REPRODUCIBLE MACHINE LEARNING

Group 50: Hongshuo Zhou(260792817), Erik Helal (260738542) Weige Qian(260763075)

I. ABSTRACT

Merity et al.'s "Regularizing and Optimizing LSTM Language Models" investigates methods of reducing overfitting in recurrent neural networks (RNNs) carrying out a language modeling task. The objective of this paper is to assess the reproducibility of Merity et al.'s work and to investigate the robustness of the model in question by performing certain ablations. First, as ablations pertaining to the model's regularization, Dropconnect, word embedding dropout, and variation dropout were all removed one by one. The result of this was more and more overfitting. Then, as ablations pertaining to the model's optimization method, momentum parameters of 0.5, 0.75, and 0.9 were attempted with both standard SGD momentum as well as Nesterov momentum. The results of these experiments were the following losses (standard momentum/Nesterov momentum): 4.23/4.26, 4.28/4.56, and 5.06/5.05, respectively. In other words, it was found that the higher the momentum, the higher the reported loss. Besides that, the use of Adam optimizer with learning rate 0.002 could reach a lower loss with the cost of overfitting problem.

II. INTRODUCTION

Recently, deep learning methods have gained in popularity within the machine learning community, as they are a way to potentially eliminate the need for domain expertise and feature engineering. Recurrent neural networks (RNN's) have been found to be particularly adept at performing tasks such as language modeling (e.g. predicting the next word in a sequence) and sentiment classification (LECTURE NOTES).

Nonetheless, like all deep learning methods, RNN's are prone to suffering from overfitting. Furthermore, for the purposes of language modeling, basic RNN's can have trouble with exploding/vanishing gradients due to excessively long chains of dependencies (i.e. lasting many time steps). A solution to this issue is to modify the internal structure of the RNN to yield a so-called long short-term memory unit (LSTM).

Accordingly, the objective of this report is to review the work of (Merity et al., 2018)[4] Merity et al.'s "Regularizing and Optimizing LSTM Language Models" seeks to explore different ways of reducing the susceptibility of LSTM's to overfitting when tasked with processing the Penn Treebank and WikiText-2's datasets. To accomplish this(Merity et al., 2018), explore elements such as dropout, randomized-length backpropagation through time (BPTT), activation regularization (AR), and temporal activation regularization (TAR). Additionally, they look into the use of averaged stochastic gradient descent (AvSGD).

In our review, we first examine the reproducibility of the (Merity et al., 2018)[4]'s work. Then, we perform certain ablations in order to assess the robustness of their model. First, we perform ablations related to the regularization techniques employed: no dropout, normal dropout,variational dropout, dropout with Dropconnect, and embedding dropout are all investigated. Then, we perform ablations related to the optimization method employed: stochastic gradient descent (SGD) with momentum, SGD with Nesterov momentum, and Adam are all investigated.

III. RELATED WORK

The paper (Merity et al., 2017)[4] that our project is based on has introduced several regularization methods. Our experiments mainly focus on DropConnect (Wan et al., 2013)[2]. DropConnect is a generalization of Dropout; instead of randomly setting a subset of activation functions to 0 in dropout layers, DropConnect randomly sets a subset of weights in the network to zero. Results from this cited paper showed that when regularizing large neural network models, DropConnect often outperforms Dropout.

Dropout (Srivastava et al., 2014)[1] is a regularization method designed to prevent neural networks from overfitting. The key idea is to randomly drop units (i.e. set a subset of activation functions to 0) from the neural network during training. To see if DropConnect could outperform normal Dropout on the datasets we have, we also perform an experiment of using Dropout on the LSTM model. The cited paper indicated that dropout has been highly effective on the SVHN, ImageNet, CIFAR-100, and MNIST datasets. Basically, dropout significantly reduces overfitting and gives major improvements over other regularization methods when training under neural networks. However, as the original paper that our project is based on states, naive application of dropout for regularization strategies is ineffective as it disrupts the RNN's ability to retain long term dependencies.

