Seminar Hochleistungsrechner - Aktuelle Trends und Entwicklungen
Wintersemester 2017/2018

Maschinelles Lernen im HPC

Manuel Rothenberg
Technische Universität München

30.01.2018

Zusammenfassung

Im vorliegenden Paper geht es um die Anwendung von Maschinellem Lernen im Bereich von High Performance Computing (HPC). Es werden grundlegende Methoden im maschinellen Lernen, wie 'Support Vector Machine' und Neuronale Netze vorgestellt. Diese werden anschließend auf verschiedene Bereiche zur Optimierung von HPC angewandt. Dazu gehören frühzeitige Fehlererkennung von Hardware, sowie Optimierungen zur Ausnutzung der vorhandenen Ressourcen, was sowohl Energie als auch Hardware betrifft. Es werden die Vorgehensweise sowie die Resultate in diesen Anwendungen vorgestellt und bewertet. Insgesamt wird gezeigt, dass mit maschinellem Lernen Optimierungen durchgeführt werden können, die sonst ungenutzt geblieben wären.

1 Einführung

Maschinelles Lernen fokussiert sich auf die Algorithmen-Konstruktion um Vorhersagen anhand von Daten zu machen. Dabei wird versucht eine Funktion $f : X \to Y$ zu finden bzw. zu erlernen, welche die Input-Domäne X (Daten) einer Output-Domäne Y (mögliche Vorhersagen) zuordnet. Die Elemente X und Y sind anwendungsspezifische Repräsentationen von Daten-Objekten bzw. Vorhersagen. [11]

Um eine Input- mit einer Outputdomäne mit Hilfe einer Funktion zu verknüpfen gibt es verschiedene Methoden. Die bekanntesten Methoden werden später noch vorgestellt. Dabei ist es jedoch immer wichtig, ein gutes Modell des Anwendungszwecks zu bestimmen. Anhand von diesem Modell kann dann eine tieferliegende Verknüpfung zwischen Input- und Outputdomäne hergestellt werden. Diese Verbindung ist oft nicht einfach über die menschliche Intuition zu finden. Auch Formel-basierte Ingenieursarbeit ist dazu oft nicht in der Lage, da die Anwendungsumgebung oft hoch-komplex und nicht-linear miteinander agiert [7].

Maschinelles Lernen ist daher interessant um Muster zu erkennen und Vorhersagen zu treffen, die z.B. im Bereich des High Performance Computing (HPC) verwendet werden können um Prozesse zu optimieren. Im Bereich von HPC agieren viele Rechner miteinander. Das führt dazu, dass diese Struktur organisiert werden muss um die Rechenkapazität eines Rechenzentrums auch effizient ausnutzen zu können. Dazu gehört unter Anderem das Scheduling, um Tasks auf dem am besten geeigneten Rechner auszuführen, sowie die Kommunikation zwischen Rechnern. Auch die Energienutzung eines ganzen Rechenzentrums muss organisiert werden. In dieser Hinsicht muss z.B. bestimmt werden, ob und welche Rechner hoch- bzw. runtergetaktet werden können. Auch die Kühlung ist ein sehr großer Posten in der Energiebilanz eines Rechenzentrums und kann mit Hilfe von Vorhersagen optimiert bzw.

reduziert werden. Des Weiteren muss auch sichergestellt sein, dass die benötigte Hardware dauerhaft verfügbar ist. In dieser Hinsicht bietet sich eine Vorhersage der Ausfallwahrscheinlichkeit von bestimmten Geräten an, um möglichst schon vor einem Defekt reagieren zu können.
All diese Prozesse können mit Hilfe von Maschinellem Lernen optimiert werden, um die Effizienz eines Rechenzentrums zu verbessern. Dies hilft letztendlich Hardware, Energie und nicht zuletzt Geld zu sparen, sowie die Umwelt durch weniger Ressourcen-Nutzung zu schonen.
Das vorliegende Dokument zielt im zweiten Kapitel darauf ab, Methoden des Maschinellen Lernens vorzustellen. Im dritten Kapitel wird auf die Anwendungsmöglichkeiten zur Optimierung von HPC mit Hilfe dieser Methoden näher eingegangen. Im vierten Kapitel folgt dann eine Zusammenfassung, sowie ein Ausblick.

2 Methoden in Maschinellem Lernen

Maschinelles Lernen kann mit Hilfe von verschiedenen Methoden angewandt werden. Dieses Kapitel soll einige davon vorstellen.

2.1 Überwachtes und unüberwachtes Lernen

In den meisten Fällen geht es beim maschinellen Lernen um die Klassifizierung von Datensätzen. Das System lernt aus vorhandenen Datensätzen und versucht anhand der Daten Muster und Gesetzmäßigkeiten zu erkennen, die auf zukünftige Datensätze angewandt werden können um diese zu klassifizieren. Alle Methoden des maschinellen Lernen können grob in zwei Kategorien aufgeteilt werden.

