Adopting and improving LSTMs as Language models

Filippo Daniotti (232087)

University of Trento

filippo.daniotti@studenti.unitn.it

1. Introduction

This document is the final report for the course *Natural Language Understanding*, held at the University of Trento in the A.Y.2021/2022. The goal of the project is to implement a language model based on LSTMs and to evaluate its performances on the word-level Penn Treebank dataset and achieve a perplexity score ≤ 90.7. In this work:

- we analyzed the dataset;
- we implemented and trained a baseline LSTM model and a refined regularized LSTM model;
- we fine-tuned the refined model and improved it with additional regularization techniques from the literature;
- we evaluated the models and performed an in-depth analysis of the best performing model behaviour;
- we created a TUI application to easily interact with the model and use it to generate sentences.

The final model achieved a perplexity score of 79.86 on the test set. The code for this project is available at `https://github.com/filippodaniotti/NLU-UniTN-2022`.

2. Task Formalisation

Language modelling is the task of developing a model which is able to predict which word comes next in a sequence.

A *language model* is a probablity distribution over sequences of words. Suppose we have a sequence of n words modeled as a joint probability: $P(w^{(n)}) = P(w_1, \dots, w_n)$; we can compute this probablity using the *chain rule*, which comes from the Bayes rule: $P(A, B) = P(A|B)P(B)$. Ultimately, a language model computes the probability of a sequence as follows:

$$P(w_1, \dots, w_n) = \prod_{k=1}^{n} P\left(w_k | w_1^{(k-1)}\right) \tag{1}$$

The underlying assumption is that sentences in natural languages have *structure*, hence we can compute the joint distribution from data, i.e. *corpora*. However, this setting is challenged by several properties of natural languages. Although we can trace some general schemes for structure - e.g. word orders like VSO, SVO, *et cetera* - natural languages allow non-standard phrasing; moreover, they allow sentences to grow up to arbitrary lengths. More generally, it is not possible to compute statistics for the sentences space, which has to be considered unbounded in the scope of this task.

To cope with this problem, we can compute statistics for a word considering a small window of preceeding words. Formally, motivated by the Markov Assumption, we think of this process as mapping the original space into a smaller one, for which we can easily compute statistics: these language models are called *n*-gram language models.

	Train	**Validation**	**Test**
Sents split percentage	85.51%	6.85%	7.64%
Words split percentage	85.62%	6.79%	7.59%
# Sentences	42,068	3,370	3,761
# Words	887,521	70,390	78,669
# Vocabulary	9,999	6,021	6,048
# OOV words	-	0	0
Sentence lengths mean	20.09	20.88	20.91
Sentence lengths median	20.0	20.0	20.0
Sentence lengths σ	10.14	9.98	10.18
Sentence max length	82	74	77
Sentence min length	1	1	1
Sentence 25% length	14	13	13
Sentence 50% length	20	20	20
Sentence 75% length	27	28	27

Table 1: *Statistics of word-level Penn Treebank*

Currently, state-of-the-art language models are *Neural language models*, i.e. language models that use neural network models to perform embeddings and approximate the distribution. In particular, the first effective models to be employed where Recurrent Neural Networks, such as LSTMs and GRUs; last developments revolve around attention-based models[1], and specifically Transformers such as BERT[2] and GPT[3][4].

3. Data Description & Analysis

Description The dataset used to train and evaluate the model is word-level Penn Treebank[5], which is a variation of the original Penn Treebank dataset built to be best suited to language modelling settings. Specifically, the original Penn Treebank has undergone a pre-processing pipeline so that it features:

- consistent lowercase;
- tokenization;
- no punctuation;
- proper nouns and other rare words have been encompassed by a `<unk>` token;
- digits have been replaced with a `N` token;
- no OOV words in *validation* and *test* sets with respect to the *training* set.

Statistics analysis In Tab.1 we are presenting a summary of the relevant statistics of the dataset.

The PTB dataset appears to be balanced enough, as sents and words are evenly splitted and the mean and standard deviance σ of the words-per-sentence distribution are very similar, being around 20 and 10.

