Assignment3 - DD2424

Silpa Soni Nallacheruvu

June 9, 2025

Gradient Check

To verify the correctness of my analytical gradient computations for my three-layer ConvNet, I implemented a comparison against the provided debug data for the gradients and other available variables. The relative mean and max errors were computed for each layer's weights, biases and more:

Comparison	**Max Difference**	**Mean Difference**
X_conv Vs conv_outputs	0.0	0.0
conv_outputs_mat Vs conv_outputs_flat	1.70×10^{-13}	6.83×10^{-16}
conv_flat	0.0	0.0
h	0.0	0.0
p	6.76×10^{-16}	5.01×10^{-17}
Fs_flat	1.64×10^{-14}	8.45×10^{-16}
W1	4.50×10^{-14}	2.03×10^{-16}
W2	4.51×10^{-15}	1.25×10^{-16}
b1	4.22×10^{-16}	4.22×10^{-17}
b2	1.31×10^{-15}	2.34×10^{-16}

Table 1: Numerical comparison of intermediate computations and gradients.

As the relative mean and max difference between provided debug values and my computed values are insignificant, I conclude that my implementation is bug free.

Initial Three layer ConvNet

The initial three-layer ConvNet was implemented with the following architecture:

- Convolutional layer with 32 filters of size 3x3, stride 1, and padding 1.

- ReLU activation function.
- Max pooling layer with a pool size of 2x2 and stride 2.
- Fully connected layer with 128 units.
- ReLU activation function.
- Output layer with softmax activation for classification.

The network was trained on the CIFAR-10 dataset with training size of 49,000 images, validation size of 1,000 images and test size of 10,000 images. The short training runs with cyclic learning rates from Assignment2 was applied with a minimum learning rate of 1e-5 and a maximum learning rate of 1e-1 for three cycles with constant steps of 800, and L2 regularization $\lambda = 0.003$.

From here, the notations f = filter size, nf = number of filters, nh = number of hidden units.

The initial model with Network architecture 2 ($f = 4, nf = 10, nh = 50$) achieved a final validation accuracy of 56.1% and test accuracy of 55.88%. The training time was reported as 13.07 seconds.

Next, we compare the final test accuracy and training time of the following four network architectures with varying f and nf values with short training runs:

- Architecture 1: $f = 2, nf = 3, nh = 50$
- Architecture 2: $f = 4, nf = 10, nh = 50$
- Architecture 3: $f = 8, nf = 40, nh = 50$
- Architecture 4: $f = 16, nf = 160, nh = 50$

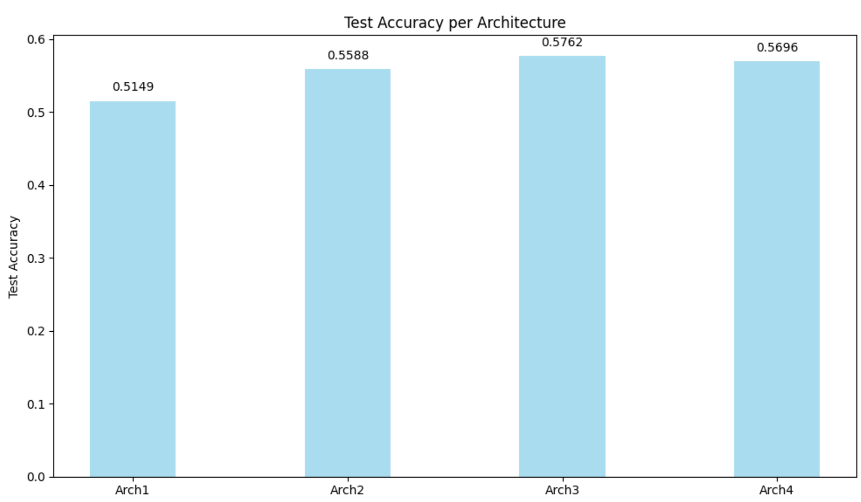

Figure 1: Comparison of final test accuracy for different network architectures.

