Neural Networks II

Mark A. Austin

austin@umd.edu

ENCE 688P, Spring Semester 2021
University of Maryland

March 24, 2021

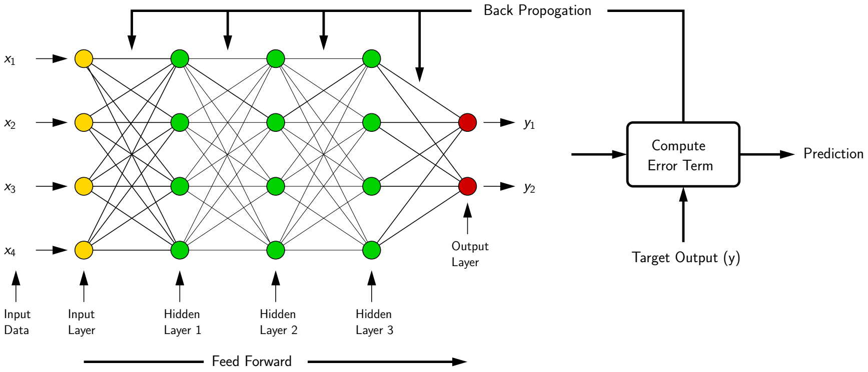
Overview

Part 02

Quick Review

Multilayer Network Training

Training Procedure: Learning the weights and biases to compute a target function (i.e., match the input-output relation of training instances drawn from the target function).



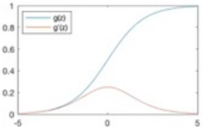
Activation Functions

Activation Function

Activations introduce nonlinearities into the neural network.

Common Activation Functions

Sigmoid Function

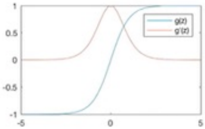


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

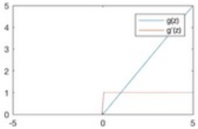


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



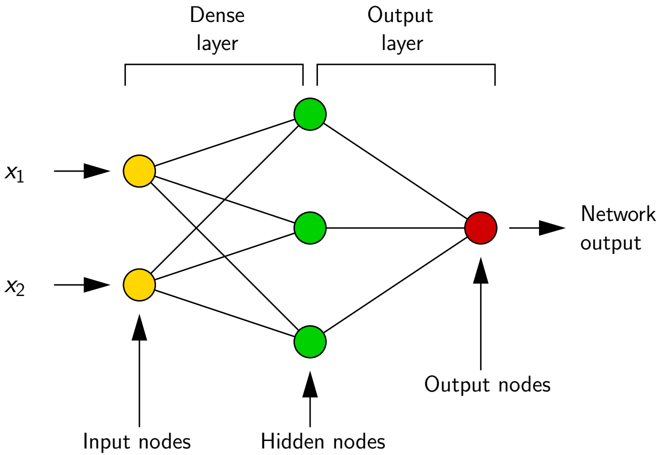
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

```
tf.nn.relu(z)
```

Networks with One Hidden Layer

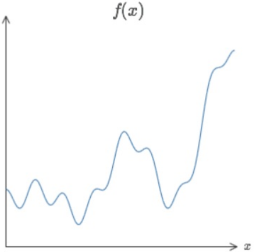
Network Architecture



Universal Approximation Theorem (1989)

Universal Approximation Theorem (1989)

A feed-forward network with a single hidden layer is sufficient to approximate to an arbitrary precision, any continuous functions.

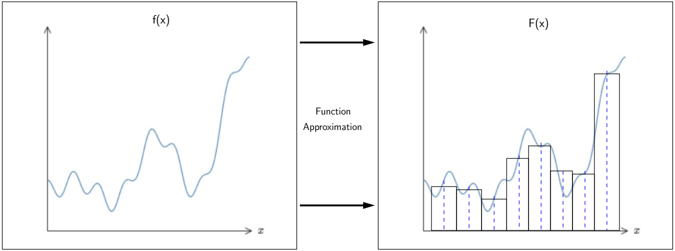


Caveats:

- The number of hidden units may be infeasibly large.
- The resulting model may not generalize.

Universal Approximation Theorem

Continuous Function to Discrete Approximation with Bumps



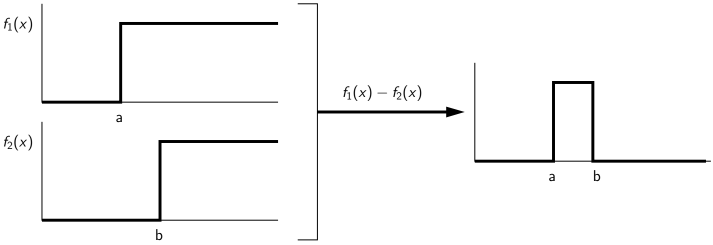
We wish to find:

$$F(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i) \tag{5}$$

such that $|F(x) - f(x)| \leq \epsilon$.

Universal Approximation Theorem

Strategy: We can systematically assemble bump function (i.e., the weighting coefficients v_i) on the interval [a,b] from pairs of steps, i.e.,

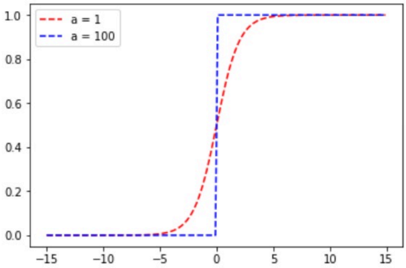


In practice, threshold steps are replaced by a smooth approximation to a step (e.g., sigmoid function).

