Ensembles for sequence learning

THÈSE N^o 3518 (2006)

presentée à la Faculté sciences et techniques de l'ingénieur

École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de docteur ès sciences

par

Christos Dimitrakakis

Bachelor of Engineering,
University of Manchester, UK

Master of Science,
University of Essex, UK

acceptée sur proposition du jury:

Prof. Hervé Bourlard, directeur de thèse
Dr. Samy Bengio, codirecteur de thèse
Prof. Holger Schwenk, rapporteur
Dr. Andrew Morris, rapporteur
Prof. Aude Billard, rapporteur

Lausanne, EPFL
2006

Version abrégée

Cette thèse explore l'application de méthodes d'ensemble à différentes tâches d'apprentissage séquentiel, ainsi qu'à leurs relations, avec emphase sur les problèmes d'apprentissage supervisé et par renforcement. On concentre sur le development et l'examination critique des nouveaux mèthodes ou de nouveaux applications des mèthodes existant.

Dans ces deux types de problemes, même après avoir observé des données on n'est souvent pas sûr de l'hypothèse correcte parmi les possibles. Cependant, dans beaucoup de méthodes pour problèmes supervisés et aussi par renforcement, cette incertitude est ignorée, dans le sens qu'il n'y a qu'une solution sélectionnée dans tout l'espace des hypothèses. En dehors de la solution classique offerte par la formulation bayesienne analytique, les méthodes d'ensemble offrent une approche alternative pour représenter cette incertitude. Cela se fait simplement en maintenant un ensemble d'hypothèses alternatives.

Le problème séquentiel supervisé qui est considèré est la reconnaissance automatique de la parole par des modèles de Markov cachés. L'application de méthodes d'ensemble sur ce problème présente des difficultés en soi, puisque la plupart d'entre elles ne peuvent être facilement adaptées à des tâches d'apprentissage séquentiel. Cette thèse propose différentes approches pour appliquer des méthodes d'ensemble à la reconnaissance automatique de la parole et ainsi développe des méthodes pour l'entraînement efficace des mélanges phonétiques avec ou sans accès des données pour l'alignement phonétique. De plus, une notion de perte esperée est introduite pour l'intégration des modèles probabilistiques avec l'approche de boosting. Dans certains cas il y a des améliorations substantielles par rapport au système de base.

Dans les problèmes d'apprentissage par renforcement le bout est d'agir d'une manière qui maximise les récompenses futures dans un certain environnement. Dans ces problèmes, l'incertitude devient importante parce que ni l'environnement ni la distribution des récompenses ne sont connus. Cette thèse présente de nouveaux algorithmes pour agir optimalement en présence d'incertitude, basés sur des considérations théorétiques. Des représentations d'incertitude basée sur des méthodes d'ensemble sont développées et testées sur quelques tâches simples, avec une performance comparable à l'état de l'art. Le thèse aussi fait des parallélismes entre les représentations d'incertitude proposées qui sont basées sur des estimations du gradient et le "prioritised sweeping" et aussi entre l'application d'apprentissage par renforcement sur le contrôle d'un ensemble des classificateur et des méthodes d'apprentissage d'ensemble supervisées classiques.

Mots clés : Ensembles, boosting, bagging, melange des experts, reconnaissance de la parole, apprentissage par renforcement, exploration-exploitation, incertitude, apprentissage des séquences, décision sé-

quentiels.

Abstract

This thesis explores the application of ensemble methods to sequential learning tasks. The focus is on the development and the critical examination of new methods or novel applications of existing methods, with emphasis on supervised and reinforcement learning problems.

In both types of problems, even after having observed a certain amount of data, we are often faced with uncertainty as to which hypothesis is correct among all the possible ones. However, in many methods for both supervised and for reinforcement learning problems this uncertainty is ignored, in the sense that there is a single solution selected out of the whole of the hypothesis space. Apart from the classical solution of analytical Bayesian formulations, ensemble methods offer an alternative approach to representing this uncertainty. This is done simply through maintaining a set of alternative hypotheses.

The sequential supervised problem considered is that of automatic speech recognition using hidden Markov models. The application of ensemble methods to the problem represents a challenge in itself, since most such methods can not be readily adapted to sequential learning tasks. This thesis proposes a number of different approaches for applying ensemble methods to speech recognition and develops methods for effective training of phonetic mixtures with or without access to phonetic alignment data. Furthermore, the notion of expected loss is introduced for integrating probabilistic models with the boosting approach. In some cases substantial improvements over the baseline system are obtained.

In reinforcement learning problems the goal is to act in such a way as to maximise future reward in a given environment. In such problems uncertainty becomes important since neither the environment nor the distribution of rewards that result from each action are known. This thesis presents novel algorithms for acting nearly optimally under uncertainty based on theoretical considerations. Some ensemble-based representations of uncertainty (including a fully Bayesian model) are developed and tested on a few simple tasks resulting in performance comparable with the state of the art. The thesis also draws some parallels between a proposed representation of uncertainty based on gradient-estimates and on "prioritised sweeping" and between the application of reinforcement learning to controlling an ensemble of classifiers and classical supervised ensemble learning methods.

Keywords: Ensembles, boosting, bagging, mixture of experts, speech recognition, reinforcement learning, exploration-exploitation, uncertainty, sequence learning, sequential decision making.

Contents

List of Figures

List of Tables

Acknowledgments

I wish to thank my supervisor, Samy Bengio, for his invaluable aid and guidance during the writing of this thesis and my thesis director, Hervé Bourlard, for his continued support. I also extend my thanks to Silvia Chiappa, Daniel-Gatica Perez, José del R. Milán, Hynek Hermansky, Joseph Keshet, Ricardo Chavarriaga, Jean-François Paiement and all the other researchers and students at IDIAP for their time, feedback and friendship as applicable. I would particularly like to thank Mikaella Keller and Petr Motlicek for comments on parts of the thesis. Special thanks go to Hélèn Paugame-Moisy and to Mikaella Keller for translating the abstract into proper french. The interesting discussions with Adam Atkinson, Sean Barrett, Zho Fang, Stephen Granade, Gunther Schmidl, Dan Schmidt, Dan Shiovitz, Jacob Wildstrom and the other people on if-MUD have helped me improve some technical and presentational aspects of the work. I would also like to thank Richard Sutton, without whose encouragement I would never have embarked upon this PhD. Finally, I am indebted to my family for their support during this period.

This work was carried out in the framework of the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2). The NCCR are managed by the Swiss National Science Foundation on behalf of the federal authorities. This work has also received financial support from both the Swiss National Science Foundation from the grants Divide And Learn (2100-61243.00) and ADASEQ (2000-65101.01) under the MULTI project (2000-068231.021/1) and from IDIAP.

Chapter 1

Introduction

Machine learning is in general concerned with the *inference* of models given some *observations*. The inferred models can then be used for *making decisions*. The inference procedure corresponds to firstly selecting a hypothesis class within which a model that explains the data is expected to be found and secondly to searching for such a hypothesis. However there is always some uncertainty remaining about which of all the possible hypotheses is correct, since more than one model might fit the observations to some degree. This uncertainty can also affect the decision making process. To motivate the following discussion, it might be useful to consider a simple example of model inference and decision making.

Consider attempting to estimate the probability of a coin coming heads or tails through observation of repeated throws. Our *belief* at any point in time can be represented by the set of models that we think are in agreement with the data together with our confidence in them. For example if we have two alternative models in our hypothesis class, firstly that the coin is fair and the secondly is that the coin is biased, observing 90 heads out of 100 throws may make us believe that the biased model is a better explanation. When we subsequently decide whether or not to accept a bet on a future outcome, this decision will be based on our new belief. Naturally, our current beliefs depend upon both our observations and our beliefs prior to having seen any observations.

One particularly important type of prior belief is to assume that decisions and observations occur independently of previous ones. In the coin inference problem we could assume that each of the coin throws was independent of each other. In other cases however there may be some type of temporal structure. In another version of the coin problem, while the coin throws themselves may be independent of each other, we will still have a sequential decision making problem if after every bet we are allowed to decide whether or not to continue betting. Then the temporal structure arises because of the fact that our current beliefs depend upon our previous ones. Another temporal dependence arises if we are playing a game such as chess against an adversary: then clearly the history of our decisions can affect his future moves, simply because we change the board state: thus, our future observations can depend upon previous ones. And clearly, our own future moves will depend on prior ones. This is true even if the two players are both simply reacting according to a fixed plan to each observed board position, since the task includes a time-dependent state of the *environment*. However, if the player makes moves according to some particular *beliefs*, those can also change after each move. This changing belief state, like in the continuous-betting coin problem, induces a temporal structure in itself.

Problems without temporal structure are frequently called *static learning* tasks, as they model dependencies between variables that are not dependent in time. These include static pattern recognition, clustering and function approximation (see for example Bishop, 1995). Tasks with temporal structure fall into the domain of *sequence learning* and include applications such as sequential decision making (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998; Govindarajulu, 2004; Wald, 1947), speech recognition (Rabiner and Juang, 1993) and system control (Bertsekas, 2001).

In most common applications involving either type of problems, inference results in the selection of a single inferred model. However there exist many cases in which there is a plurality of models, each one of which modelling the data to some extent. Multiple models can arise in the following cases: Firstly, when multiple models have been constructed out of necessity, perhaps because each one uses slightly different assumptions. Secondly, when there is a 'natural' breakdown of the problem. Thirdly, when an algorithm explicitly calls for the construction of multiple hypotheses. Lastly, when the framework in which inference is performed, such as a probabilistic framework, requires maintaining a belief over the complete hypothesis set. Methods where multiple models are maintained instead of a single model fall in the domain of ensemble methods.

This thesis shall be concerned with the application of such methods to sequence learning tasks, in particular with the problems of speech recognition and reinforcement learning. While ensemble methods have been in use for quite some time in static learning tasks, their use in sequential tasks has been limited.[1] Thus it is interesting to investigate if, how and whether it is worthwhile to apply or adapt existing ensemble methods to sequential learning tasks.

The application of ensemble methods might also serve some practical purposes. Firstly, because it may allow for model reuse; using different combinations of previously inferred models for different tasks. Secondly, because it may allow for a more tractable optimisation procedure; it can be easier to infer the optimal combination of a given set of relatively good sub-models than to directly infer an optimal overall model. Thirdly because it allows us to represent uncertainty in a natural way. Such methods have been employed successfully in static problems in the past: For example Schwenk and Bengio (2000) show increased classification performance in a pattern recognition task using Ada-Boost (Freund and Schapire, 1997). Thus it would be interesting to see whether they can enjoy a similar success in sequential problems.

Applying such techniques is generally straightforward for static models, since these can be described as a simple mapping from an input to an output space. Thus, it is very simple to combine multiple static models that map to the same space for most spaces of interest[2]. In sequential learning, however, things are not so clear-cut since for some tasks, such as sequence recognition, the output is the set of sequences of any non-zero length. Thus, combining multiple experts can seldom be done by simply considering their decisions independently. We need a formal framework to combine the experts themselves into an appropriate mixture model that would jointly take decisions.

[1]One of the major applications as of the moment of writing appears to be tracking the movement of dynamic objects via particle filtering techniques.

[2]For example, for any $\{f_i\}$ in some linear space $\mathcal{F}$, their linear combination will by definition belong to the same space.

Another type of difficulty arises when the model that we are using in order to make decisions is allowed to change each time we make decisions. For example in a speaker authentication task, we may allow our model to incorporate newly acquired data after each successful authentication of a client. However the optimal way to do this is not immediately obvious, since a successful authentication by an impostor would lead to a deterioration of our model. It would be interesting to see whether ensemble methods could be applied to sequential decision making problems such as this, by explicitly maintaining a set of alternative hypotheses such that decision will be made taking into account the uncertainty represented by this set of hypotheses.

To summarise, one may say that this thesis, as the title suggests, deals with sequence learning (the speech recognition and reinforcement learning problems) and ensembles (boosting, bagging, mixture of experts, particle filters). The main question is how to apply ensemble methods (where a model is built of many similar elements) to sequential learning problems, such as speech recognition and reinforcement learning. Perhaps Chapter 6 is an exception to the overall theme, as it describes an RL technique used as an optimisation method for training an ensemble. However the speech recognition chapters specifically address the question of using ensemble techniques for solving the problem of speech recognition, while the last two chapters specifically address the problem of exploration in reinforcement learning and (among other things) propose an ensemble (a population of estimates) as a component of an approximate solution to the problem. Thus, with the exception of Chapter6 (where the application is not sequence learning, but where the algorithm is borrowed from sequence learning) all chapters refer to ensemble methods for sequence learning to some degree.

The following section shall describe the motivation and goals of the thesis, the specific contributions made and will offer a guide on reading through the range of topics presented.

1.1 Motivation and goals

The original goal of this work was to extend ensemble methods for static learning tasks to sequential learning. Initially, the motivation was that the fact that such methods had resulted in superior generalisation performance in static tasks and it would be interesting to see whether similar gains could be obtained in a sequence learning setting. Accordingly, some such extensions are considered in the supervised learning setting, with an application to speech recognition.

As a secondary consideration, ensemble methods are interesting because they offer a way to simplify the minimisation of cost functions. On the one hand, there exist problems for which it is natural or convenient to utilise certain models for expressing the relationships between observations. On the other hand, it can be the case that the adaptation of such models in order to minimise a cost function of interest is hard: It might be easier to find the optimal way to combine a set of fixed *basis models* , rather than discovering a new model. In this thesis such a technique is utilised within a boosting framework[3] in order to improve performance [4] in a speech recognition task.

[3]Boosting refers to a set of methods for "boosting" the performance of basis models through their combination. See Section 2.2.3 for more details

[4]As measured by the word error rate (Section 2.26).

Thirdly, such models offer a way to represent uncertainty since each member of the ensemble can be viewed as corresponding to an alternative hypothesis. This can be done in either a relatively ad-hoc way, or in the Bayesian framework. We utilise bagging and a related technique, where the ensemble is composed of a number of independent estimators, in the context of speech recognition and reinforcement learning, in which case Bayesian methods are also considered.

In the speech recognition framework, the following goals were set: Firstly, to investigate whether ensemble methods could be usefully applied to the task of phoneme classification. Secondly, to see whether the resulting ensemble models could be also used in a speech recognition task; since full inference using such models is hard, another goal was to consider a number of approximate inference techniques for the speech recognition task. Thirdly, to examine whether it was possible to develop a boosting-based scheme for the minimisation of word error rates. Finally, to see whether in practice the developed techniques perform significantly better than single HMM speech recognition.

While optimisation techniques such as expectation-maximisation are very useful when exact probabilities can be inferred, in the cases where inference is hard or infeasible, an approximate training approach may be used in its stead. Part of the work presented here covers using a reinforcement learning approach for controlling the adaptation of a mixture of classifiers. The goal was to examine whether such approximate techniques for model adaptation in ensemble models are performing comparably to more conventional ensemble methods, with a view to eventually employing such methods in a sequence recognition framework.

Reinforcement learning methods usually deal with problems where optimisation is not performed over a set of given data, but where some form of sampling takes place[5]. The form of sampling used can influence the speed of convergence significantly. While asymptotic convergence is guaranteed with only weak assumptions on the type of sampling used (Jaakkola et al., 1994), it might however be possible to obtain better short-term performance through appropriate sampling.

There is an inherent trade-off between sampling so as to improve the accuracy of the current model (exploratory behaviour) and sampling so that some cost measure given our current model is minimised (greedy behaviour), which is referred to as the *exploration-exploitation* trade-off. One of the aims of the work was to create a formal framework for the exploration-exploitation trade-off, from which the optimal balance between the two extremes would immediately arise. This balance turns out to depend upon information concerning our model's accuracy. In order to represent the uncertainty of our estimates in some meaningful way, two simple techniques are explored: ([a]) 1. a simple gradient-based technique 2. an ensemble method inspired by particle filtering(see Casella et al., 1999, for example) While this work is also partly motivated by the need to formalise some ad-hoc aspects of frequently used exploration methods, the ultimate aim was to develop nearly-optimal methods for the reinforcement learning problem that remain tractable. In particular, using an ensemble of estimates as a sampling and integration method seemed particularly attractive in this context.

[5]This is one of the main differences with a conventional supervised learning setting, though active learning is an exception.

1.2 Contributions

The first set of contributions lies in the field of speech recognition. Firstly two well-known ensemble methods, boosting and bagging, which have been applied in the past to create classifier mixtures for static patterns, are applied to the problem of phoneme classification. Secondly, methods to use the resulting mixture models for speech recognition are investigated. Multi-stream techniques are employed, which had been previously used to combine together models that used different types of features. Our application of multi-stream is a slight departure from the usual case since ti is the models that are different, but they observe the same data. This approach was initially presented in the ICASSP 2004 conference (Dimitrakakis and Bengio, 2004b).

Training mixture models to improve sequence classification does not necessarily result in improved sequence recognition using the same models. A further contribution lies in the development of a technique to apply boosting methods directly to the problem of speech recognition, where the objective to be minimised is the word error rate. This objective implies that a loss function should be defined over a whole utterance. We show experimentally that a uniform loss over each utterance does not lead to improved performance while using boosting and we develop a temporal credit assignment scheme for defining appropriate loss distributions over utterances. This technique is shown to significantly reduce word error rates and our preliminary results had originally been presented in ICASSP 2005 (Dimitrakakis and Bengio, 2005a).

The second set of contributions lies in the field of reinforcement learning. Firstly we consider how it is possible to train an ensemble of classifiers using reinforcement learning techniques. This is theoretically possible, since a supervised learning problem can be cast into a reinforcement learning framework. The use of reinforcement learning in such a context is explored experimentally for training ensemble classifiers and is compared with two well-known ensemble algorithms and is shown to perform similarly. This work has been published in (Dimitrakakis and Bengio, 2005b) and presented in ESANN 2004 (Dimitrakakis and Bengio, 2004a).

Reinforcement learning problems assume active data acquisition for model inference. The trade-off between trying to collect more data in order to improve the model and between using the current model in order to make optimal decisions is referred to as the *exploration-exploitation trade-off*. We present a simple formalisation of the concept in such a way that the balance between exploration and exploitation depends on no extra hyper-parameters, but only on the problem formulation. From this formalisation follow three approximate algorithms to the intractable problem of making decisions under uncertainty. We compare these algorithms with a closely-related method based on the value of perfect information. Since these algorithms require representing uncertainty in some way, we explore two broad classes of such methods. The first is an agnostic gradient-based technique for maintaining estimates of uncertainty for our model's parameters. The second class is ensemble techniques. We consider an ensemble composed of independent estimators and one approximating a Bayesian inference procedure, using particle filtering techniques. In either case the distribution we are interested in is represented explicitly as a population of different estimates. A comparison is then made between combinations of different techniques for estimation (including the closed-form Bayesian solution) and for decision making.

Finally, we note how these techniques are related to and provide some justification for existing ad hoc measures of uncertainty and methods of exploration. The gradient-based estimates part of this work has been presented in (Dimitrakakis and Bengio, 2005c), while the work related to the nearly optimal exploration policies is presented in (Dimitrakakis, 2006)

1.3 Organisation

After a brief technical introduction in Chapter 2, the thesis is organised in two parts. The first part details contributions related to speech recognition, while the second deals with contributions related to sequential decision making in general and to reinforcement learning particular.

In the first part, Chapter 3 deals with the application of two ensemble methods, bagging (Breiman, 1996) and boosting (Freund and Schapire, 1997), to a phoneme classification task and subsequently with techniques for using the phoneme mixture models to perform speech recognition. Chapter 4 develops a boosting-related technique for creating a mixture in such a way as to minimise the word error rate. This approach is then compared with the phoneme-based approaches and with a baseline system.

The second part focuses on reinforcement learning (RL). Chapter 5 offers a brief introduction to the field and necessary concepts and technical details. Chapter 6 focuses on the relations between supervised and reinforcement learning techniques and explores the use of a simple RL-based technique for training a mixture of multi-layer perceptrons in place of the EM algorithm. Chapter 7 introduces a formalisation of the exploration-exploitation trade-off and some optimal decision thresholds based on simple bounds, which are given in terms of probability distributions over the expected reward of actions, then proceeds to describe practical algorithms. Chapter 8 explores two ways for representing this uncertainty. The first is a set of simple parametrised approaches, where an estimate of the cost gradient with respect to the parameters is used as a measure of the accuracy of the parameters. The second set of methods explicitly maintains a distribution of estimated returns. Two of those are an ensemble representation of distributions, one being a simple approach related to bagging and the other a Monte Carlo approximation of a Bayesian estimate. Finally, some experiments are performed where these expressions of uncertainty in our estimates and an analytical Bayesian estimate are used to guide exploration according to different decision making algorithms.

Readers interested in the speech recognition part of the work are advised to continue immediately to the subsequent chapter for background information and then proceed to chapter 4. Those interested in the reinforcement learning part of the work reinforcement learning are advised to read Chapters 2 and 5 for a review of the basic concepts. Readers interested more in the exploration-exploitation trade-off may skip Chapter 6 altogether and proceed directly to chapters 7 and 8.

Chapter 2

Background

2.1 Sequence learning

Sequence learning (see Sun and Giles, 2001, for example) essentially deals with the problem of discovering relationships between variables and storing such relationships in a compact form or law. The questions that are asked are similar, i.e. what is the expected value of y given x? or what is the probability density of z? What distinguishes sequential learning tasks from static ones is the fact that the relationships considered are between ordered tuples of variables: while a static task could entail learning a mapping $f : \mathcal{X} \to \mathcal{Y}$, in a sequence learning task the mapping could be of the form $f : \mathcal{X}^* \to \mathcal{Y}$. Here, and throughout the text, $\mathcal{X}^*$ denotes the set of all finite tuples $(x_1, x_2, \ldots, x_t, \ldots, x_T)$ with $x_t \in \mathcal{X}$, while $\mathcal{X}^n$ denotes the set of all such n-tuples specifically. This property makes sequential learning tasks of this type not amenable to a simple static treatment. Common sequential learning tasks include supervised learning tasks such as sequence classification (where we have to decide to what class a given sequence belongs) and recognition (where we have to determine a sequence of events that have given rise to an observation sequence), and sequential decision making tasks (where we have to make decisions in a sequential manner, adapting our behaviour as we observe new data, and planning into the future). These are discussed below.

2.1.1 Supervised learning tasks

Supervised learning (see Trevor Hastie and Friedman, 2001, for example), tasks are in general association tasks. The aim is to find a function $f : \mathcal{X} \to \mathcal{Y}$ in some class $\mathcal{F}$ such that some cost with respect to a potentially infinite set of example pairs (x, y) with $x \in \mathcal{X}$, $y \in \mathcal{Y}$ is minimised. The form and origin of the cost and the mapping f depend upon the problem and its formulation. For example in probabilistic models the mapping results from a distribution over $\mathcal{Y}$ and the optimal model is that whose conditional (or posterior) probability given the data and our prior knowledge is highest. In this respect, sequential supervised learning tasks do not differ from static ones. However, when $\mathcal{X}$ or $\mathcal{Y}$ is a sequence space, some specific application areas arise, for example in the diagnosis of cardiac diseases from electrocardiogram (ECG) data, the detection of the onset of epileptic seizures from electroencephalogram (EEG) data, the recognition of speech and the categorisation of music to name but a few. Those and other application areas within supervised sequence learning belong to one of the following different class types: sequence classification, sequence segmentation, sequence recognition and sequence prediction.

The **sequence classification** task entails assigning a sequence to one or more of a set of categories. The classification of a piece of music from a set of predetermined categories, speech-based user authentication and patient diagnosis using ECG data can all be formulated as sequence classification tasks. More formally, given a finite label set $\mathcal{Y}$ and an observation set $\mathcal{X}$, the sequence classification task entails creating a mapping $f : \mathcal{X}^* \to \mathcal{Y}$, such that for any $x \in \mathcal{X}^*$, with $x = (x_1, x_2, \ldots, x_T)$, $x_t \in \mathcal{X}, T > 0$, $f(x)$ corresponds to the predicted label, or classification decision, for the observed sequence x.

For probabilistic models, the predicted label is derived from the *conditional* probability of the class given the observations, or *posterior class probability* $P(Y = y|X = x)$.

While some models estimate this probability directly, other models estimate $p(x|Y = y)$, the conditional density of the observations given the class label. The posterior class probability can be obtained by using the definition of conditional probability in the form that is known as Bayes' rule:

$$P(Y = y|X = x) = \frac{p(x|Y = y)P(Y = y)}{p(x)}. \tag{2.1}$$

Definition 2.1 (Bayes classifier) *A classifier $f : \mathcal{X}^* \to \mathcal{Y}$ that employs (2.1) for calculating $P(Y = y|X = x)$ and makes classification decisions according to*

$$f(x) = \arg\max_{y \in \mathcal{Y}} P(Y = y|X = x) \tag{2.2}$$

is referred to as a Bayes classifier.

At least in an abstract sense, this task formulation is exactly the same as the formulation of the static classification task, the only difference being in the types of models used and the space $\mathcal{X}$, which in our case can be a sequence. Application areas where this task formulation include speaker identification and verification, music classification. Either application could be implemented in this framework by first creating a number of models equal to the number of classes, such that each model represents the class. Then, given a sequence belonging to an unknown class, the class posteriors are calculated. At that point, it is possible to directly make a decision by selecting the class with the highest posterior given the observations.[1]

In the **sequence segmentation** task, we are trying to detect the onset time of particular events, assuming that the sequence of events is known. For example given a set of EEG data containing an epileptic seizure we can attempt to find the time at which the seizure occurred. As another example, we may have a phonetic transcription of a spoken word and be tasked with determining the onset of various phonetic units from the speech signal. More formally, we are given sequences of labels $y \in \mathcal{Y}^*$ and observations $x \in \mathcal{X}^*$ in pairs (x, y) such that the length of the label sequence, $m = L(y)$, is smaller than or equal to $n = L(x)$. The task is to determine the label sequence $f \in \mathcal{Y}^n$ that maximises $P(Y = f|X = x)$.

[1]However, care should be taken as this simply selects the most probable class and makes an implicit assumption about the costs of making incorrect decisions. Discussion of varying costs for decisions will be deferred until Section 2.1.3.

Not all possible label sequences are considered, since we are limited to sequences $\{f_i\}$ for which $\exists\, t \in [1, m]$ such that $f_i = y_t \ \forall i \in [1, n]$ under the constraint that $f_{i+1} = y_t$ or $f_{i+1} = y_{t+1}$. This limits the choice of sequences to a large, but finite set of possible sequences. Thus, at least formally, the sequence segmentation task is equivalent to the sequence classification task, though in practice an exhaustive evaluation is not performed. In general this task formulation is used when the purpose is to discover the onset of events that are known to have occurred within the observation time window.

In **sequence recognition**, we attempt to determine a sequence of events from a sequence of observations. For example, given a spoken sentence we may try to infer the sequence of words that was spoken or given a musical recording we can infer the sequence of notes played. More formally, we are given a sequence of observations x and are required to determine a sequence of labels $y \in \mathcal{Y}^*$, i.e. the sequence $y = (y_1, y_2, \dots, y_k)$, $L(y) \leq L(x)$ with maximum posterior probability $P(y|x)$. In practice, models are used for which it is not necessary to exhaustively evaluate the set of possible label sequences.

In **sequence prediction** tasks, we are given a sequence of observations x and are required to predict future values of the sequence. While such a prediction can take the form of a full probability density $p(x_{t+1:t+k}|x_{1:t})$, often it consists simply of an expectation, i.e. $E[x_{t+1}|x_{1:t}]$. Dynamical system modelling, as employed for example in weather and stock market prediction, music generation and the simulation of controlled systems, falls within the sequence prediction framework. Sequence prediction is closely related to other tasks where temporal relationships between instances of a random variable are modelled, such as sequence smoothing and filtering.

2.1.2 Probabilistic inference

In order to solve any of the above supervised tasks, we must solve two different *inference* problems. Firstly, given some set of data D, we must infer a model which will be used to predict the data. Typically we attempt to find a model h^* within a class of models $\mathcal{H}$ which minimises some particular cost criterion. In a probabilistic framework, In all three cases, we assume that there is a set of models $\mathcal{H}$ representing all the possible hypotheses. Furthermore, each model $h \in \mathcal{H}$ defines a probability distribution $p(\cdot|h)$. The exact form will depend upon the model and problem. A frequent choice when modelling dependencies between variables x, y, is to use the model for either the conditional $p(y|x, h)$ or the joint density $p(y, x|h)$. For cases where h defines simply a deterministic function f, this density is singular. We also in general assume a *prior* belief ξ over the set of models $\mathcal{H}$, which we define as a probability density with value $p(h|\xi)$ for every $h \in \mathcal{H}$. In this framework, there are three types of inference that take place: maximum likelihood (ML), maximum a posteriori (MAP) and full Bayesian inference.

In MAP inference, we attempt to find the density

$$p(\cdot|h^*), \quad h^* = \arg\max_h p(h|D, \xi), \tag{2.3}$$

in other words, the model with maximum posterior probability given the data and the prior. ML inference is essentially the same as MAP, but with $p(h'|\xi) = p(h|\xi)$ for all $h, h' \in \mathcal{H}$. Frequently it is written as

$$p(\cdot|h^*), \quad h^* = \arg\max_h p(h|D). \tag{2.4}$$

Finally, in full Bayesian inference we never select a single model out of the set: we maintain a *belief*, represented by a probability density over the whole of $\mathcal{H}$ which initially is the same as the prior belief ξ and which changes as we receive more data. We can write this *posterior* density as

$$p_{\mathcal{H}}(\cdot|D, \xi) = \int_{\mathcal{H}} p(\cdot|h, D, \xi)p(h|D, \xi)dh. \tag{2.5}$$

More frequently than not, such an integration does not have a closed form solution. One attractive type of approximation is offered by Monte Carlo methods(see Casella et al., 1999), where the integration over the space $\mathcal{H}$ is approximated by a finite set of elements H, resulting in models of the type

$$p_{\mathcal{H}}(\cdot|D, \xi) \approx \sum_{h \in H} p(\cdot|h, D, \xi)p(h|D, \xi). \tag{2.6}$$

After having obtained some model, we may then use it to infer something about previously unseen data. All the supervised learning tasks mentioned in Section 2.1.1 can be cast in the probabilistic framework. Given our model and some observations we may predict any of the given quantities in a probabilistic manner, with details of how inference is performed depending largely on the chosen model. For example the Bayes Classifier presented in the previous section corresponds to the MAP inference procedure; the difference being that the inference is performed over classes rather than models.

Nevertheless, after inference having been performed, it still is necessary to make some kind of *decision*. Taking the classification task as an example, it is quite common to decide to label an unknown example with the label of the class having the maximum posterior probability given our model and the current observation. This simple way to make decisions unfortunately is only optimal when we just want to minimise the probability of a classification error. Sometimes, however, the objective might be different - for example it might be the case that each classification error carries a different cost and that we wish to take the decision that minimises the expected cost. In discrete decision problems the calculation is usually trivial when the expected cost of each type of error is known and either ([a]) 1. either we are being *greedy* in the sense that we are only interested in minimising the expected cost for the next time step or 2. each decision made is independent of future ones. If many decisions have to be taken in sequence and they depend on one another, perhaps because we are controlling a dynamical system, or simply because our model/belief may change over time[2] as we observe more data, the problem becomes much more difficult.

The field which deals specifically with making sequential decisions is called **sequential decision making**. This is a more general field, since it deals not only with the pure modelling aspect, but also with data collection and with decision making with respect to both the collection of data and the use of the model: In standard formulations of sequence classification, recognition and prediction, there is a

[2]Since the expected cost is also part of our model, if this is uncertain then our beliefs with respect to its value will also change over time

fixed model, which has been inferred from some data, and which is used to classify, recognise or predict on novel data. However in the more general setting of sequential decision making, decisions made now may have future repercussions because either the future state of the environment or the model can be affected by our current decision.

2.1.3 Sequential decision making

The supervised learning tasks described in Section 2.1.1 can be seen as special cases of sequential decision making (see Sutton and Barto, 1998; Wald, 1947; DeGroot, 1970)(SDM). In particular, supervised learning tasks can be viewed as the search for an f that minimises a cost functional, such as for example

$$C_f = \int\int c(f(x), y)p(x, y)\ dx\ dy,$$

where c is a previously defined sample cost function, and p is a potential function, such as a probability density (in which case the above integral becomes the Bayes risk). In this case c is given and minimisation can be done analytically, or at least its gradient with respect to f can be computed – something necessary for minimisation algorithms such as gradient descent (see Bertsekas, 1999) and generalised alternating minimisation algorithms (see Gunawardana and Byrne, 2005, for an interesting overview) such as Expectation Maximisation (see for example McLachlan and Krishnan, 1997).

Sequential decision making differs from the classic supervised learning setting because the decision function f represents a *sequence of decisions* on which the cost C_f depends. It is possible to factorise the cost by making it a function of *instantaneous* costs incurred after the decision made at each time-step. This means that its exhaustive calculation for all possible f becomes prohibitive as the number of individual decisions in the decision sequence increases. Another complication is that the instantaneous costs themselves may itself be initially unknown. Lastly, in a classical supervised learning setting the correct decision sequence for each observed sequence example would be known or would be trivially calculated. In an SDM setting, each decision sequence would result in a noisy sequence of instantaneous costs from which the overall cost for each f would have to be estimated.

In sequential decision problems making it is most common to refer to the problem of maximising *utility* rather than a minimising cost. The utility is a functional of sample rewards r_t over time, such as for their expected weighted sum, or their lower bound with probability larger than some δ among other possibilities. The problem is thus to make a decision *now* that maximises a functional of *future* rewards. The rewards themselves can be random variables and the optimal decision will depend on our beliefs about their distribution and all possible future decisions. To give a concrete example, deterministic games (such as chess) can be placed in this framework by considering them as tasks for which the aim is to maximise the total return $R = \sum_t r_t$, with sample rewards being 0 in all cases apart from the case when the game ends, when the reward would be 1 for a win, and -1 for a loss. In this case, the probability of a win for a player would be related to his expected return by $P(\text{win}) = (E[R] + 1)/2$.

A single game of chess in this context is frequently referred to as an *episode*. It is instructive to note than when complete episodes are considered, the total reward is known and learning the expected reward given a game is akin to a standard supervised learning task: each game constitutes an observation and the relationships between games and rewards can be modelled. In fact, for chess, the only thing that we need to know about a game is the final state: this completely determines the outcome of the game. On the other hand, when individual states in each game are considered, things become more complicated. Firstly, it is not trivial to assign probabilities of wins given a particular state. Secondly, from the point of view of the decision maker, each decision is made with a view to a *delayed reward*. The following sections give a short overview on optimal policies for immediate and delayed rewards.

n-armed Bandit problems

n-armed bandit problems (see Sutton and Barto, 1998; Mannor and Tsitsiklis, 2004; Madani et al., 2004a), are a special form of stochastic games, in which each action from some predefined set of actions results in a stochastic *reward*, whose distribution is unknown but stationary. The aim of the game is to discover the action with the maximum average reward. More formally, consider a stochastic function $r : \mathcal{A} \to \mathbb{R}$, where $\mathcal{A}$ is a discrete space of size n called the action space, such that each action a results in a stochastic reward $r(a) \equiv p(r|a)$. We will refer to this as the *reward distribution*. We would like to select actions in an optimal way, i.e. so that $E[r]$ is maximised.[3] This expectation is completely defined by a probability distribution over actions $\{P(a)|a \in \mathcal{A}\}$. Such a distribution, or method for selecting actions, is called a *policy*. Policies for which this distribution does not change over time are called *stationary policies*. The expected reward given some stationary policy π is easy to calculate:

$$E[r|\pi] = \sum_{a \in \mathcal{A}} E[r|a] P(a|\pi). \tag{2.7}$$

The task is to find π^* such that $E[r|\pi^*] \geq E[r|\pi] \quad \forall\ \pi \neq \pi^*$.

If the $E[r|a]$ is known for each action a, then it is trivial to obtain the optimal solution by setting $P(a)$ to 1 for the action with the maximum expected reward, and to 0 for the remaining actions. However, normally neither the expectation nor the exact distribution is known for r and we must resort to sampling in order to evaluate $E[r|\pi]$, either directly, or through the estimation of $\{E[r|a]\}_{a \in \mathcal{A}}$

Different sampling assumptions allow different algorithms, which, however, are all related. When it is possible to sample from all actions simultaneously, algorithms such as the Hedge algorithm described in (Freund and Schapire, 1997) can be used to determine the optimal probabilities. When sampling is limited to one action at a time, one must consider the balance between gaining new knowledge about actions that currently appear inferior, in order to be able to determine the best action as quickly and accurately as possible, and using the current knowledge in order to maximise the expected reward. This is referred to as the *exploration-exploitation trade-off* (see Sutton and Barto, 1998, chap. 2) or (MacKay, 1997, chap. 36) and it becomes significantly harder to resolve when a dynamic environment is considered. This problem is further addressed in Chapters 5 and 7.

[3]It is also possible to define a utility function that is not simply linear with respect to the reward, but which includes perhaps its variance. That ventures into risk-sensitive decision making. In the end, it is possible to reduce all problems to problems of simply maximising a single numerical value with appropriate transformation. Utility theory (see DeGroot, 1970, chap. 7) deals with formalising the notion of action preferences.

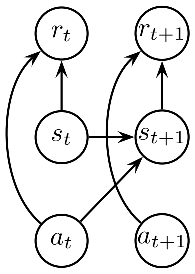


Figure 2.1: Markov decision process

Reinforcement learning

One may generalise the bandit problem where the environment consists of a static reward distribution conditioned on the actions in two ways. Firstly, it can be extended to the case where the environment is defined as a Markov decision process (MDP) rather than a static one.

Definition 2.2 (Markov decision process) *A Markov decision process (see Figure 2.1.3) is defined as the tuplet* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathfrak{T}, \mathfrak{R})$*, comprised of a set of states* $\mathcal{S}$*, a set of actions* $\mathcal{A}$*, a transition distribution* $\mathfrak{T}(s', s, a) = P(s_{t+1} = s' | s_t = s, a_t = a)$ *and a reward distribution* $\mathfrak{R}(s', s, a) = p(r_{t+1} | s_{t+1} = s', s_t = s, a_t = a)$

We may now extend the notion of a policy from the simpler n-armed bandit setting to that of MDPs. Now, a policy π defines the probability distribution $P(a_t|s_t, \pi)$. Policies for which $P(a_t|s_t, \pi) = \pi_{a,s}$ are called stationary.

The second generalisation consists of discovering a policy that does not only attempt to maximise the expected immediate reward, but also future rewards. This in general is the goal of reinforcement learning: to discover some policy π^* for which the expectation of some functional R of the reward over time is maximised:

$$\pi^* = \arg\max_{\pi} E[R|\pi].$$

This functional R is called the *return*. A common choice for R is the cumulative discounted reward:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma \in [0, 1)$ is a scalar *discount factor.*[4] One can think of γ as a mechanism for weighing the importance of rewards in the distant future relative to immediate rewards. When $\gamma = 0$, the optimal policy is the policy that maximises the expected value of the immediate reward only. As $\gamma \to 1$, the optimal policy is that which maximises the expected value of all future rewards.

[4]Setting $\gamma = 1$ corresponds to the special case of average-reward infinite-horizon problems and will not be dealt with here.

$$\begin{tikzcd} x \arrow[r] \arrow[rr, bend left] & h \arrow[r] & y \end{tikzcd}$$

(a) Conditional mixture

$$\begin{tikzcd} x \arrow[rr, bend left] & h \arrow[r] & y \end{tikzcd}$$

(b) Static mixture

Figure 2.2: Mixture models

Given a model, (i.e. when the environment dynamics and reward expectations for each state-action pair are known *a priori*), it is possible to calculate the expected return for any given policy. In fact, it is even possible to obtain the optimal stationary policy using dynamic programming (Bellman, 1957b) techniques. However in general the model is not known and stochastic approximation techniques must be used. In general such methods converge only asymptotically (Jaakkola et al., 1994). Reinforcement learning techniques fall within this domain and are more thoroughly reviewed in Chapter 5.

2.2 Mixture models

While a number of models can be used for representing our beliefs in an inference task, this thesis is mainly concerned with ensemble methods. In such methods, sometimes also called modular methods, the task is solved jointly by a set of similar models. We shall be mainly concerned with the class of mixture models in a probabilistic framework.

As a motivation, we will consider the problem of estimating a function $f : \mathcal{X} \to \mathcal{Y}$, given data (x, y), with $x \in \mathcal{X},\ y \in \mathcal{Y}$. To give a concrete example, in a classification task x might be an observation and y a vector of class membership probabilities. The following exposition will not be restricted to such tasks however, but to the more general case of estimating the conditional density $p(y|x)$.

The goal can be stated as constructing f in order to minimise some appropriate loss function. We will focus on probabilistic methods, where the aim is to estimate the conditional density $p(y|x)$ in the form of a mixture of models from a set $\mathcal{H}$ where each model h defines its own conditional density $p(y|x, h)$:

$$p(y|x) = \sum_{h \in \mathcal{H}} p(y|x, h)p(h|x). \tag{2.8}$$

This model, shown in graphical form in Figure 2.2(a), can also be stated in terms of the joint density

$$p(y, x) = \sum_{h \in \mathcal{H}} p(y, x|h)p(h). \tag{2.9}$$

This thesis deals mostly with models where one or more of these random variables lie in a sequence space. There is large number of such models, however will focus on only a few types.

The *static* mixture model, shown in Figure 2.2(b), results from the assumption that the mixing probabilities of each model do not depend on the observations, so that $p(h|x) = p(h)$. This is employed in model sampling approaches investigated herein, such as bagging (Breiman, 1996).

Another special case is the *switching* model, where only one model is responsible for the ensemble's decision at any one time, i.e. $x \in \mathcal{X},\ P(h = i|x) = 1$ for some $i \in \mathcal{H}$.

In the case of sequence data, with $x = (x_1, \ldots, x_T)$, switching can be constrained by defining probabilities $P(h_{t+1} = i|h_t = j, x)$.

2.2.1 Training and model selection

Mixture models are primarily of interest because given an *appropriate set* of base hypotheses $\mathcal{H}$, it is possible to approximate any function $f : \mathcal{X} \to \mathcal{Y}$. In fact, many types of models are of this form, including multi-layer perceptrons, generalised linear models (such as radial basis function networks) and Gaussian mixture models (see Jordan (1999); Trevor Hastie and Friedman (2001) for an overview of such models). Two questions that arise when training such models are firstly how the mixture components are selected and secondly how the mixing parameters are adapted.