The paper we based on also introduced the method of variational dropout. This method was introduced in the paper 'A Theoretically Grounded Application of Dropout in Recurrent Neural Networks'(Gal Ghahramani, 2016)[7], instead of applying different dropout masks at different times, the variational dropout only samples a binary dropout mask once, and use it repeatedly. Our ablation experiment also includes this method.

The paper that our project is based on also used quasi-recurrent neural networks (QRNNs) (Bradbury et al., 2016)[3] to explore the viability of the proposed regularization and optimization strategies. To solve the drawback of RNNs on long sequences and parallelism, QRNNs alternate convolutional layers, which apply in parallel across timesteps, and make use of a minimalist recurrent pooling function that applies in parallel across channels. These allow it address the drawbacks of CNNs and RNNs.

For the optimization method of the model, we focused on modification strategies for our task. We found several common optimizing algorithms (Ruder, et al., 2016)[5] for training. Based on the cited paper, we decided to choose SGD with momentum, Nesterov accelerated gradient and ADAM for our experiments.

IV. Dataset and Setup

For this project we have been provided with two datasets from the original paper. They are the preprocessed versions of the Penn Treebank (PTB) and the WikiText-2 (WT2) datasets. Both of them are popular datasets in language modeling. Compared to Penn Treebank, WikiText-2 has a larger data size and contains more vocabulary, which could yield more features.

To tokenize the word from the dataset, the authors of the paper use a Corpus class to store the word and the token to which it corresponds. After the Corpus object has been built, it would be stored in data persistence. Generally this procedure could save running time. Additionally, at the beginning of the model training, word embedding was used to lower the dimension of the input matrix. It could improve the performance of the model.

V. Proposed approach

The published code is based on two papers; one of them is the paper we have based ourselves off of (Merity et al., 2017)[4], and the other one is written by the same author (Merity et al., 2018)[6]. Because of limited computational resources, we only reproduce part of the results. Using the instructions on the ReadMe file, we choose three combinations of arguments. Besides DropConnect, they use all the regularizing methods mentioned in the paper. Two of them were using the LSTM model on the two datasets we mentioned in the Dataset and Setup section. The LSTM has 3 layers with 1150 hidden units. It uses SGD as the default optimizing method with a learning rate of 30. Due to the limitation of computational resources, we change the number of epochs from 500 to 60. In the dataset of WT2, we only use 1/10 of original data with 100 epoch.

The third combination of the arguments uses the QRNN model. It has 4 layers with 1550 units. Since the paper states that the main difference of effect between QRNN and LSTM on these datasets is the training time (QRNNs were 2 4 faster per epoch as compared to their LSTM counterparts), we mainly want to compare the training time by running the Code.

The basic process is data processing, RNN construction, and training with SGD. Every time when the loss reaches the lowest point, the system will save the model. But, as long as the loss is higher than a threshold, AvSGD is used to get a better result. By running the code, we could check how loss changes before and after switching to AvSGD.

For the ablation part, we mainly focus on some regularizing methods. We compare the normal situation (using Penn bank dataset and the condition we mentioned in the first paragraph of this section). We remove the dropConnect , variational dropout, dropout and dropout on input one by one. Furthermore, we also compare the difference in situations of removing the dropout in Word Embedding or not. We will plot graphs to see at which epoch overfitting occurs and how serious overfitting is at the end of the training.

For the modification part, we focus on the optimizing method. Although the optimizing method will change to AvSGD eventually, we still want to see if changing the SGD at first will yield better results than the normal situation. As we mentioned before in the related work section, we chose SGD with momentum, Nesterov momentum and Adam.

During the process of SGD finding the optima, it is possible that the algorithm gets stuck in so-called ‘flat land’, which could make it stuck at that point due to a lack of gradient. Momentum could help SGD accelerate in such a situation and dampen oscillation (Ruder, et al., 2016)[5]. The cited paper also recommended to set value of momentum at 0.9. We try to do the experiment with Momentum equals 2, 1, 0.9, 0.75 and 0.5. Even with the momentum, SGD just follows the slope blindly. Nesterov momentum could slow down the step of SGD when slope up.

