DD2424 Deep Learning - Assignment 3

Hugo Brunlid

3 May 2024

1 Introduction

In the following assignment, we expand upon the network built previously by generalizing to a k-layer network. Gradients are computed using backpropogation and used for training the network on images from the CIFAR-10 dataset. Mathematical details of the network can be summarized as follows. Given an input vector, $\mathbf{x}$, of size $d \times 1$ the network outputs a vector of probabilities, $\mathbf{p}$ $(K \times 1)$, for each possible output label. For layers $l = 1, 2, \ldots, k-1$, we have

$$\begin{aligned} \mathbf{s}^{(l)} &= W_l \mathbf{x}^{(l-1)} + \mathbf{b}_l \\ \mathbf{x}^{(l)} &= \max(0, \mathbf{s}^{(l)}) \end{aligned}$$

and then for the last layer

$$\begin{aligned} \mathbf{s} &= W_k \mathbf{x}^{(k-1)} + \mathbf{b}_k \\ \mathbf{p} &= \text{Softmax}(\mathbf{s}) \end{aligned}$$

The cost function J for the mini-batch is represented as the average cross-entropy loss, adding an L_2-regularization term. Gradients are computed using the mini-batch efficient method.

$$J(\mathcal{B}, \lambda, \mathbf{\Theta}) = \frac{1}{n} \sum_{i=1}^{n} l_{\text{cross}} \ (\mathbf{x}_i, y_i, \mathbf{\Theta}) + \lambda \sum_{i=1}^{k} \|W_i\|^2$$

As an improvement to the network architecture, batch normalization is implemented where the activations are normalized after each layer. This makes deep networks much easier to train, improves gradient flow through the network and reduces the strong dependence on initialization. In order for this to work, both the forward pass and backward pass has to be adapted, providing a new set of equations for computing the gradients and updating scale and shift parameters.

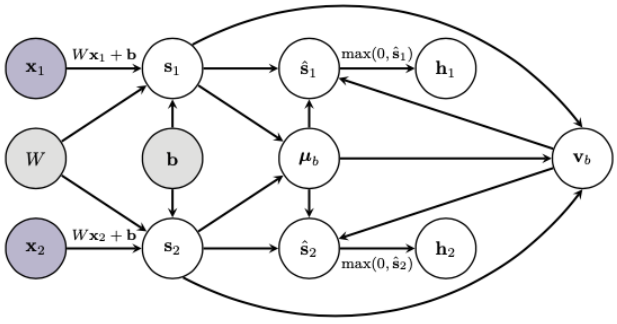


Figure 1: Computational Graph with Batch Normalization

2 Checking Gradient Computations

To assess whether the analytical gradient computations are correct, they are checked against numerical gradients computed using code adapted from `ComputeGradsNumSlow.m`. Here, we test the gradients for a 2, 3 and 4-layer network with batch normalization, and compute the average absolute deviation between the analytical and numerical computations. Due to computational requirements of centered difference method, only the first 100 features & 1000 samples are used.

Table 1: Analytical vs. Numerical Gradients

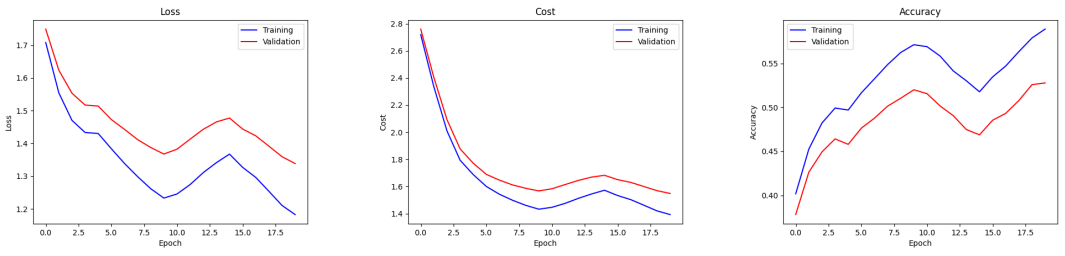
Gradients	**Layer 1**	**Layer 2**	**Layer 3**	**Layer 4**
Weights, $\mathbf{W}$	2.08e-07	1.35e-11		
Bias, $\mathbf{b}$	2.01e-18	1.47e-11		
Gamma, $\boldsymbol{\gamma}$	1.39e-11			
Beta, $\boldsymbol{\beta}$	1.45e-11			
Weights, $\mathbf{W}$	1.39e-11	1.36e-11	1.28e-11	
Bias, $\mathbf{b}$	1.33e-12	4.44e-13	1.35e-11	
Gamma, $\boldsymbol{\gamma}$	1.34e-11	1.77e-11		
Beta, $\boldsymbol{\beta}$	1.19e-11	1.58e-11		
Weights, $\mathbf{W}$	3.18e-06	8.29e-07	3.14e-07	1.35e-11
Bias, $\mathbf{b}$	1.78e-12	2.42e-18	2.86e-18	1.27e-11
Gamma, $\boldsymbol{\gamma}$	3.08e-07	1.47e-11	1.85e-11	
Beta, $\boldsymbol{\beta}$	7.82e-08	1.65e-11	1.21e-06	

