Language Modeling with Penn TreeBank Dataset

A deep learning approach

Federico Izzo (229316)

University of Trento

federico.izzo@studenti.unitn.it

1. Introduction

This document illustrates the model created and tested for the language modeling task in order to satisfy the requirements for the *Natural Language Understanding (NLU)* course project. In particular the main objectives of the assignment are:

1. implement a *State-of-The-Art (SOTA) RNN*[1] architecture in any python framework;
2. obtain a baseline score of 144 *PP* for a vanilla RNN or 90.7 *PP* for a vanilla LSTM using the *Penn Treebank (PTB)* dataset[2];
3. try to improve the provided score making some hyperparametrization or implementing some other *Deep Learning (DL)* solutions.

Along with the report, a public available GitHub repository is provided[1].

The best performing model obtained a perplexity of 86.64 PP.

2. Task formalization

In the field of *NLU* the *Language modeling (LM)* task has the goal of predicting a word given the previous sequence of words (input sentence). More formally, the goal is to learn a model M that approximates the conditional probability P of token t_i given $t_0, ..., t_{i-1}$ as much as possible:

$$P(t_i|t_1, \ldots, t_{i-1}) \approx M(t_i|t_1, \ldots, t_{i-1}) \quad (1)$$

Language models are employed in a variety of domains, such as speech recognition, machine translation, response generation, and captioning. Often the LM task is solved using DL techniques that aim to find a minimum for the non-linear function representing the probability function. In the origin, DL solutions were mainly based of RNN in order to allow a variable length input. This brings to very deep structures that suffer of vanishing gradient. To overcome this phenomenon a new kind of networks like LSTM[3] and GRU[4] were proposed.

Despite the current SOTA for LM uses transformer-based models[5], this work focuses on LSTM-based architectures as specified in the assignment.

[1] https://github.com/fedeizzo/languageModelling

3. Data Description Analysis

As already discussed in section 1, Penn Treebank is the dataset used for the model train and evaluation. It is a common benchmark for LM maintained by the university of Pennsylvania, it is characterized by over four millions and eight hundred thousand annotated words in it, all of them corrected by humans. There are different kind of annotations inside the dataset, such as:

- piece-of-speech: labels associated to tokens used for text analysis;
- syntactic skeletons: structure associated to a sentence useful for training or evaluation purposes;
- semantic skeletons: as for syntactic skeleton, a structure useful for the LM task learning.

The dataset is composed of a total of 49199 sentences and 1036580 words, for a vocabulary size of 10000 tokens. The dataset is divided into three splits: train, validation, and test.

The split ratio between the three parts is shown in table 1.

Table 1: *Count of words and sentences and split ratio into training, validation, and testing sets in the Penn Treebank dataset.*

	Sentences	Words	Sent. split	Words split
Train	42068	887521	85.50 %	85.62 %
Val	3370	70390	6.84 %	6.79 %
Test	3761	78669	7.64 %	7.58 %

Luckily, validation and test sets do not contain *out-of-vocabulary (OOV) words*, this simplifies the embedding management during the problem formulation.

The 5 most frequent words are presented in table 2 (words are shared across all three splits):

Table 2: *Occurences of most frequent words in the Penn Treebank dataset.*

	Word	Count	% over total
1	The	59421	5.7
2	<unk>	53299	5.1
3	N	37607	3.6
4	of	28427	2.7
5	to	27430	2.6

The presence of <*unk*> and *N* tokens can be easily explained because they respectively represent unknown elements and digits.

Figure 1 shows the sentence length distributions for all splits.

