Neural Networks III

Mark A. Austin

austin@umd.edu

ENCE 688P, Spring Semester 2026
University of Maryland

May 26, 2026

Overview

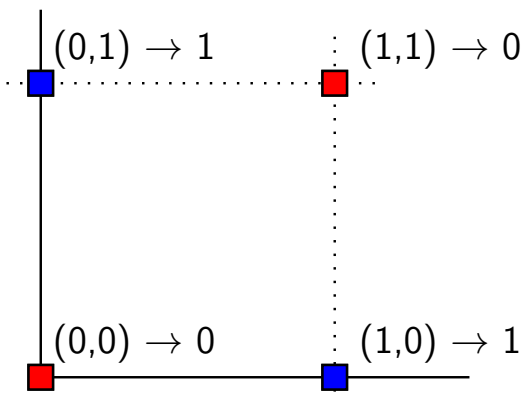
Networks with

One Hidden Layer

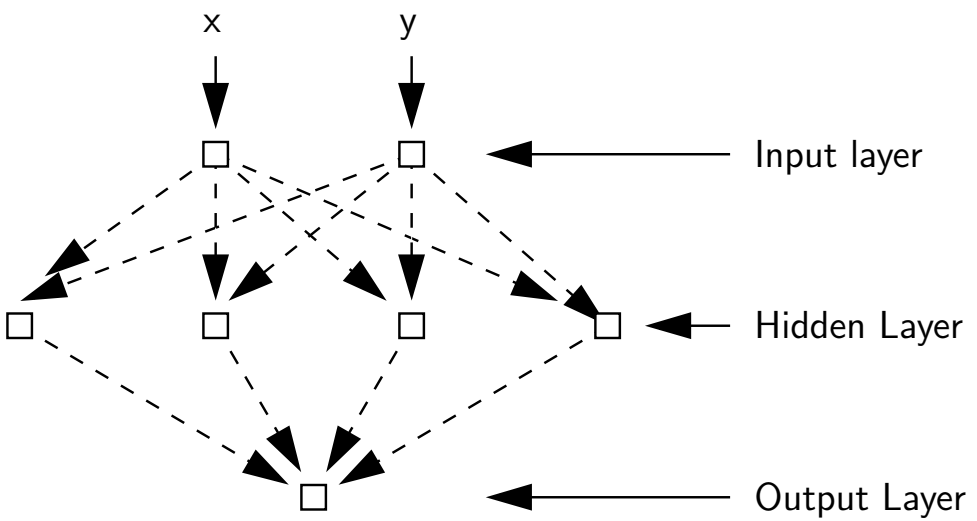
Example 3. Modeling an XOR Boolean Gate

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

XOR Problem Space



Neural Network Architecture (2 inputs, 4 hidden nodes)



Example 3. 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 3. Modeling an XOR Boolean Gate

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

```
1   # ============================================================
2   # TestMultiLayer-XOR-Gate01.py: Train and test boolean XOR gate
3   # with one hidden layer.
4   #
5   # Modified by: Mark Austin                          October, 2020
6   # ============================================================
7
8   import math
9   import matplotlib
10  import matplotlib.pyplot as plt
11  import numpy as np
12
13  # Define Sigmoid function:
14
15  def sigmoid(x):
16      return 1/(1+np.exp(-x))
17
18  # Define derivative of Sigmoid function:
19
20  def sigmoid_der(x):
21      return sigmoid(x)*(1-sigmoid(x))
22
23  # main method ...
24
25  def main():
26
27      # Define input features and targe output:
```

Example 3. Modeling an XOR Boolean Gate

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

```
28      input_features = np.array( [[0,0],[0,1],[1,0],[1,1]] )
29      target_output  = np.array([[0,1,1,0]])
30
31      # Reshaping target output into vector:
32
33      target_output = target_output.reshape(4,1)
34
35      # Initial weights and learning parameters ..
36
37      weight_hidden = np.random.rand(2,4)
38      weight_output = np.random.rand(4,1)
39
40      lr = 0.10 # Learning Rate:
41
42      # Train neural network, 300,000 epochs ...
43
44      for epoch in range(300001):
45
46          # Input for hidden layer :
47
48          input_hidden = np.dot(input_features, weight_hidden)
49
50          # Output from hidden layer :
51
52          output_hidden = sigmoid(input_hidden)
53
54          # Input for output layer :
```

Example 3. Modeling an XOR Boolean Gate

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

```
55
56          input_op = np.dot(output_hidden, weight_output)
57
58          # Output from output layer :
59
60          output_op = sigmoid(input_op)
61
62          # Phase 1: Compute derivatives for phase 1 ...
63
64          error_out = ((1 / 2) * (np.power((output_op - target_output), 2)))
65
66          # Derivatives for phase 1 :
67
68          derror_douto = output_op - target_output
69          douto_dino   = sigmoid_der(input_op)
70          dino_dwo     = output_hidden
71
72          derror_dwo = np.dot(dino_dwo.T, derror_douto * douto_dino)
73
74          # Phase 2: Compute derivatives for phase 2 ...
75
76          derror_dino  = derror_douto * douto_dino
77          dino_douth   = weight_output
78          derror_douth = np.dot(derror_dino , dino_douth.T)
79
80          douth_dinh = sigmoid_der(input_hidden)
81          dinh_dwh   = input_features
82          derror_dwh = np.dot(dinh_dwh.T, douth_dinh * derror_douth)
```

Example 3. Modeling an XOR Boolean Gate

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

```
83
84          # Update weights in hidden and output layers ...")
85
86          weight_hidden -= lr * derror_dwh
87          weight_output -= lr * derror_dwo
88
89      # Print Results ...
90
91      print("--- Final weights in hidden layer:");
92
93      print (weight_hidden)
94
95      print("--- Final weights in output layer:");
96
97      print (weight_output)
98
99      print("--- Use trained network to predict values ... ");
100     print("--- Verify input [0,0] --> 0 ... ");
101
102     single_point = np.array([0,0])
103
104     result1 = np.dot(single_point, weight_hidden)
105     result2 = sigmoid(result1)
106     result3 = np.dot( result2, weight_output)
107     result4 = sigmoid(result3)
108
109     print ("--- Result: %f ..." %(result4))
```

Example 3. Modeling an XOR Boolean Gate

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

```
111         ... lines of code removed ...
112
113         print("--- Verify input [1,1] --> 0 ... ");
114
115         single_point = np.array([1,1])
116
117         result1 = np.dot(single_point, weight_hidden)
118         result2 = sigmoid(result1)
119         result3 = np.dot(result2, weight_output)
120         result4 = sigmoid(result3)
121
122         print ("--- Result: %f ..." %(result4))
123
124     # call the main method ...
125
126     main()
```

Example 3. 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 3. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 1)

```
1  # ================================================================
2  # TestKeras-XOR-Problem.py: Use Keras to solve XOR gate problem.
3  #
4  # Written by: Mark Austin                                November 2020
5  # ================================================================
6
7  import numpy as np
8  from tensorflow import keras
9  from keras.models import Sequential
10 from keras.layers.core import Dense, Activation
11
12 # main method ...
13
14 def main():
15     print("--- Enter TestKeras-XOR-Problem1.main()       ... ");
16     print("--- ======================================== ... ");
17
18     print("--- Training data ...")
19
20     training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
21     print(training_data)
22
23     print("--- Target data for XOR problem ...")
24
25     target_data = np.array([[0],[1],[1],[0]], "float32")
26     print(target_data)
```

Example 3. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 2)

```
28      print("--- Define dense layers ...")
29
30      layer1 = Dense(2, input_dim =2, activation = 'sigmoid')
31      layer2 = Dense(1, activation = 'sigmoid' )
32
33      print("--- Assemble Sequential model ...")
34
35      model = Sequential()
36      model.add( layer1 )
37      model.add( layer2 )
38
39      print("--- Compile model ...")
40
41      model.compile( loss='mean_squared_error',
42                     optimizer='adam', metrics=['binary_accuracy'])
43
44      print("--- Train model to fit data ...")
45
46      model.fit( training_data, target_data, epochs=20000, verbose=2)
47
48      # Retrieve and print layer weights and bias values ...
49
50      np.set_printoptions(formatter={'float': '{: 0.5f}'.format})
51
52      print("--- Layer 1: weights and bias ...")
53
54      print( layer1.get_weights()[0] )
```

