Neural Network Digit Recognition Exercise

Data

The data consists of 48 (height) by 28 (width) images of MNIST digits. From random inspection, each image contains 2 digits, with one near the top and one near the bottom. There is overlap of the digits in some images. There are a total of 44 000 images, with 36 000 for training, 4000 for validation, and 4000 for testing.

MLP classifier

Architecture

The MLP model has a single hidden layer with 64 units. The 42 x 28 input is flattened to a 1176 dimension vector xf. This input vector is transformed linearly into a 64 dimension hidden representation with RELU activation to introduce nonlinearity. The hidden representation is then transformed linearly into a 20 dimension output. The first 10 dimensions of the output represents the logit predictions for the first digit. The last 10 dimensions of the output represents the logit predictions for the second digit. Finally, softmax is applied to the first and last 10 dimensions separately to obtain the probability predictions for the two digits.

Results

This architecture achieved (84%, 83%) training accuracy and (83, 82%) test accuracy. Removing the RELU activation on the hidden layer increased the training and test accuracy to (92%, 91%) and (90%, 89%).

My hypothesis is that RELU clipping the negative hidden values shrinks the possible search space of weights. A better minima in the loss landscape could be in the search space that RELU eliminated, thus the weights converge to a less optimal minima and give a worse accuracy.

CNN classifier:

Architecture

Base: The base CNN architecture consists of 3 convolutional layers and a linear output layer.

- Layer 1: outputs a B x 32 x 42 x 28 tensor with (3 x 3) kernel and same padding
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 2: outputs a B x 64 x 42 x 28 tensor
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 3: outputs a B x 64 x 42 x 28 tensor
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Flatten layer
- Linear output: outputs a B x 20 tensor where the first and last 10 dimensions are log logit predictions for the two digits

        - Softmax is applied to convert the logit to probabilities

SCNN (Split CNN): This architecture contains 2 pairs of conv layers and 1 pair of linear layers. The two digits seem to have a relatively fixed location within the image. Convolving the entire image may cause the activation that corresponds to the center of the image contain mixed signals from both digits. The idea is to split the 42 x 28 input image into two 28 x 28 images (one with the top 28 rows and one with the bottom 28 rows), and then process them separately. This may allow the overlapping parts of the other digit to be filtered out as noise when predicting the main digit.

- Layer 1_1: outputs a B x 32 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is the top x[:28, :] (the first 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 1_2: outputs a B x 32 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is x[14:, :] (the last 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 2_1: outputs a B x 64 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is the top x[:28, :] (the first 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 2_2: outputs a B x 64 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is x[14:, :] (the last 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Flatten both sets intermediate tensors
- Linear 1: outputs a B x 10 tensor (logit predictions for the top digit)
    - Softmax is applied to convert the logit to probabilities
- Linear 2: outputs a B x 10 tensor (logit predictions for the bottom digit)
    - Softmax is applied to convert the logit to probabilities

SCNN2: In SCNN, each cropped image gets its own set of conv and linear weights. However, the paired layers are trying to solve the same problem (recognize the single digit in the input), so the idea here is to share weights between corresponding layer pairs so that information from the top and bottom crop is used to train one set of weights. An additional conv and linear layer has been added to introduce more nonlinearity and parameters.

- Layer 1: outputs a B x 32 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is the top and the bottom crop
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 2: outputs a B x 64 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is the top x[:28, :] (the first 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Layer 3: outputs a B x 64 x 28 x 28 tensor with (3 x 3) kernel and same padding
    - Input is x[14:, :] (the last 28 rows and all 28 columns of the image)
    - Relu activation and (3 x 3) max pool with stride 1 and same padding
- Flatten both sets intermediate tensors
- Linear: outputs a B x 10 tensor (logit predictions for the input tensor)
    - Softmax is applied to convert the logit to probabilities

SCNN3: The idea is to reduce the number of parameters with depthwise separable convolution layers. Also trying to shrink the dimensions of the last convolution output tensor with maxpool because the last fully connected layer has a lot of weights.

CNN2: Applying the same architecture as SCNN3 but without splitting the image into top and bottom crops.

Results

CNN: With 10 epochs, 64 per batch, and SGD, the training accuracy was (88%, 80%) and the test accuracy was (87%, 80%). After multiple runs, it appears the top digit is more easily classified than the bottom digit. With 32 per batch, the train accuracy was (98%, 88%) and test accuracy was (97%, 88%).

CNN2: With 10 epochs, 64 per batch, and SGD, the training accuracy was (69%, 63%) and the test accuracy was (68%, 68%). With a decrease in the number of parameters, the expressiveness of the network fell as well.

SCNN: With 10 epochs, 64 per batch, and SGD, the training accuracy was (98%, 89%) and the test accuracy was (97%, 88%). It looks like processing the two digit crops separately does yield better results, however the problem still remains that the second digit has lower accuracy.

SCNN2: With 10 epochs, 64 per batch, and SGD, the training accuracy was (96%, 95%) and the test accuracy was (95%, 94%). It seems like sharing weights for the top and bottom crops managed to boost the accuracy of prediction for the second digit. However, this is still not quite as good as 98%. With 32 per batch, the train accuracy was (98%, 97%) and test accuracy was (97%, 96%).

