Deep Learning Assignment 1

ID: 641245

December 2020

1 Perceptron

1.1 Introduction & Simple Problem

The simplest neural network is the perceptron, initially formulated by Rosenblatt 1958. It consists of an input layer and an output layer (fig. 1). The input layer provides the inputs, x_j, and can be described by the input vector, x. Each input x_i has an associated weight, w_i, which determines the weight of the respective input. The output layer is fed the weighted input vector:

$$z = w \cdot x \tag{1}$$

where w is the weights vector and z is the scalar output. The output z is then put into a transfer function, y, which yields the final result. The problem we are trying to solve is one of binary classification: based on a binary sequence of length n, where $n = 10$ and each position x_i can take either value 1 or -1, is the result of the following equation "*positive*" or "*negative*":

$$t = \sum_{i=1}^{n} x_i \tag{2}$$

I will define $t = 0$ to be part of the "*positive*" category and arbitrarily assign the categories numericals: "*positive*" $= 1$ and "*negative*" $= 0$. To solve this problem we need to choose the appropriate transfer function y. An intuitive solution might be a step-function:

$$y = \begin{cases} 1 & \text{if} \quad w \cdot x \geq \theta \\ 0 & \text{if} \quad w \cdot x < \theta \end{cases} \tag{3}$$

where θ is some threshold. While intuitive, it proves intrinsically problematic. This is because we need to be able to adjust the weights in w so that we can optimise the perceptron. We do this by minimising an error function E. I will define the error function as follows:

$$E = \frac{1}{2}(t - y)^2 \tag{4}$$

where t is the *training signal*, i.e. the true result of equation (2) with respect to some input vector x. To effectively minimise the error function we need to perform *gradient descent*, i.e. walk in the opposite direction of the gradient of the error-surface with respect to w_i. For this we need to find the derivative of E with respect to w_i:

$$\frac{dE}{dw_i} = -(t - y) \cdot y' \cdot x_i \tag{5}$$

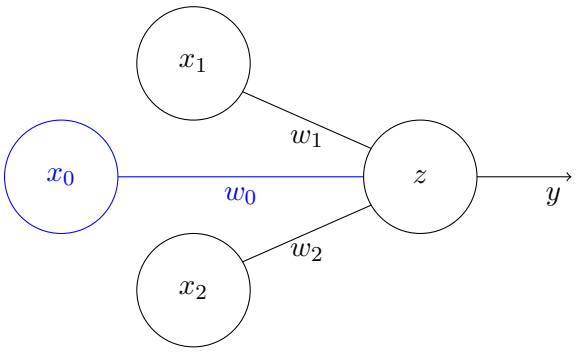


Figure 1: A simple perceptron with two inputs, x_1 and x_2, and a single output unit, z, with activation y. Each input has an associated weight, w_1 and w_2 respectively. Additionally, I treat the threshold (or the *bias*) as another input x_0 (always = -1) with its respective weight w_0.

The problem should now become apparent. The derivative y' of a step-function is 0 at all values of z, except at $z = \theta$, where it is undefined. Thus, we should to use the logistic function as our transfer function:

$$y = \frac{1}{1 - e^{-z}} \tag{6}$$

$$y' = y(1 - y) \tag{7}$$

All we need to add now is the step-size ϵ to determine the rate at which we are descending down the error-surface, giving us the change for w_i for an iteration:

$$\Delta w_i = -\epsilon \frac{dE}{dw_i} \tag{8}$$

Figure 2 shows that this problem is very easily solvable by a simple perceptron. The perceptron can draw a $N - 1$ dimensional hyperplane in N dimensional space to separate the space into two subspaces — one for each class. My perceptron is able to reach a state of $E \simeq 0$ very quickly, dependent solely on ϵ ($\epsilon = 0.01$ shown in figure 2). I do this by first dividing the total sample space S ($|S| = 1024$) into a training set A ($|A| = 819$; $\frac{|A|}{|S|} \simeq 0.8$) and a test set B ($|B| = 205$; $\frac{|B|}{|S|} \simeq 0.2$). Then I train the perceptron on A for X number of epochs ($X = 200$), updating w with each iteration and storing the updated w at the end of each epoch. Simultaneously, I calculate and store the mean epoch error.

Subsequently, I run the perceptron with each stored w on B and calculate the mean error for each w. The results are shown in figure 2. Somewhat surprisingly, the perceptron seems to be performing better on the test-set, than on the training set.

1.2 The Parity Problem

The above described perceptron proves to be both simple and powerful when trying to solve linearly-separable problems. However; it is effectively useless when presented with seemingly trivial problems such as the *XOR* problem, which is not linearly separable. The *Parity Problem* is an analogy of this, where we try to provide a binary classification to an N dimensional input vector x based on the sign of the result of $\prod_{i=1}^{N} x_i$ — each element x_i is either 1, or -1. A version of this problem, where $N = 2$ is shown in figure 3a. As can be seen, there is no hyperplane which can separate the results perfectly. This is also reflected in the jittery error-trace produced by the perceptron when trying to solve a version of

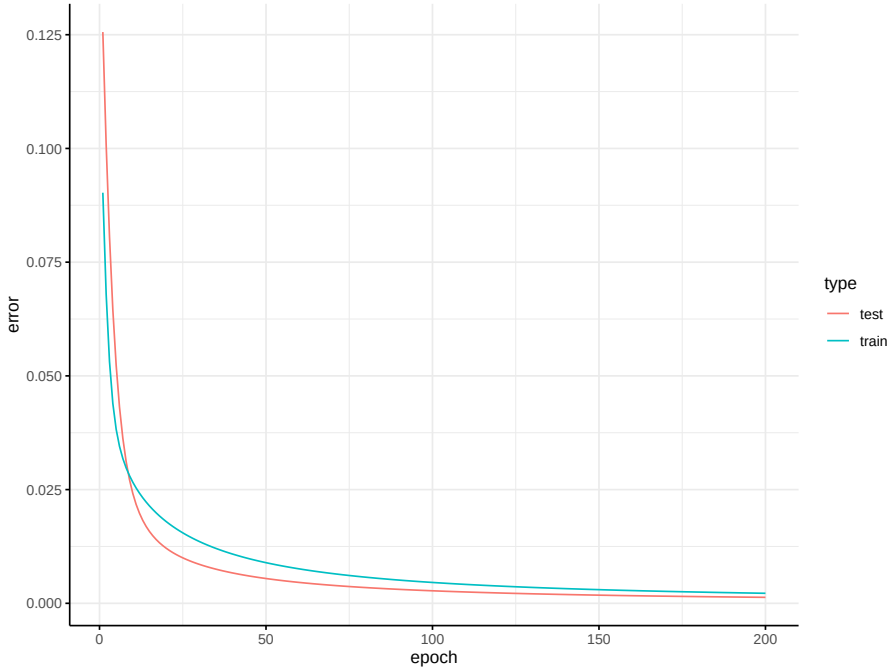


Figure 2: The mean per-epoch error of the perceptron for train and test data subsets. The perceptron converges quickly, reaching the global minimum after X epochs, depending on the step-size ϵ.

this problem where $N = 10$. The perceptron is unable to converge when presented with this problem.

