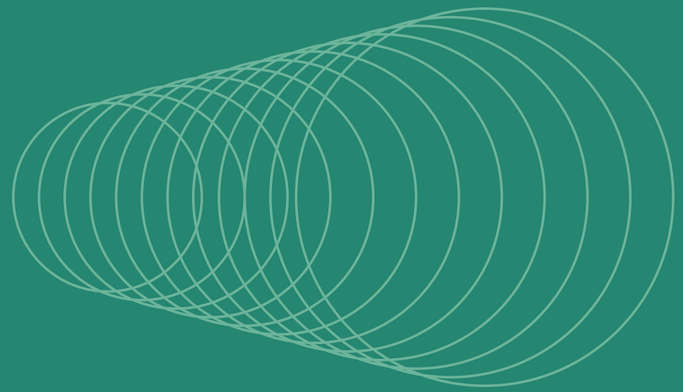
# CE889 Individual Project

Name : Hassan Moin
Reg No: 2201717

# Data Details



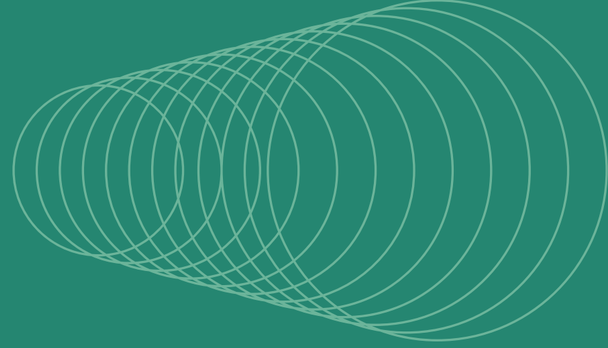
## Collection of data:

- Playing the game in a way that half of the wins is on left and half is on right
- Collected atmost 7000 rows
- Data have four columns with data type float

## Partition of data:

- Training split 70%
- Validation split 30%

# Data Details Continue



## Processing of data:

- Removed duplication in row for the better model fit
- Replaced null values with the mean of column
- Normalized the  data in order to train feature on similar scale

# Network Architectre

## Neuron Class:

```
class neuron_struct:
    def __init__(self, layer_position, activation_value, no_of_weights):
        self.activation = activation_value
        self.d_weights = []
        self.gradiant = 0
        self.weights = []
        self.initializing_random_weights(no_of_weights)
        self.position = layer_position  # setting neuron index

    def initializing_random_weights(self, no_of_weights):
        #Random weights to network
        for i in range(0, no_of_weights):
            self.weights.append(random.random())
            self.d_weights.append(0)

    #Calculations of weights
    def weigths_calculation(self, layer):
        total_sum = 0
        for i in range(0, len(layer)):
            total_sum = total_sum + (float(layer[i].activation) * float(layer[i].weights[self.position]))
        self.sigmoid(total_sum)

    #Activation Function
    def sigmoid(self, x):
        self.activation = 1 / (1 + math.exp(-(nn_lambda * x )))
```

# Network Architectre Continue

## Network Layers:

```
#Defining Neural Network layers
#input layer
input = []
#hidden layer
hidden = []
#output layer
output = []
breakpoint = 0.0
global de_normalize
de_normalize = False

#Initializing layers with neurons
for i in range(0 , input_neurons+1):
  input.append(neuron_struct(i,0,2))

for i in range(0 , hidden_neurons+1):
  hidden.append(neuron_struct(i,0,2))

for i in range(0 , output_neurons):
  output.append(neuron_struct(i,0,0))
```

## Network Operations:

```
#function to normalize the game input row
def normalization(inputs):···

#function to denormalize the precited output
def de_normalization(outputs):···

#Error calculation on training file
def error_epoch():···

#Error calculation on validation file
def error_validate():  ···

def feed_forward(input_row):···

def back_propogation(outputs_network):···

#Training Function
def training():···

#Function to save weights in txt file
def weight_save():···

#fetch weights from txt file and initialize weights into layers
def weight_fetch_and_initialize(row):···

#Function to do prediction against game input
def predictions(row):···
```

