

Học viện công nghệ bưu chính viễn thông
cơ sở tp HCM



Bài báo cáo

Đề tài:

Neural Network với Java

GVHD: *Nguyễn Thị Tuyết Hải*

Nhóm thực hiện

1	<i>Trần Nguyễn An Khang</i>	<i>N19DCCN087</i>
2	<i>Nguyễn Chí Hòa</i>	<i>N19DCCN061</i>

TP.Hồ Chí Minh

Mục lục

Bảng phân công	3
<input type="checkbox"/> <i>Source code:</i>	3
Giới thiệu	4
1. Mô hình neural network	5
1.1. Cách mà neural network học	5
1.2. Forward Propagation	5
1.3. Backpropagation	6
2. Neural network với Java	6
2.1. Activation function	6
2.2. Dataset.....	8
2.3. Class Connection	9
2.4. Class neuron.....	9
2.5. Class NeuralNetwork.....	11
2.6. Hàm Main	13
3. Mở rộng.....	15
<input type="checkbox"/> <i>Nhận xét:</i>	16
4. Tài liệu tham khảo	19

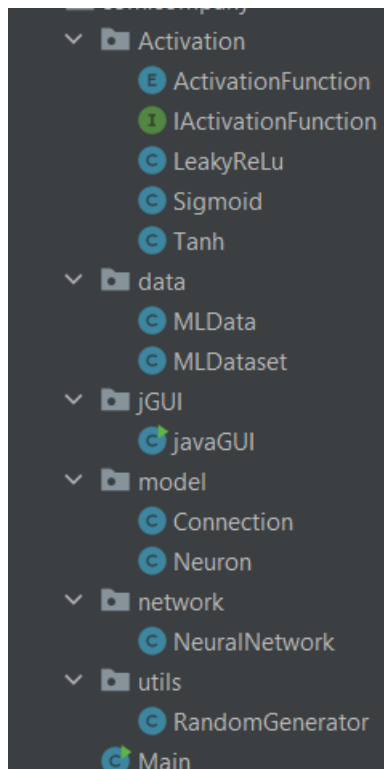
Bảng phân công

Họ và tên	Nội dung	Mức độ hoàn thành
Trần Nguyễn An Khang	Cấu trúc thuật toán, xây dựng mạng neural, forward và backward.	100%
Nguyễn Chí Hòa	Cấu trúc của ngôn ngữ Java , đồ họa.	100%

✓ **Phần mềm sử dụng:**

- IntelliJ IDE để code và run chương trình Java.

✓ **Cấu trúc chương trình:**



➤ **Source code:** <https://github.com/rocketvn/Newral-Net-with-Java>

Giới thiệu

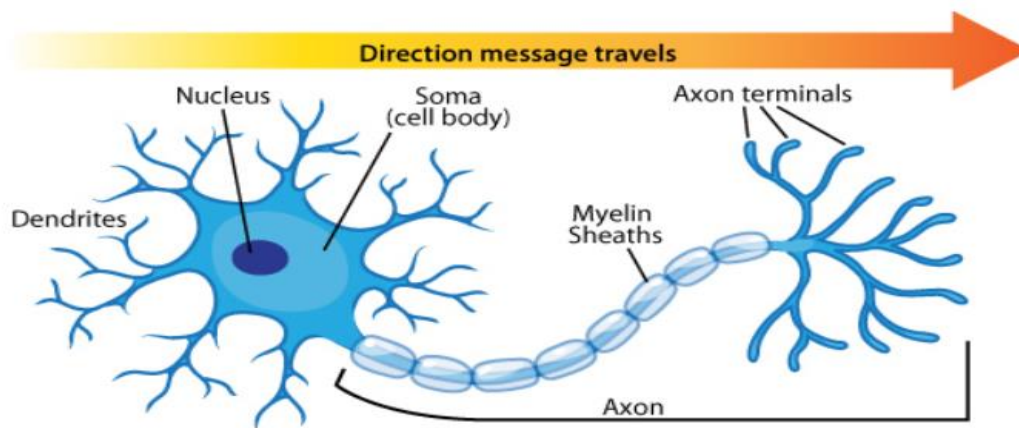
Con chó có thể phân biệt được người thân trong gia đình và người lạ hay đứa trẻ có thể phân biệt được các con vật. Những việc tưởng chừng như rất đơn giản nhưng lại cực kì khó để thực hiện bằng máy tính. Vậy sự khác biệt nằm ở đâu? Câu trả lời nằm ở cấu trúc bộ não với lượng lớn các nơ-ron thần kinh liên kết với nhau. Liệu máy tính có thể mô phỏng lại cấu trúc bộ não để giải các bài toán trên ???

