Assignment 3

Albin Larsson Forsberg

April 23, 2020

1 Introduction

The assignment consists of creating a basic one layer network that is classifying images from the CIFAR-10 dataset. The inputs consists of a input layer of size 3072, k hidden layer of different sizes, and an output layer of size 10 with activation function being a softmax. Whichever node has the highest probability assigned to it will be the prediction of the network. Between each layer batch normalization is performed.

2 Dataset

The dataset used is the CIFAR-10 dataset that is coming from the Canadian Institute For Advanced Research. It contains in total 60,000 picture from 10 categories: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. It is widely used in Machine Learning to develop and train different classification models.

3 Model

The Model used is a scalable model with $k+1$ layers, an input, k hidden, and an output layer. The input consists of the rgb data from each pixel and the three dimensional Matrix structure has been vectorized to only be in one dimension. The data is normalized before training. This vector is the fed into the network that predicts the class. The output layer is defined by function 1, while the hidden layers is defined by function 2. Where W are the weights and b the biases. The cost is defined by the cross entropy function.

$$f(X) = \text{softmax}(W^{[k]}a^{[k-1]} + b^{[k]}) \tag{1}$$

$$a^{[n]} = \text{ReLU}(W^{[n]}a^{[n-1]} + b^{[n]}) \tag{2}$$

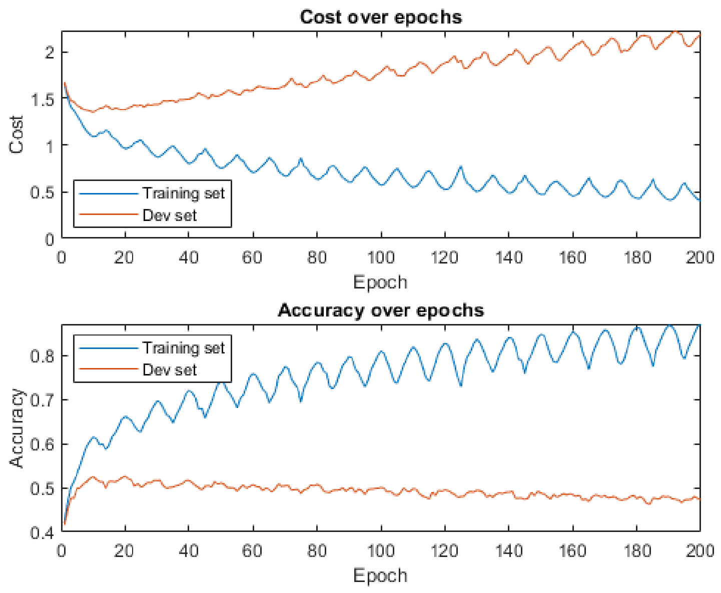


Figure 1: Example of overfit when training for too long

4 Results

The gradient was calculated analytically by hand and was implemented in code. An indication that it is correctly implemented is seen in figure 1 where it is seen that the network overfits heavily when trained for a large amount of epochs. The check that was performed to verify that the gradients are correct are done by checking the gradients of a one hidden layer network and a two hidden layer network numerically. The gradients are correct up unto a small number. However, for the γ and β variables in the lowest layer, especially for the two layer version the error is getting bigger. It is however still in the range of 10^{-3} which when described as a relative error is deemed as small enough. This error seems to stay fairly consistent as well, as when the network is expanded to k layers the error stays constant when checking the first layer. The error in the first layer is the most interesting as the error should be largest there as it has to propagate through a large amount of numerically calculated gradients before reaching there.

To investigate how much batch normalization effects the outcome of the training process two training runs were performed with the same parameter initialization with the only difference being the random order that the shuffling before each epoch was done and the lack of batch normalization (BN) in the first case. The network has two hidden layers with 50 nodes each. The result from the run without BN can be seen in figure 2 and the run with BN can be seen in figure 3. The accuracy without BN was 52.74% and with BN on the test

