Final NLU project: Language Modeling with Recurrent Neural Networks

Sara Miotto (232086)

University of Trento

sara.miotto@studenti.unitn.it

Abstract

The main purpose of this project is to implement a Language Modeling by using a basic RNN architecture and try to improve it through the implementation of some regularization techniques.

1. INTRODUCTION

The model built aims at predicting the next word within a given sequence and needs to have the architecture of a recursive neural network, namely a LSTM. The dataset chosen for the training as well as for the testing is the well-known *Penn Treebank*(PTB). The goodness of the prediction is measured both through the *cross entropy* and through the *perplexity*. In order to obtain better outcomes several regularization techniques have been taken into account, in particular: *Weight drop*, *Variational Dropout*, *Early Stopping*, *Weight Norm Regularization*, *Weight Tying* and *Gradient Clipping*.
The sections of the paper are structured as follows. First, in *Task Formalisation*, the assigned task will be described in detail. Then, in *Data Description* & *Analysis* we can find a explanation of the dataset used, its origin and composition, as well as some useful statistics. *Model* chapter deals with the neural network itself, the pipeline, the baseline, the regularization techniques and the experiments carried out. In *Evaluation* results are presented and commented, expecially there is a comparison between the baseline and the regularized model. Finally in *Conclusions* everything is summed up.

2. TASK FORMALIZATION

Language Modeling attempts at defining a probability distribution for the different logical units of a sentence. These conditional probabilities calculated over the entire vocabulary are used in conjunction with the chain rule in the following way:
$P(w_{1:n} = \prod_{i=1}^{n} P(w_i|w_{<i})$
The traditional approach is the so-called *count-based n-gram*, which relies on observing a pre-determined number of words preceding the one we want to predict and the probability of this word will be determined by counting the number of times it has appeared in the studied sequence. In the modern approach, neural networks have taken over. At first feed-forward neural networks have addressed this problem, yet, in a second moment, recurrent neural networks have proved to be better. In fact, computationally speaking, they were less expensive and furthermore they were very flexible in being invariant to translations. RNNs have made it possible to deal with inputs of any length and also to use information from several previous steps.
In particular, LSTM has been chosen as baseline because of its advantage of having an explicit context layer as well as gates controlling the flow of information in and out the various units of the neural network itself. The input sequence is codified as follows:
x_{h_t} = LSTM($E^T[x_{t-n+1}, x_{t-n+2}, ..., x_{t-1}]$).
The probability is then calculated as

$P(x_t = x_k|h_t) = \frac{exp(W_k h_t)}{\sum exp(W_j h_t)}$

with V being the size of the vocabulary and W_k the $k_t h$ row vector in the projection matrix.
As for the initialization of the weights, several techniques have been brought together harmoniously, including *Xavier initialization* for input weights, *Orthogonal initialization* for recurrent weights, *Zero initialization* for bias and *Uniform initialization* for Linear and Embedding layers.
At a later date, all the various regularization tricks were tested for their beneficial or non-beneficial effect on the LSTM.
Pytorch libraries were used and the whole code has been implemented in *Google Colab* so as to be able to use GPUs.

3. DATA DESCRIPTION & ANALYSIS

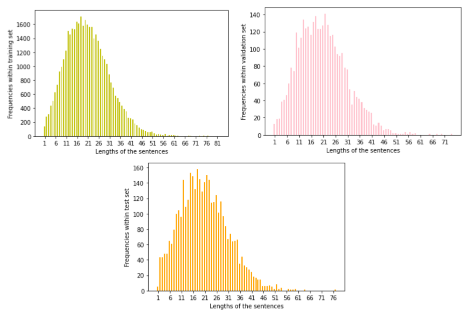

Figure 1: *Distributions of words respectively in the training, validation and test set*

3.1. Dataset origin and composition

PTB dataset was developed by researchers at the University of Pennsylvania and consists of approximately 4.5 million words of newspaper and magazine articles from the *Wall Street Journal and Associated Press*. It is heavily pre-processed, in that it has no capital letters, numbers (which were substituted with *N*) or punctuation of any kind. It also contains the token *unk*, standing for the large number of out of vocabulary tokens which are the results of the relatively reduced size of the vocabulary, equal to 10000 words. However, all the words in validation and test sets are already included in the training dataset, and so there are not OOV after the creation of the vocabulary. This last is made of unique words and a wide variety of syntactic structures, such as nominal phrases, verbal phrases, and prepositional phrases.
In the most common split of this corpus, sections $0-20$ are

used as training data (929589 tokens), sections $21 - 22$ as validation data (73760 tokens) and sections $23 - 24$ as test data (82430 tokens).

