Neural Network from Scratch

June 13, 2025

[83]:

```python
import numpy as np

class NeuralNetwork():

    #Set random seed
    np.random.seed(42)

    def __init__(self, X, y, n_hidden_neurons, output_act_fn="linear",␣
 ↪error_fn="mse"):
        self.y=y
        self.X=X
        self.n_input_neurons = X.shape[1]
        self.n_hidden_neurons = n_hidden_neurons
        self.output_act_fn = output_act_fn
        self.error_fn = error_fn

    #initialize weights and biases with random values (array of random numbers)
        self.input_hidden_weights = np.random.randn(self.n_input_neurons,self.
 ↪n_hidden_neurons)
        self.hidden_biases = np.random.randn(self.n_hidden_neurons)
        self.hidden_output_weights = np.random.randn(self.n_hidden_neurons,1)
        self.output_bais = np.random.randn(1)


    # activation function
    def activition(self, x, act_fn):
        if act_fn == 'sigmoid' :
            return 1/ (1 + np.exp(-x))
        elif act_fn == 'relu' :
            return np.maximum(0,x)
        elif act_fn == 'linear':
            return x
        else:
            raise Exception('Unknown activation function')


    def activition_derivitive(self, x, act_fn):
        if act_fn == 'sigmoid' :
            return x * (1-x)
        elif act_fn == 'relu' :
            return np.where(x>0,1,0)
        elif act_fn == 'linear':
            return 1
        else:
            raise Exception('Unknown activation function')

        # Forward Propagation
    def forward_pass(self, X):
        #input layer
        self.input = X

            #Hidden layer
        self.hidden = self.activition(np.dot(self.input, self.
↪input_hidden_weights) + self.hidden_biases , "relu")

            #Output layer
        self.output = self.activition(np.dot(self.hidden, self.
↪hidden_output_weights) + self.output_bais , self.output_act_fn)

        return self.output
    def error(self, y_true, y_pred):
        if self.error_fn == 'mse':
            return np.mean(np.square(y_true - y_pred))
        elif self.error_fn == 'cross_entropy':
            return -np.mean(y_true * np.log(y_pred+0.00001)+(1 - y_true)* np.
↪log(1 - y_pred+0.00001))
        else:
            raise Exception('Unknown error function')
    def error_derivitive(self, y_true, y_pred):
        if self.error_fn == 'mse':
            return 2* (y_pred - y_true) / y_true.size
        elif self.error_fn == 'cross_entropy':
            return (y_pred - y_true) / (y_pred * (1- y_pred+0.00001)* y_true.
↪size)
        else:
            raise Exception('Unknown activation function')

        # Backpropagation
    def backward_pass(self, X, y_true, y_pred, learning_rate):
        # output layer
        self.output_error = self.error_derivitive(y_true, y_pred) * self.
↪activition_derivitive(y_pred , self.output_act_fn)
        self.output_bais -= learning_rate * np.sum(self.output_error , axis=0)
        self.hidden_output_weights -= learning_rate * np.dot(self.hidden.T,self.
 ↪output_error)

            # Hidden layer
        self.hidden_error = np.dot(self.output_error, self.
 ↪hidden_output_weights.T) * self.activition_derivitive(self.hidden, "relu")
        self.hidden_biases -= learning_rate * np.sum(self.hidden_error,axis=0)
        self.input_hidden_weights -= learning_rate * np.dot(X.T,self.
 ↪hidden_error)

            # return weights and biases
        return self.input_hidden_weights, self.hidden_biases, self.
 ↪hidden_output_weights, self.output_bais

        # Training
    def train(self, X, y, learning_rate, epochs):
        for epoch in range(epochs):
            y_pred = self.forward_pass(X)
            wih, bh, who , bo =self.backward_pass(X, y, y_pred, learning_rate)
            if epoch % 500 == 0:
                print('Epoch: {}, Loss: {:.3f}'.format(epoch, self.error(y,␣
 ↪y_pred)))
        print('Trainig complete!')
        return wih, bh, who, bo

        # Predition
    def predict(self, X):
        if self.error_fn == "mse":
            return self.forward_pass(X)
        elif self.error_fn == 'cross_entropy':
            return np.where(self.forward_pass(X) > 0.5, 1, 0)
```