# Network Architectre Continue

## Hyper Prams & Enums:

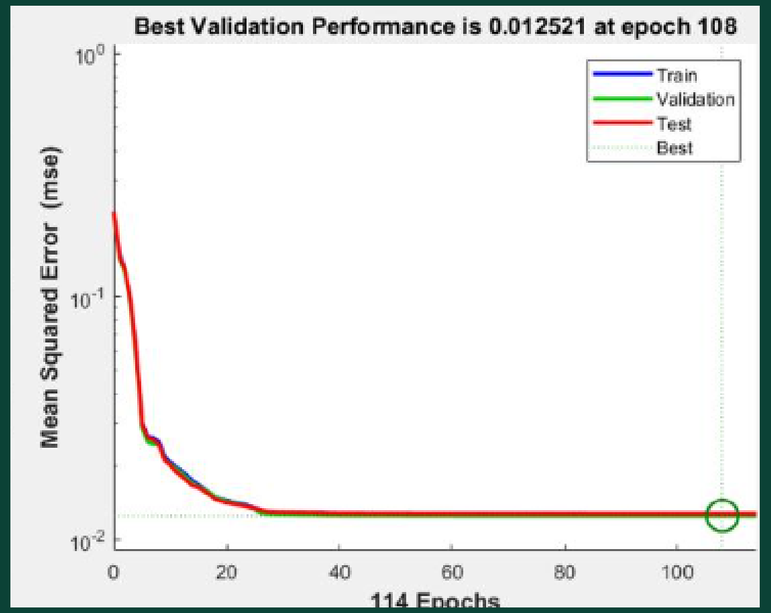
NN_hyperparams.py > ...

```
nn_lambda = 0.8
nn_lr = 0.8
nn_momentum = 0.5
hidden_neurons = 2
output_neurons = 2
input_neurons = 2
x1_max = 556.311
x2_max = 558.184
y1_max = 7.634
y2_max = 3.105
x1_min = -539.680
x2_min = 65.201
y1_min = -3.624
y2_min = -3.030
iterations = 2000
training_flag = 0
```

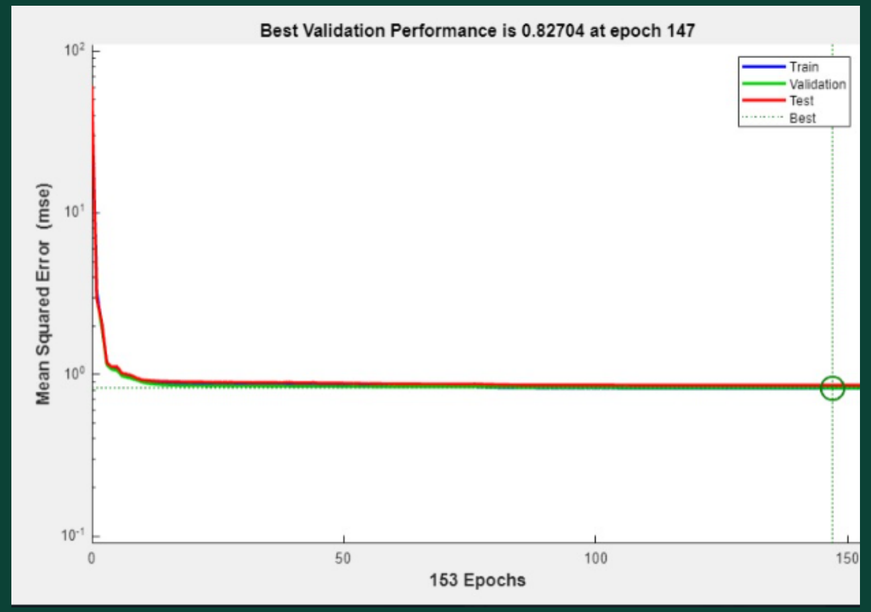
## Integration with game:

```
NetHolder.py > ...
rom NN_layers_networks import predictions
lass NeuralNetHolder:

    def __init__(self):
        super().__init__()


    def predict(self, input_row):
        input = input_row.split(",")
        output = predictions([float(input[0]), float(i
        return output
```

