Implementation of a Language Model using Weight-Dropped LSTM with Non-monotonically Triggered ASGD

Luca Miotto (mat. 229363)

University of Trento

luca.miotto-1@studenti.unitn.it

1. Introduction

Language modeling is a common task in the field of Natural Language Understanding which consists of assigning probabilities to n-grams (*i.e.,* sequences of words) [1]. The aim of this project is to implement a language model by using an LSTM-based architecture. More specifically, our architecture implements a Weight-Dropped LSTM with Non-monotonically Triggered Averaged SGD (abbreviated by the authors to AWD-LSTM for ASGD Weight-Dropped LSTM [2]). This architecture is a multi-layer LSTM which leverages a number of regularization techniques, including (but not limited to) several dropout strategies, weight decay and gradient clipping. In this project, we implement some of these regularization techniques and provide a comparison by training, validating and testing on the well-known Penn Treebank (PTB) dataset. Moreover, we implement an early-stopping mechanism in order to detect overfitting and thus avoid unnecessary waste of computational time on our limited resources.

2. Task Formalisation

According to a definition from [1], the language models (LMs) are models that assign probabilities to sequences of words. A sequence of *n* consecutive words is called an *n-gram.* In order to compute such probabilities, we can apply the chain rule of probability:

$$P(w_1, ..., w_n) = P(w_1) \cdot P(w_2|w_1) \cdot ... \cdot P(w_n|w_{1:n-1}) \quad (1)$$

$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \quad (2)$$

This formulation makes it easier to compute the joint probability, since it can be broken down into products of conditional probabilities. However, as soon as the sentences are long enough, the problem of keeping a record of all possible sequences of words becomes infeasible. As a consequence, an *n-gram model* approximates the history by only taking into account the last few words. Such an approximation is actually implying a relaxed formulation of the Markov Property to hold on the words probability space. Instead of considering the whole sequence, we are approximating the conditional probability of the next word by only considering the latest previous $N-1$ words. This can be formalized as follows:

$$P(w_k|w_{1:k-1}) \approx P(w_k|w_{k-N+1:k-1}) \quad (3)$$

For example, when $N = 2$ we are approximating with *bigrams.* Similarly, when $N = 3$ we have *trigrams.* Therefore, eq. (2) can be approximated accordingly:

$$P(w_1, ..., w_n) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1:k-1}) \quad (4)$$

Concretely, these probabilities can be estimated through maximum likelihood estimation (MLE), which can be obtained via a normalized count of the *n-grams*:

$$P(w_k|w_{k-N+1:k-1}) \approx \frac{C(w_{k-N+1}, ..., w_k)}{\sum_w C(w_{k-N+1}, ..., w_{k-1}, w)} \quad (5)$$

$$= \frac{C(w_{k-N+1}, ..., w_k)}{C(w_{k-N+1}, ..., w_{k-1})} \quad (6)$$

Where the sum in eq. (5) is performed over all possible (known) words w following the $(w_{k-N+1}, ..., w_{k-1})$ sub-sequence, and can thus be reduced to the count of the prior.

Another effective method to approximate this conditional distribution can be achieved by applying a neural language model, thanks to the property of neural networks of being universal function approximators. Moreover, neural language models can usually generalize better to unseen data, since they represent the entire prior context as an *embedding*. This behavior encourages the language model to pursue the hypothesis that – although the target word w_k has never been observed with the given prior $(w_{k-N+1}, ..., w_{k-1})$ – the model should assign a high enough conditional probability to w_k anyway, if this word has appeared in the past along with similar priors (*i.e.,* priors with similar embeddings). Further details about the model will be given in section 4.

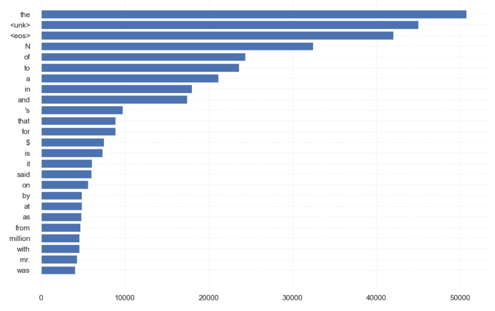
3. Data Description and Analysis

As already mentioned in section 1, we used the Penn Treebank (PTB) dataset for training, evaluating and testing our model. The training set was made of 42068 lines for a total of 10000 distinct words. Both the validation and the test set contained no out-of-vocabulary (OOV) words with respect to the training set. The validation set was made of 3370 lines. The test set was made of 3761 lines.

