Multi-Layer Perceptron

Kashif Inayat
Information Security and Machine Learning Lab, Hongik University, South Korea

Outline

- Introduction
- Architecture & It’s Implementation
- Learning Process & It’s Implementation
- Training Set/Data Set
- Questions/Answers

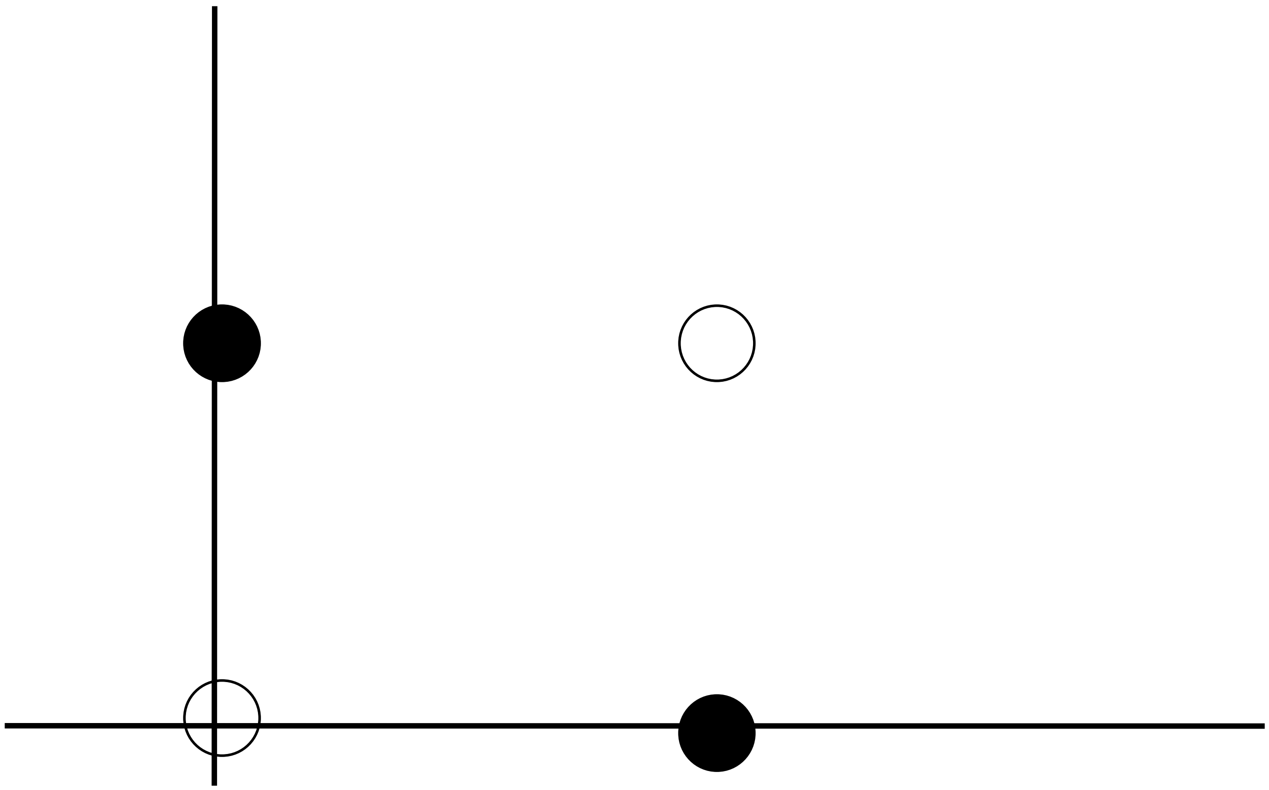