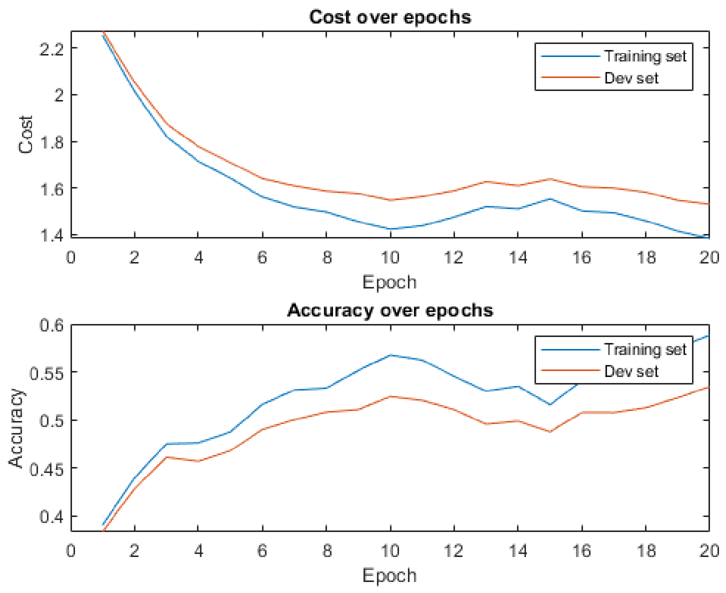


Figure 2: Loss and accuracy for a run without Batch Normalization

data was 53.46%. This difference in result is fairly consistent across different initializations and since it just after a small number of training iterations this result serves as an indication the BN helps to improve the performance of the network.

This result is even greater when looking at the result from a deep network. The figures 4 and 5 both show the same type of results as in figure 2 and 3, but in this case it is a nine layer deep network instead of three. Here it is seen that when there is a deeper network BN can be used to great effect to improve performance and training speed for a network. The accuracy on the test set is 47.32% and 51.91% respectively.

When looking for a good value of lambda a coarse search was first performed with 60 different values of Lambda with values between 10^{-5} and 10^{-1}. The result from this is seen in figure **??**. A fine search between the values of 10^{-3} and 10^{-2} was then performed. The value of $\lambda = 0.0035$ was deemed to provide a good result. The test accuracy after three training cycles was 54.04.

Trying different values if variance in the model with and without BN shows that by using BN the model becomes more insensitive for different initialization values and can manage to learn, despite small variance in the model. That is not the case when BN is not used. A proper initialization is needed. I would guess that the problem with to small variance in the model without BN is almost the same as having a model with all parameters initialized to the same value. It is thus causing the model to be more linear, this is especially the case when the ReLU activation function is used as the numbers will be

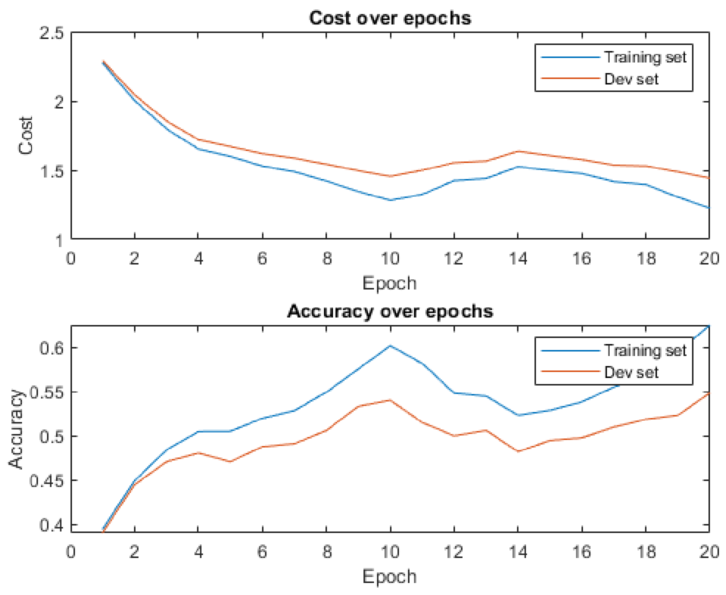


Figure 3: Loss and accuracy for a run with Batch Normalization

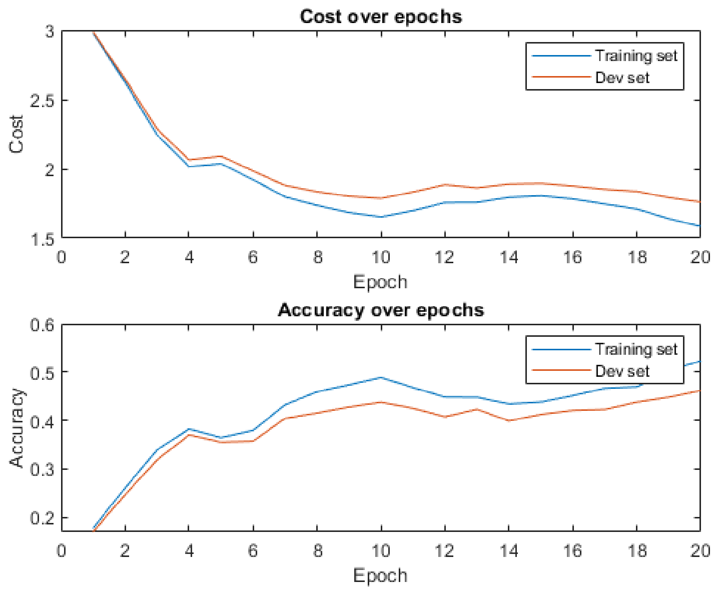


Figure 4: Loss and accuracy for a run without Batch Normalization for a deep network