[66]:

```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=1000, n_features=3 , noise=20 ,␣
 ↪random_state=42)
y = y.reshape(-1,1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[67]:

```
nn = NeuralNetwork(X_train, y_train, n_hidden_neurons=128,␣
 ↪output_act_fn='linear',error_fn='mse')
wih, bh, who, bo= nn.train(X_train, y_train, learning_rate=0.001, epochs=5000)
```

```
Epoch: 0, Loss: 13892.784
Epoch: 500, Loss: 374.455
Epoch: 1000, Loss: 367.597
Epoch: 1500, Loss: 362.922
Epoch: 2000, Loss: 359.279
Epoch: 2500, Loss: 356.310
Epoch: 3000, Loss: 354.433
Epoch: 3500, Loss: 352.962
Epoch: 4000, Loss: 351.569
Epoch: 4500, Loss: 350.323
Trainig complete!
```

[68]:

```
# Weights and biases
print('Input-hidden weights:\n', wih)
print('Hidden-output weights:\n', who)
print('Hidden biases:\n', bh)

print('Output bias:\n', bo)
```

```
Input-hidden weights:
 [[ 0.46218215  0.32542526  0.16580122  1.95488085  0.38762215 -0.20067982
   2.37748297  1.29989384 -2.04491936  0.99280742 -1.00687122 -0.45065981
   0.24196227 -2.53614614 -2.86502429 -0.57946922 -0.97141953  0.43746702
  -1.1375737  -2.10772852  1.68217445 -1.30138492  0.0675282  -1.46967514
  -0.85697598  0.04103586 -0.18329947  0.27823312  0.29107773 -1.26135854
  -0.29820138  2.74711965 -0.94704823 -1.47916521  1.36182059 -1.38830393
  -0.14503344 -2.32309602 -1.27638632  0.53413518  1.37159347  0.25528276
   0.06449208 -0.4329507  -2.22006145 -0.54760222 -0.35777331  0.84362734
   0.56385418 -0.65745027  0.81709272 -0.34187398 -0.38302591  0.90215247
   1.3512      0.82033207 -2.03080346 -0.66764091  0.45827357  2.0147918
  -0.42215082 -0.18260957 -0.91808413 -1.86473578  1.12831257  1.94606514
  -1.14753947  0.31739748  0.58125847 -1.36779867  1.01385854  3.32589176
   0.17883453  2.03555612 -3.01400746  0.62750092  0.13721074 -0.990968
   0.81978882 -2.19759049 -0.09314316  1.0601566   2.66800386 -0.15633042
  -0.87627891 -0.86861731  0.78476595  0.29750958 -0.26421085 -0.72215849
   0.27824087  2.35579203 -0.54494303  0.59275396 -1.33860321 -2.44785674
  -0.53712489  1.68819686  0.00511346  0.43626686 -1.46287535 -0.443088
  -0.04317059 -0.79546518 -0.02259556  1.44727207  2.90546878  0.97093419
  -0.42545192 -0.14622313 -1.76017047  0.83785549  0.11687791  2.78604817
  -0.29754586  0.03559718 -0.03297419 -1.11915353  1.11644524  2.31861803
   1.27818612 -1.21913875  1.15585473 -1.90729882  1.08559308  2.23150204
  -2.71706934 -1.39488468]
 [-0.00442579 -0.94091384 -2.27458144  0.54909627 -1.03743309  0.31757436
  -1.06957905  1.99265692 -0.89502155  0.42080245  0.30863459 -1.24499542
```

