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 aims to summarize the work done for the *Natural Language Understanding (NLU)* course project, in particular this required the creation and testing of a model for the language modeling task. The main objectives are the following:

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 hyper-parametrization 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:

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

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 phenomena a new kind of networks like LSTM[3] and GRU[4] were proposed.

The current SOTA for LM is based on transformers[5].

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

3. Data Description Analysis

As already introduced in section 1 the dataset used is Peen Treebank, a common benchmark for language modeling.

Penn Treebank (PTB) is a dataset maintained by the university of Pennsylvania, there are over four millions and eight hundred thousand annotated words in it, 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 that can be used for training or evaluation purposes;
- semantic skeletons: as for syntactic skeleton, a structure useful for the LM task learning.

The dataset is composed by 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: *Dataset split.*

	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: *Most frequent words.*

	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* can be easily explained because they respectively represent unknown elements and digits.

Figure 1 shows the sentence length distributions for all splits.

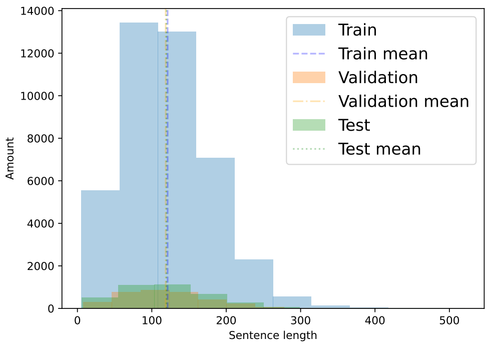

Figure 1: *Sentence lengths distribution.*

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. in order to have sentences with equal length a custom collate function is defined, it applies a common pad value ignored during the loss computation.

4.2. Architecture

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

- a list of integers representing the words in a sentence is passed as input to the model;
- each word is mapped to a vector space using a learnable embedding layer;
- the embedded input representing elements from t_0 to t_{i-1} to predict t_i is then used by the recurrent structure that takes elements;
- finally, the output of the LSTM is fed to a fully connect layer that gives 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 layer;
- *Embedding dropout*: a modified version of an embedding layer that includes dropout[2];
- *Weight dropout*: a dropout applied on the weight of a 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"*.

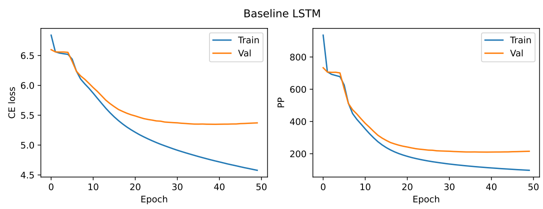

Figure 2: *Baseline overfitting.*

4.4. Optimizer

Three different optimizers were tested:

- *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. Gives a solution closer to the optimum;

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

- *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.

An additional metric was used to assess model performances, it is the *Per word Perplexity (PP)* 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:

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

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 result with respect to the baseline, after the 50-th epoch the validation loss stops decreasing while the training one keeps improving.

At first, I thought the problem was related to low model capacity, for this reason I decided to increase the depth and the size of the model. A side effect of this update was the memorization of training data directly within the model parameters. In order to overcome this consequence my next step was

- the increase of the dropout value to avoid the new overfitting effect;

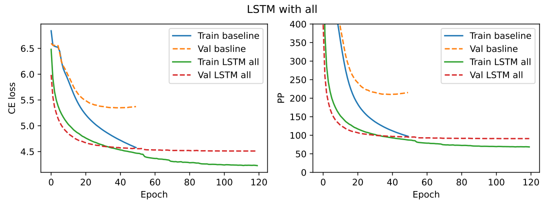

Figure 3: *LSTM with regularization.*

- the increase starting learning rate which was controlled by the learning rate scheduler.

Those updates aimed to align more the validation curve to the training one. Unfortunately, from figure 4 it is possible to notice that the overfitting phenomenon was reduced but was still present.

