# NLU course projects - Assignment 1 - LM

*Davide Cavicchini (247823)*

University of Trento

davide.cavicchini@studenti.unitn.it

## 1. Introduction

This is a report for the first assignment of the Natural Language Understanding Course. The objective is to implement various techniques that can enhance the performance of a simple language model for next-word prediction and to understand how each of these techniques influences the model.

I have experimented with the model architecture, substituting the RNN with an LSTM block, using different optimizers, and applying different regularization techniques such as Dropout, Weight Tying, Variational Dropout, and Non-Monotonically Triggered Average SGD.

## 2. Implementation details

This section will briefly touch on the architectures and techniques implemented for this assignment.

### 2.1. Architecture

The implemented architectures differ in the encoder for the text, one uses a simple RNN module while the other an LSTM, both already implemented by Pytorch.

### 2.2. Regularization

To improve the results, I implemented some regularization techniques, which should boost the generalization capabilities of the model and allow it to train longer.

Dropout forces the network to develop independent meaningful representations of the data without relying too much on putting together small units of information from multiple neurons. I placed one after the embedding layer and another before the output decoder. To implement it I used the standard pytorch module.

Weight tying significantly lowers the number of parameters of the network. In particular, by tying together the embedding and decoder parameters it ties the meanings extracted by the word embeddings to the parameters used for decoding, therefore extracting from the vocabulary the word with the highest similarity to the encoder representation. To implement it the embedding and linear layers need to coincide in shape, and use the same tensor.

Variational dropout uses the same dropout mask for a particular sequence, therefore avoiding updating the update of some parameters for the whole training sample. To implement it, it's enough to reuse the same mask for the entire sequence in a batch.

### 2.3. Optimizers

SGD and AdamW can be easily swapped from one another, the latter uses more advanced techniques enabling it to converge faster to a solution.

Since SGD tends to "jump around" the idea of AvSGD is to average over these jumps. The NT variation starts the averaging depending on the model performance on the validation set. I used a custom optimizer which inherits from SGD for the gradient computations, then implemented the logic from the pseudocode available in this paper [1].

## 3. Results

### 3.1. Architecture

As we can see from table 1, using a more complex model that mitigates the vanishing gradient problem has, as expected, a positive effect on the PPL scores.

### 3.2. Regularization

Among the techniques tested, regularization had the most significant impact on the results. It allows the model to be trained longer without triggering the patience threshold, allowing for better results at the cost of slower learning. This is evident from the reported training epochs for each model in table 1 and from the PPL and loss graphs in Figure 1 and 2. The tensorboard files are available in the assignment folder to view them better.

### 3.3. Optimizers

As hinted earlier, the AdamW optimizer helps the model to converge faster and to a solution with lower test PPL than the one achieved with the SGD optimizer. This is evident in table 1.

For NT-AvSSG, I observed that this technique is more resilient to over-fitting, as demonstrated by the PPL (perplexity) on the development set exhibiting a smooth behavior at later stages of training in Figure 3. This allows the models to be trained for longer without triggering the patience. Strangely, this technique obtains the best result among the models for the development set; while for the test set, it worsens the results slightly as shown in table 2. Lastly, looking at the performance improvement compared to the standard non-averaged model becomes more pronounced as the learning rate increases. This is to be expected since, as explained in section 2.3, the idea is to smooth out the jumps at later training iterations, which are more prominent with higher learning rates.

### 3.4. Learning Rate

I experimented with various learning rates and observed that integrating more regularization techniques during training enabled me to increase the learning rate, leading to quicker and more stable results, particularly when using NT-AvSGD as evident in Figure 3.

## 4. References

[1] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *ArXiv*, vol. abs/1708.02182, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:212756