As SGD only provides the same learning rates for all parameters, it will be unsuitable when meeting some situations. Adagrad and RMSprop solve this problem. The ADAM algorithm is just like RMSprop with momentum. We want to check if these optimizers could improve the performance of model.

VI. Results

A. Reproduce

For experiment tasks, we first reproduced the original results. Figure 1 illustrates the relationship of ppl at 600 batches and the valid ppl as a function of the number of epochs(this graph only show the result of LSTM in Penn tree bank). The two curves show a similar trend and tend to overlap as epochs increase. This indicates that minimal over-fitting happened. For the reproduction result of using LSTM in WT2 dataset, due to the reduction to the original dataset, we increase the epoch number to 100, this produces the result of 6.03/6.08 on validation/test loss. Compare to the result of Penn Tree Bank in LSTM we reproduce and the result that the author of paper gives us, it is a little bit higher than our expectation.

For the effect of AvSGD, I think the result we reproduce shows us the significant positive effect that it could bring to the training. When the loss trained by other optimizing methods has reached a stable point, Switching to AvSGD could bring us from 0.5 to 1.5 reduction on loss.

The effect of QRNN is also proved in our reproduction result. All of our training is based on GPU NVIDIA Tesla P4. For QRNN, mostly the training time per epoch is around 55s, compared to almost 100s per epoch in LSTM, it has significant progress (almost 2 times in speed)

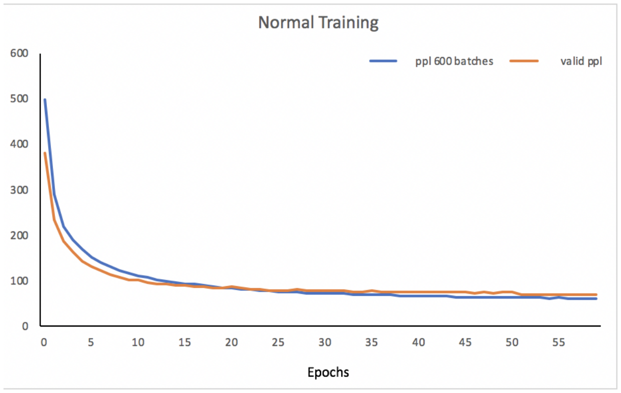

Fig. 1. Normal Training

B. Regularizing Ablation Comparison

For ablation of regularization, we performed several different experiments. Figure 2 illustrates the relationship of ppl at 600 batches and valid ppl as a function of the number of epochs, if dropconnect is removed from original model. Figure 3, Figure 4 and Figure 5 respectively illustrate the same relationship from the absence of word embedding dropout, the absence of variational dropout, no dropout and no dropout on input. It is to be expected that the degree of over-fitting increased from Figure 2 to Figure 5.

For this part, we make an invalid experiment on the normal dropout experiment. Because in Pytorch, the dropout argument on the constructor of LSTM has no use on the last layer (from documentation of Pytorch). As per LSTM we only have one layer, basically, remove or not remove normal dropout did not have any effect on the final result.

In conclusion of the ablation of regularizing methods, the results of our reproduction show us the regularizing methods that used in code do solve the overfitting problem of the training.

