Neuron
Network

Rocket Game

Data Partition

ce889_dataCollection

	X axis	Y Axis	Vel-X	Vel-Y
	A	B	C	D
1	617.0829	345.5	0.2	0
2	617.0829	345.3	0.100487	0.053951
3	617.029	345.1995	0.001583	0.034857
4	616.9941	345.1979	-0.09647	0.022692
5	616.9714	345.2944	-0.19343	0.017421
6	616.954	345.4878	-0.28906	0.019004
7	616.935	345.7769	-0.38312	0.027388
8	616.9076	346.16	-0.47537	0.042515
9	616.8651	346.6354	-0.56558	0.064319
10	616.8008	347.201	-0.65352	0.092723
11	616.708	347.8545	-0.73896	0.127644
12	616.5804	348.5935	-0.82167	0.168992
13	616.4114	349.4151	-0.90143	0.216666
14	616.1947	350.3166	-0.97802	0.27056
15	615.9242	351.2946	-1.05122	0.33056
16	615.5936	352.3458	-1.12083	0.396544
17	615.1971	353.4666	-1.19044	0.462528
18	614.7345	354.6571	-1.26005	0.528512
19	614.206	355.9171	-1.32966	0.594495

```
#Importing data and splitting them
df = pd.read_csv('ce889_dataCollection(baha).csv') #Collecting data from the CSV file
x=df.iloc[0:,:1] #this takes the coloumn i want ( remember [from which row : which row
y=df.iloc[0:,1:2]
vx=df.iloc[0:,2:3]
vy=df.iloc[0:,3:4]
```

Data Partition

ce889_dataCollection

	X axis	Y Axis	Vel-X	Vel-Y
	A	B	C	D
1	617.0829	345.5	0.2	0
2	617.0829	345.3	0.100487	0.053951
3	617.029	345.1995	0.001583	0.034857
4	616.9941	345.1979	-0.09647	0.022692
5	616.9714	345.2944	-0.19343	0.017421
6	616.954	345.4878	-0.28906	0.019004
7	616.935	345.7769	-0.38312	0.027388
8	616.9076	346.16	-0.47537	0.042515
9	616.8651	346.6354	-0.56558	0.064319
10	616.8008	347.201	-0.65352	0.092723
11	616.708	347.8545	-0.73896	0.127644
12				
13				
14				
15	615.9242	351.2946	-1.05122	0.33056
16	615.5936	352.3458	-1.12083	0.396544
17	615.1971	353.4666	-1.19044	0.462528
18	614.7345	354.6571	-1.26005	0.528512
19	614.206	355.9171	-1.32966	0.594495

```
#Importing data and splitting them
df = pd.read_csv('ce889_dataCollection(baha).csv') #Collecting data from the CSV file
x=df.iloc[0:,:1] #this takes the coloumn i want ( remember [from which row : which row
y=df.iloc[0:,1:2]
vx=df.iloc[0:,2:3]
vy=df.iloc[0:,3:4]
```

Splitting the Data

```
                    #it could've been done that both input merged and decrease the lines of code but i used that way to debug easier
# Split data into training and testing sets
X_train, X_test,y_train, y_test, vx_train, vx_test, vy_train, vy_test = train_test_split( *arrays: x,y,vx,vy, test_size=0.2, random_state=7)
```

Data Processing

Data Processing

1

```
def detect_outliers(df, threshold=3):  1 usage
    z_scores = np.abs(zscore(df))
    return (z_scores > threshold).any(axis=1)
```

Data Processing

1

```python
def detect_outliers(df, threshold=3):  1 usage
    z_scores = np.abs(zscore(df))
    return (z_scores > threshold).any(axis=1)
```

2

```python
def scaling(column):
    return (column - column.min()) / (column.max() - column.min())
```

Data Processing

1

```
def detect_outliers(df, threshold=3):  1 usage
    z_scores = np.abs(zscore(df))
    return (z_scores > threshold).any(axis=1)
```

2

```
def scaling(column):
    return (column - column.min()) / (column.max() - column.min())
```

3

```
x_t=scaling(X_train).to_numpy()
x_v=scaling(X_test).to_numpy()
y_t=scaling(y_train).to_numpy()
y_v=scaling(y_test).to_numpy()
```

Architecture Design

```
inputs = np.column_stack((x_t, y_t, np.ones(x_t.shape[0])))
# #so shape[0] will be the number of samples that we will u
actual_outputs = np.column_stack((vx_t, vy_t))
```

```
n_inputs = inputs.shape[1]
n_hidden_neurons = 15 #according to
n_outputs = actual_outputs.shape[1]
```

