CS489 Machine Learning: Assignment 5

Liam Palmer, 20534162

1.)

a)

|  |  |  |
| --- | --- | --- |
|  | Training Accuracy | Testing Accuracy |
| Softmax Regression  (1000 iterations) | 0.916582 | 0.9198 |
| Softmax Regression  (15000 iterations) | 0.932255 | 0.9243 |
| Convolutional NN  (1000 iterations) | step 0, 0.040000  step 100, 0.880000  step 200, 0.920000  step 300, 0.860000  step 400, 0.980000  step 500, 0.900000  step 600, 0.980000  step 700, 0.980000  step 800, 0.920000  step 900, 1.000000 | 0.963100 |
| Connected NN  (2000 iterations) | 0.8933 | 0.9013 |
| Connected NN  (15000 iterations) | 0.9621 | 0.9580 |

When the number of iterations is in the range 1000-2000, the Convolutional NN performs best with ~96% accuracy, followed by Softmax Regression with ~92% accuracy and Connected NN with ~90% accuracy. The Convolutional NN performs best as expected due to its high expressivity and good results with image recognition. Even thought the Connected NN is more expressive than Softmax Regression, it seems to perform the worst due to the small number of iterations (and thus small amount of training data used).

Once the number of iterations is increased to 15000, the Connected NN outperforms the Softmax Regression as expected (The Connected NN is now provided with sufficient data to get good results). Even at 15000 iterations, neither the Softmax Regression nor the Connected NN can perform as well as the Convolutional NN at only 1000 iterations.

b)

1000 Iterations

|  |  |  |
| --- | --- | --- |
|  | Training Accuracy | Testing Accuracy |
| Rectified Linear Units | step 0, 0.040000  step 100, 0.880000  step 200, 0.920000  step 300, 0.860000  step 400, 0.980000  step 500, 0.900000  step 600, 0.980000  step 700, 0.980000  step 800, 0.920000  step 900, 1.000000 | 0.963100 |
| Sigmoid Units | step 0, 0.040000  step 100, 0.080000  step 200, 0.220000  step 300, 0.160000  step 400, 0.520000  step 500, 0.620000  step 600, 0.860000  step 700, 0.780000  step 800, 0.800000  step 900, 0.840000 | 0.793000 |

Rectified Linear Units perform significantly better than Sigmoid Units (~96% to ~79%). Sigmoid units suffer from a vanishing gradient (as x increases, the gradient goes to zero) whereas the gradient does not vanish for large x in Rectified Linear Units. Thus, using Sigmoid Units results in low convergence (or no convergence) to good results, and thus Rectified Linear Units have significant performance benefits.

c)

1000 Iterations

|  |  |  |
| --- | --- | --- |
| Keep Probability | Training Accuracy | Testing Accuracy |
| 0.25 | step 0, 0.040000  step 100, 0.700000  step 200, 0.920000  step 300, 0.840000  step 400, 0.960000  step 500, 0.900000  step 600, 0.980000  step 700, 0.960000  step 800, 0.860000  step 900, 0.960000 | 0.952700 |
| 0.5 | step 0, 0.040000  step 100, 0.880000  step 200, 0.920000  step 300, 0.860000  step 400, 0.980000  step 500, 0.900000  step 600, 0.980000  step 700, 0.980000  step 800, 0.920000  step 900, 1.000000 | 0.963100 |
| 0.75 | step 0, 0.040000  step 100, 0.780000  step 200, 0.900000  step 300, 0.860000  step 400, 0.960000  step 500, 0.940000  step 600, 1.000000  step 700, 0.960000  step 800, 0.980000  step 900, 1.000000 | 0.972400 |
| 1 | step 0, 0.020000  step 100, 0.820000  step 200, 1.000000  step 300, 0.840000  step 400, 0.960000  step 500, 0.920000  step 600, 0.980000  step 700, 0.980000  step 800, 0.920000  step 900, 0.980000 | 0.973200 |

As seen above, increasing the keep\_prob to 1 increases the testing accuracy. However, we actually expect that having a keep\_prob less than 1 (equivalently a non-zero drop rate) will decrease over-fitting and improve overall accuracy, which is contradictory to what is observed. It is likely that at only 1000 iterations, not enough data is used in training to show a substantial benefit of drop out. Increasing the number of iterations (and thus the amount of training data) will likely show a substantial benefit to dropout.

d)

2000 Iterations

|  |  |  |
| --- | --- | --- |
|  | Training Accuracy | Testing Accuracy |
| 1 hidden layer (150) | 0.8923 | 0.8996 |
| 2 hidden layers (128, 32) | 0.8983 | 0.9010 |
| 3 hidden layers (85, 40, 25) | 0.8905 | 0.8936 |

10000 Iterations

|  |  |  |
| --- | --- | --- |
|  | Training Accuracy | Testing Accuracy |
| 1 hidden layer (150) | 0.9348 | 0.9367 |
| 2 hidden layers (128, 32) | 0.9470 | 0.9454 |
| 3 hidden layers (85, 40, 25) | 0.9540 | 0.9509 |

At only 2000 iterations, the three variations show little difference in accuracy with 3 hidden layers performing the worse by a small margin. This might be due to the fact that a more complex network with 3 layers required more training data to get good results.

