Modeling 10

Ryan Swope

April 21, 2021

1 Artificial Neural Networks

1.1 Introduction to ANNs

Artificial Neural Networks, or ANNs, are fairly rudimentary but powerful and diverse examples of artificial intelligence. Their primary strengths lay in pattern recognition, although their applications extend far beyond this. The appeal of ANNs for our purposes are their aforementioned robustness, simple to implement topology, and straightforward training mechanism. ANNs are comprised of layers of nodes. A node contains a single numerical value, and cannot influence nodes in it's home layer. There are three types of layer: input, hidden, and output. The input layer has the same number of nodes at there are input values, and the output layer has the same number of nodes as output values; however, the hidden layers can have any number of nodes. The function of the input and output layers is clear, and the function of the hidden layer(s) is to connect the input layer to the output. When data is passed from an input node, it contributes a certain value to each of the nodes in the following layer. This contribution is determined by the weights and biases. These are matrices that describe the transition between the N dimensional input layer vector and the M dimensional following layer vector. Weights multiply the value of a node and biases are added to it. The value going into a node, which is determined by the sum of the contribution from the previous layer, is then passed through an activation function such as the sigmoid function which normalizes the value, and this is the value stored in the node. This process is repeated, or propagated, through the network until the output layer is reached. At the output layer, each node corresponds to a "choice" by the network. The node with the highest value is the choice the network is guessing the input to be. For example, the network we wrote for the first part of this project learned to recognized the digits 0-9. If the network thought the digit it was seeing was an eight, the eighth node would have a large value (Usually $>.95$, although the only requirement is it is the largest value).

Upon initializing the network, random values for the weights and biases are chosen, so the value predicted by the network is random. To train it to correctly recognize data, a process called back-propagation is used. In essence,

[1] https://pathmind.com/wiki/neural-network

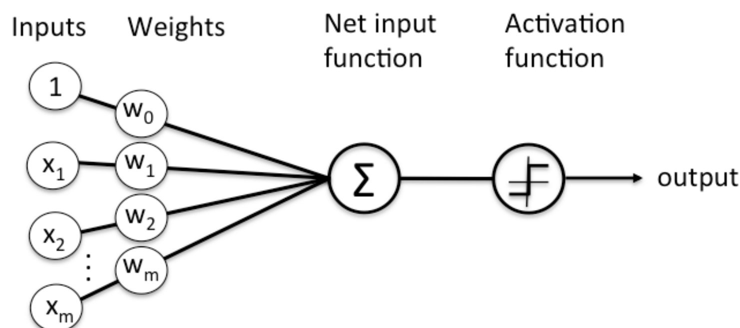


Figure 1: Example neural network topology [1]

back-propagation works as follows: An input with a known value is passed through the network and the output is "rated" by a cost function. The better the guess is, the lower the cost. Based on the cost value and a parameter called the learning rate (typically η), the network is nudged in such a way that it would produce a "more correct" output. This is done by changing the weights and biases of activated nodes by small values. The larger the learning rate, the larger these vhanges will be. Because these values are wrapped by the activation function, they will never exceed 1 or fall below 0. This is done many, many times until the network is sufficiently trained. Then, an input can be sent through the network to retrieve its output.

1.2 MNIST

As stated earlier, the goal of this network was to recognize the digits 0-9. Our training data was obtained from the MNIST, an extremely common ANN dataset. It contains 60,000 labeled digits that were hand written by school children and then digitized into 28x28 images. These images can then be unraveled into vectors of dimension 784, with each value corresponding to the gray-scale value of that pixel, between 0 and 255. This value could then be fed through the network, with an output layer of 10 nodes corresponding to each digit. MNIST was further split into three categories: training data, test data, and validation data. Training data is used to train the network through the process of back propagation, and the test data acts as "new" data the network can then test itself on. The validation data is similar, as after each training session, or epoch, the network runs through the validation data and returns this as an empirical rating of how well the network is doing. The improvement in recognition of the validation data over time is called the recognition rate.

My best performing network had 50 hidden layers, trained for 30 epochs and had $\eta = 3.0$. This seemed to be the Goldilocks zone for all three parameters.