Hyper-Parameters

Using Matlab

Hidden_neurons	alfa	momentum	Error after 1000E
3	0.5	0.2	0.028122
4			0.027984
6			0.021353
10			0.017237
20			0.018802
15			0.016487
14			0.017177
16			0.016936
15	0.5	0.2	0.016487
17			0.01799
15	0.8		0.015767
15	0.7		0.015936
15	0.9		0.015633
15	1		0.015521
15		0.6	0.015536
15		0.8	0.015537
15		0.4	0.015533
		0.3	0.01553
		0.25	0.015527
		0.15	0.015498
15		0.13	0.015452
15	1	0.14	0.015449

Hyper-Parameters

Using Grid-Search

```python
hyperparameters = {
    "learning_rate": np.linspace( start: 0.01,  stop: 0.5,  num: 10),  # Learning rate values between 0.01 and 0.5
    "momentum": np.linspace( start: 0.1,  stop: 0.9,  num: 5),  # Momentum values between 0.1 and 0.9
    "hidden_neurons": [10,12,13, 15,17, 20]  # Different numbers of neurons in the hidden layer
}

# Grid search function
def grid_search(grid, inputs, actual_outputs, inputs_val, outputs_val):  1 usage
    param_name = list(grid.keys())
    param_values = list(grid.values())
    best_params = None
    best_score = float('inf')  # Start with a big number and go down as we find a better score

    # Iterate through all combinations of hyperparameters
    for values in product(*param_values):
        hyperparams = dict(zip(param_name, values))
        print(f"Testing hyperparameters: {hyperparams}")

        # Train the model with current hyperparameters
        score = train_fn(train_neural_network, inputs, actual_outputs, inputs_val, outputs_val, **hyperparams)
        print(f"Validation score: {score}")

        # Update the best parameters if the current score is better
        if score < best_score:
            best_score = score
            best_params = hyperparams

    return best_params, best_score
```

Hyper-Parameters

Using Grid-Search

```
Epoch 36/40 | Total Error: 0.09230484508492581
Epoch 37/40 | Total Error: 0.09225759258232494
Epoch 38/40 | Total Error: 0.0922250612627867
Epoch 39/40 | Total Error: 0.092201188536776
Epoch 40/40 | Total Error: 0.09218282231439513
Validation score: 0.38449157848680204
Best Hyperparameters: {'learning_rate': np.float64(0.01), 'momentum': np.float64(0.1), 'hidden_neurons': 10}
Best Validation Score: 0.3580784796725381

Process finished with exit code 0
```