Neural là tính từ của neuron (nơ-ron), network chỉ cấu trúc, cách các nơ-ron đó liên kết với nhau, nên neural network (NN) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là thành phần quan trọng nhất của não. Đầu chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron lại liên kết với tầm 10.000 nơ-ron khác.

Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

Neuron Anatomy



Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác.

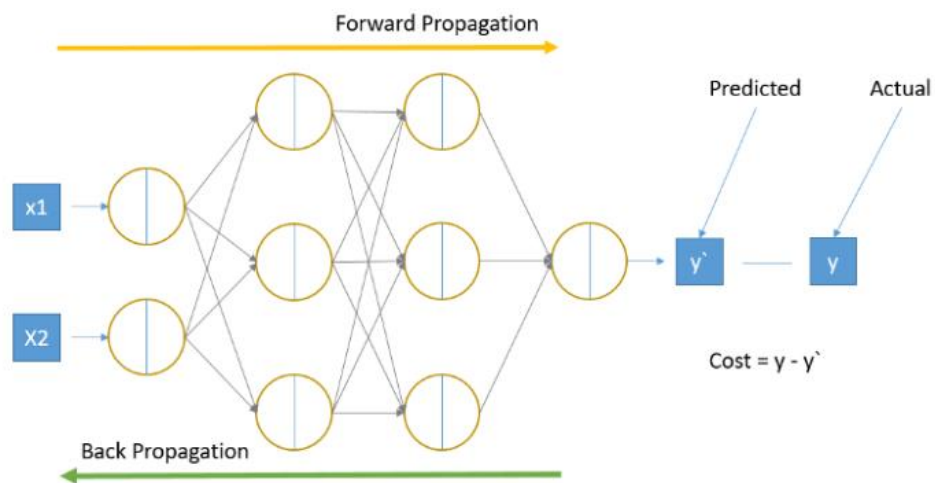
=> Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không. Tương tự các hoạt động của hàm sigmoid.

Tuy nhiên NN chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình đấy đi giải quyết các bài toán chúng ta cần.