3.2. Statistics of the Dataset

The following section provides statistical features related to the dataset used.
The number of sentences is equal to 42068, 3761 and 3370, for the training, validation and test dataset respectively. Regarding the maximum identified length of the sentences, it has turned out to be 82 for the training ones, 74 for validation and 77 for test. On the other hand, for what concerns the minimum length, it has emerged that there are no empty rows, but several have a unit size (almost 137). The most frequent average length was then analyzed, which results to be between 15 and 20 words (see graph 1). N and *unk* also appear among the most frequent words.

4. MODEL

In this section, the layout of the model is explicated (see figure 2). It starts with a common pre-processing phase and then dwells on the baseline structure, so the regularization techniques are added to improve the model itself.

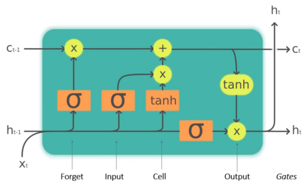

Figure 2: *Weight-dropped LSTM*

4.1. Pre-processing phase

Training, validation and test sets were already provided so it was just necessary to read and save them within three lists of sentences.
Due to the fact that the sequences were characterized by different lengths, padding has been applied. It consists of filling the shortest ones with a special value (known as *padding token*) so that they would all have exactly the same length. Also a symbol standing for the end of the line has been added.
Words were then mapped into integer numbers to be accepted by the neural network.
So, a Dataset class has been created for dealing and working with the dataset in an easier way. The Dataset was loaded into a Dataloader class, an iterable object returning a batch of features and labels, at each single step. In fact, the dataset, as it is, cannot fit in GPU memories, and so it needed to be reshuffled and split into these batches whose dimension has been set to 64.

4.2. LSTM baseline

The baseline of the LSTM consists, first of all, of an *Embedding Layer* which receives as input the sequence of integers representing the words and maps them into the so-called embedding space, one word at a time. Its basic purpose is the one of representing inputs in a continuous space and in a way that is acceptable by the neural network. The size of this space is smaller when compared with the vocabulary and so the model is allowed to give a compact representation of the words and to capture accordingly the relationships between them. From a mathematical point of view, it is implemented as a matrix, the *Embedding Matrix*, of dimensions (*vocab-size*, *embedding-dim*), where *vocab-size* is the size of the vocabulary and *embedding-dim* the one of the embedding space. Thus, the outcome is another matrix (n, *embedding-dim*), where n is the number of words in the input sequence.
The embedding layer is then followed by one LSTM layer with three different inputs, namely the current input, the previous internal state vector (the long-term information that the network is able to maintain in the long term), and the previous state. The output is the result of the updated internal state vector and the current state. The internal state changes according to the input sequence and so it is able to discern between useful and not useful information for the aim of prediction.
LSTM units give way to a fully connected layer, responsible of the final prediction of the word. Mathematically it is expressed as: $output = W * input + b$, with w being the matrix of weights and b the bias. Thus, the outcome is equal to the scalar product between the input and the weights matrix, plus the bias vector.
For the training the classical stochastic gradient descent optimizer was used.

4.3. Regularized and optimized Model

The so-called *weight dropped LSTM* has the structure observed in figure 2.

4.3.1. Weight drop

Dropout is one of the most popular regularization technique, although the random deactivation of some neurons could cause problems with the storage of long temporal sequences or moreover destroy them in the training phase.
For this reason, it was opted not for a naive implementation, but rather for the so-called weight drop. In fact, Weight drop can provide a more controlled form of regularization for an LSTM as it allows a selective weight adjustment. Unlike dropout which randomly drops neurons and potentially leads to an unstable representations, weight drop focuses on specific weights and preserves critical connections.
A dropout probability must be defined for each weight within the recurrent hidden-to-hidden matrices before the forward and the backward pass and remain the same for the entire process as weights are reused over multiple steps.

4.3.2. Variational Dropout

Variational dropout is used for all dropout operations except those involving the hidden-to-hidden transition. Unlike standard dropout, it uses the same dropout mask at each step, meaning that the units to be dropped are always the same.
While the dropout pattern is fixed, the dropout rate for each neuron is a random variable and learnt during training.

4.3.3. Early stopping

Early stopping is very popular in preventing overfitting as it involves the interruption of the training process at a certain point before the model has had the opportunity to fully converge, based on the performance of the model on a validation set. The idea behind this technique is that, as the model continues to train, at some time it will eventually starts to over-fit the training data, and this is indicated by a decrease in the performance on the validation set, even if the training seems to keep improv-

ing. It allows to achieve a certain generalization with reference to unseen data.
In the specific case of this project, the monitor has been set on the perplexity computed on the validation test and the coefficient of patience is set to three, meaning that after three times perplexity is seen to keep increasing, the training is interrupted.