This was initially shown in Minsky and Papert 1969, leading to the first "*winter*" in AI research.

2 Self-supervised learning

To overcome this issue of non-linearity, *Multi-layer Perceptrons* (MLPs) were devised (fig. 4). The MLP is made up of an input layer, X numbers of hidden layers and an output layer. Each layer contains n number of units, i.e. dimensions in the vector space. Each unit in a given layer is connected to each unit in the consecutive layer, with an associated weight to a given connection. In mathematical terms, we are performing a series of linear transformations from input vector z_i to output vector z_k using weight matrices $\{W_1, W_2, ..., W_{X+1}\}$ to map the transformations.

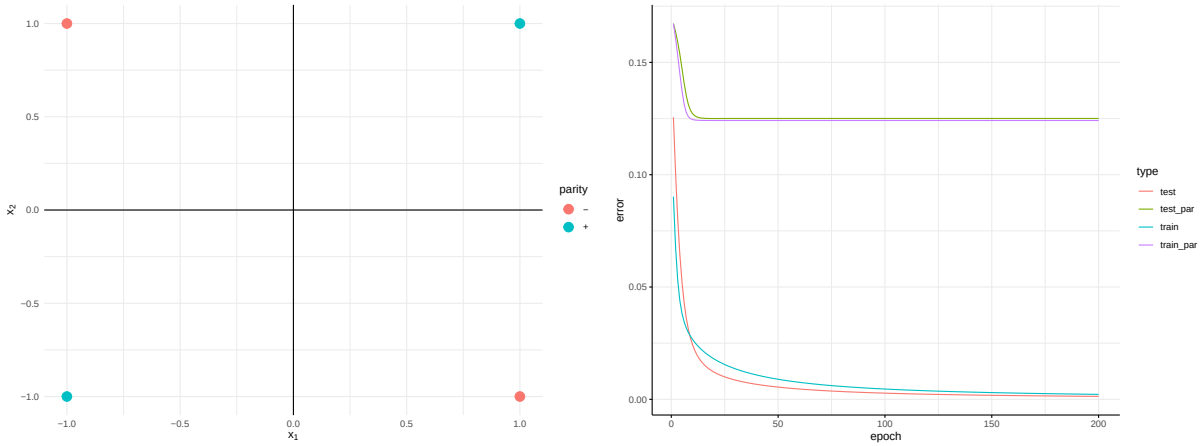
For our example, we will be using an 8 dimensional input layer, a single 3 dimensional hidden layer and an 8 dimensional output layer. The transformations are formalised as follows:

$$x_j = W_1 \cdot z_i$$
$$z_j = g(x_j)$$

$$x_k = W_2 \cdot z_j$$
$$z_k = g(x_k)$$

where $g(x)$ is a vector function, taking the weighted sum of the inputs from the previous layer and returning the activation of each unit. In our case the activation function is sigmoidal and defined in (6).

To appropriately adjust the weights when measuring the network output against some training signal t, we



(a) 2D representation

(b) error plot

Figure 3: The perceptron is unable to converge in non-linearly separable problems. Figure (a) shows a simplified 2D representation of the parity problem, analogising to a *XOR* gate. Figure (b) is the error-trace of the perceptron across epochs.

need to be able to *back-propagate* the network error. *Back-propagation* (backprop) is the approximation of the error of the hidden units, which do not have a target value. We will backprop using the simple quadratic loss function (4), using the derivation outlined by Stephen Eglen.

The target vector t is a 1-HOT encoded classifier, i.e. all positions in the vector are 0 take 1. When considering all classes, this analogises to an identity matrix, where each column signifies a unique class. In our example, we will get an 8×8 identity matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We train the model on all of these, using each column vector as both input and target, so that the model will learn to return the input vector. In machine learning terms, we are attempting to solve the self-supervision problem of the auto-encoder. An auto-encoder is used for *representation learning*, where we compress the feature space using a "bottle-neck". A bottle-neck is a hidden layer with a feature space of lower dimensionality than the input and the output space (fig. 4). The auto-encoder finds nonlinear manifolds in a data-set, which it can use to classify different inputs. More specifically, we will be looking at an "undercomplete auto-encoder", i.e. one without a regularisation term appended to the ΔW function. This is because we have a very small, clearly defined sample space I and thus do not need to worry about over-fitting.

Looking at figure 5, we can see that the MLP is able to reach $\simeq 0$ error within a few hundred epochs using batch learning. The MLP also successfully reproduces the input vector:

```
print(t_k) # prints the target vector
[0. 0. 0. 0. 0. 0. 0. 1.]
```

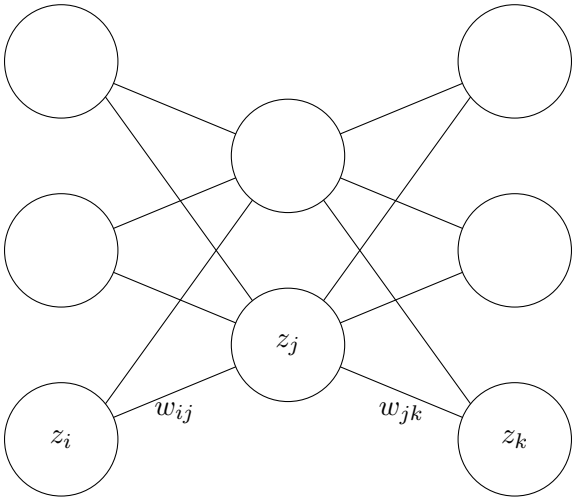


Figure 4: Schematic of a MLP with 3 layers: an input layer z_i; a hidden layer z_j; and an output layer z_k. These layers are densely connected, with each connection having an associated weight w.

```
print(np.round(z_k)) # prints the MLP output rounded to 0 decimals
[0. 0. 0. 0. 0. 0. 0. 1.]
```

The α is a hyper-parameter known as the momentum and it is implemented in the Δw function as follows:

$$\Delta w_{ij}^{t+1} = -\epsilon \frac{dE}{dw_{ij}} + \alpha \Delta w_{ij}^{t} \tag{9}$$

Inspecting figure 5, we can see that the addition of a momentum term when $\alpha = 0.01$ increases rate of convergence slightly. However, tuning the α term (and the ϵ term) can create vastly differing results in MLP output (fig. 6). In fact, the MLP is especially sensitive to increments in α, as increases past $\alpha = 0.01$ causes slower convergence, no convergence and/or emergent noise.

3 MNIST classification

The "*Hello World*" of ML, the Modified National Institute of Standards and Technology (MNIST) data-set provides a good example for experimenting with various hyper-parameters without having to worry about data pre-processing (fig. 7). Hyper-parameters includes things such as number of hidden layers, number of per-layer units, learning rate and choice of optimizer and loss-function.

ML APIs such as *Keras* make building neural nets efficient and straightforward, removing the need to concern oneself with specifics. It adds a layer of abstraction to NN construction, changing it from a mathematical challenge to an engineering one.

