Language Modeling with RNN

Laura Corso (mat. 230485)

University of Trento

laura.corso@studenti.unitn.it

1. Introduction

This document is the report for the final project of the course *Natural Language Understanding* (year 2022) at University of Trento. The aim of this project is to develop a Recurrent Neural Network to deal with the task of *Language Modeling*.

To realize the task, two different RNN models are developed: a baseline model and an improved model. The latter is an improvement of the former one through the application of the following regularization techniques:

- dropout on the hidden-to-hidden connections of the RNN
- Variational dropout (as described in the work by Merity et al. [1])
- L-2 weight decay

2. Task Formalisation

Language Models are a core part of the *Natural Language Understanding* area of research. In fact, they permit to develop algorithm for automatic completion that can be used in text editors, chat apps and even in automatic speech recognition, where these models are used to predict the most likely sentences from the input audio signal. Given the central role of these models, very different techniques has been developed during the recent years in order to deal with this task.

In particular, the research focus on *Statistical Models* which try to predict the next word given its *context* i.e. the preceding words. Therefore, the task of language modeling is approached as a *density estimation* problem. Specifically, given a corpus of tokens $X = (x_1, ..., x_t)$, the joint probability $P(X)$ factorizes as $P(X) = \prod_t P(x_t|x_{<t}) = \prod_t P(x_t|C_t)$, where $C_t = X_{<t}$ is referred to as the *context* of the conditional probability.

Initially, N-gram models were proposed, but the underlying *Markov assumption* limits their ability of prediction. In fact, the last few words (one for unigram models, two for bigram models, three for trigram models, etc.) before the one to predict are not enough in order to model the conditional probability $P(x_t|x_{<t})$ successfully.

To improve the aforementioned assumption, in the last few years, *Neural Networks* were introduced. First, *Feed Forward Networks* were implemented but then a huge revolution happened with the introduction of *Recurrent Neural Network*. These models permit to retain some sort of knowledge about the past inputs and use it to improve the prediction on the current input. In the scope of language modelling this characteristic is very important. In fact, it permits to extend the context for the prediction of the next word not only to the last few preceding words, but to a very long sequence of previously seen terms.

Obviously, given the high power of these models, in order to avoid overfitting, some regularization techniques has to be introduced in the processing pipeline. Regarding this topic, very different approaches were proposed along the years in order to reach the state-of-art performance such as Merity et al. [1] or Melis et al. [2].

In this work an *LSTM Language Model* is developed using some regularization strategies such as dropout on the connections between the recurrent hidden layers of the network and Variational dropout [1], in order to improve a baseline LSTM model.

3. Data Description & Analysis

In order to train, evaluate and test the model described in this report, the *Penn Tree bank* dataset is used. The .zip file is downloaded from the link [3].

In the unzipped folder six .txt files are available:

- *ptb.train.txt* which contains the training set;
- *ptb.valid.txt* which contains the validation set;
- *ptb.test.txt* which contains the test set;
- *ptb.char.** which contain the same data of the above mentioned files rewritten as a sequences of characters, with spaces rewritten as _ .

The first three dataset files contain the words grouped in sentences and each sentence is located on a new line of the file. These files has been preprocessed by the author of the link [3]. In fact, in each file the $< unk >$ character is already present. In addition, no *Out Of Vocabulary words* are present neither in the testing set nor in the valid set with reference to the train set. Moreover, the training set contains more words than the other two datasets.

The statistic about the three datasets are the following:

Dataset	Num. words	Num. sentences	Num. unique words
training	887521	42068	10001
valid	70390	3370	6023
testing	78669	3761	6050

where 'Num. unique words' is the number of words without repetitions.

In order to use these data in the RNN model:

1. First, a two way vocabulary of the training set is built as $\{word : id\}$ and $\{id : word\}$. Consequently, all the words contained in the training set are mapped, without repetitions, to an integer value and viceversa. This is done because the Neural Network takes as input only *Tensors* that can contain only numerical data.
2. Second, the sentences of the three datasets are converted with the previously computed word ids. In this way, the three datasets shares the same id for the same word and the model is able to generalize between them.
3. Third, the sentences, that now contain word ids and not words, are grouped into batches of uniform size along the horizontal size. This means that each batch contains the same number of sentences. Instead, all the sentences in

a batch have the same dimension but this is not uniform along batches. Therefore, each batch is a matrix of shape $[n\_sentences, length]$, where $n\_sentences$ is constant along batches and $length$ vary across them.

To equalize the size of the sentences in a batch, a *right-padding* procedure is applied. Therefore, the length of the longest sentence in the batch L is extracted, and, to all the other sentences, a number of $< pad >$ character is added to reach the L size.

4. Models

As previously mentioned, in this project two models are implemented: a baseline model (following referred to as $baseline$) and a model that improves it with some regularization techniques (following referred to as $improved$).

The $baseline$ is composed of the following layers:

- an *embedding layer* that embed each word in a vector of size 300
- a one layer *Long Short-Term Memory*
- a *linear layer* to map the embedded inputs to the corresponding word ids.

To the LSTM is fed as input each sentence without the padding. This is done in order to reduce the computation time. Then, the padding is re-applied to the output.

This model is trained, validated and tested for five times, each with 30 epochs. This permit to obtain an averaged test loss between runs since the training corpora is small.

In addition, every five epochs a test on the valid dataset is performed and, if after 3 epochs, the resulting loss is still deteriorating the training is early stopped.

The initial weights of the LSTM are initialized with a *Xavier initialization* for the input-to-hidden connections and with an *orthogonal initialization* for the hidden-to-hidden connections.

In order to train this network a mean *Cross Entropy Loss* along batches is used with the *Adam* optimizer using a learning rate of 0.0001.

