NLU Course Project - Language Modelling

Christian Dalvit (249988)

University of Trento

christian.dalvit@studenti.unitn.it

1. Introduction

This project explores language modeling using Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) units. Some regularization techniques proposed by Merity et al. [1] and optimizers are applied to improve the model performance. The effectiveness of these techniques will be evaluated on the PennTreeBank benchmark measuring the models perplexity. The code of this project is made available on Github.

2. Implementation details

All models of the project are implemented in PyTorch. The RNN implementation from Lab 4 was used as a baseline. The baseline consists of an embedding layer, a RNN and a linear output layer. The baseline optimizer is stochastic gradient descent (SGD).

First Part For the first part, the baseline model was extended by replacing RNN cells with LSTM cells [2]. LSTMs are designed to address the vanishing and exploding gradient problems of vanilla RNNs [3]. Furthermore, two dropout layers were implemented for regularization [4]. Finally, the AdamW [5] optimizer was used instead of SGD. Since PyTorch provides implementations for all the mentioned components, the implementation was straightforward.

Second Part The LSTM baseline from the first part was then extended in the second part by incrementally implementing three regularization techniques.

Weight tying [6, 7] shares the weights between the embedding and output layer [1]. This reduces the model parameters and prevents the model from having to learn a one-to-one mapping between the input and output [1]. When implementing weight tying, we must ensure that the dimensions of the embedding matrix match the output layer dimensions. In PyTorch the `weight` property can be used to implement weight tying.

Variational dropout [8] creates a dropout mask only on the first call and applies the same dropout mask for all repeated connections within a forward and backward pass [1]. In contrast, standard dropout samples a dropout mask every time the dropout function is applied [1]. Since PyTorch does not provide a variational dropout implementation, a custom version is implemented in the `VariationalDropout` class.

Non-monotonically triggered AvSGD (NT-AvSGD) [1] is a variant of averaged SGD (AvSGD). AvSGD performs the same update step as SGD, but instead of returning the last iteration as solution, an average starting from a user-defined time step T is returned. NT-AvSGD eliminates the need to set T manually [1]. NT-AvSGD switches from SGD to AvSGD if the following condition is met

$$t > n \wedge \min_{0 \leq i \leq t'} P_i$$

Where t is the current time step, n a non-monotone interval, P_i the i-th evaluation perplexity and $t' = t - n - 1$. NT-AvSGD is implemented in the `NTAvgSGD` class using PyTorch's optimizer interface. The NT-AvSGD implementation is inspired by the code provided on Github by Merity et al. [1]

3. Results

All models were tested on the PennTreeBank dataset, measuring the models perplexity. All models were trained using the cross-entropy loss and gradient clipping for 100 epochs, with an early stopping patience of 3. The tests were performed on the Marzola cluster of the University of Trento.

First Part In the first part, the RNN implementation was benchmarked against the LSTM implementations with dropout rates of 0.0 (no dropout), 0.1, 0.2 and 0.5 for different learning rates. An embedding size of 300, hidden size of 200 and a training batch size of 64 was used for all benchmarks in the first part. Table 1 shows that for the SGD optimizer LSTMs generally perform better than RNNs. Adding dropout to LSTM cells can improve the performance up to 18 perplexity points. Using the Adam optimizer instead of the SGD further improves the perplexity, especially for higher dropout rates. Figure 1 shows the convergence of selected models over the training epochs. Vanilla RNN and LSTM have rough learning curves, resulting in an early abortion of training. Adding dropout smooths the curve. The Adam optimizer has the best performance with a fast convergence, smooth learning curve and fast training.

Second Part In the second part, the LSTM baseline was benchmarked against the incremental improvements described in Section 2 for different learning rates. An embedding size of 400 and training batch size of 20 was used for training. Furthermore, a second layer and dropout between the hidden layers was added to the LSTM. Dropout rates for the embedding, output and hidden layers were 0.4, 0.1 and 0.25. Table 2 shows that the main improvement is achieved by weight tying. This is interesting because weight tying reduces the number of parameters and improves performance at the same time. Each additional regularization technique improves the performance by approximately 5 perplexity points. Compared to the first part, higher learning rates are more effective with the discussed hyperparameter choice. Figure 2 shows the convergence of selected models over the training epochs. For LSTM models with learning rate 10.0 applying weight tying improves convergence. Adding variational dropout results in slightly slower convergence, but longer training and therefore better overall performance. For the non-monotonically triggered AvSGD model we can observe the triggering of ASGD around epoch 45, which avoids abortion of the training.