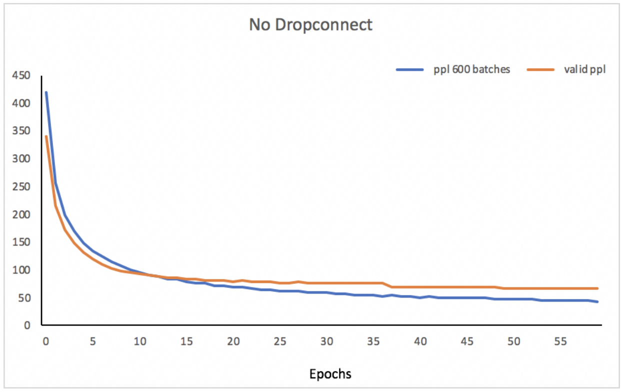

Fig. 2. No Dropconnect

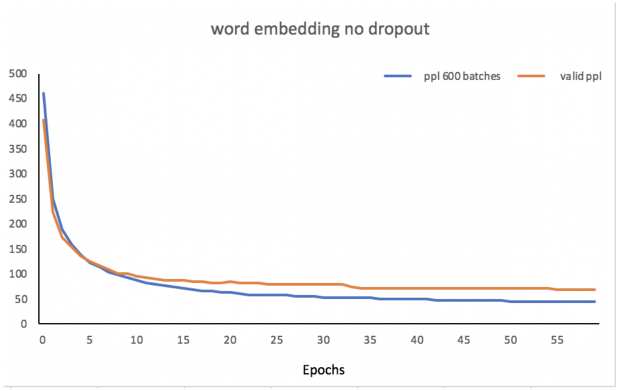

Fig. 3. wordembedding no dropout

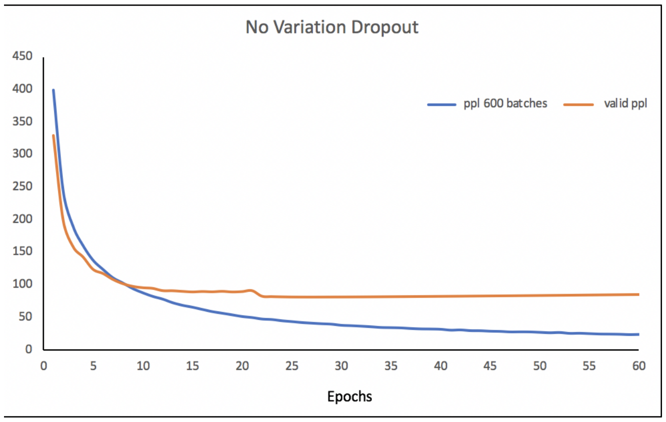

Fig. 4. No Variation Dropout

C. Optimization Modifications Comparison

For modification of optimization, we performed several different experiments. In general, we chose SGD with momentum and SGD with Nesterov momentum for comparison.We first tried to use 2 and 1 as our momentum value,but the loss of results is too large, the number show as NaN, so we decrease the momentum value .Figure 6 illustrates the relationship of ppl at 600 batches and valid ppl as a function of the number of epochs when momentum is set to 0.5. Figure 7 and Figure 8 illustrate the relationship of loss at 600 batches and valid loss as a function of the number of

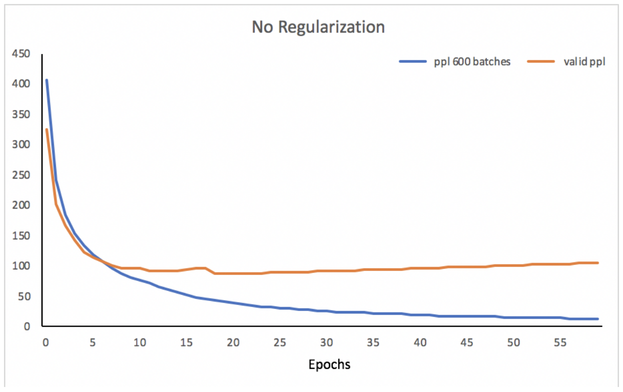

Fig. 5. No dropout on input

epochs when momentum is set to 0.75 and 0.9, respectively. Ppl being the exponential function of loss, we deduce that when momentum was set to 0.5 the valid loss was 4.23, while the valid loss of the model when momentum was set to 0.75 and 0.9 was 4.28 and 5.06, respectively. It is trivial that the valid loss decreased as we tuned momentum down.