1. Mô hình neural network

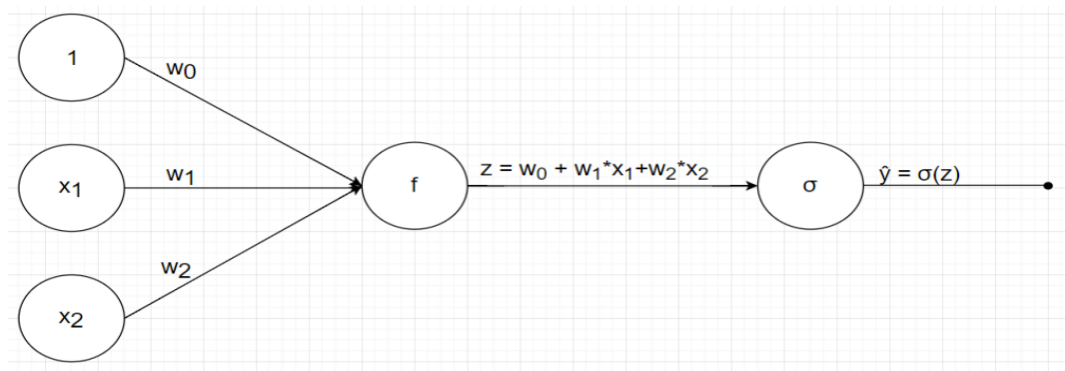
1.1. Cách mà neural network học

Neural Network học hỏi từ sự lan truyền Forward và Backward. Nó nhận đầu vào và chuyển tiếp chúng đến tất cả các nơ-ron được kết nối và cuối cùng nhận được kết quả đầu ra. Nếu kết quả đầu ra không chính xác thì nó sẽ ghép lại lỗi và cập nhật trọng số bằng cách sử dụng các dẫn xuất riêng cho đến khi đạt đến độ chính xác tối ưu do người dùng đặt. Lúc đầu, giá trị đầu ra sẽ rất khó hiểu, sau khi huấn luyện mạng nhiều lần sẽ nhận được đầu ra mong muốn của mình.



1.2. Forward Propagation

Trong forward propagation, các giá trị đầu vào được nhân với các giá trị trọng số của khớp thần kinh và sau đó chúng ta nhận được tổng có trọng số. Sau đó, áp dụng tổng có trọng số cho một hàm kích hoạt. Cho đến khi chúng ta đến lớp đầu ra, chúng ta chỉ chuyển tiếp các giá trị đầu vào từ nơ-ron đến nơ-ron. Lần đầu tiên, giá trị đầu ra sẽ trông khó hiểu.



1.3. Backpropagation

Mục đích của việc backpropagation là để giảm hàm chi phí. Trong backpropagation, tính toán lỗi ở lớp cuối cùng cũng là lớp đầu ra bằng cách sử dụng

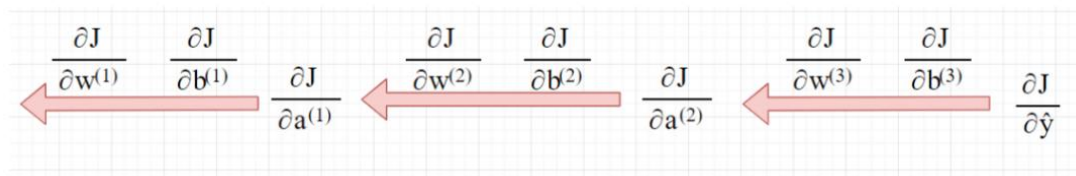
$$\text{error} = \text{actual} - \text{predicted}$$

Sau đó, tính toán gradient dựa trên lỗi ở lớp cuối cùng của neural network. Sau đó, sử dụng các gradient được tính toán này để cập nhật lớp ẩn. Sau đó, áp dụng việc cập nhật trọng số khớp thần kinh dựa trên các gradient này với learning rate để đi đến cực tiểu toàn cầu.

➤ Quá trình Forward propagation



➤ Quá trình Backpropagation



2. Neural network với Java

2.1. Activation function

Activation function là 1 thành phần rất quan trọng của neural-network. Nó quyết định khi nào thì 1 neuron được kích hoạt hoặc không. Liệu thông tin mà neuron nhận được có liên quan đến thông tin được đưa ra hay nên bỏ qua.

Activation function là 1 phép biến đổi phi tuyến tính mà chúng ta thực hiện đối với tín hiệu đầu vào. Đầu ra được chuyển đổi này sẽ được sử dụng làm đầu vào của neuron ở layer tiếp theo.