SCNN3: With 10 epochs, 64 per batch, and SGD, the training accuracy was (74%, 75%) and the test accuracy was (74%, 76%).

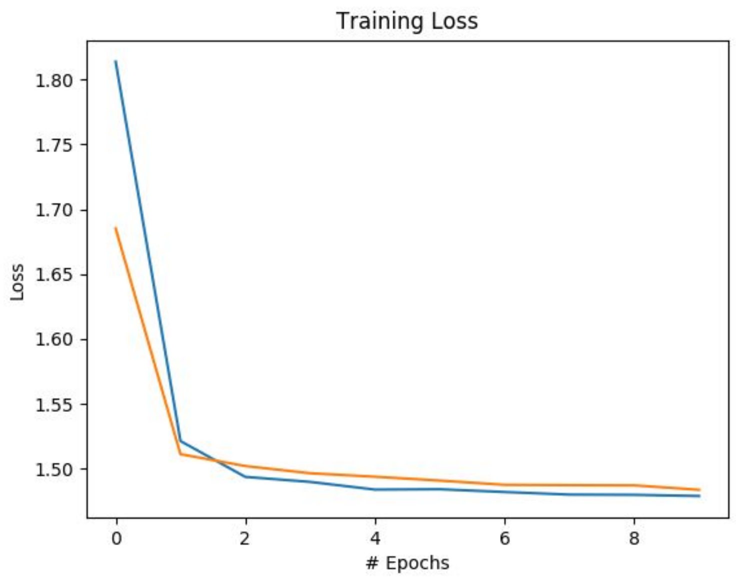
Hyperparameter Tuning

The biggest change to the accuracy came from decreasing the batch size and changing the optimizer from SGD to Adam.

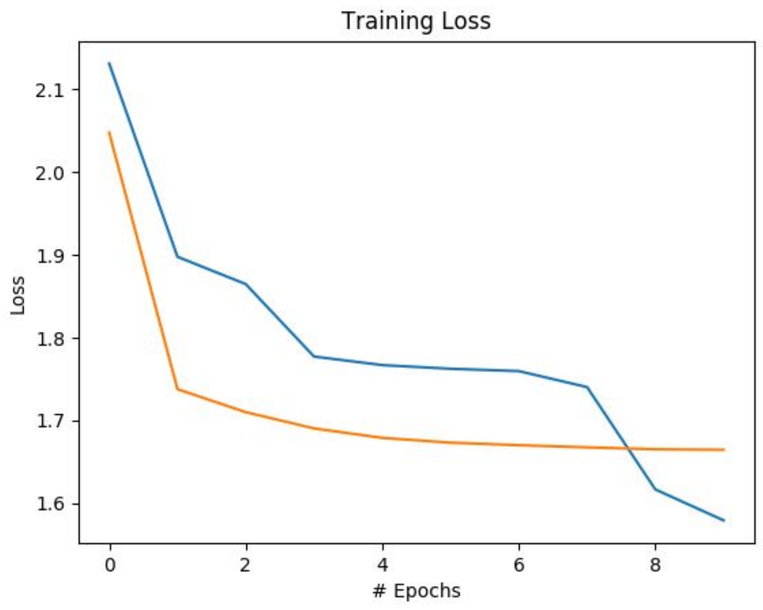
Batch size: With SGD, the bigger the batch size, the less noisy the gradients and the update step will be. However, the benefit of a smaller batch size is that the gradients are updated more often throughout an epoch and the weights can converge faster. I experimented with batch sizes of 32, 64, and 128, and found that training with 32 gave the highest accuracy. However, this is with a fixed number of epochs, which means having a smaller batch size resulted in more training iterations.

Optimizer: Switching to Adam gave the most significant performance boost. With SGD, even if momentum is used, if the loss landscape is ill formed (e.g. high curvature in certain dimensions, saddle points, flat surface), the parameters may get stuck in a local minima or make very slow progress. With Adam, an adaptive learning rate is applied to each dimension, which significantly improved the converge rate. Models that did poorly with SGD actually managed to reach the 98% threshold. For example the SCNN3 architecture reached training and test accuracies of (99%, 99%) and (98%, 97%), and the CNN2 architecture achieved (98%, 97%) training and (96%, 96%) test accuracy. The default learning rate of 0.001 was too big and caused Adam to diverge, but I found that 0.0001 worked well. Below are training loss and accuracy curves comparing the difference between SGD and Adam.

