Neural Networks I

Mark A. Austin

austin@umd.edu

ENCE 688P, Spring Semester 2021
University of Maryland

March 24, 2021

Overview

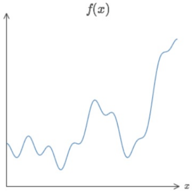
Part 02

Quick Review

Why Neural Networks?

Reasons to use Neural Networks:

- Neural networks are universal function approximators, no matter how complex:



- Neural network architectures are highly scalable and flexible.

Caveat:

- Very large neural networks may be close to impossible to train and generalize correctly.

Modeling Expectations

Capabilities of a Perceptron Model: (From Lippman, 1987)

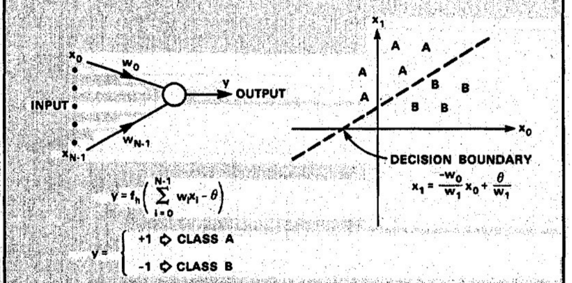
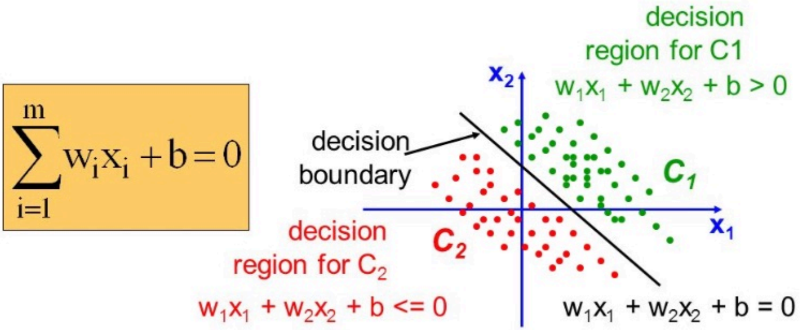


Figure 12. A single layer perceptron that classifies an analog input vector into two classes denoted A and B. This net divides the space spanned by the input into two regions separated by a hyperplane or a line in two dimensions as shown on the top right.

Perceptron Model as a Linear Classifier

Perceptron operating on real-valued vectors is a linear classifier:

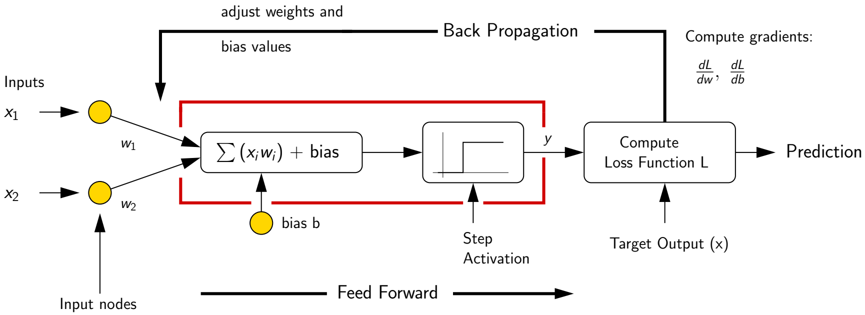


Addition of bias values expands modeling capability. No bias value → decision boundary constrained to pass through the origin.

Training a Single Perceptron Model

Training Perceptron Models

Network Architecture: Consider a single perceptron model with two input streams, weights and a bias, and step activation.



Input Data: $(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)$.

Training Perceptron Models

Training Objective

Find weight and bias (w_1, w_2, b) values to minimize difference between predictions and target values.

Quadratic Loss Function

Define a loss function L,

$$L(y, target(x)) = \frac{1}{2}\sum_{i=1}^{n}(y_i - target(x_i))^2 \tag{7}$$

Here, y_i is the network prediction for input x_i and $target(x_i)$ is the target value for learning.

