Introduction to Artificial Intelligence & Data Science

Exercise 3

University of Applied Sciences
Computer Science and Digital Communication

December 10, 2025

Tasks

1. install and establish python environment including the required packages
2. use the provided python script **handwritten-ocr-cnn.py** to run all subsequent experiments
3. study the programm code
4. optimize network topology and training the generalisation capabilities and accuracy
   number of convolutional layers, neurons/planes of the feature and
   the training to maximize the generalisation by means of dropout layers, learning rate schedules
5. for each of the above means applied: analyse the loss function over both training and validation data and justify their application
6. optimize the training process by continuing from one task to the next by using the best method respectively its parameter
7. after optimizing the training process and the performance of the models by means of the above tasks analyse confusion matrix and accuracy report
8. write a report on the experimental results and findings for each of below tasks

Install and activate virtual environment

Unix/MaxOS

```
# install environment, ensure python 3.10 version
python3 -m venv .venv

# activate environment
source .venv/bin/activate
```

Windows

```
# install environment, ensure python 3.10 version
py -m venv .venv

# activate environment
.venv\Scripts\activate
```

Note: *.env* denotes the name of the directory in which the environment is installed to. Repeating the install instruction overwrites/deletes all previously installed python packages and configurations

Readings: Install packages in a virtual environment using pip and venv

Install required python packages in virtual environment

```
# install all required packages
pip install -r Requirements.txt

#or install all required packages manually

pip install tensorflow
pip install keras
pip install matplotlib
pip install numpy
pip install scikit-learn
```

Unix

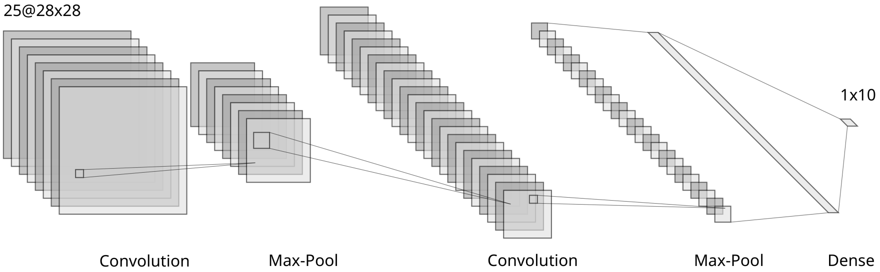
```
pip install pyqt5
```

Handwritten optical character recognition with CNN topology

Task - recognize handwritten digits 0-9

Reference: MNIST hand-written digits dataset

Handwritten optical character recognition with CNN topology

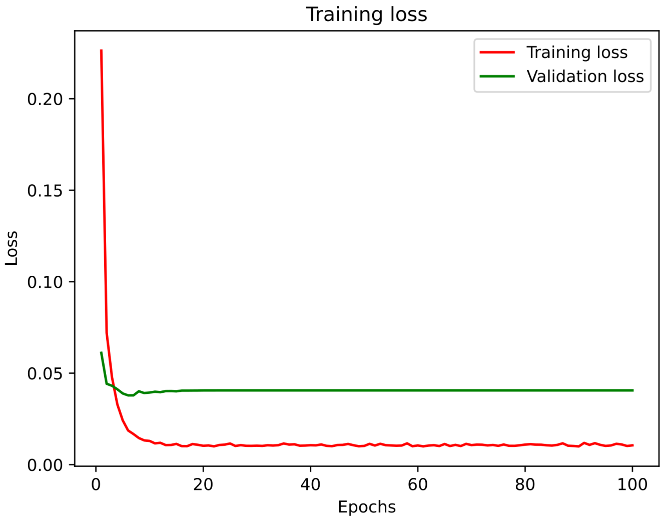


Handwritten optical character recognition - example configuration

Configuration:

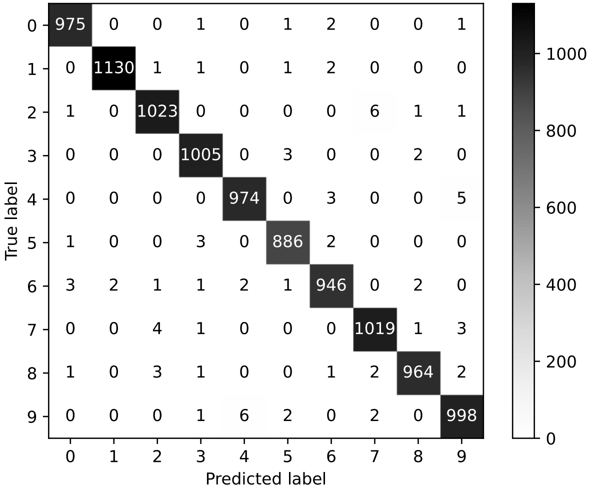
- topology:
  - 3 pairs of convolutional and pooling feature extraction layers
  - 2 feedforward full-connected classification layers (activation: softmax, loss function: cross-entropy)
- training:
  - stochastic-gradient-descent (SGD) with adaptive momentum optimization (ADAM)
- learning rate schedule - exponential decay

