Distributed Tensorflow

Michael Zipperle
259564
Hochschule Furtwangen
Michael.Zipperle@hs-furtwangen.de

***Abstract*—Der Einsatz einer Künstliche Intelligenz (KI) hat in den letzten Jahren stark zugenommen. Einerseits werden die Technologien immer ausgereifter und Big-Player wie Amazon, Microsoft, Facebook und Google treiben die Forschung in diesem Bereich stark voran. Anderseits ermöglichen die Fortschritte in Hardware wie CPU und GPU, dass diese Technologien optimal eingesetzt werden können. Eine der Hauptherausforderungen stellt die Performance dar, da komplexe neuronale Netzen noch immer eine relativ hohe Trainingszeit von mehreren Stunden oder sogar Tagen benötigen. Dieser Artikel untersucht aktuelle Frameworks, mit denen eine KI umgesetzt werden kann. Dabei wird das Framework TensorFlow von Google im Detail betrachtet. Es wird aufgezeigt, wie durch eine verteilte Umgebung über mehrere Instanzen die Performance deutlich gesteigert werden kann. Dabei wird eine kleine Beispielspielanwendung Schritt für Schritt entwickelt, die die Konfiguration dieser Umgebung aufzeigt. Eine Evaluation bestätigt die Performancesteigerung bei der Nutzung einer verteilten TensorFlow Umgebung.**

I. Einführung

Eine KI wird heutzutage immer mehr eingesetzt, um die Interaktion zwischen Mensch und Maschine zu verbessern. Zusätzliche ermöglicht eine KI vollkommen neue Möglichkeiten in Bereichen wie Robotik, Daten-Analyse, Gesundheitswesen, Autonomes Fahren und vieles mehr. Doch was steckt hinter einer KI? Eine KI versucht durch maschinelles Lernen die menschliche Wahrnehmung und das menschliche Handeln durch Maschinen nachzubilden. Bei der KI-Forschung gibt es viele Verbindungen zur Neurologie und Psychologie, da menschliches Denken erforscht und verstanden werden muss, bevor Maschinen dieses nachahmen können. Bis heute ist es noch nicht annähernd gelungen. Die meisten KIs beschränken sich auf einen bestimmten Teilbereich und optimiert für einen bestimmten Anwendungsfall [1].

Bei einer KI kommt i.d.R. maschinelles Lernen zum Einsatz, wobei von erhobenen Daten gelernt wird, um daraus Entscheidungen zu treffen. Es gibt eine Reihe vordefinierter Algorithmen die an sogenannte Entscheidungsbäume erinnern. Diese Algorithmen sind jedoch nicht dynamisch genug, um mehrere Variablen verarbeiten zu können. Mit der Einführung von Deep Learning als Zweig des maschinellen Lernens ist dieses Problem größtenteils behoben. Durch Deep Learning kann ein kontinuierlicher Lernprozess umgesetzt werden, der sich stetig an neue Situationen anpasst. Dabei basiert Deep Learning auf der Analyse von Big Data und verarbeitet riesige Mengen an Daten aus Datenbank, Internetquellen und mehr. Es nutzt neuronale Netze, um die Denkweise eines Menschen möglichst gut nachzuahmen. Deep Learning wurde bis vor wenigen Jahren nur von wenigen Unternehmen genutzt, da der Einsatz viele Ressourcen benötigte und somit relativ teuer war. Durch den Fortschritt der Technik und der besseren Unterstützung der Graphics Processing Unit (GPU) hat sich Deep Learning durchgesetzt und kommt in heutigen Anwendungen wie autonomes Fahren, maschinelle Übersetzung, Spracherkennung zum Einsatz [2].

Aktuell stellt der Lernprozess in Echtzeit eine große Herausforderung dar, oftmals benötigt dieser mehrere Stunden bzw. Tage. Für einige Anwendungsfälle dauert dies zu lange, da ein Lernprozess nahezu in Echtzeit benötigt wird. Es gibt bereits zahlreiche Deep Learning Frameworks, welche in diesem Arikel aufgezeigt werden. Sehr aktuell ist das Framework TensorFlow von Google. Dies wird im Laufe dieses Artikels genauer untersucht und es wird anhand eines Tutorials die Einrichtung einer verteilten TensorFlow Umgebung aufgezeigt. Durch diese verteilte Umgebung kann die Performance des Lernprozesses gesteigert werden.