4.3.4. Weight Norm Regularization

L_2 regularization aims at pushing the weights of the neural network towards zero, as large weights can lead to overfitting. However, this method has limitations because it affects all weights equally and can cause the weights of hidden units being stuck near zero. To address this issue, it is possible to consider a constraint on the embedding and projection matrices, whose weights are shared. Since the row vectors represent word vectors, it is more appropriate to act on their norms rather than on all the individual weight parameter. The use of a soft regularization loss term rather than hard-fixing weight norms in the forward pass is motivated by the distribution of weight norms. It can be seen that NLMs tend to learn unequal weight norms for words with different counts. Therefore, hard-fixing weight norms may limit the ability to learn of the network.

4.3.5. Weight tying

Weight tying is based on the idea of sharing the weights of certain parts of the model, in this case between embedding and softmax layer, so as to reduce the number of parameters the network has to learn and consequently to prevent overfitting. However, it is important to note that weight binding can also limit the model's ability to fit the data and reduce its ability to generalize.

4.3.6. Gradient clipping

Since a language model can generate very large gradients due to the large amount of data and dependencies between words, gradient clipping is a useful technique to deal with such a problem.

4.4. Non-monotone Target-Adaptive Stochastic Gradient Descent

In the regularized version of the neural network the traditional *SGD* has been replaced with a slightly different version of it: *Non-monotone Target-Adaptive Stochastic Gradient Descent* (NT-ASGD). It is designed specifically to improve the convergence rate of *SGD* in non-convex optimization problems ad introduces a target update rule that adaptively adjusts the learning rate based on the performance of the current iteration compared to the previous one. Taking the average is a method which helps the stabilization of the training and improves the final performance of the model. To ensure that the averaging process does not significantly influence the stability of the neural network weights, it is important to perform averaging at a point during training when the weights have reached a consistent value. One strategy to determine when to average the weights is to monitor the performances of the model on a validation set. When the validation metric fails to improve for multiple cycles, averaging can be initiated. This technique requires the use of two additional hyperparameters: *L*, the logging interval (typically set to the number of iterations in an epoch), and *n*, a non-monotone interval, set to 5. These hyperparameters are used to prevent the randomness of the process to significantly impact on the decision to average the weights.

5. EVALUATION

Below the criteria used to evaluate the functioning of both the baseline and the regularized model are made explicit. The outcomes found are then related to each other, and through an error analysis there is an attempt to give both qualitative and quantitative explanations.

5.1. Metrics used

The goodness of the language model is analyzed in terms of cross entropy and perplexity. *CE* is a logarithmic penalty, in which the prediction probability of a given class is compared with the actual desired output. The formula is the following: $L_{CE} = -\sum t_i log(p_i)$.
Perplexity is strictly related to it as it is the exponentiated average negative log probability of the model, where the probability is calculated for each word in the test set. In other words it is computed by taking the average negative log probability of the model across all the words in the test set, and then exponentiating that value. The negative log probability is a measure of the uncertainty or unpredictability of the predictions of the model, and exponentiating it helps to scale the perplexity score to a more interpretable range. A lower perplexity score indicates that the model is more confident in its predictions. Given a sentence $w_1 w_2 w_n$, the perplexity is defined as $(P(w_1 w_2 w_n))^{-\frac{1}{n}}$.

5.2. Results on evaluation

Four runs of the code have been performed in order to obtain more reliable results and the mean of these values is the one reported here below.

5.2.1. Results for the baseline

The model was trained for fifty epochs, following which the perplexity on the test set has shown to be equal to 133.225 with a standard deviation of 1.407 and a cross entropy of 4.892 (standard deviation of 0.011).

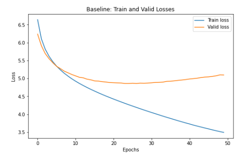

Figure 3: *Train and valid loss in the baseline*

5.2.2. Results for the regularized model

For the regularized model early stopping was applied and the value of perplexity on the test is 95.019 (standard deviation: 0.142), while the loss is 4.554 (standard deviation: 0.001). The

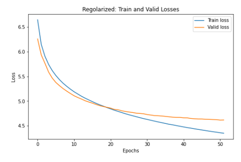

Figure 4: *Train and valid loss in the regularized model*

regularized model achieves better values of perplexity but that is not all. Observing the loss curves plotted on the graphs (3 and 4) it possible to notice that without the regularization techniques, after a little more than twenty epochs the model is prone to the phenomenon of overfitting, i.e. the value of the perplexity on the training set tends to reach very low levels creating an ever-increasing gap with the one calculated in the test set.