In figure 1, we plot the frequency distribution of the top-25 most common words in the traning set. Here, $< unk >$ represents an unknown word (which is pretty often a proper noun), $< eos >$ represents the end of a sentence, and N substitutes a generic number in the corpus.

Aside from language modeling, annotated versions of the PTB dataset are used for other common NLP tasks such as constituency parsing (*i.e.,* extracting a constituency-based parse tree of the sentence), dependency parsing (*i.e.,* analysing the dependency relationships between the words in a sentence) and part-of-speech (POS) tagging.

Figure 1: *Top-25 words in training set*

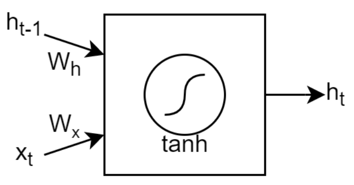


4. Model

In section 2, we briefly introduced the neural networks as a possible solution for language modeling. More specifically, among the plethora of available neural network architectures, we argue that RNNs are a very reasonable choice, due to their capability of retaining state information along an input sequence and producing an output which is dependent on both the current input (*i.e.,* the current embedded word) and the retained state. In a more formal fashion, we can describe the hidden state and output computation processes for a basic ("vanilla") RNN as:

$$\begin{aligned} h_t &= tanh(W_x x_t + W_h h_{t-1}) \\ y_t &= softmax(W_y h_t) \end{aligned} \tag{7}$$

Figure 2: *Vanilla RNN cell for hidden state update. Note that, for language models x_t usually will not be the raw word, but the result of the embedding on that word.*



Unfortunately, RNNs are known to be extremely sensitive to both vanishing and exploding gradients. Also, Back-Propagation Through Time (BPTT) becomes a very computationally expensive operation as soon as the length of the input sequence grows. This is due to the back-propagation of the error along the unfolded computational graph for the weights update. Here we show a simplified example for the hidden state weights:

$$\frac{\partial \mathcal{E}_t}{\partial W_h} = \sum_k \frac{\partial \mathcal{E}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \cdot \frac{\partial h_k}{\partial W_h} \tag{8}$$

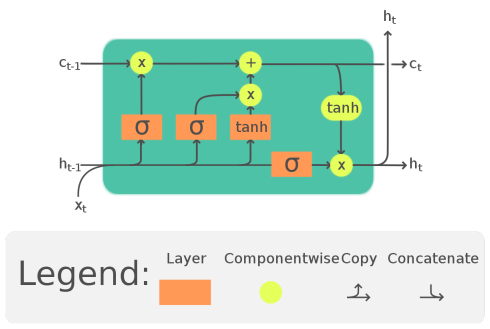
Where index k refers to all the time steps and t is the current time step.

While this latest issue can be partially overcome by applying Truncated BPTT, the vanishing and exploding gradient problems require more specific regularization techniques. Also, due to the rapid decrease of the values through time, we will need more complex network architectures explicitly designed to manage the task of retaining relevant state information for longer sequences and forget information no longer needed. One architecture with these characteristics might be the Long Short-Term Memory (LSTM) network, which we define as:

$$\begin{aligned} g_t &= tanh(W_g s_t) & \\ i_t &= \sigma(W_i s_t + b_i) & \text{input gate} \\ f_t &= \sigma(W_f s_t + b_f) & \text{forget gate} \\ o_t &= \sigma(W_o s_t + b_o) & \text{output gate} \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t & \text{cell state} \\ h_t &= o_t \odot tanh(c_t) & \text{hidden state} \end{aligned} \tag{9}$$

Where $s_t = \binom{x_t}{h_{t-1}}$ is the couple of current word embedding and previous hidden state, while $\odot$ is the Hadamard product.

Figure 3: *LSTM network. Available online at [5].*



4.1. AWD-LSTM

The specific architecture we chose for this project is called AWD-LSTM (for ASGD Weight-Dropped LSTM) [2]. As the name suggests, this architecture makes use of the Averaged SGD (ASGD).

4.1.1. NT-ASGD

"Vanilla" SGD updates the parameters of the model (in this case, the weights of the network) simply by applying:

$$w_{t+1} = w_t - \eta \nabla_w \mathcal{L}(y, f_w(x)) \tag{10}$$

Where η is the learning rate, and the gradient is computed on the loss resulting from the current prediction, performed on the model parameterized with the current set of weigths w_t.

