Language modelling with LSTM. Experimenting with power divergences for a better rare-words likelihood estimation.

Pawel Budzynski

Université Paris-Saclay, 2022

1 Introduction.

This exercise focuses on creating a *language model* using a LSTM network. The task of language modelling is training a model to estimate the distribution of following word given words that already appeared in a sequence. The model, an LSTM-based network, predicts next word which is the most likely given input words and a remembered context - the historical context is a main motivation to use LSTM network (or another recursive model).

For model's objective function, several families of power divergences will be explored. They will used as the bases of custom loss function - each power divergence is a power-function generalization of the KL divergence. Details of these functions will be explained in the following parts. The general motivation is that the natural Zipfian distribution of real-world linguistic data is reportedly better captured by a power function-based objectives.

2 Dataset

The experiments will be executed using popular PTB Corpus that includes Penn Treebank porion of the Wall Street Journal corpus with the vocabulary size limited to then thousand words. Data set published by the authors of the paper [10] is ready to use in the language modelling task thus no further preprocessing of the data was performed. The data set was preprocessed and spit into train, development and test sets of sizes 930k, 74k and 82k tokens respectively. To limit size of the vocabulary some of the rare tokens were mapped to a special "unknown token" [11].

3 LSTM architecture

The language model trained in this exercise is based on Long-Short Term Memory (LSTM) recurrent neural network [3] where the output for time step t is computed using input x_t and hidden state h_{t-1}, and memory vector c_{t-1} for $t > 0$. A LSTM cell includes a number of gates that regulates the relationship between current input and historical state. There are various variants and implementations of the LSTM architecture. In this exercise a vanilla LSTM model is used. It consists of forget gate $f^{(t)}$, input gate $i^{(t)}$ and output gate $o^{(n)}$:

$$f^{(t)} = \sigma\left(U^{(1)}\begin{bmatrix} x^{(t)} \\ h^{(t-1)} \end{bmatrix} + b^{(1)}\right), \tag{1}$$

$$i^{(t)} = \sigma\left(U^{(2)}\begin{bmatrix} x^{(t)} \\ h^{(t-1)} \end{bmatrix} + b^{(2)}\right), \tag{2}$$

$$o^{(t)} = \sigma\left(U^{(3)}\begin{bmatrix}x^{(t)}\\h^{(t-1)}\end{bmatrix} + b^{(3)}\right). \tag{3}$$

The values coming out from the gates are used to compute time step t hidden state $h^{(t)}$ and memory state $c^{(t)}$ defined as:

$$c^{(t)} = f^{(t)} \times c^{(t-1)} + i^{(t)} \times \tanh\left(U^{(4)}\begin{bmatrix}x^{(t)}\\h^{(t-1)}\end{bmatrix} + b^{(4)}\right), \tag{4}$$

$$h^{(t)} = o^{(t)} \times \tanh(c^{(t)}). \tag{5}$$

Since each gate requires similar operations the LSTM implementation could be optimized using concatenated vectors:

$$U = \begin{bmatrix}U^{(1)}\\U^{(2)}\\U^{(3)}\\U^{(4)}\end{bmatrix}, b = \begin{bmatrix}b^{(1)}\\b^{(2)}\\b^{(3)}\\b^{(4)}\end{bmatrix}, \tag{6}$$

to parallelize the computations $l = U\begin{bmatrix}x^{(t)}\\h^{(t-1)}\end{bmatrix} + b$ and $l_\sigma = \sigma(l^{(1,2,3)})$, $l_{\tanh} = \tanh(l^{(4)})$. Then l_σ contains values of the gates and $l_{\tanh}$ values required for the memory vector update.

4 Power divergences

The intuition lying behind experimenting with various families of divergences comes from two arguments. Firstly, language models are usually trained using Maximum-Likelihood Estimation (MLE) where the objective is derived from Kullback-Leiber (KL) divergence. During a training, the likelihood of a golden word given the context is 1, and for all other words it equals 0. In this case, MLE loss function is reduced to logarithm of the probability of the golden world predicted by the model. The second reason is that in natural languages frequency distributions of words in corpus follow Zipf's law [12]. Because of a significant difference between counts of high-frequency and low-frequency words makes it difficult to properly estimate the underlying distribution. In this context power divergences are expected to stress various aspects of distributions differences and their usage should have direct impact on the model's performance regarding words frequencies. As it was suggested by the authors [8] a proper choice of the parameters should lead either to generalization – concentrating probability mass on training examples or shifting probability mass to low-frequency words.