Figure 2: Comparison of total training time for different network architectures.

The above results show that the test accuracy increases and training time decreases from Arch1 to Arch3 because increasing patch size and filters improves feature extraction and reduces computation (fewer patches per image). Arch4 gets slower and less accurate because patches become too large, losing spatial info, and the network's dense layers become very large, slowing down training and hurting generalization.

We can conclude that Arch2 and Arch3 are the best performing architectures, with Arch3 being the most efficient in terms of training time and accuracy.

I was able to verify the correctness of the above cyclic learning rate schedule with learning rate curve shown below:

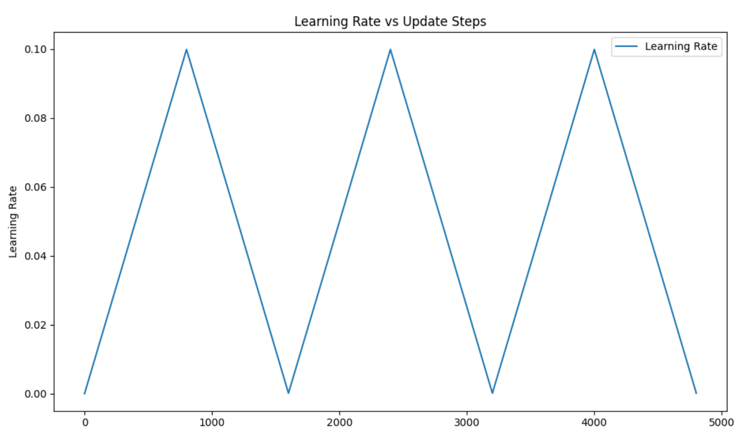

Figure 3: Cyclic learning rate schedule for the training runs.

Longer Training Runs

Here, we implement the cyclical learning rates with increasing step sizes for longer training runs. The step size is doubled after each cycle, starting from 800 steps. The minimum learning rate is set to 1e-5 and the maximum learning rate is set to 1e-1. The training runs are performed for 3 cycles with the previously mentioned best two architectures, i.e., Network Architecture 2 and 3. The curves for training Vs validation loss and accuracies is shown below, logged at every $n_s/2$ steps, where n_s is the number of steps in the current cycle.

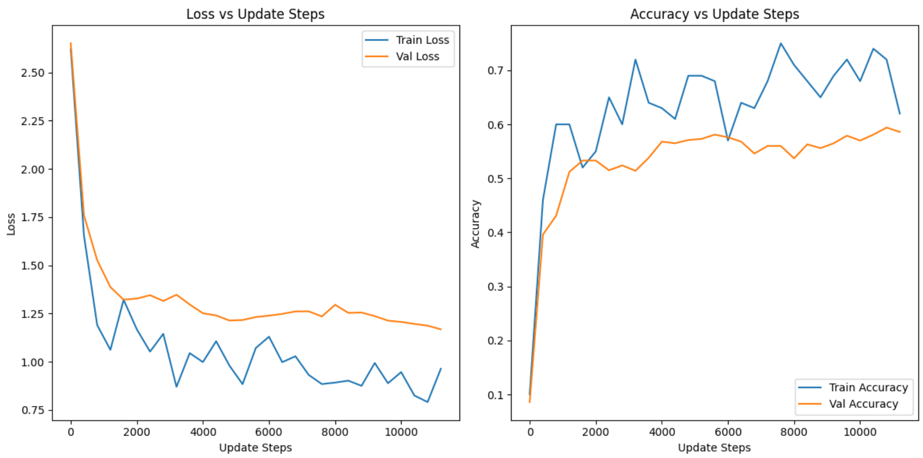

Figure 4: Training and validation loss and accuracies for longer training runs for the Network Architecture 2 ($f = 4, nf = 10, nh = 50$).

Figure 5: Training and validation loss and accuracies for longer training runs for the Network Architecture 3 ($f = 8, nf = 40, nh = 50$).

