**Letter Recognition using Deep Neural Nets with Softmax Units**

Yonggang Li (18043158)

Zoe Wang (18042783)

Rebecca Liu (17238744)

1. **Data Structure**

The basic element of the system is Neuron, a set of Neuron composes a Layer, and Layers are chained to form a NeuralNetwork. Figure 1 shows this architecture. With this architecture, layers can be added or dropped to make the network deeper or shallow.

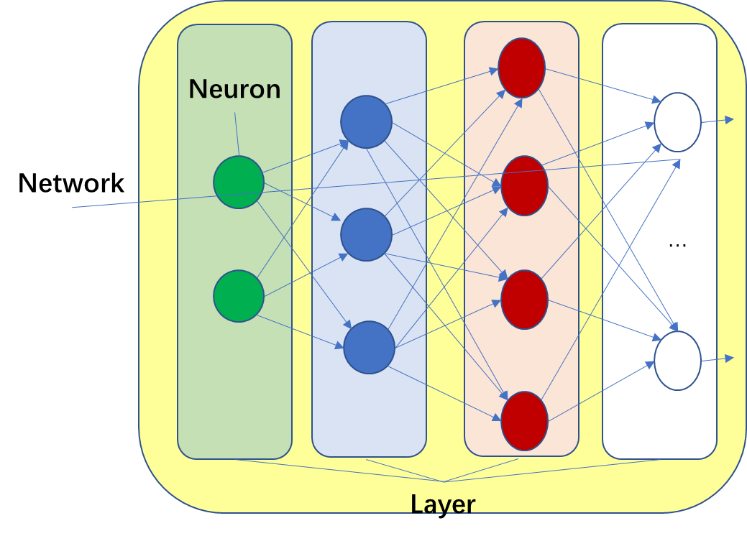


Figure 1 Data Structure

* 1. **Neuron**

There are two function pointers which refer to the activation function and its derivative being used.

typedef double (\*NeuronFunc)(double val);

class Neuron {

public:

Neuron(int connections);

void setDesiredOutput(double t);

// Activation functions, \*D means derivative of the function

static double sigmoid(double val);

static double sigmoidD(double val);

static double tanh(double val);

static double tanhD(double val);

static double relu(double val);

static double reluD(double val);

static void setActivationFunction(ActivationFunction af);

public:

int m\_connections;

double m\_o; //actual ouput

double m\_t; //desired output

double\* m\_w; //weights

//the activation function used, will be updated from UI via setActivationFunction

static NeuronFunc activate;

static NeuronFunc activateD;

};

* 1. **Layer**

Most of the methods are explained by their name, saveWeights and loadWeights are utilities for storing and loading its weights from the specified file.

class Layer

{

public:

Layer(int leftNeurons, int neurons, int rightNeurons);

virtual void calcOutput(Layer\* pLeft);

virtual void calcError(Layer\* pRight);

virtual void updateWeights(Layer\* pRight, double learningRate);

void copyWeights(Layer\* fromLayer);

void setDesiredOuput(double\* vals);

double\* getOutput();

void saveWeights(FILE\* pFile);

void loadWeights(FILE\* pFile);

public:

int m\_N, m\_rightN;

CArray<Neuron\*> m\_neurons;

Neuron\* m\_pBias;

double\* m\_err;

double\* m\_output;

private:

LayerType m\_type;

};

* 1. **NeuralNetwork**

class NeuralNetwork

{

public:

NeuralNetwork();

void initialise();

//others are omitted…

public:

CArray<Letter\_S> m\_train; //train patterns

CArray<Letter\_S> m\_test; //test patterns

private:

Layer\* m\_pInput;

Layer\* m\_pHidden0;

Layer\* m\_pHidden1;

Layer\* m\_pOutput;

double m\_learningRate;

int m\_maxEpochs;

Assess\_S m\_assess; //assess result

BOOL m\_preTrained; //need pretrain before training the network

}

* 1. **Input Patterns**

struct Letter\_S {

char symbol; // the character

double O[OUTPUT\_NEURONS];

double X[INPUT\_NEURONS];

void reset() {

symbol = 0;

memset(X, 0, sizeof(double)\*INPUT\_NEURONS);

if (SOFTMAX == 1){memset(O, 0, sizeof(double)\*OUTPUT\_NEURONS);}

else {

for (int i = 0; i < OUTPUT\_NEURONS; i++)

O[i] = 0.1;

}

}

};