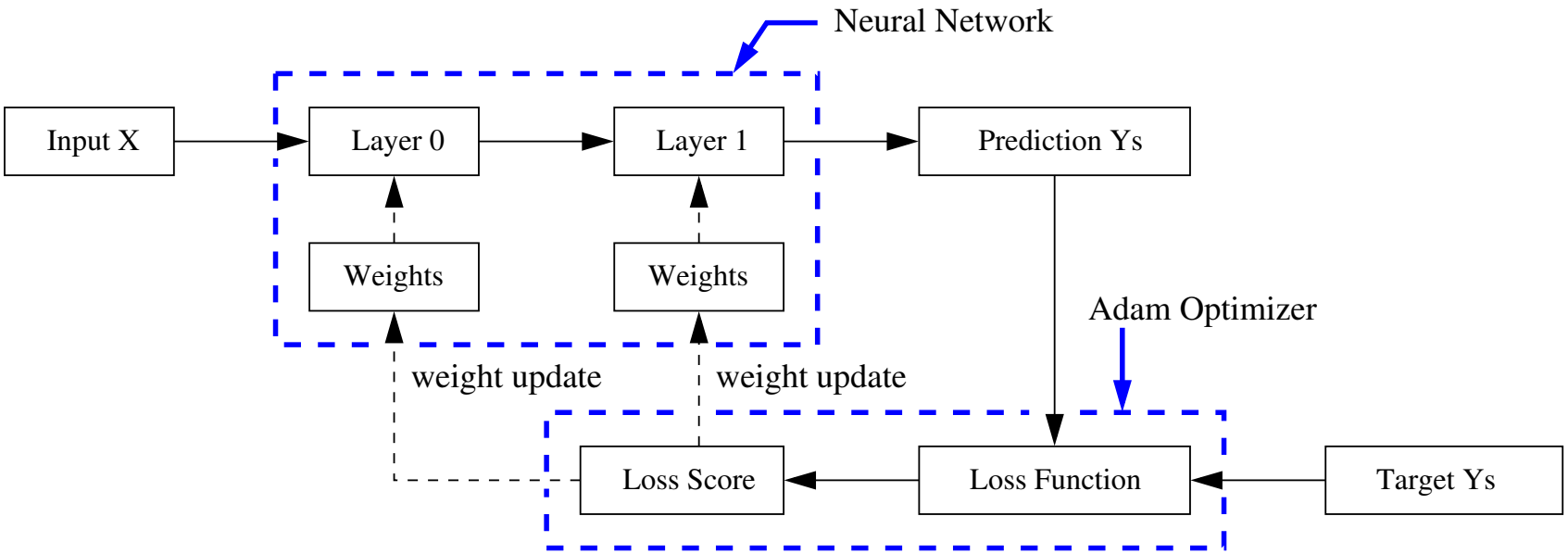
Example 3. Modeling an XOR Boolean Gate

TensorFlow 2 + Keras: Code (pg. 3)

```
55      print("--- Layer 2: weights and bias ...")
56
57      print ( layer2.get_weights()[0] )
58      print ( layer2.get_weights()[1] )
59
60      print("--- Trained model predictions ...")
61
62      print( model.predict(training_data) )
63
64      print("--- Rounded model predictions ...")
65
66      print( model.predict(training_data).round())
67
68      print("--- ======================================== ... ");
69      print("--- Finished!! ... ");
70
71  # call the main method ...
72
73  main()
```

Example 3. 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 3. 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 3. Modeling an XOR Boolean Gate

DL4J: Create training dataset:

```
1    // Create matrix of input values ...
2
3       double[][] matrixDouble = new double[][]{ {0.0, 0.0}, {1.0, 0.0},
4                                                 {0.0, 1.0}, {1.0, 1.0}};
5       INDArray input01 = Nd4j.create(matrixDouble);
6
7    // Create vector of expected output values ....
8
9       double[] vectorDouble = new double[]{0,1,1,0};
10      INDArray output01 = Nd4j.create(vectorDouble).transpose();
11
12      DataSet ds = new DataSet(input01, output01 );
```

DL4J: MultiLayer Configuration with 2 layers:

```
13   // Create neural network configuration builder ...
14
15   MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
16        .updater(new Sgd(0.1))
17        .seed(seed)
18        .biasInit(0) // Init the bias with 0 - empirical value, too
19        .miniBatch(false)
```

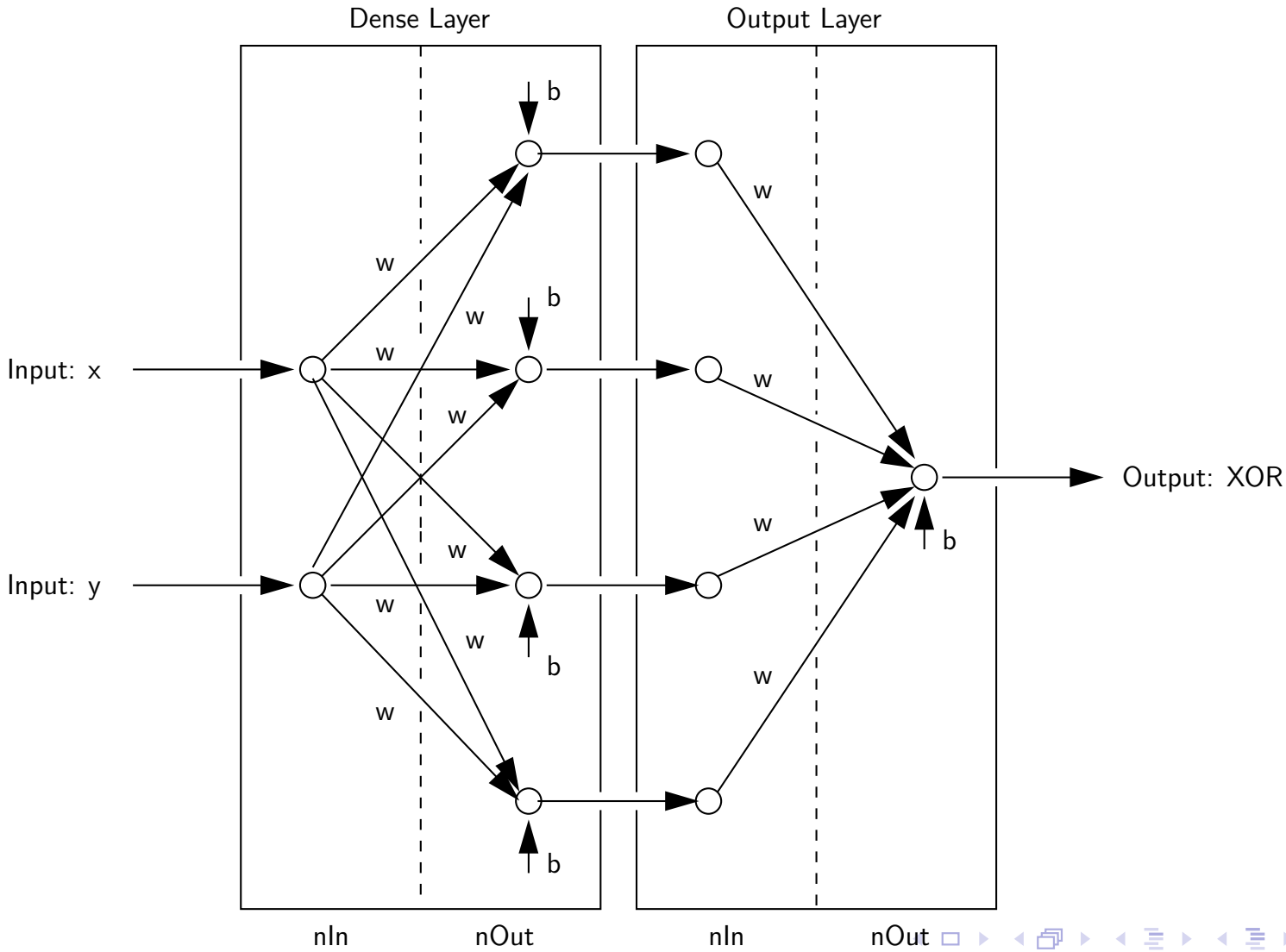
Example 3. Modeling an XOR Boolean Gate

DL4J: Create Network Configuration (cont'd):

```
23              .list()
24              .layer(new DenseLayer.Builder()
25                 .nIn(2)
26                 .nOut(2)
27                 .activation(Activation.SIGMOID)
28                 .weightInit( new UniformDistribution(0, 1) )
29                 .build())
30              .layer(new OutputLayer.Builder( LossFunctions.LossFunction.MSE )
31                 .nIn(2)
32                 .nOut(1)
33                 .activation(Activation.IDENTITY)
34                 .weightInit(new UniformDistribution(0, 1))
35                 .build())
36              .build();
37
38          // Create multilayer network ...
39
40          MultiLayerNetwork net = new MultiLayerNetwork(conf);
41          net.init();
42          net.setListeners(new ScoreIterationListener(500));
```

Example 3. Modeling an XOR Boolean Gate

DL4J: Network Model (nIn, nOut)



Example 3. 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:

```
52      for( int i=0; i <= 10000; i++ ) {
53         net.fit(ds);
54      }
```

Example 3. 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 ]
```

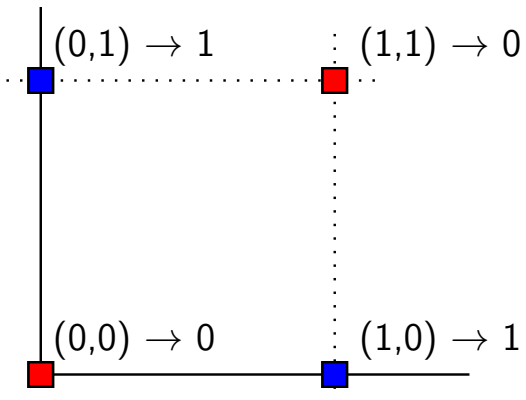
DL4J: Evaluation metrics:

