Neural Networks I

Mark A. Austin

austin@umd.edu

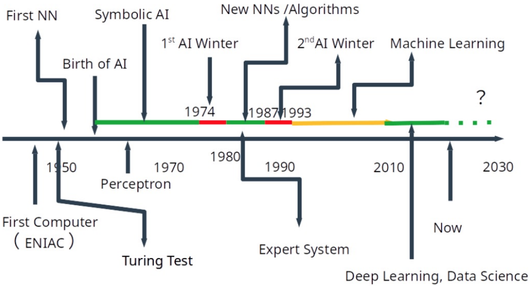
ENCE 688P, Spring Semester 2022
University of Maryland

May 27, 2022

Overview

Quick Review

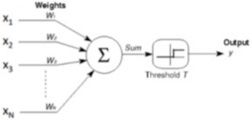
A Brief History



- 1943: First neural networks invented (McCulloch and Pitts)
- 1958-1969: Perceptrons (Rosenblatt, Minsky and Papert).
- 1980s-1990s: CNN, Back Propagation.
- 1990s-2010s: SVMs, decision trees and random forests.
- 2010s: Deep Neural Networks and deep learning.

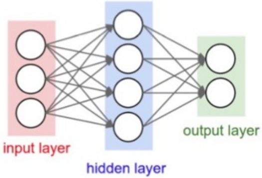
Machine Learning Capabilities (1980-1990)

Expressive Power of a Neural Network

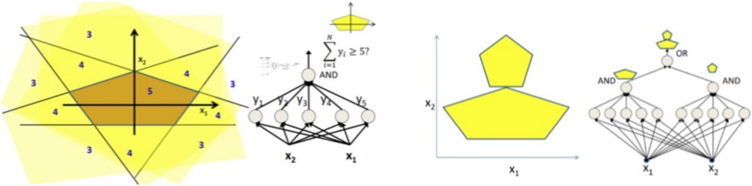


$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

Neural Network with Single Hidden Layer



Approximation of Functions / Boolean Logic



Introduction to Neural Networks

Purpose and Grand Vision (Aggarwal, 2018)

Purpose

Neural networks are a development to simulate the human nervous system for machine learning tasks by teaching the computational units in the model similar to human neurons.

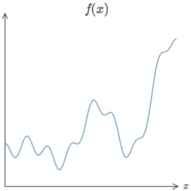
Grand Vision

Create AI by building machines whose architecture simulates computations in the human nervous system.

Why Neural Networks?

Reasons to use Neural Networks:

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



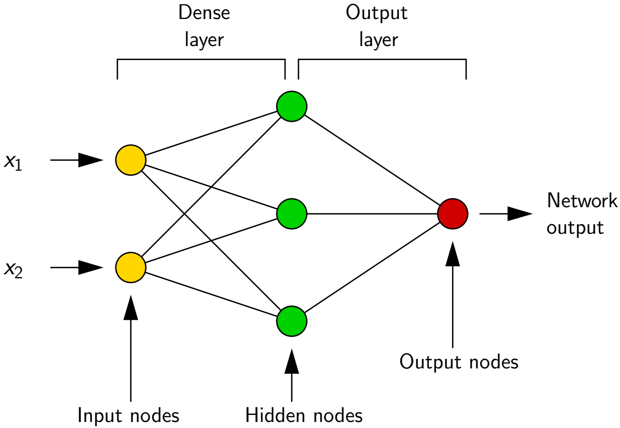
- Neural network architectures are highly scalable and flexible.