The $improved$ is composed of the following layers:

- an *embedding layer* that embed each word in a vector of size 300
- a three layer *Long Short-Term Memory* with dropout between hidden-to-hidden connections
- a *Variational dropout* layer ([1])
- a *linear layer* to map the embedded inputs to the corresponding word ids.

As for the $baseline$, to the LSTM are fed un-padded inputs and the corresponding padding is re-done on the output.

Equally, this model is trained, validated and tested for five times but with 40 epochs each. In fact, with the adding of two layers, the number of parameters of the network increases. Therefore, in order to converge to the minimum value of *Cross Entropy Loss* the *Adam* optimizer needs more time.

In addition, differently from $baseline$, in this model no early stopping is applied. Instead, a *Scheduler* for an *adaptive learning rate* is implemented. In particular, this decreases the learning rate of a 0.1 factor if for 5 epochs no improvement in the loss value (in the range of $best - 0.1$) is observed. The initial learning rate is set to 0.001.

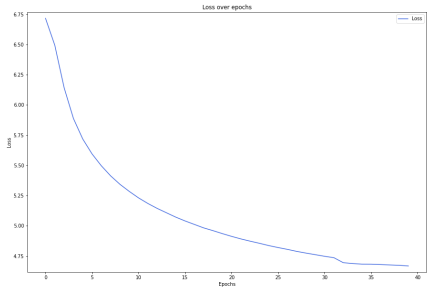

Figure 1: *Training losses over the epochs of the final run*

The initial weights of the LSTM are initialized as $baseline$ and, moreover, in this model a zero initialization of the initial hidden states is performed. This is done in order to prevent the network to use pre-initialized hidden states between different runs and different epochs augmenting the robustness of the model.

As an additional regularization strategy, a *L-2 weight decay* is applied with a regularization factor of 0.00000000001.

The *Variational dropout* [1] is applied to the embedded input and to the padded output of the model.

Some experiments have been done also adding the *Embedding dropout* from Merity et al. [1] but the additional parameters to train only worsen the performance of the model depicted above.

5. Evaluation (approx. 400-800 words)

As previously mentioned, the $improved$ improves the performance of $baseline$. In fact, the following results are obtained:

Model	Avg Loss	Avg Perplexity
baseline	5.1729	176.429
improved	5.0649	158.367

where 'Avg Loss' is the average loss across runs and 'Avg Perplexity' is the average perplexity across runs.

As it's possible to notice, even if the average losses are not so different between $improved$ and $baseline$, the former is able to gain 18.062 points over the perplexity of the baseline. This proves that the applied regularization techniques work.

In addition, from the ***Fig. 3*** and the ***Fig. 2*** it's possible to observe that the training procedure proceed smoothly towards the minimum value of both the loss and the perplexity.

From the execution it's also possible to extract the words with the highest and lowest perplexity. The results are:

- The word with the highest perplexity (131857360.33099522) is: by
- The word with the lowest perplexity (8.785339014139026e-05) is: unk.

Therefore, 'by' is mispredicted most of the time. Instead, 'unk' is predicted correctly almost every time. This can be due to the fact that 'unk' occurs very often both in the test set (4794 occurrences) and in the output of the network, consequently, most of the predicted 'unk' token are predicted correctly because the model is able to learn the underlying probability distribution of this token. Instead, regarding the token 'by', it occurs only 468

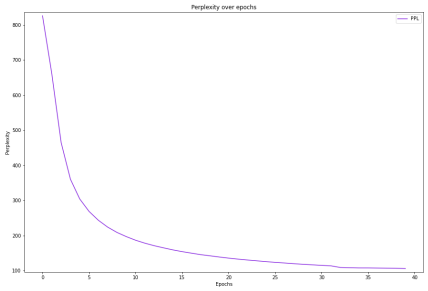

Figure 2: *Training perplexities over epochs of the final run*

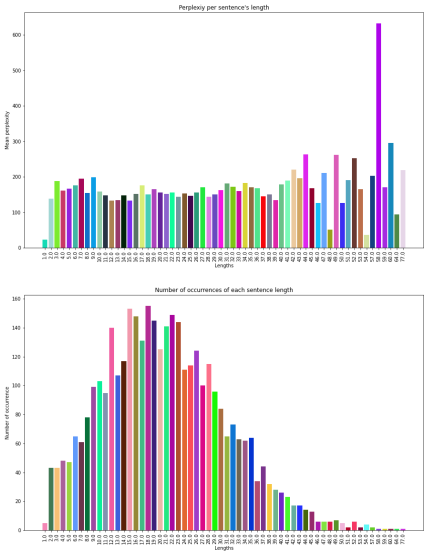

Figure 3: *Length of sentences vs Sentence perplexity & Length of sentences vs Occurrence of sentence's length*

times in the testing set, therefore, the model it's not able to learn its probability distribution.

Moreover, from ***Fig. 3*** it's possible to appreciate how the sentence with length 58, which occurs only one time, is the one with the higher perplexity (631.9688). Instead, the length that occurs more often is 18 with 155 occurrences and the length with the lowest perplexity (22.6760) is 1.

6. Conclusion

From the above described results it's possible to conclude that the regularization of Neural Network models is a very important procedure in order to boost the performances on the Language Modelling task.

In particular, the application of the described two types of dropout prevent the overfitting of the network thanks to their ability to deactivate some hidden-to-hidden connections or some inputs/outputs in a forward-backward pass. These deactivations push the network to learn at every training step redundant representations of the the input allowing a more precise prediction of the output.

In addition, the adaptive learning rate allows to perform a fine-grade search of the optimal loss value further improving the final predictions.

7. References

[1] 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

[2] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," 2017. [Online]. Available: https://arxiv.org/abs/1707.05589

[3] [Online]. Available: https://deepai.org/dataset/penn-treebank