As evident by the table, the analytical gradient computations seem to be correct as the average absolute deviation is very small. We also note that the deviations seem to increase as the number of layers in the network increases, which is intuitive as the backpropogation equations for computing the gradients are becoming increasingly complex.

3 Training the 3-layer network

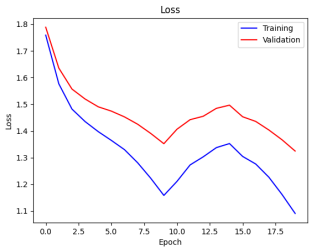
For the first training run, we use a 3 layer network with $50 \rightarrow 50 \rightarrow 10$ hidden nodes. Initially, the network is trained without batch normalization, i.e. the 3-layer extension of the previous assignment. In the next stage, we train the network with batch normalization. We use He initialization with parameter settings as follows.

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

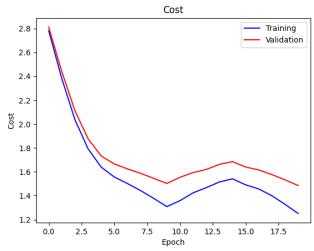


(a) Loss Function (b) Cost Function (c) Accuracy on test set 52.99%

Figure 2: Training a 3-layer network without batch normalization



(a) Loss Function



(b) Cost Function

(c) Accuracy on test set 53.51%

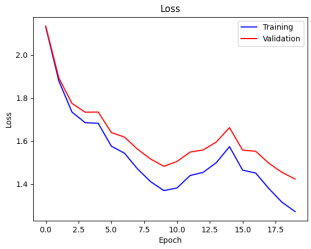
Figure 3: Training a 3-layer network with batch normalization

Looking at the cost and loss functions during the training of the 3-layer networks, we can see very similar patterns in how the cost function converges towards a minima. Both networks yield a similar accuracy on the test set, with batch normalization yielding 0.5 percentage units higher accuracy. Next, we increase the depth of the network to 9 layers.

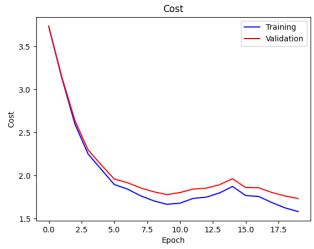
4 Training the 9-layer network

For the second model architecture setup, we train a much deeper network consisting of a total of 9 layers. Hidden nodes in each layer are $50 \to 30 \to 20 \to 20 \to 10 \to 10 \to 10 \to 10 \to 10$ with the final layer representing the 10 possible output classes. Once again, we use the default parameter settings provided in the assignment description.

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



(a) Loss Function

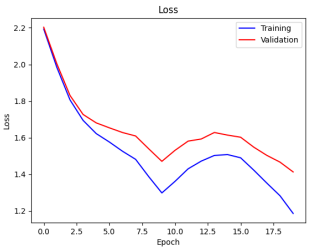


(b) Cost Function

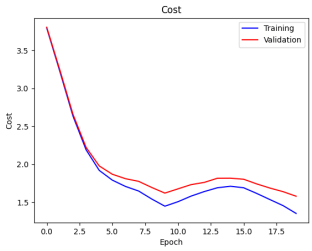
(c) Accuracy on test set 49.47%

Figure 4: Training a 9-layer network without batch normalization

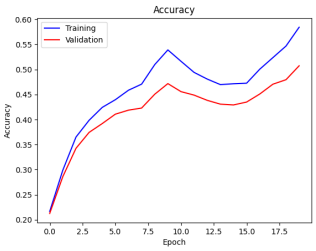
Here, we see that the performance is decreased as compared to the shallower, but wider, neural network of the 3-layer model. Moreover, we note that batch normalization seems to have a larger effect when training deeper neural networks as the final accuracy is increased by 2.2 percentage points. As a network becomes deeper, it becomes harder to train using mini-batch gradient descent and some standard decay of the learning rate. Batch normalization seems to help in training the network, as evident by the cost and loss functions looking much smoother with batch normalization than without.



