Final NLU project

Giovanni Ambrosi (232252)

University of Trento

giovanni.ambrosi@studenti.unitn.it

Abstract

This document is the final report for the Natural Language Understanding course. The main goal is to implement a Language Model and try to improve its performance through some regularization techniques.

1. Introduction

The project develops all the main steps that a Language Model framework, but also every general task in the Natural Language Understanding field, requires:

- Data Analysis and Preprocessing;
- Model training;
- Improvement of results;

First, I downloaded the dataset and started analyzing it at sentence and word levels, but also in terms of the statistical distribution of its content. Then, an LSTM was implemented for word prediction and, in a second phase, some regularization techniques were integrated in order to improve the baseline of 90.7 Perplexity score.
In the end, I conducted an analysis to verify the goodness of my regularized models, asking each of them to generate some sentences.

2. Task Formalisation

The problem is divided into three subtasks: sequence mapping, word prediction, and model regularization. The first one aims to give each word an index that represents its class/label while the second is solved in a classification fashion using the input words to predict the next one.
Since neural networks are complex structures and are composed of millions of parameters that need to be optimized, we have to consider their tendency to overfit on the training data, and here conceals the third objective of this project, which is to apply specific regularization techniques to ensure that the (final) model reaches the best possible generalization capability.

3. Data Description & Analysis

The dataset used is the Penn TreeBank dataset composed of articles taken from the Wall Street Journal. In the most common split the train set is composed of 42068 sentences and 925139 tokens, validation and test sets are made of 3370 sentences, 73328 tokens, and 3761 sentences, and 82218 tokens respectively. The train partition represents the 85.5% of the entire dataset, while validation and test sets compose the 6.86% and 7.65% respectively. Going deeper into the dataset analysis, I explored some statistical indexes about the sentences and computed the average length, standard deviation, and probability distribution of the sentences' lengths, noticing that the three sets show similar statistics. It is essential to say that I there are no Out-Of-Vocabulary words in the three splits.
Words are already lowered, numbers are substituted with the 'N' token and most of the punctuation is removed, however, some preprocessing steps are performed that are presented later.
Furthermore, I did an analysis of the label distribution noticing that the three splits are strongly represented by the 'unk' word. This information will be used for evaluating the models properly.
Table 1 and 2 summarize the main features of the PTB Dataset.

Partition	no sent	avg len	std dev len	%
train	42068	22	10.11	85.506
valid	3370	22	9.93	6.850
test	3761	22	10.18	7.644

Table 1: *Statistics of the PTB Dataset*

Word	% on train	% on valid	% on test
the	5.49	5.62	5.51
unk	4.87	4.75	5.83
eos	4.54	4.60	4.57
N	3.51	3.55	3.07
of	2.64	2.509	2.67

Table 2: *Label distribution on the PTB dataset*

4. Model

The model chosen for the project is a Long-Short-Term-Memory. The main reason is that most of the techniques taken from Merity et al.[1] have been tried on an LSTM network.
The vanilla version is composed of 1 Embedding layer, 1 LSTM layer, along with its gates that define the memory of the network, and 1 Fully Connected layer for prediction.
First, some data preprocessing on the given dataset is performed. Preprocessing guarantees that all the irrelevant parts in a corpus are removed and prepares the entire corpus for the sequence mapping step. The operations carried out are:

- **Tokenization**: separates each word/token in a sentence.
- **Void sentence removal**: removes void sentences (if any)
- **'eos' token**: inserts a user-defined 'eos' token at the end of each sequence. In this project, it is represented by the word 'eos'.

After having removed all the elements that are not useful, the dataset is ready to be handled.
A class called Vocabulary takes all the preprocessed sequences in the dataset and maps each token with a number based on the position that element occurs in the corpus.
All the words that occur less than 3 times are mapped to the 'unk' token.
At the end of this process, each word of a sentence is returned along with its label in the Vocabulary class and the label of the next word.
Practically speaking:

sentence	**I**	**eat**	**an**	**apple**	**'eos'**
Vocabulary index	1	2	3	4	0
Input	1	2	3	4	-
Labels	2	3	4	0	-