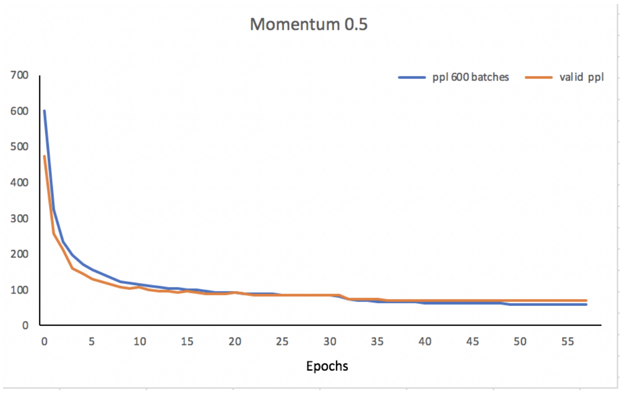

Fig. 6. Momentum 0.5

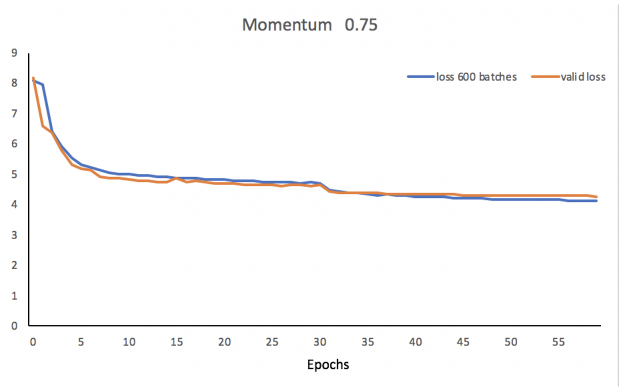

Fig. 7. Momentum 0.75

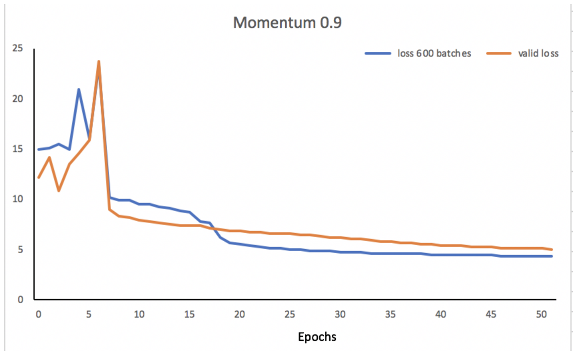

Fig. 8. Momentum 0.9

For the experiments of SGD with Nesterov momentum, the results showed similar trends to using SGD with momentum. Figure 9, Figure 10 and Figure 11 respectively illustrate the relationship of loss at 600 batches and valid loss as a function of the number of epochs when Nesterov momentum is set to 0.5, 0.75 and 0.9. Our experiment results showed that the valid loss of the model was 4.26, 4.56 and 5.05 when Nesterov momentum was set to 0.5, 0.75 and 0.9, respectively. It is to be expected that the valid loss decreased as we tuned Nesterov momentum down.

