Neural language models : Dealing with large vocabularies

Matthieu Labeau

NNT : 2018SACLS313

Neural Language Models: Dealing with Large Vocabularies

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université Paris-Sud

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 21/09/2018, par

MATTHIEU LABEAU

Composition du Jury :

Pierre ZWEIGENBAUM Senior Researcher, LIMSI-CNRS	Président
Massih-Reza AMINI Professor, Université Grenoble-Alpes	Rapporteur
Phil BLUNSOM Associate Professor, University of Oxford	Rapporteur
Armand JOULIN Research Scientist, Facebook Artificial Intelligence Research	Examinateur
André MARTINS Research Scientist, Instituto de Telecomunicações	Examinateur
Alexandre ALLAUZEN Professor, Université Paris-Sud	Directeur de thèse

Abstract

Neural networks architectures have been applied to a myriad of tasks in Natural Language Processing (NLP), mainly since the success they met following their early application to language modeling. Language models are an important component in many NLP tasks, where they provide prior knowledge on the language used. However, the cost of training a neural language model grows linearly with the number of words it considers — the number of words in the *vocabulary* — which makes training very large-vocabulary neural language models a difficult task. Another observation made on neural language models is that they are generally less effective on rare words than discrete models. This issue is linked to the size of the vocabulary; indeed, the particular repartition of word counts in a text (namely, its *Zipfian* shape) means we systematically encounter a large number of infrequent words in our corpus, number that will grow with the size of the vocabulary. This is of course even more impactful for languages with rich morphologies. Besides, since each word is associated to a particular representation, there is no relation between words that share morphological caracteristics.

This work investigates practical methods to ease training and improve performances of neural language models with large vocabularies. Despite several training tricks, the most straightforward way to limit computation time is to limit the vocabulary size, which is not a satisfactory solution for numerous tasks. Most of the existing methods used to train large-vocabulary language models revolve around avoiding the computation of the partition function, ensuring that output scores are normalized into a probability distribution. Here, we focus on sampling-based approaches, including importance sampling and noise contrastive estimation. These methods allow an approximate computation of the partition function. After examining the mechanism of *self-normalization* in noise-contrastive estimation, we first propose to improve its efficiency with solutions that are adapted to the inner workings of the method and experimentally show that they considerably ease training. Our second contribution is to expand on a generalization of several sampling based objectives as *Bregman divergences*, in order to experiment with new objectives. We use *Beta divergences* to derive a set of objectives from which noise contrastive estimation is a particular case.

Finally, we aim at improving performances on full vocabulary language models, by augmenting output words representation with subwords. We experiment on a Czech dataset and show that using character-based representations besides word embeddings for output representations gives better results. We also show that reducing the size of the output look-up table improves results even more.

Résumé

Les méthodes neuronales ont été appliquées avec succès à de nombreuses tâches en traitement automatique du langage (TAL) depuis leur application aux modèles de langues. Ceux-ci demeurent des composants particulièrement importants dans de nombreux systèmes de TAL, apportant des connaissances préalables sur la langue utilisée. Cependant, le coût d'entraînement des modèles de langues neuronaux augmente linéairement avec le nombre de mots qu'il considère — c'est à dire la taille de son *vocabulaire* — ce qui rend l'entraînement des modèles à grand vocabulaire difficile. De plus, les modèles de langues neuronaux sont en général moins efficaces sur les mots rares que les modèles discrets. Ce problème est lié à la taille du vocabulaire; en effet, la répartition des fréquences des mots dans un texte (qu'on appelle *Zipfienne*) implique que le nombre de mots rares dans un corpus sera systématiquement élevé. Ce phénomène a un impact plus important pour les langues à la morphologie riche. De plus, puisque chaque mot est associé à une représentation particulière, aucun usage n'est fait des similitudes morphologiques que partagent certains mots.

Le travail présenté dans cette thèse explore les méthodes pratiques utilisées pour faciliter l'entraînement et améliorer les performances des modèles de langues munis de très grands vocabulaires. La principale limite à l'utilisation des modèles de langue neuronaux est leur coût computationnel: il dépend de la taille du vocabulaire avec laquelle il grandit linéairement. La façon la plus aisée de réduire le temps de calcul de ces modèles reste de limiter la taille du vocabulaire, ce qui est loin d'être satisfaisant pour de nombreuses tâches. La plupart des méthodes existantes pour l'entraînement de ces modèles à grand vocabulaire évitent le calcul de la fonction de partition, qui est utilisée pour forcer la distribution de sortie du modèle à être normalisée en une distribution de probabilités. Ici, nous nous concentrons sur les méthodes à base d'échantillonnage, dont le sampling par importance et l'estimation contrastive bruitée. Ces méthodes permettent de calculer facilement une approximation de cette fonction de partition. L'examen des mécanismes de l'estimation contrastive bruitée nous permet de proposer des solutions qui vont considérablement faciliter l'entraînement, ce que nous montrons expérimentalement. Ensuite, nous utilisons la généralisation d'un ensemble d'objectifs basés sur l'échantillonnage comme *divergences de Bregman* pour expérimenter avec de nouvelles fonctions objectif.

Enfin, nous exploitons les informations données par les unités sous-mots pour enrichir les représentations en sortie du modèle. Nous expérimentons avec différentes architectures, sur le Tchèque, et montrons que les représentations basées sur les caractères permettent l'amélioration des résultats, d'autant plus lorsque l'on réduit conjointement l'utilisation des représentations de mots.

Remerciements

Je voudrais tout d'abord remercier Alexandre, qui a dirigé mon stage, puis ma thèse. J'ai eu la chance de bénéficier de ses encouragements et de ses conseils, en matière de recherche, d'enseignement, et de tout ce qui concerne de près ou de loin le travail d'un doctorant, pendant plus de 4 ans. Sa disponibilité et son appui, ainsi que nos discussions, ont été déterminants pour la réalisation de cette thèse.

Je voudrais ensuite remercier les membres du jury, qui ont accepté de participer à l'évaluation de ce travail, et les rapporteurs, pour leur relecture et leurs avis. Particulièrement, merci à Pierre, pour ses remarques et conseils.

Le LIMSI a été un environnement de travail on ne peut plus plaisant pendant ces années de thèse, et ce grâce aux membres des groupes TLP et ILES. D'abord, merci à Lauriane, avec qui j'ai eu la chance de partager un bureau pendant plusieurs années, et Franck, avec qui j'ai partagé beaucoup de pauses qui finissaient invariablement en discussion sur la dernière architecture neuronale à la mode. Les réunions autour du thême traduction ont été mes premiers pas dans la communauté scientifique; je tiens à en remercier les membres, mais tout particulièrement François et Guillaume pour les nombreux articles, toujours très intéressants, que j'ai pu lire grâce à eux, et pour toutes les problématiques que j'ai découvert durant ces groupes de lectures. Les doctorants qui m'ont précédé au sein de ce thême ont aussi ma gratitude: pour leurs conseils et leur aide, merci à Nicolas, Khanh, Benjamin et Yong. J'ai beaucoup aimé partager le quotidien des doctorants du LIMSI; Qu'elle ai été régulière ou plus épisodique, la compagnie de Pierre, d'Elena, de Julia et de Kevin a toujours été enrichissante. Merci aussi aux doctorants que j'ai vu arriver; Rachel, Pooyan, Yuming, Swen, Arnaud, pour le temps passé ensemble et les discussions, scientifiques ou non - et Charlotte, même si elle a su résister à l'attrait d'une thèse. Enfin, j'ai été très content de rencontrer et de passer du temps avec Aina, Syrielle, Benjamin, José et Aman, qui ont commencé leurs thèses alors que je profitais d'une prolongation, sans oublier Margot, qui l'a commencé juste après.

De nombreux membres permanents du Limsi m'ont aidé à comprendre un peu mieux le travail de chercheur et d'enseignant, et j'ai beaucoup apprécié les discussions que j'ai eu avec eux: merci à Thomas, Hélène, Hervé et Claude; et particulièrement à Marianna, pour les relations que j'ai pu nouer grâce à elle, et pour ses nombreux conseils pour l'après-thèse. Merci aussi à Leonardo et Jane, qui m'ont fait profiter de leur expérience de post-doctorants. Enfin, merci à Éric pour sa disponibilité, et à Laurence pour son aide - ainsi qu'à tous les autres membres du Limsi, à qui je dois ma gratitude.

Encore merci à Franck, Pooyan, et Syrielle, qui m'ont aidé à relire ce qui suit; à eux, à Lauriane, et à tous ceux avec qui nous avons pu passer un moment aux deux gares; et à Alexandre. Et bien sur, à ma famille et aux amis qui m'ont cotoyé, soutenu, supporté pendant ces 4 dernières années.

Contents

List of Figures

List of Tables

Introduction

Neural networks architectures have been applied to a myriad of tasks in Natural Language Processing (NLP), mainly since the success they met following their early application to language modeling (Schwenk and Gauvain, 2002; Bengio et al., 2003). Language models are an important component in many NLP tasks, where they provide prior knowledge on the language used. They attribute a probability to a sentence S, based on data upon which the language model is built. In language modeling, where words are the atomic units, a sentence S is a sequence of words:

$$S = (w_1, \ldots, w_t, \ldots, w_{|S|})$$

The representational power of language models is limited by how many words they have knowledge of: these words are listed in a finite vocabulary $\mathcal{V}$. Usually, this vocabulary is built from the same data as the model. Hence, the language model can only represent (at most) the words that are present in the training data. The probability of S is written as:

$$P(S) = \prod_{i=1}^{|S|} P(w_t | w_1, \ldots, w_{t-1})$$

While probabilities given by discrete language models are based on the number of occurrences of a sequence in the training corpus, neural language models jointly learn continuous representations of words, which the model scores together. These continuous vectors, that we call *embeddings*, are *distributed* representations: the joint learning procedure of neural language models implies that embeddings are learned similar for words that have similar contexts, allowing for generalization across these contexts. The development of unsupervised methods to learn such representations (Mikolov et al., 2013), of recurrent structures (Elman, 1990b; Mikolov et al., 2010) - and their improvements (Hochreiter and Schmidhuber, 1997) - made neural architectures even more popular in NLP.

While neural language models can be used as stand-alone models, to compute the probabilities of or generate sequences of words, they are necessary to many sophisticated systems. Intuitively, improving a language model should also improve the metric associated to the task it is applied to. As a matter of fact, neural language models were applied with success to Automatic Speech Recognition (ASR) (Schwenk and Gauvain, 2004; Schwenk, 2007; Mnih and Hinton, 2009; Le et al., 2011) and Statistical Machine Translation (SMT) (Le et al., 2012; Schwenk et al., 2012; Devlin et al., 2014; Cho et al., 2014c), among many other tasks. For these tasks, a reduced

vocabulary may consequently hinder performance - for example, with the recent task of neural machine translation, the translation performance decreases when the number of unknown words goes up (Cho et al., 2014a). For some tasks, a large vocabulary is necessary. Indeed, rare words may need to be represented: for example in named entity recognition, or any work on social media datasets, for which the vocabulary is very diverse due to spelling errors and informal, irregular or abbreviated words. Besides, these issues are even more pregnant for morphologically rich languages. We can take the example of Czech, which is a highly inflected language: the vocabularies generated from Czech data is far larger. For instance, the Czech words *dnes, dnešní, dnešních, dnešku, dnesnímu, ...* each have their own representation in a vocabulary, whereas in English, only a few words, among which *today* and *current*, would be represented. This shows the importance of being able to train large-vocabulary neural language models.

Training a neural language model is usually done by maximizing its likelihood on the data. This is maximum likelihood estimation (MLE). In practice, we minimize the negative log-likelihood (NLL):

$$NLL(S) = -\sum_{t=1}^{|S|} \log P(w_t|w_1, \ldots, w_{t-1})$$

Computing this objective and its gradient is however slow and costly. Indeed, in order to compute the probability of the next word, we need to compute the score given by the model to each word in the associated vocabulary $\mathcal{V}$ and to normalize these scores, via a *softmax* layer. Thus, the cost of training grows linearly with $|\mathcal{V}|$, which makes training very large-vocabulary language models a difficult task. Quite a few solutions have been proposed: *hierarchical models* modify the standard architecture of the output layer (Morin and Bengio, 2005; Mnih and Hinton, 2009; Le et al., 2011). Others, which only speed-up computation during training, approximate the training objective via sampling schemes. The most notable are *importance sampling* (Bengio and Sénécal, 2003, 2008) and *noise contrastive estimation* (Gutmann and Hyvärinen, 2012; Mnih and Teh, 2012). These approaches each have they own strenghs and weaknesses, but they seem to fall off for very large vocabularies (Chen et al., 2016).

Another observation made on neural language models is that they are generally less effective on rare words than discrete models. This issue is linked to the size of the vocabulary; indeed, the particular repartition of word counts in a text (namely, its *Zipfian* shape) means we systematically encounter a large number of infrequent words in our corpus, number that will grow with the size of the vocabulary. This is of course even more impactful for languages with rich morphologies: the Czech words we used as examples previously are, for most of them, rare words. For these words, the number of occurences is never sufficient, and the weak generalization attained by using distributed representations does not help. Besides, since each word is associated to a particular representation, there is no relation between words that share morphological caracteristics. This implies that *dnes, dnešní, dnešních* and other related forms are learnt independently. To attain a better form of general-

ization, language models can rely on subword units for input words, whether they are characters (Kim et al., 2015) or morphological features (Botha and Blunsom, 2014). This also presents the advantage of vastly reducing the number of parameters needed to represent words.

In this context, the motivations for this work are twofold: first, we wish to facilitate training of large vocabulary neural language models. Secondly, we would like to improve the performances of neural language models when trained on a full vocabulary, which means that a non-negligeable number of words will be infrequent in the training data. This implies that we will work on standalone language models. We will verify their efficiency using an intrisic measure, the *perplexity*. We will work towards our first objective by reviewing existing strategies aiming at training large vocabularies neural language models; focusing on sampling-based methods, we will specifically analyze the inner workings of noise contrastive estimation and propose solutions to ease learning with this method. Finally, we will try to replace noise contrastive estimation and other sampling-based methods in a larger class of objective functions and experimentally explore some of them. To achieve our second objective, we will take interest in subword-based representations. In order to improve performances of neural language models on infrequent words, we will not only use them for representing input words, but also to augment output word representations.

This work is organized as follows: in Chapter 1, we review language models in general and particulary neural language models. We detail the architecture of feedforward and recurrent language models and how they are usually trained. After explaining the practical aspects of training neural language models, we outline the design choices we have made for the experiments presented subsequently. Chapter 2 describes various strategies to efficiently train large vocabularies language models - mainly focusing on sampling-based approaches. The two main methods we discuss are importance sampling and noise contrastive estimation. In Chapter 3, we begin by extensively experimenting with the hyperparameters of importance sampling and noise contrastive estimation, as well as other related methods. In order to improve the efficiency of noise contrastive estimation, we examine the mechanism of *self-normalization* and how it is affected by the frequency distribution of words in the text and the size of the vocabulary. We propose solutions that are adapted to the inner workings of the method and experimentally show that they considerably ease training. Chapter 4 will expand on a generalization of several sampling based objectives as *Bregman divergences*, in order to experiment with new objectives. We use *Beta divergences* to derive a set of objectives from which noise contrastive estimation is a particular case. Finally, in Chapter 5, we aim at improving performances on full vocabulary language models, by augmenting output word representations with subwords and experimenting with various architectures. We experiment on a Czech dataset and show that using character-based representations besides word embeddings for output representations gives better results when working on the full vocabulary. We also show that in that case, reducing the size of the output look-up table improves results even more.

The main contributions presented in the Sections 3.2 and 3.3 have been published in Labeau and Allauzen (2018), while an exploration of the results of Section 3.1.2 has been published in Labeau and Allauzen (2017a). Preliminary work on subword-based neural architectures, applied to a different task, has been published in Labeau et al. (2015) and is presented in Annex D. We applied such architectures to language modeling with an extrinsic evaluation on machine translation, in Labeau and Allauzen (2017c). Finally, the main contributions of Chapter 5 were first explored in Labeau and Allauzen (2017b).

Chapter 1

From Discrete to Neural Language Models

Contents

Language models play a key role in various Natural Language Processing tasks, such as machine translation and speech recognition. Their function is to assign probabilities to sentences, *i.e* to determine how likely a sentence is in a given language. As such, we can view them as prior knowledge on the language they model. The goal is to assign a low probability to sequences that are grammatically incorrect, but also to sequences that are grammatically correct and do not make sense. In this work, we will only consider *statistical language models*, which determine the likelihood of a sentence based only on data upon which the language model is built. A sentence S is considered as a sequence of words

$$S = (w_1, \dots, w_t, \dots, w_{|S|})$$

Each of these words belongs to a finite vocabulary $\mathcal{V}$, which contains all the words the model is able to represent. We want to assign to S a probability

$$P(w_1, \dots, w_{|S|})$$

Assuming that the words composing the sentence have been generated depending on the sequence of previous words, and using the *chain rule*, we can write this probability as:

$$P(S) = \prod_{t=1}^{|S|} P(w_t | w_1, \dots, w_{t-1})$$

We choose to write each sequence of previous words $(w_1, \dots, w_{t-1})$ as a context H_t, transforming the probability of our sequence into a product of conditional probabilities of words given a context $P(w_t|H_t)$. Then, the generation of a word w given a context H can be modeled as depending on a multinomial distribution on the vocabulary $\mathcal{V}$. Our statistical language model must learn these distributions from a dataset $\mathcal{D}$, which is composed of couples (H, w). At first, to make the task easier, we can use the Markov assumption: we restrict the conditional dependency of the word w_t on the $n-1$ previous words. Then,

$$P(S) = \prod_{t=1}^{|S|} P(w_t | w_{t-n+1}, \dots, w_{t-1})$$

We call the models using this approximation n-gram models. The choice of n is a trade-off between being able to capture longer dependencies between words, and needing more parameters to model the conditional probabilities. Increasing n also brings another issue: the *data sparsity*. Indeed, since each word comes from a finite vocabulary $\mathcal{V}$, there are $|\mathcal{V}|^n$ possible n-grams. With higher values of n, it is increasingly unlikely to have a particular n-gram appear in the training data $\mathcal{D}$.

Historically, the first language modeling approaches are based upon counting n-gram occurrences, which we will discuss in Section 1.1. To deal with the issue of data sparsity, they were improved with *smoothing* methods, which allow giving non-zero probabilities to sequences that were never encountered in the training data. When improved in that manner, discrete approaches are still competitive, because of their speed and how little space they need. However, they lack the ability to generalize and imply an exponential growth of their number of parameters if we are to increase

n. The introduction of neural networks language models, that use continuous distributed representations of words, allowed to overcome this issue. Using a recurrent structure permitted to remove the constraint of keeping a fixed-size context, and to capture long-term dependencies. We will present these models in Section 1.2. However, training neural networks for language modeling presents numerous practical difficulties, among which computation time is not the least. It also implies making quite a lot of choices on the design of the model. In the last section (1.3), we will rapidly summarize some of these difficulties, and how they are usually addressed. We will then discuss the computational bottleneck of neural network language models, and how it relates to the size of the output vocabulary, which is the main topic of this work.

1.1 Discrete language models

Some of the most used language models are very simple: they are based on relative frequencies of sequences. Indeed, maximum likelihood estimation (MLE) on the training data $\mathcal{D}$ gives the following estimate:

$$P_{MLE}(w_t|w_{t-n+1},\dots,w_{t-1}) = \frac{c_{\mathcal{D}}(w_{t-n+1},\dots,w_t)}{c_{\mathcal{D}}(w_{t-n+1},\dots,w_{t-1})}$$

where $c_{\mathcal{D}}$ is the number of occurrences of a sequence in the training corpus $\mathcal{D}$. Usually, n is chosen to be 2 or 3, corresponding to *bigram* and *trigram* models. However, if the quantity of training data is not large enough for the size of the model we are training, there is a higher risk of never coming across word sequences that are still plausible, therefore attributing them $P_{MLE} = 0$. This especially is an issue for applications such as speech recognition or machine translation, where the word sequence in question could not be outputted, even if well evaluated by the acoustic or translation model. Therefore, *smoothing* methods are used to adjust the MLE probabilities and prevent zero probabilities.

Smoothing

The main idea behind smoothing is to move a small amount of the probability mass attributed to n-grams seen in the data to the n-grams that are unseen. The simplest way to do so is to add a minimum number of occurrences to each sequence: this is *additive smoothing*. However, this offers poor performance. A variety of more complex methods exist, which are reviewed and compared in Chen and Goodman (1999), or again in Goodman (2001b). These methods usually combine high-order models (with a high value of n) with lower order models. Mostly, smoothing techniques can be divided in two categories, according to how this combination is accomplished. First, *backoff* models can be described by the following rule:

$$P_{backoff}(w_t|w_{t-n+1},\dots,w_{t-1}) =$$

$$\begin{cases} \alpha(w_t|w_{t-n+1},\dots,w_{t-1}) & \text{if } c_{\mathcal{D}}(w_{t-n+1},\dots,w_t) > 0 \\ \gamma_{t-n+1}^{t-1} P_{backoff}(w_t|w_{t-n+2},\dots,w_{t-1}) & \text{if } c_{\mathcal{D}}(w_{t-n+1},\dots,w_t) = 0 \end{cases} \tag{1.1}$$

If the sequence has a non-zero count, we use a distribution α which is based on discounted relative frequencies. The discounting method varies: *Katz smoothing* (Katz, 1987) uses a ratio based on the Good-Turing estimate (Good, 1953), while other methods use an absolute discount, notably *Kneser-Ney smoothing* (Kneser and Ney, 1995b). The discounted quantity is then attributed to the sequences which have a zero-count, backing-off to a lower order model, where γ is a normalizing factor, to ensure the distribution sums to 1.

The other category gathers *interpolated* models. They are the linear interpolation of higher and lower-order n-gram models:

$$P_{interpolated}(w_t|w_{i-n+1},\dots,w_{t-1}) =$$

$$\lambda_{i-n+1}^{t-1} \frac{c_{\mathcal{D}}(w_{i-n+1},\dots,w_t)}{c_{\mathcal{D}}(w_{i-n+1},\dots,w_{t-1})} + (1-\lambda_{i-n+1}^{t-1}) P_{interpolated}(w_t|w_{i-n+2},\dots,w_{t-1}) \tag{1.2}$$

Jelinek and Mercer (1980) and Witten and Bell (1991) give examples of such models. The main difference between the two categories resides in how they estimate the probabilities of sequences which appear in the training data: interpolated models use information from lower-order distribution, while back-off models do not. Besides comparing these methods, Chen and Goodman (1999) experiment with combining them. They propose a modified version of the Kneser-Ney smoothing which is still widely used, and a recommended discrete language model for many applications.

Limitations

Several aspects of discrete language models, even with smoothing, could be improved upon. As we have seen, higher-order models often use lower-order dependencies to assign probabilities, because of the data sparsity. Besides, augmenting the size of sequences the model is able to consider has an exponential cost in parameters, which forces the context to be most of the time very short. For these reasons, longer dependencies are ignored.

This brings up another issue: most of the parameters are *free* parameters, since they directly depend on values represented by other parameters corresponding to smaller sequences. A model being able to share parameters, in order to re-use the information previously learnt for new sequences, would also be able to better *generalize* to unseen sequences. Various work tried to tackle these difficulties. Some types of language models directly incorporate knowledge about the language, whether it is in the form of structural information (as structured language models, described in Chelba and Jelinek (2000)) or by grouping words that belong to a same group together to exploit their closeness, as for class-based language models[1] (Brown et al.,

[1]This type of language models can also be used to deal with the difficulty of having a large

1992; Kneser and Ney, 1993). However, an approach based on using *continuous* representations of words with neural networks for probability estimation offers a solution to all these issues.

1.2 Neural network language models

We will first describe the standard *feedforward* language model, introduced in Bengio et al. (2003). There are two main ideas behind this model: First, using a *distributed* representation, inspired by Elman (1990b); Hinton (1986), for each word. It means that instead of representing a word as being only one element in a large vocabulary, each word is characterized by a number of features. This allows to describe a large number of objects with a relatively small number of parameters, and to generalize information to new objects easily. Secondly, making these representations *continuous*.Then, the smooth probability function that takes as input a fixed-length word sequence (a context) and outputs a probability distribution on $\mathcal{V}$ can be modeled with a neural network.

Both the parameters of the distributed representations and of the probability function are learnt jointly. Previously, several approaches applied neural networks to language modeling (Miikkulainen and Dyer, 1991; Xu and Rudnicky, 2000), but never to learn a large-scale model of the distribution of word sequences. However, we can mention maximum entropy language models (Berger et al., 1996), which can be assimilated to neural networks without a hidden layer.

1.2.1 Feedforward language models

As previously, we want to be able to assign a probability

$$P(w_1, \ldots, w_{|S|})$$

to a sequence S. We still consider the Markov assumption to be true: we want to model $P(w_t|H_t)$ where $H_t = (w_{t-n+1}, \ldots, w_{t-1})$ is the first $n-1$ words of the n-gram, while w_t is the last. Then, we consider the conditional probability distribution over $\mathcal{V}$ as a continuous function to be learned:

$$P(.|H_t) = f(H_t)$$

Word embeddings

This function is decomposed in two parts. First, the distributed representations of the words of $\mathcal{V}$. These are vectors, that we note $\mathbf{r} \in \mathbb{R}^{d_r}$. In practice, they are represented by a *Look-up matrix* $\mathbf{L} \in \mathbb{R}^{d_r \times |\mathcal{V}|}$. Each row of this matrix is a continuous vector that corresponds to a word in the vocabulary: $\mathbf{L} = [\mathbf{r}_j]_{j=1}^{|\mathcal{V}|}$. These representations are called *word embeddings*. To obtain a representation of the full

vocabulary. Thus, we will discuss them a little more in Section 2.1

context H_t, we simply concatenate the word embeddings corresponding to the words of H_t:

$$\mathbf{x}_t = [\mathbf{r}_{w_{t-n+1}}; \ldots; \mathbf{r}_{w_{t-1}}]$$

Then, $\mathbf{x}_t \in \mathbb{R}^{(n-1)d_r}$ represents the whole input sequence H_t. Then, we map this vector to a conditional probability distribution over $\mathcal{V}$.

Hidden layers

This is done by applying successive feature extracting layers: each layer consists in a simple linear transformation, using a weight matrix $\mathbf{W}^{hidden} \in \mathbb{R}^{d_h \times (n-1)d_r}$ and a bias vector $\mathbf{b}^{hidden} \in \mathbb{R}^{d_h}$, followed by the application of a non-linear function ϕ, that we call an *activation* function:

$$\mathbf{h}_t = \phi(\mathbf{W}^{hidden}\mathbf{x}_t + \mathbf{b}^{hidden})$$

If we stack at least one *hidden* layer between the input and output layers, we obtain a model called the *multilayer perceptron*. It has been shown (Hornik et al., 1989; Cybenko, 1989) that, with at least one hidden layer, and a suitable non-linear activation function, any function continuous on a closed part of $\mathbb{R}^N$ can be approximated - in our case, $N = |\mathcal{V}|$.

Output layer

Now, we need to map the resulting hidden representation $\mathbf{h}_t \in \mathbb{R}^{d_h}$ to a probability distribution on $\mathcal{V}$. This means that we need to give a probability to each word w in the vocabulary: the output of our multilayer perceptron is a vector of $|\mathcal{V}|$ positive values which sum to 1, that we note $\mathbf{P}^{H_t}$. Then,

$$P(w_t = k|H_t) = P_k^{H_t} \qquad \text{and} \qquad \sum_{k=1}^{\mathcal{V}} P_k^{H_t} = 1$$

We proceed with a layer whose activation function is the *softmax* function. First, we extract a score with a linear transformation corresponding to a specific word in the vocabulary. This first step is explicited with an intermediate scoring function s:

$$s(w_t = k, H_t) = s_k^{H_t} = \mathbf{r}_k^{out}\mathbf{h}_t + b_k^{out}$$

The vectors $\mathbf{r}_k^{out}$ each correspond to a word k of the vocabulary, as do the bias values b_k^{out}. They form the rows of a matrix that can be seen as the weight matrix of the output layer, $\mathbf{W}^{out} = [\mathbf{r}_k^{out}]_{k=1}^{|\mathcal{V}|}$, with $\mathbf{W}^{out} \in \mathbb{R}^{|\mathcal{V}| \times d_h}$, and the associated bias vector $\mathbf{b}^{out} \in \mathbb{R}^{|\mathcal{V}|}$. Equivalently, we can see these weights as *output word embeddings*[2]. Then, we make this score positive by applying the exponential function, and transform them into probabilities by normalizing them:

[2]It is possible for the *input* and *output* word embeddings to share the same parameters, which implies $d_r = d_h$. The resulting model is called *log-bilinear model* (Mnih and Hinton, 2007)

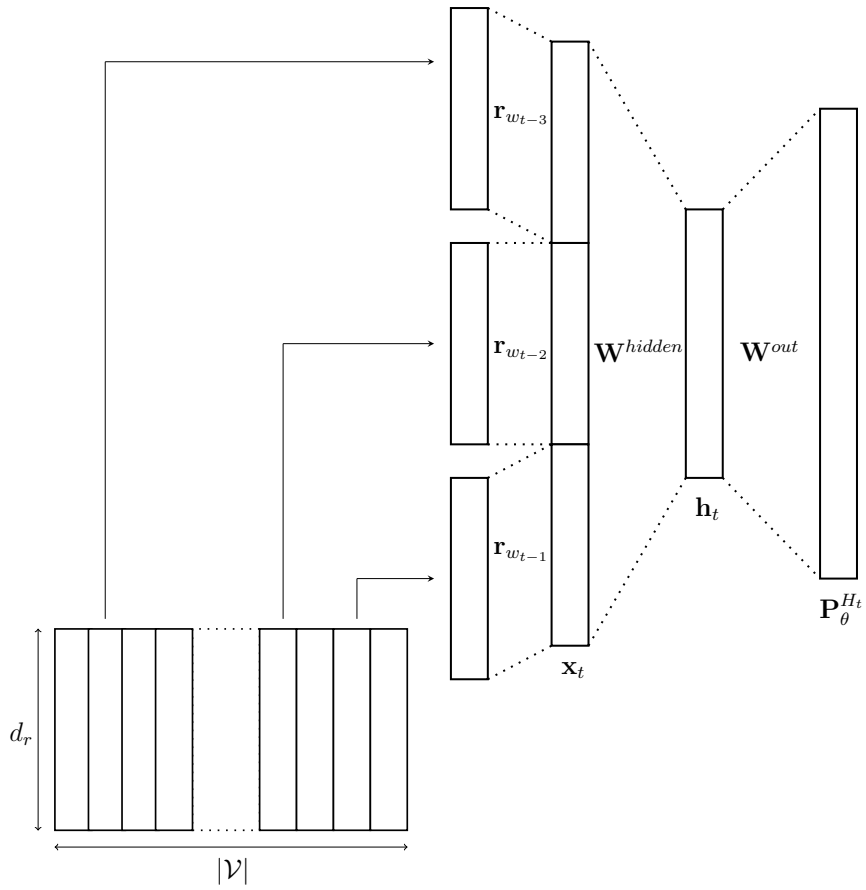

Figure 1.1: Example architecture of a feedforward neural network language model. Here, the context size $n-1=3$, and we only use one hidden layer.

$$P(w_t = k|H_t) = P_k^{H_t} = \frac{e^{s(w_t=k,H_t)}}{\sum_{l=1}^{|\mathcal{V}|} e^{s(w_t=l,H_t)}} = \frac{e^{\mathbf{r}_k^{out}\mathbf{h}_t+\mathbf{b}_k^{out}}}{\sum_{l=1}^{|\mathcal{V}|} e^{\mathbf{r}_l^{out}\mathbf{h}_t+\mathbf{b}_l^{out}}} \tag{1.3}$$

The dimensions d_r, d_h of the word embeddings and hidden representations, and the activation function ϕ, are all *hyperparameters*, that we need to choose. We will discuss hyperparameter choices in more details in Section 1.3.2. The look-up matrix $\mathbf{L}$, the hidden layer weight matrix $\mathbf{W}^{hidden}$ and bias $\mathbf{b}^{hidden}$, and the output layer weight matrix $\mathbf{W}^{out}$ and bias $\mathbf{b}^{out}$ are parameters of the model. We gather them in one parameter vector θ, which is then the set of all the free parameters of our model. The continuous function f we learn depends on θ, and outputs a conditional probability distribution. We can then note it P_θ : it is modeled by a feedforward neural network, composed of an input, hidden, and output layer. If H is fixed, we can note the conditional probability distribution associated as $P_\theta^H(.)$, a function over $\mathcal{V}$, or as a vector $\mathbf{P}_\theta^H \in \mathbb{R}^{|\mathcal{V}|}$.

Learning with gradient descent and backpropagation

In order to learn this function, we perform maximum-likelihood estimation (MLE). The objective is to find the parameters θ which minimize the negative log-likelihood of this conditional distribution for each tuple of input context and following word

$(H, w) \in \mathcal{D}$, where $\mathcal{D}$ is the training set:

$$NLL(\theta) = -\sum_{(H,w)\in\mathcal{D}} \log P_\theta(w|H) \tag{1.4}$$

We can do so with the Stochastic Gradient Descent (SGD). We then need to compute the gradient to update the parameters: in practice, this is done via *backpropagation* (Rumelhart et al., 1988). For one training example (H, w), such an update has the form:

$$\theta_{updated} = \theta - \lambda \frac{\partial}{\partial\theta} \log P_\theta(w|H) \tag{1.5}$$

where λ is the learning rate, which can be treated as an hyperparameter. Its value can be fixed or can change during training according to various strategies that we will discuss in Section 1.3.2.

1.2.2 Recurrent neural network language models

One of the disadvantage of the conventional feedforward network is that the length of the input context is fixed to $n-1$ words. The idea of *recurrent* neural networks is to use a structure adapted to working with sequences, by creating at each time step a fixed-sized vector that represents all the preceding words. The idea of recurrent neural network was introduced by Elman (1990a) and applied to language modelling by Mikolov et al. (2010).

Basic recurrent layer

Our input sequence is still $S = (w_1, \ldots, w_t, \ldots, w_{|S|})$. The main difference with the feedforward model is that we compute for each time step t a hidden state $\mathbf{h}_t$ corresponding to a word w_t. This state is computed using the previous hidden state and the computed representation $\mathbf{r}_{w_t}$, coming from the input layer. Hence,

$$\mathbf{h}_t = \phi(\mathbf{W}^{hidden}\mathbf{h}_{t-1} + \mathbf{W}^{input}\mathbf{r}_{w_t})$$

where $\mathbf{W}^{hidden}$ and $\mathbf{W}^{input}$ are weights matrices, applying linear transformations respectively to the previous hidden state and the input representation. At each time step, the output conditional probability distribution (which is now conditioned on all the previous words in the sequence) is obtained in the same way as for the feedforward model:

$$P(w_t = k|w_1, \ldots, w_{t-1}) = \frac{e^{\mathbf{r}_k^{out}\mathbf{h}_t + \mathbf{b}_k^{out}}}{\sum_{l=1}^{|\mathcal{V}|} e^{\mathbf{r}_l^{out}\mathbf{h}_t + \mathbf{b}_l^{out}}} \tag{1.6}$$

While the first occurrence of the model (Mikolov et al., 2010) was trained with *truncated backpropagation*, it was later (Mikolov et al., 2011) improved with *backpropagation through time* (Werbos, 1990). The first one consists in applying parameter

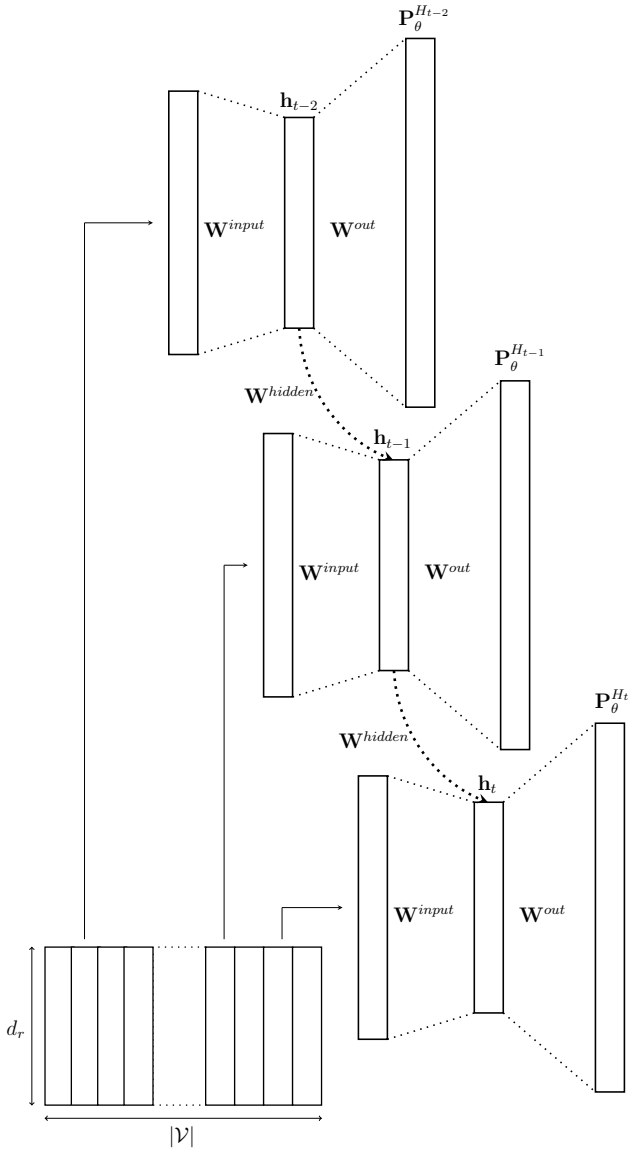

Figure 1.2: Example architecture of a recurrent neural network language model, unrolled through 3 sucessive time steps.

updates using only the gradients from the current time step, while the second allows propagating the gradients through several time steps. That number is usually limited by a fixed threshold. However, there are some issues with this kind of structures. First, it is not possible to choose what to keep and what to skip in a hidden state. It is probable that information will not be kept for very long. Secondly, even if we rely on the backpropagation through time to make the model keep relevant information, in practice, it causes *vanishing* and *exploding* gradients (Bengio et al., 1994; Pascanu et al., 2013). While it is possible to clip the gradient to deal with the latter, the former often implies that the model ignores long-term dependencies, since the gradient vanishes after a few time steps. Hence, a conventional recurrent neural network language model may not be able to use longer context than usual n-grams models.

