Final NLU project report

Andrea Bonora (232222)

University of Trento

andrea.bonora@studenti.unitn.it

1. Introduction

This paper presents the work done for the Natural Language Understanding course. This project aims to build a recurrent neural network (RNN) that can deal with a language modeling task. The model proposed in this work is an LSTM (Long-Short Term Memory) network that through the use of different regularization techniques and some other tricks is able to prevent overfitting and to reach a perplexity under the fixed baseline. The code of this project can be found at this link: `https://github.com/BonoraAndrea-unitn/NLU_project_22_23`.

2. Task Formalisation

Language modeling is a fundamental task in natural language processing (NLP) that involves predicting the next word in a sentence. In order to provide a basis for their word predictions, language models examine corpora of text data [1].

This project has as goal to build an RNN model that, given in input a sequence of words, is able to predict the word that comes next. Specifically, language models seek to model the linguistic intuition used intrinsically by humans, which is a powerful methodology to explore the grammatical structure allowed in a language, to articulate the rules that determine what is and is not part of a given language's grammar.

To satisfy the requirements of the project the final model must reach performances, in terms of perplexity, under a certain baseline. Specifically, using a Vanilla RNN model, the perplexity should be under 140. Performances under 90 PPL must be reached if the final network is instead implemented using LSTM cells.

3. Data Description & Analysis

In this work, the Penn Treebank (PTB) dataset has been used to train and test the model. PTB dataset is widely used in machine learning of NLP research. Specifically the English Penn Treebank corpus, and in particular the section of the corpus corresponding to the articles of Wall Street Journal (WSJ), is one of the most known and used corpus. The PTB dataset presents three different sets of sentences, one for training, one for validation, and one for testing. More in-depth, the training set is composed of about $900k$ words, much more than the validation set ($70k$) and the test set ($79k$). The difference in the vocabulary sizes is instead smaller, in fact, the dictionary for train, validation, and test contains respectively, $10k$ words, $6k$ words, and $6k$ words.

A crucial thing when it comes to language modeling is to understand if there are out-of-vocabulary (OOV) words which are unknown words that appear in the test/validation set but not in the train vocabulary. The presence of OOV words could introduce some difficulties in the model training. Fortunately, there are no OOV words in this dataset.

Next, an analysis of the words that have the highest frequency has been made. It turned out that the three most frequent words are the same for all the different sets. In particular, the $< unk >$ tag is always at least in the first two places in the ranking. In the PTB dataset, the $< unk >$ tag is used to represent words that occur infrequently or don't meet a certain frequency threshold.

Furthermore, analyzing the dataset, it was found that all the words are in lowercase. In fact, for each set, producing a vocabulary using the lowercase version of each word would lead to having an identical copy of the original one.

Finally, also statistics on the sentences have been collected, and they are reported in table 1.

Dataset	# of sentences	Min # of words	Max # of words	Average # of words
Train set	42068	1	82	21
Validation set	3370	1	74	20
Test set	3761	1	77	20

Table 1: *Statistics of the sentences for each set (train, validation, and test)*

4. Model

This section provides a description of the models tested in this project starting from a standard LSTM to the final model. For legibility issues, each technique and idea implemented has been explained in a different subsection. At the end of the section the model actually used is openly described.

4.1. Standard LSTM

An LSTM model is a Recurrent Neural Network that can process not only single data points but also entire sequences of data. In language modeling, there is the necessity of gathering information from a sequence of words to predict the next one correctly. Specifically, LSTM allows us to do this and it improves other RNNs which suffer from the vanishing gradient problem. LSTM units, differently from the other network units, allow gradients to flow unchanged during backpropagation.

4.2. Dropout

Dropout [2] is a technique used to improve generalization of neural networks. It consists of multiplying neural network activations by random zero-one masks during training [3]. It has proven to be an effective mechanism for regularization and preventing the co-adaptation of neurons. What proportion of the mask values are set to one is determined by a dropout probability p, which is a hyperparameter.

4.3. Skip/Residual Connections

It is pretty common, when dealing with language modeling, to stack multiple LSTM layers. To that end, the hidden state of a layer is given as input to the next layer and so on for all the layers. However, stacking many layers could introduce some problems in the backpropagation. To handle this problem skip connections or residual connections are often added [4]. These connections are used to improve the flow of information and further alleviate the vanishing gradient problem. Specifically, a residual connection can be defined as:

$$x_{l,t} = h_{l-1,t} + h_{l-2,t} \tag{1}$$

where l represents the layer and t the time step.