```
 ========================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") only
```

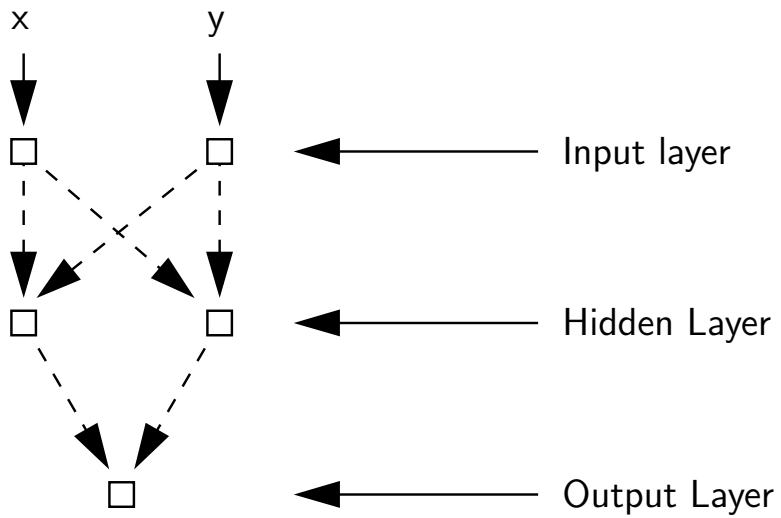
Example 3. Modeling an XOR Boolean Gate

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 3. Modeling an XOR Boolean Gate

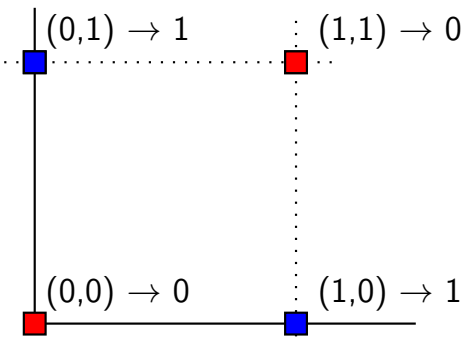
Summary of Results: TensorFlow 2 + Keras

```
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]]

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

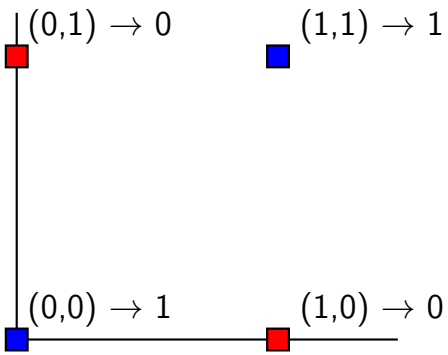
Example 4. Modeling XNOR and XOR Boolean Gates

Network Architecture:

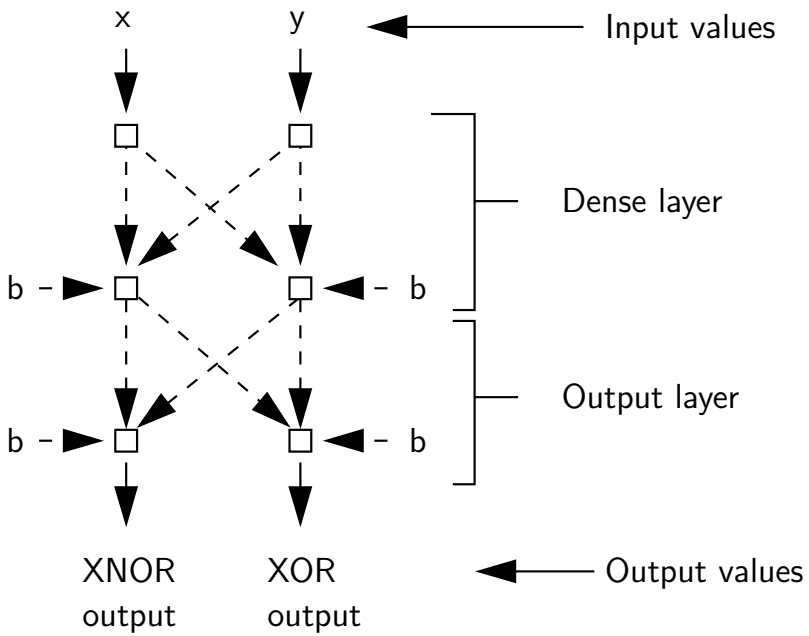
XOR Problem Space



XNOR Problem Space



Neural Network Architecture (2 inputs, 2 hidden nodes)



Example 4. 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 4. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 1)

```
1   # ================================================================
2   # TestKeras-XNOR-XOR-Problem.py: Keras implementation of [ XNOR, XOR ]
3   # neural net.
4   #
5   # Written by: Mark Austin                                November 2020
6   # ================================================================
7
8   import numpy as np
9   from tensorflow import keras
10  from keras.models import Sequential
11  from keras.layers.core import Dense, Activation
12  from keras.optimizers import SGD
13
14  # main method ...
15
16  def main():
17      print("--- Enter TestKeras-XNOR-XOR-Problem.main()  ... ");
18      print("--- ======================================== ... ");
19
20      print("--- Training data ...")
21
22      training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
23      print(training_data)
24
25      print("--- Target data for XNOR, XOR problem ...")
26
27      target_data = np.array([[1,0],[0,1],[0,1],[1,0]], "float32")
```

Example 4. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 2)

```
28      print(target_data)
29
30      print("--- Define dense layers ...")
31
32      layer1 = Dense(4, input_dim =2, activation = 'sigmoid')
33      layer2 = Dense(2, activation = 'sigmoid' )
34
35      print("--- Assemble Sequential model ...")
36
37      model = Sequential()
38      model.add( layer1 )
39      model.add( layer2 )
40
41      print("--- Compile model ...")
42
43      model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy
44
45      print("--- Train model to fit data ...")
46
47      model.fit( training_data, target_data, epochs=20000, verbose=2)
48
49      # Retrieve and print layer weights and bias values ...
50
51      np.set_printoptions(formatter={'float': '{: 0.5f}'.format})
52
53      print("--- Layer 1: weights ...")
```

Example 4. Modeling XNOR and XOR Boolean Gates

TensorFlow 2 + Keras: Code (pg. 3)

```
55      print( layer1.get_weights()[0] )
56
57      print("--- Layer 1: bias values ...")
58
59      print( layer1.get_weights()[1] )
60
61      print("--- Layer 2: weights ...")
62
63      print ( layer2.get_weights()[0] )
64
65      print("--- Layer 2: bias values ...")
66
67      print ( layer2.get_weights()[1] )
68
69      print("--- Trained model predictions ...")
70
71      print( model.predict(training_data) )
72
73      print("--- Rounded model predictions ...")
74
75      print( model.predict(training_data).round())
76
77      print("--- ======================================== ... ");
78      print("--- Finished!! ... ");
79
80  # call the main method ...
81
82  main()
```

Example 4. 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 4. 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 4. Modeling XNOR and XOR Boolean Gates

DL4J: Create training dataset:

```
1   INDArray input = Nd4j.zeros(4, 2);     INDArray labels = Nd4j.zeros(4, 2);
2
3   input.putScalar(new int[]{0, 0}, 0);   labels.putScalar(new int[]{0, 0}, 1);
4   input.putScalar(new int[]{0, 1}, 0);   labels.putScalar(new int[]{0, 1}, 0);
5
6   input.putScalar(new int[]{1, 0}, 1);   labels.putScalar(new int[]{1, 0}, 0);
7   input.putScalar(new int[]{1, 1}, 0);   labels.putScalar(new int[]{1, 1}, 1);
8
9   input.putScalar(new int[]{2, 0}, 0);   labels.putScalar(new int[]{2, 0}, 0);
10  input.putScalar(new int[]{2, 1}, 1);   labels.putScalar(new int[]{2, 1}, 1);
11
12  input.putScalar(new int[]{3, 0}, 1);   labels.putScalar(new int[]{3, 0}, 1);
13  input.putScalar(new int[]{3, 1}, 1);   labels.putScalar(new int[]{3, 1}, 0);
14
15  DataSet ds = new DataSet(input, labels);
```

DL4J: Create Network Configuration with 2 layers:

```
16  // Create neural network configuration builder ...
17
18  NeuralNetConfiguration.Builder builder = new NeuralNetConfiguration.Builder();
19  builder.updater(new Sgd(0.1));
20  builder.seed(123);
21  builder.biasInit(0);
22  builder.miniBatch(false);
```

Example 4. Modeling XNOR and XOR Boolean Gates

DL4J: Create Network Configuration (cont'd):

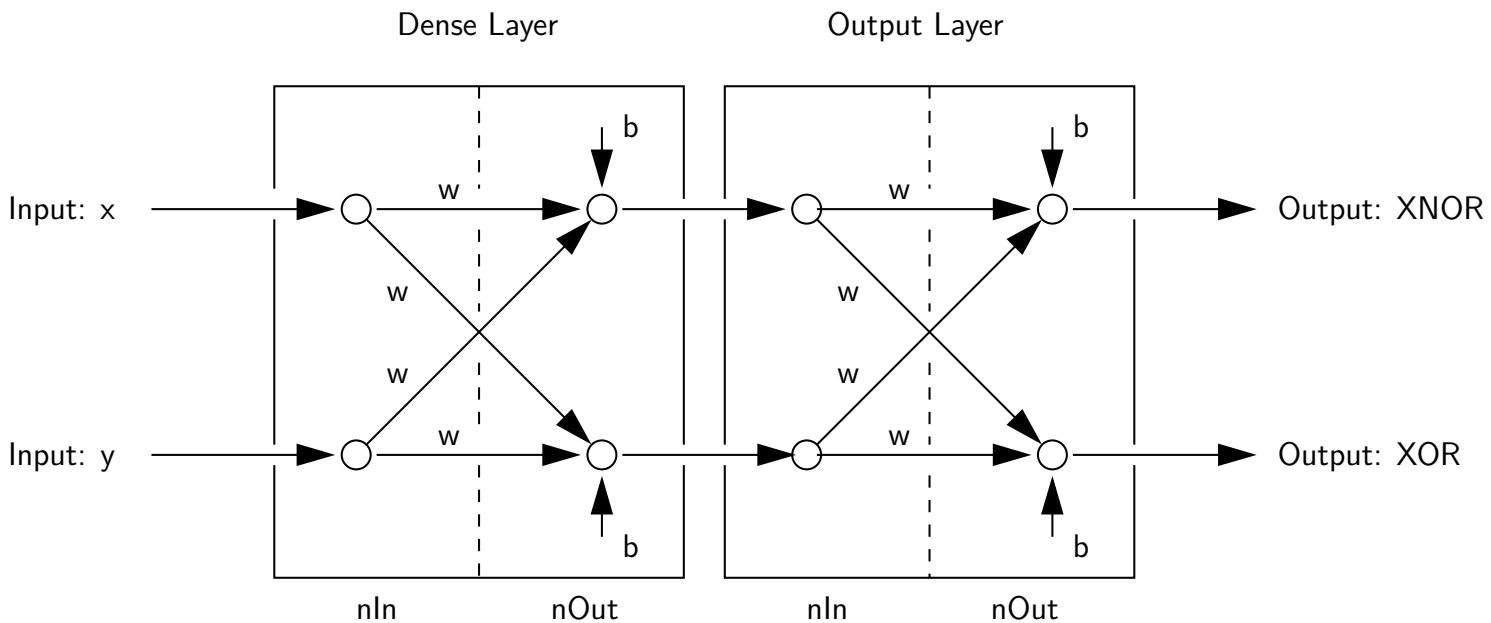
```
23  // Create dense layer with 2 input connections ...
24
25  DenseLayer.Builder hiddenLayerBuilder = new DenseLayer.Builder();
26  hiddenLayerBuilder.nIn(2);
27  hiddenLayerBuilder.nOut(2);
28  hiddenLayerBuilder.activation(Activation.SIGMOID);
29  hiddenLayerBuilder.weightInit(WeightInit.DISTRIBUTION);
30
31  // Create output layer with 2 output connections ...
32
33  Builder outputLayerBuilder = new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVE
34  outputLayerBuilder.nIn(2);
35  outputLayerBuilder.nOut(2);
36  outputLayerBuilder.activation(Activation.SOFTMAX);
37  outputLayerBuilder.dist(new UniformDistribution(0, 1));
38
39  // Create list of layers in network configuration ...
40
41  ListBuilder listBuilder = builder.list();
42  listBuilder.layer(0, hiddenLayerBuilder.build());
43  listBuilder.layer(1, outputLayerBuilder.build());
44  listBuilder.pretrain(false);
45  listBuilder.backprop(true);
```

Example 4. Modeling XNOR and XOR Boolean Gates

DL4J: Create Network Configuration (cont'd):

```
46  // Build and check the network configuration ...
47
48  MultiLayerConfiguration conf = listBuilder.build();
49  MultiLayerNetwork net = new MultiLayerNetwork(conf);
50  net.init();
51  net.setListeners(new ScoreIterationListener( 1000 ));
```

DL4J: Network Model (nIn, nOut)



Example 4. 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 4. Modeling XNOR and XOR Boolean Gates

DL4J: Train the network for 10,000 epochs:

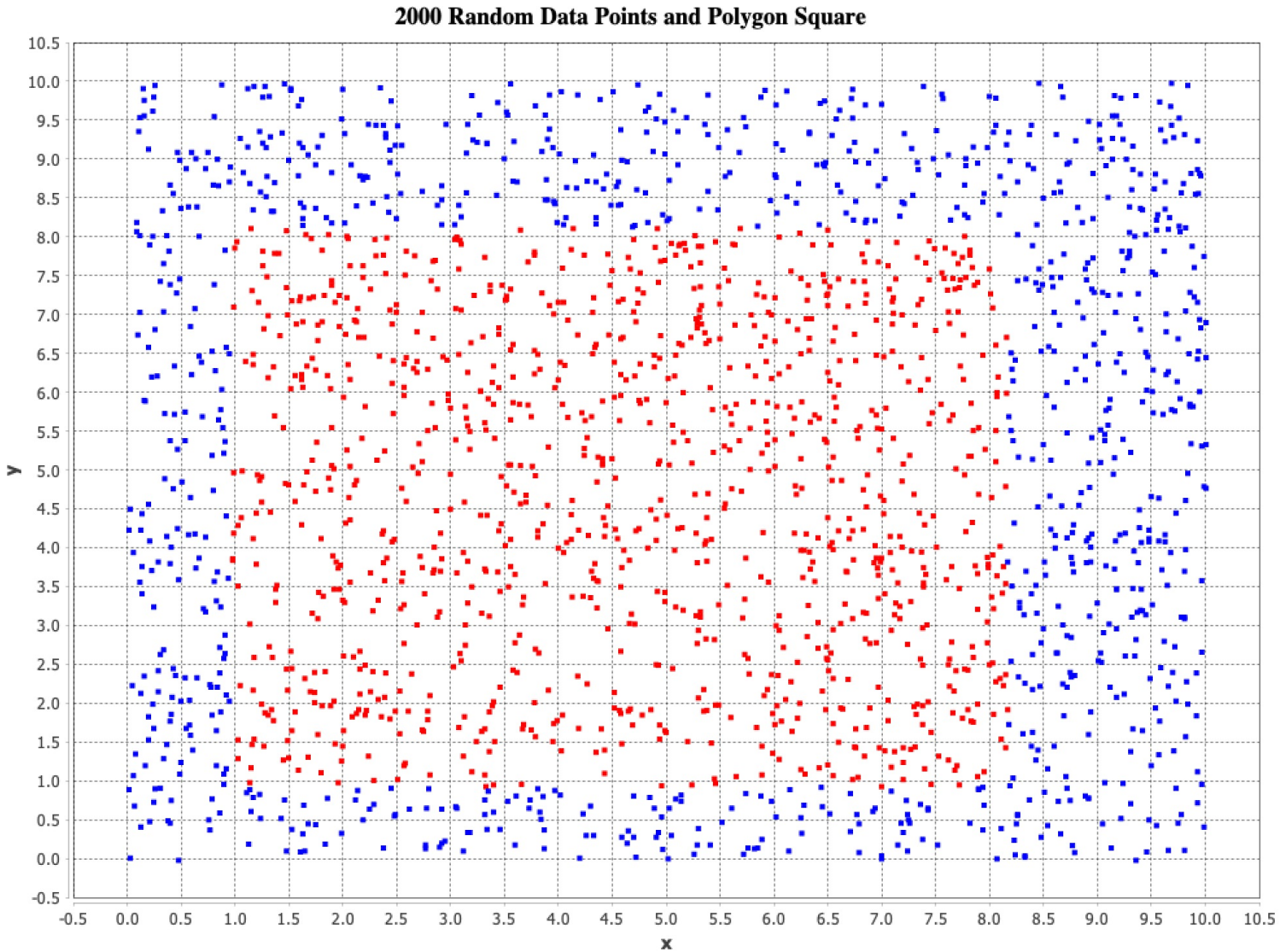
```
52      for( int i=0; i <= 10000; i++ ) {
53         net.fit(ds);
54      }
```

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 5. Points in Convex Polygon



