

Bachelorarbeit

Jan Philipp Fortowski

xx.xx.2024

Contents

1	Fully Connected Layer / Feed Forward Layer	3
1.1	ForwardPass in der Fully Connected Layer	4
1.2	Backpropagation im Fully Connected Layer	7
2	Pooling Layer	7
2.1	Forward Propagation im Pooling Layer	9
2.2	Backpropagation im Pooling Layer	10
3	Unterstützungs Templates	15
4	Einleitung	17
4.1	Motivation	17
5	Quellen	18
6	Liste der Abbildungen	19

1 Fully Connected Layer / Feed Forward Layer

In diesem Kapitel werden die Fully Connected Layer betrachtet, welche genau so aufgebaut sind, wie bei den Feed Forward Networks. genau genommen wird im Prinzip einfach ein Feed Forward network am Ende des Convolutional networks eingesetzt, denn die Fully Connected layer werden nur am Ende benutzt. Wenn in einem Netzwerk nur die Fully Connected Layer verwendet werden, dann handelt es sich um ein Feed Forward Network. Das Grundprinzip ist vergleichbar kompliziert. Jede Schicht enthält sogenannte Knoten. Diese Knoten sind mit den Neuronen in einem Gehirn vergleichbar.

TODO Bild von Neuronen und Activations/Knoten

Alle Knoten einer Schicht, sind mit allen Knoten der nächsten Schicht verbunden. Diese Verbindungen werden gewichtet, also müssen Gewichte gespeichert werden, die festlegen, wie wichtig der Input eines jeweiligen Knotens für den aktuellen Knoten ist. Diese Knoten können angepasst werden, das heißt, dass kleine inkrementelle Anpassungen mit jeder Lern-Iteration vorgenommen werden, um das Netzwerk Stück für Stück einem Fehler-Minimum anzupassen.

TODO Bild von einem Neuronalen Netzwerk mit allen Verbindungen

Besonders für lineare Probleme, die nicht durch den Nullpunkt eines Koordinatensystems gehen ist es notwendig, auch Biases mit einzurechnen. Für die Bilderkennung ist das normalerweise nicht wichtig, und da sich die Convolutions auf die Bilderkennung spezialisieren, wird es für dieses Projekt nicht notwendig sein, Biases einzuplanen.

In diesem Kapitel werden nun zunächst der generelle Aufbau der Klasse "FullyConnectedLayer" dargelegt, und in den beiden Unterkapiteln wird dann speziell auf den ForwardPass und auf die Backpropagation eingegangen, zusammen mit den mathematischen Grundlagen und dem dazugehörigen Code.

Zunächst muss die Klasse erstellt werden. Die Objektvariablen sind diese:

```
1 public class FullyConnectedLayer extends Layer {  
2     double[][] weights;  
3     int inLength;  
4     int outLength;  
5     double learnRate;  
6 }
```

```
7 private double[] lastZ;  
8 private double[] lastInput;
```

in den "weights" werden die Gewichte gespeichert zwischen den Inputs und den Knoten der Schicht. Diese können angepasst werden, um den Fehler des Netzwerkes zu minimieren, damit es korrekte Klassifizierungen vornehmen kann. Mit "inLength" ist die Menge an Inputs gemeint, das heißt zum Beispiel wie viele Pixel die Bilder haben, die in die Schicht eingespeißt werden. Mit "outLength" entspricht dann auch der Menge an Outputs welche von der Schicht erzeugt werden, das hängt davon ab, wie viele Knoten die nächste Schicht hat, oder wenn es die letzte Schicht ist, wie viele mögliche Antworten das Netzwerk hat. Die "learnRate" schränkt die grÖße der Inkremente zwischen den einzelnen gelernten Bildern ein. Dies ist Notwendig, da die berechneten inkremente, also die Steigung im Fehlergraphen, normalerweise zu groß ist, und die Fehler Minima dadurch häufig übersprungen werden. Das netzwerk soll sich aber dem Fehlerminimum annähern, also müssen die Schritte kleiner sein. Zu klein sollten sie aber auch nicht sein, weil das Netzwerk dadurch deutlich langsamer wird, und eine zu langsame Annäherung nicht Vorteilhaft ist.

TODO Bild von Lernrate zu groß und zu klein

Mit dem Array "lastZ" und "lastInput" sollen jeweils für die Backpropagation Notwendige Zwischenschritte gespeichert werden. "lastZ" entspricht den Inputs multipliziert mit den Gewichten. Das bedeutet hierbei handelt es sich um die Ergebnisse der Schicht bevor sie durch die Activation Funktion umgerechnet werden. Mit "lastInput" sind einfach die Inputs gemeint, die zuletzt in diese Schicht geleitet wurden, und noch keinen Rechnungen unterlegen sind. Genauer wird im Unterkapitel zur Backpropagation betrachtet, aber diese Werte sind in den verschiedenen Schritten durch die Kettenregel Notwendig.

1.1 ForwardPass in der Fully Connected Layer

Für den Forward Pass, also die Grundfunktion in der Query, muss ein Array erstellt werden, welches die outputs der Schicht enthält. Beim Fully Connected Layer entspricht die menge an Outputs der menge der Knoten aus der nächsten Schicht. Ein Wert in der Output menge wird berechnet, indem

jeder Input in die Schicht mit dem Gewicht multipliziert wird, welches zwischen jedem Knoten in dieser Schicht und dem Knoten der nächsten Schicht gespeichert wurde. Alle diese Werte werden dann multipliziert, und mit einer Activation Funktion umgerechnet. Dies ist dann der Output, der an die nächste Schicht übergeben wird. Hier im Code wird dieser Output durch das Array "a" dargestellt, um auf das Ergebnis der Activation hinzuweisen.

Abgesehen davon werden allerdings auch noch die Werte lastInput und das Array "Z" zwischengespeichert. Diese Werte sind für die Backpropagation notwendig, daher werden die Details auch erst im nächsten Kapitel behandelt. Aber im Prinzip steht das Array "Z" für die Zwischenergebnisse, die noch mit der Activation Funktion verrechnet werden müssen. Dieses Zwischenergebnis ist für die Backpropagation wichtig, und muss daher zwischengespeichert werden.

"lastInput" ist an sich selbsterklärend, hier wird einfach der letzte Input zwischengespeichert, also einfach die Werte, die bei der Query aus der letzten Schicht eingereicht wurden, und noch nicht weiter verrechnet wurden.

```
1 public double[] FullyConnectedForwardPass(double[] input)
2 {
3     lastInput = input;
4     double[] Z = new double[outLength];
5     double[] a = new double[outLength];
6     //Diese schleife summiert alle Inputs auf alle
7     outputs, und multipliziert die Inputs mit ihren
8     jeweiligen gewichten aus der weights-Matrix
9     for(int i=0; i<inLength; i++){
10        for(int j=0; j<outLength;j++){
11            Z[j] += input[i]*weights[i][j];
12        }
13    }
14 }
```

Nach dieser Schleife sind die Inputs mit den Gewichten verrechnet, aber die Activation Funktion wurde noch nicht angewendet. Also sollten diese Werte unter "lastZ" zwischengespeichert werden.

```
1     lastZ = Z;
2
3     //Diese Schleife wendet auf alle Outputs die Activation
4     Funktion an, in diesem Falle die Sigmoid Funktion
5     for(int i=0; i<inLength; i++){
6         for(int j=0; j<outLength;j++){
```

```

6         a[j] = Sigmoid(Z[j]);
7     }
8 }
9 return a;
10 }

```

Der Array der zurückgegeben wird, ist der Komplette berechnete Output, und kann so an die nächste Schicht weitergeleitet werden. Im Convolutional Network kommen mehrere Schichten Fully Connected layer immer ans Ende. Die Convolution Layer und die MaxPool Layer sind auf die Erkennung von Features in Bildern und auf die Komprimierung und Verdichtung relevanter Daten in den Bildern spezialisiert. Daher macht es keinen Sinn, diese Schichten nach einem Fully Connected Layer einzusetzen. Das Bild kann aus den Ausgaben einer Fully Connected Layer nicht wieder rekonstruiert werden, und ist daher für die anderen Schichten nicht mehr nutzbar. Es kann also davon ausgegangen werden, dass nach einem Fully Connected Layer nur noch weitere Fully Connected Layer auftreten werden. Außerdem soll hier angemerkt werden, dass die Convolution und Pool Layer zwar die Bildererkennung positiv beeinflussen, simple Netzwerke bereits rein und ausschließlich mit Fully Connected Layeren aufgebaut werden können. Diese sind zwar sehr anfällig auf leichte Veränderungen der Inputs, und die Genauigkeit lässt daher ein wenig zu wünschen übrig, aber durchaus verlässliche Ergebnisse können damit schon erzielt werden.

Da die Outputs der Fully Connected Layer keine direkten Rückschlüsse auf die Bilder zulassen, die zu Anfang in das Netzwerk gegeben werden, wird bei den in der Abstrakten Klasse definierten Methoden zur Rückgabe der verschiedenen Parameter auch anders verfahren, als in den anderen Layer Klassen. Die anderen Layer Klassen müssen Auskunft darüber geben können, wie viele Output Bilder sie erzeugen, und welche Dimensionen diese besitzen. Zum Beispiel gibt die Max pool Layer zwar komprimierte Bilder zurück, das heißt dass die Dimensionen kleiner geworden sind. Allerdings werden alle Bilder, die in die Schicht eintreten, mit jedem Filter Fenster verrechnet, welches die Schicht besitzt. Das heißt die Menge der Output Bilder ist die Menge der Input Bilder mal die Menge der Filter. Diese Informationen sind für die nachfolgenden Schichten von großem Wert. Bei den Fully Connected Layeren können nur Fully Connected Layer folgen, also ist auch nur die Menge der Inputs wichtig, da Fully Connected Layer die Inputs als ein einziges 1-

Dimensionales Array betrachten. Also muss nur die "getOutputElements" Methode sinnvoll betrachtet werden, diese gibt einfach die "outLength" Variable zurück. Alle anderen get-Methoden können 0 zurück geben.

```
1  @Override
2  public int getOutputElements() {
3      return outLength;
4  }
```

1.2 Backpropagation im Fully Connected Layer

Ja Hahaha hier kommt Text hin lalalal

Ich bin ein harlunke Har Har Har

Ein echter Pirat wenn du so willst du Troll

Berta der troll war ein echt wiederliches Ding, ein Troll von gigantischen ausmaßen in all den falschen stellen, Feter bauch, Fette nase Riesige Ohren, Riesige Füße, und Riesige Zähne, zumindest die Huaer, die noch übrig waren.

2 Pooling Layer

Pooling Layer werden dazu verwendet, um das sogenannte Downsampling zu betreiben. Downsampling bedeutet einfach, dass eine geringere Auswahl an Daten weitergegeben werden. In einem Netzwerk benutzt man eines dieser Verfahren, um die weiteren Schichten nur mit den nötigsten Daten zu belasten. Gerade in Anbetracht der Größe der Daten, die bei den Convolutional Layern entstehen, kann das viel Zeit sparen. Aber Hauptsächlich sorgt dies auch dafür, dass das Netzwerk weniger anfällig für Veränderungen der Daten wird. leichte Veränderungen der Daten können für gravierende Änderungen, das heißt wenn ein Bild nicht korrekt Zentriert ist, oder wenn es um ein paar Grad gedreht wird, kommt ein Netzwerk schnell durcheinander. Die Convolution Layer erstellen Feature Maps, in denen die Features allerdings auch die gleichen Positionen haben, wie in den Inputs. Dadurch können bei kleinen Veränderungen schon Fehler in den nächsten Schichten entstehen. Dem kann man durch das Downsampling entgegenwirken.

Auf eine Gewisse weise können die Daten bereits in den Convonlutional Layern "gedownsampelt" werden, wenn die Schrittgröße (Stride) beim anwenden der Filter Größer ist als 1.

Wie genau funktioniert ein Pooling Layer dann? so wie bei den Convolutional Layern werden Filter eingesetzt. Diese sind aber kleiner, und funktionieren anders. Meist ist ein Filter hier nur 2x2 groß, und bei der Anwendung wird eine Schrittgröße von 2 verwendet. Die Filter überlappen die Felder nicht, die bereits behandelt wurden. Auf die Pixel, auf die der Filter angewandt wird, werden nun eine Pooling Operation angewandt.

Zwei mögliche Pooling Operationen sind das Durchschnitts Pooling und das Maximum Pooling.

- Das Durchschnitts Pooling gibt den Durchschnitt aller Werte als Output zurück, während
- das Maximum Pooling nur den Größten Wert der betrachteten Werte zurückgibt.

Es gibt noch weitere, wie zum Beispiel das Globale Pooling, bei dem die gesamte Feature Map auf einen einzigen Wert reduziert wird. Dadurch soll auf eine möglichst aggressive Art festgestellt werden, ob ein bestimmtes Feature überhaupt in der Feature Map vorkommt. Je nach Anwendungsfall kann auch diese Methode Erfolg bringen.

Zunächst wird die Maximum Pooling Layer betrachtet. genau wie bei den Convolutional Layern, wird hier eine StepSize verwendet. Zwar ändert sich diese in den meisten Fällen nicht, aber es ist gut, die Flexibilität zu haben. Die windowSize legt fest, welche Dimensionen der Filter haben soll. Außerdem ist es hilfreich, wenn die input Parameter auch gespeichert werden, damit gemeint sind input length, input rows und input columns, also die Menge an Bildern, zusammen mit den Zeilen und Spalten die diese Bilder haben.

```
1 public class MaxPoolLayer extends Layer {  
2  
3     private int stepSize;  
4     private int windowSize;  
5  
6     private int inLength;  
7     private int inRows;  
8     private int inCols;
```


2.1 Forward Propagation im Pooling Layer

Wie oben beschrieben, wird hier ein kleiner Filter benutzt, der über die Input Bilder gleitet, und entweder den maximalen Wert, oder den Durchschnittswert zurückgibt.

TODO Bild einfügen

Zunächst wird eine Methode benötigt, die den Filter darstellt, und eine Feature Map durchläuft.

```
1 public double [][] pool(double [][] input){
2     //Die Output Dimensionen müssen berechnet werden
3     double [][] output = new double[getOutputRows()][
4         getOutputCols()];
5     //zwei Schleifen um über den Input zu iterieren
6     for(int r=0; r<getOutputRows(); r+=stepSize){
7         for(int c=0; c<getOutputCols(); c+=stepSize){
8             double max = 0.0;
9             //Zwei Schleifen um das Maximum im Fenster
10            //des Filters zu finden
11            for(int x=0; x<windowSize; x++){
12                for(int y=0; y<windowSize; y++){
13                    if(max<input[r+x][c+y]){
14                        max=input[r+x][c+y];
15                    }
16                }
17            }
18            output[r][c] = max;
19        }
20    }
21    return output;
22 }
```

Alternativ kann natürlich auch der Durchschnitt errechnet und ausgegeben werden. Dazu sind keine Großen Anpassungen nötig:

```
1 public double [][] pool(double [][] input){
2     //Die Output Dimensionen müssen berechnet werden
3     double [][] output = new double[getOutputRows()][
4         getOutputCols()];
5     //zwei Schleifen um über den Input zu iterieren
6     for(int r=0; r<getOutputRows(); r+=stepSize){
7         for(int c=0; c<getOutputCols(); c+=stepSize){
8             double average = 0.0;
9         }
10    }
11    return output;
12 }
```

```

8      //Zwei Schleifen um die Werte im Fenster des
      Filters zu addieren
9      for(int x=0; x<windowSize; x++){
10         for(int y=0; y<windowSize; y++){
11             average+=input[r+x][c+y];
12         }
13     }
14     //Der Durchschnitt muss natürlich noch durch
      die menge der im Filter Fenster
      enthaltenen Pixel geteilt werden
15     output[r][c] = average/(windowSize*windowSize
      );
16 }
17 }
18 return output;
19 }

```

Im Code wurden die Methoden `getOutputRows` und `getOutputCols` schon verwendet, also müssen diese auch noch Implementiert werden. Die Formel dazu sieht folgendermaßen [1] aus:

$$H_{out} = \text{floor}(1 + (H - \text{pool_height})/\text{stride})$$

$$W_{out} = \text{floor}(1 + (W - \text{pool_width})/\text{stride})$$

H_{out} und W_{out} entsprechen den Dimensionen des output Fensters.

Und im Code werden diese so umgesetzt:

```

1  public int getOutputRows() {
2      return (inRows - windowSize) / stepSize + 1;
3  }
4
5  public int getOutputCols() {
6      return (inCols - windowSize) / stepSize + 1;
7  }

```

2.2 Backpropagation im Pooling Layer

Für die Backpropagation in einem Pooling Layer müssen keine gewichte oder Biases angepasst werden. Denn diese gibt es hier nicht. Stattdessen muss der Fehler aber trotzdem auf einen der Inputs zurückgeführt werden, das heißt

zum Beispiel, dass beim Max Pooling aus jedem von dem Filter Fenster betrachteten Gebiet der Maximale Wert für den Fehler verantwortlich war, und die anderen Werte keine Verantwortung tragen.

TODO Bild einfügen

Am besten gelingt dies, indem die Positionen der Maximal Werte gespeichert werden. Dazu sollten erst neuen Instanzvariablen angelegt werden:

```
1 List<int [][]> lastMaxRow;  
2 List<int [][]> lastMaxCol;
```

Zwei Listen, welche 2-Dimensionale Arrays enthalten, jeweils eine liste für die Koordinate der X und eine für die Y Achse. Danach müssen diese Initialisiert werden, und zwar in der "maxPoolForwardPass" Methode. Das stellt sicher, dass sie initialisiert werden, bevor das pooling beginnt:

```
1 public List<double [][]> maxPoolForwardPass(List<double  
    [][]> input) {  
2     //Initialisierung der lastMaxRow und Col, in denen  
        die Position der Maximalen Werte gespeichert  
        werden  
3     lastMaxRow = new ArrayList<>();  
4     lastMaxCol = new ArrayList<>();
```

Um die Werte dann zu speichern, müssen diese während der pooling Operation gespeichert werden. Dazu können zwei 2-Dimensionale Integer Arrays verwendet werden, welche mit den gleichen Maßen initialisiert werden, wie die Outputs. Das heißt, dass für jeden Output die X und Y Koordinate im dazugehörigen lastMax-Array ein entsprechender Platz vorhanden ist.

```
1 public double [][] pool(double [][] input) {  
2     double [][] output = new double[getOutputRows()][  
        getOutputCols()];  
3  
4     int [][] maxRows = new int[getOutputRows()][  
        getOutputCols()];  
5     int [][] maxCols = new int[getOutputRows()][  
        getOutputCols()];
```

für den Fall, das kein maximum über 0 gefunden werden konnte, sollten Koordinaten trotzdem markiert werden, darum werden die alle Felder zuerst auf -1 gesetzt:

```
1     for (int r = 0; r < getOutputRows(); r += stepSize) {
```

```

2         for (int c = 0; c < getOutputCols(); c +=
           stepSize) {
3             double max = 0.0;
4
5             maxRows[r][c] = -1;
6             maxCols[r][c] = -1;

```

In der inneren Schleife müssen die Koordinaten natürlich auch gespeichert werden, wenn ein neues Maximum im Fenster gefunden wurde:

```

1         for (int x = 0; x < windowSize; x++) {
2             for (int y = 0; y < windowSize; y++) {
3                 if (max < input[r + x][c + y]) {
4                     max = input[r + x][c + y];
5             //Koordinaten des Maximalen Wertes werden in den dafür
              vorgesehenen Arrays gespeichert
6                     maxRows[r][c] = r + x;
7                     maxCols[r][c] = c + y;
8                 }
9             }
10        }
11        output[r][c] = max;
12    }
13 }

```

zuletzt müssen die Arrays, welche die Koordinaten enthalten noch den Listen hinzugefügt werden, die am Anfang Initialisiert wurden:

```

1        lastMaxRow.add(maxRows);
2        lastMaxCol.add(maxCols);
3        return output;
4    }

```

Die eigentliche Umsetzung der Backpropagation findet in der "backPropagation" Methode statt. Der Grundgedanke ist der, dass nur der Maximalwert zum Fehler beigetragen hat, und daher auch nur dieser Wert angepasst werden muss. Also Erstellt man eine Fehlermatrix, die überall nur 0 enthält, außer an den Stellen, von denen die Maximalwerte genommen wurden. Da werden die Fehlerwerte eingetragen, und dann, da es hier keinerlei Gewichte oder Biases gibt, wird diese Fehlermatrix dann an die nächste Schicht weitergegeben. Zuerst muss eine Liste mit den Fehlermatrizen erstellt und initialisiert werden und es wird ein Zähler benötigt, um über alle vorher angelegten Koordinaten-Matrizen zu iterieren, welche die Positionen der Maximalwerte

enthalten.

```
1 public void backPropagation(List<double[][]> dLd0) {  
2     List<double[][]> dXdL = new ArrayList<>();  
3     int l=0;
```

Dann wird über alle eingehenden Fehlermatrizen iteriert, um die jeweilige Fehlermatrize zu erstellen, die an die nächste Schicht weitergegeben wird. letztere muss zunächst initialisiert werden.

```
1     for(double[][] array : dLd0){  
2         double[][] error = new double[inRows][inCols];
```

Dann muss die eingehende Fehlermatrix durchiteriert werden. für jeden eingehenden wert wird die Position des verantwortlichen Maximalwerts ermittelt. In der entstehenden Fehlermatrix wird der eingehende fehler dann an der jeweiligen Position hinzugefügt. Wichtig ist, dass der Wert zu den bestehenden Werten hinzugefügt wird. meistens überlappen sich die betrachteten fenster in der Pooling Phase nicht, aber falls doch, kann ein ein Wert in mehreren Fenstern der Maximalwert sein, daher ist dieser wert dann auch sozusagen für mehrere Fehler verantwortlich. Das wird dadurch ausgedrückt, dass der Fehlerwert dann aufaddiert wird.

```
1         for(int r=0; r<getOutputRows(); r++){  
2             for(int c=0; c<getOutputCols(); c++){  
3                 int max_i = lastMaxRow.get(l)[r][c];  
4                 int max_j = lastMaxCol.get(l)[r][c];  
5  
6                 if(max_i != -1){  
7                     error[max_i][max_j] += array[r][c];  
8                 }  
9             }  
10        }
```

Zuletzt müssen die entstandenen Fehlermatrizen nur noch der liste hinzugefügt werden, welche dann an die nächste Schicht weitergegeben wird.

```
1        dXdL.add(error);  
2        l++;  
3    }  
4    if(previousLayer != null){  
5        previousLayer.backPropagation(dXdL);  
6    }  
7 }
```

Für den Fall, dass die eingehenden Fehler keine liste, sondern eine einfaches Array ist, also zum Beispiel weil die nächste Schicht eine Fully Connected Layer ist, erstellt man noch eine "backPropagation" Methode:

```
1     public void backPropagation(double[] dLd0) {  
2         List<double[][]> matrixList = vectorToMatrix(dLd0,  
            getOutputLength(), getOutputRows(), getOutputCols  
            ());  
3         backPropagation(matrixList);  
4     }
```

3 Unterstützungs Templates

Die Gliederung dieser Arbeit, entspricht einer organischen Herangehensweise oder einer Anleitung, wie ein Netzwerk aufgebaut wird. Jedes Kapitel entspricht einem Teil des Netzwerkes, also den Funktionen, der Initialisierung, der Query oder Abfrage und dem Lern- oder Backpropagation-Algorithmus. Jedes der Kapitel startet mit einer Übersicht, über die Funktionalität, erläutert die Theorie dahinter, teilweise auch mathematisch, und schließt, mit dem daraus resultierenden Code ab.

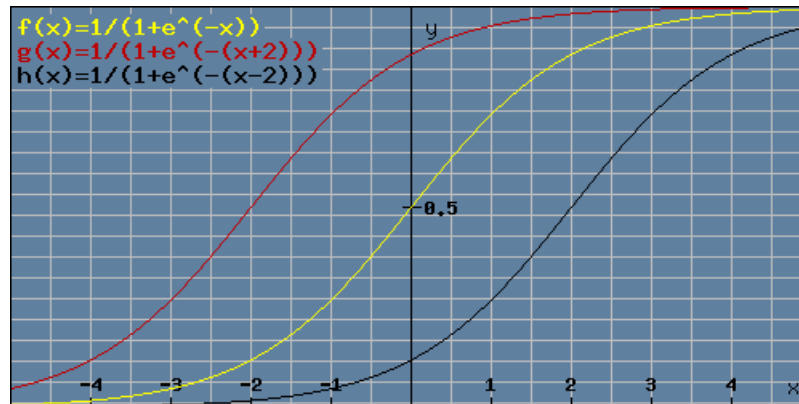


Figure 1: Sigmoidfunktion mit Bias

- Dendriten
- der Zellkörper
- das Axon

```
1 public class NeuralNetwork {
2     Layer[] layers;
3     double learnRate;    // Initialisierung
4     public NeuralNetwork(double learnRate, int... layerSizes)
5     {
6         layers = new Layer[layerSizes.length - 1];
7         for (int i = 0; i < layers.length; i++) {
8             layers[i] = new Layer(layerSizes[i], layerSizes[i
9                 + 1]);
10        }
11        this.learnRate = learnRate;
12    }
13 }
```

Aber verfolgen wir diesen Gedanken doch einmal an einem Beispiel: Sei $f(w)$ unsere Funktion:

$$f(w) = w^2 + 3$$

Wir nennen die kleine Verschiebung von w jetzt h . Dann gilt zumindest schon einmal:

$$\Delta w = (w + h) - w$$

Und gekürzt:

$$\Delta w = h$$

Dann wäre die Änderungsrate also:

$$\text{Änderungsrate} = \frac{\Delta e}{\Delta w}$$

$$\frac{\Delta e}{\Delta w} = \frac{f(w + h) - f(w)}{h}$$

Wenn wir uns dann die Mühe machen, $f(w)$ auszuschreiben, ergibt sich daraus:

$$\frac{\Delta e}{\Delta w} = \frac{(w + h)^2 + 3 - (w^2 + 3)}{h}$$

Dann fangen wir an Klammern auszurechnen:

$$\frac{\Delta e}{\Delta w} = \frac{w^2 + w * h + w * h + h^2 + 3 - w^2 - 3}{h}$$

$$\frac{\Delta e}{\Delta w} = \frac{w^2 + w * h + w * h + h^2 - w^2}{h}$$

$$\frac{\Delta e}{\Delta w} = \frac{w * h + w * h + h^2}{h}$$

h kürzen:

$$\frac{\Delta e}{\Delta w} = w + w + h$$

$$\frac{\Delta e}{\Delta w} = 2w + h$$

Und jetzt zum Interessanten Teil. Da wir h nicht gleich 0 setzen können, können wir allerdings h gegen 0 laufen lassen, dann verwenden wir die Leibniz-Notation. Das bedeutet, dass wir anstatt Δw und Δe , wobei Δ eine sehr kleine Vergrößerung darstellt, jetzt dw und de verwenden, wobei d für eine unendlich kleine Vergrößerung steht, eine sogenannte Infinitesimalzahl. Das

sieht dann ungefähr so aus:

$$\frac{de}{dw} = \lim_{h \rightarrow 0} 2w + h$$

$$\frac{de}{dw} = 2w$$

4 Einleitung

4.1 Motivation

Neuronale Netzwerke werden vor allem für Klassifikationsverfahren verwendet. In der Praxis gibt es viele Anwendungsbereiche, in denen es vorteilhaft ist, große Mengen von Daten automatisch zu klassifizieren. Einige Beispiele, wären die Bild- und Schrifterkennung, die man dazu verwendet, Kennschilder von Autos, maschinell auszulesen. Solche Technologien werden immer häufiger auf Parkplätzen und Autobahnen eingesetzt. Aber, auch fast jedes Handy kann mittlerweile Schrift erkennen, die mit der Kamera aufgenommen wird. Auch in der Medizin, beim Auswerten von Röntgenbildern, in der Biologie, zum Erkennen von Pflanzen auf Fotos und noch vielem mehr, werden Neuronale Netzwerke inzwischen eingesetzt.

Es gibt kaum einen Bereich, in dem sich nicht eine mögliche Anwendung für Neuronale Netzwerke finden lässt. Ganz besonders aufwändige Netzwerke, werden mittlerweile auch für konstruktive Aufgaben verwendet, wie zum Beispiel Sprachmodelle, unter anderem ChatGPT und Bilder generierende KIs, wie Midjourney. Typische Probleme, bei welchen neuronale Netzwerke eingesetzt werden, sind komplexe Aufgaben, mit gigantischen Datenmengen. Eine Aufgabe ist zu komplex, wenn eine Lösung nicht manuell programmiert werden kann, unter anderem, bei der Bilderkennung. Außerdem sollten solche Probleme automatisiert werden, weil es sich nicht lohnt, Menschen dazu einzusetzen. In solchen Fällen, werden neuronale Netzwerke eingesetzt. Im Laufe der Zeit sind die modernen Netzwerke immer komplexer geworden und liefern nahezu menschliche Ergebnisse, im Bruchteil der Zeit, die Menschen dafür brauchen würden. Aber wie genau funktioniert so ein Netzwerk?

5 Quellen

- 1. Alexey Kravets, "Forward and Backward propagation of Max Pooling Layer in Convolutional Neural Networks", URL: <https://towardsdatascience.com/forward-and-backward-propagation-of-pooling-layers-in-convolutional-neural-networks-11e36d169bec>
- 2. T. Rashid, Neuronale Netze selbst programmieren - Ein verständlicher Einstieg mit Python. Paderborn: O'Reilly Verlag; 2017.
- 3. S. Lague, How to Create a Neural Network (and Train it to Identify Doodles). Hrsg. auf dem YouTube Kanal [Sebastian Lague](<https://www.youtube.com/@SebastianLague>). Ohne Jahr [zitiert am 16. September 2023]. Abrufbar unter: URL: <https://www.youtube.com/watch?v=hfMk-kjRv4c>.
- 4. "Ableitung der Sigmoid-Funktion" URL: <https://ichi.pro/de/ableitung-der-sigmoid-funktion-91708302791054>
- 5. "What is the role of the bias in neural networks?" URL: <https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks>.
- 6. "How do I choose the optimal batch size?", URL: <https://ai.stackexchange.com/questions/8560/how-do-i-choose-the-optimal-batch-size>

6 Liste der Abbildungen

List of Figures

1	Sigmoidfunktion mit Bias	15
---	------------------------------------	----