

# Stop Wasting my FLOPS: Improving the Efficiency of Deep Learning Models

Présentée le 14 juin 2022

Faculté des sciences et techniques de l'ingénieur  
Laboratoire de l'IDIAP  
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

## Angelos KATHAROPOULOS

Acceptée sur proposition du jury

Prof. A. M. Alahi, président du jury  
Prof. F. Fleuret, Prof. P. Frossard, directeurs de thèse  
Dr A. Lucchi, rapporteur  
Dr G. Synnaeve, rapporteur  
Dr M. Salzmann, rapporteur



# Acknowledgements

Before continuing to the rest of this thesis, I would like to express my appreciation to several people that have directly or indirectly made this work possible and the past 5 years enjoyable.

Firstly, I would like to thank François for offering me this opportunity 5 years ago. More importantly, I want to thank him for the many interesting discussions and for showing to me what a sustainable approach to research looks like. Finally, I want to express my admiration for his unlimited enthusiasm that has been really inspirational. I will certainly miss going to his office for a morning meeting and instead discussing the latest mini-project he was playing with over the weekend.

Secondly, I would like to thank all the people at Idiap for making my PhD a smooth and enjoyable experience. To Alexandros, Nam, Bastian, Sophie, François, Apoorv, Sargam, Christian, Olivia, Angel, Suraj, Teja, Tatjana, Cijo, James, Julian, Phil, Nikos, Thiago and many other Idiapers past and present.

I would also like to thank my brother and my parents because they have had the longest influence on my life without which I would not be the person that I am today.

Moreover, I want to thank my friends and in particular “Ζωάρρα Crew”. It is very inspiring to see our little group from Greece spread all over the world in various prestigious places.

Finally, but definitely most importantly, I want to thank Despoina for making my life so much brighter and for sharing with me the highs and supporting me during the lows of this 5 year journey.

*January 13, 2022*

Angelos



# Abstract

Deep neural networks have completely revolutionized the field of machine learning by achieving state-of-the-art results on various tasks ranging from computer vision to protein folding. However, their application is hindered by their large computational and memory requirements. In this thesis, we propose methods for improving the efficiency of deep neural networks.

Firstly, we tackle the sample inefficiency of neural network training with an importance sampling algorithm suitable for deep neural networks. This algorithm allows us to focus computation on datapoints that are going to provide useful gradients for training our models and ignore the ones that will have negligible gradients. We show that our algorithm can improve the performance of various neural networks when compared to uniform sampling under a fixed computational budget.

Secondly, we design a model that is suitable for processing large input images with a fraction of the computational and memory requirements of traditional approaches. We achieve this by sampling from a data-dependent attention distribution in order to only process a portion of the input in high resolution. We demonstrate that our model can learn both the attention and the features in an end-to-end fashion using only single image-wise labels for supervision.

Subsequently, we shift our attention to transformer architectures and introduce a kernelized formulation for self-attention that reduces its quadratic complexity to linear with respect to the input sequence's length. Furthermore, we uncover the relationship between autoregressive transformers and recurrent neural networks and show that our formulation enables up to 3 orders of magnitude faster autoregressive inference.

Finally, we develop clustered attention, a method that can approximate softmax transformers with reduced computation. This is achieved by grouping elements of the input using clustering. We showcase that our formulation provides a better trade-off between performance and computation in comparison to the original transformer architecture. In addition, we demonstrate that clustered attention can approximate pretrained transformer models without any fine-tuning and with minimal loss in performance.



# Résumé

Les réseaux de neurones profonds (deep neural networks) ont révolutionné le domaine de l'apprentissage automatique (machine learning) en obtenant des résultats à la pointe de l'innovation dans des domaines comme la vision par ordinateur (computer vision) ou le repliement des protéines. Mais leur utilisation est fortement pénalisée par leurs énormes besoins en puissance de calcul et en mémoire. Dans cette thèse, notre objectif est de proposer différentes méthodes permettant d'optimiser l'efficacité de ces modèles.

Dans un premier temps, nous tentons de résoudre le problème de l'inefficacité de la phase d'échantillonnage de l'apprentissage de réseau de neurones à l'aide d'un algorithme d'échantillonnage dynamique adapté. Cet algorithme nous permet de concentrer la puissance de calcul sur les exemples d'apprentissage qui apporteront des gradients utiles à l'entraînement de nos modèles, tout en ignorant ceux qui ont des gradients négligeables. Nous démontrons que notre algorithme peut améliorer les performances de divers réseaux neuronaux par rapport à un processus d'échantillonnage uniforme, étant donné un budget computationnel fixé.

Ensuite, nous élaborons un modèle qui convient au traitement d'images de grande tailles avec seulement une fraction des besoins en matière de calcul et de mémoire demandés par les approches traditionnelles. Nous y parvenons en réalisant un échantillonnage à partir d'une fonction d'attention qui dépend des données, afin de ne traiter qu'une partie des entrées en haute résolution. Nous montrons que notre modèle peut apprendre de manière jointe aussi bien la fonction d'attention que les caractéristiques en utilisant uniquement une vérité terrain de classification.

Ensuite, nous nous intéressons aux modèles à attention du type *transformers* et nous introduisons une formulation kernelisée pour l'auto-attention (self-attention) qui réduit sa complexité quadratique en complexité linéaire par rapport à la longueur de la séquence d'entrée. Nous montrons la relation entre les *transformers* autorégressifs et les réseaux neuronaux récurrents, et nous démontrons que notre formulation permet une inférence autorégressive plus rapide de 3 ordres de grandeur.

Enfin, nous développons une méthode d'attention groupée proche des *transformers* softmax, qui elle aussi demande une puissance de calcul réduite. Cette méthode consiste à regrouper

## Résumé

---

les éléments d'entrée à l'aide d'un *clustering*. Nous montrons que notre formulation offre un meilleur compromis entre performance et puissance de calcul par rapport à l'architecture originale. Par ailleurs, nous démontrons que cette attention groupée peut égaler les modèles de *transformers* pré-entraînés sans réglage spécifique, avec une perte minimale de performance.



# Zusammenfassung

Tiefe neuronale Netze haben den Bereich des maschinellen Lernens vollständig revolutioniert, indem sie bei verschiedenen Aufgaben — von Computer Vision bis zur Proteinfaltung — Spitzenresultate erzielt haben. Ihre Anwendung wird jedoch durch ihren hohen Rechen- und Speicherbedarf behindert. In dieser Arbeit schlagen wir Methoden zur Verbesserung der Effizienz von tiefen neuronalen Netzen vor.

Erstens gehen wir die Ineffizienz des Trainings neuronaler Netze mit einem für tiefe neuronale Netze geeigneten Importance Sampling Algorithmus (Algorithmus für Stichprobenentnahme nach Wichtigkeit) an. Dieser Algorithmus ermöglicht es uns, die Berechnung auf Datenpunkte zu konzentrieren, die nützliche Gradienten für das Training unserer Modelle bieten, und diejenigen zu ignorieren, die vernachlässigbare Gradienten aufweisen. Wir zeigen, dass unser Algorithmus die Leistung verschiedener neuronaler Netze im Vergleich zu gleichmäßigem Sampling bei einem festen Berechnungsbudget verbessern kann.

Zweitens entwerfen wir ein Modell, das für die Verarbeitung grosser Eingangsbilder geeignet ist und nur einen Bruchteil der Rechen- und Speicheranforderungen herkömmlicher Ansätze benötigt. Wir erreichen dies, indem wir von einer datenabhängigen Aufmerksamkeitsverteilung (Attention Distribution) abtasten, um nur einen Teil des Inputs in hoher Auflösung zu verarbeiten. Wir zeigen, dass unser Modell sowohl die Aufmerksamkeit als auch die Merkmale durchgehend erlernen kann, indem es nur einzelne Labels per Bild zur Überwachung verwendet.

Darauffolgend verlagern wir unsere Aufmerksamkeit auf Transformer-Architekturen und führen eine kernelisierte Formulierung für die Selbstaufmerksamkeit (Self-Attention) ein, die ihre quadratische Komplexität auf eine lineare in Bezug auf die Länge der Eingabesequenz reduziert. Zudem decken wir die Beziehung zwischen autoregressiven Transformers und rekurrenten neuronalen Netzen (RNN) auf und zeigen, dass unsere Formulierung eine um bis zu drei Grössenordnungen schnellere autoregressive Inferenz ermöglicht.

Schliesslich entwickeln wir eine Methode der geclusterten Aufmerksamkeit, die Softmax-Transformers mit reduziertem Rechenaufwand approximieren kann. Dies wird durch die Gruppierung von Eingabeelementen mittels Clustering erreicht. Wir zeigen, dass unsere Formulierung im Vergleich zur ursprünglichen Transformer-Architektur einen besseren Trade-Off

## Zusammenfassung

---

zwischen Leistung und Rechenaufwand bietet. Darüber hinaus zeigen wir, dass die geclusterte Aufmerksamkeit die vortrainierte Transformer-Modelle ohne Verfeinerung und mit minimalem Leistungsverlust nähern kann.

## Περίληψη

Τα νευρωνικά δίκτυα έχουν φέρει πλήρη επανάσταση στον τομέα της μηχανικής μάθησης, επιτυγχάνοντας κορυφαία αποτελέσματα σε διάφορες εργασίες που κυμαίνονται από την υπολογιστική όραση έως την αναδίπλωση πρωτεϊνών. Ωστόσο, η εφαρμογή τους παρεμποδίζεται από τις μεγάλες απαιτήσεις τους σε υπολογιστική ισχύ και μνήμη. Στην παρούσα διατριβή προτείνουμε μεθόδους για τη βελτίωση της αποδοτικότητας των νευρωνικών δικτύων.

Πρώτον, αντιμετωπίζουμε τη δειγματοληπτική αναποτελεσματικότητα της εκπαίδευσης των νευρωνικών δικτύων με έναν αλγόριθμο δειγματοληψίας σημαντικότητας κατάλληλο για νευρωνικά δίκτυα. Αυτός ο αλγόριθμος μας επιτρέπει να εστιάσουμε τους υπολογισμούς σε σημεία δεδομένων που πρόκειται να παρέχουν χρήσιμη πληροφορία για την εκπαίδευση των μοντέλων μας και να αγνοήσουμε αυτά που δεν βοηθούν στην εκπαίδευση. Δείχνουμε ότι ο αλγόριθμός μας μπορεί να βελτιώσει την απόδοση διάφορων νευρωνικών δικτύων σε σύγκριση με την ομοιόμορφη δειγματοληψία υπό σταθερή υπολογιστική ισχύ.

Δεύτερον, σχεδιάζουμε ένα μοντέλο που είναι κατάλληλο για την επεξεργασία μεγάλων εισόδων ενώ έχει πολύ μικρότερες υπολογιστικές απαιτήσεις και απαιτήσεις μνήμης σε σχέση με τις παραδοσιακές προσεγγίσεις. Το επιτυγχάνουμε αυτό με τη δειγματοληψία από μια κατανομή προσοχής (**attention**) που εξαρτάται από τα δεδομένα, ώστε να επεξεργαζόμαστε μόνο ένα τμήμα της εισόδου σε υψηλή ανάλυση. Αποδεικνύουμε ότι το μοντέλο μας μπορεί να μάθει τόσο την κατανομή προσοχής όσο και την εξαγωγή χαρακτηριστικών με τρόπο **end-to-end** χρησιμοποιώντας μόνο ετικέτες ανά εικόνα για επίβλεψη.

Στη συνέχεια, στρέφουμε την προσοχή μας στις αρχιτεκτονικές **transformer** και εισάγουμε μια αναδιατύπωση της εξίσωσης αυτοπροσοχής (**self-attention**) που μειώνει την τετραγωνική πολυπλοκότητά της σε γραμμική σε σχέση με το μήκος της ακολουθίας εισόδου. Επιπλέον, αποκαλύπτουμε τη σχέση μεταξύ των ανατροφοδοτούμενων **transformer** και των αναδρομικών νευρωνικών δικτύων (**RNN**) και δείχνουμε ότι η διατύπωσή μας επιτρέπει έως και 3 τάξεις μεγέθους ταχύτερη εκτέλεση των ανατροφοδοτούμενων **transformer**.

Τέλος, αναπτύσσουμε μια μέθοδο ομαδοποιημένης προσοχής που μπορεί να προσεγγίσει τους παραδοσιακούς **transformer** με μειωμένες υπολογιστικές απαιτήσεις. Αυτό επιτυγχάνεται με την ομαδοποίηση στοιχείων της εισόδου ώστε να μειωθούν οι απαιτήσεις για τον υπολογισμό των κατανομών προσοχής. Παρουσιάζουμε ότι η μεθόδός μας παρέχει

## Περίληψη

---

καλύτερο συμβιβασμό μεταξύ απόδοσης και υπολογισμού σε σύγκριση με την αρχική αρχιτεκτονική. Επιπλέον, αποδεικνύουμε ότι η ομαδοποιημένη προσοχή μπορεί να προσεγγίσει προ-εκπαιδευμένα μοντέλα χωρίς καμία αλλαγή και με ελάχιστη απώλεια επιδόσεων.

# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract (English/Français/Deutsch/Ελληνικά)</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Dissertation outline and contributions . . . . .	3
<b>2 Not All Samples Are Created Equal</b>	<b>5</b>
2.1 Chapter Introduction . . . . .	5
2.2 Related Work . . . . .	6
2.2.1 Importance Sampling for Convex Problems . . . . .	6
2.2.2 Importance Sampling for Deep Learning . . . . .	7
2.2.3 Other Sample Selection Methods . . . . .	7
2.2.4 Stochastic Variance Reduced Gradient . . . . .	7
2.3 Variance Reduction for Deep Neural Networks . . . . .	8
2.3.1 Introduction to Importance Sampling . . . . .	8
2.3.2 Beyond the Full Gradient Norm . . . . .	9
2.3.3 When is Variance Reduction Possible? . . . . .	10
2.4 Experiments . . . . .	12
2.4.1 Ablation study . . . . .	13
2.4.2 Image classification . . . . .	15
2.4.3 Fine-tuning . . . . .	16
2.4.4 Pixel by Pixel MNIST . . . . .	17
2.5 Chapter Conclusions . . . . .	18
<b>3 Processing Megapixel Images with Deep Attention Sampling</b>	<b>19</b>
3.1 Chapter Introduction . . . . .	19
3.2 Related Work . . . . .	20
3.2.1 Recurrent visual attention models . . . . .	21
3.2.2 Patch based models . . . . .	21
3.2.3 Attention models . . . . .	21
3.2.4 Other methods . . . . .	22
3.3 Methodology . . . . .	22
3.3.1 Attention in neural networks . . . . .	22

## Contents

---

3.3.2	Attention sampling . . . . .	22
3.3.3	Multi-resolution data . . . . .	25
3.3.4	Implementation details . . . . .	26
3.4	Experimental evaluation . . . . .	26
3.4.1	Introduction . . . . .	26
3.4.2	Megapixel MNIST . . . . .	27
3.4.3	Histopathology images . . . . .	29
3.4.4	Speed limit sign detection . . . . .	31
3.5	Chapter Conclusions . . . . .	33
<b>4</b>	<b>Fast Autoregressive Transformers with Linear Attention</b>	<b>35</b>
4.1	Chapter Introduction . . . . .	35
4.2	Related Work . . . . .	36
4.2.1	Efficient Transformers . . . . .	36
4.2.2	Understanding Self-Attention . . . . .	37
4.2.3	Linearized softmax . . . . .	37
4.3	Linear Transformers . . . . .	38
4.3.1	Transformers . . . . .	38
4.3.2	Linearized Attention . . . . .	39
4.3.3	Causal Masking . . . . .	40
4.3.4	Transformers are RNNs . . . . .	43
4.4	Experiments . . . . .	44
4.4.1	Synthetic Tasks . . . . .	45
4.4.2	Image Generation . . . . .	46
4.4.3	Automatic Speech Recognition . . . . .	48
4.5	Chapter Conclusions . . . . .	50
<b>5</b>	<b>Fast Transformers with Clustered Attention</b>	<b>51</b>
5.1	Chapter Introduction . . . . .	51
5.2	Related Work . . . . .	52
5.2.1	Attention Improvements Before Transformers . . . . .	52
5.2.2	Non-asymptotic Improvements . . . . .	53
5.2.3	Improvements in Asymptotic Complexity . . . . .	53
5.3	Scaling Attention with Fast Clustering . . . . .	54
5.3.1	Vanilla Attention . . . . .	54
5.3.2	Clustered Attention . . . . .	55
5.3.3	Improving clustered attention . . . . .	57
5.4	Experiments . . . . .	58
5.4.1	Evaluation on Wall Street Journal (WSJ) . . . . .	59
5.4.2	Evaluation on Switchboard . . . . .	60
5.4.3	RoBERTa Approximation . . . . .	61
5.5	Conclusions . . . . .	62

---

<b>6</b>	<b>Conclusions and Future Work</b>	<b>63</b>
6.1	Future work . . . . .	64
<b>A</b>	<b>Appendix for Chapter 2</b>	<b>67</b>
A.1	Differences of variances . . . . .	67
A.2	An upper bound to the gradient norm . . . . .	68
A.3	Comparison with SVRG methods . . . . .	69
A.4	Ablation study on $B$ . . . . .	70
A.5	Importance Sampling with the Loss . . . . .	71
<b>B</b>	<b>Appendix for Chapter 3</b>	<b>73</b>
B.1	Introduction . . . . .	73
B.2	Sampling with replacement . . . . .	73
B.3	Sampling without replacement . . . . .	74
B.4	Extra related work . . . . .	76
B.5	Ablation study on the entropy regularizer . . . . .	76
B.6	Ablation study on the number of patches . . . . .	76
B.7	Qualitative results of the learned attention distribution . . . . .	78
	B.7.1 Histopathology images . . . . .	78
	B.7.2 Speed limits . . . . .	80
B.8	Network Architecture Details . . . . .	80
	B.8.1 Megapixel MNIST . . . . .	80
	B.8.2 Histopathology images . . . . .	80
	B.8.3 Speed Limits . . . . .	80
<b>C</b>	<b>Appendix for Chapter 4</b>	<b>83</b>
C.1	Gradient Derivation . . . . .	83
C.2	Training Evolution . . . . .	84
C.3	Image Generation Throughput Discussion . . . . .	85
	C.3.1 Stateful softmax attention . . . . .	85
	C.3.2 Equalizing the batch size . . . . .	85
C.4	Qualitative Results on Image Generation . . . . .	86
<b>D</b>	<b>Appendix for Chapter 5</b>	<b>91</b>
D.1	Scaling Attention with Fast Clustering . . . . .	91
	D.1.1 Clustered attention . . . . .	91
	D.1.2 Improved clustered attention . . . . .	92
D.2	Quality of the approximation . . . . .	93
D.3	Experiments . . . . .	95
	D.3.1 Time and Memory Benchmark . . . . .	95
	D.3.2 Ablation on clusters and sequence length . . . . .	96
	D.3.3 Automatic Speech Recognition . . . . .	98

## Contents

---

D.3.4 RoBERTa Approximation . . . . .	100
<b>Bibliography</b>	<b>103</b>
<b>Curriculum Vitae</b>	<b>115</b>



# 1 Introduction

Neural networks have long held the promise of general purpose artificial intelligence. In many respects, they failed to deliver and, even 50 years after their invention, methods like Support Vector Machines or decision trees with hand crafted features were preferred. However, around 2012, the *deep learning* revolution (Krizhevsky et al., 2012) has reignited that promise and deep neural networks are now ubiquitous throughout our technology stack.

From the early days of neural networks, it was obvious to both theorists and practitioners that their performance was tightly coupled to their size (be it the number of neurons or the number of layers). In general, larger networks generalize better to unseen datapoints which is surprising when considering more traditional learning theory. As a result, the most effective models nowadays have billions of parameters and require millions of dollars to be trained on large-scale datasets (Brown et al., 2020). This has spawned an active area of research that aims at improving the efficiency of deep neural networks, which is also the topic of this thesis.

Approaches to improve the efficiency of neural networks can be roughly categorized in four categories that focus on: (i) the hardware and software, (ii) the optimization algorithms, (iii) the network architecture and (iv) network compression.

**Hardware and software:** Moore's law, the approximate doubling of transistor density every two years since 1965, as well as the advent of general purpose computing on Graphical Processing Units (GPUs) in 2007 (Nickolls et al., 2008), have played a key role in the democratization of deep learning. Arguably the only development more important than the evolution of hardware has been the creation of efficient open source packages (Bergstra et al., 2011; Collobert et al., 2002; Abadi et al., 2016; Paszke et al., 2019) that enabled fast experimentation on a scale that was not possible before. This tight coupling between software and hardware has naturally influenced the design choices of both (Hooker, 2020). For example, commodity hardware such as GPUs now have specialized operations for large matrix multiplication and algorithms favor dense operations which are significantly more efficient on commodity hardware. Lately, the wide adoption of neural networks has led to the development of specialized hardware such as Tensor Processing Units (Jouppi et al., 2017) and Intelligence Processing Units (Jia et al.,

2019) that promise to further improve the efficiency of large neural networks and allow for more freedom during algorithm design by also enabling sparse computations.

**Optimization and sample efficiency:** One of the major limiting factors for scaling deep neural networks is related to the large computational cost of optimizing millions or even billions of parameters when training on datasets that contain millions of datapoints. The first crucial step towards reducing this computational cost was to move away from traditional optimization methods developed for convex problems such as LBFGS (Nocedal, 1980), conjugate gradient (Hestenes et al., 1952) or even gradient descent with line search (Nocedal and Wright, 2006) to stochastic optimization (Robbins and Monro, 1951; Qian, 1999). More recently, several optimization algorithms have been developed, particularly for deep learning, that greatly reduce the required time for training such models (Kingma and Ba, 2014; You et al., 2017). However, these methods still rely on naive uniform sampling of datapoints for their stochastic gradient estimator. In chapter 2, we introduce an importance sampling method that can be combined with all existing optimization methods to further reduce the computational requirements during training.

**Efficient Architectures:** Modern deep learning frameworks have allowed researchers to easily utilize any differentiable function  $f : \mathbb{R}^D \rightarrow \mathbb{R}^M$  as a layer in a neural network. This gave rise to a plethora of models that achieve comparable performance while having widely different computational characteristics. As a result, researchers focus on developing models that achieve state-of-the-art performance with significantly reduced computational cost. For example, MobileNet (Howard et al., 2017) retained state-of-the-art classification accuracy on the popular ImageNet image recognition challenge while reducing the computation cost by more than three times and the number of network parameters by 40%. Along this line of research, we present in Chapter 3, a network architecture that can process large images with an order of magnitude lower computational and memory requirements. Moreover, in Chapter 4 and 5, we present two efficient transformer variants that were among the first to reduce the asymptotic computational complexity of the transformer architecture (Vaswani et al., 2017) to linear with respect to the input's length. More recently, researchers have employed Neural Architecture Search for discovering efficient architectures that surpass existing models both in terms of accuracy and computation (Tan and Le, 2019).

**Network compression:** The previously discussed methods reduce the computational cost of neural networks in general, both during training and during inference. However, the need to deploy existing models on a variety of devices (e.g. servers, mobile devices, laptops etc.) gave rise to methods that reduce the computational cost of neural networks during inference. These methods can be roughly categorized into (i) network quantization methods (Courbariaux et al., 2015; Yang et al., 2019a) that approximate the inference using integers instead of floating point numbers, (ii) network compression methods (Liang et al., 2021) that either merge or remove parts of the network that do not contribute to the final prediction and (iii) distillation methods (Hinton et al., 2015) that improve the performance of smaller and more efficient networks using the output of larger architectures.

## 1.1 Dissertation outline and contributions

The following four chapters in this dissertation are based on four conference proceedings articles (Katharopoulos and Fleuret, 2018, 2019; Katharopoulos et al., 2020; Vyas et al., 2020). While all of them deal with improving the efficiency of neural networks, each work is independent and the chapters can be read in any order.

In Chapter 2, which is based on Katharopoulos and Fleuret (2018), we show that deep neural network training spends most of the computation on examples that are properly handled, and could be ignored for most of the training. We propose to mitigate this phenomenon with a principled importance sampling scheme that focuses computation on “informative” examples, and reduces the variance of the stochastic gradients during training. Our contribution is twofold: first, we derive a tractable upper bound to the per-sample gradient norm, and second we derive an estimator of the variance reduction achieved with importance sampling, which enables us to switch it on when it will result in an actual speedup. The resulting scheme can be used by changing a few lines of code in a standard SGD procedure, and we demonstrate experimentally, on image classification, CNN fine-tuning, and RNN training, that for a fixed wall-clock time budget, it provides a reduction of the train losses of up to an order of magnitude and a relative improvement of test errors between 5% and 17%.

In Chapter 3, which is based on Katharopoulos and Fleuret (2019), we address the large computational and memory constraints of deep architectures for processing large signals, such as megapixel images. To this end, we propose a fully differentiable end-to-end trainable model that samples and processes only a fraction of the full resolution input image. The locations to process are sampled from an attention distribution computed from a low resolution view of the input. We refer to our method as attention sampling and it can process images of several megapixels with a standard single GPU setup. We show that sampling from the attention distribution results in an unbiased estimator of the full model with minimal variance, and we derive an unbiased estimator of the gradient that we use to train our model end-to-end with a normal SGD procedure. This new method is evaluated on three classification tasks, where we show that it allows to reduce computation and memory footprint by an order of magnitude for the same accuracy as classical architectures. We also show the consistency of the sampling that indeed focuses on informative parts of the input images.

In chapters 4 and 5 we tackle the computational and memory requirements of self-attention, a crucial component of the transformer architecture. In particular, due to the quadratic complexity with respect to the input’s length, transformers are prohibitively slow for large sequences. To address this limitation, in Chapter 4 (Katharopoulos et al., 2020), we express the self-attention as a linear dot-product of kernel feature maps and make use of the associativity property of matrix products to reduce the complexity from  $\mathcal{O}(N^2)$  to  $\mathcal{O}(N)$ , where  $N$  is the sequence length. We show that this formulation permits an iterative implementation that dramatically accelerates autoregressive transformers and reveals their relationship to recurrent neural networks. Our linear transformers achieve similar performance to vanilla transformers

## Chapter 1. Introduction

---

and they are up to  $4000\times$  faster on autoregressive prediction of very long sequences.

In Chapter 5, which is based on [Vyas et al. \(2020\)](#), we propose clustered attention, which instead of computing the attention for every query, groups queries into clusters and computes attention just for the centroids. To further improve this approximation, we use the computed clusters to identify the keys with the highest attention per query and compute the exact key/query dot products. This results in a model with linear complexity with respect to the sequence length for a fixed number of clusters. We evaluate our approach on two automatic speech recognition datasets and show that our model consistently outperforms vanilla transformers for a given computational budget. Finally, we demonstrate that our model can approximate arbitrarily complex attention distributions with a minimal number of clusters by approximating a pretrained BERT model on GLUE and SQuAD benchmarks with only 25 clusters and no loss in performance.

Finally, in chapter 6, we summarize our findings and propose new directions for future research.

## 2 Not All Samples Are Created Equal

### 2.1 Chapter Introduction

The dramatic increase in available training data has made the use of deep neural networks feasible, which in turn has significantly improved the state-of-the-art in many fields, in particular computer vision and natural language processing. However, due to the complexity of the resulting optimization problem, computational cost is now the core issue in training these large architectures.

When training such models, it appears to any practitioner that not all samples are equally important; many of them are properly handled after a few epochs of training, and most could be ignored at that point without impacting the final model. To this end, we propose a novel importance sampling scheme that accelerates the training of any neural network architecture by focusing the computation on the samples that will introduce the biggest change in the parameters which reduces the variance of the gradient estimates.

For convex optimization problems, many works ([Bordes et al., 2005](#); [Zhao and Zhang, 2015](#); [Needell et al., 2014](#); [Canévet et al., 2016](#); [Richtárik and Takáč, 2013](#)) have taken advantage of the difference in importance among the samples to improve the convergence speed of stochastic optimization methods. On the other hand, for deep neural networks, sample selection methods were mainly employed to generate hard negative samples for embedding learning problems or to tackle the class imbalance problem ([Schroff et al., 2015](#); [Wu et al., 2017](#); [Simo-Serra et al., 2015](#)).

Recently, researchers have shifted their focus on using importance sampling to improve and accelerate the training of neural networks ([Alain et al., 2015](#); [Loshchilov and Hutter, 2015](#); [Schaul et al., 2015](#)). Those works, employ either the gradient norm or the loss to compute each sample's importance. However, the former is prohibitively expensive to compute and the latter is not a particularly good approximation of the gradient norm.

Compared to the aforementioned works, we derive an upper bound to the per sample gradient

norm that can be computed in a single forward pass. This results in reduced computational requirements of more than an order of magnitude compared to [Alain et al. \(2015\)](#). Furthermore, we quantify the variance reduction achieved with the proposed importance sampling scheme and associate it with the batch size increment required to achieve an equivalent variance reduction. The benefits of this are twofold, firstly we provide an intuitive metric to predict how useful importance sampling is going to be, thus we are able to decide when to switch on importance sampling during training. Secondly, we also provide theoretical guarantees for speedup, when variance reduction is above a threshold. Based on our analysis, we propose a simple to use algorithm that can be used to accelerate the training of any neural network architecture.

Our implementation is generic and can be employed by adding a single line of code in a standard Keras model training. We validate it on three independent tasks: image classification, fine-tuning and sequence classification with recurrent neural networks. Compared to existing batch selection schemes, we show that our method consistently achieves lower training loss and test error for equalized wall-clock time.

## 2.2 Related Work

Existing importance sampling methods can be roughly categorized in methods applied to convex problems and methods designed for deep neural networks.

### 2.2.1 Importance Sampling for Convex Problems

Importance sampling for convex optimization problems has been extensively studied over the last years. [Bordes et al. \(2005\)](#) developed LASVM, which is an online algorithm that uses importance sampling to train kernelized support vector machines. Later, [Richtárik and Takáč \(2013\)](#) proposed a generalized coordinate descent algorithm that samples coordinate sets in a way that optimizes the algorithm's convergence rate.

More recent works ([Zhao and Zhang, 2015](#); [Needell et al., 2014](#)) make a clear connection with the variance of the gradient estimates of stochastic gradient descent and show that the optimal sampling distribution is proportional to the per sample gradient norm. Due to the relatively simple optimization problems that they deal with, the authors resort to sampling proportionally to the norm of the inputs, which in simple linear classification is proportional to the Lipschitz constant of the per sample loss function.

Such simple importance measures do not exist for Deep Learning and the direct application of the aforementioned theory ([Alain et al., 2015](#)), requires clusters of GPU workers just to compute the sampling distribution.

### 2.2.2 Importance Sampling for Deep Learning

Importance sampling has been used in Deep Learning mainly in the form of manually tuned sampling schemes. [Bengio et al. \(2009\)](#) manually design a sampling scheme inspired by the perceived way that human children learn; in practice they provide the network with examples of increasing difficulty in an arbitrary manner. Diametrically opposite, it is common for deep embedding learning to sample hard examples because of the plethora of easy non informative ones ([Simo-Serra et al., 2015](#); [Schroff et al., 2015](#)).

More closely related to our work, [Schaul et al. \(2015\)](#) and [Loshchilov and Hutter \(2015\)](#) use the loss to create the sampling distribution. Both approaches keep a history of losses for previously seen samples, and sample either proportionally to the loss or based on the loss ranking. One of the main limitations of history based sampling, is the need for tuning a large number of hyperparameters that control the effects of “stale” importance scores; i.e. since the model is constantly updated, the importance of samples fluctuate and previous observations may poorly reflect the current situation. In particular, [Schaul et al. \(2015\)](#) use various forms of smoothing for the losses and the importance sampling weights, while [Loshchilov and Hutter \(2015\)](#) introduce a large number of hyperparameters that control when the losses are computed, when they are sorted as well as how the sampling distribution is computed based on the rank.

In comparison to all the above methods, our importance sampling scheme based on an upper bound to the gradient norm has a solid theoretical basis with clear objectives, very easy to choose hyperparameters, theoretically guaranteed speedup and can be applied to any type of network and loss function.

### 2.2.3 Other Sample Selection Methods

For completeness, we mention the work of [Wu et al. \(2017\)](#), who design a distribution (suitable only for the distance based losses) that maximizes the diversity of the losses in a single batch. In addition, [Fan et al. \(2017\)](#) use reinforcement learning to train a neural network that selects samples for another neural network in order to optimize the convergence speed. Although their preliminary results are promising, the overhead of training two networks makes the wall-clock speedup unlikely and their proposal not as appealing.

### 2.2.4 Stochastic Variance Reduced Gradient

Finally, a class of algorithms that aim to accelerate the convergence of Stochastic Gradient Descent (SGD) through variance reduction are SVRG type algorithms ([Johnson and Zhang, 2013](#); [Defazio et al., 2014](#); [Allen-Zhu, 2017](#); [Lei et al., 2017](#)). Although asymptotically better, those algorithms typically perform worse than plain SGD with momentum for the low accuracy optimization setting of Deep Learning. Contrary to the aforementioned algorithms, our proposed importance sampling does not improve the asymptotic convergence of SGD but

results in pragmatic improvements in all the metrics given a fixed time budget.

## 2.3 Variance Reduction for Deep Neural Networks

Importance sampling aims at increasing the convergence speed of SGD by focusing computation on samples that actually induce a change in the model parameters. This formally translates into a reduced variance of the gradient estimates for a fixed computational cost. In the following sections, we analyze how this works and present an efficient algorithm that can be used to train any Deep Learning model.

### 2.3.1 Introduction to Importance Sampling

Let  $x_i, y_i$  be the  $i$ -th input-output pair from the training set,  $\Psi(\cdot; \theta)$  be a Deep Learning model parameterized by the vector  $\theta$ , and  $\mathcal{L}(\cdot, \cdot)$  be the loss function to be minimized during training. The goal of training is to find

$$\theta^* = \operatorname{argmin}_{\theta} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\Psi(x_i; \theta), y_i) \quad (2.1)$$

where  $N$  corresponds to the number of examples in the training set.

We use an SGD procedure with learning rate  $\eta$ , where the update at iteration  $t$  depends on the sampling distribution  $p_1^t, \dots, p_N^t$  and re-scaling coefficients  $w_1^t, \dots, w_N^t$ . Let  $I_t$  be the data point sampled at that step, we have  $P(I_t = i) = p_i^t$  and

$$\theta_{t+1} = \theta_t - \eta w_{I_t} \nabla_{\theta_t} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t}) \quad (2.2)$$

Plain SGD with uniform sampling is achieved with  $w_i^t = 1$  and  $p_i^t = \frac{1}{N}$  for all  $t$  and  $i$ .

