

Deep learning

- Deep learning
 - Data laden
 - load_data
 - csv
 - zip
 - Data opsplitsen
 - handmatig
 - trainsplit
 - Shape bekijken
 - Normalisatie
 - Model maken
 - Model summary
 - Compile model
 - Model Summary
 - Model trainen
 - Model evalueren
 - Predict
 - Get predictions

Data laden

load_data

```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.mnist.load_data()
```

CSV

```
white_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"
df = pd.read_csv(white_url, sep=";")
```

zip

```
import tensorflow as tf
tf.keras.utils.get_file("wine.zip",
"https://archive.ics.uci.edu/static/public/186/wine+quality.zip", extract=True)

file_path = "/root/.keras/datasets/wine_extracted/winequality-white.csv"
wine_df = pd.read_csv(file_path, delimiter=";")
wine_raw_data = wine_df.values
```

Data opsplitsen

handmatig

```
X_train = X_train_full[:-10000]
y_train = y_train_full[:-10000]
X_val = X_train_full[-10000:]
y_val = y_train_full[-10000:]
```

trainsplit

```
X_train = X_train_full[:-10000]
y_train = y_train_full[:-10000]
X_val = X_train_full[-10000:]
y_val = y_train_full[-10000:]
```

Shape bekijken

```
print(X_train.shape)
print(y_train.shape)
print(X_val.shape)
print(y_val.shape)
print(X_test.shape)
print(y_test.shape)
```

Normalisatie

Model maken

```
def get_model():
    model = keras.Sequential()
    model.add(keras.Input(shape=(4,), name='input')) # 4 = aantal features
    model.add(keras.layers.Dense(5, activation="relu", name='hidden')) # 5 = aantal hidden (gegeven)
    model.add(keras.layers.Dense(3, activation="softmax", name='output')) # 3 aantal output (in dit geval 3 klassen)
    return model

def get_model():
```

```

model = keras.Sequential()
model.add(keras.Input(shape=(28,28,)))
model.add(keras.layers.Dense(300,activation="relu"))
model.add(keras.layers.Dense(100,activation="relu"))
model.add(keras.layers.Dense(10,activation="softmax")) # classificatie

return model

def get_model():
    model = keras.Sequential()
    model.add(keras.Input(shape=(10000,)))
    model.add(keras.layers.Dense(16,activation="relu"))
    model.add(keras.layers.Dense(16,activation="relu"))
    model.add(keras.layers.Dense(10,activation="sigmoid")) # binairy classificatie = sigmoid
    return model

def get_model(norm_layer):
    model = keras.Sequential()
    model.add(keras.Input(shape=(11,)))
    model.add(norm_layer)
    model.add(keras.layers.Dense(50,activation="relu"))
    model.add(keras.layers.Dense(50,activation="relu"))
    model.add(keras.layers.Dense(50,activation="relu"))
    model.add(keras.layers.Dense(1)) # regressie
    return model

```

Model summary

```

model = get_model()
model.summary()
# Je zou dit zelf moeten kunnen opstellen. Dit is wss een examenvraag

```

Compile model

```

optimizer = keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
['accuracy'])

# examenvraag: welke loss functie ga je gebruiken?
# binairy classificatie = loss='binary_crossentropy' EN classificatie = accuracy

```

Model Summary

```
model = get_model()
model.summary()
```

Model trainen

```
history = model.fit(X_train, y_train, batch_size=32, epochs=1, callbacks=[early_stopping_cb], validation_data=(X_val, y_val))
```

Model evalueren

```
model.evaluate(X_test, y_test)
```

Predict

```
model.predict(X_test[0:10])
```

Get predictions

```
def get_predictions(model, X, keepdims=False):
    predicted = model.predict(X)
    return keras.ops.argmax(predicted, axis=1, keepdims=keepdims)
```