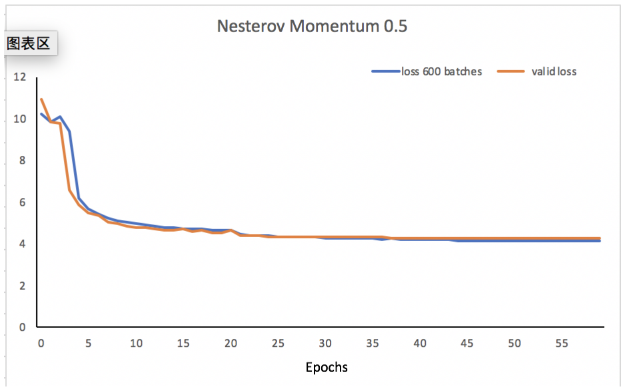

Fig. 9. Nesterov Momentum 0.5

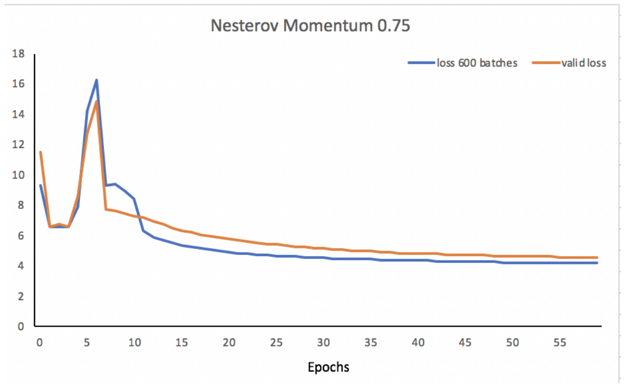

Fig. 10. NesterovMomentum 0.75

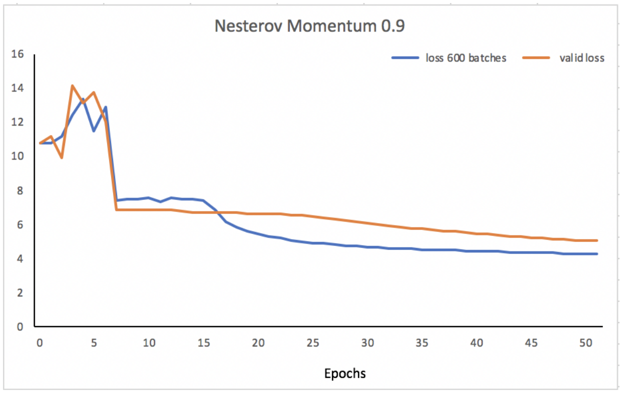

Fig. 11. NesterovMomentum 0.9

For the optimizer Adam, we first choose the same learning rate as the SGD optimizer that the author recommend, but the loss is extremely high. Then we dcrease the learning rate to 20 and 10, but the loss is still very high. After reading the documentation of Adam optimizer in Pytorch website. We decide to use the default learning rate 0.001 and its double 0.002. After running the code,we found the Adam with learning rate 0.002 could reach a little bit lower loss (4.26/4.15 on validation/test) compare to the combination of SGD and AvSGD. But on the other hand, The Adam also cause overfitting problem on our model. At the end of training, the loss of training/validation reach 3.94/4.26 and 3.92/4.18 correspond to learning rate 0.001 and 0.002 respectively.

In conclusion of optimizer part, we found Adam with learning rate 0.002 and SGD with momentum 0.5 could reach a little bit better performance. But the use of Adam will cause the problem of overfitting

VII. Discussion & Conclusions

To conclude, this was a review of Merity et al.'s "Regularizing and Optimizing LSTM Language Models". Merity et al.'s work is an investigation of the different ways of reducing the susceptibility of RNN's to overfitting when given a language modeling task with the Penn treebank and WikiText-2 datasets. In our work, we first demonstrate that, indeed, when Merity et al.'s suggested regularization and optimization strategies are applied, virtually no overfitting occurs. We then performed ablations on the regularization methods used, observing that as we removed Dropconnect, word embedding dropout, and variation dropout, the model suffered from more and more overfitting. Subsequently, we performed ablations on the optimization strategies. First, we considered a model with standard SGD momentum, and considered momentum parameters of 0.5, 0.75 and 0.9. It was found that doing so gave losses of 4.23, 4.28 and 5.06, respectively. Similarly, upon performing identical ablations on a model with Nesterov SGD momentum, we found losses of 4.26, 4.56 and 5.05. These results indicate that although higher momentum may allow for faster convergence (SGD will be less likely to get stuck in local extrema), this may be at the expense of final accuracy. Besides that, our experiment also show us the Adam optimizer could bring a nice final accuracy at the cost of overfitting.

VIII. STATEMENTS OF CONTRIBUTION

Hongshuo Zhou: Report Writing, Proof Reading

Erik Helal:Report Writing, Proof Reading

Weige Qian:Experiments, Report Writing

References

[1] Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15: 1929–1958, 2014.

[2] Wan, M. Zeiler, S. Zhang, Y LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In Proceedings of the 30th international conference on machine learning (ICML- 13), pp. 1058–1066, 2013.

[3] Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-Recurrent Neural Networks. arXiv preprint arXiv:1611.01576, 2016.

[4] Merity S, Keskar N S, Socher R. Regularizing and optimizing LSTM language models[J]. arXiv preprint arXiv:1708.02182, 2017.

[5] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv preprint arXiv:1609.04747, 2016.

[6] Merity S, Keskar N S, Socher R. An analysis of neural language modeling at multiple scales[J]. arXiv preprint arXiv:1803.08240, 2018.

[7] Yarin Gal, Zoubin Ghahramani.A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. arXiv:1512.05287