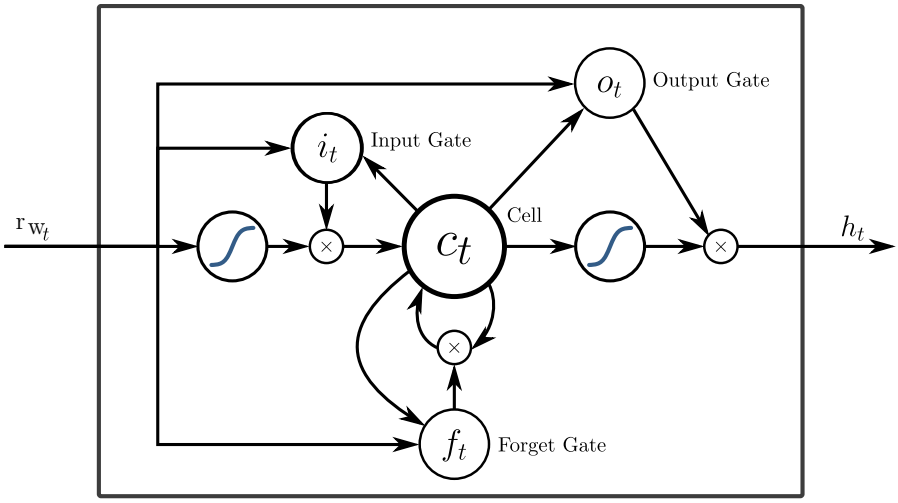

Figure 1.3: Detailed working of a LSTM Unit; see Equations 1.7.

LSTM and other improvements

The most widely used improvements on the recurrent structure are the *Long Short-Term Memory* units, or LSTMs (Hochreiter and Schmidhuber, 1997). It was specifically designed to be able to retain these long-term dependencies: through the use of gates, it allows selecting the information that is retained and transmitted. It also scales the descending gradient to avoid vanishing and exploding gradients when backpropagating through time:

$$\begin{aligned}
\mathbf{t}_t &= \phi(\mathbf{W}_t^{hidden}\mathbf{h}_{t-1} + \mathbf{W}_t^{input}\mathbf{r}_{w_t}) \\
\mathbf{f}_t &= \phi(\mathbf{W}_f^{hidden}\mathbf{h}_{t-1} + \mathbf{W}_f^{input}\mathbf{r}_{w_t}) \\
\mathbf{o}_t &= \phi(\mathbf{W}_o^{hidden}\mathbf{h}_{t-1} + \mathbf{W}_o^{input}\mathbf{r}_{w_t}) \\
\mathbf{z}_t &= \phi(\mathbf{W}_z^{hidden}\mathbf{h}_{t-1} + \mathbf{W}_z^{input}\mathbf{r}_{w_t}) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{t}_t \odot \mathbf{z}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \phi(\mathbf{c}_t)
\end{aligned} \tag{1.7}$$

which can summarized with:

$$\mathbf{h}_t = LSTM(\mathbf{h}_{t-1}, \mathbf{r}_{w_t})$$

LSTM were first applied to language modeling in Sundermeyer et al. (2012), and compared to feedforward neural networks in Sundermeyer et al. (2013). Another popular recurrent structure, using gates to retain or forget information, is the *Gated Recurrent Unit*, or GRU (Cho et al., 2014c), which were compared to LSTM in Chung et al. (2014). In this work, we chose to use LSTMs for our models.

1.3 Practical considerations

Even if we possess all the basic blocks that would allow us to build our language model, many hyperparameters must be chosen. They include architectural choices,

such as the number and type of layers, as well as their sizes, but also the learning procedure, the type of regularization, and other tricks that can help training. We will first explain how we evaluate our models, on which data, then what choices we need to consider, before moving on to our main topic.

1.3.1 Evaluation

When used for a specific application, the performance of a language model can be measured by how much it improves the system it is a part of, which is an *extrinsic* evaluation. However, to evaluate a language model on its own (an *intrinsic* evaluation), the most common used method is *perplexity*. It is the geometric average of the inverse probability the model assigns to the words, and is usually measured on a separate test dataset. For a sequence of N words,

$$PPL = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_t|H_t)}} \tag{1.8}$$

Equivalently, it can be described as the exponential of the cross-entropy of the test data, given the model. Intuitively, it is a measure of how informed the model is. Thus, a model attributing uniform probabilities to each word in a vocabulary $\mathcal{V}$ would have a perplexity of $|\mathcal{V}|$, while a model attributing a probability of 1 to the right word each time would have a perplexity of 1.

However, we should remark that perplexity has drawbacks. First, it is intrinsic, and measuring the effect of the model on the task it is applied to is arguably far more important. Secondly, it is the exponential of entropy, which means perplexity improvements which seem meaningful could mean only very small entropy changes. This issue is further discussed in Goodman (2001b): the author argues that the speed and space taken by a model are often overlooked, while the gains obtained on perplexity are not always worth their cost. They also may offer almost no gain on other measures, as the word error rate. Another aspect that is sometimes overlooked is how important the quantity and quality of data available for training are, in order to obtain a reasonable performance. However, while it is relatively easy to train discrete models on large datasets, large-scale neural language models are very costly in computation time and memory.

Datasets

One of the most used benchmarks in neural language modeling is the Penn TreeBank (PTB) (Marcus et al., 1993) corpus. As it is rather small - $929K$ training words, $73K$ validation words, and $82K$ test words - experimenting with it is not too demanding in resources. However, as argued in Józefowicz et al. (2016), with the increasing availability of data and computational resources, models should be tested on larger corpora, since a good performance on a small dataset may not hold on a larger one. The 1 Billion Word Benchmark dataset (Chelba et al., 2014) is an open dataset commonly used as a way to compare progress of large language models. About 1% of the dataset was held-out, shuffled and divided in 50 partitions that

can be used as validation/test data. As we are limited in resources, we will use the PTB for most of our experiments in this work, while we will verify our results on the 1 Billion Word Benchmark when possible, using the first and second partitions as validation and test datasets.

1.3.2 Choosing hyperparameters

The first choices to consider are the number of hidden layers (for both feedforward and recurrent models) and their dimensions, as well as the dimension of word representations. Of course, higher dimensional representations are more expressive, but also more costly to use[3]. These choices should also depend on the amount of data available. Similarly, using several stacked hidden layers can improve results. But not only is it more costly, it can also make training far more difficult, because of the phenomenon of gradient attenuation mentioned in Section 1.2.2. Although this issue was first handled by using specific initialization schemes (Glorot and Bengio, 2010; Saxe et al., 2013), one possible solution is to use *highway* layers (Srivastava et al., 2015), which regulate the information flow (and can allow propagation of the gradient) with a gating mechanism.

The choice of the activation function also impacts heavily how easily the information circulates in the network: while the *sigmoid* and *tanh* were traditionally used, they easily saturate, which makes their gradient very small. The *rectified linear unit* (Glorot et al., 2011), or ReLU, which is simply the function $f(u) = max(0, u)$, is currently the most popular activation function. Being linear when non-zero, it does not saturate, and its computation is faster.

Optimization, Regularization and Initialization

The learning scheme refers to the choice of the learning rate λ used to update parameters (in Equation 1.5). It is crucial, and using strategies like iteratively drecreasing λ after seeing enough examples facilitates learning. However, we can use algorithms that adapt the learning rate to each parameter, like *Adagrad* (Duchi et al., 2011), which uses a cache of accumulated past gradients to normalize the learning rate. One of the most used method is *Adam* (Kingma and Ba, 2014), which combines several strategies including Adagrad.

Overfitting is a common issue, and especially is of concern for smaller datasets. While we could use the usual $L1$ and $L2$ regularization constraints, a neural-network specific method, *dropout*, was recently introduced by Srivastava et al. (2014). Dropout keeps neurons active with a probability smaller than 1 during training, which has the effect of sampling a sub-network. While it was only efficient for feedforward networks at first, Zaremba et al. (2014) applied it successfully to LSTMs.

How the parameters are initialized is quite important, and can drastically affect training. The initialization scheme should be chosen in accordance with the activation function. Glorot and Bengio (2010) make the analysis that the neurons in the

[3]For example, see the results of Józefowicz et al. (2016) to note the influence of dimension choices on perplexity.

network should have similar output distributions, and provides advice on doing so, while He et al. (2015) derive an initialization scheme specifically for ReLUs. However, a recent method called *batch normalization* (Ioffe and Szegedy, 2015), which we will use, normalizes the outputs of activation function into a unit Gaussian distribution, rendering initialization less impactful.

1.3.3 The computational bottleneck

The main limitation of neural networks language models is their expensive computational cost. This was already pointed out by Bengio et al. (2003). Schwenk and Gauvain (2004) provide, for the feedforward model, the number of operations needed to obtain the output distribution:

$$((n-1) \times d_r + 1) \times d_h + (d_h + 1) \times |\mathcal{V}| \tag{1.9}$$

Since, for a language model, the quantities d_r and d_h verify $d_r \ll |\mathcal{V}|$ and $d_h \ll |\mathcal{V}|$, the computational cost mainly depends on the size of the vocabulary, with which it grows linearly. Early work on neural network language models proposed a certain number of training tricks to accelerate training, among which using a *mini-batch*, *i.e* parallelizing computation by forwarding (and backpropagating the gradient of) multiple examples at the same time, instead of just one. In practice, it is very efficient, and it only requires transforming the vectors $\mathbf{x}$, $\mathbf{h}$ and $\mathbf{P}$ of Section 1.2 into matrices. However, the most straightforward way to limit computational time is to limit the vocabulary size. Bengio et al. (2003) proposed to remove words under a frequency threshold of the vocabulary, and to map them to a same token *UNK*, which represents all unknown words. That is what we will do when we do not use the full training vocabulary. An improvement on that idea is the *short-list*, (Schwenk and Gauvain, 2004) which limits the neural language model vocabulary size, and uses the probabilities obtained from a discrete language model for the remaining words. However, this solution limits the model capacity to generalize to less frequent words.

While the training and inference costs of a neural language model are always making their practical use difficult, we will in the next chapter mainly be interested in reducing them when they are the most impactful: for language models that use a large vocabulary.

Chapter 2

Avoiding direct normalization: Existing strategies

Contents

The main drawback of Neural Probabilistic Language Models is their very long computation time. At the root of this issue lies the multinomial classification objective: its computation time is linear in the number of categories, which is the size of the target vocabulary. To understand why, we need to take a closer look at the objective function and its gradient. Our neural probabilistic language model with parameters θ outputs, for an input context H, a conditional distribution P_θ^H for the next word, over the vocabulary $\mathcal{V}$. As explained in Section 1.2, this conditional distribution is defined using the *softmax* activation function:

$$P_\theta(w|H) = \frac{e^{s_\theta(w,H)}}{\sum_{w' \in \mathcal{V}} e^{s_\theta(w',H)}} = \frac{e^{s_\theta(w,H)}}{Z_\theta(H)} \tag{2.1}$$

Here, we are not interested in the network architecture and we suppose that the model outputs a scoring function $s_\theta(w, H)$ which depends on it. The denominator is the partition function $Z_\theta(H)$, which is used to ensure that for each input context H, output scores are normalized into a probability distribution. As explained in Section 1.2.1, the objective is to minimize the negative log-likelihood of this conditional distribution for each tuple of input context and following word $(H, w) \in \mathcal{D}$ in the training data:

$$NLL(\theta) = -\sum_{(H,w)\in\mathcal{D}} \log P_\theta(w|H) \tag{2.2}$$

Using the Stochastic Gradient Descent (SGD) to train this objective implies taking the objective gradient to make the parameter updates. For one training example (H, w), the gradient of the log-probability is computed as follows:

$$\frac{\partial}{\partial\theta} \log P_\theta(w|H) = \frac{\partial}{\partial\theta} s_\theta(w, H) - \sum_{w' \in \mathcal{V}} P_\theta(w'|H) \frac{\partial}{\partial\theta} s_\theta(w', H) \tag{2.3}$$

With the first term, we are increasing the conditional log-likelihood of the word w, while we are decreasing the conditional log-likelihood of all the other words in the vocabulary with the second. Unfortunately, explicitly computing $P_\theta(w'|H)$ implies computing the partition function $Z_\theta(H)$, which means summing over the whole vocabulary $\mathcal{V}$.

While we described in Section 1.3.3 a classic workaround consisting in using a *short-list* that limits the output layer of the network to the most frequent words, the model will not be able to generalize well to words out of this list. In this chapter, we will describe methods that can be used to efficiently handle large vocabularies in language modeling. We will expose the theoretical guaranties they offer, the hyper-parameter choices they entail and their experimental advantages and weaknesses. Most of these methods revolve around avoiding the computation of the partition function $Z_\theta(H)$. Thus, an important concept in this chapter (and the remaining of this dissertation) is the *un-normalized* version of a probability distribution. This is simply the exponential of the scoring function and we will denote it as a lower-case version of the normalized probability distribution:

$$p_\theta(w|H) = e^{s_\theta(w,H)} = P_\theta(w|H) Z_\theta(H) \tag{2.4}$$

We will rapidly present *Hierarchical* approaches (Section 2.1), which reorganize the structure of the output layer into a tree structure. They provide a potential $\mathcal{O}(\log|\mathcal{V}|)$ speedup, but at the cost of building the word hierarchy that will replace $\mathcal{V}$. The main other approaches, that we will examine in detail, use sampling schemes to avoid computing the partition function. While *Importance Sampling* (Section 2.2) directly approximates the gradient computation, *Noise Contrastive Estimation* (Section 2.3.1) and related methods (Section 2.3.2 and 2.3.3) use a discriminative objective function that doesn't require normalization and let the partition function be parametrized separately. We will also present *Constrained self-normalization* (Section 2.4), where the partition function is explicitly forced to be close to 1 in the objective function, which is efficient for some specific applications.

2.1 Hierarchical language models

To avoid the costly summations in the multinomial classification objective, we can modify the output layer to make this classification hierarchical. We first predict a (series of) class or cluster and, given that class, predict the following word. While class-based models have been used extensively in language modeling (Brown et al., 1992; Kneser and Ney, 1993), they aimed at improving the model perplexity, or reducing its size. Decomposing the vocabulary into classes in order to get a speed-up in computation was first applied by Goodman (2001a), to a Maximum entropy language model. An extension of this idea was then applied to Neural probabilistic language models, first by Morin and Bengio (2005). Here, the prediction of a word from a vocabulary $\mathcal{V}$ is replaced by a series of $\mathcal{O}(\log|\mathcal{V}|)$ decisions. That process can be seen as a hierarchical decomposition of the vocabulary, following a binary tree structure. Indeed, by building a binary hierarchical clustering of the vocabulary, we can represent each word as a bit vector $\mathbf{b}(w) = (b_1(w), \ldots, b_m(w))$ and compute the conditional probability of the next word as:

$$P_\theta(w|H) = \prod_{i=1}^{m} P_\theta(b_i(w)|b_{i-1}(w), \ldots, b_1(w), H) \tag{2.5}$$

Then, the full vocabulary normalization is replaced by $\mathcal{O}(\log|\mathcal{V}|)$ binary normalizations. The main limitation, however, is that we need to build the tree of words, which is here done from prior knowledge. In Mnih and Hinton (2009), the procedure used to build this tree is automated and data-driven, using various feature based algorithms on the training set. It allows the model to outperform its non-hierarchical equivalents. Other work (Mikolov et al., 2011) used frequency to build simple classes, while Zweig and Makarychev (2013) explicitly minimizes the runtime of a class-based language model using dynamic programming. Another approach (Le et al., 2011) uses a *short-list* for the most frequent words, followed by hierarchical clustering for the rest of the vocabulary. Using the short-list avoids the drop of performance observed when all clusters are put in the leaves of the tree. Combining these last two strategies, a recently introduced method (Grave et al., 2017), the *adaptive softmax*, also takes into account the modern matrix multiplication computation time on GPU.

2.2 Importance Sampling

Importance sampling (Owen, 2013) is a general sampling technique used when we want to estimate a distribution using samples from a different distribution. If we have a random variable X, a function $f(X)$ of X with an unknown distribution and we want to estimate the expected value of f with respect to a distribution $\mathcal{P}$, the ordinary Monte-Carlo estimator of $\mu = \mathbb{E}_{\mathcal{P}}[f(X)]$ is

$$\hat{\mu}_{MC} = \frac{1}{k} \sum_{\substack{i=1 \\ x_i \sim \mathcal{P}}}^{k} f(x_i) \tag{2.6}$$

However, when we can't sample from $\mathcal{P}$ but have access to a proposal distribution $\mathcal{Q}$, we notice that by introducing the notation:

$$\rho(X) = \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}$$

we can obtain the following re-writing of the expectation:

$$\mathbb{E}_{\mathcal{P}}[f(X)] = \sum_{x \in \mathcal{X}} \mathcal{P}(x) f(x) = \sum_{x \in \mathcal{X}} \rho(x) \mathcal{Q}(x) f(x) = \mathbb{E}_{\mathcal{Q}}[\rho(X) f(X)] \tag{2.7}$$

provided that $\mathcal{Q}(x) > 0$ where $\mathcal{P}(x) f(x) \neq 0$. Then, our estimator is:

$$\hat{\mu}_{IS} = \frac{1}{k} \sum_{\substack{i=1 \\ x_i \sim \mathcal{Q}}}^{k} \rho(x_i) f(x_i) \tag{2.8}$$

This estimator can easily be shown to be consistent (which means that as the number of available samples grows, the sequence of estimates converges in probability towards the true value) and unbiased (the expected value of the estimator is the true value of the parameter) (Owen, 2013). Using this strategy, we an concentrate the sampling effort on important regions of the sampling space, hence the name *importance sampling*.

2.2.1 Application to Language Modeling

Importance sampling was first applied to neural probabilistic language models in Bengio and Sénécal (2003). The idea is to rewrite the gradient of the objective presented in Equation 2.3 as:

$$\begin{aligned} \frac{\partial \log P_\theta(w|H)}{\partial \theta} &= \frac{\partial}{\partial \theta} s_\theta(w, H) - \sum_{w' \in \mathcal{V}} P_\theta(w'|H) \frac{\partial}{\partial \theta} s_\theta(w', H) \\ &= \frac{\partial}{\partial \theta} s_\theta(w, H) - \mathbb{E}_{w' \sim P_\theta^H} \left[\frac{\partial}{\partial \theta} s_\theta(w', H) \right] \end{aligned} \tag{2.9}$$

The second term is an expectation with respect to P_θ^H that can be estimated using importance sampling with a proposal distribution $\mathcal{Q}$. From Equation 2.7:

$$\begin{aligned}\frac{\partial}{\partial\theta}\log P_\theta(w|H) &= \frac{\partial}{\partial\theta}s_\theta(w,H) - \sum_{w'\in\mathcal{V}}\mathcal{Q}(w')\frac{P_\theta(w'|H)}{\mathcal{Q}(w')}\frac{\partial}{\partial\theta}s_\theta(w',H)\\ &= \frac{\partial}{\partial\theta}s_\theta(w,H) - \mathbb{E}_{w'\sim\mathcal{Q}}\left[\frac{P_\theta(w'|H)}{\mathcal{Q}(w')}\frac{\partial}{\partial\theta}s_\theta(w',H)\right]\end{aligned} \tag{2.10}$$

And, from Equation 2.8, we get the gradient estimate:

$$\frac{\partial}{\partial\theta}\log P_\theta(w|H) \approx \frac{\partial}{\partial\theta}s_\theta(w,H) - \frac{1}{k}\sum_{\substack{i=1\\ \hat{w}_i\sim\mathcal{Q}}}^{k}\frac{P_\theta(\hat{w}_i|H)}{\mathcal{Q}(\hat{w}_i)}\frac{\partial}{\partial\theta}s_\theta(\hat{w}_i,H) \tag{2.11}$$

However, the reason we are using sampling is to avoid computing the partition function $Z_\theta(H)$: we can only compute the un-normalized version of P_θ^H.

Self-normalized Importance Sampling

As described in Bengio and Sénécal (2003), we need to use a different version of importance sampling, that is called *ratio* or *self-normalized importance sampling* (RIS). The idea is to normalize the weights, which we can do by applying classical importance sampling to the partition function. Indeed, if our distribution $\mathcal{P}$ can only be estimated up to its partition function, which means we only have access to p with $p = Z\mathcal{P}$, we can write:

$$Z = \sum_{x\in\mathcal{X}}p(x) = \sum_{x\in\mathcal{X}}\omega(x)\mathcal{Q}(x) = \mathbb{E}_{\mathcal{Q}}[\omega(X)] \tag{2.12}$$

with

$$\omega(x) = \frac{p(x)}{\mathcal{Q}(x)} \tag{2.13}$$

Using the same distribution $\mathcal{Q}$, to be able to re-use the samples, we obtain the estimator for Z:

$$\Omega = \sum_{\substack{i=1\\ x_i\sim\mathcal{Q}}}^{k}\omega(x_i) \tag{2.14}$$

Which gives us the estimator for μ,

$$\hat{\mu}_{RIS} = \sum_{\substack{i=1\\ x_i\sim\mathcal{Q}}}^{k}\frac{1}{\Omega}\omega(x_i)f(x_i) \tag{2.15}$$

Using the strong law of large numbers, we can show that the estimator $\hat{\mu}_{RIS}$ converges to the wanted expectation almost surely and is therefore consistent. However, we must have $\mathcal{Q}(w) > 0$ when $p > 0$, which is a condition a little stronger than for

the classic importance sampling estimator $\hat{\mu}_{IS}$. Besides, $\hat{\mu}_{RIS}$ is biased in $\mathcal{O}(\frac{1}{k})$ (Hesterberg, 2003) — which makes it asymptotically unbiased.

To apply this estimator to our problem, we now write:

$$\omega(\hat{w}) = \frac{p_\theta(\hat{w}|H)}{\mathcal{Q}(\hat{w})} \tag{2.16}$$

and we obtain the following gradient update:

$$\begin{aligned} \frac{\partial}{\partial\theta}\log P_\theta(w|H) &= \frac{\partial}{\partial\theta}s_\theta(w,H) - \mathbb{E}_{w'\sim\mathcal{Q}}\left[\frac{1}{Z_\theta(H)}\frac{p_\theta(w'|H)}{\mathcal{Q}(w')}\frac{\partial}{\partial\theta}s_\theta(w',H)\right] \\ &\approx \frac{\partial}{\partial\theta}s_\theta(w,H) - \sum_{\substack{i=1\\ \hat{w}_i\sim\mathcal{Q}}}^{k}\frac{1}{\Omega}\omega(\hat{w}_i)\frac{\partial}{\partial\theta}s_\theta(\hat{w}_i,H) \end{aligned} \tag{2.17}$$

Number of samples and choice of $\mathcal{Q}$

The variance of the ratio estimator is depending on the sample size in $\mathcal{O}(\frac{1}{k})$ (Hesterberg, 2003): while both the bias and the variance can be reduced by augmenting the number of samples drawn from $\mathcal{Q}$, it might be costly to do so. Another issue is that some of the weights ratio

$$r_i \approx \frac{\omega(\hat{w}_i)}{\Omega}$$

can become disproportionally large if the value of $\mathcal{Q}(\hat{w}_i)$ is too small. A weight vastly larger than the others would override all other weights and mimic a situation where we have only sampled one observation. This can be measured using the *effective sample size*(ESS):

$$ESS = \frac{\left(\sum_{i=1}^{k} r_i\right)^2}{\sum_{i=1}^{k} r_i^2} \tag{2.18}$$

which will be largely smaller than k if weights are unbalanced. We can also look at the expression of the variance (Hesterberg, 2003) to understand this issue:

$$Var(\hat{\mu}_{RIS}) = \frac{1}{k}\sum_{i=1}^{k}\frac{P_\theta(\hat{w}_i|H)^2(\frac{\partial}{\partial\theta}s_\theta(\hat{w}_i,H) - \mu\mathcal{Q}(\hat{w}_i))^2}{\mathcal{Q}(\hat{w}_i)} \tag{2.19}$$

It is clear that having $\mathcal{Q}$ too close to 0, especially when P_θ^H is not, leads to a high variance, while having $\mathcal{Q}$ approximately proportional to P_θ^H for most w leads to a smaller variance.

Experimentally, Bengio and Sénécal (2003), while using the unigram distribution as $\mathcal{Q}$, found that a sample size too low would cause divergence as training progresses:

they used the ESS to monitor the sample size and if needed increase it during training. However, as the training advances, the number of samples necessary increases rapidly. This behavior is conjectured to be caused by the fact that the model distribution becomes increasingly complex and diverges from the unigram distribution in Bengio and Sénécal (2008). The authors try to switch to more complex distributions for $\mathcal{Q}$ (interpolated bigram and trigram distributions), but it gave poorer results - which seems to be caused by the fact that these distributions are very different than those learned with neural networks. They then force their distribution $\mathcal{Q}$ to *adapt* by redistributing probability mass in order to track P_θ^H. This is done by interpolating a series of n-gram models of different orders. As a result, the number of samples needed seems to grow only linearly with time, instead of exponentially.

2.2.2 Target Sampling

A more recent approach, Jean et al. (2015), applied to machine translation, partitions the training corpus and defines a subset of the vocabulary to sample from for each partition, which contains at least all target words in the partition. Formally, we obtain separate distributions $\mathcal{Q}_i$ for each of these partitions i:

$$\mathcal{Q}_i(\hat{w}) = \begin{cases} \frac{1}{|\mathcal{V}_i|} & \text{if } \hat{w} \in \mathcal{V}_i \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Using uniform probabilities on a subset of the vocabulary simplifies the estimator since the importance weights all have the same denominator. Since we choose this subset to contain at least all the target words, the necessary condition for consistency described in 2.2.1 is verified. We obtain a gradient update that is the same we would get from a softmax applied only on the set of words sampled from $\mathcal{Q}_i$:

$$P_\theta(w|H) \approx \frac{\exp s_\theta(w, H)}{\sum\limits_{\substack{i=1 \\ \hat{w}_i \sim \mathcal{Q}_i}}^{k} \exp s_\theta(\hat{w}_i, H)} \tag{2.21}$$

Using only subsets of the full vocabulary with uniform distributions also allows for better computational efficiency.

2.2.3 Complementary Sum-Sampling

The recent work of Botev et al. (2017) shows that the high variance in the standard importance sampling estimation is simply due to not including the correct class into the set of samples. They show that by summing over a subset of classes (including the correct one) to which we add the samples from $\mathcal{Q}$, we dramatically reduce the estimation error at no additional cost.

2.3 Density estimation as a classification task: discriminative objectives

Noise Contrastive Estimation (NCE) was first described in Gutmann and Hyvärinen (2010, 2012), as a way of estimating a parametric probabilistic model from observed data, in the case where the probability function of the model is *un-normalized*. The first idea is to consider the partition function Z as a separate parameter, instead of a value dependent on all the other parameters θ. Then, a parametrized distribution P_θ is decomposed as:

$$\log P_\theta = \log p_{\theta_0} + c$$

with parameters $\theta = (\theta_0, c)$. Here, $c = -\log Z$ gives the un-normalized model proper scaling, while the other parameters make the shape of the model match the shape of the data density distribution. However, estimating separately θ_0 and c is not possible with maximum-likelihood estimation, since we can simply choose c to be as large as we want to increase likelihood. The authors then propose an objective function which mimics maximum-likelihood estimation by learning to discriminate between examples from data or generated from a *noise distribution*. This method has been applied to language modeling, as well as other approaches which also use discriminative objectives and that will be described subsequently.

2.3.1 Noise Contrastive Estimation

With NCE, we learn the relative description of the data distribution $P_\mathcal{D}$ to a reference noise distribution P_n, by learning their ratio $P_\mathcal{D}/P_n$. This ratio is learned by discriminating between the two distributions. Concretely, we draw k samples from the noise distribution for each tuple $(H, w) \in \mathcal{D}$ and optimize our model to perform a classification task between them. We can consider our example as coming from the following mixture:

$$\frac{1}{k+1}P_\mathcal{D} + \frac{k}{k+1}P_n \tag{2.22}$$

Since we don't have access to $P_\mathcal{D}$ but want to approach it with our model of parameters θ, we can consider the conditional class probabilities as:

$$P(w|C=1,H) = P_\theta(w|H) \quad \text{and} \quad P(w|C=0,H) = P_n(w|H) \tag{2.23}$$

which gives the posterior class probabilities:

$$P(C=1|w,H) = \frac{P_\theta(w|H)}{P_\theta(w|H) + kP_n(w|H)} \tag{2.24}$$

and

$$P(C=0|w,H) = \frac{kP_n(w|H)}{P_\theta(w|H) + kP_n(w|H)} \tag{2.25}$$

which can be rewritten as:

$$P(C = 1|w, H) = \sigma\left(\log \frac{1}{k} \frac{P_\theta(w|H)}{P_n(w|H)}\right) \tag{2.26}$$

The reformulation obtained in Equation 2.26 shows that training a classifier based on a logistic regression estimates the log-ratio of the two distributions: we learn P_θ relatively to P_n. Since we assume the class labels to be Bernouilli distributed and independent, the classification objective is given by maximizing the log-likelihood of the true examples to belong to class $C = 1$ and the noise samples $(\hat{w}_i)_{1 \leq i \leq k}$ to $C = 0$, which is, for one example (H, w) from $\mathcal{D}$:

$$J_\theta^H(w) = \log \frac{P_\theta(w|H)}{P_\theta(w|H) + kP_n(w|H)} + \sum_{i=1}^{k} \log \frac{kP_n(\hat{w}_i|H)}{P_\theta(\hat{w}_i|H) + kP_n(\hat{w}_i|H)} \tag{2.27}$$

In order to obtain the global objective to maximize, we sum on all examples $(H, w) \in \mathcal{D}$:

$$J_\theta = \sum_{H,w \in \mathcal{D}} J_\theta^H(w) \tag{2.28}$$

This objective is proven to reach a maximum at $P_{\theta^*} = P_\mathcal{D}$, which is unique if we have that $P_n > 0$ whenever $P_\mathcal{D} > 0$ (Gutmann and Hyvärinen, 2010, 2012). Besides, the authors prove that the optimal parameters θ_T^* obtained with an objective J_θ^T restricted to a sample T of the training data converge in probability towards θ^* when T is large enough, assuming constraints on the model that are similar to those used in Maximum Likelihood estimation. The estimation error also behaves comparably to the estimation error of MLE: it is asymptotically following a normal distribution.

Application to language modeling

This objective does not impose any normalization constraint on our model: as indicated earlier, it is possible to estimate an un-normalized distribution p_{θ_0}, by parametrizing the partition function independently. However, as described by Mnih and Teh (2012), who first applied NCE to language modelling, this parametrization is context-dependent and we have:

$$\log P_\theta^H = \log p_{\theta_0}^H + c^H \tag{2.29}$$

However, learning an additional parameter by context gets very costly as the context size increases. To avoid it, the authors argue that these context-dependent parameters c^H can be put to zero, and that given the number of free parameters, the output scores for each context $p_{\theta_0}^H$ self-normalize, which is verified in their experiments.

The gradient update is the following:

$$\frac{\partial}{\partial\theta} J_\theta^H(w) = \frac{kP_n(w|H)}{P_\theta(w|H) + kP_n(w|H)} \frac{\partial}{\partial\theta} \log P_\theta(w|H) - \sum_{i=1}^{k} \left[\frac{P_\theta(\hat{w}_i|H)}{P_\theta(\hat{w}_i|H) + kP_n(\hat{w}_i|H)} \frac{\partial}{\partial\theta} \log P_\theta(\hat{w}_i|H) \right] \tag{2.30}$$

We can note (Mnih and Teh (2012)) that this gradient converges to the maximum likelihood estimation gradient as the number of samples k grows. Also, one advantage of the NCE over importance sampling is that the weights used here are contained between 0 and 1, for any noise distribution P_n.

Choosing the noise distribution

Gutmann and Hyvärinen (2010, 2012) offer theoretical results on the impact of the noise distribution: the estimation error of parameters θ is asymptotically independent of P_n when the ratio of noise sample by example k coming from the data is large enough. They also show that having both a noise distribution close to the data distribution and a high number of samples k will lead to a better estimation error. We can then consider the choice of the noise distribution as a trade-off between using a large number of noise samples or using a noise distribution closer to our data distribution.

Mnih and Teh (2012) compared the use of the uniform and unigram distribution in their experiments, finding the unigram distribution to give far more accurate results. In the literature, NCE was then mainly used in the context of machine translation: Vaswani et al. (2013); Baltescu and Blunsom (2015) report results with the unigram distribution, while Zoph et al. (2016) used an uniform noise. Chen et al. (2015) applied NCE to speech recognition, with the unigram distribution, but fixed the parameters c^H to 9 instead of zero, choosing this value to be close to the log-partition function at initialization, as a trade-off between performance and convergence speed. Chen et al. (2016) highlighted the inconsistency of NCE training (using unigram noise) when dealing with very large vocabularies, showing very different perplexity results for close loss values.

2.3.2 BlackOut

BlackOut (Ji et al., 2015) is a more recent method using a discriminative objective, with ties to both NCE and Importance sampling. We can describe this approach from two different points of view.

NCE with a re-weighted noise distribution

First, we can see it as applying NCE with a different noise distribution, that we obtain by re-weighting samples from a distribution $\mathcal{Q}$, using the ratio between the

learned and sampling densities. As for NCE, we can work with the un-normalized model density p_θ. This re-weighted noise distribution is:

$$P_n(w|H) = \frac{1}{k} \sum_{\substack{i=1 \\ \hat{w}_i \sim \mathcal{Q}}}^{k} \frac{\mathcal{Q}(w)}{\mathcal{Q}(\hat{w}_i)} p_\theta(\hat{w}_i|H) \tag{2.31}$$

If we note the set of samples $\mathcal{S}_k = \{\hat{w}_i \sim \mathcal{Q}\}_{i=1}^k$ and the inverse probabilities $q(w) = \frac{1}{\mathcal{Q}(w)}$ we obtain the following posterior probabilities:

$$P(C = 1|w, H) = \frac{q(w)p_\theta(w|H)}{q(w)p_\theta(w|H) + \sum_{\hat{w}_i \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)} \tag{2.32}$$

We obtain the objective of importance sampling, similar to a sampled softmax, when we directly maximize the likelihood of the log-ratio $\log \frac{\omega(w)}{\Omega}$. Here, instead, we apply a logistic regression:

$$P(C = 0|w, H) = \frac{\sum_{\hat{w} \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)}{q(w)P_\theta(w|H) + \sum_{\hat{w} \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)} \tag{2.33}$$

Using the same process as for NCE, we obtain the discriminative objective:

$$\begin{aligned} J_\theta^H(w) = \log \left(\frac{q(w)p_\theta(w|H)}{q(w)p_\theta(w|H) + \sum_{\hat{w} \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)} \right) \quad + \\ \sum_{w' \in \mathcal{S}_k} \log \left(\frac{\sum_{\hat{w} \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)}{q(w')p_\theta(w'|H) + \sum_{\hat{w} \in \mathcal{S}_k} q(\hat{w})p_\theta(\hat{w}|H)} \right) \end{aligned} \tag{2.34}$$

This objective offers the same theoretical guaranties as the NCE, since the authors prove the noise distribution $P_n^{\mathcal{S}_k}$ to be a valid probability distribution. The only infringement to the NCE procedure is that the samples used in the objective are not sampled from $P_n^{\mathcal{S}_k}$ but from $\mathcal{Q}$ (since we use them to build $P_n^{\mathcal{S}_k}$): however, the expected value of $P_n^{\mathcal{S}_k}(w|H)$ when we sample $\mathcal{S}_k$ from $\mathcal{Q}$ verifies:

$$\mathbb{E}_{\mathcal{S}_k \sim \mathcal{Q}} \left[P_n^{\mathcal{S}_k}(w|H) \right] = \mathcal{Q}(w|H)$$

which shows that this approximation is still accurate.

Approximating classification probabilities with IS

If we take back the notations of Section 2.2.1:

$$\omega(\hat{w}_i) = \frac{p_\theta(\hat{w}_i|H)}{Q(\hat{w}_i)} \quad \text{and} \quad \Omega = \sum_{i=1}^{k} \omega(\hat{w}_i) \tag{2.35}$$

we can see that while the method described in Section 2.2 is equivalent to directly maximizing the likelihood of the log-ratio approximated via importance sampling $\log \frac{\omega(w)}{\Omega}$, and can be seen as a weighted softmax, blackOut amounts to applying a logistic regression to this same log-ratio, which we can interpret as a posterior probability of coming from the data ($C = 1$):

$$P(C = 1|w, H) = \sigma\left(\log \frac{\omega(w)}{\Omega}\right) \tag{2.36}$$

Then, maximizing the likelihood of classifying the true examples as such and the importance samples as noise, we obtain the following discriminative objective:

$$J_\theta^H(w) = \log P(C = 1|w, H) + \sum_{i=1}^{k} \log(1 - P(C = 1|w, H))(\hat{w}_i|H)) \tag{2.37}$$

which is the same as in Equation 2.34.

Choice of the sampling distribution $\mathcal{Q}$

In Ji et al. (2015), the chosen sampling distribution $\mathcal{Q}$ is a power-raised unigram distribution:

$$\mathcal{Q} = p_{unigram}^{\alpha}$$

where $\alpha \in [0, 1]$. This distribution can be seen as an interpolation between an uniform ($\alpha = 0$) and unigram ($\alpha = 1$) distribution, which is a trade-off between being able to sample from the most important part of the probability mass (the most frequent words) and the need to have $\mathcal{Q}$ not too close to 0, to avoid large weights which would reduce the effective sampling weights, as discussed in Section 2.2.1 for importance sampling. Experimental results show that the approach is more stable than NCE and Importance sampling and that the discriminative objective improves results over maximum likelihood estimation, which using a power-raised unigram distribution does as well. However, experiments are performed over only very small vocabularies.

2.3.3 Negative Sampling

The *Negative Sampling* algorithm was popularized by the skip-gram embedding algorithm (Mikolov et al., 2013) and while closely linked to NCE, is not able to directly optimize the likelihood of a language model and learn conditional probabilities, as detailed in Dyer (2014). Recently, Melamud et al. (2016, 2017) showed Negative Sampling to be viable for language modeling. As previously, we use k

samples $(\hat{w}_i)_{1 \leq i \leq k}$ from the unigram distribution. The objective is very simple: it maximizes the likelihood of a logistic regression that discriminates true examples from noise samples. However, here, our model directly parametrizes the log-odds with the scoring function $s_\theta(w, H)$:

$$J_\theta^H(w) = \log \sigma(s_\theta(w, H)) + \sum_{i=1}^{k} \log \sigma(-s_\theta(\hat{w}_i, H)) \tag{2.38}$$

Melamud et al. (2016) show that optimizing this objective equates to finding a low-dimensional approximation of the shifted pointwise mutual information (PMI) matrix:

$$s_{\theta^*}(w, H) \approx \log \frac{P(w, H)}{P(w)P(H)} - \log k = \log \frac{P(w|H)}{kP(w)} \tag{2.39}$$

where the probabilities P are the frequencies in the training data. We can then obtain an estimation of the conditional probability using our model score and the unigram probabilities:

$$\hat{P}(w|H) \propto P(w) \exp s_\theta(w, H) \tag{2.40}$$

Experiments on relatively small data give results slightly better than with NCE, but the main argument for this method is the simplicity of its objective function. Since we directly parametrize what is in the other methods a log-ratio of our model and a noise probability, we can make the hypothesis that the learning will be more robust: the issue of unbalanced weights (caused by a noise probability too close to 0, as discussed in Sections 2.2.1 and 2.3.2) will not occur here.

2.4 Avoiding normalization by constraining the partition function

For some specific applications, focused on reducing the inference cost of a Neural Probabilistic Language Model, the techniques presented earlier are not a good solution. Hierarchical approaches are still costly and importance sampling requires normalization at testing time. Techniques based on discriminative objectives are fast during training, but since self-normalization is difficult to monitor, using a softmax may be required during testing. To be able to efficiently use a NPLM to decode for a machine translation system, Devlin et al. (2014) introduced *Self-normalized Neural Networks*, which they want to be able to use for inference without performing a softmax explicitly. Instead, they add an explicit constraint in their objective function that makes the partition function $Z_\theta(H)$ as close to 1 as possible during training:

$$NLL_{Self-normalized}(\theta) = \sum_{(H,w) \in \mathcal{D}} \left[s_\theta(w, H) - \alpha \log^2 Z_\theta(H) \right] \tag{2.41}$$

The hyper-parameter α enables trading accuracy for self-normalization error. While this objective is not faster to train, it greatly improves inference speed, since assuming the model self-normalized in such a way allows us to only compute $s_\theta(w|H)$. Andreas et al. (2015) analyse the theoretical properties of *self-normalized* models, whether they use discriminative objectives or an explicit constraint. They proved that a procedure that makes the partition function $Z_\theta(H)$ close to 1 for training contexts H also makes it close to 1 for unseen contexts. However, this result doesn't necessarily apply to a classification objective. Nonetheless, it provides intuition that sub-sampling the explicit constraint in the objective function 2.41 may not degrade results, while providing a speed-up. They experiment with the following objective:

$$NLL_{Self-normalized}(\theta) = \sum_{(H,w)\in\mathcal{D}} s_\theta(w,H) - \frac{\alpha}{\gamma} \sum_{H\in\gamma\mathcal{H}} \log^2 Z_\theta(H) \tag{2.42}$$

Here, $\gamma\mathcal{H}$ indicates that we sum on a fraction γ of all training contexts, ordered by frequency. Their results show that choosing $\gamma = 0.1$ keeps the performance intact. We should note that Chen et al. (2016) experimented with a similar objective, but without squaring the partition function, which we can interpret as a classic Negative log-likelihood objective where we normalize only for a fraction of the most frequent contexts. In their experiments, this objective behaves very similarly to the one presented in Equation 2.42.

2.5 Conclusions

In this chapter, we present a review of methods which allow avoiding the costly summations in the multinomial classification objective. While *hierarchical* approaches can reduce the computation time from something linear to a logarithm in $\mathcal{V}$, we focused here on *sampling-based* approaches. *Constrained* objectives are only useful if we are interested in fast inference — the main methods are *Importance sampling*, which approximates the partition function, and *Noise Contrastive Estimation*, which avoids computing it. Both theses training procedures require sampling k samples from an auxiliary distribution, and the sum over the full vocabulary is replaced by a sum over these samples. Interesting variants of these methods exist: we can mention *complementary sum-sampling*, which reduces drastically the variance of the estimator of importance sampling, and the *BlackOut* algorithm, which improves on noise contrastive estimation by using a context-dependant reweighted noise distribution.

Chapter 3

Detailled analysis of Sampling-Based Algorithms

Contents

In the previous chapter, we discussed several ways to handle large vocabularies when training neural languages models. We chose in this dissertation to focus on sampling-based methods, and are interested to know how these methods adjust to very large vocabularies in practice. Indeed, the vocabulary sizes commonly used in neural language models can reach very high numbers: for example, the 1 Billion Word Benchmark dataset (Chelba et al., 2014), in its commonly used form, is equipped with a $800K$ words vocabulary. Józefowicz et al. (2016), who presented the best published results on this corpus, experiments with both importance sampling (IS) and noise contrastive estimation (NCE), showing the former to be far more data-efficient. As discussed earlier, other works, like Chen et al. (2016), question the efficiency of NCE for language modeling, since perplexity is not explicitly optimized by the objective function. Therefore, we will in this chapter extensively compare IS, NCE and derived methods.

Similarities between IS and NCE are analyzed in Józefowicz et al. (2016): the authors explain that while NCE uses a surrogate binary classification task, we can see IS as using a surrogate multinomial classification task, where each of the k noise samples represents one of $k+1$ categories and the last one is the data category. This leads the authors to make the hypothesis that the multinomial nature of the IS objective could make it a better choice. While we will explore more in-depth the relationship between the objectives in Chapter 4, we will here try to understand what leads to the experimental discrepancies between them. Practically, the main aspect in which the methods differ is how they handle the partition function: while, with IS, it is explicitly approximated (as explained in Section 2.2.1), with NCE, we can parametrize it separately (as described in Section 2.3.1). The fact that NCE does not explicitly optimize the log-likelihood is exactly what allows us to parametrize the partition function separately: indeed, as detailed in Gutmann and Hyvärinen (2013), knowing the partition function, which represents the *'scale'* of the model, is essential to compute the likelihood[1]. Then, using a separate scaling parameter would give irrelevant results when doing maximum likelihood estimation (MLE), since we could minimize the negative log-likelihood independently of the data.

We will first study, in Section 3.1, the impact of hyperparameters (which are the number of noise samples k and the noise distribution P_n) on both NCE and IS. Obtaining disappointing results with NCE, we choose to investigate it specifically. While it is theoretically proven to converge when the partition function is parametrized separately, the scaling parameter is usually fixed and the *un-normalized* model therefore *self-normalizes*, as a side effect of the training procedure. In Section 3.2, we will compare closely the training of neural language models with MLE, IS, NCE and analyze their respective behavior. We will then show experimentally that the training instability of NCE is tied to the difficulty of the model to self-normalize and how this evolves with variants of the algorithm. Finally, in Section 3.3, we will use our analysis to propose various solutions to stabilize training and bring the performance of NCE closer to what could be expected theoretically.

Throughout the chapter, we do our analysis on the Penn Treebank (PTB) corpus. Usually, the size of the vocabulary is of 10K words. Using this vocabulary, a

[1]Which is not even taking in account the fact that the concept of likelihood only applies to probability density functions, which are normalized.

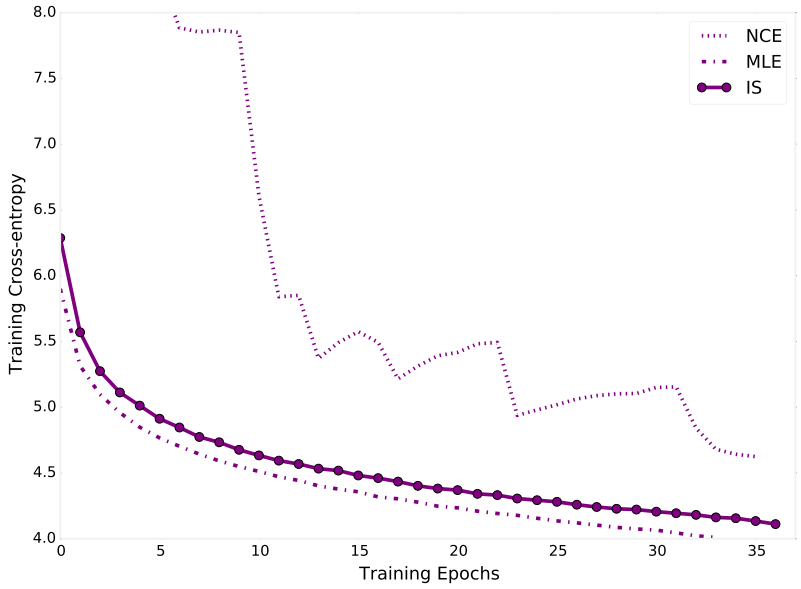

Figure 3.1: Training cross-entropy curves on PTB for models trained with MLE, IS and NCE on a vocabulary of $10K$ words.

Training method	Perplexity
MLE	66,9
IS	71,1
NCE	87,6

Table 3.1: Best final perplexities on the test set of the Penn Treebank (PTB) corpus obtained with MLE, IS and NCE on a vocabulary of $10K$ words.

comparison of training cross-entropies for models trained with MLE, NCE and IS, and a comparison of final testing perplexities, are shown in Figure 3.1 and Table 3.1. With this setting, we can observe that there are differences in the training processes and that NCE does not perform as well given the same number of training epochs. However, the model is able to reach a perplexity that is not so far from MLE and IS, and could mirror their performance with more training epochs. To work with an experimental set-up that allows us to witness even more evidently the training difficulties of NCE, we choose to use the full training vocabulary instead, which contains a little more than $44K$ words. When not specified otherwise, we use $k = 100$ noise samples drawn from the unigram distribution. Our hyperparameter choices are summarized in Table 3.2. To reduce the impact of the training criterion, models are learnt for a minimum of 30 epochs. Beyond that limit, we backtrack the epoch when no progress has been made on the validation set perplexity, stopping training after 10 consecutive backtrackings[2].

[2]For the sake of clarity, backtrackings are discarded from the graphs, keeping only the epochs used to obtain the final model.

Hyperparameter	Value
Number of noise samples k	100
Default noise distribution P_n	Unigram
Hidden Layer	LSTM
Maximum Sentence Length	30
Number of hidden layers	2
Word embedding dimension	300
Hidden layer dimension	300
Batch size (Sentences)	32
Dropout rate	0.5
Optimization method	SGD
Learning rate	1.0
Grad clipping value	5.0
Maximum number of training epochs	50

Table 3.2: Structural choices and hyperparameters used on the Penn Treebank (PTB) corpus for experiments presented in this chapter (unless specified otherwise).

3.1 Choosing k and P_n: impact of the parametrization of sampling

For both IS and NCE, there are strong theoretical results that can guide us in our choices of noise distribution P_n and of number of noise samples k. However, in the larger part of the published work where these methods are used for language modeling, practical considerations — which are, how easy it is to sample from P_n and how much of a computational burden is using k noise samples — prevail. We will here restate these results and explore experimentally reasonable (i.e, fairly practical to implement) choices for these hyperparameters. While k is easy to control, the complexity of P_n is harder to set. We choose here to experiment mainly with context-independent distributions, which are the uniform and unigram distributions. We implement the transition between the two distributions by using a distortion parameter $0 \leq \alpha \leq 1$ on the unigram distribution. P_n^α is uniform for $\alpha = 0$ and unigram for $\alpha = 1$; we will use additional intermediary values in our experiments. For all experiments, final perplexity results on the testing set of PTB are presented in Table 3.3.

3.1.1 Effects on Importance Sampling

As explained in Section 2.2.1, the bias and the variance of the self-normalized importance sampling estimator can be reduced by augmenting the number of samples k. Concerning the choice of P_n, Hesterberg (2003) gives the optimal noise distribution in terms of reducing the variance. However, there is a non-zero theoretical minimal limit on the variance that can be reached with self-normalized importance sampling. Besides, even approximations of the optimal noise distribution would

necessitate knowledge of the score function

$$\frac{\partial}{\partial \theta} s_\theta(w, H)$$

for every word w in $\mathcal{V}$, which is something we want to avoid computing. Furthermore, while previous results with IS (Bengio and Sénécal, 2003, 2008) were finding the method unstable and attempts at adapting P_n during learning gave poor results, our experiments show that training was stable for almost all choices of hyperparameters and there was no apparent need to make P_n closer to the true data distribution. In this, our experiments match those of Józefowicz et al. (2016). This stability is ex-

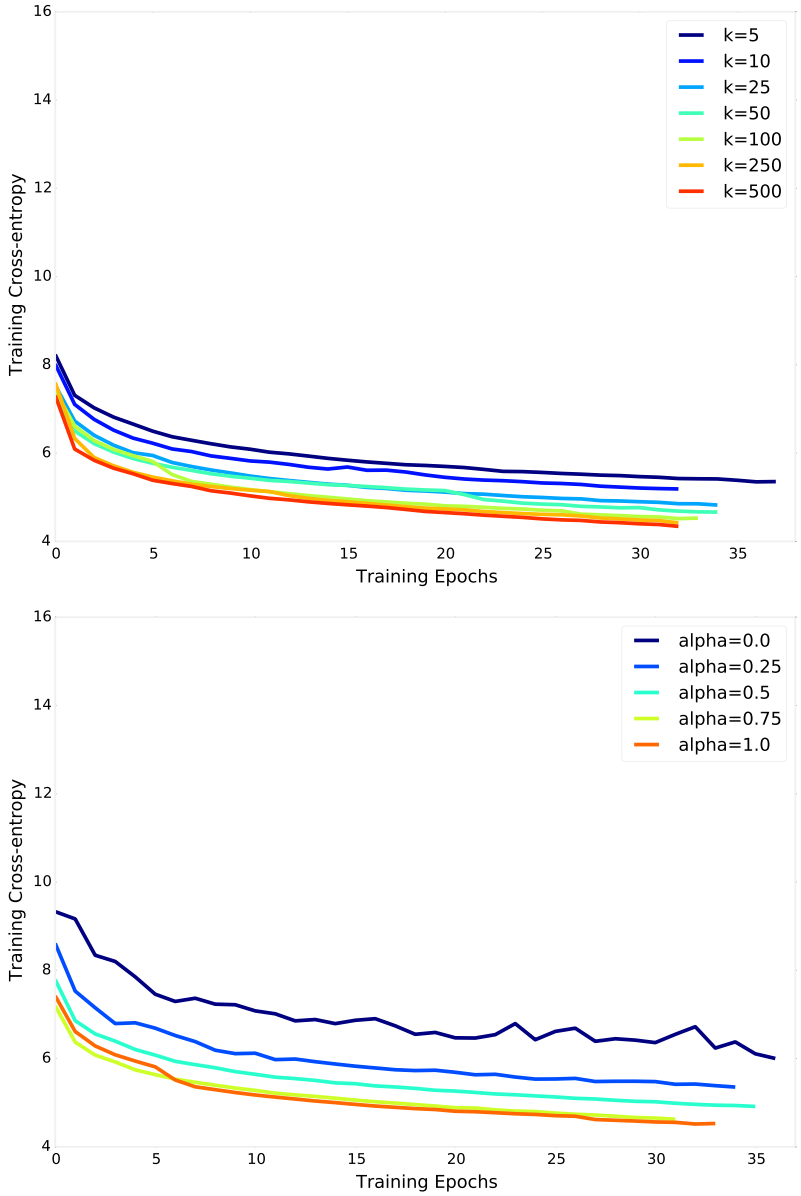

Figure 3.2: *Top*: Training cross-entropy curves on PTB for models trained with IS and various values of the number of samples k. *Bottom*: Training cross-entropy curves on PTB for models trained with IS and various values of distortion α for the noise distribution P_n.

plained by the fact that the Tensorflow function used for IS[3] is a little different than the algorithm described in Bengio and Sénécal (2003). Indeed, it always includes the target word in the sampled distribution, making the method closer to much more stable complementary sum-sampling (Botev et al., 2017), that we described in Section 2.2.3.

In Figure 3.2 are shown training cross-entropies for models using IS, with varying k and α. On the first column of Table 3.3 are the corresponding final perplexity results on the test set. We can confirm that increasing k gives a better final perplexity and that using the unigram distribution is far more efficient than using the uniform distribution.

3.1.2 Effects on Noise-Contrastive Estimation

The results presented in Section 2.3.1 were that as the number k of noise samples by example increases, the choice of the noise distribution P_n has less impact on the estimation accuracy. Besides, Gutmann and Hyvärinen (2010) explain that finding the noise distribution that minimizes the estimation error is very difficult. However, the authors also show that for a noise distribution close to the data distribution, there is a guarantee that even for small values of k, the estimation error is close to the theoretical optimum. Then, they suggest that one should first choose P_n the closest to the data as possible so that noise can be sampled easily and then make k as large as computationally possible.

Training cross-entropies for models trained with NCE, with varying k and α, are presented on Figures 3.3 and 3.4. Corresponding final perplexity results are presented in the second column of Table 3.3. To not only evaluate how well the model optimizes perplexity, but also how well it performs at the task it is actually training for, we also show the mean probabilities of classifying data samples into the data class - $P(C = 1|w, H)$, when $(w, H) \in \mathcal{D}$ - and noise samples into the noise class — $P(C = 0|\hat{w}, H)$, when $\hat{w} \sim P_n$ — during training. As we we could see on preliminary experiments on the $10K$ vocabulary, NCE is far less stable than IS. With our configuration, k needs to be of at least 100 samples for the model to attain a reasonable perplexity in 50 epochs. The top graph of Figure 3.3 shows that models with smaller values are far from reaching a similar result. However, when k varies, the classification task changes: it is easier to recognize data when k is low, and easier to recognize noise when k is high, so the second graph is harder to interpret. We can still suppose that it is having an easier classification task (a low k) that makes training less efficient.

While using a uniform distribution is not working in our case, smoothing the unigram distribution seems to make learning easier, as shown on the top graph of Figure 3.4. Still, when we get too far of the unigram distribution, learning is not as precise and the final perplexity is quite high. Looking at classification probabilities, we can see that for low values of α, it is easier to recognize data samples: again, having the classification task easier makes for a less efficient training. Finally, we can see that smoothing the unigram distribution a little is beneficial, probably thanks

[3]Which is called *sampled softmax*.

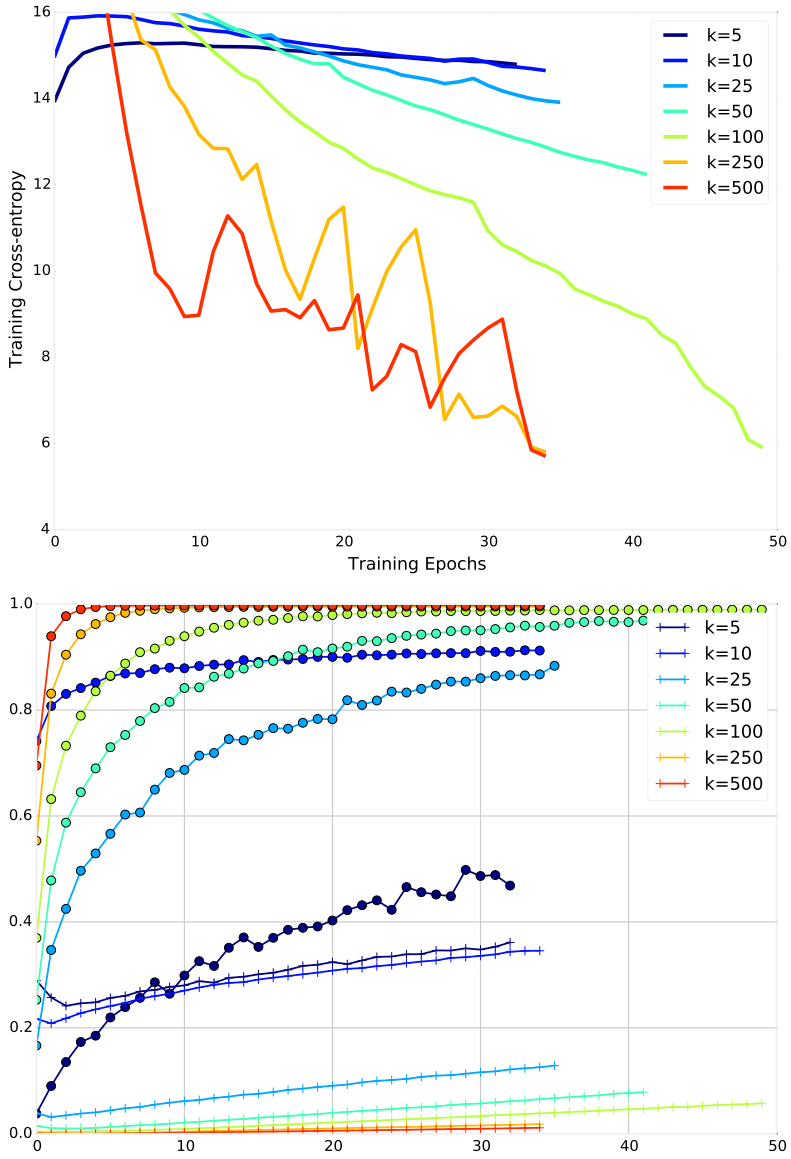

Figure 3.3: *Top*: Training cross-entropy curves on PTB for models trained with NCE and various values of the number of samples k. *Bottom*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

to having higher values of $P_n(w)$ for rare words.

Complexifying the noise distribution

In order to explore a more complex noise distribution P_n, we experimented with a bigram (and so, context-dependent) distribution. Training cross-entropies and classification probabilities are shown in Figure 3.5. It is clear that the bigram distribution is far more data-efficient than the unigram and the final perplexity is also better. While the classification task uses the same k, we can also see that the model is able to learn to recognize data samples far better with a bigram distribution. While these results are very interesting, using a context-dependent distribution is

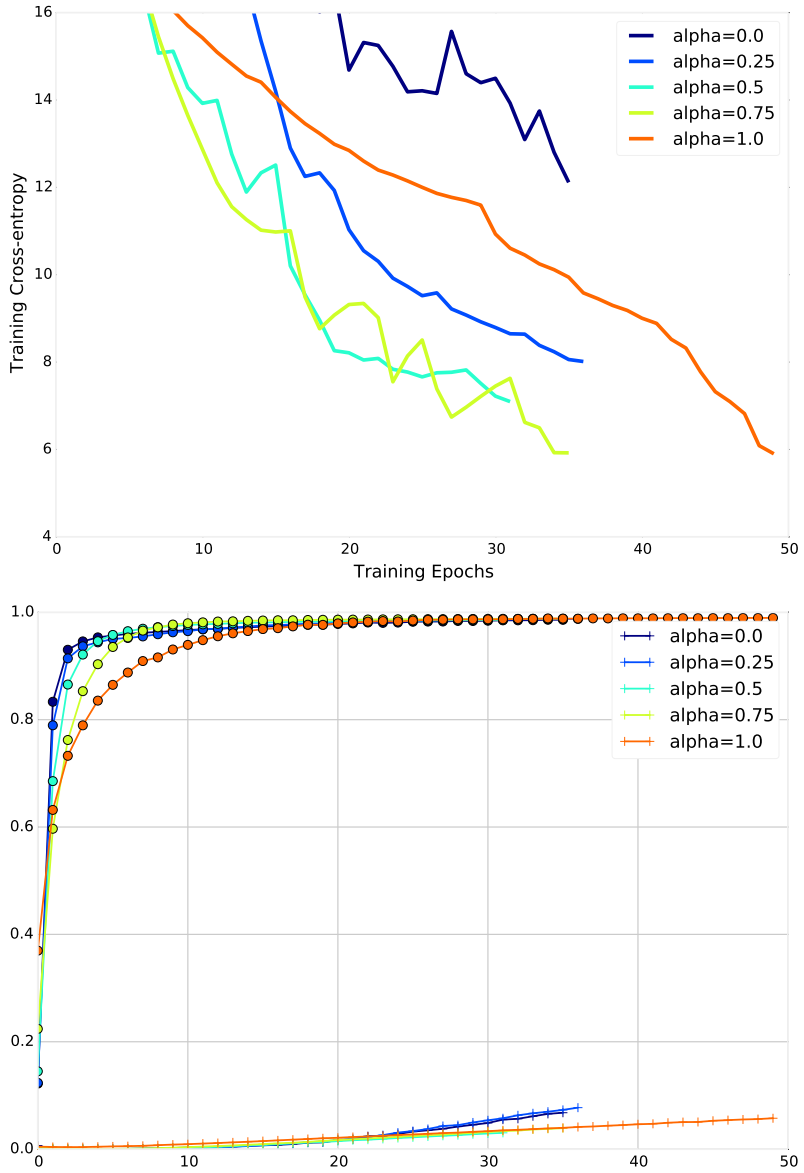

Figure 3.4: *Top*: Training cross-entropy curves on PTB for models trained with NCE and various values of distortion α for the noise distribution P_n. *Bottom*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

very slow in practice, since sampling needs to be done example by example and cannot be done once by batch. While they show that it is indeed more efficient to use a noise distribution closer to the data distribution, these results have no practical application.

Particular case of the BlackOut algorithm

BlackOut (Ji et al. (2015), described in Section 2.3.2) can be seen as NCE with a re-weighted, context-dependent noise distribution: the noise probability given to words is re-scaled using the model distribution on the noise samples. However, these noise samples are still obtained using the original noise distribution: we can expect

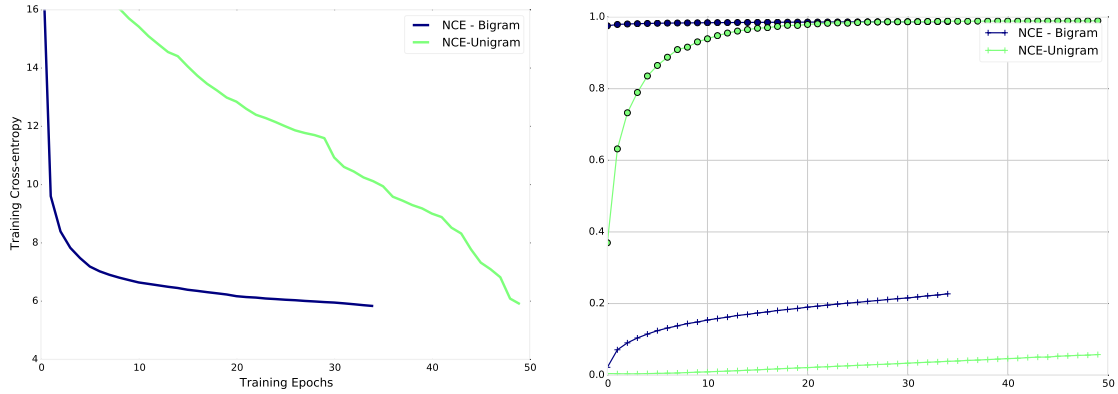

Figure 3.5: *Left*: Training cross-entropy curves on PTB for models trained with NCE with a unigram and bigram distribution as P_n. *Right*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

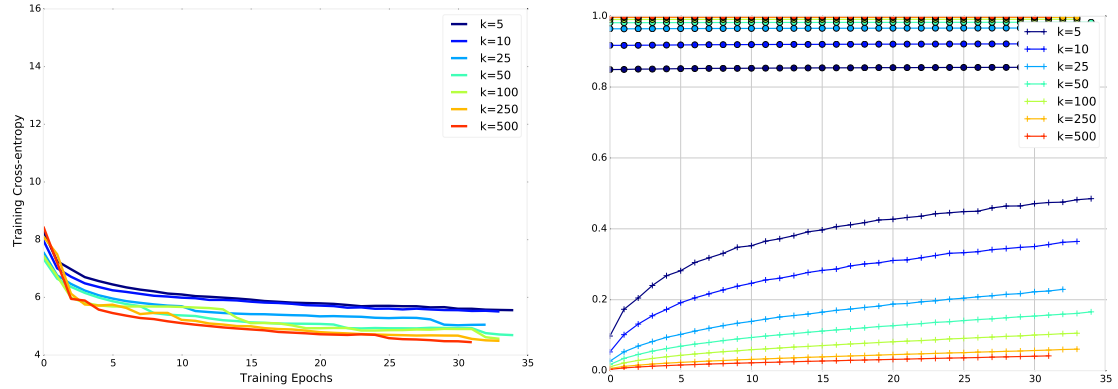

Figure 3.6: *Left*: Training cross-entropy curves on PTB for models trained with BlackOut and various values of the number of samples k. *Right*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

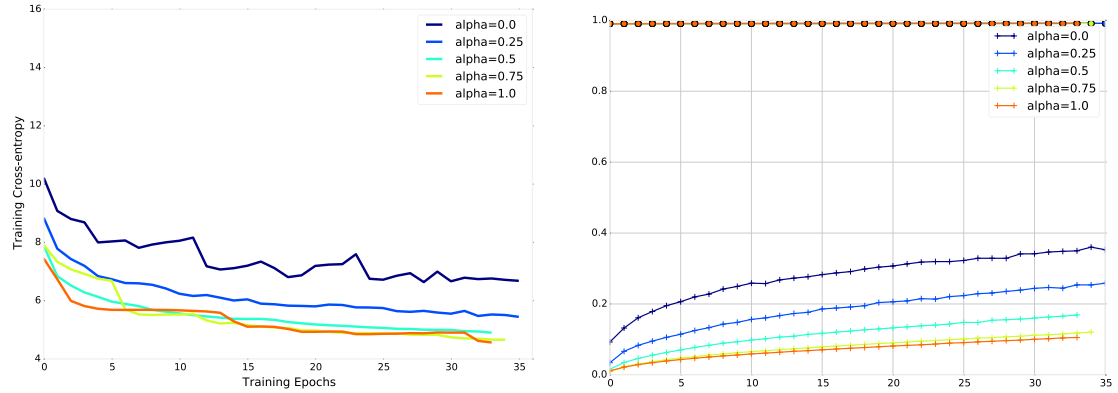

Figure 3.7: *Left*: Training cross-entropy curves on PTB for models trained with BlackOut and various values of distortion α for the noise distribution P_n. *Right*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

the distortion α to keep its impact on training. Results are shown in Figures 3.6 and 3.7, and final testing perplexities are shown in the third column of table 3.3. We can immediately see that training is far more stable than with NCE: by averaging over all samples to obtain re-weighted noise probabilities (see Equation 2.31), BlackOut uses them more effectively. Still, augmenting k improves results. Binary classification probabilities are hard to interpret here, since reweighing the noise probabilities transforms them into two separate sets in a multinomial classification: does the sample belong to category 1 or in one of the categories 2 to $k+1$? The probabilities of recognizing noise reflect this modification: they seem fixed to a value dependent of k. Here, we can notice that smoothing does not improve results.

Effects on Negative sampling

With negative sampling (NS, described in Section 2.3.3), the model directly parametrizes the log-odds and it learns what is, in the NCE, the ratio of the data and noise distribution, without knowing the values of P_n. We expect here to avoid the stability issues caused by having unbalanced weights, coming from very small values of $P_n(w)$ for rare words. Results are shown in Figures 3.8 and 3.9 and final testing perplexities are shown in the fourth column of Table 3.3.

We can first notice that training seems indeed more stable than with NCE. Interestingly, while varying k has the same predictable effect as with NCE on binary classification probabilities, the perplexity is very differently affected: having k too high seems to worsen the performance, and $k = 50$ gives the best final result. The effect of distortion is similar to the effect obtained with NCE and using $\alpha = 0.75$ gives the best result.

These results seem to confirm the analysis of Józefowicz et al. (2016) and Chen et al. (2016): IS is far more data-efficient than NCE, which, learning for a different task, does not optimize perplexity and therefore does not seem adapted to language modeling. While we have explored alternatives to NCE that obtain results that are competitive with IS, we wish to analyse in depth the inner workings of NCE and

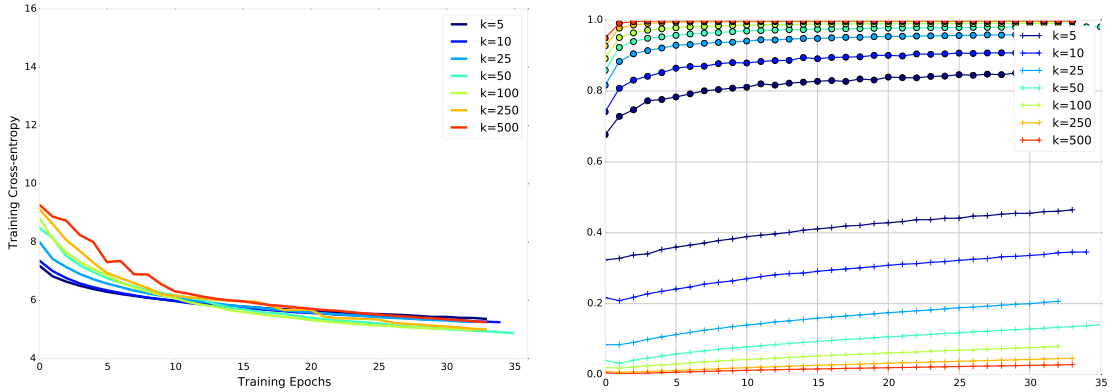

Figure 3.8: *Left*: Training cross-entropy curves on PTB for models trained with Negative Sampling and various values of the number of samples k. *Right*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

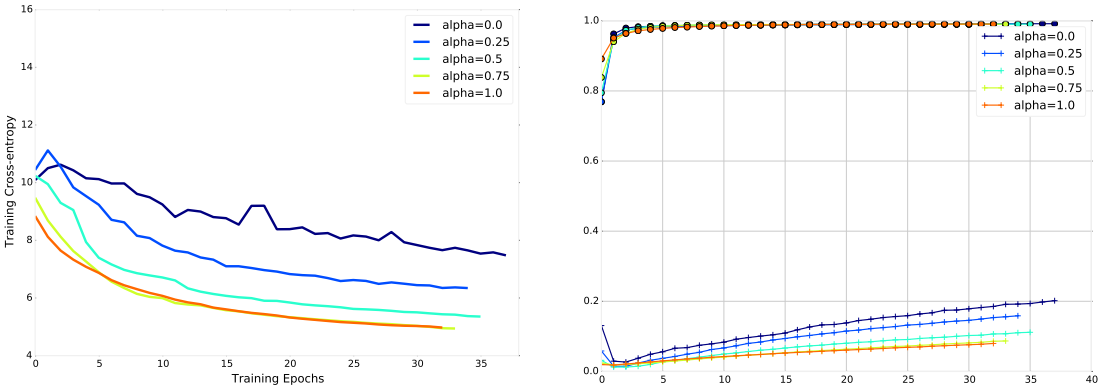

Figure 3.9: *Left*: Training cross-entropy curves on PTB for models trained with Negative Sampling and various values of distortion α for the noise distribution P_n. *Right*: Corresponding curves of classification probabilities of data examples into data (+) and noise samples into noise (o).

	IS	NCE	BlackOut	NS
Baseline model	168.3	306.0	169.0	228.3
$k = 5$	233.9	X	248.4	252.0
$k = 10$	203.1	X	250.2	255.0
$k = 25$	184.6	X	216.3	268.6
$k = 50$	181.0	X	181.1	196.2
$k = 250$	160.3	298.0	163.7	243.0
$k = 500$	153.5	235.7	157.9	302.0
$\alpha = 0.0$	366.1	X	487.4	900.9
$\alpha = 0.25$	201.9	1068.0	233.9	344.8
$\alpha = 0.50$	180.6	641.2	178.9	208.6
$\alpha = 0.75$	171.5	277.0	172.3	195.8
P_n = Bigram		269.6		

Table 3.3: Best final perplexities on the test set of the Penn Treebank (PTB) corpus obtained with IS, NCE and variations, with varying number of samples k and distortion α of the noise distribution P_n. 'X' indicates that the model did not reach a perplexity under the size of the vocabulary within the maximal number of training epochs, which is 50.

how to improve its performance.

3.2 Impact of the partition function on the training behaviour of NCE

In this section, we will closely monitor the training process of the same language model, varying only the training criterion, *i.e* MLE, IS or NCE. We should also remember that, apart from the type of classification they learn, NCE and IS are different in how they handle the partition function. In order to better understand

what happens during NCE training, we also consider an '*intermediate*' model denoted as *NCE normalized*. This normalizes the scores into a probability distribution $P_\theta(w|H)$ before computing the NCE objective function. Without any application in practice, this model allows us to better assess the impact of the normalization process.

3.2.1 Self-normalization is crucial for NCE

The training cross-entropies for the 4 models are drawn in the top graph of Figure 3.10. Whether they are trained with MLE or NCE, the normalized models exhibit very similar learning curves. However, while the behavior of the model trained with IS only slightly deviates from the normalized models, the un-normalized model trained with NCE takes far longer to reach a comparable cross-entropy, ending with a sensibly higher value. In the bottom graph, we can observe the values of minus the objective function[4] and both of its terms (the first, data-dependent and the second, containing the noise samples) for the normalized and un-normalized models trained with NCE:

$$-J_\theta = \sum_{H,w\in\mathcal{D}} \left[-\log \frac{p_\theta(w|H)}{p_\theta(w|H) + kP_n(w)} \quad -\sum_{i=1}^{k} \log \frac{kP_n(\hat{w}_i)}{p_\theta(\hat{w}_i|H) + kP_n(\hat{w}_i)} \right] \tag{3.1}$$

We can observe a small decrease of the second term for the normalized model, whereas for the un-normalized, it starts with high values and decreases to become closer to the normalized one. Moreover, the gap between the two data-dependent terms stays high at the end of training.

For a deeper analysis, we study the values of the partition function during training. For both the normalized and the un-normalized models, the partition function $Z_\theta(H)$ associated to each training context H is computed and the results over an epoch are summarized with an histogram. Each bin represents a range of values and its height represents the fraction of training contexts whose partition function belongs to this range. For a better readability, both range values and bin heights are in log-scale. These histograms are shown in Figure 3.11 for different epochs. While the repartition of the partition function values for the normalized model does not change much - which is logical, since the model is always integrated to 1 - we observe that these values for the un-normalized model are chunked together and are decreasing during learning (epochs 5, 10 and 15). However, this trend seems to slow down towards the end of training, since, at epoch 30, the repartition resemble to the one of the normalized model, only shifted to smaller values (which are still quite high). Considering these results, we dissect what happens during the training of the un-normalized model. Let us rewrite the objective J_θ as a function of the following ratio:

[4] Since the objective J_θ^H is negative, we minimize $-J_\theta^H$.

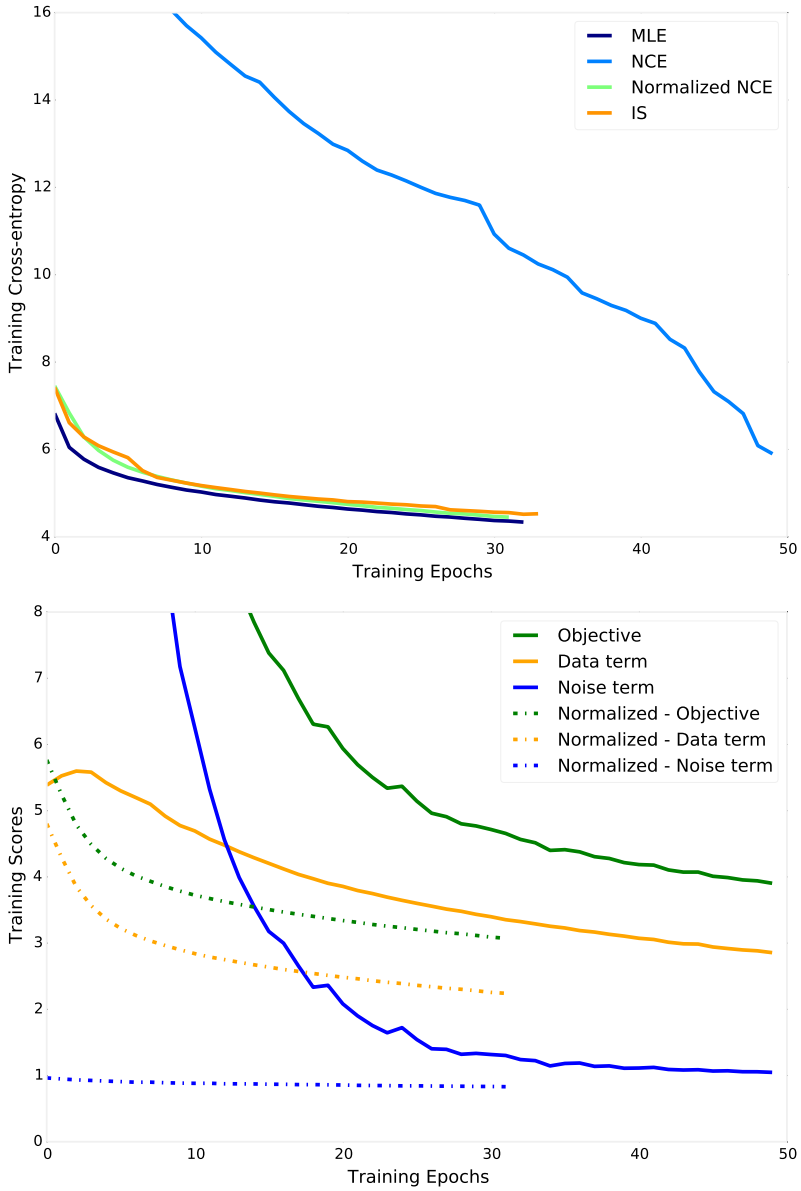

Figure 3.10: *Top*: Training cross-entropy curves on PTB with Maximum-Likelihood estimation, IS, NCE on an un-normalized model and a model normalized before application of the NCE. *Bottom*: Training scores (see equation 3.2) of the same un-normalized and normalized models trained with NCE

$$
\begin{aligned}
r_\theta(w, H) &= \frac{p_\theta(w|H)}{kP_n(w)}, \text{ then} \\
-J_\theta &= \sum_{H,w \in \mathcal{D}} \left[-\log \frac{r_\theta(w,H)}{r_\theta(w,H)+1} \quad -\sum_{i=1}^{k} \log \frac{1}{r_\theta(\hat{w}_i, H)+1} \right]
\end{aligned}
\tag{3.2}
$$

The second term of $-J_\theta$ is very high for large values of r_θ. Since values of P_n are always smaller than 1 and $k = 100$, and knowing the partition function at the beginning of training is large for all contexts, we can assume that $r_\theta(\hat{w}_i|H)$ is large for a least part of the set of noise samples $(\hat{w}_i)_{i=1}^k$. Thus, this second term is the largest part of the objective, whose minimization leads to a decrease of the model

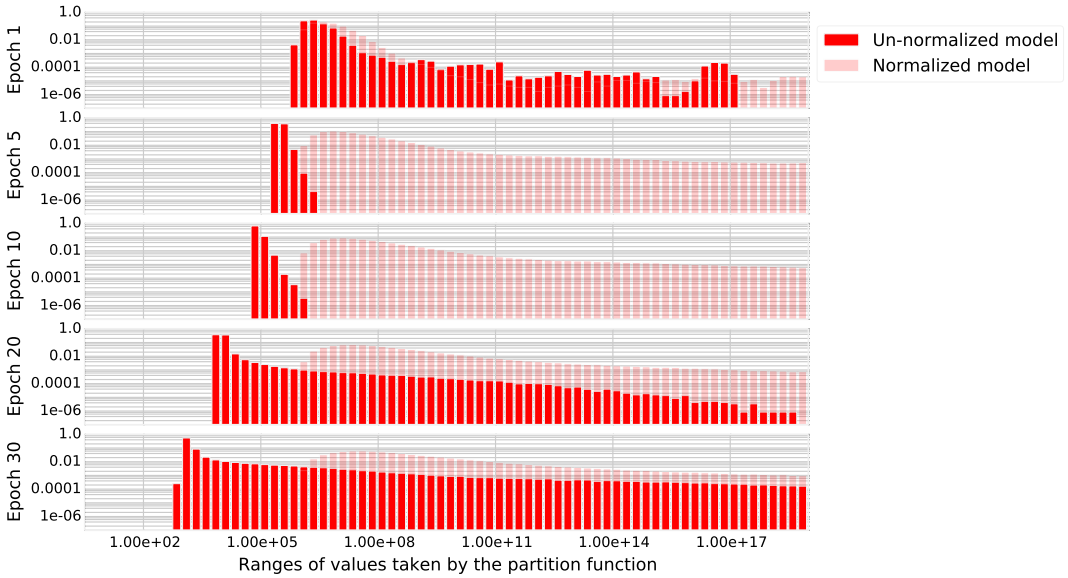

Figure 3.11: Repartition of the values of the partition function $Z_\theta(H)$ for all examples (H, w) during specific epochs of training on PTB with NCE. The fully colored bars represent the repartition for the un-normalized model, while the faded bars represent the repartition for the normalized model. Both scales are logarithmic.

scale. However, this is done at the expense of the data-dependent term, since its score increases a little during the first few epochs. If we take a look back at its gradient:

$$\frac{\partial}{\partial\theta} J_\theta^H(w) = \frac{1}{1 + r_\theta(w, H)} \frac{\partial}{\partial\theta} \log p_\theta(w|H) - \sum_{i=1}^{k} \left[\frac{r_\theta(w, H)}{r_\theta(w, H) + 1} \frac{\partial}{\partial\theta} \log p_\theta(\hat{w}_i|H) \right] \tag{3.3}$$

we see that it is always scaled by the weight $1/(1 + r_\theta(w, H))$, which means that the objective does not learn anything from data as long as the model scale has not decreased to a reasonable value. This shows that this *'self-normalization'* mechanism is crucial for NCE, but we still need to understand what makes a scaling value reasonable.

In this section, we have shown that when scoring a word w with an input context H, the scale of the model $Z_\theta(H)$ has an influence on what is learnt. Indeed, a high value of the partition function increases the value of $r_\theta(w, H)$, which reduces the importance of the data-dependent term of the gradient. That proves the necessity of self-normalizing in order to learn.

3.2.2 Influence of the shape of P_n on self-normalization

Now, we would like to understand what role the noise distribution P_n is playing in learning. Let us consider again the weight of the gradient of the data-dependent

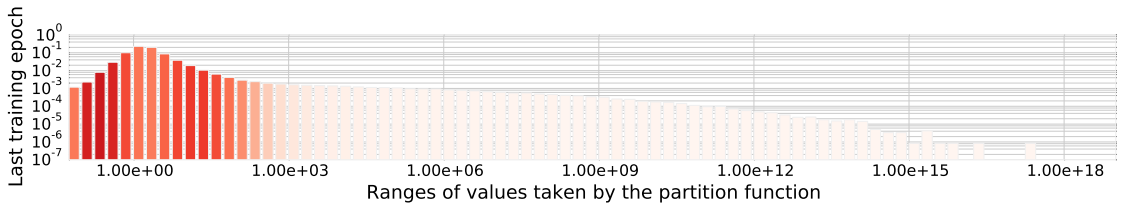

Figure 3.12: Repartition of the values of the partition function $Z_\theta(H)$ for all training examples (H, w), during the last epoch of training with NCE on PTB. For both this figure and figure 3.13, the colour of a bin indicates the proportion of training examples (H, w) for which the word w is one of the 10 most frequent according to the noise distribution P_n: the lighter the color is, the higher is that proportion. Both scales are logarithmic.

term:

$$\frac{1}{r_\theta(w|H)+1} = \frac{kP_n(w)}{p_\theta(w|H)+kP_n(w)}.$$

Having higher values of P_n means that the model does not need to scale back the partition function as much to be able to learn from data. Besides, if the noise distribution is correlated with the data distribution, an example with a high value of P_n tends to be more frequent. Therefore the model is first able to learn from words w with the higher values of $P_n(w)$. However, the lower $P_n(w)$ is, the more difficult it is for the model to learn about w. And the associated partition function has to be closer to 1 for the data-term gradient to be meaningful.

We verify our analysis experimentally by looking at the values of the repartition function at the end of training. The histogram is shown in Figure 3.12: this time, each bin is colored according to the proportion of examples associated to the highest values of P_n it contains[5]. This allows us to vizualize if the contexts associated to frequent words are less 'self-normalized'. We clearly observe that this is the case: while the majority of the partition functions have values under ≈ 100, the partition functions of contexts associated to the higher values of P_n mainly take values that are far larger.

We have just shown that the frequency $P_n(w)$ of a word affects how the model learns: the higher $P_n(w)$ is, the more important the contribution of the data-term to the objective is. This has another implication: high values of $P_n(w)$ counter the effect of high values of the partition function, which means that the model does not need to self-normalize as much for contexts associated to frequent words.

3.2.3 How do these factors affect learning ?

To visualize how learning is affected, we propose to study histograms showing the values taken by r_θ for all data and noise samples seen by the model during a training epoch. In orange, we show data samples and in blue, noise samples. As for Figure 3.12, each bin is colored according to the proportion of very high frequency

[5]Since we use the unigram distribution, it indicates here the examples who are among the 10 most frequent words.

words it has: the clearer it is, the more occurrences of the 10-highest frequency words it contains. As previously, the histograms displayed in this section are in log-scale. We first display these values at initialization, in Figure 3.13. We can see that ratios of occurrences of the frequent words have the lowest values. That makes sense, since the scores given by the model at initialization follow a normal distribution and those given to the most frequent words are divided by the higher values of P_n.

In Figure 3.14, we show the histograms for epochs 1, 5, 10, 20 and 30, during training of a normalized model. At epoch 1, we can see that noise samples are still ordered in the same way as at initialization, with the ratios of the occurences of the most frequent words having lower values. Since the model is normalized, the

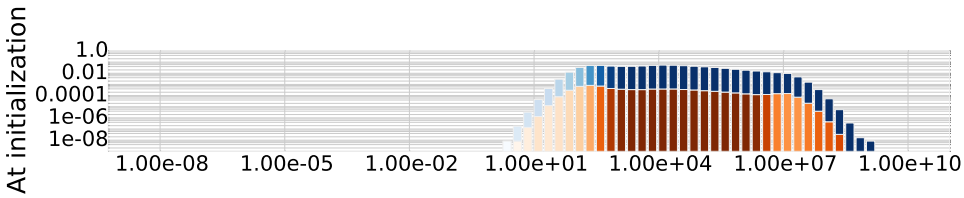

Figure 3.13: Repartition of the values of the ratio $r_\theta(w|H)$ for all positive training examples (H, w) in orange, and associated noise sample $(\hat{w}_i)_{i=1}^k$, in blue, at initialization.

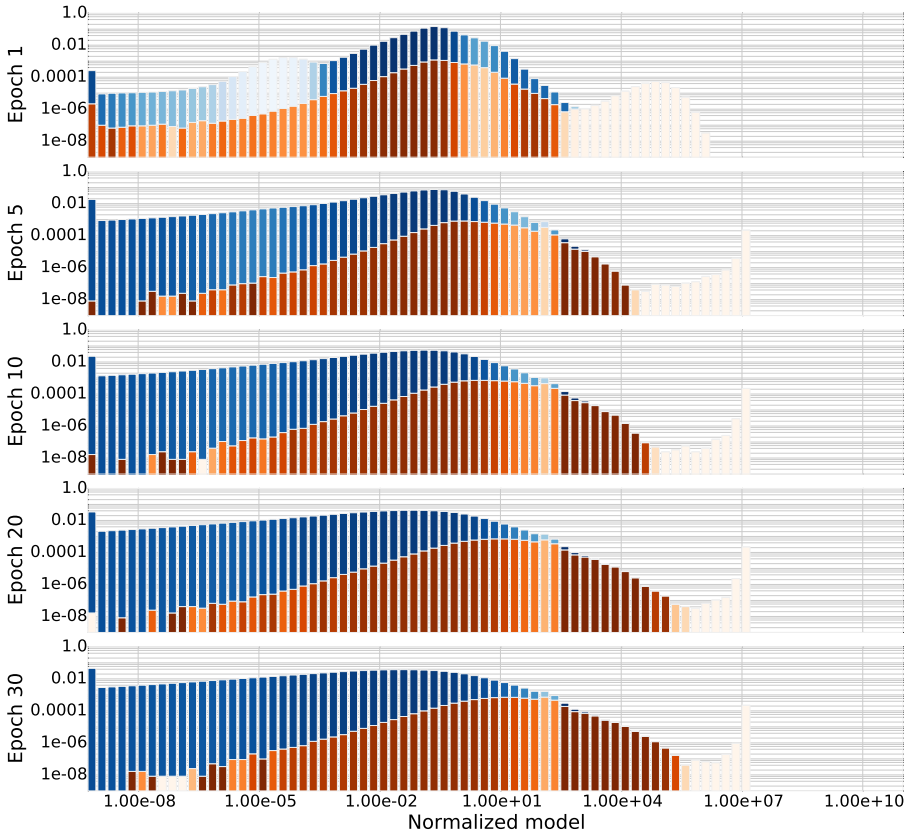

Figure 3.14: Repartition of the values of the ratio $r_\theta(w|H)$ for all positive training examples (H, w) in orange, and associated noise sample $(\hat{w}_i)_{i=1}^k$, in blue, during several training epochs of a normalized model with NCE on PTB.

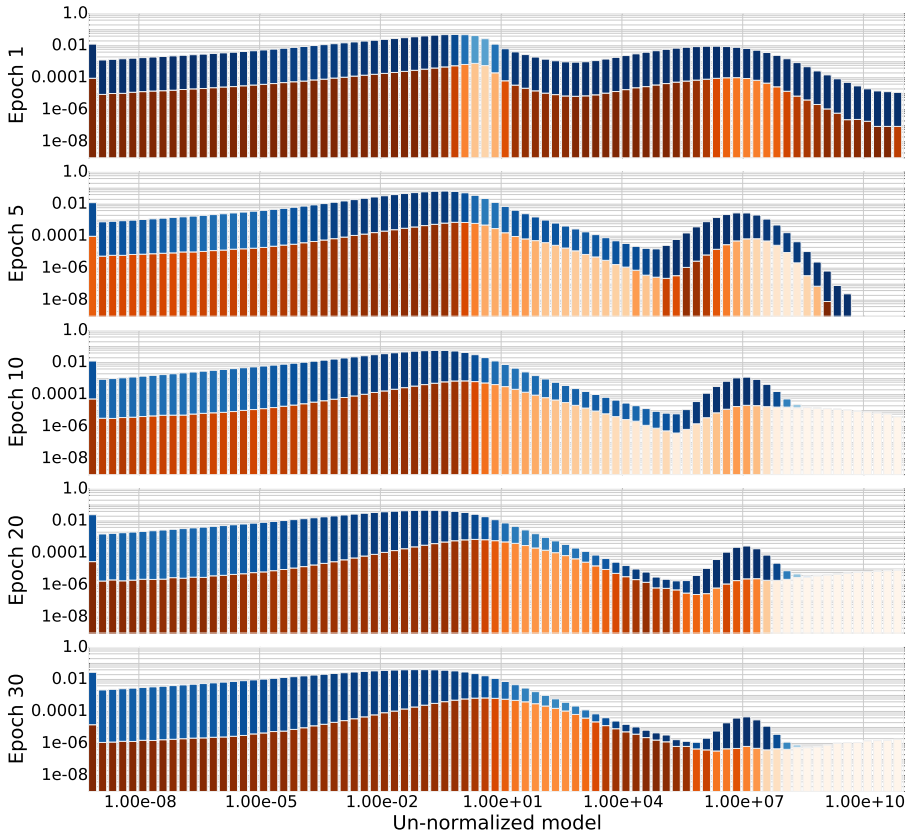

Figure 3.15: Repartition of the values of the ratio $r_\theta(w|H)$ for all positive training examples (H, w) in orange, and associated noise sample $(\hat{w}_i)_{i=1}^k$, in blue, during several training epochs of an un-normalized model with NCE on PTB.

occurences of the most frequent words in data samples are learnt faster (which aligns with our analysis in Section 3.2.3). As learning progresses, we can see that there is a large number of data samples the model has no uncertainty about: the values of their ratios r_θ become well-separated of the values of the ratios for noise samples.

In Figure 3.15, we show the histograms for epochs 1, 5, 10, 20 and 30, during training of an un-normalized model. During epoch one, the values taken by the ratios are all over the place and noise samples seem uniformly distributed. During the next epochs, a clear separation between data examples values and noise samples values takes far more time to appear. The only data samples that are completely separated of noise samples at the end of training are occurences of frequent words, which was not the case for the normalized model. This seems to confirm our analysis of Section 3.2.1: for words that are not frequent, self-normalization is necessary before learning.

3.3 Easing the training of neural language models with NCE

The observations made in the previous sections show us that an higher discrepancy between high and low P_n leads to a more difficult training with NCE - learning

is tiered according to word frequencies. As a consequence and because of the Zipfian distribution of word frequencies, using the unigram distribution is harder as the size of the vocabulary grows. As discussed previously, two values strongly impact how a sample is learnt: first, its value of P_n; then the value of the partition function. We will in this section introduce two simple tricks to reduce the negative impact they have on training with NCE. We will verify how they interact with various values of k and α, first independently, then when used together - perplexity results on the test set of the PTB are presented in Table 3.4. Finally, we will present a summary of the best configurations of sampling based methods for a fixed k and check their efficiency on a much larger corpus: the 1 Billion Word Benchmark dataset. These results are presented in Table 3.6.

3.3.1 Helping the model by learning to scale

To control the partition function, we can follow a method introduced by Chen et al. (2015): shifting all values of the partition function by a fixed value Z_c. In practice, it means we set

$$p_\theta(w|H) = \frac{e^{s_\theta(w,H)}}{Z_c}.$$

Another similar workaround consists in initializing the output bias to $-\log|\mathcal{V}|$ Vaswani et al. (2013); Melamud et al. (2017). It has the same effect as fixing $Z_c = |\mathcal{V}|$ and initializing the bias to 0. In Chen et al. (2015) this value is arbitrarily set to $\log Z_c = 9$. Here, we propose to learn the best shifting value by adding Z_c to the model parameters θ. In this case Z_c can be seen as an extra bias term that learns a normalizing constant.

Figure 3.16 represents training cross-entropy curves depending on the initial

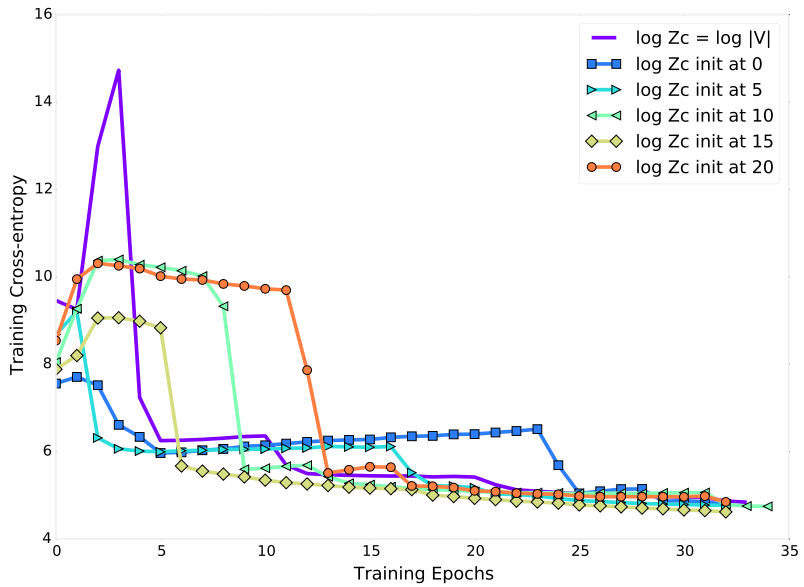

Figure 3.16: Training cross-entropy curves on PTB for un-normalized models trained with NCE while fixing, then learning a parameter Z_c that shifts the partition function, depending on the initial value of Z_c.

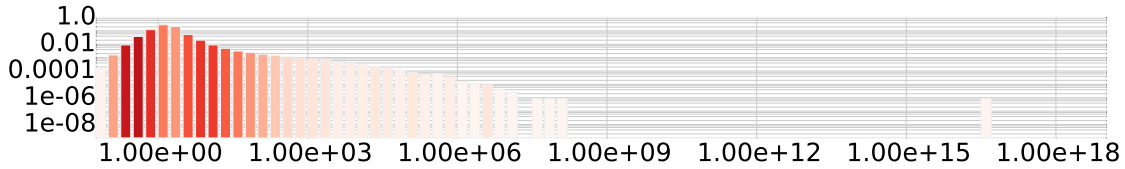

Figure 3.17: Repartition of the values of the partition function $Z_\theta(H)$ for all training examples (H, w), during the last epoch of training with NCE on PTB, where a scaling parameter Z_c is learned. The color of a bin indicates the proportion of training examples (H, w) for which the word w is one of the 10 most frequent according to the noise distribution P_n: the lighter the color is, the higher is that proportion. Both scales are logarithmic.

value of Z_c, showing its impact. Figure 3.17 shows the histogram of the repartition of values of the partition function, for the final epoch of training a model where we learn Z_c. This is to be compared with Figure 3.12, showing the same histogram, but for classic NCE. We can see that the values of the partition function are far smaller and grouped around 1. The first column of Table 3.4 shows how the choices of k and α affect this method. We can see that we obtain reasonable results even with a very low k and that smoothing the unigram distribution is only moderately helpful here. Finally, Table 3.6 shows that learning to scale achieves a similar perplexity as fixing it to the arbitrary value $|\mathcal{V}|$ on the PTB.

3.3.2 Helping the model with a well-chosen initialization

To operate on the impact of noise distribution during learning, we turn to negative sampling. We remember that with NS, the model directly parametrizes the log-odds, and we get back an estimation of the conditional probability using the model score and the noise distribution:

$$\hat{P}(w|H) \propto P_n(w) \exp\left(s_\theta^{NS}(w, H)\right) \tag{3.4}$$

Then, the initial values of $r_\theta(w|H) = e^{s_\theta^{NS}(w,H)}$ simply depend on the model initialization and the initial values of the partition function are close to 1. We notice that our models trained with negative sampling avoided the initial increase in cross-entropy that is visible with NCE, even when we fix or learn Z_c. We believe this is due to having an initial model distribution very close to P_n, which is a consequence of multiplying final scores by P_n, as showed in Equation 3.4.

To verify this, we trained a NCE model for which we initialized the output bias with values of $\log P_n(w)$. By looking at the repartition of the values of the ratio and partition function at initialization, showed in figure 3.18, we can check that having $p_\theta(w|H) \approx P_n(w)$ mitigates the main issues described in section 3.2: first, our distribution is far closer to being normalized; then, the values of $r_\theta(w|H)$ are bundled together around $1/(k+1)$.

Figure 3.19 presents the histograms showing the repartition of ratios $r_\theta(w|H)$ during training for this method. At epoch 1, both data and noise samples for non-frequent words are still concentrated around $1/(k+1)$, whereas data samples of

NCE +	$Z_c \in \theta$	Bias init. at $\log(P_n)$	Both
Baseline model	179.3	151.8	148.4
$k = 5$	247.3	221.4	231.5
$k = 10$	299.0	204.4	198.2
$k = 25$	191.9	171.2	189.6
$k = 50$	184.7	157.3	167.6
$k = 250$	167.0	141.6	144.3
$k = 500$	164.6	137.5	138.4
$\alpha = 0.0$	335.0	322.4	275.4
$\alpha = 0.25$	201.8	205.0	193.5
$\alpha = 0.50$	178.1	172.7	159.6
$\alpha = 0.75$	173.5	155.0	156.3

Table 3.4: Best final perplexities on the test set of the Penn Treebank (PTB) corpus obtained with NCE and the two tricks presented in this section: learning conjointly a parameter Z_c and initializing the output bias to the values of P_n - with varying number of samples k and distortion α of the noise distribution P_n.

frequent words see their value of r_θ increase and the noise samples of frequent words see their value of $r_\theta(w|H)$ decrease. From epoch 5, a large number of data samples of non-frequent words have higher values of r_θ and are completely separated from noise samples. At the end of training, the repartition of these ratios resembles the one obtained with the normalized model, in Figure 3.14.

The second column of Table 3.4 shows the efficiency of this method across values of k and α. Strangely, except for a few configurations, including our baseline, using this initialization jointly with learning a parameter scale Z_c gives worse results than the initialization alone. We suspect this method to be efficient enough that learning a supplementary parameter (that is very impactful) makes training more difficult.

3.3.3 Summary of results with sampling-based algorithms

As shown in Table 3.6, with our baseline parameters choices, we get our best model on PTB by both using this initialization and learning Z_c. The final perplexity is quite better than with Normalized NCE: pairing the 'learning to scale' approach

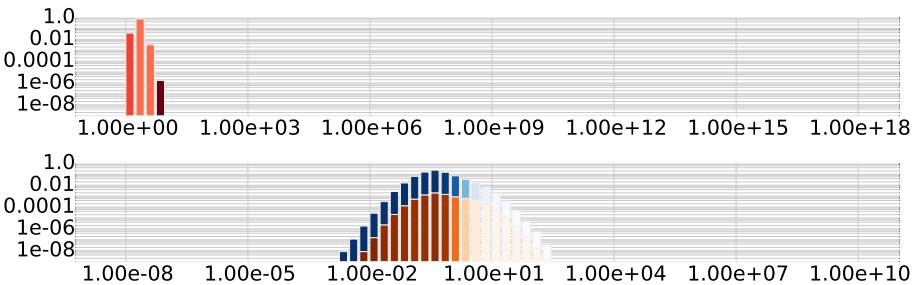

Figure 3.18: Repartition of the values of the partition function $Z_\theta(H)$ (*top*) and ratio $r_\theta(w|H)$ (*bottom*) for all training examples (H, w) at initialization, for a model whose output bias is initialized to the values of $\log P_n(w)$.

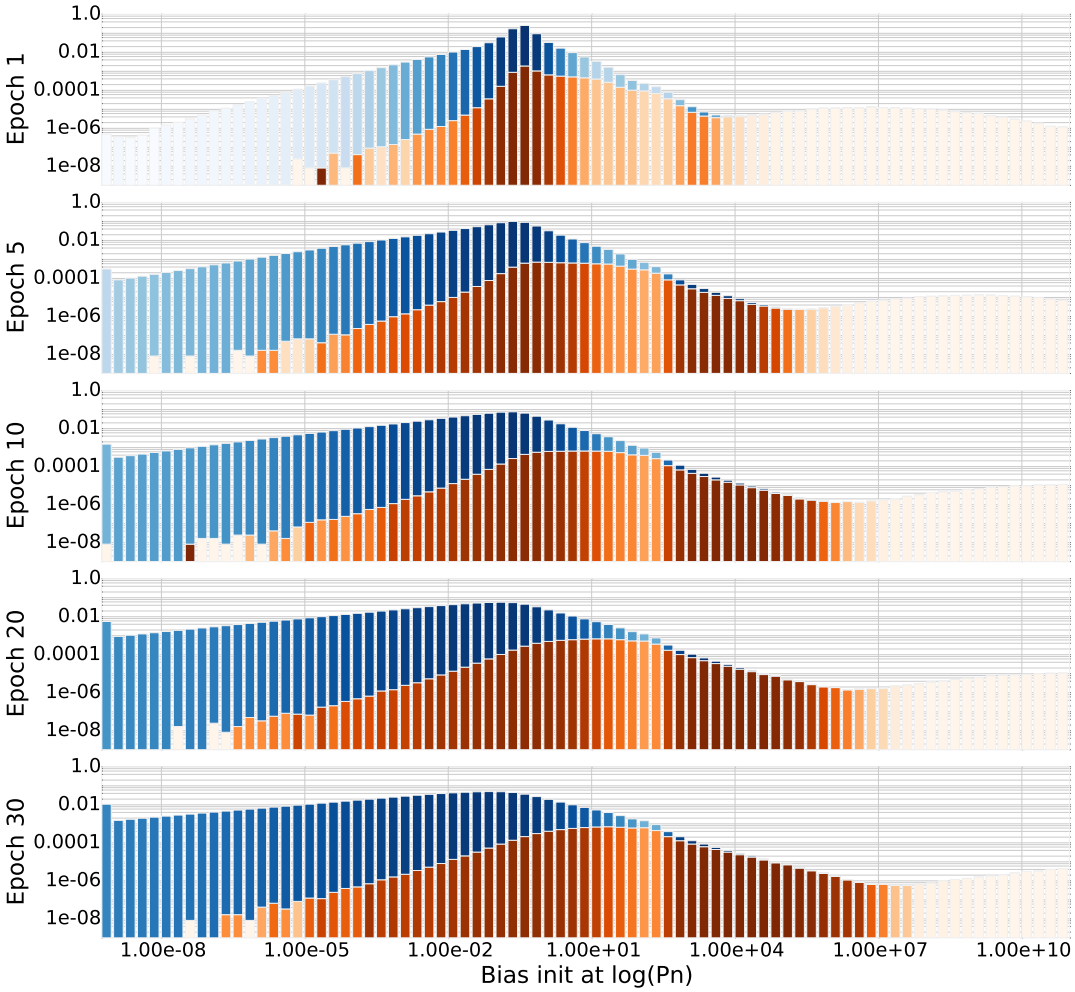

Figure 3.19: Repartition of the values of the ratio $r_\theta(w|H)$ for all positive training examples (H, w) in orange, and associated noise sample $(\hat{w}_i)_{i=1}^k$, in blue, during several training epochs on PTB. The model is trained with NCE, with a output bias initialized to the values of $\log P_n(w)$.

with the adapted initialization scheme makes NCE far more competitive, with no additional computational cost. Figure 3.20 also shows a learning curve far smoother than those of methods previously presented. We also present in Table 3.6 results with slight smoothing of the noise distribution, for the methods who profit from it.

Hyperparameter	Value
Number of noise samples k	100
Default noise distribution P_n	Unigram
Hidden Layer	LSTM
Maximum Sentence Length	30
Number of hidden layers	1
Word embedding dimension	512
Hidden layer dimension	512

Hyperparameter	Value
Batch size (Sentences)	1000
Dropout rate	0.9
Optimization method	Adam
Learning rate	0.01
ϵ	0.0001
Grad clipping value	0.0
Maximum number of training epochs	3

Table 3.5: Structural choices and hyperparameters used on the 1 Billion Words Benchmark for experiments presented in this chapter.

Training method	PTB	1BW Benchmark
MLE	150.2	-
Normalized NCE	159.3	-
NCE	306.0	X
NCE + $\alpha = 0.75$	277.0	X
Importance Sampling	168.3	77.9
Importance Sampling + $\alpha = 0.75$	171.5	67.2
BlackOut	169.0	71.8
NS	228.3	102.4
NS + $\alpha = 0.75$	195.8	83.7
NCE + $Z_c = \|\mathcal{V}\|$	178.6	72.8
NCE + $Z_c \in \theta$	172.3	70.9
NCE + Bias init. at $\log(P_n)$	151.8	74.8
NCE + Bias init. at $\log(P_n)$ + $Z_c \in \theta$	148.4	71.3

Table 3.6: Best testing perplexities obtained on PTB with a full vocabulary and on the 1 Billion Word Benchmark with a vocabulary of $64K$ words. 'X' indicates that the model did not reach a perplexity under the size of the vocabulary within the maximal number of training epochs 50.

Results on the 1 Billion Words Benchmark are presented in the second column of Table 3.6. In this case, the vocabulary contains $64K$ words, which renders normalized models very difficult to use in practice[6]. We trained the models for a minimum of one epoch and up to two more if models made progress on the validation set perplexity. The hyperparameters we used are shown in Table 3.5. Final results are a little different than with the PTB: initializing the output bias to $P_n(w)$ is here a little less efficient, while smoothing the noise distribution with $\alpha = 0.75$ is more impactful for other sampling-based method, making IS slightly better. We suppose this is due to the shape of the noise distribution, which, being cut at $64K$ words, only contains frequent words. On the one hand, it makes for a more peaky distribution for very frequent words, which makes training with basic NCE more difficult. Self-normalization is harder and 3 epochs are not enough for the testing perplexity to be brought under the size of the vocabulary. On the other hand, having no long tail composed of very rare words on the noise distribution makes training a little easier for other methods, which renders our second trick less necessary.

3.4 Conclusions

In this chapter, we present an in-depth analysis of the inner workings of noise contrastive estimation and propose solutions to ease training allowing the method to reach at least comparable results with importance sampling, which is often considered more efficient. First, an exhaustive series of experiments comparing results of importance sampling (IS), noise contrastive estimation (NCE), BlackOut and negative sampling (NS) when varying the number k of noise samples and the choice of the noise distribution P_n shows NCE to perform badly and IS to still outperf-

[6]Besides the computation time constraints, the batch size is rapidly limited by the GPU memory, while a smaller batch size also increases the computation time.

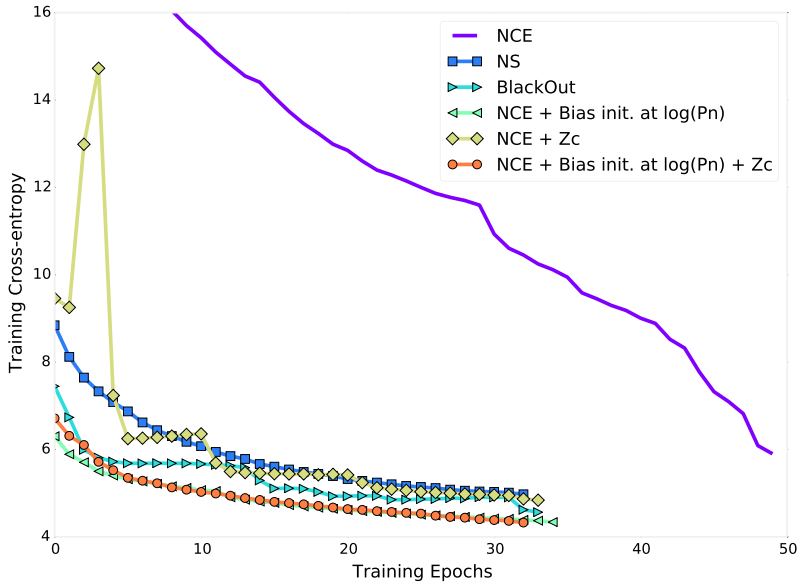

Figure 3.20: Training cross-entropy curves on PTB for un-normalized models trained with NCE, variations of NCE and alternative additions presented in Section 3.3.

morm BlackOut and NS. This leads us to closely monitor the training of two models trained with NCE, where we normalize the scores obtained by one of them.

An analyis of the objective function and its gradients first highlights how self-normalization is crucial if we want the NCE to learn. However, the higher P_n is, the less self-normalization has an impact on learning. That also means that infrequent words coming from data samples are harder to learn about and necessitate the model to self-normalize for the associated context. Hence, the large variance of values taken by the noise distribution has a key role in the difficulties met when using NCE, especially for language modeling with large vocabularies, because of the Zipfian shape of the unigram distribution. We verify this analysis using the monitored experiments.

Finally, we propose two solutions to ease training with NCE. First, when including a scaling parameter in the parameters of the model, the scaling process that should be attained via self-normalization can be both stable and efficient. Additional improvements can be obtained by initializing the bias of the output layer according to the noise distribution. This limits the effects of the Zipfian shape of the unigram distribution. Combining this 'learning to scale' approach with the adapted initialization scheme yields a very competitive, yet simple, training strategy for neural language models with large vocabularies and shows convincing results on both the Penn Treebank and the 1 Billion Words Benchmark.

Chapter 4

Extending Sampling-Based Algorithms

Contents

Whether we are performing maximum likelihood estimation or noise contrastive estimation, training a neural language model is done by otpimizing a function of our model parameters θ. This objective function is expressed depending on the probability distribution (or its un-normalized version) output by the model. But this objective is also commonly called *loss* function: that is because it is the expression of a dissimilarity between the parametrized distribution that we are learning and the distribution described by our training data. Learning is in fact minimizing this measure of divergence. For example, in the case of maximum likelihood estimation, the negative likelihood that we minimize is also the *Kullback-Leibler* divergence between the distribution output by the model and the data distribution:

$$NLL(\theta) = -\sum_{(H,w)\in\mathcal{D}} \log P_\theta^H(w) = -\sum_{(H,w)\in\mathcal{H}\times\mathcal{V}} P_\mathcal{D}^H(w) \log P_\theta^H(w) = D_{KL}(P_\mathcal{D}||P_\theta) \tag{4.1}$$

Our knowledge on the data distribution is simply represented by examples that we sample from the training data.

In this chapter, we will work with the divergence measures known as *Bregman divergences*: the Kullback-Leibler divergence or the Euclidean squared distance are examples of such divergences. We will highlight how Maximum-Likelihood Estimation and some of the sampling-based methods described in Chapter 2 have their objective functions derived from Bregman divergences, and are therefore part of a generalized estimation framework. We will do this generalization in order to gain insight on the inner workings of sampling-based algorithms and to derive new objective functions. The reasoning presented here is based on two successive generalizations. Firstly, the work of Pihlaja et al. (2012), which described NCE (2.3.1) and *score matching* (Hyvärinen and Dayan, 2005) as part of a set of sampling-based objective functions that lead to a consistent estimation of parameters. Secondly, an extension of this set to a larger class of functions, which are Bregman divergences, in Gutmann and Hirayama (2011).

In Section 4.1, we will shortly review Bregman divergences and how they can be used to design objective functions for neural probabilistic language modeling. A Bregman divergence is defined as a measure of distance corresponding to a strictly convex function: we will re-obtain methods presented earlier by using the suitable function. For example, the Kullback-Leibler divergence can be obtained by using the Bregman divergence associated to the negative entropy function on a multinomial probability distribution. By minimizing this divergence applied to the data probability distribution and the model probability distribution, we obtain the classic Maximum Likelihood Estimation objective.

Some of the methods presented earlier will be obtained by using the corresponding divergence on another domain of application: instead of working with probability distributions, we will work directly with un-normalized model distributions (which we could also call *measures*), belonging to $\mathbb{R}_+^{|\mathcal{V}|}$, in order to avoid the costly normalization. In Section 4.2, we will present how we are able to use these divergences without any summation on the full vocabulary, and experiment with example functions presented in Gutmann and Hirayama (2011). Then, we will apply to language modeling a set of Bregman divergences presenting interesting properties (*Theorem 4*,

Csiszar (1991)). Also known as *Beta divergences* (*Section 3*, Cichocki and ichi Amari (2010)), they are Bregman divergences derived with a class of generalized entropy functions (Eguchi and Kato, 2010), obtained via a parametrized power transformation.

In Section 4.3, we will show that objectives based on a surrogate binary classification task, like NCE, can also be obtained by applying a Bregman divergence to a set of binary classification probabilities — instead of multinomial classification probabilities of words into the vocabulary. A composition function allows us to go from one divergence to its corresponding "binary classification" version. We can show that the NCE objective can be derived by applying the Bregman divergence obtained from the negative entropy function to the binary probabilities of classification. This allows us to experiment with a new series of divergences, that come from the previously used class of generalized entropies, followed by the composition function.

4.1 Language model objective functions as Bregman divergences

Bregman divergences were first described in Bregman (1967). Their functional form provides a class of divergences between two functions, indexed by convex functions, which include several existing distance[1] measures, as the Euclidean distance or the Kullback-Leibler divergence. Informally, Bregman divergences measure the distance between a convex function and its tangent hyperplane. Convexity gives these divergences interesting properties, which makes them frequently used in statistical learning. A recent characterization of Bregman divergences describes them as the loss functions for which the unique optimal predictor is the conditional expectation $\mathbb{E}[X|Z]$, when predicting a random variable X from observations Z (Banerjee et al., 2005).

4.1.1 Learning by minimizing a Bregman divergence

The *functional* Bregman divergence corresponding to a function $\Phi : \mathcal{S} \to \mathbb{R}$, which is a strictly convex function defined on a convex set $\mathcal{S}$ and twice continuously differentiable on this set, is defined for all $f, g \in \mathcal{S}$ as:

$$D_\Phi(f, g) = \Phi(f) - \Phi(g) - \nabla\Phi(g)^T(f - g) \tag{4.2}$$

where ∇ denotes the gradient of a function. We can note that the functional definition is a generalization of the *vector* and *point-wise* versions (Frigyik et al., 2008). The Bregman divergence holds several nice properties, which are detailed and proved in various works (Censor and Zenios, 1997; Grünwald and Dawid, 2004; Frigyik et al., 2008).[2] The main one is **non-negativity**: it stems directly from the strict convexity

[1] That are not formal *distances*: Bregman divergences in general are not.

[2] However, the definition is not universal and may vary depending on the author's purpose. The reader may use Appendix B and C of Frigyik et al. (2008) for a list of properties and proofs.

of Φ. We can also note the **linearity** and the **affine invariance** of the divergence with respect to Φ.

We can now specify the functional objects we will manipulate. Since we work with multinomial distributions over a vocabulary $\mathcal{V}$, we can assume that our functions can be defined attributing a value to each word of that vocabulary:

$$\begin{aligned} f : \mathcal{V} &\rightarrow \mathbb{R} \\ w &\rightarrow f(w) \end{aligned} \tag{4.3}$$

and will therefore be applied to a discrete space. Then, to simplify the objects we manipulate, f can be represented by a vector $\mathbf{f}$ of $|\mathcal{V}|$ values $(f_w)_{w\in\mathcal{V}}$, that verifies $\mathbf{f} \in \mathcal{S} \subset \mathbb{R}^{|\mathcal{V}|}$. Knowing this, we can outline that in this chapter, we will only work with three possible $\mathcal{S}$: first, the set of probability distributions over $\mathcal{V}$ noted $\Delta^{\mathcal{V}}$, where

$$\Delta^{\mathcal{V}} = \left\{ \mathbf{f} : \mathbf{f} \in \mathbb{R}_+^{|\mathcal{V}|}, \sum_{w\in\mathcal{V}} f_w = 1 \right\} \tag{4.4}$$

Secondly, we will use $\mathbb{R}_+^{|\mathcal{V}|}$ when we wish to work with un-normalized distributions. Lastly, we will use $[0,1]^{|\mathcal{V}|}$ when working with binary classification probabilities for every word in the vocabulary. However, for the sake of consistency, we will retain the functional notations f and $f(w)$, as in the rest of this dissertation, the scores and probabilities outputed by a language model are mostly denoted as functions over $\mathcal{V}$ that we evaluate for a word w. In this context, and following Gutmann and Hirayama (2011), it makes sense to restrict ourselves to using a specific kind of Bregman divergence: the *separable* Bregman divergence (*Section 3.5*, Grünwald and Dawid (2004)). This means that the function Φ can be written as:

$$\Phi(f) = \sum_{w\in\mathcal{V}} \phi(f(w))\mu_{\mathcal{V}}(w) \tag{4.5}$$

where ϕ has the same properties as Φ, notably the strict convexity. Depending on our choice of $\mathcal{S}$, ϕ is defined over $[0,1]$ or $\mathbb{R}_+$. $\mu_{\mathcal{V}}(w)$ represents a probability mass function over $\mathcal{V}$ that allows us to weight the contribution of each dimension (which represent words) to Φ, without any loss of generality. It can, in the simpler case, be the uniform distribution. This allows us to compute the divergence between f and g as a weighted sum of the divergences of their values over their domain $\mathcal{V}$. This restriction on Φ is necessary in order to be able to work with un-normalized models: we do not want to have to evaluate our functions on more than one word w at a time when computing our objective function. Then, the associated divergence is written (using the linearity property):

$$D_\Phi(f,g) = \sum_{w\in\mathcal{V}} D_\phi(f(w), g(w))\mu_{\mathcal{V}}(w) \tag{4.6}$$

From the non-negativity, we obtain that

$$D_\Phi(f,g) = 0 \quad \Rightarrow \quad f = g$$

We use the divergence to approximate a function representing our data by a parametric function representing our model. Assuming the first one is fixed, our problem is to minimize the Bregman divergence with respect to the second. Thus, we note $f_{\mathcal{D}}$ the function representing our data, and f_θ the parametric function. We then minimize the following divergences:

$$D_\Phi(f_{\mathcal{D}}, f_\theta) = \sum_{w \in \mathcal{V}} D_\phi(f_{\mathcal{D}}(w), f_\theta(w)) \mu_{\mathcal{V}}(w) \tag{4.7}$$

which is equivalent to minimizing the objective:

$$L_\phi(f_\theta) = \sum_{w \in \mathcal{V}} \left[-\phi(f_\theta(w)) - \phi'(f_\theta(w))(f_{\mathcal{D}}(w) - f_\theta(w)) \right] \mu_{\mathcal{V}}(w) \tag{4.8}$$

Decomposing the objective in two terms

We can notice that our objective is divided into two parts: one that depends on the data function $f_{\mathcal{D}}$, and the second that only depends on f_θ. Following Gutmann and Hirayama (2011) again, we re-write this objective with two distinct functions to highlight this separation:

$$S_0^\phi(x) = -\phi(x) + \phi'(x)x \quad \text{and} \quad S_1^\phi(x) = \phi'(x) \tag{4.9}$$

which satisfy, on the domain of ϕ:

$$\frac{(S_0^\phi)'(x)}{(S_1^\phi)'(x)} = x \quad \text{and} \quad (S_1^\phi)'(x) > 0 \tag{4.10}$$

Now, the objective defined in Equation 4.8 can be written:

$$L_{S^\phi}(f_\theta) = \sum_{w \in \mathcal{V}} \left[S_0^\phi(f_\theta(w)) - S_1^\phi(f_\theta(w)) f_{\mathcal{D}}(w) \right] \mu_{\mathcal{V}}(w) \tag{4.11}$$

Accounting for the context

However, our multinomial distributions are conditional: they depend on a context H. Since they share their parameters θ, we cannot learn them independently with context dependent objectives, but we can define a global objective:

$$L_{S^\phi}(f_\theta) = \sum_{H \in \mathcal{H}} L_{S^\phi}^H(f_\theta^H) \mu_{\mathcal{H}}(H) \tag{4.12}$$

where $\mathcal{H}$ is the discrete set of possible contexts, and $\mu_{\mathcal{H}}(w)$ a probability mass function over $\mathcal{H}$, with:

$$L_{S^\phi}^H(f_\theta^H) = \sum_{w \in \mathcal{V}} \left[S_0^\phi(f_\theta^H(w)) - S_1^\phi(f_\theta^H(w)) f_{\mathcal{D}}^H(w) \right] \mu_{\mathcal{V}}(w) \tag{4.13}$$

We should note that this objective can be also written as the minimization of only one Bregman divergence: we can consider the functions $f_{\mathcal{D}}$ and f_θ to be defined over $\mathcal{H} \times \mathcal{V}$. Then, if we assume that $f_{\mathcal{D}}$ is included in the possible models, and that there exists a set of parameters θ_* such that $f_{\theta_*} = f_{\mathcal{D}}$, the non-negativity property of the Bregman divergence tells us that it is a minimum of the objective L_S, while the convexity property makes it unique.

How to obtain a sample objective ?

Since our goal is to approach $f_{\mathcal{D}}$, the only way for us to obtain an usable objective is to compute the second term as a sample average over examples drawn from $\mathcal{D}$. We will examine the possible ways to do so; however, the difficulty mainly resides in computing the first term efficiently.

4.1.2 Directly learning the data distribution

The most evident choice for our objective is to choose to directly learn the data probability distribution: we then set $f_{\mathcal{D}} = P_{\mathcal{D}}$ and $f_\theta = P_\theta$, and the probability mass function $\mu_{\mathcal{V}}(w)$ can be the uniform distribution.

Normalized distribution and MLE

In that case, we can begin by setting $\mathcal{S} = \Delta^{\mathcal{V}}$, as presented in Equation 4.4, and our objective is now, for a given context H:

$$L^H_{S^\phi}(P^H_\theta) = \sum_{w \in \mathcal{V}} S^\phi_0(P^H_\theta(w)) - \sum_{w \in \mathcal{V}} S^\phi_1(P^H_\theta(w)) P^H_{\mathcal{D}}(w) \tag{4.14}$$

If we choose, as convex function:

$$\begin{array}{rccc} \phi: & [0,1] & \to & \mathbb{R} \\ & x & \to & x\log(x) \end{array} \tag{4.15}$$

which is defined at the origin by continuity. Consequently, Φ is defined as the negative entropy function:

$$\begin{array}{rccc} \Phi: & [0,1]^{|\mathcal{V}|} & \to & \mathbb{R} \\ & (x_1, \dots, x_{|\mathcal{V}|}) & \to & \displaystyle\sum_{i=1}^{|\mathcal{V}|} x_i \log(x_i) \end{array} \tag{4.16}$$

It is easy to compute $\phi'(x) = \log(x) + 1$, which gives

$$S^\phi_0(x) = x \quad \text{and} \quad S^\phi_1(x) = \log(x) + 1$$

Our objective is then:

$$L^H_{KL}(P^H_\theta) = -\sum_{w \in \mathcal{V}} P^H_{\mathcal{D}}(w) \log(P^H_\theta(w)) \tag{4.17}$$

which amounts to minimizing the Kullback-Leibler (KL) divergence between $P_{\mathcal{D}}$ and P_θ - that is exactly the MLE objective. However, this implies computing the partition function.

Working with un-normalized distributions ?

We now assume working with an un-normalized parametric distribution, which implies setting $\mathcal{S} = \mathbb{R}^{|\mathcal{V}|}$. In accordance with previous notation, we now note $f_\theta = p_\theta$. If we choose ϕ as the negative entropy extended to $\mathbb{R}_+$, for a given context H, we obtain the objective:

$$L_{KL}^H(p_\theta^H) = \sum_{w \in \mathcal{V}} p_\theta^H(w) - \sum_{w \in \mathcal{V}} P_{\mathcal{D}}^H(w) \log(p_\theta^H(w)) \tag{4.18}$$

We can consider the second term as an expectation over the data distribution, and evaluate its sample average by drawing examples from the training data. However, the first term still implies summing on the whole vocabulary - and the issue persists for any other choice of ϕ. We should note that we could approximate the computation of our first term via importance sampling. However, since we are working with un-normalized distribution, it resembles *simple* importance sampling. This is probably harder to use than approximating the gradient of objective 4.17, which is self-normalized importance sampling, that is the method traditionaly used in language modeling (Section 2.2.1).

4.2 Learning un-normalized models using Bregman divergences

As explained in Gutmann and Hyvärinen (2010), an estimation problem can be formulated as a classification problem: by discriminating between observed data samples and noise samples, we can learn a decision boundary between the two. Since we 'know' the properties of the noise distribution, we can learn about the properties of the data distribution via this boundary. Following Gutmann and Hirayama (2011), we can generalize this approach to obtain a class of objectives via Bregman divergences, which are the estimators described in Pihlaja et al. (2012).

4.2.1 Learning by matching the ratio of data and noise distributions

Let us assume that we have access to a noise distribution P_n that we can sample from, and that examples sampled from the data distribution $P_{\mathcal{D}}$ (of class $C = 1$) and P_n (of class $C = 0$) are mixed together, with a ratio of k noise samples by data sample, as in Section 2.3.1. We can define the ratios of distributions:

$$r_{\mathcal{D}}^H(w) = \frac{P_{\mathcal{D}}^H(w)}{kP_n(w)} \quad \text{and} \quad r_\theta^H(w) = \frac{P_\theta^H(w)}{kP_n(w)} \tag{4.19}$$

Gutmann and Hirayama (2011) shows that learning such a ratio implies that we are learning an optimal classifier, while the inverse is not necessarily true. However, putting aside the link to the classification task, we will for now focus on the objective we obtain by choosing the probability mass function $\mu_{\mathcal{V}}(w)$ to be the noise distribution kP_n. Both the data and the model ratio are positive: here, we have $\mathcal{S} = \mathbb{R}^{|\mathcal{V}|}$. The Bregman divergence associated to Φ between $r_{\mathcal{D}}^H$ and r_θ^H can be written as a function of P_θ^H, which gives the objective function:

$$L_{S^\phi}^H(P_\theta^H) = k \sum_{w \in \mathcal{V}} S_0^\phi \left(\frac{P_\theta^H(w)}{kP_n(w)} \right) P_n(w) - \sum_{w \in \mathcal{V}} S_1^\phi \left(\frac{P_\theta^H(w)}{kP_n(w)} \right) P_{\mathcal{D}}^H(w) \tag{4.20}$$

As we have seen in Section 2.3.1, we could parametrize separately the partition function. The solution adopted then was to set these parameters to 0 and let the model self-normalize. It amounts to simply working with an un-normalized model distribution p_θ^H in the model ratio, which is what we will do in the remainder of this section. Then, the objective equals:

$$L_{S^\phi}^H(p_\theta^H) = k\mathbb{E}_{\hat{w} \sim P_n} \left[S_0^\phi \left(\frac{p_\theta^H(\hat{w})}{kP_n(\hat{w})} \right) \right] - \mathbb{E}_{w \sim P_{\mathcal{D}}^H} \left[S_1^\phi \left(\frac{p_\theta^H(w)}{kP_n(w)} \right) \right] \tag{4.21}$$

from which we obtain a working sample objective:

$$\hat{L}_{S^\phi}^H(p_\theta^H) = \sum_{\substack{i=1 \\ \hat{w}_i \sim P_n}}^{kN_{\mathcal{D}}} \left[S_0^\phi \left(\frac{p_\theta^H(\hat{w}_i)}{kP_n(\hat{w}_i)} \right) \right] - \sum_{\substack{i=1 \\ w_i \sim P_{\mathcal{D}}^H}}^{N_{\mathcal{D}}} \left[S_1^\phi \left(\frac{p_\theta^H(w_i)}{kP_n(w_i)} \right) \right] \tag{4.22}$$

The set of parameters $\hat{\theta}_*$ maximizing the global version of the sample objective $\hat{L}_{S^\phi}$ defines our estimator. Gutmann and Hirayama (2011) explain that it can be proven that, under some technical conditions, this estimator is consistent (converging in probability to the complete objective optimum θ^* when the number of samples is large enough) with an asymptotic normal mean square error. These conditions are explicited in Pihlaja et al. (2012). They are that P_n is non-zero whenever $P_{\mathcal{D}}$ is, that there is uniform convergence of the sample objective towards the Bregman divergence, and model identifiability (one-to-one correspondence between θ and f_θ) which translates into having a full-rank fisher information matrix. This is a generalization of theorems 10.13 and 10.18 for proof of consistency of Maximum Likelihood Estimation in Wasserman (2004), under similar conditions.

4.2.2 Experimenting with learning un-normalized models

We have just defined a class of objective functions that can be used to train un-normalized models, derived from a Bregman divergence that we obtain from a strictly convex function ϕ defined on $\mathbb{R}_+$. Pihlaja et al. (2012) gives several examples of objectives, with their associated functions $S_{0,1}^\phi$. They are shown in Table 4.1, along with the corresponding function ϕ, and r represents the ratio $p_\theta^H(w)/kP_n(w)$. We

Name	$\phi(r)$	$S_0^\phi(r)$	$S_1^\phi(r)$
IS	$r\log(r) - r$	r	$\log r$
PO	$\frac{r^2}{2}$	$\frac{r^2}{2}$	r
NCE	$r\log(r) - (1+r)\log(1+r)$	$-\log\left(\frac{1}{1+r}\right)$	$\log\left(\frac{r}{1+r}\right)$
InvPO	$-\frac{1}{r}$	$-\frac{1}{r}$	$-\frac{1}{2r^2}$
InvIS	$-1 - \log(r)$	$\log r$	$-\frac{1}{r}$

Table 4.1: Choices of ϕ corresponding to S_0 and S_1 functions presented in Pihlaja et al. (2012).

can see that S_0^ϕ and S_1^ϕ correspond respectively to the noise term and minus the data term of the objective. The objective denoted as IS is obtained, again, from the negative entropy extended to the axis of positive number[3]. This particular choice of ϕ makes the objective the same as if we applied importance sampling to the objective of Equation 4.13, and differs from the objective of Section 2.2 by not approximating the partition function. We can also re-derive the NCE objective. PO, InvPO and InvIS denote Polynomial, Inverse Polynomial and Inverse IS functions.

The $S_{0,1}^\phi$ functions determine how much we penalize noise and data samples depending on the relative probability mass attributed to the sample by the model compared to the noise distribution. For example, the polynomial objective, being squared on the data term, penalizes far more noise samples $\hat{w}$ where the ratio $p_\theta^H(\hat{w})/kP_n(\hat{w})$ is higher than 1. Now that we are ready to apply our objectives to language modeling, we must also be careful: we have seen in Section 3.2 that the ratio $p_\theta^H(\hat{w})/kP_n(\hat{w})$ could take from very small to very large values. This is true even at initialisation, before the NCE objective we used could have any influence on learning. While with NCE, the functions $S_{0,1}^\phi$ are stable for any values of r, it clearly is not the case for the other examples presented here.

First experimental results

In a first series of experiments, we trained models on the Penn Treebank, following the same setup that is exposed at the beginning of the previous chapter, with the hyperparameters presented in Table 3.2. We tried the IS, InvIS, PO and InvPO objectives, trained with SGD with varying learning rates (1.0, 0.1, 0.01). We also tried the tricks presented at the end of the previous chapter, in Section 3.3, to ease learning. As expected, in all configurations, the scores outputed by InvPO begin to overflow during the first epoch of training. This is also mostly the case for the IS objective, but using both training tricks and a low learning rate allows the model to progress, even if learning is very slow and the perplexity reached stays very high. The InvIS objective also overflows in most cases, but when we initialize the bias to the values of $P_n(w)$, it manages to learn; however, it stagnates after reaching a test set perplexity of 900. Inversely, the PO objective seems to be learning properly, but

[3]The ϕ corresponding to the functions for IS of Pihlaja et al. (2012) is not exactly the negative entropy, but Bregman divergences are affine invariant with respect to ϕ.

is very slow for every learning rate, and progress seems to stop for very high test set perplexity. This is still verified when we learn a scaling parameter. However, when we initialize the bias to the values of $P_n(w)$, the model overflows too.

Power-transformed entropy and beta divergence

In order to gain a better flexibility and to avoid overflows, we investigated the use of power transformations. In the four objectives presented earlier, at least one of the functions $S^{\phi}_{0,1}$ has a linear or squared (or inverse and inverse squared) term: we would like to experiment with functions that are transformed by a power β that verifies $|\beta| < 1$. To do so, we use, as function Φ, a power entropy, parametrized by a number $\beta \in \mathbb{R}$ [4]:

$$\phi_\beta(r) = \begin{cases} \frac{1}{\beta(\beta-1)}(r^\beta - \beta r + \beta - 1) & \text{if } \beta \neq 0, 1 \\ r\log(r) - r + 1 & \text{if } \beta = 1 \\ r - \log(r) - 1 & \text{if } \beta = 0 \end{cases} \tag{4.23}$$

It is easy to verify that ϕ_β is strictly convex for any β. We should note that here again, since Bregman divergences are affine invariant with respect to ϕ, modifying the constant and linear terms in the entropy above does not change the resulting objective. Knowing that, we immediately observe that this class of functions generalizes the functions ϕ we used to obtain the objectives IS and InvIS, for $\beta = 1$ and $\beta = 0$. We can also easily derive that PO corresponds to $\beta = 2$ and InvPO to $\beta = -1$. In fact, the class of Bregman divergences derived from the ϕ_β functions are known as *Beta-divergences* (Basu et al. (1998)). They possess an interesting property: they are the Bregman divergences that are homogeneous of degree β, which means they are characterized by their multiplicative scaling behaviour (Csiszar, 1991; Cichocki and ichi Amari, 2010):

$$\forall \lambda \in \mathbb{R}, D_{\Phi_\beta}(\lambda f, \lambda g) = \lambda^\beta D_{\Phi_\beta}(f, g) \tag{4.24}$$

In the case $\beta = 1$, we find that D_{Φ_1} is the KL-divergence extended to vectors of positive numbers, or *I-divergence*, while with $\beta = 2$, we obtain that D_{Φ_2} is the standard squared Euclidean distance, or L_2 norm. The functions $S^{\phi_\beta}_{0,1}$ associated to ϕ_β are:

$$S_0^{\phi_\beta}(r) = \begin{cases} \frac{r^\beta - 1}{\beta} \\ r - 1 \\ \log(r) \end{cases} \quad \text{and } S_1^{\phi_\beta}(r) = \begin{cases} \frac{r^{\beta-1}-1}{\beta-1} & \text{if } \beta \neq 0, 1 \\ \log(r) & \text{if } \beta = 1 \\ 1 - \frac{1}{r} & \text{if } \beta = 0 \end{cases} \tag{4.25}$$

Then, for our particular application, we are mainly interested in values of β that are strictly between 0 and 1. We experiment with several values, using the

[4] Which is equivalent to the Tsallis q-entropy, with the relation $\beta = q - 1$ (Eguchi and Kato, 2010)

β	Perplexity
0.01	X
0.1	404.0
0.25	236.0
0.5	251.8
0.75	302.7
0.9	593.2
0.99	X

Table 4.2: Best final perplexities on the test set of the Penn Treebank (PTB) corpus obtained with Bregman divergences derived from ϕ_β, for various values of β such that $0 < \beta < 1$. 'X' indicates that the model did not reach a perplexity under the size of the vocabulary within the maximal number of training epochs.

same hyperparameters as previously indicated, and both initializing bias with P_n and learning a scaling parameter. Results are presented in Table 4.2. While our newly defined objectives do learn, their performances are not satisfactory, and for the most part far from those obtained with the NCE in a similar configuration.

4.3 From learning ratios to directly learning classification probabilities

In the previous section, we have seen that the NCE objective can be seen as the minimization of a Bregman divergence between the ratio of data and noise distributions, and the ratio of model and noise distributions. However, we had difficulties learning with other Bregman divergences derived from simple functions - and we obtained disappointing performances with working objectives. To understand the performance of NCE compared to the other objectives we experimented with, we look into the surrogate classification task. If we note the posterior class probabilities of an example coming from the data ($C = 1$) or the noise ($C = 0$) as:

$$P^H_{\mathcal{D}}(C = 1|w) = \frac{P^H_{\mathcal{D}}(w)}{P^H_{\mathcal{D}}(w) + kP_n(w)} \quad P^H_{\mathcal{D}}(C = 0|w) = 1 - P^H_{\mathcal{D}}(C = 1|w) \tag{4.26}$$

and that we define similar posterior classification probabilities using the model distribution, we easily derive that:

$$r^H_\theta(w) = \frac{P^H_\theta(C = 1|w)P(C = 0)}{kP^H_\theta(C = 0|w)P(C = 1)} = \frac{P^H_\theta(C = 1|w)}{P^H_\theta(C = 0|w)} = \frac{P^H_\theta(C = 1|w)}{1 - P^H_\theta(C = 1|w)} \tag{4.27}$$

While this shows that the ratio can serve as discriminant function between the two classes, and that it allows us to learn an optimal classifier, it also gives us a direct link between the ratio and the quantity that is directly optimized when learning for a binary classification task: $P^H_\theta(C = 1|w)$. This quantity can be obtained from the ratio as follows:

$$P_\theta^H(C=1|w) = \frac{r_\theta^H(w)}{1+r_\theta^H(w)} \tag{4.28}$$

4.3.1 Minimizing the divergence between posterior classification probabilities and link to NCE

We can consider the probabilities of recognizing the data as functions defined on $\mathcal{V}$ taking values in $[0,1]$. That means that here, we have $\mathcal{S} = [0,1]^{|\mathcal{V}|}$ — we can then apply the general objective of Equation 4.13 to minimize the divergence between the functions $P_\mathcal{D}^H(C=1|.)$ and $P_\theta^H(C=1|.)$. The probability mass function $\mu_\mathcal{V}(w)$ is the mixture from which we draw the samples:

$$\mu_\mathcal{V}(w) = P_\mathcal{D}^H(w) + kP_n(w)$$

We obtain the following objective:

$$L_{S^\phi}^H(P_\theta^H(C=1|.)) = \sum_{w\in\mathcal{V}} \left[S_0^\phi(P_\theta^H(C=1|w)) - S_1^\phi(P_\theta^H(C=1|w))P_\mathcal{D}^H(C=1|w) \right] \mu_\mathcal{V}(w) \tag{4.29}$$

First, let us use the negative entropy of a binary random variable as convex function $\phi_{BinClassif}$:

$$\begin{array}{rccl} \phi^{BinClassif}: & [0,1] & \to & \mathbb{R} \\ & p & \to & p\log(p) + (1-p)\log(1-p) \end{array} \tag{4.30}$$

Then, we obtain the functions

$$S_0^{BinClassif}(p) = \log\left(\frac{1}{1-p}\right) \quad \text{and} \quad S_1^{BinClassif}(p) = \log\left(\frac{p}{1-p}\right) \tag{4.31}$$

which gives us the objective:

$$\begin{aligned} &L_{BinClassif}^H(P_\theta^H(C=1|.)) = \\ &\quad -\sum_{w\in\mathcal{V}} \log(P_\theta^H(C=0|w))\mu_\mathcal{V}(w) \\ &\quad -\sum_{w\in\mathcal{V}} \left[\log(P_\theta^H(C=1|w)) - \log(P_\theta^H(C=0|w))\right] P_\mathcal{D}^H(C=1|w)\mu_\mathcal{V}(w) \end{aligned} \tag{4.32}$$

that we can factorize as:

$$
\begin{aligned}
&L^H_{BinClassif}(P^H_\theta(C=1|.)) = \\
&\quad -\sum_{w\in\mathcal{V}} \log(P^H_\theta(C=0|w))(1-P^H_\mathcal{D}(C=1|w))\mu_\mathcal{V}(w) \\
&\quad -\sum_{w\in\mathcal{V}} \log(P^H_\theta(C=1|w))P^H_\mathcal{D}(C=1|w)\mu_\mathcal{V}(w)
\end{aligned}
\tag{4.33}
$$

We then use the Equations 4.26 and their model versions to re-write our objective:

$$
\begin{aligned}
&L^H_{BinClassif}(P^H_\theta(C=1|.)) = \\
&\quad -\sum_{w\in\mathcal{V}} \log\left(\frac{kP_n(w)}{p^H_\theta(w)+kP_n(w)}\right)\frac{kP_n(w)}{P^H_\mathcal{D}(w)+kP_n(w)}\mu_\mathcal{V}(w) \\
&\quad -\sum_{w\in\mathcal{V}} \log\left(\frac{p^H_\theta(w)}{p^H_\theta(w)+kP_n(w)}\right)\frac{P^H_\mathcal{D}(w)}{P^H_\mathcal{D}(w)+kP_n(w)}\mu_\mathcal{V}(w)
\end{aligned}
\tag{4.34}
$$

Here, replacing $\mu_\mathcal{V}$ by its corresponding probability mass function, we obtain:

$$
\begin{aligned}
&L^H_{BinClassif}(P^H_\theta(C=1|.)) \\
&\propto -\sum_{w\in\mathcal{V}} \log\left(\frac{kP_n(w)}{p^H_\theta(w)+kP_n(w)}\right)kP_n(w) - \sum_{w\in\mathcal{V}} \log\left(\frac{p^H_\theta(w)}{p^H_\theta(w)+kP_n(w)}\right)P^H_\mathcal{D}(w) \\
&= -k\mathbb{E}_{\hat{w}\sim P_n}\left[\log\frac{kP_n(\hat{w})}{p^H_\theta(\hat{w})+kP_n(\hat{w})}\right] - \mathbb{E}_{w\sim P^H_\mathcal{D}}\left[\log\frac{p^H_\theta(w)}{p^H_\theta(w)+kP_n(w)}\right]
\end{aligned}
\tag{4.35}
$$

of which the sample version is minus the NCE objective (that we maximize, whereas here we minimize the objective) described in Equation 2.27.

Establishing a link between ϕ^{Ratio} and $\phi^{BinClassif}$

We have just shown that minimizing the Bregman divergence between the functions $P^H_\mathcal{D}(C=1|.)$ and $P^H_\theta(C=1|.)$ that associate to a word a binary probability of classification, is exactly maximizing the NCE objective. This gives us an occurrence where minimizing a Bregman divergence that we derive from $\phi^{BinClassif}$ (the binary negative entropy) between the functions $P^H_\mathcal{D}(C=1|.)$ and $P^H_\theta(C=1|.)$ is equivalent to minimizing a Bregman divergence derived from ϕ^{Ratio} (with $\phi^{Ratio}_{NCE}(r) = r\log(r) - (1+r)\log(1+r)$) between the functions $r^H_\mathcal{D}$ and r^H_θ.

We would like to be able to work with ratios, because they are obtained directly from the output of neural language models. We would also like to be optimizing more efficiently the classification task — which we can do by using a Bregman divergence applied to the classification. To conciliate both, we outline a formal way to derive a function ϕ adapted to working with ratios from a corresponding function $\phi^{BinClassif}$:

this gives us an objective better adapted to classification, while still being usable directly at the output of the neural language model.