Moreover, since there are no OOV words, we can rely on the train split to construct the vocabulary. Its length is given by the 9999 words, to which we added two additional tokens:

- `<pad>` is a pad token to pad shorter sequences to the length of the longest sequence in a given batch;
- `<eos>` is a sentence terminator token.

The analysis of the most frequent words in the train split reveals that the most frequent words are extremely common function words, like articles (*the*, 5.7%), or prepositions (*of*, 2.7%), which is unsurprising. We are not surprised to observe that the two wrappers `<unk>` and `N` show high counts, being 5.1% and 3.7% respectively, ranking second and third most frequent words. Ultimately, the first 10 most frequent words make up almost 30% of the total word count.

4. Model

4.1. Architecture

We employed LSTMs[6] as backbone architecture, as they outperform Vanilla RNNs and GRUs in caputirng long-term dependencies, which is a key feature for language modelling. The model features the following components:

- a *word embedding* layer, which maps the input words to a sparse vector representation;
- a *recurrent LSTM* layer, which processes the sequence of embeddings and outputs a sequence of hidden states;
- a *fully-connected* layer, which maps the hidden states to a sequence of logits.

We have implemented two different architectures:

- *Baseline LSTM*: baseline model with a single-layer LSTM with 300 hidden units;
- *Merity LSTM*: refined architecture featuring a set of regularization technique, which will be discussed later.

4.2. Pipeline

The complete experiment pipeline is structured as follows.

1. *Data loading*: the dataset is downloaded, extracted and loaded into memory.
2. *Data pre-processing*: in order to be ready to be fed to the model, the following operations are performed:
    - each sentence is tokenized by splitting on whitespaces;
    - a `<eos>` token is appended to each sentence;
    - the vocabulary mappings are created.
3. *Data batching and collation*: the dataset is padded to the length of the longest sequence in the batch before the batch is collated to be fed to the model. Additionally, depending on the experiment configuration the collate function can also perform the following operations:
    - it can split sentences into chunks of a fixed length to implement TBPTT;
    - it can partially shuffle sentences.
4. *Model training*: the model is trained on the training set; we will also perform a validation epoch on the validation set in order to monitor the model learning behaviour and to tune the hyper-parameters.
5. *Model evaluation*: the model is evaluated on the test set according to the evaluation metrics and pipeline (see Sec.5).

4.3. Regularization

Sequence models are prone to overfitting, as they are trained on long sequences and they have to capture long-term dependencies. To cope with this problem, we employed the following regularization techniques suggested by [7]:

- *Variational Dropout*[8]: it samples a binary mask that drops the same units across time-steps, to ensure temporal consistency;
- *Embedding Dropout*[8]: it drops some connections in the embedding layer; it is equivalent to dropping some words from the vocabulary, in order to prevent the model from relying too much on specific words;
- *DropConnect*[9]: it shuts down some connections in the hidden-to-hidden weight matrices by setting the weights of the target connections to 0, to control overfitting on the recurrent connections;
- *Weights Initialization*: initializing the weights can help the model to converge faster; specifically, we are initializing the LSTM layers according to the *Xavier Uniform* distribution[10] and the sparse embedding from a *Uniform* distribution in the range $[-0.1, 0.1]$;
- *Weight Tying*[11]: we tye the weights of the embedding layer and the output layer, in order to reduce model parameters;
- *Gradient Clipping*: we clip the gradients to a maximum norm, in order to prevent gradient explosion;
- *Partial Shuffle*[12]: each sentence in a batch is rotated by a random offset, in order to improve generalization;

TBPTT Additionally, we used *Truncated Backpropagation Through Time* (TBPTT) to train the model. As suggested in [7], we are using $k_1 = k_2 = k$, i.e. we are splitting the sentences into chunks of a fixed length and backpropagating the error only through the chunks, allowing for a more efficient usage of training samples. Specifically, the split step k is sampled from a Gaussian $\mathcal{N}(\text{seq}, \sigma^2)$ with probability $p = 0.95$ and from $\mathcal{N}(\frac{\text{seq}}{2}, \sigma^2)$ with probability $1 - p$. However, we are using 30 in place of 70 as value for seq. The motivation is to be found in the dataset statistics. Recalling Tab.1, we know that the mean sentence length is around 20 and the length at the 75% quantile is 27, which means that the majority of the sentences are shorter than 30 words and would not undergo any processing if 70 was the mean of the Gaussian.