Table 1: *Perplexities in the first part*

Model	**Learning rate**			
SGD optimizer	**0.5**	**1.0**	**1.5**	**2.0**
RNN	157.44	159.75	165.16	168.10
LSTM	145.52	145.56	141.49	141.62
LSTM + DR0.1	127.06	123.50	123.95	124.62
LSTM + DR0.2	129.41	**122.24**	122.29	122.55
LSTM + DR0.5	159.14	144.37	140.91	140.56
Adam optimizer	**0.5e-4**	**1e-4**	**0.5e-3**	**1e-3**
LSTM + DR0.1	139.71	120.72	119.85	122.10
LSTM + DR0.2	144.71	122.50	115.95	116.32
LSTM + DR0.5	171.05	139.35	107.84	**105.75**

Note: Underlined values show the best perplexity of a model and the bold value shows the best perplexity over all models for a given optimizer.

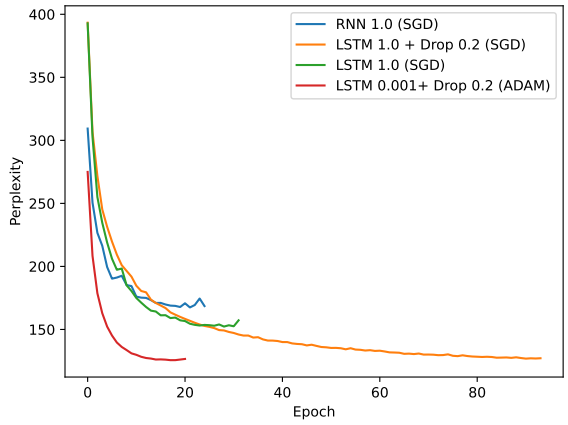

Figure 1: *The convergence over the training epochs of selected models in the first part. The labels denote the model name with the learning rate, plus if a dropout is applied together with the used dropout rate. In parentheses, the optimizer is mentioned.*

Table 2: *Perplexities in the second part*

Model	**Learning rate**				
SGD optimizer	**1.0**	**2.0**	**5.0**	**10.0**	**20.0**
LSTM	106.99	104.17	106.22	111.32	116.35
+ WT	98.57	94.66	91.72	89.41	92.09
+ VarDR	98.64	89.29	84.36	84.84	95.68
+ NT-ASGD	98.72	87.39	81.14	**79.66**	87.62

Note: Underlined values show the best perplexity of a model and the bold value shows the best perplexity over all models for a given optimizer.

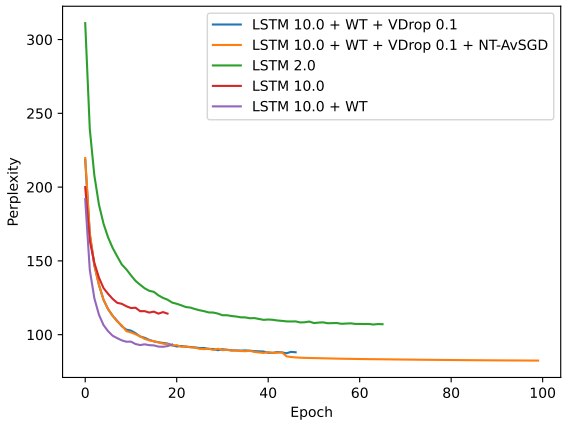

Figure 2: *The convergence over the training epochs of selected models in the second part. The labels denote the model name with the learning rate, plus which regularization techniques are applied.*

4. References

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

[2] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[3] H. Sak, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[6] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *arXiv preprint arXiv:1611.01462*, 2016.

[7] O. Press and L. Wolf, "Using the output embedding to improve language models," *arXiv preprint arXiv:1608.05859*, 2016.

[8] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in neural information processing systems*, vol. 29, 2016.