In sampling methods, a base hypothesis $h \in \mathcal{H}$ is created from the data according to some distribution $p(x|h)$. For the case where the distribution over the data is the same for all base hypotheses, the corresponding posterior distribution $p(h|x)$ is uniform. Such sampling also provides an opportunity to introduce model priors $p(h)$ in a non-analytic way. Bagging and cross-validated committees (see for example Parmanto et al., 1995) belong to this category, while particle filters and other Monte Carlo estimation methods have been long used as approximate Bayesian estimation procedures. As will be seen in Part II, such methods can also be useful for making decisions under uncertainty, where it is necessary to use a probability distribution to represent the uncertainty in our beliefs. In that case, the probability distribution over the hypothesis space will be approximated by a finite number of (highly probable) elements.

Another potential advantage of mixture methods is that the mixing parameters can be relatively few and that for certain cost functions there is a single global minimum. Thus, it can be much easier to adapt the mixing parameters of a mixture of hypotheses in a given class for the minimisation of a particular cost than to find the hypothesis in the class that minimises it. As will be seen in Part I, this can be used for word error rate minimisation in speech recognition.

A final advantage is the potential robustness to noise. By selecting only a single hypothesis within the allowed class, one completely disregards all the other hypotheses in the same class. With a limited number of data, this appears counter-intuitive from a probabilistic perspective: Initially one has a prior belief that covers all of the hypothesis space. Subsequently to observing data, one obtains a posterior belief (2.5) which potentially again covers all of the space. Although it can be difficult to perform Bayesian inference in practice, it may be possible to simulate it with a finite set of hypotheses.

However, even with a finite number of hypotheses, the overall model complexity can be extremely high. Ensemble pruning (see for example Zhang et al., 2006; Partalas et al., 2006) techniques attempt to reduce the size of the ensemble while maintaining or improving upon its performance.

Finally, there exist methods for constructing ensembles where not all members share the same input and output space. Firstly, it is possible to vary the input space across experts by using different sets of features for each one. Another option is to associate each target point in $\mathcal{Y}$ with a point in a higher dimensional space $\mathcal{Z}$ and then have each member of the ensemble perform a mapping $h_i : \mathcal{X} \to \mathcal{Z}_i$, where $\mathcal{Z}_i$ is a subspace of $\mathcal{Z}$. Then the outputs of all experts can be combined to reconstruct a vector in $\mathcal{Z}$.

The classic way of doing this for multi-class classification problems is the error-correcting output code described by Dietterich and Bakiri (1995). In this thesis we will be concerned only with ensembles in which all members share the same input and output space. Three such methods, bagging, boosting and mixture of experts, will be described in the remainder of the section.

2.2.2 Bagging

Model inference in classification tasks can be stated as the procedure of finding a hypothesis $h \in \mathcal{H}$, given some data D, that maximise the posterior density $p(h|D)$. However, there is no reason why the model inference should be restricted to a point estimate. One may equally well infer a distribution over the classifier space $\mathcal{H}$ given the data. This can be done either parametrically or via sampling. From this point of view, Bagging (Breiman, 1996) represents a method for sampling the classifier space $\mathcal{H}$. Let us assume we have observations $o \in \mathcal{O}$, which may consist of input and label pairs (x, y), and draw a random sample[5] D_i of pairs from a distribution $\mathfrak{D}$. Let us subsequently infer a model with maximum posterior probability $h_i = \arg\max_h P(h|D_i)$. After drawing N such samples D_i and creating an equal number of models, one from each sample, the models can be brought together into a mixture. This will satisfy

$$p(o) = \sum_{i=1}^{N} p(o|h_i)P(h_i|D_i)P(D_i),$$

where we made use of the fact that $p(o|h_i, D_i) = p(o|h_i)$. From the sampling theorem, this leads to

$$p(o) \approx \frac{1}{N}\sum_{i=1}^{N} p(o|h_i). \tag{2.10}$$

In practice bagging is performed by uniformly sampling from a fixed set of data D rather than the actual data distribution. In this case each sample D_i is a sample of size $\|D\|$ drawn uniformly with replacement from D, called a bootstrap replicate of D.

Bagging has attracted attention as a method to reduce estimator variance. For an unbiased model h, $E[h] = h^*$, where h^* corresponds to the 'true' model. However, while in expectation we are close to the true model, each individual sample of h might be far from the h^*. This can be expressed via the variance $\mathrm{Var}[h - h^*]$. If we consider h_i as an independent sample of an estimator and we use $h^{(N)}$ to denote a "bagged" estimator made up of N models then

$$\lim_{N\to\infty} h^{(N)} = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} h_i = E[h],$$

from which it follows that

$$\lim_{N\to\infty} \mathrm{Var}[h^{(N)} - h^*] = \mathrm{Var}[E[h] - h^*],$$

which for unbiased estimators is 0..

[5]Herein we follow the standard nomenclature of calling a random sample a set of realisations

2.2.3 Boosting

Boosting algorithms (Meir and Rätch, 2003; Schapire and Singer, 1999; Freund and Schapire, 1997) are a family of ensemble methods for improving the performance of classifiers by training and combining a number of *experts* through an iterative process that focuses the attention of each new expert to the training examples that were hardest to classify by previous ones. The most commonly used boosting algorithm for classification is Ada-Boost (Freund and Schapire, 1997), where an ensemble of experts is able to decrease the training error exponentially fast as long as each new classifier has a classification error smaller than 50%.

An algorithmic view of boosting

More precisely, an Ada-Boost ensemble is composed of a set of n_e experts, $\mathcal{E} = \{e_1, e_2, ..., e_{n_e}\}$. For each input $x \in X$, each expert e_i produces an output $y_i \in Y$. These outputs are combined according to the reliability $\beta_i \in [0, 1]$ of each expert:

$$y = \sum_{i=1}^{n_e} \beta_i y_i. \tag{2.11}$$

The expert training is an iterative process, which begins with training a single expert and subsequently trains each new expert in turn, until a termination condition is met. While there exist many variants of Ada-Boost for multi-class classification problems, in this work we will mainly concentrate onAda-Boost.M1. The experts are trained on bootstrap replicates of the training dataset $\mathcal{D} = \{d_i | i \in [1, N]\}$, with $d_i = (x_i, y_i)$. The probability of adding example d_i to the bootstrap replicate $\mathcal{D}_j$ is denoted as $p_j(d_i)$, with $\sum_i p_j(d_i) = 1$. At the end of each boosting iteration j of Ada-Boost.M1, β_j is calculated according to:

$$\beta_j = \ln \frac{1 - \varepsilon_j}{\varepsilon_j}, \tag{2.12}$$

where ε_j is the empirical expected loss of expert e_j, given by

$$\varepsilon_j = \sum_i p_j(d_i) l(d_i), \tag{2.13}$$

where $l(d_i)$ is the *sample loss* of example d_i. If, for any predicate π, we let $[\pi]$ be 1 if π holds and 0 otherwise, it can be defined as: $l(d_i) = [h_i \neq y_i]$, (i.e. the zero-one loss). After training in the current iteration is complete, the sampling probabilities are updated so that $p_j(d_i)$ is increased for misclassified examples and decreased for correctly classified examples according to:

$$p_{j+1}(d_i) = \frac{p_j(i) \exp(\beta l(d_i))}{Z_j}, \tag{2.14}$$

where Z_j is a normalisation factor to make D_{j+1} into a distribution. Thus, incorrectly classified examples are more likely to be included in the next bootstrap data set. Because of this, the expert created at each boosting iteration concentrates on harder parts of the input space.

A second multi-class extension is Ada-Boost.M2, where a weight vector $w_{i,y}$ is maintained for each class that an example can belong to and where the base classifier is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$, thus providing some confidence measure (not necessarily with a probabilistic interpretation) of how likely the data is to belong to each one of the classes. The calculation of example weights and coefficients for the linear combination of classifiers requires the calculation of these confidence measures for all the classes in $\mathcal{Y}$. This calculation may be prohibitive for applications where the number of classes is extremely large. Freund and Schapire (1997, Section 5.2) gives more details on this particular algorithm.

In general, Ada-Boost and other boosting algorithms can be viewed as greedy optimisation methods[6] or the minimisation of a cost related to the classification margin (Smola et al., 2000, see). A brief overview of classification margins follows.

2.2.4 Margins

Let us say we have a set D of inputs and target pairs $x \in \mathcal{X}$, $y \in \mathcal{Y}$ drawn from a distribution $\mathcal{D}$. Assume some margin function $m : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ and some classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$. The margin function provides some measure of the confidence with which a classifier is making a classification and has the property that $m(x, y) > 0$ for any correctly classified example (x, y). One useful such margin function is the one defined for a binary classifier with output space $\mathcal{Y} = [-1, 1]$:

Definition 2.3 (Two-class classification margin)

$$m(x, y) = yf(x) \tag{2.15}$$

This particular margin function equals 1 when the correct label is predicted with high confidence and -1 when an incorrect label is predicted with high confidence. An equivalent function can be written for the multi-class case, with $\mathcal{Y} = [0, 1]^n$, where n is the number of classes, and f_y denoting the score (which could be a probability) that the classifier assigns to class y, the margin can be written as

Definition 2.4 (Multi-class classification margin)

$$m(x, y) = f_y(x) - \max_{y' \neq y} f_{y'}(x) \tag{2.16}$$

Thus the margin can serve as a measure of how far away from the decision threshold a classification is made. In general we would like to find a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maximises some functional $\mathfrak{F}$ of the margin over all the examples (or alternatively that minimises some cost function related to $\mathfrak{F}$, such as a bound).

If the function f is parameterised with parameters w, then we can employ the chain rule of differentiation and be able to use gradient descent methods to find parameters that minimise the relevant cost. In some cases, such as in the case of the linear combination of hypothesis and exponential cost in Ada-Boost, the solution can be obtained analytically.

[6]Greedy in the sense that a decision is made which would have been optimal had it been the last one to be made. Such decisions are termed greedy since they can potentially sacrifice long-term for short-term gains.

The functional

The functional $\mathfrak{F}[m]$ can in general be written as

$$\mathfrak{F}[m(f, y)] = \sum_i z(i) m(f(x_i), y_i).$$

The following are some commonly used functionals, where we use the shorthand notation $m(i) \equiv m(f(x_i), y_i)$

Definition 2.5 (Average margin) *This is useful for constructing a cost function that maximises the margin everywhere.*

$$\mathfrak{F}_D[m] = \sum_{i \in D} m(i) \tag{2.17}$$

Definition 2.6 (Minimum margin) *Useful for constructing a cost function that maximises the minimum margin.*

$$\mathfrak{F}_D[m] = \min_{i \in D} m(i) \tag{2.18}$$

Since no gradient can be computed for the minimum margin, it is common to use approximations or bounds in order to utilise gradient-based methods for its maximisation (see Smola et al., 2000; Zemel and Pitassi, 2000; Friedman, 2001).

Definition 2.7 (Exponential margin) *This is useful for constructing a cost function that puts more emphasis on the minimum margin, however tries to increase the margin everywhere. When $\beta \to 0$, it becomes the same as the average margin and when $\beta \to \infty$, it becomes the same as the minimum margin.*

$$\mathfrak{F}_D[m] = \sum_{i \in D} m(i) \frac{\exp(-\beta m(i))}{\sum_{j \in D} \exp(-\beta m(j))} \tag{2.19}$$

2.2.5 Mixture of experts

The mixture of experts (MoE) architecture was introduced in (Jacobs et al., 1991). Unlike the mixtures in bagging and boosting, the mixture of experts model is not a static mixture since there is a dependency between h and the observations x.

The model essentially takes the form of (2.8), with multi-layer perceptrons, called *experts*, representing each term $p(y|x, h)$ and another multi-layer perceptron, called the *gate* or gating function representing the $p(h|x)$ term.

The original algorithm employed a form of gradient descent for parameter adaptation. It was later extended to a hierarchical mixture of experts with expectation-maximisation training in (Jordan and Jacobs, 1994).

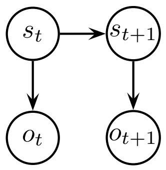


Figure 2.3: Hidden Markov model

2.3 Speech recognition with hidden Markov models

Speech recognition techniques generally consist of methods for utilising prior knowledge, such as the phonetic, lexical and grammatical morphology of the language that we wish to recognise in order to specify a class of models that correspond to the overall morphological structure, and subsequently finding the models within this class that predict the available data well, according to some well-defined optimality criterion. One such simple, yet natural, class is that of hidden Markov models.

Definition 2.8 (Hidden Markov model) *A hidden Markov model is a probabilistic model describing the relations between two variables: a state variable s in some discrete space $\mathcal{S}$, and an observation variable $o \in \mathcal{O}$. These take values s_t and o_t respectively at time t, which are related through:*

$$P(s_t|s_{t-1}, s_{t-2}, \ldots) = P(s_t|s_{t-1}) \tag{2.20}$$

$$P(o_t|s_t, o_{t-1}, s_{t-1}, o_{t-2}, \ldots) = P(o_t|s_t), \tag{2.21}$$

These dependencies admit the following factorisations of the joint distribution:

$$\begin{aligned} P(s_t, s_{t-1}, s_{t-2}, \ldots) &= P(s_t|s_{t-1})P(s_{t-1}|s_{t-2}, \ldots) \\ &= \prod_{k=1}^{t} P(s_k|s_{k-1}) \end{aligned} \tag{2.22}$$

$$\begin{aligned} P(o_t, s_t, o_{t-1}, s_{t-1} \ldots) &= P(o_t|s_t)P(s_t, o_{t-1}, s_{t-1}, o_{t-2}, \ldots) \\ &= \prod_{k=1}^{t} P(o_k|s_k)P(s_k|s_{k-1}), \end{aligned} \tag{2.23}$$

which means that only three types of distribution need to be modelled: $P(o_k|s_k)$, the observation (or emission) distribution, $P(s_k|s_{k-1})$, the transition distribution and the $P(s_1) \equiv P(s_1|s_0)$, the initial state distribution.

Training usually consists of finding a model m^* within a class of models $\mathcal{M}$ defining the distributions $P(o_k|s_k, m^*)$, $P(s_k|s_{k-1}, m^*)$ with maximum posterior density

$$m^* = \arg\max_m p(m|\mathcal{D}),$$

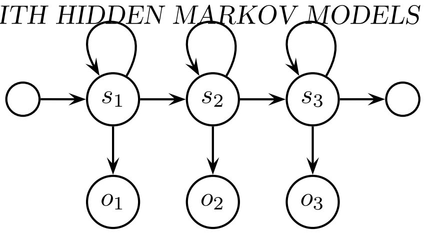


Figure 2.4: A phonetic model

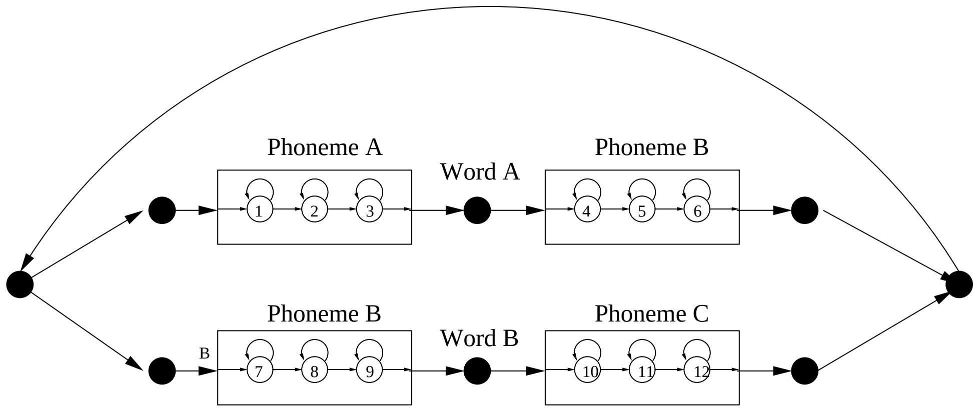


Figure 2.5: A hidden Markov model for speech recognition.

where $\mathcal{D}$ is a set of observation sequences. Most of the time $\mathcal{M}$ is restricted to models with a particular number of states and allowed transitions between states. In that case a number of optimisation techniques might be used, though in practice expectation maximisation tends to be preferred, in particular the *Baum-Welch* algorithm (Baum et al., 1970), which uses the forward-backward algorithm as the expectation step.

The most common way to apply such models to speech recognition is to associate each state s with morphological features $a \in \mathcal{A}$, such as phonemes, syllables, or words, through a distribution $P(a|s)$, which takes values in $\{0, 1\}$ in standard practice: This is done by creating a graph, as the one shown in Figure 2.5, with a set of parallel chains such that each chain maps to one word; for example, given that we are in the state $s = 4$ at some time t, then we are also definitely (i.e. with probability 1) in Word A and Phoneme B at time t. In general, we can determine the probabilities for sequences states, we can also determine most probable sequence of words or phonemes, i.e. given a sequence of observations $(o_{1:T})$, it is then possible to infer a state distribution $P(s_{1:T}|o_{1:T})$ and subsequently a distribution over morphologies, to wit the probabilities of possible word, syllable or phoneme sequences. More details are given in the following section.

2.3.1 Continuous speech recognition

Speech exhibits a hierarchical structure, whose levels correspond to different temporal scales, from short-term features such as phonemes, to long-term ones such as words and utterances. When hidden Markov models are employed, it is advantageous to construct a model at each level in such a way that the complete speech model remains a hidden Markov model.

In the Gaussian-mixture hidden Markov model applied to speech, the features in observation space $\mathcal{O}$ are features derived from a short-term speech signal, modelled using a Gaussian mixture. A commonly used such set of features are Mel-frequency cepstral coefficients (see for example Rabiner and Juang, 1993) (MFCC). Such features are derived from a linear frequency transform of a short window of audio data after pre-emphasis. The full frequency information is reduced by integrating into a fixed set of bands with Mel fiterbanks. Finally, IDFT or DCT is performed to obtain the cepstral coefficients. Frequently, first-order approximations of the first and second time derivatives of the cepstral coefficients are also included in the features. Other types of features are certainly possible, but this thesis will not be concerned with this topic.

Individual phonetic units are modelled using hidden Markov models, usually comprising of not more than 5 states. The models, such as the one depicted in Figure 2.4, have a topology such that 'backwards' transitions are not possible, i.e. $P(s_{t+1} = i|s_t = j) = 0$ for all $j > i$. Such a topology is commonly referred to as a *left-to-right* topology. Words are formed through the concatenation of phoneme models[7]. The architecture of the overall model is such that each state s in the overall model maps to one state in one phoneme HMM and one word. The word models themselves are connected according to the transitions allowed by the language model.

Given a set of utterances and speech data, it is possible to infer a hidden Markov model that explains the data given the utterances by finding the model with the highest posterior probability. Unfortunately, this inference requires the solution of an optimisation problem with many local extrema, so in practise the parameters of the HMM are initialised from reasonable values by first optimising them over a range of simpler sub-problems. In practice, speech recognition performs adequately after training each phoneme model separately using phonetically labelled speech data.

After phoneme-level training, or in cases where the data does not include time-aligned phonetic labels from the outset, a form of training called *embedded training* is used. Given a set of utterances, a set of models composed of the concatenated phoneme models comprising each utterance is trained so that we obtain ML or MAP parameter estimates given the observations. This utterance-level training has the advantage that it does not rely on a given phonetic alignment, which can be particularly noisy anyway, but on utterance labels, which are virtually noise-free. In a probabilistic framework, embedded training consists of an application of the Baum-Welch algorithm with the training data. Because the forward and backward steps in which the posterior state probabilities are calculated are time consuming, sometimes the *Viterbi approximation* can be used instead. In this case the Viterbi algorithm (see Section 2.3.1) is used to find the sequence of state with highest posterior probability. The maximisation step is performed as though the states within the sequence had a posterior probability of one and the rest a posterior probability of zero. Thus the maximisation step is sped up by a factor equal to the number of states. However, since the complexity of the expectation step remains the same, thus the overall speed up is not more than a factor of two.

[7]The sequence of phonemes that each word is composed of is called the pronunciation lexicon. It is usually determined by linguistic studies, or simply from a lexical dictionary.

Inference

We wish to find the sequence of words $w^* \in W^*$ such that $P(w^*|o) \geq P(w|o)$ for all $w \in \mathcal{W}^*$. This posterior probability of words given observations can be written as

$$P(w|o) = \sum_{s \in \mathcal{S}^T} P(w|s)P(s|o). \tag{2.24}$$

The sequence of words themselves is usually assumed to arise from some language model. In a probabilistic framework such a model would be used to obtain a probability for $P(w)$ for any sequence of words w. Frequently, models called n-grams, Markov chain models of the form $P(w_{n+1}|w_1, \ldots, w_n)$, are used. The dependency on the language model can be seen if we re-write (2.24) as $P(w|o) = p(o|w)P(w)/p(o)$.

While it is possible to calculate $P(s|o)$ with the forward-backward algorithm, the summation required in calculating $P(w|o)$ can be problematic due to the very large size of $\mathcal{S}^T$. On the other hand, the expression $\arg\max_w P(w|s)P(s|o)$ can be simplified by approximating $P(s|o)$ via an indicator function which is 1 when $s_t = s_t^*$ and 0 otherwise, where s^* is the sequence of states with the highest posterior probability given the sequence of observations. Since models are constructed so that at the most one word sequence corresponds to each state sequence, this allows inferring the word sequence directly.

In order to determine the most probable state sequence itself efficiently, the evaluation of the posterior probabilities of all state sequences is not necessary. The problem can be restated as a shortest-path problem instead. Let

$$\rho_t(i,j) = \log P(s_t = i | s_{t-1} = j, o),$$

and $\mathcal{E} = \{p(s_t = i|s_{t-1} = j)|i,j \in \mathcal{S}\}$. Given a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ and a metric $\rho_t(i,j)$, with $i,j \in \mathcal{N}$ and $t \in [0,T]$, the Viterbi algorithm (Viterbi, 1967) can be used to compute a sequence $\{s_i^*\}$ such that

$$\sum_t \rho_t(s_t^*, s_{t-1}^*) \geq \sum_t \rho_t(s_t, s_{t-1}), \tag{2.25}$$

for all sequences $\{s_i\}$. The problem is essentially a shortest path problem on a tree $\mathcal{T}$ with $T|\mathcal{N}|$ nodes and $T|\mathcal{N}|^2$ edges with metric $-\rho_t(i,j)$, whose solution is particularly efficient.[8]

Using the most probable sequence of states in order to infer a sequence of words is commonly referred to in the literature as "Viterbi decoding". While extremely fast, its main drawback is that the sequence of most probable words is not necessarily the one corresponding to the most probable sequence of states.[9]. Fortunately a trade-off between accuracy and speed can be made by using the Viterbi algorithm to create an n-best list of state sequences and then using that in place of the complete set $\mathcal{S}^T$ in (2.24) to obtain word sequence posteriors.

[8]Worst-case shortest-path complexity of Dijkstra's algorithm for general graphs with n nodes is $O(n^2)$. For trees with k nodes at each level, depth T and m connections per level, the complexity is $O(kTm)$.

[9]This because, as noted previously, the Viterbi decoding procedure corresponds to inferring the most probable sequence of words by approximating the posterior distribution over states with an indicator function.

Performance measures

Some measure of performance for determining the quality of an automatic speech recognition system is required, on the basis of some distance between the desired and actual output of the speech recogniser. Since these outputs are sequences we need an appropriate measure in the sequence space, such as the *edit* distance, otherwise known as the *Levenshtein* distance (Levenshtein, 1966). For two sequences a and b, this is defined as the number of atomic operations necessary to transform a sequence of symbols a into another a sequence b. In speech there are three types of atomic operations commonly used: word insertions, deletions and substitutions.

The most common measure of performance[10] for an automatic speech recogniser is based on the edit distance and it is called the word error rate:

$$WER = \frac{N_{ins} + N_{sub} + N_{del}}{N_{words}}, \tag{2.26}$$

where N_{ins} is the number of word insertions, N_{sub} the number of word substitutions and N_{del} the number of word deletions. These numbers are determined by finding the minimum number of insertions, substitutions, or deletions necessary to transform the target utterance into the emitted utterance for each example and then summing them for all the examples in the set.

2.3.2 Multi-stream decoding

When we wish to combine evidence from multiple models, multi-stream decoding techniques can be used as an approximation to the full mixture model (Morris et al., 2001). Such techniques derive their name from the fact that they were originally used to combine models which had been trained on different streams of data or features Misra and Bourlard (i.e. 2005).

In multi-stream decoding each sub-unit model corresponding to a morphological feature a is comprised of n sub-models $a = \{a_i | i \in [1, n])\}$ associated with the sub-unit level at which the recombination of the input streams should be performed. For any given a, the likelihood can be written as

$$p(o|a) = \sum_{i=1}^{n} p(o|a, i)p(i), \tag{2.27}$$

where $p(i)$ is a fixed weight for expert i.

We consider the case of state-locked multi-stream decoding, where all sub-models are forced to be at the same state. This can be viewed as creating another Markov model with emission distribution

$$p(o_t|s_t) = \sum_{i=1}^{n} p(o_t|s_t, i)p(i). \tag{2.28}$$

A similar method for combining models is to consider the exponentially weighted product of emission

[10]There are of course other possible measures of performance, such as those described in Morris et al. (2004).

distributions rather than the mixture.

$$p(o_t|s_t) = \prod_{i=1}^{n} p(o_t|s_t, i)^{p(i)}. \tag{2.29}$$

However this approximation does not arise from (2.27), but from assuming a factorisation of the observations

$$p(o_t|s_t) = \prod_{i=1}^{n} p(o_t^i|s_t, i), \tag{2.30}$$

which is useful when there is a different model for different parts of the observation vector.

Multi-stream techniques are hardly limited to the above. For example Misra et al. (2003) describes a system where $p(i)$ is related to the entropy of each sub-model, while Ketabdar et al. (2005a) describes a multi-stream method utilising state posteriors. In this thesis, however, we will concentrate on the two techniques outlined above and an additional one, which will be described in Section 3.4.

Part I

Ensembles for speech recognition

Chapter 3

Phoneme mixtures

In this chapter the application of ensemble methods to hidden Markov models (HMMs) for speech recognition is examined. We mainly consider two methods: bagging and boosting, with emphasis on the latter. Both methods feature a fixed mixing distribution between the mixture components, which simplifies the inference, though it does not completely trivialise it.

In the speech model considered, words are hidden Markov models composed of concatenations of phonetic hidden Markov models, where the state transitions are such that there exist no cycles other than self-transitions. In this setting it is possible to employ mixture models at any temporal level - i.e. a word can be represented by a mixture of word models and a phoneme can be represented by a mixture of phoneme models.

If mixtures at the phoneme model level are considered and data with a correct phonetic segmentation is available, then it is possible to restrict ourselves to a simple sequence classification problem in order to train a mixture model. Application of methods such as bagging and boosting to the phoneme classification task is straightforward. However, using the resulting models for continuous speech recognition poses some difficulties in terms of complexity. Section 3.4 outlines how multi-stream decoding (see Section 2.3.2) can be used to perform approximate inference in the resulting mixture model.

Chapter 4 introduces an algorithm for word error rate minimisation using boosting techniques. While it appears trivial to do so by minimising some form of loss based on the word error rate, in practice successful application additionally requires use of a probabilistic model for inferring error probabilities in parts of misclassified sequences. The concepts of expected label and expected loss are introduced, of which the latter is used in place of the conventional loss. This integration of probabilistic models with boosting allows its use in problems where labels are not available.

3.1 Prior research

The original Ada-Boost algorithm had been defined for classification and regression tasks, with the regression case receiving more attention recently (see (Meir and Rätch, 2003) for an overview). However, the amount of research in the application of boosting to sequence learning has been comparatively small.

This chapter presents methods and results for the use of boosting and bagging for the static phoneme classification. In this case the formulation of the task is essentially the same as that of static classification; the only difference being that the observations are sequences rather than single values.

Perhaps the closest approach to the one described herein was by Schwenk (1999), where boosting was used in a speech recognition task. An HMM/ANN system was used, with the ANNs used to compute the posterior phoneme probabilities at each state. Boosting itself was performed at the ANN level, using AdaBoost with confidence-rated predictions and in which the sample loss function was the frame error rate. The resulting decoder system differed from a normal HMM/ANN hybrid in that each ANN was replaced by a mixture of ANNs that had been provided via boosting. Thus such a technique avoids the difficulties of performing inference on mixtures, since the mixtures only model instantaneous distributions. Zweig and Padmanabhan (2000) appear to be using a similar technique, though the details of their approach are not evident, but with a system based on Gaussian mixtures and they additionally describe a few boosting variants for large-scale systems with thousands of phonetic units. Both papers report mild improvements in recognition. In the work presented here, we are interested in seeing whether models that result from phoneme-level boosting offer an advantage over a frame-level boosting approach.

Another interesting way to apply boosting would be to use it at the sentence level, for the purposes of explicitly minimising the word error rate. A proposed scheme for word error minimisation and related work in utterance-level boosting will be described in Section 4.1.

3.2 Data and methods

The phoneme data was based on a pre-segmented version of the OGI Numbers 95 (N95) data set (Cole et al., 1995). This data set was converted from the original raw audio data into a set of features based on Mel-Frequency Cepstrum Coefficients (MFCC) (Rabiner and Juang, 1993) (with 39 components, consisting of three groups of 13 coefficients, namely the static coefficients and their first and second derivatives) that were extracted from each frame. The data contains 27 distinct phonemes that compose 30 dictionary words. There are of 3233 training utterances and 1206 test utterances, containing 12510 and 4670 words respectively. The segmentation of the utterances into their constituent phonemes resulted in 35562 training segments and 12613 test segments, totalling 486537 training frames and 180349 test frames respectively. The feature extraction and phonetic labelling is described in more detail in (Johnny Mariéthoz and Samy Bengio, 2004).

The models employed have a number of hyper-parameters. For model comparison in the speech recognition task, these were tuned by holding out 1233 utterances from the training set and performing training on the remaining 2000 utterances. Subsequently, the hyper-parameters that resulted in the best performance in terms of word error rate were chosen for each model and then that model was trained on the complete training set. Finally, the models were compared on the independent test set. Results in figures and tables explicitly indicate whether they are reported on either the complete training set, the holdout set, or the independent test set and how the selection of hyper-parameters was performed.

The comparative performance measure used depends on the task. For the phoneme classification task, the *classification error* is used, which is the percentage of misclassified examples in the training or testing data set. For the speech recognition task, the *word error rate* (2.26) is used. Confidence values given for the classification error are based on a two-proportion z-test, assuming they are sampled from normal distributions with equal variances, a reasonable assumption for large sample sizes, while for the word error rate a bootstrap estimate is used to estimate the statistical significance of the results. The tests are described in more detail in Appendix B.4.

For the classification task, we used pre-segmented data. Thus, the classification could be performed using a Bayes classifier composed of 27 Hidden Markov Models, each one corresponding to one class. Each HMM was composed of three [1] states [2], (though for one of the experiments, only one hidden state was used) in a left-to-right topology and the distributions corresponding to each state were modelled with a Gaussian mixture model, with each Gaussian having a diagonal covariance matrix. For the initial results presented, ten Gaussian components are used, while for the final results the number of components is chosen from $\{10, 20, 30, 40\}$ by evaluating the performance on the hold-out set. In all cases, the diagonal covariance matrix elements of each Gaussian were clamped to a lower limit of 0.2 times the global variance of the data. For continuous speech recognition, transitions between word models incurred an additional likelihood penalty of $\exp(-15)$ while calculating the most likely sequence of states. Finally, in all continuous speech recognition tasks, state sequences were constrained to remain in the same phoneme for at least three acoustic frames.

For phoneme-level training, the adaptation of each phoneme model was performed in two steps. Firstly, the acoustic frames belonging to each phonetic segment were split into a number of equally-sized intervals, where the number of intervals was equal to the number of states in the phonetic model. The Gaussian mixture components corresponding to the data for each interval were initialised via 25 iterations of the K-means algorithm (see, for example (Bishop, 1995)). After this initialisation was performed, a maximum of 25 iterations of the EM algorithm were run on each model, with optimisation stopping earlier if at any point in time t, the likelihood ℓ_t satisfied the following stopping criterion:

$$\frac{\ell_t - \ell_{t-1}}{\ell_t} < \varepsilon, \quad \varepsilon > 0 \tag{3.1}$$

with $\varepsilon = 10^{-5}$ being used in all experiments that employed EM for optimisation.

For utterance-level training, the same initialisation was performed. The inference of the final model was done through expectation maximisation (using the Viterbi approximation) on concatenated phonetic models representing utterances. Note that performing the full EM computation is costlier and does not result in significantly better generalisation performance, at least in this case. The stopping criterion and maximum iterations were the same as those used for phoneme-level training.

[1] The optimal number of hidden states to use will vary depending on the data. It will predominantly depend on the minimum duration of each phoneme in terms of the number of observation frames that it occupies with the selected set of features. For this particular dataset, the performance of a single HMM was found to be optimal on a hold-out set when the number of states was 3. This value was maintained for all subsequent experiments.

[2] and an additional two non-emitting states: the initial and final states

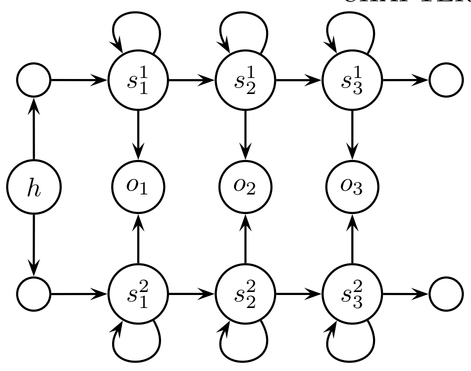


Figure 3.1: A phoneme mixture model. The generating model depends on the hidden variable h, which determines the mixing coefficients between model 1 and 2. The random variable h may in general depend on other variables. The distribution of the observation is a mixture between the two distributions predicted by the two hidden models, mixed according to the mixture model h.

3.3 Model training at the phoneme level

The simplest way to apply ensemble training techniques such as bagging and boosting to HMM training is to cast the problem into the classification framework. This is possible at the phoneme classification level, where each class $c \in \mathcal{C}$ corresponds to one of the possible phonemes. As long as the available data are annotated in time so that subsequences containing single phoneme data can be extracted, it is natural to adapt each hidden Markov model m_c to a single class c out of the possible $|\mathcal{C}|$, and combine the models into a Bayes classifier in the manner described in Section 2.1.1. Such a Bayes classifier can then be used as an expert in the Ada-Boost framework.

More specifically, each example d in the training dataset $\mathcal{D}$ will be a sequence segment corresponding to data from a single phoneme $c \in \mathcal{C}$. So each example d would be of the form $d = (s, c)$, with $s \in S^*$ being a subsequence of features corresponding to single phoneme data. At each iteration j of both boosting and bagging, a new classifier h_j is created, which consists of a set of hidden Markov models $\{m^j_1, m^j_2, ..., m^j_{|\mathcal{C}|}\}$. Each model m^j_i is adapted to the set of examples $\{d_k \in \mathcal{D}_j | y_k = c_i\}$, where $\mathcal{D}_j$ is a bootstrap replicate of $\mathcal{D}$, sampled uniformly in the case of bagging and according to (2.14) in the case of boosting. The $p(h_i)$ for the mixture components is given by the uniform distribution and (2.12) respectively. In this case the Ada-Boost method used was Ada-Boost.M1, though, since the number of classes is relatively small, other variants for multi-class classification, such as Ada-Boost.M2, could have been used instead.

Since previous studies had shown that an increase in generalisation performance may be obtained through the use of those two ensemble methods, it was expected that they would have a similar effect on performance in phoneme classification tasks. This is tested in Section 3.5. While using the resulting phoneme classification models for continuous speech recognition is not straightforward, we describe some techniques for combining the ensembles resulting from this training in order to perform sequence recognition in Section 3.4.

3.4 Continuous speech recognition with mixtures

The approach described in the previous section is only suitable for phoneme classification, which requires that the data is segmented at the phoneme level both during training and testing. However we can still employ boosting by training with segmented data to produce a number of expert models which can then be recombined during decoding on unsegmented data.

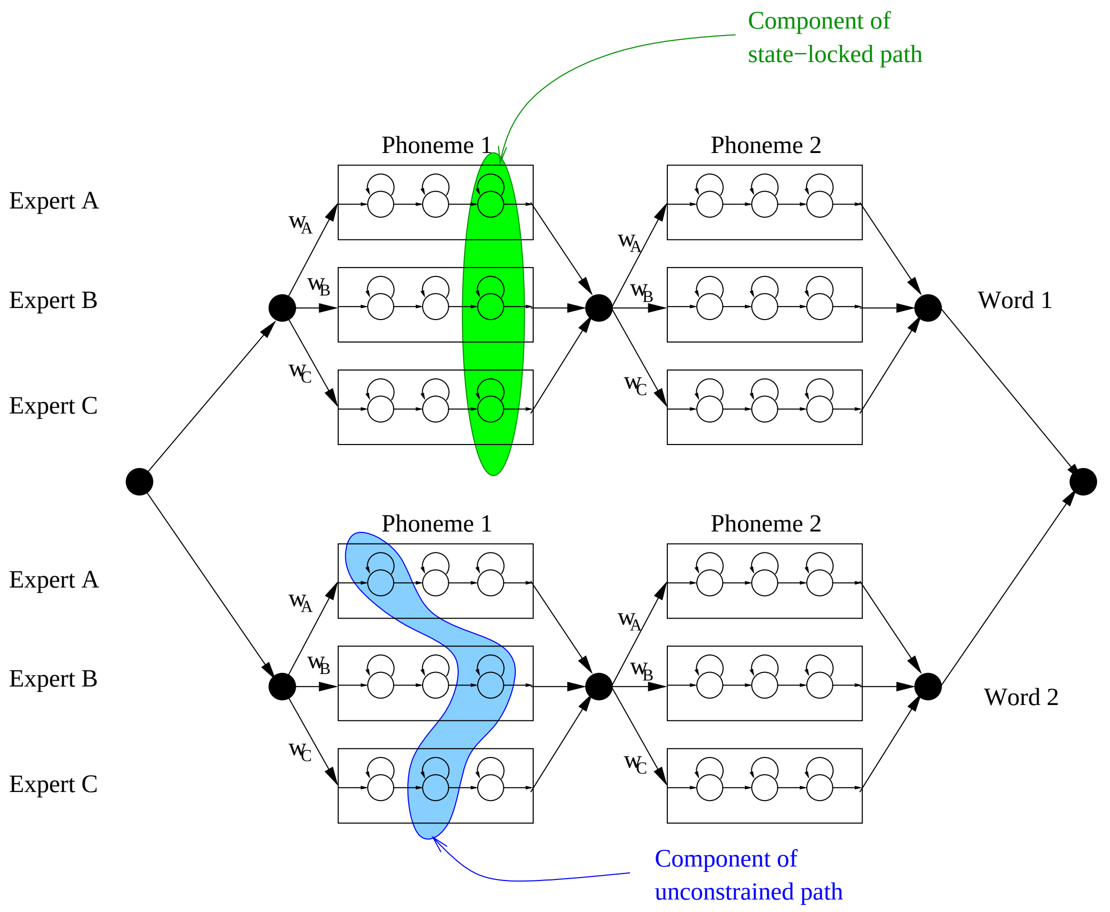


Figure 3.2: Single-path multi-stream decoding for two vocabulary words consisting of two phonemes each. When there is only one expert the decoding process is done normally. In the multiple expert case, phoneme models from each expert are connected in parallel. The transition probabilities leading from the anchor states to the Hidden Markov Model corresponding to each experts are calculated from the expert weights β_i, from equation (2.12), of each expert.

The first technique employed for sequence decoding uses an HMM comprising all phoneme models created during the boosting process, connected in the manner shown in Figure 3.2. Each phase of the boosting process creates a sub-model i, which we will refer to as *expert* for disambiguation purposes. Each expert is a classification model that employs one hidden Markov model for each phoneme. For some sequence of observations, each expert calculates the posterior probability of each phonetic class given the observation and its model. Two types of techniques are considered for employing the models for inferring a sequence of words.

In the *single stream* case, decoding is performed using the Viterbi algorithm in order to find a sequence of states maximising the posterior probability of the sequence. A normal hidden Markov model is constructed in the way shown in Figure 3.2, with each phoneme being modelled as a mixture of expert models. In this case we are trying to find the sequence of states $\{s_t = s_i^j\}$ with maximum likelihood. The transition probabilities leading from anchor states (black circles in the figure) to each model are calculated from the boosting weights β_i so that they sum to one and represent the confidence weight of each expert according to:

$$w_i = \frac{\beta_i}{\sum_j^{n_e} \beta_j}. \tag{3.2}$$

This type of decoding would have been appropriate if the original mixture had been inferred as a type of switching model, where only one sub-model is responsible for generating the data at each point in time and where switching between models can occur at anchor states.

The models may also be combined using *multi-stream* decoding (see section 2.3.2. In this case we wish to find the sequence of *combinations* of states across expert with the highest likelihood. More formally, a combination of of n experts, where expert had a state space $\mathcal{S}$, would have the state space $\mathcal{S}^n$. However, state combinations and their transitions are typically constrained by forcing some of them to have zero probability. For example a combination whereby expert A is in a state corresponding to phoneme 1 while expert B is in a state corresponding to phoneme 2 would not be allowed.

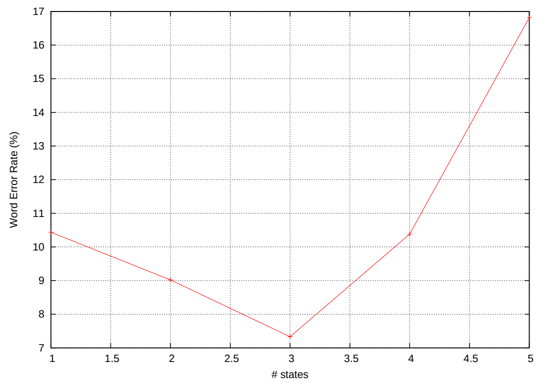
The advantage of such a method is that it uses information from all models. The disadvantage is that there are simply too many states to be considered. In order to simplify this, we consider multi-stream decoding synchronised at the state level, i.e. with the constraint that $P(s_t^i \neq s_t^j) = 0$. This corresponds to equation (2.27), where the stream weights are given by (3.2). Experiments on the hold-out set with boosting and bagging, shown in Figure 3.5, demonstrate that this should be the preferred decoding technique among the three for models where the class probability is modelled as a mixture, which is the expected result (see Appendix B.1).

3.5 Experiments

Since the available data includes segmentation information it makes sense to first limit the task to training for phoneme classification. This enables the direct application of ensemble training algorithms for this task by simply using each training segment as a training example.

Two methods were examined for this task: bagging and boosting. At each iteration of either method, a sample from the training set was made according to the distribution defined by either algorithm and then a Bayes classifier composed of N hidden Markov models, one for each phonetic class, was trained.