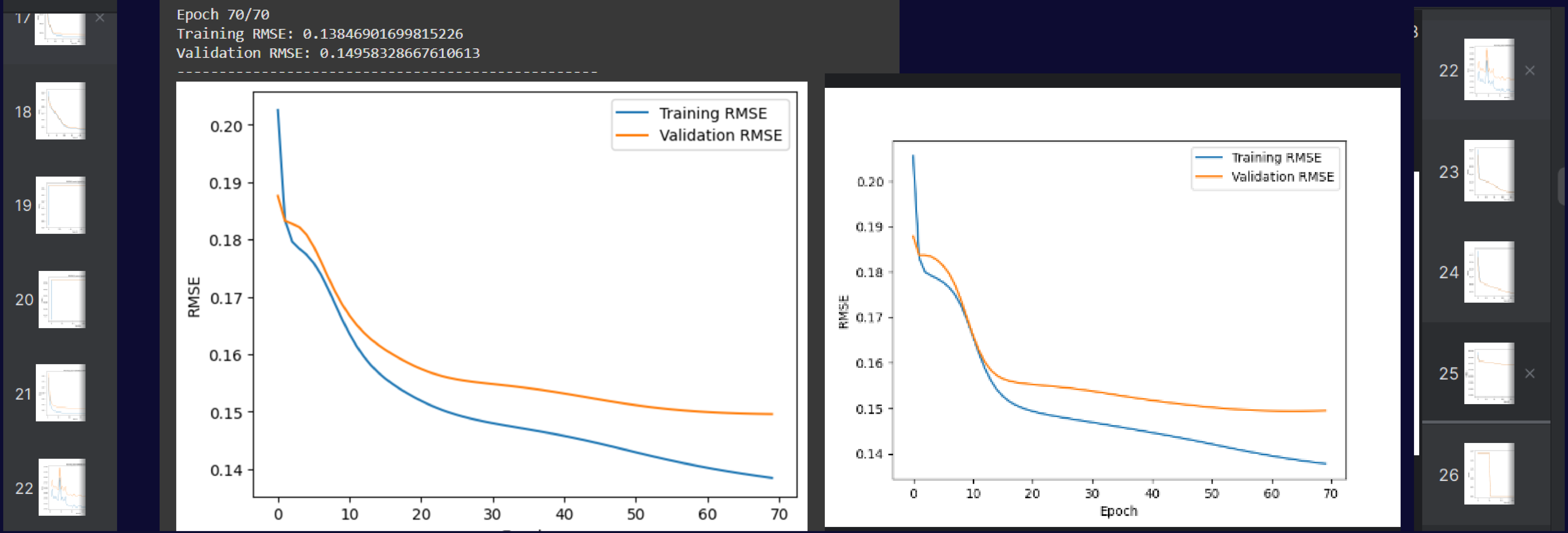
Hyper-Parameters

Using Grid-Search Library

```
"C:\Program Files\Python312\python.exe" "C:\Users\legion\OneDrive - University of Essex\Masters\Neural Networks\Lab4\Erasable.py"
Starting grid search...
Fitting 2 folds for each of 210 candidates, totalling 420 fits
Best parameters: {'batch_size': 16, 'hidden_layer_sizes': (16,), 'learning_rate_init': 0.1, 'max_iter': 100, 'momentum': 0.3}
Best RMSE: 0.1156
Test RMSE: 0.1276

Process finished with exit code 0
```

Hyper-Parameters

Using Trial and Error



Feed Forward

```python
class neuron:
    def __init__(self, inps, ws):
        self.inps = np.array(inps)
        self.ws = np.array(ws)
        self.vs = np.dot(self.inps, self.ws)  #Weighted sum by multiplaying inps by weights and then it will go to activation function
        # init some values to be filled later using the eqns
        self.hiddens = np.zeros(self.ws.shape[1])  #hidden neurons
        self.outputs = np.zeros(self.ws.shape[1])  #output neurons
        self.es = np.zeros(len(actual_outputs))  #error array
        self.gd_ys = np.zeros(self.ws.shape[1])  #gradient decent of output
        self.gd_hs = np.zeros(self.ws.shape[1])  #gradient decent of hiddden

    def activation(self, layer):
        #Sigmoid activation function      that if h is given it returns hidden layer (that is used so if i have more than 1 actv fn)
        if layer == 'h':
            self.hiddens = 1 / (1 + np.exp(-0.99*self.vs))
            return self.hiddens
        else:
            self.outputs = 1 / (1 + np.exp(-0.99*self.vs))
```

```python
for i in range(inputs.shape[0]):
    #1:Forward Pass Hidden Layer
    hidden_neuron = neuron(inputs[i], wsh)
    hidden_outputs = hidden_neuron.activation('h')

    #2:Output Layer
    output_neuron = neuron(np.append(hidden_outputs, values: 1), wsy) #with bais
    #output_neuron = neuron(hidden_outputs, wsy) #without bais
    output_values = output_neuron.activation('o')
```

Error Calculation

```
#calculating error compared to actual outputs given from the CSV file
def error_calc(self, actual_outputs):
    self.es = np.array(actual_outputs)  - self.outputs
    return self.es
```

```
#3:Error calc
error = output_neuron.error_calc(actual_outputs[i])
sqr_error = np.square(error)
total_error += np.mean(sqr_error)
```

Back Propagation

```python
class neuron:
    def __init__(self, inps, ws):
        self.inps = np.array(inps)
        self.ws = np.array(ws)
        self.vs = np.dot(self.inps, self.ws)  #Weighted sum by multiplaying inps by weights and then it will go to activation function
        # init some values to be filled later using the eqns
        self.hiddens = np.zeros(self.ws.shape[1])  #hidden neurons
        self.outputs = np.zeros(self.ws.shape[1])  #output neurons
        self.es = np.zeros(len(actual_outputs))  #error array
        self.gd_ys = np.zeros(self.ws.shape[1])  #gradient decent of output
        self.gd_hs = np.zeros(self.ws.shape[1])  #gradient decent of hiddden

    def activation(self, layer):
        #Sigmoid activation function      that if h is given it returns hidden layer (that is used so if i have more than 1 actv fn)
        if layer == 'h':
            self.hiddens = 1 / (1 + np.exp(-0.99*self.vs))
            return self.hiddens
        else:
            self.outputs = 1 / (1 + np.exp(-0.99*self.vs))
```