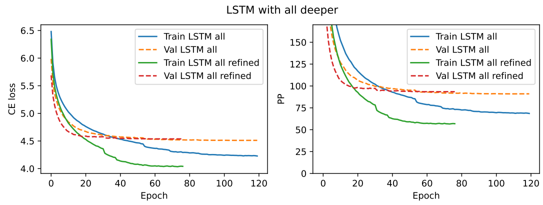

Figure 4: *LSTM all refined.*

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

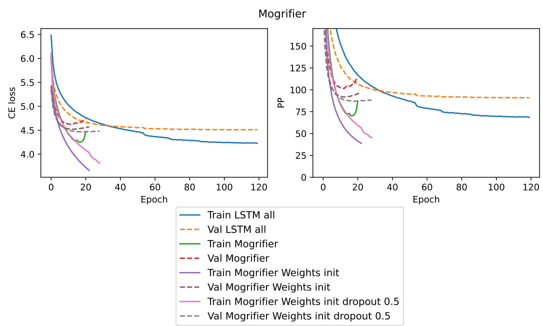

Figure 5: *Mogrifier.*

Figure 5 shows how this model structure is capable of learning better and in a faster way but at the same time exhibits poor generalization capabilities.

Technically speaking, the Mogrifier architecture has all elements to allow learning a good representation, capable of obtaining low perplexity values, however the overfitting phenomenon remains the main problem, even when applying regularization techniques.

5.3. Predictions analysis

A more in depth analysis of the predicted words of the test split was made with respect to the LSTM model presented in figure 3. An element of interest was the correlation between the effectiveness of the model and the length of the sentence. It was proved that sequence models implemented using DL tends to decrease the perplexity value with long sequences. Figure 6 shows a stable behavior across words at different position in the sentence, except ones around the 60-th one.

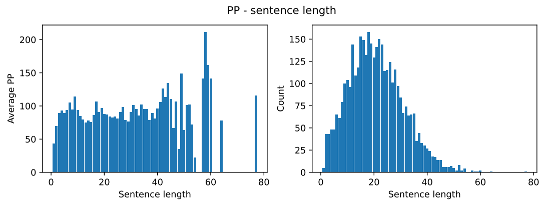

Figure 6: *PP and sentence lengths correlation.*

I think that this is not correlated to the problem previously discussed, but instead it could be a consequence of the low presence of words around that specific position as shown in the right subplot.

Table 3: *Most correct words.*

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
's	434	903
in	353	1470

From table 3 and 4 it seems that number of correctly predicted words is correlated with total number of word occurrences. This evidence is strengthened by the fact that most occurred words are shared across dataset splits and consequentially an overfitting behavior results in good performance also in the testing phase.

Table 4: *Least correct words.*

Word	Correctly predicted	Total occurrences
acquisition	1	16
acquire	1	11
accounts	1	8
account	1	10
acceptances	1	1
acceptance	1	1
abortions	1	4
1990s	1	3
13th	1	9

To finally asses this hypothesis, a final experiment was made taking into account the accuracy. From table 5 it is clear that words with highest accuracy are not necessarily the most frequent ones.

Table 5: *Words with greatest accuracy excluding low occurrences.*

Word	Accuracy	Total occurrences
jones	0.96%	23
officer	0.94%	36
york	0.90%	71
'm	0.90%	10
breakers	0.88%	8
mac	0.86%	7
lynch	0.84%	19
be	0.84%	384

6. Conclusion

Despite good results obtained with LSTM based models the quality observed with human evaluation is low. The predictions made during the first part of the input sequence are less qualitative accurate than the ones made at the end of the sentence. This could be explained by the fact that some initial words are required to understand and store the context inside the LSTM memory cell. Moreover, the prediction analysis made in section 5.3 does not highlight any possible pattern involved in the problem.

Possible future works may be a more in-depth investigation to improve the perceived quality, or 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