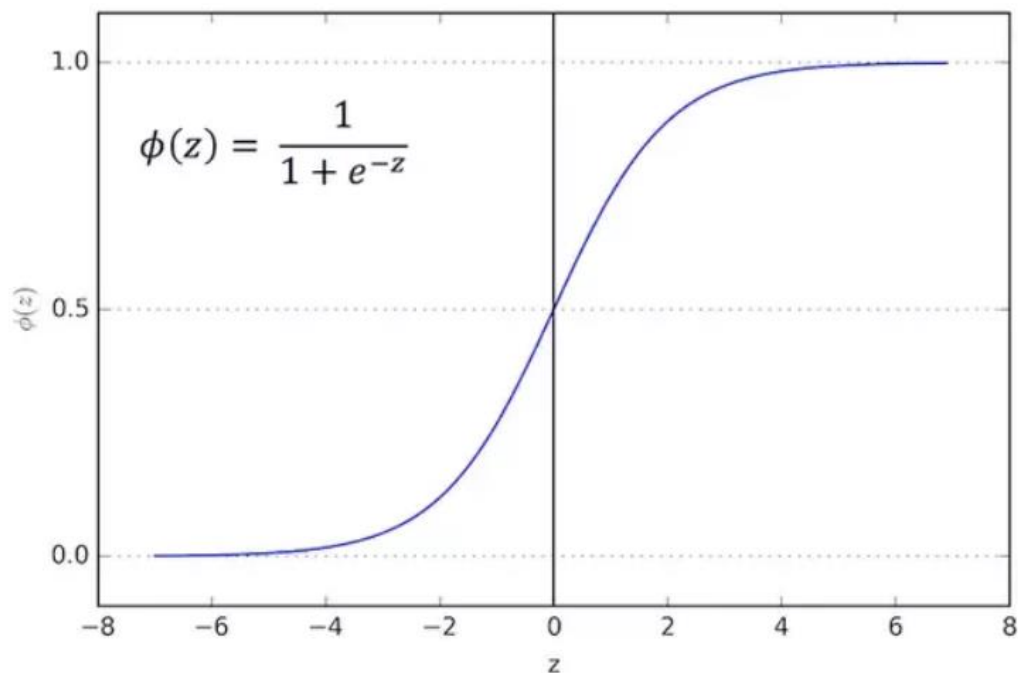
Nếu không có activation function thì weight và bias chỉ đơn giản như 1 hàm biến đổi tuyến tính. Giải 1 hàm tuyến tính sẽ đơn giản hơn nhiều nhưng sẽ khó có thể mô hình hóa và giải được những vấn đề phức tạp. Một mạng neuron nếu không có activation function thì cơ bản chỉ là 1 model hồi quy tuyến tính. Activation function thực hiện việc biến đổi phi tuyến tính với đầu vào làm việc học hỏi và thực hiện những nhiệm vụ phức tạp hơn như dịch ngôn ngữ hoặc phân loại ảnh là khả thi.

Activation function hỗ trợ backpropagation (tuyên truyền ngược) với việc cung cấp các lỗi để có thể cập nhật lại các weight và bias, việc này giúp mô hình có khả năng tự hoàn thiện.

❖ Trong project này em dùng hàm *Sigmoid* là activation

$$\text{Sigmoid: } f(x) = \frac{1}{1+e^{-x}}$$

❖ Hàm Sigmoid cho giá trị của hàm trong khoảng [0,1]



❖ Code

```

public class Sigmoid implements IActivationFunction {
    @Override
    public double output(double x) {
        return 1 / (1 + Math.exp(-x));
    }

    @Override
    // đạo hàm
    public double outputDerivative(double x) {
        return x * (1 - x);
    }
}

```

2.2. Dataset

- ❖ Class MLDataset để giữ input và label của dữ liệu truyền vào dưới dạng list.

```

public class MLDataset {
    private double[][] inputs;
    private double[][] targets;
    private List<MLData> data;

    public MLDataset() {
        data = new ArrayList<>();
    }

    public MLDataset(double[][] inputs, double[][] targets) {
        this.data = new ArrayList<>();
        this.inputs = inputs;
        this.targets = targets;
        for (int i = 0; i < this.inputs.length; i++) {
            this.data.add(new MLData(inputs[i], targets[i]));
        }
    }

    public void addMLData(MLData mlData) {
        this.data.add(mlData);
    }
}

```


2.3. Class Connection

- ❖ Class Connection có nhiệm vụ kết nối các neuron lại với nhau và giữ lại trọng số weights.

```
public class Connection {  
  
    private UUID connectionId;  
    private Neuron from;  
    private Neuron to;  
    private double synapticWeight;  
    private double synapticWeightDelta;  
  
    public Connection(Neuron from, Neuron to) {  
        this.connectionId = UUID.randomUUID();  
        this.from = from;  
        this.to = to;  
        this.synapticWeight = RandomGenerator.randomValue(-2, 2);  
    }  
  
    public void updateSynapticWeight(double synapticWeight) {  
        this.synapticWeight += synapticWeight;  
    }  
}
```