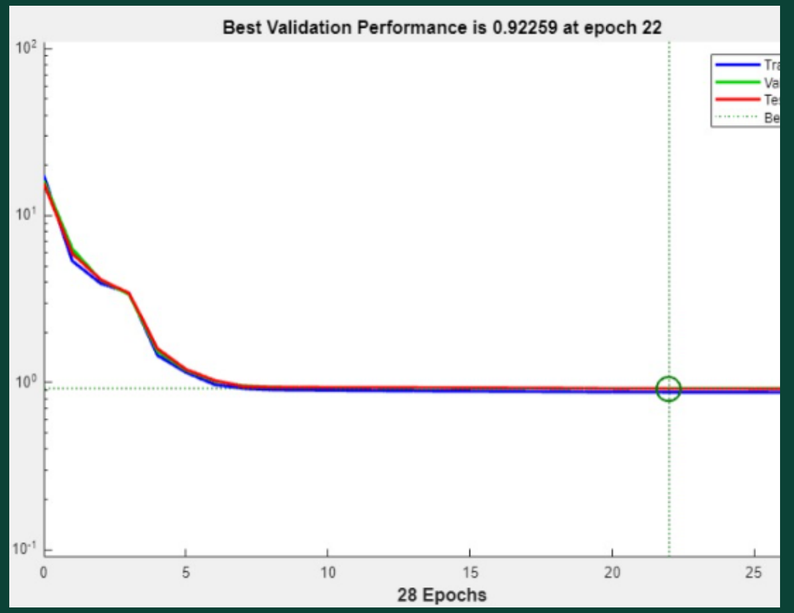
# Hyperparameters




Hidden Neuron = 2

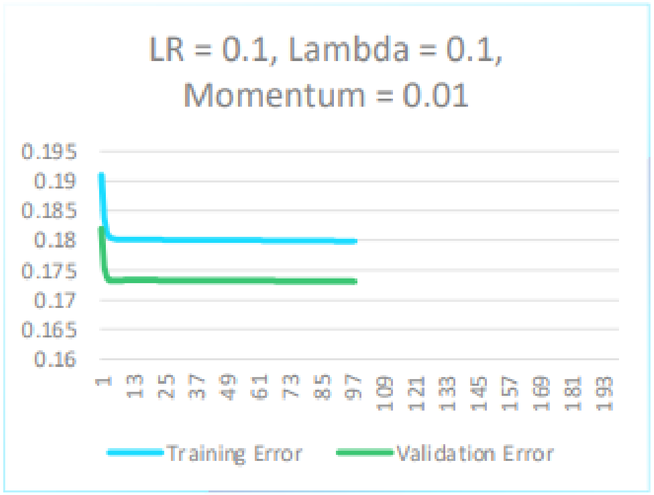


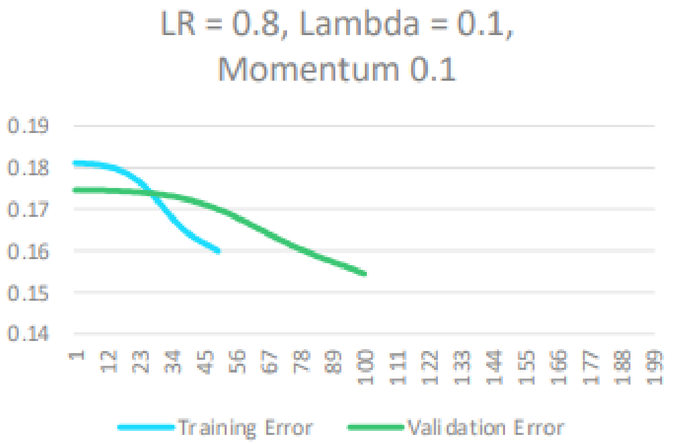

Hidden Neuron = 4

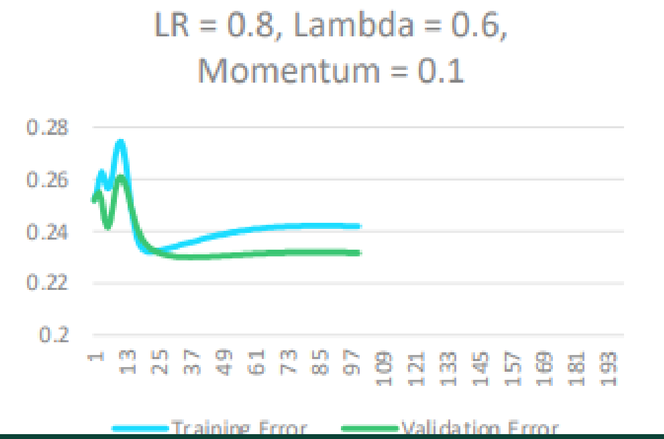


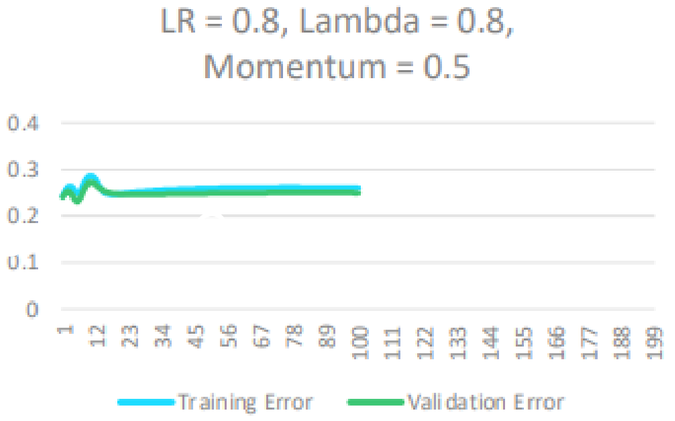

Hidden Neuron = 6

# Hyperparameters



# RMSE

## RMSE on training and testing data

```python
#Error calculation on training file
def error_epoch():
  error_epoch = []
  with open('normalized_training.csv', 'r') as training:
    data = reader(training)
    for row in data:
      feed_forward([row[0], row[1], 1])
      error_epoch.append( ( (float(row[2]) - float(output[0].activation))**2
                          (float(row[3]) - float(output[1].activation))**2
    return math.sqrt(sum(error_epoch) / len(error_epoch))

#Error calculation on validation file
def error_validate():
  total_error = []
  with open('normalized_testing.csv', 'r') as validating:
    data = reader(validating)
    for row in data:
      feed_forward([row[0], row[1], 1])
      total_error.append( ( (float(row[2]) - float(output[0].activation))**2
                          (float(row[3]) - float(output[1].activation))**2
    return math.sqrt(sum(total_error) / len(total_error))
```

```
-----------------------------------
-----------------------------------
Epoch :  1  Training Error :  0.2482139458668065  Validation Error :  0.2582418532021857
-----------------------------------
Epoch :  2  Training Error :  0.23773049661816884  Validation Error :  0.24564224221589132
-----------------------------------
Epoch :  3  Training Error :  0.23874388581502004  Validation Error :  0.24975573855220098
```

# Stopping Criteria

## Condition to stop training:

- Check for previous and current error till 7 decimal points and stop if no significant changes.

```
for i in range(iterations):
    training()
    training_error = error_epoch()
    #Stopping condition for training
    if float("{:.7f}".format(training_error)) == float("{:.7f}".format(breakpoint)):
        print("Training stopped due to defined stopping criteria")
        break
    else:
        breakpoint = training_error
    validation_error = error_validate()
```

- Stopped on number of epochs

# Thank you!