DD2424 Deep Learning - Bonus Assignment 3

Hugo Brunlid

6 May 2024

1 Introduction

In this bonus assignment, we implement changes to try to improve performance of the k-layer neural network built in the main part of the assignment. Initially, we try to improve the deeper network by using the Adam optimizer with learning rate decay. Next, the number of hidden nodes is increased to evaluate the performance of the network. As another experiment, we change how the values $\boldsymbol{\mu}_{\text{av}}^{(l)}$ and $\mathbf{v}_{\text{av}}^{(l)}$ are computed for the test set, and implement PreciseBN instead of using exponential moving averages.

2 Adam Optimizer

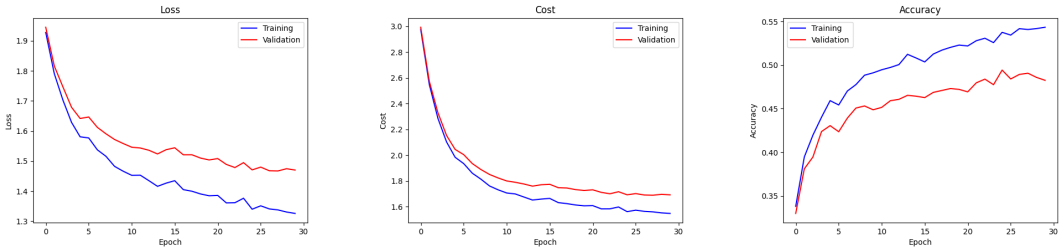
Using a slightly more sophisticated optimizer such as Adam and learning rate decay.

Adaptive Moment Estimation (ADAM) computes adaptive learning rates for each parameter by storing an *exponential decaying average* of past gradients $\mathbf{m}^{(t)}$, and past squared gradients $\mathbf{v}^{(t)}$. These are used to estimate the mean and variance of the sequence of computed gradients in each dimensions. Variance is then used to dampen/augment the update. Suggested default values of $\beta_1 = .9, \beta_2 = .999, \epsilon = 10^{-8}$ are used.

1. Let $\mathbf{g}_t = \nabla_{\mathbf{x}} f\left(\mathbf{x}^{(t)}\right)$
$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1-\beta_1)\,\mathbf{g}_t$$
$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2)\,\mathbf{g}_t . * \mathbf{g}_t$$
2. Set $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$
3. Both $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$ are biased towards zero, especially during the initial time-steps. Counter these biases by setting
$$\hat{\mathbf{m}}^{(t+1)} = \frac{\mathbf{m}^{(t+1)}}{1-\beta_1^t}, \quad \hat{\mathbf{v}}^{(t+1)} = \frac{\mathbf{v}^{(t+1)}}{1-\beta_2^t}$$
4. Adam update rule
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(t+1)}} + \epsilon}\hat{\mathbf{m}}^{(t+1)}$$

This method serves as a replacement for using the cyclical learning rates, instead we use a fixed eta of $\eta = 0.001$. Intuitively, instead of getting triangular peaks such as in for cyclical η_t, we get a more smooth convergence of the cost and loss functions. A 7-layer network is trained using the Adam optimizer with the following parameter settings.

$$\texttt{n\_batch} = 450,\ \eta = 2 \cdot 10^{-3},\ \texttt{decay} = 0.995\ \texttt{cycles} = 3,\ n_s = 5 \cdot 45000/\ \texttt{n\_batch}$$



(a) Loss Function (b) Cost Function (c) Accuracy on test set 49.36%

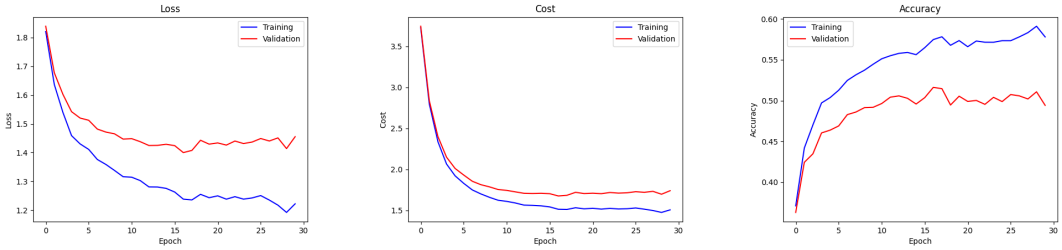
Figure 1: Training a 7-layer network with Adam Optimizer

As compared to the model from the main assignment, performance is slightly decreased. Tweaking the parameters and implementing learning rate decay varies the model performance on the validation set. I found that increasing the batch size improves performance, likely since the gradients computed are more accurately adapted to the entire data set. A lambda search was performed to find the optimal parameter value, this was once again around $\lambda = 0.005$.