2.4. Class neuron

- ❖ Class neuron có nhiệm vụ khởi tạo ra các neuron với chức năng input, output và kết nối các neuron lại với nhau.
- ❖ Trong class neuron em khởi tạo function calculateOutput() để tính forward propagation theo công thức

$$Y = \text{Activation}((\text{weight} * \text{input}) + \text{bias})$$

- *Weight*: là trọng số của đường nối

```

public void calculateOutput() {
    this.outputBeforeActivation = 0.0;
    for (Connection connection : incomingConnections) {
        this.outputBeforeActivation += connection.getSynapticWeight() * connection.getFrom().getOutput();
    }
    this.output = activationFunction.output(x this.outputBeforeActivation + bias);
}

```

- Em khởi tạo biến outputBeforeActivation để tính giá trị (weight*input).
- Sau đó output sẽ bằng Activation(outputBeforeActivation) + bias) theo như công thức trên.

❖ Cũng trong class neuron function calculateGradient() để thực hiện Backpropagation từ việc tính đạo hàm .

```

public double loss(double target) {
    return target - output;
}

// Backpropagation
public void calculateGradient(double target) {
    this.gradient = loss(target) * activationFunction.outputDerivative(output);
}

public void calculateGradient() {
    this.gradient = outgoingConnections.stream().mapToDouble
        (connection -> connection.getTo().getGradient() * connection.getSynapticWeight()).sum()
        * activationFunction.outputDerivative(output);
}

```

- Em tính hàm loss function bằng cách lấy giá trị thực tức label trừ đi output tức giá trị dự đoán.
- Em dùng mapToDouble() để tính tổng bất kỳ thuộc tính nào của một lớp tùy chỉnh.
- Ở đây biến gradient được tính từ output .
- Cuối cùng là nhân với Activation function đạo hàm.

❖ Cuối cùng trong class neuron em khởi tạo hàm updateConnectin() để cập nhật trọng số cho các node layer sau khi tính toán xong .

```

public void updateConnections(double lr, double mu) {
    for (Connection connection : incomingConnections) {
        double prevDelta = connection.getSynapticWeightDelta();
        connection.setSynapticWeightDelta(lr * gradient * connection.getFrom().getOutput());
        connection.updateSynapticWeight(connection.getSynapticWeightDelta() + mu * prevDelta);
    }
}

```

- Hàm có 2 tham số là lr (learning rate) và mu (momentum) để điều chỉnh gradient.

2.5. Class NeuralNetwork

- ❖ Class này rất quan trọng nó giúp việc khởi tạo model cũng như quá trình model thực hiện forward và backward.
- ❖ Hàm init() để khởi tạo các layer của mạng neuron bao gồm các layer input, hidden và output.

```

public void init() {
    for (int i = 0; i < inputSize; i++) {
        this.inputLayer.add(new Neuron());
    }
    for (int i = 0; i < hiddenSize; i++) {
        this.hiddenLayer.add(new Neuron(this.inputLayer, activationFunction));
    }
    for (int i = 0; i < outputSize; i++) {
        this.outputLayer.add(new Neuron(this.hiddenLayer, activationFunction));
    }
    this.initialized = true;
    logger.info("Network Initialized.");
}

```

❖ Hàm train() để thực hiện việc load data để training model.

```
public void train(MLDataset set, int epoch) {
    if (!initialized){
        this.init();
    }
    logger.info(s: "Training Starting...");
    for (int i = 0; i < epoch; i++) {
        Collections.shuffle(set.getData());

        for (MLData datum : set.getData()) {
            forward(datum.getInputs());
            backward(datum.getTargets());
        }
    }
    logger.info(s: "Training Finished.");
}
```

- Tham số epoch để truyền vào số lần train.
- Em dùng shuffle() để xáo trộn dataset giúp tăng độ chính xác của model.
- Cuối cùng là thực hiện forward và backward.