```
   0.22745993  0.29548771 -2.65461159 -0.04847006  0.32516244  0.37562201
  -1.01047592 -2.67174119  0.79936286 -0.04671264  0.25049285  0.15920662
  -1.48683963  0.09516851  0.81307274 -0.75585487  2.6197359   0.03797221
  -1.35885333  1.88079317 -1.4391623   0.11762717  1.7808744  -1.19762485
   1.44405059  0.59427192  0.75905219  2.03933201  0.20573006 -0.72073368
  -0.70176757 -0.9378861  -0.06616551  0.45844735  0.5542515   0.75592334
   0.26788389  2.25313165  0.16985023  2.7788275   1.35987084 -0.85993899
  -1.49953486  0.29078571 -0.98026318  1.11075019  0.61490588  0.85108741
  -0.71318754 -2.61000742 -0.36714651  0.28794195  0.53611805 -1.32912275
  -0.69410131  1.56653742 -0.78166424 -0.05178262  0.27903486 -1.30846656
   0.4631465   0.89099817  0.55401581  0.57984427 -1.42021078 -1.68461856
   0.76569207 -0.0861997   0.61062185  4.00623855  1.12663503  1.35869671
   0.94341078  0.49281619 -0.36869928  0.63838179 -0.42062862 -0.61557853
  -0.31910665  0.91337097  2.74870441 -1.56153159 -0.49854119 -2.20883677
  -1.7518049   2.5817312   0.06428002 -1.42756813 -0.77719381  0.55393923
  -0.7049738   0.29710192  0.16937041 -0.63759652  2.27346071  0.59148973
  -3.06137419  0.14888587  0.03693704  1.27611788 -0.9156966  -0.74853593
   0.36336082  0.84941854 -0.77134528  0.31998717 -1.15715922  0.2807826
   2.35788674  0.28047312 -1.23830583  0.55971073  2.55534557  1.7585209
  -2.31908636 -1.16643643]
 [ 1.44143401 -1.30273023  0.0982733   0.61023393 -1.66404253 -0.03067419
  -3.47219136 -0.84555609 -0.42031453 -1.15138212  1.9156877  -1.51425349
  -0.44004449  0.20817502  1.63881898 -1.40745931  1.20484096 -0.35448066
  -0.9634703   0.71574518  0.06371467 -0.82447418  0.06980208 -0.37778604
  -0.24379913  0.72297561  2.18824832 -1.30149344  1.97500375 -1.61395584
  -0.10866721  0.44511048 -0.32118724 -1.07328422  0.37333917 -1.07255418
  -0.90382202  0.93334217  0.54279918 -0.54304684  0.98241633  0.3384591
   0.764915    0.34414505 -1.37054077 -1.15988946  0.62022934  0.48661066
   0.02143086  0.01011749  1.11420243 -0.53548405  1.40004348 -0.39699959
  -0.6024473   0.88528525  0.48706769  1.30353547  1.7515929   0.25028817
   0.83859257 -0.22403764  0.56114245 -0.68554737  0.04274842  1.21467233
  -1.19844121  3.55636012 -1.14787216 -1.14145097  1.70353061  1.23759415
   0.76312118  0.14271139  0.59857882 -0.85956893  0.01195135 -0.47043683
   0.93491708 -0.33031212 -0.93372478 -0.10244442  0.33170286 -0.60153854
  -0.89814501  0.17555379  0.18705018 -0.42793725 -0.2771536   0.6545942
  -1.20497703 -1.71998109 -0.1647706   0.00852352  0.60125401  0.97628025
   0.39726967 -0.85364041 -0.01901621 -1.29851232 -0.28023824 -0.29030066
   0.72460099 -0.99323958  0.52207924  1.87842071  0.87463242 -0.15788732
   0.49843531 -0.45229578  0.32160484  0.42938905  0.51737127 -1.11377118
  -0.04258467  0.68910059  2.07189998  1.6695096   2.25143301 -0.2052847
   0.85781939  0.23552898  2.08110418 -0.88709053 -1.0037583  -1.84326642
  -2.3985645  -1.18303281]]
Hidden-output weights:
 [[ 0.77127738]
 [-1.68740343]
 [-1.6514519 ]
 [ 1.22359783]
 [-1.38787795]
```