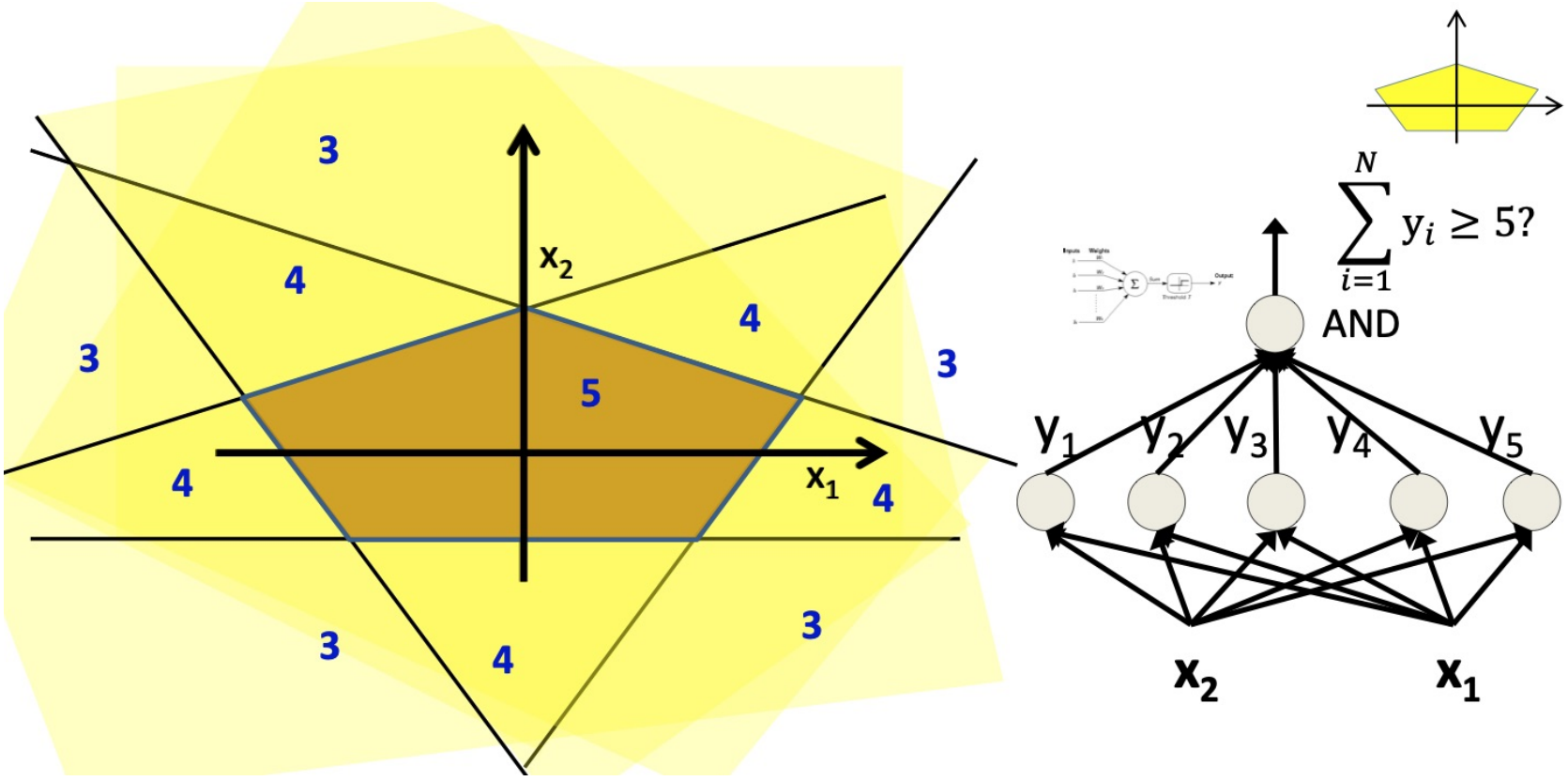
Example 5. Points in Convex Polygon

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 5. Points in Convex Polygon

DL4J: Read training dataset ...

```
1       // Read polygon square-shaped data ...
2
3       double[][] x = DataUtils.readInputsFromFile( "data/polygon-square-data.txt");
4       double[][] t = DataUtils.readInputsFromFile( "data/polygon-square-outcome.txt");
```

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

```
1       // Scale coordinates from [0,10] --> [0,1] ....
2
3       for (int i = 0; i < x.length; i = i + 1 ) {
4          x[i][0] = x[i][0]/10.0;
5          x[i][1] = x[i][1]/10.0;
6       }
```

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

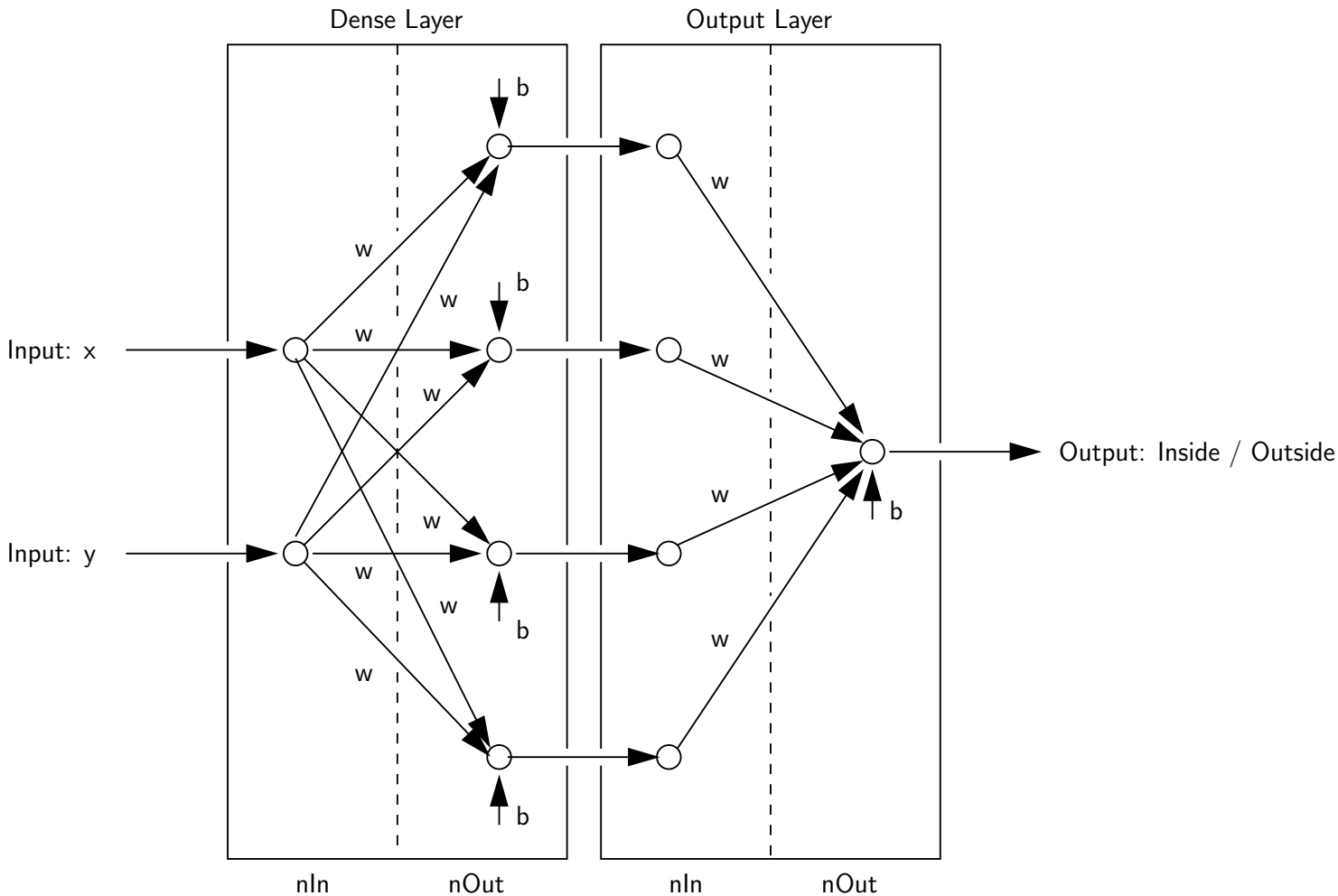
Example 5. Points in Convex Polygon

DL4J: Create Network Configuration:

```
1   MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
2        .updater(new Sgd(0.01))
3        .seed(seed)
4        .biasInit(0) // Init the bias with 0 - empirical value, too
5        .miniBatch(false)
6        .list()
7        .layer( new DenseLayer.Builder()
8             .nIn(2)
9             .nOut(8)
10            .activation(Activation.SIGMOID)
11            .weightInit(WeightInit.DISTRIBUTION)
12            .build())
13       .layer( new OutputLayer.Builder(LossFunctions.LossFunction.MSE )
14            .nIn(8)
15            .nOut(1)
16            .activation(Activation.SIGMOID)
17            .weightInit(WeightInit.DISTRIBUTION)
18            .build())
19       .pretrain(false)
20       .backprop(true)
21       .build();
22
23  MultiLayerNetwork net = new MultiLayerNetwork(conf);
24  net.init();
25  net.setListeners(new ScoreIterationListener(250));
```

Example 5. 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 5. 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:

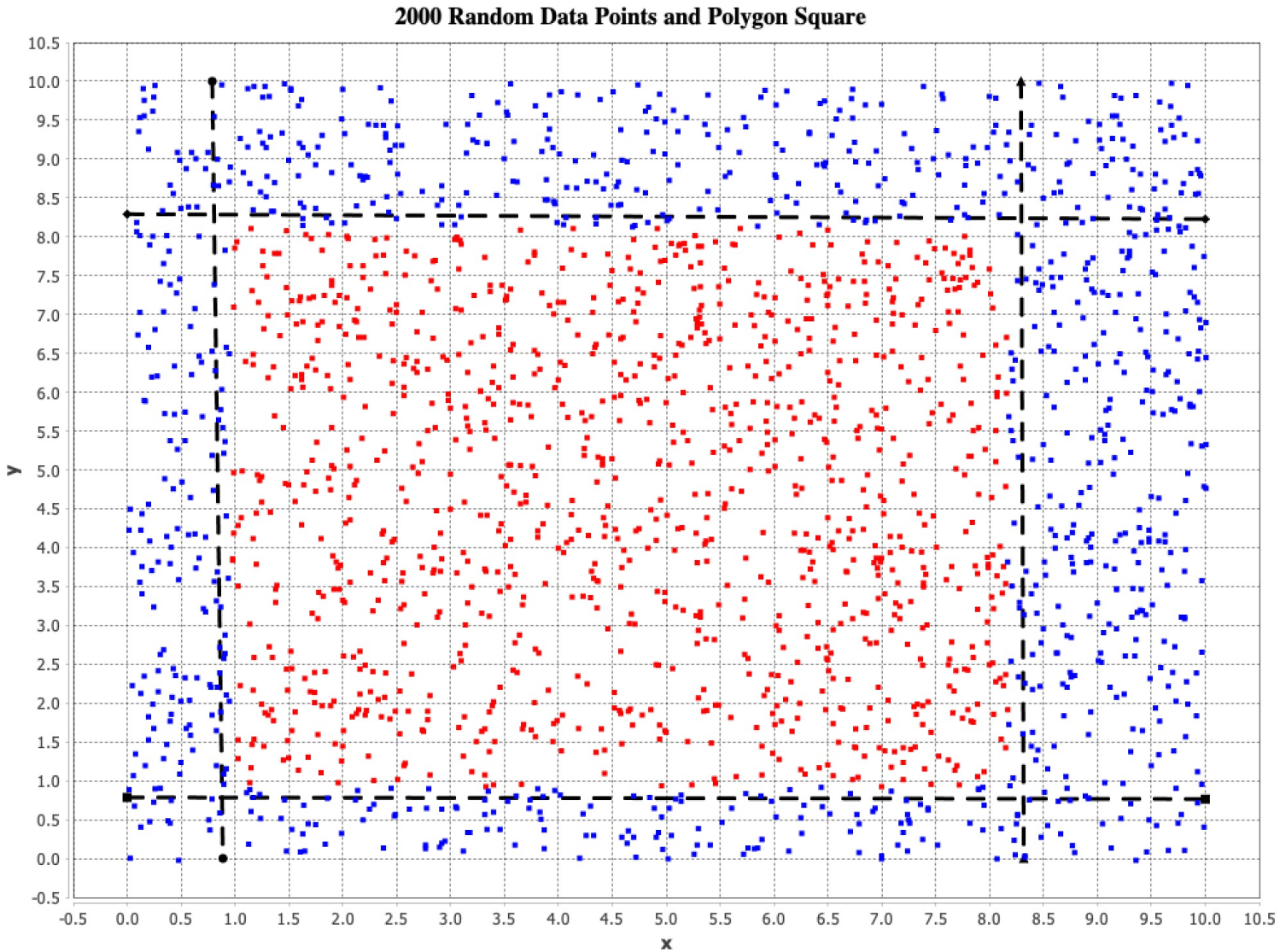
$$
\begin{aligned}
f_1(x_1, x_2) &= -0.3x_1 - 51.5x_2 + 42.7 = 0 \\
f_2(x_1, x_2) &= 59.3x_1 + 0.15x_2 - 49.3 = 0 \\
f_3(x_1, x_2) &= 69.0x_1 + 0.68x_2 - 6.12 = 0 \\
f_4(x_1, x_2) &= -0.12x_1 - 60.4x_2 + 4.74 = 0
\end{aligned}
\tag{1}
$$

Example 5. Points in Convex Polygon

Decision boundary equations scaled to $[10 \times 10]$ grid:

$$\begin{aligned} f_1(x_1, x_2) = 0 \quad &\rightarrow x_2 \approx 427/51.5 = 8.2. \\ f_2(x_1, x_2) = 0 \quad &\rightarrow x_1 \approx 493/59.3 = 8.3. \\ f_3(x_1, x_2) = 0 \quad &\rightarrow x_1 \approx 61/69 = 0.9. \\ f_4(x_1, x_2) = 0 \quad &\rightarrow x_2 \approx 47.4/60.1 = 0.8. \end{aligned} \tag{2}$$

Example 5. Points in Convex Polygon



Example 5. Points in Convex Polygon

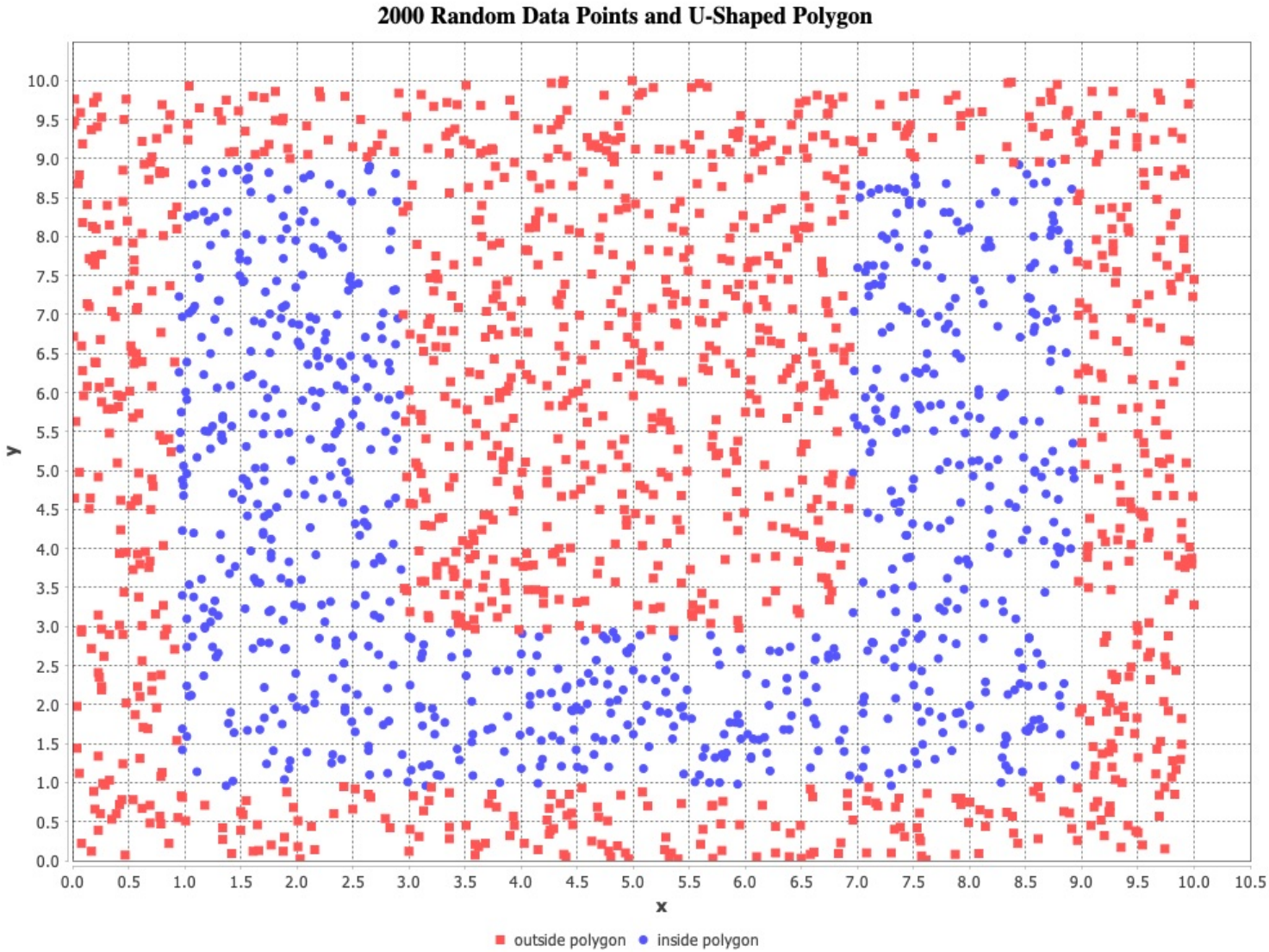
DL4J: Evaluation Metrics (2, 4 and 6 nodes on hidden layers)

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 6. Points in Non-Convex Polygon



Counter Example 6. 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 6. 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 6. 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	

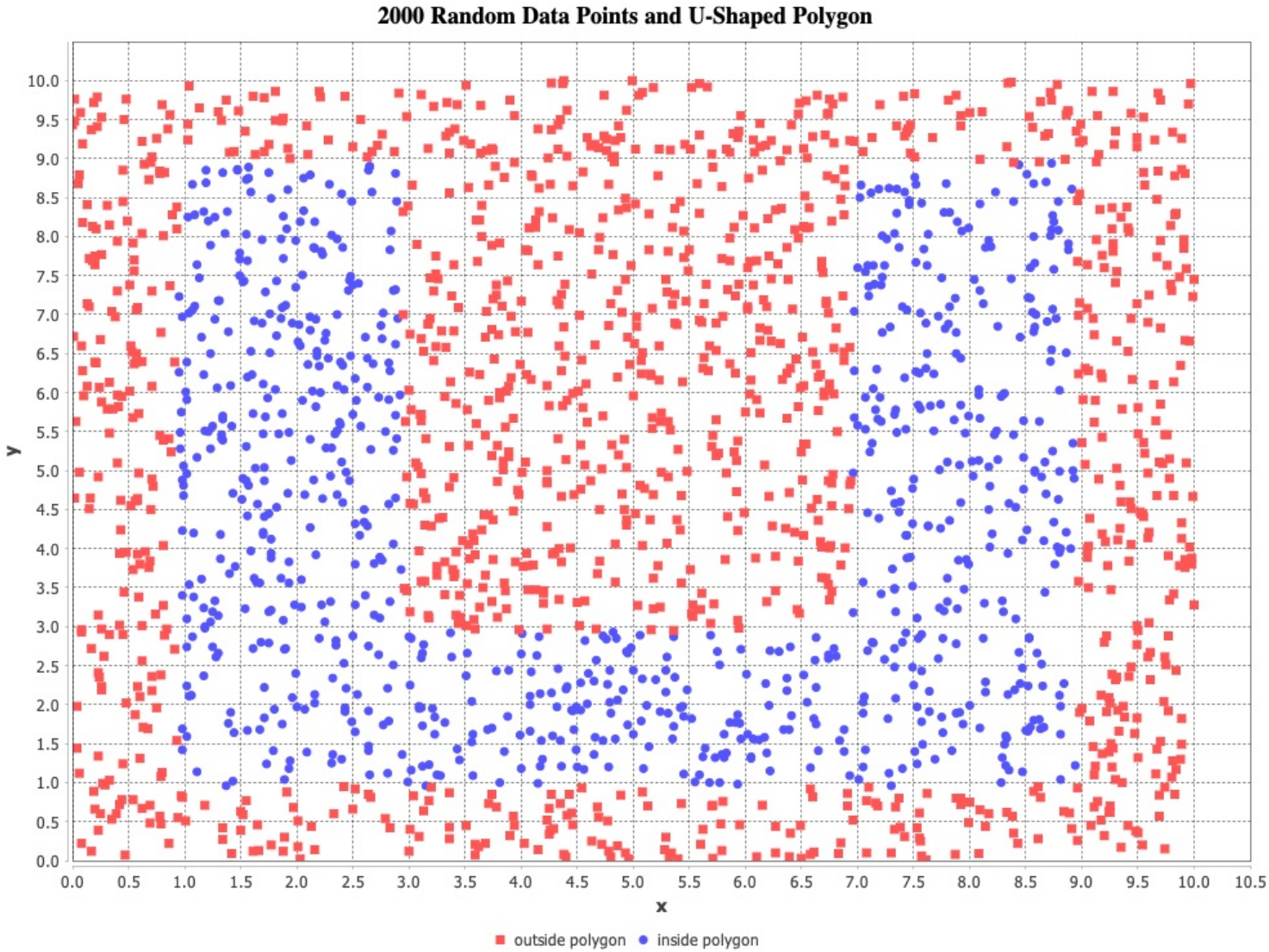
DL4J: Confusion Matrix (4 and 8 nodes on hidden layer)

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

Networks with

Two Hidden Layers

Example 7. Points in U-Shaped Polygon



Example 7. Points in U-Shaped 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.
- A neural network with only one hidden layer cannot capture the non-convex shape. Thus, the purpose of this example is to see use of two layers solves the problem.