Introduction

- **Limitations in Perceptron:**

1) Can't solve X-OR Problems
2) Can't solve Non-Linearly Separable Problems

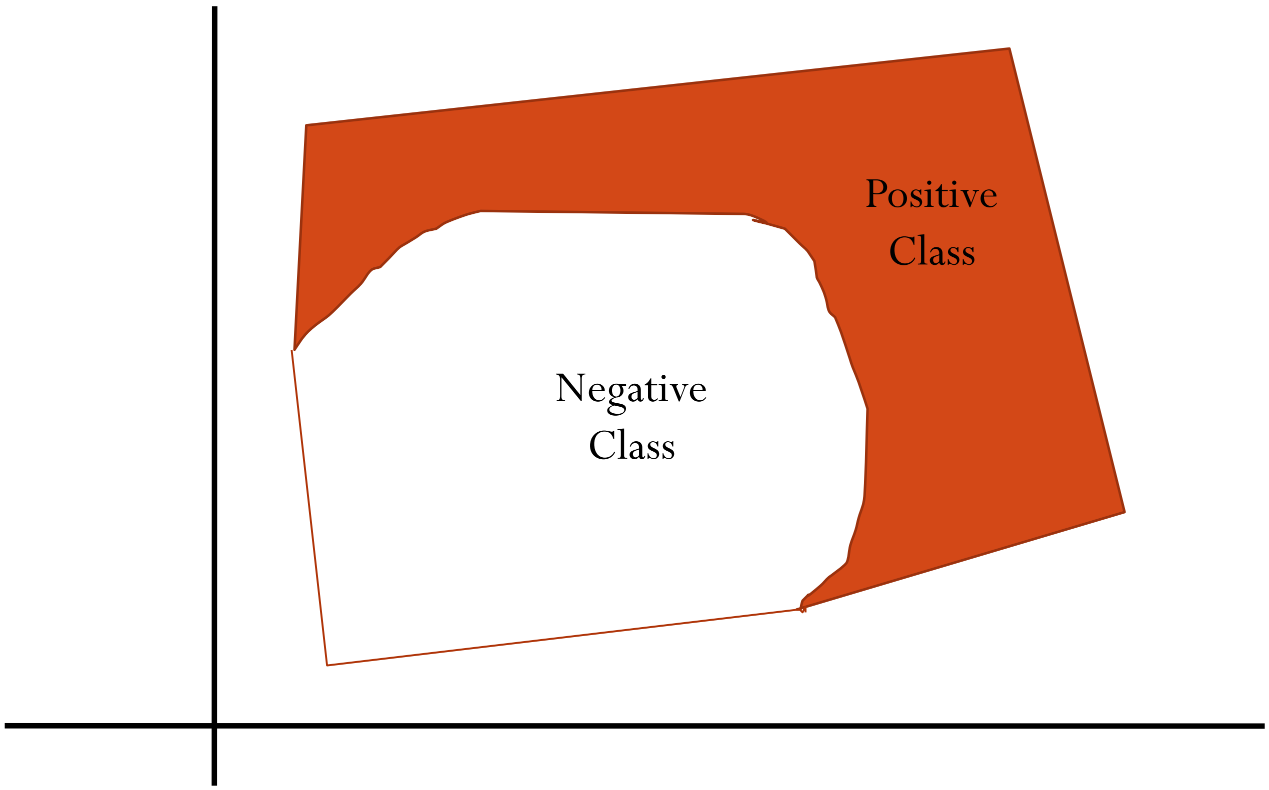
Introduction

- XOR Problem



Introduction

- Nonlinear Separable Problem



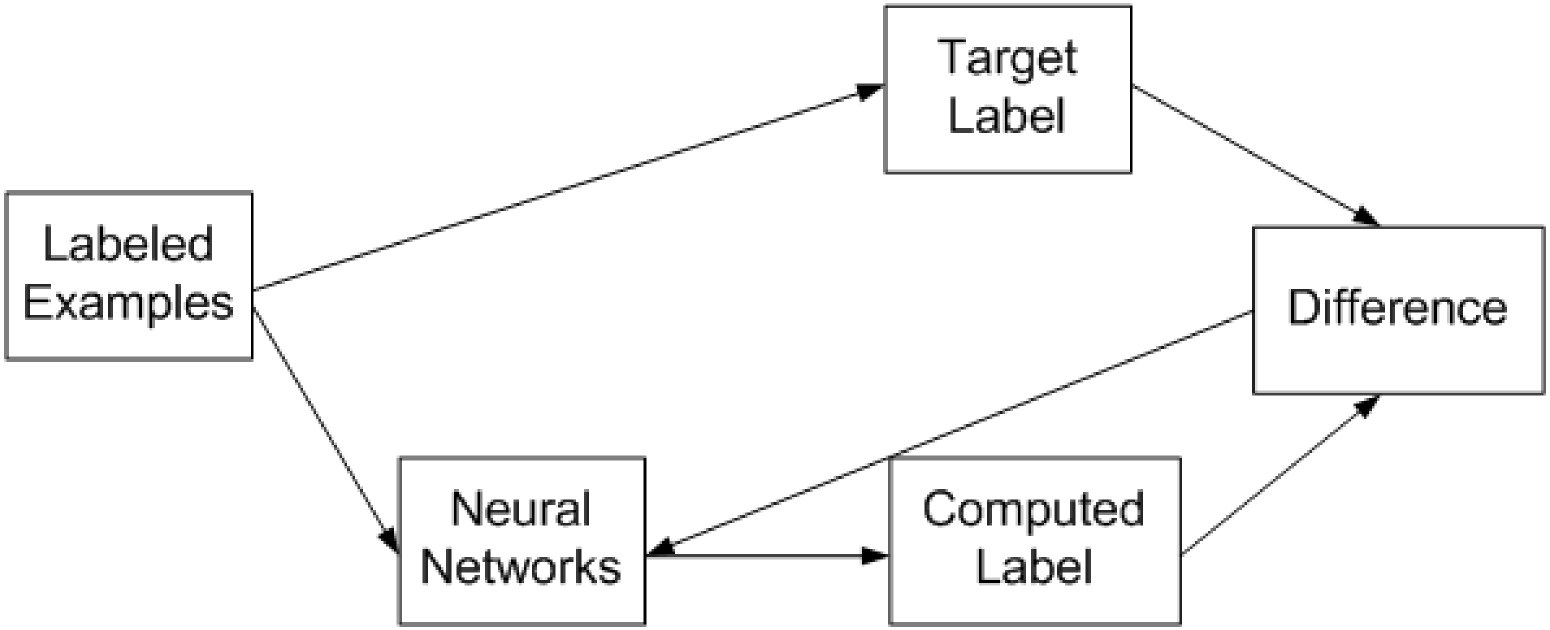
Motivations:

1) Addition of intermediate layer called hidden layer

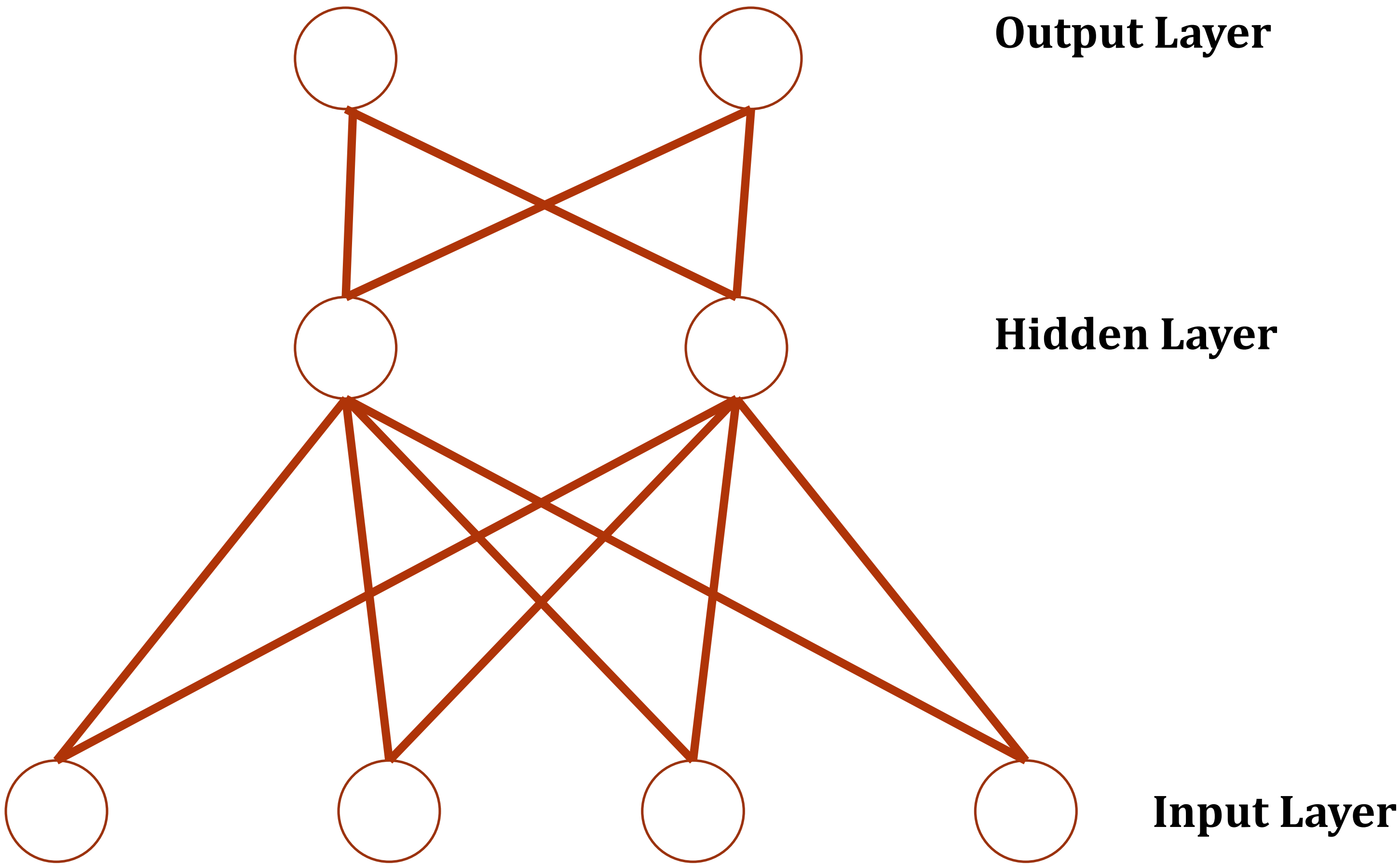
1) Solvable even to Nonlinear Separable

Introduction: Multilayer Perceptron

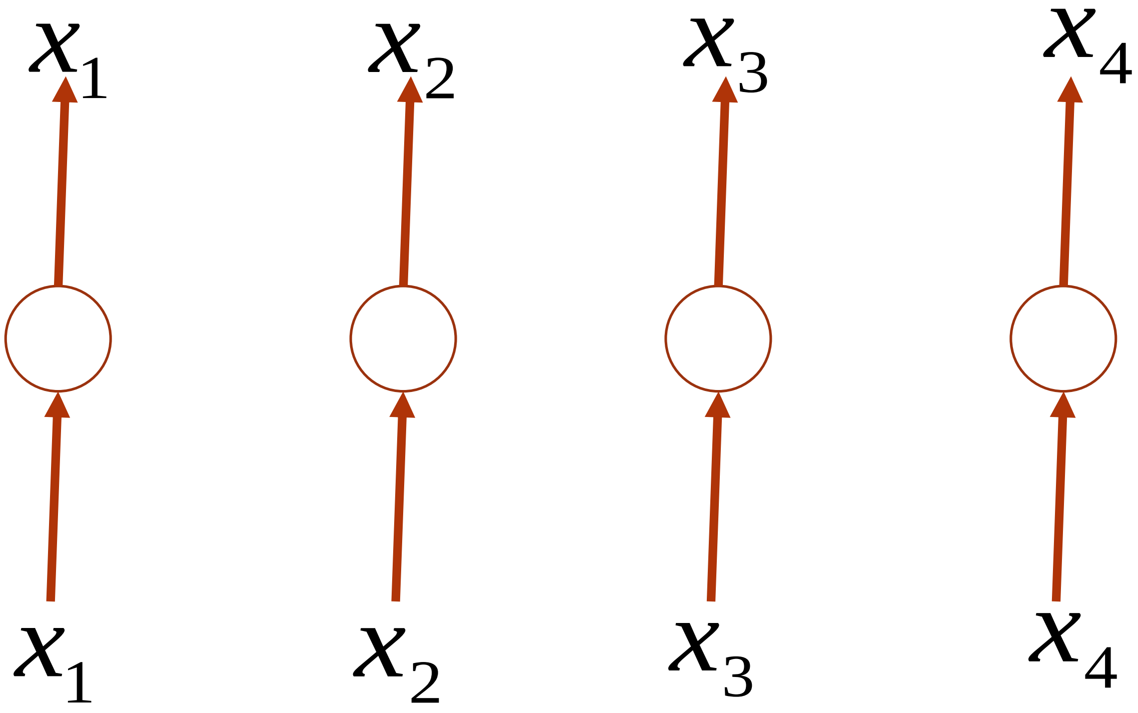
- Supervised Neural Networks



Architecture



Architecture: Input Node



Implementation : Input Layer

```java
public class InputLayer implements Layer{
    int dinput;
    NeuralVector inputLayerValues;
    public InputLayer(int inputLength){
//          inputNodes = nodes;
        dinput=inputLength;
    }

    @Override
    public void setInputNodesValues(NeuralVector inputVector) {
        inputLayerValues = inputVector;
    }

    public NeuralVector getOutputValuesOfLayer() {
        return inputLayerValues;
    }

}
```