(a) Loss Function



(b) Cost Function



(c) Accuracy on test set 51.70%

Figure 5: Training a 9-layer network with batch normalization

5 Searching for Lambda

To find the optimal lambda for the 3-layer network with batch normalization a search is performed in two stages: a coarse search and then a fine search. During the lambda search, we use the same network hyperparameters as previously, with the exception of lambda being varied.

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

5.1 Coarse Search

For the coarse search, we randomly sample values of λ in the range $[10^{-5}, 10^{-1}]$ using a log-scale to get an even dispersion. The model is trained for each sampled value of lambda, then accuracy on the test set is computed and saved. Top 3 lambdas and accuracies are presented below.

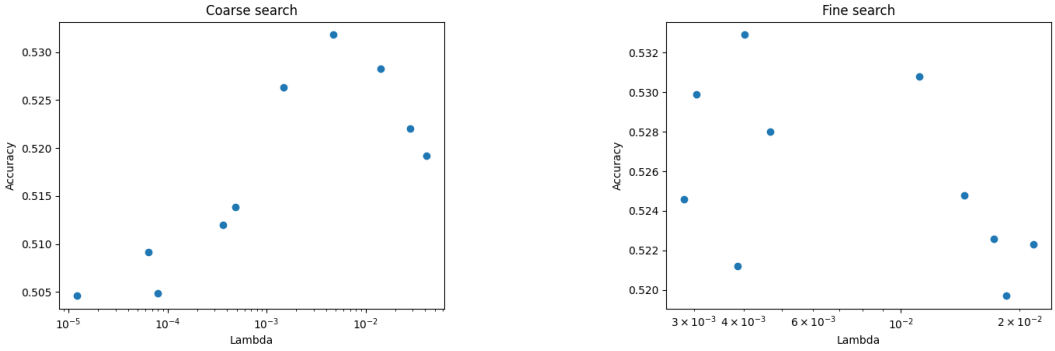
- Top 3 Accuracies: $[0.5318, 0.5283, 0.5263]$
- Top 3 Lambdas: $[0.0047, 0.0141, 0.0014]$

5.2 Fine Search

During the fine search, the two best performing lambda values are used as boundaries to decrease the interval to sample from. Since they are very close to each other, the interval is augmented by ± 0.25 in the log10-scale to capture any beneficial parameter values outside this interval. Values are uniformly sampled from this narrower interval, training the model for 10 different lambdas.

- Top 3 Accuracies: $[0.5329, 0.5308, 0.5299]$
- Top 3 Lambdas: $[0.0040, 0.0111, 0.0030]$

All 10 iterations of the coarse and fine search are presented in the plots below. We see that the best accuracy was given by $\lambda = 0.0040$ at around 53.29%. This is very close to the original value $\lambda = 0.005$ used before which gave an accuracy on the test set of 53.51%. Since we were not able to find a better value for lambda, moving forward we will use the standard setting. Random initialization, selection of validation set, and other hyperparameters all have an effect on the outcome of the search. Varying these may improve model performance further and yield a different optimal lambda setting.



