# Neuronale Netze in der Bildverarbeitung
# Versuch-1

Julian Schmidt, Vincenz Forkel

25. November 2021

# Inhaltsverzeichnis

# 1 Einführung

Ziel des ersten Versuchs des Praktikums Neuronale Netze in der Bildverarbeitung im Wintersemester 2021/22 ist es Grundlagen des Aufbaus und der Arbeit mit Neuronalen Netzen kennen zu lernen.
Konkret wird hier der MNIST-Datensatz verwendet, für den ein Neuronales Netz darauf trainiert werden soll Ziffern zwischen 0 und 9 zu klassifizieren.

Der MNIST (Modified National Institute of Standards and Technology database) ist eine öffentlich verfügbare Datenbank von handgeschriebenen Ziffern.
Wobei jede Ziffer als $28 \times 28$ Pixel großes Graustufen-Bild gespeichert ist.

Verwendet wird hierfür die Skript Sprache Matlab.

alle Skripte und Lösungen befinden sich in diesem Github Repository:
https://github.com/JulianSchmidt96/MessTechnikPraktikumNN/

## 2 Netzwerk Architektur

Um die Architektur des Netzwerkes zu erstellen wurde die Matlab-Toolbox für Neuronale Netze verwendet.

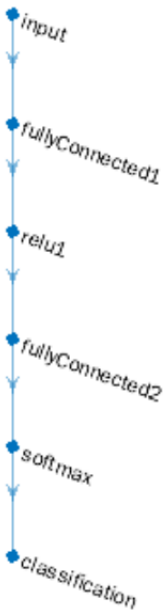
```
NN_layer = [
    imageInputLayer([28  28  1],'Normalization','none','Name','input')

    fullyConnectedLayer(1000,  'Name','fullyConnected1')
    reluLayer('Name','relu1')

    fullyConnectedLayer(10,'Name','fullyConnected2')
    softmaxLayer('Name','softmax')

    classificationLayer('Name','classification')
   ];
```

Wie hier zu sehen ist besteht das Netz aus zwei 'fully connected layer', einem 'relu layer', einem 'softmax layer' und einem 'classification layer'.




Netz Architektur

# 3 Aufgaben

## 3.1 Aufgabe 1

### 3.1.1 Aufgabenstellung

1. Nutzen Sie die gegebene Matlab-Vorlage und laden Sie Bilder und Labels aus dem MNIST-Datensatz. Teilen Sie die Daten in Trainings- und Validationsdaten auf.

### 3.1.2 Lösung

Die Daten werden aus der Neuronale Netze Toolbox Bibliothek geladen.
aus 1000 geladenen Daten pro Ziffer werden 750 für Trainings-Daten und 250 für Validierungs-Daten verwendet

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
fullDataset = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
numTrainFiles = 750;
[TrainSet,ValidationSet] = splitEachLabel(fullDataset,numTrainFiles,'randomize'
```

## 3.2 Aufgabe2

### 3.2.1 Aufgabenstellung

Erstellen Sie ein neuronales Netz zur Bildklassifizierung. Dieses sollte zum Beispiel aus einem Image input layer, einem Fully connected layer, einem ReLU layer, einem weiteren Fully connected layer, einem Softmax layer und einem Classification layer (nur bei Verwendung der Funktion trainNetwork) bestehen. Achten Sie darauf, dass das Softmax-Layer einen Vektor mit einer Länge entsprechend der zu erkennenden Klassen benötigt, um die Klassifizierung zu ermöglichen. Benutzen Sie den Befehl analyzeNetwork, um ihre Struktur auf Fehler zu prüfen. Legen Sie dann die Hyperparameter für das Training fest.

### 3.2.2 Lösung

Prinzipiell geh es hier bei um die Netzwerkarchitektur, welche bereits in Abschnitt 2 erläutert.

## 3.3 Aufgabe 3

### 3.3.1 Aufgabenstellung

Trainieren Sie das neuronale Netz mit der Funktion trainNetwork oder mit einer eigenen Trainingsschleife. Es ist wünschenswert, wenn Sie beide Varianten ausprobieren. Bei korrekter Implementierung werden beide Wege funktionieren und zu vergleichbaren Ergebnissen führen. Jedoch können Sie bei der erfolgreichen Implementierung einer benutzerdefinierten Trainingsschleife ein umfangreiches Verständnis zum Training Neuronaler Netze gewinnen. Nutzen Sie zunächst für beide Optionen den Solver Adam und evaluieren Sie das Ergebnis, indem Sie das Netz anhand der Validierungsdaten testen. Mit geeigneten Parametern sollte es möglich sein, eine Genauigkeit von mindestens 95 zu erreichen.

### 3.3.2 Lösung

Hierfür werden relevante Trainingsoptionen in der Variable 'modelOptions zwischengespeichert.

```
modelOptions = trainingOptions('adam', ...
        'InitialLearnRate',0.0001, ...
        'MaxEpochs',30, ...
        'ValidationData',ValidationSet, ...
        'ValidationFrequency',val_freq, ...
        'Verbose',false);
```

Mit dem Optimizer Adam lies sich bei einer Trainingsdauer von 30 Epochen eine finale Validierungsgenauigkeit von ca. 98 Prozent erreichen.

## 3.4 Aufgabe 4

### 3.4.1 Aufgabenstellung

Ermitteln Sie die mittlere Klassifizierungsgenauigkeit des NN in Ab- hängigkeit der eingelesenen Ziffern.
Stellen Sie das Ergebnis in einem geeigneten Diagramm ($\mu$ = f (Ziffer)) dar

### 3.4.2 Lösung

## 3.5 Aufgabe 5

### 3.5.1 Aufgabenstellung

Vergleichen Sie nun die beiden Solver Adam und Sgdm mit mindestens 6 verschiedenen Lernraten zwischen 106 und 101 . Stellen Sie die er- reichte mittlere Klassifizierungsgenauigkeit in Abhängigkeit von der Lernrate dar und verwenden Sie für die Lernrate eine logarithmische Achse.

### 3.5.2 Lösung

in zwei for-schleifen werden beide optimizier mit den verschiedenen Lernraten und sonst gleichen parametern getestet und es werden Lernrate+Optimizer mit der höchsten finalen Genauigkeit übernommen.

```
start_lr = 10^-6;
end_lr = 10^-1;

lrs = multiplicatedarray(start_lr, end_lr, 10);
epochs = 10;

val_freq = 30;



opts = {'adam','sgdm'};
optimizier = opts{1};
for opt =1:length(opts)

    ac = zeros(size(lrs));
    for lr = 1:length(lrs)
        modelOptions = trainingOptions(opts{:,opt}, ...
        'InitialLearnRate',lr, ...
        'MaxEpochs',epochs, ...
        'ValidationData',ValidationSet, ...
        'ValidationFrequency',val_freq, ...
        'Verbose',false);
        [net, results] = trainNetwork(TrainSet,NN_layer,modelOptions);
```

```
        x = results.FinalValidationAccuracy;
        ac(lr) = x;


    end

valacc = 0;

if max(ac) > valacc
    optimizier = opts(opt);
    [valacc,inlr] = max(ac);
end
end



function arr = multiplicatedarray(startval, endval, stepmulti)
    i=startval;
    k = 1;

    arr = [];

    while i~=endval
       arr(k) = i;
       i = i * stepmulti;
       k = k + 1;
    end

end
```

6

### 3.6 Aufgabenstellung

Testen Sie außerdem mindestens 5 verschiedene Größen des Mini-Batch zwischen 16 und 256. Wählen Sie aus der vorherigen Aufgabe eine Lern- rate aus, bei der der Trainingsprozess zu einer hohen Klassifizierungs- genauigkeit konvergiert. Stellen Sie die mittlere Klassifizierungsgenau- igkeit und die benötigte Trainingszeit in Abhängigkeit von der Größe des Mini-Batches dar.

### 3.7 Lösung

Es wird ein Array aus 5 verschiedenen Minibatchgrößen erstllt und dann wird das Netz in einer for-schlefe mit jeder Minibatch-Größe erneut trainiert, hierbei wird verglichen welche Minibatch-Größe die geringste Trainingszeit besitzt.

```
start_mbatch = floor(255/6);
end_mbatch = floor(255/6) * 6;

mbatches = addedarray(start_mbatch, end_mbatch, start_mbatch);



mean_val = zeros(size(mbatches));
times = zeros(size(mbatches));
for mb = 1:length(mbatches)

modelOptions = trainingOptions(opts{:,opt}, ...
          'InitialLearnRate',lr, ...
          'MaxEpochs',epochs, ...
          'ValidationData',ValidationSet, ...
          'ValidationFrequency',val_freq, ...
          'MiniBatchSize',mbatches(mb), ...
          'Verbose',false);
          %[net, results] = trainNetwork(TrainSet,NN_layer,modelOptions);
          f = @() trainNetwork(TrainSet,NN_layer,modelOptions);
          time = timeit(f);
          %x = results.ValidationAccuracy;
         % x(find(isnan(x)))=[];
          %mean_val(mb) = mean(x);
          times(mb) = time;

end
%[val,in] = max(mean_val);
[valtime,inmb] =min(times);
sprintf('%.15g seconds is the fastest training time with a minibatchsize of
minibatchsize = %.15g with the learning rate %.15g and the optimizer %s',valtim
```

```
function arr = addedarray(startval, endval, stepadd)
    i=startval;
    k = 1;

    arr = [];

    while i~=endval
        arr(k) = i;
       i = i + stepadd;
       k = k + 1;
    end
```

## 3.8 Aufgabe 7

### 3.8.1 Aufgabenstellung

Leiten Sie aus den Trainingsergebnissen Zusammenhänge zwischen den untersuchten Hyperparametern und der Trainingszeit sowie der erreichten Genauigkeit bei der Klassifizierung ab. Formulieren Sie hierfür eine kurze Diskussion.

### 3.8.2 Lösung

Es hat sich eindeutig gezeigt, dass Adam ein genauerer Optimizer als SGDM ist.
Die Lernrate scheint ebenfalls einen großen Einfluss auf die Genauigkeit des Netzes zu haben. Auch wenn sich die Genauigkeit beinahe durchgehend in einem Bereich größer 90 Prozent befindet lässt sich die Genauigkeit immernoch etwas verbessern durch das durchprobieren verschiedener Lernraten. Hierbei wurden die Lernraten jeweils um den Faktor 10 verändert.

Testweise haben wir in einem großem Durchlauf einmal den Lernratenarray so angepasst das wir ca 50 verschiedene Lernraten getestet haben.
Hierfür muss lediglich der lernratenarray angepaßt werden, das Skript findet selbstständig die ideale Lernrate. Zu Beachten ist allerdings, dass hier für eine sehr lange Laufzeit eingeplant werden muss.

Wir haben hierfür einmal das Skript über ein Wochenende laufen lassen.
Eine geschickte Anpassung der Minibatch-Größen würde die Trainings-Dauer und somit Gesamtlaufzeit verringern.
Die Ergebnisse haben gezeigt, dass Änderungen der Minibatch-Größe einen direkten Einfluss auf die Laufzeit hat.

Beim Testen hat sich gezeigt, dass für verschiedene Minibatch-Größe unterschiedliche Genauigkeiten erreicht wurden.
Das liegt allerdings mit hoher Wahrscheinlichkeit da dran, dass jede Prädiktion auf statistischen Größen basiert.
Wenn ein Neuronales Netz mehrfach mit exakt gleichen Parametern trainiert wird, ist es unmöglich immer zu 100 Prozent das gleiche Ergebnis zu erhalten.

Eine deutliche Genauigkeitsverbessserung lies sich durch das hinzufügen der Zeile:

'Shuffle','every-epoch', ...
zu den modeloptionen.
genauigkeit, allerdings steigt damit die Laufzeit sehr stark.
Die Validationsfrequenz auf einen sehr kleinen Wert zusätzen sorgt ebenfalls für eine erhöhte Grundsätzlich lässt sich mit relativ wenig Aufwand eine hohe Genauigkeit in der Vorhersage von Ziffern auf 28x28 Pixel Bilder mit dem MNIST Datensatz erreichen.