CSCE 4613 – Artificial Intelligence
Fall 2019
Assignment 3

Carson Molder
cbmolder@uark.edu
010755069

Tyler Tracy
tgtracy@uark.edu
010805685

1 General Questions

Did you implement the basic framework of a feed-forward multi-layer neural network?

Yes, the basic framework of such a neural network was implemented. The networks were written in the Python programming language using the robust machine learning library Pytorch.

Pytorch provides many functions that ease the implementation of neural networks without sacrificing capability or performance. We used the Pytorch library to implement many parts of the network, from the input and output tensors, network layers, loss calculation and backpropagation, and optimization.

Did the neural network test whether feed-forward and back-propagation work?

Yes. For each neural network, the loss is calculated every epoch and at set intervals is displayed on the console during the training process. Furthermore, comparisons between the pre-training and post-training results show that adjustments were made to the weights and biases within the networks, since both networks became more accurate at estimating their respective functions.

Which data sets did you use?

XOR neural network For the XOR neural network, a simple training set is used – the truth table for the XOR function. The labels are the results of each operation and the inputs are the two values for the two inputs that the XOR function takes in. A truth table is provided in Figure 1.

Figure 1: The truth table for the XOR function.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

To test the XOR neural network, we first ran the same truth table through the neural network to see if the values aligned with the truth table. However, a more insightful approach is to test a range of values in the interval $[0, 1]$ for both A and B.

Therefore, the final version of our XOR network is tested on a 1×1 block of points with a small step size (set by default to 0.01). The results for this test set are plotted both before and after training to see if the network has learned the XOR function. Blue areas represent a result of 0 and green areas represent a result of 1.

x^2 neural network To train the x^2 neural network, a random batch of x-values (capped to the range $[-10, 10]$ by default) is generated. By default, this batch is 100 elements long, but the training batch size can be changed in the neural network's code file. The labels for each entry are simply the x-value squared.

The x^2 neural network code also generates random y-values for each x value that are limited to the range $[\Delta y - x, \Delta y + x]$, where Δy is a value (set to 10 by default) that is configurable in the network's code file. However, the y-values are never used in the training of the network and are only used to provide samples to see if the network correctly identifies whether a y-value falls above or below its estimate for x^2.

The x^2 neural network has two testing data sets. The first is generated similarly to the training set, with a random batch of x-values associated with random y-values capped to a certain distance from the actual value of x^2. Each testing element is printed out to the console, with the neural network's estimate for x^2, the actual value of x^2, and its guess on if $y > x^2$ based on its estimate.

The second testing data set for the x^2 neural network was a set of points in the (x, y) plane, like the testing data set for the XOR neural network. A $2x \times x^2$ set of data points with a default step size of 0.25 and defaut x of 10 is generated. The domain of these points is $[-x, x]$ and the range is $[0, x^2]$. Every point generated is ran through the neural network both before and after training and the results are plotted against the function $y = x^2$. Blue areas represent values the neural network estimates are less than x^2 and green areas represent values the neural network estimates are greater than x^2.

Did you implement or think about how to change the number of layers or nodes?

During the implementation process, the amount of layers, number of neurons per layer, and activation function of the layers were all tweaked until the neural network began to adapt to the training data.

The XOR network is implemented using a 2-neuron input layer (for the two inputs), one 16-neuron hidden layer, and a 2-neuron output layer. The output layer uses one-hot vector classification, in which the first neuron is activated when the neural network estimates the output to be 0 and the second neuron is activated when the neural network estimates the output to be 1. The neural network's "guess" can be interpreted to be the neuron that is the most activated.

The x^2 network is implemented using a 1-neuron input layer (for the value of x), a 1-neuron output layer (for the estimate of x^2), and two hidden layers. The first hidden layer has 4 neurons and the second has 16, which tries to mimic a quadratic "increase" in activation. This is because the neurons use rectified linear (ReLU) activation and thus their activation amount increases linearly instead of quadratically.

Did you prepare the data points for the training examples for the XOR function and the x^2 function?

Yes. The procedures mentioned in the data set description are used to generate the data sets. Each testing point is placed inside a Pytorch tensor object, which is required for the neural network to be able to parse it. The neural network's output is also a Pytorch tensor object, and so these tensors have their contents removed from their context to be printed to the console and/or plotted on a graph using the Python library Matplotlib.

2 XOR Neural Network

The software

The code for the XOR neural network is saved in `net1.py`. If Python is already installed, the neural network can be run by opening the terminal, moving to the file's directory, and running the command `python net1.py`.

Both neural networks require the following Python libraries to be installed: Pytorch, Matplotlib, and Numpy. If the program does not load because of a missing import, installing these respective libraries using the `pip install <package>` command should fix the issue.

The first action the program does is generate a training set using the truth table shown in Figure 1. Each training element is a four-element Python list, with the first two entries being A and B respectively, and the third and fourth entries being the one-hot classification label that the neural network is trying to optimize toward.