Handwritten optical character recognition - mnist data set - loss function



Readings: MNIST database of handwritten digits

Handwritten optical character recognition - mnist data set - confusion matrix



Handwritten optical character recognition - mnist data set - accuracy report

	precision	recall	f1-score	support
0	0.9939	0.9949	0.9944	980
1	0.9982	0.9956	0.9969	1135
2	0.9913	0.9913	0.9913	1032
3	0.9911	0.9950	0.9931	1010
4	0.9919	0.9919	0.9919	982
5	0.9911	0.9933	0.9922	892
6	0.9895	0.9875	0.9885	958
7	0.9903	0.9912	0.9908	1028
8	0.9938	0.9897	0.9918	974
9	0.9881	0.9891	0.9886	1009
accuracy			0.9920	10000
macro avg	0.9919	0.9919	0.9919	10000
weighted avg	0.9920	0.9920	0.9920	10000

Task 1: Topology

1. reflect on the function of the convolution and pooling layers for 2D classification
2. enhance the feature extraction by a 3rd pair of convolution and pooling layer
3. vary the number of planes/feature maps between [10, 100] (evaluate min. 4 values)
4. change the name of the model in the *model_name* variable to identify the output files for model loss function, accuracy, and classification report
5. analyse the loss function for both training and validation data for the different configurations
6. note observations in the report together with the figures of the loss functions

Reference: https://keras.io/2.15/api/layers/

Task 1: Topology cont.

```
model_name = 'CNN_Handwritten_OCR_CNN'+str(n_cnn1planes)+ ...'
...
model = Sequential()

cnn1 = Conv2D( ... )
model.add(cnn1)
model.add(MaxPool2D( ... ))

cnn2 = Conv2D( ... )
model.add(cnn2)
model.add(MaxPool2D( ... ))

cnn3 = Conv2D( ... )
model.add(cnn3)
model.add(MaxPool2D( ... ))
```

Reference: https://keras.io/2.15/api/layers/

Task 2: Learning Rate

1. reflect on the function of the learning rate as parameter for optimization algorithms
2. use the Stochastic-Gradient-Descent (SGD) algorithm as optimizer
3. vary the learning rate between [0.001,0.01] (evaluate min. 4 values)
4. change the name of the model in the *model_name* variable to identify the output files for model loss function, accuracy, and classification report
5. analyse the loss function for both training and validation data for each learning rate
6. note observations in the report together with the figures of the loss functions

```
learning_rate=0.001
optimizer = SGD(learning_rate=learning_rate)
model.compile(loss='categorical_crossentropy', metrics=['accura
```

Reference: keras.io/2.15/api/optimizers/learning_rate_schedules/

Task 3: Learning Rate Schedules

1. reflect on the function of the learning rate as parameter for optimization algorithms and how alternative learning rate schedules effect the optimization
2. use the Stochastic-Gradient-Descent (SGD) algorithm as optimizer
3. replace the original constant learning rate by a learning rate schedule
4. change the name of the model in the *model_name* variable to identify the output files for model loss function, accuracy, and classification report
5. analyse the loss function for both training and validation data for each learning rate
6. extend the experimental scope by varying the initial learning rate between [0.001,0.1] and the type of schedule (evaluate min. 4 values)
7. note observations in the report together with the figures of the loss functions

```
learning_rate = ExponentialDecay(initial_learning_rate=1e-2,
                                 decay_steps=n_epochs,
                                 decay_rate=0.9)
optimizer = SGD(learning_rate=learning_rate)
```

Reference: keras.io/2.15/api/optimizers/learning_rate_schedules/

Task 4: Optimizers

1. reflect on the function of the momentum term if applied to the Stochastic-Gradient-Descent (SGD) algorithm
2. enhance the Stochastic-Gradient-Descent (SGD) algorithm with the momentum term
3. change the name of the model in the *model_name* variable to identify the output files for model loss function, accuracy, and classification report
4. vary the learning rate between [0.001,0.01] (evaluate min. 4 values)
5. analyse the loss function for both training and validation data for each learning rate
6. note observations in the report together with the figures of the loss functions

```
learning_rate=0.001
momentum=0.9
sgd = SGD(learning_rate=learning_rate, momentum=momentum)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
```