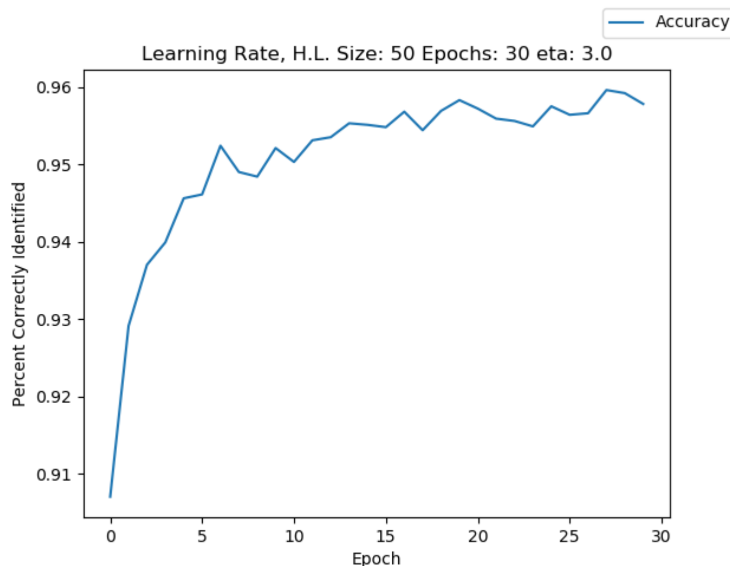


Figure 2: Best network topology learning curve.

More hidden layers caused the network to take longer to run with no improvement or even worse performance, 30 epochs was sufficient to train the network, and a learning rate of 3 was large enough to quickly converge to the local minimum of the cost function, but was not so large that the network jumped around too much to ever converge. This network maxed out at a recognition rate of about .96, which is fairly good.

Unfortunately, the network did not perform this well on the handwritten images I fed through it. They were made in a simple program called Paintbrush in 28x28 PNGs. I tested it on three sets of the digits and the network was only able to recognize 16/30 - barely better than a coin flip. This baffled me considering how well it had done on the MNIST data, although apparently this is not uncommon. Several of my peers had the same issue. I suspect there are a few factors at play. Creating the digits with a black brush meant the "activated" pixels all had values of 255, whereas the handwritten ones were likely not near that value, so the network was not used to handling these large values. The converse could be said about the white space; paper, especially after being written on by a child, will not be perfectly white. Other, more unlikely factors include my handwriting and brush thickness. However, I find both of these unlikely as I tested the network on other digits written by Rachel Price, and the results were similar. In particular the network struggled with the digits 4, 6, 7 and 9. Certain features of these digits, especially when handwritten, are present in other digits, thus offering a possible explanation for the difficulty the network had with them. The results of all thirty digits are in Table 1.

Digit	Correct (Out of 3)	Confused Digits
0	3	-
1	3	-
2	2	3
3	2	0
4	1	6, 7
5	2	3
6	0	5, 9, 2
7	0	3, 8, 3
8	2	9
9	1	4, 8

Table 1: Outcomes of handwritten values.

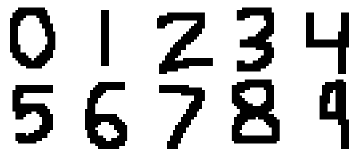
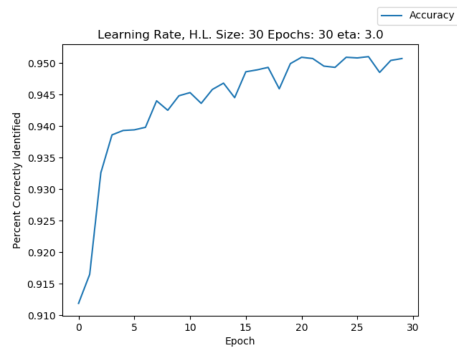


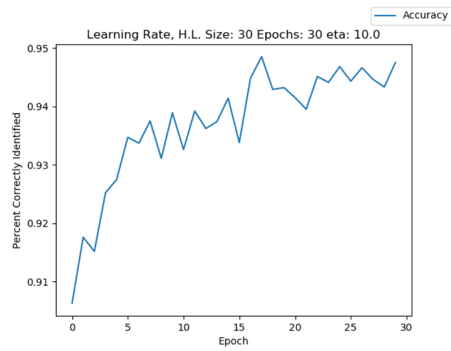
Figure 3: One set of my handwritten digits.