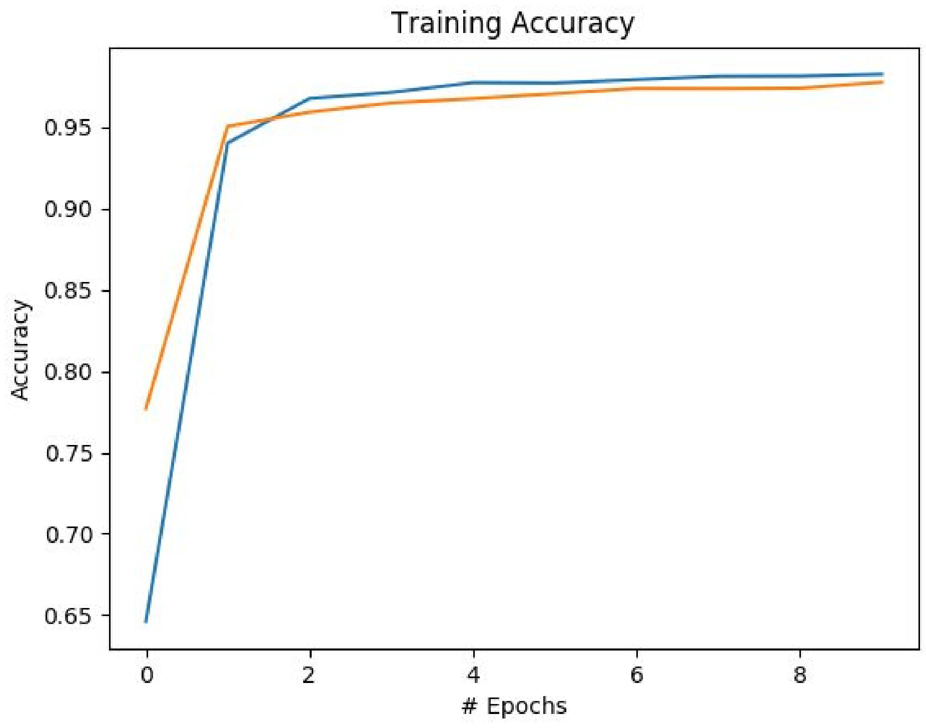
Loss with Adam:



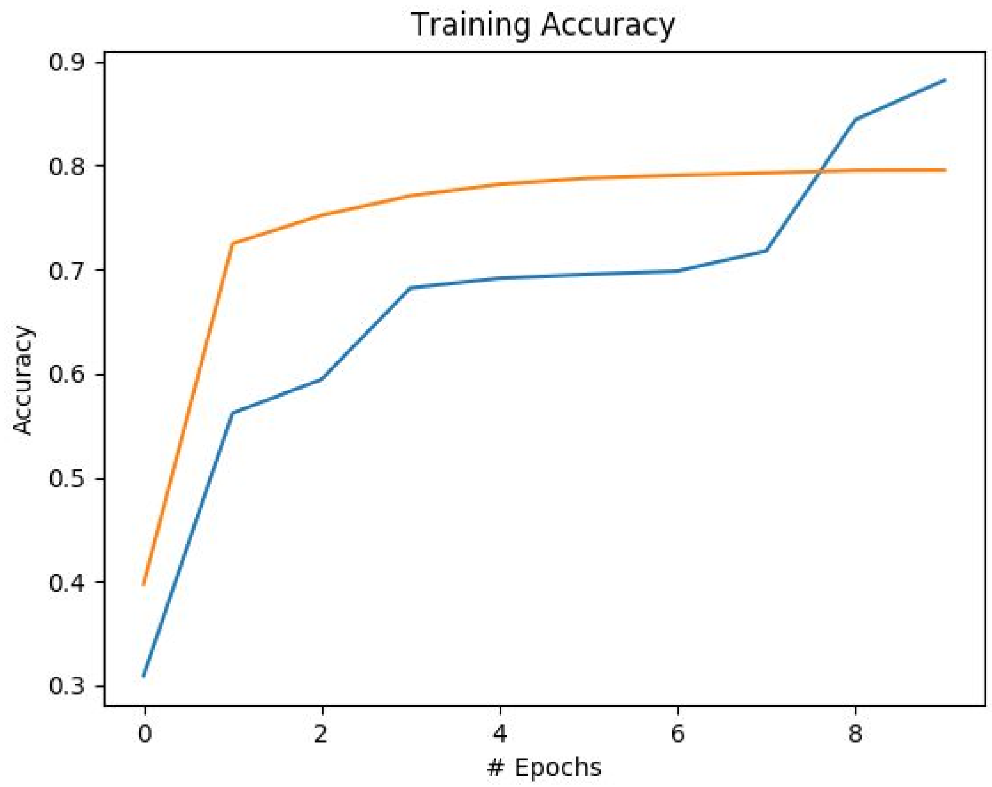
Loss with SGD



Accuracy with Adam



Accuracy with SGD



Best Architecture
The best architecture was SCNN2. This architectured performed well with both SGD and Adam. The best accuracy that it achieved was (99%, 99%) train and (98%, 98%) test with Adam and 15 epochs.

Additional Notes:
My initial attempt was to create a very deep conv net with many parameters that can overfit the training set, and then slowly reduce the architecture complexity and add regularization like dropout. However, I was able to find a method that performed well without significant difference between training and test loss/accuracy, so I did not find it necessary to regularize.

During training, sometimes the training accuracy is quite low (around 15%) and the validation accuracy was very high (around 65%), but the loss was roughly the same. I suspected that the validation data may have an unbalanced number of digits due to the random shuffling. However, examining the test data showed that both top and bottom digit have roughly 400 of each class, so the final test accuracy can be trusted.