Reference: keras.io/2.15/api/optimizers/

Task 5: Dropout Layer (Regularization)

1. reflect on the effects of introducing dropout layers particularly between convolutional layers
2. insert a dropout layer at appropriate position between instructions that compose the current topology
3. vary the dropout rate between [0.2,0.5] for all introduced dropout layers (evaluate min. 4 values)
4. change the name of the model in the *model_name* variable to identify the output files for model loss function, accuracy, and classification report
5. analyse the loss function for both training and validation data for each dropout layer insertion and dropout rate
6. note observations in the report together with the figures of the loss functions

```
model = Sequential()
cnn1 = Conv2D( ... )
model.add(cnn1)


...
model.compile(...)
```

Task 6: Final accuracy evaluation

1. evaluate the accuracy report and the confusion matrix of the test data classified using the best model

Background

Model generalisation - between underfitting and overfitting

machine learning algorithms train a model based on a finite set of training data

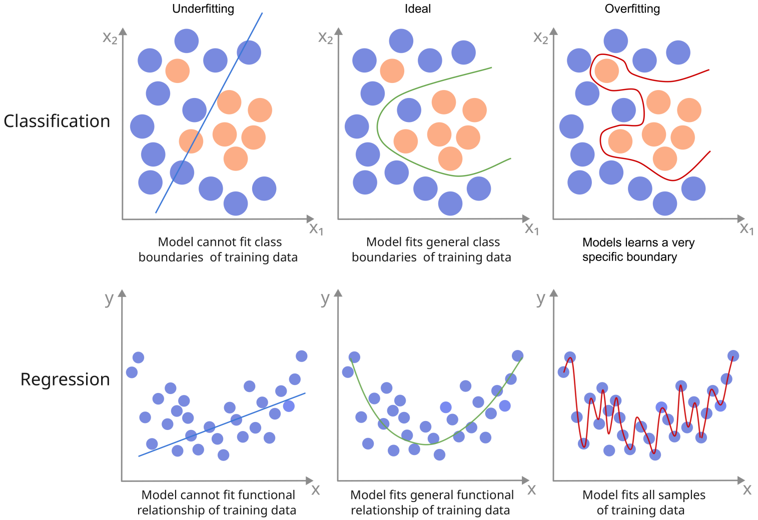
training considers the model is evaluated based on how well it predicts the observations contained in the training data

goal to obtain model that generalizes beyond training data
$\rightarrow$ predicts previously unseen observations

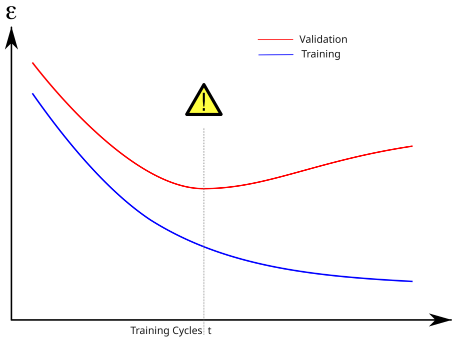
underfitting - model insufficient to fit the training data, i.e. to capture the relationship between input and output

overfitting - model fits the training data well while exhibiting larger generalization error of unseen data

Model generalisation - between underfitting and overfitting



Overfitting



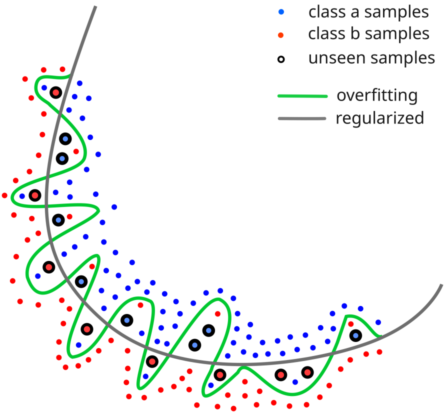
error ε of both training and validation data as a function of the number of training cycles

validation error increases (positive slope) while the training error steadily decreases (negative slope) $\rightarrow$ situation of overfitting likely occurred

validation error at minimum $\rightarrow$ best predictive and fitted model

Readings: Overfitting

Overfitting

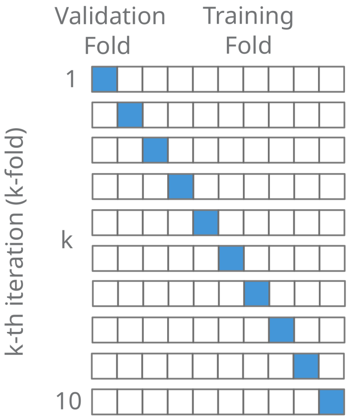


Reference: Overfitting

Overfitting - diminishing its causes

- model comparison
- cross-validation
- regularization
- early stopping
- pruning
- Bayesian priors
- dropout

