CMSC 426 – Computer Vision

Project 3: Classification with Neural Networks

Part 1 of this project involved implementing my own two-layer perceptron network and training the network to use back-propagation to classify. Part 2 involved using Pytorch to create and train a simple convolutional neural network to perform the same task

Alessandro Russo

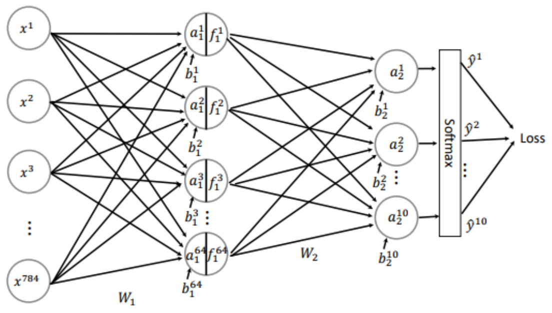
2.1 – Define the Network:

To start I had to complete the forward function to return y_hat. Y_hat, as stated in the project specifications, is equal to softmax(W2(σ(W1x + b1) + b2). Filling in the values in the forward method was simple as I followed the variable assignment that was stated in lecture.

$$a_1 = w_1 \cdot x + b_1$$
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

2.2 – Initialize the Network:

I filled out the initialization method and assigned W1 and W2 to random integers with some standard deviation of 0.1. Following the diagram in the project specification, I set W1 shape to 784 x 64 and W2's shape was set to 64 x 10. B1 and b2 were initialized to zero with b1's shape being 1 x 64 and b2's shape being 1 x 10. A side note is that I had to normalize all the labels and image vectors to ensure no runtime errors. I also stabilized the softmax function.



2.3 – Backpropagation

This was the hardest part of the project that caused a lot of confusion. I was expected to derive the following Jacobian matrices:

$$\frac{\partial a2}{\partial b2}, \frac{\partial a2}{\partial W2}, \frac{\partial a2}{\partial f1}, \frac{\partial f1}{\partial a1}, \frac{\partial a1}{\partial b1}, \frac{\partial a1}{\partial W1}.$$

I used the chain rule to get the partial derivatives with respect to the vector variables W1 and W2 and b1 and b2.

2.3 – Backpropagation

When building my model, in the early developing stages, the accuracies were only around 0.10 which suggested the network was not learning. One issue was that one of the derivatives was missing a term and therefore calculate incorrectly. The other issue was a problem in calculating the accuracy. When the derivative was corrected, the accuracy became substantially higher. Below is vectorized matrix derivatives I used.

$$\frac{\partial A_2}{\partial b_2} = matrix\ of\ ones \qquad \frac{\partial F_1}{\partial A_1} = sigmoid\ derivative\ of\ A_1$$

$$\frac{\partial A_1}{\partial W_1} = x\ (Training\ Images) \qquad \frac{\partial A_1}{\partial b_1} = matrix\ of\ ones$$

$$\frac{\partial A_2}{\partial W_2} = sigmoid\ value\ A_1 \qquad \frac{\partial A_2}{\partial F_1} = W_2$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial W_2} \qquad \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial b_2}$$

$$\frac{\partial L}{\partial F_1} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial F_1} \qquad \frac{\partial A_2}{\partial W_1} = \frac{\partial A_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial A_1} \cdot \frac{\partial A_1}{\partial W_1}$$

$$\frac{\partial L}{\partial A_1} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial A_1} \qquad \frac{\partial A_2}{\partial W_1} = \frac{\partial A_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial A_1} \cdot \frac{\partial A_1}{\partial W_1}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial A_1} \cdot \frac{\partial A_1}{\partial W_1} \qquad \frac{\partial F_1}{\partial b_1} = \frac{\partial F_1}{\partial A_1} \cdot \frac{\partial A_1}{\partial b_1}$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial A_2} \cdot \frac{\partial A_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial b_1}$$