Training Perceptron Models

Numerical Strategy: Use gradient descent algorithm to compute sequence of weight approximations, i.e.,

$$w_{n+1} = w_n - \eta \nabla L. \tag{8}$$

Here, w = matrix of network weights and η = learning rate.

Chain Rule: Network predictions y are a composition of the linear combiner + activation function.

Mathematically, L is related to x, w and b as follows:

$$L = L(f(g(x, w, b))) \rightarrow \frac{dL}{dw} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial w}. \tag{9}$$

Training Perceptron Models

Two problems with Step Activation:

- Can change weights without affecting L,
- Function is not continously differentiable.

Hence, replace step activation with sigmoid function:

$$y = f(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geq 0. \end{cases} \quad \rightarrow y = \sigma(z) = \left[\frac{1}{1+e^{-z}}\right]. \tag{10}$$

Derivative of sigmoid is easy:

$$\frac{dy}{dz} = \frac{d}{dz}\sigma(z) = \sigma(z)\left[1 - \sigma(z)\right]. \tag{11}$$

Back Propagation in Feed Forward Models

Minimize L. Here, L = L(y), where y = σ(z) and z = g(x,w,b).

Use chain rule to find derivative of L with respect to w:

$$\frac{dL}{dw} = \frac{\partial L}{\partial y} \cdot \frac{dy}{dz} \cdot \frac{\partial z}{\partial w}. \tag{12}$$

First term,

$$\frac{dL}{dy} = \frac{\partial}{\partial y}\left[\sum_{i=1}^{n}(y_i - target(x_i))^2\right] = \sum_{i=1}^{n}(y_i - target(x_i)). \tag{13}$$

Back Propagation in Feed Forward Models

Second term,

$$\frac{dy}{dz} = \frac{d}{dz}\sigma(z) = \sigma(z)\left[1 - \sigma(z)\right]. \tag{14}$$

Third term,

$$\frac{\partial z}{\partial w} = \frac{\partial}{\partial w}\left[\sum_{i=1}^{n} w_i \cdot x_i + b\right]. \tag{15}$$

Collecting terms,

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} & \frac{\partial L}{\partial w_2} & \frac{\partial L}{\partial b} \end{bmatrix}^T. \tag{16}$$

Update Weights: Plug equation 16 into equation 8. Repeat.

Metrics of Evaluation

Metrics of Evaluation

Confusion Matrix

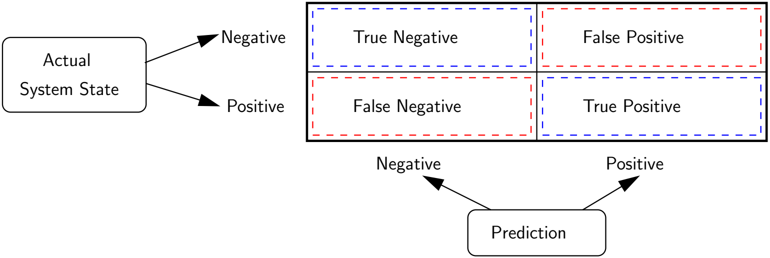
A simple metric to understand performance of a model in terms of predictions and their relationship to the actual state of a system.

Four Cases to Consider:

- True negative: The system state is negative; the model predicts negative.
- False positive: The system state is negative, but the model predicts positive.
- False negative: The system state is positive, but the model prediction is negative.
- True positive: The system state is positive and the model prediction is positive.

Metrics of Evaluation

Training Objective: We want:



- True negative and true positive numbers to be as high as possible,
- False positive and false negative to be as low possible.