Optimization We used mostly *SGD* and its variation *NT-ASGD*[7] as optimizers. We also performed some experiments with *Adam*, but we did not observe any improvement in the model performance. We also employ a *Learning Rate Scheduler* which reduces the learning rate by a factor of 0.5 when the validation loss does not improve for 3 epochs.

4.4. Experiments

Unless otherwise specified, we used a batch size of 128 and we trained the models for 50 epochs and used SGD with *learning rate* $= 1$ and *weight decay* $= 1 \cdot 10^{-6}$. All the experiments were conducted on a NVIDIA GeForce RTX 3050 GPU with 4GB of VRAM.

Baseline We perfomerd a first experiment with the default configuration to set a baseline for our study. Then, we performed a second experiment were we simply add a *Dropout* layer with $p = 0.5$ after the LSTM layer.

Merity LSTM This is our implementation of the AWD-LSTM model proposed in [7]. We started by adding the dropout and regularization techniques individually and then combining them in different configurations, in order to understand their contribution and keep only the best performing ones, in a greedy fashion. Subsequently, motivated by the fact that NNs benefit from growing in depth in many tasks and that the aforementioned techniques proved to be able to present overfitting, we increased the size of the model by increasing the sizes of both the embedding and the hidden representation. Lastly, we picked the best performing model and we performed an extensive training experiment, where we let it train until the validation loss would stop decreasing.

5. Evaluation

5.1. Metrics

In this work we are framing the learning problem as a multi-class classification problem, where the classes to predict are the words in the vocabulary. Hence, we are training the model to minimize the cross-entropy (CE) loss:

$$\text{CE}\big(f(x;\theta), y\big) = -\sum_{i=1}^{N} y_i \log f(x_i;\theta) \tag{2}$$

The main metric we use to compare and evaluate the models is the *perplexity* (PP). The reason is that PP is a well understood and established metric and it correlates well enough with the model performances on real-world tasks. Although the PP can be defined in multiple ways, it is convenient to use the definition of PP as the exponential of the cross-entropy (CE):

$$\text{PP}(f(x;\theta), y) = \exp\big(CE(f(x;\theta), y)\big) \tag{3}$$

While the CE gives a measure of how much the model is uncertain about the prediction, the PP can be interpreted as a measure of how many classes the model is considering to make a prediction (*weighted branching factor*). The lower the PP, the less uncertain the model is on which word to predict, the better it is performing.

Additionally, we are further evaluating the best-performing model to excerpt deeper insights in its behaviour. Specifically, we are considering *average PP per sequence length*, *per word predicted vs. target counts difference* and *F1 score*. Lastly, we are giving a qualitative evaluation of the model by showing some examples of generated sequences.

5.2. Results

We evaluated every model by performing an evaluation epoch on the *validation* and *test* set with batch size 1. The results of these evaluation runs are summarized in Tab.2. It is worth noting that we decided to scrap *Weight Dropout* on the hidden-to-hidden weight matrices of the LSTM layers. The reason is that the the model with the three dropout techniques (`merity_ad`) was achieving worse results than the model with only *Variational Dropout* and *Embedding Dropout* (`merity_eld`), as it can be seen in Fig.1.

Model	Validation	Test
`baseline_dropout`	116.11	113.44
`merity_ad`	95.36	91.29
`merity_ad_nohh`	95.36	91.29
`merity_ad_nohh_tbptt`	90.38	87.46
`merity_ad_nohh_1024`	85.85	82.74
`merity_ad_nohh_1024_ps`	87.40	83.18
`merity_ad_nohh_1024_ps_long`	83.01	79.86

Table 2: *Results (PP) of the experiments.*

Hence, we performed further experiments on the `merity_wd` model. We found out that `merity_wd_nohh` was achieveng better results on the validation set, as in Fig2. This behaviour is probably caused by the fact that dropping out the hidden-to-hidden weights prevents the model from effectively learn long-term dependencies in the sequences. In the end, we decided to keep the `merity_ad_nohh` model as it had the best performances and still little overfitting.