Caveat:

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

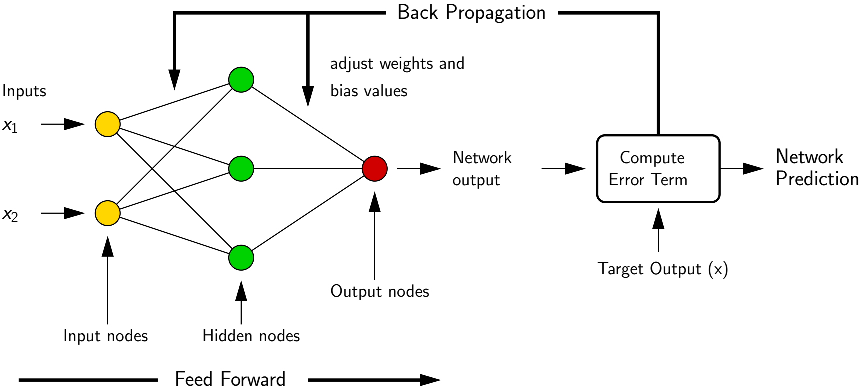
Basic Neural Network Architecture

Neural Network with One Hidden Layer:



Basic Neural Network Architecture

Training Procedure: Back Propogation



Modeling Expectations

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

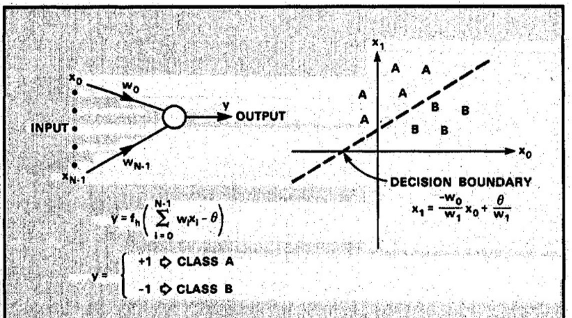


Figure 12. A single layer perceptron that classifies an analog input vector into two classes denoted A and B. This net divides the space spanned by the input into two regions separated by a hyperplane or a line in two dimensions as shown on the top right.

Modeling Expectations

Neural Networks with Hidden Layers: (From Lippman, 1987)

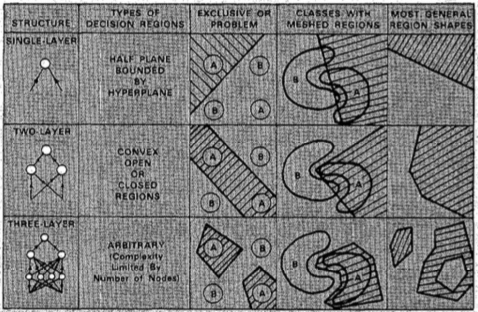


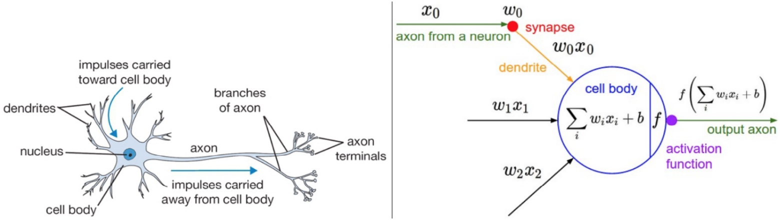
Figure 14. Types of decision regions that can be formed by single- and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A. Smooth closed contours bound input distributions for classes A and B. Nodes in all nets use hard limiting nonlinearities.

The Perceptron

Building Block of Machine Learning

A Little History / Biological Inspiration

Neural networks originally began as computational models of the brain (i.e., models of cognition).

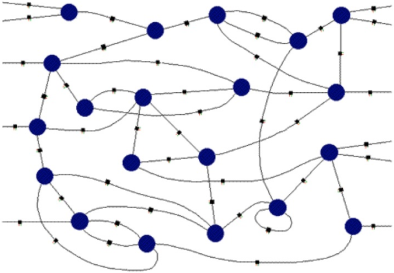


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

- Early models were based on association relationships.
- More recent models of brain are connectionist – neurons connect to neurons.

Connectionist Models

Present-day neural network models are connectionist machines.