Table 3: *Example of sequence mapping*

The second subtask of Language Model is the classification that allows the model, once trained, to predict the next word.
Initially, a vanilla LSTM is used to look at the model's behavior with the Embedding Layer composed of 300 units while the Recurrent Layer has 200 units. Multiple runs varying the batch size and the learning rate have been done and in the end, the best results have been achieved with a batch size of 32 using SGD (as written in Merity et. al.[1] the SGD outperforms other types of optimizers) with a learning rate set to 1.0.
To prevent the model from overfitting, early-stopping has been implemented. It monitors the error on the validation set and if it worsens 5 times (user-choice) during the epochs it stops the training process and saves the best weights' configuration.
The third point I focused on is the regularization of the model and different types of regularization techniques, mainly in the form of Dropout along with the previous early-stopping, have been implemented and used for both generalization performance and overfitting avoidance. The main techniques are described below:

4.1. Embedding Dropout

As the name suggests, the Embedding Dropout performs dropout on the embedding matrix at a word level, where the dropout is broadcast across all the word vector's embedding. The remaining non-dropped-out word embeddings are scaled by $\frac{1}{1-p_e}$ factor, where p_e is the probability of embedding dropout.

4.2. Variational (Locked) Dropout

In standard dropout, a new binary dropout mask is sampled each and every time the dropout function is called. Variational (Locked) Dropout ensures that the same mask is applied within the same batch of samples. Originally, Variational dropout was applied on the hidden representation but as written in Merity et al.[1] it is better to use Variational dropout for embedding and output layer.

4.3. Weight Dropout or DropConnect

For the hidden representation, Weight Drop/DropConnect [3] is used, where the dropout operation is applied on the weight matrices instead of on the activations, before the forward and backward passes.

4.4. Weigth Tying

Weight Tying improves the performance of Language Models by tying (sharing) the weights of the embedding and classification layers. This method also massively reduces the total number of parameters.

4.5. Non-Monotically triggered Average Stochastic Gradient Descent

To further improve the training process, NT-ASGD is also used. During the training process, we look at the validation loss and, as for the early-stopping, if it fails to improve for multiple epochs, suggesting that the model has stagnated, the non-monotonic criterion triggers the ASGD that starts decaying the learning rate.

5. Evaluation

The criteria used for evaluating the model are:

- **Cross-Entropy Loss** used generally for classification tasks;
$$H(W) = -\frac{1}{N}\log(p(W)) \tag{1}$$
where q is the approximating distribution on data W
- **Perplexity**
$$PP(W) = 2^{H(W)} \tag{2}$$
where $H(W)$ is the previously defined cross-entropy.
Perplexity is a measurement of how well a probability model predicts a sample. In the context of Natural Language Understanding, perplexity is a way to capture the degree of 'uncertainty' a model has in predicting (i.e. assigning probabilities to) text. The lower the perplexity, the better the model, and the **baseline** to improve is **90.7**.
- **Subjective metric**. A function that generates a sentence starting from n previous words has been implemented and it is used as a subjective metric. Through this, I want to examine the ability of a trained model to predict words and see if it is capable of generating meaningful sentences. Since the train set is strongly represented (almost 5% of all the words) by the 'unk' word, I decided to avoid the tested model outputs that token, forcing the network to output the second most probable word it predicts. The maximum length of the output sentence is set to 20.

5.1. Vanilla LSTM

As said the first model is a vanilla LSTM that shows a tendency to overfit training data and is not capable of improving for the whole 50 epochs, starting to overfit around the 30th and triggering the early stopping around epoch 40. The final results obtained are a Perplexity score of around 147.8 on the validation set and around 145.5 on the test set. Looking at the perplexity curve in Fig.1 it stops improving after around 20 epochs and starts having a "flat" pace, meaning that it might be stuck in a local minimum.

5.2. Embedding LSTM

Since in [1] the best LSTM has got 33 millions of parameters I decided to augment the dimension of my model, setting the number of units in the embedding and hidden layers to 400 (in order to apply weight tying) and the number of LSTM layers (recurrent layers of the network) to 3.
The first Dropout technique tried is the Embedding dropout where the mask is sampled with probability 0.2. The performance of the model increases a lot reaching a validation Perplexity score of around 96.1 and 94.7 on the test set but it is still subject to strong overfitting problems and the learning gets early-stopped after around 17 epochs (orange line in Fig. 1). The first consideration about the new architecture and dropout is that they help the model to start from a better position, in fact, looking at the PPL curves of all models in 1 we can clearly see that the initial perplexity is highly lower with respect to the vanilla LSTM (blue graph). This dropout technique has been revealed very effective and for this reason, I decided to add once at a time the remaining regularization strategies.