4.4. ASGD

"For the specific task of neural language modeling, traditionally SGD without momentum has been found to outperform other algorithms" [5]. Based on this observation and following [5] Averaged SGD (ASGD) has been tested. Differently from standard SGD, which updates the weights of the model using only the last iterate, the averaged version uses an average of the last $K - T$ iterates, where K is the total number of iterations and T is a hyperparameter. In this project, as in [5], a non-monotonic validation metric has been introduced in order to decide wheater to trigger or not the ASGD.

4.5. WeightDrop

Another useful mechanism to reduce overfitting, introduced by [5], is the use of DropConnect [6], very similar to the standard dropout, but here it is applied on the hidden to hidden weight matrices. As dropout, it is applied to the weight matrices before the forward and the backward pass [5]. Doing this will help to prevent overfitting from occurring in the recurrent connections of the LSTM.

4.6. Embedding Dropout

The embedding dropout [7] is equivalent to performing dropout on the weights of the embedding matrix at a word level. The purpose of embedding dropout is to prevent overfitting by randomly setting elements of the embedding matrix to zero during training. Specifically, this mechanism introduces noise and forces the model to rely on other available information to make predictions. This helps to reduce the model's reliance on specific word embeddings and encourages it to learn more robust representations that are less sensitive to individual word embeddings.

4.7. Locked Dropout

Locked dropout, differently from dropout, which samples a new binary mask every time it is called, simply samples the dropout mask only the first time it is called and then uses it repeatedly for all the repeated connections in the forward and backward pass [5].

4.8. AWS LSTM

The AWS LSTM [8] is the final model used to obtain the results that will be shown in the next section. The cited model is an LSTM network that uses some of the techniques previously described. Specifically, in the implementation of this work WeightDrop, Embedding Dropout, Locked Dropout, and tied weights are used. Specifically, this architecture uses DropConnect on the hidden-to-hidden transition within the RNN, embedding dropout in the embedding matrix, and Locked dropout for all the other dropout operations. Additionally, a new criterion to compute the loss during training has been defined. In fact, traditional cross-entropy loss has trouble in handling a large vocabulary size since the softmax operation used results computationally expensive and memory-intensive. To overcome this issue split cross-entropy loss is introduced. It addresses the large dictionary size problem by splitting the vocabulary into two or more partitions and instead of computing the softmax over all the words it is calculated for each partition separately. Furthermore, in the AWS LSTM implementation, other two regularization techniques are introduced: Activation Regularization (AR) and Temporal Activation Regularization (TAR). The first regularization mechanism tends to penalize activations that are meaningfully bigger than 0. It is defined as follows:

$$\alpha L_2(m \odot h_t) \tag{2}$$

where m indicates the dropout mask [5].

The second one instead is used to avoid significant changes in the hidden state and it is defined as

$$\beta L_2(h_t - h_{t+1}) \tag{3}$$

where β is a scaling coefficient [5].

4.9. Mogrifier AWS LSTM

The idea behind this last addition is to build a model where the two inputs x and h_{t-1} modulate one another before that the usual LSTM computation takes place [9]. Precisely, before feeding x and h_{t-1} to the LSTM layer, the value of both is computed from the interleaved sequences

$$x^i = 2\omega(Q^i h^{i-1}_{prev}) \odot x^{i-2} foroddi \in [1...r] \tag{4}$$

$$h^i_{prev} = 2\omega(R^i h^{i-1}) \odot h^{i-2}_{prev} foreveni \in [1...r] \tag{5}$$

(SISTEMARE FORMULE)

The number of steps r is a hyperparameter. Setting it to 0 means that no changes will be made on x and h_{t-1} before the LSTM layer. An example of how it works is visible in figure 1.

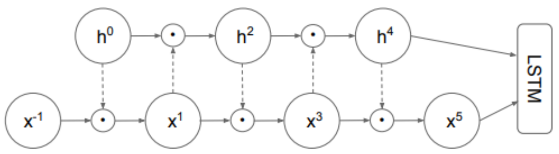

Figure 1: *Mogrifier LSTM with 5 steps.*

Different combinations of these techniques have been tested, and as reported in the next section, Mongrifier AWD LSTM seems to outperform all the other methods tried during the course of this project.

5. Evaluation

In order to evaluate the performances of the model the perplexity was used. Perplexity is a useful metric to evaluate models in natural language processing. There exist two possible ways to compute the perplexity. In this work, it has been computed as the exponential of the cross-entropy loss. Note that lower perplexity means better performances. The goal of this project is to build a model that performs below a perplexity value of 90.4.