- 'Supervised Learning' (überwachtes Lernen)
- 'Unsupervised Learning' (unüberwachtes Lernen)

Bei beiden Kategorien erhält das System viele unterschiedliche vorhandene Datensätze mit jeweils vielen verschiedenen Merkmalen, anhand derer es Muster erkennen soll. Das Ziel ist dann, dass jeder Datensatz klassifiziert werden kann.
Der Unterschied besteht darin, dass beim Supervised Learning die Trainings-Datensätze schon klassifiziert sind. Das System kann dann lernen, welche Merkmaleigenschaften zu welcher Klassifizierung führen.
Beim Unsupervised Learning ist diese Klassifizierung der Trainings-Daten nicht vorhanden. Das System muss selbst erkennen, welche charakteristischen Merkmale zusammengehören und welche sich abgrenzen. Abbildung 1 zeigt grafisch den Unterschied zwischen Supervised und Unsupervised Learning. Links sieht man, dass die Datensätze bereits klassifiziert sind und dass das System anhand dieser Daten versucht eine mögliche Trennlinie zu finden mit Hilfe derer dann in Zukunft Datensätze klassifiziert werden können.
Rechts sieht man, dass alle Datensätze als Kreise dargestellt sind, die noch nicht klassifiziert wurden. Die farblich markierte Klassifizierung wird dann während des Unsupervised Learning ermittelt. Hier muss das System selbst herausfinden, welche Datensätze zu welcher Klassifizierung gehören.

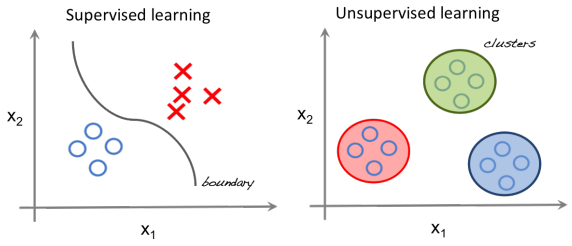

Abbildung 1: Grafische Erklärung von Supervised/Unsupervised Learning [1]

2.2 Support Vector Machine

Support Vector Machines (SVM) gehören zu den am Meisten genutzten Algorithmen zur Klassifizierung und Regression [11]. Eine SVM dient der

binären Klassifizierung. Die grundlegende Idee ist, eine Linie zu finden, die die Datensätze perfekt in ihre Klassen trennt. Dies soll auch möglich sein, wenn die Datensätze nicht linear trennbar sind. Um dies zu erreichen, muss eine Hyperebene gefunden werden, welche die Datensätze in eine höhere Dimension überführt, in der die Daten trennbar sind [3]. Diese Hyperebene ist mit Hilfe von Stützvektoren definiert. Um eine Balance zwischen korrekter und falscher Erkennung zu schaffen, kann der Algorithmus belohnt oder bestraft werden, je nachdem ob eine Klassifizierung korrekt oder falsch war.

2.3 Neuronale Netze

Neuronale Netze werden z.B. in den Bereichen Mustererkennung, Klassifizierung, Prognose und Regelung eingesetzt. Das Funktionsprinzip ist jedoch immer das gleiche [2]. Ein neuronales Netz ist ein System, das einen Datensatz als Eingabevektor erhält. Dieser Eingabevektor besteht aus den zu betrachtenden Merkmalen des Datensatzes. Anhand dieses Eingabevektors ermittelt das neuronale Netzwerk dann einen Ausgabevektor, der die gesuchte Information enthält. Ein- und Ausgabevektor müssen dabei nicht die selben Merkmale enthalten.
Das neuronale Netzwerk selbst besteht dabei aus beliebig vielen Neuronen (Units), welche den Eingabevektor verarbeiten und schließlich auch den Ausgabevektor erzeugen. Abbildung 2 zeigt die unterschiedlichen Arten von Neuronen. Neuronen die übereinander angeordnet sind, werden als Layer (Schicht) zusammengefasst. Zwischen Input-Layer und Output-Layer können mehrere sogenannte Hidden-Layer enthalten sein. Im Beispiel aus Abbildung 2 ist ein Hidden-Layer enthalten. Die Merkmale des Input-Vektors fließen von links nach rechts.

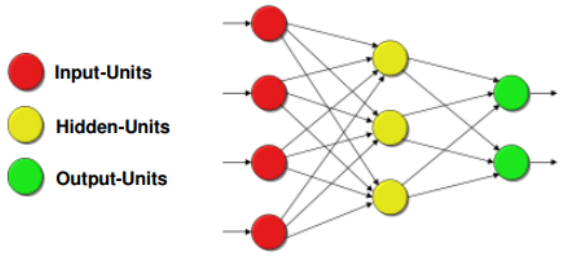

Abbildung 2: Neuronales Netz [13]

Neuronen sind über Kanten miteinander verbunden. Jede Kante hat ein Gewicht welches ausdrückt wie stark der Einfluss des sendenden Neurons auf das empfangende Neuron ist.