3.1 Model 1: Perceptron

Figure 8 shows the schematic of a simple perceptron (`model_1`) built with the `keras.Sequential()` class for classifying the MNIST data-set. The data-set is provided in $n \times 28 \times 28$ `numpy` arrays and split into training ($n = 6 \times 10^4$) and test ($n = 1 \times 10^4$) sets. First, we reshape the data using `numpy.reshape()`, which changes the dimensions of the data from an array of shape `(None, 28, 28)` to `(None, 784)` — i.e. it makes each 28×28 matrix representing an individual digit into a 784-element long row-vector. The

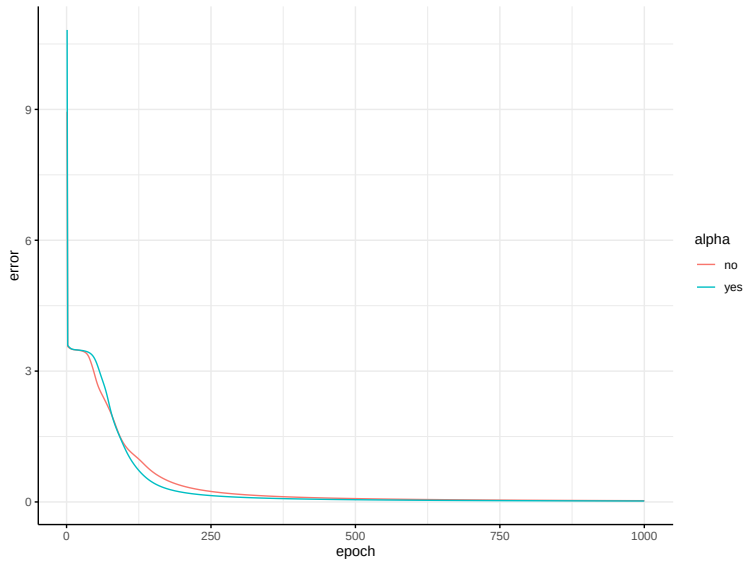
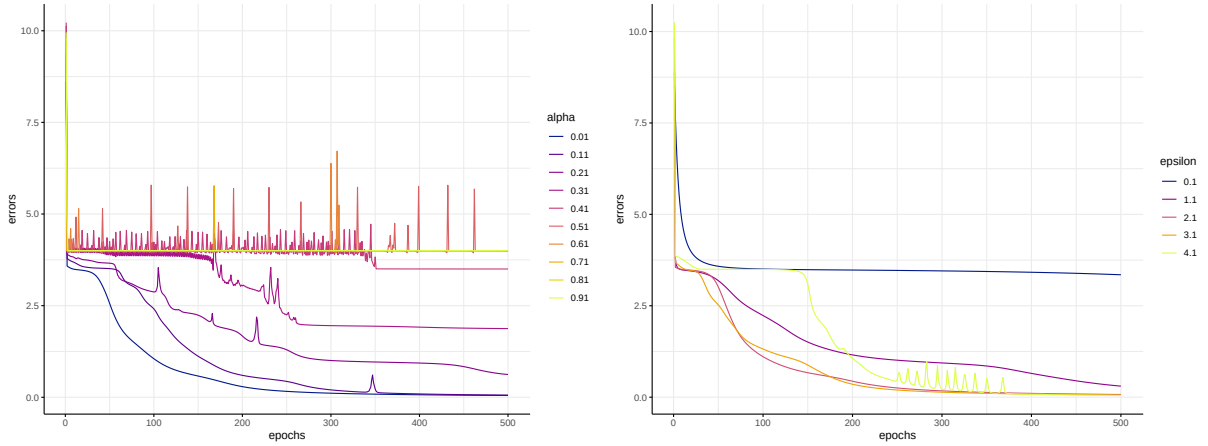


Figure 5: The change in MLP quadratic error over epochs — with and without a momentum term (i.e. alpha = yes or no).



(a) testing α (b) testing ϵ

Figure 6: Effects of changing hyper-parameters on the network learning rate. **(a)** The network error of per epoch with different values of α. **(b)** The network error per epoch with different values of ϵ.

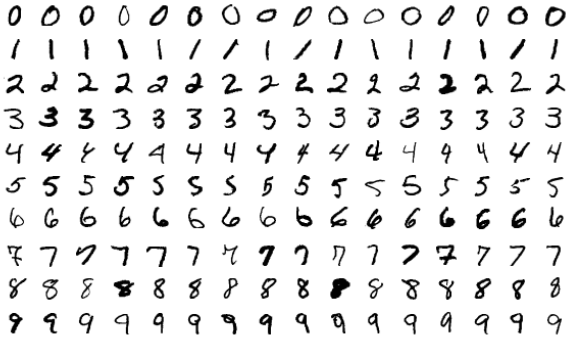


Figure 7: Example of the MNIST data-set. Handwritten digits used to train neural nets.

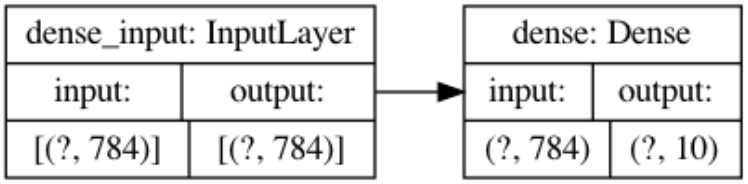


Figure 8: A schematic of the perceptron: no hidden layers.

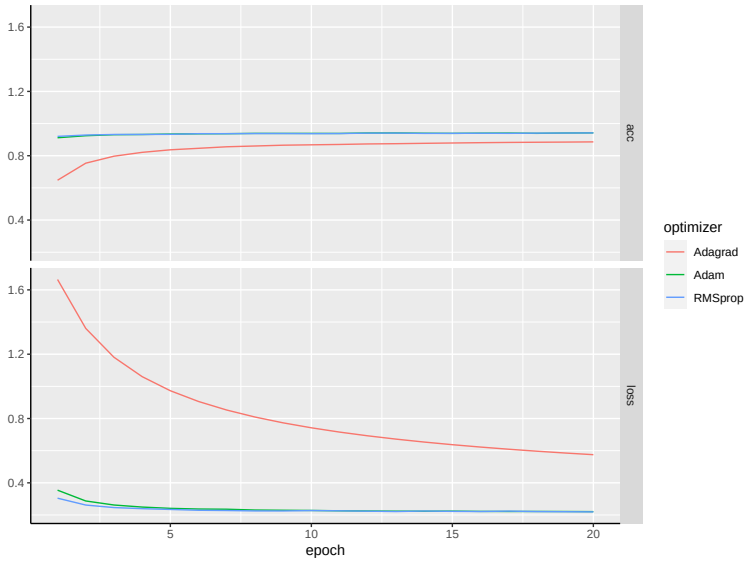


Figure 9: The changes in loss and accuracy of the validation sets over epochs of the different perceptron models.

`None` denotes "undefined", i.e. it depends on the number of data-points in the data-set. We then rescale the data, so that each element in a row vector is between 0 and 1.