❖ Kế đến là 2 hàm để khởi tạo việc forward và backward trên các layer.

<pre>private void forward(double[] inputs) { int i = 0; for (Neuron neuron : inputLayer) { neuron.setOutput(inputs[i++]); } for (Neuron neuron : hiddenLayer) { neuron.calculateOutput(); } for (Neuron neuron : outputLayer) { neuron.calculateOutput(); } }</pre>	<pre>private void backward(double[] targets) { int i = 0; for (Neuron neuron : outputLayer) { neuron.calculateGradient(targets[i++]); } for (Neuron neuron : hiddenLayer) { neuron.calculateGradient(); } for (Neuron neuron : hiddenLayer) { neuron.updateConnections(learningRate, momentum); } for (Neuron neuron : outputLayer) { neuron.updateConnections(learningRate, momentum); } }</pre>
---	---

- ❖ Cuối cùng là hàm predict() để dự đoán kết quả.

```
public double[] predict(double... inputs) {  
    forward(inputs);  
    double[] output = new double[outputLayer.size()];  
    for (int i = 0; i < output.length; i++) {  
        output[i] = outputLayer.get(i).getOutput();  
    }  
    logger.info("Input : " + Arrays.toString(inputs) + ", Predicted : " + Arrays.toString(output));  
    return output;  
}
```

2.6. Hàm Main

- ❖ Đầu tiên ta truyền dữ liệu vào.

```
private static final double[][] XOR_Input = {  
    {0, 0},  
    {0, 1},  
    {1, 0},  
    {1, 1}  
};  
  
private static final double[][] XOR_Label = {  
    {0},  
    {1},  
    {1},  
    {0}  
};
```

- Ở đây em dùng dữ liệu cổng logic XOR.
- ❖ Sau khi đã có dữ liệu ta sẽ khởi tạo model.

```
public static void main(String[] args) {

    NeuralNetwork neuralNetwork = new NeuralNetwork( inputSize: 2, hiddenSize: 10, outputSize: 1);
    neuralNetwork.init();
    neuralNetwork.setLearningRate(0.01);
    neuralNetwork.setMomentum(0.5);
    neuralNetwork.setActivationFunction(ActivationFunction.SIGMOID);

    MLDataset dataSet = new MLDataset(XOR_Input, XOR_Label);
    neuralNetwork.train(dataSet, epoch: 100000);

}
```

- Đầu tiên khởi tạo ra neuralNetwork từ class NeuralNetwork.
- Set tham số learning rate (lr) và momentum (mu).
- Cuối cùng là set activation function ở đây em cho là Sigmoid.

❖ Cuối cùng là predict dự đoán kết quả.

```
neuralNetwork.predict( ...inputs: 1, 1);
neuralNetwork.predict( ...inputs: 1, 0);
neuralNetwork.predict( ...inputs: 0.6, 0.5);
//neuralNetwork.predict(0, 0);
//neuralNetwork.predict(0.5,0.6);
```

❖ Và kết quả là:

```
- Network Initialized.
- Training Starting...
- Training Finished.
- Input : [1.0, 1.0], Predicted : [0.03639084411600611]
- Input : [1.0, 0.0], Predicted : [0.9636699210487027]
- Input : [0.6, 0.5], Predicted : [0.2944736415346451]
```

- Tỷ lệ chính xác chưa được cao chúng ta có thể chỉnh sửa tham số learning rate và momentum để tăng độ chính xác.

3. Mở rộng

❖ Một ví dụ đơn giản khi áp dụng vào thực tế.

- *Một ngân hàng có nhu cầu cho khách hàng vay, dựa trên dữ liệu về lương và thời gian làm việc của khách hàng.*

<i>Lương</i>	<i>Thời gian làm việc</i>	<i>Cho vay</i>
5	2	1
6	0.3	0
7	0.15	0
8	0.1	0
9	0.5	1
10	1	1

- 1 là cho vay, 0 là không cho vay

❖ Đầu tiên ta cho dữ liệu vào

```
private static final double[][] XOR_Input = {
    {5, 2},
    {6, 0.3},
    {7, 0.15},
    {8, 0.1},
    {9, 0.5},
    {10, 1}
};

private static final double[][] XOR_Label = {
    {1},
    {0},
    {0},
    {0},
    {1},
    {1}
};
```

❖ Kế đến ta build model

```
NeuralNetwork neuralNetwork = new NeuralNetwork( inputSize: 2, hiddenSize: 10, outputSize: 1);
neuralNetwork.init();
neuralNetwork.setLearningRate(0.01);
neuralNetwork.setMomentum(0.5);
neuralNetwork.setActivationFunction(ActivationFunction.SIGMOID);

MLDataset dataSet = new MLDataset(XOR_Input, XOR_Label);
neuralNetwork.train(dataSet, epoch: 100000);
```

- Model được xây dựng với 2 input, 10 hidden layer và 1 output đầu ra.
- Set learning rate (lr) là 0.01 và momentum (mu) là 0.5.
- Kế đến gọi hàm kích hoạt ở đây là SIGMOID.
- Train model với epoch là 100000 lần train.

❖ Cuối cùng là predict.

```
neuralNetwork.predict( ...inputs: 10, 1);
neuralNetwork.predict( ...inputs: 6, 0.3);
neuralNetwork.predict( ...inputs: 8.5, 0.3);|
//neuralNetwork.predict(0, 0);
//neuralNetwork.predict(0.5,0.6);
```

❖ Kết quả:

```
es\JetBrains\IntelliJ IDEA Community Edition 2021.2.3\lib\
- Network Initialized.
- Training Starting...
- Training Finished.
- Input : [10.0, 1.0], Predicted : [0.858614056819849]
- Input : [6.0, 0.3], Predicted : [0.030110637559084545]
- Input : [8.5, 0.3], Predicted : [0.47441466272481053]
```

➤ **Nhận xét:**

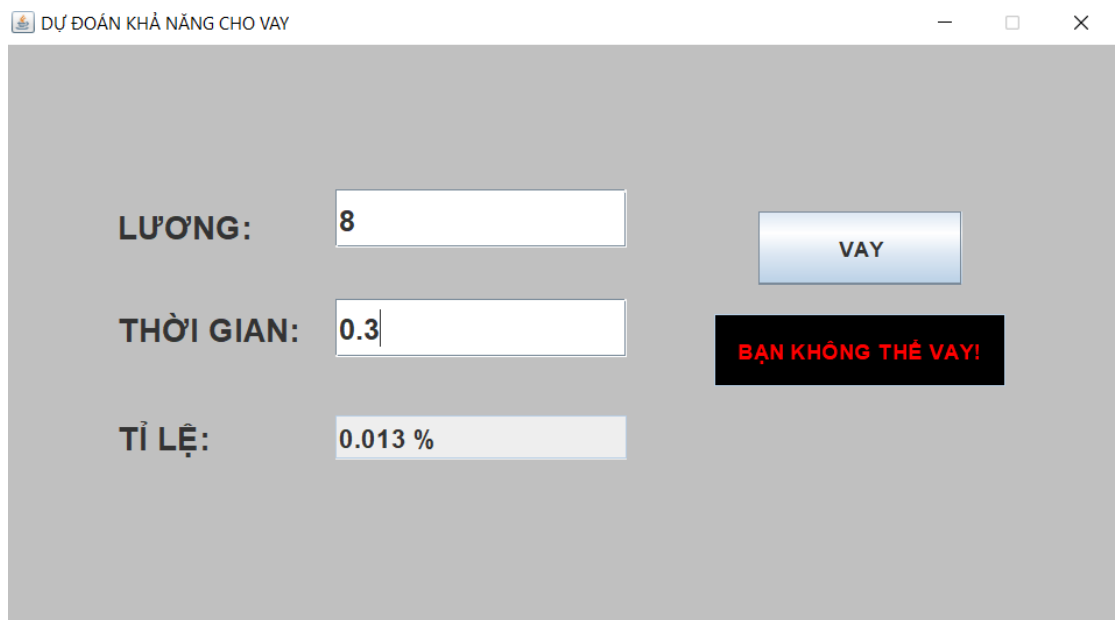
- Kết quả cho ra khá sát với kết quả trên dữ liệu.
- Độ chính xác của model vẫn chưa cao.
- Nếu kết quả lớn hơn 0.5 thì sẽ được vay và bé hơn 0.5 thì sẽ không được vay.
- Vậy nên với mức lương 8.5(triệu) và thời gian làm việc 0.3(3 tháng) thì sẽ không được cho vay.

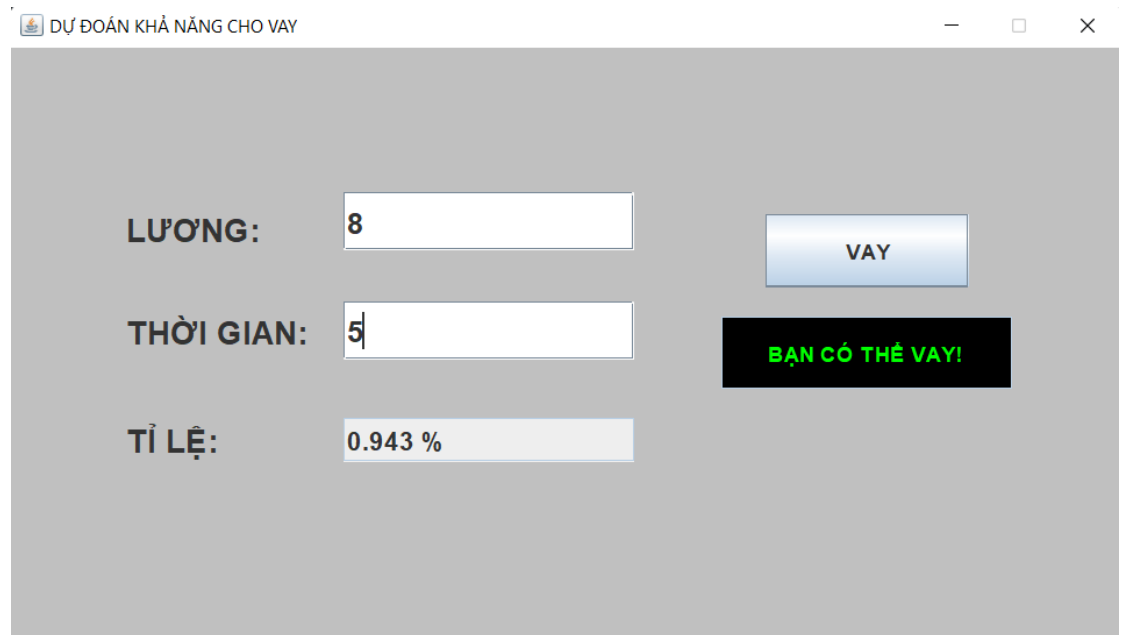
- ❖ Để thêm cho project thêm sinh động cũng như thực tế em có thêm vào một chút graphic vào.



- Ở đây em dùng thư viện Java Swing của Java để thiết kế GUI.

- ❖ Em thử nghiệm hai trường hợp predict tỉ lệ cho một khách hàng vay dựa trên lượng dữ liệu ở trên.





- Em đặt mốc là 0.5 , tức lớn hơn 0.5 thì sẽ được vay và bé hơn 0.5 thì sẽ không được vay.

```
if (num3 > 0.5) {  
    preTextField.setText("BẠN CÓ THỂ VAY!");  
    preTextField.setForeground(Color.green);  
} else {  
    preTextField.setText("BẠN KHÔNG THỂ VAY!");  
    preTextField.setForeground(Color.red);  
}
```

➤ **Note:** phần graphic nằm trong thư mục jGUI/javaGUI.java

4. Tài liệu tham khảo

<https://nttuan8.com/bai-3-neural-network/>

<https://viblo.asia/p/mang-neural-network-WAyK84zpKxX>

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

<https://medium.com/@zaxxio/build-neural-network-in-java-c3dde0ab8887>