Example 7. Points in U-Shaped Polygon

DL4J: Read training dataset ...

```
1       // Read polygon U-Shaped data ...
2
3       double[][] x = DataUtils.readInputsFromFile( "data/polygon-u-shape-data.txt");
4       double[][] t = DataUtils.readInputsFromFile( "data/polygon-u-shape-outcome.txt");
```

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

```
1       // Scale coordinates from [0,10] --> [0,1] ....
2
3       for (int i = 0; i < x.length; i = i + 1 ) {
4          x[i][0] = x[i][0]/10.0;
5          x[i][1] = x[i][1]/10.0;
6       }
```

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

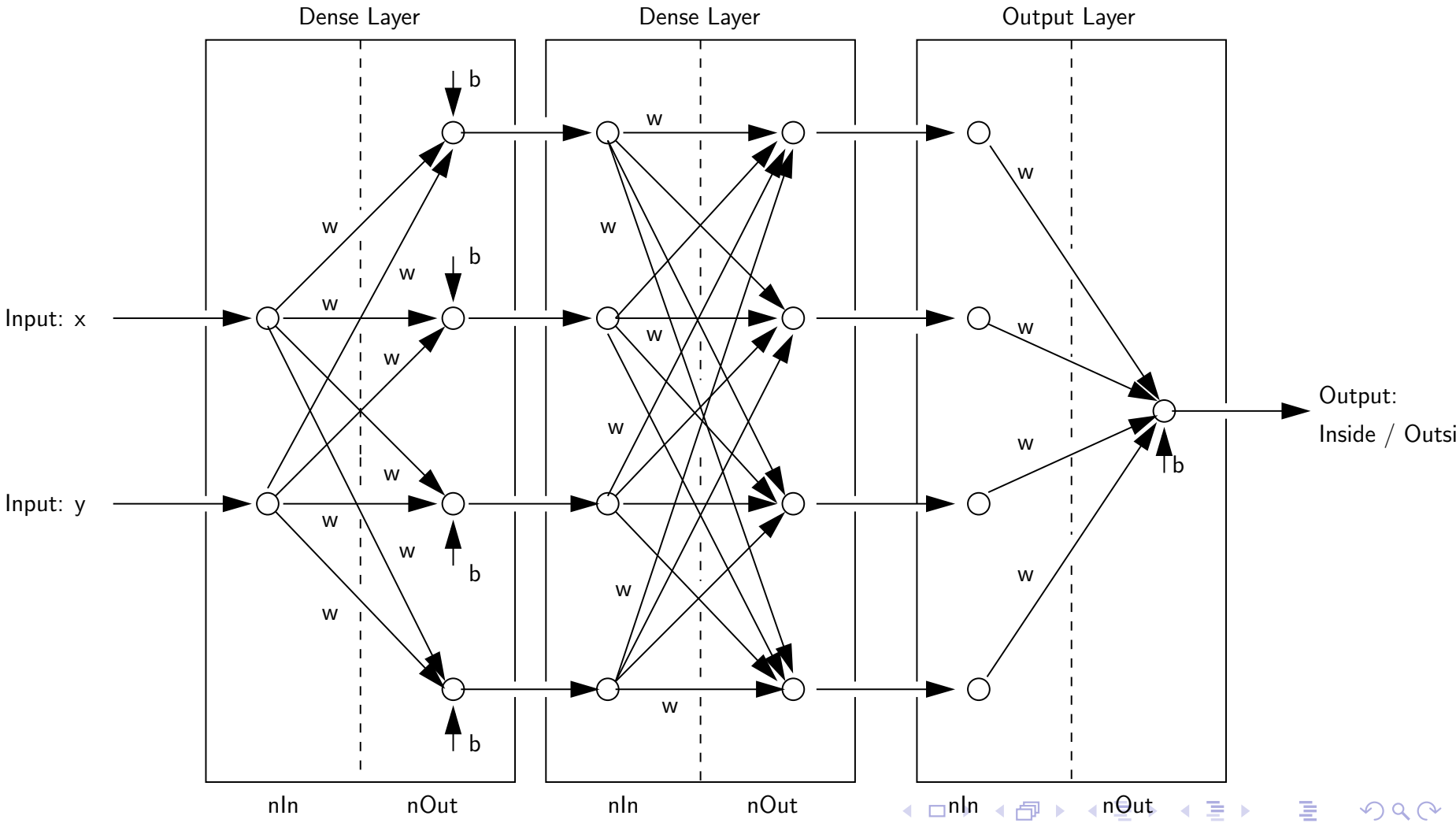
Example 7. Points in U-Shaped Polygon

DL4J: Create Network Configuration:

```
1   MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
2       .updater(new Sgd(0.01))
3       .seed(seed)
4       .biasInit(0) // Init the bias with 0 - empirical value, too
5       .miniBatch(false)
6       .list()
7       .layer( new DenseLayer.Builder()
8            .nIn(2).nOut(4)
9            .activation(Activation.SIGMOID)
10           .weightInit(WeightInit.DISTRIBUTION)
11           .build())
12      .layer( new DenseLayer.Builder()
13           .nIn(4).nOut(4)
14           .activation(Activation.SIGMOID)
15           .weightInit(WeightInit.DISTRIBUTION)
16           .build())
17      .layer( new OutputLayer.Builder(LossFunctions.LossFunction.MSE )
18           .nIn(4).nOut(1)
19           .activation(Activation.SIGMOID)
20           .weightInit(WeightInit.DISTRIBUTION)
21           .build())
22      .pretrain(false)
23      .backprop(true)
24      .build();
```

Example 7. Points in U-Shaped Polygon

DL4J: Network Model (nIn, nOut)



Example 7. Points in U-Shaped Polygon

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 (DenseLayer)      4,4         20          W:{4,4}, b:{1,4}
   layer2 (OutputLayer)     4,1         5           W:{4,1}, b:{1,1}
----------------------------------------------------------------
   Total Parameters:  37   Trainable Parameters:  37
```

DL4J: Summary of Network Model (12 nodes on hidden layer)

```
LayerName (LayerType)    nIn,nOut    TotalParams    ParamsShape
================================================================
   layer0 (DenseLayer)      2,12        36          W:{2,12}, b:{1,12}
   layer1 (DenseLayer)      12,12       156         W:{12,12}, b:{1,12}
   layer2 (OutputLayer)     4,1         5           W:{12,1}, b:{1,1}
----------------------------------------------------------------
   Total Parameters:  205  Trainable Parameters:  205
```

Example 7. Points in U-Shaped Polygon

DL4J: Training the Network (4 nodes on hidden layers 1 and 2) ...

```
16:38:21.779 Score at iteration 0 is 535.1254272460938
16:38:22.945 Score at iteration 250 is 526.7112426757812
16:38:23.796 Score at iteration 500 is 533.8363037109375
16:38:24.558 Score at iteration 750 is 540.9840087890625
16:38:25.268 Score at iteration 1000 is 538.2188720703125
16:38:25.985 Score at iteration 1250 is 534.0460815429688

  ... lines of output removed ...

16:41:34.715 Score at iteration 98750 is 116.39311981201172
16:41:35.196 Score at iteration 99000 is 115.5108642578125
16:41:35.693 Score at iteration 99250 is 116.2978515625
16:41:36.160 Score at iteration 99500 is 115.38568878173828
16:41:36.621 Score at iteration 99750 is 116.22488403320312
16:41:37.097 Score at iteration 100000 is 115.29683685302734
```

Example 7. Points in U-Shaped Polygon

DL4J: Training the Network (8 nodes on hidden layers 1 and 2) ...

```
16:13:09.127 Score at iteration 0 is 854.4296875

... lines of output removed ...