There is only one fully connected (or "dense") layer — the *output layer*. It's activation is defined as a *softmax* function. The model is compiled using various optimizers: Adagrad, Adam and RMSprop. The loss function was defined as the `sparse_categorical_crossentropy` function.

Looking at figure 9, we can see that models running Adam and RMSprop optimizers fare similarly, while Adagrad converged significantly slower. When running the test set on the Adam-model, 9164 digits were correctly classified, 561 were incorrectly classified and 275 where not classified.

3.2 Model 2: MLP

The set of type 2 models are MLP variants: figure 11a shows 3 MLP models (varying optimizer) with a single dense hidden layer using 16 units and a $ReLU$ activation function; figure 11b shows 3 MLP models (varying optimizer) with two dense hidden layers — 400 and 200 units respectively — and with $ReLU$ activation functions. The schematic of the models from figure 11b is summarised in figure 10. The data for training and testing was prepared the same way as for the perceptron models.

The single hidden layer (SHL) and multi hidden layer (MHL) models performed comparably and slighly better than the perceptron models. Interestingly, shown in figure 11b, the RMSprop optimizer MHL starts unlearning earlier than any of the other MLP models. Like the perceptron models, the optimizer

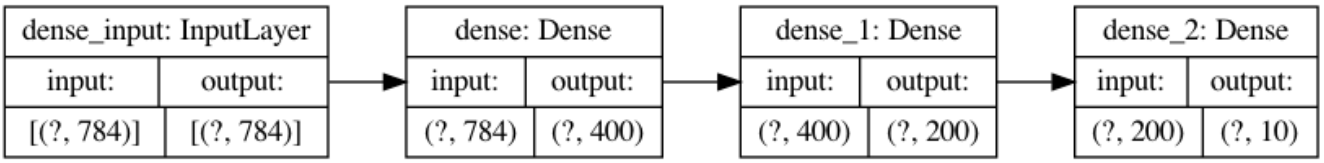
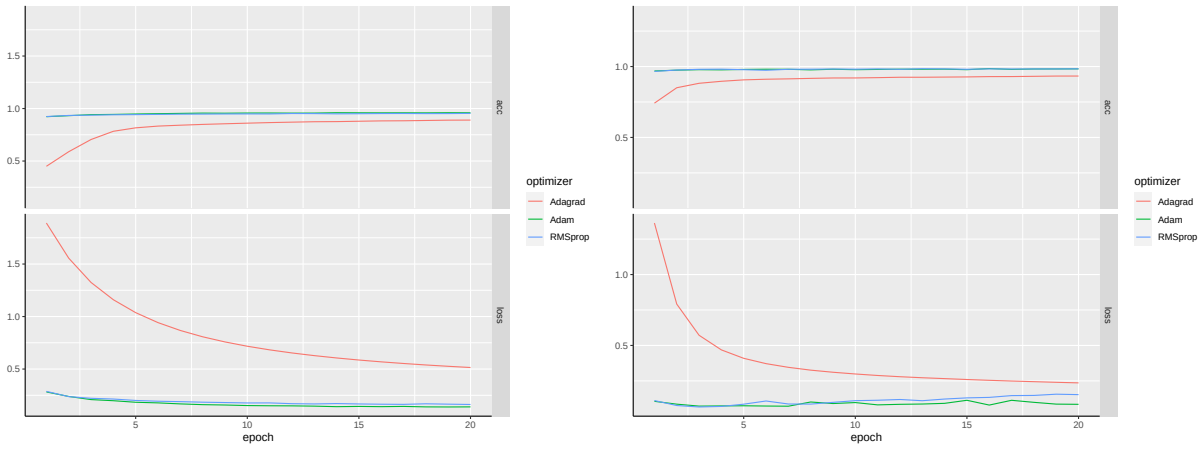


Figure 10: Schematic of the (MHL) MLP model: two hidden layers with number of units = 400 and 200 respectively.



(a) SHL (b) MHL

Figure 11: The changes in loss and accuracy of the validation sets over epochs of the different MLP models. **(a)** shows the results for the SHL models and **(b)** shows the results for the MHL models.

trends hold. Finally, the second MHL models converged earlier than the SHL models.

Running the Adam MHL model on the test data-set we receive 9829 correctly classified digits, 162 incorrectly classified digits and 9 unclassified digits.

3.3 Model 3: CNN

The third set of models are classified as convolutional neural networks (CNN). The schematic for the models is outlined in figure 12. I vary the optimizer the models are compiled with (Adam, Adagrad, RMSprop). Before training the models, I reshaped the data into a tensor with the following dimensions: (number of images) $\times$ (width of image) $\times$ (height of image) $\times$ (number of channels). For our training set, for example, the data set was in the dimensions were $60000 \times 28 \times 28 \times 1$.

As can be seen in figure 13, the cross-optimizer trend holds from the previous two models, with Adam and RMSprop performing comparably and Adagrad converging at a much reduced rate. The model compiled with the Adam optimizer returned 9891 correctly classified digits, 106 incorrectly classified digits and 3 unclassified digits when run on the test data-set.

3.4 Cross-model comparison

All models performed reasonably, however some performed better than others. In general, the most effective optimizers were Adam and RMSprop, which performed comparably. Looking at figure 14 shows that more layers consistently produced more accurate classifications, clearly displayed in figure 14d. This

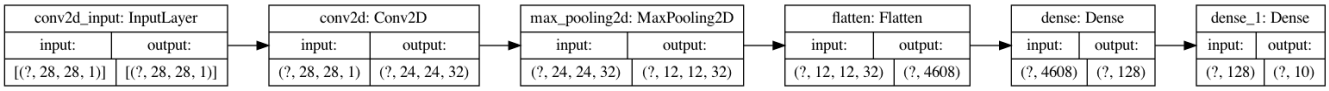


Figure 12: Schematic of the CNN models: many hidden layers, including a 2D convolution layer and a pooling layer.