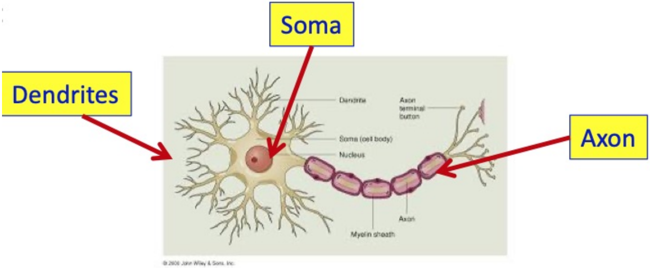


That is:

- Network of processing elements.
- Knowledge is stored in the connections between the elements.
- We need a model for these computational units.

Mathematical Model of a Single Neuron

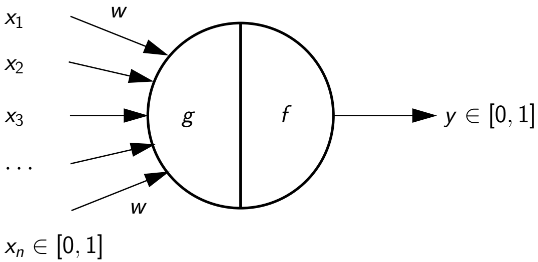
Modelling the Brain. Basic units are neurons:



- Signals come in through the dendrites into Soma.
- A signal exits via the axon to other neuron (only one axon per neuron).
- Neurons do not undergo cell division.

Mathematical Model of a Single Neuron

McCulloch and Pitts Model for a Single Neuron (1943):



First artificial neural network:

- Assumes boolean input (i.e., $x \in [0, 1]$).
- A neuron fires when its activation is 1, otherwise its activation is 0 (i.e, $y \in [0, 1]$).

Mathematical Model of a Single Neuron

Mathematical Model:

- All incoming connections have the same weight.
- Function $g()$ aggregates the inputs, i.e.,

$$g(x_1, x_2, \cdots, x_n) = g(x) = \sum_{i=1}^{n} x_i \tag{1}$$

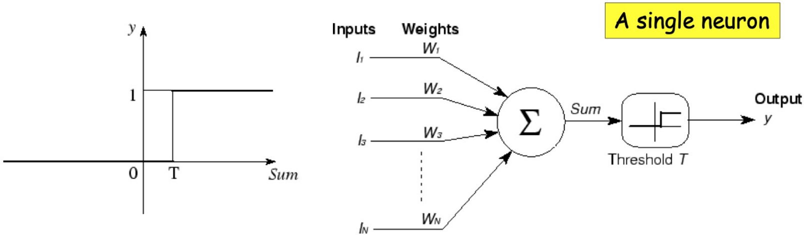
- Function $f()$ takes a decision based on this aggregation. y = 0 if any input x_i is inhibitory. Otherwise:

$$\begin{aligned} y = f(g(x)) &= 1 \text{ if } g(x) \geq \theta. \\ &= 0 \text{ if } g(x) < \theta. \end{aligned}$$

- θ is called the threshold parameter.

Mathematical Model of a Single Neuron

Behavior of a Simple Neuron Unit:

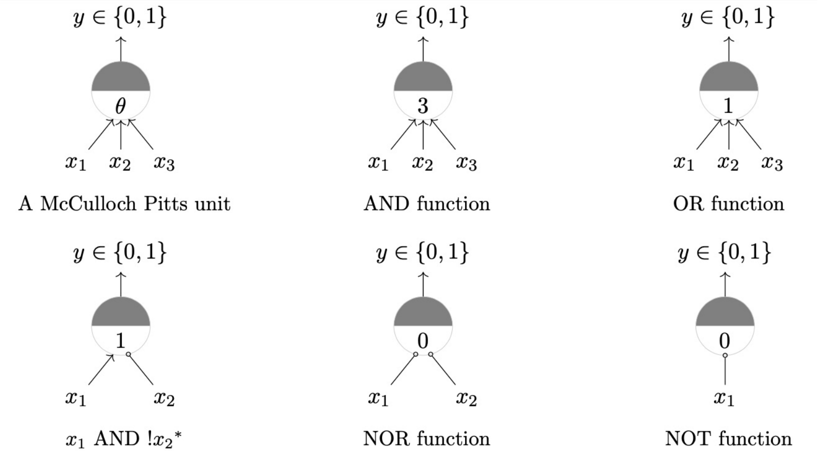


Criticisms:

- Claimed their machine could emulate a Turing machine.
- Did not provide a learning mechanism.