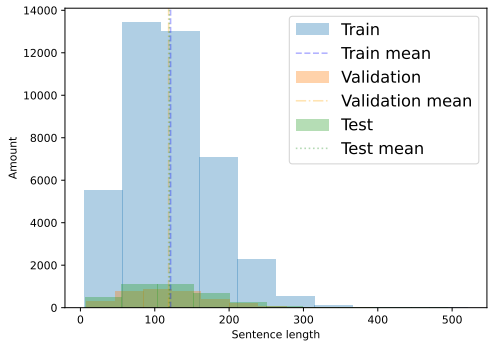

Figure 1: *Length distribution of all sentences in the training, validation, and testing sets of the Penn Treebank dataset.*

The mean values are almost identical, this implies that the input length of the RNN is constant over the training, validation, and testing phases: consequentially, any operation applied on top of the train split should behave in a similar way on the others.

4. Model

The following section explains the adopted pipeline for the dataset creation and model formulations.

4.1. Pipeline

In order to train a RNN model some dataset manipulation is required. The implemented steps are listed below:

1. the original dataset is loaded from file;
2. a <*EOS*> token is append to each sentence to declare the end of sentence;
3. each unique word is mapped to an integer number;
4. a custom collate function applies a common pad value, ignored during the loss computation, in order to have sentences with equal length.

4.2. Architecture

The baseline model is a plain LSTM structure where the forward pass is divided into a sequence of steps:

- a list of integers, representing the words in a sentence, is passed to the model as the input;
- each word is mapped to a vector space using a learnable embedding layer;
- the embedded input, which represents elements from t_0 to t_{i-1}, is then used to predict t_i by the recurrent structure;
- finally, the output of the LSTM is fed to a fully connected layer that returns the class probability for each word.

Once the required PP value was reached, a Mogrifier LSTM architecture[6] was tested to boost performance. This is a enhanced version of a canonical LSTM in which the hidden element of the step t_{i-1} is used as a gate for the input of step t.

4.3. Overfitting

From the first run it was clear that the model suffered of overfitting (figure 2). For this reason, an incremental approach was used to add several well known techniques for overfitting avoidance[7]:

- *Learning rate scheduler*: to control the impact of a single train update by changing the learning rate dynamically, in particular a *ReduceLROnPlateau* scheduler was used, to reduce the learning rate if no improvement in the validation loss is received within a user-defined patience;
- *Early Stopper*: to stop the training when the validation loss starts to increase;
- *Weights initialization*: hidden layers initialization before training;
- *Parameter Tying*: to reduce the model complexity and finding a common representation by aggregating the embedding and classification layers;
- *Embedding dropout*: a modified version of an embedding layer that includes dropout[2];
- *Weight dropout*: a dropout applied on the weight of an LSTM model[3];
- *Locked dropout*: a layer that allows to shutdown neurons in a consistent way across repeated connections within the forward and backward pass;
- *Gradient clipping*: to avoid *"exploding gradient"*.

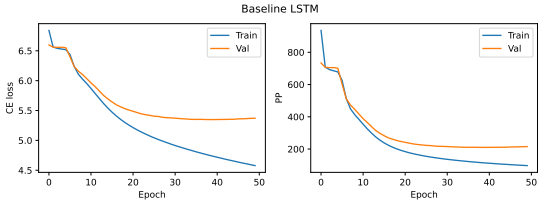

Figure 2: *Loss w.r.t. epochs (left) and PP w.r.t epochs (right) obtained using the baseline LSTM model described in section 4.2.*

[2] embeddig dropout source
[3] weight dropout source

4.4. Optimizer

The three optimizers tested, and associated impacts, are presented below:

- *Stochastic Gradient Descent*: after many epochs it stagnates;
- *Non-monotonically Triggered ASGD*: optimized version of SGD capable of taking mean values from SGD to reduce noise. It gives a solution closer to the optimum;
- *ADAM*: reaches better result than SGD and ASGD in less time when used in combination with Mogrifier LSTM.

Moreover, several tests were made also using *Truncated Back-Propagation Through Time (TBPTT)*[8].

5. Evaluation

This section contains metrics used for the evaluation phase and explains different experiments.

