НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

БАКЪЛАВЪРСКИ ФАКУЛТЕТ

ДЕПАРТАМЕНТ "ИНФОРМАТИКА"

ПРОГРАМА ИНФОРМАТИКА

Проект по невронни мрежи

КУРСОВА РАБОТА

ТЕМА: Невронна мрежа за разпознаване и класифициране на ръкописни цифри

НА СТУДЕНТА: Илиян Делчев Додеков

ФАК. № 50129

Пролетен Семестър

2014-2015 г.

Съдържание

Въведение

I. Задание

Проектът представлява реализирана невронна мрежа с три слоя за разпознаване и класифициране на ръкописни цифри. Използван е Java, като език за имплементацията.

Особености на мрежата:

- Възможност за подаване на входни данни от файлове. Всяко изображение е 28x28 пиксела със стойност от 0 до 255.
- За обучение и тестване на мрежата е ползвана MNIST (Modified National Institute of Standards and Technology) базата от ръкописни цифри.
- Три слоя – един входен, един скрит и един изходен. 784 неврона във входния слой; 387 неврона в скритият слой; 10 неврона в изходния слой. Възможност за промяна на броя на невроните посредством конфигурационен файл.
- Възможност за обучение на мрежата по алгоритъма за обратно разпространение на грешката (backpropagation).
- Активационната функция е сигмоидална ($f(x)=1/(1+e^{-x})$))
- Коефицент на обучение – 0,5, с възможност за промяна посредством конфигурационен файл.
- Възможност за записване на вече обучена мрежа във файл, за по-нататъшна употреба.
- Възможност за визуализиране на квадратична грешка
- Възможност за работа с тренировъчни, валидационни и тестови данни.
- Възможност за прекратяване на обучението против претрениране (overfitting), посредством валидационен набор от данни.
- Възможност за определяне дали тренировъчните данни да бъдат подавани последователно или разбъркано.
- Възможно за задаване на seed за случайни числа.

II. Резултати

В MNIST базата от данни има 3 типа изображания – 60 000 броя, които се ползват за трениране на мрежата, 5 000 броя за тестване на мрежата и още 5 000 броя валидационни данни, които предпазват мрежата от претрениране. Важно е да се отбележи, че в трите типа изображения няма цифри, които се повтарят.

С горепосочената мрежа, след около 10 цикъла на обучение с 60 000 изображения, е реализиран коефициент на грешка **9,68%**. След този момент мрежата започва да се претренира. Ако проверката за претрениране се премахне, реализирания коефициент на грешка стига до около **1,94%**.

Какво е невронна мрежа

Невронната мрежа е модел за обработка на информация, вдъхновен от изучаването на биоелектричните мрежи в мозъка на човека и животните, образувани от неврони и техните синапси. В наши дни учените често наричат изкуствените невронни мрежи просто невронни мрежи.

Математическия аналог на биологичната невронна мрежа представлява множество от взаимносвързани прости изчислителни елементи (неврони). Всеки неврон приема сигнали от другите (под формата на числа), сумира ги, като сумата минава през активационна функция (най-често използваната е сигмоидалната функция $y=f(x)=1/(1+e^{-x})$), и така определя своята активация (степен на възбуда), която се предава по изходящите връзки към другите неврони. Всяка връзка има тегло, което умножавайки се със сигнала, определя неговата значимост (сила). Теглата на връзките са аналогични на силата на синаптичните импулси, предавани между биологичните неврони. Отрицателна стойност на теглото съответства на подтискащ импулс, а положителна - на възбуждащ.

В невронната мрежа обикновено винаги съществуват входен и изходен слой от неврони, във входния се въвежда информацията към мрежата, след това сигналите от входните неврони преминават през един или няколко слоя от междинни (скрити) неврони, според топологията на невронната мрежа, като сигналите накрая стигат до изходния слой, откъдето се чете получената информация.

Математически е доказано, че всяка невронна мрежа с поне един скрит слой от достатъчно на брой неврони между входния и изходния слой, може да моделира поведението на всяка съществуваща функция.

Теглата на връзките между невроните определят функционалността и поведението на невронната мрежа. За да бъде една невронна мрежа използваема и приложима към даден проблем, тя трябва да бъде предварително изучена.

Изучаването на една невронна мрежа се постигне чрез промяна на теглата на връзките между невроните и се осъществява чрез правила, които определят как да се променят тези тегла. Най-разпространеното сред тях е метода на обратното разпространение на сигнал за грешка (back-propagation), където за всеки изходен неврон се изчислява разликата от желаното му поведение, като се формира сигнал за грешка, който се движи назад към входния слой и по пътя си променя теглата на връзките така, че при следващата активация на мрежата грешката да бъде по-малка от сегашната. Този начин на "обучение" на мрежата обаче води до "забравяне"- мрежата бъде обучена да разпознава един елемент и впоследствие той не се повтори във входните данни, мрежата "забравя" този елемент.

MNIST база от ръкописни изображения на цифри

MNIST (Modified National Institute of Standards and Technology) базата от ръкописни изображения на цифри представлява набор от събрани цифри, написани с ръкописно от хора, за целите на машинното обучение. Базата представлява набор от 60 000 тернировъчни изображения и 10 000 тестови изображения. Всички тези изображения са нормализирани да бъдат с еднакъв размер и центрирани на базата на тежестта на изображението. Всяко от тях е 28x28 пиксела и представлява цифра от 0 до 9. Общо това прави 784 пиксела за всяко изображение. Важно е да се отбележи, че набора от тестови и тренировъчни цифри е различен и едното не е част от другото. Идеята е тренировъчният набор да се ползва за обучение, докато тестовият набор за проверка на точността и калкулиране на грешка, след като обучението приключи.

Повече за базата от изображения може да бъде прочетено на офицялния и сайт - http://yann.lecun.com/exdb/mnist .

Някой от изображенията в базата изглеждат по следния начин:

Всичките тестови и тренировъчни изображения са записани във 2 файла (общо 4). Така четенето става по-бързо и не се забавя допълнително обучителния процес. Във всеки комплект от файлове, единият съдържа пикселите на всички изображения, а другия – коя е цифрата. Това означава, че за всеки 784 пиксела в единия файл, има 1 цифра в другия файл (отношението е 1:784).

Всеки от файловете е бинарен и има следната структура:

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

Стойностите варират между 0 и 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Пикселите имат стойност от 0 до 255. 0 означава бяло а 255 означава черно.

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

Стойностите варират между 0 и 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  10000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Пикселите имат стойност от 0 до 255. 0 означава бяло а 255 означава черно.

Имплементираната мрежа не е задължително да работи точно с тези файлове, но е важно да се спазва структурата.

Типове изображения

Както вече беше споменто, мрежата работи с 3 типа изображания – 60 000 броя, които се ползват за трениране на мрежата, 5 000 броя за тестване на мрежата и още 5 000 броя валидационни данни, които предпазват мрежата от претрениране.

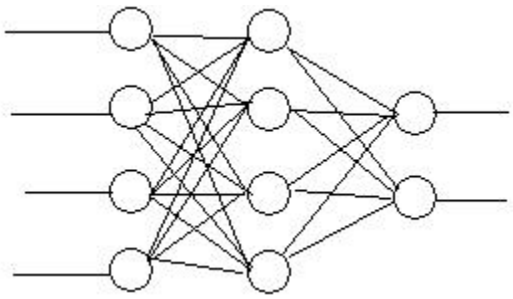
Тренировъчен тип
Ползват се за регулиране на теглата в мрежата.

Валидационен тип
Ползват се за минимизиране на претренирането. С този набор не се регулират тежестите, а просто се проверява, че увеличението на точността на мрежата над тренировъчния набор също увеличава и точността над валидационния набор (с който мрежата не е тренирана). Ако точността на мрежата над тренировъчния набор расте, но точността на мрежата над валидационния набор намалява или остава същата, значи мрежата се претренира и обучението трябва да спре.

Тестови тип
Ползват се само за тестване на вече трениран продукт за да се потвърди, че мрежата работи добре.

Реализация

Както вече беше споменато, една невронна мрежа се състои от няколко слоя (в случая 3), като всеки слой има брой неврони в себе си. Невроните от един слой са свързани с всъчки неврони от следващия слой. Входа на мрежата се вкарва във входния слой, а резултата от мрежата се взима от изходния слой.

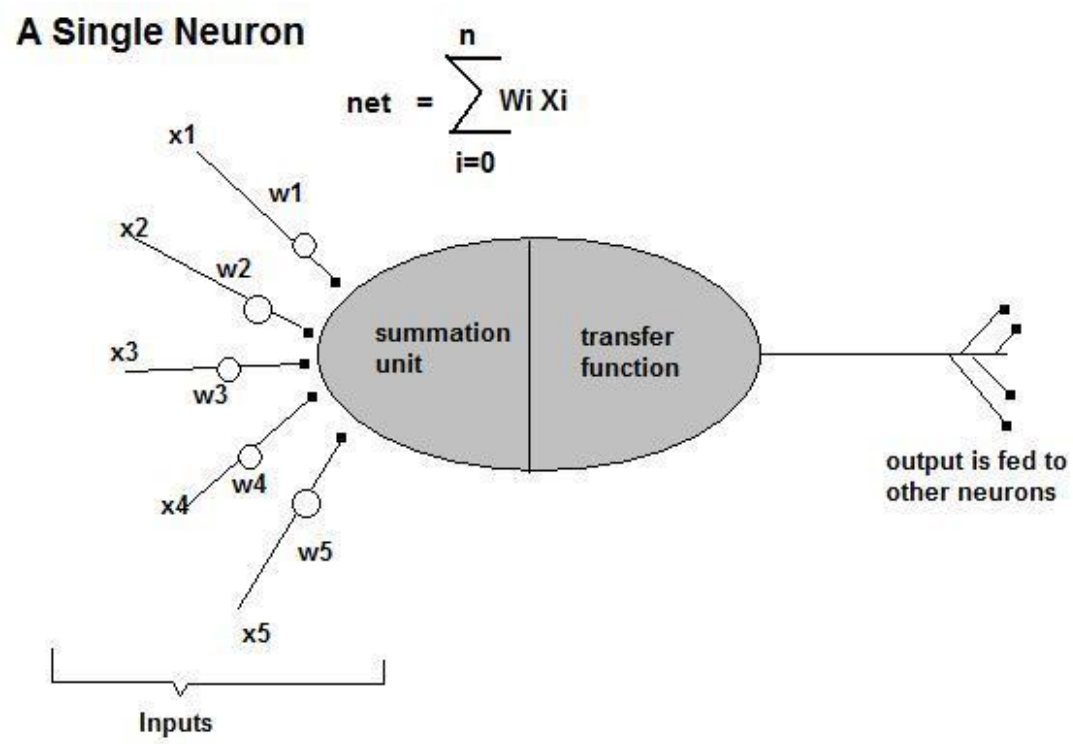


Една невронна мрежа може да бъде обучава с примери. Това включва подаване на вход и очавкан изход. На базата на грешката коефицентите се променят, така че да има по коректи резултати. Този процес се нарича трениране.

След като мрежата бъде обучена, за да се оперира с нея, е нужно да се подаде вход и мрежата да изчисли изхода. Изхода, разбира се, не винаги ще бъде коректен, но при правилно обучение, ще бъде сведен до минимум.

I. Изкуствен неврон

Един неврон, който се ползва е една изкуствена невронна мрежа, много си прилича по структура на билогичен неврон.



Неврона се състои от различен брой входове, суматор (summation unit), активационна функция (transfer function). Изхода на неврон от своя страна може да бъде предаден като вход на други неврони. Важно е да се отбележи и че за всеки входен сигнал има асоциирана съответна тежест.

➢ Суматор

За да се изчисли общия входен сигнал, суматора първо трябва да изчисли сбора на всеки входен сигнал, умножен със съответстващата му тежест. Също така обаче, всеки неврон има и прагов елемент, който променя общия входен сигнал. Праговия елемент поначно се инициализира със случайна стойност при създаване на невронната мрежа. Тежестите и праговия елемент се променят при тренирането на мрежата.

$$y_i = \sum_j W_{i,j} \quad x_j$$

Където:
y_i: сумиран входен сигнал
W_{ij}: входна тежест
X_j: входен сигнал

```
    public double findNetValue(NeuronConnections inputs, double bias) {
```

```
        double sum = bias;
        for(INeuron neuron: inputs.keySet()) {
            sum += inputs.get(neuron) * neuron.getOutputValue();
        }
        return sum;
    }
```

➢ Активационна функция

Активационната функция представлява тривиална функция, която ползва общия входен сигнал, изчислен от суматора, за да генерира изходен сигнал. Изхода от активационната функция се предава като входен сигнал на невроните в следващия слой.

В тази имплементация е ползва на сигмоидалната функция. Тази функция приема стойността генерирана от суматора и произвежда стойност между 0 и 1.

$$out = \frac{1}{(1 + e^{-sum})}$$

Където:
out: изход на неврон
sum: сумиран входен сигнал

```
    public double activation(double value) {
        return (1 / (1 + Math.exp(value * -1)));
    }
```

II. Структура на мрежата

Реализираната мрежа е разделена на 3 слоя – 1 входен, 1 скрит и 1 изходен. Невроните от входния слой са свързани с невроните от скритият слой, а невроните от скритият слой са свързани с невроните от изходния слой. Входния слой има 784 неврона, скритият слой има 387 неврона, а изходния слой има 10 неврона.

Броя на невроните във входния слой е равен на броя пиксели на всяко изображение. Входните неврони могат да имат стойност от 0.0 до 1.0. Тъй като стойността на всеки пиксел е между 0 и 255, преди стойността на пиксела да се подаде на входния неврон, тя е разделена на 255, за да може да се вмести в допустимите раници.

Броя на невроните в изходния слой е равен на броя възможни цифри (10). В зависимост от това коя е разпозната цифра, съответният неврон трябва да има

стойност клоняща към 1.0, а останалите трябва да имат стойност клоняща към 0.0. Например, ако резултата е 0, това означава че първият неврон ще има стойност близка до 1.0.

Броян на невроните в скрития слой е избран произволно и може да бъде променян, но е следвано правилото:

```
брой неврони в скрития слой = (брой неврони във входния слоя –
брой неврони в изходния слой)/2
```

III. Обучение на мрежата

В тази секция се разглежда как се обучава реализираната невронна мрежа. В тази имплементация е ползван алгоритъма за обратно разпространение на грешката.

Важно е да се отбележи, че при първоначалното създаване на невронната мрежа, тежестите и праговите елементи се инициализират със случайни стойности.

Обучение на невронна мрежа представлява постъпковата промяна на тежестите и праговите елементи, за да може да се постигат по-акуратни резултати.

Един тренировъчен цикъл се състои от 6 стъпки.

➢ Стъпка 1: Инициализиране на входовете

За да се инициализа входа на мрежата е нужно просто статично да се инициализират изходите на входния слой.

```
int i = 0;

for(INeuron neuron : this.getInputLayer()) {
    neuron.setOutputValue(t.getInputs().get(i));
    i++;
}
```

➢ Стъпка 2: Намиране на изхода

По-горе е описано как се пресмята изхода от един неврон. Изхода на невроните от входния слой се ползва за вход на невроните в скрития слой. Аналогично, изхода на невроните от скрития слой се ползва за вход на невроните в изходния слой. Изхода на невроните от изходния слой е и изхода на мрежата.

$$out = \frac{1}{(1 + e^{-\sum_j w_{i,j} x_j})}$$

Където:
out: изход на неврон
W_{ij}: входна тежест
X_j: входен сигнал

```
            NeuronLayer nl;

            for(int count = 1; count < layers.size(); count++) {
                nl = layers.get(count);
                for(INeuron neuron : nl) {
                    neuron.updateOutput();
                }
            }
```

```
    public void updateOutput() {
        double netValue = strategy.findNetValue(inputs, bias);
        output = strategy.activation(netValue);
    }
```

Както беше споменато, функцията `updateOutput()` първо се пресмята общата входна стойност и после минава през активационната функциия.

Функцията на суматора изглежда по следния начин:

$$y_i = \sum_j W_{i,j} \quad x_j$$

Където:
y_i: сумиран входен сигнал
W_{ij}: входна тежест
X_j: входен сигнал

```
    public double findNetValue(NeuronConnections inputs, double bias) {
        double sum = bias;
        for(INeuron neuron: inputs.keySet()) {
            sum += inputs.get(neuron) * neuron.getOutputValue();
        }
        return sum;
    }
```

Активационната функция изглежда по следния начин:

$$f(sum) = \frac{1}{(1 + \ e^{-sum})}$$

Където:
f(sum): изход на неврон

sum: сумиран входен сигнал

```
    public double activation(double value) {
        return (1 / (1 + Math.exp(value * -1)));
    }
```

➢ Стъпка 3: Намиране на грешката (делтата)

В тази стъпка се пресмята грешката. Грешка (делта) може да се дефинира, като разликата между получения резултат и очаквания резултат. Например, когато за пръв път пресметнем изхода на мрежата, най-вероятно ще бъде тотално грешен.

За да се пресметне грешката:

1. Първо се пресмята грешката на всеки неврон в изходния слой
2. Пресметнатата делта се ползва за да се пресметне делтата на невроните в скрития слой.
3. Пресметнатата делта от скрития слой се ползва за да се пресметне делтата на невроните във входния слой.

За пресмятането на грешката на един неврон в изходния слой, ползваме следната формула:

$$\delta_i = f'(sum_j)(t_j - y_j)$$

Където:
$\boldsymbol{\delta_i}$: грешка на даден неврон
$\boldsymbol{y_j}$: изходен сигнал на даден неврон
$\boldsymbol{t_j}$: очакван резултат за даден неврон от изходния слой
$\boldsymbol{f'(sum_j)}$**:** първа производна на активационната функция

```
    public double findDelta(double output, double errorFactor) {
        return output * (1 - output) * errorFactor;
    }
```

```
            i = 0;
            for(INeuron neuron : this.getOutputLayer()) {
                neuron.updateDelta(t.getOutputs().get(i) - neuron.getOutputValue());
                i++;
            }
```

За намирането на фактора на грешка в скрития слой обаче, сметките са малко по-различни:

1. Първо, делтата на всеки неврон, към който текущия неврон е свързанн, се умножава по тежестта на тази връзка

2. Тези произведения се събират за да се делтата за даден неврон в скрития слой.

$$\delta_i = f'(sum_j) * \sum_j \delta_j w_{kj}$$

Където:
$\boldsymbol{\delta_i}$: грешка на даден неврон
$\boldsymbol{f'(sum_j)}$**:** първа производна на активационната функция
$\boldsymbol{\delta_j}$: делта на неврон от следващия слой.
$\boldsymbol{w_{kj}}$*:* тежест на връзка до неврон от следващия слой

```
            for(i = layers.size() - 2; i>=1; i--) {
                NeuronLayer currentLayer = layers.get(i);

                for(INeuron neuron: currentLayer) {
                    double errorFactor = 0;
                    for(INeuron connectedNeuron : neuron.getForwardConnections()) {
                        errorFactor += connectedNeuron.getDeltaValue() *
connectedNeuron.getInputs().get(neuron);
                    }
                    neuron.updateDelta(errorFactor);
                }
            }
```

След края на стъпка 3, имаме делтата на всички неврони в мрежата.

➢ Стъпка 4: Промяна тежестите и праговите коефиценти

След калкулирането на грешката на всички неврони във всички слоеве, трябва да бъдат коригирани тежестите и праговите коефиценти, на базата на същата тази грешка. Всеки неврон трябва да коригира повече от една тежест, тъй като има различна тежест за всяка връзка.

```
            for(i = 1; i<layers.size(); i++) {
                for(INeuron neuron : layers.get(i)) {
                    neuron.updateFreeParams();
                }
            }
```

Функцията `updateFreeParams()` калкулира новите стойности на праговия елемент и тежестите, на базата на делтата която получихме в стъпка 3. Праговия елемент представлява тежест с коефицент на сигнала равен на 1.

```
    public void updateFreeParams() {
        strategy.findNewBias(bias, delta);
        strategy.updateWeights(inputs, delta);
    }
```

За намиране на нов прагов елемент:

$$x_0 = \delta_i * 1 * \varepsilon$$

Където:
$\boldsymbol{\delta_i}$: грешка на даден неврон
$\mathbf{x_0}$: нова стойност на праговия елемент
$\boldsymbol{\varepsilon}$: коефицент на обучение

```
    public double findNewBias(double bias, double delta) {
        return bias + Constants.LEARNING_RATE * 1 * delta;
    }
```

За намиране на нова тежест на невроните:

$$newW_{ij} = w_{ij} + \delta_j * \varepsilon * y_j$$

Където:
$\boldsymbol{\delta_j}$: грешка на неврона
$\mathbf{newW_{ij}}$: нова стойност на тежестта
$\mathbf{w_{ij}}$: стара стойност на тежестта
$\boldsymbol{\varepsilon}$: коефицент на обучение
$\boldsymbol{y_j}$: изходна стойност на неврона

```
    public void updateWeights(NeuronConnections connections, double delta) {
        for(INeuron neuron: connections.keySet()) {
            Double newWeight = connections.get(neuron) + Constants.LEARNING_RATE *
neuron.getOutputValue() * delta;
            connections.put(neuron, newWeight);
        }
    }
```

➢ Стъпка 5: Проверка за претрениране (overfitting)

След края на всеки обучителен цикъл, се прави проверка за претрениране. Тази стъпка се извършва с валидационния набор от изображения. Проверява се дали точността на мрежата над валидационния набор от изображения расте. Ако намалява или остава същата значи мрежата се претренира и обучението ще спре до тук.

За да се намери точността на мрежата се изчислява квадратичната грешка. За тази цел се изчислява резултата от мрежата за валидационния набор, но без да се променят тежестите. След всяко изчисление се пресмята квадратичната грешка по следната формула:

$$E = \frac{1}{n}\sum_{i=1}^{n}(t_j - y_j)^2$$

Където:
$\boldsymbol{E}$: Квадратична грешка
$\boldsymbol{y_j}$: изходен сигнал на неврон от изходия слой
$\boldsymbol{t_j}$: очакван резултат за даден неврон от изходния слой
$\mathbf{1/n}$: деление по броя на неврони в изходния слой

Квадратичната грешка се смята за всяко изображение от валидационния набор. Всичките грешки се събират и се делят на броя изображения. С това се намира коефициентът на точност. Ако текущия коефициент на точност е по-голям или равен на този от предишния обучителен цикъл, значи мрежата е претренирана и обучението трябва да спре.

➢ Стъпка 6: Повторение

След калкулирането След изпълнението на тези 5 стъки, най-вероятно имаме по-дора мрежа. За получаване на реално резултати обаче този процес трябва да бъде повторен многократно.

IV. Опериране с мрежата

Оперирането на мрежата включва следните 2 стъпки:

1. Подаване на входове, както е описано в стъпка 1 от миналата глава.
2. Калкулирането на изходите, както е описано в стъпка 2 от миналата глава.

Важно е да се отбележи, че за да може да се използва една мрежа, тя трябва да бъде добре обучена с достатъчно примери. Може да се твърди, че колкото и да бъде тренирана една мрежа, винаги ще има някакъв процент грешка.

Опериране с приложението

Както беше споменато, приложението е имплементирано изцяло на Java, което означава, че за да се оперира с приложението е нужно да има инсталирана Java 7.

I. Файлове

Самия пакет е архив, представляващ набор от следните файлове и директории:

```
├──doc
    ├──IliyanDodekovF50129.docx
    └──IliyanDodekovF50129.pdf
├──lib
    └──mnist-nn.jar
├──networks
    ├──9_68.nn
    ├──10_94.nn
    ├──26.nn
    ├──27.nn
    └──46.nn
├──resources
    ├──test-images.idx3-ubyte
    ├──test-labels.idx1-ubyte
    ├──train-images.idx3-ubyte
    ├──train-labels.idx1-ubyte
    ├──val-images.idx3-ubyte
    └──val-labels.idx1-ubyte

├──src
    └──com
        └──nbu
            └──cscb822
                ├──api
                    ├──INetworkFactory.java
                    ├──INeuralNetwork.java
                    ├──INeuron.java
                    └──INeuronStrategy.java
                ├──exception
                    └──NeuralNetworkException.java
                ├──impl
                    ├──BackwardPropagationNetworkFactory.java
                    ├──BackwardPropagationNeuronStrategy.java
                    ├──Mse.java
                    ├──NeuralNetwork.java
                    ├──Neuron.java
                    ├──NeuronConnections.java
                    ├──NeuronLayer.java
                    └──TrainingData.java
                ├──mnist
                    ├──MnistImage.java
                    ├──MnistImageFileReader.java
```

```
                    ├──MnistList.java
                    └──ReaderApplication.java
                ├──util
                    ├──Constants.java
                    ├──MnistRandom.java
                    ├──MseHistory.java
                    ├──NeuralNetworkInitializer.java
                    ├──NeuronCollection.java
                    ├──NeuronLayerCollection.java
                    ├──TrainingDataCollection.java
                    └──Utils.java
                ├──MNIST.java
                └──MnistHelper.java
├──nn.bat
└──nn.properties
```

В следващата таблица са описани по-важните файлове:

Име	Описание
doc/	Директория съдържаща документацията за приложението
lib/	Директория съдържаща компилирания Java код
networks/	Директория съдържаща готови невронни мрежи, записани във файл
resources/	Директория съдържаща MNIST базата от ръкописни изображения в специални файлове.
src/	Директория съдържаща сорс кода на приложението
nn.bat	Скрипт за стартиране на приложението под Windows
nn.properties	Конфигурационен файл на приложението.

II. Конфигурация

Цялата конфигурация за приложението се прави във файла nn.properties. Тий съдържа следните възможни конфигурационни опции:

Име	Описание
learning.rate	Коефицент на обучение
neurons.input.layer	Брой неврони във входния слой.
neurons.hidden.layer	Брой неврони в скрития слой.
neurons.output.layer	Брой неврони в изходния слой.
file.training.label	Файл съдържащ 60000 имена на цифри за обучителните изображения.
file.training.image	Файл съдържащ пикселите на 60000 цифри за обучителните изображения.
file.testing.label	Файл съдържащ 5000 имена на цифри за тестовите изображения.
file.testing.image	Файл съдържащ пикселите на 5000 цифри за тестовите изображения.
file.validation.label	Файл съдържащ 5000 имена на цифри за валидационни изображения.
file.validation.image	Файл съдържащ пикселите на 5000 цифри за

	валидационни изображения.
image.pixels.x	Брой хоризонтални пиксели на изображенията.
image.pixels.y	Брой вертикални пиксели на изображенията.
overfitting.check.enabled	Дали мрежата да проверява за претрениране с валидационния набор от изображения. Възможните стойности са true или false.
random.seed	Seed за случайни числа за генерирането на тежестите на нова мрежа.
training.mode	Определя дали тренировъчните данни да се подават последователно или разбъркано. Вузможните стойности са shuffle и sequential.

III. Оперирane

Реализираното приложени е изцяло конзолно, като всичките му възможности могат да бъдат достъпени с помощта на меню. За да бъде стартирано е нужно да се стартира файла **nn.bat** под Windows.

При стартиране на приложението се зареждат файловете с изображенията в паметта за по-бързо достъпване и на потребителя е представено меню:

```
E:\workspace\CSCB822NN>nn.bat
Please wait. Loading files...
Magic number for file resources\val-labels.idx1-ubyte is 2049
Entry size for file resources\val-labels.idx1-ubyte is 5000
Magic number for file resources\test-labels.idx1-ubyte is 2049
Entry size for file resources\test-labels.idx1-ubyte is 5000
Magic number for file resources\test-images.idx3-ubyte is 2051
Entry size for file resources\test-images.idx3-ubyte is 3920000
Magic number for file resources\val-images.idx3-ubyte is 2051
Entry size for file resources\val-images.idx3-ubyte is 3920000
Magic number for file resources\train-labels.idx1-ubyte is 2049
Entry size for file resources\train-labels.idx1-ubyte is 60000
Magic number for file resources\train-images.idx3-ubyte is 2051
Entry size for file resources\train-images.idx3-ubyte is 47040000

MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise:
```

След избор на дадена опция и изпълнението и, потребителят винаги бива доведен до това меню до изход от приложението.

Всяка от опциите има следното действие:

- **Create a new neural network** – Създава нова невронна мрежа със случайни стойности на тежестите и праговите коефиценти. Приложението може да работи само с една невронна мрежа и при повторно извикване на тази опция, старата мрежа ще бъде изтрита.

- **Load a neural network from a file** – Зарежда вече създадена и потенциално обучена невронна мрежа от файл. Формата на файла е специфичен за приложението и могат да бъдат зареждани само файлове, които са създадени от същото това приложение. Приложението може да работи само с една невронна мрежа и при повторно извикване на тази опция, старата мрежа ще бъде изтрита.

- **Save a neural network to a file** – Запазва обучена мрежа във файл. Формата на файла е специфичен за приложението и могат да бъдат зареждани само файлове, които са създадени от същото това приложение.
- **Train the neural network** – Започва обучение на невронна мрежа. Задава се колко обучителни цикъла да се направят. Възможно е и предварително прекратяване на тренирането при достигане на даден процент на грешка – тази опция може да се конфигурира в nn.properties файла.

- **Test the neural network** – Демонстрира се работата на невронната мрежа – подава се цифра и мрежата познава коя е тя. Цифрата се подава, като пореден номер от тестовия сет от MNIST базата. За добри резултати е нужно мрежата да бъде обучена достатъчно.

- **Print average error rate** – Калкулира се процента на грешка, спрямо всички изображения от тестовия сет от MNIST базата. Реално всичките 10000 изображения се прекарват през мрежата и излиза процента на сбъркани категоризации.

- **Print average mean square history for training data** – визуализира квадратичната грешка изчислена след всеки цикъл на обучение на мрежата.

- **Print average mean square history for validation data** - визуализира квадратичната грешка на валидационния набор от изображения изчислена след всеки цикъл на обучение на мрежата.

- **Quit** – Изход от приложението.

➢ Създаване на мрежа

За да се създаде мрежа:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 1

Neural Network successfully initialized!
```

➢ Зареждане на вече създадена мрежа от файл

За да се зареди вече създадена мрежа:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 2

Enter file to load the network from: E:\workspace\CSCB822NN\networks\nn5-
1_94.nn

Neural Network successfully loaded!
```

➢ Записване на мрежа във файл

За да се запиша невронна мрежа във файл:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
```

```
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 3

Enter file to save the network to: E:\workspace\CSCB822NN\networks\test.nn
```

- **Обучение на мрежа**

За да бъде обучена мрежа:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 4

Finished adding training data.
Finished adding validation data.
Enter the number of epochs: 5

Training is in process. Please wait.
Training mode is set to: shuffle
Shuffling training data...
Traing data succesfully shuffled.
Training data MSE for round [1] is [0.03328079540739458].
Validation data MSE for round [1] is [0.0336537310321584].
Epoch [1] is done in 18.73735 minutes.
Shuffling training data...
Traing data succesfully shuffled.
Training data MSE for round [2] is [0.02339336776163224].
Validation data MSE for round [2] is [0.01630062776223768].
Epoch [2] is done in 18.625983333333334 minutes.
Shuffling training data...
Traing data succesfully shuffled.
Training data MSE for round [3] is [0.014923125409202685].
Validation data MSE for round [3] is [0.015142307748459996].
Epoch [3] is done in 18.652933333333333 minutes.
Shuffling training data...
Traing data succesfully shuffled.
Training data MSE for round [4] is [0.014149090772041207].
Validation data MSE for round [4] is [0.01493014154354006].
Epoch [4] is done in 18.62825 minutes.
Shuffling training data...
Traing data succesfully shuffled.
```

```
Training data MSE for round [5] is [0.013676553937716985].
Validation data MSE for round [5] is [0.014629638115677301].
Epoch [5] is done in 18.43585 minutes.
Training done in 93.10598333333333 minutes.
```

В зависимост от броя обучителни итерации, времето за обучение може да бъде доста голямо. Ако е конфигурирано, тренировката може да спре преждевременно при претрениране.

Претрениране на мрежата изглежда по следния начин:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
optio
n:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 4

Finished adding training data.
Finished adding validation data.
Enter the number of epochs: 5

Training is in process. Please wait.
Training mode is set to: shuffle
Shuffling training data...
Traing data succesfully shuffled.
Training data MSE for round [1] is [0.0134509350214862].
Validation data MSE for round [1] is [0.014896809038974735].
Average MSE of validation data is bigger than in previous iteration. The
network
 is overfitting - training will stop now.
Training done in 18.647316666666665 minutes.
```

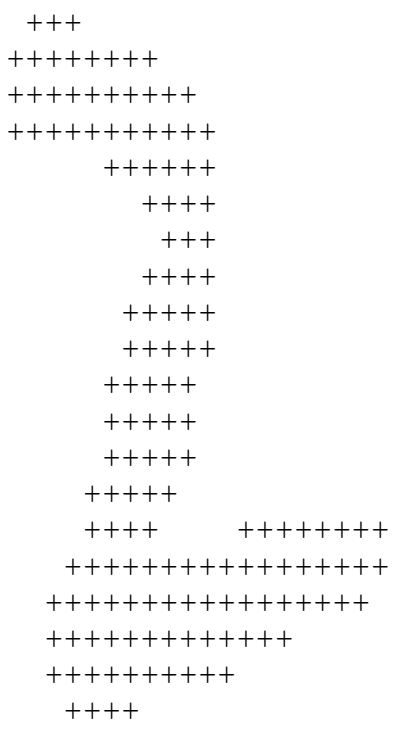
➢ Тестване на мрежа

Тестването на мрежата става посредством избор на поредна цифра от тестовия сет на MNIST базата. Мрежата се опитва да познае е коя е цифрата и съответно се принтира дали наистина е това. На конзолата се изрисува и как изглежда самото изображение, като всички пиксели, които не са 0 се изобразяват със знака „*“. Принтира се също и вектор с изхода на мрежата (стойността на 10-те неврона в изходния слой).

```
MNIST Neural Network for handwritten digit recognition. Choose
the desired optio
```

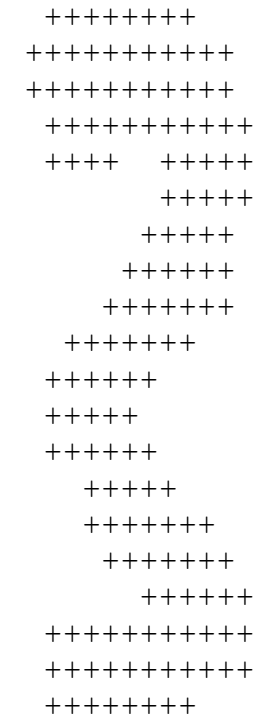
```
n:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 5

Enter the desired letter number from the testing set or [b] to
return: 371
```

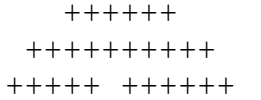


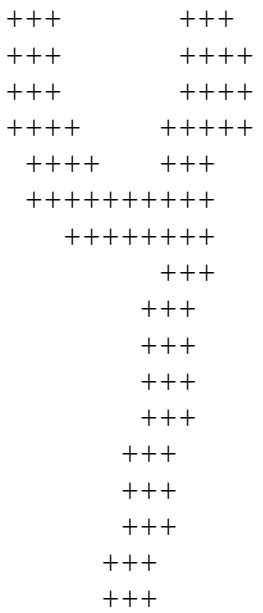
```
Testing with number: 2
This looks like the number: 2
Neural Network output - [0:0.0000 ,1:0.0000 ,2:1.0000 ,3:0.0000
,4:0.0000 ,5:0.0
000 ,6:0.0000 ,7:0.0000 ,8:0.0000 ,9:0.0000]
```

```
Enter the desired letter number from the testing set or [b] to
return: 982
```



```
Testing with number: 3
This looks like the number: 3
Neural Network output - [0:0.0000 ,1:0.0000 ,2:0.0000 ,3:1.0000
,4:0.0000 ,5:0.0
000 ,6:0.0000 ,7:0.0000 ,8:0.0000 ,9:0.0000]
Enter the desired letter number from the testing set or [b] to
return: 2345
```





```
Testing with number: 9
This looks like the number: 9
Neural Network output - [0:0.0000 ,1:0.0000 ,2:0.0000 ,3:0.0000
,4:0.0000 ,5:0.0
000 ,6:0.0000 ,7:0.0000 ,8:0.0000 ,9:1.0000]
Enter the desired letter number from the testing set or [b] to
return:
```

За да се върне в главното меню потребителя просто трябва да натисне “b”.

- **Пресмятане на процента на грешка**

При пресмятане на грешката се тества цялия тестов сет на MNSIT базата. Процента на грешка представлява процента на сбъркани цифри спрямо целия брой:

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
option:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 6
```

```
Starting to test the data...
Finished testing the data.
Error count is [484] from [5000]. That's [9.68%].
Error count for number [0] is 257
Error count for number [1] is 9
Error count for number [2] is 15
Error count for number [3] is 26
Error count for number [4] is 33
Error count for number [5] is 19
Error count for number [6] is 20
Error count for number [7] is 45
Error count for number [8] is 34
Error count for number [9] is 26
```

- **Квадратична грешка на тренировъчните данни**

Визуализира квадратичната грешка изчислена след всеки цикъл на обучение на мрежата.

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
optio
n:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 7

Training epoch [1] has finished with MSE value of [0.07434192700569693].
Training epoch [2] has finished with MSE value of [0.05444115540430707].
Training epoch [3] has finished with MSE value of [0.044637232831067346].
Training epoch [4] has finished with MSE value of [0.03406269781808672].
Training epoch [5] has finished with MSE value of [0.03328079540739458].
Training epoch [6] has finished with MSE value of [0.02339336776163224].
Training epoch [7] has finished with MSE value of [0.014923125409202685].
Training epoch [8] has finished with MSE value of [0.014149090772041207].
Training epoch [9] has finished with MSE value of [0.013676553937716985].
Training epoch [10] has finished with MSE value of [0.0134509350214862].
```

- **Квадратична грешка на валидационните данни**

Визуализира квадратичната грешка на валидационния набор от изображения изчислена след всеки цикъл на обучение на мрежата.

```
MNIST Neural Network for handwritten digit recognition. Choose the desired
optio
```

```
n:
1.  Create a new neural network.
2.  Load a neural network from a file.
3.  Save neural network to a file.
4.  Train the neural network.
5.  Test the neural network.
6.  Print average error rate.
7.  Print average mean square error history for training data.
8.  Print average mean square error history for validation data.
99. Quit.
Your choise: 8

Training epoch [1] has finished with MSE value of [0.055963184571055105].
Training epoch [2] has finished with MSE value of [0.05323948756883251].
Training epoch [3] has finished with MSE value of [0.034858559903565].
Training epoch [4] has finished with MSE value of [0.03420022121822513].
Training epoch [5] has finished with MSE value of [0.0336537310321584].
Training epoch [6] has finished with MSE value of [0.01630062776223768].
Training epoch [7] has finished with MSE value of [0.015142307748459996].
Training epoch [8] has finished with MSE value of [0.01493014154354006].
Training epoch [9] has finished with MSE value of [0.014629638115677301].
Training epoch [10] has finished with MSE value of [0.014896809038974735].
```

Тук може да се види, че при последната обучителна итерация квадратичната грешка е по-голяма от при пред-последната обучителна итерация. Съответно обучението е спряло преждевременно.

Използвани източници

За изготвянето на курсовата работа са ползвани следните източници:

Курсовете по CSCB822 Невронни Мрежи
http://wikipedia.org
http://yann.lecun.com/exdb/mnist/
http://www.codeproject.com/
http://neuralnetworksanddeeplearning.com/
http://pages.cs.wisc.edu/
http://statweb.stanford.edu/~tibs/ElemStatLearn/data.html