As mentioned earlier, the XOR neural network implements three layers: a 2-neuron input layer, one 16-neuron hidden layer, and a 2-neuron output layer. The next step for the program is to generate this neural network using a `Net` class that inherits from the Pytorch `module` class, and print its details to the console. The neural network is initialized with random weights and biases, which means the results are different every time the code is ran.

After initializing the neural network, the network is tested before training with the test set described earlier. Then, the network is trained and the test set is tested again to see if the neural network learned the XOR function. Finally, the before and after results are plotted using Matplotlib.

The training process

A function called `train` takes in the neural network and training set and performs training on the network. The function defines a Pytorch optimizer that adjusts the weights and biases of the network after every training cycle. Two constants, `EPOCHS` and `CUTOFF_LOSS`, define how many cycles the neural network will train on the testing data.

In each training cycle, known as an "epoch", the training data is split into the sample (the first two values, A and B) and the label (the expected result from the neural network, 0 or 1). The sample is ran through the neural network in a forward pass. The forward pass is defined in the `forward` function in the `Net` class, and is automatically called when a value is passed into the net.

The net returns the output tensor, which is ran through a mean-square-error loss function. This is calculated as the following:

$$\frac{1}{2}((V_0 - L_0)^2 + (V_1 - L_1)^2)$$

where V_0 and V_1 are the neural network's estimates for labels 0 and 1 and L_0 and L_1 are the actual values of the labels. One of L_0 and L_1 will always be 0, and one will always be 1–which one is has which value depends on the value of the truth table.

The optimizer's gradient is reset to zero to ensure it starts from the same location at every epoch. Then, the loss is backpropagated through the neural network using the function `loss.backward()` and the optimizer adjusts the weights based on these losses using the function `optimizer.step()`. After every epoch, it prints the loss value to the console. Ideally, this value will decrease after every epoch, but a large learning rate (defined as `LEARN_RATE` in the code) may cause the optimizer to overshoot a "valley" and cause a small increase in loss on the next epoch.

The network stops training, regardless of the number of epochs trained, when its calculated loss reaches below the `CUTOFF_LOSS`. If this threshold is never reached, it instead trains until the maximum number of training cycles, `EPOCHS`, is reached.

The end results

After the XOR neural network completes its training, the $1x1$ graph data points are run through testing again, and the Matplotlib library is used in the function fancy_plot to plot the results. The sample outputs vary every time, but generally the post-training test results fit the XOR function well, with quadrants II and IV consistently indicating a value of 1. Figure 2 shows a sample output of the Matplotlib graph.

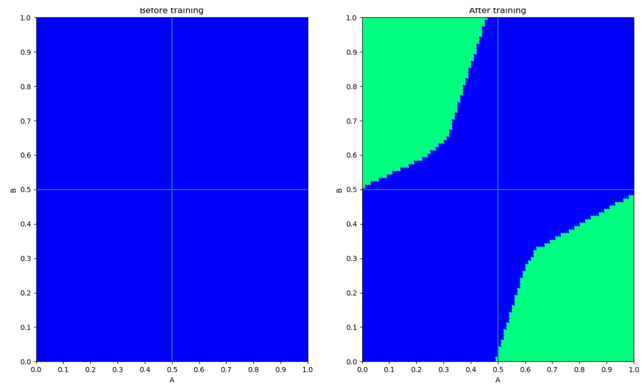

Figure 2: Sample testing results of the XOR function before (left) and after (right). Blue regions indicate a value of 0, green regions indicate a value of 1.

3 x^2 Neural Network

The software

The code for the x^2 neural network is saved in `net2.py`. Like the XOR neural network, it can be ran by running the command `python net2.py`. It also requires the same packages as the XOR neural network.

The x^2 neural network program runs similarly to the XOR neural network program. It first generates two random batches–a training batch and testing batch–of x-and-y value pairs. The training and testing batch sizes are determined by `TRAIN_BATCH_SZ` and `TEST_BATCH_SZ` respectively. The x-values range from `-MAX_X` to `MAX_X`, while the y-values vary from their associated X value by a maximum of `DELTA_Y`.

Next, the program initializes the x^2 neural network which is stored in a `Net` class similar to the one in the XOR neural network. It prints the net to the console, along with the testing and training set tensors. Then, it tests the testing set on the neural net before training and prints the results.

It also calls a function called `fancy_test` which runs the neural net on a series of points in the (x, y) plane, where the range (X) is `-FANCY_TEST_MAX_X` to `FANCY_TEST_MAX_X`, the domain (Y) is `0` to `FANCY_TEST_MAX_X` squared, and the step size between points is `FANCY_TEST_STEP`. A point is plotted as blue (or, alternatively, given a label of 0) if the neural network estimates that it lays below the $y = x^2$ parabola and green (given a label of 1) if the neural network estimates that it lays above the parabola.

Then, the program trains the neural network on the training set and tests the neural network again on the testing set and through the `fancy_test` function. Finally, the function `fancy_plot` is called and the results of the `fancy_test` function before and after the training are shown.

The training process

Similar to the XOR neural network, a function known as `train` is called to train the network. It runs for a predetermined number of epochs and does not have a cutoff loss. This is because the random X values can vary significantly and the loss function for each sample is not an error function, but a difference. For large values, the loss is calculated as follows:

$$|V - L|$$

where V is the neural network's output value (the estimate for x^2) and L is the label (the actual value of x^2).

This type of loss is known as L1 loss. However, for small values close to zero, the loss function becomes a smooth curve, unlike the regular absolute value function which has a sharp point at $x = 0$.

This alternative form of L1 loss is known as smooth L1 loss, and is what the x^2 neural network uses. Smooth L1 loss is able to handle outliers better than MSE loss or regular L1 loss, and its differentiability at zero makes calculating gradients easier.

Like the XOR network, this loss value is backpropagated through the neural network and the optimizer adjusts the network's weights and biases based on how the loss is distributed during the backpropagation process. After an `EPOCH` number of epochs, the network finishes trianing, hopefully with a lower loss value than when it was initialized with random weights and biases.

It is important to note that the y-values generated before the training process are never used in training. They are only used to see how well the network has trained—the network only considers the actual value of x^2 during the learning process. Initially, we tried to implement a one-hot classification system similar to the one for the XOR neural network that took in *both* the x and y value as inputs. This network tried to determine if y was greater or less than x^2 directly.

However, the results were not satisfactory and the network always converged on an output that was the same regardless of the values of x and y. The regression-based approach our final x^2 neural network uses is much simpler, more accurate, and more flexible.

The end results

After the x^2 neural network completes its predetermined amount of training epochs, it then runs the testing values for both the randomly-generated values and (x, y) points again. The randomly-generated values' test results are printed to the console, and the (x, y) points' test results are plotted using Matplotlib using the `fancy_plot` function.

Although the values for `max_x`, the maximum x-value used in the training set, and `fancy_test_max_x`, the maximum x-value plotted on the graph, can be independently varied, the best results appear when the two values are close or equal to each other. Figure 3 shows a sample result when these two values are both set to 10. Figure 4 shows a sample result when `max_x` is increased to 50 but `fancy_test_max_x` remains at 10. Finally, Figure 5 shows a sample result when `max_x` is decreased to 5.

As shown by the three figures, increasing the maximum x value decreases the accuracy of the neural network around

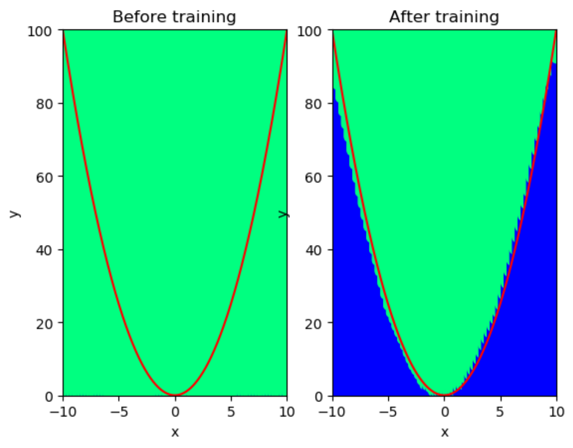

Figure 3: Sample testing results of the x^2 function before (left) and after (right), using a training maximum x value of 10. The blue region indicates where the neural network estimates the region is below x^2 is, the green region indicates where the neural network estimates the region above x^2 is.

the origin. Likewise, decreasing the maximum x value decreases the accuracy of the neural network outside of the origin.

The pre-training data may appear as entirely green or entirely blue. This is because, every time the network is code is run, a different set of initial weights and biases are randomly initialized. If the entire space is green, then the random weights and biases led the network to naively estimate that every y-value is above $y = x^2$. Likewise, if the entire space is green, then the neural network naively estimated that every y-value is below $y = x^2$.

Either way, the results indicate that the network is learning x^2 relatively well within the bounds it is given. It makes sense that increasing the maximum x-value of the training data would decrease accuracy around the origin, because fewer x-values are likely to be around the origin. Likewise, it makes sense that decreasing the maximum x-value of the training data would decrease accuracy outside the origin, because no x-points are allowed outside the bounds.

4 Appendix

Sample console output for each of the neural networks is attached at the end of this report.

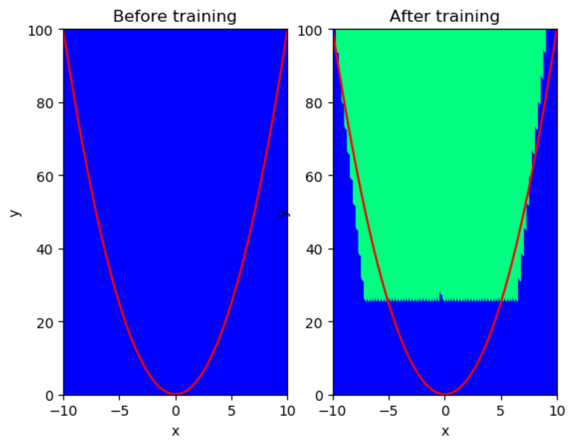

Figure 4: Sample testing results of the x^2 function before (left) and after (right), using a training maximum x value of 50.