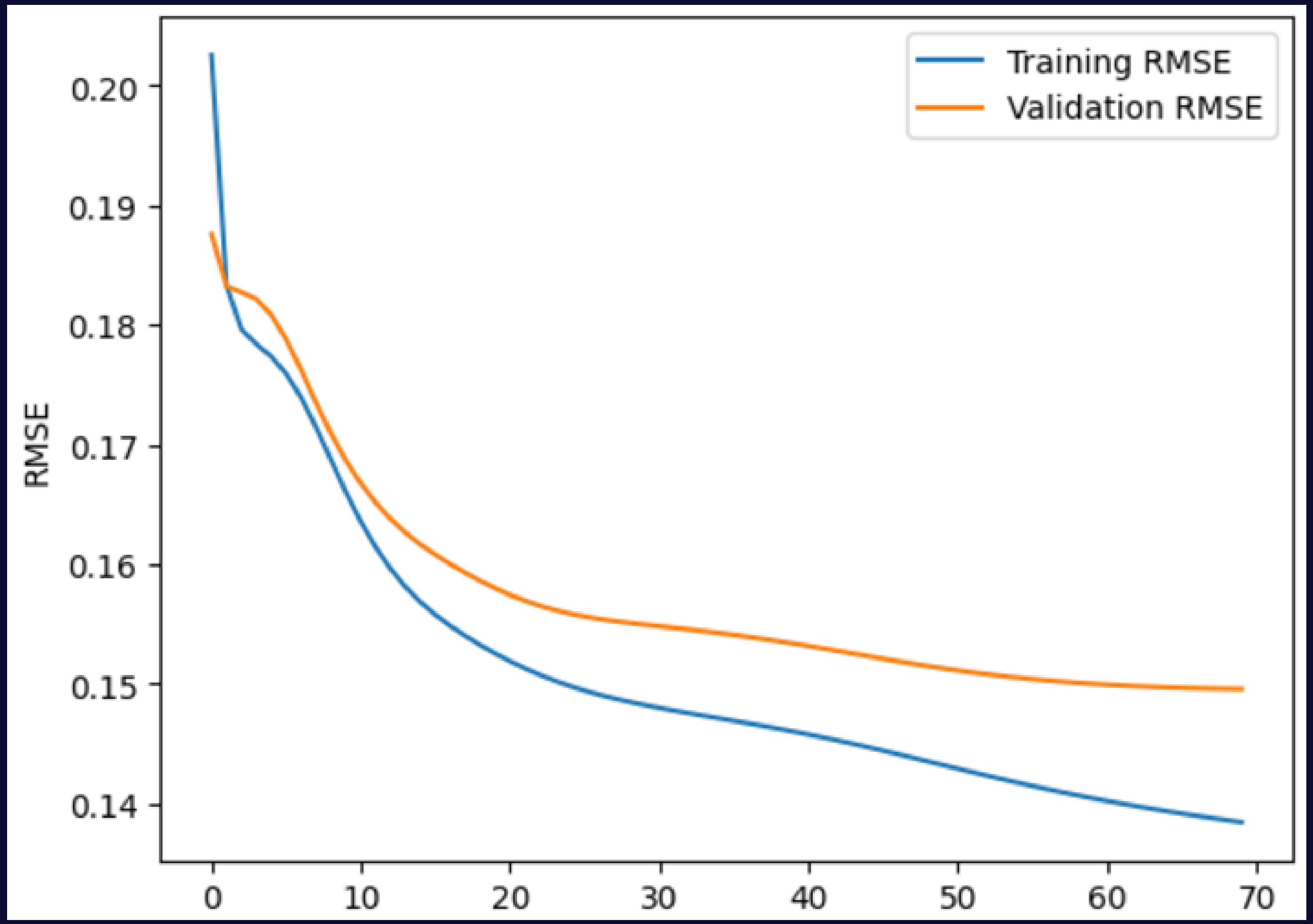
```python
for i in range(inputs.shape[0]):
    #1:Forward Pass Hidden Layer
    hidden_neuron = neuron(inputs[i], wsh)
    hidden_outputs = hidden_neuron.activation('h')

    #2:Output Layer
    output_neuron = neuron(np.append(hidden_outputs, values: 1), wsy) #with bais
    #output_neuron = neuron(hidden_outputs, wsy) #without bais
    output_values = output_neuron.activation('o')
```