(a) Coarse Search

(b) Fine Search

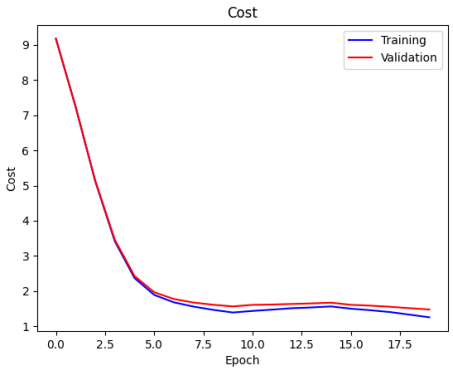
Figure 6: Accuracy during lambda search

6 Sensitivity to Initialization

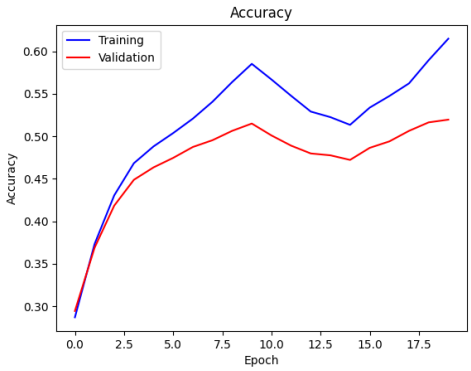
In this section, we explore model sensitivity to initialization. Prior to this point, we have used He initialization where the standard deviation of the normal distribution used to sample values from is proportional to the inverse square root of the number of layer inputs.

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}}$$

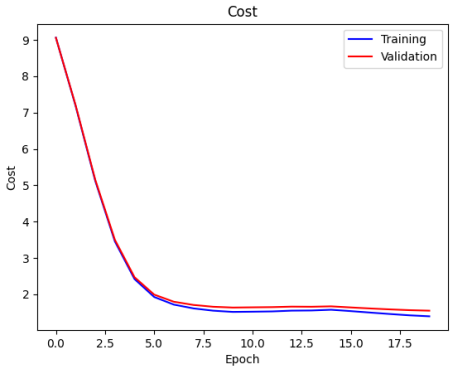
Here, we instead initialize weight and biases of each layer using a fixed standard deviation of $\sigma_{\text{init}} \in \{10^{-1}, 10^{-3}, 10^{-4}\}$ such that $W_{i,lm} \sim N\left(w; 0, \sigma^2\right)$. Running the 3-layer model with batch normalization, standard hyperparameters and the best lambda from the lambda search yields the following results.