II. Verwandte Arbeit

Bevor auf TensorFlow genauer eingegangen wird, gibt dieses Kapitel eine Übersicht über folgende Deep Learning Frameworks:

- Torch
- Caffe
- Caffe2
- Keras
- Chainer
- CNTK
- Apache MXNet
- Amazon DSSTNE
- Eclipse Deeplearning4j

A. Torch

Torch wurde ursprünglich von Facebook entwickelt und 2017 als Open-Source Projekt veröffentlicht. Das Framework bietet zwei Hauptfunktionen: Erstens, mathematische Berechnung mit starker Unterstützung der GPU. Dabei wird es oft als Ersatz für das bekannte Python Framework Numpy eingesetzt, da es durch die GPU-Unterstützung eine wesentlich bessere Performance bietet. Zweitens, zur Bildung von neuronalen Netzen für Deep Learning mit maximaler Flexibilität und Geschwindigkeit [4]. Laut DL4J zeichnet sich das Framework durch viele Modulare Funktionen aus, die sich einfach kombinieren lassen. Des Weiteren lassen sich neue Layer einfach definieren und auf der GPU ausführen. Nachteilig

ist, dass Torch keinen kommerziellen Einsatz bietet und die Dokumentation nicht vollständig sein soll [3].

B. Caffe

Caffe ist ein Open-Source Projekt, das von Yangqing Jia im Rahmen seiner Doktorarbeit beim Berkley AI Research (BAIR) initiiert wurde. Aktuell wird es vom BAIR und Community Entwicklern weiterentwickelt. Mit nur wenigen Codezeilen lassen sich Modelle erstellen. Dabei lässt sich konfigurieren, ob die Berechnungen auf der CPU oder GPU durchgeführt werden. Caffe wird hauptsächlich zur Verarbeitung von Bildern eingesetzt und kann mit nur einer GPU über 60 Millionen Bilder pro Tag verarbeiten [5]. Aktuell bietet Caffe noch keine Möglichkeit, die Berechnung verteilt auf mehreren Instanzen durchzuführen [3].

C. Caffe2

Der Erfinder von Caffe Yangqing Jia ist jetzt bei Facebook und arbeitet an einer Erweiterung von Caffe. Diese wurde unter dem Name Caffe2 veröffentlicht und bietet mehr Skalierbarkeit und Leichtgewichtigkeit gegenüber Caffe. Zudem erlaubt Caffe2 das Verteilen von Aufgaben bzw. Berechnung auf mehrere Instanzen [6].

D. Keras

Keras ist ein High-Level Application Programming Interface (API) für neuronale Netze und basiert auf TensorFlow, welches in Kapitel III genauer erläutert wird. Die Entwicklung von Keras folgte dem Motto: ”Die Fähigkeit, mit möglichst wenig Verzögerung von der Idee zum Ergebnis zu kommen, ist der Schlüssel für eine gute Forschung” [7]. Somit ermöglicht Keras das schnelle aufsetzen und testen von neuronalen Netzen. Elephans ist eine Erweiterung für Keras und erlaubt das Verteilen einer Anwendung über mehrere Instanzen [8].

E. Chainer

Chainer ist auch ein Open-Source Deep Learning Framework und bietet ein flexibles, intuitives und leistungsstarkes Mittel zur Implementierung einer ganzen Reihe von Deep-Learning-Modellen, einschließlich State-of-the-Art-Modellen wie z. B. rekurrenten neuronalen Netzen und variationalen Autoencodern. Mit Chainer können Anwendungen auf mehrere Instanzen verteilt werden und dadurch wie in Abbildung 1 zu sehen ist die Performance anderen Frameworks klar überlegen. Die optimale Performance wurde beim Verteilen einer Anwendung auf 128 GPUs erreicht [9].

F. Microsoft Cognitive Toolkit (CNTK)

Microsoft Cognitive Toolkit (CNTK) ist ein Open-Source Framework für das verteilen einer Deep-Learning Anwendung im kommerziellen Bereich. Es beschreibt neuronale Netze als eine Reihe von Rechenschritten über einen gerichteten Graphen. CNTK erlaubt dem Benutzer, populäre Modelltypen wie feed-forward DNNs, konvolutionelle neurale Netze (CNNs) und wiederkehrende neurale Netze (RNNs / LSTMs)

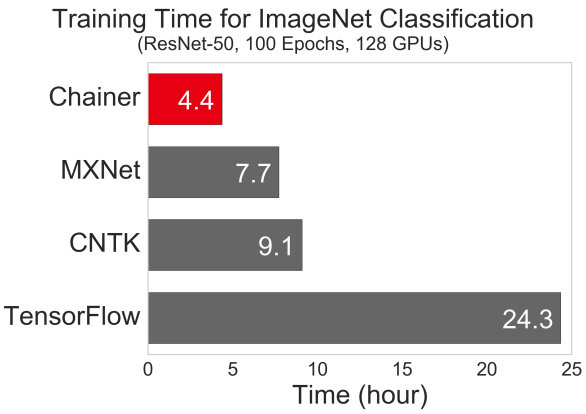

Fig. 1: Deep Learning Framework Vergleich [9]

leicht zu verwirklichen und zu kombinieren. CNTK implementiert stochastisches Gradientenabstiegsverfahren (SGD, Error Backpropagation) mit automatischer Differenzierung und Parallelisierung über mehrere GPUs und Server hinweg [10].