2.4 – Training

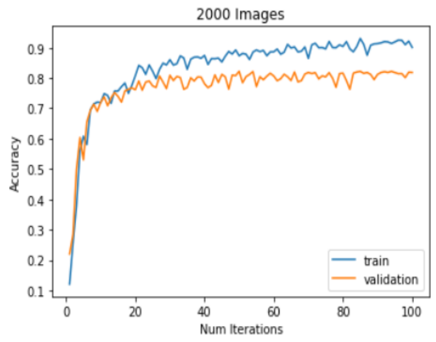
I used, as required, a batch size of 256. The loop adjusted the start point and endpoint over the whole epoch. Considerations were given to calculate the number of iterations to finish a single epoch. The endpoint adjusted correctly for the final batch size. The weights and the biases are adjusted at the end of each batch. Training was performed twice. Once on a training set of 2000 and the other a training set of 50000 images. The accuracies can be seen below.

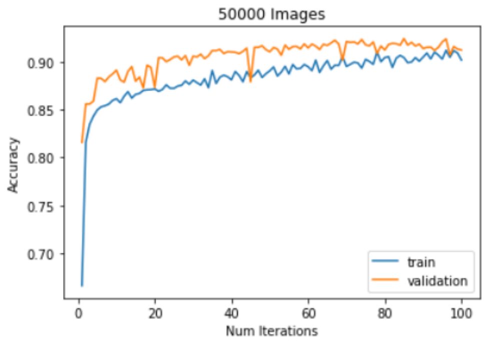
```
2000 training set accuracy list = [0.1205, 0.2565, 0.378, 0.5595, 0.608, 0.5805, 0.6945, 0.7145, 0.7205
, 0.719, 0.7485, 0.7425, 0.7165, 0.757, 0.757, 0.7715, 0.7835, 0.7495, 0.776, 0.8075, 0.8415, 0.834
, 0.812, 0.843, 0.8225, 0.798, 0.8285, 0.849, 0.843, 0.8605, 0.8425, 0.847, 0.873, 0.866, 0.8285, 0
.862, 0.8685, 0.8695, 0.865, 0.8745, 0.8445, 0.864, 0.8635, 0.8655, 0.853, 0.872, 0.8875, 0.879, 0.
8925, 0.8735, 0.881, 0.8785, 0.861, 0.8855, 0.8925, 0.887, 0.891, 0.8735, 0.886, 0.8865, 0.8955, 0.
8785, 0.885, 0.911, 0.899, 0.9025, 0.886, 0.8885, 0.9025, 0.8645, 0.9095, 0.9145, 0.9, 0.9005, 0.89
6, 0.9205, 0.9005, 0.9, 0.9095, 0.904, 0.92, 0.896, 0.8865, 0.904, 0.9305, 0.911, 0.875, 0.908, 0.9
125, 0.9135, 0.916, 0.92, 0.919, 0.914, 0.919, 0.925, 0.9245, 0.91, 0.9215, 0.901]

50000 training set accuracy list = [0.1015, 0.1365, 0.1775, 0.1445, 0.1235, 0.118, 0.147, 0.211, 0.
266, 0.3605, 0.47, 0.4535, 0.4445, 0.5365, 0.6775, 0.7075, 0.688, 0.8005, 0.7835, 0.823, 0.865, 0.8
69, 0.871, 0.893, 0.8965, 0.8945, 0.8915, 0.8895, 0.8925, 0.909, 0.9255, 0.937, 0.9345, 0.9405, 0.9
445, 0.9525, 0.953, 0.9515, 0.951, 0.958, 0.962, 0.966, 0.97, 0.969, 0.97, 0.9745, 0.9765, 0.977, 0
.978, 0.978, 0.98, 0.982, 0.983, 0.985, 0.986, 0.9865, 0.987, 0.988, 0.99, 0.991, 0.989, 0.9905, 0.
9925, 0.994, 0.994, 0.994, 0.9945, 0.994, 0.995, 0.995, 0.996, 0.9965, 0.998, 0.9975, 0.9945, 0.991
, 0.9895, 0.9835, 0.9885, 0.9955, 0.9955, 0.984, 0.984, 0.973, 0.9965, 0.992, 0.995, 0.9965, 0.998,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Notice the accuracies increasing first drastically in the first 20 iterations, then converges towards a value. The 2000 training set converges to around 0.92 and the 50000 training set converges to 1.0.

2.5 – Overfitting

We can see from the training accuracy results that an accuracy of 0.92 was achieved for the 2000 images. However, the validation never reached more than 0.82 for the 2000 images training set. This shows that the 2000 images caused an overfit for the neural network. With the 50000 training set we do not as much overfitting comparing the validation results with the training results, especially at the higher iterations. Comparing the orange line and blue line for the 50000 training set, there is not a big gap between them.





2.6 – How well did it work?

As you can see the networks learned very quickly in the first 20 iterations. This can be seen with the steep slope in the first part of the curve. After iteration 20 the accuracy for the training reached in the 2000 training set images the neural network started to converge to a value around 0.92 for the training set while the accuracy for 50000 training set images reaches 1.00. Overall, both accuracies for training and validation sets are higher than 0.8 which is a good score

2.7 – Where did It make mistakes?

I created a dictionary to track where the neural network made mistakes. The dictionary is keyed by each digit and the value is the number of misses for classifying this digit. We can see that the highest misses are for the 8 followed by the 3. This could be because they look similar. Another is the 5 and the 2 which have a similar amount of mistakes. Both 5 and 2 also look similar if one is inverted.