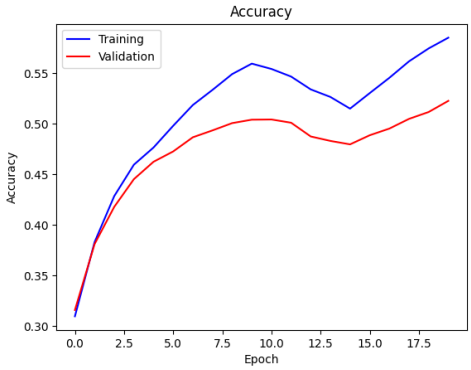
(a) Cost Function with BN



(b) Accuracy with BN, 52.68%



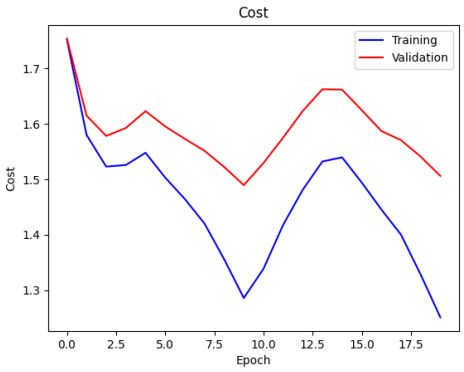
(c) Cost Function without BN



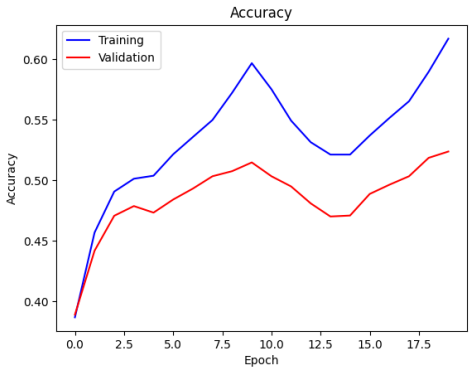
(d) Accuracy without BN, 52.34%

Figure 7: 3-layer network with $\sigma_{\text{init}} = 10^{-1}$

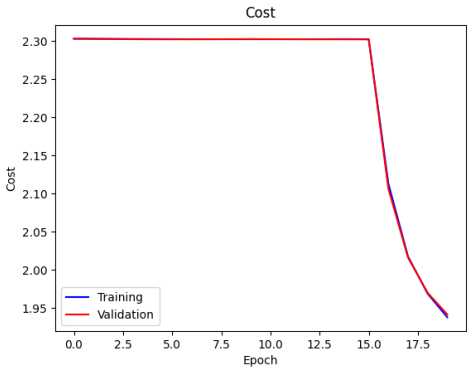
By looking at the Figures 7-9, we draw the conclusion that the model without batch normalization is much more sensitive to initialization than the one with batch normalization. For $\sigma_{\text{init}} = 10^{-1}$ both models have similar performance, but when σ_{init} decreases the non-batch normalized model crashes. When using ReLU as the activation function, a significant amount of activations are zero at each layer and absolute gradients are almost all zero. This results in training not progressing, and the model yielding poor results. Batch normalization seems to remedy this problem, providing increasing performance as σ_{init} decreases. This ultimately results in a test accuracy of 53.51%, which is in line with the He initialization used before.



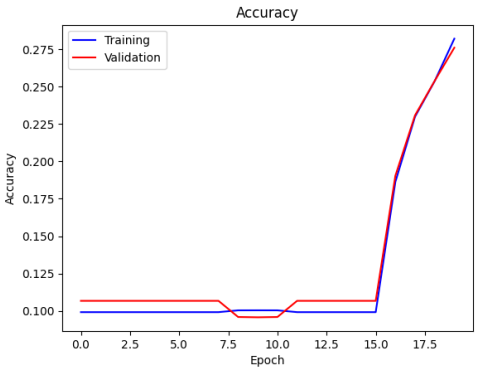
(a) Cost Function with BN



(b) Accuracy with BN, 52.88%

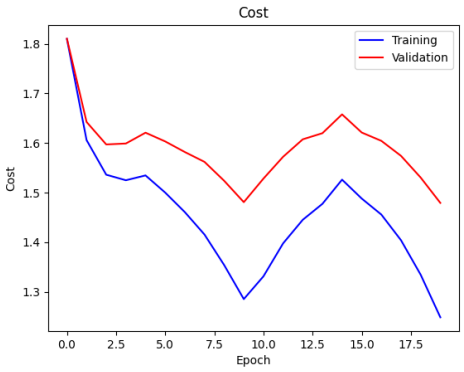


(c) Cost Function without BN

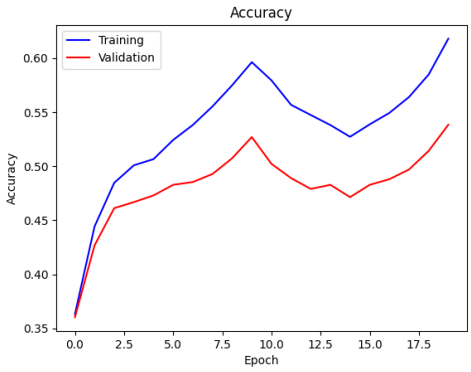


(d) Accuracy without BN, 28.10%

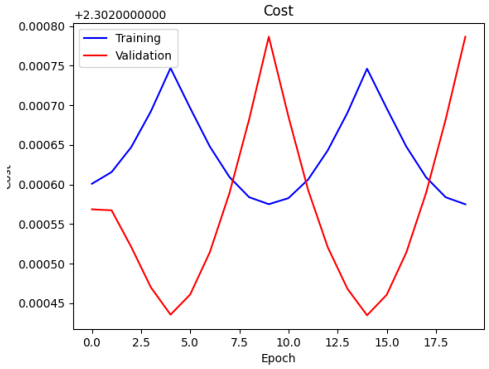
Figure 8: 3-layer network with $\sigma_{\text{init}} = 10^{-3}$



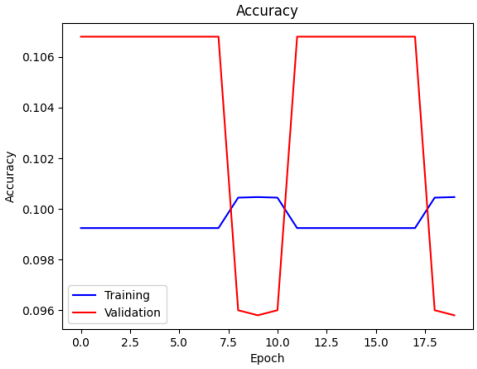
(a) Cost Function with BN



(b) Accuracy with BN, 53.51%



(c) Cost Function without BN



(d) Accuracy without BN, 10.00%

Figure 9: 3-layer network with $\sigma_{\text{init}} = 10^{-4}$