Mathematical Model of a Single Neuron

Simplified Modeling of Boolean Gates:



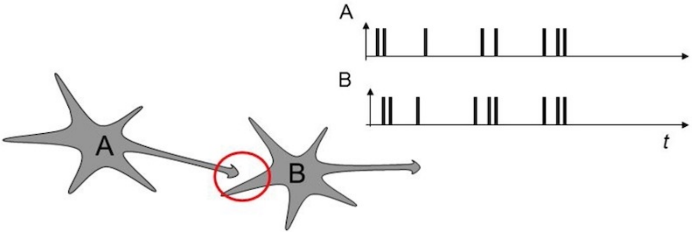
A McCulloch Pitts unit | AND function | OR function

x_1 AND $!x_2$* | NOR function | NOT function

Mathematical Model of a Single Neuron

Hebbian Learning (Donald Hebb, 1949)

When an axon of cell A excites cell B and repeatedly or presistently takes part in firing it, some growth processes or metabolic change takes place in one or both cells so that A's efficiency is increased.



Observation: In other words, neurons that fire together wire together!

Mathematical Model of a Single Neuron

Principles of Hebbian Learning

- Neurons that fire together wire together!
- If neuron x_i repeatedly triggers neuron y, the synaptic knob connecting x_i to y gets larger.
- Mathematically, we can write:

$$w_i = w_i + \eta x_i y \tag{2}$$

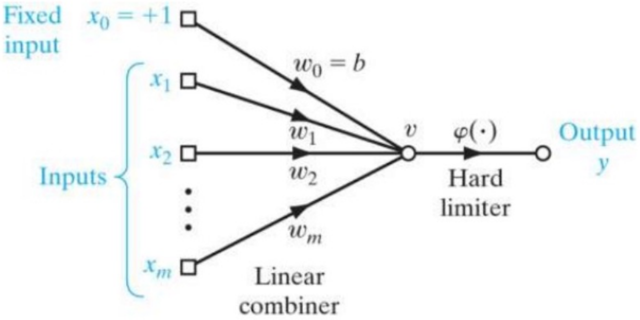
- Here, w_i is the weight of the i-th neuron's input to output neuron y.
- This simple formula is actually the basis of many learning algorithms in machine learning.

Mathematical Model of a Single Perceptron

Perceptron Model (Rosenblatt, 1958)

The simplest form of a neural network consists of a single neuron with adjustable synaptic weights and bias.

A nonlinear neuron consists of a linear combiner followed by a hard limiter.



Mathematical Model of a Single Perceptron

Perceptron Model (Rosenblatt, 1958):

- Learning algorithm:

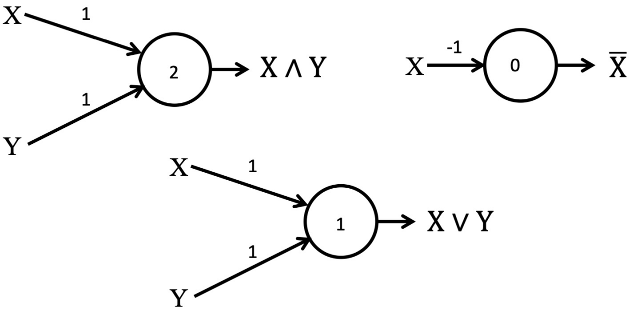
$$w(x) = w(x) + \eta \left(d(x) - y(x)\right) x. \tag{3}$$

Here:

- η is the learning rate,
- $d(x)$ and $y(x)$ are the desired and actual outputs in response to x.
- Update weights whenever the perceptron output is wrong.
- Proved convergence.
- Solution for OR and AND Boolean Gates.