Function Approximation: Modeling Steps and Bumps

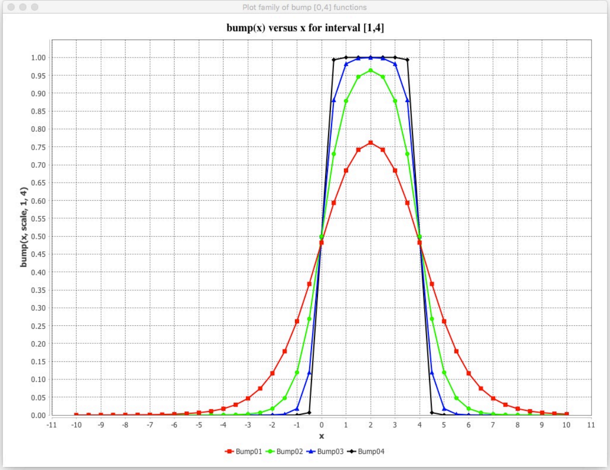
Approximation: Use Sigmoid Functions to Approximate Steps

Let $\sigma(x) = \left[\frac{1}{1+e^{-ax}}\right]$.



The blue dashed curve is a good approximation of step function.

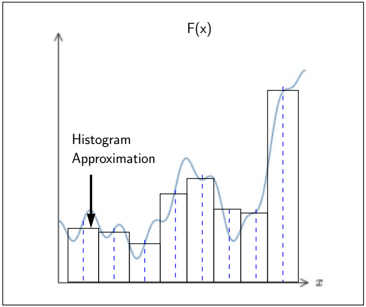
Function Approximation: Modeling Steps and Bumps



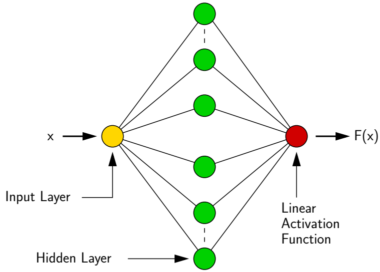
Function Approximation: Neural Network Model

Neural Network Model Approximation of Histogram

Histogram Approximation



Neural Network Model



Note

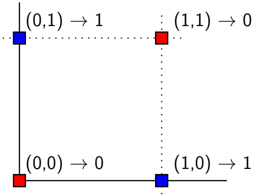
- Use linear activation function to get combined sum of individual bump functions.

Examples

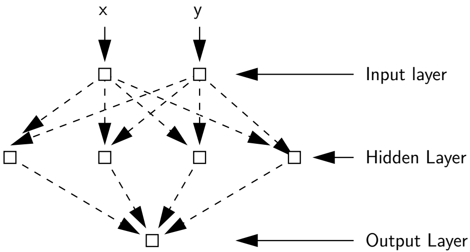
Example 4. Modeling an XOR Boolean Gate

Problem Description: XOR Boolean Gate creates a non-separable problem space:

XOR Problem Space



Neural Network Architecture (2 inputs, 4 hidden nodes)



Example 4. Modeling an XOR Boolean Gate

Gate Behavior:

```
* +---------+---------+-------------+
* | Input 1 | Input 2 | Output XOR  |
* +---------+---------+-------------+
* |    0    |    0    |      0      |
* +---------+---------+-------------+
* |    1    |    0    |      1      |
* +---------+---------+-------------+
* |    0    |    1    |      1      |
* +---------+---------+-------------+
* |    1    |    1    |      0      |
* +---------+---------+-------------+
```

Network Model: 2 input-neurons, 1 hidden-layer with 4 (or 2) hidden-neurons, 1 output-neuron.

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Manual Implementation (pg. 1).

```
# ==================================================================
# TestMultiLayer-XOR-Gate01.py: Train and test boolean XOR gate
# with one hidden layer.
#
# 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 ...

def main():

    # Define input features and targe output:
```

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Manual Implementation (pg. 2).

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

    # Reshaping target output into vector:

    target_output = target_output.reshape(4,1)

    # Initial weights and learning parameters ..

    weight_hidden = np.random.rand(2,4)
    weight_output = np.random.rand(4,1)

    lr = 0.10 # Learning Rate:

    # Train neural network, 300,000 epochs ...

    for epoch in range(300001):

        # Input for hidden layer :

        input_hidden = np.dot(input_features, weight_hidden)

        # Output from hidden layer :

        output_hidden = sigmoid(input_hidden)

        # Input for output layer :
```

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Manual Implementation (pg. 3).

```
        input_op = np.dot(output_hidden, weight_output)

        # Output from output layer :

        output_op = sigmoid(input_op)

        # Phase 1: Compute derivatives for phase 1 ...

        error_out = ((1 / 2) * (np.power((output_op - target_output), 2)))

        # Derivatives for phase 1 :

        derror_douto = output_op - target_output
        douto_dino   = sigmoid_der(input_op)
        dino_dwo     = output_hidden

        derror_dwo = np.dot(dino_dwo.T, derror_douto * douto_dino)

        # Phase 2: Compute derivatives for phase 2 ...

        derror_dino  = derror_douto * douto_dino
        dino_douth   = weight_output
        derror_douth = np.dot(derror_dino , dino_douth.T)

        douth_dinh = sigmoid_der(input_hidden)
        dinh_dwh   = input_features
        derror_dwh = np.dot(dinh_dwh.T, douth_dinh * derror_douth)
```

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Manual Implementation (pg. 4).

```

        # Update weights in hidden and output layers ...")

        weight_hidden -= lr * derror_dwh
        weight_output -= lr * derror_dwo

    # Print Results ...

    print("--- Final weights in hidden layer:");

    print (weight_hidden)

    print("--- Final weights in output layer:");

    print (weight_output)

    print("--- Use trained network to predict values ... ");
    print("--- Verify input [0,0] --> 0 ... ");

    single_point = np.array([0,0])

    result1 = np.dot(single_point, weight_hidden)
    result2 = sigmoid(result1)
    result3 = np.dot( result2, weight_output)
    result4 = sigmoid(result3)

    print ("--- Result: %f ..." %(result4))
```

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Manual Implementation (pg. 5).