5.1. Metrics

The task was addressed as a classification problem where the output of the model is a vector and each cell represents the probability of the i-th word. The *Cross Entropy (CE)* was the objective function used to learn parameters of the model

$$CE(x,y) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} log f_\theta(x_{ij}) \tag{2}$$

where:

- x, y represents inputs and labels for the model;
- N is the number of elements in the batch;
- C is the total number of classes.

Moreover, *Per word Perplexity (PP)* was used as an additional metric to assess model performances. It is defined on top of the CE loss:

$$PP(x,y) = e^{CE(\{x,y\})} \tag{3}$$

The final goal is to find a set of parameters that minimizes the PP value:

$$\theta^* = \text{argmin}_\theta PP(X,Y) \tag{4}$$

5.2. Results

The first idea was to create a baseline LSTM model that can be used later to make comparisons with enhanced implementations. No technique was used to avoid overfitting, and as expected performance on the train split is higher than the one on validation split (figure 2).

The second experiment focused on regularization techniques presented in the section 4.3. Different combinations have been tested and at the end the best result, presented in figure 3, was obtained using all regularization tools except weight dropout. Even if the performance are better with respect to the baseline, after the 50-th epoch the validation loss stops decreasing while the training one keeps improving.

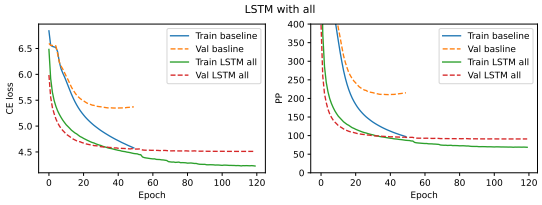

Figure 3: *Loss w.r.t. epochs (left) and PP w.r.t epochs (right) obtained using the LSTM model enhanced with regularization techniques reported in section 4.3.*

The depth and size of the model has been increased in order to increase the model capacity and hopefully to reduce the problem. However, a side effect of this update was the memorization of training data directly within the model parameters. Then, both the dropout and the starting learning rate values has been increased. The former to reduce the overfitting, the latter to counterbalance the slow down of the learning. Unfortunately, from figure 4 it is possible to notice that overfitting is still present.

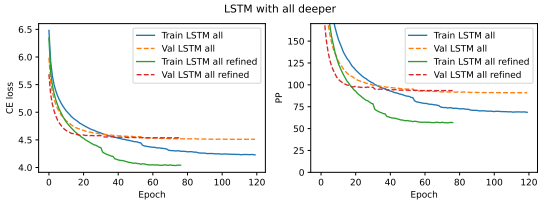

Figure 4: *Loss w.r.t. epochs (left) and PP w.r.t epochs (right) obtained using the deeper LSTM model enhanced with regularization techniques reported in section 4.3.*

At this point the most effective regularization techniques were used and further tests were made on top of the Mogrifier architecture with Adam as optimizer.

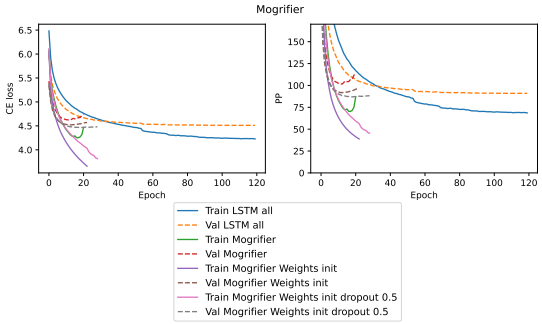

Figure 5: *Loss w.r.t. epochs (left) and PP w.r.t epochs (right) obtained using the Mogrifier LSTM model with regularization techniques reported in section 4.3.*

Figure 5 shows how this model structure is capable of learn-

ing better and in a faster way but at the same time exhibits poor generalization capabilities.

As a matter of fact, the Mogrifier architecture can learn a good representation, capable of obtaining low perplexity values. However the overfitting phenomenon remains the main problem, even when applying regularization techniques.

