CS 335 - Artificial Intelligence and Machine Learning
Lab 5 - Neural Networks
Yash Sharma - 17D070059

1 Task 2.1 - `taskSquare`

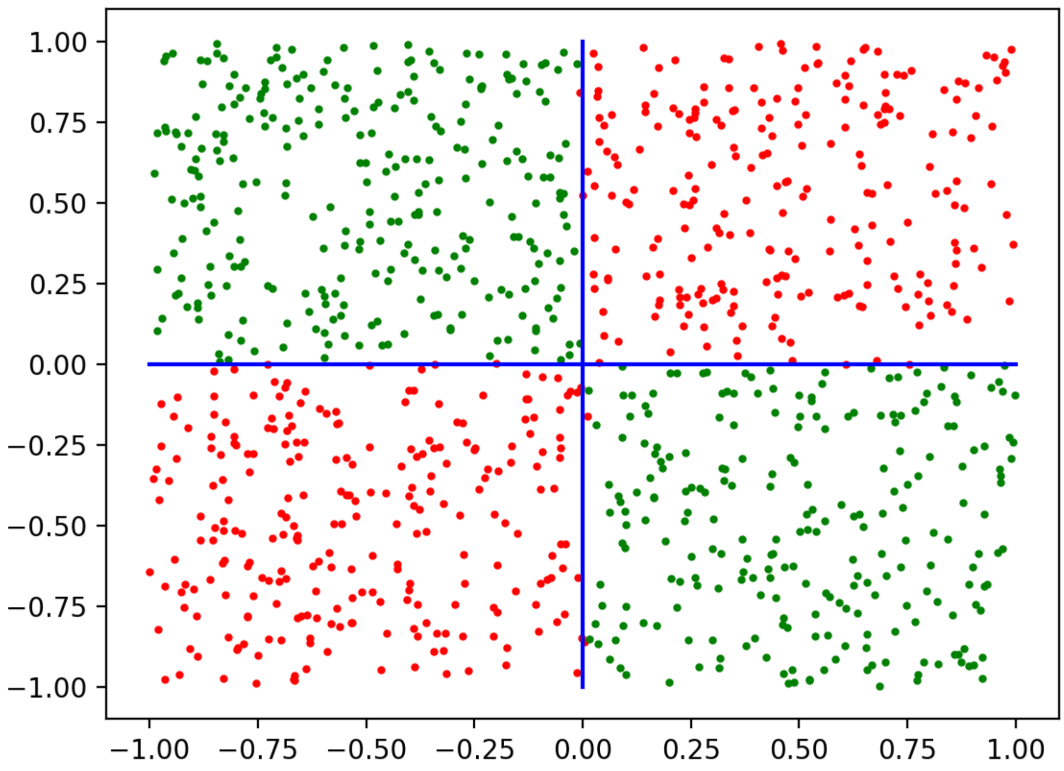

Figure 1: Result of taskSquare

[1] Architecture -

The first layer is a `FullyConnectedLayer` with 2 in_nodes and 4 out_nodes and RELU activation. With random seed as 42, 4 was the minimum number of hidden

[1]For any task, cross entropy loss has not been normalised by number of examples in the batch size, but gradients have been to avoid over-sensitivity to learning rate and batch size during backprop

nodes I could have to achieve more than 90% accuracy. With higher number of hidden nodes, say 7, its more than 90% with any random seed. This could show robustness to local minima or better perception ability.

The second layer is again a `FullyConnectedLayer`, this time with 4 in_nodes and 2 out_nodes and a softmax activation. The two out_nodes are ultimately taken as predictions for each training example feeded into the network.

Figure 1 is the result. Takes a couple of seconds to train.

Test accuracy achieved - 98.8 %

Other hyperparameters -

$$number\_of\_epochs = 10$$
$$learning\_rate = 0.1$$
$$batch\_size = 20$$

2 Task 2.2 - `taskSemiCircle`

Architecture - The implementation is kept simple and similar to `taskSquare`.

The first layer is a `FullyConnectedLayer` with 2 in_nodes and 2 out_nodes and RELU activation. With random seed as 42, 2 was the minimum number of hidden nodes I could have to achieve more than 90% accuracy. With higher number of hidden nodes, say 5 or 6, its more than 90% with any random seed.

The second layer is again a `FullyConnectedLayer`, this time with 2 in_nodes and 2 out_nodes and a softmax activation. The two out_nodes are ultimately taken as predictions for each training example feeded into the network.

Figure 2 is the result. Takes a couple of seconds to train.