5.3. Comparison and differences between regularized and non-regularized model

In comparing the regularized and the non-regularized versions of the neural network, several aspects can be taken into account: performance, complexity, training time and robustness.
First of all, the regularized version of the LSTM has proven to be able to achieve better values of the perplexity without burdening the gap between the training and validation loss.
The regularized model is surely more complex computationally speaking and as a consequence the time required for a single epoch is higher than the one spent for the baseline. Even if, thanks to early stopping, also the regularized version converges relatively fast.
Furthermore, the regularized LSTM is more robust to changes in the training data or to slight variations in the test set, compared to the baseline model.

5.4. Interpretation of errors and analysis

In order to better understand the difficulties encountered by the neural network in making predictions, I first thought to analyze the correlation between the length of the sentence and the correctness of the prediction (see graphs 5). The first graph takes into account the correctness of the word's position within the sentence, while the second only checks whether the word actually figures in the target. In the wake of this lenght analysis,

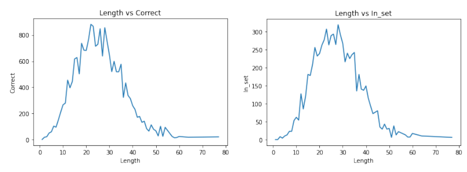

Figure 5: *Respectively: each single word is correctly predicted and is placed in the right point of the sentence, each single word is correctly predicted even if not necessarily in the right point of the sentence*

I have printed the first five sentences with the highest as well with the lowest values of the loss. In both scenario they are pretty short, made up of about 20 or 30 words. This is not surprisingly because when dealing with a relatively short sentence the neural network is not able to learn the relationships between words. In fact, the ones with a low loss contain words which appeared many times in the training dataset and the model had actually memorized them.
Sentence with the highest loss (9.236802):

- Full Sentence: as commonly understood service implies sacrifice eos;
- Target : commonly understood service implies sacrifice eos;
- Predict: a as the the the to;

Sentence with the lowest loss (0.167244):

- Full Sentence: revenue rose N N to $Nbillionfrom$ N billion eos;
- Target : rose N N to $Nbillionfrom$ N billion eos;
- Predict: rose N N to $Nbillionfrom$ N billion eos.

I have tried to relate the lenght also with the loss as can be observed in figure 6. The distribution appears to be pretty uniform and these two variables do not seem correlated at all (indeed the coefficient is very small, equal to 0.13).

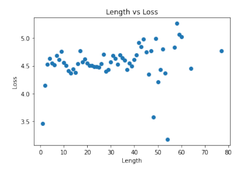

Figure 6: *Correlation between the lenght of the sentence and the calculated loss*

In 7 it is possible to find the most frequent words within the target and the prediction set. They correspond to the ones that also appear more often in the training set and are mainly prepositions, numbers and end of sentences. Essentially when the network is in trouble due to the absence of contextual information, it will tend to output exactly these ones.

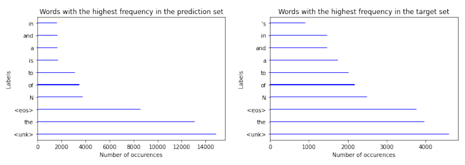

Figure 7: *Most frequent words within the prediction and the target set*

It has also been proved that the network is unable to take into account all those words which appear less than eight times in the training set.

6. CONCLUSION

This project was very useful in learning how a language model works in practice and how a recursive neural network is suited for this purpose. It was also interesting to note the impact regularization techniques have on the model under analysis. All in all, the outcomes obtained are satisfactory, although they obviously leave room for further improvements.

7. References

[1] Merity, et al., 2017, ”Regularizing and Optimizing LSTM Language Models”, *arXiv preprint arXiv: 1708.02182*.

[2] Christian Herold,et al., 2018, ”Improving Neural Language Models with Weight Norm Initialization and Regularization”, *Proceedings of the Third Conference on Machine Translation (WMT), Volume 1: Research Papers, pages 93–100*. https://aclanthology.org/W18-6310

[3] Hakan Inan, et al., 2017 ”TYING WORD VECTORS AND WORD CLASSIFIERS: A LOSS FRAMEWORK FOR LANGUAGE MODELING”, *arXiv preprint arXiv:1611.01462v3*