5.3. Embedding and Variational LSTM

After the embedding layer, I used Variational Dropout with probability 0.4. The results, highlighted with the green line in Fig. 1 show that the LSTM is less prone to overfit and the training continues almost till the end of the 50 epochs, stopping itself around the 42nd epoch. The Perplexity is even better and for the first time, the model succeeds in outperforming the baseline achieving a 86.8 validation PPL score and 86.5 PPL score on the test set. Unlike the previous models, I can experimentally conclude that the network is not optimized at its best level. In fact, I noticed that during the last epochs, the model either improved the epoch before it got stopped or got stuck in a local minimum returning the same loss on the validation set for several epochs. This suggested that it can go beyond the limit of 86.8 on the validation set (same for the test set) by properly acting on the most important part of the network, the memory.

5.4. Embedding and Variational and Weight Drop LSTM

In the first two experiments, the models have shown a tendency to trigger the early stopping.
The last dropout technique added is Drop Connect on the last recurrent layer with probability 0.2. The results have improved and the perplexity scores on the validation and test sets are 81.8 and 81.0 respectively. Initially, the choice of the layer to apply the mask was random, but based on these good results, I started an ablation study of the effect of Weight Dropout on the recurrent layers of the network.

5.4.1. Weight Drop on input recurrent layer

Using Weight Drop on the first recurrent layer leads the network to worsen slightly performance with respect to using it only on the last LSTM layer. This can be explained through the fact that the LSTM layers represent the memory of the network and removing some information on the input layer triggers a negative cascade effect on the other recurrent layers. Maybe ”masking” some of the input information makes the network unable to remember well the history of the data.

5.4.2. Weight Drop on all recurrent layers

I explored the effect of Weight Drop on all recurrent layers and as for the previous model, the network had a worse performance, confirming the thesis that applying the dropout on the first recurrent layer has an impact on all the others. For both the two models described the PPL score was about 86.7 and 86.0 on the dev set and test set.

5.4.3. Weight Drop on second and third recurrent layers

Discarded the option to apply the dropout technique on the input layer, I focused on the second and third ones and noticed that the model slightly improves the PPL score on both validation and test sets, achieving 80.8 on validation and 80.5 on test. Therefore, for the last experiment, the selected model is the Embedding and Variational and Weight Drop LSTM with DropConnect applied to the 2nd and 3rd recurrent layers (in the Table 4 only the results for this model are reported).

5.5. Average Weight Dropped LSTM

As the last technique, I used Non-Monotically Average Stochastic Gradient Descent.
Since it is not guaranteed that the optimizer switches within 50 epochs, I increased the number of epochs to 70 and added a weight-decay parameter equal to $1.2 \cdot 10^-6$. The improvement of applying NT-ASGD during training seems to be effective and the Perplexity score still decreases by around 1.5 points on both sets. The final results obtained are 79.4 PPL on the validation set and 78.9 PPL on the test set. In table 4, the performances of each model are reported

Model	Train PPL	Valid PPL	Test PPL
Vanilla LSTM	90.8	147.8	145.5
Embedding LSTM	44.7	96.1	94.7
Emb-Var LSTM	76.9	86.8	86.5
Emb-Var-WD LSTM	69.2	80.8	80.5
AWD-LSTM	65.9	**79.4**	**78.9**

Table 4: *Results on validation and test sets*

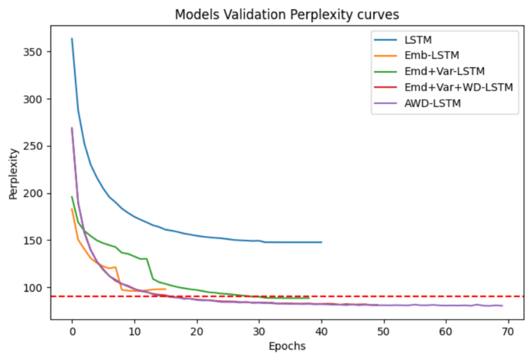

Figure 1: *Models’ perplexity curves*

As said before, each model is also evaluated through a ”subjective metric” by generating a sentence given a common input. I started with a single word and then complex the starting sequence more and more.