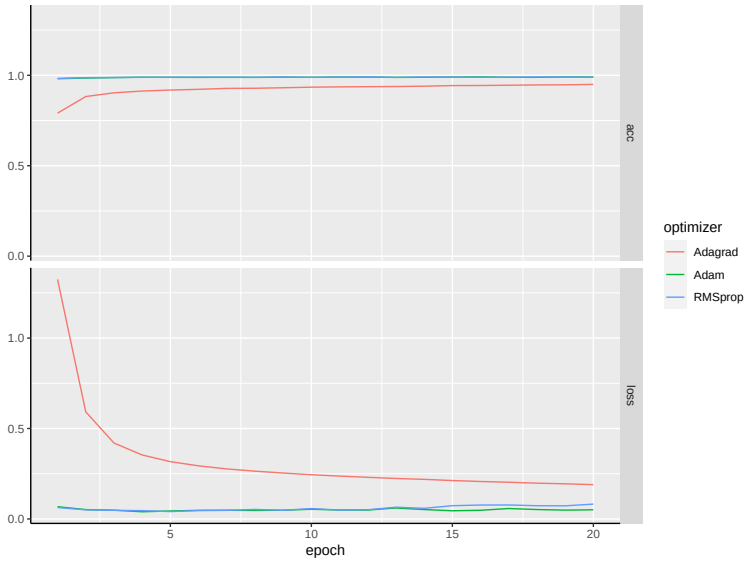
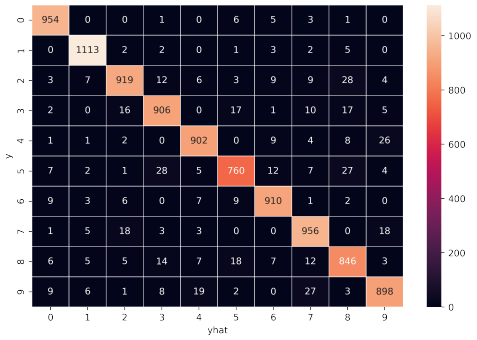
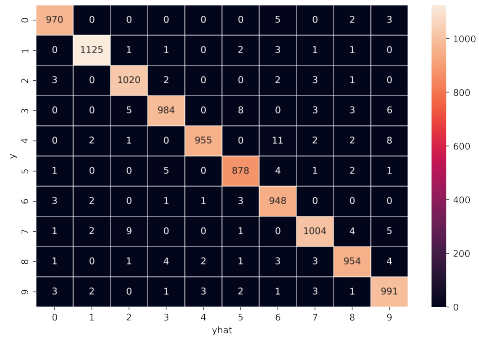


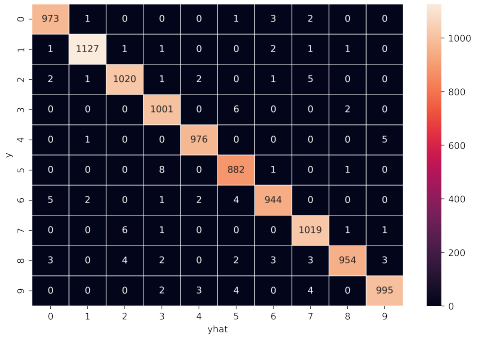
Figure 13: The changes in loss and accuracy of the validation sets over epochs of the different CNN models.



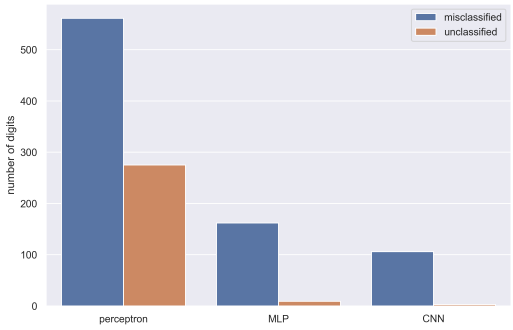
(a) perceptron



(b) MLP



(c) CNN



(d) errors

Figure 14: Comparing the predictions by the different models with the ground truth and with one another. **(a)** — **(c)** show the confusion matrices produced by each model type compiled with an Adam optimizer when run on the test set. **(d)** shows the number of incorrectly and unclassified digits for each model-type.

figure shows the number of incorrectly classified and unclassified digits for each Adam-compiled model type when run on the test set.

Inspecting the confusion matrices shown in figures 14a—14c we can see that across all models, the most difficult digit to classify was 5 and the easiest was 1.

References

Minsky, M. and S. Papert (1969). "An introduction to computational geometry". In: *Cambridge tiass., HIT*.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

A Code

A.1 Perceptron

A.1.1 Summation

```
 #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 11 20:21:59 2020

@author: benjamintenmann
"""


import numpy as np
from itertools import product
from math import exp
from scipy.stats import uniform
import matplotlib.pyplot as plt

def g(x):
    z = 1/(1+exp(-x))
    return z

def gprime(x):
    der = g(x) * (1-g(x))
    return der

def loss(t,y):
    err = 0.5 * (t-y)**2
    return err

def ifelse(condition, t=1, f=0):
    if condition == True:
        return t
    else:
        return f

def gradient_descent(t, w, x):
    z = x.dot(w)
    delta_w = (t-g(z)) * gprime(z) * x
    return delta_w

epochs = 200
epsilon = 0.01
inputs = list(product([-1,1], repeat=10))

w = uniform.rvs(size=11)


# training
ep_err = []
W = [w.copy()]
for i in range(epochs):

    err = 0

    for n in range(819):
```

```
54
55          z = np.array(inputs[n])
56          t = ifelse(sum(z) >= 0)
57
58          x = np.append(z, -1)
59
60
61          w += epsilon * gradient_descent(t, w, x)
62          err += loss(t, g(x.dot(w)))
63
64
65      W.append(w.copy())
66      ep_err.append(err/819)
67
68  plt.plot(list(range(epochs)), ep_err)
69  plt.show()
70
71  # testing
72  test_err = []
73  for w in W:
74
75      err = 0
76
77      for p in range(205):
78
79          z = np.array(inputs[p+819])
80          t = ifelse(sum(z) >= 0)
81
82          x = np.append(z, -1)
83
84          err += loss(t, g(x.dot(w)))
85
86      test_err.append(err/205)
87
88  plt.plot(list(range(epochs+1)), test_err)
89  plt.show()
90
91  import pandas as pd
92  T = pd.DataFrame({"err_train":ep_err, "err_test":test_err[:-1]})
93  T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/perceptron/perc_out.
        csv")
```

A.1.2 Parity

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Dec 11 20:21:59 2020
5
6  @author: benjamintenmann
7  """
8
9
10 import numpy as np
11 from itertools import product
12 from math import exp
13 from scipy.stats import uniform
14 import matplotlib.pyplot as plt
```

```
15
16 def prod(x):
17     r = 1
18     for i in x:
19         r *= i
20     return r
21
22 def g(x):
23     z = 1/(1+exp(-x))
24     return z
25
26 def gprime(x):
27     der = g(x) * (1-g(x))
28     return der
29
30 def loss(t,y):
31     err = 0.5 * (t-y)**2
32     return err
33
34 def ifelse(condition, t=1, f=0):
35     if condition == True:
36         return t
37     else:
38         return f
39
40 def gradient_descent(t, w, x):
41     z = x.dot(w)
42     delta_w = (t-g(z)) * gprime(z) * x
43     return delta_w
44
45 epochs = 200
46 epsilon = 0.01
47 inputs = list(product([-1,1], repeat=10))
48
49 w = uniform.rvs(size=11)
50
51
52 # training
53 ep_err = []
54 W = [w.copy()]
55 for i in range(epochs):
56
57     err = 0
58
59     for n in range(819):
60
61         z = np.array(inputs[n])
62         t = ifelse(prod(z) > 0)
63
64         x = np.append(z, -1)
65
66
67         w += epsilon * gradient_descent(t, w, x)
68         err += loss(t, g(x.dot(w)))
69
70
71     W.append(w.copy())
72     ep_err.append(err/819)
73
```

```
plt.plot(list(range(epochs)), ep_err)
plt.show()

# testing
test_err = []
for w in W:

    err = 0

    for p in range(205):

        z = np.array(inputs[p+819])
        t = ifelse(prod(z) > 0)

        x = np.append(z, -1)

        err += loss(t, g(x.dot(w)))

    test_err.append(err/205)

plt.plot(list(range(epochs+1)), test_err)
plt.show()

import pandas as pd
T = pd.DataFrame({"err_train":ep_err, "err_test":test_err[:-1]})
T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/perceptron/perc_out.
    csv")
```

A.2 MLP

A.2.1 no alpha

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 11 13:42:19 2020

@author: benjamintenmann
"""

import numpy as np
from scipy.stats import uniform
import matplotlib.pyplot as plt

def g(x):
    z = 1/(1+np.exp(-x))
    return z

def gprime(x):
    der = g(x) * (1-g(x))
    return der

def loss(t, y):
    err = 0.5 * sum((t-y)**2)
    return err

bias = -1
epsilon = 2.3
```

```

targets = np.identity(8)
inputs = np.vstack((targets, (np.ones(8)*bias)))

I = 8
J = 3
K = 8

epochs = 50

W1 = uniform.rvs(size=(J*(I+1))).reshape(J, (I+1))
W2 = uniform.rvs(size=(K*(J+1))).reshape(K, (J+1))

delta_j = np.zeros(J)

epoch_error = []

for i in range(epochs):
    DW1 = np.zeros((J, I+1))
    DW2 = np.zeros((K, J+1))

    ep_er = 0
    for n in range(inputs.shape[1]):

        z_i = inputs[:, n]
        t_k = targets[:, n]

        x_j = W1 @ z_i
        z_j = g(x_j)
        z_j = np.append(z_j, bias)

        x_k = W2 @ z_j
        z_k = g(x_k)

        ep_er += loss(t_k, z_k)
        delta_k = gprime(x_k) * (t_k - z_k)

        DW2 = DW2 + np.outer(delta_k, z_j)

        delta_j = gprime(x_j) * (delta_k @ W2[:,:-1])

        DW1 = DW1 + np.outer(delta_j, z_i)
    epoch_error.append(ep_er)
    W2 += epsilon * DW2
    W1 += epsilon * DW1


plt.plot(list(range(1,epochs+1)), epoch_error)
plt.show()

import pandas as pd
T = pd.DataFrame({"epochs":list(range(1,epochs+1)), "error":epoch_error})
T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/noalpha.csv",
    index=False)
```

A.2.2 alpha

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 11 13:42:19 2020

@author: benjamintenmann
"""

import numpy as np
from scipy.stats import uniform
import matplotlib.pyplot as plt

def g(x):
    z = 1/(1+np.exp(-x))
    return z

def gprime(x):
    der = g(x) * (1-g(x))
    return der

def loss(t, y):
    l = 0.5 * sum((t-y)**2)
    return l

bias = -1
epsilon = 2.3
alpha = 0.001

targets = np.identity(8)
inputs = np.vstack((targets, (np.ones(8)*bias)))

I = 8
J = 3
K = 8

epochs = 1000

W1 = uniform.rvs(size=(J*(I+1))).reshape(J, (I+1))
W2 = uniform.rvs(size=(K*(J+1))).reshape(K, (J+1))

delta_j = np.zeros(J)

epoch_error = []

for i in range(epochs):
    DW1 = np.zeros((J, I+1))
    DW2 = np.zeros((K, J+1))

    ep_er = 0
    for n in range(inputs.shape[1]):

        z_i = inputs[:, n]
        t_k = targets[:, n]

        x_j = W1 @ z_i
        z_j = [g(q) for q in x_j]
        z_j.append(bias)

        x_k = W2 @ z_j
        z_k = g(x_k)
```

```
61
62          ep_er += loss(t_k, z_k)
63          delta_k = gprime(x_k) * (t_k - z_k)
64
65          DW2 = ((1+alpha) * DW2) + np.outer(delta_k, z_j)
66
67          delta_j = gprime(x_j) * (delta_k @ W2[:,:-1])
68
69          DW1 = ((1+alpha) * DW1) + np.outer(delta_j, z_i)
70      epoch_error.append(ep_er)
71      W2 += epsilon * DW2
72      W1 += epsilon * DW1
73
74
75  plt.plot(list(range(1,epochs+1)), epoch_error)
76  plt.show()
77
78  import pandas as pd
79  df = pd.DataFrame({"epoch":list(range(1, epochs+1)), "error":epoch_error})
80  df.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/alpha.csv", index
        =False)
```

A.3 MNIST

A.3.1 Perceptron

```
1   import numpy as np
2   import tensorflow as tf
3   from tensorflow import keras
4   from tensorflow.keras.models import Sequential
5   from tensorflow.keras.layers import Dense, Activation
6   from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
7   from tensorflow.keras.metrics import categorical_crossentropy
8   import matplotlib.pyplot as plt
9   import seaborn as sns
10  import pandas as pd
11
12  (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
13  x_train = x_train.reshape((60000, 28*28))
14  x_train = x_train / 255
15  x_test = x_test.reshape((10000, 28*28))
16  x_test = x_test / 255
17
18  modelAdam = Sequential([
19      Dense(units=10, activation='softmax', input_shape=(784,))
20  ])
21
22  modelAdam.compile(optimizer=Adam(learning_rate=0.001), loss='
        sparse_categorical_crossentropy', metrics=['accuracy'])
23
24  batch_size = 128
25  n_epoch = 20
26  history = modelAdam.fit(x_train, y_train, validation_split = 0.1, batch_size=batch_size,
        epochs=n_epoch)
27
28  T = pd.DataFrame(history.history)
29
30  modelAdagrad = Sequential([
```

```
    Dense(units=10, activation='softmax', input_shape=(784,))
])

modelAdagrad.compile(optimizer=Adagrad(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdagrad.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

modelRMSprop = Sequential([
    Dense(units=10, activation='softmax', input_shape=(784,))
])

modelRMSprop.compile(optimizer=RMSprop(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelRMSprop.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_prc.csv")

val_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)
loss, acc = modelAdam.evaluate(val_dataset)
print("Adam:", loss, acc)
loss, acc = modelAdagrad.evaluate(val_dataset)
print("Adagrad:", loss, acc)
loss, acc = modelRMSprop.evaluate(val_dataset)
print("RMSprop:", loss, acc)

yhat = modelAdam.predict(x_test)
def f(yhat):
    if sum(yhat) != 0:
        r = np.where(yhat == 1)[0][0]
        return r
    else:
        pass
ls = list(map(f, np.round(yhat)))
l = []
for i, y in enumerate(np.round(yhat)):
    if sum(y) == 0:
        l.append(i)
def g(x):
    return x != None
f_ls = np.array(list(filter(g, ls)))
npL = np.array(l)
y_test_cut = np.array(y_test[np.in1d(np.arange(0, 10000), npL, invert=True)])
counts = []
for y in np.unique(y_test_cut):
    count = [0]*10
    for yhat in np.unique(f_ls):
        new = f_ls[y_test_cut == y]
```