Test accuracy achieved - 97 %

Other hyperparameters -

$$number\_of\_epochs = 15$$
$$learning\_rate = 0.1$$

$$batch\_size = 20$$

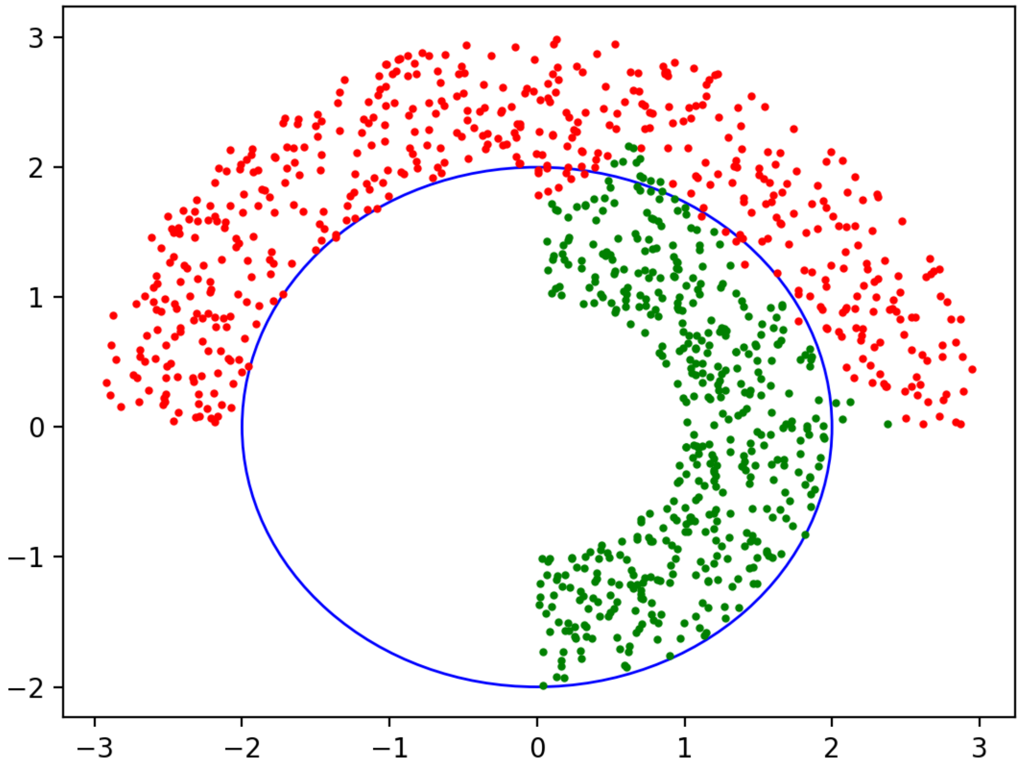

Figure 2: Result of taskSemiCircle

3 Task 2.3 - `taskMnist`

This one required a more complicated architecture to achieve a high test accuracy.%

Architecture -

The first layer is a `FullyConnectedLayer` with 784 in_nodes and 15 out_nodes and RELU activation.

The second layer again is a `FullyConnectedLayer` with 15 in_nodes and 15 out_nodes and RELU activation. As discussed by sir in class, it's common practice to keep equal number of hidden nodes across certain hidden layers.

The final layer is a `FullyConnectedLayer` with 20 in_nodes and 10 out_nodes representing the 10 classes, with softmax activation.

Test accuracy achieved - 94.55 %

Other hyperparameters -

$$number\_of\_epochs = 5$$
$$learning\_rate = 0.1$$
$$batch\_size = 50$$

Takes about 30 seconds to train. I dropped the idea of keeping the hidden nodes of the order of input for this task as a simpler model was giving me the results. Occam's Razor :)

4 Task 2.4 - `taskCifar10`

This had to be done with a CNN architecture. A simple or even complex feed forward architecture only gave upto 30% results.

The architecture used is a very simple and toned down version of common architectures found on web for CIFAR10 dataset

Input to the first layer was of size $3 \times 32 \times 32$ for each training example in the batch.

Final output is predictions vectors in 10 classes for the input RGB image.

Architecture -

First layer is a `ConvolutionLayer` that takes $3 \times 32 \times 32$ input image and outputs a deep representation of size $32 \times 10 \times 10$ by using 32×3 (out_depth X in_depth) convolution filters of size 5×5 with stride of 3, no padding. Activation is RELU.

Second layer is an `AvgPoolingLayer` that downsamples the previous layers activations to give an output of size $32 \times 4 \times 4$ by using an average filter of size 4×4, with a stride of 2.

This is then flattened to give an output of 512 nodes.

Finally a fully connected layer to map these 512 input nodes to 10 classes, using softmax activation.

Achieved a test accuracy of around 50.2 % with part of the training data (5000 samples instead of 40000), and 55.32% with the entire training data.

Roughly took 8 mins to train for the former, and 2 hours for the latter.

Other hyperparamaters -

$$number\_of\_epochs = 50$$

$$learning\_rate = 0.1$$

$$batch\_size = 200\ (50\ for\ small\ train)$$

I didn't increase alpha beyond 0.1 for the fear of overshooting a good minima.

Around 10 epochs were enough to clear the treshold of 35% accuracy, I trained it for longer just to see how far it goes.

Going a bit outside the general implementation found online. I replaced the flatten and FC layers with another Convolutional Layer (converting input into 10 x 1 x 1 output) using softmax activation and then squeezing the 1 size arrays using flatten. The results were very similar, and I now realise that a Convolutional Layer of this form would mathematically equate a Fully Connected layer.

References

1. Cross Entropy function - https://stackoverflow.com/questions/47377222/cross-entropy-function-python

2. Learning rate issues - https://stackoverflow.com/questions/37044600/sudden-drop-in-accuracy-while-training-a-deep-neural-net

3. Convolutional Neural Network Implementational Understanding - http://cs231n.github.io/convolutional-networks/

4. Understanding `np.einsum` - https://stackoverflow.com/questions/26089893/understanding-numpys-einsum and https://docs.scipy.org/doc/numpy/reference/generated/numpy.einsum.html

5. Architecture for CIFAR10 net - https://www.researchgate.net/figure/Structure-of-CIFAR10-quick-model_fig2_312170477