Final NLU project template

Andrea Bonora (mat. 232222)

University of Trento

andrea.bonora@studenti.unitn.it

1. Introduction

A variety of statistical and probabilistic approaches are used in language modeling (LM) to estimate the likelihood that a given sequence of words will appear in a sentence. In order to provide a basis for their word predictions, language models examine corpora of text data [1]. A variety of Natural Language Processing (NPL) tasks are based on LM. For instance, LM is used in machine translation tasks to assess the likelihood of the system's outputs and enhance the translation's fluency in the target language. In speech recognition tasks, the next word is predicted by combining the LM and acoustic models. This work proposes a Language Model implementation utilizing a LSTM (Long-Short Term Memory) backbone and tests the created model using data from the Penn Tree Bank.

2. Task Formalisation

A machine learning model referred to as a language model is one that is created to represent the language domain. It serves as a foundation for a variety of language-based tasks, such as:

- Question answering
- Semantic search
- Summarization

and plenty of other tasks that operate on natural language.

This project has as goal to build a 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.

The idea of this work is to build a model that should outperform the baseline in terms of perplexity in the test data. In section 4 a more detailed explanation of the actual model used is provided.

3. Data Description & Analysis

In this work, the Penn Treebank (PTB) dataset has been used. PTB 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 for the evaluation of models for sequence labeling.

The data used are extracted from the mentioned dataset, in particular, there are about $930K$ tokens for the training data, $74K$ tokens for validation data, and $82K$ tokens for testing data. The final vocabulary is limited to $10K$ words. For what concerns the number of sentences, there are about $42K$, $3K$, and $4K$ sentences for train, validation, and testing respectively.

An investigation of the used words has been done. Specifically, the frequency of each word has been calculated. Table 1 reported the 5 most used words for training, validation, and testing. Surprisingly, the word "the" does not appear in the five most used words in testing. This is an interesting data, which will be analyzed in section 5

Word	Frequency
Train	
the	50770
< unk >	45020
< eos >	42068
pierre	32481
nov.	24400
Validation	
the	4122
< unk >	3485
< eos >	3370
pierre	2603
nov.	1832
Test	
< unk >	4794
join	4529
< eos >	3761
pierre	2523
nov.	2195

Table 1: *Frequencies of the top-5 most used words for train, validation and test*

4. Model

The proposed work is based on a LSTM model, which is a Recurrent Neural Network (RNN) that can process not only single data points but also entire sequences of data. For the task of this work, using a RNN is crucial since there is the necessity of gathering the information of a sequence of words in order to predict the next one correctly. Specifically, LSTM allows to do this and furthermore, it improves other RNNs which suffer from the vanishing gradient problem. In fact, LSTM units, differently from the other network units, allow gradients to flow unchanged.

The central role of a LSTM model is held by a memory cell known as a 'cell state' that maintains its state over time. The cell state resembles a conveyor belt in some ways. With only a few minor linear interactions, it proceeds directly down the entire chain. It's easy for information to flow without any changes. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Information can pass through gates on a purely optional basis. They consist of a point-wise multiplication process and a layer of sigmoid neural networks. Indicating how much of each component should be allowed through, the sigmoid layer outputs numbers between zero and one.

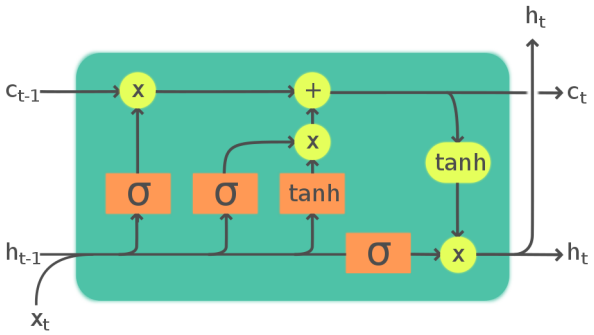

Figure 1: *Image of a LSTM cell. It can process data sequentially and keep its hidden state through time.*

In this work, a standard LSTM version provided by PyTorch has been used. The standard implementation has been improved through the use of dropout which means that a unit is dropped at training time with a given probability p. All units are present at test time, but their weights have been scaled by p. The goal is to avoid co-adaptation, which occurs when a neural network becomes overly dependent on a small number of connections and may be a sign of overfitting. Furthermore, to capture more information about the words in the sequence an extension of backpropagation is needed and it is called backpropagation through time (BPTT). In particular, the truncated BPTT version has been used, in which the error is propagated through recurrent connections back in time for a specific number of time steps [2].

To further improve the training process an investigation on the averaged SGD (ASGD) has been performed. ASGD takes steps identical to standard SGD, but instead of returning the last iterate as the solution, it returns an average summation of the weights, normalized by the number of iterations: $\frac{1}{K-T+1}\sum_{i=T}^{K}\omega_i$ [3].

