CODE SUMMARY:

I am building a **Single Layer Feed Forward Neural Network.** The goal is to research whether Winsorizing can improve the accuracy or computational effort of a single-node classification algorithm. File **test_data.py** uses normal data and **test_winsorized_data.py** uses Winsorized data.

Titanic dataset has been used for testing. Source: https://www.kaggle.com/c/titanic/data.
I have removed fields with non-integer or non-float values. The input fields are ticket class, Sex, Age, # of siblings or spouses aboard the ship, # of parents or children aboard the ship, passenger fare. The output is passenger probability of surviving.

I am using tanh as my Activation function. The pre-defined learning rate is set to 0.001 and regularization strength set to 0.01.

METHOD DESCRIPTION:

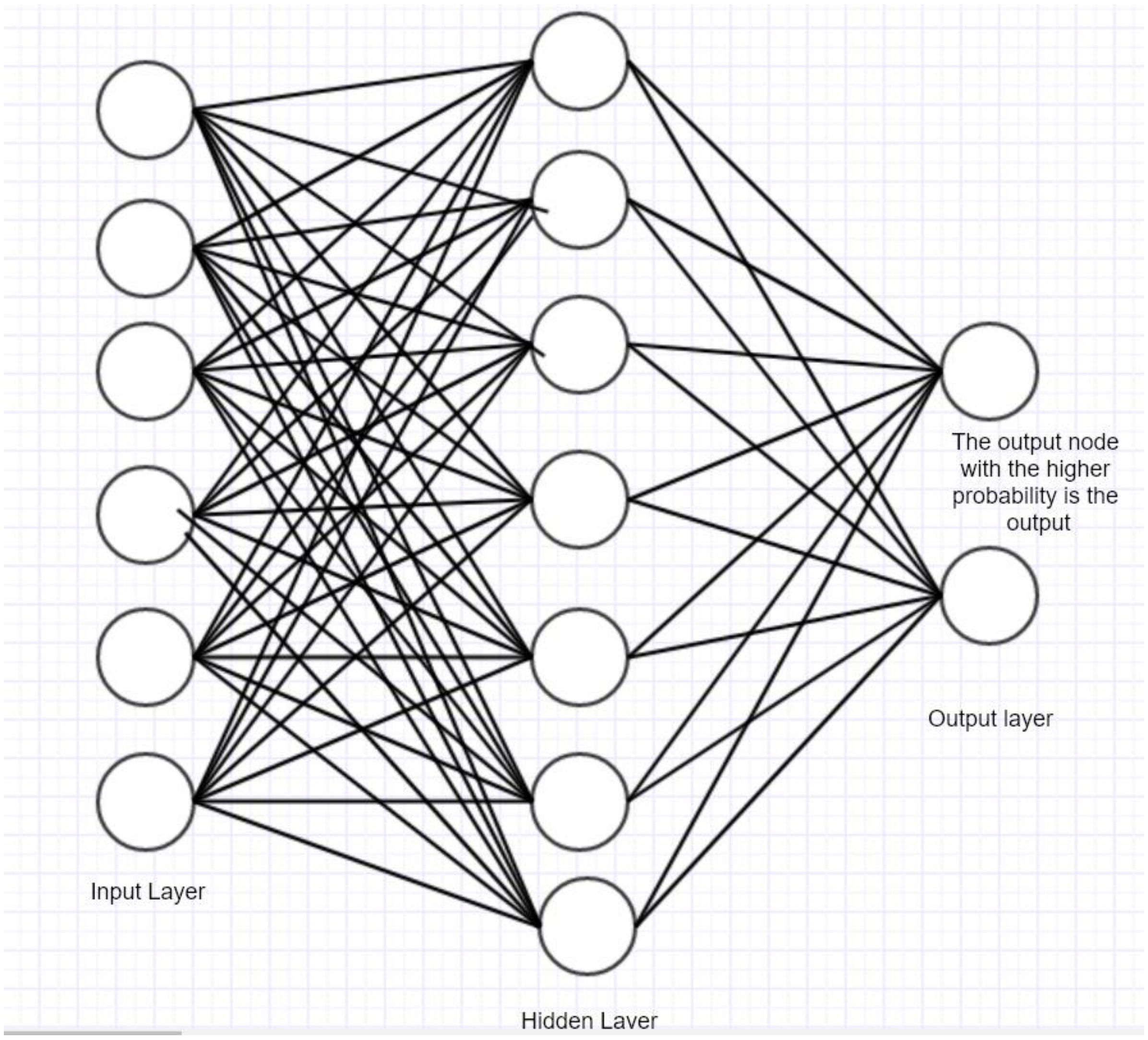
- **calculate_loss**: Helper function to evaluate total loss of the dataset.
- **predict**: Helper function for testing new data
- **build_model**: This function learns the parameter for the neural network and returns the model. No. of iterations are set to 20000 and no. of nodes in hidden layer are 7.
- **generate_data**: This function extracts data from train.csv without Winsorization.
- **winsorized_data**: This function extracts data and Winsorizes it by a factor of 0.2
- **get_accuracy**: This function finds prediction accuracy when test our model

RESULT ANALYSIS:

- If we look at the result in Output.txt we will find we can see the error loss value decrease after every iteration and is finally set to 0.499. This means our training is successful and for testing our network we use new data. Computation time was noted to be 4 seconds and ACC was found to be 80%!
- While testing the winsorized data though, the error loss value is now even smaller: 0.417. While testing our network we note an increased accuracy in our prediction, ACC = 85%. But an increased computational effort was needed. Computational time was noted to be 24 seconds!

FUTURE WORK:

The effect of winsorization was noted to be negligible while testing some other datasets. I intend to work more on this and increase the number of layers and maybe try some other activation function (ex: Relu)



This diagram was drawn using Gliffy. As we can see the input layer has 6 nodes and hidden layer has 7. We are then finding the probability of a point in the dataset belonging to a category/class and then classifying it depending on which output node has the max value.

Training (test_data.py):

- At Line 72 we are doing forward propagation where we take the dot product of the weights and input values and then use our activation function on this value.
- At Line 79 we are doing back propagation to update our weights and this guides us to a low error value and improves our accuracy