* 1. **Model Assessment**

struct Assess\_S {

double trainSSE;

double trainMse;

double trainRatio; //good classification

double testSSE;

double testMse;

double testRatio;

int confusionMatrix[OUTPUT\_NEURONS][OUTPUT\_NEURONS];

};

* 1. **Patterns shuffle algorithm**

for (int i=0; i<pattern\_size; i++)

{

Letter\_S letter = m\_train.GetAt(i);

int index = rand() % (pattern\_size-i);

m\_train[i] = m\_train[index];

m\_train[index] = letter;

}

* 1. **Calculate Output**

Each Layer depends on its left layer to feed it

void Layer::calcOutput(Layer\* pLeft) {

if (pLeft == nullptr)

return;

int size = pLeft->m\_neurons.GetSize();

for (int i = 0; i < m\_N; i++) {

double sum = 0;

for (int j = 0; j < size; j++) {

sum += pLeft->m\_neurons[j]->m\_o \* pLeft->m\_neurons[j]->m\_w[i];

}

sum += pLeft->m\_pBias->m\_w[i];

m\_neurons[i]->m\_o = Neuron::activate(sum);

}

}

* 1. **Calculate Error**

The logics for calculating the error of output and hidden layers are different. The layer type, m\_type, is automatically set when constructing the network.

void Layer::calcError(Layer\* pRight) {

if (m\_type == LayerType::LT\_OUPUT) {

for (int out = 0; out < m\_N; out++) {

Neuron\* p = m\_neurons[out];

m\_err[out] = (p->m\_o - p->m\_t) \* Neuron::activateD(p->m\_o);

}

}

else if (m\_type == LayerType::LT\_HIDDEN) {

for (int i = 0; i < m\_N; i++) {

m\_err[i] = 0.0;

Neuron\* p = m\_neurons[i];

for (int j = 0; j < pRight->m\_neurons.GetSize(); j++) {

m\_err[i] += pRight->m\_err[j] \* p->m\_w[j];

}

m\_err[i] \*= Neuron::activateD(p->m\_o);

}

}

}

* 1. **Update Weights**

The layer depends on the errors from its right layer.

void Layer::updateWeights(Layer\* pRight, double learningRate) {

if (pRight == nullptr)

return;

for (int i = 0; i < m\_N; i++) {

Neuron\* p = m\_neurons[i];

for (int j = 0; j < pRight->m\_neurons.GetSize(); j++) {

p->m\_w[j] -= (learningRate \* pRight->m\_err[j]\* p->m\_o);

}

}

//bias

for (int out = 0; out < m\_N; out++) {

m\_pBias->m\_w[out] -= (learningRate \* pRight->m\_err[out]);

}

}

1. **Network Architecture**

As shown in Figure 2, this is a 4 layers network with 16 inputs, two 32 neurons hidden layers and 26 outputs. There are also 3 bias nodes for hidden and output layers respectively.

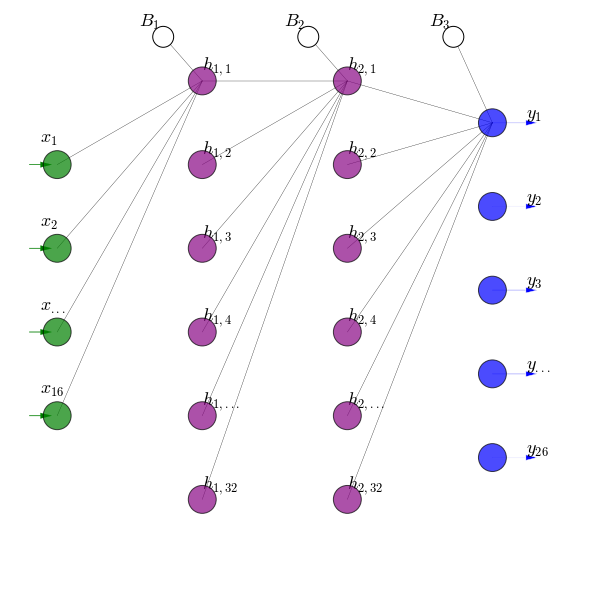


Figure 2 Network Architecture

* 1. **Activation Functions for Hidden Layers**

****Three activation functions, sigmoid, ReLU, and tanh have been tested. The weight adjustment formula is the same, the difference is formula for calculating error because the derivative of the activation function different.