```
        count[yhat] = len(new[new == yhat])
    counts.append(count)

n = np.array(counts)

f, ax = plt.subplots(figsize=(9, 6))

ax = sns.heatmap(n, annot=True, ax=ax, fmt='d', linewidths=.5)
ax.set(xlabel='yhat', ylabel='y')
fig = ax.get_figure()
fig.savefig('/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/heat_mlp.png', dpi=400)
print(fig)

print('corr class:', np.sum(np.diagonal(n)), '\nunclass:', len(l), '\nincorr class:', (10000 - np.sum(np.diagonal(n))) - len(l))
```

A.4 MLP

A.4.1 SHL

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from tensorflow.keras.metrics import categorical_crossentropy
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000, 28*28))
x_train = x_train / 255
x_test = x_test.reshape((10000, 28*28))
x_test = x_test / 255

modelAdam = Sequential([
    Dense(units=16, activation='relu', input_shape=(784,)),
    Dense(units=10, activation='softmax')
])
modelAdam.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdam.fit(x_train, y_train, validation_split = 0.1, batch_size=batch_size, epochs=n_epoch)

T = pd.DataFrame(history.history)

modelAdagrad = Sequential([
    Dense(units=16, activation='relu', input_shape=(784,)),
    Dense(units=10, activation='softmax')
])
modelAdagrad.compile(optimizer=Adagrad(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```

batch_size = 128
n_epoch = 20
history = modelAdagrad.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

modelRMSprop = Sequential([
    Dense(units=16, activation='relu', input_shape=(784,)),
    Dense(units=10, activation='softmax')
])
modelRMSprop.compile(optimizer=RMSprop(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelRMSprop.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_mlp1.csv")
```

A.4.2 MHL

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from tensorflow.keras.metrics import categorical_crossentropy
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000, 28*28))
x_train = x_train / 255
x_test = x_test.reshape((10000, 28*28))
x_test = x_test / 255

modelAdam = Sequential([
    Dense(units=400, activation='relu', input_shape=(784,)),
    Dense(units=200, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelAdam.compile(optimizer=Adam(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdam.fit(x_train, y_train, validation_split = 0.1, batch_size=batch_size,
     epochs=n_epoch)

T = pd.DataFrame(history.history)

```

```
modelAdagrad = Sequential([
    Dense(units=400, activation='relu', input_shape=(784,)),
    Dense(units=200, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelAdagrad.compile(optimizer=Adagrad(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdagrad.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

modelRMSprop = Sequential([
    Dense(units=400, activation='relu', input_shape=(784,)),
    Dense(units=200, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelRMSprop.compile(optimizer=RMSprop(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelRMSprop.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_mlp2.csv")

tf.keras.utils.plot_model(modelAdam, to_file='/Users/benjamintenmann/Desktop/CompBio/
    Assignments/DL_1/MNIST/model_2.1.png', rankdir='LR', show_shapes=True)

val_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)
loss, acc = modelAdam.evaluate(val_dataset)
print("Adam:", loss, acc)
loss, acc = modelAdagrad.evaluate(val_dataset)
print("Adagrad:", loss, acc)
loss, acc = modelRMSprop.evaluate(val_dataset)
print("RMSprop:", loss, acc)

yhat = modelAdam.predict(x_test)
def f(yhat):
    if sum(yhat) != 0:
        r = np.where(yhat == 1)[0][0]
        return r
    else:
        pass
ls = list(map(f, np.round(yhat)))
l = []
for i, y in enumerate(np.round(yhat)):
    if sum(y) == 0:
        l.append(i)
def g(x):
    return x != None
f_ls = np.array(list(filter(g, ls)))
npL = np.array(l)
```

```
y_test_cut = np.array(y_test[np.in1d(np.arange(0, 10000), npL, invert=True)])
counts = []
for y in np.unique(y_test_cut):
    count = [0]*10
    for yhat in np.unique(f_ls):
        new = f_ls[y_test_cut == y]
        count[yhat] = len(new[new == yhat])
    counts.append(count)

n = np.array(counts)

f, ax = plt.subplots(figsize=(9, 6))

ax = sns.heatmap(n, annot=True, ax=ax, fmt='d', linewidths=.5)
ax.set(xlabel='yhat', ylabel='y')
fig = ax.get_figure()
fig.savefig('/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/heat_mlp.png'
    , dpi=400)
print(fig)

print('corr class:', np.sum(np.diagonal(n)), '\nunclass:', len(l), '\nincorr class:',
    (10000 - np.sum(np.diagonal(n))) - len(l))
```

A.5 CNN

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, MaxPooling2D, Conv2D, Flatten
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from tensorflow.keras.metrics import categorical_crossentropy
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000, 28*28))
x_train = x_train / 255
x_test = x_test.reshape((10000, 28*28))
x_test = x_test / 255

x_train = x_train.reshape((60000, 28, 28, 1))
x_test = x_test.reshape((10000, 28, 28, 1))

modelAdam = Sequential([
    Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelAdam.compile(optimizer=Adam(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdam.fit(x_train, y_train, validation_split = 0.1, batch_size=batch_size,
```

```
    epochs=n_epoch)

T = pd.DataFrame(history.history)

modelAdagrad = Sequential([
    Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelAdagrad.compile(optimizer=Adagrad(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelAdagrad.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

modelRMSprop = Sequential([
    Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=10, activation='softmax')
])
modelRMSprop.compile(optimizer=RMSprop(learning_rate=0.001), loss='
    sparse_categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
n_epoch = 20
history = modelRMSprop.fit(x_train, y_train, validation_split = 0.1, batch_size=
    batch_size, epochs=n_epoch)

T = T.append(pd.DataFrame(history.history))

T.to_csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_cnn.csv")
tf.keras.utils.plot_model(modelAdam, to_file='/Users/benjamintenmann/Desktop/CompBio/
    Assignments/DL_1/MNIST/model_3.1.png', rankdir='LR', show_shapes=True)

val_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)
loss, acc = modelAdam.evaluate(val_dataset)
print("Adam:", loss, acc)
loss, acc = modelAdagrad.evaluate(val_dataset)
print("Adagrad:", loss, acc)
loss, acc = modelRMSprop.evaluate(val_dataset)
print("RMSprop:", loss, acc)

yhat = modelAdam.predict(x_test)
def f(yhat):
    if sum(yhat) != 0:
        r = np.where(yhat == 1)[0][0]
        return r
    else:
        pass
ls = list(map(f, np.round(yhat)))
l = []
```

```
for i, y in enumerate(np.round(yhat)):
    if sum(y) == 0:
        l.append(i)
def g(x):
    return x != None
f_ls = np.array(list(filter(g, ls)))
npL = np.array(l)
y_test_cut = np.array(y_test[np.in1d(np.arange(0, 10000), npL, invert=True)])
counts = []
for y in np.unique(y_test_cut):
    count = [0]*10
    for yhat in np.unique(f_ls):
        new = f_ls[y_test_cut == y]
        count[yhat] = len(new[new == yhat])
    counts.append(count)

n = np.array(counts)

f, ax = plt.subplots(figsize=(9, 6))

ax = sns.heatmap(n, annot=True, ax=ax, fmt='d', linewidths=.5)
ax.set(xlabel='yhat', ylabel='y')
fig = ax.get_figure()
fig.savefig('/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/heat_cnn.png'
    , dpi=400)
print(fig)

print('corr class:', np.sum(np.diagonal(n)), '\nunclass:', len(l), '\nincorr class:',
    (10000 - np.sum(np.diagonal(n))) - len(l))

sns.set_theme()
f, ax = plt.subplots(figsize=(9, 6))

ax = sns.barplot(x=['perceptron', 'perceptron', 'MLP', 'MLP', 'CNN', 'CNN'], y=[561,
    275, 162, 9, 106, 3], hue=['misclassified', 'unclassified']*3)
ax.set(ylabel='number of digits')
fig = ax.get_figure()
fig.savefig('/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/err_barplot.
    png', dpi=400)
print(fig)
```