```
    ... lines of code removed ...

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

    single_point = np.array([1,1])

    result1 = np.dot(single_point, weight_hidden)
    result2 = sigmoid(result1)
    result3 = np.dot(result2, weight_output)
    result4 = sigmoid(result3)

    print ("--- Result: %f ..." %(result4))

# call the main method ...

main()
```

Example 4. Modeling an XOR Boolean Gate

Python + NumPy: Abbreviated Results.

```
--- Final weights in hidden layer:

[[    7.5956455  -2.67775051 -3.71266848  5.49406133]
 [  -3.65519292  -2.85282227  7.74148801  5.54965051] ]

--- Final weights in output layer:

[[ -10.42115829 ][ -4.44622987 ][ -10.46429239 ][ 15.76218393 ]]

--- Use trained network to make predictions ...

--- Verify input [0,0] --> 0, result: 0.008287 ...
--- Verify input [1,0] --> 1, result: 0.991392 ...
--- Verify input [0,1] --> 1, result: 0.991305 ...
--- Verify input [1,1] --> 0, result: 0.008498 ...
```

Example 4. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 1)

```
# ===============================================================
# TestKeras-XOR-Problem.py: Use Keras to solve XOR gate problem.
#
# Written by: Mark Austin                          November 2020
# ===============================================================

import numpy as np
from tensorflow import keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation

# main method ...

def main():
    print("--- Enter TestKeras-XOR-Problem1.main()      ... ");
    print("--- ======================================= ... ");

    print("--- Training data ...")

    training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
    print(training_data)

    print("--- Target data for XOR problem ...")

    target_data = np.array([[0],[1],[1],[0]], "float32")
    print(target_data)
```

Example 4. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 2)

```
print("--- Define dense layers ...")

layer1 = Dense(2, input_dim =2, activation = 'sigmoid')
layer2 = Dense(1, activation = 'sigmoid' )

print("--- Assemble Sequential model ...")

model = Sequential()
model.add( layer1 )
model.add( layer2 )

print("--- Compile model ...")

model.compile( loss='mean_squared_error',
               optimizer='adam', metrics=['binary_accuracy'])

print("--- Train model to fit data ...")

model.fit( training_data, target_data, epochs=20000, verbose=2)

# Retrieve and print layer weights and bias values ...

np.set_printoptions(formatter={'float': '{: 0.5f}'.format})

print("--- Layer 1: weights and bias ...")

print( layer1.get_weights()[0] )
```

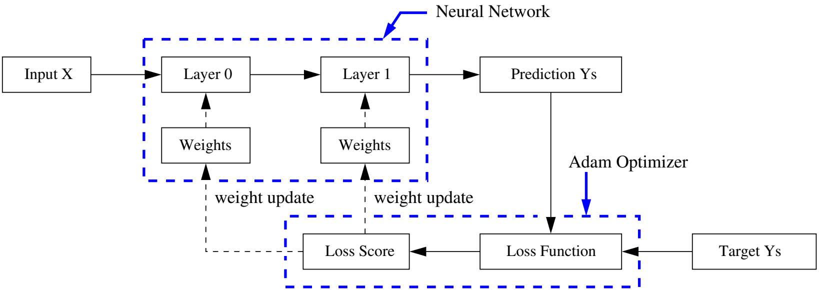
Example 4. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 3)

```
    print("--- Layer 2: weights and bias ...")

    print ( layer2.get_weights()[0] )
    print ( layer2.get_weights()[1] )

    print("--- Trained model predictions ...")

    print( model.predict(training_data) )

    print("--- Rounded model predictions ...")

    print( model.predict(training_data).round())

    print("--- ======================================= ... ");
    print("--- Finished!! ... ");

# call the main method ...

main()
```

Example 4. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Solution Procedure with Adam Optimizer.



The Adam optimization algorithm (2015) is an extension of stochastic gradient descent.

Example 4. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Abbreviated Results

```
=====================================================================
Layer 1: weights and bias values

Weights:      [ [ 9.19988  5.42391  4.75831 -5.60893]
                [-6.43348 -7.64395  6.11223  7.93375]]
Bias values: [   1.93753 -3.06313 -0.28112  3.10157]

Layer 2: weights and bias values

Weights:      [ [-11.47534] [ 7.33922] [ 7.02013]  [-6.50343] ]
Bias values: [ 5.22003]
---------------------------------------------------------------------

Trained prediction: [ [ 0.00046]  [ 0.99628] [ 0.99900] [ 0.00380] ]
Rounded prediction: [ [ 0.00000]  [ 1.00000] [ 1.00000] [ 0.00000] ]
=====================================================================
```

Example 4. Modeling an XOR 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[]{0,1,1,0};
      INDArray output01 = Nd4j.create(vectorDouble).transpose();

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

DL4J: MultiLayer Configuration with 2 layers:

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

   MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .updater(new Sgd(0.1))
        .seed(seed)
        .biasInit(0) // Init the bias with 0 - empirical value, too
        .miniBatch(false)
```