RMSE

```
#calculating the RMSE for the training for each epoch
RMSE_t = np.sqrt(total_error / inputs.shape[0])
rmse_training[epoch] = RMSE_t
```

```
RMSE_v = np.sqrt(total_val_error / inputs_val.shape[0])
rmse_validation[epoch] = RMSE_v
```

```
#Plotting RMSE
plt.plot( *args: rmse_training, label='Training RMSE')
plt.plot( *args: rmse_validation, label='Validation RMSE')
plt.xlabel('Epoch')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```



Stopping Criteria

```
def stopping_cr(epochs,rmse_validation, threshold, lookback):  1 usage
    global stopping_counter
    if epochs < lookback:
        return False
    dif=rmse_validation[epochs] - rmse_validation[epochs-lookback]
    if abs(dif) < threshold or dif>0:
        stopping_counter+=1
    else:
        stopping_counter=0
    if stopping_counter == 3:
        return True
```


```
if stopping_cr(epoch,rmse_validation, threshold: 0.0001, lookback: 7):
    print(f"Early stopping triggered at epoch {epoch + 1}")
    break
else:
    print(f"pass counter is {stopping_counter}")
```

Game Integration

```python
import numpy as np

class neuron:
    def __init__(self, inps, ws):
        self.inps = np.array(inps)
        self.ws = np.array(ws)
        self.vs = np.dot(self.inps, self.ws)  # Weighted sum
        self.hiddens = np.zeros(self.ws.shape[1])  # Initialize hidden activations
        self.outputs = np.zeros(self.ws.shape[1])  # Initialize output activations
        #self.es = np.zeros(len(actual_outputs))  # Error initialization
        #self.gd_ys = np.zeros(self.ws.shape[1])  # Output layer delta
        #self.gd_hs = np.zeros(self.ws.shape[1])  # Hidden layer delta

    def activation(self, layer):
        # Sigmoid activation function
        if layer == 'h':
            self.hiddens = 1 / (1 + np.exp(-0.9*self.vs))
            return self.hiddens
        else:
            self.outputs = 1 / (1 + np.exp(-0.9*self.vs))
            return self.outputs
```

Game Integration

```
class NeuralNetHolder:

    def __init__(self):
        super().__init__()
        self.weightsh=[[ 1.03836913,  2.36806616 , 1.27618052, -1.39276823 , 6.69020476, -1.07442192,
   8.21462095 ,-0.07544661],
 [ 0.4515546 ,  1.32233404 , 1.35507259 , 1.02626652, -0.41703071 , 0.62394654,
  -0.19060612 , 5.58428387],
 [ 0.46187807 , 0.04951344 , 0.37191654 , 0.99390402, -3.01751414,  0.67940092,
  -3.27905539 , 0.25740749]]


        self.weightsy=[[  0.01372197 , -1.77085417],
 [  0.74317918,  -6.91405856],
 [ -1.16739402 , -2.97859069],
 [  2.25162679 ,  6.35944113],
 [-11.16891962 ,  6.85191718],
 [  2.36269928 ,  4.652875341],
 [ 11.39313472 , -0.32452071],
 [ -9.36612209 ,  0.88832898],
 [  4.57763116  ,-1.6717935 ]]
```

Game Integration

```
def predict(self, input_row):
    # WRITE CODE TO PROCESS INPUT ROW AND PREDICT X_Velocity and Y_Velocity
    input_row = list(map(float, input_row.split(','))) #taking the CSV row and making it a list

    max_x=640.113
    max_y=650.402
    max_vx=7.999
    max_vy=7.864
    min_x=-641.203
    min_y=66.1015
    min_vx=-6.728294
    min_vy=-7.920305
    #Normalizing input by the same way the neural network was trained
    input_row[0]=(input_row[0]-min_x)/(max_x-min_x)
    input_row[1]=(input_row[1]-min_y)/(max_y-min_y)
    #adding the bais
    input_row.append(1)
```

Game Integration

```
#Forward Pass
hidden_neuron = neuron(input_row, self.weightsh)
hidden_outputs = list(hidden_neuron.activation('h'))
hidden_outputs.append(1) #adding the second bais

output_neuron = neuron(hidden_outputs, self.weightsy)
output_values = output_neuron.activation('o')

#Step4 Denormaliz:
vx=output_values[0]*(max_vx-min_vx)+min_vx
vy=output_values[1]*(max_vy-min_vy)+min_vy
denormalized_output = [vx,vy]


return denormalized_output
```

Thank You