```
{0: 5, 1: 14, 2: 31, 3: 39, 4: 17, 5: 34, 6: 15, 7: 14, 8: 63, 9: 19}
```

2.8 – Visualize the Weights

2000 training network running for the test data set gave an accuracy of 0.8078.
50000 training network running for the test data set gave an accuracy of 0.9116
Weights from the network were saved, loaded and the test images/labels were ran and pass ed through my accuracy function to get these scores.

3.1 – Describe the Network Architecture

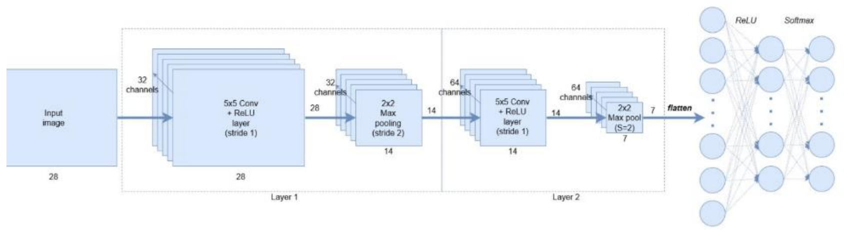


Fig 1

Shapes are the same in layers above

Comparing the diagram with the cnn. The input layer is the images of size 28 by 28. The convolution and the maximum pooling layers are shown in the diagram with the exact number used in the code. The conv1d spec is : in channels = 1, out channels = 6, kernel =5, padding = 0, dilation = 1. The maximum pooling is 2 by 2. The conv2d spec is: in channels = 6, out channels = 16, kernel =5, padding = 0, dilation = 1. Max pooling again with 2 by 2 spec. The last 3 layers are a rectified linear unit (relu) that refers to activation function. The number of nodes for this layer is 120. The layer after in another rectified linear unit (relu) with the number of nodes of 84. The output layer is a linear layer of dimensions 10.

3.2 – Train CNN (UNFINISHED)

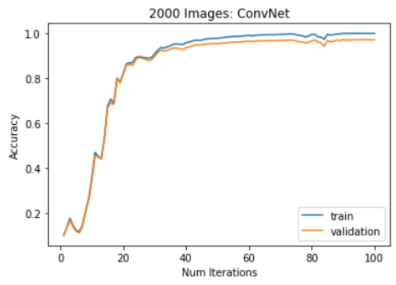
Just like before I used a batch size of 256. The loop adjusted the start point and endpoint over the whole epoch. Considerations were given to calculate the number of iterations to finish a single epoch. The endpoint adjusted correctly for the final batch size. The weights and the biases are adjusted at the end of each batch. The convolutional training was performed twice. Once on a training set of 2000 and the other a training set of 50000 images. The accuracy seemed too low, so I increased the learning rate to 0.01 to improve this. The accuracies on the training set can be seen below.

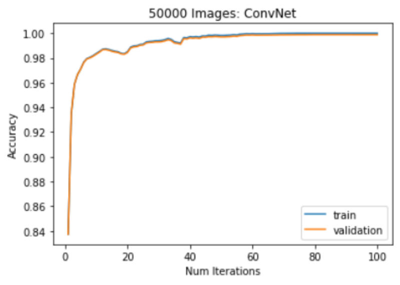
```
20000 = [0.1125, 0.1125, 0.1125, 0.1125, 0.1125, 0.1125, 0.1125, 0.1145, 0.135, 0.202, 0.282, 0
.4345, 0.5495, 0.664, 0.6855, 0.7695, 0.734, 0.7975, 0.83, 0.84, 0.8495, 0.8635, 0.885, 0.883,