Once the number of iterations is increased to 10000, we see a significant improvement (~1%) at each layer increase. Thus, we achieve a higher accuracy with more layers (most likely due to higher expressivity and a higher number of parameters in the network).

Increasing the number of nodes per layer will increase the number of training parameters in the network, thus increasing overall expressivity and likely increasing test accuracy.

2.)

a)

Consider the following 24x24 image of zeros with a 1 in the 12th column and 12th row:

000000000000000000000000

000000000000000000000000

.

000000000001000000000000

.

000000000000000000000000

Let there be one node in the hidden layer with the following 5x5 patch function of 1s:

11111

11111

11111

11111

11111

After applying the 5x5 patching to the original image, we obtain the following 20x20 set of pixels, with the 5x5 block of 1s starting at the 8th row and 8th column:

00000000000000000000

.

00000000000000000000

00000001111100000000

00000001111100000000

00000001111100000000

00000001111100000000

00000001111100000000

00000000000000000000

.

00000000000000000000

Now finally, say we are predicting probabilities between 20x20=400 classes, and the weight parameters for the ith class is the one-hot vector with a 1 at the ith position (the weight parameters to be used in the Softmax function).

Then the probability of class 9x20+1 = 181 (corresponding the 10th row 1st column in the image above) is:

e0/(375e0 + 25e1) = 1/(375 + 25e) ~= 0.002258

Now we consider the following 24x24 image of zeros with a 1 in the 2nd column and 12th row, which is a translation of our original image 10 pixels to the left.

000000000000000000000000

000000000000000000000000

.

010000000000000000000000

.

000000000000000000000000

After applying the same 5x5 patching to the translated image, we obtain the following 20x20 set of pixels, with the 5x2 block of 1s starting at the 8th row and 1st column:

00000000000000000000

.

00000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

00000000000000000000

.

00000000000000000000

Now finally, as before we are predicting probabilities between 20x20=400 classes, and the weight parameters for the ith class is the one-hot vector with a 1 at the ith position (the weight parameters to be used in the Softmax function).

Then the probability of class 9x20+1 = 181 (corresponding the 10th row 1st column in the image above) is:

e1/(390e0 + 10e1) = e/(390 + 10e) ~= 0.006516

Thus, under a translation of 10 pixels the image produces a different probability in the same network. This is a counter example.

b)

Consider the following 24x24 image of zeros with a 1 in the 12th column and 12th row:

000000000000000000000000

000000000000000000000000

.

000000000001000000000000

.

000000000000000000000000

Let there be one node in the hidden layer with the following 5x5 patch function of 1s:

11111

11111

11111

11111

11111

After applying the 5x5 patching to the original image, we obtain the following 20x20 set of pixels, with the 5x5 block of 1s starting at the 8th row and 8th column:

00000000000000000000

.

00000000000000000000

00000001111100000000

00000001111100000000

00000001111100000000

00000001111100000000

00000001111100000000

00000000000000000000

.

00000000000000000000

After applying the max pooling as described, we obtain the following 5x5 image.

00000

01100

01100

00000

00000

Now finally, say we are predicting probabilities between 5x5=25 classes, and the weight parameters for the ith class is the one-hot vector with a 1 at the ith position (the weight parameters to be used in the Softmax function).

Then the probability of class 5x2+1 = 11 (corresponding the 3rd row 1st column in the image above) is:

e0/(21e0 + 4e1) = 1/(21 + 4e) ~= 0.03137

Now we consider the following 24x24 image of zeros with a 1 in the 2nd column and 12th row, which is a translation of our original image 10 pixels to the left.

000000000000000000000000

000000000000000000000000

.

010000000000000000000000

.

000000000000000000000000

After applying the same 5x5 patching to the translated image, we obtain the following 20x20 set of pixels, with the 5x2 block of 1s starting at the 8th row and 1st column:

00000000000000000000

.

00000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

11000000000000000000

00000000000000000000

.

00000000000000000000

After applying the max pooling as described, we obtain the following 5x5 image.

00000

10000

10000

00000

00000

Now finally, as before we are predicting probabilities between 5x5=25 classes, and the weight parameters for the ith class is the one-hot vector with a 1 at the ith position (the weight parameters to be used in the Softmax function).

Then the probability of class 5x2+1 = 11 (corresponding the 3rd row 1st column in the image above) is:

e1/(23e0 + 2e1) = e/(23 + 2e) ~= 0.0956

Thus, under a translation of 10 pixels the image produces a different probability in the same network. This is a counter example.