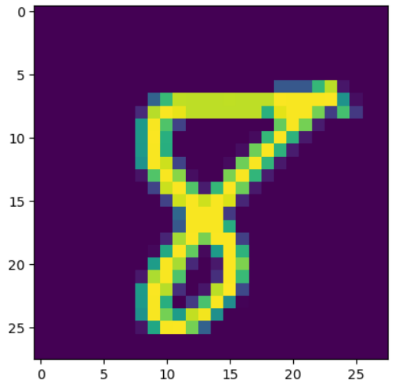
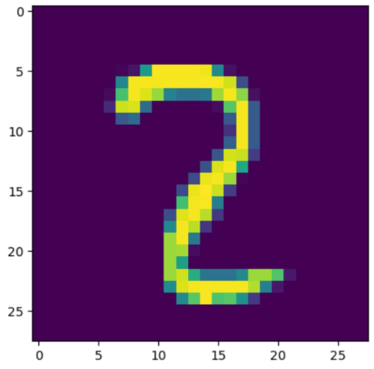
Project 3: Neural Nets

Jonas Ferguson

Classification: MNIST

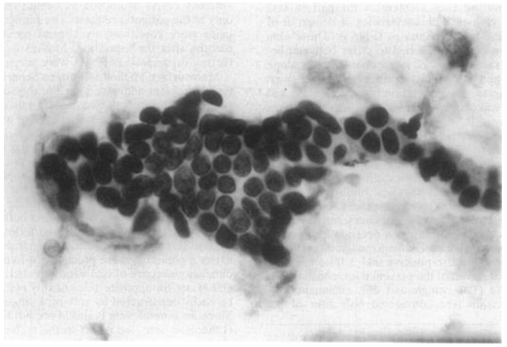
I built a convolutional neural net to classify handwritten digits 0-9. These digits were pulled from the 50000 image training set and tested on the 10000 image test set. I tried doing this while varying dropout rate, batch size, drop layer size, and learning rate.





Regression: WDBC

I built a multi-layer perception neural net to perform regression on the Wisconsin Diagnostic Breast Cancer (WDBC) database. There are 569 instances with 30 features on which to perform the regression. One is the target feature for cancer being either malignant or benign. I ran this NN while varying batch size, learning rate, first layer size, and second layer size.



Methods

MNIST

I built a relatively simple CNN for classifying MNIST. It takes an input of 28x28 to match the image. It runs two layers, each including a convolutional step, ReLU, pooling, and dropout. The first has 32 filters of size 5x5, the second had 64 filters of size 5x5. After both layers, we run an MLP classifier which consists of a fully connected layer hidden layer, ReLU, dropout, and another fully connected layer that serves as the output. The hidden MLP 'dropout' layer had a size of 128.

The CNN was run entirely through PyTorch. I used a batch size of 16 and a learning rate of 0.0001 and ran for one training epoch. I generated a confusion matrix of results as well as a correct classification rate, averaged across many runs.

To the right is an example of a CNN for MNIST with a fairly similar architecture to mine, although some of my values differed.

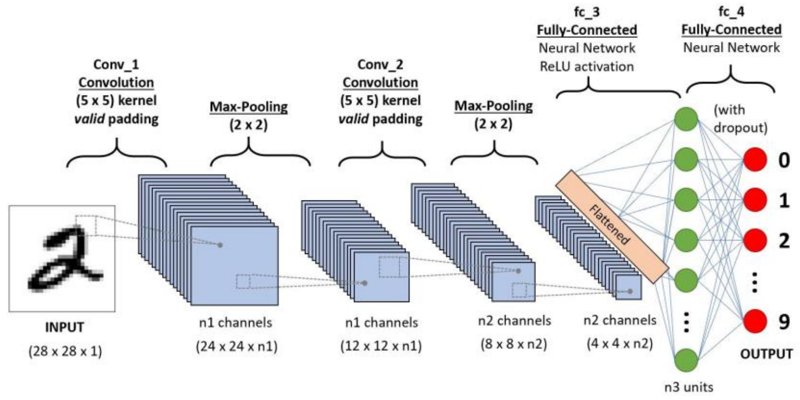
WDBC

For WDBC regression, I used a simple MLP. The input layer takes in all 30 input values and passes them to a hidden layer of size 64, another hidden layer of size 32, and an output layer of size 1.

This MLP was run through PyTorch. I used a batch size of 32 and a learning rate of 0.001 as default. I ran this for 10 epochs.

For results, we create a scatter plot of predictions vs. actual data values.