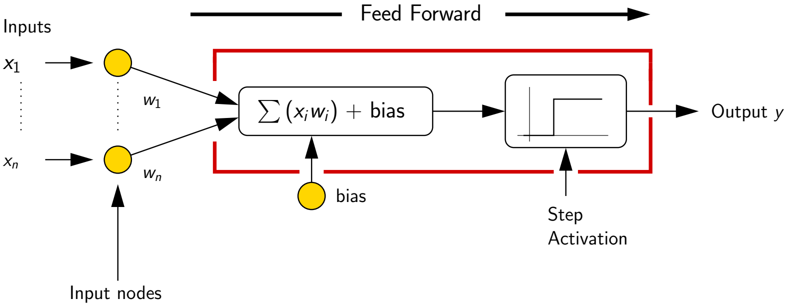
Mathematical Model of a Single Perceptron

Perceptron Model for OR and AND Boolean Gates



Individual elements are weak – no solution for XOR problem.
Networked elements are required.

The Perceptron Model: Forward Propagation



Here:

- Inputs $x_1, x_2, x_3, \cdots x_n$ are real valued.
- Weights $w_1, w_2, w_3, \cdots w_n$ are real valued.
- The output y can also be real valued.

The Perceptron Model: Forward Propagation

Step 1: Linear combiner:

$$z = g(x) = \sum_{i=1}^{n} w_i x_i + \text{bias}. \tag{4}$$

Step 2: Step activation:

$$y = f(z) = \begin{cases} 0, & z < \theta, \\ 1, & z \geq \theta. \end{cases} \tag{5}$$

Here, θ is the threshold parameter.

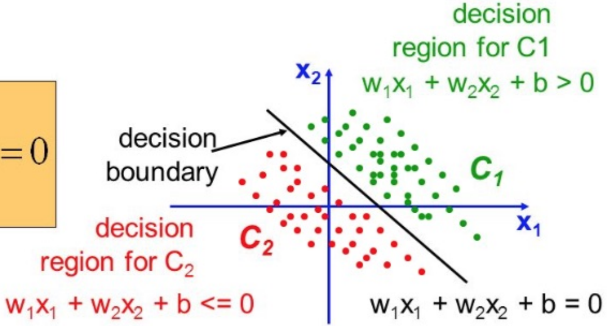
Composition of steps 1 and 2:

$$y = f(g(x)) \tag{6}$$

Perceptron Model as a Linear Classifier

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

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

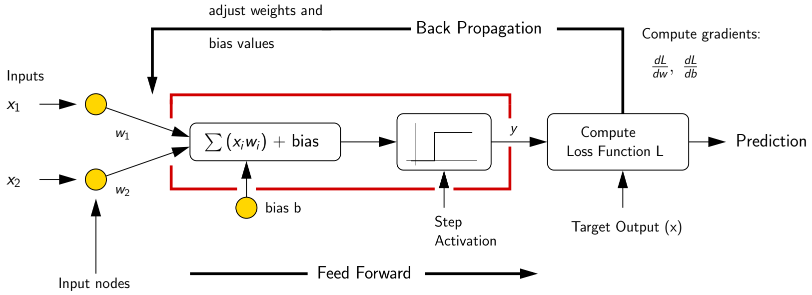


Addition of bias values expands modeling capability. No bias value $\rightarrow$ decision boundary constrained to pass through the origin.

Training a Single Perceptron Model

Training Perceptron Models

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



Input Data: $(x_{11}, x_{21}), (x_{12}, x_{22}), \cdots (x_{1n}, x_{2n})$.

Target Data: target (x_1), $\cdots$ target (x_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,

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

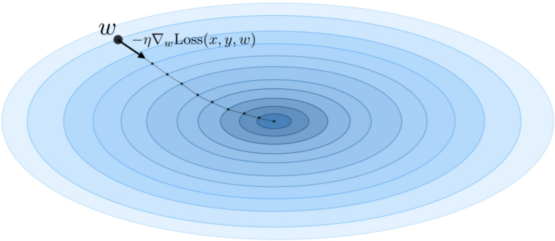
Update Weights: Plug equation 16 into equation 8. Repeat.

Iterations of Learning in Feed Forward Models

Stochastic Gradient Descent

Strategies of neural network learning are iterative.

$$w_{n+1} = w_n - \eta \bigtriangledown_w \text{Loss}(x, y, w). \tag{17}$$



They employ training datasets to update the model.

Metrics of Evaluation

Metrics of Evaluation

Confusion Matrix

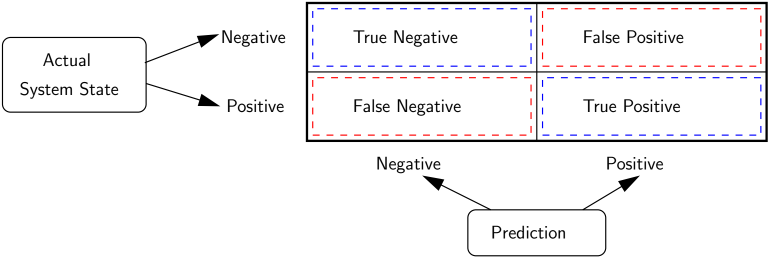
A simple metric to understand performance of a model in terms of predictions and their relationship to the actual state of a system.

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:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{18}$$

Precision:

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

Recall:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{20}$$

F1 Score:

$$\text{F1} = 2\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{21}$$

Metrics of Evaluation

Simple Example

Insert venn diagram ...

Assessment:

Metrics of Evaluation

Confusion Matrix

Insert description and simple example ...

Simple Matrix

Insert simple example ...

Single-Layer

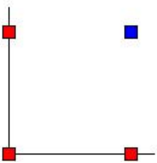
Perceptron Examples

Modeling Boolean Gates

Problem Description:

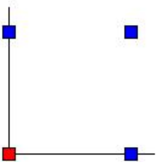
AND

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



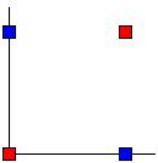
OR

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 Boolean 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 Boolean 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 Boolean 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 Boolean 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 Boolean 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 Boolean 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 Boolean 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 2. 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 2. 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 2. 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 2. Modeling an OR Boolean Gate

DL4J: Trained weights and bias:

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

Decision boundary:

$$f(x_1, x_2) = 6.9568\,(x_1 + x_2) - 3.2378 = 0.0. \tag{22}$$

DL4J: Trained model predictions:

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

Example 2. 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
```

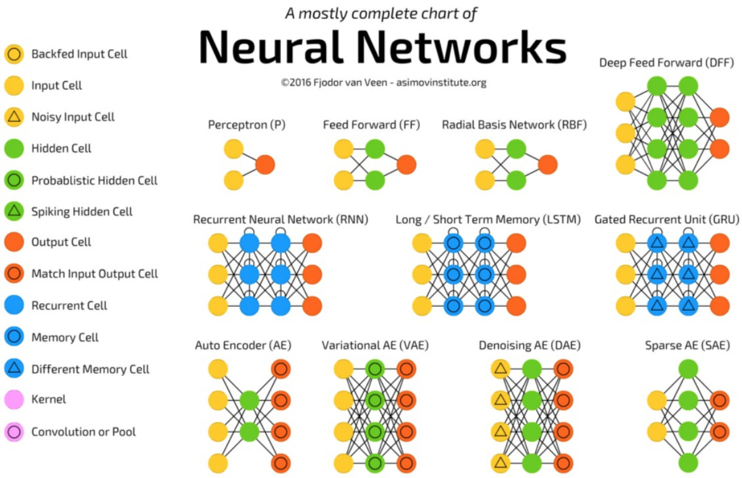
References

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

Appendix A

Chart of Neural Network Architectures

Neural Network Architectures



Neural Network Architectures