0.877, 0.874, 0.893, 0.915, 0.9225, 0.916, 0.915, 0.9115, 0.901, 0.911, 0.9295, 0.931, 0.949, 0
.95, 0.9525, 0.956, 0.9555, 0.954, 0.9595, 0.9625, 0.966, 0.971, 0.975, 0.977, 0.979, 0.9835, 0
.986, 0.9875, 0.9865, 0.9875, 0.9895, 0.991, 0.9915, 0.9895, 0.9925, 0.9865, 0.9825, 0.98, 0.98
45, 0.988, 0.985, 0.9845, 0.9855, 0.993, 0.996, 0.9945, 0.996, 0.999, 0.9975, 0.995, 0.9985, 0.
9985, 0.997, 0.9995, 0.998, 0.998, 1.0, 0.9985, 0.9995, 0.9995, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1
.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

```
50000 = [0.83742, 0.93458, 0.95868, 0.96658, 0.9711, 0.97672, 0.97956, 0.98052, 0.98192, 0.9836
6, 0.98522, 0.98704, 0.98742, 0.98684, 0.98608, 0.98544, 0.98496, 0.98376, 0.9835, 0.98506, 0.9
8876, 0.98972, 0.9899, 0.99088, 0.99114, 0.99298, 0.99334, 0.99352, 0.9939, 0.99394, 0.99424, 0
.99484, 0.9958, 0.99502, 0.99314, 0.99254, 0.99208, 0.9965, 0.99628, 0.9972, 0.99698, 0.99726,
0.9969, 0.99782, 0.99772, 0.9984, 0.99822, 0.9985, 0.99836, 0.99818, 0.99824, 0.99836, 0.99852,
0.99884, 0.99864, 0.9992, 0.99938, 0.99966, 0.99956, 0.99968, 0.99958, 0.99962, 0.99958, 0.9996
2, 0.9996, 0.99966, 0.99974, 0.9998, 0.99988, 0.99988, 0.9999, 0.99992, 0.99994, 0.99998, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

3.3 – Overfitting?

The 2000 training set reaches 0.97 accuracy score while the validation for the 2000 reaches 0.965. The 5000 training set reaches 1.0 accuracy score while the validation for the 50000 reaches 1.0 also. We can conclude not much overfitting happened with the training data for the cnn.





3.4 – How well does it work?

Higher accuracies were reached using cnn compared with the np neural network and there was less of an overfit. In addition, the runtime was much faster, and I checked this by measuring the times with the time module even though it was very obvious. Lastly, I could have improved the accuracy in the np networks by regularization or some form of augmentation.

References:

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Convolution Diagram (fig 1) taken from

https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/