Additionally, I track MSE loss on the test set.



https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

Datasets

MNIST

The MNIST database contains 60,000 examples (50,000 train, 10,000 test) of handwritten digits from 0-9. These were taken from US high school students and census data. Each digit has been standardized into a 28x28 greyscale image, creating an input size of 28x28x1.

This dataset was very straightforward to work with. It can be directly loaded through PyTorch and is already divided into training and test datasets.

Suvanjanprasai, CC BY-SA 4.0, via Wikimedia Commons

WDBC

The WDBC database contains 569 sets (80% for train, 20% for test) of 30 features. These features are the mean, standard error, and mean of three largest values, for ten metrics measured on cell nuclei. These are radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. The measured cells are all from breast lumps suspected of being cancerous. Additionally, the target value is encoded as B for benign and M for malignant.

As all the features are already numeric, no further processing was needed to use these. The target values were changed from B and M to 0 and 1 for encoding numerically. All the data was split into a training and testing sets, and the converted into PyTorch tensor datasets.

Experiments - MNIST

Experiment 1

Experiment 1 tested dropout rate values from 0 – 0.9 with a 0.1 interval.

Dropout Rate	Classification Rate
0.0	0.97618
0.1	0.9736
0.2	0.9739
0.3	0.97126
0.4	0.96928
0.5	0.9636
0.6	0.95604
0.7	0.94548
0.8	0.9121
0.9	0.24028

Interestingly, we find that no dropout rate works the best. I think this might be because I applied the same dropout to both the convolutional steps and the MLP. Perhaps we'd see different results if these were separated.

Experiment 1-4 Format

Experiments 1-4 were all conducted with the same format. For each value being tested, the CNN was run 5 times, for one epoch each. The classification rate was averaged across these 5 tests.

Experiment 2

Experiment 2 tested batch sizes from 4 - 512 that doubled each test.

Batch Size	Classification Rate
4	0.98216
8	0.97832
16	0.9744
32	0.96624
64	0.9544
128	0.93474
256	0.9145
512	0.87192

Here we find that the smallest batch sizes work best. This is to be expected because we're letting the weights improve many more times. However, these smaller batch sizes take substantially longer to run, making this a tradeoff one must choose.

Experiment 5

Now we take what we have learned and attempt to get the best performance that we can. We will use dropout of 0, batch size of 4, drop layer of 256, and a learning rate of 0.001.

Experiment 3

Experiment 3 tested drop layer sizes from 4 – 1024 that doubled.

Drop Layer Size	Classification Rate
4	0.3901
8	0.90248
16	0.95808
32	0.96468
64	0.97144
128	0.97246
256	0.97864
512	0.97978
1024	0.97894

Here we find that a larger drop layer size results in increased performance, although a larger size requires longer runtimes. This seems to level out at around 256 neurons.

After training for 10 epochs, we find a classification rate of 98.87%.

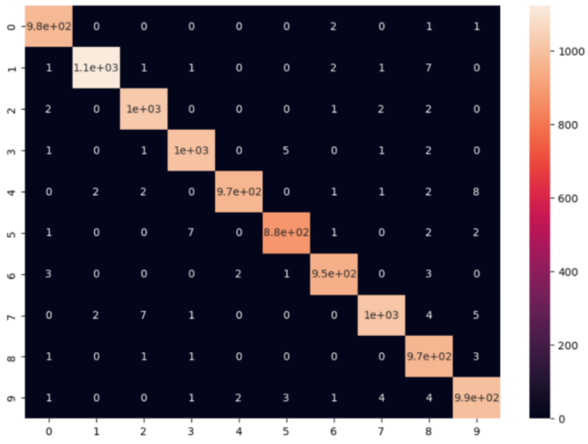
We can also see this data visualized on a confusion matrix.

Experiment 4

Experiment 4 tested learning rates from 0.000001 – 1, exponentially increasing.

Learning Rate	Classification Rate
0.000001	0.66022
0.00001	0.90178
0.0001	0.97492
0.001	0.9858
0.01	0.95562
0.1	0.10538
1	0.10156

Here we find best performance with a learning rate around 0.001. As it gets too small or too large, we notice a huge drop, with 0.1 and above performing essentially at random.



Experiments - WDBC

Experiment 1

Experiment 1 tested batch size from 4-512 that doubled each test.

Batch Size	MSE Test Loss
4	0.029195
8	0.029523
16	0.032435
32	0.040142
64	0.052932
128	0.071059
256	0.097215
512	0.131650

Here we see that once again, a smaller batch size has better performance. A smaller batch size took significantly longer with this data set, and thus this improved loss comes at a substantially increase of runtime.