* + 1. **Sigmoid**

****

D() =

* + 1. **ReLU**

*ma*x(0, val)

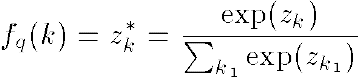
D() is 1.0 if > 0, otherwise 0

* + 1. **Tanh**

D() =

* 1. **Activation Functions for Output Layers**

Softmax is adopted for output layer, the error and weights for leading to the output are adjusted as below,

****

****

1. **Network Initialisation**

When loading patters, the input will be scaled to 0~1 and the output will be a vector with 26 elements and all are 0 except the one for the corresponding letter, as the pseudo code below shows,

letter.O[letter.symbol - 'A'] = 1

The weights are assigned randomly with a value between -1.0 and 1.0.

The activation function is ReLU initially unless changed from the user interface.

Learning rates are different for different activation functions,

const double SIGMOID\_LEARNING\_RATE = 0.01;

const double RELU\_LEARNING\_RATE = 0.0001;

const double TANH\_LEARNING\_RATE = 0.0005;

1. **Pseudo Code**
   1. Build network

m\_pInput = new Layer(0, INPUT\_NEURONS, HIDDEN\_NEURONS);

m\_pHidden0 = new Layer(INPUT\_NEURONS, HIDDEN\_NEURONS, HIDDEN\_NEURONS);

m\_pHidden1 = new Layer(HIDDEN\_NEURONS, HIDDEN\_NEURONS, OUTPUT\_NEURONS);

m\_pOutput = new SoftmaxLayer(HIDDEN\_NEURONS, OUTPUT\_NEURONS, 0);

* 1. Train data shuffling

The shuffle algorithm is very simple, generate a random number between the current pattern index ***i*** and the total pattern size, swap the pattern at ***i*** with the generated index.

int pattern\_size = m\_train.GetSize();

for (int i=0; i<pattern\_size; i++)

{

Letter\_S letter = m\_train.GetAt(i);

int index = rand() % (pattern\_size-i);

m\_train[i] = m\_train[index];

m\_train[index] = letter;

}

* 1. Feed forward

pInput->setInput(input);

m\_pHidden0->calcOutput(m\_pInput);

m\_pHidden1->calcOutput(m\_pHidden0);

m\_pOutput->calcOutput(m\_pHidden1);

* 1. Back propagation

m\_pOutput->calcError(nullptr);

m\_pHidden1->updateWeights(m\_pOutput, m\_learningRate);

m\_pHidden1->calcError(m\_pOutput);

m\_pHidden0->updateWeights(m\_pHidden1, m\_learningRate);

m\_pHidden0->calcError(m\_pHidden1);

* 1. Set Activation Function

if (af == ActivationFunction::AF\_RELU) {

Neuron::activate = relu;

Neuron::activateD = reluD;

}else if (af == ActivationFunction::AF\_SIGMOID) {

Neuron::activate = sigmoid;

Neuron::activateD = sigmoidD;

}else{

Neuron::activate = tanh;

Neuron::activateD = tanhD;

}

By this way, Layer code for calling the activation function and its derivative functions can be simple, for example

m\_err[out] = (p->m\_o - p->m\_t) \* Neuron::activateD(p->m\_o);

m\_err[i] \*= Neuron::activateD(p->m\_o);