The experiments performed will use α, β and γ power divergences described in [8] as objective functions. The formulas of the divergences for normalized probabilities are given as follows:

$$Obj_\alpha(\theta) = \frac{1}{\alpha(\alpha-1)} \sum_{(x,y)\in\mathcal{D}} \left(p_\theta(y|x)\right)^{(1-\alpha)}, \tag{7}$$

$$Obj_\beta(\theta) = \frac{1}{\beta(\beta-1)} \sum_{(x,y)\in\mathcal{D}} \left[(\beta-1) \sum_{y'\in\mathcal{Y}} \left(p_\theta(y'|x)\right)^\beta - \beta \left(p_\theta(y|x)\right)^{(\beta-1)} \right], \tag{8}$$

$$Obj_\gamma(\theta) = \sum_{(x,y)\in\mathcal{D}} \left[\log p_\theta(y|x) - \frac{1}{\gamma} \log \sum_{y'\in\mathcal{Y}} \left(p_\theta(y'|x)\right)^\gamma \right]. \tag{9}$$

5 Noise-contrastive estimation

Noise contrastive estimation (NCE) is a technique that aims to avoid computing the expensive softmax function. Softmax function, usually used as a way to trainform logits into likelihoods, contains a partition function over the entire output space. NCE employs a sampling-based objective to improve performance of estimating the likelihood measures. It is quite relevant for language modeling and other problems where the output space is immense - in this exercise there is a separate output neuron for every single word in the corpus which result in quite a large prediction space.

Below a brief overview of NCE and its theoretical basis is presented.

While using softmax function, trying to predict the next word w_{i+1} from vocabulary V assuming context c_i one should compute following:

$$P_\theta^{MLE}\left(w_{i+1} \mid c_i\right) = \frac{\exp\left(s_\theta(w_{i+1})\right)}{\sum_{j=1}^{|V|} \exp\left(s_\theta(w_j)\right)} = \frac{\exp\left(s_\theta(w_{i+1})\right)}{Z} \tag{10}$$

However, the Z term is expensive to compute given a large size of a vocabulary. Instead ignoring the partition function is possible and sampling could be use in the formula.

Instead of computing the MLE for a multinomial classification problem, it could be reduced to a binary classification problem - whenever a target word is generated, another k words samples from a known noise distribution are generated.

Let denote whether the word is from the target or noise distribution with D, on a target distribution p_θ and noise distribution q. Then:

$$p(D=1|w,c) = \frac{1}{k+1} p_\theta(w|c); \quad p(D=0|w,c) = \frac{k}{k+1} q(w) = 1 - p(D=1|w,c), \tag{11}$$

which could be reformulated as:

$$p(D=1|w,c) = \frac{p(D=1|w,c)}{p(D=1|w,c) + p(D=0|w,c)}, \tag{12}$$

and replacing the terms with the (11) substitutions, it gives:

$$p(D=1|w,c) = \frac{p_\theta(w|c)}{kq(w)+p_\theta(w|c)}; \quad p(D=0|w,c) = \frac{kq(w)}{kq(w)+p_\theta(w|c)}. \tag{13}$$

Unfortunately, $p_\theta(w|c)$ is still intractable because it involves computing a partition function. However, there is an easy "hack": assume the probabilities are already normalized, and that the partition function Z equals 1. By that, one can write $p_\theta(w|c) = \frac{\sigma_\theta(w,c)}{Z}$ as just $\sigma_\theta(w,c)$

Plugging this into (13), results in:

$$p(D=1|w,c) = \frac{\sigma_\theta(w,c)}{kq(w)+\sigma_\theta(w,c)}; \quad p(D=0|w,c) = \frac{kq(w)}{kq(w)+\sigma_\theta(w,c)} \tag{14}$$

The obtained formula can be used to define a loss objective. Most often, a binary cross-entropy objective is used. Using the KL divergence between the ground truth and the model predictions, here is a BCE example:

$$J(\theta) = KL(p_{Data}(D=1|w,c)||p_\theta(D=1|w,c)) + KL(p_{Data}(D=0|w,c)||p_\theta(D=0|w,c)) \tag{15}$$

However - the alpha, beta and gamma divergences presented here are all generalizations of the KL divergence. Hence, it should be possible to substitute them for the KL divergence in the above formula and formulate a new noise-contrastive loss objective based on this formula.

6 Approximated softmax

For compuation speed-up, to avoid computation of partitioning function over all logits an *aproximated softmax function* can be used. The formula based on importance sampling was given in the paper [8] as

$$p_\theta(y|x) \approx \frac{\frac{\exp(s_\theta(x,y))}{q(y)}}{\frac{\exp(s_\theta(x,y))}{q(y)} + \sum_{i=1,\hat{y}_i\sim q}^{k} \frac{\exp(s_\theta(x,\hat{y}_i))}{q(\hat{y}_i)}}, \tag{16}$$

where p_θ is a probability parametrized by the language model and s_θ are model's outputs. q is a noise distribution that is ideally as similar as possible to the unknown distribution p that the model is trying to achieve. The formula is based on the connection between NCE and importance sampling [4] and was presented in [9] as a *Ranking-based NCE.* However, in the original paper the formula was given as

$$p_\theta(y|x) \approx \frac{\exp(\bar{s}(x,y;\theta)}{\sum_{i=1,\hat{y}_i\sim q}^{k} \exp(\bar{s}(x,\hat{y}_i;\theta))}, \tag{17}$$

where $\bar{s}(x,y;\theta) = s_\theta(x,y) - \log(q(y))$. The additional element in the denominator introduced in Equation 16 seems to be ensuring that approximated probability value is not exceeding value of one.

The distribution q should be chosen carefully as on the one hand is should be similar to p but also it should be easy to sample from it. A selection of possible noise probabilities q is presented in the Section 7.

7 Noise distributions

Importance sampling, noise-contrastive estimation and approximated softmax require simple distribution that can be used as a substitute of the real distribution of words. In this project three typical distributions are considered and some of smoothing techniques will be presented.

7.1 Distributions

Typically, uniform, unigram and bigram distributions are used in language models as they are relatively simple and intuitive. Higher order n-grams distributions can be used although they are more complicated and expensive to compute. In the following formulas, $c(x)$ represents counts occurrences of item x in the corpus C.

1. Uniform distribution assigns equal probability for each word coming from the vocabulary V.
$$q(w) = \frac{1}{|V|}. \tag{18}$$

2. Unigram distribution constructed from corpus C assigns probability proportional to the amount of occurrences of the word in the corpus.
$$q(w_i) = \frac{c(w_i)}{|C|}. \tag{19}$$

3. Bigram distribution is based on the occurrences of pair of words (w_{prev}, w) normalized by the number of times word w_{prev} appears before another word.
$$q(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{|\{w' : c(w_{i-1}, w') > 0\}|} \tag{20}$$

However, the raw bigram distribution appears to be impractical to use since many bigrams do not appear in the corpus. Because of that, their probabilities are equal to 0 which leads to problems with computations hence a small value ϵ is added to avoid zero values (however, it is already the simplest form of smoothing called *additive smoothing*).

7.2 Smoothing

The general idea of smoothing distributions is to move part of the probability mass from the common words towards less frequent ones, or towards unseen words, or n-grams. A number of smoothing techniques were introduced over years ranging from the most simple ones like *additive smoothing* towards more sophisticated ones based on interpolation or backoff model [6]. One of the most popular techniques is *Good-Turing smoothing* or derived from it *Katz smoothing*. However, Good-Turing comes with a number of issues as it is based on the estimation of a number of words with certain number of occurrences. The main idea is to move a part of the probability towards unseen words and discount the rest of the probabilities. In this task an issue of predicting unseen words is omitted, this this technique is was not used.

Instead, uniform distribution will be smoothed using distortion coefficient as in [7]. For coefficient $0 < \alpha < 1$ the formula is given as

$$q(w) = \frac{c(w)^\alpha}{Z}. \tag{21}$$

There are various approaches for computing Z but for the simplicity it can be assumed to be 1. However, since q needs to be a distribution to sample from and it will be normalized by PyTorch function anyway Z can be assumed to be a sum of all distorted values $Z = \sum_{w \in V} c(w)^\alpha$, where V is the vocabulary (set of words coming from the data set).

More interesting technique will be used to smooth the bigram distribution since it gives place for more manipulation, as some probability mass can be shifted towards bigrams that do not occur in the data set. For that *Kneser-Ney smoothing* variant presented in [1] was used. Following [5] the discounting constant α will be set $\alpha \in (0, 1)$. Even though modified Kneser-Ney smoothing 5-gram is reported to be pretty effective language model itself [11][4], it is beyond the scope of this project to use it.

$$q_{KN}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - \alpha, 0)}{\sum_{w'} c(w_{i-1}, w')} + \lambda_{w_{i-1}} q_{KN}(w_{i-1}), \tag{22}$$

where normalizing constant λ is given by

$$\lambda_w = \frac{\alpha}{\sum_{w'} c(w, w')} |\{w' : c(w, w') > 0\}|, \tag{23}$$

and the unigram distribution $q_{KN}(w)$ is

$$q_{KN}(w) = \frac{|\{w' : c(w', w) > 0\}|}{|\{(w', w'') : c(w', w'') > 0\}|}. \tag{24}$$

8 Perplexity

To evaluate the model a perplexity measure will be used. This intrinsic evaluation method is formulated as the geometric average of the inverse probability assigned to the words by the model[6]. The intuition behind perplexity is to measure probability of seeing sentences coming from a real world data (test set in this case). The perplexity for a sentence S_N of N words is given as

$$PPL(S_N) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} p(w_i|h_i)}}, \tag{25}$$

where w_i is the i-th word of the sentence S_N and h_i is the history of previous words. Perplexity can be computer for the entire test set or per sentence. In this exercise perplexity will be computed for each sentence separately and the averaged result will be used. Given the fact that performance of the model is already being evaluated with cross entropy it is possible to transform the perplexity formula to save computations. Since cross entropy returns negative log likelihood, cross entropy loss can be used directly to obtain perplexity value ($PPL = PPL(S_N)$ for shortcut):

$$PPL = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} \exp\left(\log(p(w_i))\right)}} = \sqrt[N]{\frac{1}{\exp\left(\sum_{i=1}^{N} \log(p(w_i))\right)}}. \tag{26}$$

Furthermore, it is possible to get rid of N root:

$$PPL^N = \frac{1}{\exp\left(\sum_{i=1}^{N} \log(p(w_i))\right)}. \tag{27}$$

Since $\log(a^N) = N \cdot \log(a)$:

$$N \cdot \log(PPL) = -\log\left(\exp\left(\sum_{i=1}^{N} \log(p(w_i))\right)\right) = -\sum_{i=1}^{N} \log(p(w_i)). \tag{28}$$

Finally

$$PPL = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log(p(w_i))\right), \tag{29}$$

where value in the exponent is simply cross entropy loss for a single sentence.

9 Experiments

In the following section results of training the model with different settings will be discussed. Model was trained for a limited number of epochs since the main goal of the experiments was to study changes in behaviour in reference to parameters and objective functions used in the setting.

9.1 Power divergences

The model with embedding size of 60 and one hidden layer of size 600 was trained using various loss functions. Standard softmax function was used. The model was trained for 15 epochs, optimized using ADAM optimizer with learning rate of 1e-4 and a batch size of 16.

9.1.1 α-divergence experiments

The results of experiments ran comparing the performance of alpha divergence across different parameters were presented on the plots. Figures 1,2,3 and 4 present changes in the learning process as learning curves behaviours change. For example for $\alpha = 1.3$ model seems to be overfitting the training set.

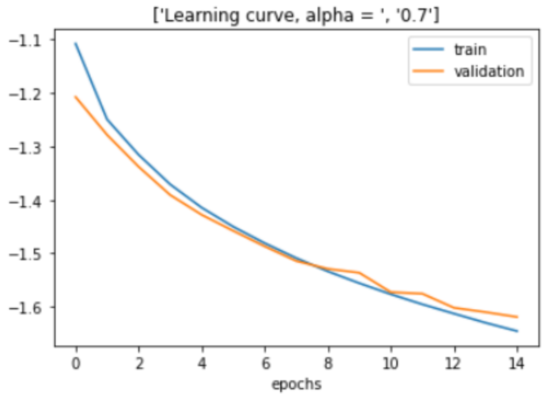

Fig. 1: Learning curve for training with alpha divergence, $\alpha = 0.7$

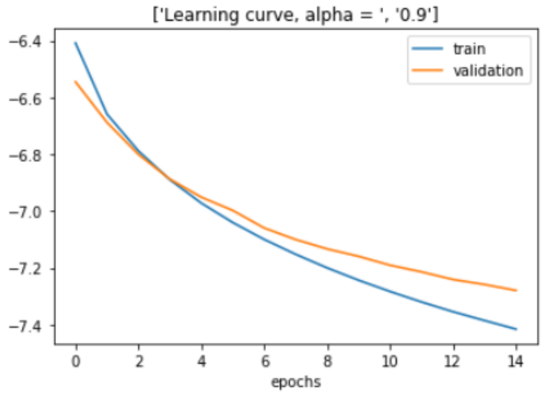

Fig. 2: Learning curve for training with alpha divergence $\alpha = 0.9$

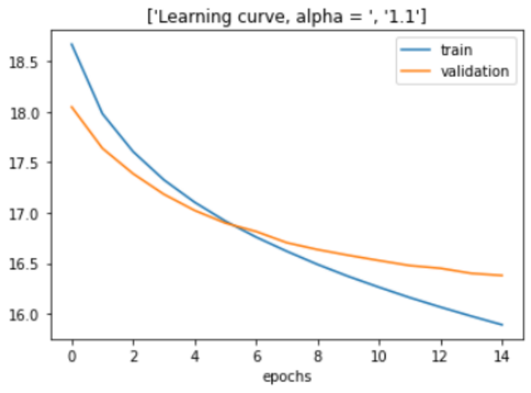

Fig. 3: Learning curve for training with alpha divergence $\alpha = 1.1$

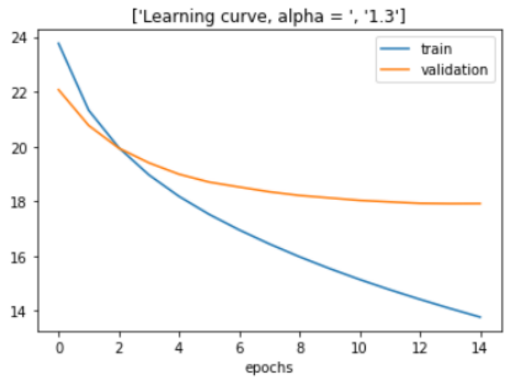

Fig. 4: Learning curve for training with alpha divergence, $\alpha = 1.3$

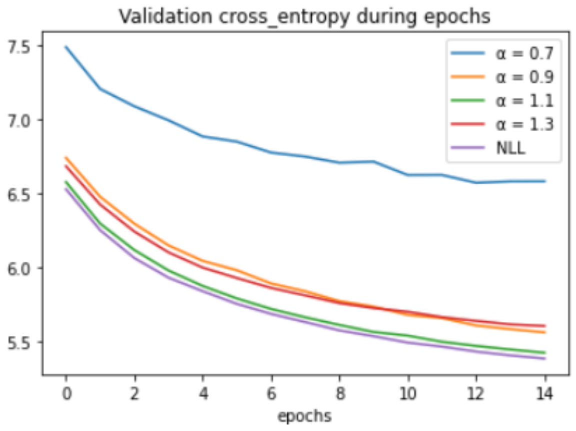

Fig. 5: Validation cross-entropy over epochs for various α values.

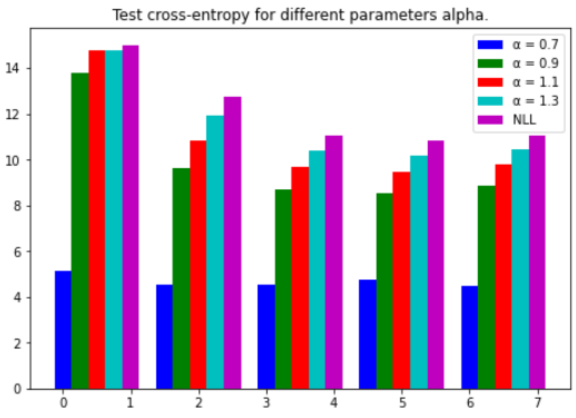

Fig. 6: Validation cross-entropy across frequency bins for various α values.