```
[-0.81706864]
[-2.07535953]
[ 1.63401409]
[-2.61559157]
[ 0.62517808]
[-1.00070499]
[ 0.82169753]
[-0.76325916]
[-2.22028514]
[-3.59788847]
[ 0.26688467]
[-0.2442513 ]
[-1.13964814]
[-0.48387187]
[-3.2579528 ]
[ 1.1950648 ]
[-1.73886255]
[-0.65183611]
[ 0.14251482]
[-1.68898119]
[-0.54336465]
[ 1.82168908]
[ 0.08466454]
[ 1.62217277]
[-1.09681149]
[-0.56478992]
[ 2.92179914]
[-3.33271513]
[-1.46694335]
[ 1.8627603 ]
[-1.41900868]
[ 1.93783147]
[-2.38230332]
[-0.05273362]
[ 0.09860665]
[ 1.59489029]
[ 0.13923718]
[ 0.32647709]
[-0.51220271]
[-1.87415922]
[-0.75269223]
[ 0.86339984]
[-0.48237294]
[ 0.5946124 ]
[ 2.22733168]
[ 0.92419535]
[ 0.54468774]
[ 1.83476189]
```

```
[-0.93158284]
[-2.8154273 ]
[-0.42320537]
[-2.73130807]
[-1.49232127]
[ 1.07311388]
[ 2.57086843]
[ 0.30180414]
[-2.0550943 ]
[ 0.69411754]
[-2.76071315]
[ 0.96240502]
[ 2.05629809]
[-2.2048558 ]
[ 3.28019096]
[ 1.13609466]
[-1.14835529]
[ 1.69011895]
[ 3.81512183]
[ 0.57057891]
[ 1.33966561]
[-1.42359787]
[-0.93613482]
[ 1.1320221 ]
[-1.79371717]
[ 1.10323956]
[-0.96556529]
[ 0.72202702]
[ 1.34911882]
[ 2.53479484]
[-0.82588938]
[-0.74821561]
[-1.13061931]
[-0.06684489]
[ 0.34919607]
[ 0.64528216]
[-1.78478673]
[ 0.87502155]
[ 2.84092817]
[ 1.25890978]
[ 1.54281163]
[-1.47945244]
[-2.24153225]
[-2.36117002]
[ 3.57469381]
[ 0.65436566]
[-1.30886287]
[-0.8685134 ]
```

```
 [-1.02147502]
 [ 3.04993551]
 [-0.47395864]
 [-0.07916169]
 [ 1.89952411]
 [ 2.44328571]
 [ 0.68141195]
 [-2.4004152 ]
 [ 0.65237837]
 [ 1.16312199]
 [ 2.10812542]
 [ 2.09311468]
 [ 1.96323762]
 [-0.82030826]
 [ 0.57936264]
 [-1.35913466]
 [ 2.03576393]
 [-2.03407222]
 [ 2.59587243]
 [ 2.00553461]
 [-0.79808752]
 [-1.05215708]
 [-2.12987003]
 [ 2.076819  ]
 [ 2.51687888]
 [-3.44033733]
 [-2.31646109]]
Hidden biases:
 [-0.83217041  0.48078105  0.60500092  1.83181869  0.84768687 -0.43361308
 -0.31495565  0.50567809 -1.26771871  1.72089313  0.86846817 -0.59073022
 -1.71313453  1.64559666 -0.66618967  1.31707125 -1.57484585 -1.05687093
 -0.02871066  0.42936581 -0.82510897  0.48114156 -1.06762043 -0.18754183
  0.12980313  0.63167599  1.14661419 -0.88686223 -1.37384133  1.28384716
  0.29103455 -0.56827477  2.50065608  0.59289856  1.32468887 -0.00681422
  2.53579178  2.59179591 -0.3579881   0.65795527  0.58878343  1.35347752
 -1.14431306  0.77684639  0.83432256 -1.6018695  -1.42044452 -2.2075279
 -0.27662089  1.0778175   1.55049274 -0.19984998  1.42375252 -1.43011273
 -2.12283033  0.09133101  0.73402503 -0.04115782 -1.95155762  0.43518948
 -1.30749202  0.53624658  0.47927843 -1.36619846 -0.55337524 -1.2208661
 -0.50360656  1.06612    -1.3447997   0.44376862 -0.48235377 -0.49798926
 -0.04761146 -1.03443846 -0.68489279 -1.65346779  2.10230124  0.03115642
 -0.89415368  0.37019393 -0.23489184  0.09947136  0.57638062  0.81055735
 -0.73941678 -0.60578679 -0.32259746 -2.35380474 -1.7089166   1.83537318
  1.8712451   0.03133935  0.55625421 -0.03510943  3.07908381  0.5831429
 -0.24818138 -1.61537568 -1.60644632  0.35414385 -0.9858581  -1.39102994
 -1.88373002 -1.02342941  1.68566408  0.91367684  0.16209647  1.41598164
 -0.30243442 -0.94781571  1.19579611  1.18321205 -1.56529744 -0.91770512
 -0.72903446 -1.45742261  1.17371663  2.23042967 -1.34104751  1.00281478
 -1.04570909 -0.32285165 -1.27323543 -1.16332243 -0.85025504 -1.38503437
 -0.12145419 -0.08248742]
Output bias:
 [-0.4295871]
```

[72]:

```python
# Accuracy
from sklearn.metrics import r2_score
y_pred = nn.predict(X_test)
print('R2 Score: {:.3f}'.format(r2_score(y_test , y_pred)))
```

```
R2 Score: 0.975
```

[73]:

```python
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print('R2 Score: {:.3f}'.format(r2_score(y_test, y_pred)))
```

```
R2 Score: 0.976
```

[76]:

```python
# MlpRegressor
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(128,), activation='relu', solver='adam',␣
 ↪max_iter=5000, random_state=42)
y_train = y_train.reshape(-1)
mlp.fit(X_train, y_train)
```

[76]:

```
MLPRegressor(hidden_layer_sizes=(128,), max_iter=5000, random_state=42)
```

[77]:

```python
y_train.shape
```

[77]:

```
(750,)
```

[78]:

```python
y_pred = mlp.predict(X_test)
y_test = y_test.reshape(-1)
print('R2 Score: {:.3f}'.format(r2_score(y_test, y_pred)))
```

```
R2 Score: 0.976
```

[84]:

```python
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,␣
 ↪n_redundant=5, random_state=42)
y = y.reshape(-1, 1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[85]:

```
nn = NeuralNetwork(X_train, y_train,  n_hidden_neurons=128,␣
 ↪output_act_fn='sigmoid', error_fn='cross_entropy')
wih, bh, who, bo = nn.train(X_train, y_train, learning_rate=0.001, epochs=5000)
```

```
Epoch: 0, Loss: 5.929
Epoch: 500, Loss: 4.358
Epoch: 1000, Loss: 2.837
Epoch: 1500, Loss: 0.827
Epoch: 2000, Loss: 0.657
Epoch: 2500, Loss: 0.580
Epoch: 3000, Loss: 0.525
Epoch: 3500, Loss: 0.486
Epoch: 4000, Loss: 0.454
Epoch: 4500, Loss: 0.424
Trainig complete!
```