During the course of this work, twelve models have been tested and in this section, a description of all of them is provided. For all the models, except when explicitly said, the embedding size and the hidden size have been set to 650. This value is the perfect trade-off between the computation capability and time-demanding in training.

All the models that will be described are trained for 50 epochs except for when the early stopping mechanism stops the training sooner than excepted in order to avoid overfitting. More precisely, early stopping is triggered after 6 epochs in a row with no improvements.

The optimizer used is the stochastic gradient descent (SGD) with the annealing of the learning rate if no improvements happened for 4 epochs in a row. The initial learning rate is always set to 30 and the weight decay to 1.2×10^{-6}. Finally, the batch size has been set to 70 for the train set, 10 for the validation set, and 1 for the test set.

All the results of the model that will be described here below are reported in table 2.

The first model implemented is a simple LSTM network. As imaginable, the performance is very poor and the behavior of the model during training is not optimal (figure 2), in fact, the model is able to perform very well on the train set but it provides a very poor performance on the validation set. The training of this model lasted only for 10 epochs thanks to early stopping.

To overcome the problems of the previous model a new network with the use of dropout has been tested. The final result is significantly better. Specifically, the value of the perplexity reached is 105.15, a good -27.14 wrt. to the standard LSTM implementation, but there is still a huge gap with the imposed baseline.

The third model tested is similar to model number two, so it is an LSTM implementation with the use of dropout, but in this network, the weights of the encoder and the decoder are tied [10][11]. As shown in table 2 tying weights lead to a considerable boost in performances, with a perplexity value that starts to approach the baseline. Seeing the improvements provided by the tied weights, this technique has been used in all the remaining models.

Subsequently, two tests have been made to try the addition of residual connections and averaged SGD. Unfortunately, both of the two models do not improve the performances obtained since this moment. For this reason, both residual connection and ASGD have been dropped after these two tests.

As mentioned in the section before, the final designed model to solve the task of this project is called AWD LSTM [5] [8]. Specifically, the implementation of AWD LSTM provides the use of weightDrop, embedding dropout, locked dropout, tied weights, and a new criterion called split cross-entropy. To assess the usefulness of the mentioned regularization techniques I have built models that introduce one technique at a time. As reported in table 2, all of these dropout techniques are useful in lowering the perplexity of the model. In particular, the introduction of locked dropout is the key to boosting the performance, leading to a -6.91 with respect to the model that only uses weight dropout and embedding dropout. Introducing also the split cross-entropy into the training allows another minor boost in performance.

Finally, the last model tried is the same one of the previous test, but with the idea of the mogrify LSTM presented by [9] and

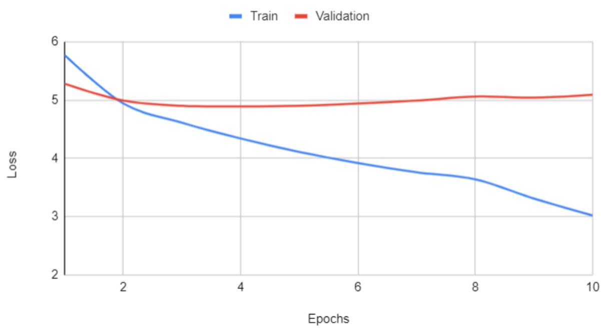

Figure 2: *Loss during training of the standard LSTM model*

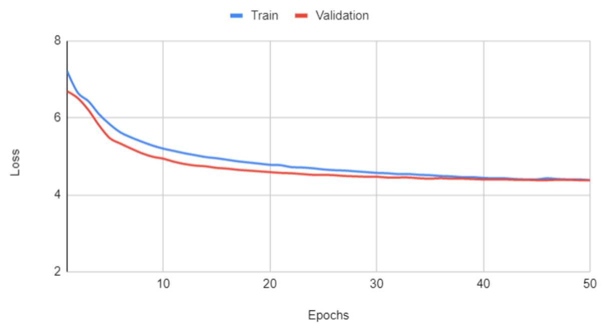

Figure 3: *Loss during training of the Mongrifier AWD LSTM model*

described in the previous section. Using the input and the previous hidden state of the model to modulate each other can help the network to find a more robust representation of the data and consequently lower the perplexity further more. Specifically, for this test the number of mogrify steps were set to 5. The final results are indeed as hoped, making the mogrify AWD LSTM the best model tried since now ($76.24 PPL$).