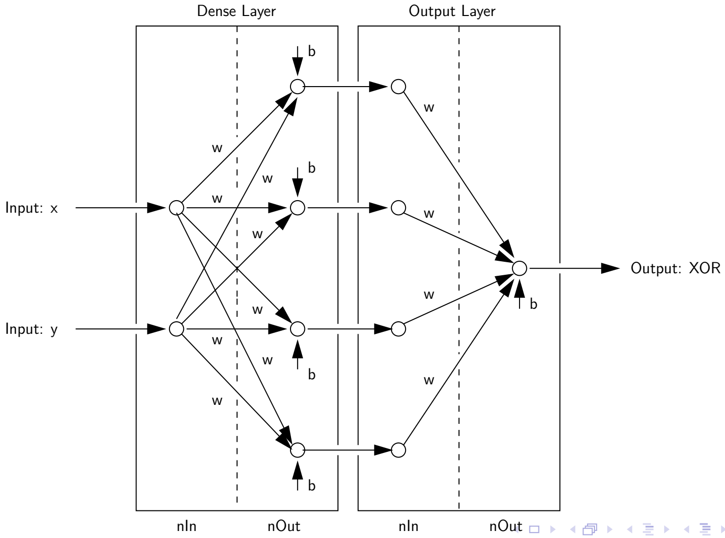
Example 4. Modeling an XOR Boolean Gate

DL4J: Create Network Configuration (cont'd):

```
            .list()
            .layer(new DenseLayer.Builder()
               .nIn(2)
               .nOut(2)
               .activation(Activation.SIGMOID)
               .weightInit( new UniformDistribution(0, 1) )
               .build())
            .layer(new OutputLayer.Builder( LossFunctions.LossFunction.MSE )
               .nIn(2)
               .nOut(1)
               .activation(Activation.IDENTITY)
               .weightInit(new UniformDistribution(0, 1))
               .build())
            .build();

        // Create multilayer network ...

        MultiLayerNetwork net = new MultiLayerNetwork(conf);
        net.init();
        net.setListeners(new ScoreIterationListener(500));
```

Example 4. Modeling an XOR Boolean Gate

DL4J: Network Model (nIn, nOut)



Example 4. Modeling an XOR Boolean Gate

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

```
LayerName (LayerType)   nIn,nOut   TotalParams   ParamsShape
=================================================================
   layer0 (DenseLayer)      2,4        12          W:{2,4}, b:{1,4}
   layer1 (OutputLayer)     4,1         5          W:{4,1}, b:{1,1}
-----------------------------------------------------------------
   Total Parameters:  17   Trainable Parameters:  17
=================================================================
```

DL4J: Train the network for 10,000 epochs:

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

Example 4. Modeling an XOR Boolean Gate

DL4J: Trained model predictions:

```
Trained Model Predictions: [-0.000001,  1.0000,  1.0000, -0.000001 ]
Rounded Model Predictions: [ 0.000000,  1.0000,  1.0000,  0.000000 ]
```

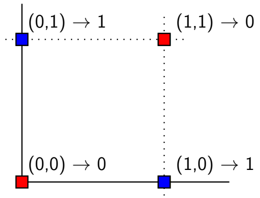
DL4J: Evaluation metrics:

```
 ========================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") onl
```

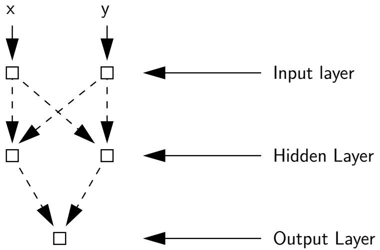
Example 4. Modeling an XOR Boolean Gate

Minimal Network Architecture:

XOR Problem Space



Neural Network Architecture (2 inputs, 2 hidden nodes)



Hidden layer can be modeled with 4 neurons. But strictly speaking, we only need 2 to work.

Example 4. Modeling an XOR Boolean Gate

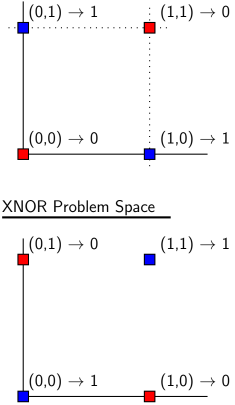
Summary of Results: TensorFlow 2 + Keras

```
4 nodes on hidden layer          2 nodes on hidden layer
=============================================================

Trained model predictions        Trained model predictions
[[ 0.00046]                      [[ 0.00428]
 [ 0.99628]                        [ 0.99536]
 [ 0.99900]                        [ 0.99529]
 [ 0.00380]]                       [ 0.00509]]

Rounded model predictions        Rounded model predictions
[[ 0.00000]                      [[ 0.00000]
 [ 1.00000]                        [ 1.00000]
 [ 1.00000]                        [ 1.00000]
 [ 0.00000]]                       [ 0.00000]]

=============================================================
```

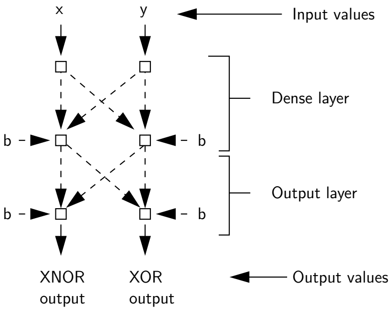
Example 5. Modeling XNOR and XOR Boolean Gates

Network Architecture:

XOR Problem Space



Neural Network Architecture (2 inputs, 2 hidden nodes)



Example 5. Modeling XNOR and XOR Boolean Gates

Gate Behavior:

```
* +---------+---------+-------------+------------+
* | Input 1 | Input 2 | Output XNOR | Output XOR |
* +---------+---------+-------------+------------+
* |    0    |    0    |      1      |     0      |
* +---------+---------+-------------+------------+
* |    1    |    0    |      0      |     1      |
* +---------+---------+-------------+------------+
* |    0    |    1    |      0      |     1      |
* +---------+---------+-------------+------------+
* |    1    |    1    |      1      |     0      |
* +---------+---------+-------------+------------+
```

DL4J Implementation: 2 input-neurons, 1 hidden-layer with 2 (or 4) hidden-neurons, 2 output-neurons.