The validation cross-entropy of models across epochs is as presented on Figure 5. Additionally, the impact of the various model's parameters was showcased across word frequency bins and can be seen on Figure 6. The entire corpus was ranked by frequency and partitioned into 5 bins of equal size; each choice of alpha parameter is compared with the baseline NCE model on its performance across different word frequency groups.

Table 1: The performance after various alpha parameter choices vs NLL on perplexity and cross-entropy, ranked.

Parameter values	Cross-Entropy	Perplexity
$\alpha = 0.7$	6.4066	2204.0598
$\alpha = 0.9$	5.4167	598.6445
$\alpha = 1.1$	5.2794	531.3553
$\alpha = 1.3$	5.4567	491.1907
NCE	5.2411	443.9385

It could be noticed from the results presented in Table 1 that alpha values above 1 offer better results than values below 1. Losses are not directly comparable because different objective functions are used, but it is possible to compare cross-entropy and perplexity and see that for alpha divergences, $\alpha = 0.7$ offers the worst performance on both metrics, with $\alpha = 1.1$ and $\alpha = 1.3$ offering better performance depending on whether minimizing cross-entropy or perplexity is preferred. This intuitively makes sense, as α parameter values above 1 are supposed to favorize more frequent words and thus better conform to the Zipf-like natural distribution of words. However, all alpha divergence models offer worse performance than just using the standard model trained using NLL/NCE.

9.1.2 β-divergence experiments

Analogously to the previous section, but with the beta divergence loss. Figures 7,8,9 and 10 presents learning curves from training the model when beta divergence was used with varying β values. It is visible that for smaller β values model is overfitting the training set.