An extension of dropout has also been tried. In fact, in standard dropout, a new binary dropout mask is sampled each and every time the dropout function is called. This means that new dropout masks are sampled even if the given connection is repeated, such as the input x_0 to a LSTM timestep $t = 0$ receiving a different dropout mask than the input x_1 def to the same LSTM at $t = 1$. The variation used in this work is called variational dropout. It aims to sample a binary dropout mask only once upon the first call and then to repeatedly use that locked dropout mask for all repeated connections within the forward and backward pass [3].

In addition to dropout, other two techniques have been investigated in order to reduce overfitting: activation regularization (AR) and Temporal Activation Regularization (TAR). The first one "*consists in using the L_2-regularization on the weights of the network to control the norm of the resulting model*" [3]. Specifically, the formula of AR is the following:

$$\alpha L_2(m \odot h_t) \tag{1}$$

where m is the dropout mask, $L_2(\cdot) = ||\cdot||_2$, h_t is the output of the RNN at timestep t, and α is a scaling coefficient. The second "*use the L_2 decay on the individual unit activations and on the difference in outputs of an RNN at different time steps*" [3]. The equation for computing TAR is reported here.

$$\beta L_2(h_t - h_{t+1}) \tag{2}$$

where β is a scaling coefficient.

In order to regularize the network, AR specifically penalizes activations that are significantly larger than 0. Instead, TAR belongs to the broad class of "*slowness* regularizers," which discourage models from making significant changes to the hidden state.

5. Evaluation

This section reports all the experiments performed. All experiments use a two-layer LSTM model with 1500 units in the hidden layer and an embedding size of 1500. These two parameters are set equal in order to use tied weights which means to use the same weights for the encoder and the decoder [4] [5]. Other parameters have been fixed during all the experiments, specifically, the dropout is set always to 0.65, the initial learning rate to 20, the number of epochs to 40 and the batch size is 20 for the training phase and 10 for validation and testing.

The optimizer used was always the stochastic gradient descent (SGD), with the annealing of the learning rate every time a worsening in performance happened. In order to use the L2 regularization, the weight decay was set to $1.2e-6$.

In order to evaluate 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.

Several configurations of the model has been tried and the results are reported in table 2. To better understand how the experiments have been performed, an explanation of the various test is reported below:

- Model1: a standard LSTM model with the use of dropout that was set to 0.65.
- Model2: identical to Model1, but here the dropout for the input was set to 0.1
- Model3: identical to Model1, but variational dropout was used instead of standard dropout
- Model4: similar to model3 with the addition of activation regularization ($\alpha = 2$) and Temporal Activation Regularization ($\beta = 1$)
- Model5: similar to model1 with the addition of activation regularization ($\alpha = 7$)
- Model6: variation of model 3 with the use of ASGD when the performances don't improves.

As the results show the simple addition of dropout to the LSTM model boosts the performance of the model. The improvement is significantly high (-11.59). Other methods presented a smaller enhancement in perplexity with respect to model1. Specifically, using variational dropout and the two regularization techniques lead to a further -2.66 in performance, which is the better result obtained. The results obtained suggest that dropout and regularization are crucial when it comes to train a language model. Different value for α and β need to be tried in order to get the ones that produce the better results. This has not been done since, as said before, using larger value for AR and TAR lead to a higher training time. Due to constrains imposed by google colab these tests could not be carried out. Finally, among all the tested models, the one that involved the use of ASGD did not produce any improvement. On the contrary, it lead to a worst result.

Model's name	Perplexity	Gain
Baseline	90.7	-
Model1	77.33	-13.37
Model2	77.47	-13.23
Model3	76.68	-14.02
Model4	75.47	-15.23
Model5	74.52	-16.18
Model6	77.44	-13.26

Table 2: *Results of the performed experiments. Performance measured using perplexity, so lower value means better performance*

Word	# errors
the	11713
<unk>	10284
<eos>	4062
mr.	1978
form	1564
corp.	1457
pierre	1268
join	1117
a	1056
striking	982

Table 3: *Top-10 word for number of errors.*

One last data examinated is the words that produce the highest number of errors (table 3). As can be predicted, the word "the" is the one that causes an error in the prediction the highest number of time when predicted as next word. This is probably due to the fact that the word "the" is the one with the highest frequency in the traning dataset, and is not that popular on the test dataset.

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 LSTM model with the combinations of some techinques like variational dropout, backpropagation through time, activation regularization and temporal activation regularization. The results shows some improvements with respect to the baseline, but some other improvement could be done. Specifically, an extension of the built-in dropout of the LSTM pytorch implementation could be developed, as done in [3], which proposed the use of DropConnect on the recurrent hidden to hidden weight matrices. Another idea is to reduce the size of the model without compromising the performances and reduce the training time.

7. References

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

[2] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 5528–5531.

[3] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[4] O. Press and L. Wolf, "Using the output embedding to improve language models," *arXiv preprint arXiv:1608.05859*, 2016.

[5] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *arXiv preprint arXiv:1611.01462*, 2016.