Example 5. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 1)

```
# ===================================================================
# TestKeras-XNOR-XOR-Problem.py: Keras implementation of [ XNOR, XOR ]
# neural net.
#
# Written by: Mark Austin                                November 2020
# ===================================================================

import numpy as np
from tensorflow import keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

# main method ...

def main():
    print("--- Enter TestKeras-XNOR-XOR-Problem.main()   ... ");
    print("--- ======================================== ... ");

    print("--- Training data ...")

    training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
    print(training_data)

    print("--- Target data for XNOR, XOR problem ...")

    target_data = np.array([[1,0],[0,1],[0,1],[1,0]], "float32")
```

Example 5. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 2)

```
    print(target_data)

    print("--- Define dense layers ...")

    layer1 = Dense(4, input_dim =2, activation = 'sigmoid')
    layer2 = Dense(2, activation = 'sigmoid' )

    print("--- Assemble Sequential model ...")

    model = Sequential()
    model.add( layer1 )
    model.add( layer2 )

    print("--- Compile model ...")

    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy

    print("--- Train model to fit data ...")

    model.fit( training_data, target_data, epochs=20000, verbose=2)

    # Retrieve and print layer weights and bias values ...

    np.set_printoptions(formatter={'float': '{: 0.5f}'.format})

    print("--- Layer 1: weights ...")
```

Example 5. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 3)

```
    print( layer1.get_weights()[0] )

    print("--- Layer 1: bias values ...")

    print( layer1.get_weights()[1] )

    print("--- Layer 2: weights ...")

    print ( layer2.get_weights()[0] )

    print("--- Layer 2: bias values ...")

    print ( layer2.get_weights()[1] )

    print("--- Trained model predictions ...")

    print( model.predict(training_data) )

    print("--- Rounded model predictions ...")

    print( model.predict(training_data).round())

    print("--- ======================================== ... ");
    print("--- Finished!! ... ");

# call the main method ...

main()
```

Example 5. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: 4 nodes on hidden layer:

```
=================================================================
Layer 1 weights: [ [ 6.72854 -6.70459  7.59369 -7.70765]
                   [-9.43216 -5.13656 -5.93510  5.67913] ]
Layer 1 biases:    [-2.14363  0.92774  3.59704 -3.29682]

Layer 2: weights: [ [-11.12897 11.41488] [ 4.90443 -4.64064]
                    [ 6.27834  -6.88757] [-7.29352  6.94774] ]
Layer 2 biases:   [  -0.71356   1.17832]
=================================================================
```

TensorFlow 2 + Keras: 2 nodes on hidden layer:

```
=================================================================
Layer 1 weights: [ [ 8.82153   9.08939] [ -9.03718  -9.08577] ]
Layer 1 biases:    [ 4.90436  -4.88207]

Layer 2 weights: [ [11.61453 -11.17453] [-11.53318  11.23555] ]
Layer 2 biases:    [-5.82071   5.56507]
=================================================================
```

Example 5. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Abbreviated Results (20,000 epochs)

```
4 nodes on hidden layer           2 nodes on hidden layer
=================================================================


Trained model predictions         Trained model predictions
[[ 0.99944  0.00061]              [[ 0.99639  0.00431]
 [ 0.00115  0.99896]               [ 0.00355  0.99545]
 [ 0.00433  0.99623]               [ 0.00380  0.99577]
 [ 0.99555  0.00387]]              [ 0.99632  0.00440]]

Rounded model predictions         Rounded model predictions
[[ 1.00000  0.00000]              [[ 1.00000  0.00000]
 [ 0.00000  1.00000]               [ 0.00000  1.00000]
 [ 0.00000  1.00000]               [ 0.00000  1.00000]
 [ 1.00000  0.00000]]              [ 1.00000  0.00000]]

=================================================================
```

Example 5. Modeling XNOR and XOR Boolean Gates

DL4J: Create training dataset:

```
INDArray input = Nd4j.zeros(4, 2);     INDArray labels = Nd4j.zeros(4, 2);

input.putScalar(new int[]{0, 0}, 0);   labels.putScalar(new int[]{0, 0}, 1);
input.putScalar(new int[]{0, 1}, 0);   labels.putScalar(new int[]{0, 1}, 0);

input.putScalar(new int[]{1, 0}, 1);   labels.putScalar(new int[]{1, 0}, 0);
input.putScalar(new int[]{1, 1}, 0);   labels.putScalar(new int[]{1, 1}, 1);

input.putScalar(new int[]{2, 0}, 0);   labels.putScalar(new int[]{2, 0}, 0);
input.putScalar(new int[]{2, 1}, 1);   labels.putScalar(new int[]{2, 1}, 1);

input.putScalar(new int[]{3, 0}, 1);   labels.putScalar(new int[]{3, 0}, 1);
input.putScalar(new int[]{3, 1}, 1);   labels.putScalar(new int[]{3, 1}, 0);

DataSet ds = new DataSet(input, labels);
```

DL4J: Create Network Configuration with 2 layers:

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

NeuralNetConfiguration.Builder builder = new NeuralNetConfiguration.Builder();
builder.updater(new Sgd(0.1));
builder.seed(123);
builder.biasInit(0);
builder.miniBatch(false);
```

Example 5. Modeling XNOR and XOR Boolean Gates

DL4J: Create Network Configuration (cont'd):

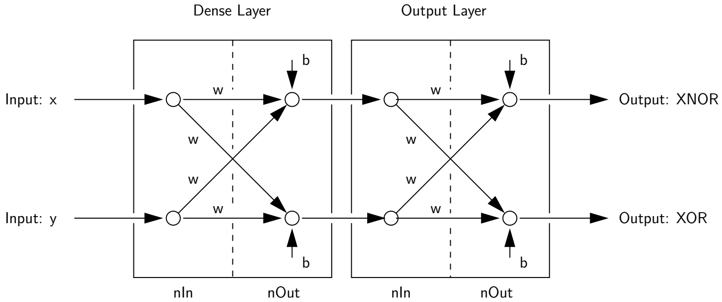
```
// Create dense layer with 2 input connections ...