The final test accuracies and training time for the above two architectures after the longer training runs are as follows:

```
Network Arch2: Final test accuracy = 57.35% in training time of 31.04 seconds
Network Arch3: Final test accuracy = 60.38% in training time of 30.49 seconds
```

Listing 1: Final test accuracies and training time Arch2 and Arch3 for longer training runs

We were able to achieve the final test accuracy > 60% with one of the network architectures, i.e., Network Architecture 3. To get an indication where layer width is a critical factor, we re-run training for the Network Architecture 2 with $nf = 40$. Below is the training and validation curves for this run:

Figure 6: Training and validation loss and accuracies for longer training runs for the updated Network Architecture 2 ($f = 4, nf = 40, nh = 50$).

The final test accuracies and training time for the above updated architecture after the longer training runs are as follows:

```
Updated Network Arch2: Final test accuracy = 63.51% in training time of 58.46 seconds
```

Listing 2: Final test accuracy and training time for updated Arch2 longer training runs

We can see that increasing the layer width (number of filters) in the convolutional layer significantly decreases the loss and improves the final test accuracy, achieving a final test accuracy of 63.51% compared to 57.35% with the original architecture, also greater than the test accuracy of 60.38% with the Network Architecture 3 ($f = 8, nf = 40, nh = 50$). This is because smaller patches with the same number of filters lead to higher dimension feature representation, more detailed learning and in turn, better accuracy. Larger patch sizes reduce spatial detail, while more filters increase representational power, but very large patches may cause loss of detail and overfitting

I was able to verify the correctness of the above increased cyclic learning rate schedule with learning rate curve shown below:

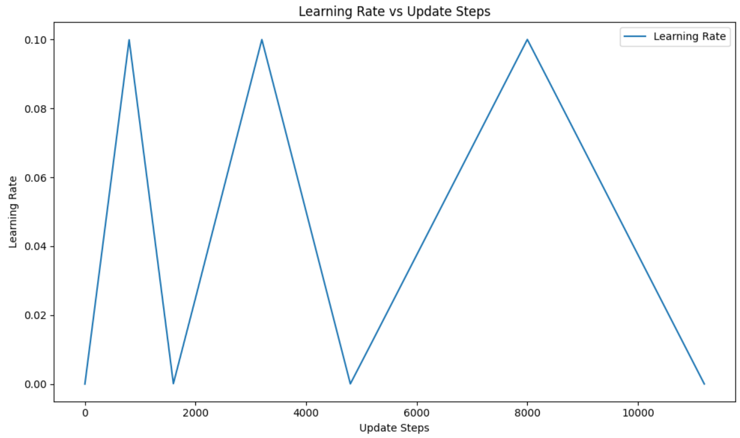

Figure 7: Increased Cyclic learning rate schedule for longer training runs.

Label Smoothing Vs No Label Smoothing

We introduce a new Network Architecture 5 ($f = 4, nf = 40, nh = 300$) with a larger fully connected layer and apply label smoothing to the output layer. The label smoothing is applied by modifying the target labels to be a mixture of the original one-hot encoded labels and a uniform distribution over all classes. The training runs are performed for 4 cycles with the same cyclic learning rate schedule as before, with $\lambda = 0.0025$. The training and validation curves for above run with and without label smoothing are shown below:

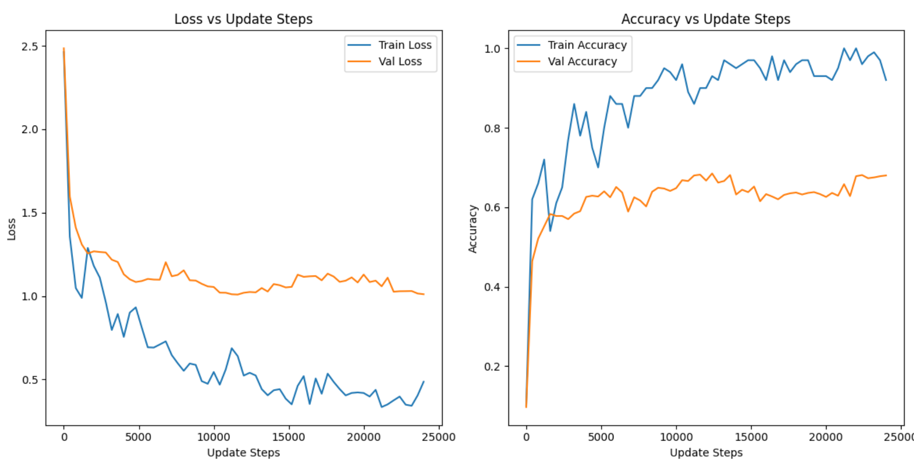

Figure 8: Training and validation loss and accuracies for Network Architecture 5 ($f = 4, nf = 40, nh = 300$) with label smoothing.

Figure 9: Training and validation loss and accuracies for Network Architecture 5 ($f = 4, nf = 40, nh = 300$) without label smoothing.

Without label smoothing, the training loss decreases rapidly and the training accuracy reaches nearly 1.0, while the validation accuracy plateaus and a clear gap emerges between training and validation performance, indicating overfitting.

With label smoothing, the training accuracy does not reach 1.0 and the training loss is higher, but the validation loss decreases more smoothly and the validation accuracy is consistently higher and more stable. The gap between training and validation curves is reduced, showing that label smoothing acts as a regularizer and improves generalization.

In summary, label smoothing prevents the model from becoming overconfident on the training set, resulting in better generalization and higher validation accuracy, by preventing the network from assigning full confidence to any single class.

The final validation, test accuracies and training time for the above two architectures after the longer training runs are as follows:

```
Label smoothing: Final test accuracy = 66.05% in training time of 231.30 seconds
No label smoothing: Final test accuracy = 65.96% in training time of 237.41 seconds
```

Listing 3: Final test accuracies and training time of Arch5 with and without label smoothing

The results in Listing 3 show that label smoothing yields a marginal improvement in test accuracy (66.05% vs. 65.96%) for this architecture, while the training times are nearly identical. This suggests that label smoothing slightly improves the model's generalization without incurring any noticeable additional computational cost.

Although the difference in accuracy is small for this run, label smoothing also tends to reduce overfitting, as observed from the smoother validation curves and reduced gap between training and validation performance in the learning curves. Thus, label smoothing is a simple and effective regularization technique, especially for longer training runs.

I was able to verify the correctness of the above increased cyclic learning rate schedule for 4 cycles with learning rate curve shown below:

Figure 10: Increased Cyclic learning rate schedule for 4 cycles for Arch5.

Improvements and Future Work

The current implementation of the three-layer ConvNet can be improved further by:

- Implementing data augmentation techniques such as random cropping, flipping, and rotation to increase the diversity of the training data.
- Experimenting with different activation functions like Leaky ReLU or ELU to improve convergence.
- Using dropout layers to reduce overfitting, especially in the fully connected layers.
- Tuning hyperparameters such as learning rate, batch size, and regularization strength more systematically using grid search or random search.
- Implementing early stopping based on validation loss to prevent overfitting.
- Exploring more complex architectures like ResNet or DenseNet for potentially better performance.
- Using transfer learning with pre-trained models on larger datasets to leverage learned features.
- Implementing different styles of increased cyclic learning rates, such as triangular or cosine annealing, to improve convergence.

These improvements can help achieve better performance and generalization on the CIFAR-10 dataset.

From the experience of the final project, I believe among these, the best bang for buck wrt final test accuracy would be data augmentation and dropout layers, as they can significantly improve the model's ability to generalize to unseen data without requiring a complete overhaul of the architecture. The hyperparameter search and early stopping can also help in finding the best configuration for the model without requiring extensive computational resources and it helped a lot during my transfer learning final project. As seen in this assignment, increased cyclic learning rates can also help in achieving better convergence and generalization, especially with larger architectures.