DD2424 - Assignment 3 (bonus)

Summary

For this extension, I have explored `batch normalization` further, looking at `precise batch normaliztion` and `adaptive batch normalization`. The improvements seem to be marginal. Then, I have also attempted to increase the efficiency and performance when training a deep network, specifically testing variants of a network with 7 hidden layers - one similar to that in the assignment, as well as a deeper variants with more hidden units. I have attempted to train these networks with a cyclical learning rate as well as attempting to gauge the improvements of instead using `Adam` and `AdaGrad`.

The code for the assignment has been written in `python`. I have implemented the neural network as a class. For the hand-in, all of the code has been put toghether in a main file with all the functions and the class declared at the top. For the hand-in, I have also commented out the saving of generated figures and results in JSON files, as well as omitting some of the case-specific testing and gradient testing.

Oskar Stigland

DD2424
Spring 2023

Exploring `batch normalization` further

My initial extension to the third assignment is a further exploration of `batch normalization`, for which I have looked into so-called precise and `adaptive batch normalization`, respectively. For all experiments in this first section, I have used a 4-layer neural network with 100 units in each layer and $\lambda = 0.01$, with a cyclical learning rate, training for 30 epochs with $\eta_{min} = 1e-4$, $\eta_{max} = 1e-2$ and `ns`$= 900$.

Precise `batch normalization`

For `precise batch normalization`, I computed the layer-to-layer normalization parameters at for each 20th batch, such that after e.g. 20 batches, the parameters would be computed for those 20 preceding batches, and so on for 100 batches for each epoch. Then, after each epoch I computed the normalization parameters on the entire training dataset and checked the accuracy on the testing set. The results are provided as a comparison with `adaptive batch normalization` and vanilla `batch normalization` in a figure below. The effect seems to be marginal with respect to loss metrics and the testing accuracy. The loss curve is perhaps slightly smoother.

Adaptive `batch normalization`

For `adaptive batch normalization`, I saved the normalization parameters after each epoch and recomputed them on the testing dataset. For this latter part, I also initially used an image flip probability of $p = 0.5$ at each epoch when comuting the parameters. Again, the effects are very marginal - perhaps even bordering on indiscernible. To compare the results, I have plotted the training loss and testing accuracies for vanilla `batch normalization`, `precise batch normalization`, `adaptive batch normalization` and a combined approach, with an image flip probability of $p = 0.1$. For the exponential moving average, $\alpha = 0.8$ is used.

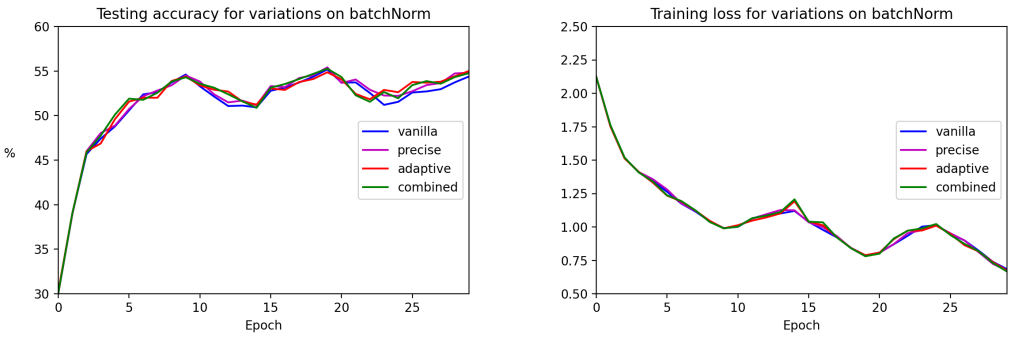

Figure 1: Testing accuracy and training loss for four variations on `batch normalization`, trained for 30 epochs with a cyclical learning rate and a 4-layer network with 100 nodes in each layer and $\lambda = 0.01$.

As is evident from the figure, there is no clear improvement for any of the models. The three modifications (precise, adaptive and combined) seem to achieve a slight bump in accuracy towards the end as compared to the vanilla implementation - but it is indeed slight. All models achieve a maximum accuracy of some $\sim 55\%$.

Improving training for a deep network

To test different optimization paradigms and potentially improve the training process for a deep network, I have ventured into testing `Adam` with a fixed learning rate and `AdaGrad` with a decaying learning rate, which decays according to

$$\eta_n = \eta_0 \times \exp(-\Delta\eta \times t)$$

where $\delta\eta$ is the decay rate, set to $\Delta\eta = 10^{-4}$, and $\eta_0 = 0.01$. For `Adam`, we have a fixed $\eta = 0.001$. Further, for this particular experiment, I have chosen to use `adaptive batch normalization` and for the exponential moving average I use $\alpha = 0.8$. Additionally, I have also employed `He` initialization in order to improve gradient calculations across the training process, and $\lambda = 0.01$. For the model, I have used 7 hidden layers with two variations on depth:

$$\boldsymbol{m}_1 = [50, 50, 50, 20, 20, 10, 10]$$
$$\boldsymbol{m}_2 = [100, 100, 100, 50, 50, 20, 20]$$

The results for all models are presented in the figures below.

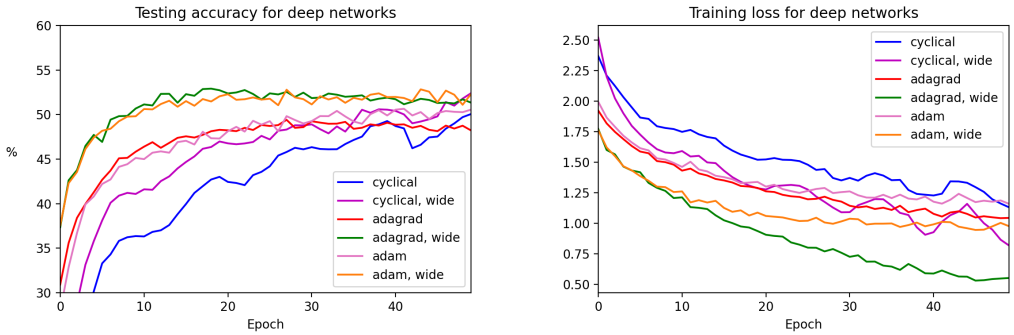

Figure 2: Testing accuracy and training loss for deep networks, trained for 50 epochs with a cyclical learning rate, `AdaGrad` or `Adam`.

In this particular case, the wide network, i.e. with $\boldsymbol{m}_2$ and `AdaGrad` outperforms the others - including the same network with `Adam` with respect to the training loss. However, the accuracy is more or less identical for the two cases. Surprisingly, seems also the cyclical learning rate scheme reaches a similar accuracy with $\boldsymbol{m}_2$ after 50 epochs. However, `AdaGrad` and `Adam` provide more stable training, which is evident from both the accuracy and loss plots, although the final result is no dramatic improvement over the cyclical learning rate. Looking at the figure, it is evident that hidden layers with more units perform better, and that models train with `Adam` and `AdaGrad` converge much faster than when a cyclical learning rate is used.