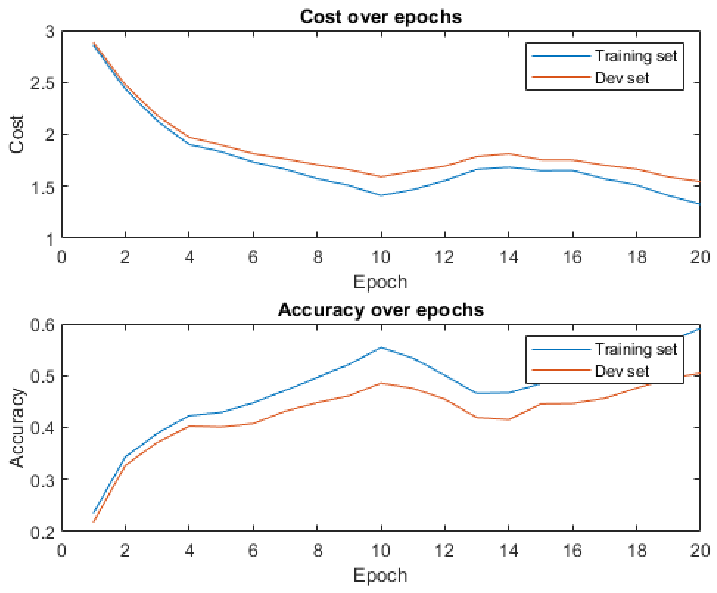
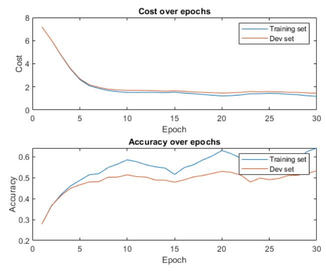
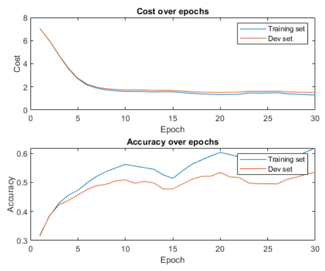


Figure 5: Loss and accuracy for a run with Batch Normalization for a deep network

close to zero anyhow. Batch normalization solves this by making each layer be normalized before activation causing the inputs to each layer to be on a standard distribution. The plots for the different cases are seen in figure **??**, **??**, and **??**.

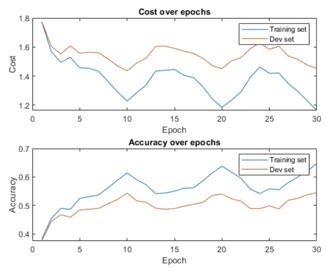


(a) With Batch Normalization

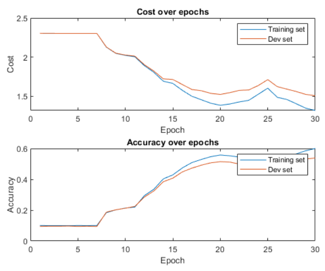


(b) Without Batch Normalization

Figure 6: Loss and accuracy for initialization of $\sigma = 10^{-1}$

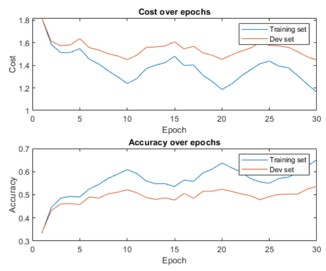


(a) With Batch Normalization

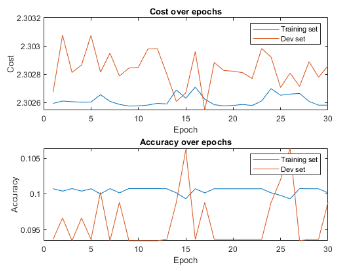


(b) Without Batch Normalization

Figure 7: Loss and accuracy for initialization of $\sigma = 10^{-3}$



(a) With Batch Normalization



(b) Without Batch Normalization

Figure 8: Loss and accuracy for initialization of $\sigma = 10^{-4}$