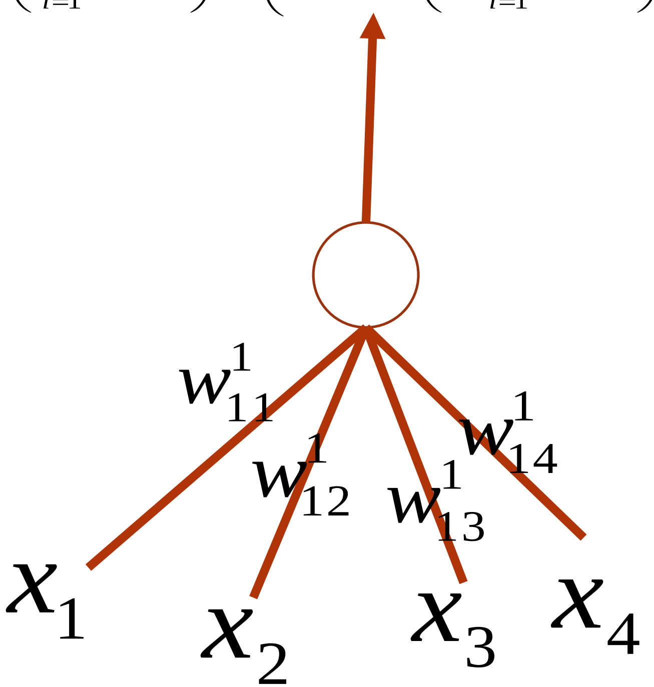
Implementation : Input Node

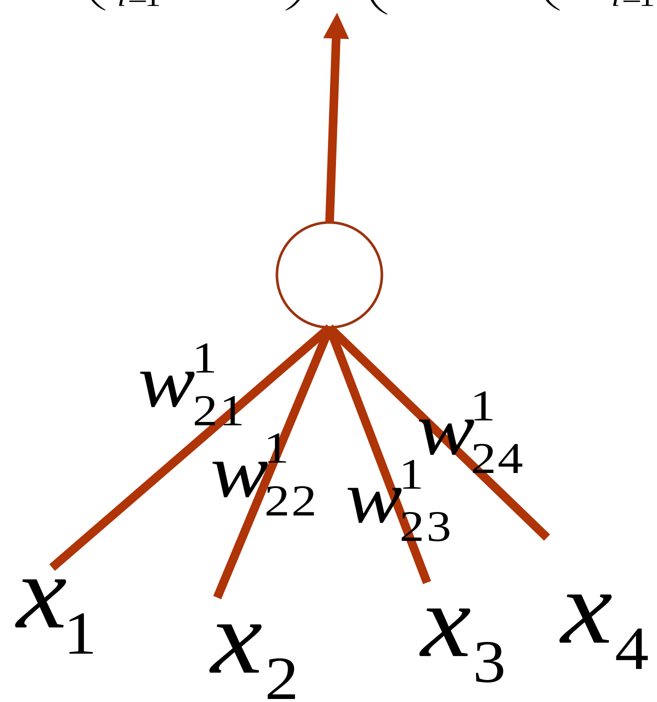
```
public void setValues(List<Double> value) {
    input = (Vector)value ;
}

@Override
public List<Double> getValues() {
    return input;
}



@Override
public double get(int i) {
    return input.elementAt(i);
}
```

Architecture: Hidden Node

$$h_1 = f\left(\sum_{i=1}^{4} x_i w_{1i}^1\right) = \left(1 + \exp\left(-\sum_{i=1}^{4} x_i w_{1i}^1\right)\right)^{-1}$$

$$h_2 = f\left(\sum_{i=1}^{4} x_i w_{2i}^1\right) = \left(1 + \exp\left(-\sum_{i=1}^{4} x_i w_{2i}^1\right)\right)^{-1}$$





Implementation : Hidden Layer

```
public class HiddenLayer implements Layer {

    List<HiddenNode> hiddenNodes;
    InputLayer inputValues;
    int dinput;
    double learningRate;

    int mhidden;

    public HiddenLayer(int numberOfHiddenNodes, int inputLength, double hiddenLayerLearingRate, InputLayer inputLayerValue
        dinput = inputLength;
        inputValues = inputLayerValue;
        learningRate = hiddenLayerLearingRate;
        hiddenNodes = new Vector<HiddenNode>();
        mhidden = numberOfHiddenNodes;

        //      inputNodes=inputnodes;
    }
```

Implementation : Hidden Layer

```java
    public void initilize() {

        for (int j = 0; j < mhidden; j++) {
            for (int i = 0; i < dinput; i++) {
                Random r = new Random();
                MultiLayerPerceptron.hiddenLayerWeights[j][i] = ((r.nextDouble() * (-0.1)) + 0.1);
            }
        }

        for (int i = 0; i < mhidden; i++) {
            hiddenNodes.add(new HiddenNode(dinput, mhidden, this));
        }

```

```java
    @Override
    public NeuralVector getOutputValuesOfLayer() {
        InputVector vectorFromInputLayer = (InputVector) inputValues.getOutputValuesOfLayer();
        int nodeIndex = 0;
        double[] hiddenLayerOutput = new double[mhidden];
        for (HiddenNode hiddenNode : hiddenNodes) {
            hiddenNode.calculateNetInput(vectorFromInputLayer, nodeIndex);
            hiddenLayerOutput[nodeIndex] = hiddenNode.getOutput();
            nodeIndex++;
        }
        return new HiddenLayerVector(hiddenLayerOutput);

    }

}

