NLU course projects - Lab 4 (LM)

Matteo Mascherin (247183)

University of Trento

matteo.mascherin@studenti.unitn.it

1. Introduction

Modelling language can be done in several ways, using several approaches, which vary from statistical methods to neural one. In this work, a neural approach is used in order to model language and to be able to perform next token prediction. The task consists in modelling the **probability distribution over a sequence**, which is able to reason given a specific context. One of the most suitable neural architectures for language model task is the **Recurrent Neural Network** (RNN), which is able to capture information over a input **sequence**.

2. Implementation details

The neural architecture consists in three main layer, the first one performing the embedding of a certain token, the second one computing the dependencies between tokens through a RNN and the third one is simply a linear layer to map the tokens back into the dictionary.

2.1. Part 1

For the first part of the assignment three main modification were applied to the model:

- the RNN network was substituted by an **LSTM** due to its advantages in term of performance and stability
- **dropout** layers were added to improve training stability and to allow better generalization capability
- SGD optimizer replaced by **ADAMW** optimizer for better training stability

All the modifications are implemented using standard torch library. The dropout layers are positioned after the embedding layer and before the output layer. The dropout probability is set according to the numbers suggested by [1]. For the experiment reported 0.3 was used in both layers. Since the SGD has been replaced by ADAMW the learning has been decreased from 1.5 used with SGD to 0.0015.

2.2. Part 2

In the second part the paper of Stephen Merity et al [1] was a reference to further improve the performance of the model. Few advanced techniques are introduced:

- **weight tying** to improve the stability and save parameters
- **variational dropout** to perform the dropout at sequence level to treat each sentence in the batch in the same way
- non-monotonically triggered **average SGD** optimizer to provide a more stable convergence in the critical phase of the training.

Weight tying was applied into the embedding and the output layer since they work with the same dimensions once we set hidden size equal to output size. Variational dropout is implemented extending the torch.nn.Module class. Non-monotonically triggered ASGD was included using the standard implementation provided by Torch.

Model	Params	Validation	Test
Baseline(RNN)	6M	260	230
LSTM	6.7M	146	136
+ dropout	6.7M	138	129
+ adamW optimizer	6.7M	130	120
Part 2 LSTM	5.3M	100	95
Part 2 LSTM	8.9M	**96**	**88**

Table 1: *Results in term of perplexity for all the different models trained*

3. Results

The tasks presented in the last sections are evaluated on the **perplexity**: a measure of uncertainty of a probability distribution over a vocabulary. The target perplexity was set to 250 for every point of the project.

3.1. Part 1

Starting from the baseline (RNN network), all the results are listed in table 1. The result referring to the first part shows how the different modifications are guiding the model to the right direction. With this achievements the importance of dropout can be further seen if we compare training and validation losses. As can be seen in the images 1, 2, the model tends to overfit if dropout is not inserted.

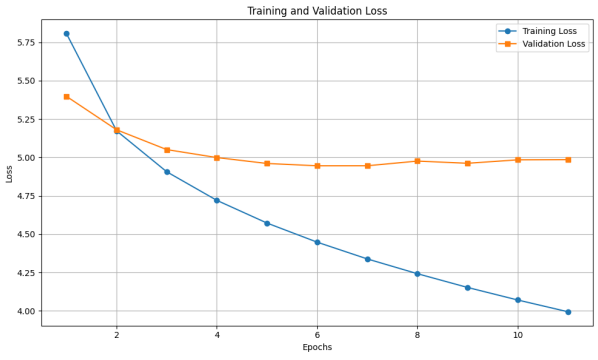

Figure 1: *Training losses without regularization techniques*

3.2. Part 2

With the previous findings, the best model is the one putting together all that has been learnt so far. In order to enhance more

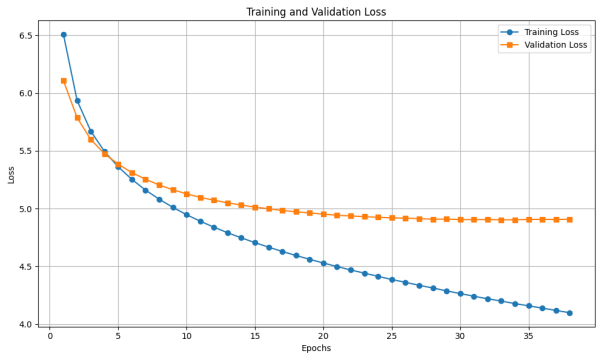

Figure 2: *Training losses with dropout enabled*

the result, hyperparameters were tuned to take the best from the model. Turns out that the major factors are:

1. **Non-monotonically triggered ASGD**: strongly influence the performance once the model is converging, finding the best parameters for the model
2. **dropout**: when increased with an upper bound of 0.5 influence the overall performance
3. **learning rate adaptation**: in order to reduce training time and to maintain great convergence capability is useful to introduce a scheduler for the learning rate decay.

With reference to the point above, the batch size was set to 20 which turns out to be a great mini-batch size for SGD to work with. To see the effectiveness of the ASGD is enough to look at the perplexity schema 3 which shows how the training is able to improve in its final steps. The variational dropout was increased to 0.5 for the output layer for further improve performances and generalization capabilities. The last and main feature introduced is the scheduler that is able to control the learning rate as the training goes on and tune it accordingly. This was crucial to limit the training time without losing performance when converging. The same technique is applied also when the optimizer switches to ASGD with a reduced learning rate and a higher decreasing rate. The massive improvement comes from the adoptation of the non-monotonical triggered AvSGD which store the mean of the weights. Following the implementation of [1] the losses are stored and then ASGD is triggered once the loss does not improve after the model is converging. Once the ASGD is active, the mean of the weights is used in evaluation to test and store the best model. Concluding, weight decay was used inside the optimizer with default value to provide training stability.

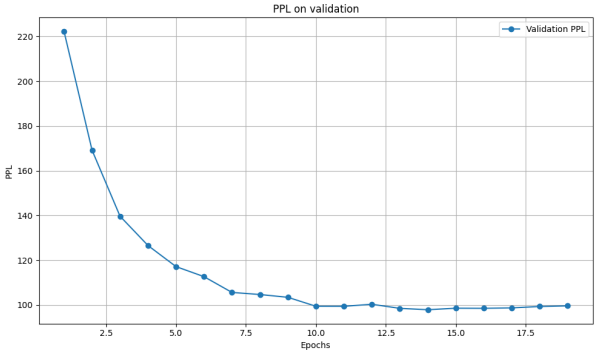

Figure 3: *Perplexity on ASGD training*

4. References

[1] N. S. K. . R. S. Stephen Merity, "Regularizing and optimizing lstm language models," *ICLR 2018*, 2018.