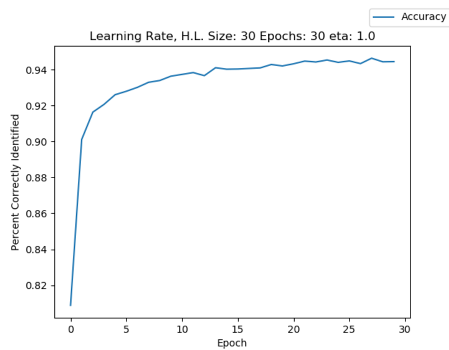
1.3 Network Topology

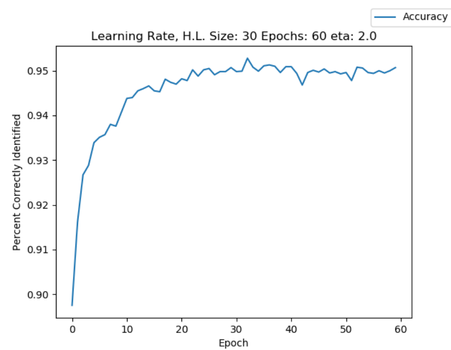
As I mentioned before, this was my nest performing network, although it was not the only one I tested. I tested several others with various hidden layer sizes, epochs of training, and learning rates. For epochs and hidden layers it was mostly a question of diminished return. Although in theory training the network for longer will results in better recognition rate, in practice the maximum possible value is asymptotically approached, and beyond 30 epochs there was no perceivable improvement. Similarly, adding hidden layer nodes increased the length of time it took the network to train, but no tangible improvements could be seen. In fact, more hidden layer nodes often made the network worse. In contrast, the learning rate did have a more or less optimal value, that could approach the local minimum of the cost function quickly (less epochs needed) without being too large that the network could never settle into this minimum. My optimal value of 3.0 was mostly a guess and check value, but a more analytical solution could certainly be found, although this difference is easily compensated by simply letting the network run for sufficiently many epochs.

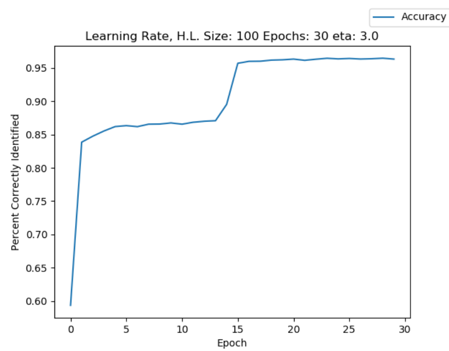
As you can see, the network struggles to achieve recognition rates above .95. There are a number of factors at play, including bad images that even a human couldn't recognize, but considering the rudimentary construction of our network I'd consider a .95 recognition rate a success.

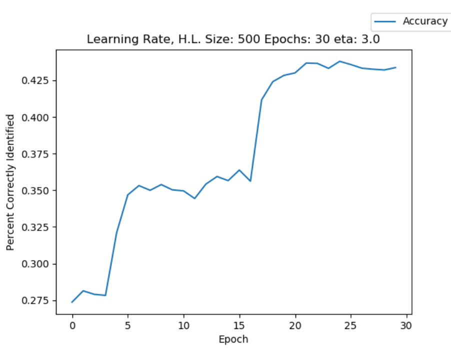

Accuracy
Learning Rate, H.L. Size: 30 Epochs: 30 eta: 3.0
Percent Correctly Identified
0.950
0.945
0.940
0.935
0.930
0.925
0.920
0.915
0.910
0 5 10 15 20 25 30
Epoch


Accuracy
Learning Rate, H.L. Size: 30 Epochs: 30 eta: 10.0
Percent Correctly Identified
0.95
0.94
0.93
0.92
0.91
0 5 10 15 20 25 30
Epoch


Accuracy
Learning Rate, H.L. Size: 30 Epochs: 30 eta: 1.0
Percent Correctly Identified
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0 5 10 15 20 25 30
Epoch


Accuracy
Learning Rate, H.L. Size: 30 Epochs: 60 eta: 2.0
Percent Correctly Identified
0.95
0.94
0.93
0.92
0.91
0.90
0 10 20 30 40 50 60
Epoch


Accuracy
Learning Rate, H.L. Size: 100 Epochs: 30 eta: 3.0
Percent Correctly Identified
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0 5 10 15 20 25 30
Epoch


Accuracy
Learning Rate, H.L. Size: 500 Epochs: 30 eta: 3.0
Percent Correctly Identified
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0 5 10 15 20 25 30
Epoch

1.4 Comments on Optimization

When considering optimization with ANNs, or any artificial intelligence, there is a clear trade off: optimized recognition or optimized training. In theory, with a much larger training dataset, multiple networks or a more robust detection algorithm (incorporation of structures larger than 1 pixel), our network could be drastically improved, but this is much more difficult to implement. For our purposes our network was, in my opinion, the right one to use. However, for large-scale implementations in, say, mobile check deposits where it is imperative the recognition rate is far greater than .95, the more complex network would be necessary.