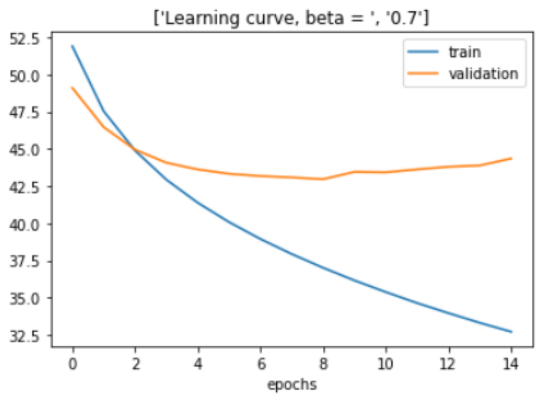

Fig. 7: Learning curve for training with beta divergence, $\beta = 0.7$

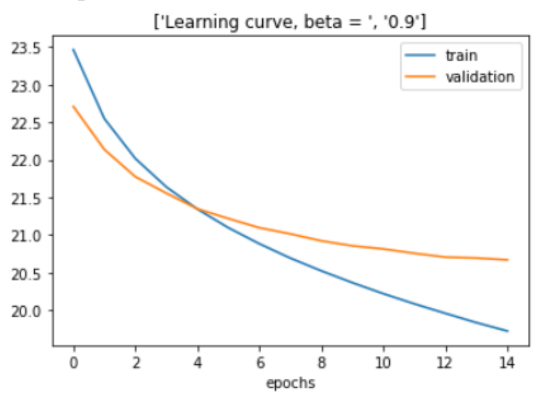

Fig. 8: Learning curve for training with beta divergence, $\beta = 0.9$

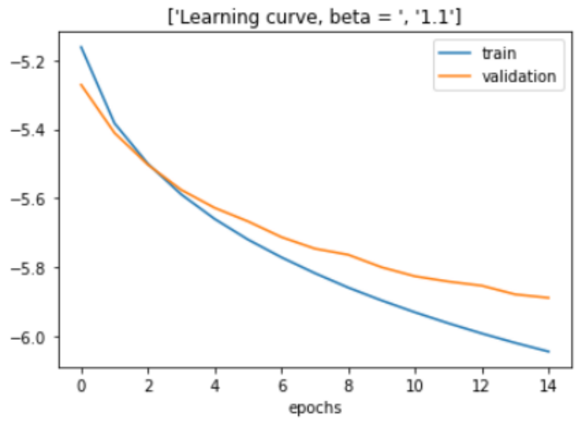

Fig. 9: Learning curve for training with beta divergence, $\beta = 1.1$

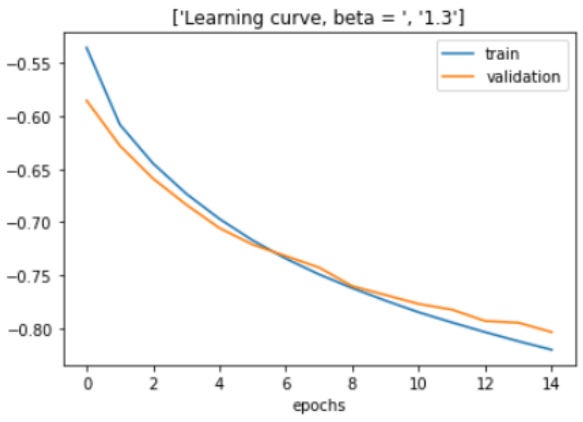

Fig. 10: Learning curve for training with beta divergence, $\beta = 1.3$

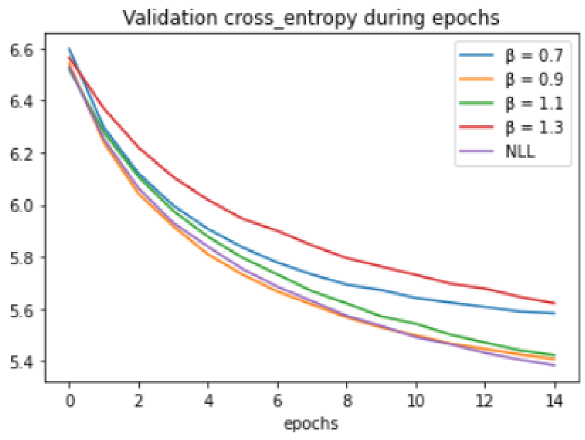

Fig. 11: Validation cross-entropy over epochs for various β values.

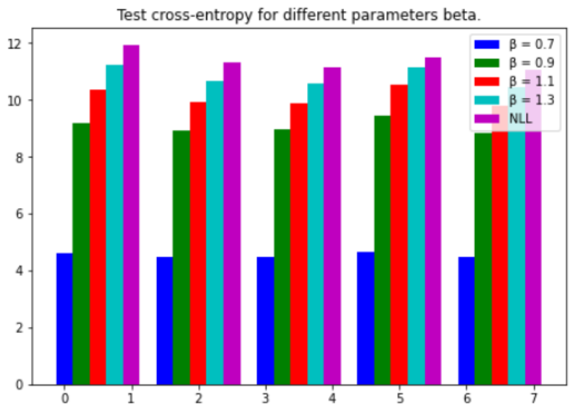

Fig. 12: Validation cross-entropy across frequency bins for various β values.

The validation cross-entropy of models across epochs is as presented on Figure 11. Additionally, the impact of the various model's parameters was showcased across word frequency bins and can be seen on Figure 12.

Table 2: The performance of beta parameter choices vs NLL on perplexity and cross-entropy, ranked.

Parameter values	Cross-Entropy	Perplexity
$\beta = 0.7$	5.4335	729.5884
$\beta = 0.9$	5.2620	507.6068
$\beta = 1.1$	5.2773	440.0366
$\beta = 1.3$	5.4711	470.2264
NCE	5.2411	443.9385

Similarly to the alpha divergence, as it was presented in Table 2, higher beta values achieve better results. $\beta = 0.7$ does not suffer so drastically compared to other parameter settings this time, but it is markedly worse across both metrics.

What's interesting to note is that $\beta = 0.9$, a value below 1, offers the best cross-entropy performance. This could be rationalized by looking at the properties of the beta divergence compared to the alpha - the generalization of the KL divergence used is constructed in a way that values of beta below 1 favorize outliers instead of overfitting as in the case of the alpha. [2]

Again, all beta divergence models offer worse performance than just using the standard model trained using NLL/NCE.

9.2 Approximated softmax

The results of training with approximated softmax function with $k = 1000$ samples were presented on Figure 13. The number of samples equals roughly 10% of the vocabulary size. The function was used together with alpha divergence and various noise distributions. Unigram 0.25 is distorted unigram distribution. Bigram 0.75 is bigram distribution with Kneser-Ney smoothing with discount 0.75.