G. Apache MXNet

Apache MXNet ist ein modernes Open-Source Deep Learning Framework, mit dem neuronale Netze trainiert werden können. Es ist skalierbar, ermöglicht schnelles Modelltraining und unterstützt ein flexibles Programmiermodell und mehrere Programmiersprachen. Die MXNet-Bibliothek ist portabel und kann auf mehreren GPUs und mehreren Instanzen skaliert werden. MXNet wird von den wichtigsten Public Cloud-Anbietern wie Amazon Web Services (AWS) und Azure unterstützt [11]. Amazon und Microsoft arbeiteten zusammen an einer API für Apache MXNet und veröffentlichten im Oktober 2017 Gluon, eine neue Open-Source Deep-Learning Schnittstelle, die es Entwicklern ermöglicht, einfach und schnell Machine-Learning-Modelle zu erstellen, ohne dabei die Performance zu beeinträchtigen [13].

H. Amazon DSSTNE

Amazon Deep Scalable Sparse Tensor Network Engine (DSSTNE) ist eine Open-Source Deep Learning Framework zum entwickeln von Empfehlungsmodellen. DSSTNE wurde bei Amazon eingesetzt, um personalisierte Produktempfehlungen für ihre Kunden zu erstellen. Es ist für den produktiven Einsatz von realen Anwendungen ausgelegt, bei denen Geschwindigkeit und Skalierbarkeit gegenüber experimenteller Flexibilität im Vordergrund stehen. Training und Vorhersagen werden skaliert, wobei Berechnung und Speicherung für jede Arbeitsschicht modellparallel verteilt werden [14].

I. Eclipse Deeplearning4j

Eclipse Deeplearning4j (DL4J) ist die erste kommerzielle, Open-Source, Deep-Learning-Bibliothek für Java und Scala. Durch die Integration mit Hadoop und Apache Spark bringt DL4J KI vor allem zu Geschäftsumgebungen, bei der Hadoop

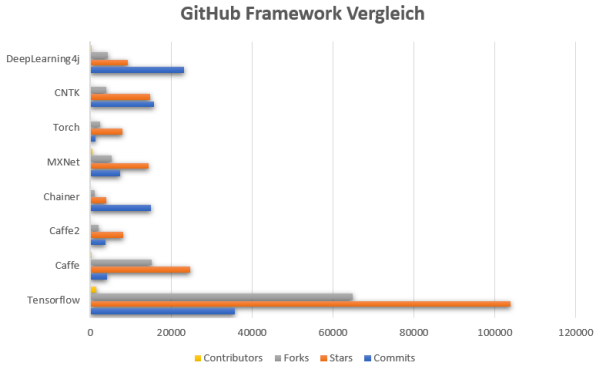

Fig. 2: GitHub: Vergleich des Interesses an o.g. Deep Learning Frameworks

bereits im Einsatz ist. DL4J zielt darauf ab, Plug-and-Play auf dem neuesten Stand der Technik zu sein, mehr Konvention als Konfiguration. Dies ermöglicht ein schnelles Prototyping für Data Scientists, Machine-Learning-Praktiker und Softwareentwickler. DL4J ist im Maßstab anpassbar. Unter der Apache 2.0-Lizenz veröffentlicht, gehören alle Derivate von DL4J ihren Autoren. DL4J kann Modelle aus Frameworks wie TensorFlow, Caffe und Theano importieren [15].

Wie zu sehen, gibt es aktuell viele Deep Learning Frameworks. Die meisten dieser Framework sind Open-Source Projekte und auf GitHub veröffentlicht. Heutzutage ist es wichtig, dass sich diese Technologie schnell weiterentwickelt. Dies erfordert eine große Gemeinschaft an Entwicklern. Abbildung 2 zeigt das Interesse dieser Gemeinschaft an den verschiedenen Deep Learning Frameworks vom 1. Juli 2018. Das Framework TensorFlow besitzt besonders viel Interesse, deshalb wird dieses Framework im nächsten Kapitel genauer untersucht.