Experiment 1-4 Format

Experiments 1-4 were all conducted with the same format. For each value being tested, the MLP was run 500 times, for one epoch each. The classification rate was averaged across these 500 tests.

Experiment 2

Experiment 2 first layer sizes from 4-512 that doubled each test

First Layer Size	MSE Test Loss
4	0.155518
8	0.078991
16	0.060661
32	0.049175
64	0.040021
128	0.036586
256	0.037419
512	0.044507

It seems that the optimal first layer size is around 128. All known data points seem to support this conclusion.

Experiment 5

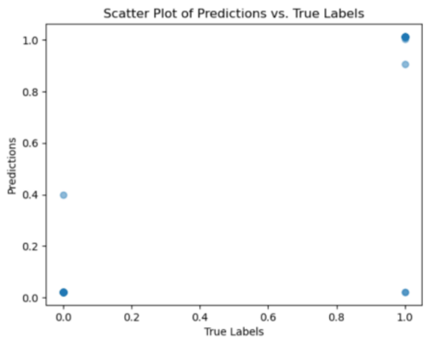
Now we take what we have learned and attempt to get the best performance that we can. We will use batch size of 4, first layer of size 128, second layer size 128, and a learning rate of 0.01.
After training for 500 epochs, we find a MSE test loss of 0.018601. We can see this on a scatter plot and histogram.

Experiment 3

Experiment 3 tested second layer sizes from 4 – 512 that doubled.

Second Layer Size	MSE Test Loss
4	0.040408
8	0.040509
16	0.040445
32	0.040207
64	0.040117
128	0.040802
256	0.040569
512	0.040017

Here we find that second layer size seems to have limited impact on loss. This likely would be different if we combined second layer and first layer growth, but with a small first layer, as was tested with, the second layer simply doesn't matter much.

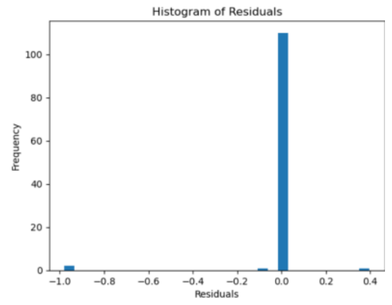


Experiment 4

Experiment 4 tested learning rates from 0.000001 – 1, exponentially increasing.

Learning Rate	Classification Rate
0.000001	0.378701
0.00001	0.337139
0.0001	0.101918
0.001	0.039951
0.01	0.026398
0.1	0.156390
1	884.03712

Here we find an ideal learning rate around 0.01. Smaller learning rates don't perform great, and as we increase the rate things get worse very quickly.



We see that nearly everything gets correctly classified, save for a few examples which are false negatives, and one slight false positive.

Summary

MNIST

Overall, building the CNN and using it on MNIST was a surprisingly straightforward process. I expected it to be much more difficult, and also require much more training than one epoch to perform well. I would be interested to see how much performance could really be improved by continuing to scale up the size of the CNN or if MNIST has a natural breaking point at which all the examples left are so odd that the computer has no idea what to do with them.

If I were to continue working with this, I would like to apply the same CNN to the EMNIST database (which includes all characters, not just digits) to see how it would perform at that scale. Would the same patterns of performance apply, or would we need to scale the CNN much larger to account for many more filter combinations? I would be very interested to see how this works.

WDBC

WDBC was a bit more difficult to work with, and I also struggled a bit more with understanding the concept of regression for NNs in general. Once I got the data loaded in properly and I was able to convert the M and B notation into numeric values, using PyTorch was actually quite simple and the MLP went together quite quickly.

I would be interested to see how well a similar regression method with an MLP would perform on a larger dataset. With only 569 pieces of data to work with and most of that being needed to train, there was not much data to actual run tests on. This is despite that fact that each datapoint had 30 values to look at, enough to surely improve classification further.

Overall Thoughts

It was very interesting to get to have hands-on experience with one of the most notable topics in computing right now. I feel that I learned a lot and gained a better intuition about NNs because of this project.
PyTorch does make building NNs very simple and straightforward. Once data is properly loaded in and a loop is made for training, building the actual network requires almost no work. Additionally, it was very easy to change things for testing. Adjusting the size of a layer or even adding another one comes down to just a line or two of code. It made it very easy to play around with the NN and see what effects changes to the structure could have on performance.

Continually with this project, my biggest limitation was computing power. If I were able to run these tests with a more powerful machine than a less-than-prime Dell laptop, I would have been able to scale up the NNs substantially and be able to train with more than one or a few epochs.

Additionally, datasets such as CIFAR-10 or CIFAR-100 would be very interesting to experiment with, but computational power limitations made these datasets out of my reach for this project.