| Part | Model Name | Learning Rate (LR) | epochs | Dev PPL | Test PPL |
|---|---|---|---|---|---|
| 1.0 | RNN | 0.5 | 55 | 169.3 | 162.0 |
| 1.0 | RNN | 1.0 | 36 | **167.8** | **160.6** |
| 1.0 | RNN | 2.0 | 29 | 172.6 | 162.7 |
| 1.1 | LSTM | 0.5 | 107 | 159.4 | 152.7 |
| 1.1 | LSTM | 1.0 | 63 | **156.5** | 150.3 |
| 1.1 | LSTM | 2.0 | 39 | 157.8 | **148.3** |
| 1.2 | LSTM+drop | 0.5 | 179 | 144.9 | 139.5 |
| 1.2 | LSTM+drop | 1.0 | 195 | 125.9 | 122.1 |
| 1.2 | LSTM+drop | 2.0 | 112 | **124.8** | **120.2** |
| 1.3 | LSTM+drop(AdamW) | 0.001 | 40 | **122.9** | **114.06** |
| 2.1 | LSTM+WT | 0.5 | 199 | 143.8 | 137.9 |
| 2.1 | LSTM+WT | 1.0 | 119 | 133.5 | 130.5 |
| 2.1 | LSTM+WT | 2.0 | 68 | **130.5** | **126.8** |
| 2.2 | LSTM+WT+VD | 0.5 | 199 | 144.1 | 139.9 |
| 2.2 | LSTM+WT+VD | 1.0 | 199 | 114.9 | 112.3 |
| 2.2 | LSTM+WT+VD | 2.0 | 133 | 108.9 | 106.15 |
| 2.2 | LSTM+WT+VD | 2.5 | 84 | 112.4 | 109.9 |
| 2.2 | LSTM+WT+VD | 3.0 | 85 | **108.9** | **<u>106.0</u>** |
| 2.3 | LSTM+WT+VD(NT-AvSGD) | 0.5 | 199 | 144.5 | 144.4 |
| 2.3 | LSTM+WT+VD(NT-AvSGD) | 1.0 | 199 | 115.6 | 113.5 |
| 2.3 | LSTM+WT+VD(NT-AvSGD) | 2.0 | 199 | 105.1 | 106.8 |
| 2.3 | LSTM+WT+VD(NT-AvSGD) | 2.5 | 199 | 103.9 | **106.8** |
| 2.3 | LSTM+WT+VD(NT-AvSGD) | 3.0 | 185 | **<u>103.3</u>** | 107.5 |

Table 1: *Performance of trained models for each configuration.* ***bold*** *values represent the best model for each part, and* <u>*underlined*</u> *values the best overall.*

| Learning Rate (LR) | Delta Dev PPL | Delta Test PPL |
|---|---|---|
| 0.5 | +0 | +4.2 |
| 1.0 | +2.1 | +2.5 |
| 2.0 | -1.7 | +2.9 |
| 2.5 | -2.9 | +2.6 |
| 3.0 | **-4.1** | **+2.0** |

Table 2: *Delta on the PPL scores before and after averaging for the LSTM+WT+VD(NT-AvSGD) models on dev and test sets*