III. TensorFlow

TensorFlow ist eine Open-Source-Softwarebibliothek für die numerische Hochleistungsberechnung. Die flexible Architektur ermöglicht die einfache Implementierung von Berechnungen auf einer Vielzahl von Plattformen (Central Processing Unit (CPU)s, GPUs, TPUs) und von Desktops über Cluster von Servern bis hin zu mobilen und Edge-Geräten. Ursprünglich von Forschern und Ingenieuren des Google Brain-Teams in der KI-Organisation von Google entwickelt, bietet es eine starke Unterstützung für maschinelles Lernen und Deep Learning. Der Kern der flexiblen numerischen Berechnung wird in vielen anderen wissenschaftlichen Bereichen eingesetzt [16].

A. Installation

TensorFlow kann auf den folgenden Plattformversionen und höher installiert werden:

- macOS 10.12.6
- Ubuntu 16.04
- Windows 7

Standardmäßig werden Anwendungen für TensorFlow in der Programmiersprache Python geschrieben, es lassen sich jedoch Entwicklungsumgebungen für die Programmiersprachen C++, Go, Swift und Java einrichten.

B. Unterstützung von mobilen Plattformen

Google stellt verschiedene Versionen ihres Frameworks bereit. Unter anderem auch Versionen für mobile Endgeräte, die für diese optimiert sind und mit weniger zur Verfügung stehenden Ressourcen auskommen. Im folgenden werden die verschiedenen Versionen kurz vorgestellt.

1) TensorFlow Mobil: TensorFlow Mobil wurde für den Einsatz auf einem mobilen Gerät optimiert und enthält alle Funktionen die das Framework zu bieten hat. Jedoch bietet TensorFlow Mobil nicht die optimale Performance für mobile Geräte. Deshalb wurde TensorFlow Light entwickelt, welches sich aktuell noch im Entwicklungsstatus befindet. Dabei werden von TensorFlow Light nur eine begrenzte Anzahl an Funktionen unterstützt, sodass die binäre Größe kleiner ist und damit ein Performance-Steigerung erzielt werden kann. Ob TensorFlow Mobil oder Light für eine Anwendung eingesetzt wird, hängt von deren Anforderungen ab [16].

2) TensorFlow Light: TensorFlow Lite ist eine Lösung für mobile und eingebettete Geräte wie beispielsweise Smartphones oder Smartwatches. Es ermöglicht maschinelles Lernen auf dem Gerät mit geringer Wartezeit und einer kleinen binären Größe. TensorFlow Lite unterstützt auch Hardware-Beschleunigung mit der Android Neural Networks API. TensorFlow Lite verwendet viele Techniken, um niedrige Latenzzeiten zu erreichen, wie z. B. die Optimierung der Kernel für mobile Apps, vorfusionierte Aktivierungen und quantisierte Kernel, die kleinere und schnellere Modelle ermöglichen [16]. Gerade der Einsatz von einem Deep Learning Framework direkt auf einem mobilen Gerät ermöglicht viele neue Anwendungsfälle und erlaubt auch den Einsatz des Frameworks, wenn das mobile Gerät keine Internetverbindung hat. Meist reicht es auch aus, wenn auf dem mobilen Gerät ein bereits trainiertes Modell genutzt wird, sofern kein weiteres Training des Modells nötig ist.

3) TensorFlow.js: TensorFlow.js ist eine JavaScript-Bibliothek zum Trainieren und Bereitstellen von Machine Learning Modellen im Browser und auf Node.js. Es können entweder vorhandene Modelle genutzt und weiter trainiert oder neue Modelle mit einer innovativen API gebaut und bereitgestellt werden [17]. Dies ermöglicht die Integration von maschinellem Lernen in Webseite, die auf jeder Plattform mit einem Browser genutzt werden kann.

IV. Verteiltes TensorFlow

Nachdem im vorherigen Kapitel ein kurzer Überblick über TensorFlow gegeben wurde, soll jetzt die Verteilung einer TensorFlow Umgebung auf mehrere Instanzen näher betrachtet werden. Dieses Kapitel stellt nötiges Wissen bereit, das im nächsten Kapitel im Rahmen eines Tutorials zur Einrichtung einer verteilten TensorFlow Umgebung eingesetzt wird.

A. Architektur

Im folgenden wird die Architektur einer verteilten TensorFlow Umgebung genauer beschrieben. Dabei wird zwischen dem Parameterserver und Arbeiter unterschieden.

1) Parameterserver: Der Parameterserver verwaltet das aktuelle Modell mit seinen Gewichtungen und Variablen und verteilt es an die Arbeiter. Falls durch mehrere Arbeiter zu viele Ein-/Ausgabe Anfragen an den Parameterserver erzeugt werden, lässt sich durch mehrere Parameterserver die Last verteilen. Mehreren Parameterserver synchronisieren das Modell untereinander. Abbildung 3 zeigt, wie das Verteilen des Modells an die Arbeiter aussehen kann.