[89]:

```
# Weights and biases
print('Input-hidden weights:\n', wih)
print('Hidden-output weights:\n', who)
print('Hidden biases:\n', bh)
print('Output bias:\n', bo)
```

```
Input-hidden weights:
 [[ 4.91183828e-01 -1.00468647e-01  6.44106378e-01 …  2.18844993e+00
  -9.56418207e-01 -4.09697867e-01]
 [ 9.15288413e-02 -4.90635188e-01 -1.53993291e+00 …  1.04335534e+00
  -1.47839834e+00 -4.42668812e-01]
 [ 1.26456992e+00 -7.14245936e-01  4.96105956e-01 … -6.09409034e-01
  -2.14842526e+00 -5.82169478e-01]
 …
 [-1.53604276e-01  1.68445391e-01  1.13970488e+00 … -5.75899104e-01
  -2.39451060e-02  2.23720422e+00]
 [ 1.72978081e+00  4.83170695e-01 -1.35860334e-03 … -8.48581350e-01
  -9.56506553e-01 -1.96117134e+00]
 [-6.46226342e-01 -1.31219010e+00  1.66662767e+00 …  3.94603584e-01
   2.12066826e-01 -2.76227811e-01]]
Hidden-output weights:
 [[-2.62953365e-01]
 [-3.29512105e-01]
 [ 1.20847033e+00]
 [ 3.13681443e-01]
 [ 3.67716485e-01]
 [ 2.14878029e+00]
 [-9.06006210e-01]
 [ 7.28395787e-01]
 [ 2.49850589e-01]
 [ 1.79070340e-01]
 [ 1.62902030e+00]
 [ 1.50191246e-01]
 [ 6.30643759e-01]
 [ 7.44890741e-01]
 [-3.55293161e-01]
 [-1.16744705e+00]
 [-1.11634281e+00]
 [-2.08801813e+00]
 [ 1.94963110e+00]
 [-9.62831219e-01]
 [-2.01166385e-01]
 [-4.93545795e-01]
 [ 3.14794395e-01]
 [ 9.12193259e-01]
 [ 7.69160070e-01]
 [-5.09639344e-01]
 [-8.08503772e-02]
 [ 2.20992921e-01]
 [-3.52910269e-01]
 [ 1.29207373e+00]
 [ 4.65078921e-01]
 [ 6.73924164e-01]
 [ 3.79560790e-01]
 [ 1.02327370e+00]
 [-1.72077933e-01]
 [-1.20128929e+00]
 [ 9.38372818e-02]
 [-3.49494225e-01]
 [-5.76451809e-01]
 [ 5.41559672e-01]
 [ 1.18880083e+00]
 [-7.78755940e-02]
 [ 5.01976625e-01]
 [ 1.65749483e+00]
 [-2.86965768e-02]
 [ 2.60216650e+00]
 [ 5.59467650e-01]
 [-1.66037264e+00]
 [ 7.14018871e-01]
 [ 3.32919905e-01]
 [ 1.13311262e+00]
 [ 3.85118614e-01]
 [-3.44928082e-01]
 [-1.33809662e+00]
 [-3.00449579e-01]
 [-8.47666304e-01]
```

```
[-4.07347343e-01]
[ 3.95955381e-01]
[ 1.67934345e+00]
[ 3.11628861e-01]
[-1.14507415e+00]
[ 3.02528376e-01]
[ 1.14150893e+00]
[-7.99877586e-01]
[-2.04818232e-01]
[-7.91226509e-02]
[-8.29164617e-01]
[ 4.58560775e-01]
[-2.16914496e-01]
[ 2.98886535e-01]
[-8.81705287e-01]
[-4.57696367e-01]
[ 6.98224350e-01]
[-3.21102981e-01]
[ 1.81459685e+00]
[-3.97360629e-01]
[ 1.07694466e-01]
[ 1.04910342e+00]
[-1.40829218e+00]
[-1.82906818e+00]
[-1.61942176e+00]
[ 7.91287626e-01]
[-6.68901975e-01]
[ 2.53968644e+00]
[-7.04036200e-01]
[ 3.53896673e-02]
[ 1.51545031e+00]
[ 2.07393002e+00]
[ 2.06609933e+00]
[ 1.07752347e+00]
[ 8.06221241e-01]
[ 1.33097768e-01]
[ 6.12878291e-01]
[-5.15349945e-01]
[-1.45379295e+00]
[ 4.56259955e-02]
[-1.43021254e-01]
[-6.08086704e-01]
[ 6.13906711e-01]
[ 8.12328265e-01]
[ 2.84189715e-02]
[ 1.47680333e+00]
[-1.36056616e+00]
[-6.51691554e-02]
```

```
 [-8.49853393e-01]
 [-1.86319506e+00]
 [-2.65477942e-01]
 [-6.54351808e-01]
 [ 1.52254581e+00]
 [ 5.83214630e-01]
 [-1.16891117e+00]
 [ 1.61803160e+00]
 [ 8.49584949e-01]