Here are some results:

Input word *"the"*:

- **Vanilla LSTM**: *"the company said the company is expected to acquire N N in the third quarter from N billion"*;
- **Embedding LSTM**: *"the company said it will build a new line of the new company <eos>"*;
- **Emb-Var LSTM**: *"the company said it expects to post a third-quarter loss of N or N a fully operational N <eos>"*;
- **Emb-Var-WeightDrop LSTM**: *"the company said it expects to report a loss for its fourth quarter ending sept. N <eos>".*
- **AWD LSTM**: *"the company said it will sell its N N stake to the company 's N million common shares <eos>".*

Using a one-word input we can notice that all the models are able to generate a good sentence and the general argument of the generated sequences appears quite similar among the models (all of them write about "a company").

Input words *"new york"*:

- **Vanilla LSTM**: *"new york stock closed yesterday at N up from the previous quarter ended oct. about half the price level at"*.;
- **Embedding LSTM**: *"new york stock exchange volume was N shares up from N million shares friday and at about half the trading"*.;
- **Emb-Var LSTM**: *"new york city <eos>".*
- **Emb-Var-WeightDrop LSTM**: *"new york stock exchange volume totaled about N million shares <eos>".*
- **AWD LSTM**: *"new york stock exchanges were suspended in a trading session in the past N days <eos>"*

With a two-word input, we have the first curious case. In some cases, the Embedding and Variational dropout LSTM is not able to generate a sentence given the "new york" input. This is quite surprising, considering that it is the first model to improve the baseline and I expect that it performs better with respect to the previous ones. Moreover, it looks like the vanilla LSTM can output a better sentence with respect to the Embedding LSTM. The last two models are instead very good.

Input words *"he said many of the countries"*

- **Vanilla LSTM**: *"he said many of the countries will be a good time for a few years and the market 's own"*;
- **Embedding LSTM**: *"he said many of the countries are likely for the sale of the company <eos>"*;
- **Emb-Var LSTM**: *"he said many of the countries will have to be able to make the new products <eos>"*;
- **Emb-Var-WeightDrop LSTM**: *"he said many of the countries have been working on the u.s. in a joint meeting with mr. bush and the administration 's chief executive".*
- **AWD LSTM**: *"he said many of the countries are still being forced to keep the u.s. trade gap in N <eos>*

Complexing the output more we finally find a meaning to the numerical results of the models. The vanilla LSTM outputs almost a correct sentence, but the other models are capable of going beyond and generating sentences that are not comparable with the vanilla one in terms of complexity and correctness.

6. Conclusions

In conclusion, we can say that a Language Model task can be solved using an LSTM model properly regularized. In particular, looking at the structure of an LSTM we can distinguish different layers having different subtasks i.e. the embedding layer has to learn the best embedding of each word, the classification layer has to predict the next word, etc... This might be the reason why we need ad-hoc dropout techniques that require some fine-tuning.
The most sensitive part to regularize has revealed the recurrent layers and this is understandable because the gates along with their weights manage all the memory of the network and the capability of the model to retain longer sequences. For that reason, the first layer must remain non-dropped, otherwise, the memory mechanism of the network gets affected by consequences that worsen the performance.
Considering the generation of sentences, we can conclude that all the models are capable of outputting sequences having a certain grade of meaning. In particular, we can say that at an inference level, the PPL score does not explain everything and it is not guaranteed that a better model can generate better results with respect to a worse one. But, this is valid only for the easier and shorter inputs, and as we give a more complex and longer sequence as a starting point the best models can clearly generate better outputs.

7. References

[1] Stephen Merity, Nitish Shirish Keskar, Richard Socher “Regularizing and Optimizing LSTM Language Models“, 2017

[2] Toma´s Mikolov, Stefan Kombrink1, Luka´s Burget, Jan “Honza” Cernock, Sanjeev Khudanpur “EXTENSIONS OF RECURRENT NEURAL NETWORK LANGUAGE MODEL“, June 2011.

[3] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus “Regularization of Neural Networks using DropConnect“, 2013.