DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Optimierung des MNIST Model von Tenserflow

Projektentwurf - Modul T3ELF3804.4

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik, Schwerpunkt Embeeded IT

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Selina Domes

Abgabedatum:	5. Januar 2024
Bearbeitungszeitraum:	
Kurs:	TFE21-2
Gutachter:	Mark Schutera

1. Einleitung

Im Rahmen des Projektentwurfs soll das Wissen im Bereich Deep Learning vertieft werden. Dazu wird ein Modell mit Hilfe von Tensorflow am Datensatz MNIST trainiert. In der Vorlesung bei Herrn M. Schutera wurden bereits die Grundlagen vermittelt und erste Schritte bei der Programmierung eines Modells unternommen. Zu Beginn der selbständigen Arbeit wurde das Programm mit Hilfe von KI erstellt und erste Versuche zur Parameterbehandlung durchgeführt. Das generierte Modell hatten nie eine Genauigkeit von 99 Prozent, jedoch ist dies jedoch mit einer hohen Anzahl von Bildern . Daher ist die Frage für diese Arbeit: Ist es möglich, eine Genauigkeit von 99 Prozent zu erreichen, ohne dabei über eine hohe Rechenleistung zu verfügen?

2. Vorgehen

Um diese Frage zu beantworten, wurde zunächst der aktuelle Stand der Technik erfasst und nach Optimierungsmethoden gesucht, die eine hohe Genauigkeit versprechen. Die interessantesten Optimierungsmethoden sind die Definition der Hyperparameter, da bereits bei wenigen Trainingsrunden hohe Genauigkeiten erreicht werden können. Die andere Methode besteht darin, den Datensatz durch Pre-Modeling zu bewerten und nur mit den vielversprechendsten Bildern zu trainieren. Für diese Arbeit wurde entschieden, dass die erste Methode besser geeignet ist. Um die Hyperparameter zu definieren, müssen zuerst die Vergleichsparameter bestimmt werden. Die bestimmten Parameter sind: Genauigkeit und Loss-Werte beim Testdatensatz sowie das Lernkurvendiagramm. Um eine spätere Vergleichbarkeit der Ergebnisse zu gewährleisten.

Nach Festlegung der Vergleichsparameter musste der Modellaufbau bestimmt werden. Hierfür sind folgende Netzwerkmodelle möglich: Feedforward-Netzwerk (FFN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) und Long Short-Term Memory Network (LSTM). Jedes dieser Modelle hat verschiedene Vorteile, weshalb sie in einem Test verglichen werden sollten. Für diesen Zweck wurden mithilfe von Blackbox.ai die aktuell besten Architekturen zusammengetragen und in ein Programm integriert. Dabei stellte sich das Problem heraus, dass FNN und CNN mit einem zweidimensionalen Input-Layer arbeiten, während RNN und LSTM mit einem vierdimensionalen Input-Layer arbeiten. Deshalb müssten für den Test alle vier

Modelle mit verschiedenen Datensätzen getestet werden. Aus diesem Grund wurde beschlossen, nur die FNN- und CNN-Architekturen weiter zu testen. Für die nächsten Tests wurden alle möglichen vorkonfigurierten Optimizer und Loss-Funktionen in einem Array gespeichert, um die verschiedenen Kombinationen durch Schleifen zu prüfen. Und die Ergebnisse werden jeweils in einem Array gespeichert. Für die nächsten Tests wurden sämtliche vorkonfigurierten Optimierer und Verlustfunktionen in einem Array gespeichert, um die verschiedenen Kombinationen mittels Schleifen zu prüfen. Die Ergebnisse wurden jeweils in einem Array gespeichert. Zusätzlich dazu wurde eine dynamische Lernrate einführt: Die Fehlerrate sollte sich mit jedem Trainingslauf verbessern. Das wurde im Notebook Öptimierungsmethode 1"durchgeführt.

Jedoch ist bei Tests aufgefallen, dass extrem schlechte Loss- und Accuracy-Werte produziert wurden, obwohl bereits bei der Einführung bekannt war, dass man mit einfachen Architekturen hohe Accuracy-Werte erreichen kann. Aus diesem Grund wurde ein weiteres Notebook erstellt, welches sich von dem anderen in der Form der Trainings- und Testdaten unterscheidet. Daher wurden auch zwei andere Architekturen erzeugt und verwendet. Der Unterschied besteht darin, dass der Datensatz jetzt nur normalisiert und nicht linearisiert wurde.

3. Auswertung

Wie bereits erwähnt wurden bei ersten Versuch der Optimierung wurden maximale Accuracy von maximalen 20 Prozent erreicht. Was sich nach einigen Test auf das Shape des Datensatz zurück verfolgen lässt.
Im zweiten Notebook konnten viel besser Validierungs-Accurancy erreichen lassen. Hier liegt der Spitzenwert bei 99,2 Prozent beim Model zweiten Model mit Nadam, doch die Lernkurve weißt darauf hin, das ein Overfitting statt. Weshalb eine andere Kombination vorgezogen werden sollte oder die Lernrate wird manuell angepasst. Auch das erste Model erzeugt akzeptable Werte , wie mit Adagrad , welche die beste Lernkurve erzielt. Das gesamte Ergebnisse kann im Anhang A nachgeschaut werden

4. Fazit

Zusammenfassend lässt sich sagen, dass das Thema Deep Learning ein weitreichendes

Themengebiet ist. Eine einfachere Einführung in die Thematik bietet die Arbeit mit MNIST in Zusammenarbeit mit Tensorflow und Keras. Für eine tiefgehende Programmierung und Optimierung sind gute Python-Kenntnisse erforderlich. Ein weiteres Problem bei der Arbeit war die begrenzte Rechenleistung, wodurch das Training entweder sehr lange dauerte oder gar nicht erst begann. Eine Lösung dafür ist die automatische Zwischenspeicherung. Allerdings ist dies mit begrenzten Python-Kenntnissen nicht realisierbar.

Anhang A

In [ ]:
```
#Programmcode für den Projektentwurf für das Modul T3ELF3804.4
#Autor: Selina Domes
#template: https://github.com/schutera/DeepDive.git Zuletzt am gerufen am 02-01-2024
# und: https://michaelkipp.de/deeplearning/Optimierungsmethoden.html Zuletzt am geru
```

In [1]:
```
# Used libraries
import numpy as np
print('Numpy version:', np.__version__)
import matplotlib.pyplot as plt
import seaborn as sn
print('Seaborn version:', sn.__version__)
import pandas as pd
print('Pandas version:', pd.__version__)

import os
import time
import math

import tensorflow as tf
print('Tensorflow version:', tf.__version__)

from tensorflow.keras import datasets, layers, models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

```
Numpy version: 1.24.3
Seaborn version: 0.12.2
Pandas version: 2.0.3
WARNING:tensorflow:From C:\Users\sd021\anaconda3\Lib\site-packages\keras\src\losses.
py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use t
f.compat.v1.losses.sparse_softmax_cross_entropy instead.

Tensorflow version: 2.15.0
```

In [2]:
```
from tensorflow.keras.datasets import mnist

# Loading MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Printing the shape
print('x_train:', x_train.shape)
print('y_train:', y_train.shape)
print('x_test:', x_test.shape)
print('y_test:', y_test.shape)
```

```
x_train: (60000, 28, 28)
y_train: (60000,)
x_test: (10000, 28, 28)
y_test: (10000,)
```

In [3]:
```
# Normilize the data - Interval [0,1], lineariesieren
x_train = x_train/255.0
x_test = x_test/255.0

# Printing the shape
print('x_train:', x_train.shape)
print('x_test:', x_test.shape)
```