 [ 4.63230371e-01]
 [-2.55350397e-01]
 [-1.13597249e+00]
 [-4.07339336e-01]
 [ 9.05416839e-01]
 [ 1.84922231e+00]
 [ 1.44765154e+00]
 [-5.22369209e-01]
 [-1.12947191e+00]
 [ 2.45430768e-03]
 [-1.71877497e+00]
 [ 2.15801035e-02]
 [-2.51423751e-01]
 [-3.59440498e-01]
 [-1.56587912e+00]]
Hidden biases:
 [ 1.50684680e-01 -1.22454373e-01  5.81112020e-01  9.13443866e-01
 -8.20916545e-01 -1.36609613e+00 -1.77491381e+00  4.22247188e-01
 -1.10081470e+00 -2.15739063e+00  3.66696126e-01  2.45745054e+00
  2.19488430e-03  8.15567871e-01  7.82697777e-02 -4.84083570e-03
  9.72036244e-01 -1.85451013e-01  1.35197124e-01  3.37483153e-01
 -6.70137701e-01  1.03576276e+00 -1.69921258e-01 -7.69145973e-01
  4.81055752e-01 -9.07268126e-01  3.59179358e-02 -3.05324994e-02
  1.10251096e+00  3.27580979e-01 -4.74393310e-02  7.29487603e-01
  1.35969231e+00  4.57646189e-01  1.55932770e-03 -1.31511416e+00
 -1.09375048e+00 -2.89926459e-01 -5.82521663e-01 -2.14709735e-01
 -8.72640002e-02  5.35119892e-01 -8.91716563e-02  3.54203591e-01
  5.84489088e-02 -1.99997576e+00 -9.43431971e-01 -7.37698748e-02
 -1.23036626e+00  5.72936000e-01  1.35367806e+00  1.17017337e+00
 -2.05834951e-01  1.59255845e+00  1.64583245e-01 -3.12462346e-01
 -6.19756921e-01 -3.39757709e-01 -3.23994426e-01  1.38817512e-01
  2.00547472e-01 -1.61303620e-02  3.47276110e-01  1.21213906e+00
  9.76759234e-01 -1.48536395e+00 -2.51383502e+00  8.89003020e-01
 -1.35378062e+00 -2.43809373e-01 -1.17243364e+00 -1.78921225e+00
  4.97402028e-01  7.71311955e-01 -5.64867749e-01 -2.59446949e+00
 -5.60251648e-01  3.41921728e-01 -1.49989368e+00  2.71897591e-01
  1.85622729e-01  5.06162968e-01  1.65916671e-01 -1.08088382e+00
  1.26429254e+00 -1.66665792e-01 -4.88947958e-02 -9.80846846e-01
 -4.83571012e-01 -9.59810243e-01 -2.84250508e-01  1.66395775e+00
 -1.40129638e+00 -1.58873116e+00  1.62528613e+00 -2.08792029e-01
 -6.70189863e-01  1.07337435e+00 -6.48697237e-01  1.72914071e+00
  6.80467186e-01 -5.21072231e-01  2.27513730e+00 -6.00428720e-01
  8.64144802e-01  4.80856513e-01  1.32319774e+00  1.58728203e+00
 -1.70327417e-02 -7.70037001e-01  6.03358522e-01 -7.89145683e-01
  1.90108268e+00  9.62479779e-02 -3.66931436e-01  2.40674264e-01
 -1.33502981e+00 -1.02701156e+00  1.09885395e+00 -7.97438633e-01
 -1.03845233e+00  5.70107233e-01  1.18231294e+00  8.29450545e-01
  9.77880161e-01 -7.50850031e-01 -1.41932568e+00  1.67609508e+00]
Output bias:
 [0.40399672]
```

[86]:

```python
y_pred = nn.predict(X_test)

from sklearn.metrics import accuracy_score
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.908
```

[87]:

```python
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
y_train = y_train.reshape(-1)

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.792
```

[88]:

```python
# MLPClassifier
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(128,), activation='relu',␣
 ↪solver='adam', max_iter=5000, random_state=42)
mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.940
```

[ ]:

```python

```