```

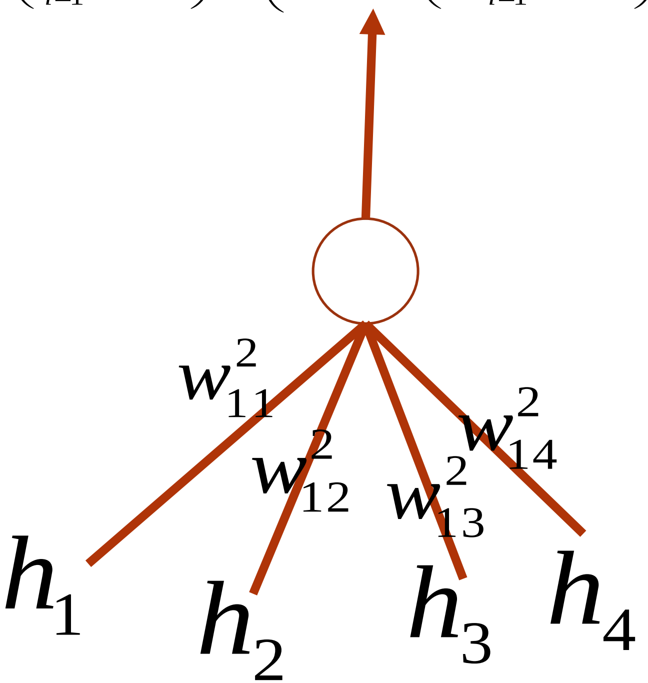
Implementation : Hidden Node

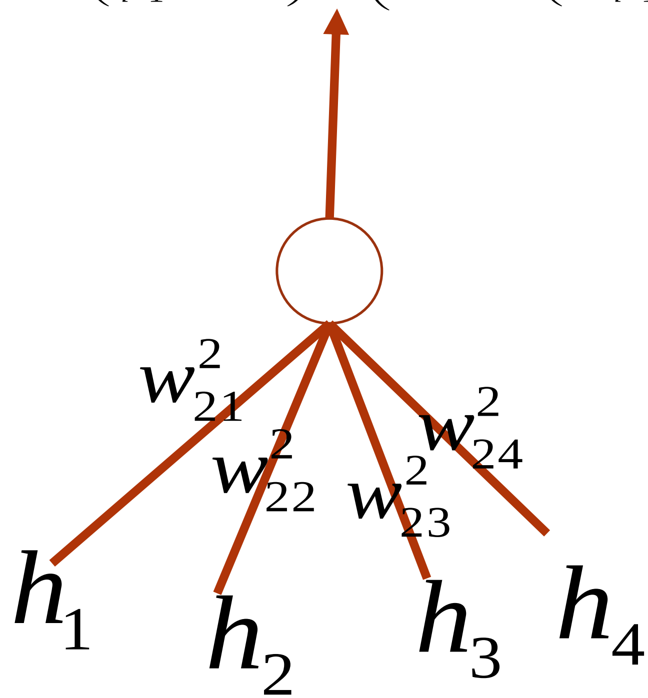
```java
@Override
void calculateNetInput(NeuralVector input, int nodeSubscript) {
    netInput = 0;
    for (int i = 0; i < inputLayerSize; i++) {
        netInput += (input.get(i) * MultiLayerPerceptron.hiddenLayerWeights[nodeSubscript][i]);
    }
}

@Override
public double getOutput() {
    outPutOfCurrentNode = (1 / (1 + Math.exp(-(netInput))));
    return outPutOfCurrentNode;
}
```

Architecture: Output Node

$$o_1 = f\left(\sum_{i=1}^{4} h_i w_{1i}^2\right) = \left(1+\exp\left(-\sum_{i=1}^{4} h_i w_{1i}^2\right)\right)^{-1}$$

$$o_2 = f\left(\sum_{i=1}^{4} h_i w_{2i}^2\right) = \left(1+\exp\left(-\sum_{i=1}^{4} h_i w_{2i}^2\right)\right)^{-1}$$





Architecture: Output Layer

```java
public class OutputLayer extends LearnableLayer {

    List<OutputNode> outputNodes;
    HiddenLayer hiddenLayer;
    int mhidden;
    int coutput;
    //double[][] weights;
    double learningRate;

    public OutputLayer(int outputLength, int hiddenLength, double hiddenLayerLearningRate, HiddenLayer hiddenLayerValue)
        mhidden = hiddenLength;
        coutput = outputLength;
        hiddenLayer = hiddenLayerValue;
        learningRate = hiddenLayerLearningRate;
        outputNodes = new Vector<OutputNode>();

    }

```

Architecture: Output Layer

```java
    public void initilize() {


        for (int j = 0; j < coutput; j++) {
            for (int i = 0; i < mhidden; i++) {
                Random r = new Random();
                MultiLayerPerceptron.outputLayerWeight[j][i] = ((r.nextDouble() * (-0.1)) + 0.1);
            }
        }

        for (int i = 0; i < coutput; i++) {
            outputNodes.add(new OutputNode(/*hiddenLayerValue,*/mhidden, coutput, this,0.52));
        }
    }

```

```java
    @Override
    public NeuralVector getOutputValuesOfLayer() {
        HiddenLayerVector vectorFromHiddenLayer = (HiddenLayerVector) hiddenLayer.getOutputValuesOfLayer();
        int nodeIndex = 0;
        double[] finalOutput = new double[coutput];
        for (OutputNode outputNode : outputNodes) {
            outputNode.calculateNetInput(vectorFromHiddenLayer, nodeIndex);
            finalOutput[nodeIndex] = outputNode.getActivationFunctionOutput();
            nodeIndex++;
        }
        return new OutputVector(finalOutput);
    }
}

```

Implementation : Output Node

```java
@Override
void calculateNetInput(NeuralVector input, int nodeSubscript) {
    netInput = 0;
    for (int i = 0; i < hiddenLayerSize; i++) {
        netInput += (input.get(i) * MultiLayerPerceptron.outputLayerWeight[nodeSubscript][i]);
    }
}


public double getActivationFunctionOutput(){

    activationFunctionOutput = (1 / (1 + Math.exp(-(netInput))));
    System.out.println("Finaloutput: " +outputOfCurrentNode+"Bias"+bias );
    return activationFunctionOutput;
}
```