It then becomes possible to apply the boosting and bagging algorithms by using Bayes Classifiers as the experts. The N95 data was pre-segmented into training examples, so that each one was a segment containing a single phoneme. Thus, bootstrapping was performed by sampling through these examples. Furthermore, the classification error of each classifier is used to calculate the weights necessary for the



(a)

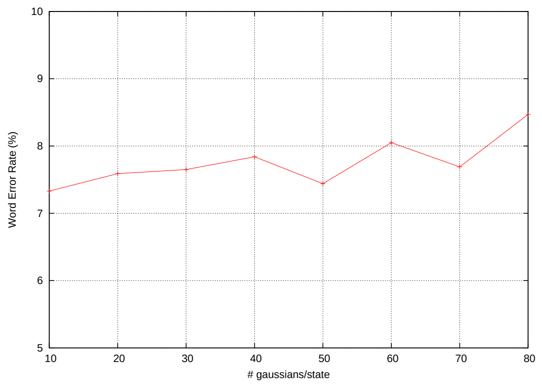
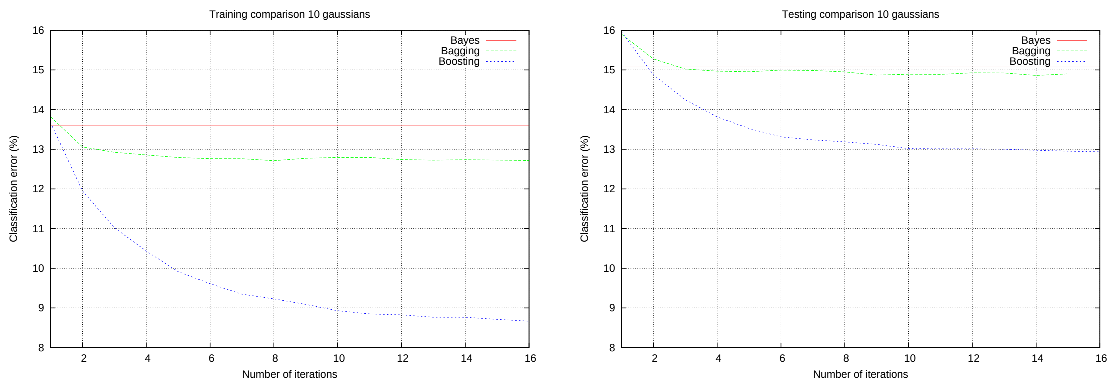


Figure 3.3: In the experiments reported in this chapter, the number of states and number of Gaussian mixtures per state were tuned on a hold-out set prior to the analysis. Figure 3.3(a) displays the word error rate performance of an HMM with 10 Gaussians per state when the number of emitting states per phoneme is varied, which varies rather dramatically with the number of states. Figure 3.3(b) displays the word error rate performance of an HMM with 3 emitting states as number of Gaussians per state varies. In this case, the effect on generalisation is markedly lower.

weighted voting mechanism. The data that was used for testing was also segmented to subsequences consisting of single phoneme data, so that the models could be tested on the phoneme classification tasks. The results in training and test sets, shown in Figures 3.4(a) and 3.4(b), were validated against the performance of a single Bayes classifier that was trained on the complete data set.

As can be seen in Figure 3.4(a), both bagging and boosting manage to reduce the phoneme classification error considerably in the training, with boosting continuing to make improvements until the maximum number of iterations. For bagging, the improvement in classification was limited to the first 4 iterations, after which performance remained constant. The situation was similar when comparing the models in the independent test set (Figure 3.4(b)).

Finally, a comparison between the models on the task of continuous speech recognition was made. Firstly, it was necessary to decide on a method for performing decoding when dealing with multiple models. The three relatively simple methods of single stream and multi-stream decoding (the latter employing either weighted product or weighted sum) were evaluated by using approximately one third of the training set as a hold out set. As can be seen in Figure 3.5, the weighted sum method that it is the only one among the three techniques that could consistently offer improvement. This was expected since it was the only method with some justification in our particular case, as it arises out of constraining the full state inference problem on the mixture. The multi-stream product method would have been justified if (2.30) held, which was not the case here, since each model had exactly the same observation variables. The single-stream model could perhaps be justified under the assumption of a switching model, where a different expert can be responsible for the observations in each phoneme. That might explain the fact that that its performance is not degrading in the case of bagging, as the components of each mixture should be quite similar to each other, something which is definitely not the case with boosting where each



(a) Training (b) Testing

Figure 3.4: Classification errors for a bagged and a boosted ensemble of Bayes Classifiers as the number of experts is increased at each iteration of boosting. For reference, the corresponding errors for a single Bayes Classifier trained on the complete training set are also included. There were 10 Gaussians per state and 3 states per phoneme for all models. Results are shown in the full training dataset.

model is trained on a different distribution of the data.

After having performed this initial analysis, the models that had been trained on the full training set were used on the test set with a set of data. The comparison shown in Figure 3.6 was performed using the same type of models for both boosting and bagging. It can be seen that while boosting manages to continuously increase performance both in terms of classification error and word error rate, the bagging approach continues to improve upon the word error rate even after it has stopped improving upon the classification error, to the extend that in the end it actually matches boosting. A fuller comparison between the two methods will be given in the next chapter, where the number of Gaussian units per state and the number of experts will be tuned on a hold-out set. Furthermore, an alternative boosting method specifically for minimising the word error rate will be explored.

3.6 Discussion

The experimental results indicate that boosting at the phoneme level can significantly improve phoneme classification performance, while also increasing the performance for sentence recognition. Perhaps surprisingly, bagging has a similar effect in sentence recognition for a much smaller effect in classification. While after training the models there is only one way to combine them in order to perform phoneme classification, the task of sentence recognition does not present an obvious (optimal, yet tractable) method for the combination of models. Indeed, the performance varies significantly depending on the combination method used, of which the multi-stream weighted sum method, arguably the most suitable method of the three for mixture models. An interesting subject of further research would be to pursue better methods for expert combination during decoding, as an alternative to the current multi-stream methods, such as perhaps a Monte Carlo approximation to the full posterior estimation of word sequences.

It is interesting to contrast the bagging and boosting methods for creating the mixture components.

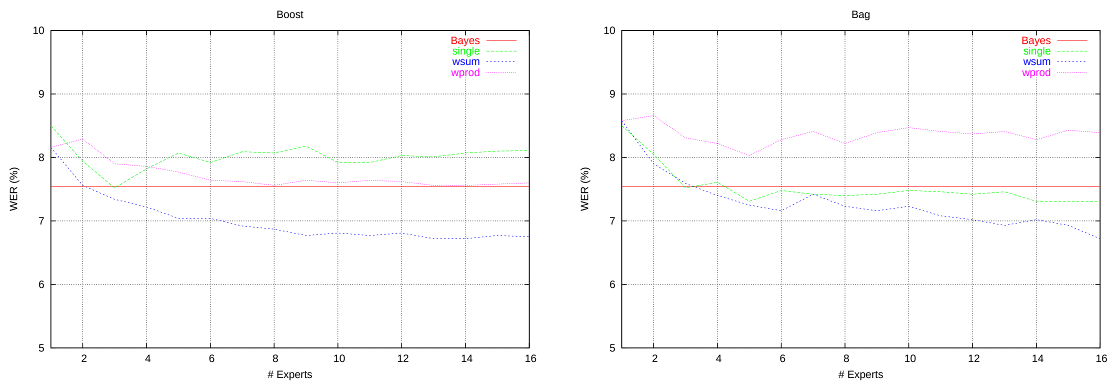


Figure 3.5: Generalisation performance on the hold-out set in terms of word error rate after training with segmentation information. Results are shown for both boosting and bagging, using three different methods for decoding. Single-path and multi-stream. Results are shown for three different methods single-stream (**single**), and state-locked multi-stream using either a weighted product (**wprod**) or weighted sum (**wsum**) combination.

While boosting performs much better than bagging in the phoneme classification task, this improvement is not so evident in the continuous speech recognition task. This is made clearer in Figure 3.6, where it can be seen that bagging continues to degrease the word error rate while the classification error remains approximately the same, though in any case the differences are not great. Taking this under consideration suggests that it might be advantageous to perform boosting in order to minimise the word error rate directly. Such an approach is examined in the subsequent chapter.

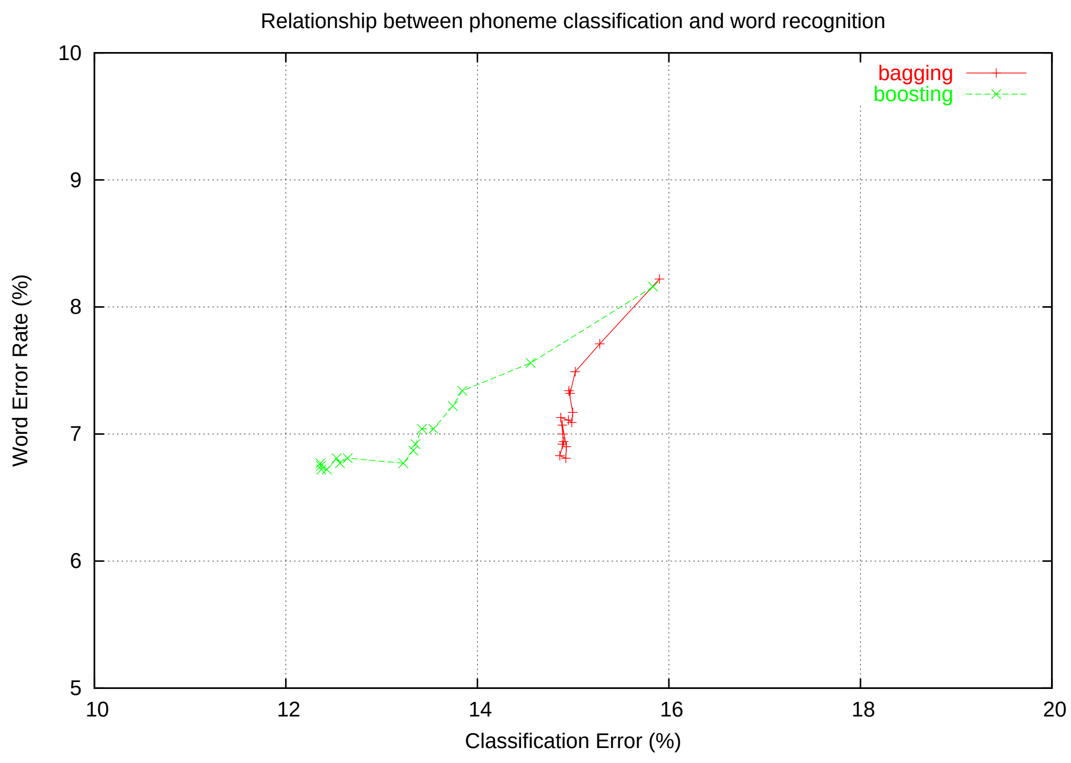


Figure 3.6: Relationship between phoneme classification errors and word recognition errors in testing for two different ensemble methods using 10 Gaussian components per mixture.

Chapter 4

Expectation boosting

It is also possible to apply ensemble training techniques at the utterance level. As before, the basic models used are HMMs that employ Gaussian mixtures to represent the state observation distributions. Attention is restricted to boosting algorithms in this case. In particular, we shall develop a method that uses boosting to simultaneously utilise information about the complete utterance, together with an estimate about the phonetic segmentation. Since this estimate will be derived from bootstrapping our own model, it is unreliable. The method that is developed in this chapter will take into account this uncertainty.

More specifically, similarly to (Cook and Robinson, 1996), sentence-level labels (sequences of words without time indications) are used to define the error measure that we wish to minimise. The measure used is related to the word error rate, as defined in (2.26). In addition to a loss function at the sentence level, a probabilistic model is used to define a distribution for the loss at the frame level. Combined, the two can be used for the greedy selection of the next base hypothesis. This is further discussed in the following section.

4.1 Boosting for word error rate minimisation

In the previous chapter (and (Dimitrakakis and Bengio, 2004b)) we have applied boosting to speech recognition at the phoneme level. In that framework, the aim was to reduce the *phoneme classification error* in pre-segmented examples. The resulting boosted phoneme models were combined into a single speech recognition model using *multi-stream* techniques. It was hoped that we could reduce the word error rate as a side-effect of performing better phoneme classification and three different approaches were examined for combining the models in order to perform continuous speech recognition. However, since the measure that we are trying to improve is the word error rate and since we did not want to rely on the existence of segmentation information, minimising the word error rate directly would be desirable. This chapter describes such a scheme using boosting techniques.

Previous approaches for the reduction of word error rate include (Bahl et al., 1988), which employed a “corrective training scheme” and an approach that also used boosting (Cook and Robinson, 1996). In the latter, the authors employed a boosting scheme where the sentences with the highest error rate were classified as 'incorrect' and the rest 'correct', irrespective of the absolute word error rate of each sentences. The weights of all frames constituting a sentence were adjusted equally and boosting was applied at the frame level. This however does not manage to produce as good results as the other schemes described by the authors. In our view, which is partially supported by the experimental results, this could have been partially due to the lack of a temporal credit assignment mechanism such as the one we present in this chapter.

In other work on utterance-level boosting, Zhang and Rudnicky (2003) compares use of the posterior probability of each possible utterance for adjusting the weights of each utterance with a "non-boosting" where the same weights are adjusted according to some function of the word error rate. In either case, utterance posterior probabilities are used for recombining the experts. Since the number of possible utterances is infinite, not all possible utterances are used, but an N-best list. For recombination, the authors consider two methods: Firstly, choosing the utterance with maximal sum of weighted posterior (where the weights have been determined by boosting). Secondly, they consider combining via ROVER, a dynamic programming method for combining multiple speech recognisers (see Fiscus, 1997). Since the authors' use of ROVER entails using just one hypothesis from each expert to perform the combination, in (Zhang and Rudnicky, 2004b) they consider a scheme where the N-best hypotheses are reordered according to their *estimated* word error rate. Finally, in (Zhang and Rudnicky, 2004a) they consider a scheme similar to the one proposed herein for assigning weights to frames, rather than just to complete sentences. More specifically, they use the currently estimated model to obtain the probability that the correct word has been decoded at any particular time, i.e. the posterior probability that the the word at time t is a_t given the model and the sequence of observations. In our case we use a slightly different formalism in that we calculate the expectation of the loss according to an independent model.

In terms of model recombination, possibly the work closest to the one presented here is that of Meyer and Schramm (2006). While the scheme is not exactly the same as multi-stream weighted sum decoding, it is nevertheless performs very similar inference. The authors there also employ Ada-Boost.M2, utilising the posterior probability of each utterance rather than a WER-based loss function. Use of M2 is not particularly straightforward, since the algorithm requires calculating the posterior of every possible class (in this case an utterance) given the data. The required calculation however can be approximated by calculating the posterior only for the subset of the top N utterances and assuming the rest are zero.

In this chapter we describe a new training method (introduced in (Dimitrakakis and Bengio, 2005a), specific to boosting and hidden Markov models (HMMs), for word error rate reduction. We employ a score that is exponentially related to the word error rate of a sentence example. The weights of the frames constituting a sentence are adjusted depending on our expectation of how much they contribute to the error. Finally, boosting is applied at the sentence and frame level simultaneously. This method has arisen from a two-fold consideration: firstly, we need to have an accurate measure of performance, which is the word error rate. Secondly, we need a way to more exactly specify which parts of an example most probably have contributed to errors in the final decision. Using boosting it is possible to focus training on parts of the data which are most likely to give rise to errors, while at the same time doing it in such a manner as to increase an accurate measure of performance. We find that both aspects of training have an important effect.

The remainder of this chapter is organised as follows: Section 4.1.1 describes word error rate-related loss functions that can be used for boosting. Section 4.1.2 introduces the concept of *expected error*, for the case when no labels are given for the examples. This is important for the task of word error rate minimisation. Previous sections on HMMs and multi-stream decoding, describe how the boosted models are combined for performing the speech recognition task. Experimental results are outlined in section 4.1.3. The chapter concludes with an experimental comparison between different methods in Section 4.2, followed by a discussion.

4.1.1 Sentence loss function

A commonly used measure of optimality for speech recognition tasks is the word error rate (2.26). We would like to minimise this quantity using boosting techniques. In order to do this, a dataset is considered where each example is a complete sentence and where the *loss* $l(d)$ for each example d is given by some function of the word error rate for the sentence.

The word error rate for any particular sentence can take values in $[0, \infty)$, while the Ada-Boost algorithm that is employed herein requires a sample loss function with range $[-1, 1]$. For this reason we employ the *ad hoc*, but reasonable, mapping $l : [0, \infty) \to (-1, 1]$

$$l(x) = 1 - 2e^{-\eta x}, \tag{4.1}$$

where x is the word error rate. When $l(x) = -1$, an example is considered as classified correctly and when $l(x) = 1$, the example is considered to be classified incorrectly. This mapping includes a free parameter $\eta > 0$. Increasing the parameter η increases the sharpness of the transition, as shown in Figure 4.1. This function is used for $l(\cdot)$ in equation (2.14).

While this scheme may well result in some improvement in word recognition with boosting, while avoiding relying on potentially erroneous phonetic labels, there is some information that is not utilised. Knowledge of the required sequence of words, together with the obtained sequence of words for each decoded sentence results in a set of errors that are fairly localised in time. The following sections discuss how it is possible to use a model that capitalises on such knowledge in order to define a distribution of errors over time.

4.1.2 Error expectation for boosting

In traditional supervised settings we are provided with a set of examples and labels, which constitute our training set, and thus it is possible to apply algorithms such as Boosting. However this becomes problematic when labels are noisy (see for example Raetsch et al., 2001). Such an example is a typical speech recognition data set. Most of the time such a data set is composed of a set of sentences, with a corresponding set of transcriptions. However, while the transcriptions may be accurate as far as the intention of the speakers or the hearing of the transcriber is concerned, subsequent translation of the transcription into phonetic labels is bound to be error prone, as it is quite possible for either the speaker

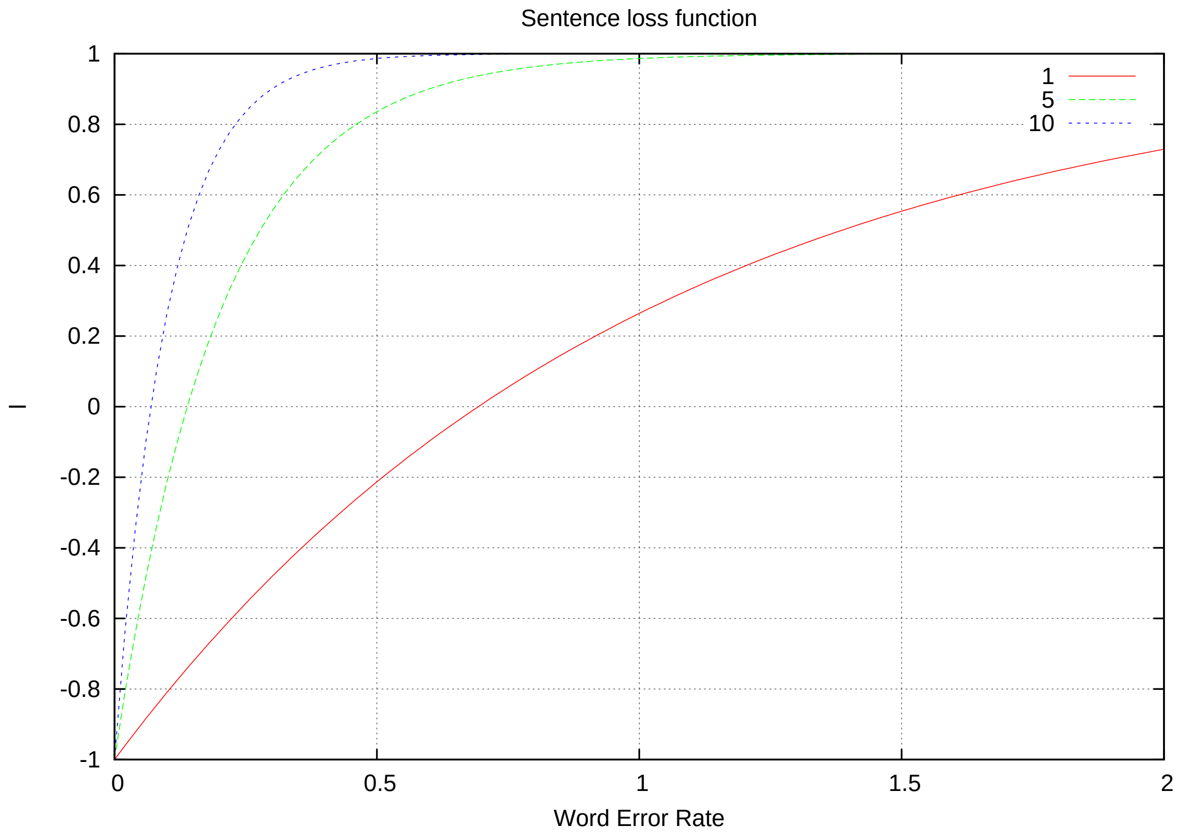


Figure 4.1: The sentence loss function (4.1) for $\eta \in \{1, 5, 10\}$.

to mispronounce words, or for the model that performs the automatic segmentation to make mistakes. In such a situation, adapting a model so that it minimises the errors made on the segmented transcriptions might not automatically lead into a model that minimises the word error rate, which is the real goal of a speech recognition system.

For this purpose, the concept of error expectation is introduced. Thus, rather than declaring with absolute certainty that an example is incorrect or not, we simply define $l(d_i) = P(y_i \neq h_i)$, so that the sample loss is now the probability that a mistake was made on example i and we consider y_i to be a random variable. Since boosting can admit any sample loss function(Freund and Schapire, 1997), this is perfectly reasonable and it's possible to use this loss as a sample loss in a boosting context. The following section discusses some cases for the distribution of y in the following section which are of relevance to the problem of speech recognition.

Error distributions in sequential decision making

In sequential decision making problems the knowledge about the correctness of decisions is delayed. Furthermore, it frequently lacks detailed information concerning the temporal location of errors. A common such case is knowing that we have made one or more errors in the time interval $[1, T]$. This form occurs in a number of settings. In the setting of individual sentence recognition a sequence of decisions is made which corresponds to an inferred utterance. When this is incorrect, there is little information to indicate where mistakes were made. This difficulty is even more pronounced in episodic reinforcement learning tasks (see (Sutton and Barto, 1998) for an overview), where in some settings no information may be given as to the correctness of behaviour apart from a single scalar evaluation at the end of the episode.

In such cases it is necessary to define a model [1] for the probability of having made an erroneous decision at different points in times t, given that there has been at least one error in the interval $[1, T]$. Let us denote the probability of having made an error at time $t \in [1, T]$,as $P(y_t \neq h_t | y_1^T \neq h_1^T)$. A trivial example of such a model is to assume that the error probability is uniformly distributed. This can be expressed via the flat prior

$$P(y_t \neq h_t | y_1^T \neq h_1^T) \propto 1/T \tag{4.2}$$

Another useful model is to assume an exponential prior such that

$$P(y_t \neq h_t | y_1^T \neq h_1^T) \propto \lambda^{t-T}, \quad \lambda \in [0, 1), \tag{4.3}$$

such that the expectation of an error near the end of the decision sequence is much higher. This is useful in tasks where it is expected that the decision error will be temporally close to the information that an error has been made. For example, if you crash while driving your car, you may assume that this was the result of a bad decision in the last few seconds, though ultimately it might have been the result of something occurring much earlier, such as having too much to drink or driving with worn-out tyres - or it could be a combination of all of these. Ultimately, such models incorporate very little knowledge about the task, apart from this simple temporal structure.

In this case we focus on the application of speech recognition, which has some special characteristics that can be used to more accurately estimate possible locations of errors. For the case of labelled sentence examples it is possible to have a procedure that can infer the location of an error in time. This is because correctly recognised words offer an indication of where possible errors lie. Assume some procedure that creates an indicator function I_t such that $I_t = 1$ for instances in time where an error could have been made. We can then estimate the probability of having an error at time t as follows:

$$P(y_t \neq h_t | y_1^T \neq h_1^T) = \frac{\gamma^{I_t}}{\sum_{k=1}^T \gamma^{I_k}}, \tag{4.4}$$

where the parameter $\gamma \in [1, \infty)$ expresses our confidence in the accuracy of I_t. A value of 1 will cause the probability of an error to be the same for all moments in time, irrespective of the value of I_t, while when γ approaches infinity we have absolute confidence in the inferred locations. Similar relations can be defined for an exponential prior and they can be obtained through the convolution of (4.3) and (4.4).

In order to apply boosting to temporal data, where classification decisions are made at the end of each sequence, we use a set of weights $\{\psi_t\}$ corresponding to the set of frames in an example sentence.

[1]Even if no model is explicitly defined, there is always one implicit.

At each boosting iteration j the weights are adjusted through the use of (4.4), resulting in the following recursive relation:

$$\psi_t^{(j+1)} = \frac{\psi_t^{(j)} \gamma^{I_t}}{\sum_{k=1}^{T} \psi_k^{(j)} \gamma^{I_k}} \tag{4.5}$$

In this manner, the loss incurred by the whole sentence is distributed to its constituent frames, although the choice is rather ad-hoc. A different approach was investigated by Zhang and Rudnicky (2004a), where the loss on the frames was related to the probability of the relevant word being uttered at time t, but their results leaves things unclear as to whether this is a good choice compared to the simple utterance-level training scheme.

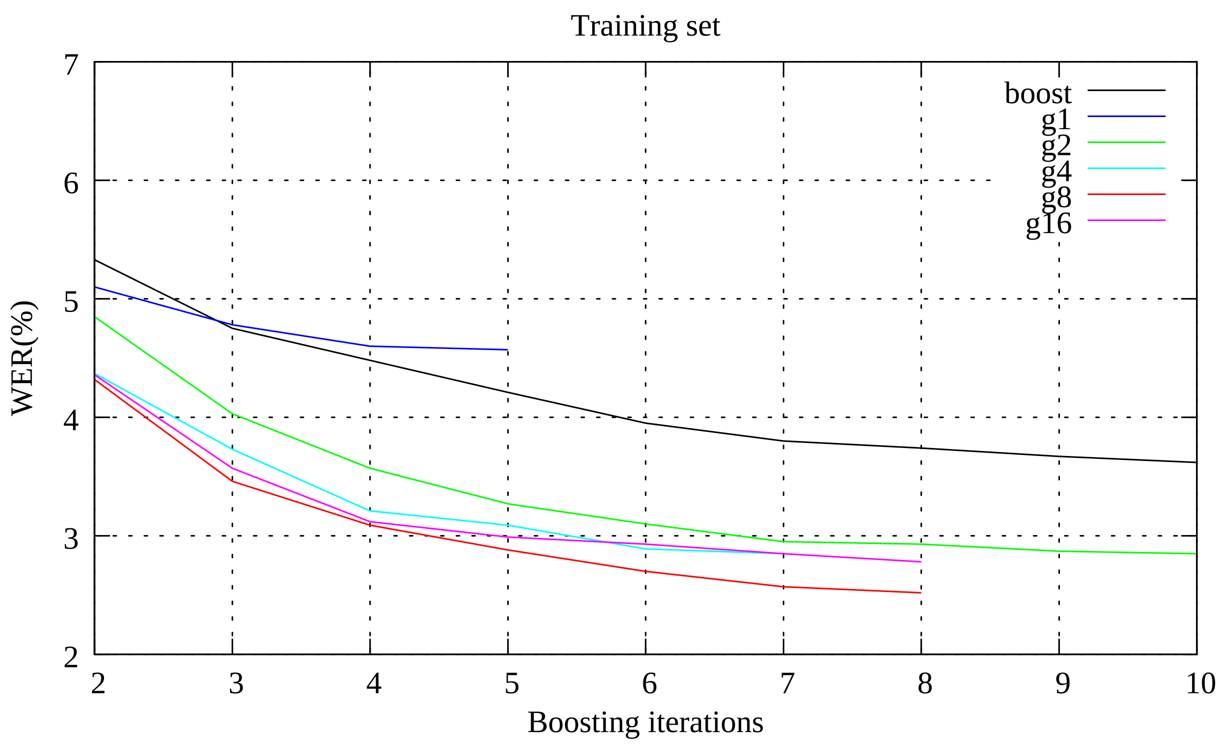


Figure 4.2: Training word error rates for various values of gamma, compared with the previous boosting approach.

4.1.3 Experimental results

We experimented on the OGI Numbers 95 (N95) data set (Cole et al., 1995) (details about the setup and dataset are given in Section 3.2). The experiment was performed as follows: firstly, a set of HMMs e_0, composed of one model per phoneme, was trained using the available phonetic labels. This has the role of a starting point for the subsequent expert models. At each boosting iteration t we take the following steps: firstly, we sample with replacement from the distribution of training sentences given by the AdaBoost algorithm. We create a new expert e_t, initialised with the parameters of e_0. The expert is trained

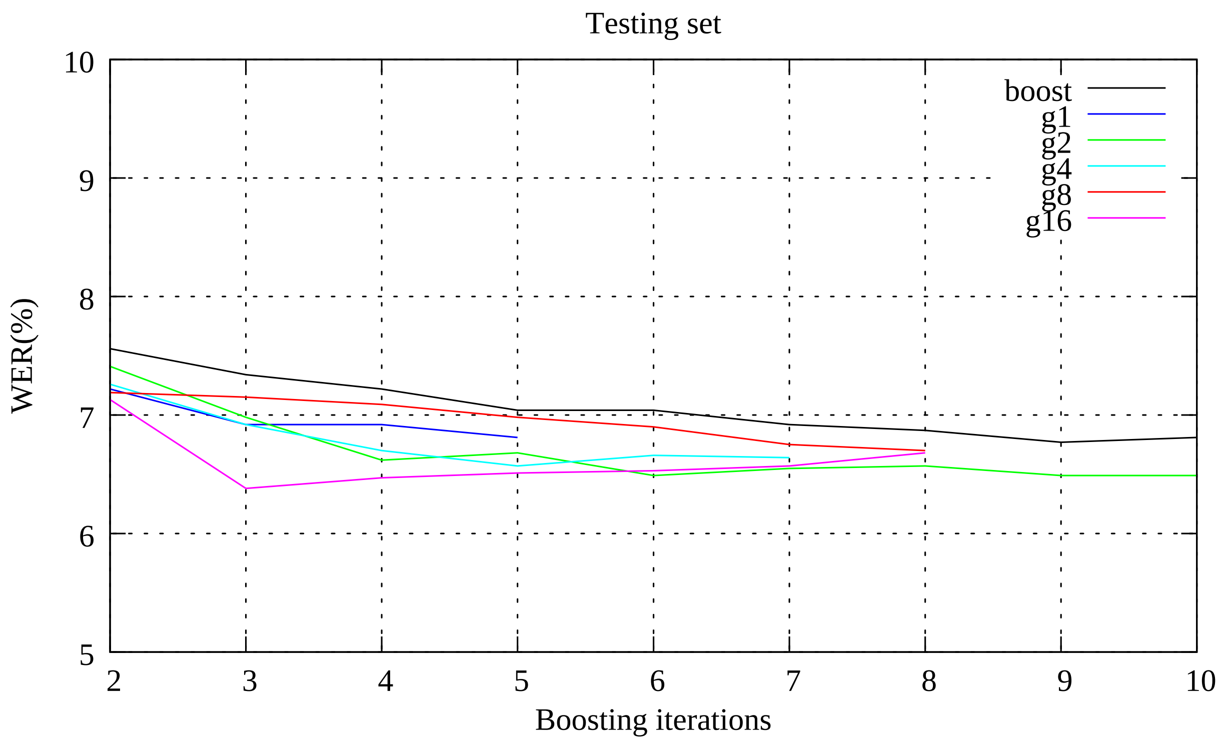


Figure 4.3: Test word error rates for various values of gamma, compared with the previous boosting approach.

on the sentence data using EM with the Viterbi approximation in the expectation step to calculate the expectation. The frames of each sequence carry an importance weight ψ_t, computed via (4.5), which is factored into the training algorithm by incorporating it in the posterior probability of the model h given the data x and time t, which, if we assume independence of x and t, can be written as

$$p(h|x,t) \propto p(h|x)p(h|t)p(h).$$

In this case, $p(h|t)$ will correspond to our importance weight ψ_t.

After training, all sequences are decoded with the new expert. The weights of each sentence is increased according to (4.1), with $\eta = 10$. This value was chosen so that any sentence decodings with more than 50% error rate would be considered nearly completely erroneous (see Figure 4.1). For each erroneously decoded sentence we calculate the edit distance using a shortest path algorithm. All frames for which the inferred state belonged to one of the words that corresponded to a substitution, insertion, or deletion are then marked. The weights of marked frames are adjusted according to (4.4). The parameter γ corresponds to how smooth we want the temporal credit assignment to be.

In order to evaluate the combined models we use the multi-stream method described in equation (2.28), where the weight of each stream is given by (3.2)

Experimental results comparing the performance of the above techniques to that of an HMM using segmentation information for training are shown in Figure 4.2, for the training data and figure 4.3 for the test data. The figures include results for our previous results with boosting at the phoneme level. We have included results for values of $\gamma \in \{1, 2, 4, 8, 16\}$. Although we do not improve significantly upon our previous work with respect to the generalisation error, we found that the convergence of boosting in this setting is significantly faster. On the training set, while boosting with pre-segmented phoneme examples had previously resulted in a reduction of the error to 3% after approximately 30 iterations (not shown), the sentence example training, combined with the error probability distribution over frames, converged to the same error after approximately 6 iterations. The situation was similar in testing, with the new approach converging to a good generalisation error at 10 iterations, while the previous approach reached requires 16 iterations to reach the same performance. A great drawback of the new approach, however, is the need for specifying two new hyper-parameters: (a) the shape of the cost function and (b) the shape of the expected error distribution As mentioned previously in the chapter, for a we are using (4.1) with $h = 10$ and for b we have chosen a mix between a uniform distribution and an indicator function, with γ being a free parameter. Choosing γ is not trivial (i.e. it cannot be chosen in the training set), since apparently large values can lead to overfitting, while values that are too small seem to provide no benefit. For the experiments described in Section 4.2 we will be holding out part of the training set in order to select an appropriate value for γ.

While the two boosting approaches are equivalent performance-wise respect, in our view the sentence training approach represents a more interesting alternative, for a number of reasons. Firstly, we are minimising the word error rate directly, which is a more principled approach since this is the real objective. Secondly, we don't in principle need to rely on segmentation information during training. Lastly, the temporal probability distribution, derived from the temporal structure of word errors and the state inference, provides us with a method to assign weights to parts of the decoded sequence. Its importance becomes obvious when we compare the performance of the method for various values of γ. When the distribution is flat (i.e. when $\gamma = 1$), the performance of the model drops significantly. This at least supports the idea of using a probabilistic model for the errors over training sentences.

4.2 Generalisation performance comparison

In a real-world application one would have to use the training set for selecting hyper-parameters to use in unknown data. To perform such a comparison, the training data set was split in two parts; holding out 1/3rd of it for validation. For each model, we used the number of Gaussians (selected from a possible set of values $\{10, 20, 30, 40, 50\}$, the number of experts and any other hyper-parameters, such as γ, giving the best results in the validation set to train a model on the full training set. This was then evaluated on the independent test set.

Table 4.2 summarises the results obtained, indicating the number of Gaussians per phoneme and the word error rate obtained for each model. If one considers only those models that were created strictly using the classification task, that is without adapting word models, ensemble methods perform

Model	Gaussians	Word error rate (%)
GMM	30	8.31
GMM embed	40	8.12
Boost GMM	10×30	7.41
HMM	10	7.52
HMM embed	10	7.04
Boost HMM	10×10	6.81
E-Boost HMM	7×10 (γ=8)	6.75
Bag HMM	16×20	5.97

Table 4.1: Test set performance comparison of models selected on a validation set. The second column indicates the number of Gaussians per phoneme. For ensemble methods, $n \times m$ denotes n models, each having m Gaussian components per state. *GMM* indicates a model consisting of a single Gaussian mixture for each phoneme. *HMM* indicates a model consisting of three Gaussian mixtures per phoneme. Thus the HMMs, the total number of Gaussians is three times that of the GMMs with an equal number of components per state. *Boost* and *Bag* models indicate models trained using the standard boosting and bagging algorithm respectively on the phoneme classification task, while *E-boost* indicates the expectation boosting algorithm for word error rate minimisation. Finally *embed* indicates that embedded training was performed subsequently to initialisation of the model.

significantly better (with more than 99% confidence[2]. Against the baseline *HMM embed* model, however, not all methods perform so well, as can be seen in Figure 4.2. In particular, the estimated probability that *Boost* is better than *HMM embed* is merely 51% and the difference in performance is just 0.23%, while against the simple *HMM* the result is statistically significant with a confidence of 91%. Slightly better performance is offered by *E-Boost*, with significance with respect to the *HMM* and *HMM embed* models at 98% and 65% respectively. Overall bagging works best, performing better than other methods with a confidence of at least 99% in all cases, while approximately 97.5% of the probability mass lies above the 0.5% differential word error rate when it is compared to the baseline model, as can be seen in Figure 4.4(a).

However these results are not quite near the state of the art on this database. Other researchers (Lathoud et al., 2005; Ketabdar et al., 2005b; Athineos et al., 2004; Hermansky and Sharma, 1998; Doss, 2005, for example), have achieved word error rates $5.0 \pm 0.3\%$, mainly through the use of different phonetic models. Accordingly, some preliminary experiments were performed with Markov models using a more complex phonetic model (composed of 80 tri-phones, i.e. phonemes with contextual information). A single such model achieved word error rates of $4.8 \pm 0.1\%$ (not shown in the table) which is in agreement with published state of the art results. This indicates that using a model that is closer to what we are trying to model could be better than using mixtures of simpler models. Further experiments to test whether even better results could be obtained by considering ensembles of tri-phone models indicated that the boosting-based approaches could not increase generalisation performance, achieving a word error rate of 5.1% at best, while the simpler bagging approach managed to reach a performance of 4.5%. Even though the reasons for this are not apparent, it is tempting to conclude that the label noise combined with

[2]The signifance was measured with a bootstrap estimate, which is described in Section B.4.2.

the variance-reducing properties of bagging are at least partially responsible for this success. Although it should be kept in mind that the aforementioned tri-phone results are merely are preliminary, they nevertheless indicate that in certain situations ensemble methods and especially bagging may be of some use to the speech recognition community.

4.3 Discussion

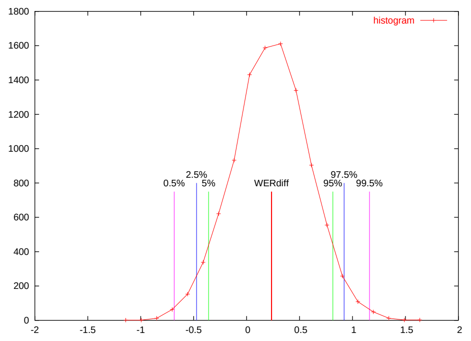
In this and the previous chapter we presented some techniques for the application of ensemble methods to HMMs. The ensemble training was performed for complete HMMs at either the phoneme or the utterance level, rather than at the frame level. Using boosting techniques at the utterance level was thought to lead to a method for reducing the word error rate. Interestingly, this word error rate reduction scheme did not improve generalisation performance for boosting, while the simplest approach of all, bagging, performed the best.

There are a number of probable causes. The first one is that the amount of data is sufficiently large for ensemble techniques to have little impact on performance, i.e. there is enough data to train sufficiently good base models. The second is that the state-locked multi-stream decoding techniques were investigated for model recombination lead to an increase in generalisation error as the inference performed is very approximate. The third is that the boosting approach used is simply inappropriate. The first case mustn't be true, since bagging does achieve considerable improvements over the other methods. There is some evidence for the second case, since the GMM ensembles are the only ones that should not be affected by the multi-stream approximations and while a more substantial performance difference can be observed, it nevertheless is not much greater. The fact that bagging's phoneme mixture components are all trained on samples from the same distribution of data and that it outperforms boosting is also in agreement with this hypothesis. This leaves the possibility that the type of boosting training used is inappropriate, at least in conjunction with the decoding method used, open.

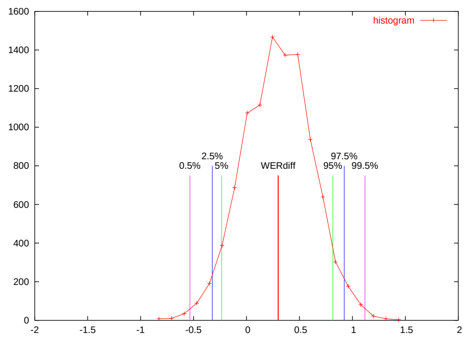
Future research in this direction might include the use of other approximations for decoding than constrained multi-stream methods. Such an approach was investigated by Meyer and Schramm (2006), where the authors additionally consider the harder problem of large vocabulary speech recognition (for which even inferring the most probable sequence of states in a single model may be computationally prohibitive). It could thus be also possible to use the methods developed herein for large vocabulary problems by borrowing some of their techniques. The first method, (also used by Zhang and Rudnicky, 2003),relies on finding an n-best list of possible utterances, assuming there are no other possible utterances and then fully estimating the posterior probability of the n alternatives. The second method is based upon a technique developed by Schramm and Aubert (2006) for combining multiple pronunciation models. In this case each model arising from boosting could be used in lieu of different pronunciation models. Another possible future direction is to consider different algorithms. Both Ada-Boost.M1, which was employed here, and Ada-Boost.M2, are using greedy optimisation for the mixture coefficients. Perhaps better optimisation procedures, such as those proposed by Mason et al. (2000), may have an advantage.

4.4 Afterword

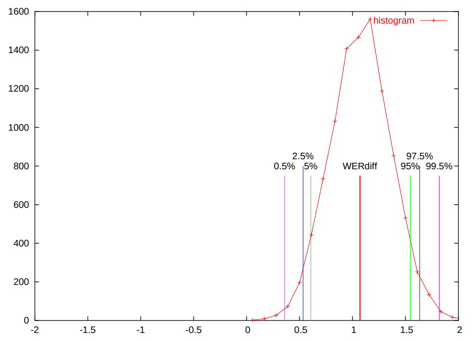
So far, even though we have seen that the application of ensemble methods can be extended in practice to sequential tasks, our view has been limited to supervised learning tasks. The tasks that we have examined, sequence (i.e. phoneme) classification and sequence (i.e. speech) recognition, are special cases of sequential decision making tasks, in the sense that a decision is made after each novel observation: in one case the decision is a class label, and in the other a sequence of words. These problems somewhat restricted because (a) there is a fixed set of data from which we can sample with virtually no cost and (b) the data contains directly gives the optimal mapping from observations to decisions. Because of (a), we do not need to be overly concerned with the requirements of estimation algorithms. We can simply make the most of the data that we have now, aiming to maximise performance later. Naturally, in some applications where data acquisition is expensive, or where the model might need to be updated as time passes, (a) is no longer true. Item (b) means that, for probabilistic modelling, simply the decision with the highest posterior probability given the observations needs to be taken. This only occurs because we implicitly assume a known fixed cost that is equal for all erroneous decisions. We shall explore these matters further in the second part of the thesis.



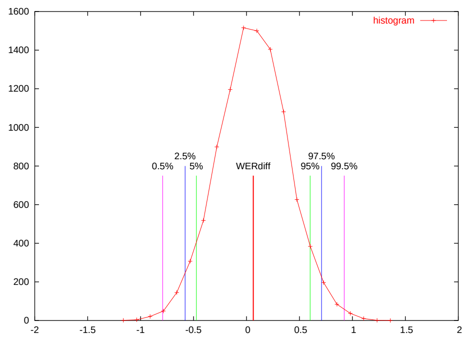
(a) Boost vs HMM embed



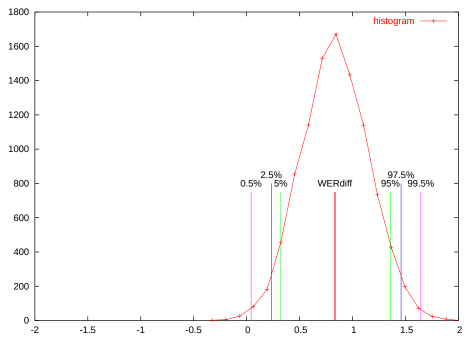
(b) E-Boost vs HMM embed



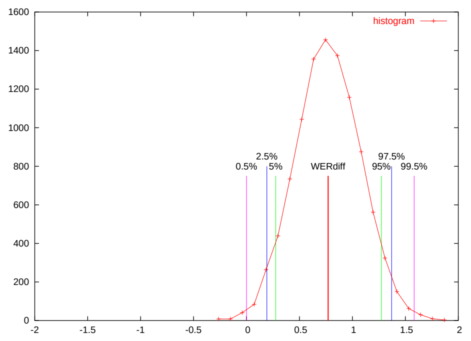
(c) Bag vs HMM embed



(d) E-Boost vs Boost



(e) Bag vs Boost



(f) Bag vs E-Boost

Figure 4.4: Significance levels of word error rate difference between the top four models. The histograms are created from 10,000 bootstrap samples of the test data, as described in Appendix B.4.2.

Part II

Ensembles and sequential decision making

Chapter 5

An overview of reinforcement learning

In sequence recognition the aim is to associate a sequence of observations $o = (o_1, \ldots, o_n)$, with $o_i \in \mathcal{O}$, to a classification decision $y \in \mathcal{Y}$. This can be done by using an injection $f : \mathcal{A}^* \to \mathcal{Y}$ to map a sequence of decisions $a = (a_1, \ldots, a_m)$, $a_i \in \mathcal{A}$, $m \leq n$ onto the class space. More generally, we could define a probabilistic model that is factorisable into

$$p(y|o, h) = \sum_{a \in \mathcal{A}^*} p(y|o, h)p(a|o, h).$$

This follows from the standard conditional mixture model (2.8) under the assumption that $p(y|a, o, h) = p(y|o, h)$. Given some data pairs $d = (o, y)$ and a prior distribution $p(h)$, the aim is to find a decision $h^* \in \mathcal{H}$ such that $p(h^*|y, o) \geq p(h|y, o)$ for all $f \in \mathcal{F}$. Such a problem is ultimately a modelling problem and decision-making amounts to simply selecting the most probable output sequence given some observations and our model.

A more general framework than that of sequence recognition is that of sequential decision making (SDM). The framework makes use of the concept of a Markov decision process[1], in which the agent attempts to maximise some measure of future rewards in an unknown environment. This is not simply a modelling problem. Even with a complete model of the MDP, it would be necessary to perform dynamic programming in order to find the optimal solution to the problem. However, the typical reinforcement learning setting starts from limited knowledge of the MDP and the algorithms must be able to discover the optimal.

This chapter will describe reinforcement learning in some more detail. Firstly, we shall give a definition of the reinforcement learning problem. Then we shall introduce the concept of a value function, which describes how 'good' a state or state-action pair is in a particular MDP and discuss reinforcement learning methods that utilise such functions in order to asymptotically discover the optimal policy in a given

[1]Or more generally, a partially observable Markov decision process (POMDP), see Definition A.1.

environment. The chapter concludes with a short introduction to the exploration-exploitation tradeoff in reinforcement learning.

In the following chapter we attempt an ensemble consisting of a number of base hypotheses trained using an adaptive policy. This policy is learnt through a reinforcement-learning inspired technique, where for every example seen a decision is made as to which of the base hypotheses will be used to predict its label. This method's effectiveness for an essentially supervised task is demonstrated by experimental results on several UCI benchmark databases. Furthermore, some parallels are drawn between the reinforcement learning based method and a mixture of experts (Jacobs et al., 1991) model trained using Expectation Maximisation.

In the above task the reinforcement learning method was simply choosing the hypotheses with the maximum expected reward (i.e. minimum expected classification error). Thus it is essentially a generalisation of a multi-armed bandit problem (see Section 2.1.3), with two differences: Firstly, that the expected reward distribution over the possible actions depends on the observation, and secondly that this distribution is not stationary as all the base hypothesis slowly change during training. However, even the basic stationary non-associative bandit problem is not as simple as it might seem at first glance. While straightforward algorithms exist for determining the expected reward from each possible action asymptotically, it is difficult to do so while simultaneously maximising the reward received. The sampling from the reward distribution in order to determine it more accurately is frequently referred to as *exploration*, while the use of our current knowledge of this distribution in order to maximise the expected return is referred to as *exploitation*. Chapter 7 elaborates on the trade-off between exploration and exploitation and talks about basic mechanisms for implementing such a trade-off algorithmically. Subsequent chapters consider two classes of methods for estimating uncertainty in estimates of expected returns: the first is based on estimates of parameter accuracy, while the second class explicitly maintains a distribution of rewards using a number of approaches. Apart from the analytical Bayesian estimate, we consider two ensemble methods that uses a population of estimators to represent a distribution of estimates in a manner. One such method is similar to bagging, while the other is a particle grid filter: an explicit Monte Carlo approximation of the analytical Bayesian solution. This part concludes with a short discussion on how such estimates of uncertainty can be used to create optimal exploration policies.

5.1 Reinforcement learning

As previously mentioned in Section 2.1.3, reinforcement learning is generally concerned with the problem of acting in an unknown environment. It is a special problem in sequential decision making where the utility is some measure of the future return

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad \gamma \in [0, 1], \tag{5.1}$$

where r_t is generated by a Markov decision process $\mathcal{M}$ (see Definition 2.2).

Since the future values of r are unknown, calculating R is not possible. However its statistical distribution can be calculated for any particular policy π and MDP $\mathcal{M}$. In particular, it is possible to estimate its expectation. The expected return for a policy π can be written as

$$E[R_t|\pi] = \sum_{k=0}^{\infty} \gamma^k E[r_{t+k}|\pi] = E[r_t|\pi] + \gamma E[R_{t+1}|\pi]$$

The objective can be formulated as finding π^* such that

$$E[R_t|\pi^*] \geq E[R_t|\pi] \quad \forall \pi$$

Other conditional expectations of the return are also of interest, and lead to important algorithms. Such forms are referred to as value functions and are used in value-based reinforcement learning methods such as temporal-difference learning (Sutton, 1988), Q-learning (Watkins and Dayan, 1992) and SARSA (Rummery and Niranjan, 1994; Sutton, 1996)

5.2 Value functions

The expected value of the return if we start from state s and follow policy π there after forms the *state value function*

$$Q^\pi(s) = E[R_t|\pi, s_t = s] \tag{5.2}$$

The expected return if we take action a in state s and follow π there after forms the *state-action value function*

$$Q^\pi(s, a) = E[R_t|\pi, s_t = s, a_t = a] \tag{5.3}$$

These conditional forms are especially useful. If we know Q^*, the value function of the optimal policy π^*, we can select actions optimally. On the other hand, as the policy improvement theorem states (see for example (Sutton and Barto, 1998)), if we know the state-action value function for any particular policy π, we can immediately improve it by creating a new, greedy policy π' for which $P(a_t = a|s_t = s) \geq P(a_t = a')$ for $a = \arg\max_i Q^\pi(s, i)$.

If only state values are known, then calculating an improved policy requires knowledge of the MDP's transition probabilities $p(s_{t+1}|s_t, a_t)$. Our attention however will be restricted to the *model-free* case, i.e. when there is no explicit model for the MDP, either one given or one estimated through observation.

5.2.1 Value function estimation

The most obvious way to estimate the value function for a particular policy is through Monte Carlo (MC) sampling (see Casella et al., 1999; Doucet et al., 2001). By selecting actions according to the policy we wish to evaluate, and recording all rewards received, it will be possible to simply calculate the expectation of return for every state-action pair through averaging the observed samples of the return that occurred

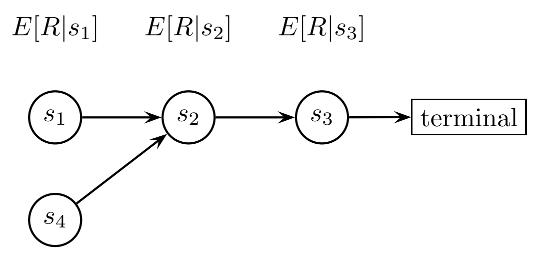


Figure 5.1: Relationships between estimates. We consider a model with four states and a terminal state, which has a reward of 0. There is a dependence on the return (and its expectation) among linked states. This dependence can be utilised to improve estimates of the return on less-frequently visited states, when those are linked to more frequently visited ones.

after each state-action pair. This can be written as follows:

$$\hat{Q}^{\pi}(s) = \hat{E}[R_t|\pi, s_t = s] = \frac{1}{N}\sum_{n=1}^{N} R_{t_n}, \quad s_{t_n} = s \tag{5.4}$$

$$\hat{Q}^{\pi}(s,a) = \hat{E}[R_t|\pi, s_t = s, a_t = a] = \frac{1}{N}\sum_{n=1}^{N} R_{t_n}, \quad s_{t_n} = s, a_{t_n} = a \tag{5.5}$$

This straightforward formulation unfortunately requires that tasks terminate because it requires generating multiple samples of the return.

Such Monte Carlo methods tend to also make inefficient use of data: The estimate of the expected return for each state-action pair is determined solely by the observed returns. However, fact there exists an intrinsic dependence between our estimates, which can be exploited in order to make better use of our observations. For example, consider the model shown Figure 5.2.1. If we have already observed some trajectories for s_1, s_2, s_3 then we have an estimate for $E[R|s_2]$. Then, upon observing a transition $s_4 \to s_2$, we can update our estimate of $E[R|s_4]$ immediately, without waiting for arrival at the terminal state, because of the additive nature of the return. This form of *bootstrapping* is used in temporal-difference methods and it allows the estimation of returns in non-terminating environments.

5.2.2 Temporal-difference (TD) value function estimation

The random variable we are sampling from is the return R_t. While the MC uses actual samples of the return, TD methods employ bootstrapping by using the expected value of the return instead. So a sample R_t is replaced by an estimate of the form $r_t + \gamma \hat{E}_t[R_{t+1}|\cdot]$. This way it is possible to make use of the current observation, $(s_t, r_t, s_{t-1}, a_{t-1})$, to update our value function estimates.

In general, updates are performed so as to minimise some measure of the discrepancy between our estimates and our observations. One useful such measure is the $\|d_t\|^2$, where d_t is referred to as the *temporal difference error*

$$d_t = r_t + \gamma \hat{E}_t[R_{t+1}|\cdot] - \hat{E}_t[R_t|\cdot].$$

In the algorithms which we will examine, value functions are represented by some parametrised function $f(w, \cdot) \equiv \hat{E}[R|\cdot]$ and the value function approximation is performed by minimising the expected value of $\|d_t\|^2$ via stochastic gradient descent methods. Note that the gradient of the parameters with respect to the cost will be

$$\nabla_\theta \|d_t\|^2 = \nabla_\theta f \nabla_f \|d_t\|^2 = 2\nabla_\theta f d_t$$

Specific forms of the temporal difference error lead to specific algorithms. The TD-learning algorithm (Sutton, 1988) maintains an estimate of the state values of the current policy and uses the update

$$d_t = r_t + \gamma \hat{E}_t[R_{t+1}|\pi, s_t = s] - \hat{Q}^\pi_t(s) \approx r_t + \gamma \hat{Q}^\pi_t(s') - \hat{Q}^\pi_t(s) \tag{5.6}$$

The SARSA algorithm maintains an estimate of the state-action values of the current policy and uses the update

$$d_t = r_t + \gamma \hat{E}_t R_{t+1}|\pi, s_t = s] - \hat{Q}^\pi_t(s, a) \approx r_t + \gamma \hat{Q}^\pi_t(s', a') - \hat{Q}^\pi_t(s, a) \tag{5.7}$$

Q-learning maintains an estimate of the state-action values of the optimal policy and uses the update

$$d_t = r_t + \gamma \hat{E}_t[R_{t+1}|\pi^*, s_t = s] - \hat{Q}^*_t(s, a) \approx r_t + \gamma \max_{a'} \hat{Q}^*_t(s', a') - \hat{Q}^*_t(s, a) \tag{5.8}$$

The first two methods are referred to as *on-policy* methods, while the Q-learning is an *off-policy* method. In the first case, the policy being evaluated is the same as the policy used to sample the environment, while in the second case the sampling policy differs from the one being evaluated. When we simply maintain individual estimates for each possible state-action pair, the value function can be updated according to

$$\hat{Q}_t(s, a) = \hat{Q}_{t-1}(s, a) + \eta_t d_t, \tag{5.9}$$

where $\eta_t > 0$ is a step-size parameter. However, as we will see below, this is just a special case of a general steepest stochastic gradient descent update.

5.2.3 Gradient descent implementation

If we consider the cost $C = E[\|d_t\|^2]$, and a parametrised function Q with parameters θ for minimising it, then the gradient descent update can be simply written as

$$\delta_t = \nabla_\theta Q \ \nabla_Q C = 2\nabla_\theta Q \ d_t,$$

where d_t is the temporal-difference error. Then updating the parameters θ with steepest gradient descent amounts to

$$\theta_{t+1} = \eta_t \nabla_\theta Q d_t. \tag{5.10}$$

The gradient of the evaluation function with respect to the parameters depends upon the function's architecture. We shall make a further distinction between *tabular* and *approximating* architectures for value function estimation. In the tabular case, a discrete state-action space is assumed, and a different estimate $\theta_{s,a}$ is maintained for each one of the possible values $Q(s,a)$. This can be written as

$$Q(s,a) = I_s \begin{bmatrix} \theta_{1,1} & \cdots & & \\ \vdots & \theta_{s,a} & \vdots & \\ \vdots & \ddots & \vdots & \\ & & \theta_{m,n} & \end{bmatrix} I'_a,$$

where I_s and I_a are indicator vectors of the form $(0,\dots,0,1,0,\dots,0)$, such that the i-th component is 1 when $s = i$ or $a = i$ respectively. In that case (5.10) becomes simply (5.9), with each estimate $Q(s,a)$ corresponding to a parameter $\theta_{s,a}$.

Any other choice of for the parametrisation of Q is usually referred to as an approximating architecture, with linear models and sigmoidal neural networks being the mostly used in practice.

5.2.4 Eligibility traces

Both MC and TD methods maintain a model for some conditional expectation of the return, $E[R_t|\cdot]$. In MC methods, the model is

$$\mathcal{R}_\infty \equiv \hat{E}[R_t|r_{t+1},\dots,r_\infty] = R_t,$$

the actual sampled value for that particular episode. If we take the expectation we see that $E[\mathcal{R}_\infty] = E[R_t]$. In TD methods, we only use a partial sample up to time $t+K$ and for the remaining time an estimate is used based on our current model is used (setting $K = 0$ corresponds to the estimates given in the previous section):

$$\mathcal{R}_K \equiv \hat{E}[R_t|r_{t+1},\dots,r_{t+K}] = \sum_{i=0}^{K} \gamma^i r_{t+i} + \gamma^{K+1}\hat{E}[R_{t+1+K}].$$

It is possible to mix the two approaches through a convex combination

$$E[R_t] = \alpha\mathcal{R}_\infty + (1-\alpha)\mathcal{R}_0,$$

with $\alpha \in [0,1]$. This gives rise to the update

$$d_t = \alpha(\mathcal{R}_\infty - Q_t) + (1-\alpha)(\mathcal{R}_0 - Q_t),$$

which is a mixture between the Monte Carlo and the temporal-difference updates. Alternatively, all estimates can be mixed in order to obtain

$$d_t \propto \sum_i \alpha_i(\mathcal{R}_i - Q_t).$$

If we define $\alpha_i = \lambda^i$, $\lambda \in [0,1]$, we can use eligibility traces, which can be viewed as importance weights on the parameters (see (Precup et al., 2000) and Appendix B.3.2 for further discussion). In particular, we may define the eligibility trace for a state-action pair (or simply a state or an action) as:

$$e_t(\cdot) = \gamma\lambda e_{t-1}(\cdot) + \nabla_{\theta_t}\hat{E}[R_t|\cdot]. \tag{5.11}$$

For tabular action value methods, this simply corresponds to what is referred to as *accumulating eligibility traces*

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + I_t(s,a) \tag{5.12}$$

where $I_t(s,a) = 1$ if $s_t = a$, $a_t = a$ and 0 otherwise. Alternatively, one may use *replacing eligibility traces*,

$$e_t(s,a) = \left(\gamma\lambda e_{t-1}(s,a)\right)\left(1 - I_t(s,a)\right) + \left(I_t(s,a)\right), \tag{5.13}$$

which may be useful in particular problemsSutton and Barto (see 1998, chap. 6,7).

For tabular action-value methods, eligibility traces are used as importance weights for updating estimates, that is

$$\hat{Q}_t(s,a) = \hat{Q}_{t-1}(s,a) + e_t(s,a)\eta_t d_t. \tag{5.14}$$

5.3 Exploration in reinforcement learning

One of the main assumptions in reinforcement learning is that all state-action pairs (s,a) will be sampled infinitely often, and given this assumption, most algorithms are only guaranteed to asymptotically converge(Jaakkola et al., 1994). This sampling requirement explains why in most cases a purely greedy policy is not used. The sampling of the state-action space is referred to as either the *exploration policy* or the *action selection method*[2]

Definition 5.1 (ϵ-greedy action selection) *In this action selection mechanism, the highest evaluated action is selected with probability* $1-\epsilon$ *and a random action is selected otherwise, leading to the following probabilities.*

$$P(a_t = i|s_t = s) = \begin{cases} (1-\epsilon) + \epsilon/|\mathcal{A}| & i = \arg\max_{a\in\mathcal{A}} Q(s,a) \\ \epsilon/|\mathcal{A}| & \textit{otherwise} \end{cases}$$

Definition 5.2 (softmax action selection)

$$P(a_t = i|s_t = s) = \frac{\exp(\beta Q(s,a))}{\sum_{a'\in\mathcal{A}} \exp(\beta Q(s,a'))}$$

Aside from explicitly forcing a sampling of the environment, sampling can be encouraged by adding a constant exploration bonus ρ to the estimated expected return of unvisited state-action pairs. The method of *optimistic initial values*, where the initial estimates of return for all state action pairs is set to

[2]Such policies are inherently non-stationary, since they depend on how our estimates of the value of actions change over time. However, under certain conditions they converge to stationary policies.

a high value is the most elementary such methods and has the advantage that a fully greedy policy can be used. In *Dyna-Q* learning (Sutton, 1990) an exploration bonus is added to all state-action pairs that have not been recently visited. The Dyna algorithm includes an environment model, which enables efficient exploration of states that have not been visited in the past. Finally, techniques related to exploration have been used in *prioritised sweeping* (Moore and Atkeson, 1993), for the purpose of restricting the number of parameter updates to be performed in the model.

In order to motivate the development of better exploration techniques, consider the scenario of selecting actions in an episodic environment. Greedy action selection would always select the actions with the highest expected return, which is optimal in terms of expected return according to our currently inferred model if this is the last episode to be experienced. However, if there remain many episodes to be experienced, perhaps a better strategy would be to explore the environment in the next few episodes, so our model of it can be improved, until our model is sufficiently accurate that it virtually guarantees that using it will give us the maximum possible return. Thus intuitively it seems that the randomness in action selection should be increased when the number of episodes is larger, since then more time is available to collect rewards, the more we can potentially benefit from reducing the *uncertainty* in our model.

A formal description of how the trade-off between exploration and exploitation naturally arises in rational agents under uncertainty will be given in Chapter 7, together with some practical algorithms and some illustratory experimental results. The methods used to estimate and represent uncertainty form a crucial aspect of any such algorithm and Chapter 8 introduces two such methods. The first method is based on estimating the gradient and Hessian of the cost function, while the second is based on sampling from a prior distribution of models in order to create an ensemble in a manner reminiscent of bagging and particle filtering (see Casella et al., 1999), but with a different set of goals: firstly, to maintain a distribution over possible values of actions, and secondly to use this distribution in order to sample efficiently from the environment.

Chapter 6

Reinforcement learning ensembles for classification

The problem of pattern classification has been addressed in the past using mainly supervised learning methods. In this context, a set of N example patterns $\hat{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ is presented to the learning machine, which adapts its parameter vector so that when input vector x_i is presented to it the machine outputs the corresponding class $y_i \in \{1, 2, \dots, |\mathcal{C}|\}$, where $|\mathcal{C}| \in \mathbb{N}$ is the number of classes. Let us denote the output of a learning machine for a particular vector x_i as $h(x_i)$. The classification error for that particular example can be designated as $l_i = 1$ if $h(x_i) \neq y_i$ and 0 otherwise. Thus, the classification error for the set of examples $\hat{D}$ can be summarised as the empirical error $\hat{L} = \sum_i l_i/N$. If $\hat{D}$ is a sufficiently large representative sample taken from a distribution D, then $\hat{L}$ should be close to the generalisation error, $L = \int p_D(x)l(x)$. In practice, however, the training set provides limited sampling of the distribution D, leading to problems such as overfitting. Thus it is expected that $L > \hat{L}$.

Since the generalisation error cannot be directly observed, it has been common to use a part of the training data for validation in order to estimate it. This has led to the development of techniques mainly aimed at reducing the over-fitting caused by limited sampling, such as early stopping and K-fold cross-validation.

Another possible solution is offered by ensemble methods, such as the mixture of experts (MOE) architecture (Jacobs et al., 1991), bagging (Breiman, 1996) and boosting (Freund and Schapire, 1997). The boosting algorithm Ada-Boost has been shown to significantly outperform other ensemble techniques for low-noise data (see (Bauer and Kohavi, 1999; Dietterich, 2000)). While the good performance of MOE and bagging is frequently attributed to the independence of experts and the reduction of classifier variance, results explaining the effectiveness of Ada-Boost relate it to the *margin of classification* (Schapire et al., 1998). See Section 2.2.4 for a description of margins.

In this chapter, which is was previously presented at ESANN 2004 (Dimitrakakis and Bengio, 2005b), the possibility of using an adaptive rather than a fixed policy for training and combining base classifiers is investigated. The field of reinforcement learning (RL) (Sutton and Barto, 1998) provides natural candidates for use in adaptive policies. In particular, the policy is adapted here using Q-learning (Watkins and Dayan, 1992), a method that improves a policy through the iterative approximation of an evaluation function Q (see Section 5.2). Previously Q-learning had been used in a similar mixture model applied to a control task (Anderson and Hong, 1994). An Expectation Maximisation based mixtures of experts (MOE) algorithm for supervised learning was presented in (Jordan and Jacobs, 1994). Herein we attempt to solve the same task as in the standard MOE model, but through the use of reinforcement learning rather than expectation maximisation techniques. A description of the similarities between reinforcement learning and expectation maximisation methods for multi-expert architectures was presented in (Toussaint, 2002).

This section is organised as follows. The framework of reinforcement learning is introduced in Section 6.1. Section 6.1.1 outlines how the RL methods are employed in this work and describes how the system is implemented. Experiments are described in Section 6.2, followed by conclusions and suggestions for future research.

6.1 General architecture

The objective in classification tasks is to reduce the expected value of the error, $E[l]$. The empirical loss $\hat{L}$ provides a biased estimate of this error. The suggested classifier ensemble consists of a set of n base classifiers, or experts, $\mathcal{E} = \{e_1, e_2, ..., e_n\}$ and a controlling agent that selects the experts to make classification decisions and to train on particular examples. The controlling agent must learn to make decisions so that $E[l]$ is minimised. We employ reinforcement learning for the purpose of finding an appropriate behaviour for the agent.

The following section will detail how Q-learning can be employed in classification tasks and potential problems with the technique are discussed. On the whole, however, it is estimated that reinforcement learning can provide an interesting alternative to supervised learning techniques even for supervised-learning tasks.

6.1.1 Implementation

The aim of this work was to apply RL techniques to the problem of training an ensemble model. In order to achieve this, we use a set of initially untrained classifiers, and a controlling agent, which utilises Q-learning. During training, the agent makes decisions about which classifiers will be trained on a given example. During testing, it determines how the labels output by the classifiers will be used to make the classification decision for a given example.

More specifically, we employ an architecture with n experts, implemented as multi-layer perceptrons (MLPs), and a further MLP with n outputs and parameters θ which acts as the controlling agent. All the MLPs have a single hidden layer with hyperbolic tangent units and are trained using steepest gradient descent. The expert MLPs use a softmax output and a cross-entropy criterion (see i.e.(Bishop, 1995)), which are suitable for a maximum likelihood training[1]. In this setting we attempt to minimise

$$E_{\hat{D}}[y \log h(x)],$$

with $h : \mathcal{X} \to \mathcal{Y}$, mapping to a probability space. The state space of the controlling agent is $\mathcal{S} \equiv \mathcal{X}$, the same as the classifiers' input space and its outputs approximate $Q(s, a_j)$. Thus, it is implemented with an MLP which has the same number of inputs as the expert MLPs and with a number of outputs equal to the number of possible actions.

At each time step t a new example x is presented to the ensemble and each expert i emits a classification decision $h_i : \mathcal{X} \to [0, 1]^{|\mathcal{C}|}$. The ensemble makes a classification decision of the form

$$f(x) = \sum_i w_i h_i(x) \tag{6.1}$$

$\sum_i w_i = 1$. We examine the case where the number of actions is equal to the number of experts and in which taking action a_j corresponds to setting $w_i = 1$ for $i = j$ and $w_i = 0$ otherwise. Thus, taking action a_j results in expert e_j making the classification decision.[2] We also chose to use the action a_j to select the expert to be trained on the particular example. As an aside, note that under a given policy, the expected value of w_i given x corresponds to $E[w_i|x] = p(a_i|x)$, the probability of action a_i given x. In this manner one could write, for the softmax action selection method,

$$E[w_i|x] = p(a_i|x) = \frac{\exp(Q(x, a_i))}{\sum_j \exp(Q(x, a_j))}. \tag{6.2}$$

During training, the classification decision at time t results in a reward $r_{t+1} \in \{0, 1\}$, which is 1 if the example is classified correctly and 0 otherwise. As noted before, we use the gradient form of the Q-learning update (5.8). The derivative of the cost function (the cost is the expected squared approximation error; see Section 5.2.2) with respect to the network outputs is $\delta = r_{t+1} + \gamma \max_i Q(s', a_i) - Q(s, a_j)$. We use stochastic steepest gradient descent with learning rate $\eta > 0$. Note also that when $\gamma = 0$, the Q-learning update is indistinguishable from other state-action value temporal difference updates such as SARSA (see Section 5.2.2). The algorithm is implemented as follows:

1. Select example x_t randomly from $\mathcal{X}$.

2. Given $s = x_t$, choose $a_j \in \mathcal{A}$ according to a policy derived from Q (for example using ϵ-greedy action selection) .

3. Take action a_j, observe r_{t+1} and the next state $s' = x_{t+1}$, chosen randomly from $\mathcal{X}$.

4. $\delta = r_{t+1} + \gamma \max_i Q_t(s', a_i) - Q_t(s, a_j)$.

5. $\theta_{t+1} = \theta_t + \eta \delta \nabla_\theta Q_t(s, a_j)$.

[1]Since we are trying to find the model with maximum likelihood given the data.
[2]However note that when using stochastic action selection, all experts are trained to some extent on all the data.

6. $s = s'$.

7. Loop to 2, unless termination condition is met.

During testing the situation is subtly different. Firstly, no more rewards are observed and no model adaptation occurs. Secondly, selecting actions stochastically can be disadvantageous. In order to eliminate the stochasticity, we simply take the expectation from (6.1) to obtain

$$E[f(x)|x] = \sum_i E[w_i|x]h_i(x).$$

Two specific forms were considered, independently of the action selection mechanism during training. Firstly, that of setting $p(a_i|x) = 1$ for $i = \arg\max_j Q(x, a_j)$, and secondly using the softmax form (6.2). The former case would correspond to a belief that our evaluation function is accurate on unseen data, which is probably not true. Thus, the smoothing performed in the latter case might be advantageous.

Choice of γ

In the algorithm we have described, the state is completely determined by the example x_t. Since this example is selected randomly (steps 1,3), we have $p(s_{t+1} = s'|s_t = s, a_t = a) = p(s_{t+1} = s')$, leading to

$$\begin{aligned}
E[R_t|s_t = s, a_t = a] &= E[r_{t+1}|s_t = s, a_t = a] + \sum_{k=1} \gamma^k E[r_{t+k+1}|s_t = s, a_t = a] \\
&= E[r_{t+1}|s_t = s, a_t = a] + \sum_{k=1} \gamma^k \sum_{s'} E[r_{t+k+1}|s_{t+k} = s']p(s_{t+k} = s'|s_t = s, a_t = a) \\
&= E[r_{t+1}|s_t = s, a_t = a] + \sum_{k=1} \gamma^k \sum_{s'} E[r_{t+k+1}|s_{t+k} = s']p(s_{t+k} = s') \\
&= E[r_{t+1}|s_t = s, a_t = a] + E[r] \sum_{k=1} \gamma^k
\end{aligned}$$

where all expectation are taken with respect to the policy π (so there is an implicit dependency on the policy). Thus, there is no temporal structure to be exploited by the full reinforcement learning framework, at least in the visible part of the state. In other words, the classification task is similar to an n-armed bandit problem [3] since the next state is not influenced by the agent's actions. For the above reasons, we have set the value of γ to zero. Maximisation of the expected value of equation (5.1), when $\gamma = 0$ amounts to maximising

$$E[R_t] = E[r_{t+1}].$$

Since the optimal policy π^* is the policy that maximises this value, we have

$$\pi^* = \arg\max_{\pi} E[r_{t+1}|\pi].$$

[3] In the n-armed bandit problem (see Section 2.1.3) the objective is to choose an optimal action among n. The reward at each time step only depends upon the action taken and a state s, but the state s does not depend upon the action taken. Thus, the optimal policy is the same no matter what the value of γ is, as the action taken at time t only influences r_{t+1} and not any later rewards.

Because of our definition of the reward, this is equivalent to finding the policy that minimises the empirical error.

This loss of temporal structure might be considered unfortunate. Indeed, the task is more accurately described as a partially observable process since the parameters of the classifiers constitute a state which changes depending on the agent's actions. This would formally necessitate the need for $\gamma > 0$, and potentially the need to approximate the hidden state with some kind of model. Nevertheless, it seems reasonable to argue that the part of the system state which can be expressed as a function of the classifiers' parameters will change rapidly at the initial stages of learning and then stabilise when each local expert approaches its region of convergence. If this is true, then the problem is similar to a *semi-stationary* bandit problem[4] and a value of $\gamma = 0$ is still appropriate, i.e. there is nothing to be gained by adding temporal structure since old states can never be revisited, at least not with the particular set of actions we have defined.

However there exist some sequence classification applications for which this is not so. These include event detection tasks, such as the detection of the onset of failures in dynamical systems. In particular, if we are defining a model for the state that defines a joint distribution for actions, observations and state, then the state may no longer be degenerate. This is so in the case where each expert is a hidden Markov model, and where we use the action to switch between models. We, however, concentrate on the simple semi-stationary case, for which interesting parallels with the mixture of experts algorithm can be drawn.

6.1.2 Comparison with mixture of experts

The mixture of experts algorithm shares a number of similarities with the one presented here. A comparison between a mixture of experts using a modified version of the EM algorithm and the Q-learning algorithm was presented in (Toussaint, 2002). We refrain from introducing new symbols whenever possible in this section, in order to emphasise the relations between algorithms.

In the mixtures of experts framework each expert i makes a classification decision $h_i : \mathcal{X} \to [0,1]^{|\mathcal{C}|}$, with $|h_i|_1 = 1$, where $|\cdot|_1$ denotes the l_1 norm (A.1). Thus $h_i(x)$ can be described as probability distribution over the classes given the data x. We use $p(y|x,i)$ to note the probability that expert i outputs class y, given x. Similarly, the gating mechanism is used to create a probability distribution over the experts given the data, $p(i|x)$, commonly referred to as the *prior* of each expert. Thus in order to find the probability of each class given the data we simply use $p(y|x) = \sum_i p(y|x,i)p(i|x)$.

In order to adjust the parameters of the gating mechanism, both in the gradient and the EM versions of the algorithm, we estimate the corresponding *posterior* as

$$p(i|x,y) = \frac{p(y|x,i)p(i|x)}{p(y|x)}.$$

This is the main difference with the reinforcement learning method employed herein, since the action selection mechanism only considers binary decisions made by the classifiers. Instead of actually calculating $p(i|x,y)$ we are treating the reinforcement r_t as a stochastic variable that depends on the action a_i and

[4]In such a problem the expected reward from each bandit changes slowly with time

Dataset	N. Test	MLP	Boost	MOE	RL	Ref.	nhu
breast-cancer	165	4.24%	**1.21**% (4)	4.84% (2)	6.06% (4)	6.30%[*]	25 (50)
forest	565895	31.2%	**26.2**% (32)	30.2% (4)	28.32% (8)	30.0%[†]	100 (50)
heart	80	15.0%	13.7% (2)	16.3% (2)	**12.5**% (16)	22.1%[‡]	10 (10)
ionosphere	151	**5.30**%	7.28% (2)	9.27% (32)	5.96% (16)	4.00%[*]	100 (25)
letter	4000	4.45%	**2.55**% (32)	4.03% (4)	3.55% (2)	20.0%[†]	100 (50)
optdigits	2394	**1.84**%	2.2% (4)	2.29% (16)	2.26% (4)	2.00%[*]	100 (25)
pendigits	3498	3.20%	3.33% (16)	**2.71**% (4)	3.29% (2)	2.26%[*]	100 (50)
spambase	2269	6.89%	6.57% (8)	7.58% (2)	**6.43**% (32)	7.00%[§]	50 (25)
vowel	462	**56.9**%	66.7% (32)	64.0% (32)	66.0% (32)	44.0%[*]	100 (100)

Table 6.1: Test classification error on 9 UCI benchmark datasets. Results are shown for a single MLP (**MLP**), and mixtures of 32 experts that have been trained with boosting (**Boost**), mixture of experts (**MoE**), and Q-learning (**RL**). **Ref** indicates a reference result. **N. Test** shows the number of test samples, while **nhu** shows the number of hidden units for the base networks and for the gating network (in parenthesis). Results in bold indicate that this was the best result obtained for a particular dataset.

for which $E[r|x, y, \pi, a_i] = p(i|x, y)$.

6.2 Experimental results

In order to evaluate the effectiveness of this approach we have performed a set of experiments on 9 datasets that are available from the UCI Machine Learning Repository (Blake and Merz, 1998). For each dataset there was a separate training and test set. We used 10-fold cross-validation on the training set in order to select the number of hidden units from $\{10, 25, 50, 100\}$ for the base classifier. Each classifier was then trained on the whole training set for 100 epochs per expert. The optimisation algorithm used was steepest stochastic gradient descent with a learning rate of $\eta = 0.01$. The number of hidden units for the gating mechanism was selected so that the temporal difference error would be minimised on the training set. The same number of gate hidden units was used in the mixture of experts in every case. The discount parameter γ for the controlling agent was set to 0, for the reasons explained in Section 6.1.1. For each one of the ensemble methods shown, the number of experts was selected through splitting the training dataset in two subsets: 2/3 of the data was used for training and 1/3rd was used for evaluation. according to which the number of experts in $\{2, 4, 8, 16, 32, 64\}$ was selected. The results reported here are for ϵ-greedy actions selection, with $\epsilon = 0.1$. Results with softmax action selection do not appear significantly different[5], however there is a marked difference when good, however.

A comparison was made between the RL-controlled mixture, a single MLP, the Mixture of Experts and Ada-Boost.M1 using MLPs. As Table 6.1 summarises, the ensembles generally manage to improve

[5]For this particular problem, and with $\gamma = 0$, the expected return of the best action can be at most 1 while that of the worst action can be at 0. The probability of the greedy action in ϵ-greedy methods, given n actions, is $n + 1/n - \epsilon$. For the softmax method, we would have a similarly flat distribution if all other experts have a similar evaluation, which is to be expected for this particular problem.

[*]k-nearest neighbours

[†]Neural network

[‡]Logistic regression

[§]Unknown

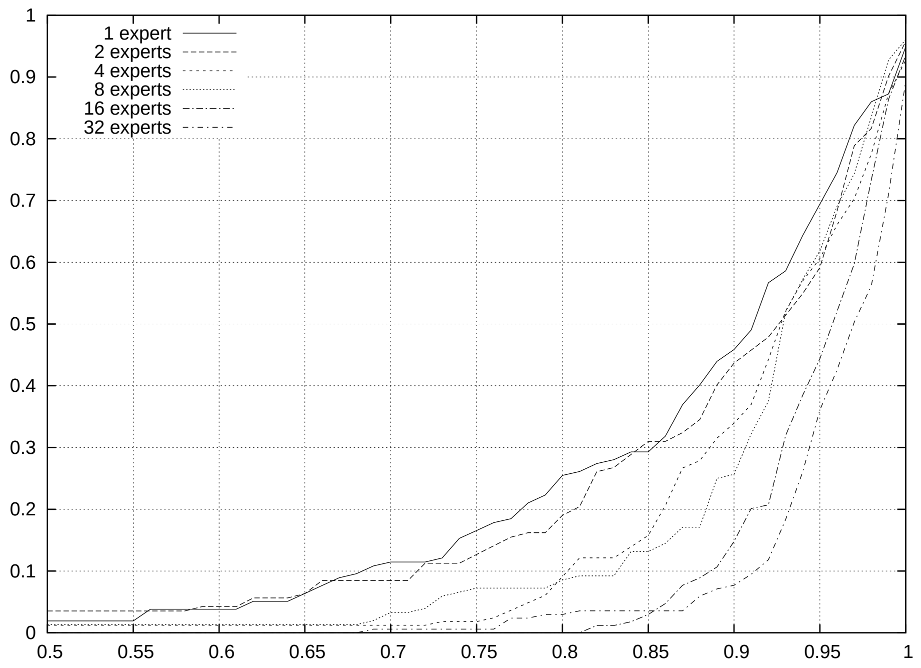


Figure 6.1: Cumulative margin distribution for RL on the ionosphere dataset, with an increasing number of experts. The x axis shows $P(m(X) < x)$, the probability that the margin of some example X is smaller than x. See Section 2.2.4 for an explanation of margins.

test performance compared to that of the base classifier. There are three cases where the MLP performs better, though only in the vowel dataset is its performance significantly better than that of the second best system ($p < 0.016$). In turn, Boosting performs significantly better with respect to the second-best system in all three cases: breast-cancer ($p < 0.046$), forest ($p < 0.0001$) and letter ($p < 0.0046$). Finally, MOE was only best once and the RL approach twice, but in all cases with $p > 0.1$. These significance figures were calculated from a two proportion z-test[6]. For each dataset we have also calculated the cumulative margin distribution: the empirical probability that an example X in the training set will have margin smaller than x, i.e. $P(m(X) < x)$. For the RL mixture there was a constant improvement in the distribution in most datasets when the number of experts was increased (see for example Figure 6.1). This may provide some explanation for the improvement in generalisation performance, since for example Schapire et al. (1998) links the minimum margin (2.18) with a bound on the generalisation error, although this behaviour has not been consistent, as is evident from the example given. While the conditions under which the margin is maximised are not investigated here, some explanations are given by Rosset et al. (2003) and Collobert and Bengio (2004).

All the datasets originate from the UCI machine learning repository (Blake and Merz, 1998) and are accompanied by information files with descriptions and reference results. It is particularly interesting perhaps to note that the reference results given for most of those datasets were obtained with the extremely

[6]See Appendix B.4.1 for a description

simple 1-nearest-neighbour method and that they are frequently very close to the best method of the ones used here. The **breast-cancer** dataset, the aim is to classify a tumour as either benign or malignant. The **forest** dataset examines forest cover type - the aim is to predict a qualitative measure (cover type) through twelve quantitative cartographic measures. The number of classes is 7, and unclear boundaries between different classes may account for some amount of label noise. The **heart** dataset is used for predicting the presence of heart disease from 13 quantitative variables and it is assumed that there is but little label noise. The **ionosphere** dataset is used to predict the 'good' and 'bad' radar returns from radar pulses. The **letter** dataset is used for a 26-character classification task, where 16 quantitative attributes corresponding to statistical moments and edge counts are used to describe each observed character. The **optdigits** database contains handwritten digits (0-9) scanned into an 8x8 matrix. The **pendigits** database contains handwritten digits from a small number of writers, collected using a graphics tablet. The data consists of the coordinates of 16 regularly spaced points on the numbers trajectory. The data in the test set has been generated by a different set of writers than those in the training set. The **spambase** database is used for the task of deciding whether a given email is spam or not and the attributes fore each instance consist of summary information about each email, such as word frequencies. Finally, the **vowel** dataset contains 11 different vowels, spoken by 90 different speakers, with the training dataset contains data from the first 48 speakers only. In this and other of cases the data in the test set arises from a different distribution than the data in the training set. Thus, estimates of generalisation error from validation in the training set are inherently unreliable for optimally selecting hyper-parameters, or indeed models. In fact, for most of these datasets it appears that selecting the best method and associated hyper-parameters via cross-validation or a hold-out does not reliably predict the best performing method on the test data: A selection in this manner results in making the correct decision only 33.3% of the time. On the other hand, the worst method is predicted as being the best only once, on the breast-cancer dataset. However research on the bias of model selection methods is beyond the scope of this study.

6.3 Conclusions and future research

The aim of this work was to demonstrate the feasibility of using adaptive policies to train and combine a set of base classifiers. While this purpose has arguably been reached, there still remain some questions to be answered, such as under what conditions the margin of classification is increased when using this approach.

An interesting aspect of this problem is the state space of the agents. As has been noted in Section 6.1.1, the initial parameters of the experts constitute a part of the (in our case, unobservable) state space which is only briefly visited by the agent. As learning progresses, the parameters of each expert converge to a steady state. For the case where information about the expert parameters is not included in the state, the problem becomes a slowly changing n-armed bandit task, which in the end becomes stationary. If we include such information in the state, then we are faced with a slightly different reinforcement learning problem than the one commonly encountered. This occurs because there exist a subspace of the state vector (related to the data) which is sampled frequently and another subspace (related to the state of the experts) where only a single trajectory is sampled. The question is firstly what techniques, short of resetting the experts to an initial state, can be applied to sample more trajectories and secondly how can knowledge from more trajectories be used to aid in the search for a better stationary point.

A very similar RL technique was recently described in Partalas et al. (2006), where actions where instead used to select the experts from a set, in a form of ensemble pruning. There the state does not include the observations and thus the state space does not become unmanageably large, while the problem itself becomes fundamentally different.

Enlarging the space of actions poses another interesting problem. Suppose for example that the best decision that we can make for a particular input is to combine the outputs of two experts, rather than use a single expert's output. In order to generalise for this case, we define a set of possible weight combinations; each possible combination constitutes a different action. In (6.2) we defined the expectation of expert weights for a particular input under a softmax policy. In general, however, it is possible to maintain a probability distribution for the weights, rather than a simple expectation. After assuming a joint distribution for the weights we can estimate the conditional density of the return given the weights. Action selection could be done by sampling from the joint distribution of weights, or else importance sampling techniques could be used. This is part of our current work in the field of action selection.

An alternative to action value methods for such enlarged spaces is provided by direct gradient descent in policy space (Baxter and Bartlett, 2000). These have also been theoretically proven to converge in the case of multiple agents and could be much more suitable for problems in partially observable environments and with large state-action spaces.

Chapter 7

Optimal exploration

In reinforcement learning, the dilemma between selecting actions to maximise the expected return according to the current world model and to improve the world model such as to *potentially* be able to achieve a higher expected return is referred to as the *exploration-exploitation trade-off.* This has been the subject of much interest before, one of the earliest developments being the theory of sequential sampling in statistics, as developed by Wald (1947). This dealt mostly with making sequential decisions for accepting one among a set of particular hypothesis, with a view towards applying it to jointly decide the termination of an experiment and the acceptance of a hypothesis. A more general overview of sequential decision problems from a Bayesian viewpoint is offered by DeGroot (1970). The concept naturally appears also in a game theoretic context (see Luce and Raiffa, 1957, chap. 13) when decision making under uncertainty is considered.

This chapter will describe a novel framework for trading exploration and exploitation nearly optimally, without having to set up any prior parameters other than our prior belief on the possible set of models. How this is possible will be made plain in Section 7.1, where the intuitive concept of trading exploration and exploitation will be seen to arise as a natural consequence of the *definition* of the problem of reinforcement learning. After the problem definitions which correspond to either extreme (only explore versus only exploit) are identified, Section 7.2 offers a summary of related work. Subsequently, Section 7.3 derives a threshold for switching from exploratory to greedy behaviour in bandit problems. This threshold is found to depend on the effective reward horizon of the optimal policy and on our current belief distribution of the expected rewards of each action. A sketch of the extension to MDPs is presented in Section 7.4. Section 7.5 uses an upper bound on the value of exploration to derive practical algorithms, which are then illustrated experimentally in Section 7.6. We conclude with a discussion on the relations with other methods.

7.1 Exploration Versus Exploitation

Let us assume a standard multi-armed bandit setting, where a reward distribution $p(r_{t+1}|a_t)$ is conditioned on actions in $a_t \in \mathcal{A}$, with $r_t \in \mathbb{R}$. The aim is to discover a policy $\pi = \{P(a_t = i)|i \in \mathcal{A}\}$ for selecting actions such that $E[r_{t+1}|\pi]$ is maximised. It follows that the optimal gambler, or oracle, for this problem would constitute a policy which always chooses $i \in \mathcal{A}$ such that $E[r_{t+1}|a_t = i] \geq E[r_{t+1}|a_t = j]$ for all $j \in \mathcal{A}$. Given the conditional expectations, implementing the oracle is trivial. However this tells us little about the optimal way to select actions when the expectations are unknown. As it turns out, the optimal action selection mechanism will depend upon the problem formulation. We initially consider the two simplest cases in order to illustrate that the exploration/exploitation trade-off is and should be viewed in terms of problem and model definition.

In the first problem formulation the objective is to discover a parameterised probabilistic policy $\pi = \{P(a_t|\theta_t) \mid a_t \in \mathcal{A}\}$, with parameters θ_t, for selecting actions such that $E[r_{t+1}|\pi]$ is maximised. If we consider a model whose parameters are the set of estimates $\theta_t = \{q_i = \hat{E}_t[r_{t+1}|a_t = i] \mid i \in \mathcal{A}\}$, then the optimal choice is to select a_t for which the estimated expected value of the reward is highest, because according to our current belief any other choice will necessarily lead to a lower expectation. Thus, stating the bandit problem in this way does not allow the exploration of seemingly lower, but potentially higher value actions and it results in a *greedy* policy.

In the second formulation, we wish to minimise the discrepancy between our estimate q_i and the true expectation. This could be written as the following minimisation problem:

$$\sum_{i \in \mathcal{A}} E\big[\|r_{t+1} - q_i\|^2 \mid a_t = i\big].$$

For point estimates of the expected reward, this requires sampling *uniformly* from all actions and thus represents a purely exploratory policy. If the problem is stated as simply minimising the discrepancy asymptotically, then uniformity is not required and it is only necessary to sample from all actions infinitely often. This condition holds when $P(a_t = i) > 0 \ \forall i \in \mathcal{A}, \ t > 0$ and can be satisfied by mixing the optimal policies for the two formulations, with a probability ϵ of using the uniform action selection and a probability $1 - \epsilon$ of using the greedy action selection. This results in the well-known ϵ-greedy policy (see Sutton and Barto, 1998, for example), with the parameter $\epsilon \in [0, 1]$ used to control exploration.

This formulation of the exploration-exploitation problem, though leading to an intuitive result, does not lead to an obvious way to optimally select actions. In the following section we shall consider bandit problems for which the functional to be maximised is

$$E\left[\sum_{k=0}^{N} g(k) r_{t+k+1} \middle| \pi\right], \quad g(k) \in [0, 1], \ N \geq 0,$$

with $\sum_{k=0}^{\infty} g(k) < \infty$. In this formulation of the problem we are not only interested in maximising the expected reward at the next time step, but in the subsequent N steps, with the $g(\cdot)$ function providing another convenient way to weigh our preference among short and long-term rewards. Intuitively it is expected that the optimal policy for this problem will be different depending on how long-term are the rewards that we are interested in. As will be shown later, by lengthening the effective reward horizon through manipulation of g and N, i.e. by changing the definition of the problem that we wish to solve, the exploration bias is increased automatically.

7.2 Related work

There has been a considerable body of work in this field, which relates to the bound and the algorithms derived from it that are proposed in this chapter. Firstly, there are theoretical results that are relevant to the discussion. Secondly, similar methods that attempt to solve the same problem. Thirdly, are related methods that attempt to solve slightly different problems.

Specifically for bandit problems, the optimal, but intractable, Bayesian solution was given in (Bellman, 1957a), while recently tight bounds on the sample complexity of exploration have been found (Mannor and Tsitsiklis, 2004). In the more general case (not considered here) where each bandit has state with Markovian dynamics, Gittins indices (Gittins, 1989) can be used to formulate an optimal Bayesian solution. In fact, the result presented in this chapter is related to the proof of optimality of Gittins indices offered by Weber (1992), where instead of searching for the supremum of the indices over all policies we merely search for the supremum over a limited set of policies - in this case, the greedy policy and the 1-step exploratory and subsequently greedy policies. There has also been work on such problems with adversarial agents by Auer et al. (2002).

The work most similar to the one presented here was an approximation to the full Bayesian case for the general reinforcement learning problem, given by Dearden et al. (1998). Some of the approximations were with respect to the model inference. However we are not explicitly concerned with this in this chapter, simply assuming that there is some probabilistic model available. The remaining approximations were related to selecting actions given the model uncertainty. There are two approaches, presented, one of which is identical to Algorithm 3, while the other, a method based on the value of perfect information (VPI), is similar related to the bound on the value of exploration that is proposed here. The similarities will be further discussed in Section 7.7, however for the moment we will note that the VPI is used as an additional exploration bonus to the estimated value of an action, then the highest value is chosen.

Many approaches for trading exploration and exploitation rely on the notion of an *exploration bonus* to be added to the point estimate of the action evaluation. One representative such approach is Meuleau and Bourgine (1999), which additionally distributes these bonuses to linked states. In this thesis, a similar approach will be described in Chapter 8. In this and other cases, exploration is performed adaptively in a non-Bayesian manner by selecting the action that maximises an upper bound on a confidence interval for its value. One of the earliest such approaches was investigated by Kaelbling (1990), which, like all such methods, suffers from the significant drawback that a percentile for the confidence interval must be specified *a priori*. A more modern such method is E^3 (Kearns and Singh, 1998), which is extremely simple, general and effective.

A somewhat different approach from all of the above is that proposed by Karakoulas (1995). In this particular variant of Q-learning, a model of the environment is maintained, which can be sampled. The author then uses efficient methods to find optimal policies with high probability according to the current learnt model of the environment. However, this is not sufficient for performing optimal exploration and the author augments the utility function with a risk-aversion parameter, which includes the effects of return variance in the utility function without making an attempt to separate model uncertainty from environmental stochasticity. Due to these reasons, this approach is not directly comparable with those that simply attempt to act optimally under a specific cost and belief probability distribution. On the other hand, the sampling-based approach is very promising: similar techniques shall be employed in order to implement the algorithms that will be developed in this section.

Other related work includes an alternative technique based on eliminating actions which are confidently estimated as low-value. This approach was used by Even-Dar et al. (2006). Bresina et al. (2002) did some work on continuous time planning under uncertainty, while Zlochin et al. (2004) worked on model-based combinatorial optimisation. A slightly different technique from the full Bayesian approach to estimating uncertainty was given by Wyatt (2001), where an optimistically evaluated state, which can never be reached is used to direct exploration. Reynolds (2001a) deals with problems related to having optimistic expectations about unknown quantities. In his thesis, Reynolds (2002) talks about how the desired problem can be solved while performing exploration - in particular it talks about methods such as Q-learning, which ostensibly try and determine the value function for the optimal policy.

7.3 Optimal Exploration Threshold for Bandit Problems

We want to know when it is a better decision to take action i rather than some other action j, with $i, j \in \mathcal{A}$, given that we have estimates q_i, q_j for $E[r_{t+1}|a_t = i]$ and $E[r_{t+1}|a_t = j]$ respectively[1]. We shall attempt to see under which conditions it is better to take an action different than the one whose expected reward is greatest. For this we shall need the following assumption:

Assumption 7.1 (Expected rewards are bounded from below) *There exists $b \in \mathbb{R}$ such that*

$$E[r_{t+1}|a_t = i] \geq b \quad \forall\ i \in \mathcal{A}, \tag{7.1}$$

The above assumption is necessary for imposing a lower bound on the expected return of exploratory actions: no matter what action is taken, we are guaranteed that $E[r_t] > b$. Without this condition, exploratory actions would be too risky to be taken at all.

Given two possible actions to take, where one action is currently estimated to have a lower expected reward than the other, then it might be worthwhile to pursue the lower-valued action if the following conditions are true: (a) there is a degree of uncertainty such that the lower-valued action can potentially be better than the higher-valued one, (b) we are interested in maximising more than just the expectation of the next reward, but the expectation of a weighted sum of future rewards, (c) we will be able to accurately determine whether one action is better than the other quickly enough, so that not a lot of resources will be wasted in exploration.

We now start viewing q_i as random variables for which we hold belief distributions $p(q_i)$, with $\bar{q}_i = E[q_i] = \hat{E}[r_{t+1}|a_t = i]$. The problem can be defined as deciding when action i, is better than taking action j, under the condition that doing so allows us to determine whether $q_i > q_j + \delta$ with high probability after $T \geq 1$ exploratory actions. For this reason we will need the following bound on the expected return of exploration.

[1]For bandit problems with states in a state space $\mathcal{S}$, similar arguments can be made by considering $i, j \in \mathcal{S} \times \mathcal{A}$.

Lemma 7.1 (Exploration bound) *For any return of the form $R_t = \sum_{k=0}^{N} g(k) r_{t+k+1}$, with $g(k) \geq 0$, assuming* (7.1) *holds, the expected return of taking action i for T time-steps and following a greedy policy thereafter, when $\bar{q_i} > \bar{q_j}$, is bounded below by*

$$U(i,j,T,\delta,b) = \sum_{k=T}^{N} g(k)\big((\bar{q_j}+\delta)P(q_i > q_j+\delta) + \bar{q_j}P(q_i \leq q_j + \delta)\big) \\ + \sum_{k=0}^{T-1} g(k)\big((\bar{q_j}+\delta)P(q_i > q_j+\delta) + bP(q_i \leq q_j+\delta)\big) \quad (7.2)$$

for some $\delta > 0$.

This follows immediately from Assumption 7.1. The greedy behaviour supposes we are following a policy where we continue to perform i if we know that $P(q_i > q_j + \delta) \approx 1$ after T steps and switch back to j otherwise.

Without loss of generality, in the sequel we will assume that $b = 0$ (If expected rewards are bounded by some $b \neq 0$, we can always subtract b from all rewards and obtain the same). For further convenience, we set $p_i = P(q_i \geq q_j + \delta)$. Then we may write that we must take action i if the expected return of simply taking action j is smaller than the expected return of taking action i for T steps and then behaving greedily, i.e. if the following holds:

$$\sum_{k=0}^{N} g(k)\bar{q_j} < \sum_{k=T}^{N} g(k)\big((\bar{q_j}+\delta)p_i + \bar{q_j}(1-p_i)\big) + \sum_{k=0}^{T-1} g(k)(\bar{q_j}+\delta)p_i \quad (7.3)$$

$$\sum_{k=0}^{T-1} g(k)\big(\bar{q_j} - (\bar{q_j}+\delta)p_i\big) < \sum_{k=T}^{N} g(k)\big(\delta p_i\big) \quad (7.4)$$

Let $g(k) = \gamma^k$, with $\gamma \in [0,1]$. In this case, any choice of T can be made equivalent to $T = 1$ by dividing everything with $\sum_{k=0}^{T-1} \gamma^k$. We explore two cases: $\gamma < 1,\ N \to \infty$ and $\gamma = 1,\ N < \infty$. In the first case, which corresponds to infinite horizon exponentially discounted reward maximisation problems, we obtain the following:

$$\bar{q_j} - (\bar{q_j}+\delta)p_i < \sum_{k=1}^{\infty} \gamma^k \delta p_i \quad (7.5)$$

$$\frac{\bar{q_j} - (\bar{q_j}+\delta)p_i}{(1-p_i)\bar{q_j}} < \gamma. \quad (7.6)$$

It possible to simplify this expression considerably. When $P(q_i \geq \bar{q_j} + \delta) = 1/2$, it follows from (7.6) that

$$\gamma > \frac{\bar{q_j} - (\bar{q_j}+\delta)/2}{\bar{q_j}/2} = \frac{\bar{q_j} - \delta}{\bar{q_j}}. \quad (7.7)$$

Thus, for infinite horizon discounted reward maximisation problems, when it is known that the all expected rewards are non-negative, all we need to do is find δ such that $P(q_i \geq q_j + \delta) = 1/2$. Then (7.7) can be used to make a decision on whether it is worthwhile to perform exploration. Although it might seem strange the q_i is omitted from this expression, its value is implicitly expressed through the value of δ.

In the second case, finite horizon cumulative reward maximisation problems, exploration should be performed when the following condition is satisfied:

$$N\delta p_i > \bar{q_j} - (\bar{q_j} + \delta)p_i \tag{7.8}$$

Here the decision making function is of a different nature, since it depends on both estimates. However, in both cases, the longer the effective horizon becomes and the larger the uncertainty is, the more the bias towards exploration is increased. We furthermore note that in the finite horizon case, the backward induction procedure can be used to make optimal decisions (see DeGroot, 1970, Sec. 12.4).

7.3.1 Solutions for Specific Distributions

If we have a specific form for the distribution $P(q_i > q_j + \delta)$ it may be possible to obtain analytical solutions. To see how this can be achieved, consider that from (7.6), we have:

$$\begin{aligned} \gamma\bar{q_j} &> \bar{q_j} - \delta\frac{p_i}{1-p_i} \\ 0 &< \delta\frac{P(q_i > q_j + \delta)}{1 - P(q_j > q_j + \delta)} - (1-\gamma)\bar{q_j}, \end{aligned} \tag{7.9}$$

recalling that all mean rewards are non-negative.

If this condition is satisfied for some δ then exploration must be performed. We observe that if the first term is maximised for some δ^* for which the inequality is not satisfied, then there is no $\delta \neq \delta^*$ that can satisfy it. Thus, we can attempt to examine some distributions for which this δ^* can be determined. We shall restrict ourselves to distributions that are bounded below, due to Assumption 7.1.

7.3.2 Solutions for the Exponential Distribution

One such distribution is the exponential distribution, defined as

$$P(X > \delta) = \int_\delta^\infty \beta e^{-\beta(x-\mu)}dx = e^{-\beta(\delta-\mu)}$$

if $\delta > \mu$, 1 otherwise. We may plug this into (7.9) as follows

$$f(\delta) = \delta\frac{P(q_i > q_j + \delta)}{1 - P(q_i > q_j + \delta)} = \delta\frac{e^{-\beta_i(\mu_j+\delta-\mu_i)}}{1 - e^{-\beta_i(\mu_j+\delta-\mu_i)}} = \frac{\delta}{e^{\beta_i(\mu_j+\delta-\mu_i)} - 1}$$

Now we should attempt to find $\delta^* = \arg\max_\delta f(\delta)$. We begin by taking the derivative with respect to δ. Set $g(\delta) = e^{h(\delta)} - 1$, $h(\delta) = \beta_i(\bar{q}_j + \delta - \mu_i)$

$$\nabla f(\delta) = \frac{g(\delta) - \delta \nabla g(\delta)}{g(\delta)^2} = \frac{g(\delta) - \delta \beta_i \nabla_h g(\delta)}{g(\delta)^2} = \frac{e^{h(\delta)}(1 - \delta\beta_i) - 1}{(e^{h(\delta)} - 1)^2}$$

Necessary and sufficient conditions for some point δ^* to be a local maximum for a continuous differentiable function $f(\delta)$ are that $\nabla_\delta f(\delta^*) = 0$ and $\nabla^2_\delta f(\delta^*) < 0$. The necessary condition for δ results in

$$e^{\beta_i(q_k + \delta - \mu_i)}(1 - \delta\beta_i) = 1. \tag{7.10}$$

Unfortunately (7.10) has no closed form solution, but it is related to the Lambert W function for which iterative solutions do exist (Corless et al., 1996). The found solution can then be plugged into (7.9) to see whether the conditions for exploration are satisfied.

7.4 Extension to the General Case

In the general reinforcement learning setting, the reward distribution does not only depend on the action taken but additionally on a state variable. The state transition distribution is conditioned on actions and has the Markov property. Each particular task within this framework can be summarised as a Markov decision process:

Definition 7.1 (Markov decision process) *A Markov decision process is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition distribution $\mathfrak{T}(s', s, a) = P(s'_{t+1}|s_t = s, a_t = a)$ and a reward distribution $\mathfrak{R}(s', s, a) = p(r_{t+1}|s_{t+1} = s', s_t = s, a_t = a)$.*

The simplest way to extend the bandit case to the more general one of MDPs is to find conditions under which the latter reduces to the former. This can be done for example by considering choices not between simple actions but between *temporally extended actions*, which we will refer to as *options* following Sutton et al. (1999). We shall only need a simplified version of this framework, where each possible *option x* corresponds to some policy $\pi^x : \mathcal{S} \times \mathcal{A} \to [0, 1]$. This is sufficient for sketching the conditions under which the equivalence arises.

In particular, we examine the case where we have two options. The first option is to always select actions according to some exploratory principle, such picking them from a uniform distribution. The second is to always select actions greedily, i.e. by picking the action with the highest expected return.

We assume that each option will last for time T. One further necessary component for this framework is the notion of mixing time

Definition 7.2 (Exploration mixing time) *We define the exploration mixing time for a particular MDP $\mathcal{M}$ and a policy π $T_\epsilon(\mathcal{M}, \pi)$ as the expected number of time steps after which the state distribution is close to the stationary state distribution of π after we have taken an exploratory action i at time step*

t, i.e. the expected number of steps T such that the following condition holds:

$$\frac{1}{\|\mathbb{S}\|}\sum_s \|P(s_{t+T} = s|s_t, \pi) - P(s_{t+T} = s|a_t = i, s_t, \pi)\| < \epsilon$$

It is of course necessary for the MDP to be ergodic for this to be finite. If we only consider switching between options at time periods greater than $T_\epsilon(\mathcal{M}, \pi)$, then the option framework's roughly corresponds to the bandit framework, and T_ϵ in the former to T in the latter. This means that whenever we take an exploratory action i (one that does not correspond to the action that would have been selected by the greedy policy π), the distribution of states would remain to be significantly different from that under π for $T_\epsilon(\mathcal{M}, \pi)$ time steps. Thus we could consider the exploration to be taking place during all of T_ϵ, after which we would be free to continue exploration or not. Although there is no direct correspondence between the two cases, this limited equivalence could be sufficient for motivating the use of similar techniques for determining the optimal exploration exploitation threshold in full MDPs.

7.5 Optimistic Evaluation

In order to utilise Lemma 7.1 in a practical setting we must define T in some sense. The simplest solution is to set $T = 1$, which results in an optimistic estimate for exploratory actions as will be shown below. By rearranging (7.2) we have

$$U(i, j, T, \delta, b) = \sum_{k=0}^{N} g(k)\bar{q_j} + \sum_{k=0}^{N} g(k)\delta p_i + (1 - p_i)\left(\sum_{k=0}^{T-1} g(k)(b - \bar{q_j}))\right) \tag{7.11}$$

from which it is evident, since $q_j \geq b$ and $g(k) \geq 0$, that $U(i, j, T_1, \delta, b) \geq U(i, j, T_2, \delta, b)$ when $T_1 < T_2$, thus $U(i, j, 1, \delta, b) \geq U(i, j, T, \delta, b)$ for any $T \geq 1$. This can now be used to obtain Algorithm 1 for optimistic exploration.

Nevertheless, testing for the existence of a suitable δ can be costly since, barring an analytic procedure it requires an exhaustive search. On the other hand, it may be possible to achieve a similar result through sampling for different values of δ. Herein, the following sampling method is considered: Firstly, we determine the action j with the greatest $\bar{q_j}$. Then, for each action i we take a sample x from the distribution $p(q_i)$ and set $\delta = x - \bar{q_j}$. This is quite an arbitrary sampling method, but we may expect to obtain a $\delta > 0$ with high probability if i has a high probability[2] to be significantly better than j. This method is summarised in Algorithm 2.

An alternative exploration method is given by Algorithm 3, which samples each action with probability equal to the probability that its expected reward is the highest. It can perhaps be viewed as a crude approximation to Algorithm 2 when $\gamma \to 1$ and has the advantage that it is extremely simple.

[2]Even if the probability of sampling a positive δ is low, one may simply increase the number of samples taken to better approximate Algorithm 1.

Algorithm 1 Optimistic exploration

```
if $\exists\ \delta\ :\ U(i, j, 1, \delta, b) > \sum_{k=0}^{N} g(k)\bar{q_j}$ then
  $a \Leftarrow i$
else
  $a \Leftarrow j$
end if
```

Algorithm 2 Optimistic stochastic exploration

```
$j \Leftarrow \arg\max_i \bar{q}_i$.
$u_j = \sum_{k=0}^{N} g(k)\bar{q_j}$.
for all $i \neq j$ do
  $\delta \Leftarrow x - \bar{q_j}, \quad x \sim p(q_i)$
  $u_i \Leftarrow U(i, j, 1, \delta, b)$
end for
$a \Leftarrow \arg\max_i u_i$
```

7.6 Experiments on bandit problems

We will study the performance of the algorithm on random bandit problems. Such problems are interesting mainly because the allow one to analyse the behaviour of the algorithm easily and represent the simplest case under which an exploration-exploitation trade-off arises. More specifically, we consider n-armed bandit problems with rewards $r_t \in \{0, 1\}$ drawn from a Bernoulli distribution. For each experimental run, the expectations $p_i \equiv P(r_t = 1|a_t = i)$ were drawn from a uniform distribution on $[0, 1]$. Unless otherwise noted, the expected rewards are stationary. Under these conditions, the expected reward is $E[\max(p_i)] = \frac{n}{n+1}$ under the oracle policy. [3] Algorithm 2 was used with $g(k) = \gamma^k$ and $b = 0$, which is in agreement with the distribution. This was compared with Algorithm 3, which can be perhaps viewed as a crude approximation to Algorithm 2 when $\gamma \to 1$. The performance of ϵ-greedy action selection with $\epsilon = 0.01$ was evaluated for reference.

The ϵ-greedy algorithm used point estimates for $\bar{q}_i$, which were updated with gradient descent with a step size of $\alpha = 0.01$, such that for each action-reward observation tuple $(a_t = i, r_{t+1})$, $\bar{q}_i \Leftarrow \alpha(r_{t+1} - \bar{q}_i)$, with initial estimates being uniformly distributed in $[0, 1]$.

For Algorithms 2 and 3 the distribution of q_i was represented through a population $\{p_i^k\}_{k=0}^K$ of point estimates, with $K = 16$. Each point estimate in the population was maintained in the same manner as the single point estimates in the ϵ-greedy approach, with each point estimate being updated independently of the others. Sampling actions values was performed by sampling uniformly from the members of the

[3]The oracle policy is optimal policy if the expected rewards are known exactly. As for the expected maximum, it arises because for a sample of independent random variables $(X_1, X_2, \ldots, X_n)$ drawn from a uniform distribution on $[0, 1]$, we can write that $P(max(X) < x) = P(X_1 < x, X_2 < x, \ldots, X_n < x) = P(X < x)^n$. On the $[0, 1]$ interval, this is the c.d.f. $c(x) = x^n$, whose p.d.f is just $p(x) = nx^{n-1}$ and $E[X] = p(x) = nx^(n - 1)$.

Algorithm 3 Sampling-greedy

```
$a \Leftarrow i$ with probability $P(a = i) = P(q_i > q_j)\ \forall j \neq i$
```

population for each action. This approach is further discussed in Section 8.5.

The results for two different bandit tasks, one with 16 and the other with 128 arms, averaged over 1,000 runs, are summarised in Figure 7.1. For each run, the expected reward of each bandit was sampled uniformly from $[0, 1]$. As can be seen from the figure, the ϵ-greedy approach performs relatively well

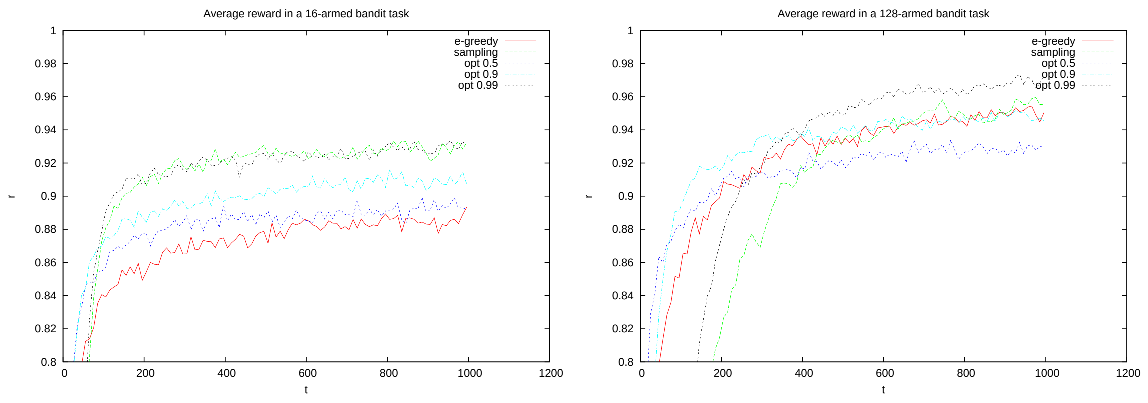


Figure 7.1: Average reward in an multi-armed bandit task averaged over 1,000 experiments, smoothed with a moving average over 10 time-steps. Results are shown for ϵ-greedy (**e-greedy**), sampling-greedy (**sampling**) and Algorithm 2 (**opt**) with $\gamma \in \{0.5, 0.9, 0.99\}$.

when used with reasonable first initial estimates. The sampling greedy approach, while having the same complexity, appears to perform better asymptotically. More importantly, Algorithm 2 exhibits better long-term versus short-term performance when the effective reward horizon is increased as $\gamma \to 1$. Ideally we would like the algorithm to be optimal for the value of γ which has been selected, i.e. that the algorithm achieving maximal expected return for a particular γ is the one whose objective function uses the same γ. However, since the algorithms is slightly optimistic, it is expected that it will actually be optimal for a value of γ slightly higher than what has been selected. Nevertheless, this will also depend on a number of addition factors, including the method used to estimate and sample from the distribution of expected rewards. More experiments analysing this behaviour will be presented in Section 8.5.3.

7.7 Discussion and conclusion

This chapter has presented a formulation of an optimal exploration-exploitation threshold for a n-armed bandit task, which links the need for exploration to the effective reward horizon and model uncertainty. Additionally, practical algorithms, based on an optimistic bound on the value of exploration, are introduced. Experimental results show that this algorithm exhibits the expected long-term versus short-term performance trade-off when the effective reward horizon is increased in the problem specification.

While the above formulation fits well within a reinforcement learning framework, other useful formulations may exist. In *budgeted learning*, any exploratory action results in a fixed cost. Such a formulation is used in (Madani et al., 2004a) for the bandit problem (see also Madani et al., 2004b, for the active learning case). Then the problem essentially becomes that of how to best sample from actions in the next T moves such that the expected return of the optimal policy *after* T moves is maximised and corresponds to $g(k) = 0 \; \forall k < T$ in the framework presented in this chapter. A further alternative, described in (Even-Dar et al., 2006), is to stop exploring those parts of the state-action space which lead to sub-optimal returns with high probability.

When a distribution or a confidence interval is available for expected returns, it is common to use the optimistic side of the confidence interval for action selection (Auer, 2005, 2002, for example). This practice can be partially justified through the framework presented herein, or alternatively, through considering maximising the expected information to be gained by exploration, as proposed by Bernardo (1979). In a similar manner, other methods which represent uncertainty as a simple additive factor to the normal expected reward estimates, acquire further meaning when viewed through a statistical decision making framework. For example the Dyna-Q+ algorithm (see Sutton and Barto, 1998, chap. 9) includes a slowly increasing *exploration bonus* for state-action pairs which have not been recently explored. From a statistical viewpoint, the exploration bonus corresponds to a model of a non-stationary world, where uncertainty about past experiences increases with elapsed time elapsed.

It is of interest to note one action selection method that is similar to the optimistic and optimistic-stochastic algorithms presented in this chapter. This is a method based on the value of perfect information (VPI), as originally developed by Howard (1966) and used in (Dearden et al., 1998, 1999) in order to select actions in a Bayesian reinforcement learning setting. The similarity is due to the fact that the model also considers the potential gains of selecting an action other than the greedy one, though the formulation is slightly different. The VPI action selection method always selects the action that maximises the expected return plus the expected gain due to acquired information. This expected gain is defined with respect to action i, the action with maximum expected return $U_i \equiv \hat{E}[R_t | a_t = i] = \bar{q}_i/(1 - \gamma)$. and action i', the second best action. More specifically, we sample the distribution of rewards [4] and obtain samples q_j for each action $j \in \mathcal{A}$ (or MDPs $m \in \mathcal{M}$). We then use U_j^* to denote the expected return of the optimal policy in such a sample after selecting action j q_j. This will simply be

$$U_j^* \equiv q_j + \frac{\gamma}{1 - \gamma} \max_k q_k \tag{7.12}$$

for the stationary bandit problem (for complete MDPs this value will have to be calculated using dynamic programming for each sample taken). Then, the *gain* of taking action j given U_j^* is

$$G_j(U_j^*) = \begin{cases} U_{i'} - U_j^* & \text{if} j = i, U_j^* < U_{i'} \\ U_j^* - U_i & \text{if} j \neq i, U_j^* > U_i \\ 0 & \text{otherwise,} \end{cases} \tag{7.13}$$

where the expected return of the best and second-best action is U_i and $U_{i'}$ respectively. Finally, the

[4] Or more generally the distribution of MDPs (see Definition 2.2, page 13). This requires maintaining a distribution over the set of possible MDPs and can be done in for example in the manner described by Dearden et al. (1999).

expected value of perfect information about action j is defined as

$$VPI(j) = \int_{-\infty}^{\infty} G_j(x)p(U_j^* = x)dx \tag{7.14}$$

and the action chosen is that which maximises

$$U_j + VPI(j).$$

The main similarity between the methods is the evaluation of the optimal policy after some time of exploration. The main differences are (a) In the proposed formulation we consider a lower bound for the expected rewards. This excludes certain types of beliefs, such as Gaussians, for the distribution of expected rewards. (b) The VPI method makes use of a second-best action. (c) Algorithms 1 and 2 search over and sample from values of δ, while VPI performs an integration. In order to see whether those differences have a significant impact on the performance of the algorithm, we will experimentally test Algorithms 2 and 3 with both VPI and E^3 (Kearns and Singh, 1998) in the next chapter.

In general, the conditions defined in Section 7.3, as well as the probabilistic formulation of Dearden et al., require maintaining some type of belief over the expected return of actions in order to be able to make an informed choice between exploratory and greedy behaviour. A natural choice for this would be to use a fully analytical Bayesian framework. In some cases the analytical expression is not possible - for example if we consider an exponential prior for q_i, and a Gaussian distribution for rewards given their expectation, we obtain a second-order exponential family posterior distribution with finite support. In other cases, such as when we use a beta prior and a Bernoulli distribution for the rewards given their expectation, we can remain within the analytical framework. Unfortunately, even in the latter case it can be difficult to calculate $P(q_i - q_j > \delta)$ for Algorithm 2, or for calculating (7.14) in the VPI framework. Thus, it might be better to consider simple numerical approaches from the outset. The following chapter considers two such types of estimates. The first one relies on estimating the gradient of the return with respect to the parameters. Then the estimated gradient is used as a measure of parameter variance. This very simple method is tested against a few baseline systems using the sampling-greedy approach. The estimates of the second type are more sophisticated. They attempt to explicitly represent a belief distribution for the rewards of each action.[5] Within that framework we consider three further cases: (a) A population of independent estimators (b) a Monte Carlo approximation of Bayesian inference and (c) fully Bayesian inference. The methods are tested in combination with optimistic stochastic exploration, sampling-greedy and VPI and against simple ϵ-greedy action selection and E^3.

[5]The extension to the full reinforcement learning case has not been considered further than the sketch given in Section 7.4, thus we do not present methods for maintaining distributions over MDPs.

Chapter 8

Estimates of return distributions

As seen in the previous chapter, the exploration-exploitation trade-off that arises when one considers simple point estimates of expected returns no longer appears under the Bayesian framework when full distributions are considered. This chapter aims to develop methods for maintaining such distributions in arbitrary models. These distributions can then be used to direct exploration, through one of the algorithms developed in Chapter 7. While that chapter had focused mostly on the development of nearly optimal algorithms, this chapters attempts to develop and examine methods for the representation of uncertainty. It is found that there is an interaction between the exploration algorithm, the representation of uncertainty and the problem on which the methods are tested.

The first such method is one of which had originally been presented in (Dimitrakakis and Bengio, 2005c). It is based on a simple gradient-based model for the estimation of the accuracy with which a parameter is known. This model is applied to the estimation of probability distributions of returns over actions in value-based reinforcement learning, where each parameter corresponds to the expected return of a different state-action pair under a particular policy. While this approach is similar to other techniques that maintain a confidence measure for action-values, it nevertheless offers an insight into other techniques that use the gradient with respect to the parameters as an accuracy measure. The greatest advantages of the method are its simplicity and ease of integration with existing action-value reinforcement learnign algorithms. On the other hand, it is limited by the fact that it does not have direct relation to probabilistic estimation and that since distributions are maintained over expected returns, only Algorithm 3 is applicable.

In order to overcome the limitations of the gradient-based techniques an attempt was made to move towards an approximation of fully Bayesian inference through the use of ensemble approaches for representing and estimating uncertatinty. Two such approaches are investigated: An independent estimator approach related to bagging (Breiman, 1996) and a randomised grid filter approach related to particle filtering (see Casella et al., 1999). The first is a rather ad-hoc method, while the second one approximates posterior distributions via a mixture model. We test the two approaches against an analytical Bayesian estimation procedure (which has a closed-form solution under the randomised bandit problem considered) and against other current methods for optimal exploration in reinforcment learning.

8.1 Gradient-based estimates

A large number of problems in both supervised and reinforcement learning are solved with parametric methods. In this framework we attempt to approximate a function $f^*(\cdot)$ via a parameterised function $f(\theta, \cdot)$, given samples of f^*, with parameters $\theta \in \mathbb{R}^n$. We focus on incremental optimisation methods for which an optimisation operator $\mathcal{M}(C, \theta)$, where C is an appropriately defined cost, can be defined as a stochastic process that is continuous with respect to θ. We define the sequence $\{\theta\}$ as $\theta_{t+1} = \mathcal{M}(C, \theta_t)$.

In reinforcement learning, samples of f^* are generated actively. Asymptotic convergence results exist for such methods under appropriate sampling assumptions (Jaakkola et al., 1994). If we maintain a distribution of θ_t (rather than a point estimate), we may be able to use it to generate samples in an optimal sense. For example such a distribution might be used to determine whether (7.7) holds.

In this Section we explore simple gradient-based methods for measuring the accuracy of our parameter estimates [1]. More specifically, we explore ways to measure the accuracy of parameters estimated via stochastic gradient descent methods. This involves estimating the gradient vector (and possibly the Hessian) and subsequently using it as a measure of convergence. Two cases are considered: variance estimates and gradient estimates. A naive variance estimate, arising from smoothness assumptions, is given and its relation to the gradient is detailed. The relation of the gradient to convergence is outlined and finally a simple gradient estimate is given.

8.1.1 Variance estimates

In the general setting, for each θ_t we sample a single value M_t from $\mathcal{M}(C, \theta_t)$, where $\mathcal{M}$ is considered as a random process. In our setting we will attempt to also maintain a confidence measure for our parameters. We will attempt to do this by measuring the variance of the process at the current point θ_t.

Firstly, we assume that M_t is bounded[2] and we attempt to estimate $\hat{E}[M_t] \approx E[M_t]$.

We may further assume that $\mathcal{M}$ is Lipschitz continuous with respect to θ, (a function f satisfies a Lipschitz continuity assumption in some set S if there exists $L \in \mathbb{R}$ such that $\|\nabla f(a) - \nabla f(b)\| \leq L\|a - b\|$ for all $a, b \in S$). An alternative, though not strictly equivalent, way of expressing this continuity is to place a prior over time for the statistics of the operator. The following simple relation follows from the assumption of an exponential prior dependency (see Appendix B.2) for the variance of the operator $\mathcal{M}$:

$$V_{t+1} = (1 - \zeta)V_t + \zeta(\hat{E}[M_t] - \theta_{t+1})(\hat{E}[M_t] - \theta_{t+1})', \tag{8.1}$$

with $\zeta \in [0, 1]$, where we have but one sample of $\mathcal{M}(C, \theta_t)$ for each time t and we make use of our smoothness assumptions for estimating variances. Now we may use V_t for our estimate of the variance of $\mathcal{M}(C, \theta_t)$.

In order to get useful estimates, we need some expressions for $\hat{E}[M_t]$. We examine the two simplest cases:

[1] Intuitively, when the gradient with respect to the parameters is close to zero, our algorithm has converged

[2] For stochastic gradient methods, under the condition that the partial derivative of the cost with respect to the parameters is bounded, all M_t are bounded.

Definition 8.1 (Naive variance estimate) *By assuming that $\mathcal{M}$ is a zero-mean process, i.e. that $E[M_t] = \theta_t$, we have:*

$$V_{t+1} = (1-\zeta)V_t + \zeta(\theta_t - \theta_{t+1})(\theta_t - \theta_{t+1})'. \tag{8.2}$$

Definition 8.2 (Counting variance estimate) *By assuming $E[M_t] = \theta_{t+1}$, e.g. when $\mathcal{M}$ is a deterministic process, we have:*

$$V_{t+1} = (1-\zeta)V_t. \tag{8.3}$$

The latter method is equivalent to a class of counting schemes whereby we increase our certainty about the mean of some random variable with each observation. With an appropriate choice for ζ such schemes can be adequate for some problems.

If it is desirable to disallow the estimate reaching zero, then we may want to clamp it to a lower limit. This can be achieved simply by adding a small positive constant to the above updates, or by just limiting the variance to always be larger or equal to a fixed *variance threshold*. The latter method is the one employed herein.

In the case where we maintain a set of parameters which are updated separately (such as in tabular reinforcement learning methods, which are further discussed in Section 8.2.2), then it is also appropriate to maintain separate variance estimates. In the following section we discuss how such estimates are related to the convergence of the stochastic operator $\mathcal{M}$ for the case when it expresses a stochastic gradient descent step.

8.1.2 Relation of variance estimates to convergence

In estimation problems we wish to find the unknown parameters θ^* from observations. In iterative estimation methods we would like to the distance between the current estimate θ and the unknown parameters, so that the process can be stopped. Unfortunately, estimating $|\theta - \theta^*|$, the distance to a solution, can be as difficult as determining θ^* itself. On the other hand, it is generally not possible to determine convergence, in certain special cases it presents a manageable task. To give a simple example, when the cost surface is quadratic (i.e $C = a(\theta^* - \theta)^2$) we have $|\theta^* - \theta| = a|\nabla_\theta C|$ and the magnitude of the steps we are taking is directly related to the convergence. It is easy to show that the mean update we have defined is an approximate measure of the gradient under some conditions.

From (8.1), we have

$$\begin{aligned} V_{t+1} &= (1-\zeta)V_t + \zeta\eta(\delta_t + e_t)'(\delta_t + e_t) \\ &= (1-\zeta)^t V_1 + \sum_{k=1}^{t}(1-\zeta)^{t-k}\zeta\eta(\delta_k + e_k)'(\delta_k + e_k) \\ &= (1-\zeta)^t V_1 + \zeta\eta(\sum_{k=1}^{t}(1-\zeta)^{t-k}\delta_k'\delta_k + \sum_{k=1}^{t}(1-\zeta)^{t-k}e_k'e_k) + 2\sum_{k=1}^{t}(1-\zeta)^{t-k}\delta_k'e_k) \end{aligned} \tag{8.4}$$

where e_k is a noise process such as the stochastic gradient error term. For the case when $\eta = 1/t$ we have, with better approximation as $t \to \infty$, and if $\delta_k = C(\theta)$ for all k (i.e. when $\alpha \to 0$)

$$\operatorname{trace}(V) \propto \|\nabla C(\theta)\|^2 + E[e^2],$$

where the right hand term from the stochastic gradient method is proportional to the variance of the observation noise in the linear case.

8.1.3 Gradient estimates

The relation of those estimates to the gradient is of interest because of the relationship of the gradient to the distance from the minimum under certain conditions. In particular, when $\nabla^2 C(\theta)$ is positive definite, the following holds:

Lemma 8.1 *Let θ^* be a local minimum of C and $\theta \in S$, with $S = \{\theta : \|\theta - \theta^*\| < \delta\}$, $\delta > 0$. If there exists $m > 0$ such that*

$$m\|z\|^2 \leq z'\nabla^2 C(\theta)z, \quad \forall\ z \in \mathbb{R}^n, \tag{8.5}$$

then every $\theta \in S$ satisfying $\|\nabla C(\theta)\| \leq \epsilon$ also satisfies

$$\|\theta - \theta^*\| \leq \epsilon/m, \quad C(\theta) - C(\theta^*) \leq \epsilon^2/m.$$

The proof is quite straightforward and is given in Appendix B.5.1. We may now define a simple estimate for the gradient itself.

Definition 8.3 (Gradient estimate) *By using similar assumptions as in the variance estimates, we may obtain an estimate of the gradient at time t:*

$$V_{t+1} = (1 - \zeta)V_t + \zeta(\hat{E}[M_t] - \theta_{t+1}) \tag{8.6}$$

Here V_t is our current estimate of the gradient vector and ζ represents our belief on how fast it changes between iterations. This can be seen by considering the equivalence of (8.6) and (B.14). In the latter, λ monotonically depends on the accuracy of the Gaussian in (B.11), which represents the prior knowledge on how fast the v_t may change with time.

Both the naive variance estimate and the gradient estimates can be used to determine convergence of parameters. It is perhaps interesting to note that for gradient methods with errors, the variance estimate includes the noise term. For reinforcement learning problems with stochastic rewards at each state or stochastic transitions between states this is significant, because it is related to the variance of the return. If we attempt to use such convergence criteria to select actions, either estimate may prove advantageous depending on the task.

8.2 Action selection

Most, if not all, reinforcement learning (RL) methods can be viewed as a combination of estimation and sampling. Given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, an agent selects actions $a \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The aim of reinforcement learning is described as finding a policy π^* that maximises a utility function, for which the only available information is reward samples r_t. The utility function we shall be examining is the expectation of the return given the policy (See Section 5.1 for details).

An important subset of reinforcement learning methods is formed by value-based methods (which are the focus of (Sutton and Barto, 1998)). These generate an evaluation for every possible action and state pair and the policy is defined in terms of this. State-action evaluations are usually noted in short-hand as $Q(s, a) = \hat{E}[R_t | s_t = s, a_t = a, \pi]$, i.e. the expected cost/return if we take action a at state s while following policy π. Value function updates typically employ temporal-difference methods, whereby parameters are adjusted in the direction of the temporal-difference error, which has the form $\delta = r_t + \gamma \hat{E}[R_{t+1} | s_{t+1}, a_t, \pi] - Q(s, a)$. In some cases parameters are adjusted according to an importance weight, which usually takes the form of an *eligibility trace* e_i, defined for each parameter θ_i.

8.2.1 Application of variance estimates to action values

These variance estimates can be applied with relative ease to action value reinforcement learning using either a tabular or an approximation architecture (see Section 5.2.2). The naive variance estimate (8.2) is particularly interesting because, for the tabular case, its use results in algorithm that is similar to (Sakaguchi and Takano, 2004). For this reason we shall concentrate on this particular estimate, but we will also be contrasting it to a gradient-related estimate.

In the following short sections we consider the application of such an estimate to reinforcement learning; firstly in the tabular and secondly in the function approximation case. Lastly, we describe action selection mechanisms, using the developed variance estimates, that can be applied to either case.

8.2.2 Tabular action value methods

The tabular reinforcement learning case can be obtained by defining a θ for each state-action pair Q, so that we maintain separate variance estimates for each one. Then we consider that at each time step the operator sample M_t can be defined as $M_t \equiv Q_{t+1}(s, a) = Q_t(s, a) + \alpha(r_t + \hat{E}[R_{t+1}] - Q_t(s, a))$. By substituting this into (8.2), we obtain

$$V_{t+1} = (1 - \zeta)V_t + \zeta \delta \delta', \tag{8.7}$$

where $\delta = Q_{t+1} - Q_t$ is the temporal-difference error vector (scaled by the step-size constant α). For the standard tabular case, all elements of δ will be 0 apart from the element corresponding to the action a, which is the one to be updated and the covariance matrix $\delta\delta'$ will have a single non-zero diagonal element.

By re-arranging the terms of (8.7) we arrive at

$$V_{t+1} - V_t = \zeta(\delta\delta' - V_t) \tag{8.8}$$

which can be written in expanded form as

$$V_{t+1}(s,a) - V_t(s,a) = \zeta(\delta(s,a) - V_t(s,a)). \tag{8.9}$$

In the following we briefly describe how eligibility traces can be integrated in our framework.

8.2.3 Eligibility traces and variance estimates

Let us assume that the return R_t is given by a probability distribution of the form $p(R_t|s_t, a_t, \pi)$. Clearly, we may estimate $E[R_t|s_t, a_t, \pi]$ by averaging the returns while following policy π. However, we can assume that the distribution of R_t depends upon the distribution of R_{t+1}. We assume an exponential distribution for this prior dependency and thus we have $p(R_{t+1}|s_{t+1}, a_{t+1}, \pi) = \lambda p(R_{t+1}|s_t, a_t, \pi) + (1-\lambda)\mathcal{W}$, where $\mathcal{W}$ is the distribution of some unknown process, while $\lambda \in [0, 1]$ represents our prior belief in the dependency between values at different times.

The relation to eligibility traces is clear. We assume that an exponential prior in time governs the probability distribution of R_t. Thus, we can perform importance sampling on our parameters through the use of this prior: in other words each new sample should influence each parameter according to its importance weight.

In RL methods employing eligibility traces, the update δ is applied to all the evaluations of all state-action pairs (s, a) proportionally to the eligibility trace $e(s, a)$. By viewing eligibility traces as importance weights we can integrate them easily with our variance estimates. This results in the following update for each parameter's estimate.

$$V_{t+1}(s,a) = (1 - \zeta e(s,a))V_t(s,a) + \zeta e(s,a)\delta\delta', \tag{8.10}$$

or in compact form

$$V_{t+1} = (I - \zeta e)V_t + \zeta e\delta\delta', \tag{8.11}$$

where I is the identity matrix.

8.2.4 Function approximation methods

We consider approaches where the value function is approximated with a parametrised function $Q_\theta : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$.

Gradient methods are a commonly used method for adapting the parameters θ. Given $\frac{\partial Q}{\partial \theta}\frac{\partial C}{\partial Q} \equiv \nabla_\theta Q \nabla_Q C$, we consider an update of the form $M_t = \theta_t + d_t$ for our parameters, where d_t is the gradient descent update. Then we simply apply (8.7) for this case and we obtain a covariance matrix for the parameters.

8.2.5 Methods for action selection

Two methods are proposed for making use the variance estimates. The first is an intuitive action selection mechanism for the case of linear approximations (including the tabular case), since the variance of Q depends linearly on those, which allows us to analytically define a distribution over actions. An alternative, and in our view more interesting, approach is to sample directly from the distribution of parameters. The two methods are described in more detail in the following.

Sampling actions from action distributions

Consider the probability $P(a^* = a|s)$ of action a being the optimal action for some state s. We need to obtain the posterior distribution of this for all actions, given the distribution of Q and the state, [3] denoted $P(a^* = a|Q, s)$. The Bayes rule gives

$$P(a^* = a|Q, s) = \frac{p(Q|a^* = a, s)P(a^* = a|s)}{\sum_{b \in \mathcal{A}} p(Q|a^* = b, s)P(a^* = b|s)}, \tag{8.12}$$

where we have made use of the fact that $\sum_{b \in \mathcal{A}} P(a^* = b|s) = 1$. Now we must assume a distribution family for $p(Q|a^*, s)$. We consider the Boltzmann distribution which can be written as

$$p(E|i) = \exp(-E_i/K\tau)$$

and has a physical interpretation of the distribution of the energies E of particles in a given state i depending on the temperature τ and a constant factor K. We will be using this in the following to define a softmax method for selection actions:

Definition 8.4 (weighted-softmax) *Select actions a according to probabilities:*

$$P(a^* = a|Q, s) = \frac{\exp(Q(s, a)/\sqrt{v_{s,a}})}{\sum_b \exp(Q(s, b)/\sqrt{v_{s,b}})} \ . \tag{8.13}$$

For the tabular case, $v_{s,a}$ at time t is simply $V_t(s, a)$. For the linear case, in which $Q(s, a) = \sum_i w_{i,a} s_i$, where w are components of a weight matrix and s_i is the i-th component of the state vector, the variance is simply $v_{s,a} = \sum_i w_{i,a} V_t(i, a)$ where $V_t(i, a)$ is the variance of the weight $w_{i,a}$. Of course we could also consider a full covariance matrix.

Another possibility is to consider an approximation to the action distribution, given the distribution of parameters, which is certainly more elegant. However, it is probably simpler to sample directly from the distribution of parameters and this is the approach we outline below.

[3] In our model Q is no longer a single value but a distribution characterised by the variance V. In this section we make no distinction between our estimate of the mean and the actual distribution in order to minimise symbol use.

Sampling actions via parameter sampling

The second method applies to the more general function approximation case, as well as the tabular case. Here we have to choose a distribution for our parameters; and then we sample from it in order to generate actions, rather than postulating a distribution over actions and sampling from that. This is because in the general case it is difficult to determine the distribution over actions from that of the parameters, while in any case it is sufficient to sample from the parameter distribution directly.

Definition 8.5 (Sampling-Greedy) *In this method, action sampling arises from sampling in the parameter space. At each time, we select action a^* such that*

$$a^* = \arg\max_a Q(s, a, \Theta), \tag{8.14}$$

where $\Theta = \mathcal{N}(\theta, V_t)$ is a sample randomly drawn from a normal distribution with mean θ and variance V_t.

This will select action a^* with probability $\prod_{a \in \mathcal{A}} P(Q(s, a^*) > Q(s, a)|\Theta)$, if we consider Θ as our belief distribution. This means that it will select each action with probability proportional to the probability of it being the best action. The reader will have noticed that this is almost the same as Algorithm 3, but with two differences. Firstly, the distribution is a special case. Secondly, in this case we are not sampling from reward, but from return distributions.

8.3 Alternative approaches

As discussed in Section 7.2, there have been previous applications of such methods to the problem of optimal action selection in reinforcement learning. Another method that is similar to the one presented here is that of Meuleau and Bourgine (1999), where they also estimate a local measure of uncertainty which is back-propagated in a similar manner to linked states. However, the closest approach to the gradient estimates proposed herein that we are currently aware of is the Reliability Index method, described in (Sakaguchi and Takano, 2004). This method has substantial similarity to our own for tabular action value methods using the naive variance update (8.2), so we pause for a moment to ponder the differences. In this method, actions are sampled according to a Boltzmann distribution:

Definition 8.6 (Reliability Index)

$$p(a|Q, s) = \frac{\exp(\phi Q(s, a)/\sqrt{v_s})}{\sum_{b \in A} \exp(\phi Q(s, b)/\sqrt{v_s})} \tag{8.15}$$

where $v_s > 0$ is defined $\forall\ s \in \mathcal{S}$ and is a variance estimate *for each one of our Q estimates and ϕ is a free parameter.*

In that method, the variance update assumes the form $V_{t+1}(s) - V_t(s) = \zeta(\delta(s)\delta(s)' + \gamma V_t(s') - V_t(s))$, with a common V for all actions, or of a type $V_{t+1}(s, a) - V_t(s, a) = \zeta(\delta(s)\delta(s)' + \gamma V_t(s', a) - V_t(s, a))$.

This is then averaged over all states to obtain $V_t(s) = \frac{1}{|\mathcal{A}|}\sum_a V_t(s,a)$. In either case, actions are selected according to (8.15). It is perhaps important to note that a temporal-difference type of update is used (since $V_t(s')$ is the estimated variance of next state's evaluation). The authors postulate that this represents the dependency between the reliability of one state and the ones close to it. In our view, parameter variances are directly related to parameter updates and γ is related to the utility function rather than to assumptions about parameter dependencies. However, a model for setting priors about parameter dependencies is offered by the exponential prior, commonly used in eligibility traces. This is the method we have employed, as explained in Section 8.2.3.

Because of the close relation between this method and ours, our results can be viewed as, firstly, a partial justification of the RI method and secondly, the generalisation of the concept to arbitrary action-value methods.

8.4 Experiments

In this section we discuss results on a few simple discrete state tasks: a n-armed bandit problem, graph walking and pole balancing. In the n-armed **bandit task**, the environment is composed of a single state and n actions, with each action a having a reward $r_a \in \{0,1\}$ governed by a Bernoulli distribution. In theis environment, we used the same set up as that described in section 7.6. It is important to note here a drawback of all those methods - while as seen in the previous chapter, we would normally like to maximise a specific utility function, such as $\sum_t E[r_t]\gamma^t$, the naive gradient-based methods explored herein do not take into account the horizon when estimating the value of exploration. Nevertheless, for a quantitative performance measure we estimate their cumulative expected return for different values of γ. An upper bound on the performance on this task can be given by the asymptotically best-achievable average[4], $E[\max(r)] = n/(n+1)$, i.e. the expected reward when the statistics of all arms are known exactly and we always pick the arm with the highest expected reward. Numerical results are given in terms of the ratio of performance of each method with respect to this, i.e. the ratio $\sum_{t=0}^{N}\gamma^t E[r|\cdot]/E[\max(r)]$. When $\gamma \to 1$ and $N \to \infty$ this ratio approaches one for all methods that asymptotically converge, so for this case another measure, the logarithmic regret, is used(see for example Auer, 2002).

The **graph task** consists of m nodes and n actions. Actions result in transitions from one node to another deterministically. The first action always results in a transition from one node to the next, forming a ring, i.e. a topologically closed path. Thus, starting from some node i it is possible to visit all nodes in m moves by simply choosing action 1. Other actions' transitions are determined randomly at the beginning of each experimental round. The reward distribution for each state is Bernoulli, with mean in $[0,b]$ selected randomly at the beginning of each round from the bounded distribution

$$P(X < x) = 1/b^2 \int_0^b \frac{b - x/t}{b} dt,$$

where b was set to 0.5.

[4]See also Section 7.6.

$F_s = K(L_0 - L)\|L_0 - L\| - K_d dL/dt$	spring force
$N = m_c g + F_s \cos(\theta)$	ground reaction
$F_f = -\tanh(n_k u)Nn$	ground friction
$du_b/dt = F_s/m_b + g$	ball acceleration
$dx_b/dt = u_b$	ball velocity
$du_c/dt = (F_{in} + F_f + F_s \sin(\theta))/m_c$	cart acceleration
$dx_c/dt = u_c$	cart velocity
$\omega = d\theta/dt$	angular velocity

Table 8.1: The above equations describe the simulation model in the pole-balancing task. The g vector has a downwards direction, while F_s is co-linear with the pole. F_{in} (the input force) and F_f are horizontal forces. The cart is constrained to move only horizontally.

8.4.1 The pole balancing task

For **the simulation** of the pole balancing task we used a discrete first-order approximation, performed with $dt = 0.01$, with a semi-elastic pole with a ball at one end and a cart sliding on a 4 meter long surface, with walls at either side, upon which the cart bounced elastically. The simulation details are shown in Table 8.1. The mass of the cart was $m_c = 1kg$, while that of the ball was $m_b = 0.1kg$. The pole was massless, elastic, with length $L_0 = 1m$, a stiffness of $K = 200Nm^{-1}$ and a damping factor of $K_d = 1Nsm^{-1}$. Gravitational acceleration was set to $g = 9.82Nm^{-2}$. The ground friction dependency on reaction force was factorised into a linear part, with $n = 0.05$ modelling the maximum possible friction given ground reaction N, and a non-linear part, with $n_k = 10$ giving the amount of non-linear dependence of friction with speed.

For **the controller** we used the following set-up. The state consisted of the angular velocity and angle of pole, and speed and position of the cart. These were discretised to form a state space of size 384, through the splitting the variables θ, ω, x_c, u_c into four regions each. The controller made decisions at a frequency of $20Hz$, choosing between 2 possible actions in $\mathcal{A}$. Each action resulted in a lateral force $F_{in} = \pm 10N$ being instantaneously applied to the cart, i.e. there was hysteresis or other engine effects. The reward was 0 for all time, apart from failure, when it was -1 and the apparatus was reset to a random position. For the reinforcement learning algorithms we used Q-learning updates, with a discounting factor $\gamma = 0.99$ and accumulating eligibility traces with $\lambda = 0.7$. The learning rate was set to $\alpha = 0.1$. For the adaptive exploration methods, the variance threshold was set to 0, such that they become purely greedy at the limit.

We have compared all the possible combinations of accuracy estimates (counting, naive and velocity), and action selection methods (sampling and weighted-softmax) against standard action selection techniques and the reliability index method. For tasks with state, we also explored the eligibility trace update option for variance estimates, as outlined previously. Each method has a free parameter, such as the temperature, ϵ, or ζ, which we varied in the range $[10^{-4}, 1]$.

8.4.2 Results

In the **bandit task**, the number of arms was varied in the set $\{16, 32, 64, 128\}$. For a small number of arms the various methods performed equally well. For increasing numbers of arms, the standard softmax and ϵ-greedy methods failed to reach a satisfying solution when not started with optimistic estimates, while the sampling method tended to perform slightly better than other methods. In this task, the variance threshold was set to 10^{-4} and the learning rate was set to a constant $\eta = 0.1$.

Figure 8.1 shows results of the best methods (Additional results showing how the performance of each method with different parameters are given in AppendixC.1) for 16 and 128 arms, plotted against the asymptotically best achievable result. The weighted-softmax was performing generally worse than the other adaptive methods. In this task we did not observe a large difference between the naive variance, the velocity and the simple counting estimate. Note that when using naive variance updates, the only difference between our method and RI lies in the action selection mechanism. It would thus appear to be strange that the weighted-softmax is performing worse than RI. However, this is probably due to the fact that, if all rewards are positive, actions with high variances are penalised by this method and it frequently halts quickly at a sub-optimal solution.

Pessimistic initial estimates

γ	greedy	sm c 0.9	sm c 0.999	sm v 0.9	sm v 0.999	ws c 0.999	ws v 0.999
0.000000	51%	55%	52%	50%	49%	52%	53%
0.500000	57%	54%	52%	51%	52%	56%	53%
0.900000	67%	55%	57%	58%	59%	66%	54%
0.990000	72%	79%	78%	82%	81%	73%	67%
0.999000	73%	90%	85%	90%	88%	75%	90%
1.000000	74%	92%	87%	91%	88%	76%	95%

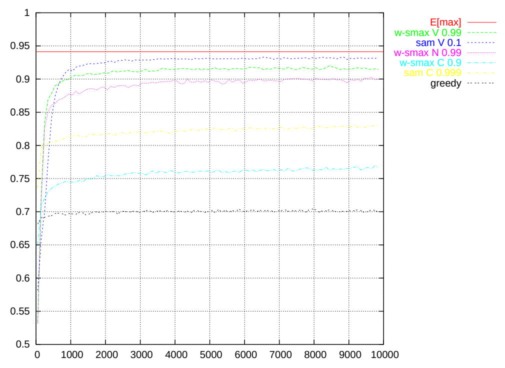
Optimistic initial estimates

γ	greedy	sm c 0.9	sm c 0.999	sm v 0.9	sm v 0.999	ws c 0.999	ws v 0.999
0.000000	50%	52%	54%	53%	50%	53%	54%
0.500000	52%	51%	53%	53%	50%	53%	59%
0.900000	59%	53%	57%	49%	48%	57%	69%
0.990000	82%	77%	81%	53%	47%	78%	82%
0.999000	95%	94%	95%	72%	70%	94%	90%
1.000000	98%	98%	98%	81%	79%	98%	91%

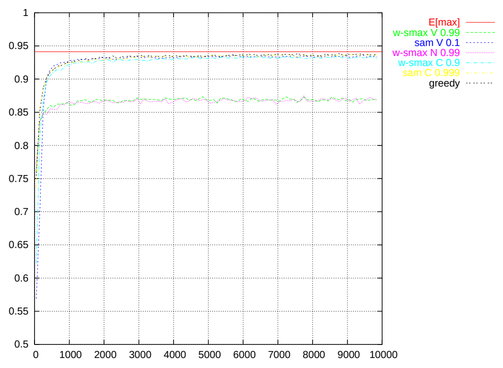
Table 8.2: Performance in 1000 random 16-arm bandit problems with pessimistic initial estimates, in terms of expected cumulative discounted reward for different values of γ over 10000 time steps. The shown are to percentage achievable with respect to the expected best when the statistics of each arm are known a priori. Results are shown for greedy action selection with optimistic initial estimates (**greedy**), sampling-greedy (**sm**) with $\zeta \in \{0.9, 0.999\}$ and weighted-softmax (**ws**) with $\zeta = 0.999$; using either velocity (**v**) or counting (**c**) estimates of uncertainty.

As far as the difference between the performance of the naive variance, velocity and counting estimates of uncertainty is concerned, it appears that the velocity estimates produce the most consistent performance. This is especially true when they are paired with sampling-greedy action selection.

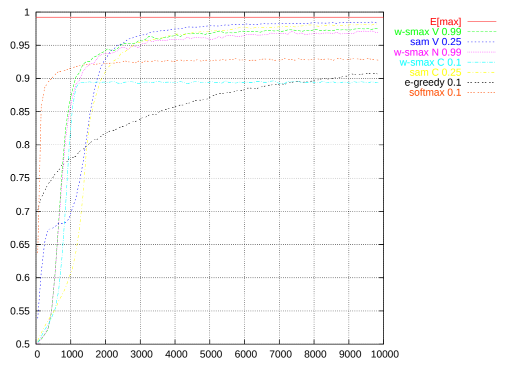
For the **graph task**, we varied the number of states in $\{4, 16, 64\}$ and the number of actions in



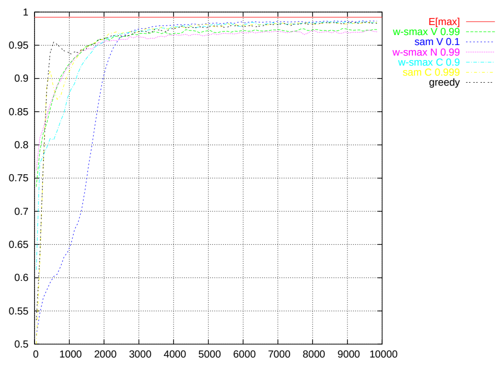
(a) pessimistic initial values, 16 arms



(b) optimistic initial values, 16 arms



(c) pessimistic initial values, 128 arms



(d) optimistic initial values, 128 arms

Figure 8.1: Average rewards of various ad-hoc methods on the 16-arm bandit tasks with initial estimates of the mean rewards of actions set to 0 and to 1 for pessimistic and optimistic initial values respectively. **E[max]** is the maximum achievable reward, **sam** is sampling-greedy and **w-smax** is the weighted softmax action selection (with C and V standing for Counting and Velocity estimates) while **greedy** is greedy action selection.

$\{4, 16, 64\}$. While in general all methods' performance degraded with an enlargement of the space, and all methods performed similarly for small spaces, we found that as the size of the space increased, the sampling method performed significantly better than other methods. The weighted softmax method appeared to be the most sensitive to the size of the space, behaving quite badly in small spaces and much better than anything else in the 64×64 space. We also found that the velocity estimate offered some improvement, especially with regard to sensitivity to ζ, while the naive and counting estimates had an essentially identical performance. The standard softmax method achieved as good a solution as the best methods, but only for a very limited range of values for τ, while the ϵ-greedy method performed the worst.

Figure 8.2 summarises those results. Those indicate that for smaller sizes of the state-action space, the standard softmax is much less sensitive to the selection of the temperature parameter. Larger spaces, however, put it at some disadvantage. For the largest case examined here, the adaptive sampling methods proposed perform significantly better. All such methods tend to perform better when $\zeta \to 1$ and this effect most pronounced for the largest spaces. A tentative hypothesis for an explanation of this behaviour could be given by considering the interaction of two factors: Firstly, the determinism of the environment transitions, and secondly the fact that the number of iterations used is barely sufficient for trying out all possible state-action pairs once in the largest environment. In such an environment the most obvious approach would be to never try a seemingly inferior more than once - if the number of states is large enough, soon a good but sub-optimal solution would be found.

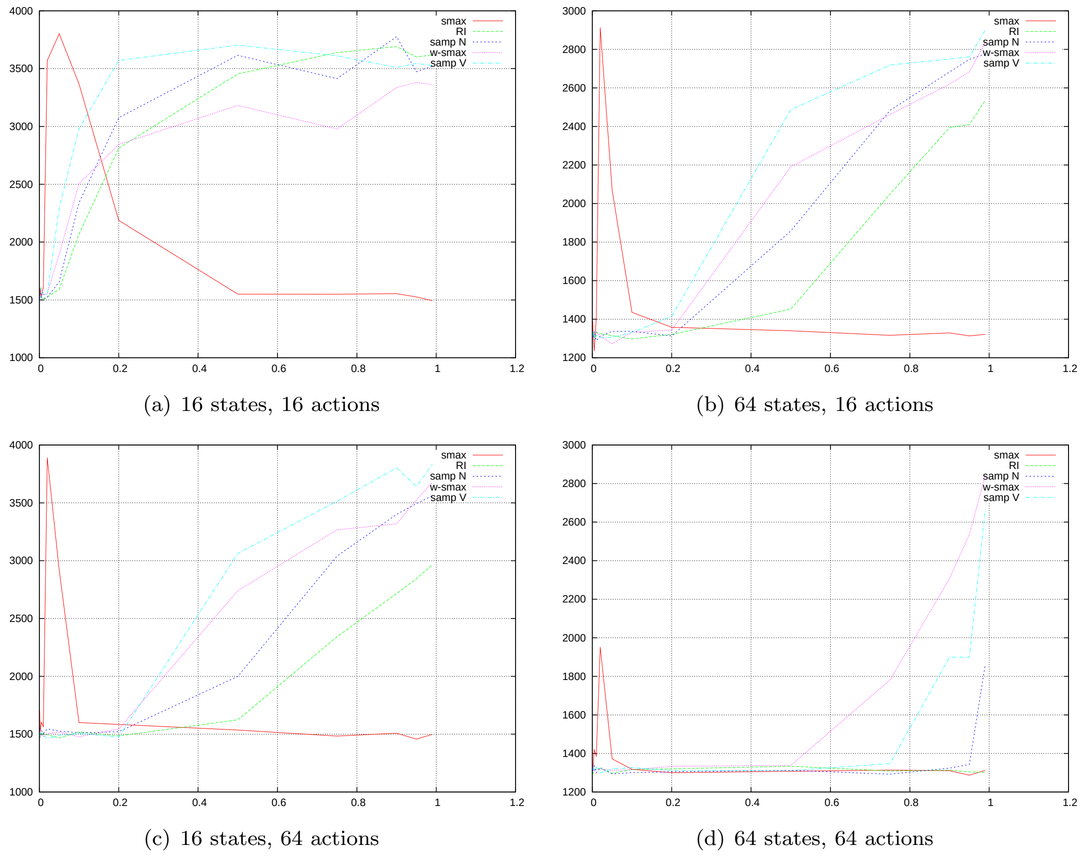


Figure 8.2: Total reward in the graph task after 10,000 iterations averaged over 100 experiments. Results are shown for softmax (**smax**), reliability index (**RI**), sampling-greedy with Naive (**samp N**) and Velocity (**samp V**) uncertainty estimates and weighted-softmax (**w-smax**), also with Velocity estimates. The x-axis is the smoothing parameter ζ, or for softmax, the temperature τ.

We used the **pole balancing task** as a more difficult problem, where it is natural to use optimistic initial estimates of return. In this task a greedy policy outperformed everything else by a large margin. This is to be expected, since the nature of the reward used is sufficient for exploration to take place. In fact, for the choice of reward function, -1 upon failure and 0 upon all other times, we have that

$$\lim_{\gamma \to 1} E[R_t | s_t = s, a_t = a, \pi] = -P(\text{failure} | s_t = s, a_t = a, \pi).$$

Thus greedy action selection mechanism in this case will simply be selecting the action with smallest probability of leading to failure. This turns out to be a good overall strategy for this problem, since the agent learns to maintain the pole at a low-risk configuration. Even a slight amount of radnomness, as can be seen in Figure 8.3(a), can result in a large drop in performance, so any additional exploration is not likely to benefit this task.

To summarise, the adaptive exploration methods discussed herein seem to be useful in some settings. The optimal value for ζ appears to be similar across action selection mechanisms and variance estimates. It is interesting to note that using the sample velocity estimate for updates, which disregards the variance of the return, results in optimal behaviour over a larger set of values for ζ than either the sample variance or counting estimate.

We have also briefly explored the response of the various methods to scaling of the variance estimate as follows: The method, as described so far, used the gradient direction scaled by the learning rate to compute the variance update. However one could remove this dependency on the learning rate. Removing it had a strongly detrimental effect on the weighted-softmax and RI action selection methods, while the sampling-greedy method was unaffected.

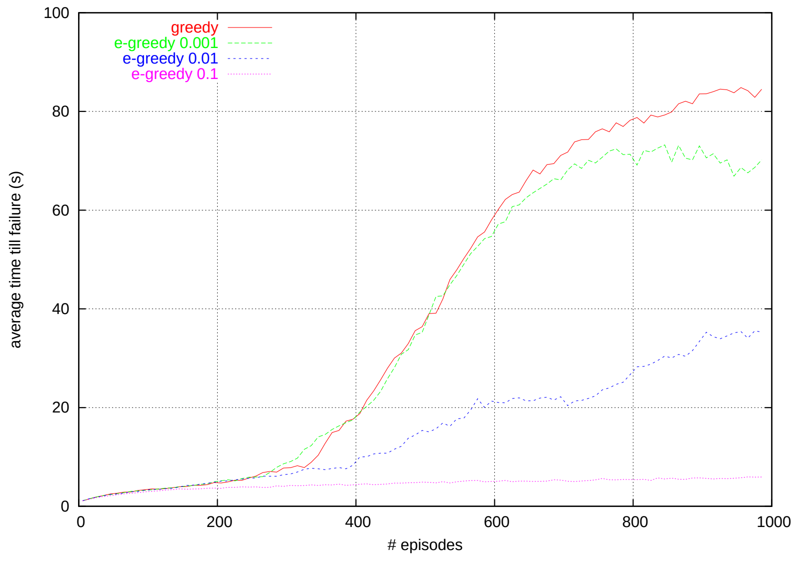
Furthermore, we would like to mention that we observed no experimental differences when using an eligibility-type update for the variance estimates. Of course, this could have very well not been so for other tasks.

8.4.3 Discussion

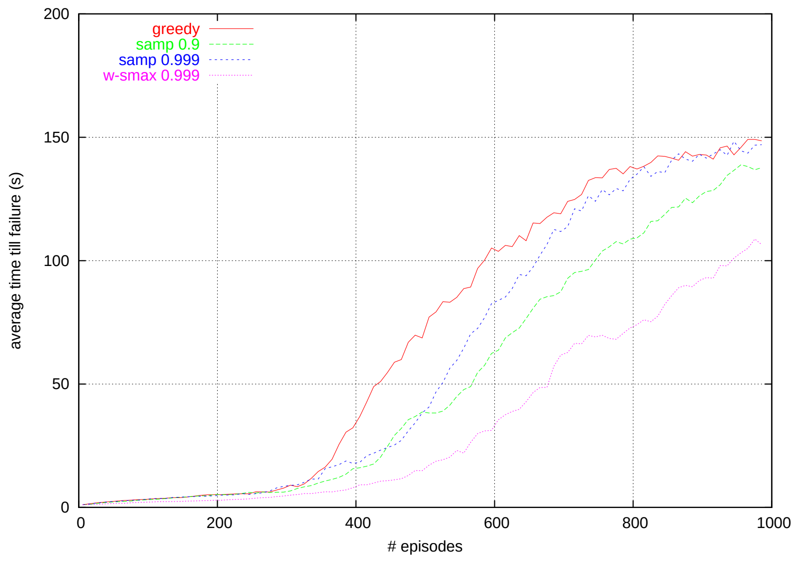
In this section, some of simple techniques for estimating parameter distributions were proposed. These can be applied to the development of action selection mechanisms. In this restricted domain it was found that of the estimates used, the smoothed gradient estimate and the simpler counting estimate can be used to direct exploration relatively well in some cases. Of interest is the fact that the naive variance estimates that are outlined are a generalisation of simple counting schemes and the scheme used in the prioritised sweeping algorithm (Moore and Atkeson, 1993) and that used in the RI method (Sakaguchi and Takano, 2004). The connection between those estimates, the gradient, and its relation to convergence offers some justification to the previously ad hoc use of such techniques. Particularly for prioritised sweeping, the use of such an update might be advantageous for the case of stochastic rewards, since the current naive estimation of accuracy might lead the algorithm to update some states far more often than necessary. Since the aim of prioritised sweeping is to limit the amount of parameter updating to be performed, such simple methods could be useful. There are also some interesting theoretical questions, such as the relationship of this model and its possible application to policy-gradient methods (see for example Baxter and Bartlett, 2000). Such methods also maintain an explicit estimate of the gradient and perform sampling in the policy space (through sampling from the joint distribution of parameters) similarly to the sampling-greedy approach.

In these experiments we have used the sampling-greedy method for action selection, wherein the actions are selected proportionally to the probability of their being the best action. There are two problems with this approach: First, that the sampling of each parameter is done independently, which is only justifiable in some settings, as for example in the randomised bandit problem considered. In more realistic problems sampling from the joint distribution of parameters would be better since there is a dependency between variables. Secondly, even if the sampling is performed correctly, the sampling-greedy algorithm is somewhat ad-hoc, i.e. it does not specifically take into account the discounting when deciding between actions, while Algorithm 2 does. Furthermore, this approach does not even take into account our uncertainty about the underlying MDP. In the next section, we shall take a look at a conceptually simpler, but potentially more powerful method, inspired from particle filtering, maintaining a set of beliefs from which we can sample conveniently.

These methods have other disadvantages. Firstly, it's not easy to estimate what the smoothing parameter and the variance threshold should be from simple prior beliefs. In tasks where the difference between two estimates is very small this leads to many problems. What we would like to do is develop methods for which little or no tuning is necessary, in the sense that the incorporation of prior knowledge can be natural. Indeed, the next section will attempt to develop such methods, where the only tuning required is the prior knowledge about the distribution of rewards/MDPs. Such methods could also be useful in partially observable problems, by maintaining a belief state over the possible MDPs. However there are some technical challenges, which we shall go over in more detail in the next section.



(a) Effects of randomness



(b) Comparison between methods

Figure 8.3: Performance in the pole balancing task, averaged over 100 runs and smoothed over 10 episodes. Effects of randomness in performance in the pole balancing task. Figure 8.3(a) shows how performance changes according to the setting of the mixing parameter ϵ in ϵ-greedy action selection. Setting $\epsilon = 0.01$ can be seen to have a severe effect on performance, while even taking just one random action out of a thousand decreases performance noticeably Results are shown for greedy action selection (**greedy**), sampling-greedy (**samp**) with $\zeta \in \{0.9, 0.999\}$ and weighted-softmax (**w-smax**) with $\zeta = 0.999$; both using counting estimates of uncertainty (Definition 8.2). The discounting parameter was set to $\gamma = 0.99$ and the eligibility race decay was set to $\lambda = 0.7$.

8.5 Ensemble estimates of return distributions

Another interesting scenario is the use of ensemble methods for representing beliefs about the expected value of the reward or the return, or more generally about the full MDP. In contrast to the previous sections of the chapter, where the aim was to agnostically estimate parameter uncertainty, the methods that will be presented in this section will attempt to explicitly represent uncertainty about quantities of interest in the observed MDP.

In this section we investigate two such approaches, one relying on independent estimators with useful properties (for the problem considered) and the other relying on Bayesian estimates. The first approach is similar to bagging in the sense that an ensemble of estimators is used, each one of which observes only a sample of the observations. This effectively leads to a population-based representation of our belief for the expected value of the return. The approach is very similar to standard Bayesian particle-filtering methods (see Casella et al., 1999), but is substantially simpler. The second approach considered is that of an actual particle filter without resampling. For completeness, an analytical Bayesian approach is also applied to the estimation problem.

Those three procedures are compared and contrasted in terms of performance in estimation accuracy, level of expected return and amount CPU usage when used with different action selection algorithms. More specifically, we compare the optimistic-stochastic, sampling-greedy and VPI algorithms under all of the different estimation procedures on randomised n-armed bandit tasks. Results for greedy action selection and the E^3 algorithm are also included.

8.5.1 Ensembles of independent estimators

A population of N estimators is initialised with randomly selected parameters corresponding to our prior belief about the joint distribution of parameters. In the simplest case, all observations are sampled uniformly, weighed equally and the same amount of exponential forgetting is used for all estimators. For steepest stochastic gradient descent estimation methods this is equivalent to changing parameters $w^{(n)}$, for each population member i, according to

$$w_{t+1}^{(n)} = w_t^{(n)} + \eta_t^{(n)} \frac{\partial C}{\partial w^{(n)}}.$$

We may consider two categories of such updates. The deterministic case, where $\eta_{t+1}^{(n)} = \eta_t^{(n)}\ \forall i \in [1, N]$ and the stochastic chase, where $\eta_t^{(n)}$ is a random variable such that $E[\eta_{t+1}^{(n)}] = E[\eta_t^{(n)}]\ \forall t$. We may additionally consider the special cases where $\eta_t^{(n)} = \eta_t^{(j)}$ and $E[\eta_t^{(n)}] = E[\eta_t^{(n')}]$, $\forall n, n' \in [1, N]$ for the deterministic and stochastic case respectively.

In the deterministic case, where all population members use the same step sizes, they should all converge to the same values in the limit in the linear approximation convex cost case, with variance bounded by the variance of error term in the gradient descent equation and the step-size (see Bertsekas, 1999, Proposition 1.5.1.).

In the case of stochastic step-sizes, or deterministic step-sizes that vary across the population, our set of estimates forms a distribution which includes the stochasticity of the environment and not just the uncertainty of our estimates. This is because a smaller step-size corresponds to a large correlation between subsequent values of the reward distribution $E[r_t]$ (see Appendix B.2). Thus a model using large step sizes will result in estimates close to our most recent observations, while a model using smaller step sizes will result in estimates close to the average of our observations. To put that in context, consider the linear estimation of $E[r_t]$. If we start with an estimate w_0, after t iterations we would have

$$w_t = (1-\eta)^t w_0 + \eta \sum_{k=1}^{t} (1-\eta)^{t-k} r_k. \tag{8.16}$$

Now consider $r_t = E[r_t] + e_t$, where e_t is a random variable from a stationary zero-mean distribution $\mathcal{E}$. Furthermore assume that $E[r_t] = \bar{r}\ \forall t$, i.e. that the expected reward is stationary. Then, we may re-write (8.16) as:

$$w_t = (1-\eta)^t w_0 + \eta \left(\bar{r} \sum_{k=1}^{t} (1-\eta)^{t-k} + \sum_{k=1}^{t} (1-\eta)^{t-k} e_k \right). \tag{8.17}$$

By taking expectations we have, since $\mathcal{E}$ is zero-mean,

$$E[w_t] = (1-\eta)^t E[w_0] + \bar{r}\eta \sum_{k=1}^{t} (1-\eta)^{t-k} = (1-\eta)^t E[w_0] + \bar{r}(1-\eta) \tag{8.18}$$

As $\eta \to 0$, $\lim_{t\to\infty} E[w_t] = \bar{r} = E[r_t]$. On the other hand, if we consider (8.17), there is the additional noise term $\eta \sum_{k=1}^{t} (1-\eta)^{t-k} e_k$, whose variance at the limit becomes simply $\frac{\eta}{2-\eta} E[e_k^2]$ when the e_k are independent zero-mean random variables (see Appendix B.5.2 for a detailed proof). This confirms the intuitive notion that the variance among estimators with a larger step-size parameter will be higher than among estimators with a smaller parameter. In the simple linear estimation procedure of the mean of an unknown random variable, this variance will be proportional to the variance of the random variable itself.

In the context of reinforcement learning, it may be useful to create a population of estimators with of a diverse set of step-size parameters. Then it would be possible to use this population to simultaneously estimate the variance of the observations and the accuracy of our mean estimates while also considering different assumptions about the stationarity of the process. In this case however we will be simply utilising the population as our current belief about the distribution of expected rewards for each action.

Sampling from the Ensemble

Assume a population of parameters $\{w^{(n)}\}_{n=1}^{N}$, sampled from the probability measure P_Θ which describes our current belief. Each set of parameters $w^{(n)}$ defines an evaluation function

$$Q_t^{(n)}(s,a) = E[r_t | s_t = s, a_t = a, n].$$

To use the same notation as Chatper 7, we will write $q_i^{(n)}$ to represent the n-th estimate of the expected reward of state-action pair i, or in the case of state-less problem, simply an action i. Assuming the members of $\{q_i^{(n)}\}_{n=1}^N$ have been sampled from our current belief, making them each one as likely as the other.

Algorithm 3 allows a natural and efficient sampling method for this representation. In order to sample from the distribution, all that is necessary is to uniformly choose between the members of the population, thus the complexity in that case is $O(1)$. However, Algorithm 2 requires the calculation of $P(q_i > q_j + \delta)$, which can be approximated by sampling from the joint distribution of q_i, q_j. This can be done by counting the number of times that a member sampled from one distribution will be larger than one sampled from another. This requires doing N comparisons with j for each of N members of i, i.e. calculating

$$P(q_i > q_j + \delta) \approx \sum_{n=1}^N \sum_{m=1}^N I(q_i^{(n)} > q_j^{(m)} + \delta) P(q_i^{(n)}, q^{(m)_j}) \tag{8.19}$$

$$= \frac{1}{N^2} \sum_{n=1}^N \sum_{m=1}^N I(q_i^{(n)} > q_j^{(n)}), \tag{8.20}$$

if we assume the samples $q_i^{(n)}$ are coming from the distribution of q_i, i.e. our belief. The $O(N^2)$ complexity of this, however, is prohibitive. We can reduce that to $O(N)$ by going through all the members $q_i^{(n)}$ of action i and then comparing with a randomly selected member of action j, $q_j^{(m)}$, with m selected uniformly in $[1, N]$. This gives an estimate which is equal to (8.20) in *expectation*.

While this estimation method is particularly simple and easy to use, it suffers from disadvantages which result from the fact that it does not arise from a probabilistic formalism.[5] Firstly, it is not easy to express our prior beliefs about what the process looks like and secondly, the updating of those beliefs is only approximate at best. For this reason we shall take a look at proper Bayesian methodologies for maintaining such beliefs.

8.5.2 Bayesian methods

Assume a set of actions $\mathcal{A}$ and a reward $r \in \mathbb{R}$ with unknown probability distribution $p(r|a)$ such that $E[r|a] = q_a$. Given a prior belief ξ over these distributions we need to determine a posterior belief after having seen some data. More specifically, the prior belief ξ defines (a) The density of the reward given a mean, $p(r|q, \xi)$. (b) The density of the mean reward $p(q|\xi)$. Through the definition of conditional probability we have

$$p(q|r, \xi) = \frac{p(r|q, \xi) p(q|\xi)}{p(r|\xi)}.$$

We will investigate special forms of the reward density.

[5]Each point estimate does actually correspond to a probabilistic inference procedure for tracking a discrete-time gaussian process, as can be seen in Appendix B.2, when η is constant. However the belief over the set of estimates, i.e. the probability that each one of them had been correct, is not maintained.

Closed-form solution for Bernoulli rewards

Let's assume ξ such that under it r is drawn from a Bernoulli distribution such that $P(r = 1|a = i) = q_i$. We can then write

$$p(q|r, \xi) = \frac{p(r|q, \xi)p(q|\xi)}{\int_{-\infty}^{\infty} p(r|q, \xi)p(q|\xi)dq} = \frac{1}{Z}[qr + (1 - q)(1 - r)]p(q|\xi).$$

Since the beta distribution is conjugate to the Bernoulli (see DeGroot, 1970, sec. 9.2), we may use it to represent the prior and posterior distributions. In particular, we set

$$p(q|\xi) = q^{\alpha-1}(1 - q)^{\beta-1}/B(\alpha, \beta),$$

where $B(\cdot)$ is the beta function and

$$p(q|r, \xi) \equiv p(q|\xi') = q^{\alpha'-1}(1 - q)^{\beta'-1}/B(\alpha', \beta'),$$

with $\alpha' = \alpha + r$, $\beta' = \beta + (1 - r)$. Now ξ' is our new prior distribution which we will use to obtain a new posterior distribution after observing one more realisation.

In order to use this Bayesian estimate in the algorithms described in Chapter 7, we require two types of computations. Firstly, sampling from the beta distribution. This was done using the sampling algorithms developed by Cheng (1978), as provided in the C library RANLIB (Brown and Lovato, 1994). Secondly, estimating the probability that $P(x > y)$ for x, y drawn independently from two beta distributions. This can be estimated easily by sampling both variables and counting the number of times one is larger than the other.

Ensemble estimates and particle filters

In the experimental set up described in Section 7.6 the rewards are Bernoulli distributed with a mean given by a uniform prior. This corresponds to a beta prior with the parameters α, β specified as $1, 1$. However in many other cases there may not exist an analytic Bayesian estimation procedure, therefore it may also be of interest to examine a type of ensemble estimate that attempts to approximate the prior and posterior distributions arising in Bayesian inference via a mixture model. There is a large body of literature concerning such models, usually referred to as particle filters in their main application, which is tracking of non-stationary state variables.

In such models we approximate the density $p(q|\cdot)$ via a set of weighted samples $\{(q^{(m)}, w^{(m)})\}_{m=1}^{K}$, called particles:

$$p(q|\cdot) \approx \sum_m p(q|q^{(m)}, \cdot)p(q^{(m)}|\cdot) \tag{8.21}$$

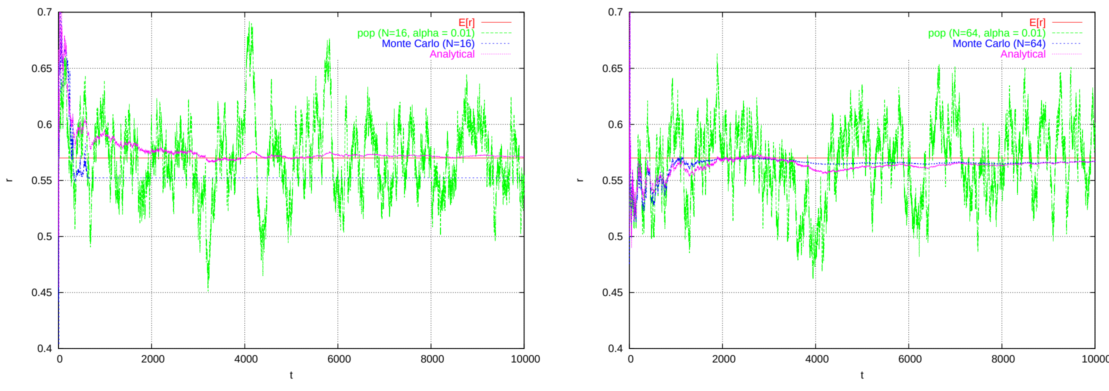
where $p(q^{(m)}|\cdot)$ corresponds to the particle weight. Broadly, there are two interacting factors that need to be considered in order for such methods to be used successfully: ([a]) 1. the form of $p(q|q^{(m)}, \cdot)$ and 2. how new particles will be generated after an observation. The simplest way to choose $p(q|q^{(m)}, \cdot)$ is to use a fixed kernel function. However this imposes a lower limit on the variance of the distribution of q.

The simplest way to update the particles is to keep their position fixed and update their weights. This can be highly inefficient, since after a few iterations most of the weights will approach zero.

In this work we use a randomised discretisation of the distribution to create an approximate grid-based representation (see Arulampalam et al., 2002, section IV.B), where the particles remain stationary. As mentioned by Arulampalam et al. (2002), a proper particle filter would be more appropriate for state estimation rather than for parameter estimation, while had the expected reward been allowed to vary over time, particle filter methods would have been the method of choice.

8.5.3 Evaluation on bandit tasks

Apart from evaluating the methods on the random bandit task described in Section 7.6, it may be also instructive to examine their performance in terms of the estimation accuracy of the mean of the observed rewards. Figure 8.4 displays the estimated mean for a particular value of $E[r]$, for the three methods examined in this section. As the Bayesian approach's prior perfectly matches the experimental conditon, we see it performing excellently. The Monte Carlo approximation follows it relatively closely for 16 particles and its precision increases as the number of particles is increased to 64, as expected. The ensemble method is not very accurate at all, which is to be expected since the gradient descent method with a fixed step-size is more suitable for tracking a non-stationary mean (see Appendix B.2). Indeed, in such tasks the method can perform quite well, as can be seen in Figures 8.5 and 8.6, where it has been used in conjunction with sampling-greedy action selection. The prior of the Bayesian approach is

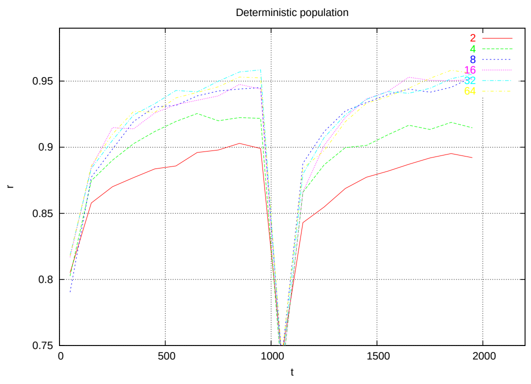


(a) 16 samples (b) 64 samples

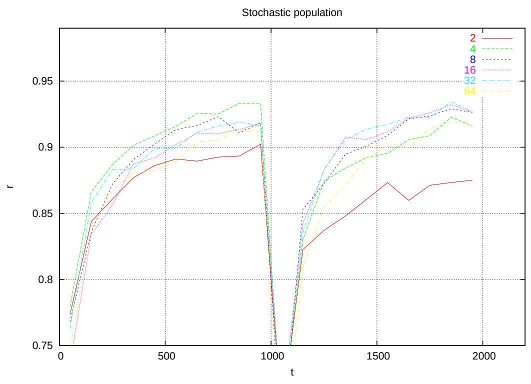
Figure 8.4: An analytical Bayesian estimate (purple line) of the mean (red line) of a Bernoulli random variable is compared with an approximate Monte Carlo approach (blue line), based on a random set of fixed particles and with the population estimate described in Section 8.5.1, with 16 and 64 samples. For this figure, the ensemble members use a learning rate of $\eta = 0.01$.

perfectly satisfied in this experiment and thus should be the method of choice. However in other cases it may not be possible to find a closed-form solution for the posterior distribution. Thus, a Monte Carlo approach may also be of interest. If we are just interested in estimating a fixed unknown parameter, such as the expected reward, then the standard particle filters are not the method choice and it is expected that they perform similarly to the independent estimator approach proposed here. For other forms of the reward distribution, the grid filter or fixed particle approach should be working relatively well, although a critical parameter in that case would be the number of samples required to approximate the belief distribution sufficiently well. In particle filter methods, one would also have to appropriately select the transition distribution. In the independent ensemble method, the learning rate parameter plays a role similar to that of the transition distribution for generating new particles in particle filters (see for example Appendix B.2). In the analytic Bayesian formulation there are no hyper-parameters to select apart from the form of the prior and observation distributions, which have to also be selected explicitly in the Monte Carlo methods, while they are implicitly selected in the independent estimator approach.

Apart from using such models for estimating the mean and uncertainty, it is of particular interest to see if they can be applied to exploration in reinforcement learning. Figure 8.5 illustrates the independent of population type and size in a standard bandit task with 128 actions, expected reward $E[r_t] \in [0, 1]$, and $r_t \in \{0, 1\}$. The expected reward was selected uniformly in $[0, 1]$ at the beginning and at the middle of each trial. Because of the stochasticity of the process, results shown are averaged over 100 trials and smoothed over 100 time steps. The results shown result from using the sampling-greedy algorithm for action selection.



(a) Deterministic population



(b) Stochastic population

Figure 8.5: Average reward on a 128-arm bandit task with Bernoulli distribution of rewards for each action, averaged over 100 experiments and 10 time steps. Results are shown for a deterministic and stochastic populations with population size in {2, 4, 8, 16, 32, 64}.

The deterministic population, whose variance summarises the uncertainty in our estimates, can be used for action selection efficiently. An increase in the number of population members results in a corresponding increase in average reward, which is our stated objective. The stochastic population, on the other hand, is not particularly useful for this task, as its variance includes the stochasticity of the environment.

As a quick illustration of the effectiveness of the approach, can be seen in Figure 8.6, where it is compared with the popular method of optimistic initial values. As might have been expected, the latter approach is very sensitive to the actual initial value chosen, with the value closest to the actual upper

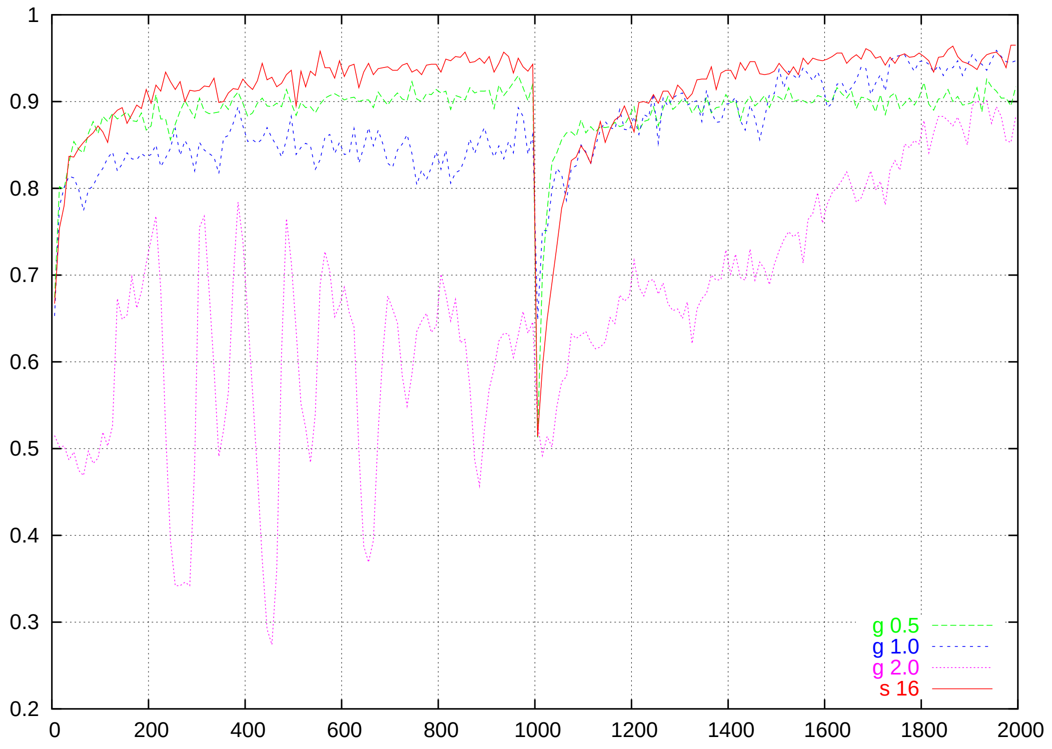


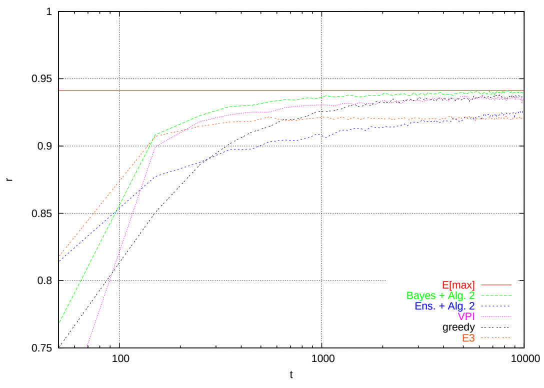
Figure 8.6: Average reward on a 128-arm bandit task with Bernoulli distribution of rewards for each action, averaged over 100 experiments and 100 time steps. Results are shown for a greedy policy (**g**) for an initial value of expected reward 0.5, 1, 2 and for two different ensemble policies with 16 ensemble members. The **s 16** indicates a population with 16 members, stochastic initialisation and deterministic updates.

bound in expected rewards performing the best (initial values lower than 0.5 performed much worse). In this respect it appears to outperform the methods described in the previous section. As far as the optimistic intial values method is concerned, it is interesting to note that there exists a period where its continuously exploring, alternating between sets of actions, until its estimates approach the actual values of the best actions.

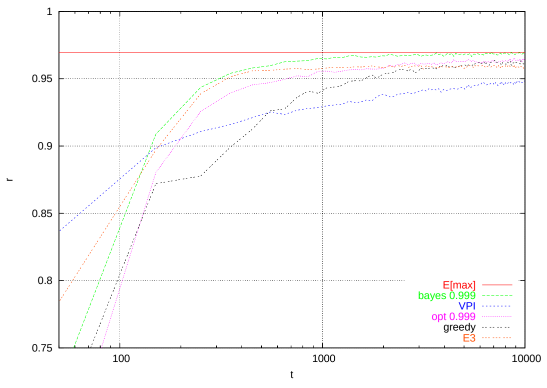
This is particularly true when overly optimistic initial values are used. The issue can be worse in problems with state when methods using an estimate of maximal state-action values are used.[6] As Reynolds (2001a,b) points out, the max operator delays the updates of the predecessor states when action values in successor states are over-optimistic. However this additional delay is measured only with respect to the convergence of action values under uniform sampling. When there is an explicit need to trade off exploration and exploitation, optimistic initial values are a reasonably good candidate.

Another comparison was made between the Bayesian model, the particle filter and the ad-hoc ensemble estimate for 16 actions and 16 samples on a 10000 episode of the randomised n-armed bandit task over 1000

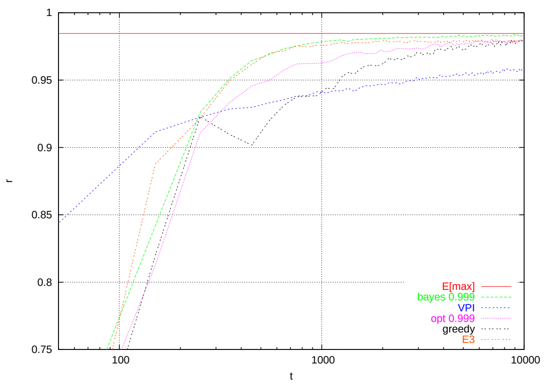
[6] Q-learning being the canonical example of such methods.



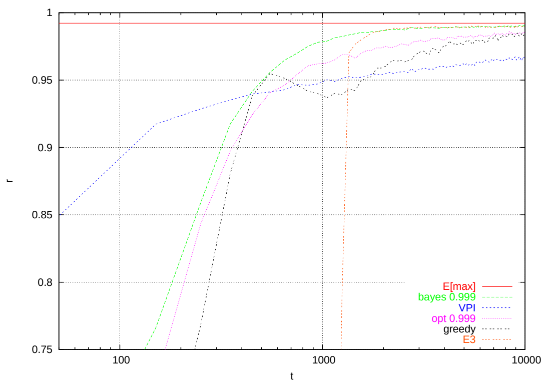
(a) 16 actions



(b) 32 actions



(c) 64 actions



(d) 128 actions

Figure 8.7: Performance of best methods in 1000 random problems, smoothed over 100 time steps. **E[max]** is the best possible asymptotic performance, while **bayes** and **opt** is a bayesian and ensemble estimates using Algorithm 2 with $\gamma = 0.999$. **VPI** is the VPI action selection with ensemble estimates. **E3** is the E^3 algorithm.

runs shows that. While apparently the ensemble estimates are performing slightly better asymptotically, in terms of the performance measure that we are interested in, namely, the expected return for a given discount factor, the particle filter methods are less optimistically biased and thus obtain a maximal performance at a point close to the selected value of γ.

Figure 8.7 shows the performance of some of the best methods on the bandit task. Near the asymptote, it can be seen that the Bayesian estimates coupled with the optimistic-stochastic algorithm perform better than the other algorithms. As expected, the simpler ensemble estimates using the same algorithm are always worse than the Bayesian ones. In this figure it is also easy to see the deficiencies of the simple greedy estimation with optimistic initial values. As the number of actions, becomes larger, the sampling becomes quite inefficient. The appearence of a small drop in performance at around 500 steps in Figure 8.7(c) and 1000 steps in Figure 8.7(d) is related to the fact that sometimes there remains a number of actions which are still optimistically evaluated at that point.

γ	grd opt	grd rnd	bay 0.9	bay 0.99	bay 0.999	ens 0.9	ens 0.99	ens 0.999	VPI 0.9	E3 0.9
0.0	53%	48%	54%	53%	55%	53%	52%	51%	55%	50%
0.5	53%	55%	57%	54%	53%	55%	53%	52%	60%	58%
0.9	58%	68%	72%	62%	60%	66%	57%	56%	73%	72%
0.99	82%	83%	89%	88%	85%	87%	81%	79%	88%	89%
0.999	95%	91%	94%	97%	97%	95%	96%	95%	95%	96%
1.0	98%	94%	95%	99%	99%	97%	99%	98%	97%	97%

Table 8.3: Performance in 1000 random 16-arm bandit problems lasting for 10^4 steps each. The numbers show the expected samples used to estimate quantities in each method. Performance with more samples (up to 128 were tried) does not give an improvement of more than 1%. The **grd (o)** method was initialised with an optimistic initial estimate for all actions, while the **grd (r)** method used a random initial estimate in $[0, 1]$. The **Bayes** methods use Bayesian estimates with prior matched to the experimental conditions and Algorithm 2, while the **Ens.** methods use the ensemble estimates and the either the algorithm or **VPI**. **E3** refers to E^3 action selection. The numbers refer to the value of γ, apart from VPI and E^3 where it was set to $\gamma = 0.999$

γ	grd opt	grd rnd	bay 0.9	bay 0.99	bay 0.999	ens 0.9	ens 0.99	ens 0.999	VPI 0.9	E3 0.9
0.0	50%	51%	50%	51%	51%	53%	52%	48%	52%	48%
0.5	49%	56%	50%	51%	51%	53%	51%	48%	56%	50%
0.9	50%	68%	60%	54%	53%	58%	53%	53%	70%	50%
0.99	59%	83%	84%	76%	68%	81%	68%	66%	87%	50%
0.999	87%	93%	91%	94%	91%	93%	91%	90%	94%	64%
1.0	96%	97%	93%	97%	98%	96%	98%	97%	96%	93%

Table 8.4: Performance in 1000 random 128-arm bandit problems lasting for 10^4 steps each. The numbers show the expected samples used to estimate quantities in each method. The labels are the same as those used in Table 8.3.

Table 8.3 and Table 8.4, compare the performance of methods quantitatively,[7] where it can be seen that Bayesian and the ensemble estimation have a performance peak approximately at the point where the γ parameter in the evaluation measure matches the γ parameter in the algorithm itself. Naturally, because the Algorithm 2 is slightly optimistic, the method explores more than is necessary for a given reward horizon. Interestingly, the VPI method [8] is able to work particularly well at the initial stages, as can be seed in the both the figures and the tables. Overall the VPI estimation appears to behave much better for all the values of $\gamma < 1$ that were examined, even when the optimistic-stochastic algorithm was tuned to the particular horizon that the measurements were taken against.

[7]Since the results have been averaged over 1000 runs, and each run lasts for 10000 steps and the rewards are Bernoulli, the probability that a 1% difference in error is due to chance is smaller than 0.05 for $\gamma \geq 0.9$,while it becomes infinitesimally small for larger values.

[8]This reports results with the VPI method using the ensemble estimates. Results with the Bayesian estimation are worse asymptotically, as the method stops all exploration very early. This effect seems to be related to the averaging, as when using just one sample for calculating the VPI the Bayesian estimation was generally better than the ensemble, though in any case worse than using more samples and and the ensemble estimate.

Method	Average time per trial (ms)		
	4 act. 16 sam.	4 act. 64 sam.	16 act. 16 sam.
Bayesian + Alg. 2	1692	6176	8045
Bayesian + Alg. 3	68	69	256
PF + Alg. 2	285	4055	1187
PF + Alg. 3	10	62	67
Ensemble + Alg. 2	23	58	92
Ensemble + Alg. 3	10	13	28
Ensemble + VPI	47	146	169
E^3	6	6	7
Greedy	4	4	5

Table 8.5: Average CPU usage (as measured on a 900MHz Athlon AMD) of exploration methods on random bandit problems. Results are shown for the 16-action bandit problem with bandits, averaged over 1000 trials for the optimistic-stochastic and sampling-greedy algorithms with analytic Bayesian (**Bayesian**), particle filter Bayesian (**PF**) and independent ensemble (**Ensemble**) estimation methods. Note that the number of samples is also relevant for the analytic estimation approach, since there are still some quantities which have to be estimated via sampling.

The E^3 method, is able to perform almost as well as the Bayesian estimates if it is appropriately tuned, for example in Figures 8.7(c) and 8.7(c). algorithms in all cases apart from when $\gamma \rightarrow 1$. The parameters for the E^3 method were set to $\epsilon = 0.1$, $\delta = 0.01$ and a scaling coefficient for the polynomial of 10^{-10}. These values were selected so that there would be at least some period of greedy behaviour for $\gamma = 0.9$ with 128 actions. However selecting a single set of values that would work in all environments was not possible.

Since it is theoretically possible to solve the problem exactly through a complete enumeration, it is worthwhile taking a look at the comparative CPU consumption of various methods. Table 8.5 shows the computational requirements of the various methods. For Algorithm 2, the Bayesian method is the slowest because it requires estimating $P(X - Y > \delta)$. The consequent sampling from the beta distributions of X and Y in order to perform this estimation seemed to slow the method down considerably. The Monte Carlo Bayesian approximation, while slightly faster than the pure Bayesian approach, requires a lot more samples in order to approach its performance. Thus, for this particualr problem, it appears better to use the Bayesian method, given the choice.

The third consideration must be the use of hyper-parameters in each method. The simplest method of optimistic initial estimates requires setting an appropriate step-size (or step-size schedule) and an upper bound on the expected rewards. While in this case the upper bound is trivial to set, it is not clear how it could have been chosen had it been known that the expected rewards were drawn from a normal distribution in place of the beta distribution. On the other hand, sometimes what is known is just an upper bound, such as in the pole-balancing task. For the ϵ-greedy policies, one would have to also select an appropriate randomness parameter. Similarly, for the methods seen in the previous section, one would have to choose the smoothing parameter and the type of estimate (counting, velocity or naive variance), which is not clear how to do. More importantly, none of these methods can be easily tuned to the choice of γ, being merely exploratory heuristics. For example, one could say that intuitively ϵ should be higher for higher γ and close to zero when $\gamma \to 0$, but beyond that not much can be said. The other methods do take into account the horizon to some extent and the only prior knowledge for the optimal-stochastic and VPI action selection methods is the prior on the environment. This might be difficult to obtain in some cases, but even simple techniques such as the ensemble estimate that is proposed in this thesis seem to work relatively well - though there one would still have to select the appropriate learning rate and initial distribution of estimates. The E^3 algorithm requires just upper and lower bounds on the reward and a number of nuisance parameters for expressing how good one would like the final solution to be. Adjusting the values is not particularly intuitive, and in more complicated problems they might be even harder to tune.

In summary, one could conclude that E^3 has the potential to work well and is extremely fast, compared to other methods. On the other hand it is slightly difficult to select its many hyper-parameters correctly. As far as estimation goes, the Bayesian methodology has the obvious advantages of being easy to specify a prior for and that it is extremely accurate. Unfortunately, even when Bayesian estimation can be analytically formulated, action selection using the estimates is not analytical, not even using the approximate optimistic-stochastic or VPI evaluation methods, which can make them relatively slow, as can be seen in Table 8.5. Lastly, they may not be always applicable. Monte Carlo methods can be applied even where analytical estimation procedures do not exist, but this may incur a further computation penalty and it may be difficult to select the required approximation. However they can perfectly match the performance of the analytical appropach. The independent estimator ensemble, while fast, can not achieve the performance of the Bayesian estimates, at least in this setting. This is to be expected since the problem is stationary, while the estimator ensemble assumes non-stationarity - the more the prior knowledge matches the experimental conditions, the better results one should expect.

8.6 Conclusion

A set of simple population-based models for estimating uncertainty has been presented. Such models are computationally interesting, since they enable the simultaneous representation of uncertainty in our estimates and of stochasticity in our observations, while maintaining beliefs about different prior assumptions on the stationarity of the environment. Another interesting aspect of population-based methods is their potential relevance to neural modelling, as for example proposed in the framework by Pouget et al. (2003). The question of whether and how biological organisms take into account uncertainty remains largely unanswered, with recent evidence showing that, at least in some cases, uncertainty does not influence choice (Daw et al., 2006).

We have presented two such methods, an independent ensemble and a grid particle filter derived from a Bayesian formulation of the estimation problem. The methods have been compared with each other and with analytical Bayesian estimation in simple estimation and in action selection in bandit problems. For optimal action selection, three different algorithms were combined with the methods: sampling-greedy, optimistic-stochastic and VPI. The results presented here indicate that such methods may be useful for efficient exploration in more complicated tasks. Similar methods can be applied to model-based exploration, where our different beliefs about the world can be expressed via a population of world models. Model-based exploration techniques, where an explicit model for the complete environment is maintained instead of only value functions or parametrised policies, should be able to allow for much more efficient exploration. This appears like a worthwhile topic for future research. Finally, it is tempting to ponder whether performance could be enhanced further by devising an action selection mechanism similar to E^3 that utilises full distributions rather than distribution-free bounds.

Chapter 9

Conclusion

This thesis has been examining the use of ensemble methods for maintaining multiple beliefs under uncertainty in sequence learning problems. The tasks we focused on were speech recognition and reinforcement learning problems, while we looked at a number of algorithms for both maintaining beliefs and for making decisions using those beliefs.

In speech recognition tasks we have applied the well-known bagging and boosting algorithms initially to phoneme classification, where we developed mixture models for each phoneme. These mixtures were then used in order to make speech recognition decisions. This work resulted in a paper presented at ICASSP'04 (Dimitrakakis and Bengio, 2004b). The practice of training phonetic mixtures, though successful, relies on potentially unreliable phonetic segmentation. Thus, this work was later extended with a boosting approach specifically addressing the problem of word error minimisation, in a paper presented at ICASSP'05 (Dimitrakakis and Bengio, 2005a). Further results, presented in this thesis, indicate that bagging can be more effective in reducing the effects of noisy phonetic segmentation, even when compared with the more sophisticated boosting technique, at least for this particular dataset. The application of the method to large vocabulary speech recognition remains a future topic of study.

Since we are interested in the interaction of ensemble methods and sequence learning, we have also looked at ensemble training as a sequential learning problem. We compared three algorithms, (a) bagging, (b) mixture of experts (MoE) with gradient-descent training and (c) a variant of mixture of experts using a technique based on methods for reinforcement learning, with each other and with a baseline MLP, in a work presented in the Neurocomputing journal (Dimitrakakis and Bengio, 2005b). In a sense, all three ensemble algorithms make sequential decisions: Ada-Boost performs a step-wise greedy search by creating a new classifier at every step and is the dual of the Hedge algorithm described by Freund and Schapire (1997); MoE allocates credit to each one of the experts deterministically; while in the RL mixture the same thing is done stochastically. They are also all making decisions greedily, since they choose the next point estimate without a view to what possible future estimates might be.

We then consider exactly this problem: how to optimally behave under conditions of uncertainty, when future beliefs are likely to change and the manner in which they change is influenced by the action taken now. Such a situation requires striking a balance between greedy and exploratory behaviour and it arises in many optimisation problems in general. We have studied this problem mainly in the context of bandit tasks, which is the simplest type of tasks where it can be encountered, and have developed a simple set of algorithms for near-optimal exploration, which has been presented in ICANN'06 (Dimitrakakis, 2006). The presented bound upon which these algorithms rely is quite similar to the VPI method, with which some comparisons were made.

All these algorithms require maintaining a probability distribution that expresses our belief. For this reason, we developed a few simple algorithms based on gradient estimates (Dimitrakakis and Bengio, 2005c), which can then be used to drive exploration. We have found that in some cases such methods can be advantageous, but nevertheless give only modest gains. We have furthermore considered three other types of estimates: (a) An analytical Bayesian estimate suitable for stationary distributions, (b) a grid-filter Monte Carlo approximation of the Bayesian estimate, (c) an ad-hoc independent ensemble estimate suitable for non-stationary distributions. All methods were found to be working quite well when used in conjunction with the developed algorithms or VPI, and they have their own advantages and disadvantages in terms of applicability, accuracy, speed and performance.

From here on it could be possible to continue in a number of directions. One could look at other types of mixture models or inference procedures for speech recognition and the potential extension of suggested methods to large scale speech recognition systems. A thorough comparison between different such methods in the field, which are applied at various temporal scales and are utilising various algorithms, would certainly be of some interest to the speech recognition practitioner, who may otherwise be bewildered by this array of offerings. Another possibility is to examine other approximate optimal exploration methods in uncertain environments and to formally expand the developed methods into the MDP case. It is also worthwhile to consider the types of uncertainty which arise when the MDP is only partially observable and when the state-action space is continuous. While there has been some progress towards both directions in a Bayesian framework, the question of optimal exploration has not yet been addressed in these cases. This remains a wide-open research topic, waiting to be explored.

Appendix A

Definitions and Notation

A.1 Notation

A.1.1 Sets, sequences and probabilities

Sets will be denoted by calligraphic characters, i.e. $\mathcal{S}$. The cardinality of a set $\mathcal{S}$ of will be denoted by $|\mathcal{S}|$. Open intervals will be denoted by $(\cdot)$, while closed intervals by $[\cdot]$.

The set of n elements $\{x_1, x_2, \ldots, x_n\}$ will be denoted in shorthand as $\{x_i\}_{i=1}^n$. The summation over elements $x \in \mathcal{X}$ will be written as

$$\sum_{x \in \mathcal{X}} x$$

or equivalently

$$\sum_{i=1}^{N} x_i.$$

The probability of a random variable X taking the value x will be noted as $P(X = x)$. When the context is clear, the probability density of a random variable X drawn from some distribution $\mathfrak{D}$ is noted as $p(x)$, otherwise as $f_X(t)$. However $p(x)$ may be employed for both densities and probabilities in order to make generalising statements. The expectation operator is noted as $E[\cdot]$. Furthermore we shall use $E_{x\in\mathcal{X}}[x]$ to indicate the expectation of the random variable x over the set. The expectation over a probability measure $\mathcal{X}$ will be written as $E_{\mathcal{X}}[x]$, or alternatively if we define a prior $\xi : x \sim \mathcal{X}$ we can use the conditional notation $E[x|\xi]$. If $x \sim \mathcal{X}$ unambiguously then we may simply write $E[x]$. An empirical estimate of the expectation, given some data D and a prior ξ, will be written in short-hand as $E[x|D, \xi] \equiv \hat{E}[x] \equiv \bar{x}$. The variance operator is denoted as $\mathrm{Var}[\cdot]$.

An ordered sequence of n values of a variable taking value x_t at time t will be denoted as $x = (x_1, \ldots, x_n)$. We will also define the range $x_{a:b} = (x_a, \ldots, x_n)$.

The notation $\mathcal{X}^*$ is used to denote the set of all ordered n-tuples $(x_1, \ldots, x_n)$, with $x_i \in \mathcal{X}$, for all $n \in \mathbb{N}$. For example $\mathcal{B}^* \equiv \{0, 1\}^*$ denotes all possible binary sequences.

A.1.2 Scalars, vectors, norms and gradients

Vectors and scalars are denoted with small italics and we shall use x_i for the ith member of vector x. All vectors are column vectors unless transposed. Matrices will be denoted with capital italics. We will use x' for the transpose of a vector x and A' for the transpose of a matrix A.

For vectors $x \in \mathbb{R}^n$, $|x|_p$, denotes the l_p norm:

$$|x|_p = \left(\sum_{i=1}^{n} x_i^p\right)^{1/p}, \tag{A.1}$$

while we will frequently use $\|x\|$ to denote the l_2 norm.

The gradient of some function f with respect to x will be written as

$$\nabla f(x) \equiv \frac{\partial f(x)}{\partial(x)}.$$

The notation for the derivative of a function f with respect to x at some point $x = x^*$ can be written in the following different ways

$$\nabla_x f(x^*) \equiv \nabla f(x^*) \equiv \frac{\partial f(x^*)}{\partial x} \equiv \left.\frac{\partial f}{\partial x}\right|_{x=x^*}.$$

The notation when multiple variables are involved is similar. For example, when $x = (x_1, x_2, \ldots, x_n)$, we can write

$$\nabla_x f(x^*) = (\partial f(x^*)/\partial x_1, \partial f(x^*)/\partial x_1, \ldots, \partial f(x^*)/\partial x_n).$$

A.1.3 Commonly used symbols

Although there is some occassional symbol re-use, the usual meanings of symbols in the thesis are summarised in Table A.1.3.

A.2 Additional Definitions

While in the standard MDP framework the policy can be formulated directly in terms of the MDP's current state, there exists an interesting superclass of problems for which the state is only partially observable. These can be formalised as partially observable Markov decision processes:

Definition A.1 (Partially observable Markov decision process) *A partially observable Markov decision process (POMDP) is defined as the tuplet $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}\mathfrak{T}, \mathfrak{R})$, comprised of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition distribution $\mathfrak{T}(s, a) = P(s_{t+1} = s', o_{t+1} = o' | s_t = s, a_t = a)$ and a reward distribution $\mathfrak{R}(s', s, a) = p(r_{t+1} | s_{t+1} = s', s_t = s, a_t = a)$.*

The difference between an MDP and a POMDP is the fact that instead of observing the state s_t of the MDP directly, we instead observe o_t, which may only give partial information about the state. For this reason the optimal policies in POMDPs are in general non-stationary.

We may now extend the notion of a policy from the simpler n-armed bandit setting to that of MDPs. Now, a policy π defines the probability distribution $P(a_t | s_t, \pi)$. Policies for which $P(a_t | s_t, \pi) = \pi_{a,s}$ are called stationary.

Any mixture of two stationary policies is also stationary (for the related game-theoretic notion of *mixed strategies*, see for example Luce and Raiffa (1957, sec. 4.7-4.10))

Formally, however, the distinction between stationary and non-stationary policies is not absolute. Any non-stationary policy with respect to the MDP of the environment is equivalent to a *stationary* policy by that uses an *augmented state* such that the problem becomes Markov. Similarly, a stationary policy on an MDP will appear to be non-stationary to an observer that only partially observes the MDP state.

	General
θ	model parameters
ξ	prior belief
c	class
C	cost
$\mathcal{D}$	data distribution
D	data set
h	a hypothesis/model
$\mathcal{H}$	the set of all hypotheses
$l(\cdot)$	sample loss
L	loss
$\mathcal{N}$	the normal distribution
s	state
o	observation
w	a vector of mixing weights
x, y	input, and output, or simply scratch random variables
	Speech recognition and Boosting
β	boosting mixture coefficients
ε	empirical expected loss
$\rho(\cdot)$	an arbitrary measure
η	utterance boosting loss function hyper-parameter
ψ	uterrance boosting temporal credit assignment
a	morphological feature (word, syllable, or phoneme)
	Reinforcement learning and decision making
α, η, ζ	step sizes
β	accuracy in softmax action selection
Q	value function
q_i	an estimate/sample of the expected reward of action or state-action pair i
π	policy
R_t	return at time t
r_t	reward at time t
U	utility (usually equal to the expected return)
γ	Decay rate for discounted cumulative rewards
$g(k)$	Function for arbitrarily weighting rewards
λ	Decay rate parameter for eligibility traces
ϵ	randomness parameter for ϵ-greedy
d_t	temporal difference error at time t
b	a lower bound on the expected rewards
e_k	noise process (in the context of gradient descent)
e_t	eligiblity trace (in the context of reinforcement learning)

Table A.1: Common meanings of symbols.

A.2.1 The E^3 algorithm

This section briefly describes the E^3 (Kearns and Singh, 1998) algorithm and how it is used in this thesis.

The E^3 algorithm uses the fact that after visiting a state at least m_{known} times in some MDP $\mathcal{M}$, where

$$m_{known} = O(NTR_{max}/\epsilon)^4 \operatorname*{Var}_{max} \log(1/\delta), \tag{A.2}$$

the estimated reward and transition probabilitites for this state will be within $O((\epsilon/NTR_{max})^2)$ of its true value with probability $(1-\delta)$. Here N is the number of states, T is the reward horizon R_{max} is the maximum possible reward, Var_{max} is the maximum possible reward variance in the MDP, ϵ is the required approximation and δ is the probability of failing to reach the required approximation. While the number of visits is less than m_{known}, the algorithm randomly selects actions and then it behaves greedily.

To apply this algorithm in our case, we first note that it uses the convention that the rewards are conditoinal on states, rather than state-action pairs. The algorithm can be applied to the bandit problem by considering each action as state that can be visited at arbitrary times. Since we are using discounted rewards instead of a finite horizon, the horizon variable becomes, as Kearns also notes, $T = 1/(1-\gamma)$.

Appendix B

Miscellany

B.1 Optimality of multi-stream

This section details how a classifier mixture model, where each mixture member is a generative model, can be used for sequence recognition. We begin by detailing the classifier mixture and then we show how the most likely sequence.

Assume an observed continuous random variable $O \in \mathcal{O}$ and a discrete class membership random variable $Y \in \mathcal{Y}$. Assume that we have a model of conditional densities $\{p(o|y)|y \in \mathcal{Y}, o \in \mathcal{O}\}$. We wish to determine $P(y|o)$ the probability of o belonging to class y. From the definition of the joint density, $p(o, y) = p(o|y)P(y) = P(y|o)p(o)$, we have:

$$P(y|o) = \frac{p(o|y)P(y)}{p(o)} \tag{B.1}$$

Consider a set of classifiers $\mathcal{H}$ that employ this rule to determine class posterior probabilities, such that each classifier h estimates for each class y, $P(y|o, h)$. Furthermore, consider a reliability term w_h, which we assume determines the prior probability that expert h is correct,[1] i.e. for each $h \in \mathcal{H}$,

$$P(h) = w_h / \sum_i w_i$$

and $\sum_{h \in \mathcal{H}} P(h) = 1$.

For static mixtures, i.e. assuming $P(h|o) = P(h)$, we can marginalise to obtain the probability of class y given data o as a weighted sum of the posterior probabilities given by each expert:

$$P(y|o) = \sum_{h \in \mathcal{H}} P(h)P(y|o, h) \frac{\sum_{h \in \mathcal{H}} w_h P(y|o, h)}{\sum_{h \in \mathcal{H}} w_h}. \tag{B.2}$$

[1]This corresponds to the weights in the static mixture model created by boosting. Meyer and Schramm (2006) use the boosting weights in a similar manner.

This can then be extended to:

$$P(y|o) = \frac{\sum_{h\in\mathcal{H}} w_h p(o|y,h)P(y|h)}{p(o)\sum_{h\in\mathcal{H}} w_h}. \tag{B.3}$$

After we make the simplifying assumption that the prior probabilities of classes are the same for each model h, i.e. that $P(y|h) = P(y)\ \forall\ h$ and since $p(o|h) = p(o)$ we have:

$$P(y|o) = \frac{P(y)\sum_{h\in\mathcal{H}} w_h p(o|y,h)}{p(o)\sum_{h\in\mathcal{H}} w_h}. \tag{B.4}$$

B.1.1 Decoding

If each of the models is a path, o being now a sequence of inputs $o_1, o_2, .., o_T$, then $p(o|h)$ (approximately, assuming $p(o_t|o_{t-1}, s_t) = p(o_t|s_t)$, $P(s_t|s_{t-1}, s_{t-2}, \ldots, s_1) = P(s_t|s_{t-1})$ and dropping the c subscript) is:

$$p(x|y,h) \approx \prod_{t=1}^{T} p(o_t|s_t^{h*})P(s_t^{h*}|s_{t-1}^{h*}) \tag{B.5}$$

where $\{s_t^{h*}\}_{t\in[1,T]}$ is the MAP state sequence for model h.

Likewise, the data density, given a path through all models is:

$$p(x|y) = \sum_{h\in\mathcal{H}} w_h \prod_{t=1}^{T} p(o_t|s_t^h)P(s_t^h|s_{t-1}^h). \tag{B.6}$$

In order to perform decoding, we construct a single trellis diagram V such that $V(i,t)$ is the likelihood for the MAP path given that the state at time t is i. In order to simplify this, we *constrain* the state across models so that $s_t^h = i\ \forall h$. Then $V^h(i,t)$ is the corresponding path leading to state i for model h at time t. We denote

$$m = \arg\max_k V(k, t-1) = \arg\max \sum_{h\in\mathcal{H}} \beta_h V^h(k,t) \tag{B.7}$$

as the from which follows:

$$V(i,t) = \sum_{h\in\mathcal{H}} \beta_h p(o_t|s_t^h = i)P(s_t^h = i|s_{t-1}^h = m)V^h(m,t), \tag{B.8}$$

Thus, in the constrained case it is possible to infer the MAP state sequence using the Viterbi algorithm by simply considering the weighted sum of likelihoods and transition probabilities locally at each state.

B.2 Exponential-family Priors in Time

Let us express the dependency of two random variables in time v_t and v_{t+k} via some prior ξ. In some cases the expectation of v_{t+k} given the prior and some observations can be written in linear form. One such case is the following.

Suppose a random variable V, generated by Markov process of the form

$$p(v_t|v_1,\ldots,v_{t-1}) = p(v_t|v_{t-1}) \tag{B.9}$$

We introduce another random variable $X \in \mathbb{R}$ with realisations x_t, for which we assume that $p(x_t|v_t, v_{t-1}) = p(x_t|v_t)$, resulting in:

$$p(v_t|x_t, v_{t-1}) = \frac{1}{p(x_t|v_{t-1})} p(x_t|v_t)p(v_t|v_{t-1}). \tag{B.10}$$

We now use consider a that the transition distribuion is Gaussian

$$p(v_t|v_{t-1}) \propto e^{-a\|v_t - v_{t-1}\|^2}, \tag{B.11}$$

with $a > 0$ and a Gaussian distribution with unit variance $p(x_t|v_t) \propto e^{-b\|x_t - v_t\|^2}$ to obtain:

$$p(v_t|x_t, v_{t-1}) \propto \frac{1}{p(x_t|v_{t-1})} e^{-b\|v_t - x_t\|^2} e^{-a\|v_t - v_{t-1}\|^2}. \tag{B.12}$$

We wish to find v_t with maximum a posteriori probability. This requires finding v_t that minimises

$$f(v_t) = b\|v_t - x_t\|^2 + a\|v_t - v_{t-1}\|^2, \tag{B.13}$$

where we note that the term $p(x_t|v_{t-1})$ does not influence the minimisation procedure and can be ignored. A necessary condition for a minimum is that the first derivative with respect to the v_t is zero. From this it follows that

$$\begin{aligned} \nabla f(v_t) &\propto b(v_t - x_t) + a(v_t - v_{t-1}) = 0 \\ v_t^* &= \frac{bx_t + av_{t-1}}{a+b} = (1-\lambda)x_t + \lambda v_{t-1} \end{aligned} \tag{B.14}$$

where $\lambda = \frac{a}{a+b}$. Thus, a linear parameter update corresponds to the above simple discrete-time Gaussian process inference procedure. The step-size λ expresses our belief as to how much of the observed randomness is due to the stochasticity of the state rather than the observation given the state. When $b \ll a$, i.e. when the process is assumed to be almost stationary or when the accuracy of the observations is supposed to be very low, not as much importance is placed upon new observations for updating our estimate.

B.3 Model-based reinforcement learning

Completely determining the return distribution for some policy π requires the calculation of

$$\begin{aligned} p(r_{t+k}|s_t = i, \pi) &= \sum_j p(r_{t+k}, s_{t+k} = j | s_t = i, \pi) \\ &= \sum_j p(r_{t+k}|s_{t+j} = j) p(s_{t+k} = j | s_t = i, \pi) \end{aligned}$$

for all $i \in \mathcal{S}$ and all $n > 0$. We can expand the above expression using the recursive relation

$$p(s_{t+1}|s_t, \pi) = \sum_i p(s_{t+1}|a_t = i, s_t) p(a_t = i | s_t, \pi).$$

B.3.1 World models

When there is a model for the $p(s_{t+k} = i | s_t = j, \pi)$ we can use samples of the distribution $p(r_{t+k}|s_{t+k} = i)$ to estimate $p(r_{t+k}|s_t = j, \pi)$ for all k rather than rely on the Bellman relations. In some cases we may model the transitions between each state pair explicitly.

In some a simpler model can be used. One commonly used such model is offered by eligibility traces, which can be seen as an agnostic way to perform importance sampling in an unknown environment. The following section outlines the links between eligibility traces, importance sampling and a prior belief in the determinism of the underlying environment-agent Markov chain that generates the observed samples.

B.3.2 Eligibility traces

It is possible to derive eligibility traces in the form of a model for state transition probabilities where the parameter λ arises from a prior on the amount of randomness exhibited by the Markov chain formed when selecting actions on some MDP $\mathcal{M}$ according to some policy π. This prior model is useful as a type of smoothing when performing parameter updates using sampling techniques.

We denote the vector of state probabilities as $x = (x_1, x_2, \ldots, x_{|\mathcal{S}|})$ and $p(x|\xi)$ will be used for our prior belief over possible state transition probabilities. Together with some observations D, this can be used to calculate

$$p(x|D, \xi) = \frac{p(x|\xi) p(D|x, \xi)}{p(D|\xi)}.$$

In this case we will consider D being a single observation (s_t, s_{t+k}). For ξ we shall employ a Dirichlet prior:

Definition B.1 (Dirichlet distribution) *The Dirichlet distribution can be used as a conjugate prior for multinomial sampling over* n *discrete outcomes*

$$p(x|\xi) = \frac{\Gamma(A)}{\prod_{i=1}^n \Gamma(a_i)} \prod_{i=1}^n x_i^{a_i - 1}, \tag{B.15}$$

where $A = \sum_{i=1}^{n} a_i$ *and* $a_i > 0, i \in \{1, \ldots, n\}$.

We would like to employ this distribution as a prior to describe the expected randomness. Let $a_i = 1 - \lambda$ for all $i \in \{1, \ldots, n\}$, with $\lambda \in [0, 1]$. For $\lambda = 1$, we have a uniform prior over transition probabilities. When $\lambda \to 0$, the transition probabilities tend to become deterministic.

B.4 Hypothesis tests

There are two statistical tests used throughout this thesis in order to assess the significance of results in supervised learning tasks. The first one, the z-test, is used for classification tasks, while a bootstrap estimate of confidence intervals is employed for speech recognition tasks. In either case, one should always keep in mind that such tests only give us some information about the probability ϵ that the difference in scores will be at least δ, should the methods be tested on different data coming from the same distribution.

B.4.1 Two-proportion z-test

In a two-proportion z-test of two independent Bernoulli variables, where we observe x_i positive results for n_i tests, we start by measuring the empirical mean of each one $\hat{p}_i = \frac{x_i}{n_i}$. We want to know something about the distribution of $\hat{p}_1 - \hat{p}_2$. We know that the variances are simply $\mathrm{Var}[\hat{p}_i] = p_i(1 - p_i)/n_i$ and that

$$\mathrm{Var}[\hat{p}_1 - \hat{p}_2] = \mathrm{Var}[\hat{p}_1] + \mathrm{Var}[\hat{p}_2]$$

If we assume the same variances for both variables then we can write

$$\mathrm{Var}[\hat{p}_1 - \hat{p}_2] = p(1 - p)\left(\frac{1}{n_1} + \frac{1}{n_2}\right).$$

For large n_1, n_2, $\hat{p}_1 - \hat{p}_2$ is approximately normal. We replace p with $\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$ and obtain

$$z = \frac{(\hat{p}_1 - \hat{p}_2)}{\sqrt{\hat{p}(1 - \hat{p})(1/n_1 + 1/n_2)}}$$

will be Gaussian distributed with mean 0, variance 1. Now we integrate over the tails of the corresponding Gaussian to obtain

$$P(\|\hat{p}_1 - \hat{p}_2\| > x | p_1 = p_2) = 1 - \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$$

There are two problems with this test. Firstly, to the Gaussianity assumption this test is not very reliable for small n_1, n_2. The second is the assumption of independence, as pointed out in (Dietterich, 1998), does not make such a test very suitable for comparing classifiers, since a lot of times there is considerable overlap in the sets of misclassified examples. Experimental results provided by Dietterich indicate that while its type I error is acceptable it is not very powerful in the sense that its probability of rejecting the null hypothesis (that the two classifiers are identical) is small. Thus it is probably not wise to draw very firm conclusions from this test.

B.4.2 Bootstrap estimate for speech recognition

Bootstrap methods(Efron and Tibshirani, 1993, see) are useful methods for simulating the calculation of estimates $\{f_i\}$ from multiple samples $\{D_i\}$ drawn from some distribution $\mathcal{D}$. Since we normally have but a finite-size sample D, the samples D_i are drawn with replacement from D, the empirical distribution, rather than from $\mathcal{D}$, the distribution of interest.

The method herein was originally advocated for evaluating speech recognition performance by Bisani and Ney (2004). It amounts to using the results of speech recognition on a test set of sentences as an empirical distribution of errors. More specifically, for comparing two systems A and B we draw identical bootstrap samples D_k from the test set. For each sentence $i \in D_k$, containing n_i words, we obtain the number of errors made by each system, which we denote as ε_i^A and ε_i^B, respectively. Then we calculate the difference in word error rate on this sample:

$$\Delta W_k = \frac{\sum_{i \in D_k} (\varepsilon_i^A - \varepsilon_i^B)}{\sum_{i \in D_k} n_i}.$$

We thus obtain a sample $S = \{\Delta W_k\}_{k=1}^K$ of K bootstrap estimates of the difference in word error rate. It is then possible to use this as an empirical distribution to estimate quantities of interest. In our case, we are interested in

$$P(\Delta W > 0) = \int_0^\infty p(\Delta W) d\Delta W \approx \frac{1}{B} \sum_{k=1}^K u(\Delta W_k),$$

where $u(x) = 0$ if $x \leq 0$ and 1 otherwise. This quantity approximates the probability that system A is better than system B.

B.5 Proofs

B.5.1 Distance Bound

The following bound is useful for deciding on the termination of gradient methods(Bertsekas, 1999, Section 1.2 and Exercise 1.2.10).

Lemma B.1 (Distance bound) *Let θ^* be a local minimum of C and $\theta \in S$, with $S = \{\theta : \|\theta - \theta^*\| < \delta\}$, $\delta > 0$. If there exists $m > 0$ such that*

$$m\|z\|^2 \leq z' \nabla^2 C(\theta) z, \quad \forall\ z \in \mathbb{R}^n, \tag{B.16}$$

then, for all $\theta \in S$,

$$\|\theta - \theta^*\| \leq \|\nabla C(\theta)\|/m, \quad C(\theta) - C(\theta^*) \leq \nabla \|C(\theta)\|^2/m.$$

For any twice continuously differentiable function f, it holds that:

$$\nabla f(y) = \nabla f(x) + \int_0^1 \nabla^2 f\big(x + t(y - x)\big)(y - x) dt.$$

We apply this to C and note that $\nabla C(\theta^*) = 0$ to obtain:

$$\begin{aligned}\nabla C(\theta) &= \int_0^1 \nabla^2 C\big(\theta^* + t(\theta - \theta^*)\big)(\theta - \theta^*)dt \\ (\theta - \theta^*)'\nabla C(\theta) &= \int_0^1 (\theta - \theta^*)'\nabla^2 C\big(\theta^* + t(\theta - \theta^*)\big)(\theta - \theta^*)dt.\end{aligned}$$

From (B.16), we have:

$$\begin{aligned}(\theta - \theta^*)'\nabla C(\theta) &\geq m\|\theta - \theta^*\|^2 \\ \|\theta - \theta^*\|\|\nabla C(\theta)\| &\geq m\|\theta - \theta^*\|^2 \\ \|\theta - \theta^*\| &\leq \|\nabla C(\theta)\|/m,\end{aligned}$$

which concludes the first part of the proof.

The second statement can be proven by using the following second order expansion that holds for every function f that is twice continuously differentiable over an open sphere f centred at x, and with $y : x + y \in S$:

$$f(x+y) = f(x) + y'\nabla f(x) + \frac{1}{2}y'\nabla^2 f(x)y + o(\|y\|^2) \tag{B.17}$$

from which it follows that:

$$f(y) - f(x) = f(x + (y-x)) - f(x) = (y-x)'\nabla f(x) + \frac{1}{2}(y-x)'\nabla^2 f(x)(y-x) + o(\|y-x\|^2) \tag{B.18}$$

We also need the fact that

$$\min_{y\in\mathbb{R}^n}\left\{(y-x)'\nabla f(x) + m\|y-x\|^2/2\right\} = -\frac{1}{2m}\|\nabla f(x)\|^2. \tag{B.19}$$

(This can be proven by the fact that at the minimum, the derivative of the argument of the minimum operator will have a derivative of 0, resulting in $y^* = \frac{-\nabla f(x)}{m} + x$. A substitution completes the proof.)

From (B.16) and (B.18), we have:

$$\begin{aligned}f(x + (y-x)) - f(x) &= (y-x)'\nabla f(x) + \frac{1}{2}(y-x)'\nabla^2 f(x)(y-x) + o(\|y-x\|^2) \\ &\geq (y-x)'\nabla f(x) + \frac{m}{2}\|y-x\|^2\end{aligned}$$

We can then replace the right hand side with its minimum, as given by (B.19), which gives:

$$f(x + (y-x)) - f(x) \geq -\frac{1}{m}\|\nabla f(x)\|^2.$$

We now replace

$$C(\theta) - C(\theta^*) \leq \frac{1}{2m}\|\nabla C(\theta)\|^2,$$

which concludes the second part of the proof.

We further note that if $\|C(\theta)\| \leq \epsilon$ then the following inequalities also hold:

$$\|\theta - \theta^*\| \leq \epsilon/m, \quad C(\theta) - C(\theta^*) \leq \epsilon^2/m.$$

B.5.2 Noise residual

We consider some update process defined such as the one in (8.17), where the noise term is a random variable $e_k \sim \mathcal{E}$, for which the following conditions hold:

$$E[e_k] = 0 \tag{B.20}$$

$$p(e_1, \ldots, e_t) = \prod_k^t p(e_k) \tag{B.21}$$

$$\tag{B.22}$$

we have, for $\eta \in [0, 1]$

$$E\left[\big(\eta \sum_{k=1}^t (1-\eta)^{t-k} e_k\big)^2\right] = \eta^2 E\left[\big(\sum_{k=1}^t (1-\eta)^{t-k} e_k\big)^2\right] \tag{B.23}$$

We note that due to (B.21) we have:

$$\lim_{t\to\infty} E\left[\big(\sum_{k=1}^t \gamma^{t-k} e_k\big)^2\right] = \lim_{t\to\infty} \sum_{k=1}^t \gamma^{2(t-k)} E[e_k^2] = \frac{E[e_k^2]}{1-\gamma^2} \tag{B.24}$$

so finally plugging this into (B.23) we obtain

$$\lim_{t\to\infty} \eta^2 E\left[\big(\sum_{k=1}^t (1-\eta)^{t-k} e_k\big)^2\right] = \eta^2 \frac{E[e_k^2]}{1-(1-\eta)^2} = \frac{\eta}{2-\eta} E[e_k^2]. \tag{B.25}$$

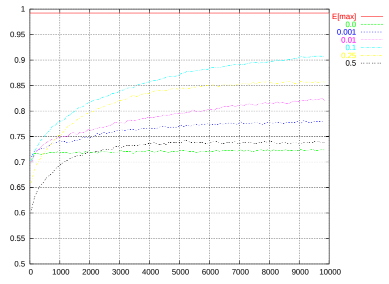
The error residual is approaches the noise variance when η is close to 1 and becomes 0 when $\eta = 0$.

Appendix C

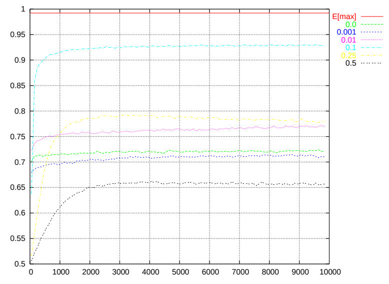
Supplementary results

C.1 Random bandit problems

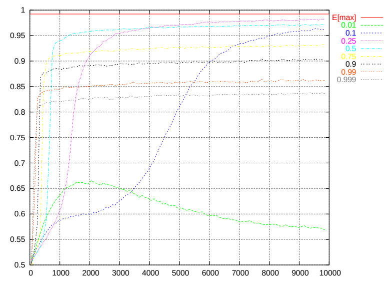
This appendix looks the behaviour of the heuristic methods in the bandit tasks in some more detail. The first obvious thing is that in general the use of pessimistic initial values (Figure C.1)results in worse performance than optimistic initial values (Figure C.1). The sampling-greedy and weighted-softmax seem to be less affected by the choice of initial values, though the selection of the ζ parameter poses an additional difficulty.



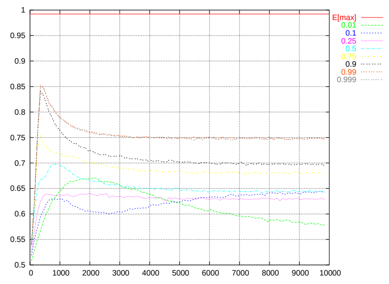
(a) ϵ-greedy



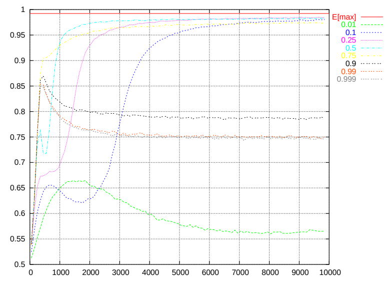
(b) softmax



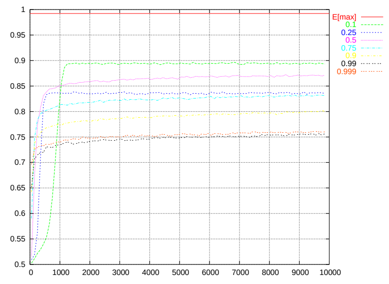
(c) sampling-greedy counting



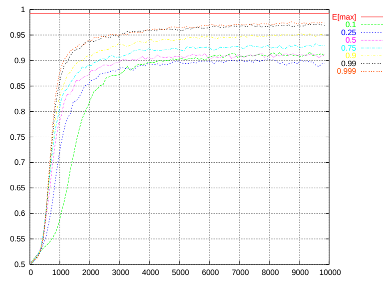
(d) sampling-greedy variance



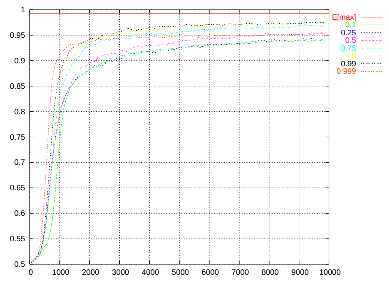
(e) sampling-greedy velocity



(f) weighted softmax counting

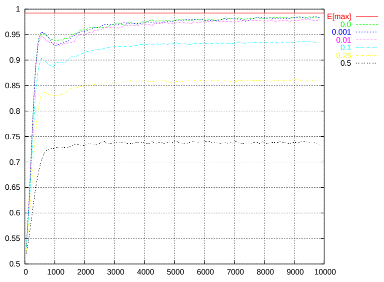


(g) weighted softmax variance

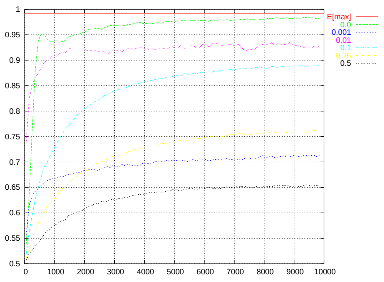


(h) weighted softmax velocity

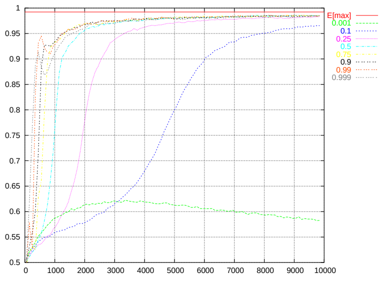
Figure C.1: Average rewards in 128-arm bandit tasks with pessimistic initial values.



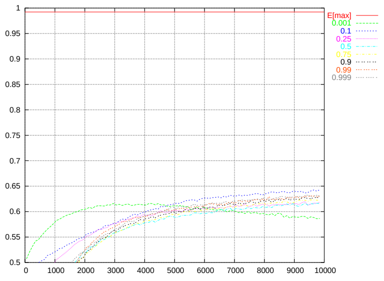
(a) ϵ-greedy



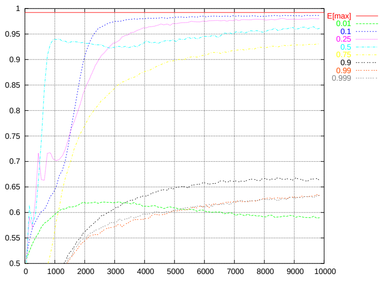
(b) softmax



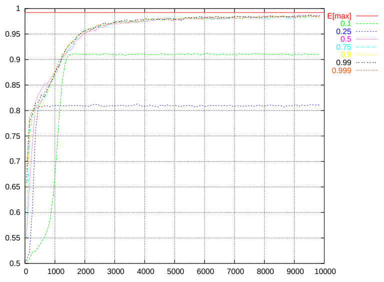
(c) sampling-greedy counting



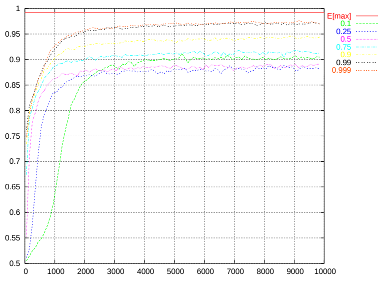
(d) sampling-greedy variance



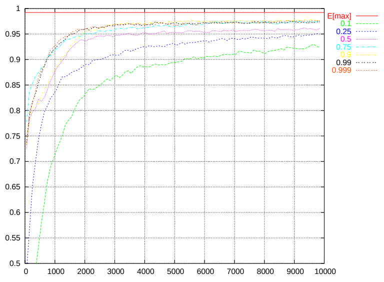
(e) sampling-greedy velocity



(f) weighted softmax counting



(g) weighted softmax variance



(h) weighted softmax velocity

Figure C.2: Average rewards in 128-arm bandit tasks with optimistic initial values.

Abbreviations

AR Auto-regressive
ANN Artificial neural network
DCT Discrete cosine transform
DFT Discrete Fourier transform
DP Dynamic programming
EM Expectation maximisation
GMM Gaussian mixture model
HMM Hidden Markov model
IDFT Inverse discrete Fourier transform
MAP Maximum a posteriori
MC Monte Carlo
MCMC Markov Chain Monte Carlo
MDP Markov decision process
MFCC Mel-frequency Cepstrum coefficients
ML Maximum likelihood
MLP Multi-layer perceptron
MOE Mixture of experts
MSE Mean square error
POMDP Partially observable Markov decision process
RI Reliability index
RL Reinforcement learning
SARSA State-action-reward-state-action
SVM Support vector machine
TD Temporal difference
WER Word error rate

Index

Glossary

basis model: one of a set of models comprising an ensemble model, 3

exploitation: try to maximise the expected return according to our current knowledge of the environment, 54

exploration: using resources in order to improve our knowledge of the environment, 54

morphology:the form and structure(of an utterance), 20

return: a functional, usually a weighted sum, of future rewards, 54

reward: a single scalar value r_t observed at time t. It is sometimes referred to as *payoff* in the literature., 11

stationary policy: a probability distribution over actions that does not change with time, 12

utility: the quantity to be maximised in a given decision making problem, 11

value function: a function mapping states or state-action pairs to expected returns given a policy, 55

Bibliography

Chuck. W. Anderson and Zhaohui Hong. Reinforcement learning with modular neural networks for control. In *Proceedings of NNACIP'94, the IEEE International Workshop on Neural Networks Applied to Control and Image Processing*, pages 90–93, 1994. URL `citeseer.nj.nec.com/anderson94reinforcement.html`.

M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussianbayesian tracking. *Signal Processing, IEEE Transactions on*, 50:174–188, February 2002. ISSN 1053-587X. doi: 10.1109/78.978374. URL `http://ieeexplore.ieee.org/iel5/78/21093/00978374.pdf`. A condensed version is available at http://citeseer.ist.psu.edu/maskell01tutorial.html.

Marios Athineos, Hynek Hermansky, and Daniel P.W. Ellis. LP-TRAP: Linear predictive temporal patterns. In *International Conference on Spoken Language Processing (ICSLP)*, 2004. IDIAP RR 04-59.

P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Machine Learning Research*, 3(Nov):397–422, 2002. A preliminary version has appeared in *Proc. of the 41th Annual Symposium on Foundations of Computer Science*.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256, 2002. A preliminary version has appeared in *Proc. of the 15th International Conference on Machine Learning*.

Peter Auer. Models for trading exploration and exploitation using upper confidence bounds. In *PASCAL workshop on principled methods of trading exploration and exploitation*. PASCAL Network, 2005.

L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. A new algorithm for the estimation of hidden Markov model parameters. In *IEEE Inernational Conference on Acoustics, Speech and Signal Processig, ICASSP*, pages 493–496, 1988.

Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105, 1999.

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *Proc. 17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000. URL `citeseer.nj.nec.com/baxter00reinforcement.html`.

Richard Ernest Bellman. A problem in the sequential design of experiments. *Sankhya*, 16:221–229, 1957a.

Richard Ernest Bellman. *Dynamic Programming*. Princeton University Press, 1957b. Republished by Dover in 2004.

José M. Bernardo. Expected information as expected utility. In *The Annals of Statistics*, volume 7, pages 686–690. Institute of Mathematical Statistics, 1979.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2001.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

M. Bisani and H. Ney. Bootstrap estimates for confidence intervals in asr performance evaluation. In *Proceedings IEEE International Conference on coustics, Speech, and Signal Processing, (ICASSP'04*, volume 1 of *Proceedings of the IEEE*, pages 409–412, 2004.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. URL `citeseer.nj.nec.com/breiman96bagging.html`.

John Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David Smith, and Rich Washington. Planning under continuous time and uncertainty: A challenge for ai. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 77–84. Morgan Kaufmann, San Francisco, CA, 2002.

Barry W. Brown and James Lovato. RANLIB.C: Library of C routines for random number generation. Part of the NETLIB package, 1994. `http://www.netlib.no/netlib/random/`.

George Casella, Stephen Fienberg, and Ingram Olkin, editors. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer-Verlag, 1999.

R. C. H. Cheng. Generating beta variates with nonintegral shape parameters. *Communications of the ACM*, 21:317–322, 1978.

R. A. Cole, K. Roginski, and M. Fanty. The OGI numbers database. Technical report, Oregon Graduate Institute, 1995.

Ronan Collobert and Samy Bengio. Links between perceptrons, MLPs and SVMs. In *ICML*, page 23, 2004. URL `http://doi.acm.org/10.1145/1015330.1015415`.

G. Cook and A. Robinson. Boosting the performance of connectionist large vocabulary speech recognition. In *Proc. ICSLP '96*, volume 3, pages 1305–1308, Philadelphia, PA, 1996. URL `citeseer.nj.nec.com/cook96boosting.html`.

R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

Nathaniel D. Daw, John P. O'Doherty, Peter Dayan, Ben Symour, and Raymond J. Dolan. Cortical substrates for exploratory decisions in humans. *Nature*, 441:876–879, June 2006.

Richard Dearden, Nir Friedman, and Stuart J. Russell. Bayesian Q-learning. In *AAAI/IAAI*, pages 761–768, 1998. URL `citeseer.ist.psu.edu/dearden98bayesian.html`.

Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In Kathryn B. Laskey and Henri Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 150–159, San Francisco, CA, July 30–August 1 1999. Morgan Kaufmann, San Francisco, CA.

Morris H. DeGroot. *Optimal Statistical Decisions.* John Wiley & Sons, 1970. Republished in 2004.

Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):1, 2000.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. URL `citeseer.ist.psu.edu/dietterich95solving.html`.

Christos Dimitrakakis. Nearly optimal exploration-exploitation decision thresholds. In *Int. Conf. on Artificial Neural Networks (ICANN)*, 2006. IDIAP-RR 06-12.

Christos Dimitrakakis and Samy Bengio. Online policy adaptation for ensemble classifiers. In *12th European Symposium on Artificial Neural Networks, ESANN 04*, 2004a.

Christos Dimitrakakis and Samy Bengio. Boosting HMMs with an application to speech recognition. In *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP*, 2004b. IDIAP-RR 03-41.

Christos Dimitrakakis and Samy Bengio. Boosting word error rates. In *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP*, 2005a.

Christos Dimitrakakis and Samy Bengio. Online policy adaptation for ensemble classifiers. *Neurocomputing*, 64:211–221, 2005b.

Christos Dimitrakakis and Samy Bengio. Gradient-based estimates of return distributions. In *PASCAL workshop on principled methods of trading exploration and exploitation*. PASCAL Network, 2005c.

Mathew Magimai Doss. *Using Auxiliary Sources of Knowledge for Automatic Speech Recognition*. PhD thesis, École Polytechnique Fédérale de Lausanne, Computer Science Department, Lausanne, Switzerland, 2005. thesis #3263 (IDIAP-RR 05-90).

Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*, volume 57 of *Monographs on Statistics & Applied Probability*. Chapmann & Hall, November 1993. ISBN 0412042312.

Eyeal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed and reinforcement learning problems. *Journal of Machine Learning Research*, 2006. to appear.

J.G. Fiscus. A post-processing system to yield reduced error word rates: Recognizer output voting error reduction (ROVER). In *IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 347–354, 1997.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

C. J. Gittins. *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, New Jersey, US, 1989.

Zakkula Govindarajulu. *Sequential Statistics*. World Scientific Publishing, 2004.

Asela Gunawardana and William Byrne. Convergence theorems for generalized alternating minimization procedures. *Journal of Machine Learning Research*, 6:2049–2073, 2005.

Hynek Hermansky and Sangita Sharma. TRAPs - classifiers of temporal patterns. In *Proceedings of International Conference on Speech and Language Processing (ICSLP'98)*, 1998.

Ronald A. Howard. Information value theory. *IEEE Transactions on Systems, Science and Cybernetucs*, 2(1):22–26, 1966.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Johnny Mariéthoz and Samy Bengio. A new speech recognition baseline system for Numbers 95 version 1.3 based on Torch. IDIAP-RR 04-16, IDIAP, 2004.

Michael I. Jordan, editor. *Learning in Graphical Models*. Adaptive Computation and Machine Learning series. MIT Press, 1999. URL `http://mitpress.mit.edu/promotions/books/JORLPS99`.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, ept of Computer Science, Stanford, 1990.

Grigoris I. Karakoulas. Probabilistic exploration in planning while learning. In *Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 352–361. Morgan Kaufmann, San Francisco, CA, 1995.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA, 1998. URL `citeseer.ist.psu.edu/kearns98nearoptimal.html`.

Hamed Ketabdar, Hervé Bourlard, and Samy Bengio. Hierarchical multi-stream posterior based speech recognition system. IDIAP-RR 25, IDIAP, 2005a.

Hamed Ketabdar, Jithendra Vepa, Samy Bengio, and Hervé Bourlard. Developing and enhancing posterior based speech recognition systems. In *Proceedings of Interspeech*, Lisbon, Portugal, 2005b. IDIAP-RR 05-23.

Guillaume Lathoud, Mathew Magimai.-Doss, Bertrand Mesot, and Hervé Bourlard. Unsupervised Spectral Subtraction for Noise-Robust ASR. In *Proceedings of the 2005 IEEE ASRU Workshop*, San Juan, Puerto Rico, December 2005. IDIAP RR 05-42.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, 163(4):845–848, 1965.

R. Duncan Luce and Howard Raiffa. *Games and Decisions*. John Wiley and Sons, 1957. Republished by Dover in 1989.

David J. C. MacKay. *Information Theory, Probability and Neural Networks*. Draft Version, 1997. URL `http://wol.ra.phy.cam.ac.uk/mackay/itprnn/`.

Omid Madani, Danie J. Lizotte, and Russel Greiner. The budgeted multi-armed bandit problem. In *Learning Theory: 17th Annual Conference on earning Theory, COLT 2004*, volume 3120 of *Lecture Notes in Computer Science*, pages 643–645. Springer-Verlag, 2004a.

Omid Madani, Danie J. Lizotte, and Russel Greiner. Active model selection. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 357–365, Banff, Canada, 2004b. AUAI Press, Arlington, Virginia.

Shiee Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5:623–648, 2004.

Llew Mason, Peter L. Bartlett, and Jonathan Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38(3):243, 2000.

Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, New York, 1997.

Ron Meir and Gunnar Rätch. An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning*, LNCS, pages 119–184. Springer-Verlag, 2003.

Nicolas Meuleau and Paul Bourgine. Exploration of multi-state environments: Local measures and Back-Propagation of uncertainty. *Machine Learning*, 35:117, 1999.

Carsten Meyer and Hauke Schramm. Boosting hmm acoustic models in large vocabulary speech recognition. *Speech Communication*, 48:532–548, 2006.

Hemant Misra and Hervé Bourlard. Spectral Entropy Feature in Full-Combination Multi-Stream for Robust ASR. In *Proceedings of ISCA European Conference on Speech Communication and Technology (Eurospeech)*, Lisbon, Portugal, September 2005. IDIAP-RR 2005 10.

Hemant Misra, Hervé Bourlard, and Vivek Tyagi. New entropy based combination rules in HMM/ANN multi-stream ASR. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Hong Kong, April 2003. IDIAP-RR 2002 31.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103, 1993.

A. Morris, A. Hagen, H. Glotin, and H. Bourlard. Multi-stream adaptive evidence combination for noise robust ASR. *Speech Communication*, pages 25–40, 2001.

Andrew C. Morris, Viktoria Maier, and Phil Green. From WER and RIL to MER and WIL. In *Proceedings of International Conference on Spoken Language Processing (ICSLP 2004)*, 2004.

Bambang Parmanto, Paul W. Munro, and Howard R. Doyle. Improving committee diagnosis with resampling techniques. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *NIPS*, pages 882–888. MIT Press, 1995. ISBN 0-262-20107-0. URL `http://nips.djvuzone.org/djvu/nips08/0882.djvu`.

Ioannis Partalas, Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Ensemble pruning using reinforcement learning. In Grigoris Antoniou, George Potamias, Costas Spyropoulos, and Dimitris

Plexousakis, editors, *Proceedings of the 4th Hellenic Conference on Artificial Intelligence, SETN 06*, Lecture Notes in Artificial Intelligence 3955, pages 301–310, Heraklion, Crete, Greece, May 2006. Springer-Verlag.

Alexander Pouget, Peter Dayan, and Richard S. Zemel. Inference and computation with population codes. *Annual Review Neuroscience*, 3:381–410, 2003.

Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 759–766. Morgan Kaufmann, San Francisco, CA, 2000.

Lawrence R. Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition.* PTR Prentice-Hall, Inc., 1993.

G. Raetsch, T. Onoda, and K.-R. Mueller. Soft margins for adaboost. *Machine Learning*, 42(3):287, 2001.

Stuart I. Reynolds. The curse of optimism. In *Proceedings of the Fifth European Workshop on Reinforcement Learning*, Utrecht, The Netherlands, October 2001a.

Stuart I. Reynolds. Optimistic initial q-values and the max operator. In *First UK Workshop on Computational Intelligence (UKCI'01)*, Edinburgh, Scotland, September 2001b.

Stuart Ian Reynolds. *Reinforcement Learning with Exploration.* PhD thesis, School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK, December 2002.

Saharon Rosset, Ji Zhu, and Trevor Hastie. Margin maximizing loss functions. In *NIPS*, 2003. URL `http://books.nips.cc/papers/files/nips16/NIPS2003_LT22.pdf`.

G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUEF/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

Yutaka Sakaguchi and Mitsuo Takano. Reliability of internal prediction/estimation and its application. I. adaptive action selection reflecting reliability of value function. *Neural Networks*, 17(7):935–952, 2004.

Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297, 1999.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of statistics*, 26(5):1651–1686, 1998.

Hauke Schramm and Xavier L. Aubert. Efficient integration of multiple pronunciations in a large vocabulary decoder. In *Proceedings of the International Conference on Acoustic, Speech and Signal Porocessing (ICASSP-00)*, volume 3, pages 143–146, 2006.

Holger Schwenk. Using boosting to improve a hybrid HMM/neural network speech recogniser. In *Proc. ICASSP '99*, pages 1009–1012, 1999. URL `citeseer.nj.nec.com/schwenk99using.html`.

Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural Computation*, 12(8):1869–1887, 2000.

A. Smola, P. Bartlett, B. Sch olkopf, and D. Schuurmans. Advances in large margin classifiers, 2000.

Ron Sun and C. L. Giles, editors. *Sequence Learning: Paradigms, Algorithms, and Applications*. Springer-Verlag: LNAI 1828, 2001. URL `http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-41597-1`.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9, 1988.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1038–1044. MIT Press, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999. URL `citeseer.ist.psu.edu/sutton99between.html`.

Marc Toussaint. A neural model for multi-expert architectures. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002)*, 2002.

Robert Tibshirani Trevor Hastie and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer-Verlag, 2001.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, pages 260–269, 1967.

Abraham Wald. *Sequential Analysis.* John Wiley & Sons, 1947. Republished by Dover in 2004.

Christopher J.C.H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279, 1992.

Richard Weber. On the Gittins index for multiairmed bandits. *The Annals of Applied Probability*, 2(4): 1024–1033, 1992.

J. Wyatt. Exploration control in reinforcement learning using optimistic model selection. In A. Danyluk and C. Brodley, editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

Richard S. Zemel and Toniann Pitassi. A gradient-based boosting algorithm for regression problems. In *NIPS*, pages 696–702, 2000. URL `citeseer.nj.nec.com/zemel01gradientbased.html`.

Rong Zhang and Alexander I. Rudnicky. Comparative study of boosting and non-boosting training for constructing ensembles of acoustic models. In *Proceedings of Eurospeech-2003*, pages 1885–1888, 2003.

Rong Zhang and Alexander I. Rudnicky. A frame level boosting training scheme for acoustic modeling. In *Proceedings of ICSLP 2004*, pages 417–420, 2004a.

Rong Zhang and Alexander I. Rudnicky. Apply n-best list re-ranking to acoustic model combinations of boosting training. In *Proceedings of ICSLP 2004*, pages 1949–1952, 2004b.

Yi Zhang, Samuel Burer, and W. Nick Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7:1315–1338, Jul 2006.

Mark Zlochin, Mauro Birattari, Nicolas Meuleau, and Marco Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.

Geoffrey Zweig and Mukund Padmanabhan. Boosting gaussian mixtures in an LVCSR system. In *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP*, August 2000. URL `http://citeseer.ist.psu.edu/428040.html; http://www.research.ibm.com/voicemail/postscript/icassp2000a.ps`.

Curriculum Vitae

Christos Dimitrakakis

Permanent address:	4 Agias Sofias, Drama 66100, Greece	Phone:	+41764971039
Current address:	4 Rue de Simplon, Martigny 1920, Switzerland	email:	dimitrak@idiap.ch
Date of Birth:	26 August 1975	Nationality :	Greek

Education

2003 –	Docteur dès Sciences (anticipated 2006). The Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland. Thesis title: *Ensembles for Sequence Learning*.
1997–1998	Master of Science in Telecommunications and Information Systems. Department of Electronic Systems Engineering, University of Essex, Colchester, UK. Thesis title: *Genetic Programming for Network Topology Design*.
1993–1997	Bachelor of Engineering in Electronic Systems Engineering, University of Manchester, Manchester, UK.

Professional Experience

2001–	IDIAP Research Institute, Martigny, Switzerland. Machine Learning Group, Research Assistant
2000 – 2001	ATMEL, Patras, Greece Software/firmware developer & technical manager
1999 – 2000	Military service, Greece
Aug 1998 – 1999	Freelance programmer, UK

Other Experience

2002–	Participation in TORCS, The Open Racing Car Simulator project. Software developer (simulation, control, sound)

Languages

English (fluent), French (passable), Greek (native).

Computer Experience

Operating Systems:	Linux, UNIX, Windows, AmigaOS
Languages:	C, C++, Java, Assembly (various), Octave

Publications

Book Chapters and Theses

C. Dimitrakakis, (1998). *Genetic programming for network topology design*, MSc Thesis, Electronic Systems Engineering, University of Essex, UK.

Journal Publications

Christos Dimitrakakis and Samy Bengio, (2005). Online policy adaptation for ensemble classifiers *Neurocomputing*, 64:211–221, 2005.

Conference and Workshop Publications

Christos Dimitrakakis (2006). Nearly optimal exploration-exploitation decision thresholds. In *International Conference on Artificial Neural Networks* (ICANN 2006), Athens, Greece.

Christos Dimitrakakis and Samy Bengio, (2005). Gradient-based estimates of return distributions. In *Pascal Workshop on Principled Methods of Trading Exploration and Exploitation*, London, UK.

Christos Dimitrakakis and Samy Bengio, (2005). Boosting word error rates. In *Proceedings of the 2005 IEEE International Conference on Acoustic Speech Signal Processing* (ICASSP 2005), Philadelphia, USA.

Christos Dimitrakakis and Samy Bengio, (2004). Online policy adaptation for ensemble classifiers *12th European Symposium on Artificial Neural Networks* (ESANN 04), Bruges, Belgium.

Christos Dimitrakakis and Samy Bengio, (2004). Boosting HMMs with an application to speech recognition. In *Proceedings of the 2004 IEEE International Conference on Acoustic Speech Signal Processing* (ICASSP 2004), Quebec, Canada.

Technical Reports

Christos Dimitrakakis (2006). Online statistical estimation for vehicle control : A tutorial IDIAP-RR 06-13, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2006). A note on exploration-exploitation decision thresholds. IDIAP-RR 06-12, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2005). Gradient estimates of return. IDIAP-RR 05-28, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2004). Estimates of parameter distributions for optimal action selection. IDIAP-RR 04-72, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2004). Boosting word error rates. IDIAP-RR 04-49, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2003). Online Policy Adaptation for Ensemble Classifiers. IDIAP-RR 03-69, IDIAP, Martigny, Switzerland.

Christos Dimitrakakis and Samy Bengio (2003). Boosting HMMs with an application to speech recognition. IDIAP-RR 03-41, IDIAP, Martigny, Switzerland.

Not peer-reviewed

Christos Dimitrakakis (1999). Reinforcement learning with continuous action values. `http://www.idiap.ch/~dimitrak/papers/RLContAction.ps.gz`