ASGD improves this optimization method by assigning to the weights the average value returned by eq. (10) over a window of width T, that is the average of the T most recent values:

$$w_{t+1} = \frac{1}{t - T + 1} \sum_{k=T}^{t} \hat{w}_k \tag{11}$$

In particular, here we implement Non-monotonicly Triggered ASGD (NT-ASGD). Instead of switching from "vanilla" SGD to ASGD after a fixed number of steps T – which should be tuned for each specific application – this method uses T both as a lower bound for the epochs, and as a width for the window of recent validation losses that are taken into account. This window of validation losses is in fact the main trigger for switching to ASGD. The switch is triggered when the current validation loss happens not to be smaller than all the T previous ones.

4.1.2. *Weight-Dropped LSTM*

In addition, this architecture implements a method known as DropConnect [4], which is a generalization of the dropout technique such that all connections (instead of only the outputs) can be dropped with a given probability $1 - p$.

4.2. Regularization Techniques

Along with the AWD-LSTM architecture, we followed the lead of [2] and implemented some of the additional regularization techniques they proposed in their paper and implemented in their code, which is publicly available at [6].

4.2.1. *Weight Tying*

Weight tying consists in sharing the weights between the word embedding layer and the softmax layer. This technique allows us to reduce the number of parameters and removes the problem of learning a one-to-one correspondence between input and output, which would be much more difficult when the last layer has a different (usually, much bigger) size than the word embedding.

4.2.2. *Dropouts*

The following dropouts were applied:

- *Embedded Dropout*: creates a mask to apply to the embedding object before actual emebedding is performed.
- *Input Dropout*: applied to the word embeddings before they're passed as inputs to the model.
- *Hidden Dropout*: applied to the outputs of an LSTM layer before they're passed as inputs to the next one.
- *Layer Dropout*: applied to the outputs of the last layer.

4.2.3. *Gradient Clipping*

Gradient clipping is a regularization technique that aims at reducing vanishing and exploding gradients by limiting their absolute value up to a given threshold.

4.2.4. *Other techniques*

We also implemented two more techniques. The first one – even if not explained in [2] – can be found in their public code [6] as an option. This consists in reducing the learning rate by a given factor at some previously established epochs, in order to delay the activation of the ASGD mode. The second one is called *Early Stopping*, it is a regularization technique that interrupts that training loop after the validation loss has increased for a given consecutive amount of times (known as *patience*).

5. Evaluation

In this section we discuss the performances of the model on the validation and test sets.

5.1. Metrics

The metrics used to evaluate the model were the *cross-entropy loss* for the loss estimation, which have been later used to compute the *perplexity*, which is probably the most widely accepted evaluation metric for language models.

The *perplexity* of a language model on a test set can be seen as the normalized inverse probability of the test set:

$$PP(W) = P(w_1, ..., w_N)^{-\frac{1}{N}} \tag{12}$$

$$= \sqrt[N]{\frac{1}{P(w_1, ..., w_N)}} \tag{13}$$

$$= \sqrt[N]{\prod_{k=1}^{N} \frac{1}{P(w_k|w_{1:k-1})}} \tag{14}$$

This way, to a high conditional probability is assigned a low perplexity. Thus, minimizing the perplexity is equivalent to maximizing the probability of the language model to behave correctly on the test set.

5.2. Hyperparameters

For model training and validation we used several hyperparameters. We decided to train up to 100 epochs (many attempts stopped before though, due to the early-stopping mechanism). It was used a batch size of 40 for the training, 10 for the validation and 1 for the testing (*i.e.,* one sequence at a time). About the model, the size of the word embedding was 400, the number of hidden units per layer was 1150 and the layers were 3. The dropout constants were (0.4, 0.3, 0.4, 0.1) for input, hidden-to-hidden, output and embedding respectively. We used a constant weight of 0.5 for the weight-dropping mechanism. The optimizer is run with a 1.2e-16 weight decay and an initial learning rate of 30. The constant T for the NT-ASGD was set to 5. The gradient clipping threshold is 0.25. The patience for the early-stopping is of 10 epochs. Finally, the learning rate reduction was applied at epoch 13 with a factor of 10 (*e.g.,* learning rate reduced from 30 down to 3).

5.3. Validation and Testing

We tried different approaches by removing some of the regularization techniques we discussed in subsection 4.2. In particular, the learning rate reduction has been applied only in the first attempt, while all the other attempts did not use it. Except for the learning reduction and the currently tested regularization technique, all the parameters remain the same between one attempt and the others.