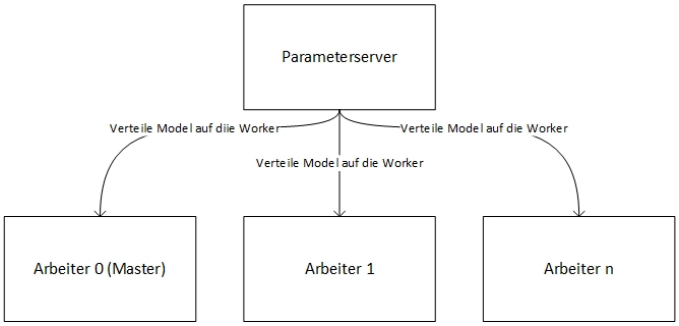

Fig. 3: Verteilen des Modells auf die Arbeiter

2) Arbeiter: Arbeiter führen ressourcenintensive Berechnungen zur Optimierung des vom Parameterserver bereitgestellten Modells durch. Nach der Optimierung sendet der Arbeiter das neue Modell mit seinen Gewichtungen und Variablen zum Parameterserver. Je mehr Arbeiter eingesetzt werden, desto schneller und effizienter kann ein Modell trainiert werden. Abbildung 4 zeigt, wie die Berechnung des Modells und die Aktualisierung des Modells auf dem Parameterserver aussehen kann. Der Master Arbeiter koordiniert die Trainingsvorgänge und kümmert sich um die Initialisierung des Modells, Zählen der Anzahl der ausgeführten Trainingsschritte, Speichern und Wiederherstellen von Modellkontrollpunkten.

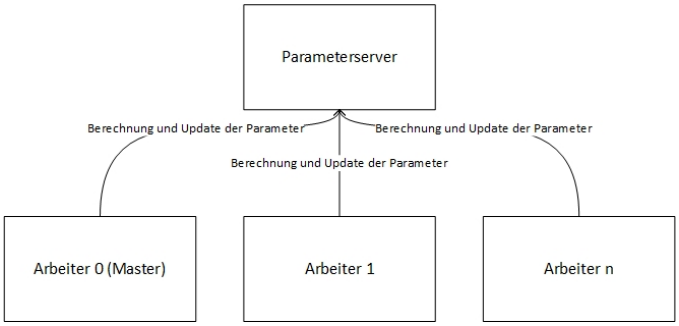

Fig. 4: Berechnung des Modells auf Arbeiter + Updaten der Variablen auf dem Parameterserver

B. Datenparallelität

Es gibt zwei verschiedene Möglichkeiten für Datenparallelität:

1) Synchron: Die Arbeiter lesen gleichzeitig das Modell vom Parameterserver und berechnen mit diesem ihre Trainingsoperationen und warten, bis die anderen Arbeiter fertig sind. Dann werden die Ergebnisse gemittelt und eine Aktualisierung des Modells wird an den Parameterserver gesendet. Somit hat zu jedem Zeitpunkt jeder Arbeiter das gleiche Modell.

2) Asynchron: Die Arbeiter lesen asynchron von den Parameterserver, berechnen ihre Trainingsoperation und senden asynchrone Aktualisierungen an den Parameterserver. Zu jedem Zeitpunkt können zwei verschiedene Arbeiter unterschiedliche Modelle besitzen.

C. Weitere Möglichkeiten

Bisher wurde die einfachste Methode beschrieben, wie ein verteilte TensorFlow Umgebung eingerichtet werden kann. Es gibt auch die Möglichkeit eine verteilte TensorFlow Umgebung auf Basis von Hadoop [18], Kubernetes [19], Docker [20] und Apache Spark [21] zu erstellen.

V. Tutorial: Erstellung einer verteilten TensorFlow Umgebung

Nachdem im vorherigen Kapitel das Konzept einer verteilten TensorFlow Umgebung erläutert wurde, wollen wir Schritt für Schritt eine solche Umgebung einrichten.

A. Vorbereitung der Instanzen

Als erstes müssen wir vier Instanzen aufsetzen. Wenn möglich sollte jede Instanz die gleichen Ressourcen zur Verfügung haben. In meinem Fall habe ich vier virtuelle Maschinen in der StudiCloud der Hochschule Furtwangen erstellt. Jede virtuelle Maschine hat eine virtuelle CPU, 1GB RAM und 10GB Festplattenspeicher zur Verfügung. Auf den virtuellen Maschinen wurde Ubuntu Server 16.04 installiert. Um die Kommunikation unter den virtuellen Maschinen zu vereinfachen, setzen wir wie in Abbildung 5 für jede virtuelle Maschine einen Hostname und die anderen virtuellen Maschinen als Hosts.

```
clouduser@tensorflow0:~$ cat /etc/hosts
127.0.0.1 localhost,tensorflow0

141.28.106.55 tensorflow0
141.28.106.56 tensorflow1
141.28.106.6 tensorflow2
141.28.106.7 tensorflow3
```

Fig. 5: Exemplarische Hosts Konfiguration

Im Idealfall sollten Instanzen mit wesentlich mehr CPU Leistung und einer zusätzlichen Grafikkarte eingesetzt werden, da nur so eine optimale Performance möglich ist. Im Rahmen dieser Arbeit standen mir diese Mittel nicht zur Verfügung.