Metrics of Evaluation

Accuracy:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{17}$$

Precision:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{18}$$

Recall:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{19}$$

F1 Score:

$$\text{F1} = 2\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{20}$$

Single-Layer

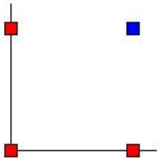
Perceptron Examples

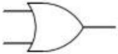
Example 1: Modeling Boolean Gates

Problem Description:

AND

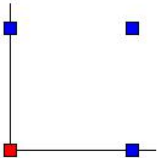
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1





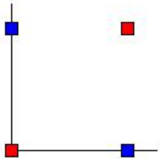
OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 1).

```
# ================================================================
# TestNeural-BooleanORGate.py: Perceptron model for boolean OR gate:
#
# Reference: Shukla, et al., Neural Networks from Scratch with
# Python Code and Math in Detail, Towards AI, 2020
#
# Modified by: Mark Austin                              October, 2020
# ================================================================

import math
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Define Sigmoid function:

def sigmoid(x):
   return 1/(1+np.exp(-x))

# Define derivative of Sigmoid function:

def sigmoid_der(x):
   return sigmoid(x)*(1-sigmoid(x))

# main method ...
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 2) ...

```
def main():
    print("--- Enter TestNeuralNetwork01.main()       ... ");
    print("--- ==================================== ... ");

    input_features = np.array( [[0,0],[0,1],[1,0],[1,1]] )

    print (input_features.shape)
    print (input_features)

    # Define target output:

    target_output = np.array([[0,1,1,1]])

    # Reshaping target output into vector:

    target_output = target_output.reshape(4,1)
    print (target_output)

    weights = np.array([[1.0],[2.0]])
    print(weights.shape)
    print (weights)

    bias = 0.3       # Bias weight:
    lr   = 0.05      # Learning Rate:
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 3) ...

```
    # Main loop for training network ...

    for epoch in range(20000):

       # feedforward input, feedforward output, back propogation ...

       inputs = input_features
       in_o    = np.dot(inputs, weights) + bias
       out_o   = sigmoid(in_o)

       # Calculate error in computed output ...

       error = out_o - target_output

       # Calculate derivative:

       derror_douto = error
       douto_dino = sigmoid_der(out_o)

       # Multiplying individual derivatives:

       deriv = derror_douto * douto_dino

       # Finding the transpose of input_features:

       inputs = input_features.T
       deriv_final = np.dot(inputs,deriv)
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 4) ...

```
    # Update the weights values:

    weights -= lr * deriv_final
    for i in deriv:
        bias -= lr * i #Check the final values for weight and biasprint (weights)

# Print summary of results ...

print("--- Weights:");

print (weights)

print("--- Bias: %f ... \n" %(bias))

print("--- Use trained network to predict values ... ");

print("--- Verify input [1,0] --> 1 ... ");

single_point = np.array([1,0]) #1st step:
result1 = np.dot(single_point, weights) + bias #2nd step:
result2 = sigmoid(result1) #Print final result

print("--- Result 1: %f ..." %(result1))
print("--- Result 2: %f ..." %(result2))

print("--- Verify input [0,1] --> 1 ... ");
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 5) ...

```
    single_point = np.array([0,1]) #1st step:
    result1 = np.dot(single_point, weights) + bias #2nd step:
    result2 = sigmoid(result1) #Print final result

    print("--- Result 1: %f ..." %(result1))
    print("--- Result 2: %f ..." %(result2))

    print("--- Verify input [1,1] --> 1 ... ");

    single_point = np.array([1,1]) #1st step:
    result1 = np.dot(single_point, weights) + bias #2nd step:
    result2 = sigmoid(result1) #Print final result

    print("--- Single input point [1,1] ...")
    print("--- Result 1: %f ..." %(result1))
    print("--- Result 2: %f ..." %(result2))

    print("--- Verify input [0,0] --> 0 ... ");

    single_point = np.array([0,0]) #1st step:
    result1 = np.dot(single_point, weights) + bias #2nd step:
    result2 = sigmoid(result1) #Print final result

    print("--- Single input point [0,0] ...")
    print("--- Result 1: %f ..." %(result1))
    print("--- Result 2: %f ..." %(result2))
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Step-by-step solution (pg 6) ...

```
    print("--- ==================================== ... ");
    print("--- Finished TestNeuralNetwork01.main()    ... ");

# call the main method ...

main()
```

Python + NumPy Code: Abbreviated Results ...