1. **User Interface Design and Usage**

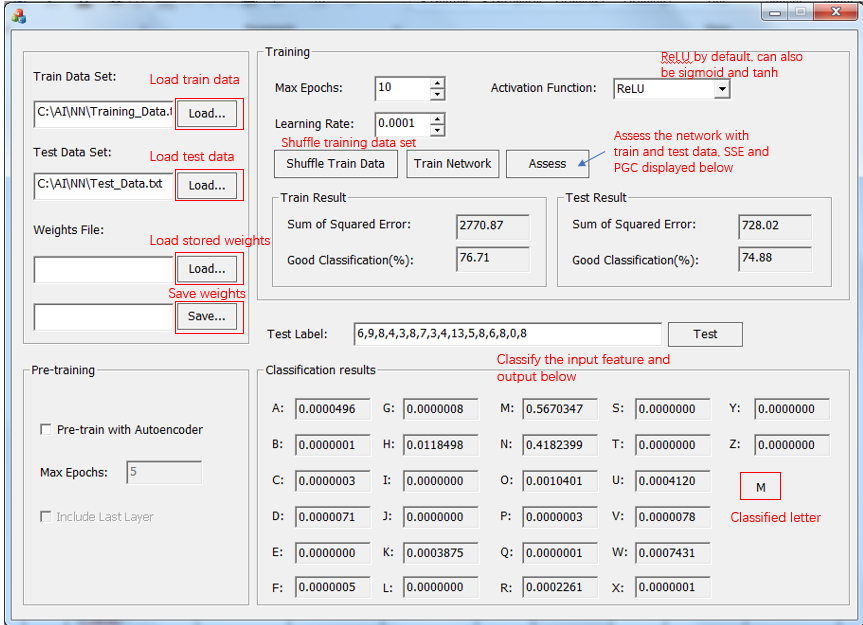


Figure 3 User Interface

* 1. **Use case one**

1. Load train data
2. Load test data
3. Set “Max Epochs”
4. Set “Learning Rate” or just use the default
5. Select “Activation Function”
6. Click “Shuffle Train Data”
7. Click “Train Network” and waiting, the SSE and PGS will be populated on the UI once training completed
8. Copy 16 features of a letter and paste into “Test Label”, then click Test. The “Classification Result” will be updated and the recognized letter will displayed at the right-bottom as shown above.
9. You may adjust some parameter and click “Train Network”, the network will be trained continuously based on the current state.
10. Click “Save” in weights file part to save the network parameter and weights to local file.
    1. **Use case two**
11. Assume you have saved the network in 5.1
12. Load train and test data
13. Load weights file
14. Click “Assess” and the SSE and PGS will be populated on the UI
15. When accessing completed, the application will ask if you want to save the confusion matrix, click “Yes” and select a location to save the file if you agree.
16. You may adjust some parameter and click “Train Network”, the network will be trained continuously based on the current state.
17. Save the network

1. **Result**
   1. **Top 3 performance network**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Architecture**  **(Softmax as output for all)** | **Learning Rate** | **Initial Weights** | **Trained Weights** | **Performance on**  **Training Data** | | **Performance on Validation Data** | | **Max.**  **Training epochs** |
| **MSE** | **PGC** | **MSE** | **PGC** |  |
| 1 | 16-32-32-26-x (Relu) | 0.0001 | init-relu.txt | 5000-relu.txt | 0.085 | 88.61 | 0.10 | 85.75 | 5000 |
| 2 | 16-32-32-26-x (tanh) | 0.0005 | init-tanh.txt | 5000-tanh.txt | 0.049 | 93.62 | 0.079 | 89 | 5000 |
| 3 | 16-32-32-26-x (sigmoid) | 0.01 | init-sigmoid.txt | 5000-sigmoid.txt | 0.014 | 98.38 | 0.047 | 93.83 | 5000 |

* 1. **Network performance**

Figure 3 shows PGC for different activation function on training and testing data sets. Among the three groups, ReLU, sigmoid, and tanh, given the same epochs, sigmoid gives the highest PGC, (98.38%) followed by tanh (93.62%) and ReLU is the lowest (88.61%). Within each group, PGC for train is higher than it for test and most importantly PGC on train will continuously increase as we train more times, give bigger epochs in other words. But PGC for test won’t grow and even drop down. This is reasonable, training more times results in overfitting the training data and therefore the neural network performs bad on the testing data.

We can also see from the figure, as epochs increase from 100 to 500, PGC grows significantly, after 500 the growing speed slow down, and after 1000, PGC is almost stuck or gets little increase with big epochs.

Figure 4 shows that SSE decrease as the epochs get bigger. It also echoes Figure 3, the higher the PGC, the lower the SSE. For epochs from 100 to 500, SSE drops quickly for all activation functions on both training and testing data, after 500, the dropping speed slows down and after 1000 it becomes very slowly. Similarly, SSE keeps dropping on training data but might fluctuate on testing data. See “Performance.xlsx” for details.

Figure 3 Percent of Good Classification

Figure 4 Sum of Squared Errors

* 1. **Confusion matrix**

Check the following files

* ConfusionMatrix-relu.txt
* ConfusionMatrix-sigmoid.txt
* ConfusionMatrix-tanh.txt

1. **Extra Task**

Extra task for this larger group are developed in Python and submitted as a Jupyter Notebook as well as html page for quick review.

1. **Testing File List**
   1. Initialize weights files:

init-relu.txt

init-sigmoid.txt

init-tanh.txt

* 1. Trained weights files

The following weights files can be loaded directly and access the train and test data.

5000-relu.txt

5000-sigmoid.txt

5000-tanh.txt

* 1. Confusion Matrix files

ConfusionMatrix-relu.txt

ConfusionMatrix-sigmoid.txt

ConfusionMatrix-tanh.txt

* 1. Epochs, PGC and SSE

Performance.xlsx

1. **References**

[1] 159740 course lectures

[2] 159740 course assignment 2 start-up code

[3] Fisher–Yates shuffle <https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle>