Lab 7 Report: Tic-Tac Neural Network v2

Stuart Harley & Valerie Djohan

Introduction:

This week's lab is a pair-programming lab and the main intention was to make a personalized neural network that has the ability to recognize a Tic-Tac-Toe board that incorporates "blanks" as an input from a modified .csv file. The student pairs were expected to find and propose a solution to modify the implemented network structure from Lab 6 to the instructor and implement the proposed solution as well as documenting the overall process.

Strategy Proposal:

As a proposal, we proposed a way in changing the load_tictactoe_csv(filepath) method by adding an additional number like 2 or -1 to represent the blanks and try to utilize a different activation function (ReLU) to train the network.

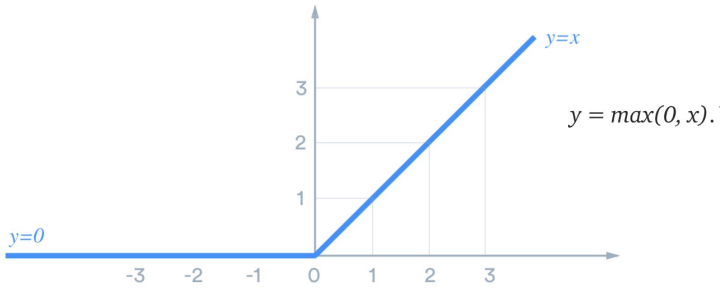

Figure 1: ReLU Activation Function

ReLU is a very common activation function that is used in CNNs and we thought it would be a good choice since ReLU does not have a vanishing gradient problem like the sigmoid function and it is likely to converge faster.

Development Process:

During the initial development process, we decided to incorporate 2 to our tic-tac-toe board in order to represent the blanks and do a standard print() to see whether it read the file correctly. Then, we tried to implement ReLU and it's derivative and compare the losses with the nn.loss() that was present in the test_nn.py. However, after using ReLU, somehow the losses that we received on every test and every training set appeared to be constant. Our instructor suggested adding a constant to the initial weights because ReLU's nature might cause the overall loss to be stuck when it's very close to zero. Unfortunately, we were not able to determine a constant to make our losses converge even after several experiments with different constants. Shortly, we decided to change our activation function from ReLU to numpy's tanh function and implement its derivative to see whether the losses could converge better than ReLU since tanh is able to map the values from -1 to 1.

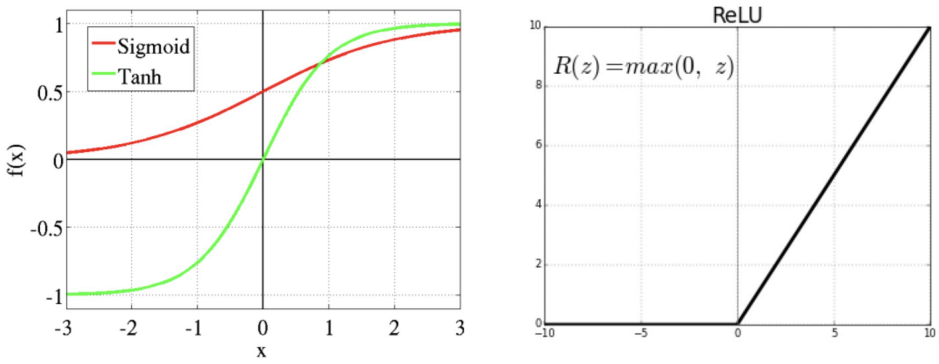

Figure 2: Tanh Activation Function Compared to Sigmoid and ReLU

Network Structure

The structure of our Neural Network does not differ much other than the fact that we used the tanh function during our feedforward and it's derivative in the backpropagation phase. Here below is the representation picture that we took from Lab 6 Description:

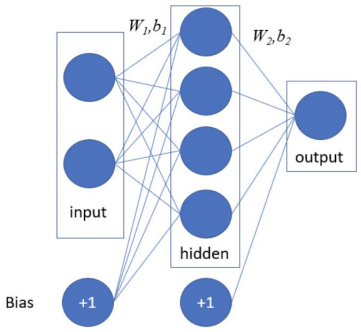

Figure 3: Neural Network Structure based on Lab 6

Training Approaches and Its Results:

We had several training approaches that we tried to use in this lab.

First Experiment: Entire Dataset

The first one was to use the whole dataset for the nn to train itself and evaluate the results. With this approach, we did lots of hyperparameter tuning in order to increase the accuracies. Here below are some sample experiments of the different hyperparameters that we used to test our implementation:

Learning Rate	**# Hidden Nodes**	**Epochs**	**Accuracy (%)**	**Loss**
0.0003	10	100	66.5%	0.335

0.0002	10	1000	60.6%	0.335
0.0002	15	10,000	66.45%	0.335
0.0003	9	10,000	77.1%	0.0996
0.001	10	10,000	66.454%	0.3545
0.0005	10	10,000	89.012%	0.0833
0.0007	10	10,000	89.56%	0.906
0.0007	11	10,000	86.73%	0.117
0.0007	12	10,000	88.03%	0.0744
0.00066	10	10,000	89.0%	0.0988
0.00066	11	100,000	96.9%	0.0674
0.00066	11	100,000	98.7% (bias=0.5)	0.00345
0.00066	12	200,000	95.5%	0.0514

Table 1: Training Experiment 1 Results

The observation that we were able to make was the fact that the accuracy would improve as the Epochs are increased, but the accuracy will also depend on the number of hidden nodes and also the learning rate that is tuned for the neural network. However, we assumed that the losses are dependent on the randomized weights since sometimes, the end loss result could display a very low loss and sometimes it could display a very high loss (higher than 0.1).