2 Tensorflow Object Recognition

2.1 Introduction

Without getting into too many gritty details, Tensorflow is an open-source machine learning platform maintained by Google. It has a very high learning curve which I experienced firsthand, but with that comes a huge amount of power, both literally in the form of GPU acceleration and figuratively in its applications within deep learning and and other advanced machine learning techniques. One of its best features for entry level users such as myself are their pretrained neural networks. These networks are specific to a certain application and can be used as-is if they are already capable of performing the desired task, or they can be retrained on a new set of training data without much of the heavy lifting involved in trained a complex neural network.

2.2 Object Detection Training

I used two pretrained models, the `fast_rcnn_inception_v2` model and the `ssd_mobilenet_v2_coco` in an attempt to train a network to recognize a knife in either an image or a video. After the incredibly arduous task of setting up Tensorflow, I experience significant difficulty with `fast_rcnn_inception_v2`. It would evolve as expected until about step 500, at which point the loss would explode to a value on the order of 1e13, from somewhere around. Loss is a measure of how well a network is doing and less is better, so this presented a serious problem. I was unable to find others with the same issue (that it would explode after several hundred steps), so I tried capping the training at 500 steps. Unfortunately, the network was simply no well trained enough and was not able to recognize knives in images. The results of the training could be seen in Tensorboard, which provides learning curve rates for your models.

The `ssd_mobilenet_v2_coco` model trained successfully overnight, and the losses seemed to be approaching an asymptotic value. Ideally, I would've been able to train it for longer, as the Regulation Losses were still increasing, meaning my model was still improving, but that was not a possibility. Overall, however, the model was able to train for over 3,000 steps, which was about as good as I

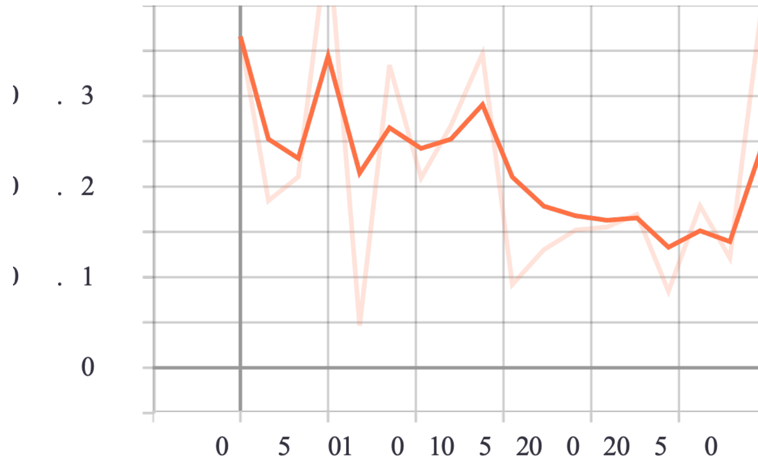


Figure 4: Total Losses during training. `fast_rcnn_inception_v2` The axes turned out odd.

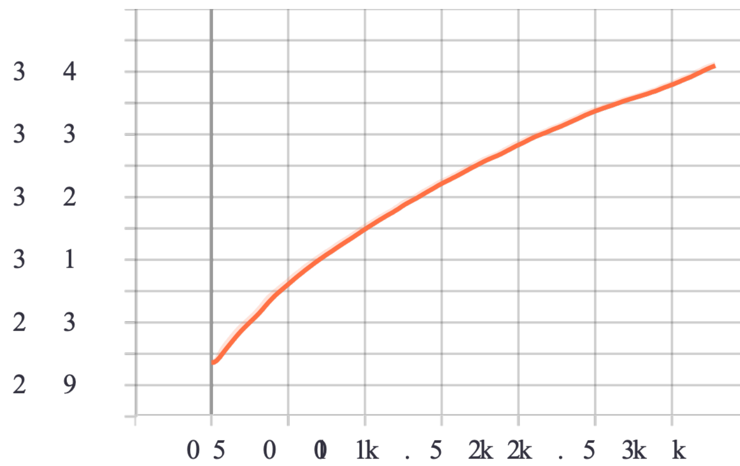


Figure 5: Regulation Losses during `ssd_mobilenet_v2_coco` training. The axes turned out odd.