B. Installation von TensorFlow

Nun müssen wir auf den virtuellen Maschinen Python und TensorFlow installieren. Python benötigen wir zum Entwickeln unserer Beispielanwendung und wir nutzen Python's Paketmanager pip zur Installation von TensorFlow. Dazu müssen folgende Befehle auf den virtuellen Maschinen ausgeführt werden.

- Installation von Python zur Entwicklung:
  `sudo apt-get install python3-dev`
- Installation von Python's Paketmanager pip
  `sudo apt-get install python3-pip`
- Installation von TensorFlow via pip
  `pip3 install tensorflow`

C. Entwicklung einer Anwendung

Die virtuellen Maschinen mit einer Python- und TensorFlow Installation sind jetzt einsatzbereit. Jetzt wollen wir eine einfache TensorFlow Anwendung erstellen, die eine virtuelle Maschine als Parameterserver und zwei virtuelle Maschinen als Arbeiter nutzt. Wir entwickeln die Anwendung lokal auf unserem Computer und verteilen sie danach auf die virtuelle Maschinen. Dazu erstellen wir die Datei *"train.py"* und öffnen sie in einer Entwicklungsumgebung unser Wahl (in meinem Fall Visual Studio Code).

1) Erstellung eines Clusters: Als erstes müssen wir ein Cluster aus Arbeiter und Parameterserver Instanzen definieren. Diese werden bei Anwendungsstart in den Argumenten "*–worker_hosts*" und "*–ps_hosts*" übergeben. Dies bietet den Vorteil, dass wir die gleiche Anwendung für Arbeiter bzw. Parameterserver verwenden können.

- Parsen der Argumenten:
```
ps_hosts = FLAGS.ps_hosts.split(",")
ps_hosts = FLAGS.ps_hosts.split(",")
```
- Erstellung des Clusters mit den Arbeitern und Parameterservern anhand der Parameter:
```
cluster = tf.train.ClusterSpec("ps":
ps_hosts, "worker": worker_hosts)
```

2) Starten der Instanz: Nachdem wir ein Cluster erstellt haben, müssen wir die jeweilige Instanz starten, dazu nutzen wir die Argumente "*–job_name*" und "*–task_index*".

- Starten der Instanz:
```
server = tf.train.Server(cluster,
job_name=FLAGS.job_name,
task_index=FLAGS.task_index)
```

3) Parameterserver: Als nächstes ist abzuklären, ob die aktuelle Instanz als Arbeiter oder Parameterserver dienen soll. Wenn die Instanz als Parameterserver dient, soll sie keine Berechnung durchführen, sondern warten bis ein Arbeiter Attribute anfordert oder aktualisieren will.

- Initialisierung einer Instanz als Parameterserver:
```
if FLAGS.job_name == 'ps':
server.join()
```

4) Arbeiter: Wenn eine Instanz kein Parameterserver ist, ist sie automatisch ein Arbeiter und führt Berechnungen durch. Die Arbeiter führen ihre Berechnungen synchron durch und warten deshalb nach jeder Iteration, bis jeder Arbeiter die Variablen auf dem Parameterserver synchronisiert hat. Anschließend verteilt der Parameterserver wieder die aktualisierten Variablen an die Arbeiter. Wir wollen dabei nicht genauer auf die Berechnungen eingehen und nutzen daher einen Beispielcode eines Tutorials von TensorFlow und schauen uns an, wie eine Berechnung auf einem Arbeiter durchgeführt wird:

- Zuweisung einer Berechnung zu einem Arbeiter:
```
with tf.device(tf.train
    .replica_device_setter(
    worker_device="/job:worker/task:%d",
    % FLAGS.task_index,
cluster=cluster)):
```
- Parsen des Arguments "is_chief", welches angibt, ob ein Arbeiter als Master fungiert:
```
is_chief = (FLAGS.task_index == 0)
```
- Starten einer neuen Session auf einem Arbeiter:
```
sess = tf.train.MonitoredTrainingSession(
master=server.target,
is_chief=is_chief)
```
- Starten einer Berechnung in der Session:
```
sess.run(task)
```
- Beenden einer Session, sobald die Berechnung beendet wurde:
```
sess.close()
```

Nun haben wir alle wichtigen Konfigurationen für unsere verteilte TensorFlow Umgebung vorgenommen. Der komplette Quellcode der Beispielanwendung kann unter "https://github.com/MichZipp/Exploring-Distributed-Tensorflow/Code" gefunden werden.