Overfitting - k-fold cross validation



- split data into a number of k chunks one chunk as data set for validation the remaining $k - 1$ chunks are merged into data set for training
- rotate the chunk for validation
- analyse the error metrics by averaging their values for each rotation step

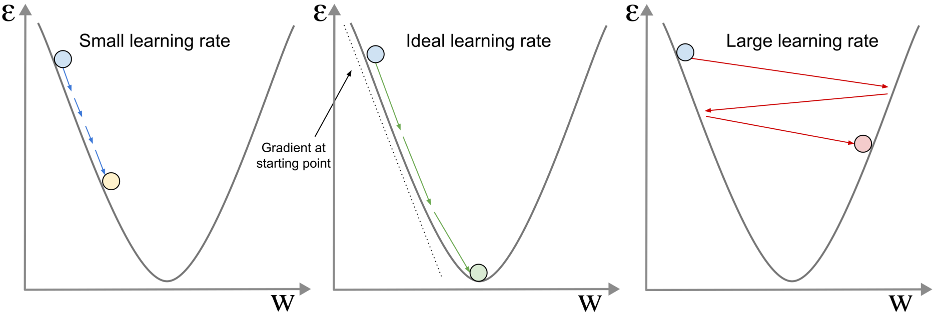
Adaptation of hyper-parameter

up to the user to structure and optimize the topology and the means of the training process until it gives desirable outputs

tune topology, number of layers, number of neurons, activation functions, optimisers, learning rate, regularization

search for optimal setup - like grid search - very expensive

Adaptation of hyper-parameter - learning rate

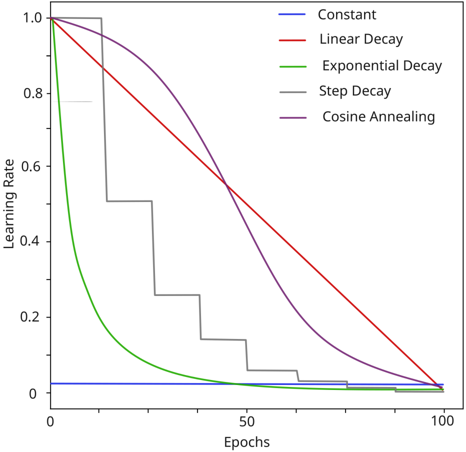


Adaptation of hyper-parameter - learning rate

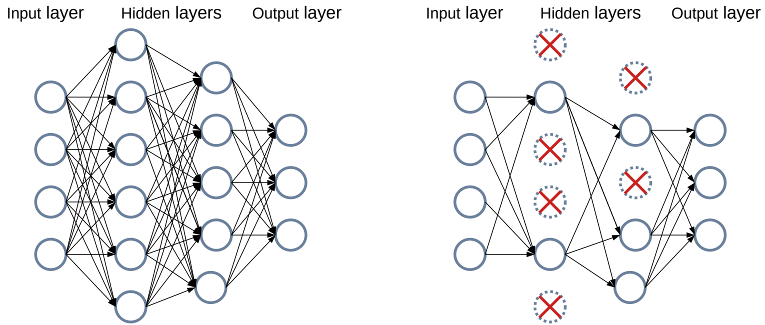
conventional learning rate - fixed

make gradient descent adaptive, more efficient

- learning rate decay reduces the learning rate by a factor each epoch



Regularisation through dropout



only applied during the training - during inference, all neurons are active

dropout rate: fraction of neurons to drop out, common rates range [0.2, 0.5], optimal rate varies depending on the dataset and architecture

convolutional neural networks: dropout of entire planes/feature maps instead of individual neurons

Reference: Dropout: a simple way to prevent neural networks from overfitting

Stochastic-Gradient-Descent - Momentum

$$\Delta w_{ij} = \eta \frac{\delta E}{\delta w_{ij}}$$

η - the learning rate

Δw_{ij} - the update of parameters w_{ij}

$$\Delta w_{ij}^{t} = \eta \frac{\delta E}{\delta w_{ij}} + \lambda \Delta w_{ij}^{t-1}$$

λ - the exponential decay factor [0,1] denoting the relative contribution of the current gradient and earlier gradients to the change of parameters

Δw_{ij}^{t-1} the update of parameters $w_i j$ obtained during the previous iteration t

effects: preventing oscillations, local minimum

Readings: Stochastic Gradient Descent

Stochastic-Gradient-Descent - Momentum