Obtaining ϕ^{Ratio} is simply a matter of composing $\phi^{BinClassif}$ with a function σ which allows to obtain the classification probabilities from the ratio (as described in Equation 4.28):

$$\begin{array}{rccc} \sigma: & \mathbb{R}_+ & \rightarrow & [0,1] \\ & r & \rightarrow & \dfrac{r}{1+r} \end{array} \tag{4.36}$$

We immediately notice that, thanks to this composition, the values manipulated during training stay between 0 and 1, which was not the case for all the objectives we experimented in Section 4.2.2. However, to obtain the right transformation, we need to account for the fact that we used different probability mass functions to integrate over $\mathcal{V}$ for different divergences. We then multiply $\phi_{BinClassif}$ by the ratio of the probability mass function used in this section and the one used in Section 4.2.1. This ratio is written as:

$$\frac{P^H_{\mathcal{D}}(w) + kP_n(w)}{kP_n(w)} = 1 + r^H_{\mathcal{D}}(w) \tag{4.37}$$

Then, we get the transformation:

$$\begin{array}{rccc} \phi^{Ratio}: & \mathbb{R}_+ & \rightarrow & \mathbb{R} \\ & r & \rightarrow & (1+r)\phi^{BinClassif}\left(\dfrac{r}{1+r}\right) \end{array} \tag{4.38}$$

Proving that the obtained ϕ^{Ratio} is convex ensures that we are still within our framework. Then, the divergences derived from ϕ^{Ratio} and $\phi^{BinClassif}$ verify:

$$D_{\Phi^{Ratio}}(r^H_{\mathcal{D}}, r^H_\theta) = D_{\Phi^{BinClassif}}(P^H_{\mathcal{D}}(C=1|.), P^H_\theta(C=1|.))$$

We keep both proofs for the Appendix A.

4.3.2 Directly applying β-divergences to binary classification

We have seen previously that using the binary negative entropy as $\phi_{BinClassif}$ yields the NCE objective. If we would like to extend this to the power entropy functions defined in Equation 4.23, we should mimic the derivation used to obtain NCE and use the function:

$$\phi^{BinClassif}_\beta = \phi_\beta(p) + \phi_\beta(1-p)$$

In that case, with $\beta = 1$, we would re-derive the NCE objective. However, we could try to simply use

$$\phi^{BinClassif}_\beta(p) = \phi_\beta(p)$$

We experimented with both ways. While they give almost the same results in the case where $\beta = 1$, only the second one seemed to provide stable learning for the other values of β, which is a result we plan to investigate on. Applying Equation 4.38, we obtain:

$$\begin{array}{rccl} \phi_\beta^{Ratio}: & \mathbb{R}_+ & \rightarrow & \mathbb{R} \\ & r & \rightarrow & (1+r)\phi_\beta\left(\dfrac{r}{1+r}\right) \end{array} \tag{4.39}$$

which gives:

$$\phi_\beta^{Ratio}(r) = \begin{cases} \frac{1}{\beta(\beta-1)}(r^\beta(1+r)^{1-\beta} - r + \beta - 1) & \text{if } \beta \neq 0, 1 \\ r\log\left(\frac{r}{1+r}\right) - 1 & \text{if } \beta = 1 \\ -(1+r)\log\left(\frac{r}{1+r}\right) - 1 & \text{if } \beta = 0 \end{cases} \tag{4.40}$$

and the associated functions $S_{0,1}^{\phi_\beta^{Ratio}}$ are:

$$S_0^{\phi_\beta^{Ratio}}(r) = \begin{cases} \frac{1}{\beta}\left(\frac{r}{1+r}\right)^\beta - \frac{1}{\beta} & \text{if } \beta \neq 0, 1 \\ \frac{r}{1+r} - 1 & \text{if } \beta = 1 \\ \log\left(\frac{r}{1+r}\right) & \text{if } \beta = 0 \end{cases} \tag{4.41}$$

and:

$$S_1^{\phi_\beta^{Ratio}}(r) = \begin{cases} \frac{1}{\beta-1}\left(\frac{r}{1+r}\right)^{\beta-1} - \frac{1}{\beta}\left(\frac{r}{1+r}\right)^\beta - \frac{1}{\beta(\beta-1)} & \text{if } \beta \neq 0, 1 \\ \log\left(\frac{r}{1+r}\right) + \frac{1}{1+r} & \text{if } \beta = 1 \\ -\log\left(\frac{r}{1+r}\right) - \frac{1}{r} & \text{if } \beta = 0 \end{cases} \tag{4.42}$$

We should notice that, as for the function associated to NCE $S_{0,1}^{NCE}$ showed in in Table 4.1, the ratio r only appears inside of the term $r/(1+r)$ for the functions $S_{0,1}^{\phi_\beta^{Ratio}}$ if $\beta \neq 0$. We expect learning procedures to be far more stable and resemble those of NCE.

Experimental results

For this series of experiments, we use a larger range of values for β. We experiment on the PTB again and verify the efficiency of our objectives on the 1 Billion Word Benchmark, using the same setup and hyperparameters as in the previous chapter (see Table 3.5). Again, we use both the training tricks we defined earlier. Results are presented in Table 4.3. Here again, the influence of β on the quality of learning is clear - however, for both corpora, we match and even mildly improve the results obtained previously with NCE.

β	PTB	1BW Benchmark
-1.0	X	X
-0.25	X	X
0.0	X	X
0.25	338.9	X
0.5	191.8	103.4
0.75	167.8	79.7
0.9	157.9	72.9
1.0	150.2	72.6
1.1	149.7	71.4
1.25	144.4	71.8
1.5	149.5	78.1
2.0	177.2	116.9

Table 4.3: Best final perplexities on the test set of the Penn Treebank (PTB) corpus and of the 1 Billion Word (1BW) Benchmark, obtained with Bregman divergences derived from ϕ_β^{Ratio}, for various values of β. 'X' indicates that the model did not reach a perplexity under the size of the vocabulary within the maximal number of training epochs.

For $\beta = 1$, we obtain perplexities that are very close to the equivalent results obtained with NCE at the end of the previous chapter (148.4 for the PTB and 71.3 for the 1BW Benchmark). The values of the β parameter that give the best results can be inferred by looking at previous experiments made with the Beta-divergence: indeed, for other applications, it has been shown (Cichocki et al., 2011; Basu et al., 1998) that the choice of β implies a trade-off between robustness to outliers (for $\beta > 1$) and efficiency (for β close to 1). Nevertheless, these results show that the application of these divergences to language modeling should be explored further; indeed, we believe they can be used judiciously to ease training in the difficult case of training the model with a very large vocabulary.

4.4 Conclusions

We present in this chapter a generalization of the maximum-likelihood estimation objective and of the noise contrastive estimation objective, as minimizations of Bregman divergences. We can apply these divergences to probability distributions of the data and model, but to avoid computing the partition function, we chose to work with the un-normalized distribution outputed by the model. We use a surrogate classification task, which introduces a noise distribution that we sample from: learning an optimal optimizer for this classification task can be done by minimizing the divergence between the ratios of data and noise probabilities and of model and noise probabilities.

We experiment with Bregman divergences derived from simple functions, presented in Pihlaja et al. (2012). However, given the very large range of values taken by the model ratios, these functions overflowed during learning. Using a power-

transformed entropy function, determined by a parameter β, we experiment with *Beta divergences*. Learning is successful only for values of β for which model ratios have an exponent that is in absolute value strictly smaller than 1.

We then show that the NCE objective can be seen as a Bregman divergence applied to functions associating to each word in the vocabulary their binary classification probability. We derive a composition rule, allowing us to re-write Bregman divergences applied to these functions as divergences applied to ratios. This composition rule has the advantage of transforming the ratios into values that are in the inverval $[0, 1]$, ensuring stability during learning. We finally experiment with these new classes of divergences, that we derive from the power-transformed entropy functions. We obtain far more convincing results, that show that the best values of β are a little higher than 1.

Chapter 5

Output Subword-based representations for language modeling

Contents

In the previous chapters of this dissertation, we mainly focused our efforts on methods designed to reduce the training time of large-vocabulary neural language models. However, large computation time is not the only practical issue met when training these models. A different, but not less important issue, affects the distributed representation of infrequent words. While in Section 1.1, we discussed how discrete models need a large number of parameters and are not able to generalize, we encounter a similar problem with continuous models as their vocabulary grow. Indeed, because of the Zipfian shape of the frequency distribution of words in a text, a large part of the vocabulary is composed by infrequent words that only occur a few times in the training data. Since they are optimized on only a few examples, the parametrized representations that are jointly learnt for these infrequent words are not suitable.

On the other hand, the discrete correspondence between a word and its embedding implies that the word structure is overlooked, even though it could provide us precious information on these infrequent words. Besides, a model should be able to use information learnt on frequent words to better represent infrequent words that are similar. Another drawback of learning a different embedding for each word, independently of their structure and frequence, is that it becomes very costly in memory as the vocabulary grows. Moreover, while these issues are already impactful for a morphologically poor language like English, for morphologically-rich languages, like Czech or German, there is a combinatorial explosion of word forms, most of which are hardly observed on training data, which renders the language modelling task even more difficult.

As we discussed previously, rare words can be replaced by a special token *UNK*. This can be seen as using a word class to merge very different words without any distinction. Using different word classes to handle out-of-vocabulary words (OOVs) Allauzen and Gauvain (2005) does not really solve this difficulty, since infrequent words are difficult to classify. A way to deal with this issue is to not stop at the level of word units, but to also use subword units. Whether they are built via a different supervised method with embedded language knowledge, or from the training data, exploiting information from subword units has been attempted many times. It has especially been the case for speech recognition, where they mainly allow to reduce the number of OOV words. While most subword-based models were focused on a specific language, Creutz et al. (2007) is a representative example of a model applied to several morphologically-rich languages. One of the first occurrences of general language models integrating morphological features to represent words is the *factored language model* (Bilmes and Kirchhoff, 2003) and its neural version (Alexandrescu and Kirchhoff, 2006). Input words are represented by their embeddings, plus several other features, including morphemes. To alleviate the impact of OOVs, Mueller and Schuetze (2011) used morphological features for class-based predictions when input words are unknown, obtaining state-of-the-art results on English. While recurrent neural networks have shown excellent performances for character-level language modeling (Sutskever et al., 2011; Hermans and Schrauwen, 2013), the results of such models are usually worse than those that use word-level prediction, since they have to consider a far longer history of tokens to be able to predict the next one correctly. While more recent work (Hwang and Sung, 2017) seems to obtain very satisfactory results with a supplementary word-level layer that allows a better processing of the

longer history, we will in our work always use word-level prediction.

While replacing the input word embeddings with this kind of representations in neural language models has been experimented with exhaustively (Kim et al., 2015; Vania and Lopez, 2017; Verwimp et al., 2017), the predicted words are only scored based on their word embeddings. As previously explained, if the vocabulary contains infrequent words, these representations may be barely learnt. Since we aim at improving infrequent word representations, we will, in this chapter, experiment with augmenting or replacing output word embeddings with representations built from subwords. To the best of our knowledge, such an idea has only been experimented by Józefowicz et al. (2016), which evaluates the use of convolutional and LSTM layers to build word representations for output words. They allow the model to trade size against perplexity, since their model performs worse than the classic softmax approach, but with far less parameters.

We will, in Section 5.1, begin with a presentation of the various architectures we will use to obtain these word representations from subwords. We then propose to study the training of a language model which augments or completely replaces output words representations with character-based representations. In Section 5.2, we will apply our model on a Czech dataset, and present a series of experiments comparing the effect of different architectures, as well as the effect of different sizes of output look-up tables. Finally, we use the results obtained in Chapter 3 to try to improve our models, and extend our experiments to English, with the PennTreebank, for comparaison.

5.1 Representing words

Usually, input and output word embeddings are parameters, stored in look-up matrices $\mathbf{W}$ and $\mathbf{W}_{out}$. As we explained in Section 1.2, the word embedding $\mathbf{r}_w^{word}$ of a word w is simply the column of $\mathbf{W}$ corresponding to its index in the vocabulary $\mathcal{V}$: $\mathbf{r}_w^{word} = [\mathbf{W}]_w$. We present in this section different ways to build embeddings $\mathbf{r}_w^{subword}$ as a function of a decomposition of the word w into subwords.

Quite a lot of work have been recently done with models representing words as function of subwords units. Some are using morphological features, with a recursive structure (Luong et al., 2013), or an additive one (Botha and Blunsom, 2014). However, language models that extract features directly from the character sequence are gaining popularity, whether they use character n-grams (Sperr et al., 2013), or characters composed by a convolutional layer (Santos and Zadrozny, 2014;

Representation	Decomposition
Word	počátku
Characters	p+o+č+á+t+k+u
Character 3-grams	poč+očá+čát+átk+tku
Lemma + Tags	počátek+N+MascIn+Sg+Loc+Act

Table 5.1: Example of subword decompositions used for a Czech word.

Kim et al., 2015) or a Bi-LSTM layer (Ling et al., 2015), as they avoid using an external morphological analyser. These types of models have also been applied with success to several other task, including learning word representations (Qiu et al., 2014; Cotterell et al., 2016; Bojanowski et al., 2016; Wieting et al., 2016), POS tagging[1] (Plank et al., 2016; Ma and Hovy, 2016; Heigold et al., 2017), Named entity recognition (Gillick et al., 2016), Parsing (Ballesteros et al., 2015) and Machine translation (Costa-jussà and Fonollosa, 2016). Recently, an exhaustive summary of previous work on building word representations by composing subword units was presented in Vania and Lopez (2017). This work also compares the different types of subword unit, how they are composed, and their impact on various morphological typologies. Among those, we consider two types of representations: decomposing words into characters (or n-grams of characters), and decomposing them into a Lemma and positional tags using a morphological analysis. An example of these different decompositions is shown in Table 5.1.

5.1.1 Decomposition into characters

A word w is a character sequence $\{c_1, .., c_{|w|}\}$, where the characters belong to a character vocabulary $\mathcal{V}_c$. Let $\mathbf{C} \in \mathbb{R}^{d_c \times |\mathcal{V}_c|}$ be the character embedding matrix. Then, the characters are represented by embeddings of dimension d_c, $\{\mathbf{r}^{char}_{c_1}, .., \mathbf{r}^{char}_{c_{|w|}}\}$, where $\mathbf{r}^{char}_{c_i} = [\mathbf{C}]_{c_i}$ denotes the vector associated to the character c_i. We can note that the character vocabulary is typically far smaller than a word vocabulary. To infer a word embedding from its character embeddings, we use two different architectures.

Using a convolution layer

First, a *convolution layer* (Waibel et al., 1990; Collobert et al., 2011), similar to layers used in Santos and Zadrozny (2014); Kim et al. (2015), which applies a convolution matrix $\mathbf{W}^{CNN}_{n_c} \in \mathbb{R}^{d_c n_c \times d_r}$ over a sliding window of n_c characters, producing local features:

$$\mathbf{x}^n_{n_c} = \mathbf{W}^{CNN}_{n_c}(\mathbf{r}^{char}_{c_{n-n_c+1}} : .. : \mathbf{r}^{char}_{c_n})^T + \mathbf{b}^{CNN}_{n_c} \tag{5.1}$$

We add $n_c - 1$ *padding* vectors on each side of the word, and $\mathbf{x}^n_{n_c}$ is a vector obtained for each of the $n + n_c - 1$ possible character n_c-gram. Each coordinate $[\mathbf{x}^n_{n_c}]_i$ of this vector depends on the corresponding column $[\mathbf{W}^{CNN}_{n_c}]_i$ of the convolution matrix. We can see each of these columns as different convolution *filters*: using d_r of these filters results in a representation of dimension d_r. The embedding of w is then obtained by applying a max-pooling function, followed by an activation function ϕ, over each of these coordinates:

$$[\mathbf{r}^{n_c}_w]_i = \phi\left(\max_{n=1}^{|w|-n_c+1} [\mathbf{x}^n_{n_c}]_i\right) \tag{5.2}$$

We can use filters of different sizes $n_{c_1}, \ldots, n_{c_f}$ and concatenate their results:

[1] Which is a task we also addressed with some of these architectures: see Annex D

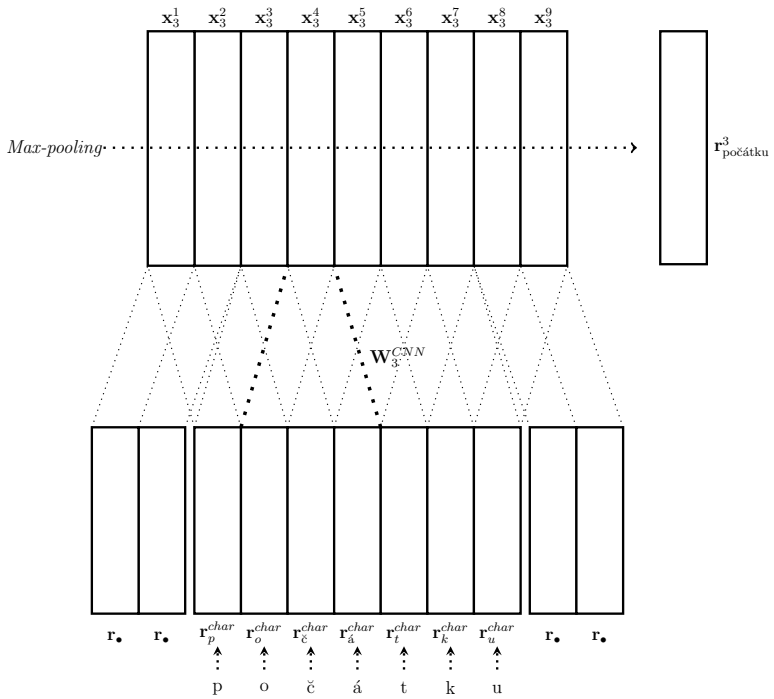

Figure 5.1: Example architecture of a convolutional layer CharCNN for building character-based representations of words. Here, the convolution matrix is applied on a window of 3 characters. The • symbol indicates the specific character token used for padding.

$$\mathbf{r}_w^{CharCNN} = (\mathbf{r}_w^{n_{c_1}} : \ldots : \mathbf{r}_w^{n_{c_f}}) \tag{5.3}$$

An exemple of such a layer with filters of size 3 is shown in Figure 5.1. We note this layer CharCNN (for *Character Convolutional Neural Network*).

Using a recurrent layer

We use a *bi-LSTM* (Hochreiter and Schmidhuber, 1997; Graves et al., 2005), on characters, similarly to Ling et al. (2015). It combines the final states $\overrightarrow{\mathbf{h}_{|w|}}$ and $\overleftarrow{\mathbf{h}_1}$ of two LSTMs, respectively over the character sequence and the reverse character sequence, which are computed as such:

$$\overrightarrow{\mathbf{h}_i} = LSTM(\mathbf{r}_{c_i}^{char}, \overrightarrow{\mathbf{h}_{i-1}})$$

$$\overleftarrow{\mathbf{h}_j} = LSTM(\mathbf{r}_{c_j}^{char}, \overleftarrow{\mathbf{h}_{j+1}})$$

We then concatenates them to obtain the representation:

$$\mathbf{r}_w^{CharBiLSTM} = \overrightarrow{\mathbf{h}_{|w|}} : \overleftarrow{\mathbf{h}_1} \tag{5.4}$$

An example of such a layer is shown in Figure 5.2. We note it CharBiLSTM.

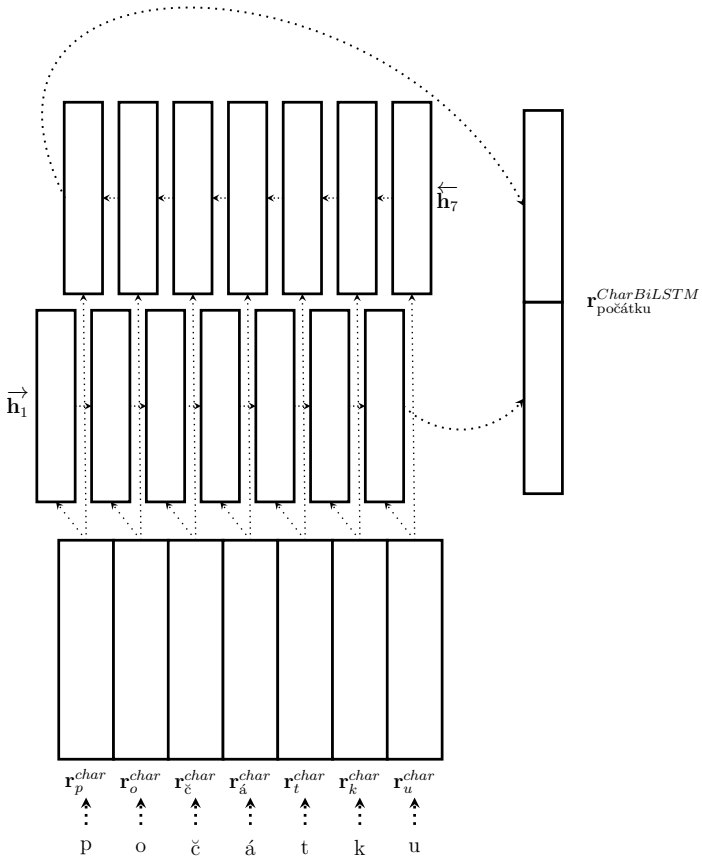

Figure 5.2: Example architecture of a bi-recurrent layer CharBiLSTM for building character-based representations of words.

5.1.2 Decomposing morphologically

For morphologically-rich languages, the different morphological properties of a word (gender, case, ...) are usually encoded using multiple tags, as shown in Table 5.1. Therefore, a word w is decomposed into a lemma l along with a set of associated sub-tags $T = \{t_1, .., t_{|T|}\}$ of fixed size $|T|$. For a given word, a single tag can be created by the concatenation of the subtags. However, this implies a large tagset and mitigates the generalization power since some sub-tags combinations can remain unobserved on training data. In this work, we prefer a factored representation where each sub-tag is considered independently.

Lemmas, similarly to surface forms, are represented by $|\mathcal{V}_l|$ vectors stored in a look-up matrix $\mathbf{l} \in \mathbb{R}^{d_l \times |\mathcal{V}_l|}$, and $\mathbf{r}^{lemma}_w = [\mathbf{L}]_w$. Each sub-tag has its own vocabulary and its own look-up matrix. However, the additional cost is negligible given their small size (which we will see with Czech in Table 5.3b). To infer a word embedding from a sub-tags set, we also use two methods. The first method consists in concatenating their embeddings:

$$\mathbf{r}^{TagConcat}_T = \mathbf{r}^{tag_1}_{t_1} : \ldots : \mathbf{r}^{tag_i}_{t_i} : \ldots : \mathbf{r}^{tag_{|T|}}_{t_{|T|}} \tag{5.5}$$

The second method uses a bidirectionnal LSTM on the sequence of tags T, using

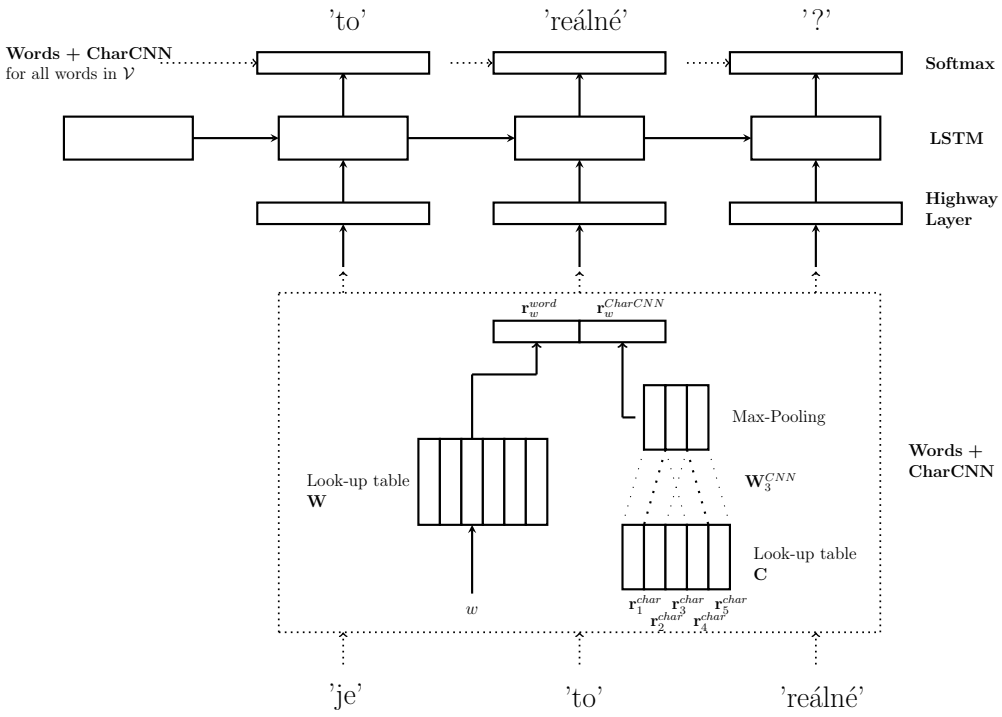

Figure 5.3: Example architecture of our language model, when using word embeddings and a character CNN to build both input and output word representations.

exactly the same structure as in Section 5.1.1:

$$\mathbf{r}_T^{TagBiLSTM} = \overrightarrow{\mathbf{h}_{|T|}} : \overleftarrow{\mathbf{h}_1} \tag{5.6}$$

We note these layers TagsConcat and TagsBiLSTM.

In several works, an intermediate layer is used between the obtained subword-based representations and the hidden layer. It can be seen as a way to blend together representations coming from different sources (whether it is word embeddings, the representations resulting from differently sized convolution filters, or from the two directions of a BiLSTM), instead of simply concatenating them. Kim et al. (2015) uses a highway layer (Srivastava et al., 2015) to do so, which we use here too. Finally, Verwimp et al. (2017) implements the simpler solution of concatenating character embeddings, which we believe to be impractical to use for output representations, since it is in practice difficult to apply a highway layer to blend those character embeddings.

5.2 Application to language modeling

As previously, the experiment are conducted using a recurrent language model, with LSTMs (Section 1.2.2). Integrating subword-based output representations in a language model is straigthforward: as illustrated in Figure 5.3, we concatenate word, character-based or lemma and tags embeddings, to obtain input and output word representations. Table 5.2 summarizes the various representations we used in our experiments, for both input and output words. However, since the traditional

Input representation	$\mathbf{r}_w$	Eq
Words	$\mathbf{r}_w^{word}$	
CharCNN	$Hw(\mathbf{r}_w^{CharCNN})$	5.2
CharBiLSTM	$Hw(\mathbf{r}_w^{CharBiLSTM})$	5.4
Words + CharCNN	$Hw(\mathbf{r}_w^{word} : \mathbf{r}_w^{CharCNN})$	
Words + CharBiLSTM	$Hw(\mathbf{r}_w^{word} : \mathbf{r}_w^{CharBiLSTM})$	
Lemma + TagsConcat.	$Hw(\mathbf{r}_l^{lemma} : \mathbf{r}_T^{TagConcat})$	5.5
Lemma + TagsBiLSTM	$Hw(\mathbf{r}_l^{lemma} : \mathbf{r}_T^{TagBiLSTM})$	5.6
Output representation	$\mathbf{r}_w^{out}$	
Words	$\mathbf{r}_w^{word}$	
CharCNN	$\mathbf{r}_w^{CharCNN}$	
CharBiLSTM	$\mathbf{r}_w^{CharBiLSTM}$	
Words + CharCNN	$\mathbf{r}_w^{word} : \mathbf{r}_w^{CharCNN}$	
Words + CharBiLSTM	$\mathbf{r}_w^{word} : \mathbf{r}_w^{CharBiLSTM}$	

Table 5.2: Detail of the various input and output representations used in our experiments. Hw indicates the use of a Highway layer.

output layer of a neural language model includes a softmax activation function, we face a significant issue: not only do we need to sum scores over the full vocabulary, we also need to compute representations using a specific subword-based layer for every word in the vocabulary, in order to compute those scores for each example. This is, of course, very costly.

Training a language model with subword-based output representations

To avoid such a heavy training cost, we train our language model with a sampling-based method, which we presented in Chapter 2. To begin, we choose importance sampling, which is used by the authors of Józefowicz et al. (2016) to train the model they built with character-based output representations. This allows us to only compute the subword-based representations of target words, plus the words drawn as noise samples during training. However, we still need to compute these representations for all words when doing inference.

We experimented with various ways to link the subword-based layers to the model and between themselves. We first apply a highway layer to the output representation, but it seems almost always counter-productive, rendering training more unstable. In all experiments we present, weights are not tied between input and output representations, since our preliminary experiments with tied weights always gave worst results. Besides, we do not mix structures for character-level representations (for example, using an input CharCNN and output CharLSTM) since our first experiments gave systematically worse results than using the same structures). When using different types of representations, we also kept consistency between vocabularies: if both lemmas and words are used in a model, any lemma considered unknown has its corresponding word unknown, and inversely. The same (or corresponding) vocabularies are used for all inputs representations, all outputs rep-

resentation, and evaluation. The only exception is presented in Section 5.3.3. When using a character-based output representation, during evaluation, the representation of the *unknown* token is built from a specific unknown character token, a specific unknown lemma token, and specific unknown tags tokens that are parameters of the model.

The following experiments aim at comparing potential uses of those different subword-based word representations, and thus are not directed towards performance. For this reason, we use the same implementation for all experiments and do not specifically try to optimize the general model structure or the dimensional hyper-parameters. We also use representations of the same size, indifferently of how we build them, to verify the potential gain they bring. A more complete comparaison of the systems we experiment with should take in account both the total dimension of the layers and the total number of parameters they use.

To fully exploit the potential of subword-based representations, we choose to apply our model to a morphologically complex language, Czech. We then apply our model to English with the Penn Treebank (PTB), in order to verify the impact of subword-based representations on this morphologically simpler language.

Czech data and setup

We used data from the parallel corpus *News-commentary* 2015, from the WMT News MT Task. The data consists in 210K word sequences, amounting in about 4,7M tokens. We divide the data into a training, development and testing sets. The development and testing sets are comprised of 10K word sequences and 150K tokens each. In our experiments, we use different vocabulary sizes, which we obtain by varying a frequency threshold f_{Th}: words are selected when their frequency in the training data is strictly higher than the threshold. Table 5.3a shows the correspondences between vocabulary sizes and thresholds. Since this data comes from a parallel corpus, we can compare the distributions of word frequencies in English and Czech. These distributions are shown in Figure 5.4. The main difference is that the tail of the Czech distribution is far longer, which means it contains far more infrequent words than the English distribution. It also implies that the Czech vocabulary is far larger, being more than twice the size of the English vocabulary, which justifies our approach.

f_{Th}	$\lvert\mathcal{V}_{Th}\rvert$
0 (All words)	159142
1	89587
2	66743
5	37010
10	25295

(a) Vocabulary sizes for different frequency thresholds

$\lvert\mathcal{V}_C\rvert$	$\lvert\mathcal{V}_L\rvert$	$\lvert\mathcal{V}_{tag_i}\rvert_{i=1..\lvert T\rvert}$
155	61364	[12, 65, 11, 6, 9, 6, 3, 5, 5, 4, 3, 3]

(b) Vocabulary sizes for subword units

Table 5.3: Various vocabulary sizes for Czech on *News-commentary* 2015.

The lemma and tags decomposition presented in Section 5.1.2 was obtained with Morphodita (Straková et al., 2014). There are 12 tag categories for Czech. Vocabulary sizes for characters, lemmas and tags are detailed in Table 5.3b. As previously mentioned, the vocabulary sizes for characters and tags are far smaller than for words and lemmas, which makes these representations far less costly in term of memory usage.

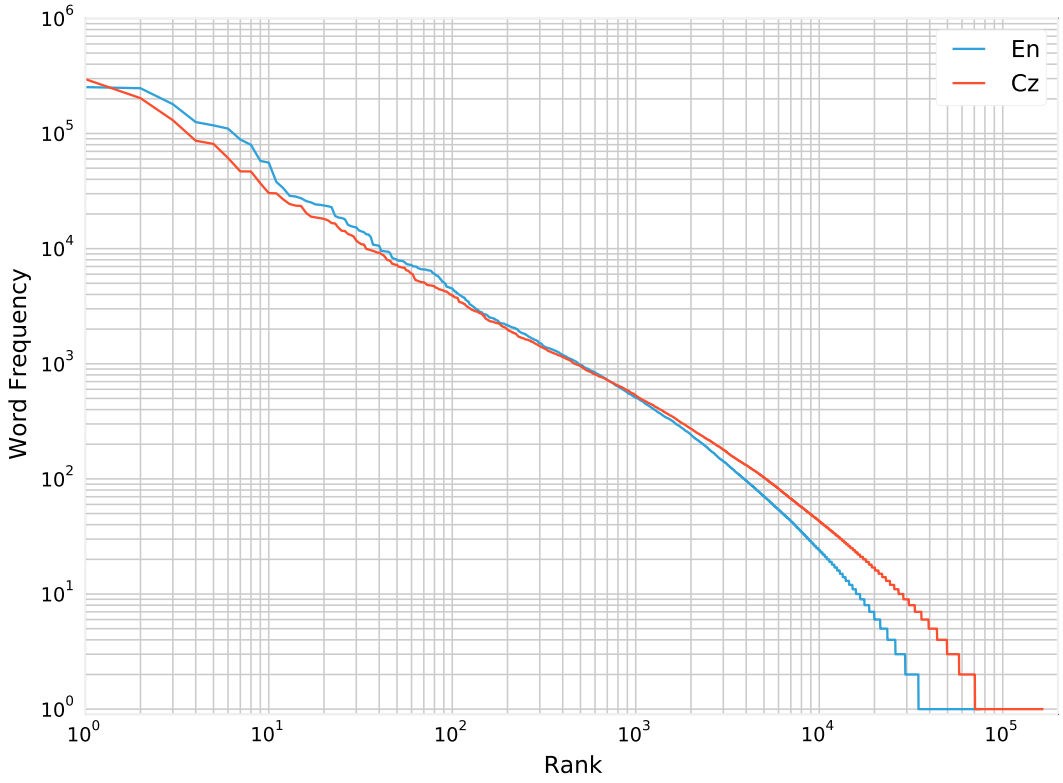

Figure 5.4: Distribution of word frequencies, ordered by rank, for the English and Czech versions of the parallel corpus *News-commentary* 2015.

The hyperparameters we use are shown in Table 5.4. We choose our subword and word embedding dimensions in order to obtain, for each type of representation, an embedding dimension of 150. In the case of the CharCNN, we use filters of 3, 5 and 7 characters, respectively of dimension 30, 50, and 70. In the case of the CharBiLSTM, each LSTM has a dimension of 75. Tags embeddings also add up to a size of 150.

Hyperparameter	Value
Number of noise samples k	500
Default noise distribution P_n	Unigram
Hidden Layer	LSTM
Maximum Sentence Length	30
Number of hidden layers	2
Input representation dimension	$d_{\mathbf{r}_w}$
Hidden Layer dimension	$d_{\mathbf{r}_w^{out}}$

Hyperparameter	Value
Batch size (Sentences)	128
Dropout rate	0.5
Optimization method	Adam
Learning rate	0.001
ϵ	$1e^8$
Grad clipping value	0.0
Maximum number of training epochs	5

Table 5.4: Structural choices and hyperparameters used on the *News-commentary* 2015 Czech corpus experiments presented in this chapter.

Finally, the input layer dimension depends on how many input representations we use, while the hidden layer size depends on the number of output representations. Given the large size of the vocabulary, we used $k = 500$ noise samples by batch. As before, we backtrack learning when no progress has been made on the validation set perplexity, stopping training after 10 consecutive backtrackings.

Preliminary experiments on input representations and vocabulary sizes

First, we train 'classical' models, where we only change the type of input representation we use, and we keep the classic output representation based only on words. Since we want to evaluate the difficulty of representing infrequent words, we begin by training models with various vocabulary sizes (which are detailed in Table 5.3a). For this set of experiments, we choose models using word based input representations, character based input representations, a concatenation of both, and a concatenation of lemma and tag based input representations. To observe variability in the final perplexity results, we present the average of the results obtained on 5 models, and the standard deviation. These values are shown in Table 5.5. Note that the baseline result for the full vocabulary is a final mean perplexity of 388.1.

The larger the vocabulary is, the less stable the training is. The standard deviation especially grows when we go from excluding words that only appear twice from the vocabulary ($f^W_{Th} = 2$), to training with the full vocabulary ($f^W_{Th} = 0$). That tendency is stronger with models using only subword based representations (CharCNN). Some of that instability can be pinned on the fact that we do not have any unkown words in the training data, which makes the associated embedding useless at evaluation. However, we suppose it mostly comes from the fact that a large number of words only appear once in the training data, and are not adequatly learned. This is probably accentuated by the fact that our training method almost never samples these words: their representation may only be updated once by epoch.

Despite these variations, it is clear that using both word-based and character-based input representations (Words + CharCNN) is better than our baseline of only word embeddings, which still gives far better results than using only character-based input representation. The efficiency of decomposing the words as a lemma and a

f^W_{Th} Input Representation	10	5	2	1	0
Words	139.6 ± 1.3	168.9 ± 2.7	231.5 ± 5.2	266.6 ± 4.1	388.1 ± 6.9
CharCNN	156.7 ± 3.1	187.8 ± 2.8	256.3 ± 3.5	281.9 ± 5.5	427.2 ± 14.1
Words + CharCNN	**130.2 ± 1.5**	**161.3 ± 2.5**	**218.9 ± 2.3**	**251.9 ± 4.2**	378.2 ± 6.2
Lemmas + TagsConcat	142.2 ± 3.0	176.8 ± 1.7	233.6 ± 2.4	265.3 ± 4.8	**371.4 ± 6.4**

Table 5.5: Average test perplexities obtained when training 5 models using word-based output representations with IS, for various input representations and vocabulary sizes. Results in bold are the best models for a given vocabulary size. The underlined results are the baselines, which use only words for the input and output representations.

set of tags seems to depend on the size of the vocabulary: while it is comparatively not efficient for smaller vocabularies, it works well when we include all words, even giving the best perplexity results. It could indicate that this decomposition allows better generalization to infrequent words.

5.3 Experiments on Czech with subword-based output representations

In this section, we experiment with augmenting and replacing output representations with character-based embeddings. We also investigated working with tags, but they were difficult to use in practice. The main issue is that tags can be ambiguous across several occurrences of the same words, which is something we need to normalize in order to predict words. To do so, we tried using specific tokens, or choosing the most frequent tags, but in any configuration, using tags for output representations made the model severly overfit.

5.3.1 Influence of the vocabulary size

First, we would like to study the potential usefulness of subword-based output representations for various vocabulary sizes. Using the representations that gave, as a whole, the best results during the preliminary experiments (Words+CharCNN), we train models with the same vocabulary sizes than before, detailed in Table 5.3a. The mean final perplexity and standard deviation across 5 trainings are shown in Table 5.6, while the perplexity curves of the best of these 5 models are shown in Figure 5.5. Clearly, our model fails to improve upon the conventional word model when the output vocabulary size is relatively small. More precisely, models that use a Words+CharCNN output layer seem unable to learn after a few epochs. We link this behavior to the difficulty met by the authors in Józefowicz et al. (2016): since most logits are tied when we use an output character-based representation - as opposed to independently learned word embeddings, the function mapping from word to word representation is smoother and training becomes more difficult. They used a smaller learning rate and a low dimensional correction factor, learned for each word, as a work-around.

However, increasing the vocabulary size reduces this effect. This is especially clear with the full training vocabulary: in this setup, using a character-based rep-

f^W_{Th} Output Representations	10	5	2	1	0
Words	**130.2 ± 1.5**	**161.3 ± 2.5**	**218.9 ± 2.3**	**251.9 ± 4.2**	378.2 ± 6.2
Words + CharCNN	635.6 ± 71.6	555.1 ± 37.3	422.6 ± 50.6	370.0 ± 20.8	**327.2 ± 8.1**

Table 5.6: Average test perplexities obtained when training 5 models using word and CharCNN input and output representations with IS, for various vocabulary sizes. Results in bold are the best models for a given vocabulary size.

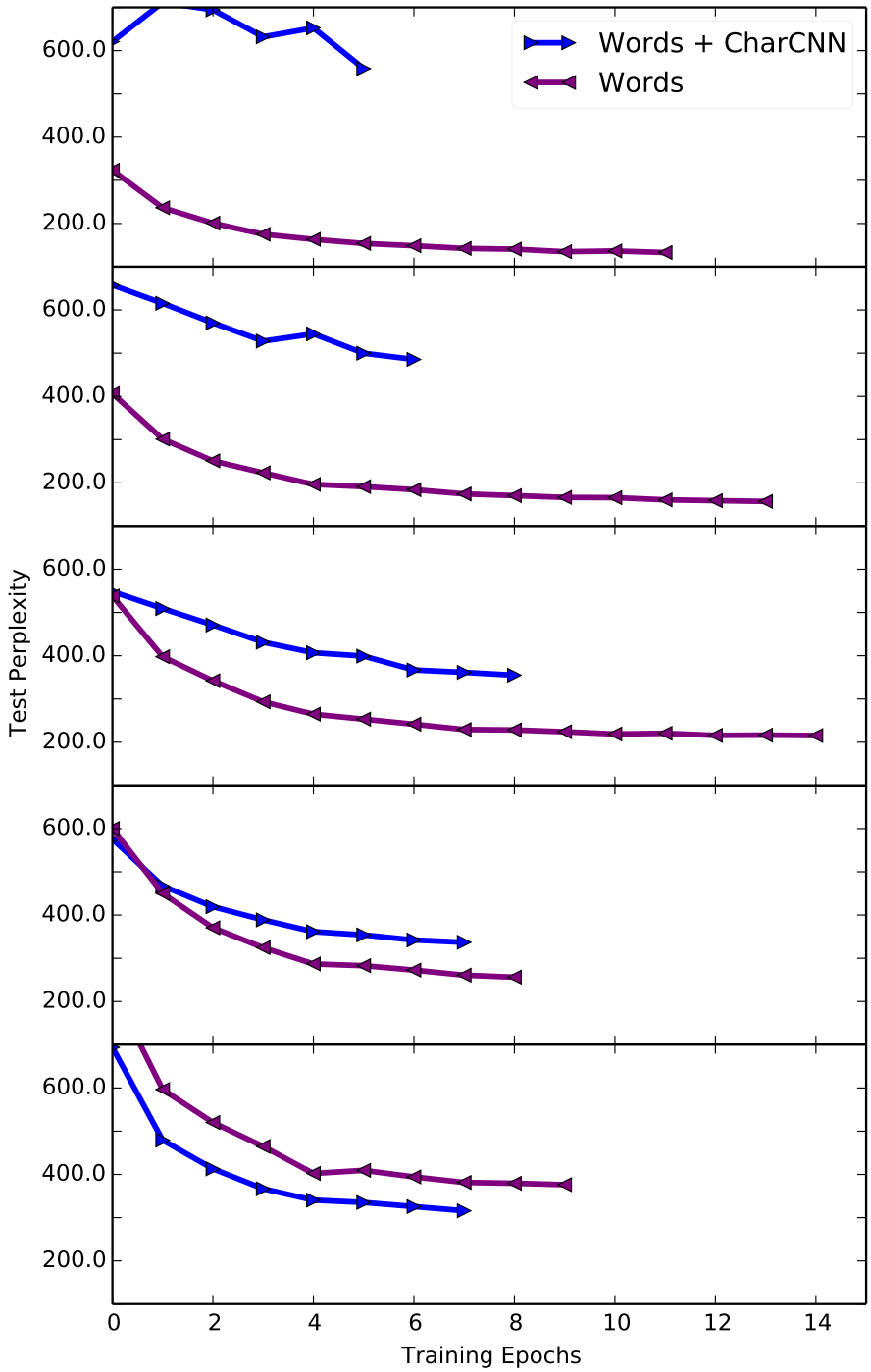

Figure 5.5: Testing perplexity curves for models trained with Words+CharCNN input representations, with IS, for various vocabulary sizes. Vocabulary sizes are given, from top to bottom, by taking $f_{Th}^{W} = 10, 5, 2, 1, 0$ in Table 5.3a

resentation improves the performance of the model. We can assume that character based representations compensate for the insufficient number of updates performed on rare words embeddings during training. Indeed, combining word and character-based embeddings allows the model to better counteract the Zipfian shape of the word occurences distribution.

Output Representation / Input Representation		Words	Words + Char		Char	
			CNN	BiLSTM	CNN	BiLSTM
Words		388.1 ± 6.9	333.2 ± 7.0	352.6 ± 3.0	917.5 ± 35.5	1943.6 ± 82.4
Char	CNN	427.2 ± 14.1	363.2 ± 7.6	-	**783.2 ± 32.8**	-
	BiLSTM	630.0 ± 15.1	-	576.4 ± 19.0	-	1983.3 ± 37.0
Words + Char	CNN	378.2 ± 6.2	**327.2 ± 8.1**	-	827.2 ± 37.1	-
	BiLSTM	376.0 ± 4.2	-	349.6 ± 11.6	-	1889.9 ± 63.3
Lemmas + Tags	Concat.	371.4 ± 6.4	**326.3 ± 6.4**	338.0 ± 5.9	828.9 ± 25.7	1990.9 ± 194.2
	BiLSTM	**368.1 ± 3.0**	335.4 ± 9.2	**332.8 ± 7.4**	821.3 ± 29.2	**1856.1 ± 57.6**

Table 5.7: Average test perplexities obtained when training 5 models with IS, for various input/output representations. Results in bold are the best models for a given output representation. The underlined result is the baseline, obtained using only words for the input and output representations.

5.3.2 Effects of the representation choice

We now experiment with various representation choices, in order to assess which combination of input and output representations gives the best performance. However, we limit these experiments to models trained with the full vocabulary, having established that, in our current configuration, augmenting output representations with subwords renders models difficult to train for smaller vocabulary sizes. Table 5.7 gathers the main experimental results.

For any input representation, augmenting the output representation with a character-based embedding, by using a Words+CharCNN output layer, improves the performance of the model. It is especially true for convolutional layers. We also can notice that the improvement is more important for models that performed badly with basic output word embeddings. When using only words as output representations, the best input representations seems to be the Lemmas+Tags input layers (with both architectures), but the Words+CharCNN/BiLSTM input layers are both close. With a Words+CharCNN output layer, all input representations give close results, except for the CharCNN layer. Among them is the best improvement on the baseline, which is a final mean perplexity of 326.3. Overall, biLSTMs perform worse than their convolution/concatenation counterparts. The mean and standard deviations are especially high when characters are used as input representation, making the decomposition into lemma and tags the best choice when using Words + CharbiLSTM output layer.

Our experiments with only character-based embeddings as output representations give results far worse than those using also word-based embeddings. Training is also far more unstable. The observations we previously made are however still valid, with the exception of a model using CharCNN layers for both input and output, which behaves better than other models using the same output representation. We believe these seemingly bad results are linked to the difficulties mentioned in Józefowicz et al. (2016) and in Section 5.3.1.

Input Representation	f^W_{Th}	Words + CharCNN All words	Frequent words	Rare words
Words	0	333.2 ± 7.0	214.8 ± 4.7	5560.6 ± 174.0
	1	317.9 ± 5.0	205.3 ± 2.7	**5077.3 ± 273.2**
	2	303.9 ± 4.9	195.7 ± 2.9	**5071.6 ± 322.7**
	5	**291.3 ± 4.0**	**188.1 ± 2.2**	**5049.4 ± 397.5**
	10	306.5 ± 10.2	191.2 ± 6.6	7999.2 ± 459.1
CharCNN	0	363.2 ± 7.6	234.2 ± 3.5	5588.5 ± 552.6
	1	356.9 ± 6.1	230.9 ± 2.9	**5432.1 ± 575.2**
	2	**350.9 ± 10.7**	**227.8 ± 7.2**	**5178.9 ± 451.0**
	5	**355.3 ± 9.3**	**228.1 ± 5.3**	6039.7 ± 765.3
	10	364.8 ± 13.7	227.2 ± 9.1	8582.0 ± 279.7
Words + CharCNN	0	327.2 ± 8.1	208.6 ± 4.7	5790.2 ± 631.9
	1	291.0 ± 11.6	189.0 ± 7.0	**4577.1 ± 446.0**
	2	283.1 ± 8.1	184.3 ± 4.7	**4354.4 ± 435.2**
	5	**268.2 ± 3.5**	**172.9 ± 2.2**	**4641.6 ± 328.6**
	10	278.4 ± 8.1	174.4 ± 4.4	6725.1 ± 587.0
Lemma + TagsConcat.	0	326.3 ± 6.4	214.2 ± 5.1	5782.4 ± 538.7
	1	312.8 ± 13.8	202.4 ± 8.4	**4747.7 ± 312.8**
	2	318.6 ± 11.3	204.8 ± 6.6	5429.0 ± 526.0
	5	**303.7 ± 6.7**	193.6 ± 2.3	6035.1 ± 718.3
	10	308.3 ± 3.2	**192.0 ± 1.4**	7378.0 ± 616.3
Lemma + TagsBiLSTM	0	335.4 ± 9.2	208.0 ± 4.0	5733.2 ± 573.2
	1	319.0 ± 10.6	206.0 ± 6.9	5099.1 ± 301.4
	2	314.3 ± 11.9	203.8 ± 8.0	**4782.3 ± 221.9**
	5	**310.3 ± 14.1**	197.7 ± 7.8	5864.7 ± 526.5
	10	**307.8 ± 5.8**	**192.6 ± 3.4**	7063.3 ± 542.8

Table 5.8: Test perplexity averaged on 5 models trained with IS, for various input representations and output word look-up table sizes. Corresponding vocabulary sizes are given in Table 5.3a. Test perplexities are given for all words, frequent words (frequency > 10) and rare words (frequency < 10). In bold are the best models for a given input representation.

5.3.3 Influence of the word embeddings vocabulary size

Following our observations in Section 5.3.1, we assess the effect of reducing the word look-up table size for the Words+CharCNN output representation, which precedently gave the best results. We do not change the size of the event space: when constructing output representations for words under a chosen frequency, we simply do not learn a specific word representation. For example, using a threshold of $f^W_{Th} = 10$ means that words that appear less thant ten times won't have their own word embedding, and are represented by the unknown word token combined with their character-based representation. Results are shown in Table B.2.

We can see that for all input representations, using a specific unknown token in place of a specific word embedding for words appearing less than 5 times in

training data gives the best performance, or close to, given the standard deviation among the threshold tested. Reducing the look-up table to words only appearing more than 10 times gives globally worse results, even if they are still better than if we keep the full table. Still, with the Lemmas + Tags input layer, both these values seem to give an equivalent result. However, when looking at the perplexities computed on rare words only (which are very hard to interpret given their very high standard deviation) it seems that reducing the lookup table too much damages the model performance. Since these infrequent words contribute so little to the global perplexity, this is mostly unnoticeable on the final results.

Overall, we improve again our best results, obtaining 268.2 as final mean perplexity, from a baseline of 388.1. While in our experiments, the subword-based output representations made training on smaller vocabularies stop early, they improved the full-vocabulary model performances. In this same setup, replacing independent word embeddings by the unknown token for rare words yields further improvement. It is worth noticing that this also opens the vocabulary, since our model can be used to rescore unknown words.

5.4 Supplementary results and conclusions

In this section, we verify our previous results by training the same models with NCE while applying the two training tricks developed in Chapter 3. We first update the results we obtained with importance sampling in Czech, then apply our best models to the PTB dataset.

5.4.1 Training with improved NCE on Czech

The main change we obtained when repeating the previous section's experiments using the improved NCE training method concerns using subword-based representations with smaller vocabularies. While, as described in Section 5.3.1, models trained with importance sampling stopped after a few epochs, with NCE, training unfolds normally, giving better results with an output Words + CharCNN layer than only word-based representations for all vocabulary sizes. These results are shown in Table 5.9. Supplementary experiments showed that the determining factor in this new result was to initalize bias weights with the unigram frequencies (in Section 3.3.2); understanding why requires further investigation.

f^W_{Th} Output Representations	10	5	2	1	0
Words	136.6 ± 6.5	174.4 ± 9.3	223.3 ± 9.6	248.6 ± 8.1	389.7 ± 19.1
Words + CharCNN	**124.2 ± 7.5**	**150.5 ± 7.4**	**199.8 ± 8.1**	**221.6 ± 5.5**	**320.6 ± 4.4**

Table 5.9: Average test perplexities obtained when training 5 models using word and CharCNN input and output representations with NCE, for various vocabulary sizes.

Output Representation Input Representation	Words	Words + CharCNN $\mathcal{V}_{Th}^{W} = 0$	 $\mathcal{V}_{Th}^{W} = 10K$
Words	164.2 ± 1.7	158.8 ± 1.6	135.4 ± 2.1
CharCNN	**161.2 ± 1.1**	**141.7 ± 2.8**	138.1 ± 2.8
Words + CharCNN	**160.9 ± 6.1**	147.8 ± 1.3	**130.8 ± 4.4**

Table 5.10: Average test perplexities obtained when training 5 models with NCE, for various input representations and output word look-up table sizes, on PTB.

Other detailed results on representation and the word embeddings vocabulary size are omitted here, but can be found in the Appendix B. They mostly confirm the two main tendancies outlined at the end of the previous section: augmenting word embeddings with subword-based output representations yields sizable improvement, while replacing independent word embeddings by the unknown token for rare words improves perplexity even more. However, some notable changes are, first, that the standard deviation is globally quite higher with NCE, especially for models using BiLSTMs. Secondly, that while the mean perplexity is overall a little higher, the models using CharBiLSTM only output representations obtain perplexities far better than with importance sampling (but are still the worst output representation).

5.4.2 Comparative experiments on English

In order to verify the usefulness of our setup on a morphologically simpler language, we now apply it to the PTB. When working on English, we only use character-based representations, since the morphology does not carry as much information in English as it does in Czech. We limit our experiments to the better-performing CharCNN layers. Each input and output representations are of the same dimension, 150, and we use 2 highway layers after the input as in the rest of this chapter, while every other hyperparameter is the same than those indicated in Table 3.2. We should note that if we want to compare the obtained results to those obtained in previous chapters, we should only do so for models that use both Words + CharCNN input and output layers. Indeed, other models use smaller input and/or output dimensions and are therefore at a disadvantage. For a more detailled experimental investigation of the effect of dimension sizes, please refer to Annex C.

We can see that, while augmenting the word input representation with a CharCNN layer is not that impactful, augmenting the word ouptut representation with the same layer yields a nice improvement - and restricting the word embedding table size to a smaller value (here, we arbitrarily chose $10K$ words) also significantly improves perplexity. If we wish to compare our results to the perplexity obtained with the same objective at the end of Chapter 3 (which was 148.4), we can see that models using both Words + CharCNN layers reach a similar perplexity. However, we obtain this results by using a little more than half the parameters, given the small amount of parameters needed by the CharCNN layer. Besides, restricting the word embedding table size to $10K$ words both reduces greatly the number of parameters and improves on the final perplexity, to reach a mean value of 130.8.

5.5 Conclusions

In this chapter, we show how augmenting the output word embeddings of neural language models with subword-based representations can improve performance, especially on very large vocabularies. We first present several ways to decompose words into subwords and the neural architectures we use to build representations from those subwords. We describe the various layers that we use to generate input or output representations, notably the Words + CharCNN layer, that concatenates a word embedding with the output of a convolution layer based on the characters. Computing representations through this kind of layer for the whole vocabulary would be very costly, which is why we use sampling-based methods for training.

We carry out experiments on Czech, with data from the parallel corpus *News-commentary* 2015. In those experiments, we train models whose output representations are simple word embeddings and models with a Words + CharCNN output layer, for various vocabulary sizes. When trained with a full vocabulary, the model equipped with augmented output representations outperforms the classical one. We then experiment with various types of input and output representations: consistently, using a concatenation of words and character-based embeddings improves the perplexity of the mode, while the performances of character-based only representations are far behind. We also obtain better results with convolution layers than with biLSTMs. Lastly, we manage to improve upon augmented models by reducing the size of the word look-up table in the Words + CharCNN output layer: we only learn output word embeddings for words that are frequent enough and use the embedding of the unknown word token for the others. The other half of these infrequent words representations are determined by their characters. While it seems to hurt the performance of the model on infrequent words, we obtain overall sizable gains in perplexity.

Finally, we reproduce these experiments with the NCE training tricks presented in Section 3, obtaining similar improvements. Besides, initializing the output bias with unigram probabilities drastically improves models trained on smaller vocabularies, making models augmented with character-based representations better than the classic models at every vocabulary size tested. We verify our results on the Penn Treebank, improving performances once by augmenting the output representations, then again by reducing the size of the output look-up table.

Conclusion

This work investigates practical methods to ease training and improve performances of neural language models with large vocabularies. The main limitation of neural language models is their expensive computational cost: it depends on the size of the vocabulary, with which it grows linearly. Despite several training tricks, the most straightforward way to limit computation time is to limit the vocabulary size, which is not a satisfactory solution for numerous tasks. Most of the existing methods used to train large-vocabulary language models revolve around avoiding the computation of the partition function, which is used to ensure that output scores are normalized into a probability distribution. While *Importance Sampling* (IS, Section 2.2) directly approximates the gradient computation, *Noise Contrastive Estimation* (NCE, Section 2.3.1) and related methods use a discriminative objective function that does not require normalization and let the partition function be parametrized separately.

In Chapter 3 we observe how IS is more efficient than NCE, accross various choices of the number k of noise samples and the noise distribution P_n. We show that this difference is linked to how normalization is handled by the methods. An analyis of the NCE objective function shows that *self-normalization* is necessary for the method to learn. It also shows that the Zipfian shape of the unigram distribution plays an important role in the difficulties met when using NCE. We experiment with including a scaling parameter in the parameters of the model, to complement the self-normalization process. Additionally, to reduce this negative impact of the large variance of values taken by the unigram distribution, we initialize the bias of the output layer according to the noise distribution, which adapts the scaling to the word frequency. Those two approaches show convincing results on both the Penn Treebank and the 1 Billion Words Benchmark.

In Chapter 4, we show that the maximum-likelihood estimation objective and the noise contrastive estimation objective derive from the same class of divergences, called *Bregman* divergences. Using these divergences to solve a surrogate classification task allows us to work with un-normalized distributions. If we apply them directly to functions associating to each word in the vocabulary their binary classification probability, we obtain objective functions for which the weights manipulated are contained in the inverval $[0, 1]$: this makes learning stable. We experiment with parametrized *Beta divergences*, of which NCE is a particular case, for the parameter $\beta = 1$. Results on both the Penn Treebank and the 1 Billion Words Benchmark show that this class of objectives is especially efficient for values of β a little above 1, where the perplexities reached are slightly better than those obtained with NCE.

Computation time is not the only issue met when training large-vocabularies neural language models: a large part of the vocabulary is composed by infrequent words that are hard to represent, damaging the performances of the model. To allow better generalization for these rare words, we rely on subword units. In Chapter 5, we implement various layers building subword-based representations of words. We carry out experiments on Czech, with data from the parallel corpus *News-commentary 2015*. In particular, we show that appending character-based representations to output word embeddings when training a neural language model with a full vocabulary consistently improves the perplexity of the model. We also experiment with reducing the size of the output look-up table : we only learn output word embeddings for words that are frequent enough, and use the embedding of the unknown word token for the others. The other half of these infrequent words representations are determined by their characters. While the performance of the model on infrequent words seem to decrease, we overall improve on the perplexity results again.

Future work

To extend the results we obtained on NCE, experiments should be done with even larger vocabularies. Especially, we would like to explore the impact of the long tails of Zipfian distributions on training. Another interesting research direction could be to exploit the knowledge we have gained in Chapter 3 to better employ context-dependant distributions. Indeed, our experiments so far have shown interesting results with the bigram distribution (in Section 3.1.2), but it is impractical to use. We believe that an efficient use of more complex noise distribution should be explored.

Since Chapter 4 defines a class of objectives, there is certainly quite a lot of directions we would like to explore. Besides experimenting with other possible objectives, it could prove useful to re-evaluate the use of an exponential function to make the neural language model outputs positive. Indeed, it is possible to apply the divergences on any kind of convex space. Alternative objectives exploiting that fact may provide other advantages than those presented here.

The experiments we presented in Chapter 5 explore which kind of representation can prove useful. However, performances should be compared as function of the training cost, which could be the training time, but also the number of parameters used. The language model could also be adapted to predict a lemma and a set of tags instead of a word, since these vocabularies are far smaller than the word vocabulary. We could then generate an associated word exploiting these informations, using recent progress on morphological reinflection tasks. This kind of approach would be particularily interesting, since it would allow to generate words unseen during training.

List of publications

Franck Burlot, Matthieu Labeau, Elena Knyazeva, Thomas Lavergne, Alexandre Allauzen, and François Yvon. Limsi@iwslt'16: Mt track. 2016.

Matthieu Labeau and Alexandre Allauzen. An experimental analysis of noise-contrastive estimation: the noise distribution matters. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 15–20, Valencia, Spain, April 2017a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-2003.

Matthieu Labeau and Alexandre Allauzen. Character and subword-based word representation for neural language modeling prediction. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 1–13, Copenhagen, Denmark, September 2017b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W17-4101.

Matthieu Labeau and Alexandre Allauzen. Représentations continues dérivées des caractères pour un modèle de langue neuronal à vocabulaire ouvert. In *Actes de la 24ème Conférence sur Traitement Automatique des Langues Naturelles*, pages 32–46, Orléans, France, Juin 2017c. URL http://taln2017.cnrs.fr/wp-content/uploads/2017/06/actes_TALN_2017-vol1.pdf#page=42.

Matthieu Labeau and Alexandre Allauzen. Learning with noise-contrastive estimation: Easing training by learning to scale. In *To be published in the Proceedings of COLING 2018, the 27th International Conference on Computational Linguistics*, Santa Fe, United States, August 2018. Association for Computational Linguistics.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1025.

Benjamin Marie, Alexandre Allauzen, Franck Burlot, Quoc-Khanh Do, Julia Ive, elena knyazeva, Matthieu Labeau, Thomas Lavergne, Kevin Löser, Nicolas Pécheux, and François Yvon. Limsi@wmt'15 : Translation task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 145–151, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/W15-3016.

References

Andrei Alexandrescu and Katrin Kirchhoff. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4, New York City, USA, June 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N06/N06-2001.

A. Allauzen and J.L Gauvain. Open vocabulary asr for audiovisual document indexation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2005.

Jacob Andreas, Maxim Rabinovich, Michael I. Jordan, and Dan Klein. On the accuracy of self-normalized log-linear models. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1783–1791, 2015. URL http://papers.nips.cc/paper/5806-on-the-accuracy-of-self-normalized-log-linear-models.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1041.

Paul Baltescu and Phil Blunsom. Pragmatic neural language modelling in machine translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 820–829, Denver, Colorado, May–June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N15-1083.

Arindam Banerjee, Xin Guo, and Hui Wang. On the optimality of conditional expectation as a bregman predictor. *IEEE Trans. Information Theory*, 51(7): 2664–2669, 2005. doi: 10.1109/TIT.2005.850145. URL https://doi.org/10.1109/TIT.2005.850145.

Ayanendranath Basu, Ian. R Harris, Nils L. Hjort, and M. C. Jones. Robust and efficient estimation by minimising a density power divergence. *Biometrika*, 85(3): 549–559, 1998. URL http://oro.open.ac.uk/24027/.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL http://dx.doi.org/10.1109/72.279181.

Yoshua Bengio and Jean-Sébastien Sénécal. Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

Yoshua Bengio and Jean-Sébastien Sénécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Trans. Neural Networks*, 19(4):713–722, 2008.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137 1155, 2003. http://machinelearning.wustl.edu/mlpapers/paper_files/BengioDVJ03.pdf.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Université De Montréal, and Montréal Québec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1):39–71, March 1996. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=234285.234289.

Jeff A. Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Companion Volume of the Proceedings of HLT-NAACL 2003–short Papers - Volume 2*, NAACL-Short '03, pages 4–6, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1073483.1073485. URL https://doi.org/10.3115/1073483.1073485.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL http://arxiv.org/abs/1607.04606.

Aleksandar Botev, Bowen Zheng, and David Barber. Complementary Sum Sampling for Likelihood Approximation in Large Scale Classification. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1030–1038, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/botev17a.html.

Jan A. Botha and Phil Blunsom. Compositional Morphology for Word Representations and Language Modelling. In *Proceedings of the International Conference of Machine Learning (ICML)*, Beijing, China, jun 2014.

Guillaume Bouchard, Théo Trouillon, Julien Perez, and Adrien Gaidon. Accelerating stochastic gradient descent via online learning to sample. *CoRR*, abs/1506.09016, 2015. URL http://arxiv.org/abs/1506.09016.

L. M. Bregman. The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. In *USSR Computational Mathematics and Mathematical Physics*, page 7:200–217, 1967.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=176313.176316.

Franck Burlot and François Yvon. Learning morphological normalization for translation from and into morphologically rich language. *The Prague Bulletin of Mathematical Linguistics (Proc. EAMT)*, 108:49–60, June 2017.

Miguel Á. Carreira-Perpiñán and Geoffrey E. Hinton. On contrastive divergence learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*, 2005. URL http://www.gatsby.ucl.ac.uk/aistats/fullpapers/217.pdf.

Yair Al Censor and Stavros A. Zenios. *Parallel Optimization: Theory, Algorithms and Applications*. Oxford University Press, 1997. ISBN 019510062X.

Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech and Language*, 14(4):283 – 332, 2000. ISSN 0885-2308. doi: https://doi.org/10.1006/csla.2000.0147. URL http://www.sciencedirect.com/science/article/pii/S0885230800901475.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 2635–2639, 2014. URL http://www.isca-speech.org/archive/interspeech_2014/i14_2635.html.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318, Santa Cruz, California, USA, June 1996. Association for Computational Linguistics. doi: 10.3115/981863.981904. URL http://www.aclweb.org/anthology/P96-1041.

Stanley F. Chen and Joshua Goodman. Regular article. *Computer Speech and Language*, 13(4):359–394, 1999. doi: 10.1006/csla.1999.0128.

Wenlin Chen, David Grangier, and Michael Auli. Strategies for training large vocabulary neural language models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1975–1985, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1186.

Xie Chen, Xunying Liu, Mark J. F. Gales, and Philip C. Woodland. Recurrent neural network language model training with noise contrastive estimation for

speech recognition. In *ICASSP*, pages 5411–5415. IEEE, 2015. ISBN 978-1-4673-6997-8. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2015.html#ChenLGW15a.

Colin Cherry. An empirical evaluation of noise contrastive estimation for the neural network joint model of translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 41–46, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1006.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, 2014a. doi: 10.3115/v1/W14-4012. URL http://www.aclweb.org/anthology/W14-4012.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, 2014b. doi: 10.3115/v1/W14-4012. URL http://www.aclweb.org/anthology/W14-4012.

Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014c. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv e-prints*, abs/1412.3555, 2014. URL https://arxiv.org/abs/1412.3555. Presented at the Deep Learning workshop at NIPS2014.

Andrzej Cichocki and Shun ichi Amari. Families of alpha- beta- and gamma- divergences: Flexible and robust measures of similarities. *Entropy*, 12(6):1532–1568, 2010. URL http://dblp.uni-trier.de/db/journals/entropy/entropy12.html#CichockiA10.

Andrzej Cichocki, Sergio Cruces, and Shun-ichi Amari. Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization. *Entropy*, 13(1):134–170, 2011. ISSN 1099-4300. doi: 10.3390/e13010134. URL http://www.mdpi.com/1099-4300/13/1/134.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078186.

Marta R. Costa-jussà and José A. R. Fonollosa. Character-based neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 357–361, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2058.

Ryan Cotterell, Hinrich Schütze, and Jason Eisner. Morphological smoothing and extrapolation of word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1651–1660, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1156.

Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Trans. Speech Lang. Process.*, 5(1):3:1–3:29, December 2007. ISSN 1550-4875.

Imre Csiszar. Why least squares and maximum entropy? an axiomatic approach to inference for linear inverse problems. *Ann. Statist.*, 19(4):2032–2066, 12 1991. doi: 10.1214/aos/1176348385. URL https://doi.org/10.1214/aos/1176348385.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-1129.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

Chris Dyer. Notes on noise contrastive estimation and negative sampling. *CoRR*, abs/1410.8251, 2014. URL http://arxiv.org/abs/1410.8251.

Shinto Eguchi and Shogo Kato. Entropy and divergence associated with power function and the statistical application. *Entropy*, 12 (2):262–274, 2010. URL https://pdfs.semanticscholar.org/7404/8a5d7b51bdbf8d693ff00a0f5312291ee161.pdf.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990a. URL http://dblp.uni-trier.de/db/journals/cogsci/cogsci14.html#Elman90.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990b. doi: 10.1016/0364-0213(90)90002-E. URL http://groups.lis.illinois.edu/amag/langev/paper/elman90findingStructure.html.

Manaal Faruqui, Yulia Tsvetkov, Graham Neubig, and Chris Dyer. Morphological inflection generation using character sequence to sequence learning. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 634–643, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1077.

Bela A. Frigyik, Santosh Srivastava, and Maya R. Gupta. Functional bregman divergence and bayesian estimation of distributions. *IEEE Trans. Information Theory*, 54(11):5130–5139, 2008.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1155.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

Irving John Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264, 1953.

Joshua Goodman. Classes for fast maximum entropy training. In *ICASSP*, pages 561–564. IEEE, 2001a. ISBN 0-7803-7041-4. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2001.html#Goodman01.

Joshua T. Goodman. A bit of progress in language modeling. *Comput. Speech Lang.*, 15(4):403–434, October 2001b. ISSN 0885-2308. doi: 10.1006/csla.2001.0174. URL http://dx.doi.org/10.1006/csla.2001.0174.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1302–1310, 2017. URL http://proceedings.mlr.press/v70/grave17a.html.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In *Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, 15th International Conference, Warsaw, Poland, September 11-15, 2005, Proceedings, Part II*, pages 799–804, 2005.

Peter D. Grünwald and Philip Dawid. Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory. *The Annals of Statistics*, 32: 1367–1433, 2004.

Michael Gutmann and Junichiro Hirayama. Bregman divergence as general framework to estimate unnormalized statistical models. In *UAI*, 2011.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 297–304, 2010. URL http://www.jmlr.org/proceedings/papers/v9/gutmann10a.html.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012. URL http://dl.acm.org/citation.cfm?id=2188396.

M.U. Gutmann and A. Hyvärinen. Estimation of unnormalized statistical models without numerical integration. In *Proceedings of the Workshop on Information Theoretic Methods in Science and Engineering*, 2013. URL http://www.me.inf.kyushu-u.ac.jp/witmse2013/.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123. URL http://dx.doi.org/10.1109/ICCV.2015.123.

Georg Heigold, Guenter Neumann, and Josef van Genabith. An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 505–513, Valencia, Spain, April 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-1048.

Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf.

Timothy Classen Hesterberg. Advances in importance sampling, 2003.

Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002. URL http://dx.doi.org/10.1162/089976602760128018.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8. URL http://dx.doi.org/10.1016/0893-6080(89)90020-8.

Kyuyeon Hwang and Wonyong Sung. Character-level language modeling with hierarchical recurrent neural networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 5720–5724, 2017. doi: 10.1109/ICASSP.2017.7953252. URL https://doi.org/10.1109/ICASSP.2017.7953252.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL http://dblp.uni-trier.de/db/conf/icml/icml2015.html#IoffeS15.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1001.

Fred Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In Edzard S. Gelsema and Laveen N. Kanal, editors, *Proceedings, Workshop on Pattern Recognition in Practice*, pages 381–397. North Holland, Amsterdam, 1980.

Shihao Ji, S. V. N. Vishwanathan, Nadathur Satish, Michael J. Anderson, and Pradeep Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *CoRR*, abs/1511.06909, 2015. URL http://arxiv.org/abs/1511.06909.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016. URL http://arxiv.org/abs/1602.02410.

Katharina Kann, Ryan Cotterell, and Hinrich Schütze. Neural multi-source morphological reinflection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 514–524, Valencia, Spain, April 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-1049.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987. URL http://dblp.uni-trier.de/db/journals/tsp/tsp35.html#Katz87.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14.

Reinhard Kneser and Hermann Ney. Improved clustering techniques for class-based statistical language modelling. In *EUROSPEECH*. ISCA, 1993. URL http://dblp.uni-trier.de/db/conf/interspeech/eurospeech1993.html#KneserN93.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '95, Detroit, Michigan, USA, May 08-12, 1995*, pages 181–184, 1995a. URL http://dx.doi.org/10.1109/ICASSP.1995.479394.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 181–184, Detroit, Michigan, May 1995b.

Hai-Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and Francois Yvon. Structured output layer neural network language model. In *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*, pages 5524–5527, Prague, Czech Republic, 2011.

Hai-Son Le, Alexandre Allauzen, and François Yvon. Continuous space translation models with neural networks. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 39–48, Montréal, Canada, June 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N12-1005.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1176.

Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-3512.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1101.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2): 313–330, June 1993. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972470.972475.

Mercedes Garcia Martinez, Loïc Barrault, and Fethi Bougares. Factored neural machine translation architectures. In *International Workshop on Spoken Language Translation (IWSLT'16)*, Seattle (USA), 2016.

Oren Melamud, Ido Dagan, and Jacob Goldberger. PMI matrix approximations with applications to neural language modeling. *CoRR*, abs/1609.01235, 2016. URL http://arxiv.org/abs/1609.01235.

Oren Melamud, Ido Dagan, and Jacob Goldberger. A simple language model based on pmi matrix approximations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1861–1866, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1198.

Risto Miikkulainen and Michael G. Dyer. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15(3):343–399, 1991. doi: 10.1207/s15516709cog1503_2. URL https://doi.org/10.1207/s15516709cog1503_2.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010. URL http://www.isca-speech.org/archive/interspeech_2010/i10_1045.html.

Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *ICASSP*, pages 5528–5531. IEEE, 2011. ISBN 978-1-4577-0539-7. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2011.html#MikolovKBCK11.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Yasumasa Miyamoto and Kyunghyun Cho. Gated word-character recurrent language model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1992–1997, Austin, Texas, November 2016. Association for Computational Linguistics. URL https://aclweb.org/anthology/D16-1209.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 641–648, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273577. URL http://doi.acm.org/10.1145/1273496.1273577.

Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1081–1088. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3583-a-scalable-hierarchical-distributed-language-model.pdf.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012. URL http://icml.cc/2012/papers/855.pdf.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252. Society for Artificial Intelligence and Statistics, 2005. URL http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf.

Thomas Mueller and Hinrich Schuetze. Improved modeling of out-of-vocabulary words using morphological classes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 524–528, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-2092.

Luis E. Ortiz and Leslie Pack Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 446–454, San Francisco,

CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. URL http://dl.acm.org/citation.cfm?id=2073946.2073998.

Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043083.