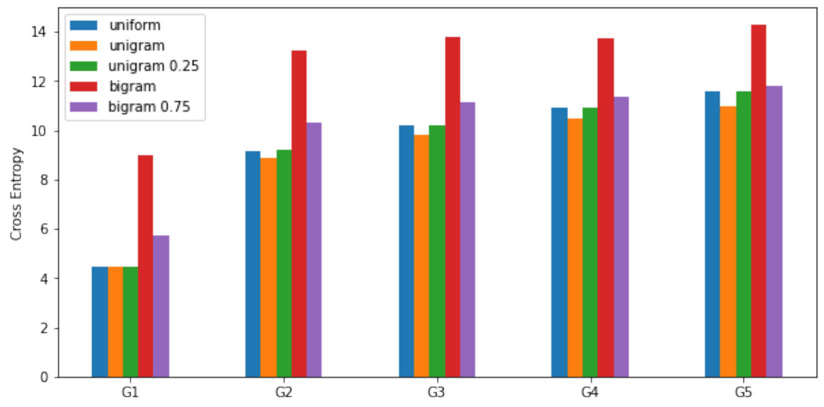

Fig. 13: Cross entropy scores among frequency groups for various noise distributions in approximated softmax.

Even though a function of smoothing is to move distribution mass towards infrequent words it had no or little impact on performance of the model among frequency groups. Furthermore, bigram distribution was expected to perform better however it had lead to worse results. It might be caused by the short training or mistake in the implementation however the least was not found.

10 Conclusions

The exercises included a discussion of theoretical aspects of language modelling together with a review of important topics and techniques.

During the experiments, different combinations of alpha and beta divergence parameters were explored using the standard softmax, and also experiments with importance-sampling derived softmax approximations were performed.

All power divergence models investigated offered worse performance than just using the standard MLE-derived loss function. The impact on various frequency bins was marginal and the results obtained do not allow to judge about effectiveness of these methods. It is possible that these kind of objective functions require longer training. Additionally, further work could be done on adapting the NCE to a power divergence objective and investigating other divergences such as the Renyi divergence.

References

1. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. Computer Speech & Language **13**(4), 359–394 (1999)
2. Cichocki, Andrzej; Amari, S.i.: Families of alpha- beta- and gamma- divergences: Flexible and robust measures of similarities. Entropy vol. 12 iss. 6 **12** (jun 2010). https://doi.org/10.3390/e12061532, libgen.li/file.php?md5=e32fb374f9995e6c6b39132cf0bc730d
3. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
4. Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., Wu, Y.: Exploring the limits of language modeling. arXiv preprint arXiv:1602.02410 (2016)
5. Kneser–Ney smoothing: Kneser–ney smoothing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Kneser-Ney_smoothing, [Online; accessed 21-December-2021]
6. Labeau, M.: Neural language models: Dealing with large vocabularies. Ph.D. thesis, Université Paris-Saclay (ComUE) (2018)
7. Labeau, M., Allauzen, A.: An experimental analysis of noise-contrastive estimation: the noise distribution matters. In: 15th Conference of the European Chapter of the Association for Computational Linguistics:. pp. 15–20 (2017)
8. Labeau, M., Cohen, S.B.: Experimenting with power divergences for language modeling. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 4104–4114 (2019)
9. Ma, Z., Collins, M.: Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. pp. 3698–3707. Association for Computational Linguistics, Brussels, Belgium (Oct-Nov 2018). https://doi.org/10.18653/v1/D18-1405, https://aclanthology.org/D18-1405
10. Mikolov, T.: Ptb dataset (2015), https://github.com/townie/PTB-dataset-from-Tomas-Mikolov-s-webpage
11. Mikolov, T., Deoras, A., Kombrink, S., Burget, L., Cernocky, J.H.: Empirical evaluation and combination of advanced language modeling techniques. In: Interspeech. ISCA (August 2011), https://www.microsoft.com/en-us/research/publication/empirical-evaluation-and-combination-of-advanced-language-modeling-techniques/
12. Powers, D.M.: Applications and explanations of zipf's law. In: New methods in language processing and computational natural language learning (1998)