NLU course project - LM RNN optimizations (first assignement)

Stefano Bonetto (247179)

University of Trento

stefano.bonetto@studenti.unitn.it

1. Introduction

The assignment starts by implementing an RNN architecture, progressively enhancing its performance with various techniques.

Various techniques have been used to improve the model. The goal was to maintain $PPL < 250$ and progressively, for each step try to let the PPL decrease.

First of all, I replace the initial RNN with a Long-Short Term Memory (LSTM) network [1]. Then I add two dropout layers [2], one after the embedding layer and one before the output one. I replace the Stochastic Gradient Descent optimizer with the AdamW optimizer.

The second stack of improvements have been made to improve the first results. Starting from the virgin LSTM (point 1), I implement weight tying, Variational Dropout (no DropConnect), and non-monotonically Triggered AvSGD. All of those techniques are described in the Stephen Merity et al. paper[3].

2. Implementation details

I've structured my work into two primary phases: firstly, implementing techniques into the code, followed by an extensive period dedicated to fine-tuning the model's hyperparameters.

The first two steps required simply to apply some features from pyTorch library, in particular adding two dropout layers[2]: one after the embedding layer and one before the last linear layer.

When applying dropout layers, we can choose a hyperparameter p that specifies the probability of each element in the input tensor being zeroed during training. For instance, I set $p = 0.1$, that means each element has a 10% chance of being zeroed out. With this tecnique, input x_0 at time $t = 0$ receive a different dropout mask than input x_1 at time $t = 1$. Using different dropout masks at each timestep in an LSTM causes inconsistent information flow and unstable training, making it hard for the model to learn and remember patterns over time. Consistent masks across timesteps help maintain stability and improve learning.

For this reason Variational Dropout is a useful tecnique to improve performances: it applies the same dropout mask across all timesteps in an LSTM, ensuring consistent information flow and stable training. This helps the model effectively learn and retain temporal patterns. In this case I mantain the default value of $p = 0.5$ for the probability. As we can see in the class $VariationalDropout$, the mask is generated only if there isn't (first iteration) or if the input size is different from the mask one.

As we can see in the Table 1 in particular for the experiments 5 (Dropout) and 6 (VariationalDropout) the improvement is significant (-14.8%).

Then I implemented also weight tying, a technique that involves sharing the same set of weights across different layers within a neural network. Specifically, I applied weight tying by sharing parameters between the output layer and the embedding layer.

Maybe the most interesting part to implement has been the NT-AvSGD part: in my implementation, the model starts with a normal SGD optimizer, but if the development loss has not improved over a defined number of epochs ($NONMONO$) and the loss of the current epoch is higher than the minimum loss from a specific number of previous epochs, the optimizer is switched into AvSGD.

The AvSGD is a variant of SGD: during evaluation, it temporarily replaces model parameters with their averaged counterparts and then restores the original parameters.

I tuned my model's learning rate according to the optimizer I was using:

- SGD: 1.5 on the first part, 5 in the second one;
- AdamW: $1e - 3$.

In the second part of the experiments I raised the learning rate first to 2.5 and then to 10 and 5.

Another tecnique that may be used to ease local minimum problem is to halve the learning rate when after n epoch the PPL hasn't decreased.

3. Results

I'm pleased with the progress I've made in improving the perplexity (PPL) of my model. Through iterative refinement, I've successfully adapted the model by fine-tuning hyperparameters whenever certain techniques didn't yield the desired results. This approach has allowed me to continually enhance the performance of the model, resulting in satisfactory outcomes.

In particular, correctly tuning the learning rate for Experiment 5 has required big efforts. Since using lower learning rates resulted in suboptimal outcomes.

These are the descriptions of the experiments:

1. RNN with SGD ($lr = 1.5$);
2. LSTM with SGD ($lr = 5$);
3. LSTM with SGD ($lr = 5$) + Dropout;
4. LSTM with AdamW ($lr = 1e - 3$) + Dropout;
5. LSTM with SGD ($lr = 5$) + Weight Tying;

6a. LSTM with SGD ($lr = 5$) + Dropout + Weight Tying;

6b. LSTM with SGD ($lr = 5$) + Variational Dropout + Weight Tying;

7. LSTM[1] with SGD ($lr = 5$) + NT-AvSGD + Variational Dropout + Weight Tying;

Results are stored in Table 1 (in the `/bin` folder of second part is not present the model for experiment $6a$, I just put there the result to compare dropout and variational dropout performances).

[1]Just for this experiment I use an embedding size and an hidden size of 600, while in the other experiments I used 300.

n° of experiment	**best_PPL**
1	198.82
2	136.71
3	121.92
4	120.81
5	110.36
6a	109.27
6b	89.09
7	88.36

Table 1: *Results of experiments.*

4. References

[1] pyTorch. (2024) Lstm. pyTorch. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[2] ——. (2024) Dropout. pyTorch. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

[3] N. S. K. . R. S. Stephen Merity, “Regularizing and optimizing lstm language models,” *ICLR*, 2018.