A.6 Data Visualisation

```
library(tidyverse)
library(dplyr)
library(ggplot2)
library(plotrix)
library(colorspace)

data.frame(x = c(1, 1, -1, -1),
           y = c(1, -1, 1, -1),
           parity = c('+', '-', '-', '+')) %>%
  ggplot(., aes(x=x, y=y, color=parity))+
  geom_point(size = 4)+
  xlab(expression(paste('x'[1], sep='')))+
  ylab(expression(paste('x'[2], sep='')))+
  theme_minimal()+
  geom_hline(yintercept = 0)+
```

```
  geom_vline(xintercept = 0)

read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/eps_test.csv")%>%
  ggplot(., aes(epsilon, final_error))+
  geom_path(color = "steelblue")+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())

df1 <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/noalpha.csv
    ")
df2 <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/alpha.csv")

data.frame(error = c(df1$error, df2$error),
           epoch = rep(df1$epochs, 2),
           alpha = rep(c('yes', 'no'), each = length(df1$epochs)))%>%
  group_by(alpha)%>%
  ggplot(., aes(epoch, error, color=alpha))+
  geom_path()+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())


df <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/alpha_test.
    csv")
df$alpha <- as.factor(rep(seq(0.01, 1, 0.1), each = 500))

pal = sequential_hcl(10, palette = 'Plasma')
df %>%
  group_by(alpha) %>%
  ggplot(., aes(x=epochs, y=errors, color=alpha))+
  geom_path()+
  theme_minimal()+
  scale_color_manual(values = pal)+
  theme(axis.line = element_line(), axis.ticks = element_line())

library(data.table)
data <- fread("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MLP/epsilon_test.
    csv", sep = ",")

data$epsilon <- as.factor(rep(seq(0.1, 5.09, 0.01), each = 500))
pal2 <- sequential_hcl(length(seq(0.1, 5.09, 1)), palette = "Plasma")

data %>%
  filter(epsilon %in% as.factor(seq(0.1, 5.09, 1))) %>%
  group_by(epsilon)%>%
  ggplot(., aes(x=epochs, y=errors, color=epsilon))+
  geom_path()+
  theme_minimal()+
  scale_color_manual(values = pal2)+
  theme(axis.line = element_line(), axis.ticks = element_line())


dat <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/perceptron/perc
    _out.csv")
data.frame(error = c(dat$err_train, dat$err_test),
           epoch = rep(seq(1,200,1), 2),
           type = rep(c('train', 'test'), each=200))%>%
  group_by(type)%>%
  ggplot(., aes(epoch, error, color=type))+
```

```
  geom_path()+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())




dat2 <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/perceptron/perc_out.csv")
data.frame(error = c(dat2$err_train, dat2$err_test, dat$err_train, dat$err_test),
           epoch = rep(seq(1,200,1), 4),
           type = rep(c('train_par', 'test_par', 'train', 'test'), each=200))%>%
  group_by(type)%>%
  ggplot(., aes(epoch, error, color=type))+
  geom_path()+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())


dada <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_2.csv")
data.frame(data = c(dada$loss, dada$acc, dada$val_loss, dada$val_acc),
           type_A = rep(c('train', 'validation'), each=40),
           type_B = rep(rep(c('loss', 'acc'), each = 20), 2),
           ints = rep(rep(c(0.2625, 0.9287), each=20), 2),
           epoch = rep(seq(1,20,1), 4))%>%
  group_by(type_B)%>%
  ggplot(., aes(epoch, data, color=type_A))+
  geom_path()+
  theme_minimal()+
  scale_color_discrete(name='type')+
  theme(axis.line = element_line(), axis.ticks = element_line(), axis.title.y = element_blank())+
  facet_grid(type_B~.)+
  geom_hline(aes(yintercept=ints), linetype='dashed', color='black', alpha=0.6)



dad <- read.csv("/Users/benjamintenmann/Desktop/CompBio/Assignments/DL_1/MNIST/hist_cnn.csv")
data.frame(data = c(dad$val_loss, dad$val_accuracy),
           type_B = rep(c('loss', 'acc'), each = 60),
           typ = rep(rep(c('Adam', 'Adagrad', 'RMSprop'), each = 20), 2),
           epoch = rep(rep(seq(1,20,1), 3), 2))%>%
  group_by(type_B)%>%
  ggplot(., aes(x=epoch, y=data, color=typ))+
  geom_path()+
  scale_color_discrete(name='optimizer')+
  theme(axis.line = element_line(), axis.ticks = element_line(), axis.title.y = element_blank())+
  facet_grid(type_B~.)

data.frame(accuracy = c(0.9267, 0.8623, 0.9272, 0.9834, 0.9219, 0.9817, 0.9892, 0.9387, 0.9892),
           optimizer = rep(c('Adam', 'Adagrad', 'RMSprop'), 3),
           model = rep(c('perceptron', 'MLP', 'CNN'), each = 3))%>%
  group_by(model)%>%
  ggplot(., aes(optimizer, accuracy, fill=model))+
  geom_bar(stat='identity', position = 'dodge')+
```

```
  scale_fill_brewer(palette = 'Paired')+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())

data.frame(accuracy = c(0.9834, 0.9219, 0.9817),
           optimizer = c('Adam', 'Adagrad', 'RMSprop'))%>%
  ggplot(., aes(optimizer, accuracy, fill=optimizer))+
  geom_bar(stat='identity')+
  theme_minimal()+
  theme(axis.line = element_line(), axis.ticks = element_line())

data.frame(data = c(561, 275, 162, 9, 106, 3),
           type_a = rep(c('misclassified', 'unclassified'), 3),
           type_b = rep(c('perceptron', 'MLP', 'CNN'), each = 2))%>%
  group_by(type_b)%>%
  ggplot(., aes(type_b, data, fill=type_a))+
  geom_bar(stat='identity', position = 'dodge')+
  theme_minimal()+
  scale_fill_brewer(palette = 'Paired', name='error type')+
  xlab('model')+
  ylab('number of digits')+
  theme(axis.line = element_line(), axis.ticks = element_line())
```