Learning Rule: Back Propagation

- Update weight between output and hidden

$$E = \frac{1}{2}\sum_{k=1}^{c}(t_k - o_k)^2 \qquad o_j = \left[1 + \exp\left(-net_j^o\right)\right]^{-1} \quad net_j^o = \sum_{k=1}^{m} h_k w_{jk}^2$$

$$\frac{\partial E}{\partial w_{ji}^2} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j^o}\frac{\partial net_j^o}{\partial w_{ji}^2}$$

$$\frac{\partial o_j}{\partial net_j^o} = \left[\left(1 + \exp\left(-net_j^o\right)\right)^{-1}\right]' = o_j\left(1 - o_j\right)$$

$$\frac{\partial E}{\partial o_j} = -\left(t_j - o_j\right) \qquad \frac{\partial net_j^o}{\partial w_{ji}^2} = h_i$$

$$w_{ji}^2 = w_{ji}^2 + \eta\left(t_j - o_j\right)o_j\left(1 - o_j\right)h_i$$

Implementation : Learning Rule: Back Propagation

- Update weight between output and hidden

```
void learn(double expectedValue, int nodeSubscript, NeuralVector hiddenLayerOutput) {
    getOutput();
    for (int i = 0; i < hiddenLayerSize; i++) {

        MultiLayerPerceptron.outputLayerWeight[nodeSubscript][i] += outputLayer.learningRate * (expectedValue
    - activationFunctionOutput) * activationFunctionOutput * (1 - activationFunctionOutput) * hiddenLayerOutput.get(i);

    }

}
```

Learning Rule: Back Propagation

- Update weight between hidden and input

$$\frac{\partial E}{\partial w_{ji}^1} = \sum_{k=1}^{c} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k^o} \frac{\partial net_k^o}{\partial h_j} \frac{\partial h_j}{\partial net_j^h} \frac{\partial net_j^h}{\partial w_{ji}^1}$$

$$h_j = \left[1 + \exp\left(-net_j^h\right)\right]^{-1} \qquad net_j^h = \sum_{i=1}^{d} x_k w_{ji}^1$$

$$\frac{\partial E}{\partial o_k} = -\left(t_k - o_k\right) \qquad \frac{\partial o_k}{\partial net_k^o} = o_k\left(1 - o_k\right)$$

$$\sum_{k=1}^{c} \frac{\partial net_k^o}{\partial h_j} = \sum_{k=1}^{c} w_{kj}^2 \qquad \frac{\partial h_j}{\partial net_j^h} = \left(1 - h_j\right)h_j \qquad \frac{\partial net_i^h}{\partial w_{ji}^1} = x_i$$

$$w_{ji}^1 = w_{ji}^1 + \eta \sum_{k=1}^{c} \left(t_k - o_k\right) o_k \left(1 - o_k\right) w_{kj}^2 \left(1 - h_j\right) h_j x_i$$

Implementation : Learning Rule: Back Propagation

- Update weight between hidden and input

```
void learn(double[] expectedOutputValues, OutputVector outputValues, NeuralVector input, int nodeSubscript) {
    getOutput();
    for (int i = 0; i < inputLayerSize; i++) {
        double multipleOfLearningRate  = 0;
        //Why no increment in output index?
        for(int outputIndex = expectedOutputValues.length - outputValues.length();
                outputIndex < expectedOutputValues.length; outputIndex++){//How we separate the target velue ?
            int c=outputIndex-inputLayerSize;
            double outputValue = outputValues.get(c);//We don't iterating output?

            double expectedVale = expectedOutputValues[outputIndex];
            //System.out.println("c Index:"+c+"output Index:"+outputIndex  );
            multipleOfLearningRate += (expectedVale - outputValue) * outputValue * (1 - outputValue) *
MultiLayerPerceptron.outputLayerWeight[c][nodeSubscript] * (1 - outPutOfCurrentNode)*outPutOfCurrentNode * input.get (i);
        }
```

Learning Rule: Back Propagation

- Interactive Learning

Input: Training Examples

$$E = \sum_{i=1}^{c} \left(t_i - o_i\right)^2$$

Initialize Weights at Random
Iterate T times
    for each training example{
        compute values of hidden nodes
        compute value of output nodes
        compute average error
        update weights between output and hidden
        update weights between hidden and input
    }
Output: Optimized Weights

Implementation : Learning Rule Back Propagation

- Interactive Learning

```
public void train(ArrayList<double[]> trainingExamples, int trainingIterations) throws Exception {
    if (trainingExamples.size() < 1) {
        throw new Exception("training examples must have some values.");
    }
    if (trainingExamples.get(0).length != numberOfInputNodes + numberOfOutputNodes) {
        throw new Exception("Length of training Example is not consistant with dimensions of MLP.");
    }
    //why this running for 1 time
    for (int i = 0; i < trainingIterations; i++) {//How many time inner loop will this run?29
        for (int j = 0; j < trainingExamples.size(); j++) {
            InputVector iv = new InputVector(trainingExamples.get(j), numberOfInputNodes);
            iv.printScreen();
            inputLayer.setInputNodesValues(iv);
            OutputVector ov = (OutputVector) outputLayer.getOutputValuesOfLayer();
            outputLayer.learn(trainingExamples.get(j));
            hiddenLayer.learn(trainingExamples.get(j),ov);
            ov.printScreen();

        }
    }
```

