NLU course projects lab 4: LM

Nicola Muraro (248449)

University of Trento

nicola.muraro@studenti.unitn.it

1. Introduction

The purpose of this assignment is to implement a language model based on neural architectures. Specifically, the task of this model is to predict the next token given a set of tokens, referred to as the context. To achieve this goal, I used an LSTM on which several modifications were made to improve its performance. In the first part of the assignment, various optimisers were analysed alongside the use of dropout. In the second part, instead I implemented more sophisticated regularisation techniques described in the article written by Merity et al. [1]. This report briefly summarises the studies conducted and the results.

2. Implementation details

The only requirement for both parts of the assignment is that the perplexity must be below 250.

The first part of this assignment focuses on studying LSTM as the foundation of our architecture, followed by two incremental modifications: adding dropout layers and using AdamW. I started by replacing the original architecture (based on RNN) with LSTM to establish an initial baseline. Next, I made the first modification by adding dropout layers: one after the embedding layer and another before the output layer. In both cases, the most effective dropout value was 0.1. Finally, I implemented the last required modification by replacing SGD with AdamW. Since changing the optimiser also requires adjusting the learning rate, I made the necessary adjustments to the learning rate to ensure an optimal performance.

The second part involves starting from the initial network (LSTM with SGD) and incrementally applying regularisation techniques outlined by Merity et al. [1]. First, I implemented weight tying between the embedding layer and the output layer. This technique allows multiple layers in the network to share the same weights, aiming to regularise the entire architecture. To enable weight sharing, the layers involved must have the same dimensions, so I adjusted their sizes to ensure compatibility. The second modification involved using Variational Dropout, a technique that applies the same dropout mask. In practice, this means reusing the same dropout mask instead of sampling a new one at each model call. With this setup, the mask can be changed once per epoch and/or differ between layers. During my tests, I experimented with both approaches to determine the most effective configuration. Finally, I implemented the last modification: using Non-monotonically Triggered AvSGD. This is a variant of AvSGD where the switch to AvSGD is based on the network's state during training. Specifically, if SGD fails to improve the solution over several consecutive epochs, it is replaced with AvSGD in the hope of further optimizing the solution. Since the timing of this switch is not predetermined, this technique is expected to provide additional performance improvements compared to both standard SGD and AvSGD.

3. Results

The results used to evaluate the effectiveness of the implemented techniques were obtained using the evaluation set. This approach ensures better generalisation. Using the evaluation set to fine-tune hyper-parameters and find an optimal configuration allows for more reliable results on the test set. In general, the best machine learning models have high capacity and are properly regularised. Since many of the previously applied techniques are forms of regularisation, I started with a relatively large model. For all configurations, the batch size for the training segment was kept constant at 64, while for the evaluation it was set at 128.

In the first part, the initial model was created with the following specifications: $emb\_size = 600, hidden\_size = 500, lr = 1.5$. The use of dropout yielded better results with a probability of $p = 0.3$. Finally, after replacing the optimizer with AdamW, I adjusted the learning rate to 0.0005. As we can see from the results shown in table 1, only the first two modifications led to improved performance. We can observe that the addition of AdamW does not result in an improvement in performance on the validation set. However, it leads to an improvement on the test set. This might suggest better generalization by this optimizer, but not consistently across both datasets. Consequently, as it does not improve the values on the validation set, this modification was disregarded in the final model. For optimal generalization we cannot implement a modification based solely on its performance on the test set, since we generally would not have access to such data. The training process of the best model is depicted in Figure 1.

Model	Eval PPL	Test PPL
LSTM	144.09	139.51
LSTM with dropout	115.45	111.07
LSTM with dropout, AdamW	119.01	109.17

Table 1: *Perplexity values for the first part of the assignment*

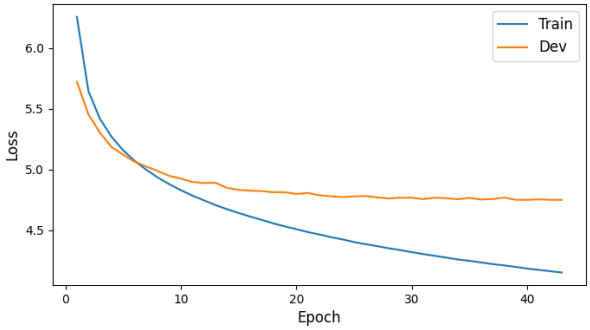

Figure 1: *Trainig loss for the best model of the first part*

For the second part, starting with the same configuration as before ($emb\_size = 600, hidden\_size = 500, lr = 1.5$), I modified the layer sizes to use weight tying. The model now has $emb\_size = hid\_size = 600$. Regarding Variational Dropout, I tested all four possible combinations for changing the mask: (1) one mask for both layers (fixed); (2) one mask for both layers (changed every epoch); (3) two masks (one per layer, fixed); (4) two masks (one per layer, changed every epoch). The best performance was achieved with the third option: using a different mask for each layer of dropout and changing it every epoch. Finally, as shown in table 2, I completed the assignment using NT-AvSGD. The last model switches to AvSGD only in the last few epochs, yet this still enables the model to reach a lower perplexity value. In Figure 2, the model's training progression is represented.

Tecnique	**Eval PPL**	**Test PPL**
W.Tying	130.78	128.35
W.Tying, Var.Drop.	129.43	125.85
W.Tying, Var.Drop., NT-AvSGD	124.56	122.22

Table 2: *Perplexity values for the second part of the assignment*

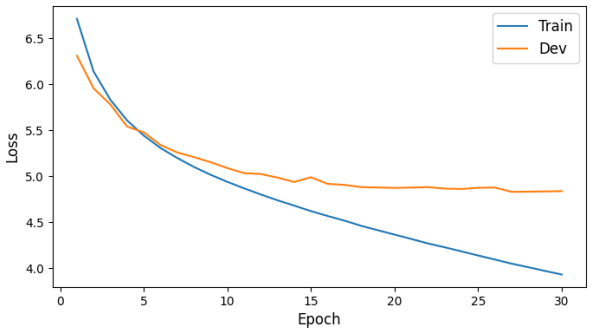

Figure 2: *Trainig loss for the best model of the second part*

4. References

[1] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.