If we define the convergence speed  $S$  of SGD as the reduction of the distance of the parameter vector  $\theta$  from the optimal parameter vector  $\theta^*$  in two consecutive iterations  $t$  and  $t + 1$

$$S = -\mathbb{E}_{P_t} \left[ \|\theta_{t+1} - \theta^*\|_2^2 - \|\theta_t - \theta^*\|_2^2 \right], \quad (2.3)$$

and if we have  $w_i = \frac{1}{N p_i}$  such that

$$\mathbb{E}_{P_t} \left[ w_{I_t} \nabla_{\theta_t} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t}) \right] \quad (2.4)$$

$$= \nabla_{\theta_t} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\Psi(x_i; \theta_t), y_i), \quad (2.5)$$

and set  $G_i = w_i \nabla_{\theta_t} \mathcal{L}(\Psi(x_i; \theta_t), y_i)$ , then we get (this is a different derivation of the result by



Wang et al., 2016)

$$\begin{aligned}
 S &= -\mathbb{E}_{P_t} \left[ (\theta_{t+1} - \theta^*)^T (\theta_{t+1} - \theta^*) - (\theta_t - \theta^*)^T (\theta_t - \theta^*) \right] \\
 &= -\mathbb{E}_{P_t} \left[ \theta_{t+1}^T \theta_{t+1} - 2\theta_{t+1}^T \theta^* - \theta_t^T \theta_t + 2\theta_t^T \theta^* \right] \\
 &= -\mathbb{E}_{P_t} \left[ (\theta_t - \eta G_{I_t})^T (\theta_t - \eta G_{I_t}) + 2\eta G_{I_t}^T \theta^* - \theta_t^T \theta_t \right] \\
 &= -\mathbb{E}_{P_t} \left[ -2\eta (\theta_t - \theta^*)^T G_{I_t} + \eta^2 G_{I_t}^T G_{I_t} \right] \\
 &= 2\eta (\theta_t - \theta^*)^T \mathbb{E}_{P_t} [G_{I_t}] - \eta^2 \mathbb{E}_{P_t} [G_{I_t}^T G_{I_t}] - \\
 &\quad \eta^2 \text{Tr}(\mathbb{V}_{P_t} [G_{I_t}])
 \end{aligned} \tag{2.6}$$

Since the first two terms, in the last expression, are the speed of batch gradient descent, we observe that it is possible to gain a speedup by sampling from the distribution that minimizes  $\text{Tr}(\mathbb{V}_{P_t} [G_{I_t}])$ . Several works (Needell et al., 2014; Zhao and Zhang, 2015; Alain et al., 2015) have shown the optimal distribution to be proportional to the per-sample gradient norm. However, computing this distribution is computationally prohibitive.

### 2.3.2 Beyond the Full Gradient Norm

Given an upper bound  $\hat{G}_i \geq \|\nabla_{\theta_t} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2$  and due to

$$\underset{P}{\text{argmin}} \text{Tr}(\mathbb{V}_{P_t} [G_{I_t}]) = \underset{P}{\text{argmin}} \mathbb{E}_{P_t} \left[ \|G_{I_t}\|_2^2 \right], \tag{2.7}$$

we propose to relax the optimization problem in the following way

$$\min_P \mathbb{E}_{P_t} \left[ \|G_{I_t}\|_2^2 \right] \leq \min_P \mathbb{E}_{P_t} \left[ w_{I_t}^2 \hat{G}_{I_t}^2 \right]. \tag{2.8}$$

The minimizer of the second term of equation 2.8, similar to the first term, is  $p_i \propto \hat{G}_i$ . All that remains, is to find a proper expression for  $\hat{G}_i$  which is significantly easier to compute than the norm of the gradient for each sample.

In order to continue with the derivation of our upper bound  $\hat{G}_i$ , let us introduce some notation specific to a multi-layer perceptron. Let  $\theta^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$  be the weight matrix for layer  $l$  and  $\sigma^{(l)}(\cdot)$  be a Lipschitz continuous activation function. Then, let

$$x^{(0)} = x \tag{2.9}$$

$$z^{(l)} = \theta^{(l)} x^{(l-1)} \tag{2.10}$$

$$x^{(l)} = \sigma^{(l)}(z^{(l)}) \tag{2.11}$$

$$\Psi(x; \Theta) = x^{(L)} \tag{2.12}$$

Although our notation describes simple fully connected neural networks without bias, our analysis holds for any affine operation followed by a slope-bounded non-linearity ( $|\sigma'(x)| \leq K$ ).

With

$$\Sigma'_l(z) = \text{diag}\left(\sigma'^{(l)}(z_1), \dots, \sigma'^{(l)}(z_{M_l})\right), \quad (2.13)$$

$$\Delta_i^{(l)} = \Sigma'_l(z_i^{(l)})\theta_{l+1}^T \dots \Sigma'_{L-1}(z_i^{(L-1)})\theta_L^T, \quad (2.14)$$

$$\nabla_{x_i^{(l)}} \mathcal{L} = \nabla_{x_i^{(l)}} \mathcal{L}(\Psi(x_i; \Theta), y_i) \quad (2.15)$$

we get

$$\|\nabla_{\theta_l} \mathcal{L}(\Psi(x_i; \Theta), y_i)\|_2 \quad (2.16)$$

$$= \left\| \left( \Delta_i^{(l)} \Sigma'_L(z_i^{(L)}) \nabla_{x_i^{(L)}} \mathcal{L} \right) \left( x_i^{(L-1)} \right)^T \right\|_2 \quad (2.17)$$

$$\leq \|\Delta_i^{(l)}\|_2 \left\| \Sigma'_L(z_i^{(L)}) \nabla_{x_i^{(L)}} \mathcal{L} \right\|_2 \|x_i^{(L-1)}\|_2 \quad (2.18)$$

$$\leq \underbrace{\max_{l,i} \left( \|x_i^{(L-1)}\|_2 \|\Delta_i^{(l)}\|_2 \right)}_{\rho} \left\| \Sigma'_L(z_i^{(L)}) \nabla_{x_i^{(L)}} \mathcal{L} \right\|_2 \quad (2.19)$$

Various weight initialization (Glorot and Bengio, 2010) and activation normalization techniques (Ioffe and Szegedy, 2015; Ba et al., 2016) uniformise the activations across samples. As a result, the variation of the gradient norm is mostly captured by the gradient of the loss function with respect to the pre-activation outputs of the last layer of our neural network. Consequently we can derive the following upper bound to the gradient norm of all the parameters

$$\|\nabla_{\Theta} \mathcal{L}(\Psi(x_i; \Theta), y_i)\|_2 \leq L\rho \underbrace{\left\| \Sigma'_L(z_i^{(L)}) \nabla_{x_i^{(L)}} \mathcal{L} \right\|_2}_{\hat{G}_i}, \quad (2.20)$$

which is marginally more difficult to compute than the value of the loss since it can be computed in a closed form in terms of  $z^{(L)}$ . However, our upper bound depends on the time step  $t$ , thus we cannot generate a distribution once and sample from it during training. This is intuitive because the importance of each sample changes as the model changes.

### 2.3.3 When is Variance Reduction Possible?

Computing the importance score from equation 2.20 is more than an order of magnitude faster compared to computing the gradient norm for each sample. Nevertheless, it still costs one forward pass through the network and can be wasteful. For instance, during the first iterations of training, the gradients with respect to every sample have approximately equal norm; thus we would waste computational resources trying to sample from the uniform distribution. In addition, computing the importance score for the whole dataset is still prohibitive and would render the method unsuitable for online learning.

In order to solve the problem of computing the importance for the whole dataset, we *pre-sample* a large batch of data points, compute the sampling distribution for that batch and

re-sample a smaller batch with replacement. The above procedure upper bounds both the speedup and variance reduction. Given a large batch consisting of  $B$  samples and a small one consisting of  $b$ , we can achieve a maximum variance reduction of  $\frac{1}{b} - \frac{1}{B}$  and a maximum speedup of  $\frac{B+3b}{3B}$  assuming that the backward pass requires twice the amount of time as the forward pass.

Due to the large cost of computing the importance per sample, we only perform importance sampling when we know that the variance of the gradients can be reduced. In the following equation, we show that the variance reduction is proportional to the squared  $L_2$  distance of the sampling distribution,  $g$ , to the uniform distribution  $u$ . The complete derivation is included in appendix A.1. Let  $g_i \propto \|\nabla_{\theta_t} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2 = \|G_i\|_2$  and  $u = \frac{1}{B}$  the uniform probability.

$$\text{Tr}(\mathbb{V}_u[G_i]) - \text{Tr}(\mathbb{V}_g[w_i G_i]) \quad (2.21)$$

$$= \mathbb{E}_u[\|G_i\|_2^2] - \mathbb{E}_g[w_i^2 \|G_i\|_2^2] \quad (2.22)$$

$$= \left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2 B \|g - u\|_2^2. \quad (2.23)$$

Equation 2.23 already provides us with a useful metric to decide if the variance reduction is significant enough to justify using importance sampling. However, choosing a suitable threshold for the  $L_2$  distance squared would be tedious and unintuitive. We can do much better by dividing the variance reduction with the original variance to derive the increase in the batch size that would achieve an equivalent variance reduction. Assuming that we increase the batch size by  $\tau$ , we achieve variance reduction  $\frac{1}{\tau}$ ; thus we have<sup>1</sup>

$$\frac{\left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2 B \|g - u\|_2^2}{\text{Tr}(\mathbb{V}_u[G_i])} \geq \quad (2.24)$$

$$\frac{\left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2 B \|g - u\|_2^2}{\frac{1}{B} \sum_{i=1}^B \|G_i\|_2^2} = \quad (2.25)$$

$$\frac{1}{\sum_{i=1}^B g_i^2} \|g - u\|_2^2 = 1 - \frac{1}{\tau} \iff \quad (2.26)$$

$$\frac{1}{\tau} = 1 - \frac{1}{\sum_{i=1}^B g_i^2} \|g - u\|_2^2 \quad (2.27)$$

Using equation 2.27, we have a hyperparameter that is very easy to select and can now design our training procedure which is described in pseudocode in algorithm 1. Computing  $\tau$  from equation 2.27 allows us to have guaranteed speedup when  $B + 3b < 3\tau b$ . However, as it is shown in the experiments, we can use  $\tau_{th}$  smaller than  $\frac{B+3b}{3b}$  and still get a significant speedup.

The inputs to the algorithm are the pre-sampling size  $B$ , the batch size  $b$ , the equivalent batch size increment after which we start importance sampling  $\tau_{th}$  and the exponential

<sup>1</sup>In the first version we mistakenly assume  $\frac{1}{\tau^2}$  which made the algorithm unnecessarily conservative. All the experiments are run using the square root of line 17 in Algorithm 1.

## Chapter 2. Not All Samples Are Created Equal

---

### Algorithm 1 Deep Learning with Importance Sampling

---

```
1: Inputs  $B, b, \tau_{th}, a_\tau, \theta_0$ 
2:  $t \leftarrow 1$ 
3:  $\tau \leftarrow 0$ 
4: repeat
5:   if  $\tau > \tau_{th}$  then
6:      $\mathcal{U} \leftarrow B$  uniformly sampled datapoints
7:      $g_i \propto \hat{G}_i \quad \forall i \in \mathcal{U}$  according to eq 2.20
8:      $\mathcal{G} \leftarrow b$  datapoints sampled with  $g_i$  from  $\mathcal{U}$ 
9:      $w_i \leftarrow \frac{1}{B g_i} \quad \forall i \in \mathcal{G}$ 
10:     $\theta_t \leftarrow \text{sgd\_step}(w_i, \mathcal{G}, \theta_{t-1})$ 
11:   else
12:      $\mathcal{U} \leftarrow b$  uniformly sampled datapoints
13:      $w_i \leftarrow 1 \quad \forall i \in \mathcal{U}$ 
14:      $\theta_t \leftarrow \text{sgd\_step}(w_i, \mathcal{U}, \theta_{t-1})$ 
15:      $g_i \propto \hat{G}_i \quad \forall i \in \mathcal{U}$ 
16:   end if
17:    $\tau \leftarrow a_\tau \tau + (1 - a_\tau) \left( 1 - \frac{1}{\sum_i g_i^2} \left\| g - \frac{1}{|\mathcal{U}|} \right\|_2^2 \right)^{-1}$ 
18: until convergence
```

---

moving average parameter  $a_\tau$  used to compute a smooth estimate of  $\tau$ .  $\theta_0$  denotes the initial parameters of our deep network. We would like to point out that in line 15 of the algorithm, we compute  $g_i$  for free since we have done the forward pass in the previous step.

The only parameter that has to be explicitly defined for our algorithm is the pre-sampling size  $B$  because  $\tau_{th}$  can be set using equation 2.27. A small analysis on the impact of  $B$  is provided in appendix A.4.

## 2.4 Experiments

In this section, we analyse experimentally the performance of the proposed importance sampling scheme based on our *upper-bound* of the gradient norm. In the first subsection, we compare the variance reduction achieved with our upper bound to the theoretically maximum achieved with the true gradient norm. We also compare against sampling based on the loss, which is commonly used in practice. Subsequently, we conduct experiments which demonstrate that we are able to achieve non-negligible wall-clock speedup for a variety of tasks using our importance sampling scheme.

In all the subsequent sections, we use *uniform* to refer to the usual training algorithm that samples points from a uniform distribution, we use *loss* to refer to algorithm 1 but instead of sampling from a distribution proportional to our upper-bound to the gradient norm  $\hat{G}_i$  (equations 2.8 and 2.20), we sample from a distribution proportional to the loss value and finally *upper-bound* to refer to our proposed method. All the other baselines from published

methods are referred to using the names of the authors.

In addition to batch selection methods, we compare with various SVRG implementations including the accelerated Katyusha (Allen-Zhu, 2017) and the online SCSG (Lei et al., 2017) method. In all cases, SGD with uniform sampling performs significantly better. The detailed results are provided in appendix A.3.

Experiments were conducted using Keras (Chollet et al., 2015) with TensorFlow (Abadi et al., 2016), and the code can be found at <http://github.com/idiap/importance-sampling>. For all the experiments, we use Nvidia K80 GPUs and the reported time is calculated by subtracting the timestamps before starting one epoch and after finishing one; thus it includes the time needed to transfer data between CPU and GPU memory.

Our implementation provides a wrapper around models that substitutes the standard uniform sampling with our importance-sampling method. This means that *adding a single line of code* to call this wrapper before actually fitting the model is sufficient to switch from the standard uniform sampling to our importance-sampling scheme. And, as specified in § 2.3.3 and Algorithm 1, our procedure reliably estimates at every iteration if the importance sampling will provide a speed-up and sticks to uniform sampling otherwise.

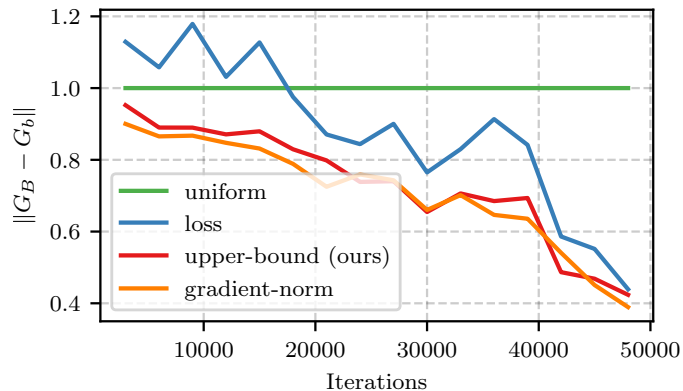


Figure 2.1 – The y-axis denotes the  $L_2$  distance of the average gradient of the large batch ( $G_B$ ) and the average gradient of the small batch ( $G_b$ ) normalized with the distance achieved by uniform sampling. The sampling of the small batch is done 10 times and the reported results are the average. The details of the experimental setup are given in § 2.4.1.

### 2.4.1 Ablation study

As already mentioned, several works (Loshchilov and Hutter, 2015; Schaul et al., 2015) use the loss value, directly or indirectly, to generate sampling distributions. In this section, we present experiments that validate the superiority of our method with respect to the loss in terms of variance reduction. For completeness, in appendix A.5, we include a theoretical analysis that explains why sampling based on the loss also achieves variance reduction during the late stages of training.

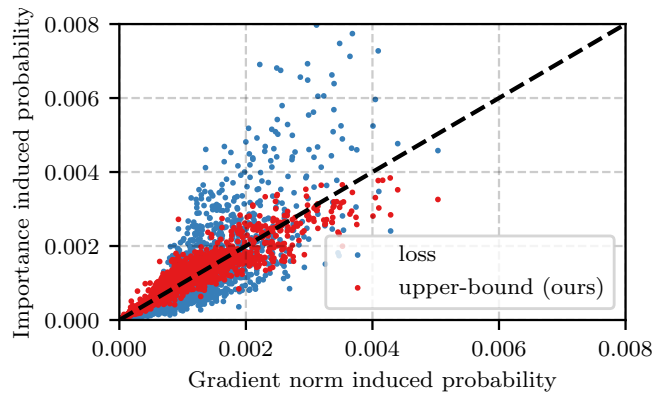


Figure 2.2 – The probabilities generated with the *loss* and our *upper-bound* are plotted against the ideal probabilities produced by the *gradient-norm*. The black line denotes perfect correlation. The details of the experimental setup are given in § 2.4.1.

Our experimental setup is as follows: we train a wide residual network (Zagoruyko and Komodakis, 2016) on the CIFAR100 dataset (Krizhevsky, 2009), following closely the training procedure of Zagoruyko and Komodakis (2016) (the details are presented in § 2.4.2). Subsequently, we sample 1,024 images uniformly at random from the dataset. Using the weights of the trained network, at intervals of 3,000 updates, we resample 128 images from the large batch of 1,024 images using *uniform* sampling or importance sampling with probabilities proportional to the *loss*, our *upper-bound* or the *gradient-norm*. The *gradient-norm* is computed by running the backpropagation algorithm with a batch size of 1.

Figure 2.1 depicts the variance reduction achieved with every sampling scheme in comparison to *uniform*. We measure this directly as the distance between the mini-batch gradient and the batch gradient of the 1,024 samples. For robustness we perform the sampling 10 times and report the average. We observe that our upper bound and the gradient norm result in very similar variance reduction, meaning that the bound is relatively tight and that the produced probability distributions are highly correlated. This can also be deduced by observing figure 2.2, where the probabilities proportional to the *loss* and the *upper-bound* are plotted against the optimal ones (proportional to the *gradient-norm*). We observe that our upper bound is almost perfectly correlated with the gradient norm, in stark contrast to the *loss* which is only correlated at the regime of very small gradients. Quantitatively the sum of squared error of 16,384 points in figure 2.2 is 0.017 for the *loss* and 0.002 for our proposed upper bound.

Furthermore, we observe that sampling hard examples (with high loss), increases the variance, especially in the beginning of training. Similar behaviour has been observed in problems such as embedding learning where semi-hard sample mining is preferred over sampling using the *loss* (Wu et al., 2017; Schroff et al., 2015).

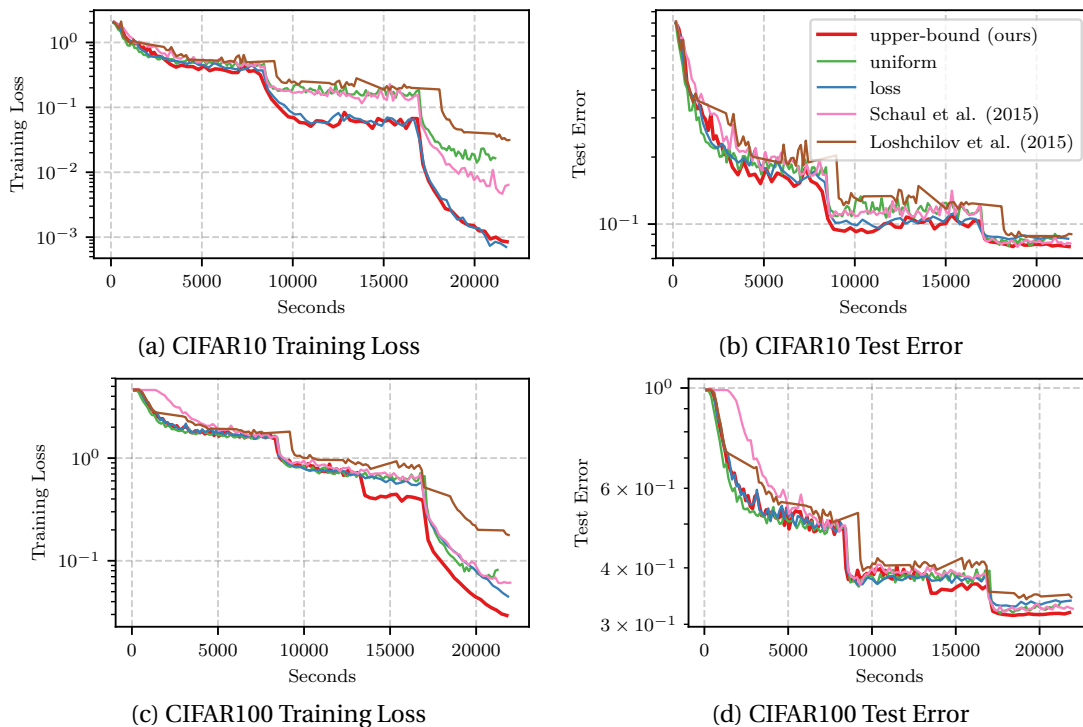


Figure 2.3 – Comparison of importance sampling using the *upper-bound* with *uniform* and *loss* based importance sampling. The details of the training procedure are given in § 2.4.2. Our proposed scheme is the only one achieving a speedup on CIFAR100 and results in 5% smaller test error. All presented results are averaged across 3 independent runs.

### 2.4.2 Image classification

In this section, we use importance sampling to train a residual network on CIFAR10 and CIFAR100. We follow the experimental setup of [Zagoruyko and Komodakis \(2016\)](#), specifically we train a wide resnet 28-2 with SGD with momentum. We use batch size 128, weight decay 0.0005, momentum 0.9, initial learning rate 0.1 divided by 5 after 20,000 and 40,000 parameter updates. Finally, we train for a total of 50,000 iterations. In order for our history based baselines to be compatible with the data augmentation of the CIFAR images, we pre-augment both datasets to generate  $1.5 \times 10^6$  images for each one. Our method does not have this limitation since it can work on infinite datasets in a true online fashion. To compare between methods, we use a learning rate schedule based on wall-clock time and we also fix the total seconds available for training. A faster method should have smaller training loss and test error given a specific time during training.

For this experiment, we compare the proposed method to *uniform*, *loss*, online batch selection by [Loshchilov and Hutter \(2015\)](#) and the history based sampling of [Schaul et al. \(2015\)](#). For the method of [Schaul et al. \(2015\)](#), we use their proportional sampling since the rank based is very similar to [Loshchilov and Hutter \(2015\)](#) and we select the best parameters from the grid  $a = \{0.1, 0.5, 1.0\}$  and  $\beta = \{0.5, 1.0\}$ . Similarly, for online batch selection, we use  $s = \{1, 10, 10^2\}$

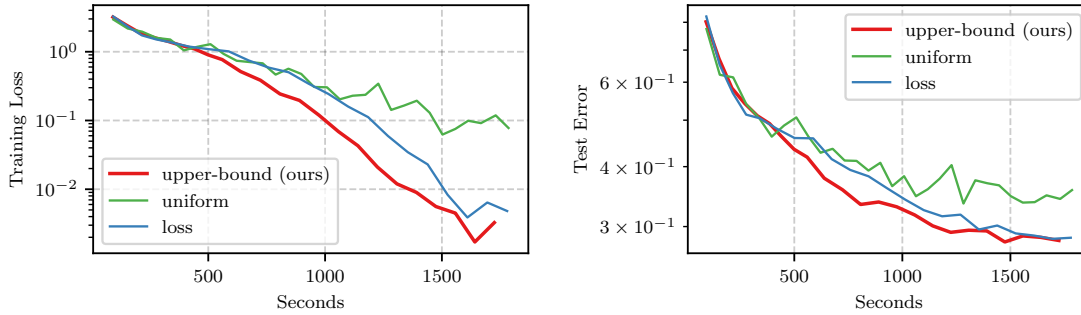


Figure 2.4 – Comparison of importance sampling for fine-tuning on MIT67 dataset. The details of the training procedure are given in § 2.4.3. Our proposed algorithm converges very quickly to 28.06% test error in approximately half an hour, a relative reduction of 17% to uniform sampling. For robustness, the results are averaged across 3 independent runs.

and a recomputation of all the losses every  $r = \{600, 1200, 3600\}$  updates.

For our method, we use a presampling size of 640. One of the goals of this experiment is to show that even a smaller reduction in variance can effectively stabilize training and provide wall-clock time speedup; thus we set  $\tau_{th} = 1.5$ . We perform 3 independent runs and report the average.

The results are depicted in figure 2.3. We observe that in the relatively easy CIFAR10 dataset, all methods can provide some speedup over uniform sampling. However, for the more complicated CIFAR100, only sampling with our proposed *upper-bound* to the gradient norm reduces the variance of the gradients and provides faster convergence. Examining the training evolution in detail, we observe that on CIFAR10 our method is the only one that achieves a significant improvement in the test error even in the first stages of training (4,000 to 8,000 seconds). Quantitatively, on CIFAR10 we achieve more than an order of magnitude lower training loss and 8% lower test error from 0.087 to 0.079 while on CIFAR100 approximately 3 times lower training loss and 5% lower test error from 0.34 to 0.32 compared to *uniform* sampling.

At this point, we would also like to discuss the performance of the *loss* compared to other methods that also select batches based on this metric. Our experiments show, that using “fresh” values for the loss combined with a warmup stage so that importance sampling is not started too early outperforms all the other baselines on the CIFAR10 dataset.

### 2.4.3 Fine-tuning

Our second experiment shows the application of importance sampling to the significant task of fine tuning a pre-trained large neural network on a new dataset. This task is of particular importance because there exists an abundance of powerful models pre-trained on large datasets such as ImageNet (Deng et al., 2009).



Our experimental setup is the following, we fine-tune a ResNet-50 (He et al., 2015) previously trained on ImageNet. We replace the last classification layer and then train the whole network end-to-end to classify indoor images among 67 possible categories (Quattoni and Torralba, 2009). We use SGD with learning rate  $10^{-3}$  and momentum 0.9. We set the batch size to 16 and for our importance sampling algorithm we pre-sample 48. The variance reduction threshold is set to 2 as designated by equation 2.27.

To assess the performance of both our algorithm and our gradient norm approximation, we compare the convergence speed of our importance sampling algorithm using our *upper-bound* and using the *loss*. Once again, for robustness, we run 3 independent runs and report the average.

The results of the experiment are depicted in figure 2.4. As expected, importance sampling is very useful for the task of fine-tuning since a lot of samples are handled correctly very early in the training process. Our *upper-bound*, once again, greatly outperforms sampling proportionally to the loss when the network is large and the problem is non trivial. Compared to uniform sampling, in just half an hour importance sampling has converged close to the best performance (28.06% test error) that can be expected on this dataset without any data augmentation or multiple crops (Razavian et al., 2014), while uniform achieves only 33.74%.

#### 2.4.4 Pixel by Pixel MNIST

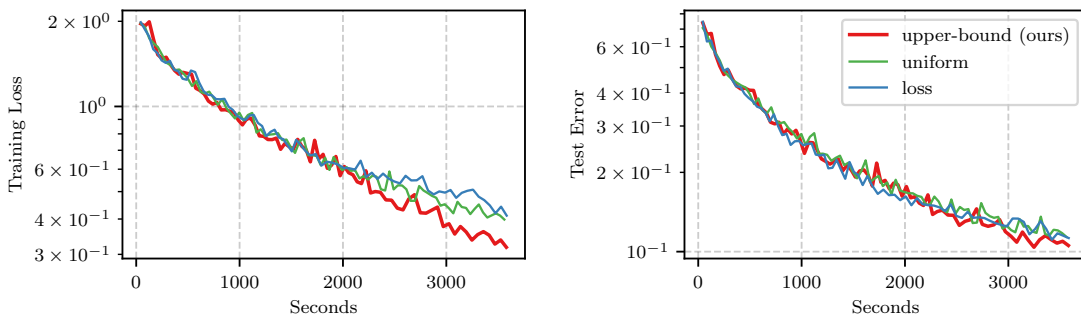


Figure 2.5 – Comparison of importance sampling on pixel-by-pixel MNIST with an LSTM. The details of the training procedure are given in § 2.4.4. Our proposed algorithm speeds up training and achieves 7% lower test error in one hour of training (0.1055 compared to 0.1139). We observe that sampling proportionally to the loss actually hurts convergence in this case.

To showcase the generality of our method, we use our importance sampling algorithm to accelerate the training of an LSTM in a sequence classification problem. We use the pixel by pixel classification of randomly permuted MNIST digits (LeCun et al., 2010), as defined by Le et al. (2015). The problem may seem trivial at first, however as shown by Le et al. (2015) it is particularly suited to benchmarking the training of recurrent neural networks, due to the long range dependency problems inherent in the dataset (784 time steps).

For our experiment, we fix a permutation matrix for all the pixels to generate a training set of

60,000 samples with 784 time steps each. Subsequently, we train an LSTM (Hochreiter and Schmidhuber, 1997) with 128 dimensions in the hidden space,  $\tanh(\cdot)$  as an activation function and  $\text{sigmoid}(\cdot)$  as the recurrent activation function. Finally, we use a linear classifier on top of the LSTM to choose a digit based on the hidden representation. To train the aforementioned architecture, we use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of  $10^{-3}$  and a batch size of 32. We have also found gradient clipping to be necessary for the training not to diverge; thus we clip the norm of all gradients to 1.

The results of the experiment are depicted in figure 2.5. Both for the *loss* and our proposed *upper-bound*, importance sampling starts at around 2,000 seconds by setting  $\tau_{th} = 1.8$  and the presampling size to 128. We could set  $\tau_{th} = 2.33$  (equation 2.27) which would only result in our algorithm being more conservative and starting importance sampling later. We clearly observe that sampling proportionally to the loss hurts the convergence in this case. On the other hand, our algorithm achieves 20% lower training loss and 7% lower test error in the given time budget.

## 2.5 Chapter Conclusions

We have presented an efficient algorithm for accelerating the training of deep neural networks using importance sampling. Our algorithm takes advantage of a novel upper bound to the gradient norm of any neural network that can be computed in a single forward pass. In addition, we show an equivalence of the variance reduction with importance sampling to increasing the batch size; thus we are able to quantify both the variance reduction and the speedup and intelligently decide when to stop sampling uniformly.

Our experiments show that our algorithm is effective in reducing the training time for several tasks both on image and sequence data. More importantly, we show that not all data points matter equally in the duration of training, which can be exploited to gain a speedup or better quality gradients or both.

Our analysis opens several avenues of future research. The two most important ones that were not investigated in this work are automatically tuning the learning rate based on the variance of the gradients and decreasing the batch size. The variance of the gradients can be kept stable by increasing the learning rate proportionally to the batch increment or by decreasing the number of samples for which we compute the backward pass. Thus, we can speed up convergence by increasing the step size or reducing the time per update.

In the next chapter, we utilize the idea of importance sampling to focus computation not on informative parts of the dataset but on informative parts of a single input.

# 3 Processing Megapixel Images with Deep Attention Sampling

## 3.1 Chapter Introduction

For a variety of computer vision tasks, such as cancer detection, self driving vehicles, and satellite image processing, it is necessary to develop models that are able to handle high resolution images. The existing CNN architectures, that provide state-of-the-art performance in various computer vision fields such as image classification (He et al., 2016), object detection (Liu et al., 2016), semantic segmentation (Wu et al., 2019) etc., cannot operate directly on such images due to computational and memory requirements. To address this issue, a common practice is to downsample the original image before passing it to the network. However, this leads to loss of significant information possibly critical for certain tasks.

Another research direction seeks to mitigate this problem by splitting the original high resolution image into patches and processing them separately (Hou et al., 2016; Golatkar et al., 2018; Nazeri et al., 2018). Naturally, these methods either waste computational resources on uninformative patches or require ground truth annotations for each patch. However, per patch labels are typically expensive to acquire and are not available for the majority of the available datasets.

The aforementioned limitations are addressed by two disjoint lines of work: the recurrent visual attention models (Mnih et al., 2014; Ba et al., 2014) and the attention based multiple instance learning (Ilse et al., 2018). The first seeks to limit the wasteful computations by only processing some parts of the full image. However, these models result in hard optimization problems that limit their applicability to high resolution images. The second line of work shows that regions of interest can be identified without explicit patch annotations by aggregating per patch features with an attention mechanism. Nevertheless, such methods do not address the computational and memory issues inherent in all patch based models.

This work aims at combining the benefits of both. Towards this goal, we propose an end-to-end trainable model able to handle multi-megapixel images using a single GPU or CPU. In particular, we sample locations of “informative patches” from an “attention distribution”

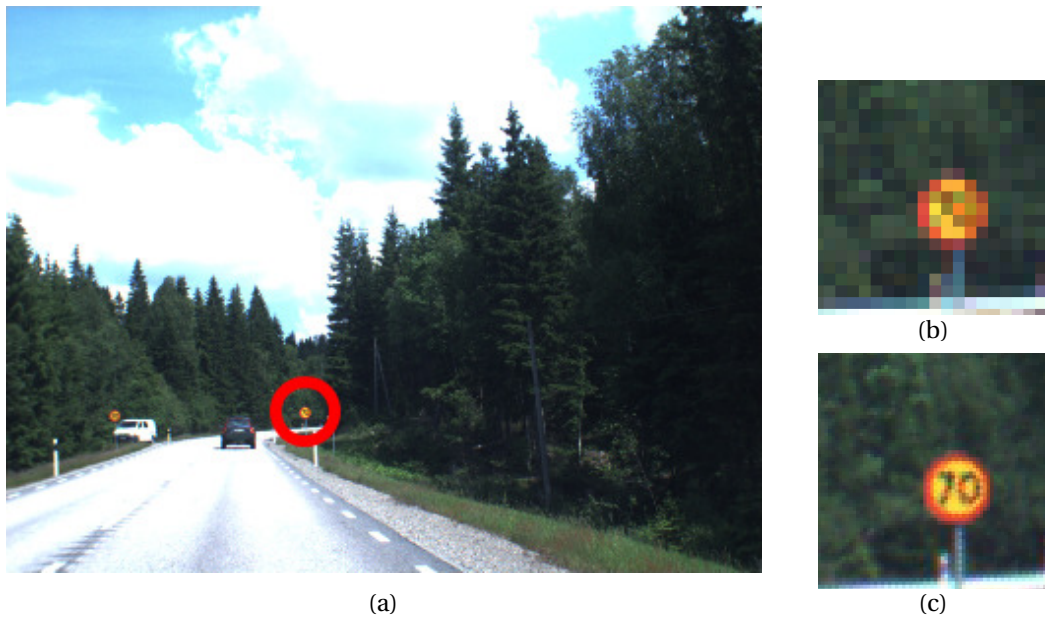


Figure 3.1 – Common practice to process megapixel images with CNNs is to downsample them, however this results in significant loss of information (b, c).

computed on a lower resolution version of the original image. This allows us to only process a fraction of the original image. Compared to previous works, due to our attention based weighted average feature aggregation, we are able to derive an unbiased estimator of the gradient of the virtual and intractable “full model” that would process the full-scale image in a standard feed-forward manner, and do not need to resort to reinforcement learning or variational methods to train. Furthermore, we prove that sampling patches from the attention distribution results in the minimum variance estimator of the “full model”.

We evaluate our model on three classification tasks and we show that our proposed *attention sampling* achieves comparable test errors with [Ilse et al. \(2018\)](#), that considers all patches from the high resolution images, while being up to  $25\times$  faster and requiring up to  $30\times$  less memory.

## 3.2 Related Work

In this section, we discuss the most relevant body of work on attention-based models and techniques to process high resolution images using deep neural networks, which can be trained from a scene-level categorical label. Region proposal methods that require per-patch annotations, such as instance-level bounding boxes ([Girshick et al., 2014](#); [Redmon et al., 2016](#); [Liu et al., 2016](#)), do not fall in that category.

### 3.2.1 Recurrent visual attention models

This line of work includes models that learn to extract a sequence of regions from the original high resolution image and only process these at high resolution. The regions are processed in a sequential manner, namely the distribution to sample the  $n$ -th region depends on the previous  $n - 1$  regions. [Mnih et al. \(2014\)](#) were the first to employ a recurrent neural network to predict regions of interest on the high resolution image and process them sequentially. In order to train their model, which is not differentiable, they use reinforcement learning. In parallel, [Ranzato \(2014\)](#); [Ba et al. \(2014\)](#) proposed to additionally downsample the input image and use it to provide spatial context to the recurrent network. [Ramapuram et al. \(2018\)](#) improved upon the previous works by using variational inference and Spatial Transformer Networks ([Jaderberg et al., 2015](#)) to solve the same optimization problem.

All the aforementioned works seek to solve a complicated optimization problem that is non differentiable and is approximated with either reinforcement learning or variational methods. Instead of employing such a complicated model to aggregate the features of the patches and generate dependent attention distributions that result in a hard optimization problem, we propose to use an attention distribution to perform a weighted average of the features, which allows us to directly train our model with SGD.

### 3.2.2 Patch based models

Such models ([Hou et al., 2016](#); [Liu et al., 2017](#); [Nazeri et al., 2018](#)) divide the high resolution image into patches and process them separately. Due to the lack of per patch annotations, the above models need to introduce a separate method to provide labels for training the patch level network. Instead *attention sampling* does not require any patch annotations and through the attention mechanism learns to identify regions of interest in arbitrarily large images.

### 3.2.3 Attention models

[Xu et al. \(2015\)](#) were the first to use soft attention methods to generate image captions. More related to our work is the model of [Ilse et al. \(2018\)](#), where they use the attention distribution to aggregate a bag of features. To apply their method to images, they extract patches, compute features and aggregate them with an attention distribution that is computed from these features. This allows them to infer regions of interest without having access to per-patch labels. However, their model wastes computational resources by handling all patches, both informative and non-informative. Our method, instead, learns to focus only on informative regions of the image, thus resulting in orders of magnitude faster computation while retaining equally good performance.

### 3.2.4 Other methods

Jaderberg et al. (2015) propose Spatial Transformer Networks (STN) that learn to predict affine transformations of a feature map than includes cropping and rescaling. STNs employ several localization networks, that operate on the full image, to generate these transformations. As a result, they do not scale easily to megapixel images or larger. Recasens et al. (2018) use a low resolution view of the image to predict a saliency map that is used in conjunction with the differentiable STN sampler to focus on useful regions of the high resolution image by making them larger. In comparison, *attention sampling* focuses on regions by weighing the corresponding features with the attention weights.

## 3.3 Methodology

In this section, we formalize our proposed *attention-sampling* method. Initially, we introduce a generic formulation for attention and we show that sampling from the attention distribution generates an optimal approximation in terms of variance that significantly reduces the required computation. In § 3.3.2, we derive the gradient with respect to the parameters of the attention and the feature network through the sampling procedure. In § 3.3.3 and § 3.3.4, we provide the methodology that allows us to speed up the processing of high resolution images using *attention sampling* and is used throughout our experiments.

### 3.3.1 Attention in neural networks

Let  $x, y$  denote an input-target pair from our dataset. We consider  $\Psi(x; \Theta) = g(f(x; \Theta); \Theta)$  to be a neural network parameterized by  $\Theta$ .  $f(x; \Theta) \in \mathbb{R}^{K \times D}$  is an intermediate representation of the neural network that can be thought of as  $K$  features of dimension  $D$ , e.g. the last convolutional layer of a ResNet architecture or the previous hidden states and outputs of a recurrent neural network.

Employing an attention mechanism in the neural network  $\Psi(\cdot)$  at the intermediate representation  $f(\cdot)$  is equivalent to defining a function  $a(x; \Theta) \in \mathbb{R}_+^K$  s.t.  $\sum_{i=1}^K a(x; \Theta)_i = 1$  and changing the definition of the network to

$$\Psi(x; \Theta) = g \left( \sum_{i=1}^K a(x; \Theta)_i f(x; \Theta)_i \right), \quad (3.1)$$

given that the subscript  $i$  extracts the  $i$ -th row from a matrix or the  $i$ -th element from a vector.

### 3.3.2 Attention sampling

By definition,  $a(\cdot)$  is a multinomial distribution over  $K$  discrete elements (e.g. locations in the images). Let  $I$  be a random variable sampled from  $a(x; \Theta)$ . We can rewrite the attention in the neural network  $\Psi(\cdot)$  as the expectation of the intermediate features over the attention

distribution  $a(\cdot)$

$$\Psi(x; \Theta) = g \left( \sum_{i=1}^K a(x; \Theta)_i f(x; \Theta)_i \right) \quad (3.2)$$

$$= g \left( \mathbb{E}_{I \sim a(x; \Theta)} [f(x; \Theta)_I] \right). \quad (3.3)$$

Consequently, we can avoid computing all  $K$  features by approximating the expectation with a Monte Carlo estimate. We sample a set  $Q$  of  $N$  i.i.d. indices from the attention distribution,  $Q = \{q_i \sim a(x; \Theta) \mid i \in \{1, 2, \dots, N\}\}$  and approximate the neural network with

$$\Psi(x; \Theta) \approx g \left( \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q \right). \quad (3.4)$$

### Relation with importance-sampling

We are interested in deriving an approximation with minimum variance so that the output of the network does not change because of the sampling. In the following paragraphs, we show that sampling from  $a(x; \Theta)$  is optimal in that respect.

Let  $P$  denote a discrete probability distribution on the  $K$  features with probabilities  $p_i$ . We want to sample from  $P$  such that the variance is minimized. Concretely, we seek  $P^*$  such that

$$P^* = \underset{P}{\operatorname{argmin}} \mathbb{V}_{I \sim P} \left[ \frac{a(x; \Theta)_I f(x; \Theta)_I}{p_I} \right]. \quad (3.5)$$

We divide by  $p_I$  to ensure that the expectation remains the same regardless of  $P$ . One can easily verify that  $\mathbb{E}_{I \sim P} \left[ \frac{a(x; \Theta)_I f(x; \Theta)_I}{p_I} \right] = \mathbb{E}_{I \sim a(x; \Theta)} [f(x; \Theta)_I]$ . We continue our derivation as follows:

$$\underset{P}{\operatorname{argmin}} \mathbb{V}_{I \sim P} \left[ \frac{a(x; \Theta)_I f(x; \Theta)_I}{p_I} \right] \quad (3.6)$$

$$= \underset{P}{\operatorname{argmin}} \mathbb{E}_{I \sim P} \left[ \left( \frac{a(x; \Theta)_I}{p_I} \right)^2 \|f(x; \Theta)_I\|_2^2 \right] \quad (3.7)$$

$$= \underset{P}{\operatorname{argmin}} \sum_{i=1}^K \frac{a(x; \Theta)_i^2}{p_i} \|f(x; \Theta)_i\|_2^2. \quad (3.8)$$

The minimum of equation 3.8 is

$$p_i^* \propto a(x; \Theta)_i \|f(x; \Theta)_i\|_2, \quad (3.9)$$

which means that sampling according to the attention distribution is optimal when we do not have information about the norm of the features. This can be easily enforced by constraining the features to have the same  $L_2$  norm.

### Gradient derivation

In order to use a neural network as our attention distribution we need to derive the gradient of the loss with respect to the parameters of the attention function  $a(\cdot; \Theta)$  through the sampling of the set of indices  $Q$ . Namely, we need to compute

$$\frac{\partial \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q}{\partial \theta} \quad (3.10)$$

for all  $\theta \in \Theta$  including the ones that affect  $a(\cdot)$ .

By exploiting the Monte Carlo approximation and the multiply by one trick, we show that

$$\frac{\partial}{\partial \theta} \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q \approx \mathbb{E}_{I \sim a(x; \Theta)} \left[ \frac{\frac{\partial}{\partial \theta} [a(x; \Theta)_I f(x; \Theta)_I]}{a(x; \Theta)_I} \right]. \quad (3.11)$$

In equation 3.11, the gradient of each feature is weighed inversely proportionally to the probability of sampling that feature. This result is expected, because the “effect” of rare samples should be increased to account for the low observation frequency (Kahn and Harris, 1951). This allows us to derive the gradients of our *attention sampling* method as follows:

$$\frac{\partial}{\partial \theta} \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q = \frac{1}{N} \sum_{q \in Q} \frac{\frac{\partial}{\partial \theta} [a(x; \Theta)_q f(x; \Theta)_q]}{a(x; \Theta)_q}, \quad (3.12)$$

which requires computing only the rows of  $f(\cdot)$  for the sampled indices in  $Q$ . A detailed derivation of equation 3.11, can be found in appendix B.2.

### Sampling without replacement

In our initial analysis, we assume that  $Q$  is sampled i.i.d. from  $a(x; \Theta)$ . However, this means that it is probable to sample the same element multiple times, especially as the entropy of the distribution decreases during the training of the attention network. To avoid computing a feature multiple times and to make the best use of the available computational budget we propose sampling without replacement.

We model sampling without replacement as follows: Initially, we sample a position  $i_1$  with probability  $p_1(i) \propto a(x; \Theta)_i \forall i$ . Subsequently, we sample the second position  $i_2$ , given the first, with probability  $p_2(i | i_1) \propto a(x; \Theta)_i \forall i \neq i_1$ . Following this reasoning, we can define sampling the  $n$ -th position with probability

$$\forall i \notin \{i_1, i_2, \dots, i_{n-1}\}, \quad p_n(i | i_1, i_2, \dots, i_{n-1}) \propto a(x; \Theta)_i \quad (3.13)$$

Simply averaging the features, as in equation 3.4, would result in a biased estimator. Instead,



we use

$$\mathbb{E}_{I_1, I_2, \dots, I_n} \left[ \sum_{k=1}^{n-1} a(x; \Theta)_{I_k} f(x; \Theta)_{I_k} + \right. \quad (3.14)$$

$$\left. f(x; \Theta)_{I_n} \sum_{t \in \{I_1, I_2, \dots, I_{n-1}\}} a(x; \Theta)_t \right] = \quad (3.15)$$

$$\mathbb{E}_{I_1, I_2, \dots, I_n} \left[ \sum_{i=1}^K a(x; \Theta)_i f(x; \Theta)_i \right] = \quad (3.16)$$

$$\mathbb{E}_{I \sim a(x; \Theta)} [f(x; \Theta)_I]. \quad (3.17)$$

We assume that  $I_1$  to  $I_n$  are sampled from  $p_1(i)$  to  $p_n(i)$  accordingly. Following the reasoning of § 3.3.2, we compute the gradient through the sampling in an efficient and numerically stable way. The complete analysis is provided in appendix B.3.

### 3.3.3 Multi-resolution data

For most implementations of attention in neural networks,  $a(\cdot)$  is a function of the features  $f(\cdot)$  (Ilse et al., 2018). This means that in order to compute the attention distribution we need to compute all the features. However, in order to take advantage of our Monte Carlo Estimation of equation 3.4 and avoid computing all  $K$  features, we use a lower resolution view of the data. This allows us to gain significant speedup from *attention sampling*.

Given an image  $x \in \mathbb{R}^{H \times W \times C}$  where  $H$ ,  $W$ ,  $C$  denote the height, width and channels respectively, and its corresponding view  $V(x, s) \in \mathbb{R}^{h \times w \times C}$  at scale  $s$  we compute the attention as

$$a(V(x, s); \Theta) : \mathbb{R}^{h \times w \times C} \rightarrow \mathbb{R}^{hw}, \quad (3.18)$$

where  $h < H$  and  $w < W$ . We also define a function  $P(x, i)$  that extracts a patch from the full resolution image  $x$  centered around the corresponding  $i$ -th pixel in  $V(x, s)$ .

Based on the above, we derive a model capable of only considering few patches from the full size image  $x$ , as follows:

$$\Psi(x) = g \left( \sum_{i=1}^{hw} a(V(x, s))_i f(P(x, i)) \right) \quad (3.19)$$

$$\approx g \left( \frac{1}{N} \sum_{q \in Q} f(P(x, q)) \right). \quad (3.20)$$

Note that both the attention and feature functions have trainable parameters  $\Theta$  which we omit for clarity. In the formulation of equation 3.20, we do not consider the location of the sampled patches  $P(x, q)$ . This is not an inherent limitation of the model since we can simply pass the location as a parameter in our feature function  $f(\cdot)$ .

### 3.3.4 Implementation details

In this section, we discuss the specifics of our proposed *attention sampling*. Equation  $f(\cdot)$  is implemented by a neural network which we refer to as *feature network*. Similarly  $a(\cdot)$  is another neural network, typically, significantly smaller, referred to as *attention network*. Finally, function  $g(\cdot)$  is a linear classification layer.

In order to control the exploration-exploitation dilemma we introduce an entropy regularizer for the attention distribution. Namely given a loss function  $\mathcal{L}(x, y; \Theta)$  we use

$$\mathcal{L}^l(x, y; \Theta) = \mathcal{L}(x, y; \Theta) - \lambda \mathcal{H}(a(x; \Theta)), \quad (3.21)$$

where  $\mathcal{H}(x)$  denotes the entropy of the distribution  $x$ . This regularizer prevents the attention network from quickly deciding which patches are informative. This results in an exploration of the available patch space during the initial stage of training. The impact of this regularizer is quantitatively evaluated in section B.5 in the appendix.

As already mentioned in § 3.3.2, normalizing the features in terms of the  $L_2$  norm guarantees that the attention distribution produces the minimum variance estimator of the “full model”; thus in all the feature networks we add  $L_2$  normalization as the final layer.

## 3.4 Experimental evaluation

In this section, we analyse experimentally the performance of our *attention sampling* approach on three classification tasks. We showcase the ability of our model to focus on informative parts of the input image which results in significantly reduced computational requirements. We refer to our approach as *ATS* or *ATS-XX* where *XX* denotes the number of sampled patches. Note that we do not consider per-patch annotations for any of the used datasets. The code used for the experiments can be found in <https://github.com/idiap/attention-sampling>.

### 3.4.1 Introduction

#### Baselines

Most related to our method is the patch based approach of [Ilse et al. \(2018\)](#) that implements the attention as a function of the features of each patch. For the rest of the experiments, we refer to this method as *Deep MIL*. For *Deep MIL*, we specify the patches to be extracted from each high resolution image by a regular grid of varying size depending on the dimensions of the input image and the patch size. Note that the architecture of the feature network and the patch size used for *Deep MIL* is always the same as the ones used for our *attention sampling* method.

To showcase that existing CNN architectures are unable to operate on megapixel images, we

also compare our method to traditional CNN models. Typically, the approach for handling high resolution images with deep neural networks is to downsample the input images. Thus; for a fair comparison, we train the CNN baselines using images at various scales. The specifics of each network architecture are described in the corresponding experiment and in detail in section B.8 in the appendix.

Finally, to show that the learned attention distribution is non-trivial, we replace the attention network of our model with a fixed network that predicts the uniform distribution and compare the results. We refer to this baseline as  $U-XX$  where  $XX$  denotes the number of sampled patches.

#### Metrics

Our proposed model allows us to trade off computation with increased performance. Therefore, besides reporting just the achieved test error, we also measure the computational and memory requirements. To this end, we report the per sample wall-clock time for a forward/backward pass and the peak GPU memory allocated for training with a batch size of 1, as reported by the TensorFlow (Abadi et al., 2016) profiler. Note that for *attention sampling*, extracting a patch, reading it from main memory and moving it to the GPU memory is always included in the reported time. Regarding the memory requirements of our baselines, it is important to mention that the maximum used memory depends on the size of the high resolution image, whereas for *attention sampling* it only depends on the number sampled patches and the patch size. For a fair comparison in terms of both memory and computational requirements, with *Deep MIL*, we make sure that the patches are extracted from a grid with a stride at least half the size of the patch. Finally, extensive qualitative results of the learned attention distribution are provided in appendix B.7.

#### 3.4.2 Megapixel MNIST

We evaluate *attention sampling* on an artificial dataset based on the MNIST digit classification task (LeCun et al., 2010). We generate 6000 empty images of size  $1500 \times 1500$  and we place patches of random noise at 50 random locations. The size of each patch is equal to an MNIST digit. In addition, we randomly position 5 digits sampled from the MNIST dataset, 3 belonging to the same class and 2 to a random class. The task is to identify the digit with the most occurrences. We use 5000 images for training and 1000 for testing.

For ATS, the attention network is a three layer convolutional network and the feature network is inspired from LeNet-1 (LeCun et al., 1995). To compute the attention, we downsample the image to  $180 \times 180$  which results in 32,400 patches to sample from. The sampled patches from the high resolution image have size  $50 \times 50$  pixels. For the CNN baseline, we train it on the full size images. Regarding uniform sampling, we note that it does not perform better than random guessing, due to the very large sampling space (32,400 possible patches); thus

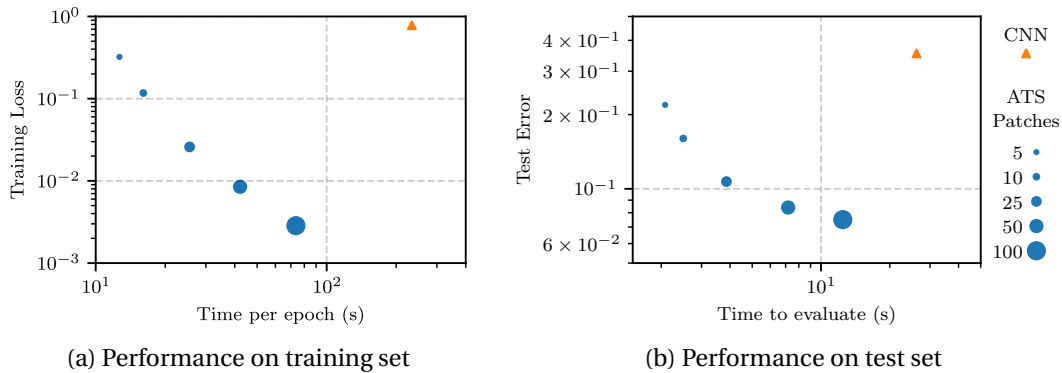


Figure 3.2 – Comparison of *attention sampling* (ATS) with a CNN on Megapixel MNIST. We observe that we can trade optimization accuracy for time by sampling fewer patches.

we omit it from this experiment. Furthermore, we also omit *Deep MIL* because the required memory for a batch size of 1 exceeds the available GPU memory.

### Performance

Initially, we examine the effect of the number of sampled patches on the performance of our method. We sample  $\{5, 10, 25, 50, 100\}$  patches for each image which corresponds to 0.01% to 0.3% of the available sampling space. We train our models 5 independent runs for 500 epochs and the averaged results are depicted in figures 3.2.a and 3.2.b. The figures show the training loss and test error, respectively, with respect to wall clock time both for ATS and the CNN baseline. Even though the CNN has comparably increased capacity, we observe that ATS is order of magnitudes faster and performs better.

As expected, we observe that *attention sampling* directly trades performance for speed, namely sampling fewer patches results in both higher training loss and test error. Although the CNN baseline performs better than random guessing, achieving roughly 40% error, it is still more than an order of magnitude higher than ATS.

### Evolution of the attention distribution

The quantitative results of the previous section demonstrate that *attention sampling* processes high resolution images both faster and more accurately than the CNN baseline. However, another important benefit of using attention is the increased interpretability of the decisions of the network. This can be noticed from Figure 3.3, where we visualize the evolution of the attention distribution as the training progresses. In particular, we select a patch from a random image from the dataset that contains 6 distinct items, 3 pieces of noise and 3 digits, and draw the attention distribution for that patch. We observe that the attention distribution starts as uniform. However, during training, we note that the attention network first learns to distinguish empty space from noise and digits and subsequently even noise from digits. This explains why by only sampling 5 patches we achieve approximately 20% error, even though it

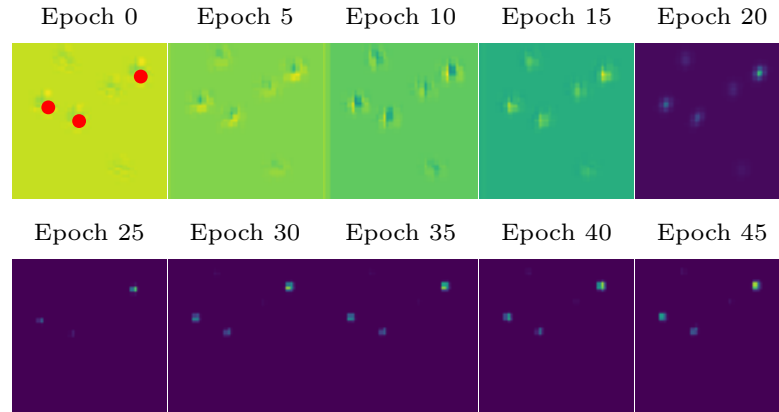


Figure 3.3 – The evolution of the attention distribution on Megapixel MNIST. Yellow means higher attention. At the first image (epoch 0) we mark the position of the digits with the red dots. The attention finds the three digits and focuses on them instead of the noise which can be clearly seen in epochs 10 and 15.

is the minimum required to be able to confidently classify an image.

### 3.4.3 Histopathology images

Method	Scale	Train Loss	Test Error	Time/sample	Memory/sample
U-10	0.2/1	$0.210 \pm 0.031$	$0.156 \pm 0.006$	1.8 ms	19 MB
U-50	0.2/1	$0.075 \pm 0.000$	$0.124 \pm 0.010$	4.6 ms	24 MB
CNN	0.5	$0.002 \pm 0.000$	$0.104 \pm 0.009$	4.8 ms	65 MB
CNN	1	$0.002 \pm 0.000$	$0.092 \pm 0.012$	18.7 ms	250 MB
<i>Deep MIL</i> (Ilse et al., 2018)	1	$0.007 \pm 0.000$	$0.093 \pm 0.004$	48.5 ms	644 MB
ATS-10	0.2/1	$0.083 \pm 0.019$	$0.093 \pm 0.014$	1.8 ms	21 MB
ATS-50	0.2/1	$0.028 \pm 0.002$	$0.093 \pm 0.019$	4.5 ms	26 MB

Table 3.1 – Performance comparison of *attention sampling* (ATS) with a CNN and *Deep MIL* on the *colon cancer* dataset comprised of H&E stained images. The experiments were run 5 times and the average ( $\pm$  a standard error of the mean) is reported. ATS performs equally well to *Deep MIL* and CNN in terms of test error, while being at least **10x** faster.

In this experiment, we evaluate *attention sampling* on the *colon cancer* dataset introduced by Sirinukunwattana et al. (2016) to detect whether epithelial cells exist in a hematoxylin and eosin (H&E) stained image.

This dataset contains 100 images of dimensions  $500 \times 500$ . The images originate both from malignant and normal tissue and contain approximately 22,000 annotated cells. Following the experimental setup of Ilse et al. (2018), we treat the problem as binary classification where the positive images are the ones that contain at least one cell belonging in the epithelial class. While the size of the images in this dataset is less than one megapixel, our method can easily

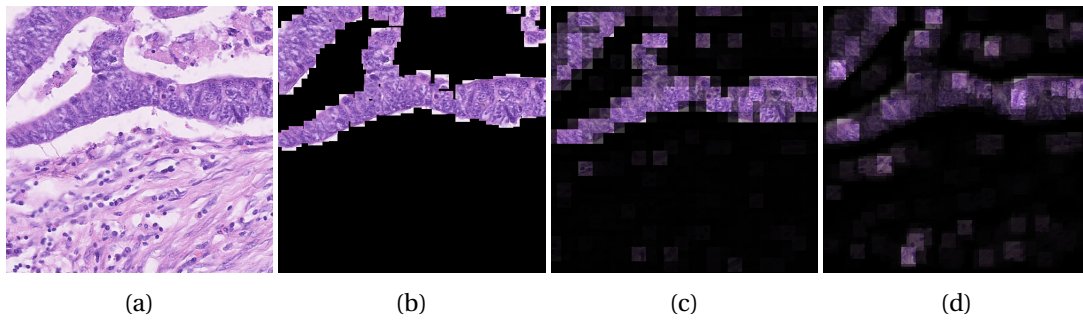


Figure 3.4 – Visualization of the learned attention distributions for *Deep MIL* (c) and our *attention sampling* (d) on an H&E stained image from the *colon cancer* dataset. (a) depicts the raw image and (b) depicts the cells that belong to the epithelial class. Images (c) and (d) are created by multiplying every patch in (a) by the corresponding normalized attention weight. Both methods localize the attention distribution effectively on the informative parts of the image.

scale to datasets with much larger images, as the computational and memory requirements depend only on the size and the number of the patches. However, this does not apply to our baselines, where both the memory and the computational requirements scale linearly with the size of the input image. As a result, this experiment is a best case scenario for our baselines.

For our model, we downsample the images by a factor of 5 and we use the attention network described in § 3.4.2. The feature network of our model is the same as the one proposed by [Ilse et al. \(2018\)](#) with input patches of size  $27 \times 27$ . For *Deep MIL*, we extract 2,500 patches per image at a regular grid. Regarding the CNN baseline, we use a ResNet ([He et al., 2016](#)) architecture. Furthermore, we perform data augmentation by small random adjustments to the brightness and contrast of each image. Following [Ilse et al. \(2018\)](#), we perform 5 independent runs and report the mean and the standard error of the mean.

#### Performance

The results of this experiment are summarized in Table 3.1. We observe that sampling from the uniform distribution 10 and 50 patches is clearly better than random guessing by achieving 15.6% and 12.4% error respectively. This stems from the fact that each positive sample contains hundreds of regions of interest, namely epithelial cells, and we only need one to classify the image. As expected, *attention sampling* learns to focus only on informative parts of the image thus resulting in approximately 35% lower test error and 3 times lower training loss. Furthermore, compared to *Deep MIL* and CNN, ATS-10 performs equally well while being **25x** and **10x** faster respectively. Moreover, the most memory efficient baseline (CNN) needs at least **3x** more memory compared to *attention sampling*, while *Deep MIL* needs **30x** more.

### Attention Distribution

To show that our proposed model indeed learns to focus on informative parts of the image, we visualize the learned attention distribution at the end of training. In particular, we select an image from the test set and we compute the attention distribution both for *Deep MIL* and *attention sampling*. Subsequently, we weigh each corresponding patch with a normalized attention value that is computed as  $w_i = \frac{a_i - \min(a)}{\max(a) - \min(a)}$ . For reference, in Figure 3.4, apart from the two attention distributions, we also visualize the patches that contain an epithelial cell. Both models identify epithelial cells without having access to per-patch annotations. In order to properly classify an image as positive or not, we just need to find a single patch that contains an epithelial cell. Therefore, despite the fact that the learned attention using *attention sampling* matches less well the distribution of the epithelial cells (Figure 3.4b), compared to *Deep MIL*, it is not necessarily worse for the classification task that we are interested in. However, it is less helpful for detecting regions of interest. In addition, we also observe that both attentions have significant overlap even on mistakenly selected patches such as the bottom center of the images.

#### 3.4.4 Speed limit sign detection

Method	Scale	Train Loss	Test Error	Time/sample	Memory/sample
U-5	0.3/1	1.468 ± 0.317	0.531 ± 0.004	7.8 ms	39 MB
U-10	0.3/1	0.851 ± 0.408	0.472 ± 0.008	10.8 ms	78 MB
CNN	0.3	0.003 ± 0.001	0.311 ± 0.049	6.6 ms	86 MB
CNN	0.5	0.002 ± 0.001	0.295 ± 0.039	15.6 ms	239 MB
CNN	1	0.002 ± 0.000	0.247 ± 0.001	64.2 ms	958 MB
<i>Deep MIL</i> (Ilse et al., 2018)	1	0.077 ± 0.089	0.083 ± 0.006	97.2 ms	1,497 MB
ATS-5	0.3/1	0.162 ± 0.124	0.089 ± 0.002	8.5 ms	86 MB
ATS-10	0.3/1	0.082 ± 0.032	0.095 ± 0.008	10.3 ms	118 MB

Table 3.2 – Performance comparison of *attention sampling* (ATS) with a CNN and *Deep MIL* on the *speed limits* dataset. The experiments were run 3 times and the average ( $\pm$  a standard error of the mean) is reported. ATS performs equally well as *Deep MIL* while being at least **10x** faster. Regarding CNN, we note that both *Deep MIL* and *attention sampling* perform significantly better.

In this experiment, we seek to classify images based on whether they contain no speed limit or a limit sign of 50, 70 or 80 kilometers per hour. We use a subset of the Swedish traffic signs dataset (Larsson and Felsberg, 2011), for which we do not use explicit annotations of the signs, just one label for each image. The dataset contains 3,777 images annotated with 20 different traffic sign classes. Each image is 1.3 megapixels, namely  $960 \times 1280$  pixels. As some classes contain less than 20 samples, we limit the classification task to the one described above. The resulting dataset consists of 747 training images and 684 test images, distributed approximately as 100 images for each speed limit sign and 400 for the background class, namely no limit sign.

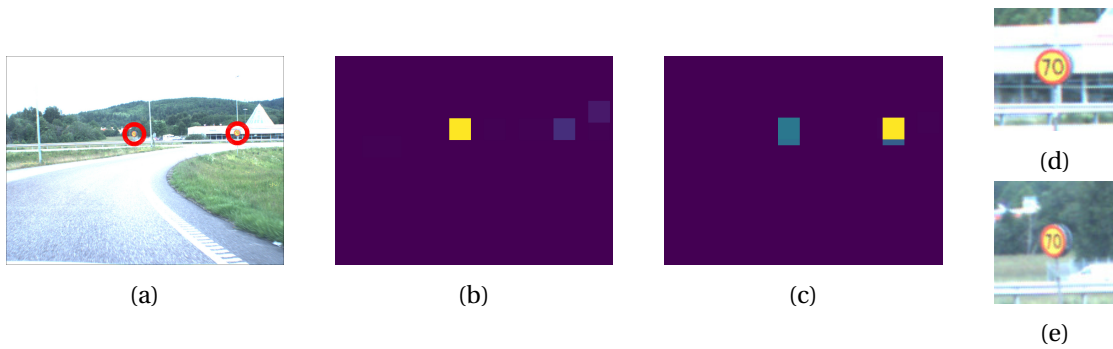


Figure 3.5 – Visualization of the learned attention distributions for *Deep MIL* (b) and our *attention sampling* (c) on an image from the speed limits dataset (a). (d) and (e) depict the marked regions from (a) which are also selected by *attention sampling*. We observe that both of them contain speed limit signs unrecognizable in the low resolution image.

An interesting fact about this dataset is that in order to properly classify all images it is mandatory to process them in high resolution. This is illustrated in Figure 3.1, where from the downsampled image one can deduce the existence of a speed limit sign, without being able to identify the number of kilometers written on it. Objects that are physically far from the moving camera become unrecognizable when downsampling the input image. This property might be critical, for early detection of pedestrians or collision avoidance in a self-driving car scenario.

For *attention sampling*, we downsample the original image by approximately a factor of 3 to  $288 \times 384$ . The attention network is a four layer convolutional network and the feature network of both our model and *Deep MIL* is a simple ResNet. For *Deep MIL*, we extract 192 patches on a grid  $12 \times 16$  of patch size  $100 \times 100$ . For a fair comparison, we evaluate the CNN baseline using images at various resolutions, namely scales 0.3, 0.5 and 1.0.

Again also for this dataset, we perform data augmentation, namely random translations and contrast brightness adjustments. In addition, due to class imbalance, for all evaluated methods, we use a crossentropy loss weighted with the inverse of the prior of each class. We perform 3 independent runs and report the mean and the standard error of the mean.

**Performance**

Table 3.2 compares the proposed model to our baselines on the speed limits dataset. We observe that although the CNN learns the training set perfectly, it fails to generalise. For the downsampled images, this is expected as the limits on the traffic signs are indistinguishable. Similarly, due to the small number of informative patches, uniform sampling fails to correctly classify both the training set and the test set. We observe that *attention sampling* achieves comparable test error to *Deep MIL* by using just 5 patches, instead of 192. This results in significant speedups of more than an order of magnitude. Regarding the required memory,



*attention sampling* needs **17x** less memory compared to *Deep MIL*.

#### Attention Distribution

In this section, we compare qualitatively the learned attention distribution of *Deep MIL* and *attention sampling* on an image from the test set of the speed limits dataset. In Figure 3.5a, we mark the positions of speed limit signs with red circles and visualize the corresponding patches in figures 3.5d and 3.5e. We observe that the attention distribution from our proposed model has high probability for both patches whereas *Deep MIL* locates both but selects only one. Also in this dataset, both models identify regions of interest in the images without being given any explicit per-patch label.

### 3.5 Chapter Conclusions

We have presented a novel algorithm to efficiently process megapixel images in a single CPU or GPU. Our algorithm only processes fractions of the input image, relying on an attention distribution to discover informative regions of the input. We show that we can derive the gradients through the sampling and train our model end-to-end with SGD. Furthermore, we show that sampling with the attention distribution is the optimal approximation, in terms of variance, of the model that processes the whole image.

Our experiments show that our algorithm effectively identifies the important regions in two real world tasks and an artificial dataset without any patch specific annotation. In addition, our model executes an order of magnitude faster and requires an order of magnitude less memory than state of the art patch based methods and traditional CNNs.

The presented line of research opens several directions for future work. We believe that a nested model of *attention sampling* can be used to efficiently learn to discover informative regions and classify up to gigapixel images using a single GPU. In addition, *attention sampling* can be used in resource constrained scenarios to finely control the trade-off between accuracy and spent computation.

In the following chapter, we study the most commonly used attention mechanism in neural networks, the scaled dot product attention used in the transformer architecture ([Vaswani et al., 2017](#)).



# 4 Fast Autoregressive Transformers with Linear Attention

## 4.1 Chapter Introduction

Transformer models were originally introduced by Vaswani et al. (2017) in the context of neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and have demonstrated impressive results on a variety of tasks dealing with natural language (Devlin et al., 2019), audio (Sperber et al., 2018), and images (Parmar et al., 2019). Apart from tasks with ample supervision, transformers are also effective in transferring knowledge to tasks with limited or no supervision when they are pretrained with autoregressive (Radford et al., 2018, 2019) or masked language modeling objectives (Devlin et al., 2019; Yang et al., 2019b; Song et al., 2019; Liu et al., 2020b).

However, these benefits often come with a very high computational and memory cost. The bottleneck is mainly caused by the global receptive field of self-attention, which processes contexts of  $N$  inputs with a quadratic memory and time complexity  $\mathcal{O}(N^2)$ . As a result, in practice transformers are slow to train and their context is *limited*. This disrupts temporal coherence and hinders the capturing of long-term dependencies. Dai et al. (2019a) addressed the latter by attending to memories from previous contexts albeit at the expense of computational efficiency.

Lately, researchers shifted their attention to approaches that increase the context length without sacrificing efficiency. Towards this end, Child et al. (2019) introduced sparse factorizations of the attention matrix to reduce the self-attention complexity to  $\mathcal{O}(N\sqrt{N})$ . Kitaev et al. (2020) further reduced the complexity to  $\mathcal{O}(N\log N)$  using locality-sensitive hashing. This made scaling to long sequences possible. Even though the aforementioned models can be efficiently trained on large sequences, they do not speed-up autoregressive inference.

In this chapter, we introduce the *linear transformer* model that significantly reduces the memory footprint and scales linearly with respect to the context length. We achieve this by using a kernel-based formulation of self-attention and the associative property of matrix products to calculate the self-attention weights (§ 4.3.2). Using our linear formulation, we also

express causal masking with linear complexity and constant memory (§ 4.3.3). This reveals the relation between transformers and RNNs, which enables us to perform autoregressive inference orders of magnitude faster (§ 4.3.4).

Our evaluation on image generation and automatic speech recognition demonstrates that *linear transformer* can reach the performance levels of transformer, while being up to three orders of magnitude faster during inference.

## 4.2 Related Work

In this section, we provide an overview of the most relevant works that seek to address the large memory and computational requirements of transformers. Furthermore, we discuss methods that theoretically analyze the core component of the transformer model, namely self-attention. Finally, we present another line of work that seeks to alleviate the softmax bottleneck in the attention computation.

### 4.2.1 Efficient Transformers

Existing works seek to improve memory efficiency in transformers through weight pruning (Michel et al., 2019), weight factorization (Lan et al., 2020), weight quantization (Zafir et al., 2019) or knowledge distillation. Clark et al. (2020) proposed a new pretraining objective called replaced token detection that is more sample efficient and reduces the overall computation. Lample et al. (2019) used product-key attention to increase the capacity of any layer with negligible computational overhead.

Reducing the memory or computational requirements with these methods leads to training or inference time speedups, but, fundamentally, the time complexity is still quadratic with respect to the sequence length which hinders scaling to long sequences. In contrast, we show that our method reduces both memory and time complexity of transformers both theoretically (§ 4.3.2) and empirically (§ 4.4.1).

Another line of research aims at increasing the “context” of self-attention in transformers. Context refers to the maximum part of the sequence that is used for computing self-attention. Dai et al. (2019a) introduced Transformer-XL which achieves state-of-the-art in language modeling by learning dependencies beyond a fixed length context without disrupting the temporal coherence. However, maintaining previous contexts in memory introduces significant additional computational cost. In contrast, Sukhbaatar et al. (2019a) extended the context length significantly by learning the optimal attention span per attention head, while maintaining control over the memory footprint and computation time. Note that both approaches have the same asymptotic complexity as the vanilla model. In contrast, we improve the asymptotic complexity of the self-attention, which allows us to use significantly larger context.

More related to our model are the works of Child et al. (2019) and Kitaev et al. (2020). The

former (Child et al., 2019) introduced sparse factorizations of the attention matrix reducing the overall complexity from quadratic to  $\mathcal{O}(N\sqrt{N})$  for generative modeling of long sequences. More recently, Kitaev et al. (2020) proposed Reformer. This method further reduces complexity to  $\mathcal{O}(N\log N)$  by using locality-sensitive hashing (LSH) to perform fewer dot products. Note that in order to be able to use LSH, Reformer constrains the keys, for the attention, to be identical to the queries. As a result this method cannot be used for decoding tasks where the keys need to be different from the queries. In comparison, *linear transformers* impose no constraints on the queries and keys and scale linearly with respect to the sequence length. Furthermore, they can be used to perform inference in autoregressive tasks three orders of magnitude faster, achieving comparable performance in terms of validation perplexity.

#### 4.2.2 Understanding Self-Attention

There have been few efforts to better understand self-attention from a theoretical perspective. Tsai et al. (2019) proposed a kernel-based formulation of attention in transformers which considers attention as applying a kernel smoother over the inputs with the kernel scores being the similarity between inputs. This formulation provides a better way to understand attention components and integrate the positional embedding. In contrast, we use the kernel formulation to speed up the calculation of self-attention and lower its computational complexity. Also, we observe that if a kernel with positive similarity scores is applied on the queries and keys, linear attention converges normally.

More recently, Cordonnier et al. (2020) provided theoretical proofs and empirical evidence that a multi-head self-attention with sufficient number of heads can express any convolutional layer. Here, we instead show that a self-attention layer trained with an autoregressive objective can be seen as a recurrent neural network and this observation can be used to significantly speed up inference time of autoregressive transformer models.

#### 4.2.3 Linearized softmax

For many years, softmax has been the bottleneck for training classification models with a large number of categories (Goodman, 2001; Morin and Bengio, 2005; Mnih and Hinton, 2009). Recent works (Blanc and Rendle, 2017; Rawat et al., 2019), have approximated softmax with a linear dot product of feature maps to speed up the training through sampling. Inspired from these works, we linearize the softmax attention in transformers. Concurrently with this work, Shen et al. (2020) explored the use of linearized attention for the task of object detection in images. In comparison, we do not only linearize the attention computation, but also develop an autoregressive transformer model with linear complexity and constant memory for both inference and training. Moreover, we show that through the lens of kernels, every transformer can be seen as a recurrent neural network.

### 4.3 Linear Transformers

In this section, we formalize our proposed *linear transformer*. We present that changing the attention from the traditional *softmax* attention to a feature map based dot product attention results in better time and memory complexity as well as a causal model that can perform sequence generation in linear time, similar to a recurrent neural network.

Initially, in § 4.3.1, we introduce a formulation for the transformer architecture introduced in (Vaswani et al., 2017). Subsequently, in § 4.3.2 and § 4.3.3 we present our proposed *linear transformer* and finally, in § 4.3.4 we rewrite the transformer as a recurrent neural network.

#### 4.3.1 Transformers

Let  $x \in \mathbb{R}^{N \times F}$  denote a sequence of  $N$  feature vectors of dimensions  $F$ . A transformer is a function  $T : \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^{N \times F}$  defined by the composition of  $L$  transformer layers  $T_1(\cdot), \dots, T_L(\cdot)$  as follows,

$$T_l(x) = f_l(A_l(x) + x). \quad (4.1)$$

The function  $f_l(\cdot)$  transforms each feature independently of the others and is usually implemented with a small two-layer feedforward network.  $A_l(\cdot)$  is the self attention function and is the only part of the transformer that acts across sequences.

The self attention function  $A_l(\cdot)$  computes, for every position, a weighted average of the feature representations of all other positions with a weight proportional to a similarity score between the representations. Formally, the input sequence  $x$  is projected by three matrices  $W_Q \in \mathbb{R}^{F \times D}$ ,  $W_K \in \mathbb{R}^{F \times D}$  and  $W_V \in \mathbb{R}^{F \times M}$  to corresponding representations  $Q$ ,  $K$  and  $V$ . The output for all positions,  $A_l(x) = V'$ , is computed as follows,

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \\ A_l(x) &= V' = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V. \end{aligned} \quad (4.2)$$

Note that in the previous equation, the softmax function is applied rowwise to  $QK^T$ . Following common terminology, the  $Q$ ,  $K$  and  $V$  are referred to as the “queries”, “keys” and “values” respectively.

Equation 4.2 implements a specific form of self-attention called softmax attention where the similarity score is the exponential of the dot product between a query and a key. Given that subscripting a matrix with  $i$  returns the  $i$ -th row as a vector, we can write a generalized

attention equation for any similarity function as follows,

$$V'_i = \frac{\sum_{j=1}^N \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^N \text{sim}(Q_i, K_j)}. \quad (4.3)$$

Equation 4.3 is equivalent to equation 4.2 if we substitute the similarity function with  $\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$ .

### 4.3.2 Linearized Attention

The definition of attention in equation 4.2 is generic and can be used to define several other attention implementations such as polynomial attention or RBF kernel attention (Tsai et al., 2019). Note that the only constraint we need to impose to  $\text{sim}(\cdot)$ , in order for equation 4.3 to define an attention function, is to be non-negative. This includes all kernels  $k(x, y) : \mathbb{R}^{2 \times F} \rightarrow \mathbb{R}_+$ .

Given such a kernel with a feature representation  $\phi(x)$  we can rewrite equation 4.2 as follows,

$$V'_i = \frac{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j)}, \quad (4.4)$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j)}. \quad (4.5)$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$(\phi(Q) \phi(K)^T) V = \phi(Q) (\phi(K)^T V). \quad (4.6)$$

Note that the feature map  $\phi(\cdot)$  is applied rowwise to the matrices  $Q$  and  $K$ .

From equation 4.2, it is evident that the computational cost of softmax attention scales with  $\mathcal{O}(N^2)$ , where  $N$  represents the sequence length. The same is true for the memory requirements because the full attention matrix must be stored to compute the gradients with respect to the queries, keys and values. In contrast, our proposed *linear transformer* from equation 4.5 has time and memory complexity  $\mathcal{O}(N)$  because we can compute  $\sum_{j=1}^N \phi(K_j) V_j^T$  and  $\sum_{j=1}^N \phi(K_j)$  once and reuse them for every query.

### Feature Maps and Computational Cost

For softmax attention, the total cost in terms of multiplications and additions scales as  $\mathcal{O}(N^2 \max(D, M))$ , where  $D$  is the dimensionality of the queries and keys and  $M$  is the dimensionality of the values. On the contrary, for linear attention, we first compute the feature maps of dimensionality  $C$ . Subsequently, computing the new values requires  $\mathcal{O}(NCM)$  additions and multiplications.

The previous analysis does not take into account the choice of kernel and feature function. Note that the feature function that corresponds to the exponential kernel is infinite dimensional, which makes the linearization of exact softmax attention infeasible. On the other hand, the polynomial kernel, for example, has an exact finite dimensional feature map and has been shown to work equally well with the exponential or RBF kernel (Tsai et al., 2019). The computational cost for a linearized polynomial transformer of degree 2 is  $\mathcal{O}(ND^2M)$ . This makes the computational complexity favorable when  $N > D^2$ . Note that this is true in practice since we want to be able to process sequences with tens of thousands of elements.

For our experiments, that deal with smaller sequences, we employ a feature map that results in a positive similarity function as defined below,

$$\phi(x) = \text{elu}(x) + 1, \quad (4.7)$$

where  $\text{elu}(\cdot)$  denotes the exponential linear unit (Clevert et al., 2015) activation function. We prefer  $\text{elu}(\cdot)$  over  $\text{relu}(\cdot)$  to avoid setting the gradients to 0 when  $x$  is negative. This feature map results in an attention function that requires  $\mathcal{O}(NDM)$  multiplications and additions. In our experimental section, we show that the feature map of equation 4.7 performs on par to the full transformer, while significantly reducing the computational and memory requirements.

#### 4.3.3 Causal Masking

The transformer architecture can be used to efficiently train autoregressive models by masking the attention computation such that the  $i$ -th position can only be influenced by a position  $j$  if and only if  $j \leq i$ , namely a position cannot be influenced by the subsequent positions. Formally, this causal masking changes equation 4.3 as follows,

$$V'_i = \frac{\sum_{j=1}^i \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^i \text{sim}(Q_i, K_j)}. \quad (4.8)$$

Following the reasoning of § 4.3.2, we linearize the masked attention as described below,

$$V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)}. \quad (4.9)$$



By introducing  $S_i$  and  $Z_i$  as follows,

$$S_i = \sum_{j=1}^i \phi(K_j) V_j^T, \quad (4.10)$$

$$Z_i = \sum_{j=1}^i \phi(K_j), \quad (4.11)$$

we can simplify equation 4.9 to

$$V'_i = \frac{\phi(Q_i)^T S_i}{\phi(Q_i)^T Z_i}. \quad (4.12)$$

Note that,  $S_i$  and  $Z_i$  can be computed from  $S_{i-1}$  and  $Z_{i-1}$  in constant time hence making the computational complexity of linear transformers with causal masking linear with respect to the sequence length.

### Gradient Computation

A naive implementation of equation 4.12, in any deep learning framework, requires storing all intermediate values  $S_i$  in order to compute the gradients. This increases the memory consumption by  $\max(D, M)$  times; thus hindering the applicability of causal linear attention to longer sequences or deeper models. To address this, we derive the gradients of the numerator in equation 4.9 as cumulative sums. This allows us to compute both the forward and backward pass of causal linear attention in **linear time** and **constant memory**. A detailed derivation is provided in the supplementary material.

Given the numerator  $\bar{V}_i$  and the gradient of a scalar loss function with respect to the numerator  $\nabla_{\bar{V}_i} \mathcal{L}$ , we derive  $\nabla_{\phi(Q_i)} \mathcal{L}$ ,  $\nabla_{\phi(K_i)} \mathcal{L}$  and  $\nabla_{V_i} \mathcal{L}$  as follows,

$$\nabla_{\phi(Q_i)} \mathcal{L} = \nabla_{\bar{V}_i} \mathcal{L} \left( \sum_{j=1}^i \phi(K_j) V_j^T \right)^T, \quad (4.13)$$

$$\nabla_{\phi(K_i)} \mathcal{L} = \left( \sum_{j=i}^N \phi(Q_j) \left( \nabla_{\bar{V}_j} \mathcal{L} \right)^T \right) V_i, \quad (4.14)$$

$$\nabla_{V_i} \mathcal{L} = \left( \sum_{j=i}^N \phi(Q_j) \left( \nabla_{\bar{V}_j} \mathcal{L} \right)^T \right)^T \phi(K_i). \quad (4.15)$$

The cumulative sum terms in equations 4.9, 4.13-4.15 are computed in linear time and require constant memory with respect to the sequence length. This results in an algorithm with computational complexity  $\mathcal{O}(NCM)$  and memory  $\mathcal{O}(N \max(C, M))$  for a given feature map of  $C$  dimensions. A pseudocode implementation of the forward and backward pass of the numerator is given in algorithms 2 and 3 respectively.

## Chapter 4. Fast Autoregressive Transformers with Linear Attention

---

### Algorithm 2 Forward pass for linear transformers with causal masking

---

```
1: Inputs  $\phi(Q), \phi(K), V$ 
2:  $\bar{V} \leftarrow 0$ 
3:  $S \leftarrow 0$ 
4: for  $i = 1, \dots, N$  do
5:    $S \leftarrow S + \phi(K_i) V_i^T$ 
6:    $\bar{V} \leftarrow \phi(Q_i) S$ 
7: end for
8: Return  $\bar{V}$ 
```

equation 4.10

---

### Algorithm 3 Backward pass for linear transformers with causal masking

---

```
1: Inputs  $\phi(Q), \phi(K), V$  and  $G$ 
2:  $G$  is the gradient of the loss with respect to the output of algorithm 2
3:  $S \leftarrow 0$ 
4:  $\nabla_{\phi(Q)} \mathcal{L} \leftarrow 0$ 
5: for  $i = 1, \dots, N$  do
6:    $S \leftarrow S + \phi(K_i) V_i^T$ 
7:    $\nabla_{\phi(Q_i)} \mathcal{L} \leftarrow G_i S^T$ 
8: end for
9:  $S \leftarrow 0$ 
10:  $\nabla_{\phi(K)} \mathcal{L} \leftarrow 0$ 
11:  $\nabla_V \mathcal{L} \leftarrow 0$ 
12: for  $i = N, \dots, 1$  do
13:    $S \leftarrow S + \phi(Q_i) G_i^T$ 
14:    $\nabla_{V_i} \mathcal{L} \leftarrow S^T \phi(K_i)$ 
15:    $\nabla_{\phi(K_i)} \mathcal{L} \leftarrow S V_i$ 
16: end for
17: Return  $\nabla_{\phi(Q)} \mathcal{L}, \nabla_{\phi(K)} \mathcal{L}, \nabla_V \mathcal{L}$ 
```

equation 4.13

equation 4.15  
equation 4.14

---

### Training and Inference

When training an autoregressive transformer model the full ground truth sequence is available. This makes layerwise parallelism possible both for  $f_i(\cdot)$  of equation 4.1 and the attention computation. As a result, transformers are more efficient to train than recurrent neural networks. On the other hand, during inference the output for timestep  $i$  is the input for timestep  $i + 1$ . This makes autoregressive models impossible to parallelize. Moreover, the cost per timestep for transformers is not constant; instead, it scales with the square of the current sequence length because attention must be computed for all previous timesteps.

Our proposed *linear transformer* model *combines the best of both worlds*. When it comes to training, the computations can be parallelized and take full advantage of GPUs or other accelerators. When it comes to inference, the cost per time and memory for one prediction is constant for our model. This means we can simply store the  $\phi(K_j) V_j^T$  matrix as an internal state and update it at every time step like a recurrent neural network. This results in inference **thousands of times faster** than other transformer models.

#### 4.3.4 Transformers are RNNs

In literature, transformer models are considered to be a fundamentally different approach to recurrent neural networks. However, from the causal masking formulation in § 4.3.3 and the discussion in the previous section, it becomes evident that any transformer layer with causal masking can be written as a model that, given an input, modifies an internal state and then predicts an output, namely a Recurrent Neural Network (RNN). Note that, in contrast to Universal Transformers (Dehghani et al., 2018), we consider the recurrence with respect to time and not depth.

In the following equations, we formalize the transformer layer of equation 4.1 as a recurrent neural network. The resulting RNN has two hidden states, namely the attention memory  $s$  and the normalizer memory  $z$ . We use subscripts to denote the timestep in the recurrence.

$$s_0 = 0, \quad (4.16)$$

$$z_0 = 0, \quad (4.17)$$

$$s_i = s_{i-1} + \phi(x_i W_K) (x_i W_V)^T, \quad (4.18)$$

$$z_i = z_{i-1} + \phi(x_i W_K), \quad (4.19)$$

$$y_i = f_i \left( \frac{\phi(x_i W_Q)^T s_i}{\phi(x_i W_Q)^T z_i} + x_i \right). \quad (4.20)$$

In the above equations,  $x_i$  denotes the  $i$ -th input and  $y_i$  the  $i$ -th output for a specific transformer layer. Note that our formulation does not impose any constraint on the feature function and it can be used for representing *any transformer* model, in theory even the ones using softmax attention. This formulation is a first step towards better understanding the relationship

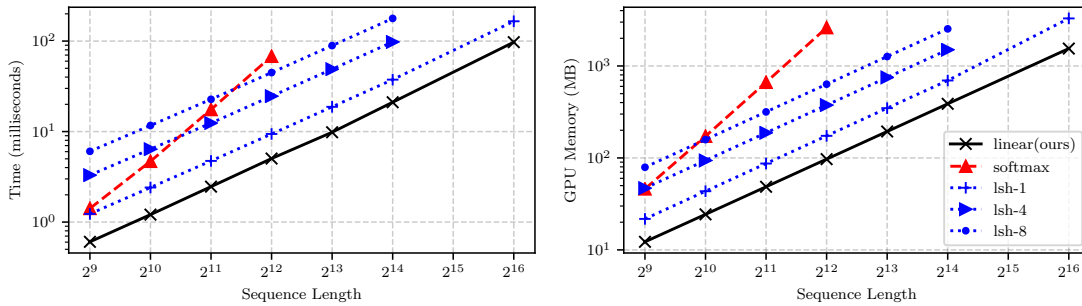


Figure 4.1 – Comparison of the computational requirements for a forward/backward pass for Reformer (lsh-X), softmax attention and linear attention. Linear and Reformer models scale linearly with the sequence length unlike softmax which scales with the square of the sequence length both in memory and time. Full details of the experiment can be found in § 4.4.1.

between transformers and popular recurrent networks (Hochreiter and Schmidhuber, 1997) and the processes used for storing and retrieving information.

## 4.4 Experiments

In this section, we analyze experimentally the performance of the proposed *linear transformer*. Initially, in § 4.4.1, we evaluate the linearized attention in terms of computational cost, memory consumption and convergence on synthetic data. To further showcase the effectiveness of *linear transformers*, we evaluate our model on two real-world applications, image generation in § 4.4.2 and automatic speech recognition in § 4.4.3. We show that our model achieves competitive performance with respect to the state-of-the-art transformer architectures, while requiring significantly less GPU memory and computation.

Throughout our experiments, we compare our model with two baselines, the full transformer with softmax attention and the Reformer (Kitaev et al., 2020), the latter being a state-of-the-art accelerated transformer architecture. For the Reformer, we use a PyTorch reimplementation of the published code and for the full transformer we use the default PyTorch implementation. Note that for Reformer, we do not use the reversible layers, however, this does not affect the results as we only measure the memory consumption with respect to the self attention layer. In all experiments, we use **softmax** (Vaswani et al., 2017) to refer to the standard transformer architecture, **linear** for our proposed *linear transformers* and **lsh-X** for Reformer (Kitaev et al., 2020), where X denotes the hashing rounds.

For training the *linear transformers*, we use the feature map of equation 4.7. Our PyTorch (Paszke et al., 2019) code with documentation and examples can be found at <https://linear-transformers.com/>. The constant memory gradient computation of equations 4.13-4.15 is implemented in approximately 200 lines of CUDA code.

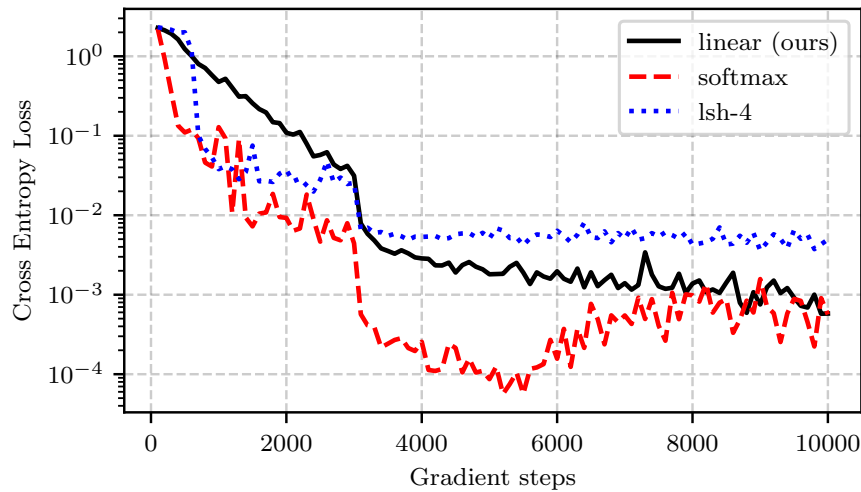


Figure 4.2 – Convergence comparison of *softmax*, *linear* and *reformer* attention on a sequence duplication task. *linear* converges stably and reaches the same final performance as softmax. The details of the experiment are in § 4.4.1.

#### 4.4.1 Synthetic Tasks

##### Convergence Analysis

To examine the convergence properties of *linear transformers* we train on an artificial copy task with causal masking. Namely, the transformers have to copy a series of symbols similar to the sequence duplication task of Kitaev et al. (2020). We use a sequence of maximum length 128 with 10 different symbols separated by a dedicated separator symbol. For all three methods, we train a 4 layer transformer with 8 attention heads using a batch size of 64 and the RAdam optimizer (Liu et al., 2019a) with a learning rate of  $10^{-3}$  which is reduced to  $10^{-4}$  after 3000 updates. Figure 4.2 depicts the loss with respect to the number of gradient steps. We observe that *linear* converges smoothly and reaches a lower loss than *lsh* due to the lack of noise introduced by hashing. In particular, it reaches the same loss as *softmax*.

##### Memory and Computational Requirements

In this subsection, we compare transformers with respect to their computational and memory requirements. We compute the attention and the gradients for a synthetic input with varying sequence lengths  $N \in \{2^9, 2^{10}, \dots, 2^{16}\}$  and measure the peak allocated GPU memory and required time for each variation of transformer. We scale the batch size inversely with the sequence length and report the time and memory per sample in the batch.

Every method is evaluated up to the maximum sequence length that fits the GPU memory. For this benchmark we use an NVidia GTX 1080 Ti with 11GB of memory. This results in a maximum sequence length of 4,096 elements for *softmax* and 16,384 for *lsh-4* and *lsh-8*. As

Method	Bits/dim	Images/sec
Softmax	0.621	0.45 (1×)
LSH-1	0.745	0.68 (1.5×)
LSH-4	0.676	0.27 (0.6×)
Linear (ours)	0.644	<b>142.8 (317×)</b>

Table 4.1 – Comparison of autoregressive image generation of MNIST images. Our linear transformers achieve almost the same bits/dim as the full softmax attention but more than 300 times higher throughput in image generation. The full details of the experiment are in § 4.4.2.

expected, softmax scales quadratically with respect to the sequence length. Our method is faster and requires less memory than the baselines for every configuration, as seen in figure 4.1. We observe that both Reformer and linear attention scale linearly with the sequence length. Note that although the asymptotic complexity for Reformer is  $\mathcal{O}(N \log N)$ ,  $\log N$  is small enough and does not affect the computation time.

#### 4.4.2 Image Generation

Transformers have shown great results on the task of conditional or unconditional autoregressive generation (Radford et al., 2019; Child et al., 2019), however, sampling from transformers is slow due to the task being inherently sequential and the memory scaling with the square of the sequence length. In this section, we train causally masked transformers to predict images pixel by pixel. Our achieved performance in terms of bits per dimension is on par with *softmax* attention while being able to generate images **more than 1,000 times faster** and with **constant memory per image** from the first to the last pixel. We refer the reader to our supplementary for comparisons in terms of training evolution, quality of generated images and time to generate a single image. In addition, we also compare with a faster softmax transformer that caches the keys and values during inference, in contrast to the PyTorch implementation.

#### MNIST

First, we evaluate our model on image generation with autoregressive transformers on the widely used MNIST dataset (LeCun et al., 2010). The architecture for this experiment comprises 8 attention layers with 8 attention heads each. We set the embedding size to 256 which is 32 dimensions per head. Our feed forward dimensions are 4 times larger than our embedding size. We model the output with a mixture of 10 logistics as introduced by Salimans et al. (2017). We use the RAdam optimizer with a learning rate of  $10^{-4}$  and train all models for 250 epochs. For the reformer baseline, we use 1 and 4 hashing rounds. Furthermore, as suggested in Kitaev et al. (2020), we use 64 buckets and chunks with approximately 32 elements. In particular, we divide the 783 long input sequence to 27 chunks of 29 elements each. Since the sequence

Method	Bits/dim	Images/sec
Softmax	3.47	0.004 (1×)
LSH-1	3.39	0.015 (3.75×)
LSH-4	3.51	0.005 (1.25×)
Linear (ours)	3.40	<b>17.85 (4,462×)</b>

Table 4.2 – We train autoregressive transformers for 1 week on a single GPU to generate CIFAR-10 images. Our linear transformer completes 3 times more epochs than softmax, which results in better perplexity. Our model generates images 4,000× faster than the baselines. The full details of the experiment are in § 4.4.2.

length is relatively small, namely only 784 pixels, to remove differences due to different batch sizes we use a batch size of 10 for all methods.

Table 4.1 summarizes the results. We observe that linear transformers achieve almost the same performance, in terms of final perplexity, as softmax transformers while being able to generate images more than 300 times faster. This is achieved due to the low memory requirements of our model, which is able to simultaneously generate 10,000 MNIST images with a single GPU. In particular, the memory is constant with respect to the sequence length because the only thing that needs to be stored between pixels are the  $s_i$  and  $z_i$  values as described in equations 4.18 and 4.19. On the other hand, both softmax and Reformer require memory that increases with the length of the sequence.

Image completions and unconditional samples from our MNIST model can be seen in figure 4.3. We observe that our linear transformer generates very convincing samples with sharp boundaries and no noise. In the case of image completion, we also observe that the transformer learns to use the same stroke style and width as the original image effectively attending over long temporal distances. Note that as the achieved perplexity is more or less the same for all models, we do not observe qualitative differences between the generated samples from different models.

### CIFAR-10

The benefits of our linear formulation increase as the sequence length increases. To showcase that, we train 16 layer transformers to generate CIFAR-10 images (Krizhevsky, 2009). For each layer we use the same configuration as in the previous experiment. For Reformer, we use again 64 buckets and 83 chunks of 37 elements, which is approximately 32, as suggested in the paper. Since the sequence length is almost 4 times larger than for the previous experiment, the full transformer can only be used with a batch size of 1 in the largest GPU that is available to us, namely an NVidia P40 with 24GB of memory. For both the linear transformer and reformer, we use a batch size of 4. All models are trained for 7 days. We report results in terms of bits per dimension and image generation throughput in table 4.2. Note that although the main point

of this experiment is not the final perplexity, it is evident that as the sequence length grows, the fast transformer models become increasingly more efficient per GPU hour, achieving better scores than their slower counterparts.

As the memory and time to generate a single pixel scales quadratically with the number of pixels for both Reformer and softmax attention, the increase in throughput for our linear transformer is even more pronounced. In particular, **for every image generated** by the softmax transformer, **our method can generate 4,460 images**. Image completions and unconditional samples from our model can be seen in figure 4.4. We observe that our model generates images with spatial consistency and can complete images convincingly without significantly hindering the recognition of the image category. For instance, in figure 4.4b, all images have successfully completed the dog’s nose (first row) or the windshield of the truck (last row).

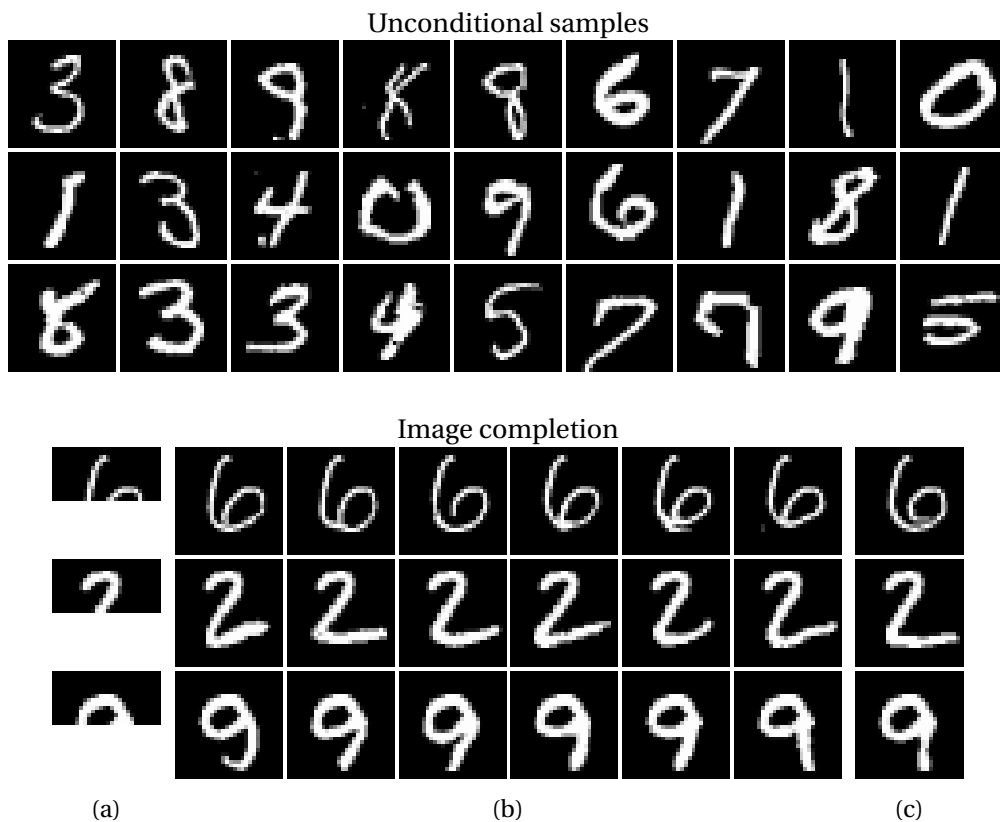


Figure 4.3 – Unconditional samples and image completions generated by our method for MNIST. (a) depicts the occluded original images, (b) the completions and (c) the original. Our model achieves comparable bits/dimension to softmax, while having more than **300 times** higher throughput, generating **142 images/second**. For details see § 4.4.2.

### 4.4.3 Automatic Speech Recognition

To show that our method can also be used for non-autoregressive tasks, we evaluate the performance of linear transformers in end-to-end automatic speech recognition using Con-



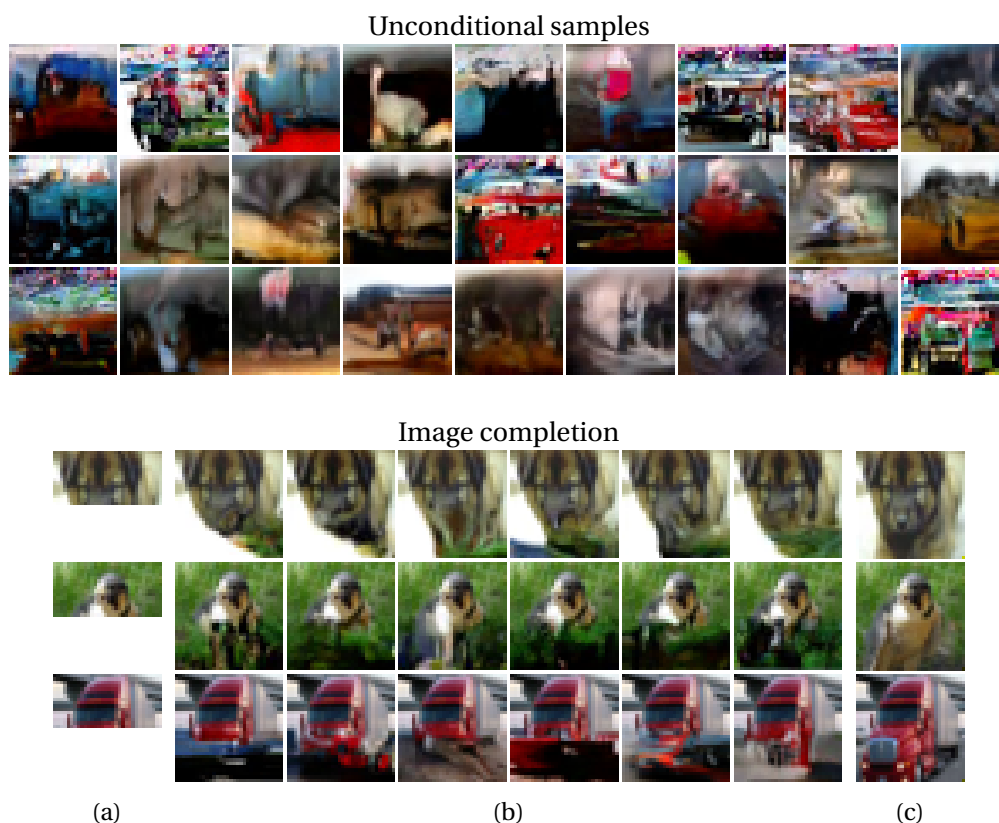


Figure 4.4 – Unconditional samples and image completions generated by our method for CIFAR-10. (a) depicts the occluded original images, (b) the completions and (c) the original. As the sequence length grows linear transformers become more efficient compared to softmax attention. Our model achieves more than **4,000 times** higher throughput and generates **17.85 images/second**. For details see § 4.4.2.

nectionist Temporal Classification (CTC) loss (Graves et al., 2006a). In this setup, we predict a distribution over phonemes for each input frame in a non autoregressive fashion. We use the 80 hour WSJ dataset (Paul and Baker, 1992a) with 40-dimensional mel-scale filterbanks without temporal differences as features. The dataset contains sequences with 800 frames on average and a maximum sequence length of 2,400 frames. For this task, we also compare with a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) with 3 layers of hidden size 320. We use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of  $10^{-3}$  which is reduced when the validation error stops decreasing. For the transformer models, we use 9 layers with 6 heads with the same embedding dimensions as for the image experiments. As an optimizer, we use RAdam with an initial learning rate of  $10^{-4}$  that is divided by 2 when the validation error stops decreasing.

All models are evaluated in terms of phoneme error rate (PER) and training time per epoch. We observe that linear outperforms the recurrent network baseline and Reformer both in terms of performance and speed by a large margin, as seen in table 4.3. Note that the softmax

Method	Validation PER	Time/epoch (s)
Bi-LSTM	10.94	1047
Softmax	5.12	2711
LSH-4	9.33	2250
Linear (ours)	8.08	<b>824</b>

Table 4.3 – Performance comparison in automatic speech recognition on the WSJ dataset. The results are given in the form of phoneme error rate (PER) and training time per epoch. Our model outperforms the LSTM and Reformer while being faster to train and evaluate. Details of the experiment can be found in § 4.4.3.

transformer, achieves lower phone error rate in comparison to all baselines, but is significantly slower. In particular, *linear transformer* is more than  $3\times$  faster per epoch. We provide training evolution plots in the supplementary.

### 4.5 Chapter Conclusions

In this work, we presented *linear transformer*, a model that significantly reduces the memory and computational cost of the original transformers. In particular, by exploiting the associativity property of matrix products we are able to compute the self-attention in time and memory that scales linearly with respect to the sequence length. We show that our model can be used with causal masking and still retain its linear asymptotic complexities. Finally, we express the transformer model as a recurrent neural network, which allows us to perform inference on autoregressive tasks thousands of time faster.

This property opens a multitude of directions for future research regarding the storage and retrieval of information in both RNNs and transformers. Another line of research to be explored is related to the choice of feature map for linear attention. For instance, approximating the RBF kernel with random Fourier features could allow us to use models pretrained with softmax attention.

In the next chapter we present a different approach for accelerating the attention computation but also more importantly for approximating pretrained transformer models without the need for additional finetuning.

# 5 Fast Transformers with Clustered Attention

## 5.1 Chapter Introduction

Sequence modelling is a fundamental task of machine learning, integral in a variety of applications such as neural machine translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015), summarization (Maybury, 1999), automatic speech recognition (Dong et al., 2018) and synthesis (Oord et al., 2016) etc. Transformers (Vaswani et al., 2017) have been proven a powerful tool significantly advancing the state-of-the-art for the majority of the aforementioned tasks. In particular, transformers employ self-attention that allows them to handle long sequences without the vanishing-gradient problem inherent in RNNs (Hochreiter et al., 2001; Arjovsky et al., 2016).

Nonetheless, despite their impressive performance, the use of self-attention comes with computational and memory requirements that scale quadratic to the sequence length, limiting their applicability to long sequences. The quadratic complexity becomes apparent if we consider the core mechanism of self-attention, namely splitting the input sequence into queries and keys and then each query attending to all keys. To this end, recently, there has been an increasing interest for developing methods that address this limitation (Dai et al., 2019b; Sukhbaatar et al., 2019b; Child et al., 2019; Kitaev et al., 2020).

These methods can be broadly categorized into two distinct lines of work, those that focus on improving the asymptotic complexity of the self-attention computation (Child et al., 2019; Lee et al., 2019; Kitaev et al., 2020; Roy et al., 2020) and those that aim at developing techniques that make transformers applicable to longer sequences without addressing the quadratic complexity of self-attention (Dai et al., 2019b; Sukhbaatar et al., 2019b). The former limits the amount of keys that each query attends to, thus reducing the asymptotic complexity. The latter increases the length of the sequence that a transformer can attend to without altering the underlying complexity of the self-attention mechanism.

In this work, we propose *clustered attention* which is a fast approximation of self-attention. Clustered attention makes use of similarities between queries and groups them in order to

reduce the computational cost. In particular, we perform fast clustering using locality-sensitive hashing and K-Means and only compute the attention once per cluster. This results in linear complexity for a fixed number of clusters (§ 5.3.2). In addition, we showcase that we can further improve the quality of our approximation by separately considering the keys with the highest attention per cluster (§ 5.3.3). Finally, we provide theoretical bounds of our approximation quality with respect to the full attention (§ 5.3.2, § 5.3.3) and show that our model can be applied for inference of pre-trained transformers with minimal loss in performance.

We evaluate our model on two automatic speech recognition datasets and showcase that clustered attention consistently achieves better performance than vanilla attention when the computational budget is equalized. Moreover, we demonstrate that our proposed attention can approximate a pretrained BERT model on the popular GLUE and SQuAD benchmarks with only 25 clusters and without loss in performance.

## 5.2 Related Work

In this section, we discuss the most relevant works on scaling transformers to larger sequences. We start by presenting approaches that aim to speed up the attention computation in general. Subsequently, we discuss approaches that speed up transformers without changing the complexity of the attention layer and finally, we summarize the most related works on improving the asymptotic complexity of the attention layer in transformer models.

### 5.2.1 Attention Improvements Before Transformers

Attention has been an integral component of neural networks for sequence modelling for several years (Bahdanau et al., 2014; Xu et al., 2015; Chan et al., 2016). However, its quadratic complexity with respect to the sequence length hinders its applicability on large sequences.

Among the first attempts to address this was the work of Britz et al. (2017) that propose to aggregate the information of the input sequence into fewer vectors and perform attention with these fewer vectors, thus speeding up the attention computation and reducing the memory requirements. However, the input aggregation is performed using a learned but fixed matrix that remains constant for all sequences, hence significantly limiting the expressivity of the model. Similarly, Chiu and Raffel (2017) limit the amount of accessible elements to the attention, by attending monotonically from the past to the future. Namely, if timestep  $i$  attends to position  $j$  then timestep  $i + 1$  cannot attend to any of the earlier positions. Note that in order to speed up the attention computation, the above methods are limiting the number of elements that each layer attends to. Recently, some of these approaches have also been applied in the context of transformers (Ma et al., 2020).

### 5.2.2 Non-asymptotic Improvements

In this section, we summarize techniques that seek to apply transformers to long sequences without focusing on improving the quadratic complexity of self-attention. The most important are Adaptive Attention Span Transformers (Sukhbaatar et al., 2019b) and Transformer-XL (Dai et al., 2019b).

Sukhbaatar et al. (2019b) propose to limit the self-attention context to the closest samples (attention span), in terms of relative distance with respect to the time step, thus reducing both the time and memory requirements of self-attention computation. This is achieved using a masking function with learnable parameters that allows the network to increase the attention span if necessary. Transformer-XL (Dai et al., 2019b), on the other hand, seeks to increase the effective sequence length by introducing segment-level recurrent training, namely splitting the input into segments and attending jointly to the previous and the current segment. The above, combined with a new relative positional encoding results in models that attend to more distant positions than the length of the segment used during training.

Although both approaches have been proven effective, the underlying limitations of self-attention still remains. Attending to an element that is  $N$  timesteps away requires  $\mathcal{O}(N^2)$  memory and computation. In contrast, our model trades-off a small error in the computation of the full attention for an improved *linear* asymptotic complexity. This makes processing long sequences possible.

### 5.2.3 Improvements in Asymptotic Complexity

Child et al. (2019) factorize the self-attention mechanism in local and strided attention. The local attention is computed between the  $C$  nearest positions and the strided attention is computed between positions that are  $C$  steps away from each other. When  $C$  is set to  $\sqrt{N}$  the total asymptotic complexity becomes  $\mathcal{O}(N\sqrt{N})$  both in terms of memory and computation time. With the aforementioned factorization, two self-attention layers are required in order for any position to attend to any other position. In addition, the factorization is fixed and data independent. This makes it intuitive for certain signals (e.g. images), however in most cases it is arbitrary. In contrast, our method automatically groups the input queries that are similar without the need for a manually designed factorization. Moreover, in our model, information flows always from every position to every other position.

Set Transformers (Lee et al., 2019) compute attention between the input sequence  $X$ , of length  $N$  and a set of trainable parameters,  $I$ , called inducing points to get a new sequence  $H$ , of length  $M \ll N$ . The new sequence  $H$  is then used to compute the attention with  $X$  to get the output representation. For a fixed  $M$ , the asymptotic complexity becomes linear with respect to the sequence length. Inducing points are expected to encode some global structure that is task specific. However, this introduces additional model parameters for each attention layer. In contrast to this, we use clustering to project the input to a fixed sequence of smaller

length without any increase in the number of parameters. Moreover, we show that not only our method has the same asymptotic complexity, it can also be used to speed up inference of pretrained models without additional training.

Recently, [Kitaev et al. \(2020\)](#) introduced Reformer. A method that groups positions based on their similarity using locality-sensitive hashing (LSH) and only computes the attention within groups. For groups of fixed size, the asymptotic complexity of Reformer becomes linear with respect to the sequence length. Note that Reformer constrains the queries and keys of self-attention to be equal. As a result, it cannot be applied to neural machine translation, image captioning or memory networks, or generally any application with heterogenous queries and keys. In addition, as it uses hash collisions to form groups it can only handle a small number of bits, thus significantly reducing the quality of the grouping. Instead, our method uses clustering to group the queries, resulting in significantly better groups compared to hash collisions.

### 5.3 Scaling Attention with Fast Clustering

In this section, we formalize the proposed method for approximate softmax attention. In § 5.3.1, we first discuss the attention mechanism in *vanilla transformers* and present its computational complexity. We then introduce *clustered attention* in § 5.3.2 and show that for queries close in the Euclidean space, the attention difference can be bounded by the distance between the queries. This property allows us to reduce the computational complexity by clustering the queries. Subsequently, in § 5.3.3 we show that we can further improve the approximation by first extracting the top- $k$  keys with the highest attention per cluster and then computing the attention on these keys separately for each query that belongs to the cluster. A graphical illustration of our method is provided in appendix D.1.

#### 5.3.1 Vanilla Attention

For any sequence of length  $N$ , the standard attention mechanism that is used in transformers is the dot product attention introduced by [Vaswani et al. \(2017\)](#). Following standard notation, we define the attention matrix  $A \in \mathbb{R}^{N \times N}$  as,

$$A = \text{softmax} \left( \frac{QK^T}{\sqrt{D_k}} \right), \quad (5.1)$$

where  $Q \in \mathbb{R}^{N \times D_k}$  denotes the *queries* and  $K \in \mathbb{R}^{N \times D_k}$  denotes the *keys*. Note that  $\text{softmax}(\cdot)$  is applied row-wise. Using the attention weights  $A$  and the values  $V \in \mathbb{R}^{N \times D_v}$ , we compute the new values  $\hat{V}$  as follows,

$$\hat{V} = AV. \quad (5.2)$$

An intuitive understanding of the attention, as described above, is that given  $Q, K, V$  we create new values  $\hat{V}$  as the weighted average of the old ones, where the weights are defined

by the attention matrix  $A$ . Computing equation 5.1 requires  $\mathcal{O}(N^2 D_k)$  operations and the weighted average of equation 5.2 requires  $\mathcal{O}(N^2 D_v)$ . This results in an asymptotic complexity of  $\mathcal{O}(N^2 D_k + N^2 D_v)$ .

### 5.3.2 Clustered Attention

Instead of computing the attention matrix for all queries, we group them into  $C$  clusters and compute the attention only for these clusters. Then, we use the same attention weights for queries that belong to the same cluster. As a result, the attention computation now becomes  $\mathcal{O}(NCD_k)$ , where  $C \ll N$ .

More formally, let us define  $S \in \{0, 1\}^{N \times C}$ , a partitioning of the queries  $Q$  into  $C$  non-overlapping clusters, such that,  $S_{ij} = 1$ , if the  $i$ -th query  $Q_i$  belongs to the  $j$ -th cluster and 0 otherwise. Using this partitioning, we can now compute the *clustered attention*. First, we compute the cluster centroids as follows,

$$Q_j^c = \frac{\sum_{i=1}^N S_{ij} Q_i}{\sum_{i=1}^N S_{ij}}, \quad (5.3)$$

where  $Q_j^c$  is the centroid of the  $j$ -th cluster. Let us denote  $Q^c \in \mathbb{R}^{C \times D_k}$  as the centroid matrix. Now, we can compute the clustered attention as if  $Q^c$  were the queries. Namely, we compute the clustered attention matrix  $A^c \in \mathbb{R}^{C \times N}$

$$A^c = \text{softmax} \left( \frac{Q^c K^T}{\sqrt{D_k}} \right) \quad (5.4)$$

and the new values  $\hat{V}^c \in \mathbb{R}^{C \times D_v}$

$$\hat{V}^c = A^c V. \quad (5.5)$$

Finally, the value of the  $i$ -th query becomes the value of its closest centroid, namely,

$$\hat{V}_i = \sum_{j=1}^C S_{ij} \hat{V}_j^c. \quad (5.6)$$

From the above analysis, it is evident that we only need to compute the attention weights and the weighted average of the values *once per cluster*. Then, we can broadcast the same value to all queries belonging to the same cluster. This allows us to reduce the number of dot products from  $N$  for each query to  $C$  for each cluster, which results in an asymptotic complexity of  $\mathcal{O}(NCD_k) + \mathcal{O}(CND_v)$ .

Note that in practice, we use multi-head attention, this means that two queries belonging to the same cluster can be clustered differently in another attention head. Moreover, the output of the attention layer involves residual connections. This can cause two queries belonging to the same cluster to have different output representations. The combined effect of residual

connections and multi-head attention allows new clustering patterns to emerge in subsequent layers.

### Quality of the approximation

From the above, we show that grouping queries into clusters can speed-up the self-attention computation. However, in the previous analysis, we do not consider the effects of clustering on the attention weights  $A$ . To address this, we derive a bound for the approximation error. In particular, we show that the difference in attention can be bounded as a function of the Euclidean distance between the queries.

**Proposition 1.** *Given two queries  $Q_i$  and  $Q_j$  such that  $\|Q_i - Q_j\|_2 \leq \epsilon$ ,*

$$\|\text{softmax}(Q_i K^T) - \text{softmax}(Q_j K^T)\|_2 \leq \epsilon \|K\|_2, \quad (5.7)$$

where  $\|K\|_2$  denotes the spectral norm of  $K$ .

*Proof.* Given that  $\text{softmax}(\cdot)$  has Lipschitz constant less than 1 (Gao and Pavel, 2017),

$$\begin{aligned} & \|\text{softmax}(Q_i K^T) - \text{softmax}(Q_j K^T)\|_2 \\ & \leq \|Q_i K^T - Q_j K^T\|_2 \\ & \leq \epsilon \|K\|_2 \end{aligned} \quad (5.8)$$

□

Proposition 1 shows that queries that are close in Euclidean space have similar attention distributions. As a result, the error in the attention approximation for the  $i$ -th query assigned to the  $j$ -th cluster can be bounded by its distance from the cluster centroid  $Q_j^c$ .

### Grouping the Queries

From the discussion, we have shown that given a representative set of queries, we can approximate the attention with fewer computations. Thus, now the problem becomes finding this representative set of queries. K-Means clustering minimizes the sum of squared distances between the cluster members, which would be optimal given our analysis from § 5.3.2. However, for a sequence of length  $N$  one iteration of Lloyd's algorithm for the K-Means optimization problem has an asymptotic complexity  $\mathcal{O}(NCD_k)$ . To speed up the distance computations, we propose to use *Locality-Sensitive Hashing* (LSH) on the queries and then K-Means in Hamming space. In particular, we use the sign of random projections (Shrivastava and Li, 2014) to hash the queries followed by K-Means clustering with hamming distance as the metric. This results in an asymptotic complexity of  $\mathcal{O}(NCL + CBL + ND_k B)$ , where  $L$  is the number of Lloyd iterations and  $B$  is the number of bits used for hashing.



### 5.3.3 Improving clustered attention

In the previous section, we show that clustered attention provides a fast approximation for softmax attention. In this section, we discuss how this approximation can be further improved by considering separately the keys with the highest attention. To intuitively understand the importance of the above, it suffices to consider a scenario where a key with low attention for some query gets a high attention as approximated with the cluster centroid. This can happen when the number of clusters are too low or due to the convergence failure of K-Means. For the clustered attention, described in § 5.3.2, this introduces significant error in the computed value. The variation discussed below addresses such limitations.

After having computed the clustered attention  $A^c$  from equation 5.4, we find the  $k$  keys with the highest attention for each cluster. The main idea then is to improve the attention approximation on these top- $k$  keys for each query that belongs to the cluster. To do so, we first compute the dot product attention as defined in equation 5.1 on these top- $k$  keys for all queries belonging to this cluster. For any query, the computed attention on these top- $k$  keys will sum up to one. This means that it cannot be directly used to substitute the clustered-attention on these keys. To address this, before substitution, we scale the computed attention by the total probability mass assigned by the clustered attention to these top- $k$  keys.

More formally, we start by introducing  $T \in \{0, 1\}^{C \times N}$ , where  $T_{ji} = 1$  if the  $i$ -th key is among the top- $k$  keys for the  $j$ -th cluster and 0 otherwise. We can then compute the probability mass, let it be  $\hat{m}_j$ , of the top- $k$  keys for the  $j$ -th cluster, as follows

$$\hat{m}_j = \sum_{i=1}^N T_{ji} A_{ji}^c. \quad (5.9)$$

Now we formulate an improved attention matrix approximation  $A^t \in \mathbb{R}^{N \times N}$  as follows

$$A_{il}^t = \begin{cases} \frac{\hat{m}_j \exp(Q_i K_l^T)}{\sum_{r=1}^N T_{jr} \exp(Q_i K_r^T)} & \text{if } T_{jl} = 1 \\ A_{jl}^c & \text{otherwise} \end{cases}. \quad (5.10)$$

Note that in the above,  $i$  denotes the  $i$ -th query belonging to the  $j$ -th cluster and  $\sqrt{D_k}$  is omitted for clarity. In particular, equation 5.10 selects the clustered attention of equation 5.4 for keys that are not among the top- $k$  keys for a given cluster. For the rest, it redistributes the mass  $\hat{m}_j$  according to the dot product attention of the queries with the top- $k$  keys. The corresponding new values,  $\hat{V} \in \mathbb{R}^{N \times D_v}$ , are a simple matrix product of  $A^t$  with the values,

$$\hat{V} = A^t V. \quad (5.11)$$

Equation 5.11 can be decomposed into clustered attention computation and two sparse dot products, one for every query with the top- $k$  keys and one for the top- $k$  attention weights with the corresponding values. This adds  $\mathcal{O}(Nk \max(D_k, D_v))$  to the asymptotic complexity of the attention approximation of equation 5.4.

### Quality of the approximation

In the following, we provide proof that improved clustered attention (eq. 5.10) is a direct improvement over the clustered attention (eq. 5.4), in terms of the  $L_1$  distance from the attention matrix  $A$ .

**Proposition 2.** *For the  $i$ -th query belonging to the  $j$ -th cluster, the improved clustered attention  $A_i^t$  and clustered attention  $A_j^c$  relate to the full attention  $A_i$  as follows,*

$$\|A_i^t - A_i\|_1 \leq \|A_j^c - A_i\|_1 \quad (5.12)$$

The proof of the above proposition is presented in section D.2 in the appendix. From equation 5.12 it becomes evident that improved clustered attention will always approximate the full attention better compared to clustered attention.

## 5.4 Experiments

In this section, we analyze experimentally the performance of our proposed method. Initially, we show that our model outperforms our baselines for a given computational budget on a real-world sequence to sequence task, namely automatic speech recognition on two datasets, the Wall Street Journal dataset (§ 5.4.1) and the Switchboard dataset (§ 5.4.2). Subsequently, in § 5.4.3, we demonstrate that our model can approximate a pretrained BERT model (Liu et al., 2019b) on the GLUE (Wang et al., 2019) and SQuAD (Rajpurkar et al., 2018) benchmarks with minimal loss in performance even when the number of clusters is less than one tenth of the sequence length. We also provide, in appendix D.3.1, a thorough benchmark that showcases the linear complexity of *clustered attention* and an ablation study regarding how the number of clusters scale with respect to the sequence length.

We compare our model with the vanilla transformers (Vaswani et al., 2017), which we refer to as **full** and the Reformer (Kitaev et al., 2020), which we refer to as **lsh-X**, where  $X$  denotes the rounds of hashing. We refer to *clustered attention*, introduced in § 5.3.2, as **clustered-X** and to *improved clustered attention*, introduced in § 5.3.3, as **i-clustered-X**, where  $X$  denotes the number of clusters. Unless mentioned otherwise we use  $k = 32$  for the top- $k$  keys with improved clustered.

All experiments are conducted using NVidia GTX 1080 Ti with 11GB of memory and all models are implemented in PyTorch (Paszke et al., 2019). For Reformer we use a PyTorch port of the published code. Note that we do not use reversible layers since it is a technique that could be applied to all methods. Our PyTorch code can be found at <https://clustered-transformers.github.io>.

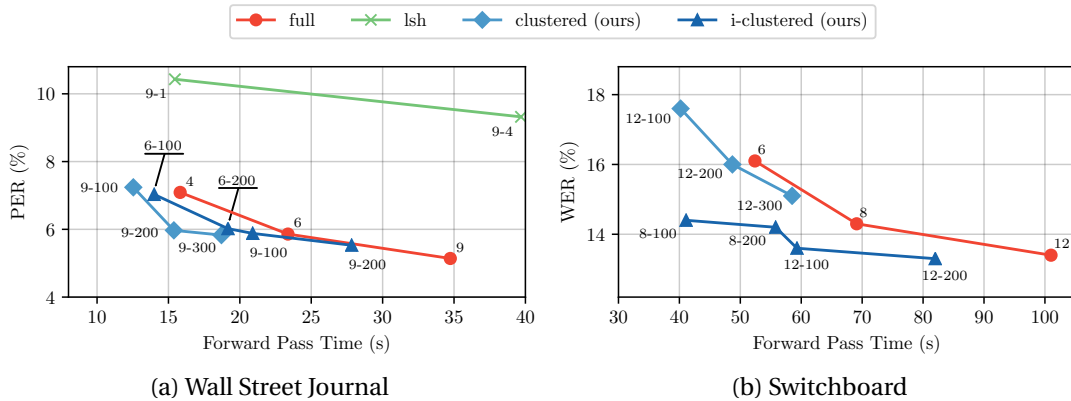


Figure 5.1 – We compare the achieved performance of various transformer models under an equalized computational budget. The numbers near the datapoints denote the number of layers and number of clusters or hashing rounds where applicable. *i-clustered* is consistently better than all baselines for a given computational budget both in WSJ and Switchboard datasets. The details can be found in § 5.4.1 and § 5.4.2 respectively.

#### 5.4.1 Evaluation on Wall Street Journal (WSJ)

In our first experiment, we employ the Wall-Street Journal dataset (Paul and Baker, 1992b). The input to all transformers is 40-dimensional filter-bank features with fixed positional embeddings. We train using Connectionist Temporal Classification (CTC) (Graves et al., 2006b) loss with phonemes as ground-truth labels. The approximate average and maximum sequence lengths for the training inputs are 780 and 2500 respectively.

**Speed Accuracy Trade-off:** We start by comparing the performance of our proposed model with various transformer variants under an equalized computational budget. To this end, we train *full* with 4, 6 and 9 layers to get a range of the required computation time and achieved *phone error rate* (PER). Similarly, we train *i-clustered* with 6 and 9 layers. Both models are trained with 100 and 200 clusters. We also train *clustered* with 9 layers, and 100, 200 and 300 clusters. Finally, we train Reformer with 9 layers, and 1 and 4 hashing rounds. We refer the reader to appendix D.3.3 for the specifics of all transformer architectures as well as their training details. In figure 5.1a, we plot the achieved PER on the validation set with respect to the required time to perform a full forward pass. Our *i-clustered* achieves lower PER than all other baselines for a given computational budget.

**Approximation Quality:** To assess the approximation capabilities of our method, we train different transformer variants on the aforementioned task and evaluate them using other self-attention implementations during inference. As the Reformer requires the queries to be identical to the keys to evaluate its approximation ability we also train a full attention model with shared queries and keys, which we refer to as **shared-full**. Note that both clustered attention and improved clustered attention can be used for approximating shared-full,

## Chapter 5. Fast Transformers with Clustered Attention

simply by setting keys to be equal to queries. Table 5.1 summarizes the results. We observe that improved clustered attention (7-8 rows) achieves the lowest phone error rate in every comparison. This implies that it is the best choice for approximating pre-trained models. In addition, we also note that as the number of clusters increases, the approximation improves as well. Furthermore, to show that the top keys alone are not sufficient for approximating

		Train with					
		full	shared-full	lsh-1	lsh-4	clustered-100	i-clustered-100
Evaluate with	full	<u>5.14</u>	-	-	-	7.10	5.56
	shared-full	-	<u>6.57</u>	25.16	41.61	-	-
	lsh-1	-	71.40	<u>10.43</u>	13.76	-	-
	lsh-4	-	64.29	9.35	<u>9.33</u>	-	-
	clustered-100	44.88	40.86	68.06	66.43	<u>7.06</u>	18.83
	clustered-200	21.76	25.86	57.75	57.24	6.34	8.95
	i-clustered-100	9.29	13.22	41.65	48.20	8.80	<u>5.95</u>
	i-clustered-200	6.38	8.43	30.09	42.43	7.71	5.60
	oracle-top	17.16	77.18	43.35	59.38	24.32	6.96

Table 5.1 – We report validation phone error rate (PER) on the WSJ dataset (§ 5.4.1). We train with one model and evaluate with another to assess the approximation abilities of different models. Underline denotes training and testing with the same model. Improved cluster (rows 7-8) approximates the full and the shared-full significantly better than all the other fast attention methods.

*full*, we also compare with an attention variant, that for each query only keeps the 32 keys with the highest attention. We refer to the latter as **oracle-top**. We observe that oracle-top achieves significantly larger phone error rate than improved clustered in all cases. This implies that improved clustered attention also captures the significant long tail of the attention distribution.

**Convergence Behaviour:** In Table 5.2, we report the required time per epoch as well as the total training time for all transformer variants with 9 layers. For completeness, we also provide the corresponding phone error rates on the test set. We observe that clustered attention is more than two times faster than full (per epoch) and achieves significantly lower PER than both Reformer variants (lsh-1 and lsh-4). Improved clustered is the only method that is not only faster per epoch but also in total wall-clock time required to converge.

### 5.4.2 Evaluation on Switchboard

We also evaluate our model on the Switchboard dataset (Godfrey et al., 1992), which is a collection of 2,400 telephone conversations on common topics among 543 strangers. All transformers are trained with lattice-free MMI loss (Povey et al., 2016) and as inputs we use 80-dimensional filter-bank features with fixed positional embeddings. The average input

	full	lsh-1	lsh-4	clustered-100	i-clustered-100
PER (%)	5.03	9.43	8.59	7.50	5.61
Time/Epoch (s)	2514	1004	2320	803	1325
Convergence Time (h)	87.99	189.64	210.09	102.15	72.14

Table 5.2 – We report the test PER, the time per training epoch (in seconds) and the wall-clock time required for the convergence of each model (in hours).

sequence length is roughly 534 and the maximum sequence length is approximately 3850. Details regarding the transformer architectures as well as their training details are provided in appendix D.3.3.

**Speed Accuracy Trade-off:** Similar to § 5.4.1, we compare the performance of various transformer models given a specific computational budget. To this end, we train *full* with 6, 8 and 12 layers. Similarly, we train *i-clustered* with 8 and 12 layers; both with 100 and 200 clusters. Finally, we also train *clustered* with 12 layers, and 100, 200 and 300 clusters. In figure 5.1b, we plot the achieved word error rate (WER) in the validation set of Switchboard with respect to the required time to perform a full forward pass. Our *i-clustered* is consistently better than *full* for a given computational budget. In particular, for a budget of approximately 50 seconds, improved clustered achieves more than 2 percentage points lower WER. Furthermore, we note that it is consistently better than clustered attention for all computational budgets.

**Convergence Behaviour:** Table 5.3 summarizes the computational cost of training the transformer models with 12 layers in the Switchboard dataset as well as the WER in the test set. We observe that due to the larger sequences in this dataset both clustered and i-clustered are faster to train per epoch and with respect to total required wall-clock time.

	full	clustered-100	i-clustered-100
WER (%)	15.0	18.5	15.5
Time/Epoch (h)	3.84	1.91	2.57
Convergence Time (h)	228.05	132.13	127.44

Table 5.3 – We report the test set WER, the time per training epoch (in hours) and the wall-clock time required for the convergence of each model (in hours).

### 5.4.3 RoBERTa Approximation

To highlight the ability of our model to approximate arbitrarily complicated attention distributions, we evaluate our proposed method on the approximation of a fine-tuned RoBERTa model (Liu et al., 2019b) on the GLUE (Wang et al., 2019) and SQuAD (Rajpurkar et al., 2018) benchmarks. In particular, we evaluate on 10 different tasks, among which there are tasks

## Chapter 5. Fast Transformers with Clustered Attention

---

such as question answering (SQuAD) and textual entailment (RTE), which exhibit arbitrary and sparse attention patterns. We refer the reader to [Wang et al. \(2019\)](#); [Rajpurkar et al. \(2018\)](#) for a detailed analysis of all tasks.

For the GLUE tasks, the maximum sequence length is 128 while for SQuAD, it is 384. For each task, we use 25 clusters for approximation which is less than 20% and 10% of the input sequence length for GLUE and SQuAD tasks respectively. In Table 5.4, we summarize the performance per task. We observe that improved clustered performs as well as the full transformer in all tasks but SQuAD, in which it is only marginally worse. Moreover, we note that clustered performs significantly worse in tasks that require more complicated attention patterns such as SQuAD and RTE. For inference time, *full* was faster than the *clustered* attention variants due to short sequence lengths.

	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI	SQuAD
full	0.601	0.880	0.868	0.929	0.915	0.682	0.947	0.900	0.437	0.904
clustered-25	0.598	0.794	0.436	0.746	0.894	0.498	0.944	0.789	0.437	0.006
i-clustered-25	0.601	0.880	0.873	0.930	0.915	0.704	0.947	0.900	0.437	0.876

Table 5.4 – We report the performance on GLUE and SQuAD benchmarks. Following common practice, we report accuracy for all tasks except STS-B and SQuAD, where we report Pearson correlation and F1-score respectively. For all metrics higher is better.

## 5.5 Conclusions

We have presented *clustered attention* a method that approximates vanilla transformers with significantly lower computational requirements. In particular, we have shown that our model can be up to  $2\times$  faster during training and inference with minimal loss in performance. In contrast to recent fast variations of transformers, we have also shown that our method can efficiently approximate pre-trained models with full attention while retaining the linear asymptotic complexity.

The proposed method opens several research directions towards applying transformers on long sequence tasks such as music generation, scene flow estimation etc. We consider masked language modeling for long texts to be of particular importance, as it will allow finetuning for downstream tasks that need a context longer than the commonly used 512 tokens.

## 6 Conclusions and Future Work

In this thesis, we have presented novel ways for improving the efficiency of deep neural networks. Firstly, we improved the sample inefficiency of neural networks using importance sampling. Subsequently, we introduced a method that focuses the computation of neural networks to informative parts of the input and finally, we improved the asymptotic complexity of the transformer architecture with our linear and clustered attention.

In more detail, in Chapter 2, we show that most of the computation during the training of deep neural networks is spent on examples that have negligible gradients and thus can be ignored. In order to develop an algorithm that is suitable for deep neural networks, we derive an upper bound to the per-sample gradient norm that can be computed in a single forward pass. Subsequently, we show that we can quantify the variance reduction of the stochastic gradient estimator achieved when sampling the data points with importance instead of uniform. Using these two insights, we develop an importance sampling algorithm, suitable for deep learning models that requires no tuning and results in improvements in both training loss and test set performance for a given computational budget.

After focusing the computation on useful parts of the dataset, in Chapter 3, we use importance sampling again to focus the computation on informative parts of a single large input. In particular, we use a small and efficient neural network to compute an attention distribution over an input image and then sample parts from the attention distribution in order to process only a tiny fraction of the full image to make our prediction. Our key contribution lies in our algorithm for training both the attention network and the feature extraction network in an end-to-end fashion despite the sampling procedure and only using image wide labels for supervision. As a result, we achieve the same performance as state-of-the-art models using an order of magnitude less computation and memory.

Subsequently, in Chapter 4, we shifted our attention to the transformer architecture. Specifically, we tackle the quadratic complexity of the self-attention component in transformer models. We showed that expressing the similarities of the queries and keys as dot products of kernel feature maps allows for computing the attention with linear complexity both in

terms of memory and computation. Moreover, by extending this formulation to autoregressive transformers, we increase the throughput of autoregressive inference by up to 3 orders of magnitude and uncover the relationship of transformers to recurrent neural networks.

Finally, in Chapter 5, we present clustered attention which is an approximation method for the commonly used softmax attention. In particular, we utilize the fact that queries that are close in Euclidean space have similar attention distributions to group them in cluster and only compute the attention for the cluster centroids. We show that using improved clustered attention results in better performance than softmax attention for a fixed computational budget. More importantly, we also demonstrate that our proposed method can approximate pre-trained transformers without any fine-tuning and no loss in performance.

### 6.1 Future work

In this thesis, we focused our attention on approximating computationally expensive operations with more efficient counterparts. In the first half of our work, we primarily utilized importance sampling to achieve this approximations. Whereas in the second part, we focused on improving the asymptotic complexity of the transformer architecture. We believe that our work opens up a multitude of new research directions.

In modern deep embedding learning problems the state-of-the-art results are achieved with the use of ranking losses such as contrastive loss or triplet loss, which result in  $\mathcal{O}(N^2)$  and  $\mathcal{O}(N^3)$  possible samples, where  $N$  is the size of the dataset (Schroff et al., 2015). In these problems, importance sampling becomes a necessity. A promising future research direction is to extend our importance sampling algorithm such that it takes advantage of the specific structure of this problem and efficiently sample among millions of possible triplets or pairs. We demonstrated that importance sampling, in addition to providing a speedup, can also stabilize training by providing consistent gradients with lower variance. Generative adversarial networks (GANs) (Goodfellow et al., 2014) are notoriously hard to train; thus we would like to investigate the effects of importance sampling on this hard optimization problem. Specifically, in vanilla GANs the generated samples might be easily recognized by the discriminator resulting in small gradients that significantly slow down learning the parameters of the generator. In the literature, there have been many extensions of the original vanilla GAN to mitigate this issue. However, the most popular of them, the Wasserstein GAN (Arjovsky et al., 2017) and its improved variant (Gulrajani et al., 2017), require computing second order gradients and are significantly slower per parameter update. To this end, we propose investigating whether importance sampling of the latent space of vanilla GANs can produce meaningful gradients for the generator without using significantly more computational resources.

For a variety of computer vision tasks, such as cancer detection, self driving vehicles and satellite image processing, it is necessary to develop models that process signals of extremely high resolution. For example, the typical image size that is used to analyze tissue samples from biopsies is several gigapixels. State-of-the-art approaches for these kind of data typically



divide the input into patches and process them separately (Hou et al., 2016; Nazeri et al., 2018; Golatkar et al., 2018). As a result, these approaches, waste computation on patches that either contain little information for the task or require patch level annotations, which are hard to acquire. A natural extension of our work on *attention sampling* for megapixel images is to utilize attention at several resolutions in order to be able to process even larger inputs.

Finally, our work on efficient transformer architectures enables their application on modalities and problems that were previously computationally prohibitive. In particular, in future research, we would like to investigate the application of linear transformers on autoregressive modelling of video and audio that typically have an input length larger than tens of thousands of tokens. Moreover, we believe that clustered attention is particularly well suited for images, where pixels can be naturally grouped into clusters. Finally our kernelized attention formulation from Chapter 4 has spawned a significant amount of research on appropriate kernel feature maps for self attention (Choromanski et al., 2020; Peng et al., 2021; Schlag et al., 2021). Nevertheless, the expressivity of the feature map, in these works, is proportional to an increase in computational complexity. We believe that a promising research direction is the development of a sparse feature map that allows for increased expressivity without increasing the computational requirements.



# A Appendix for Chapter 2

## A.1 Differences of variances

In the following equations we quantify the variance reduction achieved with importance sampling using the gradient norm. Let  $g_i \propto \|\nabla_{\theta_t} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2 = \|G_i\|_2$  and  $u = \frac{1}{B}$  the uniform probability.

We want to compute

$$\text{Tr}(\mathbb{V}_u[G_i]) - \text{Tr}(\mathbb{V}_g[w_i G_i]) = \mathbb{E}_u[\|G_i\|_2^2] - \mathbb{E}_g[w_i^2 \|G_i\|_2^2]. \quad (\text{A.1})$$

Using the fact that  $w_i = \frac{1}{B g_i}$  we have

$$\mathbb{E}_g[w_i^2 \|G_i\|_2^2] = \left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2, \quad (\text{A.2})$$

thus

$$\text{Tr}(\mathbb{V}_u[G_i]) - \text{Tr}(\mathbb{V}_g[w_i G_i]) \quad (\text{A.3})$$

$$= \frac{1}{B} \sum_{i=1}^B \|G_i\|_2^2 - \left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2 \quad (\text{A.4})$$

$$= \frac{(\sum_{i=1}^B \|G_i\|_2)^2}{B^3} \sum_{i=1}^B \left( B^2 \frac{\|G_i\|_2^2}{(\sum_{i=1}^B \|G_i\|_2)^2} - 1 \right) \quad (\text{A.5})$$

$$= \frac{(\sum_{i=1}^B \|G_i\|_2)^2}{B} \sum_{i=1}^B (g_i^2 - u^2). \quad (\text{A.6})$$

Completing the squares at equation A.6 and using the fact that  $\sum_{i=1}^B u = 1$  we complete the

derivation.

$$\text{Tr}(\mathbb{V}_u[G_i]) - \text{Tr}(\mathbb{V}_g[w_i G_i]) \quad (\text{A.7})$$

$$= \frac{(\sum_{i=1}^B \|G_i\|_2)^2}{B} \sum_{i=1}^B (g_i - u)^2 \quad (\text{A.8})$$

$$= \left( \frac{1}{B} \sum_{i=1}^B \|G_i\|_2 \right)^2 B \|g - u\|_2^2. \quad (\text{A.9})$$

## A.2 An upper bound to the gradient norm

In this section, we reiterate the analysis from § 2.3.2 with more details.

Let  $\theta^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$  be the weight matrix for layer  $l$  and  $\sigma^{(l)}(\cdot)$  be a Lipschitz continuous activation function. Then, let

$$x^{(0)} = x \quad (\text{A.10})$$

$$z^{(l)} = \theta^{(l)} x^{(l-1)} \quad (\text{A.11})$$

$$x^{(l)} = \sigma^{(l)}(z^{(l)}) \quad (\text{A.12})$$

$$\Psi(x; \Theta) = x^{(L)}. \quad (\text{A.13})$$

Equations A.10-A.13 define a simple fully connected neural network without bias to simplify the closed form definition of the gradient with respect to the parameters  $\Theta$ .

In addition we define the gradient of the loss with respect to the output of the network as

$$\nabla_{x_i^{(L)}} \mathcal{L} = \nabla_{x_i^{(L)}} \mathcal{L}(\Psi(x_i; \Theta), y_i) \quad (\text{A.14})$$

and the gradient of the loss with respect to the output of layer  $l$  as

$$\nabla_{x_i^{(l)}} \mathcal{L} = \Delta_i^{(l)} \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(L)}} \mathcal{L} \quad (\text{A.15})$$

where

$$\Delta_i^{(l)} = \Sigma_l'(z_i^{(l)}) \theta_{l+1}^T \dots \Sigma_{L-1}'(z_i^{(L-1)}) \theta_L^T \quad (\text{A.16})$$

propagates the gradient from the last layer (pre-activation) to layer  $l$  and

$$\Sigma_l'(z) = \text{diag}\left(\sigma'^{(l)}(z_1), \dots, \sigma'^{(l)}(z_{M_l})\right) \quad (\text{A.17})$$

defines the gradient of the activation function of layer  $l$ .

Finally, the gradient with respect to the parameters of the  $l$ -th layer can be written

$$\|\nabla_{\theta_l} \mathcal{L}(\Psi(x_i; \Theta), y_i)\|_2 \quad (\text{A.18})$$

$$= \left\| \left( \Delta_i^{(l)} \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(l)}} \mathcal{L} \right) \left( x_i^{(l-1)} \right)^T \right\|_2 \quad (\text{A.19})$$

$$\leq \left\| x_i^{(l-1)} \right\|_2 \left\| \Delta_i^{(l)} \right\|_2 \left\| \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(l)}} \mathcal{L} \right\|_2. \quad (\text{A.20})$$

We observe that  $x_i^{(l)}$  and  $\Delta_i^{(l)}$  depend only on  $z_i$  and  $\Theta$ . However, we theorize that due to various weight initialization and activation normalization techniques those quantities do not capture the important per sample variations of the gradient norm. Using the above, which is also shown experimentally to be true in § 4.1, we deduce the following upper bound per layer

$$\|\nabla_{\theta_l} \mathcal{L}(\Psi(x_i; \Theta), y_i)\|_2 \quad (\text{A.21})$$

$$\leq \max_{l,i} \left( \left\| x_i^{(l-1)} \right\|_2 \left\| \Delta_i^{(l)} \right\|_2 \right) \left\| \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(l)}} \mathcal{L} \right\|_2 \quad (\text{A.22})$$

$$= \rho \left\| \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(l)}} \mathcal{L} \right\|_2, \quad (\text{A.23})$$

which can then be used to derive our final upper bound

$$\|\nabla_{\Theta} \mathcal{L}(\Psi(x_i; \Theta), y_i)\|_2 \leq L \rho \underbrace{\left\| \Sigma_L'(z_i^{(L)}) \nabla_{x_i^{(l)}} \mathcal{L} \right\|_2}_{\hat{G}_i}. \quad (\text{A.24})$$

Intuitively, equation A.24 means that the variations of the gradient norm are mostly captured by the final classification layer. Consequently, we can use the gradient of the loss with respect to the pre-activation outputs of our neural network as an upper bound to the per-sample gradient norm.

### A.3 Comparison with SVRG methods

For completeness, we also compare our proposed method with Stochastic Variance Reduced Gradient methods and present the results in this section. We follow the experimental setup of § 4.2 and evaluate on the augmented CIFAR10 and CIFAR100 datasets. The algorithms we considered were SVRG (Johnson and Zhang, 2013), accelerated SVRG with Katyusha momentum (Allen-Zhu, 2017) and, the most suitable for Deep Learning, SCSG (Lei et al., 2017) which in practice is a mini-batch version of SVRG. SAGA (Defazio et al., 2014) was not considered due to the prohibitive memory requirements for storing the per sample gradients.

For all methods, we tune the learning rate and the epochs per batch gradient computation ( $m$  in SVRG literature). For SCSG, we also tune the large batch (denoted as  $B_j$  in Lei et al. (2017)) and its growth rate. The results are depicted in figure A.1. We observe that SGD with momentum performs significantly better than all SVRG methods. Full batch SVRG and Katyusha perform a small number of parameter updates thus failing to optimize the networks.

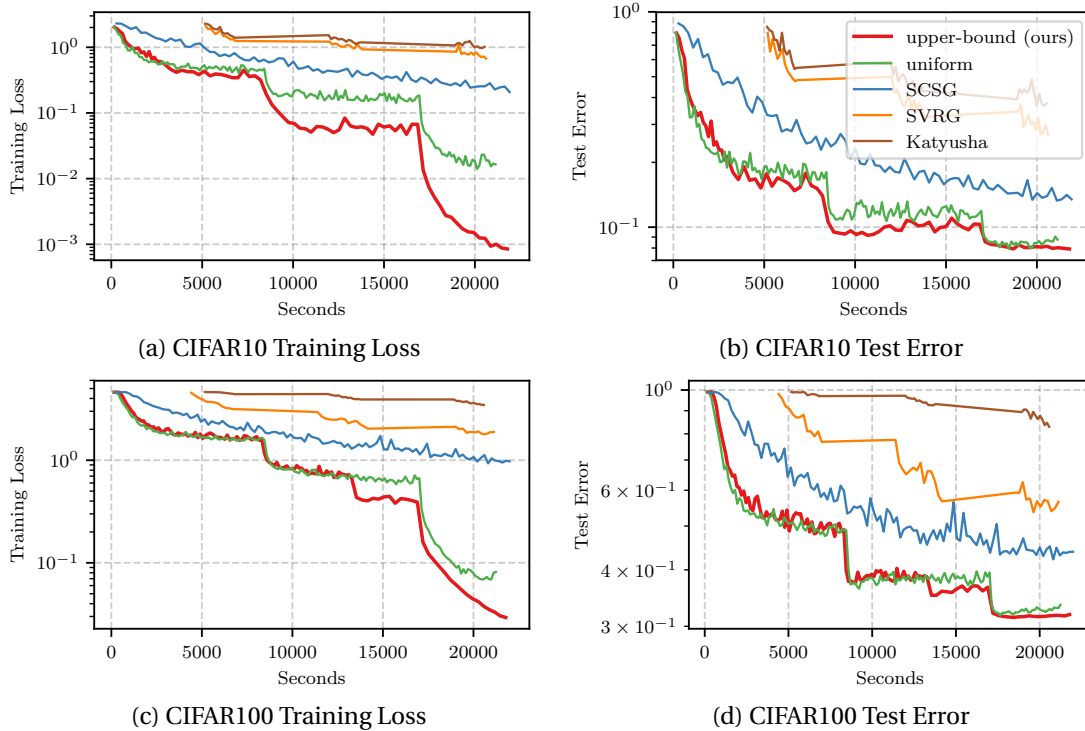


Figure A.1 – Comparison of our proposed importance sampling scheme (*upper-bound*) to SGD with uniform sampling and variance reduced methods. Only SCSG can actually perform enough iterations to optimize the network. However, SGD with uniform sampling and our *upper-bound* greatly outperform SCSG.

In all cases, the best variance reduced method achieves more than an order of magnitude higher training loss than our proposed importance sampling scheme.

### A.4 Ablation study on $B$

The only hyperparameter that is somewhat hard to define in our algorithm is the pre-sampling size  $B$ . As mentioned in § 2.3.3, it controls the maximum possible variance reduction and also how much wall-clock time one iteration with importance sampling will require.

In figure A.2 we depict the results of training with importance sampling and different pre-sampling sizes on CIFAR10. We follow the same experimental setup as in § 2.4.2.

We observe that larger presampling size results in lower training loss, which follows from our theory since the maximum variance reduction is smaller with small  $B$ . In this experiment we use the same  $\tau_{th}$  for all the methods and we observe that  $B = 384$  reaches first to 0.6 training loss. This is justified because computing the importance for 1,024 samples in the beginning of training is wasteful according to our analysis.

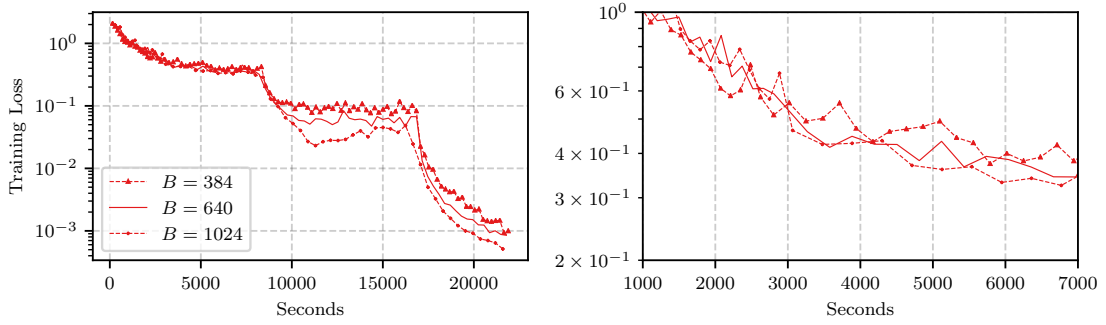


Figure A.2 – Results on training with different  $B$  on CIFAR10. See § 2.4.2 for the experimental setup.

According to this preliminary ablation study for  $B$ , we conclude that choosing  $B = kb$  with  $2 < k < 6$  is a good strategy for achieving a speedup. However, regardless of the choice of  $B$ , pairing it with a threshold  $\tau_{th}$  designated by the analysis in the chapter guarantees that the algorithm will be spending time on importance sampling only when the variance can be greatly reduced.

## A.5 Importance Sampling with the Loss

In this section we will present a small analysis that provides intuition regarding using the loss as an approximation or an upper bound to the per sample gradient norm.

Let  $\mathcal{L}(\psi, y) : D \rightarrow \mathbb{R}$  be either the negative log likelihood through a sigmoid or the squared error loss function defined respectively as

$$\begin{aligned} \mathcal{L}_1(\psi, y) &= -\log\left(\frac{\exp(y\psi)}{1 + \exp(y\psi)}\right) & y \in \{-1, 1\} & \psi \in \mathbb{R} \\ \mathcal{L}_2(\psi, y) &= \|y - \psi\|_2^2 & y \in \mathbb{R}^d & \psi \in \mathbb{R}^d \end{aligned} \quad (\text{A.25})$$

Given our upper bound to the gradient norm, we can write

$$\|\nabla_{\theta_t} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2 \leq L\rho \|\nabla_{\psi} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2. \quad (\text{A.26})$$

Moreover, for the losses that we are considering, when  $\mathcal{L}(\psi, y) \rightarrow 0$  then  $\|\nabla_{\psi} \mathcal{L}(\Psi(x_i; \theta_t), y_i)\|_2 \rightarrow 0$ . Using this fact in combination to equation A.26, we claim that so does the per sample gradient norm thus small loss values imply small gradients. However, large loss values are not well correlated with the gradient norm which can also be observed in § 2.4.1.

To summarize, we conjecture that due to the above facts, sampling proportionally to the loss reduces the variance only when the majority of the samples have losses close to 0. Our assumption is validated from our experiments, where the *loss* struggles to achieve a speedup

## **Appendix A. Appendix for Chapter 2**

---

in the early stages of training where most samples still have relatively large loss values.



# B Appendix for Chapter 3

## B.1 Introduction

This supplementary material is organised as follows: In § B.2 and § B.3 we provide the detailed derivation of the gradients for our *attention sampling*. Subsequently, in § B.4 we mention additional related work that might be of interest to the readers. In § B.5 and § B.6 we present experiments that analyse the effect of our entropy regularizer and the number of patches sampled on the learned attention distribution. In § B.7, we visualize the attention distribution of our method to show it focuses computation on the informative parts of the high resolution images. Finally, in § B.8 we provide details with respect to the architectures trained for our experiments.

## B.2 Sampling with replacement

In this section, we detail the derivation of equation 11 in our main submission. In order to be able to use a neural network as our attention distribution we need to derive the gradient of the loss with respect to the parameters of the attention function  $a(\cdot; \Theta)$  through the sampling of the set of indices  $Q$ . Namely, we need to compute

$$\frac{\partial \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q}{\partial \theta} \tag{B.1}$$

for all  $\theta \in \Theta$  including the ones that affect  $a(\cdot)$ .

By exploiting the Monte Carlo approximation and the multiply by one trick, we get

$$\frac{\partial}{\partial \theta} \frac{1}{N} \sum_{q \in Q} f(x; \Theta)_q \quad (\text{B.2})$$

$$\approx \frac{\partial}{\partial \theta} \sum_{i=1}^K a(x; \Theta)_i f(x; \Theta)_i \quad (\text{B.3})$$

$$= \sum_{i=1}^K \frac{\partial}{\partial \theta} [a(x; \Theta)_i f(x; \Theta)_i] \quad (\text{B.4})$$

$$= \sum_{i=1}^K \frac{a(x; \Theta)_i}{a(x; \Theta)_i} \frac{\partial}{\partial \theta} [a(x; \Theta)_i f(x; \Theta)_i] \quad (\text{B.5})$$

$$= \mathbb{E}_{I \sim a(x; \Theta)} \left[ \frac{\frac{\partial}{\partial \theta} [a(x; \Theta)_I f(x; \Theta)_I]}{a(x; \Theta)_I} \right]. \quad (\text{B.6})$$

### B.3 Sampling without replacement

In this section, we derive the gradients of the attention distribution with respect to the feature network and attention network parameters. We define

- $f_i = f(x; \Theta)_i$  for  $i \in \{1, 2, \dots, K\}$  to be the  $K$  features
  
- $a_i = a(x; \Theta)_i$  for  $i \in \{1, 2, \dots, K\}$  to be the probability of the  $i$ -th feature from the attention distribution  $a$
  
- $w_i = \sum_{j \neq i} a_j$

We consider sampling without replacement to be sampling an index  $i$  from  $a$  and then sampling from the distribution  $p_i(j)$  defined for  $j \in \{1, 2, \dots, i-1, i+1, \dots, K\}$  as follows,

$$p_i(j) = \frac{a_j}{w_i}. \quad (\text{B.7})$$

Given samples  $i, j$  sampled from  $a$  and  $p_i$ , we can make an unbiased estimator for  $\mathbb{E}_{I \sim a}[f_I]$  as

follows,

$$a_i f_i + w_i f_j \simeq \quad (\text{B.8})$$

$$\mathbb{E}_{I \sim a} [\mathbb{E}_{J \sim p_I} [a_I f_I + w_I f_J]] = \quad (\text{B.9})$$

$$\mathbb{E}_{I \sim a} [a_I f_I + \mathbb{E}_{J \sim p_I} [w_I f_J]] = \quad (\text{B.10})$$

$$\mathbb{E}_{I \sim a} \left[ a_I f_I + \sum_{j \neq I} a_j f_j \right] = \quad (\text{B.11})$$

$$\mathbb{E}_{I \sim a} \left[ \sum_{j=1}^K a_j f_j \right] = \quad (\text{B.12})$$

$$\mathbb{E}_{I \sim a} [f_I]. \quad (\text{B.13})$$

Using the same  $i, j$  sampled from  $a$  and  $p_i$  accordingly, we can estimate the gradient as follows,

$$\frac{\partial}{\partial \theta} \mathbb{E}_{I \sim a} [f_I] = \quad (\text{B.14})$$

$$\frac{\partial}{\partial \theta} \mathbb{E}_{I \sim a} [\mathbb{E}_{J \sim p_I} [a_I f_I + w_I f_J]] = \quad (\text{B.15})$$

$$\frac{\partial}{\partial \theta} \sum_{i=1}^K \sum_{j \neq i} a_i p_i(j) (a_i f_i + w_i f_j) = \quad (\text{B.16})$$

$$\sum_{i=1}^K \sum_{j \neq i} \frac{\partial}{\partial \theta} a_i p_i(j) (a_i f_i + w_i f_j) = \quad (\text{B.17})$$

$$\sum_{i=1}^K \sum_{j \neq i} \frac{a_i p_i(j)}{a_i p_i(j)} \frac{\partial}{\partial \theta} a_i p_i(j) (a_i f_i + w_i f_j) = \quad (\text{B.18})$$

$$\mathbb{E}_{I \sim a} \left[ \mathbb{E}_{J \sim p_I} \left[ \frac{\frac{\partial}{\partial \theta} a_I p_I(J) (a_I f_I + w_I f_J)}{a_I p_I(J)} \right] \right] \simeq \quad (\text{B.19})$$

$$\frac{\frac{\partial}{\partial \theta} a_i p_i(j) (a_i f_i + w_i f_j)}{a_i p_i(j)} = \quad (\text{B.20})$$

$$\frac{p_i(j) (a_i f_i + w_i f_j) \frac{\partial}{\partial \theta} a_i}{a_i p_i(j)} + \frac{a_i \frac{\partial}{\partial \theta} p_i(j) (a_i f_i + w_i f_j)}{a_i p_i(j)} = \quad (\text{B.21})$$

$$(a_i f_i + w_i f_j) \frac{\frac{\partial}{\partial \theta} a_i}{a_i} + \frac{\frac{\partial}{\partial \theta} p_i(j) (a_i f_i + w_i f_j)}{p_i(j)} = \quad (\text{B.22})$$

$$(a_i f_i + w_i f_j) \frac{\partial}{\partial \theta} \log(a_i) + \frac{\frac{\partial}{\partial \theta} p_i(j) (a_i f_i + w_i f_j)}{p_i(j)}. \quad (\text{B.23})$$

When we extend the above computations for sampling more than two samples, the logarithm in equation B.23 allows us to avoid the numerical errors that arise from the cumulative product at equation B.20.

### B.4 Extra related work

For completeness, in this section we discuss parts of the literature that are tangentially related to our work.

[Combalia and Vilaplana \(2018\)](#) consider the problem of high-resolution image classification from the Multiple Instance Learning perspective. The authors propose a two-step procedure; initially random patches are sampled and classified. Subsequently, more patches are sampled around the patches that resulted in confident predictions. The most confident prediction is returned. Due to the lack of the attention mechanism, this model relies in identifying the region of interest via the initial random patches. However, in the second pass the prediction is finetuned if informative patches are likely to be spatially close with each other.

[Maggiori et al. \(2017\)](#) propose a neural network architecture for the pixelwise classification of high resolution images. The authors consider features at several resolutions and train a pixel-by-pixel fully connected network to combine the features into the final classification. The aforementioned approach could be used with our *attention sampling* to approach pixelwise classification tasks such as semantic segmentation.

### B.5 Ablation study on the entropy regularizer

To characterize the effect of the entropy regularizer on our *attention sampling*, we train with the same experimental setup as for the histopathology images of § 3.4.3 but varying the entropy regularizer  $\lambda \in \{0, 0.01, 0.1, 1\}$ . The results are depicted in Figure B.1. Using no entropy regularizer results in a very selective attention distribution in the first 60 epochs of training. On the other hand, a high value for  $\lambda$ , the entropy regularizer weight, drives the sampling distribution towards uniform.

In our experiments we observed that values close to 0.01 (e.g. 0.005 or 0.05) had no observable difference in terms of the final attention distribution.

### B.6 Ablation study on the number of patches

According to our theory, the number of patches should not affect the learned attention distribution. Namely, the expectation of the gradients and the predictions should be the same and the only difference is in the variance.

In Figure B.2, we visualize, in a similar fashion to B.5, the attention distributions learned when sampling various numbers of patches per image for training. Although the distributions are different in the beginning of training after approximately 100 epochs they converge to a very similar attention distribution.

## B.6. Ablation study on the number of patches

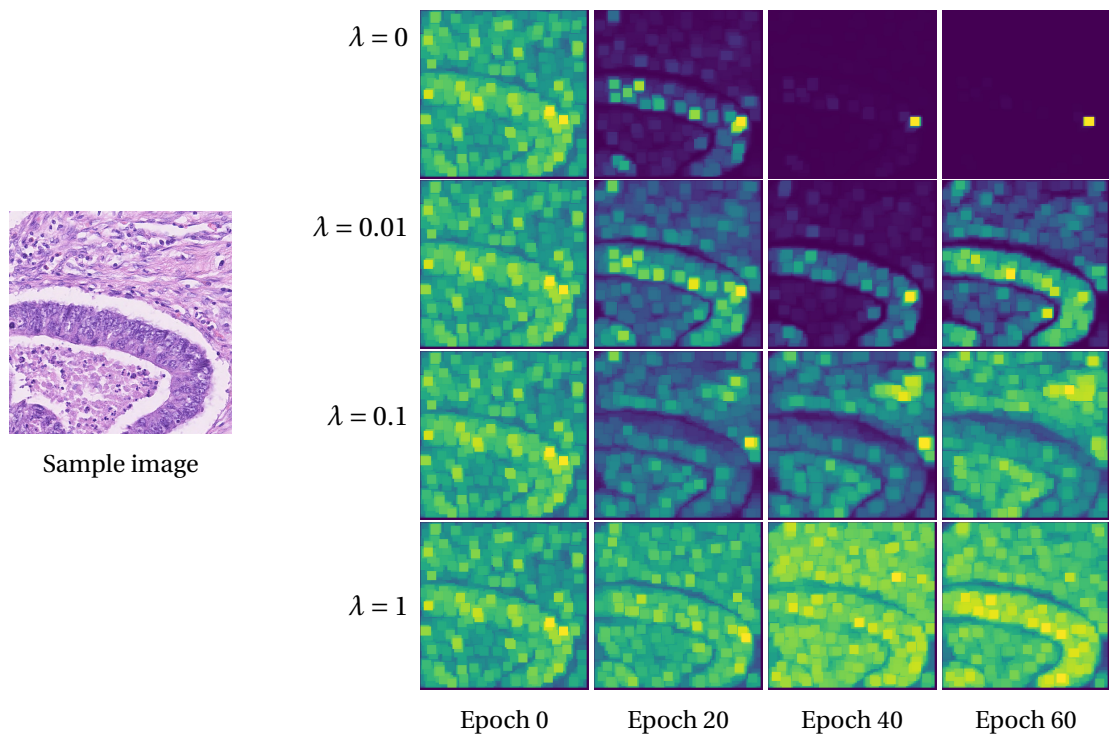


Figure B.1 – We visualize the effects of the entropy regularizer on the sampling distribution computed from a test image of the *colon cancer* dataset in the first 60 epochs of training. We observe that no entropy regularizer results in our attention becoming very selective early during training which might hinder the exploration of the sampling space.

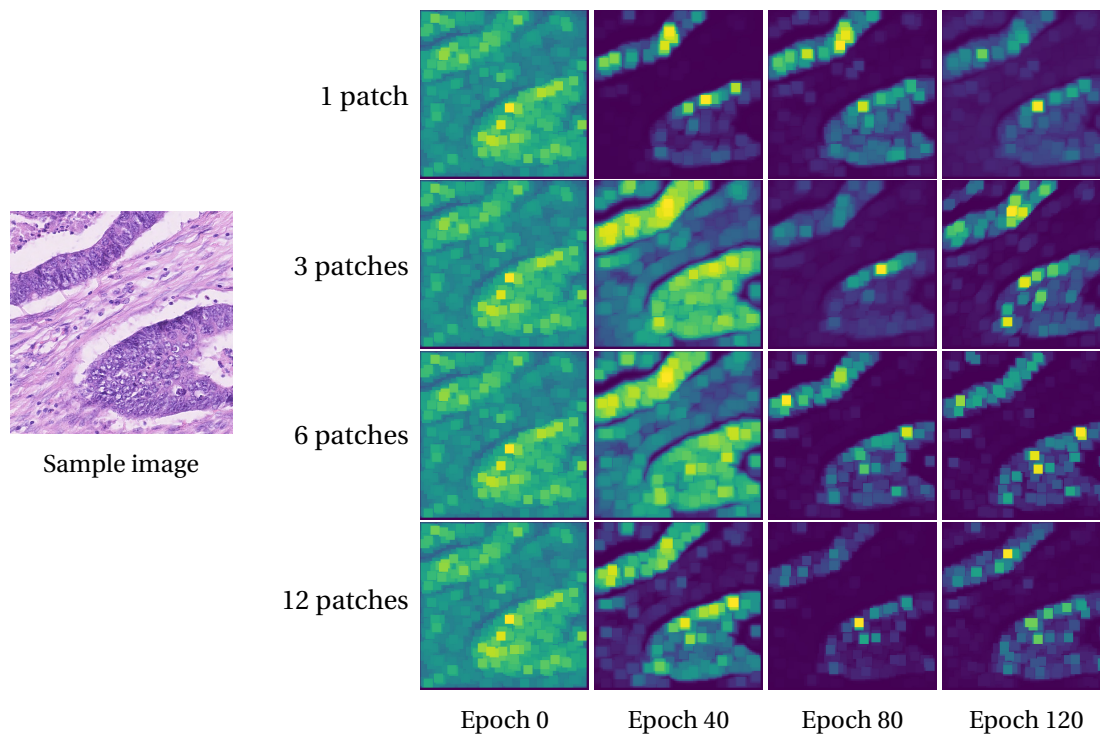


Figure B.2 – Visualization of the attention distribution when training with varying number of patches. All the distributions converge to approximately the same after ~100 epochs.

## B.7 Qualitative results of the learned attention distribution

In this section, we provide additional visualizations of the learned attention distribution using both *attention sampling* and *Deep MIL* on our two real world datasets, namely the Histopathology images § 3.4.3 and the Speed limits § 3.4.4.

### B.7.1 Histopathology images

In Figure B.3 we visualize the learned attention distribution of *attention sampling* and we compare it to *Deep MIL* and the ground truth positions of epithelial cells in a subset of the test set.

We observe that the learned attention distribution is very similar to the one learned by *Deep MIL* even though our model processes a fraction of the image at any iteration. In addition, it is interesting to note that the two methods produce distributions that agree even on mistakenly tagged patches, one such case is depicted in figures 11 and 12 where both methods the top right part of the image to contain useful patches.

## B.7. Qualitative results of the learned attention distribution

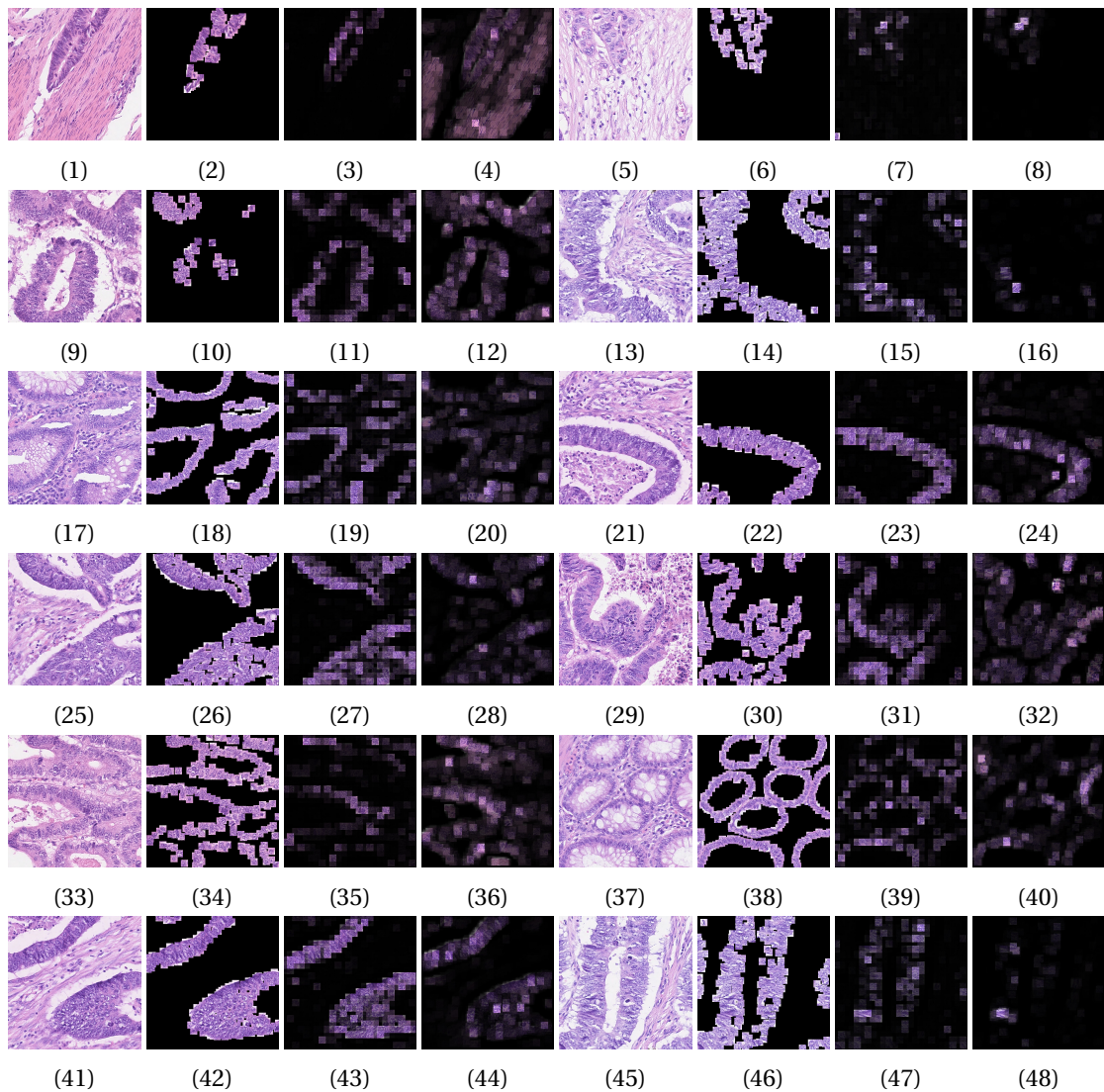


Figure B.3 – We visualize in groups of 4, the H&E stained image, the ground truth positions of epithelial cells, the attention distribution of *Deep MIL* and the attention distribution of *attention sampling*. We observe that indeed our method learns to identify regions of interest without per patch annotations in a similar fashion to *Deep MIL*.

### B.7.2 Speed limits

Figure B.4 compares the attention distributions of *Deep MIL* and *attention sampling* on the Speed Limits dataset (§ 3.4.4). This dataset is hard because it presents large variations in scale and orientation of the regions of interest, namely the speed limit signs. However, we observe that both methods locate effectively the signs even when there exist more than one in the image. Note that for some of the images, such as 6 and 15, the sign is not readable from the low resolution image.

## B.8 Network Architecture Details

In this section, we detail the network architectures used throughout our experimental evaluation. The ultimate detail is always code, thus we encourage the reader to refer to the github repository <https://github.com/idiap/attention-sampling>.

### B.8.1 Megapixel MNIST

We summarize the details of the architectures used for the current experiment. For ATS, we use a three layer convolutional network with 8 channels followed by a ReLU activation as the attention network and a convolutional network inspired from LeNet-1 (LeCun et al., 1995) with 32 channels and a global max-pooling as a last layer as the feature network. We also use an entropy regularizer with weight 0.01. The CNN baseline is a ResNet-16 that starts with 32 channels for convolutions and doubles them after every two residual blocks.

We train all the networks with the Adam (Kingma and Ba, 2014) optimizer with a fixed learning rate of  $10^{-3}$  for 500 epochs.

### B.8.2 Histopathology images

We summarize the details of the architecture used for the experiment on the H&E stained images. For ATS, we use a three layer convolutional network with 8 channels followed by ReLU non linearities as the attention network with an entropy regularizer weight 0.01. The feature network of is the same as the one proposed by (Ilse et al., 2018). Regarding, the CNN baseline, we use a ResNet (He et al., 2016) with 8 convolutional layers and 32 channels instead.

We train all the networks for 30,000 gradient updates with the Adam optimizer with learning rate  $10^{-3}$ .

### B.8.3 Speed Limits

We detail the network architectures used for the current experiment. For *attention sampling*, we use an attention network that consists of four convolutions followed by ReLU



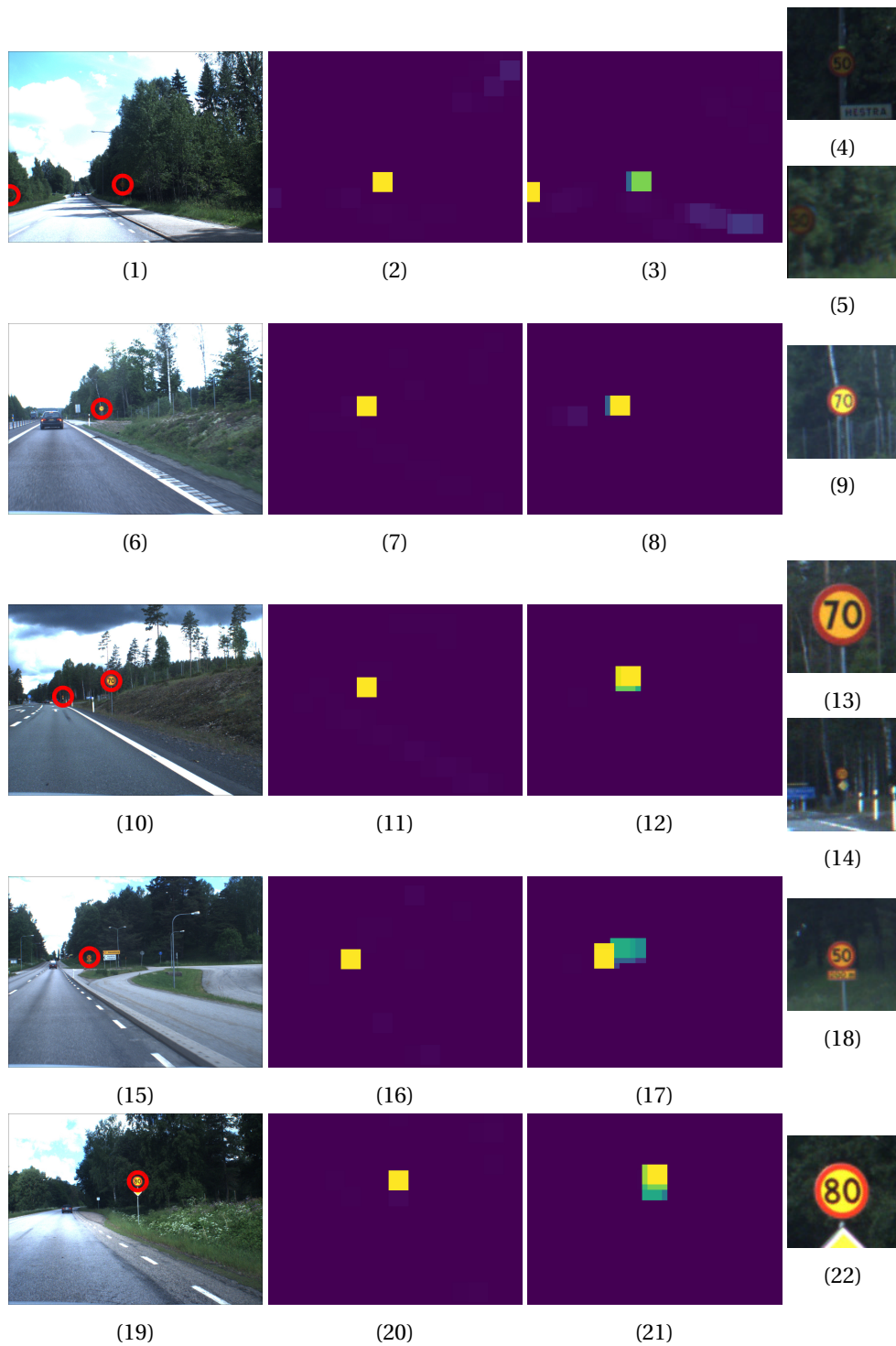


Figure B.4 – Visualization of the positions of the speed limit signs in test images of the dataset as well as the two attention distributions of *Deep MIL* (left) and *attention sampling* (right) and the patches extracted from the high resolution image at the positions of the signs. Both methods identify effectively the speed limit in the high resolution image.

## Appendix B. Appendix for Chapter 3

---

non-linearities starting with 8 channels and doubling them after each layer. Furthermore, we add a max pooling layer with pool size 8 at the end to reduce the sampling space and use an entropy regularizer weight of 0.05. The feature network of both our model and *Deep MIL* is a ResNet with 8 layers and 32 channels. The CNN baseline is a ResNet-16 that starts with 32 channels for convolutions and doubles them after every two residual blocks.

Again, we use the Adam ([Kingma and Ba, 2014](#)) optimizer with a fixed learning rate of  $10^{-3}$  for 300,000 iterations.

# C Appendix for Chapter 4

## C.1 Gradient Derivation

In the first section of our supplementary material, we derive in detail the gradients for causally masked linear transformers and show that they can be computed in linear time and constant memory. In particular, we derive the gradients of a scalar loss with respect to the numerator of the following equation,

$$V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)}. \quad (\text{C.1})$$

The gradient with respect to the denominator and the fraction are efficiently handled by autograd. Without loss of generality, we can assume that  $Q$  and  $K$  already contain the vectors mapped by  $\phi(\cdot)$ , hence given the numerator

$$\bar{V}_i = Q_i^T \sum_{j=1}^i K_j V_j^T, \quad (\text{C.2})$$

and  $\nabla_{\bar{V}} \mathcal{L}$  we seek to compute  $\nabla_Q \mathcal{L}$ ,  $\nabla_K \mathcal{L}$  and  $\nabla_V \mathcal{L}$ . Note that  $Q \in \mathbb{R}^{N \times D}$ ,  $K \in \mathbb{R}^{N \times D}$  and  $V \in \mathbb{R}^{N \times M}$ . To derive the gradients, we first express the above equation for a single element without using vector notation,

$$\bar{V}_{ie} = \sum_{d=1}^D Q_{id} \sum_{j=1}^i K_{jd} V_{je} = \sum_{d=1}^D \sum_{j=1}^i Q_{id} K_{jd} V_{je}. \quad (\text{C.3})$$

Subsequently we can start deriving the gradients for  $Q$  by taking the partial derivative for any  $Q_{lt}$ , as follows

$$\frac{\partial \mathcal{L}}{\partial Q_{lt}} = \sum_{e=1}^M \frac{\partial \mathcal{L}}{\partial \bar{V}_{le}} \frac{\partial \bar{V}_{le}}{\partial Q_{lt}} = \sum_{e=1}^M \frac{\partial \mathcal{L}}{\partial \bar{V}_{le}} \left( \sum_{j=1}^l K_{jt} V_{je} \right). \quad (\text{C.4})$$

If we write the above equation as a matrix product of gradients it becomes,

$$\nabla_{Q_i} \mathcal{L} = \nabla_{\bar{V}_i} \mathcal{L} \left( \sum_{j=1}^i K_j V_j^T \right)^T, \quad (\text{C.5})$$

proving equation 4.13. In equation C.4 we made use of the fact that  $Q_{lt}$  only affects  $\bar{V}_l$  hence we do not need to sum over  $i$  to compute the gradients. However, for  $K$  and  $V$  this is not the case. In particular,  $K_j$  affects all  $\bar{V}_i$  where  $i \geq j$ . Consequently, we can write the partial derivative of the loss with respect to  $K_{lt}$  as follows,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial K_{lt}} &= \sum_{e=1}^M \sum_{i=l}^N \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} \frac{\partial \bar{V}_{ie}}{\partial K_{lt}} = \sum_{e=1}^M \sum_{i=l}^N \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} \frac{\partial \left( \sum_{d=1}^D \sum_{j=1}^i Q_{id} K_{jd} V_{je} \right)}{\partial K_{lt}} \\ &= \sum_{e=1}^M \sum_{i=l}^N \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} Q_{it} V_{le}. \end{aligned} \quad (\text{C.6})$$

As for  $Q$  we can now write the gradient in vectorized form,

$$\nabla_{K_i} \mathcal{L} = \left( \sum_{j=i}^N Q_j \left( \nabla_{\bar{V}_j} \mathcal{L} \right)^T \right) V_i, \quad (\text{C.7})$$

proving equation 4.14. Following the same reasoning, we can compute the partial derivative of the loss with respect to  $V_{lt}$  and prove equation 4.15. Note that the cumulative sum matrices for the gradient with respect to  $Q$  and  $K$  have the same size, however one is computed in the forward direction (summing from 1 to  $N$ ) similarly to the forward pass and the other is computed in the backwards direction (summing from  $N$  to 1) similar to backpropagation through time done in RNNs.

## C.2 Training Evolution

In figure C.1 we present the training evolution of all transformer models in our experiments. For the MNIST experiment (Fig. C.1a) we train all methods for 250 epochs. The sequence length is small enough so that the training time does not vary significantly for all methods. We observe that our method converges on par with softmax attention outperforming significantly both reformer variants.

On the other hand, for CIFAR-10 (Fig. C.1b) we train all methods for a fixed amount of time, namely 7 days. We observe that *lsh-1* and *linear* complete significantly more epochs than softmax and *lsh-4* and achieve better performance. This gap is expected to increase with a further increase in sequence length.

Finally, in our last experiment on automatic speech recognition (Fig. C.1c), softmax outperforms significantly both Reformer and linear in terms of convergence. Note that linear is  $3\times$  faster per epoch which means it has completed approximately 4 times more epochs in

### C.3. Image Generation Throughput Discussion

comparison to softmax. Even though softmax attention is better in this task, we observe that *linear transformers* significantly outperform Reformer both in terms of convergence and final performance.

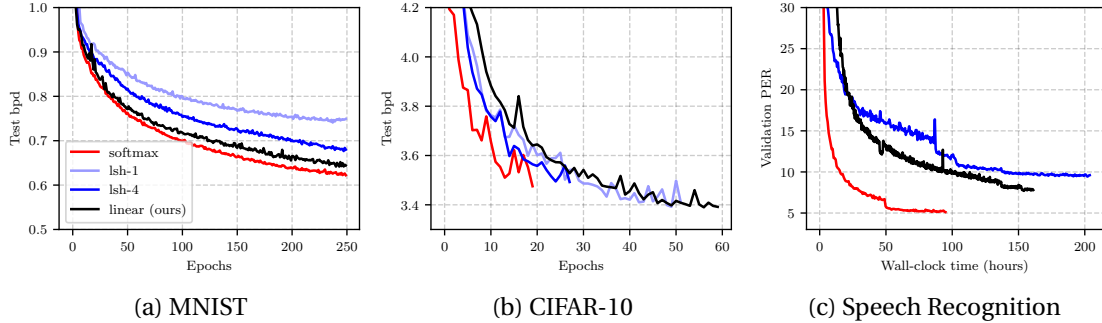


Figure C.1 – Training evolution of transformers for all our experiments. It can be observed that *linear transformers* converge consistently faster than Reformer and in the autoregressive experiments on par with softmax. For MNIST all methods are trained for 250 epochs while for CIFAR we train for 7 days. In the speech recognition experiments all methods are trained to convergence. The details of the experiments can be found in § 4.4.2 and § 4.4.3.

## C.3 Image Generation Throughput Discussion

### C.3.1 Stateful softmax attention

In § 4.4.2, we report the image generation throughput and we compare with **softmax** transformer and **lsh**. In this section we create another baseline, denoted as **stateful-softmax**, that implements a softmax autoregressive transformer as a recurrent model. Namely, all the keys and values are saved and then passed to the model again when predicting the next element of the sequence. The state of this recurrent model is the set of keys and values which has size proportional to the sequence length. This is qualitatively different to our proposed model that has a state with fixed dimensions and computing the  $i$ -th state given the previous one has fixed computational cost regardless of  $i$ .

Table C.1 summarizes the results. We observe that stateful-softmax is significantly faster than vanilla transformers. However, its complexity is still quadratic with respect to the sequence length and our formulation is more than  $50\times$  faster for CIFAR-10. Moreover, we would like to point out that implementing a similar stateful attention for Reformer is not a trivial task as the sorting and chunking operations need to be performed each time a new input is provided.

### C.3.2 Equalizing the batch size

In the previous sections we evaluate the throughput of all transformer variants for the task of autoregressive image generation. However, another important factor to consider is latency, namely the total time required to produce a single image. To this end, we use a batch size of

## Appendix C. Appendix for Chapter 4

Method	Bits/dim	Images/sec	Method	Bits/dim	Images/sec
Softmax	0.621	0.45 (1×)	Softmax	3.47	0.004 (1×)
Stateful-softmax	0.621	7.56 (16.8×)	Stateful-softmax	3.47	0.32 (80×)
LSH-1	0.745	0.68 (1.5×)	LSH-1	3.39	0.015 (3.75×)
LSH-4	0.676	0.27 (0.6×)	LSH-4	3.51	0.005 (1.25×)
Linear (ours)	0.644	<b>142.8 (317×)</b>	Linear (ours)	3.40	<b>17.85 (4,462×)</b>

(a) Image generation on MNIST

(b) Image generation on CIFAR-10

Table C.1 – Comparison of autoregressive image generation throughput of MNIST and CIFAR-10 images. The experiment can be found in § 4.4.2. For stateful-softmax we save the keys and values and reuse them for predicting the next element. A detailed description of this extra baseline can be found in § C.3.1.

1 and measure the time required by all methods to generate a single image. In addition to running the inference on the GPU, we also evaluate the time required on CPU. The results are reported in table C.2.

Method	Seconds (CPU)	Seconds (GPU)	Method	Seconds (CPU)	Seconds (GPU)
Softmax	72.6 (13.2×)	10.2 (1.4×)	Softmax	8651.4 (191.8×)	300.1 (4.9×)
Stateful-softmax	7.4 (1.3×)	10.4 (1.42×)	Stateful-softmax	71.9 (1.6×)	70.4 (1.14×)
LSH-1	46.0 (8.3×)	19.2 (2.6×)	LSH-1	2318.9 (51.4×)	221.6 (3.6×)
LSH-4	112.0 (20×)	55.8 (7.6×)	LSH-4	5263.7 (116.7×)	683.9 (11.1×)
Linear (ours)	<b>5.5 (1×)</b>	<b>7.3 (1×)</b>	Linear (ours)	<b>45.1 (1×)</b>	<b>61.3 (1×)</b>

(a) Image generation on MNIST

(b) Image generation on CIFAR-10

Table C.2 – Comparison of the time required to generate a single image with autoregressive transformers on MNIST and CIFAR-10. We run all methods with a batch size of 1 both on CPU and GPU and report the total time in seconds. For all numbers in the table, lower is better.

We observe that all methods underutilize the GPU and achieve significantly smaller image generation throughput than the one shown in table C.1. The proposed linear transformer is faster than all the methods and in particular it is almost 6.6× faster than softmax transformers for generating an image on CIFAR-10. Note that our linear autoregressive transformer is the only method that is faster on the CPU than on the GPU in every case. This is due to the fact that computing the attention as an RNN has such a low cost that the main computational bottleneck becomes the inevitable outer loop over the sequence.

## C.4 Qualitative Results on Image Generation

In this section we provide qualitative results for our image generation experiments. Since the perplexity of all models is approximately the same, as expected, the qualitative differences are not significant. A rather interesting observation however is that the Reformer models provide significantly fewer variations in their unconditional samples. Moreover, we observe that image

#### C.4. Qualitative Results on Image Generation

completion is a significantly easier task than unconditional generation as all models perform significantly better.

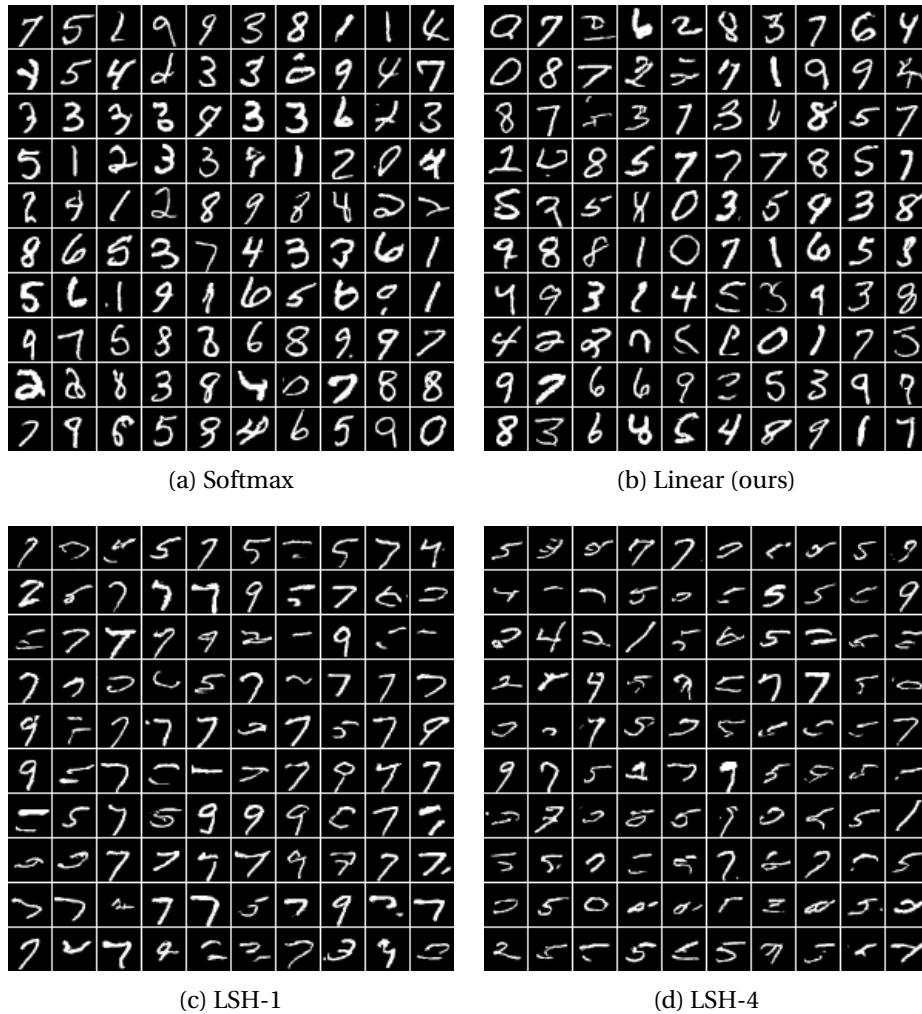
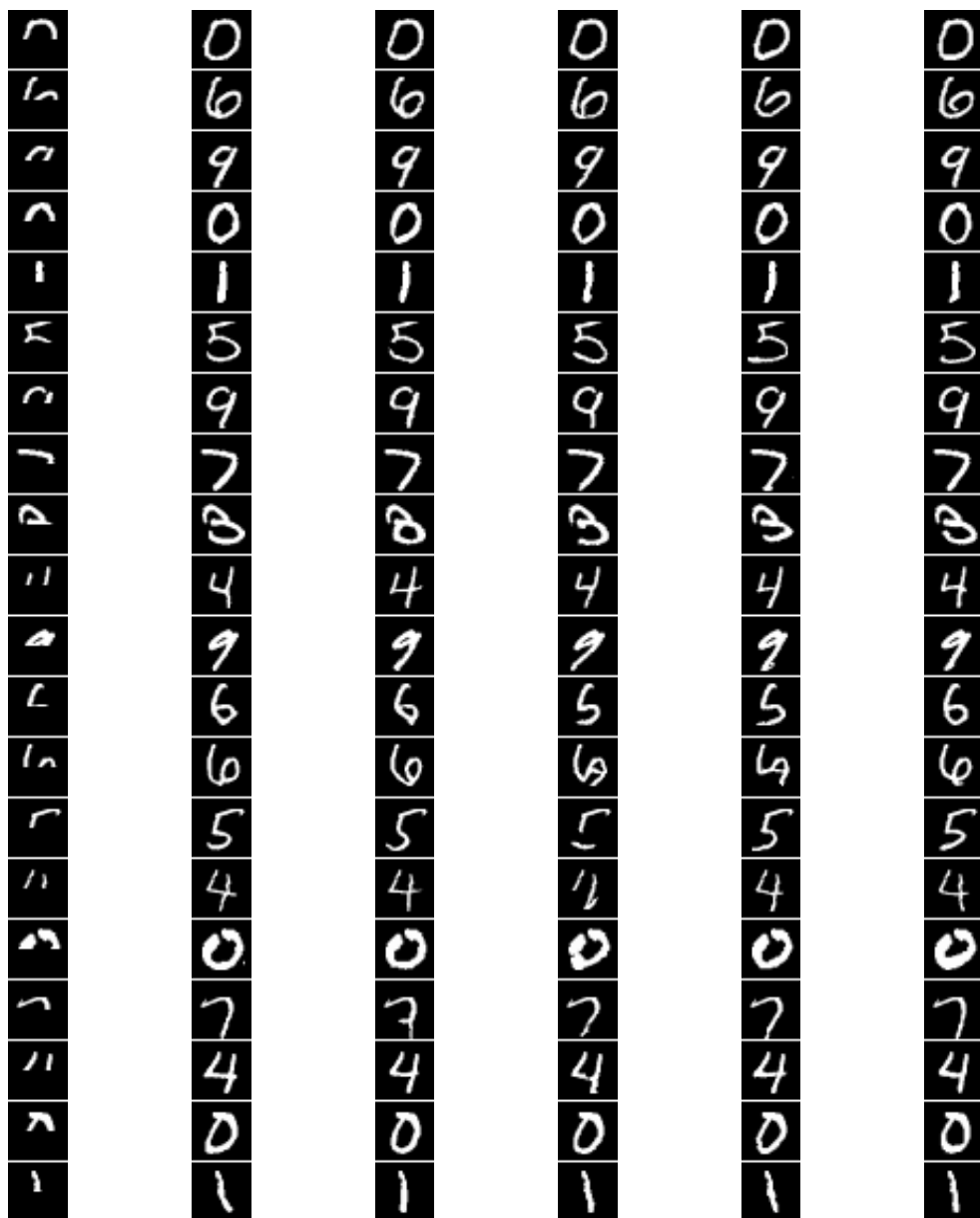


Figure C.2 – Unconditional samples from the transformer models trained with MNIST. See § 4.4.2 for more details.



(a) Occluded (b) Softmax (c) Linear (ours) (d) LSH-1 (e) LSH-4 (f) Original

Figure C.3 – MNIST digit completion from all trained models. See § 4.4.2 for more details.



## C.4. Qualitative Results on Image Generation



Figure C.4 – Unconditional samples from the transformer models trained with CIFAR-10. See § 4.4.2 for more details.



# D Appendix for Chapter 5

## D.1 Scaling Attention with Fast Clustering

### D.1.1 Clustered attention

In figure D.1, we present the steps involved in *clustered* attention computation for an example sequence with 8 queries and the number of clusters set to 3. We first cluster the queries  $Q$  using K-means to output  $S$  which indicates the membership of queries to different clusters. We use different colors to represent different clusters. After clustering, the centroids  $Q^c$  are used to compute the attention weights  $A^c$  and the new values  $V^c$  for the centroids. Finally, the values are broadcasted to get the new values  $\hat{V}$  corresponding to each query.

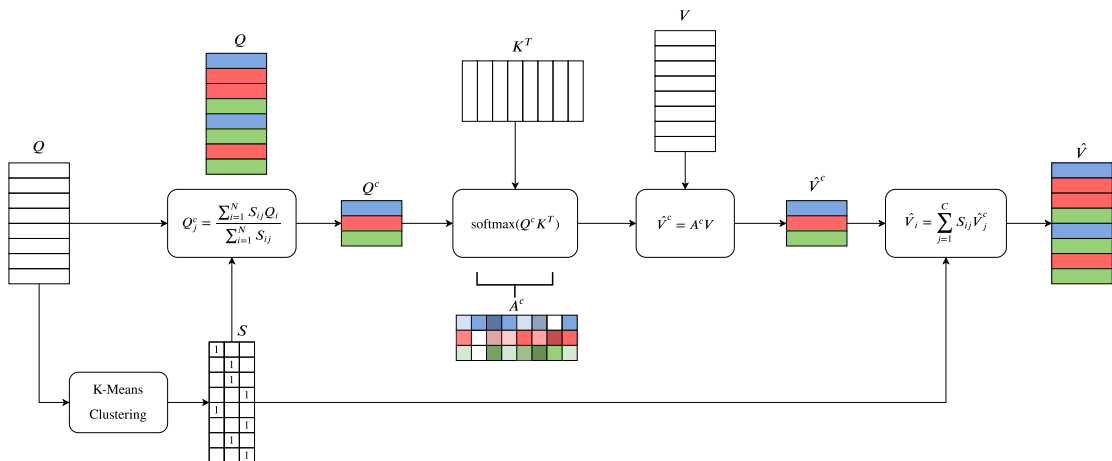


Figure D.1 – Flow-chart demonstrating the computation for *clustered* attention. We use different colors to represent the query groups and the computed centroids. The same colors are then used to show the attention weights  $A^c$ , new values for the centroids  $\hat{V}^c$ , and the resulting values  $\hat{V}$  after broadcasting.

### D.1.2 Improved clustered attention

In this section, we first describe how we can efficiently compute *i-clustered* attention using sparse dot products with the top- $k$  keys and values. We then present a flow chart demonstrating the same in Figure D.2.

As discussed in § 5.3.3, the improved attention matrix approximation  $A_i^t$  for the query,  $Q_i$  belonging to the cluster  $j$  is computed as follows:

$$A_{il}^t = \begin{cases} \frac{\hat{m}_j \exp(Q_i K_l^T)}{\sum_{r=1}^N T_{jr} \exp(Q_i K_r^T)} & \text{if } T_{jl} = 1 \\ A_{il}^c & \text{otherwise} \end{cases}, \quad (\text{D.1})$$

where,  $T \in \{0, 1\}^{C \times N}$ , stores the top- $k$  keys for each cluster.  $T_{ji} = 1$  if the  $i$ -th key is among the top- $k$  keys for the  $j$ -th cluster and 0 otherwise. In addition,  $\hat{m}_j$  is the total probability mass on the top- $k$  keys for the  $j$ -th cluster given by:

$$\hat{m}_j = \sum_{r=1}^N T_{jr} A_{jr}^c. \quad (\text{D.2})$$

Note that we can compute the attention weights  $A_i^t$  on the top- $k$  keys by first taking sparse dot-product of  $Q_i$  with the top- $k$  keys followed by the softmax activation and rescaling with the total probability mass  $m_j$ . For the rest of the keys, the attention weight is the clustered-attention weight  $A_i^c$ .

Similarly, the new values  $\hat{V}_i$  can be decomposed into the following two terms,

$$\hat{V}_i = \hat{V}_i^t + \hat{V}_i^b, \quad (\text{D.3})$$

where  $\hat{V}_i^t$  is the weighted average of the values corresponding to the top- $k$  keys with weights being the improved attention on the top- $k$  keys. On the other hand,  $\hat{V}_i^b$  is the weighted average of the rest of the values with weights being the clustered attention  $A_i^c$ . The following equations show how we compute  $\hat{V}_i^t$  and  $\hat{V}_i^b$ ,

$$\hat{V}_i^t = \sum_{l=1}^N T_{jl} A_{il}^t V_l, \quad (\text{D.4})$$

$$\hat{V}_i^b = \sum_{l=1}^N (1 - T_{jl}) A_{il}^c V_l, \quad (\text{D.5})$$

Note that  $\hat{V}_i^t$  is weighted average of  $k$  values for each query and thus requires  $\mathcal{O}(NkD_v)$  operations.  $\hat{V}_i^b$  only needs to be computed once per-cluster centroid and thus requires  $\mathcal{O}(NCD_v)$  operations.

In figure D.2 we present the *i-clustered* attention computation for the same example sequence with 8 queries and the number of clusters and top- $k$  keys set to 3. The lower half of the

figure shows the new value  $\hat{V}^t$ , computed by first taking sparse dot-products with the top 3 keys to get the attention weights. This is followed by taking the weighted average of the 3 corresponding values. The top half of the figure shows the  $\hat{V}^b$  computation. This is same as clustered attention computation but with attention weights corresponding to top 3 keys set to 0 for  $A^c$ . The resulting values  $\hat{V}$  is the sum of  $\hat{V}^b$  and  $\hat{V}^t$ .

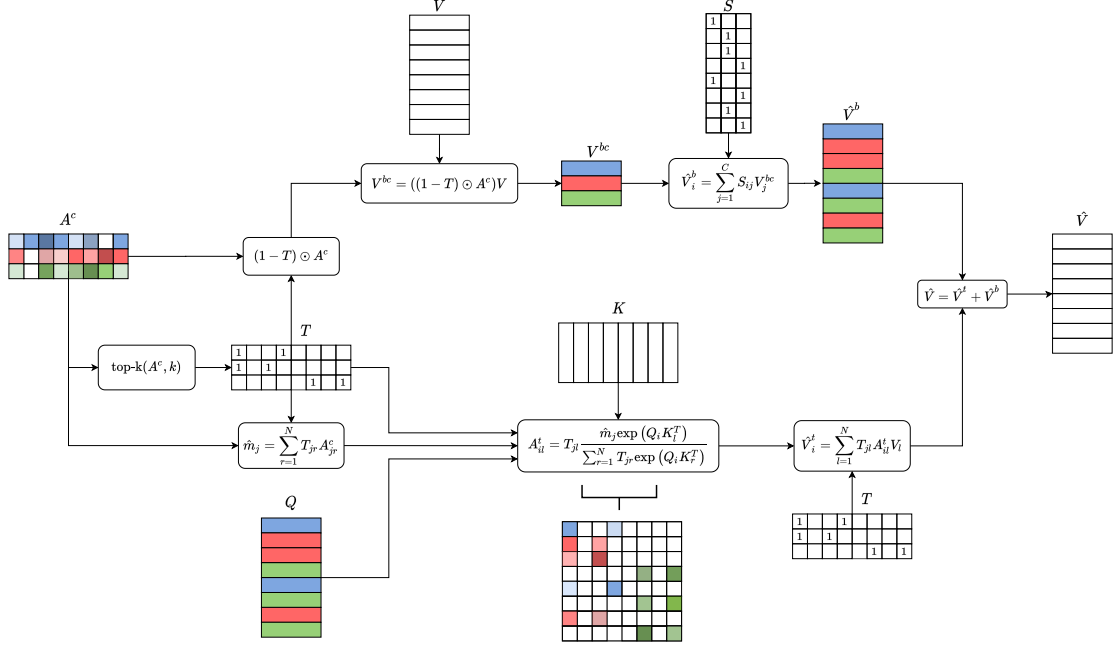


Figure D.2 – Flow-chart demonstrating the computation for  $i$ -clustered attention. The lower half of the figure shows the new value  $\hat{V}^t$ , computed by sparse dot-products with the keys  $K$  and values  $V$  corresponding to the the top- $k$  keys in  $T$ . The top half of the figure shows the computation for  $\hat{V}^b$  which is the weighted average of the rest of the values with weights coming from the clustered attention  $A^c$ . The resulting values  $\hat{V}$  is the sum of  $\hat{V}^b$  and  $\hat{V}^t$ . Further details are provided in § 5.3.3 and § D.1.2.

## D.2 Quality of the approximation

**Proposition 3.** For the  $i$ -th query belonging to the  $j$ -th cluster, the improved clustered attention  $A_i^t$  and clustered attention  $A_j^c$  relate to the full attention  $A_i$  as follows,

$$\|A_i^t - A_i\|_1 \leq \|A_j^c - A_i\|_1 \quad (\text{D.6})$$

*Proof.* As discussed before, the improved attention matrix approximation  $A_i^t$  for the query,  $Q_i$  is computed as follows:

$$A_{il}^t = \begin{cases} \frac{\hat{m}_j \exp(Q_i K_l^T)}{\sum_{r=1}^N T_{jr} \exp(Q_i K_r^T)} & \text{if } T_{jl} = 1 \\ A_{il}^c & \text{otherwise} \end{cases}, \quad (\text{D.7})$$

## Appendix D. Appendix for Chapter 5

---

where,  $T \in \{0, 1\}^{C \times N}$ , stores the top- $k$  keys for each cluster,  $T_{ji} = 1$  if the  $i$ -th key is among the top- $k$  keys for the  $j$ -th cluster and 0 otherwise.  $\hat{m}_j$  is the total probability mass on the top- $k$  keys for the  $j$ -th cluster, computed as follows:

$$\hat{m}_j = \sum_{r=1}^N T_{jr} A_{jr}^c. \quad (\text{D.8})$$

Given the full attention  $A_i$ , equation D.7 can be simplified to

$$A_{il}^t = \begin{cases} \frac{\hat{m}_j}{m_i} A_{il} & \text{if } T_{jl} = 1 \\ A_{il}^c & \text{otherwise} \end{cases}, \quad (\text{D.9})$$

where,  $m_i$  is the total probability mass on the same top- $k$  keys for the  $i$ -th query, computed using the true attention  $A_i$ , as follows:

$$m_i = \frac{\sum_{r=1}^N T_{jr} \exp(Q_i K_r^T)}{\sum_{r=1}^N \exp(Q_i K_r^T)} \quad (\text{D.10})$$

$$= \sum_{r=1}^N T_{jr} A_{ir}. \quad (\text{D.11})$$

Without loss of generality, let us assume,  $T_{jl} = 1 \quad \forall \quad l \in \{1, \dots, k\}$  and  $T_{jl} = 0 \quad \forall \quad l \in \{k+1, \dots, N\}$ .

In this case, equation D.9 can be written as:

$$A_{il}^t = \begin{cases} \frac{\hat{m}_j}{m_i} A_{il} & \text{if } l \leq k \\ A_{il}^c & \text{if } l \geq k+1 \end{cases}. \quad (\text{D.12})$$

The total probability masses on the top- $k$  keys,  $m_i$  and  $\hat{m}_j$  can now be expressed as:

$$m_i = \sum_{r=1}^k A_{ir}. \quad (\text{D.13})$$

$$\hat{m}_j = \sum_{r=1}^k A_{jr}^c. \quad (\text{D.14})$$

From equation D.12 it is clear that the clustered attention,  $A_i^c$ , and the improved clustered attention,  $A_i^t$ , only differ on the keys  $\{1, \dots, k\}$ . Thus, it suffices to show that  $A_i^t$  has lower approximation error on these keys. The approximation error on the top- $k$  keys  $\{1, \dots, k\}$ , let it be  $e_t$ , between the  $i$ -clustered attention and the full attention is as follows:

$$e_t = \sum_{l=1}^k |A_{il} - A_{il}^t| \quad (\text{D.15})$$

$$= \sum_{l=1}^k \left| A_{il} - A_{il} \frac{\hat{m}_j}{m_i} \right| \quad (\text{D.16})$$

$$= \sum_{l=1}^k A_{il} \left| 1 - \frac{\hat{m}_j}{m_i} \right| \quad (\text{D.17})$$

$$= \left| 1 - \frac{\hat{m}_j}{m_i} \right| \sum_{l=1}^k A_{il} \quad (\text{D.18})$$

$$= m_i \left| 1 - \frac{\hat{m}_j}{m_i} \right| \quad (\text{D.19})$$

$$= |m_i - \hat{m}_j| \quad (\text{D.20})$$

$$= \left| \sum_{l=1}^k A_{il} - A_{jl}^c \right| \quad (\text{D.21})$$

$$\leq \sum_{l=1}^k |A_{il} - A_{jl}^c| \quad (\text{D.22})$$

Therefore,

$$\|A_i - A_i^t\|_1 = \sum_{l=1}^k |A_{il} - A_{il}^t| + \sum_{l=k+1}^N |A_{il} - A_{il}^t| \quad (\text{D.23})$$

$$= \sum_{l=1}^k |A_{il} - A_{il}^t| + \sum_{l=k+1}^N |A_{il} - A_{jl}^c| \quad (\text{D.24})$$

$$\leq \sum_{l=1}^k |A_{il} - A_{jl}^c| + \sum_{l=k+1}^N |A_{il} - A_{jl}^c| \quad (\text{D.25})$$

$$\leq \|A_i - A_i^c\|_1 \quad (\text{D.26})$$

□

## D.3 Experiments

### D.3.1 Time and Memory Benchmark

To measure the computational cost, we compare the memory consumption and computation time on artificially generated sequences of various lengths. For clustered attention we use 100 clusters, 63 bits for the LSH, and 10 Lloyd iterations for the K-Means. For the improved clustered attention, we use the same configuration with  $k = 32$ . For Reformer, we evaluate on two variants using 1 and 4 rounds of hashing. All models consist of 1 layer with 6 attention heads, embedding dimension of 64 for each head, and a feed-forward dimension of 1536.

## Appendix D. Appendix for Chapter 5

In this experiment, we measure the required memory and GPU time *per single sequence element* to perform a forward/backward pass for the various self-attention models. Figure D.3 illustrates how these metrics evolve as the sequence length increases from  $N = 2^9$  to  $N = 2^{15}$ . For a fair comparison, we use the maximum possible batch size for each method and we divide the computational cost and memory with the number of samples in each batch and the sequence length.

We note that, in contrast to all other methods, vanilla transformer scales quadratically with respect to the sequence length and does not fit in GPU memory for sequences longer than  $2^{13}$  elements. All other methods scale linearly. Clustered attention becomes faster than the vanilla transformer for sequences with 1000 elements or more, while improved clustered attention surpasses it for sequences with 2000 elements. Note that with respect to per sample memory, both clustered and improved clustered attention perform better than all other methods. This can be explained by the fact that our method does not require storing intermediate results to compute the gradients from multiple hashing rounds as Reformer does. It can be seen, that lsh-1 is faster than the improved clustered attention, however, as also mentioned by (Kitaev et al., 2020) Reformer requires multiple hashing rounds to generalize.

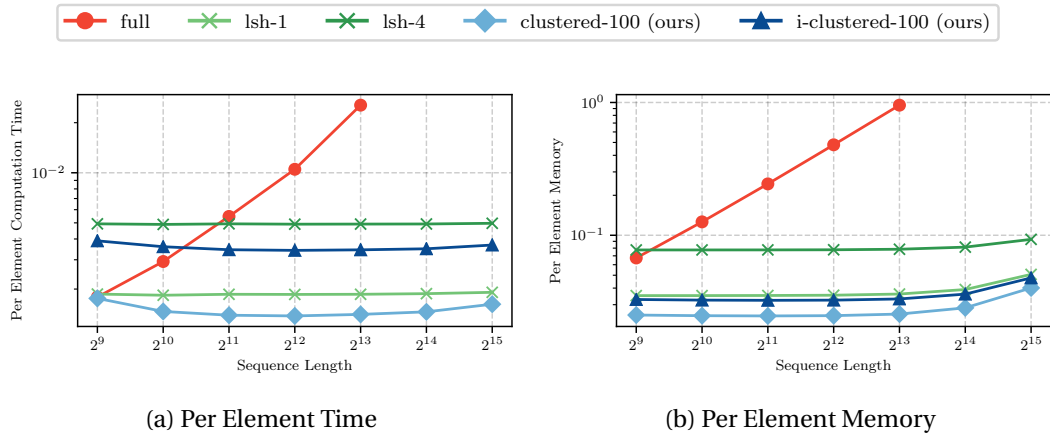


Figure D.3 – Per element GPU time and memory consumption for a forward/backward pass. All models, except full, scale linearly with respect to the sequence length since they have constant time and memory per element.

### D.3.2 Ablation on clusters and sequence length

Following (Kitaev et al., 2020), we introduce a synthetic task to analyze the relationship between the number of clusters and sequence length. In our task, the transformer models need to copy some symbols that are masked out from either the first or second half of the sequence. In particular, we generate a random sequence of tokens and we prepend a unique separator token, let it be 0. The sequence is then copied to get a target of the form  $0w0w$ , where  $w \in \{1, \dots, C\}^L$ ,  $C$  is the number of possible symbols and  $L$  is the sequence length. To



Accuracy with respect to clusters and hashing rounds

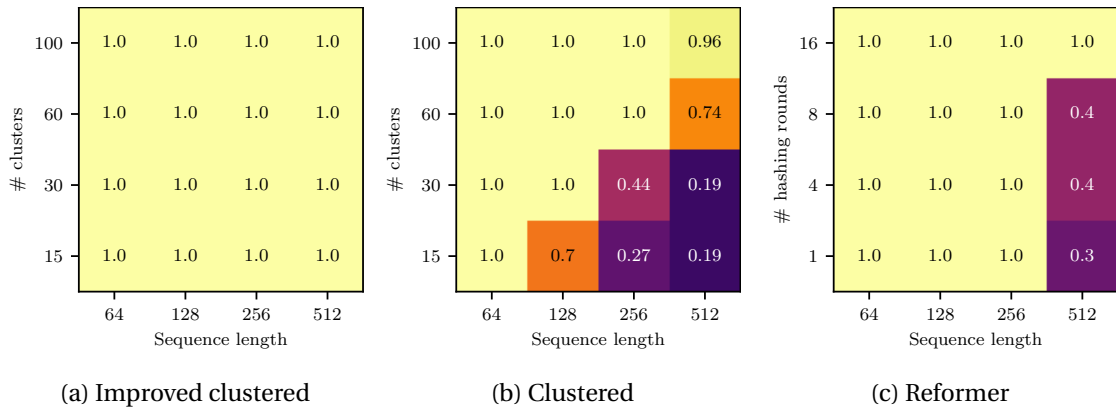


Figure D.4 – The heatmaps depict the achieved accuracy on an artificial copy task (§ D.3.2) as the sequence length, the number of clusters and the number of hashing rounds varies. Improved clustered (D.4a) is the only fast transformer variant that can solve the task perfectly for any sequence length and number of clusters combination.

generate the input, we replace some symbols from the first half of the sequence and some different symbols from the second half, such that the target sequence can be reconstructed from the input. An example of an input output pair with  $L = 4$  can be seen in figure D.5. Note that to solve this task, transformers simply need to learn to attend to the corresponding tokens in the two identical halves of the sequence.

<b>Input</b>	0	4	M	2	2	0	4	5	M	2
<b>Output</b>	0	4	5	2	2	0	4	5	2	2

Figure D.5 – Example of an input and output pair for the masked copy task. M denotes the masked out tokens.

We set the sequence length  $L$  to one of  $\{31, 63, 127, 255\}$  which means the input length varies between  $N = 2^6$  and  $N = 2^9$ . For each sequence, we sample tokens uniformly from  $\{1, \dots, 10\}$  and randomly mask out 20% of the tokens. To analyze the impact of number of clusters on performance, we train full transformer as well as clustered variants with different number of clusters and Reformer with different number of hashing rounds.

All transformer variants consist of 4 layers, 6 attention heads, embedding dimension of 32 for each head, and feed-forward dimension of 768. For both clustered and improved clustered attention, we set the number of bits for LSH to 63 and the number of Lloyd iterations for the K-Means to 10. Both clustered and improved clustered attention are trained with 15, 30, 60 and 100 clusters. We also train Reformer with 1, 4, 8 and 16 hashing rounds. Finally, all models are trained using R-Adam optimizer (Liu et al., 2020a) with a learning rate of 0.0002, batch size of 32 for 5000 iterations.

In figure D.4, we illustrate the results of this experiment as heatmaps depicting the achieved accuracy for a given combination of number of clusters and sequence length for clustered transformers and number of hashing rounds and sequence length for Reformer. Note that the vanilla transformer solves the task perfectly for all sequence lengths. We observe that both clustered (Fig. D.4b) and Reformer (Fig. D.4c) require more clusters or more rounds as the sequence length increases. However, improved clustered achieves the same performance as vanilla transformers, namely *perfect accuracy*, for every number of clusters and sequence length combination. This result increases our confidence that the required number of clusters for our method is not a function of the sequence length but of the task at hand.

### D.3.3 Automatic Speech Recognition

In this section, we present the details for the ASR experiments such as transformer architecture, optimizer and learning rate schedule. As mentioned in § 5.4, for *i-clustered*, unless specified otherwise,  $k$  is set to 32. Furthermore, all transformers have 6 heads with an embedding dimension of 32 for each head and feed-forward dimension of 768. Other architectural details specific to each experiment are described in the corresponding section.

#### Wall Street Journal

##### Convergence Behaviour:

For this experiment, we train a transformer with the full, clustered and Reformer attention variants. All models consist of 9 layers. For Reformer, we train two variants with 1 and 4 rounds of hashing with chunk size fixed to 32 as suggested. For clustered and improved clustered attention we set the number of clusters to 100. We also set the number of Lloyd iterations for K-Means to 10 and the bits for LSH to 63. All models are trained to convergence using the R-Adam optimizer (Liu et al., 2020a) with a learning rate of 0.0001, max gradient norm set to 10.0 and weight decay of 0.01. The learning rate is dropped when the validation loss plateaus. For each model we select the largest batch size that fits the GPU. The *full* attention model was trained with a batch size of 2 while the clustered variants: *clustered* and *i-clustered* could fit batch sizes of 14 and 10 respectively. For Reformer variants: *lsh-1* and *lsh-4*, batch sizes of 8 and 6 were used.

In figure D.6a, we show the training loss convergence for different transformer variants. It can be seen that *i-clustered* has a much faster convergence than the *clustered* attention. This shows that the improved clustered attention indeed approximates the full attention better. More importantly, only the *i-clustered* attention has a comparable wall-clock convergence. Given that *full* has a much smaller batch size, it makes many more updates per-epoch. We think that a slightly smaller batchsize with more updates would have been a better choice for the clustered transformers w.r.t. the wall-clock convergence. This is reflected in the Switchboard experiments where the batchsizes for the clustered variants were smaller due

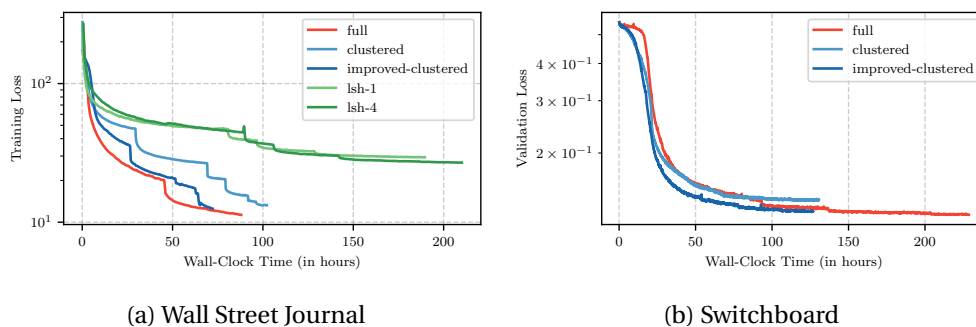


Figure D.6 – We show training/validation loss convergence for different transformer variants. Only *i-clustered* has a faster or comparable wall-clock convergence to full attention. Both the clustered variants are have a significantly better convergence than both *lsh-1* and *lsh-4*. Note that due to a smaller batch size *full* makes many more updates than all other transformer variants. More details can be found in § D.3.3 and § D.3.3.

to more layers. Finally, as it can be seen from the wall-clock convergence, the clustered transformers significantly outperform Reformer.

#### Speed-Accuracy Tradeoff:

As described in § 5.4.1, for this task we additionally train *full* with 4 and 6 layers. Similarly, we train *clustered* with 9 layers, and 200 and 300 clusters. We also train an *i-clustered* model with 9 layer and 200 clusters, and smaller models with 6 layers, and 100 and 200 clusters.

For *clustered* and *i-clustered* variants with 9 layers, we finetuned the previously described models trained with 100 clusters. We finetuned for 15 epochs with a learning rate of 0.00001. We train *full* with 4 and 6 layers to convergence in a similar fashion to the *full* with 9 layers described previously. Finally, for *i-clustered*, we first trained a model with 6 layers and 100 clusters using the training strategy used for 9 layers and 100 clusters. We then finetuned this model for 15 epochs using 200 clusters and a learning rate of 0.00001.

#### Switchboard

##### Convergence Behaviour:

For this experiment, we train a transformer with the full and clustered attention variants. All models consist of 12 layers. For the clustered and improved clustered attention we set the number of clusters to 100. We also set the number of Lloyd iterations for K-Means to 10 and the bits for LSH to 63.

Following common practice for flat-start lattice-free MMI training, we train over multiple gpus with weight averaging for synchronization as described in (Povey et al., 2015). Specifically, we modify the *e2e* training recipe for the **Wall Street Journal** in Kaldi (Povey et al., 2011) with the

following two key differences: first, the acoustic model training is done in PyTorch and second, we use R-Adam optimizer instead on natural stochastic gradient descent.

All models are trained using the R-Adam optimizer with a learning rate of 0.0002, max gradient norm set to 10.0 and weight decay of 0.01. The learning rate is dropped when the validation loss plateaus. We use the word error rate (WER) on the validation set for early stopping and model selection. The *full* attention model is trained with a batch size of 2 while the clustered variants: *clustered* and *i-clustered* are trained with a batch size of 6.

In figure D.6b, we show the training loss convergence for different transformer variants. It can be seen that *i-clustered* has the fastest convergence for this setup. Note that the overall training time for *clustered* attention is still less than that of *full* as it starts to overfit early on the validation set WER.

### Speed-Accuracy Tradeoff:

For this task we additionally train *full* with 6 and 8 layers. Similarly, we train *clustered* with 12 layers, and 200 and 300 clusters. We also train *i-clustered* with 12 layer and 200 clusters, and smaller models with 8 layers, and 100 and 200 clusters.

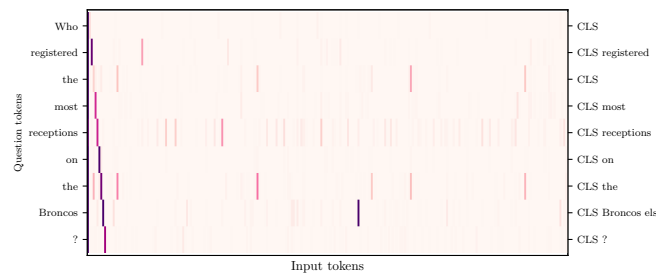
For the *clustered* and *i-clustered* variants with 12 layers, we finetuned the previously described models trained with 100 clusters. We finetuned for 5 epochs with a learning rate of 0.00001. Once again, *full* with 6 and 8 layers were trained to convergence similar to *full* with 12 layers described previously. Finally, for *i-clustered* with 8 layers, we first train a model with 100 clusters using the training strategy used for 12 layers and 100 clusters. We then finetuned this model for 5 epochs using 200 clusters and a learning rate of 0.00001.

### D.3.4 RoBERTa Approximation

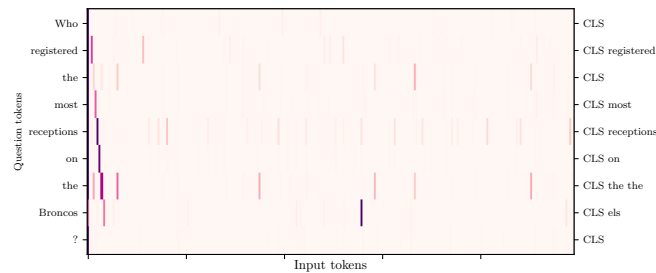
In this section we provide a qualitative comparison between the *full* attention, and the clustered attention variants *clustered* and *i-clustered* used for approximation. As described in § 5.4.3, we use 25 clusters for both attention variants. In Figure D.7 we show the attention distribution for the question tokens for a randomly selected question-context tuple from the SQuAD dataset. For each token in the question we show the attention distribution over the input sequence formed by concatenating the question and context tokens with *CLS* and *SEP* tokens appended. It can be seen that with only a few clusters, improved clustered approximates the softmax attention very closely even when the attention distribution has complicated and sparse patterns. In contrast, clustered attention fails to approximate such attention distributions. Moreover, it can further be seen that for almost all question tokens, both full and improved clustered have the same tokens with the highest attention weights. This further strengthens our believe that improved clustered attention can approximate a wide range of complicated attention patterns.

Manning finished the year with a career-low 67.9 passer rating, throwing for 2,249 yards and nine touchdowns, with 17 interceptions. In contrast, Osweiler threw for 1,967 yards, 10 touchdowns and six interceptions for a rating of 86.4. Veteran receiver **Demaryius Thomas** led the team with 105 receptions for 1,304 yards and six touchdowns, while Emmanuel Sanders caught 76 passes for 1,135 yards and six scores, while adding another 106 yards returning punts. Tight end Owen Daniels was also a big element of the passing game with 46 receptions for 517 yards. Running back C. J. Anderson was the team's leading rusher 863 yards and seven touchdowns, while also catching 25 passes for 183 yards. Running back Ronnie Hillman also made a big impact with 720 yards, five touchdowns, 24 receptions, and a 4.7 yards per carry average. Overall, the offense ranked 19th in scoring with 355 points and did not have any Pro Bowl selections.

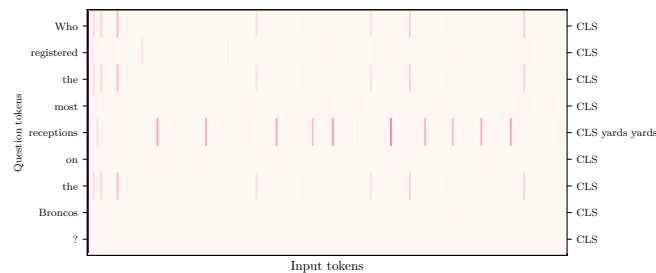
(a) context



(b) full



(c) improved-clustered



(d) clustered

Figure D.7 – Attention matrices for question-context tuples for *full* attention, and *clustered* and *i-clustered* attention used for approximation. D.7a shows the the context for the question with answer highlighted in red. D.7b shows the attention distribtution for *full*, D.7c and D.7d show the approximation using *i-clustered* and *clustered* respectively. Note that *i-clustered* has attention patterns very similar to *full* while *clustered* shows qualitatively different attention patterns. For each question token, we also present the tokens with highest attention above a threshold on the right axis. For more information refer to § D.3.4.



# Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.
- Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. (2015). Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*.
- Allen-Zhu, Z. (2017). Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1200–1205. ACM.
- Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.
- Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.
- Ba, J., Mnih, V., and Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.
- Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the 5th International Conference on Learning Representations*, San Diego, CA, USA.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.
- Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Warde-Farley, D., Goodfellow, I., Bergeron, A., et al. (2011). Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3, pages 1–48. Citeseer.

## Bibliography

---

- Blanc, G. and Rendle, S. (2017). Adaptive sampled softmax with kernel based sampling. *arXiv preprint arXiv:1712.00527*.
- Bordes, A., Ertekin, S., Weston, J., and Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6(Sep):1579–1619.
- Britz, D., Guan, M. Y., and Luong, M.-T. (2017). Efficient attention using a fixed-size memory representation. *arXiv preprint arXiv:1707.00110*.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Canévet, O., Jose, C., and Fleuret, F. (2016). Importance sampling tree for large-scale empirical expectation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1454–1462.
- Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE.
- Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.
- Chiu, C.-C. and Raffel, C. (2017). Monotonic chunkwise attention. *arXiv preprint arXiv:1712.05382*.
- Chollet, F. et al. (2015). keras. <https://github.com/fchollet/keras>.
- Choromanski, K., Likhoshesterov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.
- Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.
- Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*.
- Collobert, R., Bengio, S., and Mariéthoz, J. (2002). Torch: a modular machine learning software library. Technical report, Idiap.



- Combalia, M. and Vilaplana, V. (2018). Monte-carlo sampling applied to multiple instance learning for whole slide image classification.
- Cordonnier, J.-B., Loukas, A., and Jaggi, M. (2020). On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*.
- Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*.
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. (2019a). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.
- Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019b). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.
- Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654.
- Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. (2018). Universal transformers. *arXiv preprint arXiv:1807.03819*.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888. IEEE.
- Fan, Y., Tian, F., Qin, T., Bian, J., and Liu, T.-Y. (2017). Learning what data to learn. *arXiv preprint arXiv:1702.08635*.
- Gao, B. and Pavel, L. (2017). On the properties of the softmax function with application in game theory and reinforcement learning.
- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

## Bibliography

---

- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks.
- Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992). Switchboard: Telephone speech corpus for research and development. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 517–520. IEEE.
- Golatkar, A., Anand, D., and Sethi, A. (2018). Classification of breast cancer histology using deep learning. In *International Conference Image Analysis and Recognition*, pages 837–844. Springer.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- Goodman, J. (2001). Classes for fast maximum entropy training. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volume 1, pages 561–564. IEEE.
- Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006a). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.
- Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006b). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*.
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Hestenes, M. R., Stiefel, E., et al. (1952). *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC.
- Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.
- Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Hooker, S. (2020). The hardware lottery. *arXiv preprint arXiv:2009.06489*.
- Hou, L., Samaras, D., Kurc, T. M., Gao, Y., Davis, J. E., and Saltz, J. H. (2016). Patch-based convolutional neural network for whole slide tissue image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2424–2433.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
- Ilse, M., Tomczak, J., and Welling, M. (2018). Attention-based deep multiple instance learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2127–2136, Stockholmsmässan, Stockholm Sweden. PMLR.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.
- Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025.
- Jia, Z., Tillman, B., Maggioni, M., and Scarpazza, D. P. (2019). Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*.
- Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323.
- Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12.
- Kahn, H. and Harris, T. E. (1951). Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30.
- Katharopoulos, A. and Fleuret, F. (2018). Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Katharopoulos, A. and Fleuret, F. (2019). Processing megapixel images with deep attention-sampling models. In *Proceedings of the International Conference on Machine Learning (ICML)*.

## Bibliography

---

- Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kitaev, N., Kaiser, Ł., and Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.
- Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.
- Lample, G., Sablayrolles, A., Ranzato, M. A., Denoyer, L., and Jegou, H. (2019). Large memory layers with product keys. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Álché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8546–8557. Curran Associates, Inc.
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.
- Larsson, F. and Felsberg, M. (2011). Using fourier descriptors and spatial models for traffic sign recognition. In *Scandinavian Conference on Image Analysis*, pages 238–249. Springer.
- Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.
- LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2.
- LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia.
- Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*.
- Lei, L., Ju, C., Chen, J., and Jordan, M. I. (2017). Non-convex finite-sum optimization via scsg methods. In *Advances in Neural Information Processing Systems*, pages 2345–2355.

- Liang, T., Glossner, J., Wang, L., and Shi, S. (2021). Pruning and quantization for deep neural network acceleration: A survey. *arXiv preprint arXiv:2101.09671*.
- Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019a). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.
- Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2020a). On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.
- Liu, Y., Gadepalli, K., Norouzi, M., Dahl, G. E., Kohlberger, T., Boyko, A., Venugopalan, S., Timofeev, A., Nelson, P. Q., Corrado, G. S., et al. (2017). Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2020b). RoBERTa: A robustly optimized BERT pretraining approach.
- Loshchilov, I. and Hutter, F. (2015). Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*.
- Ma, X., Pino, J. M., Cross, J., Puzon, L., and Gu, J. (2020). Monotonic multihead attention. In *International Conference on Learning Representations*.
- Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2017). High-resolution image classification with convolutional networks. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 5157–5160. IEEE.
- Maybury, M. (1999). *Advances in automatic text summarization*. MIT press.
- Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one? In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 14014–14024. Curran Associates, Inc.
- Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088.
- Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212.
- Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.

## Bibliography

---

- Nazeri, K., Aminpour, A., and Ebrahimi, M. (2018). Two-stage convolutional neural network for breast cancer histology image classification. In *International Conference Image Analysis and Recognition*, pages 717–726. Springer.
- Needell, D., Ward, R., and Srebro, N. (2014). Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025.
- Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53.
- Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782.
- Nocedal, J. and Wright, S. J. (2006). Line search methods. *Numerical optimization*, pages 30–65.
- Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- Parmar, N., Ramachandran, P., Vaswani, A., Bello, I., Levskaya, A., and Shlens, J. (2019). Stand-alone self-attention in vision models. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 68–80. Curran Associates, Inc.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.
- Paul, D. B. and Baker, J. M. (1992a). The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics.
- Paul, D. B. and Baker, J. M. (1992b). The design for the wall street journal-based csr corpus. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91.
- Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. (2021). Random feature attention. *arXiv preprint arXiv:2103.02143*.
- Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. (2011). The kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society.

- Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech*, pages 2751–2755.
- Povey, D., Zhang, X., and Khudanpur, S. (2015). Parallel training of dnns with natural gradient and parameter averaging. In *In International Conference on Learning Representations: Workshop track*.
- Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.
- Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE.
- Radford, A., Narasimhan, K., Salimans, T., , and Sutskever, I. (2018). Improving language understanding by generative pre-training. In *OpenAI report*.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.
- Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.
- Ramapuram, J., Diephuis, M., Webb, R., and Kalousis, A. (2018). Variational saccading: Efficient inference for large resolution images. *arXiv preprint arXiv:1812.03170*.
- Ranzato, M. (2014). On learning where to look. *arXiv preprint arXiv:1405.5488*.
- Rawat, A. S., Chen, J., Yu, F. X. X., Suresh, A. T., and Kumar, S. (2019). Sampled softmax with random fourier features. In *Advances in Neural Information Processing Systems*, pages 13834–13844.
- Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE.
- Recasens, A., Kellnhofer, P., Stent, S., Matusik, W., and Torralba, A. (2018). Learning to zoom: a saliency-based sampling layer for neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 51–66.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.
- Richtárik, P. and Takáč, M. (2013). On optimal probabilities in stochastic coordinate descent methods. *arXiv preprint arXiv:1310.3438*.

## Bibliography

---

- Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.
- Roy, A., Saffar, M., Vaswani, A., and Grangier, D. (2020). Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:1908.03265*.
- Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017). Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*.
- Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
- Schlag, I., Irie, K., and Schmidhuber, J. (2021). Linear transformers are secretly fast weight memory systems. *arXiv preprint arXiv:2102.11174*.
- Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.
- Shen, Z., Zhang, M., Zhao, H., Yi, S., and Li, H. (2020). Efficient attention: Attention with linear complexities. *arXiv preprint arXiv:1812.01243*.
- Shrivastava, A. and Li, P. (2014). Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329.
- Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., and Moreno-Noguer, F. (2015). Discriminative learning of deep convolutional feature point descriptors. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 118–126. IEEE.
- Sirinukunwattana, K., Raza, S. E. A., Tsang, Y.-W., Snead, D. R., Cree, I. A., and Rajpoot, N. M. (2016). Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE transactions on medical imaging*, 35(5):1196–1206.
- Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). MASS: Masked sequence to sequence pre-training for language generation. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936, Long Beach, California, USA. PMLR.
- Sperber, M., Niehues, J., Neubig, G., Stüker, S., and Waibel, A. (2018). Self-attentional acoustic models. In *19th Annual Conference of the International Speech Communication Association (InterSpeech 2018)*, Hyderabad, India.
- Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. (2019a). Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy. Association for Computational Linguistics.



- Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. (2019b). Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.
- Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR.
- Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. (2019). Transformer dissection: An unified understanding for transformer’s attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4343–4352, Hong Kong, China. Association for Computational Linguistics.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Vyas, A., Katharopoulos, A., and Fleuret, F. (2020). Fast transformers with clustered attention. In *Proceedings of the international conference on Neural Information Processing Systems (NeurIPS)*.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
- Wang, L., Yang, Y., Min, M. R., and Chakradhar, S. (2016). Accelerating deep neural network training with inconsistent stochastic gradient descent. *arXiv preprint arXiv:1603.05544*.
- Wu, C.-Y., Manmatha, R., Smola, A. J., and Krahenbuhl, P. (2017). Sampling matters in deep embedding learning. In *The IEEE International Conference on Computer Vision (ICCV)*.
- Wu, Z., Shen, C., and Van Den Hengel, A. (2019). Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.
- Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., and Hua, X.-s. (2019a). Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7308–7316.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. (2019b). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

## Bibliography

---

- You, Y., Gitman, I., and Ginsburg, B. (2017). Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*.
- Zafir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). Q8BERT: quantized 8bit BERT. *CoRR*, abs/1910.06188.
- Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P., editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press.
- Zhao, P. and Zhang, T. (2015). Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1–9.

# Angelos Katharopoulos

## PhD Student in EPFL

✉ Bahnhofstrasse 34 angelos.katharopoulos@idiap.ch  
🏠 Rümlang 🌐 angeloskath.github.io  
Switzerland 🔄 github.com/angeloskath

---

- EDUCATION**
- Doctor of Philosophy (PhD)** 2017 – Today  
École Polytechnique Fédérale de Lausanne, Switzerland  
**PhD Advisor:** François Fleuret (<https://fleuret.org/francois>)  
**Research interests:** Making Deep Neural Networks efficient.
- Electrical and Computer Engineering** 2008 - 2015  
Aristotle University of Thessaloniki, Greece  
**Degree:** Diploma in Electrical and Computer Engineering  
**Thesis:** Learning discriminative codebooks for local feature aggregation
- PUBLICATIONS**
- [1] Despoina Paschalidou, **Angelos Katharopoulos**, Andreas Geiger, Sanja Fidler. “Neural Parts: Learning Expressive 3D Shape Abstractions with Invertible Neural Networks”. IEEE/CVF Conference on Computer Vision and Pattern Recognition (*CVPR*), 2021.
  - [2] Apoorv Vyas, **Angelos Katharopoulos**, François Fleuret. “Fast Transformers with Clustered Attention”. Neural Information Processing Systems (*NeurIPS*), 2020.
  - [3] **Angelos Katharopoulos**, Apoorv Vyas, Nikolaos Pappas, François Fleuret. “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”. In Proceedings of the International Conference on Machine Learning (*ICML*), 2020.
  - [4] **Angelos Katharopoulos**, François Fleuret. “Processing Megapixel Images with Deep Attention-Sampling Models”. In Proceedings of the International Conference on Machine Learning (*ICML*), 2019.
  - [5] **Angelos Katharopoulos**, François Fleuret. “Not All Samples Are Created Equal: Deep Learning with Importance Sampling”. In Proceedings of the International Conference on Machine Learning (*ICML*), 2018.
  - [6] **Angelos Katharopoulos\***, Despoina Paschalidou\*, Christos Diou, Anastasios Delopoulos. “Learning Local Feature Aggregation Functions with Backpropagation”. In Signal Processing Conference (*EUSIPCO '17*).
  - [7] **Angelos Katharopoulos\***, Despoina Paschalidou\*, Christos Diou, Anastasios Delopoulos. “Fast Supervised LDA for discovering micro-events in large-scale video datasets”. ACM International conference on Multimedia (*MM '16*).
- PROFESSIONAL EXPERIENCE**
- Research Intern** June 2021 – October 2021  
Facebook AI Research, Menlo Park, CA (remotely)  
Working on efficient transformers for structured computer vision tasks.

**Research Intern** January 2021 – June 2021  
Naver Labs Europe, Grenoble, France (remotely)  
Working on transformer networks for visual localization.


**Graduate Research Assistant** March 2017 – January 2021  
Idiap Research Institute, Martigny, Switzerland  
Reducing the computational complexity of deep neural networks using sampling or approximations.

**Senior Software Engineer** March 2014 – January 2015  
Total Eclipse, Thessaloniki, Greece  
Designed the software architecture for in-house frameworks in C++ and C#, implemented 2D visual effects with OpenGL shaders.


**Founder & Lead Software Engineer** November 2009 – February 2014  
Yourse, Thessaloniki, Greece  
Yourse became the most successful job search engine in Greece utilizing its advantage in Greek natural language processing. Among others, I implemented the search engine crawler, the query parser, the stemmers and the search quality and ranking.


OPEN SOURCE  
SOFTWARE  
DEVELOPMENT


**Fast Transformers** June 2020 – Today  
<https://fast-transformers.github.io>  887★  
A library for efficient self-attention and transformer implementations.

**simple-3dviz** August 2019 – Today  
<https://simple-3dviz.com>  58★  
Fast and easy-to-use 3D visualization library with python and OpenGL.

**Attention Sampling** July 2019 – Today  
<https://attention-sampling.com>  76★  
Allows neural networks to process extremely large images with a complexity decoupled from the size of the images.

**Keras Importance Sampling** July 2017 – Today  
<https://importance-sampling.com>  260★  
Keras Importance Sampling is a library that speeds up neural network training by selecting important examples from the dataset for each iteration.

**LDA++** February 2016 – Today  
<http://ldaplusplus.com>  19★  
A C++ library for fast supervised and unsupervised Latent Dirichlet Allocation.

**PHP NlpTools** July 2012 – January 2018  
<http://php-nlp-tools.com>  663★  
NlpTools is the most comprehensive PHP library for natural language processing. It provides a relatively large collection of algorithms for text classification, clustering, topic modeling, stemming, preprocessing and more.

## SKILLS

- **Programming Languages:** C, C++, C#, Python, PHP, Java 7/8, JavaScript, Go, Matlab/Octave, GLSL, bash
- **Parallel and Distributed Computing:** Nvidia CUDA, MPI
- **Deep Learning Frameworks:** Keras, TensorFlow, PyTorch, Theano, Caffe
- **Grid Infrastructure:** Sun Grid Engine, Amazon EC2, Google Compute Engine
- **Software testing:** JUnit, PHPUnit, googletest, unittest
- **Build automation:** GNU Make, CMake
- **RDBMSs:** PostgreSQL, MySQL, Sqlite
- **Other:** OpenCV, VLFeat, Atmel AVR assembly, Linux, Git, L<sup>A</sup>T<sub>E</sub>X

## ACADEMIC SERVICE

- Reviewer for ICCV (2019), ECCV (2020), NeurIPS (2019, 2020), AAAI (2020, 2021), ICLR (2020), CVPR (2020, 2021), ICML (2021)

## DISTINCTIONS

- Outstanding Reviewer Award (ECCV 2020, CVPR 2021)
- IEEE Xtreme Country Winner (2012, 2013, 2014, 2015)  
55th, 55th, 62nd, 54th respectively worldwide

## TEACHING EXPERIENCE

### **Deep Learning (Assistant)**

African Masters of Machine Intelligence, Kigali, Rwanda

October 2018

### **Deep Learning (Assistant)**

EPFL, Lausanne, Switzerland

Spring 2018, Spring 2017