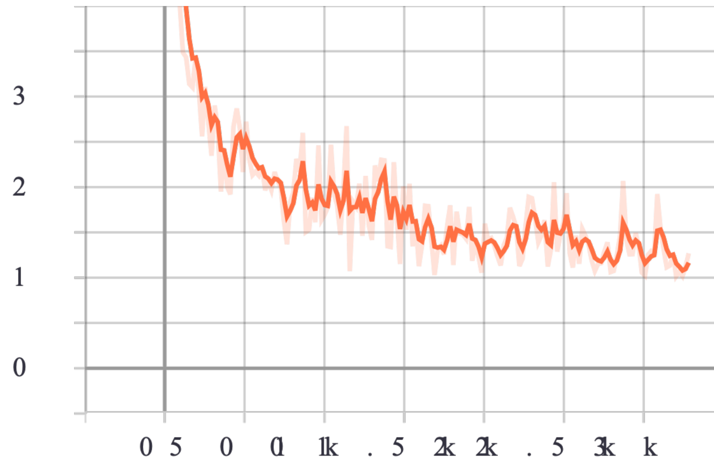


Figure 6: Total Losses during `ssd_mobilenet_v2_coco` training. The axes turned out odd.

could've hoped for without GPU acceleration. But, there were more problems with Tensorflow. While the `ssd_mobilenet_v2_coco` model successfully trained, I then had trouble saving this model and then applying it to images. It was unable to recognize the knife in most of the images, and for some reason they're not labeled, but eventually I was able to get the network to work on a few images, and considering all the headache this has cause up to this point, that is a success.

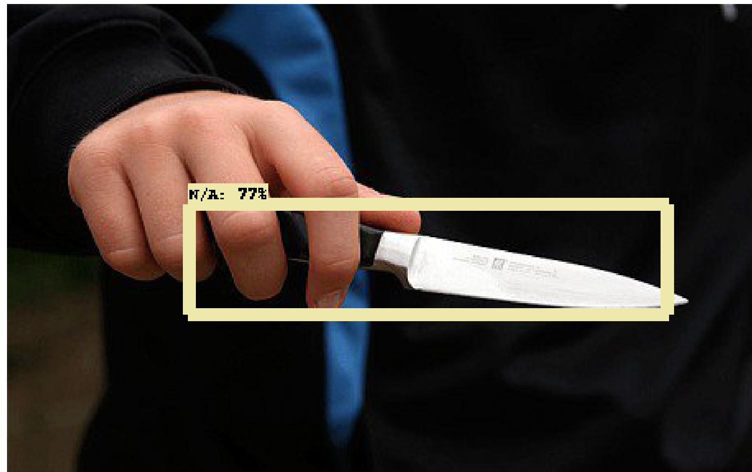


Figure 7: Recognized Image

Figure 8: Recognized Image

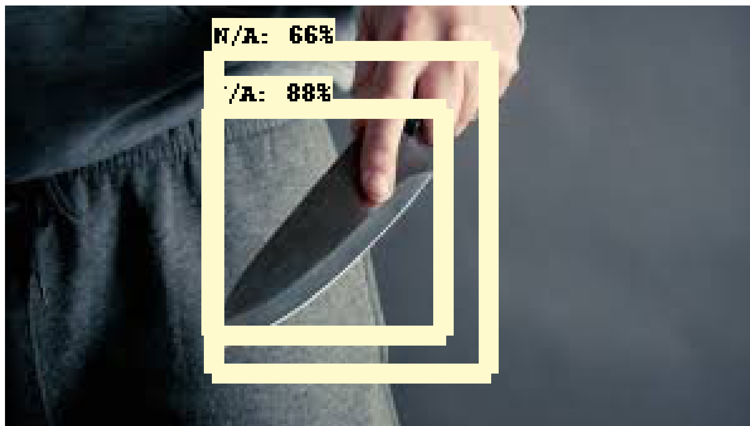


Figure 9: Recognized Image

2.3 Closing Remarks on Tensorflow

Tensorflow assumes a certain knowledge from its users, and I am not proficient enough to fully understand Tensorflow. As a result, my experience was very much a trial by fire. That being said, I'm glad I went through it, and I would hope that were I to attempt this again, I would have learned many valuable lessons this time around that would save me time in the future. Tensorflow 2.0 is also relatively new, and unfortunately most tutorials are for Tensorflow 1.1x,

so as time goes on I'm hopeful tutorials are updated to reflect the significant changes that were made between the two versions. Despite what was essentially a surface level expedition into Tensorflow, its power and versatility were more than apparent, and with more time to learn I think it would be an incredibly powerful tool. There may be a bit more to say, but I'm exhausted from the last several days I spent wrestling with Tensorflow, and despite my marginal success I am a bit disappointed with what little I have to show for it, so I'll leave it at that.

3 References

I would like to thank EdjeElectronics for his tutorial on using Tensorflow in Github, and Rachel Price for providing her handwritten numbers and help debugging code.