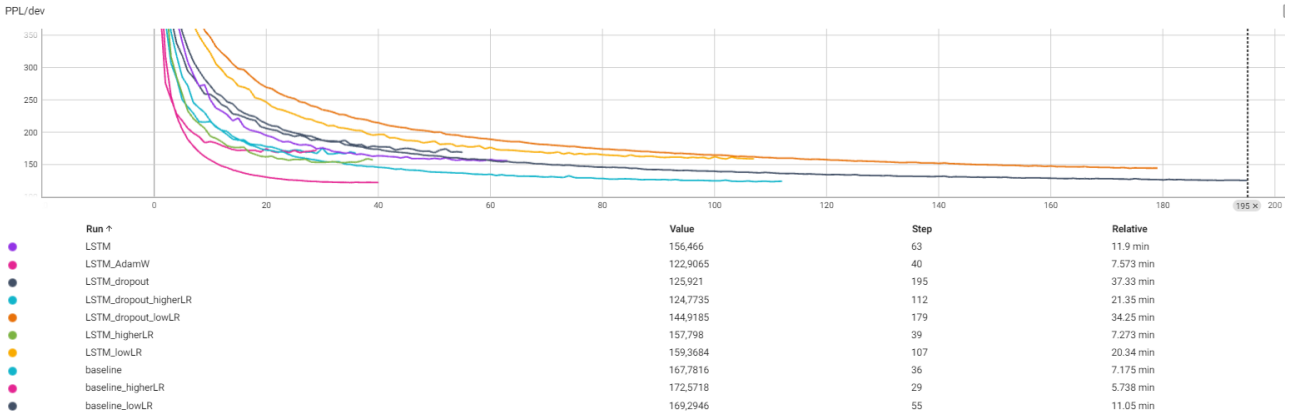


Figure 1: *PPL dev curves for models in part 1*

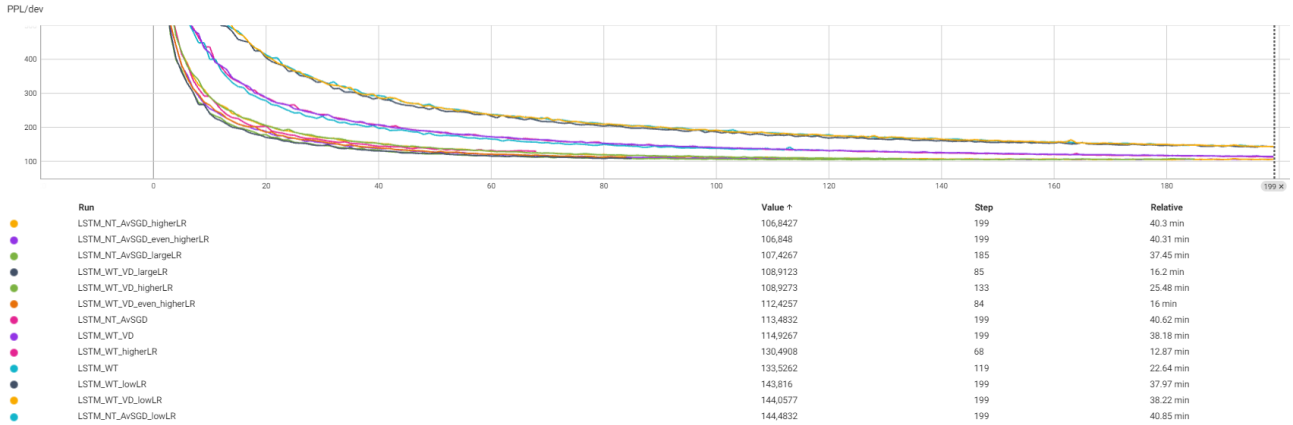


Figure 2: *PPL dev curves for models in part 2*

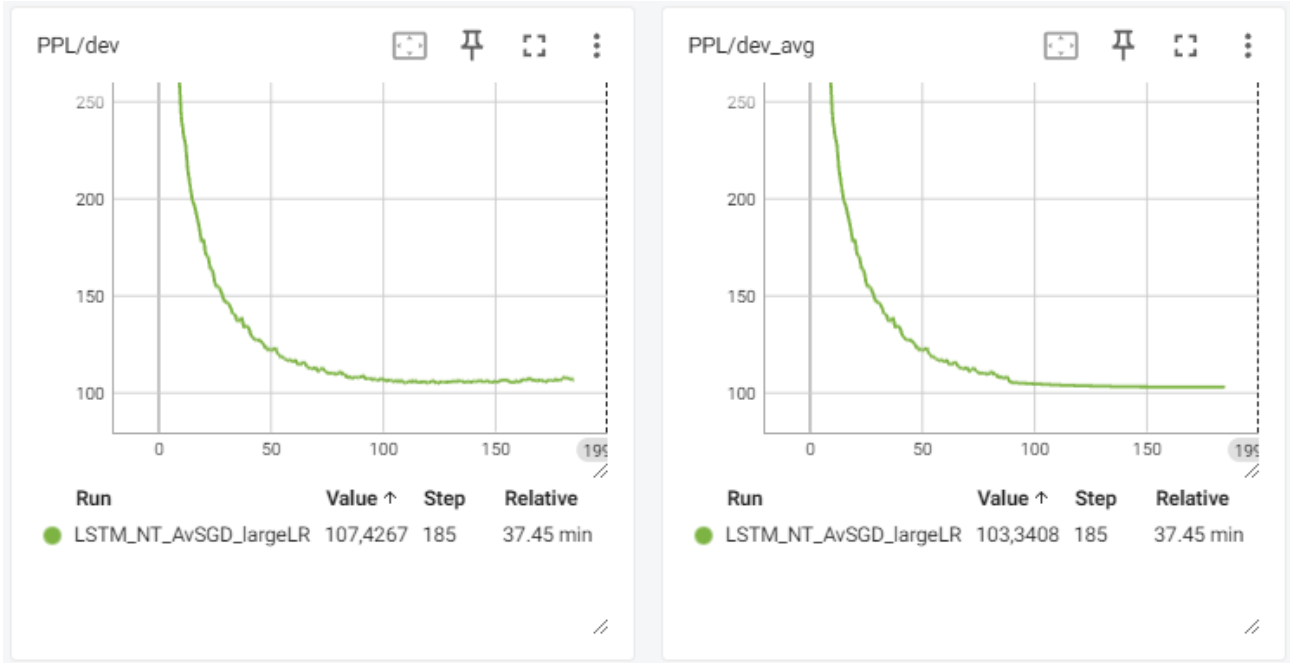


Figure 3: *PPL dev curve comparison between the normal and averaged model*