```
--- Summary of weights and biases ...
--- Weights:

    [ [8.46406006]
      [8.46563981] ]

--- Bias: -3.886041 ...
```

Example 1: Modeling an OR Gate

Python + NumPy Code: Abbreviated Results ...

```
--- Use trained network to predict values ...

--- Verify input [1,0] --> 1 ...
--- Result 1:  4.578019, result 2: 0.989829 ...

--- Verify input [0,1] --> 1 ...
--- Result 1:  4.579599, result 2: 0.989845 ...
---
--- Verify input [1,1] --> 1 ...
--- Result 1: 13.043659, result 2: 0.999998 ...
---
--- Verify input [0,0] --> 0 ...
--- Result 1: -3.886041, result 2: 0.020114 ...
```

Example 1. Modeling an OR Boolean Gate

DL4J: Create training dataset:

```
// Create matrix of input values ...

   double[][] matrixDouble = new double[][]{ {0.0, 0.0}, {1.0, 0.0},
                                             {0.0, 1.0}, {1.0, 1.0}};
   INDArray input01 = Nd4j.create(matrixDouble);

// Create vector of expected output values ....

   double[] vectorDouble = new double[]{0,1,1,1};
   INDArray output01 = Nd4j.create(vectorDouble).transpose();

   DataSet ds = new DataSet(input01, output01 );
```

DL4J: Dataset values:

```
Matrix of input values     Vector of output values
[ [       0,       0],     [       0,
  [ 1.0000,       0],       1.0000,
  [       0, 1.0000],       1.0000,
  [ 1.0000, 1.0000] ]       1.0000 ]
```

Example 1. Modeling an OR Boolean Gate

DL4J: Create Network Configuration:

```
    // Create neural network configuration builder ...

       MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
           .updater(new Sgd(0.1))
           .seed(seed)
           .biasInit(0)
           .miniBatch(false)
           .list()
           .layer(new OutputLayer.Builder( LossFunctions.LossFunction.MSE )
              .nIn(2)
              .nOut(1)
              .activation(Activation.SIGMOID)
              .weightInit(new UniformDistribution(0, 1))
              .build())
           .build();

    // Create multilayer network ...

       MultiLayerNetwork net = new MultiLayerNetwork(conf);
       net.init();
       net.setListeners(new ScoreIterationListener(1000));
```

Example 1. Modeling an OR Boolean Gate

DL4J: Summary of Network Model (4 nodes on hidden layer)

```
LayerName (LayerType)   nIn,nOut   TotalParams   ParamsShape
=================================================================
layer0 (OutputLayer)    2,1        3             W:{2,1}, b:{1,1}
-----------------------------------------------------------------
Total Parameters:  3    Trainable Parameters:  3
=================================================================
```

DL4J: Train the network for 10,000 epochs:

```
    for( int i=0; i <= 10000; i++ ) {
       net.fit(ds);
    }
```

Example 1. Modeling an OR Boolean Gate

DL4J: Trained weights and bias:

```
--- Layer: layer0 ...
--- ------------------------------------
--- Weights: [  6.9568,   6.9568 ] ...
--- Bias:       -3.2378 ...
```

Decision boundary:

$$f(x_1, x_2) = 6.9568\,(x_1 + x_2) - 3.2378 = 0.0. \tag{21}$$

DL4J: Trained model predictions:

```
[ 0.0378,
  0.9763,
  0.9763,
  1.0000 ]
```

Example 1. Modeling an OR Boolean Gate

DL4J: Evaluation Metrics:

```
# of classes:    2
Accuracy:        1.0000
Precision:       1.0000
Recall:          1.0000
F1 Score:        1.0000
Precision, recall & F1: reported for positive class (class 1 - "1") o
```

DL4J: Confusion Matrix:

```
 0 1
-----
 1 0 | 0 = 0
 0 3 | 1 = 1
```

References

- Lippmann R.P., An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987.
- Bhiksha R., Introduction to Neural Networks, Lisbon Machine Learning School, June, 2018.