DenseLayer.Builder hiddenLayerBuilder = new DenseLayer.Builder();
hiddenLayerBuilder.nIn(2);
hiddenLayerBuilder.nOut(2);
hiddenLayerBuilder.activation(Activation.SIGMOID);
hiddenLayerBuilder.weightInit(WeightInit.DISTRIBUTION);

// Create output layer with 2 output connections ...

Builder outputLayerBuilder = new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVE
outputLayerBuilder.nIn(2);
outputLayerBuilder.nOut(2);
outputLayerBuilder.activation(Activation.SOFTMAX);
outputLayerBuilder.dist(new UniformDistribution(0, 1));

// Create list of layers in network configuration ...

ListBuilder listBuilder = builder.list();
listBuilder.layer(0, hiddenLayerBuilder.build());
listBuilder.layer(1, outputLayerBuilder.build());
listBuilder.pretrain(false);
listBuilder.backprop(true);
```

Example 5. Modeling XNOR and XOR Boolean Gates

DL4J: Create Network Configuration (cont'd):

```
// Build and check the network configuration ...

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

DL4J: Network Model (nIn, nOut)



Example 5. Modeling XNOR and XOR Boolean Gates

DL4J: Summary of Network Model (2 nodes on hidden layer)

```
LayerName (LayerType)    nIn,nOut    TotalParams    ParamsShape
=================================================================
   layer0 (DenseLayer)      2,2         6           W:{2,2}, b:{1,2}
   layer1 (OutputLayer)     2,2         6           W:{2,2}, b:{1,2}
-----------------------------------------------------------------
   Total Parameters:   12   Trainable Parameters:   12
=================================================================
```

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

```
LayerName (LayerType)    nIn,nOut    TotalParams    ParamsShape
=================================================================
   layer0 (DenseLayer)      2,4         12          W:{2,4}, b:{1,4}
   layer1 (OutputLayer)     4,2         10          W:{4,2}, b:{1,2}
-----------------------------------------------------------------
   Total Parameters:   22   Trainable Parameters:   22
=================================================================
```

Example 5. Modeling XNOR and XOR Boolean Gates

DL4J: Train the network for 10,000 epochs:

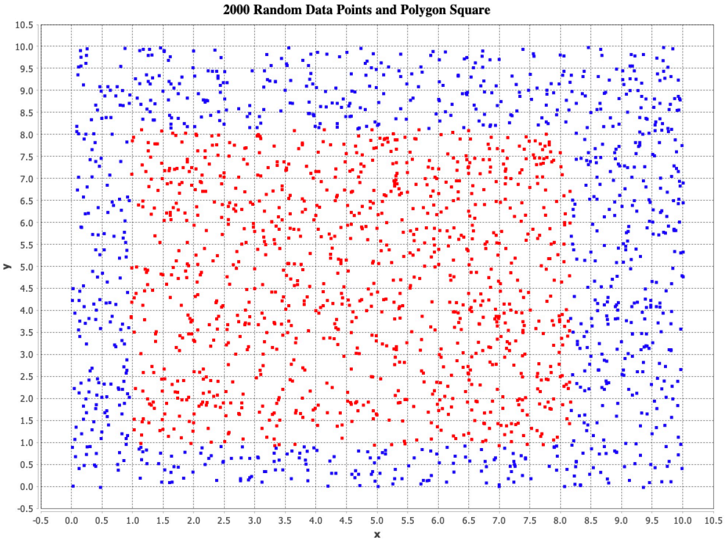
```
    for( int i=0; i <= 10000; i++ ) {
       net.fit(ds);
    }
```

DL4J: Trained model predictions:

```
------------------------------------------------------------
Hidden Layer (2 nodes)                Hidden Layer (4 nodes)
============================================================
[[ 0.9984,    0.0016],                [[ 0.9993,    0.0007],
 [ 0.0012,    0.9988],                 [ 0.0018,    0.9982],
 [ 0.0012,    0.9988],                 [ 0.0004,    0.9996],
 [ 0.9987,    0.0013]]                 [ 0.9987,    0.0013]]
=============================================================
```

Legend: Column 1 $\rightarrow$ XNOR output, Column 2 $\rightarrow$ XOR output.

Example 6. Points in Convex Polygon



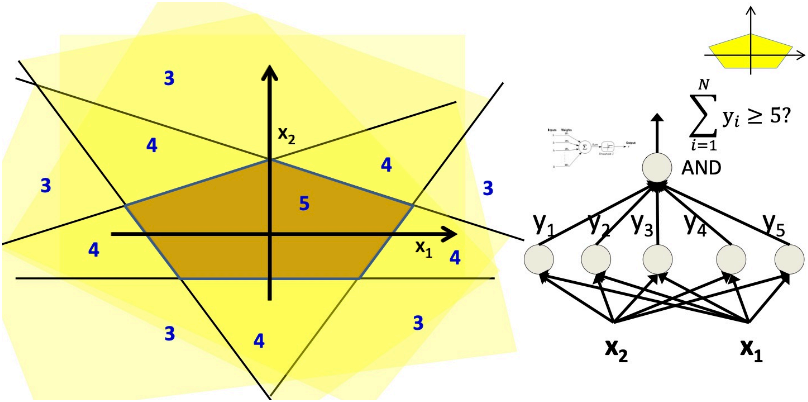
Example 6. Points in Convex Polygon

Problem Description:

- Consider a 10×10 grid containing a smaller rectangle that has area approximately equal to 50.
- If a point (x,y) is selected at random within the grid, then there is approximately a 50% chance that the point will be inside the smaller rectangle. And there is a 50% chance it will be outside.
- We wish to train a neural network to determine whether or not a specific coordinate is inside or outside the smaller polygon.
- 2,000 coordinate points are generated at random. The red dots lie outside the smaller rectangle; the blue dots are inside.
- This is the training data for our neural network.

Composition of Decision Boundaries

Network will fire when an input point is inside Region 5.



Source: Bhiksha, 2018.

Example 6. Points in Convex Polygon

DL4J: Read training dataset ...

```
// Read polygon square-shaped data ...

double[][] x = DataUtils.readInputsFromFile( "data/polygon-square-data.txt");
double[][] t = DataUtils.readInputsFromFile( "data/polygon-square-outcome.txt");
```

DL4J: Scale training dataset to [0,1] range ...

```
// Scale coordinates from [0,10] --> [0,1] ....

for (int i = 0; i < x.length; i = i + 1 ) {
   x[i][0] = x[i][0]/10.0;
   x[i][1] = x[i][1]/10.0;
}
```

Scaling the dataset from [0,10] range to [0,1] range helps to avoid vanishing gradient problem.

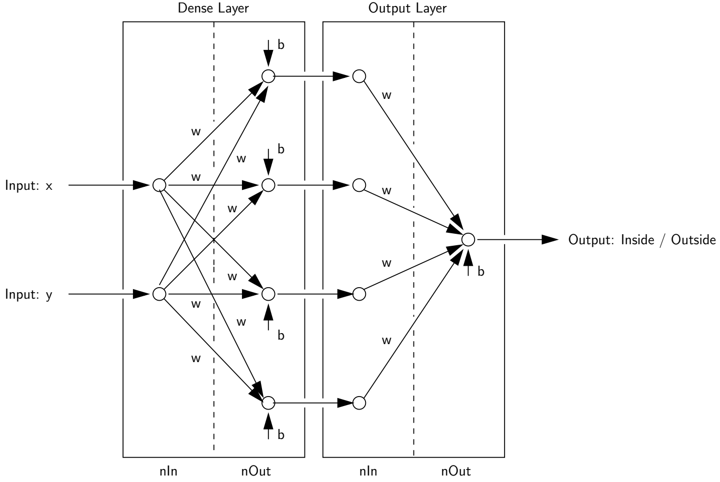
Example 6. Points in Convex Polygon

DL4J: Create Network Configuration:

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .updater(new Sgd(0.01))
        .seed(seed)
        .biasInit(0) // Init the bias with 0 - empirical value, too
        .miniBatch(false)
        .list()
        .layer( new DenseLayer.Builder()
                .nIn(2)
                .nOut(8)
                .activation(Activation.SIGMOID)
                .weightInit(WeightInit.DISTRIBUTION)
                .build())
        .layer( new OutputLayer.Builder(LossFunctions.LossFunction.MSE )
                .nIn(8)
                .nOut(1)
                .activation(Activation.SIGMOID)
                .weightInit(WeightInit.DISTRIBUTION)
                .build())
        .pretrain(false)
        .backprop(true)
        .build();

MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.init();
net.setListeners(new ScoreIterationListener(250));
```

Example 6. Points in Convex Polygon

DL4J: Network Model (nIn, nOut)



Example 6. Points in Convex Polygon

DL4J: Training the Network (2 nodes on hidden layer) ...

```
15:41:43.476 Score at iteration 0 is 498.15325927734375

.... lines of output removed ....

15:42:36.933 Score at iteration 49750 is 291.03997802734375
15:42:37.199 Score at iteration 50000 is 291.0295104980469
```

DL4J: Training the Network (4 nodes on hidden layer) ...

```
15:46:59.732 Score at iteration 0 is 242.2577667236328

.... lines of output removed ....

15:47:06.647 Score at iteration 9900 is 3.6843810081481934
15:47:06.711 Score at iteration 10000 is 3.668752670288086
```

Example 6. Points in Convex Polygon

DL4J: Weights and Bias Values (4 nodes on hidden layer)

```
==================================================================
Layer 0 weights: [[   -0.3276,   59.2762,   68.9790,   -0.1253],
                  [  -51.5552,    0.1561,    0.6804,  -60.4115]]
Layer 0 biases:  [[   42.7406,  -49.2985,   -6.1249,    4.7453]]
------------------------------------------------------------------
Layer 1 weights: [    28.5693,  -34.2605,   28.7222,  -37.4116 ]
Layer 1 biases:       -46.9808
==================================================================
```

Decision boundary equations on $[1 \times 1]$ grid:

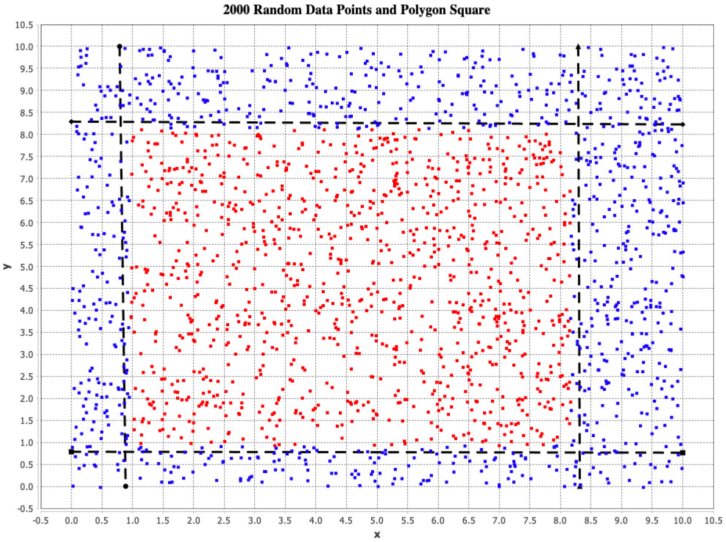
$$
\begin{aligned}
f_1(x_1, x_2) &= -0.3x_1 - 51.5x_2 + 42.7 = 0 \\
f_2(x_1, x_2) &= 59.3x_1 + 0.15x_2 - 49.3 = 0 \\
f_3(x_1, x_2) &= 69.0x_1 + 0.68x_2 - 6.12 = 0 \\
f_4(x_1, x_2) &= -0.12x_1 - 60.4x_2 + 4.74 = 0
\end{aligned} \tag{6}
$$

Example 6. Points in Convex Polygon

Decision boundary equations scaled to [10 × 10] grid:

$$\begin{aligned}
f_1(x_1, x_2) &= 0 \quad && \rightarrow x_2 \approx 427/51.5 = 8.2. \\
f_2(x_1, x_2) &= 0 \quad && \rightarrow x_1 \approx 493/59.3 = 8.3. \\
f_3(x_1, x_2) &= 0 \quad && \rightarrow x_1 \approx 61/69 = 0.9. \\
f_4(x_1, x_2) &= 0 \quad && \rightarrow x_2 \approx 47.4/60.1 = 0.8.
\end{aligned} \tag{7}$$

Example 6. Points in Convex Polygon



Example 6. Points in Convex Polygon

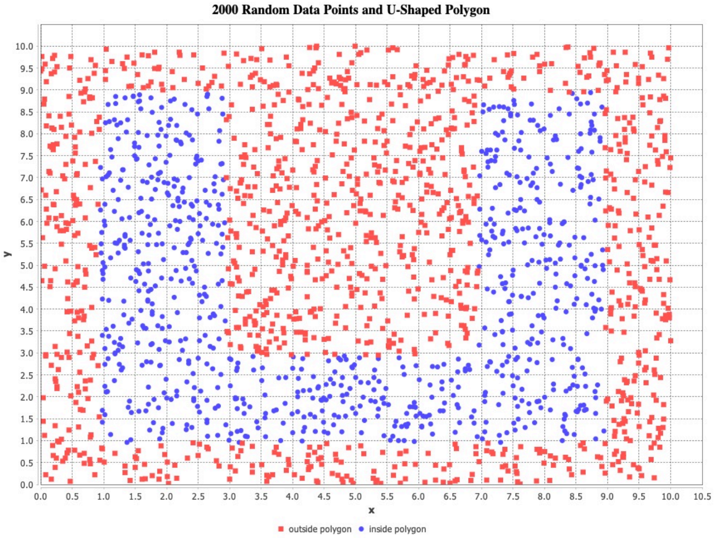
DL4J: Evaluation Metrics (2, 4 and 6 nodes on hidden layers)

Metric	2 nodes	4 nodes	6 nodes
Accuracy:	0.8015	0.9965	0.9955
Precision:	0.8427	0.9965	0.9953
Recall:	0.7924	0.9965	0.9957
F1 Score:	0.8371	0.9967	0.9957

DL4J: Confusion Matrix (2, 4 and 6 nodes on hidden layers)

```
    2 nodes              4 nodes              6 nodes
==========================================================
     0    1               0    1               0    1
-------------        -------------        -------------
   583  366 |          945    4 |          948    1 | 0 = 0
    31 1020 |            3 1048 |            8 1043 | 1 = 1
==========================================================
```

Counter Example 7. Points in Non-Convex Polygon



Counter Example 7. Points in Non-Convex Polygon

Problem Description:

- Consider a 10×10 grid containing a smaller U-shaped polygon.
- We wish to train a neural network to determine whether or not a specific coordinate is inside or outside the letter U shape.
- 2,000 coordinate points are generated at random. The red dots lie outside the U-shaped polygon; the blue dots are inside.
- This is the training data for our neural network.
- Notice that the U-shaped polygon is non-convex. Hence, if the network model contains only one hidden layer then this model should fail.

Counter Example 7. Points in Non-Convex Polygon

DL4J: Training the Network (4 nodes on hidden layer) ...

```
10:26:44.531 Score at iteration 0 is 548.747314453125

  ... lines of output removed ...

10:27:48.434 Score at iteration 49750 is 295.26971435546875
10:27:48.724 Score at iteration 50000 is 295.2453308105469
```

DL4J: Training the Network (8 nodes on hidden layer) ...

```
10:32:00.784 Score at iteration 0 is 1062.9676513671875

... lines of output removed ...

10:33:14.822 Score at iteration 49750 is 528.657958984375
10:33:15.163 Score at iteration 50000 is 528.65771484375
```

Counter Example 7. Points in Non-Convex Polygon

DL4J: Evaluation Metrics (4 and 8 nodes on hidden layer)

```
   Metric              4 nodes      8 nodes
==========================================
   Accuracy:            0.7470       0.5635  <--- Very poor accuracy.
   Precision:           0.8112       0.5896
   Recall:              0.6926       0.5885
   F1 Score:            0.5675       0.5697
```

DL4J: Confusion Matrix (4 and 8 nodes on hidden layer)

```
    4 nodes             8 nodes
=======================================
     0    1              0    1
--------------       -------------
  1162   32   |        549   645 | 0 = 0  <--- As expected,
   474  332   |        228   578 | 1 = 1       it fails!
=======================================
```

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.
- Sun J., Fundamental Belief: Universal Approximation Theorems, Computer Science and Engineering, University of Minnesota, Twin Cities, 2020.