We also tried to find some parts where the network mis-classifies the output:

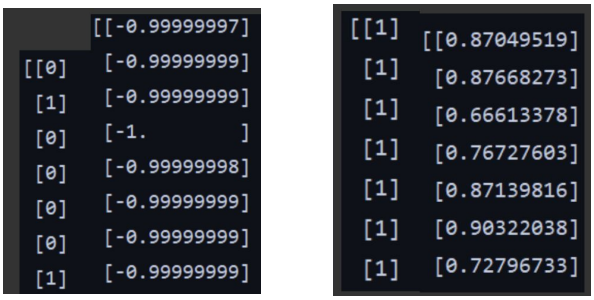

Figure 4: Mis-Classification of Inputs. In both Images,
the left array is the predicted values and the right array is the output

This mis-classification was initially caused by the shuffle in the dataset. In the first image, we accidentally shuffled both the predicted and the board values that were read from the .csv files which

messed the value mappings, while in the second image, we shuffled only the board that was read from the .csv file.

The accuracy results in Table 1 can be compared with Lab 6 accuracies. The results will be compared using the default parameters:

Lab 6 Accuracy	Lab 7 Accuracy	Lab 6 Loss	Lab 7 Loss
1.0	0.5313	0.0013287	0.469
1.0	0.513	5.195 * 10^-5	0.468
1.0	0.665	0.0002887	0.335
1.0	0.335	2.1952 * 10^-5	2.994
1.0	0.664	6.0089 * 10^-5	0.335

Table 2: Accuracy and Loss Comparison with Lab 6 Results

From the results above, the accuracy of the network trained without the blanks within the tic-tac-toe.csv and tic-tac-toeFull.csv is far higher than the ones with the blank values. It was mentioned in Lab 6 that the blanks made the end-of-game configurations increase from 78 to 958 possibilities. The increase in game configuration could be a reason why the accuracies went down by a huge factor in Lab 7. Additionally, the low accuracy might also be caused by the different activation functions that were used in both labs, and since tanh has a higher value scope, additional hyperparameter tuning was needed in order to fit the neural network with tanh nature and also to increase the accuracy.

Second Experiment: Split Dataset

We split the dataset into a training set and a validation set. We split this randomly to try to avoid either set being biased in some way. We made this a 75/25 split respectfully.We decided to go with this ratio after researching common data set splits. Generally they seemed to be somewhere between 60/20/20 and 80/10/10 (training/validation/testing). We did not need both a validation and testing set, so we felt a 75/25 split was about in the middle of the common splits and would fit our needs.

After splitting the data we trained the neural network using the best hyperparameters that we found from the first experiment. (lr = .00066, 11 hidden nodes, 100,000 epochs). We split the dataset several times and tested it to get values for loss and accuracy. We found that the values of the network trained on the training set to be basically the same as the network trained on the entire data set. (~97% accurate and ~.07 loss).

Then we tested the resulting networks against the validation sets. We found that there was a wider range of accuracies when testing against the validation sets. The lowest accuracy we found was about 81% and the highest was about 94%. And we got accuracies of everything in between. So it would appear that all

of the networks had some amount of overfitting, but some had much more than others. So, some of the random splits created training sets that better trained the network than others. Therefore, in order to build the best network, it would be necessary to run these tests again and analyze the data sets that produced the best results, and then build our own custom training dataset based on the results we found. This process is similar to what is done in the real world, where data sets are built very carefully and deliberately so that the network trained using them can get the best results.

Conclusion:

It is possible to build a pretty good neural network that identifies the winner of a tic-tac-toe board. If you create a good training set it is possible to achieve accuracies of about 95% using our network structure. And it would likely be able to achieve better results if you added an extra layer to the network, because it could probably identify more patterns. However, a neural network is not the best way to determine the winner of a tic-tac-toe game. Because the end game board is very simple, it would be much easier to look at each possible row, column, and diagonal individually (only 8 in total) to determine if there are 3 of the same entry in any of them. This method would have an accuracy of 100%, it would be easier to build, you wouldn't have to train it, and it would still determine a winner very quickly.

Bibliography

Lie, D. (2017, November 17). *A Practical Guide to ReLU: Start using and understanding ReLU without BS or fancy equations* . Retrieved from Medium.com: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

Sharma, S. (2017, November 6). *Activation Functions in Neural Networks: Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED !!!* . Retrieved from Medium.com: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6