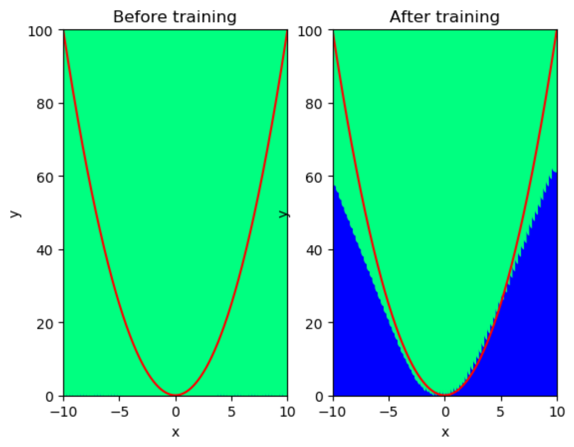

Figure 5: Sample testing results of the x^2 function before (left) and after (right), using a training maximum x value of 5.

XOR neural network sample output

```
--- Net: ---
 Net(\\
  (fc1): Linear(in_features=2, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=2, bias=True)
)

--- Training set: ---
 tensor([[0., 0., 1., 0.],
        [0., 1., 0., 1.],
        [1., 0., 0., 1.],
        [1., 1., 1., 0.]])

--- Testing set: ---
 tensor([[0., 0., 1., 0.],
        [0., 1., 0., 1.],
        [1., 0., 0., 1.],
        [1., 1., 1., 0.]])

--- Testing data before training... ---

--- Training now... ---
Epoch #1  loss: 0.453
Epoch #2  loss: 0.175
Epoch #3  loss: 0.203
Epoch #4  loss: 0.247
Epoch #5  loss: 0.288
Epoch #6  loss: 0.298
Epoch #7  loss: 0.264
Epoch #8  loss: 0.219
Epoch #9  loss: 0.189
Epoch #10 loss: 0.18
Epoch #11 loss: 0.183
Epoch #12 loss: 0.18
Epoch #13 loss: 0.154
Epoch #14 loss: 0.113
Epoch #15 loss: 0.078
Epoch #16 loss: 0.056
Epoch #17 loss: 0.018
Epoch #18 loss: 0.004

--- Testing data after training... ---

--- Plotting test results before and after... ---
```

x^2 neural network sample output

```
--- Net: ---
 Net(
  (fc1): Linear(in_features=1, out_features=4, bias=True)
  (fc2): Linear(in_features=4, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=16, bias=True)
  (fc4): Linear(in_features=16, out_features=1, bias=True)
)

--- Training Set: ---
 tensor([[ 4.5680e+00,  2.5745e+01,  2.0867e+01],
        [-5.2050e+00,  2.0354e+01,  2.7092e+01],
        [-9.6550e+00,  8.4205e+01,  9.3219e+01],
        [-8.0050e+00,  6.4284e+01,  6.4080e+01],
        [-5.4600e+00,  2.1445e+01,  2.9812e+01],
        [-8.9760e+00,  8.2067e+01,  8.0569e+01],
        [ 2.9600e+00,  4.5146e+00,  8.7616e+00],
        [ 4.8940e+00,  3.1543e+01,  2.3951e+01],
        [ 9.6900e-01, -8.1110e+00,  9.3896e-01],
        [-1.4090e+00,  9.4773e+00,  1.9853e+00],
        [-6.1970e+00,  3.2883e+01,  3.8403e+01],
        [-1.3320e+00, -5.8598e+00,  1.7742e+00],
        [-6.4240e+00,  4.2088e+01,  4.1268e+01],
        [-6.5560e+00,  4.8644e+01,  4.2981e+01],
        [ 6.8310e+00,  4.9639e+01,  4.6663e+01],
        [ 6.8090e+00,  5.3262e+01,  4.6362e+01],
        [-5.0140e+00,  2.5017e+01,  2.5140e+01],
        [-7.0020e+00,  4.5674e+01,  4.9028e+01],
        [ 4.3240e+00,  2.7689e+01,  1.8697e+01],
        [-3.7330e+00,  1.6052e+01,  1.3935e+01],
        [ 4.2920e+00,  1.4174e+01,  1.8421e+01],
        [-2.3570e+00, -1.7546e+00,  5.5554e+00],
        [ 3.1410e+00,  4.7869e+00,  9.8659e+00],
        [-3.6300e+00,  1.2093e+01,  1.3177e+01],
        [ 1.8760e+00,  7.7004e+00,  3.5194e+00],
        [-3.3500e-01,  8.3862e+00,  1.1223e-01],
        [-4.6360e+00,  2.1332e+01,  2.1492e+01],
        [ 6.6580e+00,  5.1139e+01,  4.4329e+01],
        [-7.3520e+00,  5.9207e+01,  5.4052e+01],
        [ 6.5020e+00,  4.6316e+01,  4.2276e+01],
        [ 1.6280e+00, -4.0366e+00,  2.6504e+00],
        [-2.1870e+00,  8.1430e+00,  4.7830e+00],
        [-2.6000e+00,  1.3084e+01,  6.7600e+00],
        [-2.3650e+00,  1.0290e+01,  5.5932e+00],
        [-9.4230e+00,  8.9538e+01,  8.8793e+01],
        [ 2.2170e+00,  6.1021e+00,  4.9151e+00],
        [ 7.8770e+00,  6.1500e+01,  6.2047e+01],
        [ 3.8230e+00,  5.8873e+00,  1.4615e+01],
        [-3.9360e+00,  1.8837e+01,  1.5492e+01],
        [-8.6700e-01, -6.3813e+00,  7.5169e-01],
        [-6.0030e+00,  2.9917e+01,  3.6036e+01],
        [ 1.2570e+00,  1.0499e+01,  1.5800e+00],
        [ 6.4130e+00,  4.5257e+01,  4.1127e+01],
        [ 1.4640e+00,  1.1723e+01,  2.1433e+00],
        [-9.1770e+00,  8.3734e+01,  8.4217e+01],
        [-6.8170e+00,  5.2834e+01,  4.6471e+01],
        [ 7.5130e+00,  4.7813e+01,  5.6445e+01],
        [ 1.7610e+00, -6.2579e+00,  3.1011e+00],
        [ 6.1100e+00,  2.8610e+01,  3.7332e+01],
        [ 6.4000e-02, -6.1219e+00,  4.0960e-03],
        [ 1.2260e+00,  4.0281e+00,  1.5031e+00],
        [-9.2210e+00,  7.9473e+01,  8.5027e+01],
        [ 4.9850e+00,  2.3583e+01,  2.4850e+01],
        [ 7.6040e+00,  5.6397e+01,  5.7821e+01],
        [-1.8050e+00,  6.6640e+00,  3.2580e+00],
        [-2.5310e+00,  8.6796e-01,  6.4060e+00],
        [-1.7530e+00,  1.0070e+01,  3.0730e+00],
        [-6.5980e+00,  3.3687e+01,  4.3534e+01],
        [-5.9890e+00,  3.7171e+01,  3.5868e+01],
        [-3.4010e+00,  1.7398e+00,  1.1567e+01],
        [ 6.8400e-01, -2.4751e+00,  4.6786e-01],
        [ 4.2250e+00,  1.1133e+01,  1.7851e+01],
        [-7.4300e+00,  5.6515e+01,  5.5205e+01],
        [ 5.1890e+00,  3.3831e+01,  2.6926e+01],
        [-9.7550e+00,  9.8195e+01,  9.5160e+01],
        [-9.7740e+00,  8.9955e+01,  9.5531e+01],
        [ 6.1190e+00,  3.7158e+01,  3.7442e+01],
        [ 4.8910e+00,  2.0757e+01,  2.3922e+01],
        [ 7.9220e+00,  6.5631e+01,  6.2758e+01],
        [ 3.4710e+00,  5.3428e+00,  1.2048e+01],
        [-2.5700e-01,  1.6510e+00,  6.6049e-02],
        [-6.8450e+00,  5.2909e+01,  4.6854e+01],
        [-1.5560e+00,  4.7861e+00,  2.4211e+00],
        [ 7.0370e+00,  5.2636e+01,  4.9519e+01],
        [-9.4800e+00,  8.3930e+01,  8.9870e+01],
        [ 3.1710e+00,  1.6089e+01,  1.0055e+01],
        [ 4.8000e-02, -9.4067e+00,  2.3040e-03],
        [-6.1390e+00,  3.3601e+01,  3.7687e+01],
        [ 4.3790e+00,  1.6309e+01,  1.9176e+01],
        [-2.5430e+00,  1.2062e+01,  6.4668e+00],
        [ 9.9800e+00,  9.6831e+01,  9.9600e+01],
        [-2.1240e+00, -1.0806e+00,  4.5114e+00],
        [ 3.1270e+00,  2.0551e+00,  9.7781e+00],
        [ 8.0660e+00,  6.0086e+01,  6.5060e+01],
        [-9.6080e+00,  8.9419e+01,  9.2314e+01],
        [-3.8950e+00,  9.0240e+00,  1.5171e+01],
        [-5.4700e+00,  3.1723e+01,  2.9921e+01],
        [ 3.8400e+00,  2.1270e+01,  1.4746e+01],
        [-8.7680e+00,  7.7110e+01,  7.6878e+01],
        [ 6.1450e+00,  3.7973e+01,  3.7761e+01],
        [-6.4160e+00,  4.1645e+01,  4.1165e+01],
        [-6.4770e+00,  4.1891e+01,  4.1952e+01],
        [ 4.7610e+00,  1.7107e+01,  2.2667e+01],
        [ 5.2300e-01,  2.1315e+00,  2.7353e-01],
        [-1.5700e-01, -2.8324e+00,  2.4649e-02],
        [-5.4250e+00,  2.0977e+01,  2.9431e+01],
        [-9.4310e+00,  8.4326e+01,  8.8944e+01],
        [-9.1490e+00,  7.7201e+01,  8.3704e+01],
        [-3.8960e+00,  5.2988e+00,  1.5179e+01],
        [ 4.3340e+00,  1.3124e+01,  1.8784e+01]])

--- Testing Set: ---
 tensor([[ -9.5580,  81.8604,  91.3554],
        [  6.2660,  48.4128,  39.2628],
        [ -4.4880,  29.3431,  20.1421],
        [  6.5400,  41.1536,  42.7716],
        [ -3.4370,  10.9670,  11.8130],
        [  3.3360,  12.7529,  11.1289],
        [ -2.4500,   1.7215,   6.0025],
        [  4.0200,  25.0464,  16.1604],
        [  1.8580,   8.1072,   3.4522],
        [  7.3110,  56.2577,  53.4507],
        [ -7.1550,  54.2660,  51.1940],
        [ -3.7030,   7.1792,  13.7122],
        [ -3.5820,  10.7837,  12.8307],
        [ -3.2530,  12.7440,  10.5820],
        [  4.5010,  11.1350,  20.2590],
        [  4.2830,  14.2571,  18.3441],
        [ -3.0450,  16.9840,   9.2720],
        [ -9.3410,  82.4973,  87.2543],
        [  0.9800,  -2.3906,   0.9604],
        [  7.5060,  54.7270,  56.3400],
        [ -5.4660,  22.1042,  29.8772],
        [  8.8860,  73.6110,  78.9610],
        [  2.5050,  14.1680,   6.2750],
        [  5.5110,  34.9281,  30.3711],
        [ -3.1140,   4.8020,   9.6970],
        [  8.1970,  67.0458,  67.1908],
        [ -8.3960,  65.3668,  70.4928],
        [ -9.8480, 105.3471,  96.9831],
        [ -1.6180,   3.5789,   2.6179],
        [ -4.7990,  22.3844,  23.0304],
        [ -3.0580,  11.2564,   9.3514],
        [  1.0440,  -1.7331,   1.0899],
        [  4.9050,  26.0560,  24.0590],
        [ -7.0560,  53.3661,  49.7871],
        [ -8.3320,  75.6162,  69.4222],
        [ -5.1580,  25.8730,  26.6050],
        [  4.4050,  20.1090,  19.4040],
        [ -6.8430,  55.2617,  46.8266],
        [  7.7160,  66.5817,  59.5367],
        [  8.3580,  68.3002,  69.8562],
        [ -7.5820,  63.3227,  57.4867],
        [ -9.8290,  91.3742,  96.6092],
        [  1.8710,   4.0296,   3.5006],
        [ -9.6350,  89.5142,  92.8332],
        [  6.3610,  36.7063,  40.4623],
        [  8.9890,  84.3911,  80.8021],
        [ -0.7590,  -5.1369,   0.5761],
        [ -7.3250,  53.5326,  53.6556],
        [ -7.8640,  71.2075,  61.8425],
        [  5.8340,  32.0336,  34.0356]])

--- Testing data before training... ---
x          | y          | x2 NN      | x2 label   | y vs x2 NN | y vs x2 label | correct?
-----------+------------+------------+------------+------------+---------------+---------
-9.558     | 81.86      | -0.673     | 91.355     | + (y > x2) | - (y < x2)    | no
6.266      | 48.413     | -0.515     | 39.263     | + (y > x2) | + (y > x2)    | yes
-4.488     | 29.343     | -0.425     | 20.142     | + (y > x2) | + (y > x2)    | yes
6.54       | 41.154     | -0.528     | 42.772     | + (y > x2) | - (y < x2)    | no
-3.437     | 10.967     | -0.366     | 11.813     | + (y > x2) | - (y < x2)    | no
3.336      | 12.753     | -0.36      | 11.129     | + (y > x2) | + (y > x2)    | yes
-2.45      | 1.722      | -0.309     | 6.003      | + (y > x2) | - (y < x2)    | no
4.02       | 25.046     | -0.399     | 16.16      | + (y > x2) | + (y > x2)    | yes
1.858      | 8.107      | -0.274     | 3.452      | + (y > x2) | + (y > x2)    | yes
7.311      | 56.258     | -0.565     | 53.451     | + (y > x2) | + (y > x2)    | yes
-7.155     | 54.266     | -0.557     | 51.194     | + (y > x2) | + (y > x2)    | yes
-3.703     | 7.179      | -0.381     | 13.712     | + (y > x2) | - (y < x2)    | no
-3.582     | 10.784     | -0.374     | 12.831     | + (y > x2) | - (y < x2)    | no
-3.253     | 12.744     | -0.355     | 10.582     | + (y > x2) | + (y > x2)    | yes
4.501      | 11.135     | -0.425     | 20.259     | + (y > x2) | - (y < x2)    | no
4.283      | 14.257     | -0.413     | 18.344     | + (y > x2) | - (y < x2)    | no
-3.045     | 16.984     | -0.343     | 9.272      | + (y > x2) | + (y > x2)    | yes
-9.341     | 82.497     | -0.662     | 87.254     | + (y > x2) | - (y < x2)    | no
0.98       | -2.391     | -0.218     | 0.96       | - (y < x2) | - (y < x2)    | yes
7.506      | 54.727     | -0.574     | 56.34      | + (y > x2) | - (y < x2)    | no
-5.466     | 22.104     | -0.477     | 29.877     | + (y > x2) | - (y < x2)    | no
8.886      | 73.611     | -0.64      | 78.961     | + (y > x2) | - (y < x2)    | no
2.505      | 14.168     | -0.312     | 6.275      | + (y > x2) | + (y > x2)    | yes
5.511      | 34.928     | -0.479     | 30.371     | + (y > x2) | + (y > x2)    | yes
-3.114     | 4.802      | -0.347     | 9.697      | + (y > x2) | - (y < x2)    | no
8.197      | 67.046     | -0.606     | 67.191     | + (y > x2) | - (y < x2)    | no
-8.396     | 65.367     | -0.616     | 70.493     | + (y > x2) | - (y < x2)    | no
-9.848     | 105.347    | -0.687     | 96.983     | + (y > x2) | + (y > x2)    | yes
-1.618     | 3.579      | -0.259     | 2.618      | + (y > x2) | + (y > x2)    | yes
-4.799     | 22.384     | -0.442     | 23.03      | + (y > x2) | - (y < x2)    | no
-3.058     | 11.256     | -0.344     | 9.351      | + (y > x2) | + (y > x2)    | yes
1.044      | -1.733     | -0.222     | 1.09       | - (y < x2) | - (y < x2)    | yes
4.905      | 26.056     | -0.448     | 24.059     | + (y > x2) | + (y > x2)    | yes
-7.056     | 53.366     | -0.553     | 49.787     | + (y > x2) | + (y > x2)    | yes
-8.332     | 75.616     | -0.613     | 69.422     | + (y > x2) | + (y > x2)    | yes
-5.158     | 25.873     | -0.462     | 26.605     | + (y > x2) | - (y < x2)    | no
4.405      | 20.109     | -0.42      | 19.404     | + (y > x2) | + (y > x2)    | yes
-6.843     | 55.262     | -0.542     | 46.827     | + (y > x2) | + (y > x2)    | yes
7.716      | 66.582     | -0.584     | 59.537     | + (y > x2) | + (y > x2)    | yes
8.358      | 68.3       | -0.614     | 69.856     | + (y > x2) | - (y < x2)    | no
-7.582     | 63.323     | -0.577     | 57.487     | + (y > x2) | + (y > x2)    | yes
-9.829     | 91.374     | -0.686     | 96.609     | + (y > x2) | - (y < x2)    | no
1.871      | 4.03       | -0.275     | 3.501      | + (y > x2) | + (y > x2)    | yes
-9.635     | 89.514     | -0.677     | 92.833     | + (y > x2) | - (y < x2)    | no
6.361      | 36.706     | -0.52      | 40.462     | + (y > x2) | - (y < x2)    | no
8.989      | 84.391     | -0.645     | 80.802     | + (y > x2) | + (y > x2)    | yes
-0.759     | -5.137     | -0.205     | 0.576      | - (y < x2) | - (y < x2)    | yes
```

```
-7.325    | 53.533    | -0.565    | 53.656    | + (y > x2) | - (y < x2)    | no
-7.864    | 71.207    | -0.591    | 61.842    | + (y > x2) | + (y > x2)    | yes
5.834     | 32.034    | -0.495    | 34.036    | + (y > x2) | - (y < x2)    | no

# tested  : 50
# correct : 27

# of y > x2 label: 24
# of y < x2 label: 26

# of y > x2 NN: 47
# of y < x2 NN: 3

% correct guess : 54.0%
% x2 NN and label diff (median) : 101.588%
% x2 NN and label diff (average): 103.682%

--- Training now... ---
Epoch #5  loss: 2.185
Epoch #10 loss: 1.629
Epoch #15 loss: 1.382
Epoch #20 loss: 0.795
Epoch #25 loss: 0.179
Epoch #30 loss: 0.668

--- Testing data after training... ---
x         | y         | x2 NN     | x2 label  | y vs x2 NN | y vs x2 label | correct?
----------+-----------+-----------+-----------+------------+---------------+---------
-9.558    | 81.86     | 80.124    | 91.355    | + (y > x2) | - (y < x2)    | no
6.266     | 48.413    | 33.762    | 39.263    | + (y > x2) | + (y > x2)    | yes
-4.488    | 29.343    | 18.167    | 20.142    | + (y > x2) | + (y > x2)    | yes
6.54      | 41.154    | 36.8      | 42.772    | + (y > x2) | - (y < x2)    | no
-3.437    | 10.967    | 10.985    | 11.813    | - (y < x2) | - (y < x2)    | yes
3.336     | 12.753    | 10.295    | 11.129    | + (y > x2) | + (y > x2)    | yes
-2.45     | 1.722     | 6.146     | 6.003     | - (y < x2) | - (y < x2)    | yes
4.02      | 25.046    | 14.966    | 16.16     | + (y > x2) | + (y > x2)    | yes
1.858     | 8.107     | 3.43      | 3.452     | + (y > x2) | + (y > x2)    | yes
7.311     | 56.258    | 45.348    | 53.451    | + (y > x2) | + (y > x2)    | yes
-7.155    | 54.266    | 43.618    | 51.194    | + (y > x2) | + (y > x2)    | yes
-3.703    | 7.179     | 12.801    | 13.712    | - (y < x2) | - (y < x2)    | yes
-3.582    | 10.784    | 11.975    | 12.831    | - (y < x2) | - (y < x2)    | yes
-3.253    | 12.744    | 9.728     | 10.582    | + (y > x2) | + (y > x2)    | yes
4.501     | 11.135    | 18.257    | 20.259    | - (y < x2) | - (y < x2)    | yes
4.283     | 14.257    | 16.762    | 18.344    | - (y < x2) | - (y < x2)    | yes
-3.045    | 16.984    | 8.394     | 9.272     | + (y > x2) | + (y > x2)    | yes
-9.341    | 82.497    | 76.693    | 87.254    | + (y > x2) | - (y < x2)    | no
0.98      | -2.391    | 1.033     | 0.96      | - (y < x2) | - (y < x2)    | yes
7.506     | 54.727    | 47.678    | 56.34     | + (y > x2) | - (y < x2)    | no
-5.466    | 22.104    | 24.989    | 29.877    | - (y < x2) | - (y < x2)    | yes
8.886     | 73.611    | 69.498    | 78.961    | + (y > x2) | - (y < x2)    | no
2.505     | 14.168    | 6.351     | 6.275     | + (y > x2) | + (y > x2)    | yes
5.511     | 34.928    | 25.476    | 30.371    | + (y > x2) | + (y > x2)    | yes
-3.114    | 4.802     | 8.779     | 9.697     | - (y < x2) | - (y < x2)    | yes
8.197     | 67.046    | 58.604    | 67.191    | + (y > x2) | - (y < x2)    | no
-8.396    | 65.367    | 61.751    | 70.493    | + (y > x2) | - (y < x2)    | no
-9.848    | 105.347   | 84.709    | 96.983    | + (y > x2) | + (y > x2)    | yes
-1.618    | 3.579     | 2.736     | 2.618     | + (y > x2) | + (y > x2)    | yes
-4.799    | 22.384    | 20.301    | 23.03     | + (y > x2) | - (y < x2)    | no
-3.058    | 11.256    | 8.443     | 9.351     | + (y > x2) | + (y > x2)    | yes
1.044     | -1.733    | 1.204     | 1.09      | - (y < x2) | - (y < x2)    | yes
4.905     | 26.056    | 21.028    | 24.059    | + (y > x2) | + (y > x2)    | yes
-7.056    | 53.366    | 42.52     | 49.787    | + (y > x2) | + (y > x2)    | yes
-8.332    | 75.616    | 60.739    | 69.422    | + (y > x2) | + (y > x2)    | yes
-5.158    | 25.873    | 22.764    | 26.605    | + (y > x2) | - (y < x2)    | no
4.405     | 20.109    | 17.598    | 19.404    | + (y > x2) | + (y > x2)    | yes
-6.843    | 55.262    | 40.159    | 46.827    | + (y > x2) | + (y > x2)    | yes
7.716     | 66.582    | 50.999    | 59.537    | + (y > x2) | + (y > x2)    | yes
8.358     | 68.3      | 61.15     | 69.856    | + (y > x2) | - (y < x2)    | no
-7.582    | 63.323    | 48.88     | 57.487    | + (y > x2) | + (y > x2)    | yes
-9.829    | 91.374    | 84.409    | 96.609    | + (y > x2) | - (y < x2)    | no
1.871     | 4.03      | 3.479     | 3.501     | + (y > x2) | + (y > x2)    | yes
-9.635    | 89.514    | 81.341    | 92.833    | + (y > x2) | - (y < x2)    | no
```

```
6.361     | 36.706    | 34.815    | 40.462    | + (y > x2) | - (y < x2)    | no
8.989     | 84.391    | 71.127    | 80.802    | + (y > x2) | + (y > x2)    | yes
-0.759    | -5.137    | 0.512     | 0.576     | - (y < x2) | - (y < x2)    | yes
-7.325    | 53.533    | 45.503    | 53.656    | + (y > x2) | - (y < x2)    | no
-7.864    | 71.207    | 53.339    | 61.842    | + (y > x2) | + (y > x2)    | yes
5.834     | 32.034    | 28.972    | 34.036    | + (y > x2) | - (y < x2)    | no

# tested  : 50
# correct : 35

# of y > x2 label: 24
# of y < x2 label: 26

# of y > x2 NN: 39
# of y < x2 NN: 11

% correct guess : 70.0%
% x2 NN and label diff (median) : 12.199%
% x2 NN and label diff (average): 10.896%

--- Plotting test results before and after... ---
```