Training Data Set

```
traningExamples.add(new double[]{68, 29, 82, 0, 1, 0});
traningExamples.add(new double[]{43, 34, 42, 1, 0, 0});
traningExamples.add(new double[]{8, 91, 37, 0, 1, 1});
traningExamples.add(new double[]{71, 16, 95, 1, 0, 1});
traningExamples.add(new double[]{91, 15, 49, 1, 1, 1});
traningExamples.add(new double[]{26, 44, 24, 0, 0, 0});
traningExamples.add(new double[]{71, 94, 22, 1, 0, 0});
traningExamples.add(new double[]{74, 45, 64, 0, 1, 0});
traningExamples.add(new double[]{89, 37, 77, 1, 1, 1});
traningExamples.add(new double[]{69, 76, 53, 1, 0, 1});
```

```
traningExamples.add(new double[]{1,0 , 1});
traningExamples.add(new double[]{1,1 , 0});
traningExamples.add(new double[]{0,0 , 0});
traningExamples.add(new double[]{0,1 , 1});
traningExamples.add(new double[]{1,1 , 0});
traningExamples.add(new double[]{1,0 , 1});
traningExamples.add(new double[]{1,1 , 0});
traningExamples.add(new double[]{0,0 , 0});
traningExamples.add(new double[]{1,0 , 1});
traningExamples.add(new double[]{0,1 , 1});
```

```
traningExamples.add(new double[]{96, 95, -3, 1, 0, 0});
traningExamples.add(new double[]{102, 102, 3, 1, 0, 0});
traningExamples.add(new double[]{101, 100, -4, 1, 0, 0});
traningExamples.add(new double[]{97, 99, 2, 1, 0, 0});
traningExamples.add(new double[]{102, 96, -2, 1, 0, 0});
traningExamples.add(new double[]{3, 97, 105, 0, 1, 0});
traningExamples.add(new double[]{-5, 99, 97, 0, 1, 0});
traningExamples.add(new double[]{5, 105, 102, 0, 1, 0});
traningExamples.add(new double[]{0, 96, 101, 0, 1, 0});
traningExamples.add(new double[]{5, 101, 103, 0, 1, 0});
traningExamples.add(new double[]{0, 103, 98, 0, 1, 0});
traningExamples.add(new double[]{-1, 99, 95, 0, 1, 0});
traningExamples.add(new double[]{-4, 98, 96, 0, 1, 0});
```

Output Results:

Usages | Output - MLPClass (run)

```
output: 0.6839953832380323 0.3642024538651048 0.4001094931926O186
input: 3.0 97.0 105.0
output: 0.7057351234625642 0.33543245687028955 0.3660777840132758
input: -5.0 99.0 97.0
output: 0.6581158773973538 0.3866311980137963 0.3370429308216462
input: 5.0 105.0 102.0
output: 0.6065452744694959 0.4394678223552267 0.312281905431864
input: 0.0 96.0 101.0
output: 0.5537139396911628 0.49094971515415603 0.2910963364126225
input: 5.0 101.0 103.0
output: 0.5026116961376481 0.5385786839989182 0.27285714120695836
input: 0.0 103.0 98.0
output: 0.4556111860254098 0.580937017863102 0.2570472789667228
input: -1.0 99.0 95.0
output: 0.4139827115667947 0.6176641112638019 0.24323998286531912
input: -4.0 98.0 96.0
output: 0.3779717638921623 0.6490972937388158 0.23109268716330417
input: 0.0 101.0 104.0
output: 0.34717813780057244 0.6758940809817994 0.22032728310736738
input: 5.0 103.0 104.0
output: 0.3209310531145426 0.6987737494083545 0.21072913641661511
input: -5.0 4.0 4.0
output: 0.38618361104279814 0.6248748264376521 0.3206502693886243
input: 1.0 4.0 2.0
output: 0.3656514870912087 0.618115878875692 0.3229143470810782
input: 0.0 -4.0 2.0
output: 0.3795826670930059 0.5843807957618913 0.36554329913143646
input: -5.0 -5.0 1.0
output: 0.40441740636236184 0.5537517723981625 0.4082931377424167
input: -5.0 -2.0 1.0
output: 0.3945788979035574 0.5487902244378303 0.41128832012429584
input: 1.0 2.0 -1.0
output: 0.34680598931904877 0.5571782741568988 0.38754313000128393
BUILD SUCCESSFUL (total time: 0 seconds)
```

Result

- Can be used as supervised network, or for classification.

Questions?