Figure 4: *Validation perplexities using different regularization techniques.*

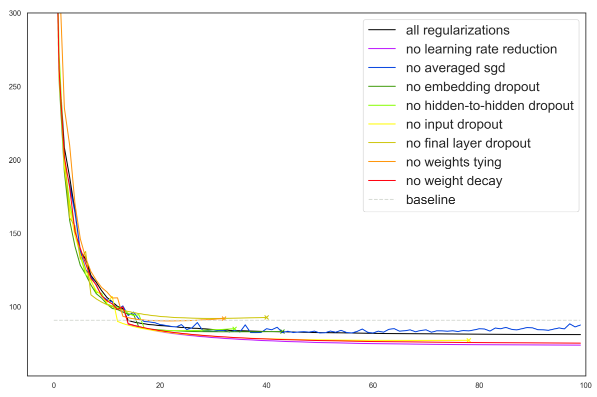
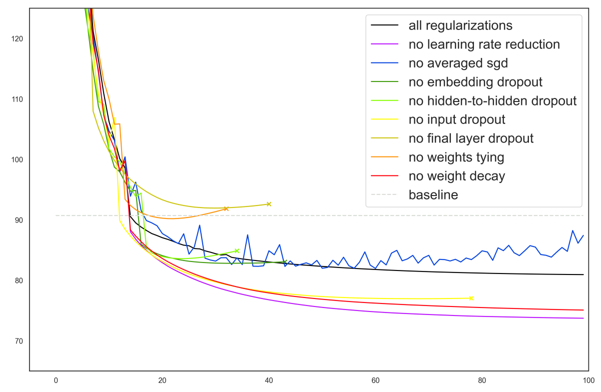


Figure 5: *A closer look around the baseline.*



The figures above show clearly how important it is the dropout in all positions along our architecture. In fact, all the attempts without a dropout did not reach the end of the training phase (*i.e.,* early-stopping applied). This evidence seems to corroborate the importance of the dropout in order to prevent overfitting.

Another relevant observation is about the ASGD. The only attempt where ASGD was never applied started oscillanting after about 20 epochs and kept behaving irregularly ever since.

Another interesting fact is provided by the first attempt, where all of the regularizations were applied (*i.e.,* the only with the learning rate reduction applied manually after a number of epochs). As you can see, retarding the activation of the ASGD does not seem to have a good impact on our architecture, that struggles to reduce perplexity after about 40 epochs.

Finally, we observe that the regularization with the smaller impact in this test was clearly weight decay. This is probably due to the fact that a very small weight decay was set. Otherwise, we can assume that the lack of weight decay should have a greater impact on the performances. Another relevant observation about it is that the red and the purple lines follow a very similar trajectory for many epochs, but start diverging right after the activation of the ASGD. This might be due to the fact that a greater weight decay will penalize greater weights more, and thus the model might be more slowed down when running with ASGD and no weight decay, since in this scenario it seems fair to assume that there will be more irregular weights to average.

After each training-validating routine, the best model for each attempt was saved. These models were used for testing:

regularizations	test ppl
all regularizations	77.891
no lr reduction	**71.787**
no averaged sgd	79.817
no embedding drop	80.397
no hidden drop	81.062
no input drop	74.122
no layer drop	89.184
no weights tying	87.691
no weight decay	71.931

We can see that the resulting perplexity for the best overall model (which turns out being the one with all the regularizations except for the learning rate reduction) on the test set is by far smaller than the baseline. And also the other attempts showed good results.

6. Conclusion

In conclusion, the AWD-LSTM proved once again to be a valid architecture for language modeling. Among the tested regularization techniques, the ones that proved to be more essential are the dropouts on the input and on the final (output) layer respectively. As concerns the learning rate reduction, it seems safe to say that it's probably not helpful, although it seems plausible that – after a thorough tuning of the epochs and reduction factors – the ASGD activation delay might actually end up improving the overall performance.

7. References

[1] D. Jurafsky and J. H. Martin, “Speech and Language Processing,” *Pearson*, 2013.

[2] S. Merity, N. S. Keskar and R. Socher, “Regularizing and Optimizing LSTM Language Models,” 2017.

[3] I. Goodfellow, Y. Bengio and A. Courville “Deep Learning,” *The MIT Press*, 2016.

[4] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, R. Fergus “Regularization of Neural Networks Using Dropconnect,” 2013

[5] “Long short-term memory,” *Wikipedia*, 2022, `https://en.wikipedia.org/wiki/Long_short-term_memory`

[6] “LSTM and QRNN Language Model Toolkit,” *GitHub*, 2018, `https://github.com/salesforce/awd-lstm-lm`