EE 559- Deep Learning : Mini Project Report -1

Ankita Humne
ankita.humne@epfl.ch

Elliott Pinel
elliott.pinel@epfl.ch

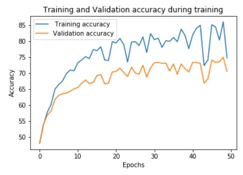
Harshvardhan
harshvardhan.harshvardhan@epfl.ch

May 22, 2020

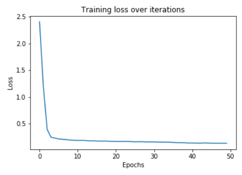
1 Project 1

In project 1, we first extract features using the "base layer", which is followed by a comparator layer and then a classifier to incorporate auxiliary loss. The best chosen solution contains convolution layers with max pool to detect the numbers, followed by a layer to compare the digits. This solution has weight sharing, batch normalization but no auxiliary loss. However, we get similar results using auxiliary loss and no weight sharing as shown in figure 4. But if we include both auxiliary loss and weight sharing, the results are not as good as shown in figures 4i and 4j. Weight sharing reduces the number of parameters, helping to make the training easier. In our case, the variance between images is less hence, weight sharing fastens the process by incorporating useful inductive priors about the problem. In our case, we share weights by using a base model to generate feature vectors for each image. If the actual label of the image can be predicted accurately, we can also predict the label for each pair of images. Thus, using the true label information and the loss associated with it, we should, in principle improve the model performance. To include this loss, we add an additional layer to predict the label of the image from the generated features. As expected, inclusion of weight sharing and auxiliary loss improve the model's performance, however, when both these methods are used together, we do not obtain the best possible results as they counteract the effects of each other.

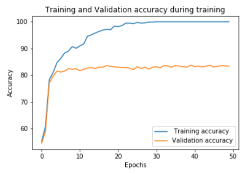
Batch normalization is known to reduce the epochs used to train and hence, was added to the network. This is verified in Figure 2. The activation function used is ReLU. We did not find much difference between results obtained with ReLU and LeakyReLU for the optimal case and decided to go with LeakyReLU because sometimes we encountered results as shown in Figure 3. This is because Leaky ReLU eliminates situation when the gradient becomes zero and we have vanishing gradients. With ReLU, we end up with a neural network that never learns if the neurons are not activated at the start, while Leaky ReLU makes us independent of initialization.



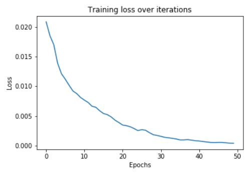
(a) Accuracy - 1 layer



(b) Loss - 1 layer

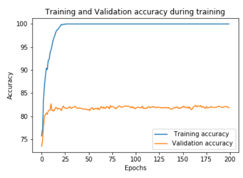


(c) Accuracy - 2 layers

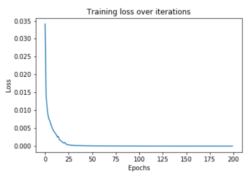


(d) Loss - 2 layers

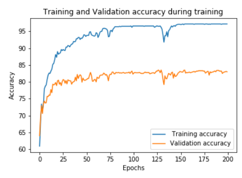
Figure 1: Comparison of layers



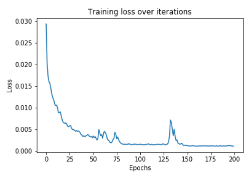
(a) Accuracy - with batch norm



(b) Loss - with batch norm

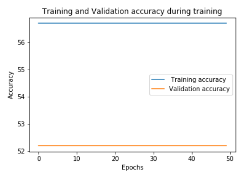


(c) Accuracy - without batch norm

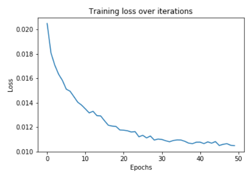


(d) Loss - without batch norm

Figure 2: Effect of batch normalization

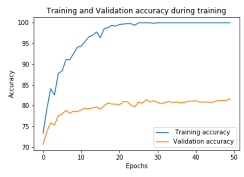


(a) Accuracy

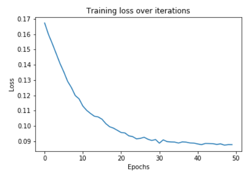


(b) Loss

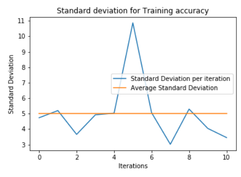
Figure 3: Faulty result with ReLU



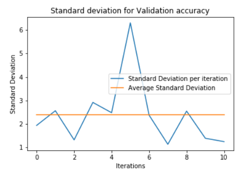
(a) Accuracy - auxiliary loss



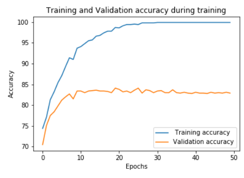
(b) Loss - auxiliary loss



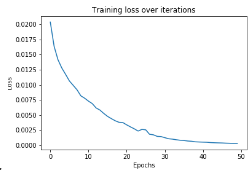
(c) Std dev training - auxiliary loss



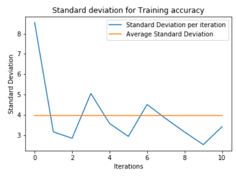
(d) std dev validation - auxiliary loss



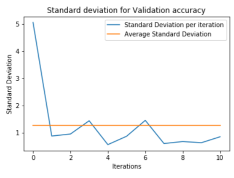
(e) Accuracy - weight sharing



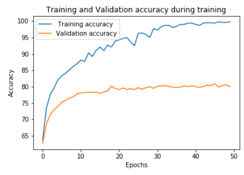
(f) Loss - weight sharing



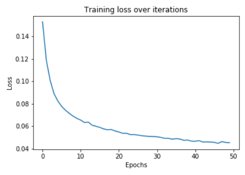
(g) std dev training - weight sharing



(h) std dev validation - weight sharing



(i) Accuracy - with both



(j) Loss - with both

Figure 4: Best obtained results