3 More Hidden Nodes

Increasing the number of hidden nodes at each layer.

To evaluate whether increasing the number of hidden nodes in the network increases overall performance, we change the model architecture to $100 \to 100 \to 50 \to 50 \to 30 \to 20 \to 10$. Training the model with Adam optimizer yields an accuracy on the test set of 50.70%. This is better performance than cyclical learning rate without batch normalization, but worse than the case with batch normalization.



(a) Loss Function (b) Cost Function (c) Accuracy on test set 50.70%

Figure 2: Training a 7-layer network with more hidden nodes

This deeper network seems to fit well to the training data, but does not generalize too well to the validation data. Even after parameter optimization, the network still doesn't match the performance of the one with cyclical learning rate, which gives 51.86% with the same parameter settings. In general, cyclical learning rates seem to give better performance across the board as compared to using the Adam optimizer in the context of this network.

4 PreciseBN & Adaptive Batch Norm

Change how the $\boldsymbol{\mu}_{\text{av}}^{(l)}$'s and $\mathbf{v}_{\text{av}}^{(l)}$'s are calculated. In this assignment an exponential moving average is used, but it is better to compute the means and standard deviations with a fixed model over a large chunk of training data and it is termed PreciseBN (perhaps calculated only a few times per epoch during training) and then again after training.

"Adaptive Batch Norm" is where $\boldsymbol{\mu}_{\text{av}}^{(l)}$'s and $\mathbf{v}_{\text{av}}^{(l)}$ are estimated from the test data. This can be particularly useful when there is a big shift between the training and test input data. The effect is probably not so big for Cifar-10 but it might interesting to see what happens. I would suggest augmenting the test data with some small augmentations and use these augmented images to compute the population statistics for the test set. Then apply these test population statistics when you run your network on the test set.

Now we are back to the case with the cyclical learning rates. Here, we use the best parameter settings for the 3-layer network with $50 \to 50 \to 10$ hidden nodes, as follows.

$\texttt{n\_batch} = 100$, $\eta_{\min} = 10^{-5}$, $\eta_{\max} = 10^{-1}$, $\lambda = 0.005$, $\texttt{cycles} = 2$, $n_s = 5 \cdot 45000/\ \texttt{n\_batch}$

To implement PreciseBN, we use the method described in the paper *Rethinking "Batch" in BatchNorm* (Wu and Johnson, 2021). Instead of using exponential moving averages for computing $\boldsymbol{\mu}_{\text{av}}^{(l)}$'s and $\mathbf{v}_{\text{av}}^{(l)}$, we use the entire training set to compute population statistics (means and variances) to be used in the batch normalization. This method seems to yield about the same performance as previously, deviating by 0.5 percentage points in accuracy on the test set, depending on parameter settings.

In the next stage, we implement adaptive batch norm where population mean and variance are computed on the test dataset itself. Running the same model but using the testing mean and variance increases final accuracy very slightly. All in all, the CIFAR-10 dataset does not seem to be significantly affected by these changes to batch normalization.

Table 1: Accuracy on the test dataset

	No Precise BN	**Precise BN**
No Adaptive BN	52.75%	52.20%
Adaptive BN	52.81%	52.38%

In order to evaluate whether Adaptive Batch Normalization has a greater effect when there is a larger difference between training and testing data, the testing dataset is augmented by flipping 50% of the images horizontally. Training the model with the same parameters as above, we get an accuracy on the test set of 52.85% with AdaptiveBN and 52.60% without. The improvement is the same independently of the data augmentation performed, signifying that the flipping does not provide sufficient alteration. Another, more likely, explanation is that the images in training and testing datasets are already non-overlapping, therefore not being affected by the changes.