16:17:47.179 Score at iteration 99750 is 26.462177276611328
16:17:47.988 Score at iteration 100000 is 39.20866775512695
```

DL4J: Training the Network (12 nodes on hidden layers 1 and 2) ...

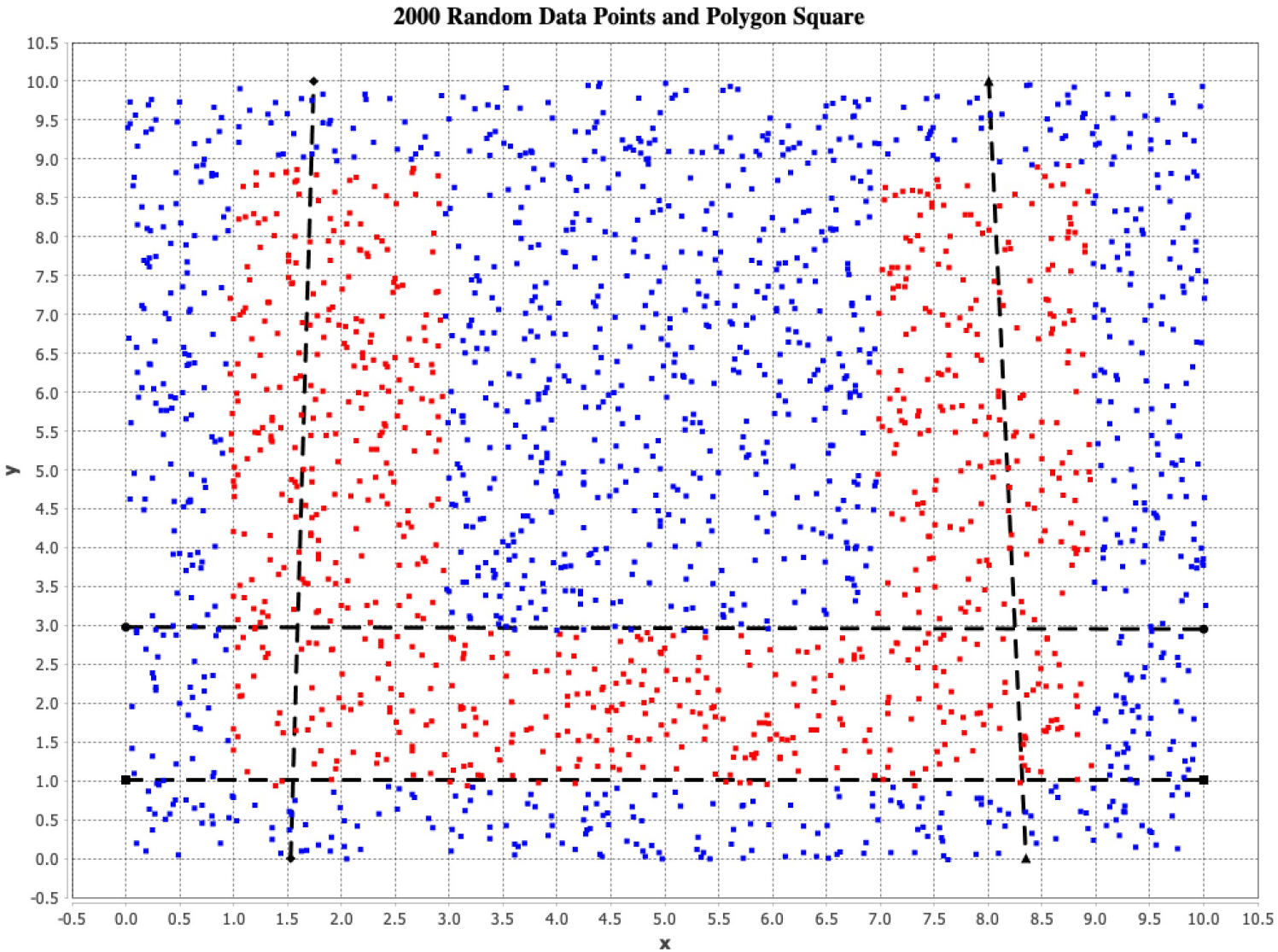
```
16:28:16.513 Score at iteration 0 is 1169.819091796875

  ... lines of output removed ...

16:33:34.319 Score at iteration 99750 is 18.23417091369629
16:33:35.162 Score at iteration 100000 is 3.9009976387023926
```

Example 7. Points in U-Shaped Polygon

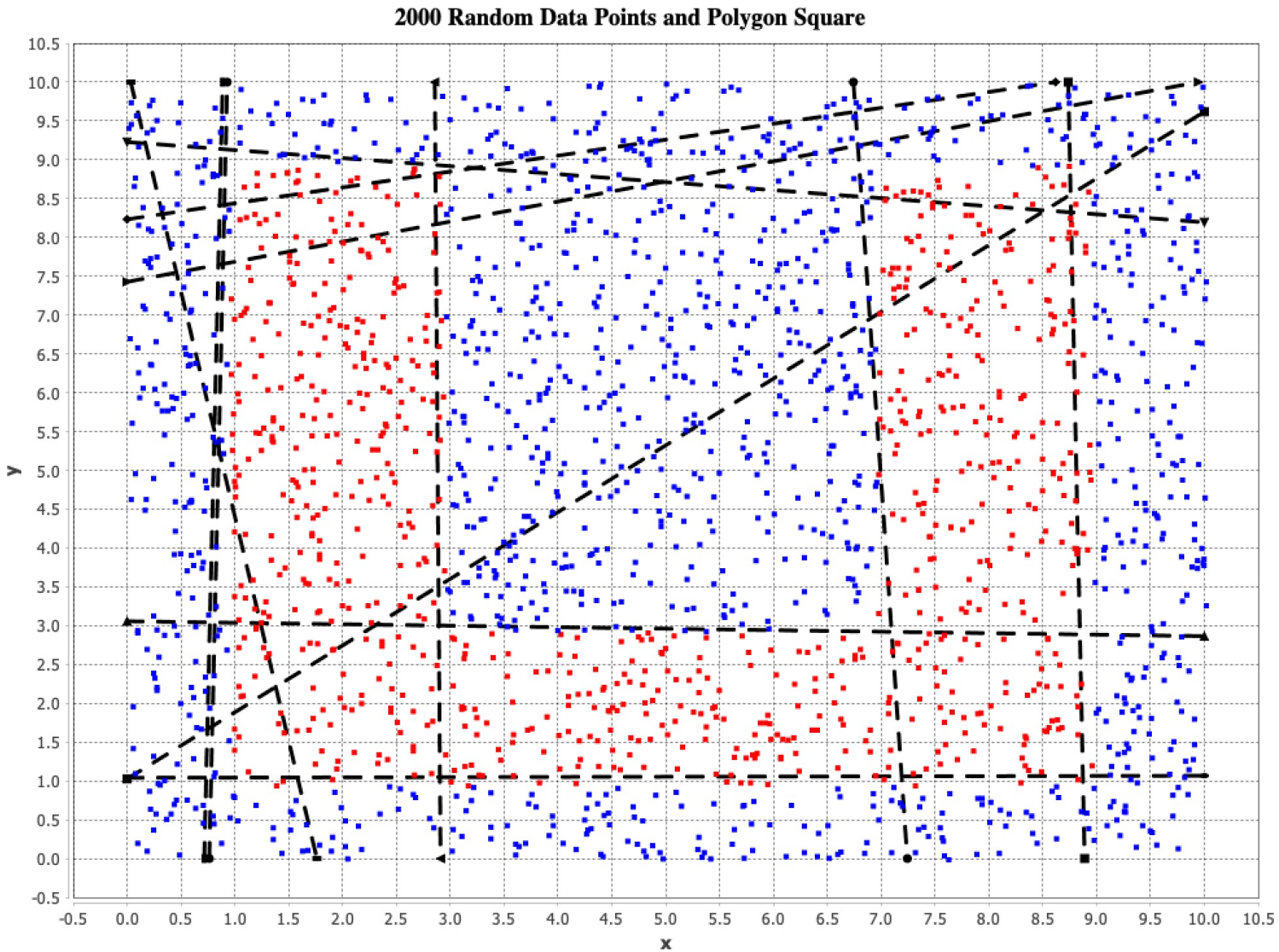
DL4J: Weights and Bias Values (4 nodes on hidden layers)

```
================================================================
Layer 0 weights: [[  -33.6689,   38.2139,    0.1660,   -0.0018],
                  [    0.7178,    1.3214,   62.7854,  -82.9784]]
Layer 0 biases:  [[    5.1473,  -31.9166,  -18.6890,    8.3968]]
----------------------------------------------------------------

Layer 1 weights: [[  -16.0345,   82.5613,   -8.2395,  -53.1885],
                  [  -15.8898,   83.7063,   -5.4736,  -52.4685],
                  [  -33.8128,   -0.2496,  -10.7231,    3.3472],
                  [   -5.2890,  -47.5059,   -6.5428,  -75.1653]]
Layer 1 biases:  [[   14.5033,   -0.1769,    1.9136,   48.5209]]
----------------------------------------------------------------

Layer 2 weights:  [   13.1383,   12.1596,    6.3566,    7.4789 ]
Layer 2 biases:      -17.7825
================================================================
```

Example 7. Points in U-Shaped Polygon (4 nodes)



Example 7. Points in U-Shaped Polygon (12 nodes)



Example 7. Points in U-Shaped Polygon

DL4J: Evaluation Metrics (4, 8 and 12 nodes on hidden layers)

Metric	4 nodes	8 nodes	12 nodes
Accuracy:	0.9200	0.9815	0.9985
Precision:	0.9147	0.9817	0.9985
Recall:	0.9217	0.9799	0.9983
F1 Score:	0.9036	0.9769	0.9981

DL4J: Confusion Matrix (4, 8 and 12 nodes on hidden layers)

4 nodes		8 nodes		12 nodes		
0	1	0	1	0	1	
1090	104	1180	14	1193	1	0 = 0 <-- It works!
56	750	23	783	2	804	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.
- Sun J., Fundamental Belief: Universal Approximation Theorems, Computer Science and Engineering, University of Minnesota, Twin Cities, 2020.