In order to compare each tested model according to the PP value on the validation split of Penn Treebank dataset, the following table is presented:

Table 3: *Comparison among each tested model based on PP value results obtained on validation and test splits of Penn Treebank dataset.*

Model	Val PP value	Test PP value
Baseline LSTM	227.56	235.66
LSTM + regularization	93.62	93.22
Deeper LSTM + reg.	92.33	93.98
Mogrifier LSTM	85.77	86.64

As shown, the model with the lowest PP value is the one with the Mogrifier structure, but it is important to underline that this metric does not take into consideration the overfitting behavior. For this reason, it was chosen to use the `LSTM + regularization` model to make further analysis (section 5.3), since it is the model that best align the train and validation splits.

5.3. Predictions analysis

A more in depth analysis of the predicted words of the test split was made with respect to the `LSTM + regularization` model. An element of interest was the correlation between the effectiveness of the model and the length of the sentence. Figure 6 shows a stable behavior across words at different position in the sentence, except ones around the 60-th one. It is well known that sequence DL models fed with very long input are not able to capture all the context due to the limited size of the vector used for latent space encoding.

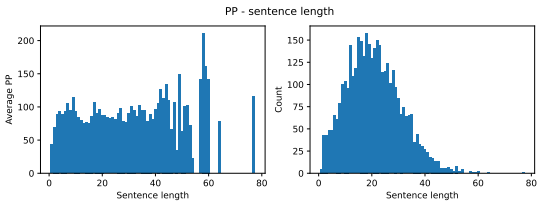

Figure 6: *PP values w.r.t. sentence lengths (left) histograms of sentence lengths (right).*

Although, in this specific case, it could be a consequence of the low occurrence of words around that specific position as shown in the right subplot of figure 6.

Secondly, from table 4 it would seem that number of correctly predicted words is correlated with total number of word occurrences.

Table 4: *Words ranked by the correclty predicted number of times.*

Word	Correctly predicted	Total occurrences
the	2761	3968
<eos>	2671	3761
<unk>	2214	4606
N	1757	2494
of	1402	2182
to	1100	2024
a	478	1739
...	...	...
accounts	1	8
account	1	10
acceptances	1	1
acceptance	1	1
abortions	1	4
1990s	1	3
13th	1	9

However, the highest numbers of correctly predicted words are associated to the most frequent tokens in the dataset (table 1). For this reason, in order to validate the previous statement, an additional experiment was made taking into account the accuracy of each word. From table 5 it is clear that words with highest accuracy are not necessarily the most frequent ones.

Table 5: *Words ranked by greatest accuracy.*

Word	Accuracy	Total occurrences
jones	96%	23
officer	94%	36
york	90%	71
'm	90%	10
breakers	88%	8
mac	86%	7
lynch	84%	19
be	84%	384

6. Conclusion

In this work, two LSTM architectures have been implemented for solving the Language Modeling task, and they have been tested on the popular Penn Treebank dataset.

After many experiments the Mogrifier model has obtained the lowest PP value (86.64) on the test split. The qualitative carried out analysis has shown that a common pattern for error estimation has not be found.

Possible future works may be a more in-depth investigation on the human perceived quality and more experiments made to address the overfitting problem associated with Mogrifier LSTM.

7. References

[1] T. Mikolov, M. Karafiát, L. Burget, Jan, H. . Černocký, and S. Khudanpur, "Recurrent neural network based language model." in *In INTERSPEECH 2010,*, pp. 1045–1048.

[2] 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.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[6] G. Melis, T. Kočiský, and P. Blunsom, "Mogrifier lstm," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJe5P6EYvS

[7] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," 2017. [Online]. Available: https://arxiv.org/abs/1708.02182

[8] C. Tallec and Y. Ollivier, "Unbiasing truncated backpropagation through time," 2017. [Online]. Available: https://arxiv.org/abs/1705.08209