Two final attempts to achieve a better result have been performed using the mogrify AWD LSTM. Specifically, at first, the previous model described has been made to train for 150 epochs, and then as a second attempt the embedding size and the hidden size has been increased to 1150 making the model train for 150 epochs. Surprisingly, the model with lower embedding and hidden sizes performs better than the other one. More in-depth, the training of the first one lasted for about 130 epochs reaching a perplexity of 70.35 while the second one was trained for "only" 74 epochs achieving a result of 73.06 PPL. Considering the difference in the number of parameters ($24M$ vs $50M$) of the two models the result is astonishing.

In the end, I can say that for the sake of this project, the mogrifier AWD LSTM network implemented successfully reached and lowered the baseline imposed.

One more analysis to bring out is the prevention of overfitting by the final model. A comparison of the standard LSTM and the mogrifier AWD LSTM is here reported. As imaginable, and as it can be seen in figure 2, the standard LSTM has a huge discrepancy between train performances and validation/test performances. Specifically, during training the loss always decreases, instead the validation loss decrease only for the first 4 epochs, starting to increase from epoch 5 and triggering early stopping at epoch 10. The behavior of the model during training of the mogrifier AWD LSTM model is definitely

Model	Perplexity	Δ
Baseline	90.7	-
LSTM	132.29	+41.59
LSTM + Dropout	105.15	+14.45
LSTM + Dropout + Tied Weights	94.61	+3.91
LSTM + Dropout + Tied Weights + Residual Connections	99.13	+8.43
LSTM + Dropout + Tied Weights + ASGD	99.83	+9.13
LSTM + WeightDropout	86.86	-3.84
LSTM + WeightDropout + Embedding Dropout	86.57	-4.13
LSTM + WeightDropout + Embedded Dropout + Locked Dropout	79.66	-11.04
AWS LSTM	77.98	-12.72
Mogrifier AWS LSTM	76.24	-14.46
Mogrifier AWS LSTM upgrade*	73.06	-17.64
Mogrifier AWS LSTM (130 epochs)	70.35	-20.35

Table 2: *Models perplexity on the test set for the Penn Treebank language modeling task.*

more stable (figure 3), and both train and validation losses follow the same behavior. It is also noticeable that the training loss is higher than the validation one for the course of almost all the training process.

6. Conclusion

This project developed for the Natural Language Understanding course aimed to build a language model to predict a word given a sequence of precedents words. This has been done by proposing a series of LSTM-based models that used different combinations of some regularization techniques. The various test showed the importance of dropout and its variations in order to prevent the overfitting of the model. Specifically, the use of the dropout techniques showed in [5] lead to an improvement wrt. using the basic dropout implementation. Another important aspect that has contributed to lowering the perplexity of the model is the trick of tied weights. Using the same weights for both the encoder and the decoder turned out to be very useful in all the models trained. Finally, the use of split cross-entropy seems to overcome the possible issue of the standard cross-entropy loss in dealing with a large vocabulary size. In fact, using the split version boosted the performance a little bit. In the end, the best model presented achieved and surpass the baseline imposed by the task reaching a final perplexity of 70.35. Possible future works could be to test different optimizers and try different combinations of hyperparameters. For what concerns the first idea, the literature often refers to SGD as the best optimizer for language modeling tasks, for this reason, other optimizers have not been used in this project. The trial of different hyperparameters instead has not been possible due to constraints imposed by Google Colab.

7. References

[1] K. Jing and J. Xu, "A survey on neural network language models," *arXiv preprint arXiv:1906.03591*, 2019.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[3] G. Cheng, V. Peddinti, D. Povey, V. Manohar, S. Khudanpur, and Y. Yan, "An exploration of dropout with lstms," 08 2017, pp. 1586–1590.

[4] F. Godin, J. Dambre, and W. D. Neve, "Improving language modeling using densely connected recurrent neural networks," *CoRR*, vol. abs/1707.06130, 2017. [Online]. Available: http://arxiv.org/abs/1707.06130

[5] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *CoRR*, vol. abs/1708.02182, 2017. [Online]. Available: http://arxiv.org/abs/1708.02182

[6] 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

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

[8] S. Merity, N. S. Keskar, and R. Socher, "An analysis of neural language modeling at multiple scales," *CoRR*, vol. abs/1803.08240, 2018. [Online]. Available: http://arxiv.org/abs/1803.08240

[9] G. Melis, T. Kočiský, and P. Blunsom, "Mogrifier lstm," 2020.

[10] O. Press and L. Wolf, "Using the output embedding to improve language models," *CoRR*, vol. abs/1608.05859, 2016. [Online]. Available: http://arxiv.org/abs/1608.05859

[11] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *CoRR*, vol. abs/1611.01462, 2016. [Online]. Available: http://arxiv.org/abs/1611.01462