D. Starten einer Anwendung

Jetzt müssen wir die Anwendung noch starten, dazu müssen wir zuerst unsere *"train.py"* mit dem File Transfer Protocol (FTP) auf unsere Instanzen kopieren. Als nächstes müssen wir die Anwendung auf jeder Instanz mit den richtigen Argumenten starten. Um diesen Prozess zu vereinfachen, schreiben wir uns ein kleines Bash-Skript, das wir dann nur auf dem Master Arbeiter starten müssen:

- Bash-Script *"run.sh"* für ein Cluster aus zwei Arbeitern und einem Parameterserver:
```
#!/bin/bash
ssh clouduser@tensorflow2
    'python3  /Tensorflow/trainer.py
    --ps_hosts="tensorflow2:2222"
    --worker_hosts="tensorflow0:2222,
      tensorflow1:2222"
    --job_name="ps" --task_index=0'
ssh clouduser@tensorflow1
    'python3  /Tensorflow/trainer.py
    --ps_hosts="tensorflow2:2222"
    --worker_hosts="tensorflow0:2222,
      tensorflow1:2222"
```

```
    --job_name="worker" --task_index=1'
python3  /Tensorflow/trainer.py
    --ps_hosts="tensorflow2:2222"
    --worker_hosts="tensorflow0:2222,
      tensorflow1:2222"
    --job_name="worker" --task_index=0
```

- Ausführen des Bash-Skripts:
  `bash run.sh`

Um die Cluster-Zusammenstellungen zu verändern, muss nur das Bash-Skript angepasst werden. Bei vier Instanzen können folgende Cluster-Zusammenstellungen verwendet werden:

- 1 Arbeiter, 1 Parameterserver
- 2 Arbeiter, 1 Parameterserver
- 3 Arbeiter, 1 Parameterserver
- 1 Arbeiter, 2 Parameterserver
- 1 Arbeiter, 3 Parameterserver
- 2 Arbeiter, 2 Parameterserver

Wie dieses Tutorial zeigt, sind nicht viele Schritte notwendig, um eine verteilte TensorFlow Umgebung einzurichten.

VI. Evaluierung

Die Nutzung einer verteilten TensorFlow Umgebung zeigt deutliche Performance Vorteile. Bei der im letzten Kapitel entwickelten Beispielanwendung führen Arbeiter einfache Berechnungen durch und aktualisieren nach jeder Iteration die Variablen auf dem Parameterserver. Bei dieser Anwendung verkürzt sich jedoch die Ausführungszeit nicht, da die Berechnung zu wenig Zeit in Anspruch nimmt und die Anzahl an Iterationen vorgegeben ist. Es ist aber zu sehen, dass mit steigender Anzahl der eingesetzten Arbeiter sich das Ergebnis verbessert. D.h. wenn anstatt einem Arbeiter zwei Arbeiter eingesetzt würden, brauchen wir pro Arbeiter nur die Hälfte an Iteration und somit die halbe Zeit, um das selbe Ergebnis zu erhalten. Um dies zu demonstrieren, können wir die Anwendung mit unterschiedlicher Anzahl an Arbeitern starten und manuell die Anzahl an Iterationen anpassen. Folgendes Ergebnis hat sich dabei ergeben:

- Ein Arbeiter und ein Parameterserver:
  - Die Anzahl an Iterationen für eine Session wurde auf 1000 gesetzt:
  - Ergebnis:

```
[17.000729 15.826864]
[17.166576 15.995058]
[17.332096 16.162918]
Total Time: 92.45s
```

Fig. 6: Ein Arbeiter und ein Parameterserver

- Ein Arbeiter und zwei Parameterserver:
  - Die Anzahl an Iterationen für eine Session wurde auf 500 gesetzt:
  - Ergebnis:

```
[16.106974 18.34111 ]
[16.44194  18.667156]
[16.775562 18.991898]
Total Time: 51.41s
```

Fig. 7: Ein Arbeiter und zwei Parameterserver

- Ein Arbeiter und drei Parameterserver:
  - Die Anzahl an Iterationen für eine Session wurde auf 333 gesetzt:
  - Ergebnis:

```
[15.961393 17.170582]
[16.480898 17.682613]
[16.98057  18.175095]
Total Time: 34.25s
```

Fig. 8: Ein Arbeiter und drei Parameterserver

Dies zeigt, dass durch das verteilen der Anwendungen auf mehrere Arbeiter Berechnungen parallel durchgeführt werden können, wodurch sich die Performance bzw. Berechnungszeit verbessert.