For similar reason we decided to discard TBPTT as training algorithm, as it was not improving the performances (Fig.3), and stick with unconstrained BPTT. On the other end, the models with increased embedding and hidden sizes achieved better results (Fig.3); we also reduced the number of LSTM layers from 3 to 2 and increased the probabilities of the dropout techniques to account for the increased model size. Similar considerations hold for the usage of *Partial Shuffle*, which ensures less overfitting at the expensive of slightly higher PPs (Fig.3); the reason is that partially shuffling the sentences increases variety in the training data, acting as a sort of data augmentation technique. Lastly, we performed an extensive training experiment on the `merity_ad_nohh_1024_ps` model, where we let it train until the validation loss would stop decreasing at epoch 96; this final and best performing model model scored 83.01 PP on the validation set and 79.86 on the test set.

5.3. Analysis

This section is dedicated to the analysis of the best performing model, `merity_ad_nohh_1024_ps_long`. We start by providing some examples of generated sequences[1], in the format: `<prompt>` (<temperature >) → <inferred >:

- `the` (1.0) → the major `<unk>` million foreign equipment received a $ N million offer for federated and packaged goods companies;
- `the price` (0.8) → the price for the giant company is about $ N million;
- `the price` (1.0) → the price advanced increase in the second quarter in workstations prompting the dollar to a steep rise in the week 's sharp climb in the imports growth of the;

We can see that the model is able to generate sequences that are roughly syntactically correct. However, the generated sequences are not semantically solid. This is probably due to the fact that the model is not able to learn the meaning of the words, but only their distribution in the sequences. Moreover, it seems that the model has consistent performances on shorter sequences and that it struggles more on longer sequences. This is

[1] In order to try and use the model, we recommend to use the TUI application in the repository attached to this work.

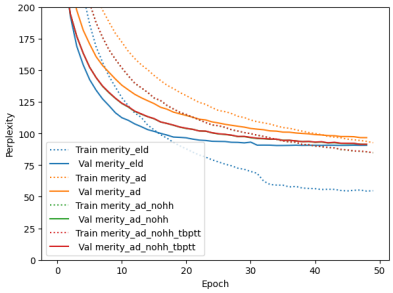

Figure 1: *Train and valid PP of the models with combined dropout techniques.*

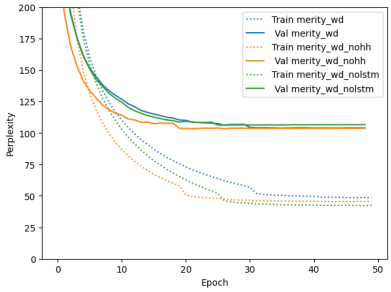

Figure 2: *Ablation study on* `merity_wd`.

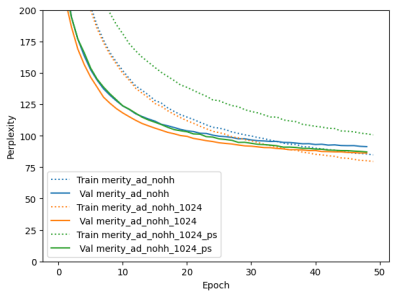

Figure 3: *Impact of larger model size and partial shuffle.*

also visible on the average PP per sequence length, as in Fig.4: sentences of length shorter than 40 tokens cause the model to score $60 \leq PP \leq 100$, slightly increasing as the length decreases. On the other hand, sentences of length longer than 40 tokens have PP scores as high as 160 and as low as 20.

