## Einführung in TensorFlow

TensorFlow ist eine Open-Source-Bibliothek für maschinelles Lernen und Deep Learning, die von Google entwickelt wurde. Sie ermöglicht es, neuronale Netze zu erstellen, zu trainieren und für verschiedene Aufgaben wie Bilderkennung, natürliche Sprachverarbeitung oder Zeitreihenanalyse einzusetzen.

TensorFlow bietet eine hohe Flexibilität und wird häufig mit Keras verwendet, einer API, die das Erstellen von neuronalen Netzen vereinfacht.

## Installation von TensorFlow

Falls TensorFlow noch nicht installiert ist, kann es mit folgendem Befehl installiert werden:

```
pip install tensorflow
```

Danach kann TensorFlow importiert werden:

```
import tensorflow as tf
from tensorflow import keras
```

## Ein einfaches neuronales Netz mit TensorFlow

Wir werden ein neuronales Netz trainieren, um handgeschriebene Ziffern aus dem MNIST-Datensatz zu klassifizieren.

Der MNIST-Datensatz besteht aus 70.000 Bildern von handgeschriebenen Ziffern (0–9), die in Trainings- und Testdaten aufgeteilt sind. Jedes Bild hat eine Größe von 28x28 Pixeln.

Zunächst laden wir die Daten:

```
import numpy as np
import matplotlib.pyplot as plt

# MNIST-Datensatz laden
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
# Beispielhafte Darstellung einer Ziffer
plt.imshow(x_train[0], cmap="gray")
plt.show()
```

Damit das neuronale Netz effizient trainieren kann, normalisieren wir die Daten. Normalisierung bedeutet, dass die Pixelwerte (die ursprünglich zwischen 0 und 255 liegen) auf den Bereich 0 bis 1 skaliert werden:

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

## Aufbau eines neuronalen Netzes

Ein neuronales Netz besteht aus mehreren **Schichten (Layers)**.

Wir definieren unser Modell mit folgenden Komponenten:

1. **Flatten Layer**: Wandelt das 28x28-Bild in einen eindimensionalen Vektor um.
2. **Dense Layer (128 Neuronen, ReLU-Aktivierung)**: Eine vollständig verbundene Schicht mit 128 Neuronen.
3. **Dropout Layer (20%)**: Hilft, Overfitting zu reduzieren.
4. **Dense Layer (10 Neuronen, Softmax-Aktivierung)**: Die Ausgabeschicht, die Wahrscheinlichkeiten für jede der 10 Ziffern ausgibt.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),   # Wandelt 28x28 Bild in V
    keras.layers.Dense(128, activation='relu'),   # Versteckte Schicht mit
    keras.layers.Dropout(0.2),                    # 20% der Neuronen werden
    keras.layers.Dense(10, activation='softmax')  # Ausgabe mit 10 Klassen
])
```

**Warum Dropout?**
Dropout deaktiviert zufällig einen Teil der Neuronen während des Trainings. Dadurch wird verhindert, dass das Modell zu stark an die Trainingsdaten angepasst wird (Overfitting).

## Kompilieren und Trainieren des Modells

Bevor wir das Modell trainieren, müssen wir es kompilieren. Dabei legen wir fest:

- **Optimizer**: `adam` – Eine moderne Optimierungsmethode, die den Lernprozess effizient gestaltet.
- **Loss Function**: `sparse_categorical_crossentropy` – Eine Verlustfunktion für Klassifikationsprobleme mit ganzen Zahlen als Labels.
- **Metrik**: `accuracy` – Wir messen die Genauigkeit des Modells.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Nun trainieren wir das Modell für 5 Epochen (Durchläufe über die Trainingsdaten):

```
model.fit(x_train, y_train, epochs=5)
```

## Evaluierung des Modells

Nach dem Training testen wir das Modell mit den Testdaten:

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Genauigkeit auf Testdaten: {test_acc:.4f}')
```

Die Genauigkeit ( `test_acc` ) gibt an, wie gut das Modell mit neuen Daten funktioniert.