VII. Zusammenfassung

Eine KI benötigt viele Ressourcen, um eine gute Performance zu gewährleisten. Es gibt derzeit einige Deep Learning Frameworks, mit der eine KI implementiert werden kann. Am Anfang des Artikels wurden einige Frameworks vorgestellt. Im Detail wurde auf das Framework TensorFlow von Google eingegangen. Dabei wurden die unterstützen Plattformversionen und Möglichkeiten, wie TensorFlow auf einem mobilen Endgerät eingesetzt werden kann beschrieben. Meist reicht eine Instanz nicht aus, um die Anforderungen einer Anwendung zu erfüllen. Deshalb gibt es die Möglichkeit, mehrere Instanzen zu nutzen, um die Performance des Frameworks zu verbessern. Dafür wurde zuerst eine Wissensgrundlage geschaffen, indem die Architektur einer verteilten TensorFlow Umgebung aufgezeigt wurde. Anschließend wurde erläutert, welche Aufgabe dabei ein Arbeiter und Parameterserver hat. Anhand eines Tutorials wurden die wichtigsten Schritte für das Aufsetzten einer solchen Umgebung beschrieben, wobei eine kleine Beispielanwendung entwickelt wurde. Diese wurde genutzt, um eine Evaluierung durchzuführen, welche bestätigt, dass der Einsatz von mehreren Arbeitern die Performance des Frameworks deutlich verbessert. Dabei wurden verschiedene Konfigurationen, unterschiedliche Anzahl an Arbeiter und Parameterserver, für die TensorFlow Umgebung genutzt. Die Performance einer KI wird auch in Zukunft eine Herausforderung darstellen, denn die Einsatzgebiete werden immer komplexer und breitgefächerter.

References

[1] ”Planet Wissen - Künstliche Intelligenz” https://www.planet-wissen.de/technik/computer_und_roboter/kuenstliche_intelligenz/ Accessed 20.06.2018

[2] ”Big Data Insider - Maschinelles Lernen und Deep Learning” https://www.bigdata-insider.de/ki-maschinelles-lernen-und-deep-learning-das-sind-die-unterschiede-a-588067/ Accessed 20.06.2018

[3] ”DL4J - Deep Learning Frameworks Comparison” https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch Accessed 20.06.2018

[4] ”Torch” https://pytorch.org/about/ Accessed 20.06.2018

[5] ”Caffe” http://caffe.berkeleyvision.org/ Accessed 21.06.2018

[6] ”Caffe2” https://caffe2.ai/ Accessed 21.06.2018

[7] ”Keras” https://keras.io/ Accessed 24.06.2018

[8] ”Elephas” https://github.com/maxpumperla/elephas Accessed 24.06.2018

[9] ”Chainer” https://chainer.org/blog/ Accessed 01.07.2018

[10] ”Microsoft Cognitive Toolkit (CNTK)” https://docs.microsoft.com/de-de/cognitive-toolkit/ Accessed 01.07.2018

[11] ”Wikipedia - Apache MXNet” https://en.wikipedia.org/wiki/Apache_MXNet Accessed 01.07.2018

[12] ”AWS Apache MXNet” https://aws.amazon.com/de/mxnet/ Accessed 01.07.2018

[13] ”AWS Introducing Gluon” https://aws.amazon.com/de/blogs/aws/introducing-gluon-a-new-library-for-machine-learning-from-aws-and-microsoft/ Accessed 01.07.2018

[14] ”Amazon DSSTNE” https://aws.amazon.com/de/blogs/aws/introducing-gluon-a-new-library-for-machine-learning-from-aws-and-microsoft/ Accessed 01.07.2018

[15] ”Elipse Deeplearning4j (DL4J)” https://deeplearning4j.org/i Accessed 01.07.2018

[16] ”Tensorflow” https://www.tensorflow.org/ Accessed 01.07.2018

[17] ”Tensorflow.js” https://js.tensorflow.org/ Accessed 03.07.2018

[18] ”TensorFlow on Hadoop” https://www.tensorflow.org/deploy/hadoop Accessed 06.07.2018

[19] ”TensorFlow on Kubernetes” https://banzaicloud.com/blog/tensorflow-on-k8s/ Accessed 06.07.2018

[20] ”TensorFlow on Docker” https://github.com/tensorflow/ecosystem/tree/master/docker Accessed 06.07.2018

[21] ”TensorFlow on Apache Spark” https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html Accessed 06.07.2018