Miika Pihlaja, Michael Gutmann, and Aapo Hyvärinen. A family of computationally efficient and simple estimators for unnormalized statistical models. *CoRR*, abs/1203.3506, 2012. URL http://arxiv.org/abs/1203.3506.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2067.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 141–150, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics. URL http://www.aclweb.org/anthology/C14-1015.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Internal Representations by Error Propagation, pages 673–695. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104449.

Cicero D. Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826. JMLR Workshop and Conference Proceedings, 2014. URL http://jmlr.org/proceedings/papers/v32/santos14.pdf.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013. URL http://arxiv.org/abs/1312.6120.

Holger Schwenk. Continuous space language models. *Comput. Speech Lang.*, 21(3): 492–518, July 2007. ISSN 0885-2308.

Holger Schwenk and Jean-Luc Gauvain. Connectionist language modeling for large vocabulary continuous speech recognition. In *ICASSP*, pages 765–768. IEEE, 2002. ISBN 0-7803-7402-9. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2002.html#SchwenkG02.

Holger Schwenk and Jean-Luc Gauvain. Neural network language models for conversational speech recognition. In *ACM Transactions on Speech and Language Processing - TSLP*, 01 2004.

Holger Schwenk, Anthony Rousseau, and Mohammed Attik. Large, pruned or continuous space language models on a gpu for statistical machine translation. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, WLM '12, pages 11–19, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=2390940.2390942.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the Association for Computational Linguistics*, pages 1715–1725, Berlin, Germany, August 2016.

Henning Sperr, Jan Niehues, and Alex Waibel. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 30–39, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-3204.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

Jana Straková, Milan Straka, and Jan Hajič. Open-source tools for morphology, lemmatization, pos tagging and named entity recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-5003.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH*, 2012.

Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, B. Freiberg, Ralf Schlüter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *ICASSP*, pages 8430–8434. IEEE, 2013. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2013.html#SundermeyerOGFSN13.

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the*

28th International Conference on Machine Learning (ICML-11), ICML '11, pages 1017–1024, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Clara Vania and Adam Lopez. From characters to words to in between: Do we capture morphology? *CoRR*, abs/1704.08352, 2017. URL http://arxiv.org/abs/1704.08352.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D13-1140.

Lyan Verwimp, Joris Pelemans, Hugo Van hamme, and Patrick Wambacq. Character-word LSTM language models. In *EACL*, 2017.

Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. *Readings in Speech Recognition*, chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Publishing Company, Incorporated, 2004. ISBN 1441923225, 9781441923226.

P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

Paul J Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 70(10):1550–1560, 1990.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Charagram: Embedding words and sentences via character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1504–1515, Austin, Texas, November 2016. Association for Computational Linguistics. URL https://aclweb.org/anthology/D16-1157.

I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

Wei Xu and Alex Rudnicky. Can artificial neural networks learn language models? In *INTERSPEECH'00*, pages 202–205, 2000.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1409.html#ZarembaSV14.

Barret Zoph, Ashish Vaswani, Jonathan May, and Kevin Knight. Simple, fast noise-contrastive estimation for large RNN vocabularies. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1217–1222, San

Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1145.

Geoffrey Zweig and Konstantin Makarychev. Speed regularization and optimality in word classing. IEEE, January 2013. URL https://www.microsoft.com/en-us/research/publication/speed-regularization-and-optimality-in-word-classing/.

Appendices

Appendix A

Proofs on Bregman divergences

Proof that $\phi^{BinClassif}$ is convex $\Rightarrow$ ϕ^{Ratio} is convex

Let the following function be convex:

$$\begin{array}{rccc} \phi^{BinClassif}: & [0,1] & \rightarrow & \mathbb{R} \\ & p & \rightarrow & \phi^{BinClassif}(p) \end{array} \tag{A.1}$$

We would like to prove that the following function is convex:

$$\begin{array}{rccc} \phi^{Ratio}: & \mathbb{R}_+ & \rightarrow & \mathbb{R} \\ & r & \rightarrow & (1+r)\phi^{BinClassif}\left(\dfrac{r}{1+r}\right) \end{array} \tag{A.2}$$

Our departure point is that $\forall r, s \in \mathbb{R}_+$, and $\forall t \in [0,1]$,

$$1 + tr + (1-t)s = t(1+r) + (1-t)(1+s) \tag{A.3}$$

Then, $\forall r, s \in \mathbb{R}_+$, and $\forall t \in [0,1]$,

$$\begin{aligned} &\phi^{Ratio}(tr + (1-t)s) = (1 + tr + (1-t)s)\phi^{BinClassif}\left(\frac{tr + (1-t)s}{1 + tr + (1-t)s}\right) \\ &= (t(1+r) + (1-t)(1+s))\times \\ &\phi^{BinClassif}\left(\frac{t(1+r)}{t(1+r) + (1-t)(1+s)}\frac{r}{1+r} + \frac{(1-t)(1+s)}{t(1+r) + (1-t)(1+s)}\frac{s}{1+s}\right) \end{aligned} \tag{A.4}$$

Since $\phi^{BinClassif}$ is convex, and since

$$\begin{cases} \frac{r}{1+r} \in [0,1] \\ \frac{s}{1+s} \in [0,1] \\ \frac{t(1+r)}{t(1+r)+(1-t)(1+s)} \in [0,1] \\ \frac{(1-t)(1+s)}{t(1+r)+(1-t)(1+s)} \in [0,1] \\ \frac{t(1+r)}{t(1+r)+(1-t)(1+s)} + \frac{(1-t)(1+s)}{t(1+r)+(1-t)(1+s)} = 1 \end{cases} \tag{A.5}$$

we have that:

$$\begin{aligned}\phi^{Ratio}(tr+(1-t)s) &\leq t(1+r)\phi^{BinClassif}\left(\frac{r}{1+r}\right)\\ &\quad+(1-t)(1+s)\phi^{BinClassif}\left(\frac{s}{1+s}\right)\\ &= t\phi^{Ratio}(r)+(1-t)\phi^{Ratio}(s)\end{aligned} \tag{A.6}$$

which makes ϕ^{Ratio} convex on $\mathbb{R}_+$.

Proof that minimizing $L_{\Phi^{Ratio}}(\theta) \Leftrightarrow$ minimizing $L_{\Phi^{BinClassif}}(\theta)$

We will now show that minimizing the Bregman divergence between the functions $P^H_{\mathcal{D}}(C=1|.)$ and $P^H_{\theta}(C=1|.)$ is equivalent to minimizing a different Bregman divergence, between the ratios of the data and noise distributions $r^H_{\mathcal{D}}$, and model and noise distributions $r^H_{\theta}(w)$, that are:

$$r^H_{\mathcal{D}}(w)=\frac{P^H_{\mathcal{D}}(w)}{kP_n(w)} \quad \text{and} \quad r^H_{\theta}(w)=\frac{P^H_{\theta}(w)}{kP_n(w)} \tag{A.7}$$

Hence, we will prove that:

$$D_{\Phi^{Ratio}}(r^H_{\mathcal{D}}, r^H_{\theta}) = D_{\Phi^{BinClassif}}(P^H_{\mathcal{D}}(C=1|.), P^H_{\theta}(C=1|.)) \tag{A.8}$$

By using Equations 4.6, 4.2 and 4.38, we obtain that:

$$\begin{aligned}D_{\Phi^{Ratio}}(r^H_{\mathcal{D}}, r^H_{\theta}) = \sum_{w\in\mathcal{V}} \Bigg[&(1+r^H_{\mathcal{D}}(w))\phi^{BinClassif}\left(\frac{r^H_{\mathcal{D}}(w)}{1+r^H_{\mathcal{D}}(w)}\right)\\ &-(1+r^H_{\theta}(w))\phi^{BinClassif}\left(\frac{r^H_{\theta}(w)}{1+r^H_{\theta}(w)}\right)\\ &-\phi'^{Ratio}(r^H_{\theta}(w))(r^H_{\mathcal{D}}(w)-r^H_{\theta}(w))\Bigg]\mu^{Ratio}_{\mathcal{V}}(w)\end{aligned} \tag{A.9}$$

By taking the derivative of the composition function presented in Equation 4.38, we obtain that:

$$\nabla\phi^{ratio}(r)=\phi^{BinClassif}\left(\frac{r}{1+r}\right)+\frac{1}{1+r}\phi'^{BinClassif}\left(\frac{r}{1+r}\right) \tag{A.10}$$

From Equation A.7, and as seen in Equation 4.39, since

$$\frac{\mu^{BinClassif}_{\mathcal{V}}(w)}{\mu^{Ratio}_{\mathcal{V}}(w)}=1+r^H_{\mathcal{D}}(w) \tag{A.11}$$

We obtain the following:

$$D_{\Phi^{Ratio}}(r^H_{\mathcal{D}}, r^H_\theta) = \sum_{w\in\mathcal{V}} \Bigg[\phi^{BinClassif}(P^H_{\mathcal{D}}(w)) - \phi^{BinClassif}(P^H_\theta(w)) - \phi'^{BinClassif}(P^H_\theta(w))\frac{(r^H_{\mathcal{D}}(w) - r^H_\theta(w))}{(1+r^H_{\mathcal{D}}(w))(1+r^H_\theta(w))}\Bigg]\mu^{BinClassif}_{\mathcal{V}}(w) \tag{A.12}$$

Lastly,

$$\begin{aligned} P^H_{\mathcal{D}}(w) - P^H_\theta(w) &= \frac{r^H_{\mathcal{D}}(w)(1+r^H_\theta(w)) - r^H_\theta(w))(1+r^H_{\mathcal{D}}(w))}{(1+r^H_{\mathcal{D}}(w))(1+r^H_\theta(w))} \\ &= \frac{(r^H_{\mathcal{D}}(w) - r^H_\theta(w))}{(1+r^H_{\mathcal{D}}(w))(1+r^H_\theta(w))} \end{aligned} \tag{A.13}$$

Which finally gives:

$$D_{\Phi^{Ratio}}(r^H_{\mathcal{D}}, r^H_\theta) = D_{\Phi^{BinClassif}}(P^H_{\mathcal{D}}(C=1|.), P^H_\theta(C=1|.)) \tag{A.14}$$

Appendix B

Subword-based models: supplementary results with NCE

Output Representation / Input Representation		Words	Words + Char		Char	
			CNN	BiLSTM	CNN	BiLSTM
Words		421.9 ± 26.2	333.7 ± 12.0	355.3 ± 25.9	939.6 ± 54.8	1481.8 ± 61.8
Char	CNN	437.1 ± 7.4	386.0 ± 12.2	-	941.1 ± 39.4	-
	BiLSTM	665.1 ± 43.6	-	608.4 ± 39.8	-	1638.1 ± 53.1
Words + Char	CNN	389.7 ± 19.1	320.6 ± 4.4	-	888.2 ± 33.6	-
	BiLSTM	402.9 ± 38.7	-	338.5 ± 19.0	-	1408.9 ± 52.7
Lemmas + Tags	Concat.	406.3 ± 17.1	335.7 ± 9.9	343.0 ± 18.1	985.2 ± 25.4	1491.9 ± 68.6
	BiLSTM	431.1 ± 30.0	365.1 ± 30.0	350.3 ± 29.6	946.5 ± 66.4	1462.2 ± 45.8

Table B.1: Average test perplexities obtained when training 5 models with NCE, for various input/output representations.

Input Representation	f_{Th}^{W}	Words + CharCNN		
		All words	Frequent words	Rare words
Words	0	333.7 ± 12.0	218.5 ± 7.5	4431.3 ± 364.9
	1	326.4 ± 10.6	213.3 ± 6.8	4352.6 ± 449.2
	2	312.9 ± 9.7	204.5 ± 6.7	4323.1 ± 263.8
	5	317.3 ± 16.2	204.8 ± 11.1	5141.5 ± 170.9
	10	314.1 ± 17.3	196.1 ± 11.1	7485.5 ± 279.5
CharCNN	0	386.0 ± 12.2	252.6 ± 7.5	4857.4 ± 368.5
	1	374.1 ± 23.6	245.0 ± 15.0	4813.8 ± 308.1
	2	358.8 ± 7.2	235.2 ± 4.1	4557.8 ± 359.6
	5	363.7 ± 18.2	235.0 ± 12.1	5727.9 ± 454.0
	10	370.9 ± 13.7	231.6 ± 8.3	8332.6 ± 577.8
Words + CharCNN	0	311.7 ± 3.2	205.4 ± 1.9	4021.0 ± 207.0
	1	302.1 ± 7.4	197.8 ± 4.8	4182.5 ± 280.1
	2	288.4 ± 12.1	188.7 ± 7.3	3971.5 ± 391.0
	5	297.0 ± 12.0	191.5 ± 7.9	4991.0 ± 215.7
	10	287.5± 10.2	180.0 ± 6.8	6735.8 ± 312.9
Lemma + TagsConcat.	0	335.7 ± 9.9	218.5 ± 7.0	4702.6 ± 267.8
	1	328.4 ± 10.2	214.2 ± 7.7	4503.3 ± 104.6
	2	322.8 ± 6.6	208.8 ± 4.6	4779.3 ± 296.6
	5	325.9 ± 13.6	207.7 ± 8.9	5783.4 ± 418.3
	10	341.1 ± 22.9	212.1 ± 15.0	8368.1 ± 445.4
Lemma + TagsBiLSTM	0	395.9 ± 36.7	257.9 ± 24.0	5412.0 ± 476.2
	1	366.8 ± 25.9	239.1 ± 16.1	5074.2 ± 522.9
	2	344.7 ± 22.6	223.8 ± 14.2	4868.2 ± 244.3
	5	324.6 ± 13.8	207.2 ± 8.1	5493.8 ± 269.0
	10	341.6 ± 24.2	213.5 ± 16.4	7542.6 ± 762.4

Table B.2: Test perplexity averaged on 5 models trained with NCE, for various input representations and output word look-up table sizes. Corresponding vocabulary sizes are given in Table 5.3a. Test perplexities are given for all words, frequent words (frequency > 10) and rare words (frequency < 10).

Appendix C

Subword-based models: supplementary results on embedding sizes influence

Output Representation Input Representation	Words	Words + CharCNN $\mathcal{V}_{Th}^{W} = 0$	Words + CharCNN $\mathcal{V}_{Th}^{W} = 10K$
Words	137.3 ± 1.8 27,0	147.8 ± 2.6 20,6	131.5 ± 1.6 15,4
Words + CharCNN	133.8 ± 6.3 20,6	147.1 ± 1.3 13,8	**130.8 ± 4.4** 10,1

Table C.1: Average test perplexities obtained when training 5 models with NCE, for various input representations and output word look-up table sizes, on PTB, followed by the number of parameters of the corresponding model (in millions)

In Table C.1, all models share the same input and output representation total size: 300. We explain the difference between the (underlined) baseline result of 137.3 and the perplexity of 148.4 obtained at the end of Chapter 3 by the presence of the highway layers, which here amounts to adding 2 hidden layers. While using the same total representation sizes shows that the perplexity improvements are limited, we still improve results, while consequently decreasing the number of necessary parameters.

Appendix D

Previous work on subword-based POS tagging

Non-lexical neural architecture for fine-grained POS Tagging

Matthieu Labeau, Kevin Löser, Alexandre Allauzen
Université Paris-Sud and LIMSI-CNRS,
Rue John von Neumann
91403 Orsay cedex
France
firstname.lastname@limsi.fr

Abstract

In this paper we explore a POS tagging application of neural architectures that can infer word representations from the raw character stream. It relies on two modelling stages that are jointly learnt: a convolutional network that infers a word representation directly from the character stream, followed by a prediction stage. Models are evaluated on a POS and morphological tagging task for German. Experimental results show that the convolutional network can infer meaningful word representations, while for the prediction stage, a well designed and structured strategy allows the model to outperform state-of-the-art results, without any feature engineering.

1 Introduction

Most modern statistical models for natural language processing (NLP) applications are strongly or fully lexicalized, for instance part-of-speech (POS) and named entity taggers, as well as language models, and parsers. In these models, the observed word form is considered as the elementary unit, while its morphological properties remain neglected. As a result, the vocabulary observed on training data heavily restricts the generalization power of lexicalized models.

Designing subword-level systems is appealing for several reasons. First, words sharing morphological properties often share grammatical function and meaning, and leveraging that information can yield improved word representations. Second, a subword-level analysis can address the out-of-vocabulary issue *i.e* the fact that word-level models fail to meaningfully process unseen word forms. This allows a better processing of morphologically rich languages in which there is a combinatorial explosion of word forms, most of which are not observed during training. Finally, using subword units could allow processing of noisy text such as user-generated content on the Web, where abbreviations, slang usage and spelling mistakes cause the number of word types to explode.

This work investigates models that do not rely on a fixed vocabulary to make a linguistic prediction. Our main focus in this paper is POS tagging, yet the proposed approach could be applied to a wide variety of language processing tasks. Our main contribution is to show that neural networks can successfully learn unlexicalized models that infer a useful word representation from the character stream. This approach achieves state of-the-art performance on a German POS tagging task. This task is difficult because German is a morphologically rich language[1], as reflected by the large number of morphological tags (255) in our study, yielding a grand total of more than 600 POS+MORPH tags. An aggravating factor is that these morphological categories are overtly marked by a handful of highly ambiguous inflection marks (suffixes). We therefore believe that this case study is well suited to assess both the representation and prediction power of our models.

The architecture we explore in section 2 differs from previous work that only consider the character level. Following (Santos and Zadrozny, 2014), it consists in two stages that are jointly learnt. The lower stage is a convolutional network that infers a word embedding from a character string of arbitrary size, while the higher network infers the POS tags based on this word embedding sequence. For the latter, we investigate different architectures of increasing complexities: from a feedforward and context-free inference to a bi-recurrent network that predicts the global sequence. Experimental results (section 4) show that the proposed approach can achieve state of the art performance

[1]Besides inflected forms, German is characterized by a possibly infinite and evolving set of compound nouns.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 232–237,
Lisbon, Portugal, 17-21 September 2015.

and that the choice of architecture for the prediction part of the model has a significant impact.

2 Network Architectures

The different architectures we propose act in two stages to infer, for a sentence $s = \{w_1, \ldots, w_{|s|}\}$, a sequence of tags $\{t_1, \ldots, t_{|s|}\}$. Each tag belongs to the tagset $\mathcal{T}$. The first stage is designed to represent each word *locally*, and focuses on capturing the meaningful morphological information. In the second stage, we investigate different ways to predict the tag sequence that differ in how the *global* information is used.

2.1 From character to word level

To obtain word embeddings, the usual approach introduced by (Bengio et al., 2003) relies on a fixed vocabulary $\mathcal{W}$ and each word $w \in \mathcal{W}$ is mapped to a vector of n_f real valued features by a look-up matrix $W \in \mathbb{R}^{|\mathcal{W}|*n_f}$. To avoid the use of a fixed vocabulary, we propose to derive a word representation from a sequence of character embedding: if $\mathcal{C}$ denotes the finite set of characters, each character is mapped on a vector of n_c features gathered in the look-up matrix C.

To infer a word embedding , we use a *convolution layer* (Waibel et al., 1990; Collobert et al., 2011), build as in (Santos and Zadrozny, 2014). As illustrated in figure 1, a word w is a character sequence $\{c_1, .., c_{|w|}\}$ represented by their embeddings $\{C_{c_1}, .., C_{c_{|w|}}\}$, where C_{c_i} denotes the row in C associated to the character c_i. A convolution filter $W^{conv} \in \mathbb{R}^{n_f} \times \mathbb{R}^{d_c * n_c}$ is applied over a sliding window of d_c characters, producing local features :

$$x_n = W^{conv}(C_{c_{n-d_c+1}} : .. : C_{c_n})^T + b^{conv},$$

where x_n is a vector of size n_f obtained for each position n in the word[2]. The i-th element of the embedding of w is the maximum over the i-th elements of the feature vectors :

$$[f]_i = \tanh(\max_{1 \leq n \leq |s|} [x_n]_i)$$

Using a maximum after a sliding convolution window ensures that the embedding combines local features from the whole word, and selects the more useful ones. The parameters of the layer are the matrices C and W^{conv} and the bias b^{conv}.

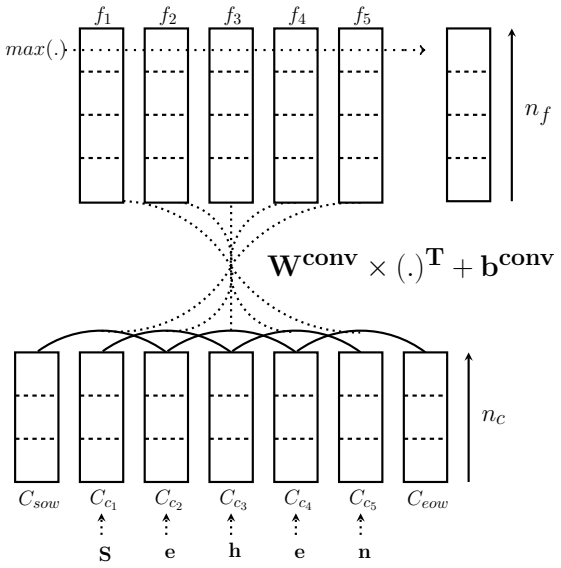

Figure 1: Architecture of the layer for character-level encoding of words.

2.2 From words to prediction

To predict the tag sequence associated to a sentence s, we first use **a feedforward architecture**, with a single hidden layer. To compute the probability of tagging the n-th word in the sentence with tag t_i, we use a window of d_w word embeddings[3] centered around the word w_n:

$$x_n = f_{n-\frac{d_w-1}{2}} : ... : f_{n+\frac{d_w-1}{2}},$$

followed by a hidden and output layers:

$$s_n = W^o \tanh(W^h x_n + b^h) + b^o. \qquad (1)$$

The parameters of the hidden an output layers are respectively W^h, b^h and W^o, b^o.

We also experiment with a **a bidirectional recurrent layer**, as described in (Graves et al., 2013). The forward and backward passes allow each prediction to be conditioned on the complete past and future contexts, instead of merely a neighboring window. As illustrated in figure 2, the forward hidden state, at position n, will be computed using the previous forward hidden state and the word embedding in position n:

$$\overrightarrow{h^n} = \tanh(\overrightarrow{W^{fh}} f_n + \overrightarrow{W^{hh}} \overrightarrow{h^{n-1}} + b^h)$$

[2]Two padding character tokens are used to deal with border effects. The first is added at the beginning and the second at the end of the word, as many times as it is necessary to obtain the same number of windows than the length of the word. Their embeddings are added to C.

[3]Similarly, we use special word tokens for padding.

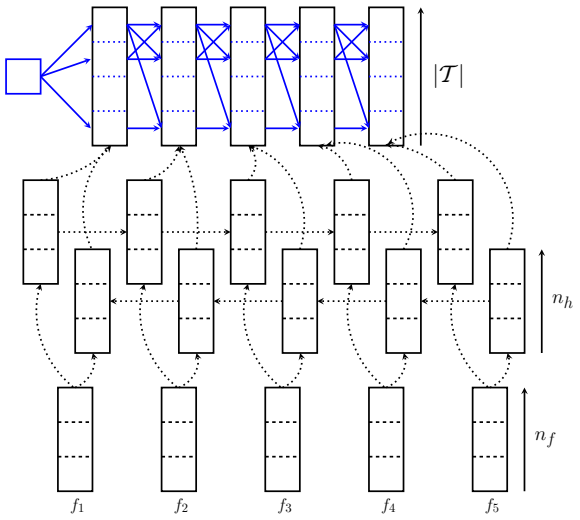

Figure 2: Bidirectional recurrent architecture for tag prediction. The upper part is used in the case of structured inference.

$\overrightarrow{W^{fh}}$ and $\overrightarrow{W^{hh}}$ are the transition matrices of the forward part of the layer, and b_h is the bias. The backward hidden states are computed similarly, and the hidden states of each direction are concatenated to pass through an output layer:

$$s_n = W^o(\overrightarrow{h^n} : \overleftarrow{h^n}) + b^o. \quad (2)$$

2.3 Inference and Training

To infer the tag sequence from the sequence of output layers defined by equations 1 or 2, we explore two strategies. The first simply applies a softmax function to the output layer of the network described in the previous section. In this case, each tag prediction is made independently of the surrounding predictions.

For sequence labeling, a more appropriate solution relies on the approach of (Collobert, 2011), also used in (Santos and Zadrozny, 2014). Let consider each possible tag sequence $\{t_1, \ldots, t_{|s|}\}$ as a possible path over a sequence of hidden states. We can add a transition matrix W^{trans} and then compute the score of a sequence as follows:

$$s(\{t\}_1^{|s|}, \{w\}_1^{|s|}) = \sum_{1 \leq n \leq |s|} \left(W^{trans}_{t_{n-1}, t_n} + [s_n]_{t_n} \right)$$

The Viterbi algorithm (Viterbi, 1967) offers an exact solution to infer the path that gives the maximum score. It is worth noticing that both these strategies can be applied to the feedforward and bidirectional recurrent networks. For both strategies, the whole network can estimate conditional log-likelihood of a tag sequence given a sentence s and the set of parameters θ. This criterion can then be optimized using a stochastic gradient ascent with the back-propagation algorithm.

3 Related Work

The choice to consider words from the character level has recently been more and more explored. While its raw application to language modeling did not achieve clear improvement over the word-based models (Mikolov et al., 2012), this approach shown impressive results for text generation (Sutskever et al., 2011; Graves, 2013). However, for this line of work, the main issue is to learn long range dependencies at the character level since the word level is not considered by the model.

More recently, the character level was considered as more interpretable and convenient way to explore and understand recurrent networks (Karpathy et al., 2015). In (Zhang and LeCun, 2015), the authors build a text understanding model that does not require any knowledge and uses hierarchical feature extraction. Here the character level allows the model to ignore the definition *a priori* of a vocabulary and let the model build its own representation of a sentence or a document, directly from the character level. To some extent, our work can be considered as an extension of their work, tailored for POS tagging.

(Santos and Zadrozny, 2014) applies a very similar model to the POS tagging of Portuguese and English. (Luong et al., 2013) also descends lower than the word level, using a dictionary of morphemes and recursive neural networks to model the structure of the words. Similarly, this allows a better representation of rare and complex words, evaluated on a word similarity task.

4 Experiments and Results

Experiments are carried out on the Part-of-Speech and Morphological tagging tasks using the German corpus TIGER Treebank (Brants et al., 2002). To the best of our knowledge, the best results on this task were published in (Mueller et al., 2013), who applied a high-order CRF that includes an intensive feature engineering to five different languages. German was highlighted as having 'the most ambiguous morphology'. The corpus, de-

Architecture	Encoding	Output	POS		POS+Morph	
			Dev	Test	Dev	Test
Feedforward	Lex.	Simple	4.22 ± 0.05	5.89 ± 0.07	13.97 ± 0.14	17.46 ± 0.14
		Struct.	3.90 ± 0.05	5.33 ± 0.09	12.22 ± 0.13	15.34 ± 0.13
	Non-lex.	Simple	3.31 ± 0.07	4.22 ± 0.07	13.50 ± 0.16	16.23 ± 0.13
		Struct.	2.92 ± 0.02	3.82 ± 0.04	11.65 ± 0.11	14.43 ± 0.19
	Both	Simple	2.59 ± 0.05	3.34 ± 0.09	11.89 ± 0.14	14.63 ± 0.22
		Struct.	$2.22 \pm 0.03^*$	$2.86 \pm 0.03^*$	9.11 ± 0.14	11.29 ± 0.06
biRNN	Lex	Simple	6.03 ± 0.06	8.05 ± 0.05	17.83 ± 0.11	21.33 ± 0.26
		Struct.	3.89 ± 0.06	5.26 ± 0.05	11.88 ± 0.05	17.78 ± 0.12
	Non-Lex	Simple	4.46 ± 0.08	5.84 ± 0.19	16.61 ± 0.18	19.39 ± 0.12
		Struct.	2.74 ± 0.07	3.59 ± 0.07	10.09 ± 0.09	12.88 ± 0.28
	Both	Simple	3.63 ± 0.06	4.63 ± 0.04	14.83 ± 0.11	17.54 ± 0.13
		Struct.	$2.21 \pm 0.04^*$	$2.86 \pm 0.05^*$	$\mathbf{8.63 \pm 0.21}^*$	$\mathbf{10.97 \pm 0.19}^*$
	CRF		**2.06**	**2.56**	9.40	11.42

Table 1: Comparison of the feedforward and bidirectional recurrent architectures for predictions, with different settings. The non-lexical encoding is convolutional. *CRF* refers to state-of-the-art system of (Mueller et al., 2013). *Simple* and *Struct.* respectively denote the position-by-position and structured prediction. $*$ indicates our best configuration.

scribed in details in (Fraser et al., 2013), contains a training set of 40472 sentences, a development and a test set of both 5000 sentences. We consider the two tagging tasks, with first a coarse tagset (54 tags), and then a morpho-syntactical rich tagset (619 items observed on the the training set).

4.1 Experimental settings

All the models are implemented[4] with the Theano library (Bergstra et al., 2010). For optimization, we use Adagrad (Duchi et al., 2011), with a learning rate of 0.1. The other hyperparameters are: the window sizes, d_c and d_w, respectively set to 5 and 9, the dimension of character embeddings, word embeddings and of the hidden layer, n_c, n_f and n_h, that are respectively of 100, 200 and 200[5]. The models were trained on 7 epochs. Parameter initialization and corpus ordering are random, and the results presented are the average and standard deviation of the POS Tagging error rate over 5 runs.

[4]Implementation is available at `https://github.com/MatthieuLabeau/NonlexNN`

[5]For both the learning rate and the embedding sizes, results does not differ in a significant way in a large range of hyperparameters, and their impact resides more in convergence speed and computation time

4.2 Results

The first experiment aims to evaluate the efficiency of a convolutional encoding with the basic feedforward architecture for prediction. We compare a completely non-lexicalized model which relies only on a character-level encoding with a lexicalized model where we use conventional word embeddings stored with a fixed vocabulary[6]. Results are reported in Table 1 along with with the state-of-the-art results published in (Mueller et al., 2013). Results show that a character-level encoding yields better results than the conventional word-level encoding. Moreover, the structured inference allows the model to achieve accuracy reasonably close to the performance of a high-order CRF that uses handcrafted features. Finally, the model that uses the concatenation of both the character and word-level embeddings outperforms the state-of-the-art system on the more difficult task, without any feature engineering.

To give an idea of how a simple model would perform on such task, the reader can refer to (Schmid and Laws, 2008) and (Mueller et al., 2013). For instance in the former, by choosing the most probable tag position-by-position, the error rate on the development set of the TIGER dataset

[6]Every word that appears in the training set.

is 32.7 for the simple POS Tagging task.

We further analyze the results by looking at the error rates respectively on known and unknown words[7]. From table 2, we observe that the number of unknown words wrongly labeled is divided by 3 for POS and almost divided by 2 for POS+Morph tagging, showing the ability of character-level encoding to generalize to new words. Moreover, a strictly non-lexical encoding makes slightly more mistakes on words already seen, whereas the model that concatenates both embeddings will make less mistakes for both unknown and known words.

This shows that information from the context and from the morphology are complementary, which is conjectured in (Mueller et al., 2013) by using a morphological analyzer in complement of higher-order CRF.

		Lex.	Non-lex.	Both
POS	Unknown	2970	1054	1010
	Known	1974	2981	1620
POS+Morph	Unknown	5827	3472	3384
	Known	8652	10205	7232

Table 2: Error counts for known/unknown words in the test set, with a structured feedforward prediction model for the tagging task.

In the second set of experiments, we evaluate the convolutional encoding with a bidirectional recurrent network for prediction. Results are presented in the second half of Table 1. Surprisingly, this architecture performs poorly with simple inference, but clearly improves when predicting a structured output using the Viterbi algorithm, both for training and testing. Moreover, a non-lexical model trained to infer a tag sequence with the Viterbi algorithm achieves results that are close to the state-of-the-art, thus validating our approach. We consider that this improvement comes from the synergy between using a global training objective with a global hidden representation, complexifying the model but allowing a more efficient solution. Finally, the model that uses the combination of both the character and word-level embeddings yields the best results. It is interesting to notice that the predictive architecture has no influence on the results of the simple task when the prediction is structured, but improves them on the difficult task. This also shows that the contribution of word embeddings to our model corresponds to a difference of 1.5 to 2 points in performance.

[7]Unknown words refer to words present in the development or test sets, but not in the training set.

5 Conclusion

In this paper, we explored new models that can infer meaningful word representations from the raw character stream, allowing the model to exploit the morphological properties of words without using any handcrafted features or external tools. These models can therefore efficiently process words that were unseen in the training data. The evaluation was carried out on a POS and morphological tagging task for German. We described different architectures that act in two stages: the first stage is a convolutional network that infers a word representation directly from the character stream, while the second stage performs the prediction. For the prediction stage, we investigated different solutions showing that a bidirectional recurrent network can outperform state-of-the-art results when using a structured inference algorithm.

Our results showed that character-level encoding can address the unknown words problem for morphologically complex languages. In the future, we plan to extend these models to other tasks such as syntactic parsing and machine translation. Moreover, we will also investigate other architectures to infer word embeddings from the character level. For instance, preliminary experiments show that bidirectional recurrent network can achieve very competitive and promising results.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and suggestions. This work has been partly funded by the European Unions Horizon 2020 research and innovation programme under grant agreement No. 645452 (QT21).

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137 1155.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and

GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, pages 24–41.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November.

Ronan Collobert. 2011. Deep learning for efficient discriminative parsing. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 224–232.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July.

Alexander Fraser, Helmut Schmid, Richárd Farkas, Renjing Wang, and Hinrich Schütze. 2013. Knowledge sources for constituent parsing of German, a morphologically rich and less-configurational language. *Comput. Linguist.*, 39(1):57–85, March.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013*, pages 273–278.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pages 104–113.

Tomas Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. Unpublished.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October. Association for Computational Linguistics.

Cicero D. Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826. JMLR Workshop and Conference Proceedings.

Helmut Schmid and Florian Laws. 2008. Estimation of conditional probabilities with decision trees and an application to fine-grained pos tagging. In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 777–784, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024, New York, NY, USA, June. ACM.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theor.*, 13(2):260–269, April.

Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang, 1990. *Readings in Speech Recognition*, chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Xiang Zhang and Yann LeCun. 2015. Text understanding from scratch. *CoRR*, abs/1502.01710.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Modèles de langue neuronaux: Gestion des grands vocabulaires

Mots clés : Réseaux de Neurones, Modèles de Langue, Grands Vocabulaires

Résumé : Le travail présenté dans cette thèse explore les méthodes pratiques utilisées pour faciliter l'entraînement et améliorer les performances des modèles de langues munis de très grands vocabulaires. La principale limite à l'utilisation des modèles de langue neuronaux est leur coût computationnel: il dépend de la taille du vocabulaire avec laquelle il grandit linéairement. La façon la plus aisée de réduire le temps de calcul de ces modèles reste de limiter la taille du vocabulaire, ce qui est loin d'être satisfaisant pour de nombreuses tâches. La plupart des méthodes existantes pour l'entraînement de ces modèles à grand vocabulaire évitent le calcul de la fonction de partition, qui est utilisée pour forcer la distribution de sortie du modèle à être normalisée en une distribution de probabilités. Ici, nous nous concentrons sur les méthodes à base d'échantillonnage, dont le sampling par importance et l'estimation contrastive bruitée. Ces méthodes permettent de calculer facilement une approximation de cette fonction de partition. L'examen des mécanismes de l'estimation contrastive bruitée nous permet de proposer des solutions qui vont considérablement faciliter l'entraînement, ce que nous montrons expérimentalement. Ensuite, nous utilisons la généralisation d'un ensemble d'objectifs basés sur l'échantillonnage comme *divergences de Bregman* pour expérimenter avec de nouvelles fonctions objectif. Enfin, nous exploitons les informations données par les unités sous-mots pour enrichir les représentations en sortie du modèle. Nous expérimentons avec différentes architectures, sur le Tchèque, et montrons que les représentations basées sur les caractères permettent l'amélioration des résultats, d'autant plus lorsque l'on réduit conjointement l'utilisation des représentations de mots.

Title : Neural language models: Dealing with large vocabularies

Keywords : Neural Networks, Language Modelling, Large Vocabularies

Abstract : This work investigates practical methods to ease training and improve performances of neural language models with large vocabularies. The main limitation of neural language models is their expensive computational cost: it depends on the size of the vocabulary, with which it grows linearly. Despite several training tricks, the most straightforward way to limit computation time is to limit the vocabulary size, which is not a satisfactory solution for numerous tasks. Most of the existing methods used to train large-vocabulary language models revolve around avoiding the computation of the partition function, ensuring that output scores are normalized into a probability distribution. Here, we focus on sampling-based approaches, including importance sampling and noise contrastive estimation. These methods allow an approximate computation of the partition function. After examining the mechanism of *self-normalization* in noise-contrastive estimation, we first propose to improve its efficiency with solutions that are adapted to the inner workings of the method and experimentally show that they considerably ease training. Our second contribution is to expand on a generalization of several sampling based objectives as *Bregman divergences*, in order to experiment with new objectives. We use *Beta divergences* to derive a set of objectives from which noise contrastive estimation is a particular case. Finally, we aim at improving performances on full vocabulary language models, by augmenting output words representation with subwords. We experiment on a Czech dataset and show that using character-based representations besides word embeddings for output representations gives better results. We also show that reducing the size of the output look-up table improves results even more.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France