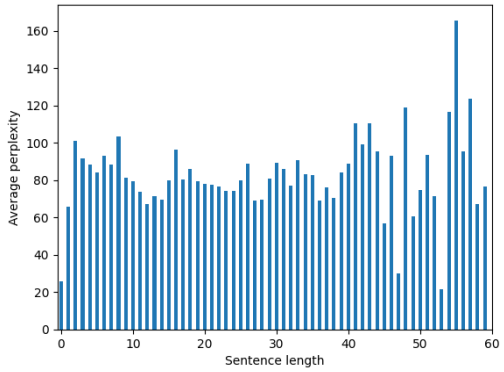

Figure 4: *Average PP per sequence length.*

If we inspect the counts of predicted and target words in Tab.3, we can see that the model heavily relies on the most present words in the dataset: the 5 most guessed words make up over 50% of the total predictions, while the 5 most present words in the target sequences make up only 20% of the total target words. All of the 10 most guessed words are over-represented in the predictions, except `and` - roughly on par - and `in` - which shows a slightly negative difference. Quite interestingly, the fact that the `<eos>` token is over-represented may indicate that the model has not learned when it is most appropriate to terminate a sentence.

Word	**Targets**	**Preds**	**Diff**
<unk >	4606 (5.85%)	15026 (19.10%)	10420
the	3968 (5.04%)	12405 (15.77%)	8437
<eos>	3761 (4.78%)	9125 (11.60%)	5364
N	2494 (3.17%)	3874 (4.92%)	1380
of	2182 (2.77%)	3236 (4.11%)	1054
to	2024 (2.57%)	2878 (3.66%)	854
a	1739 (2.21%)	2315 (2.94%)	576
and	1471 (1.87%)	1480 (1.88%)	9
in	1470 (1.87%)	1366 (1.74%)	-104
's	903 (1.15%)	1341 (1.70%)	438

Table 3: *Per word predicted vs. target counts difference.*

Finally, we can get some interesting insights by looking at the f1-score in Tab.4. Quite expectedly, given the huge size of the targets space and the nature of the task of language modelling, most of the classes have a rather low f1-score; this is particularly noticeable in the most common words, like `<unk>` (0.23) and `the` (0.34). However, it is interesting to notate than some very specific words score higher values. If we take a look at the words with highest f1-score with support ≥ 100, we can see interesting examples like `million` (0.68) and `N` (0.62), which may indicate that the model was effective in learning where to place numbers and to phrase them together. Scores can get even higher if we reduce the support threshold to ≥ 10: words that may appear in very specific phrasings show pretty interesting scores (`angeles` (0.95), `share` (0.8), `estate` (0.79)).

support ≥ 100	**support** ≥ 10	**full support**
hutton 1.0	share 0.8	<eos >0.4
kong 1.0	million 0.68	on 0.22
lehman 0.95	N 0.62	<unk >0.23
angeles 0.95	than 0.55	N 0.62
lynch 0.89	co. 0.54	the 0.34
officer 0.84	billion 0.51	$ 0.45
share 0.8	year 0.51	as 0.31
estate 0.79	president 0.51	a 0.23
loan 0.79	n't 0.49	is 0.17
cents 0.78	of 0.46	of 0.46

Table 4: *predicted words ranked by F1 score with various filters on support.*

6. Conclusions

In this study we demonstrated that LSTM-based language models can benefit greatly from the usage of dropout techniques, as well as from an increased model size. We also showed that partially shuffling the training sentences can help in reducing overfitting, at the expense of slightly higher perplexity. In addition, we inspected the model behaviour and discovered that the best performing model still heavily relies on the most frequent token in the dataset, and it has inconsistent behaviour with longer sentences. We also showed that the model is effective in learning to compose words so that they comply to common phrasings such as numbers. Furthermore, the model is able to generate syntactically correct and somewhat meaningful sentences, as long as they are not too long. Finally, we created a TUI application to easily interact with the model and generate sentences.

7. References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[5] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, jun 1993.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," 2017.

[8] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," 2016.

[9] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. [Online]. Available: https://proceedings.mlr.press/v28/wan13.html

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[11] O. Press and L. Wolf, "Using the output embedding to improve language models," 2017.

[12] O. Press, "Partially shuffling the training data to improve language models," 2019.