- $Gewicht > 0$: erregender Einfluss
- $Gewicht = 0$: kein Einfluss
- $Gewicht < 0$: hemmender Einfluss

Das Gewicht drückt das *Wissen* des neuronalen Netzes aus. Lernen ist demzufolge eine Anpassung des Kantengewichts.
Der Input, den ein Neuron von einem anderen Neuron erhält, hängt von dem Output des sendenen Neurons ab. Dieser Output wird mit dem Gewicht der verbindenden Kante multipliziert. Der gesamte Input eines Neurons wird Netzinput genannt und besteht aus den aufsummierten Inputs von allen Neuronen, die an dieses Neuron senden. [13]
Ein Neuron verarbeitet diesen Netzinput dann mit einer internen Übertragungsfunktion, die einem Netzinput ein Aktivitätslevel zuordnet. Diese Funktion kann z.B. linear, binär oder sigmoid sein. Das Aktivitätslevel wird in den meisten Fällen direkt als Output dieses Neurons verwendet. Abbildung 3 zeigt exemplarisch ein einzelnes Neuron und die

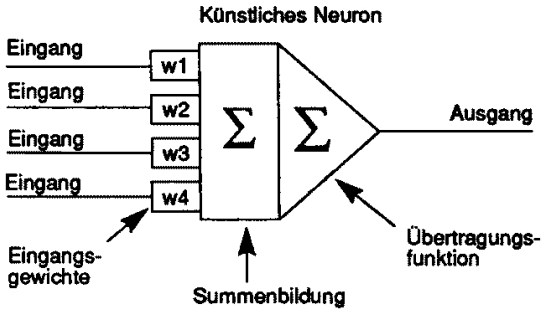

Abbildung 3: Ein einzelnes Neuron [5]

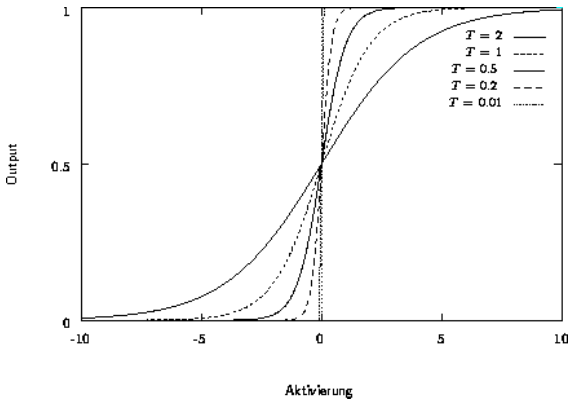

Abbildung 4: Sigmoide Funktionenschar [12]

Verarbeitung der Inputs. Abbildung 4 zeigt exemplarisch einige Sigmoid-Funktionen, die als Übertragungsfunktion dienen können.
Die Kombination vieler Neuronen und deren Verbindung ergibt dann ein neuronales Netz. Dieses Netz muss trainiert werden, was mit Hilfe von Trainingsdaten erfolgt. Diese Phase wird Trainingsphase genannt und kann sowohl in die Kategorie Supervised als auch Unsupervise Learning fallen. Während der Trainingsphase werden die Kantengewichte angepasst um ein funktionierendes Netz zu erhalten. Dies erfolgt mit Hilfe von Lernregeln. Anschließend erfolgt eine Testphase, in der das neuronale Netz nicht mehr angepasst wird, sondern nur noch verwendet wird um Ergebnisse zu validieren.

3 Maschinelles Lernen zur Optimierung von HPC

Nachdem nun ein Grundverständniss für maschinelles Lernen vorhanden ist, wird im folgenden Kapitel näher auf die Anwendung von maschinellem Lernen zur Optimierung von HPC eingegangen. Es werden dabei verschiedene Optimierungsansätze erläutert.

3.1 Ausfallvorhersage/Fehlertoleranz

In großen Rechenzentren wird im Regelfall viel Hardware betrieben. Dies hat zur Folge, dass mehr Geräte ausfallen können. Um den Betrieb nicht zu gefährden, wäre es deshalb von Vorteil, wenn vorhergesagt werden könnte, welche Geräte mit hoher Wahrscheinlichkeit in naher Zukunft ausfallen. Man könnte dann diese Geräte vorsorglich austauschen.
Errin Fulp, Glenn Fink und Jereme Haack haben in ihrem Artikel zum *USENIX Workshop on the Analysis of System Logs* vorgestellt, wie SVMs helfen können, Ausfälle von Festplatten anhand von System-Log Dateien vorherzusagen, siehe dazu auch [8]. Im folgenden Text wird deren Vorgehen näher erläutert.

3.1.1 Vorgehensweise

In großen Cluster-Systemen werden üblicherweise System-Log Einträge gespeichert, die über den Systemzustand und über Systemereignisse Auskunft geben können. Wird für dieses Logging das Unixtool *syslog* verwendet, besteht ein Log-Eintrag unter Anderem aus einer Identifikation des Nodes, einem Zeitstempel, einer Nachricht und einem Tag, der die Priorität der Nachricht repräsentiert. Umso kleiner der Tag-Wert, umso wichtiger ist dabei die Nachricht.
Des Weiteren haben inzwischen fast alle Festplatten *Self-Monitoring Analysis and Reporting Technology (SMART)* integriert. Dies ist ein System, welches Festplattenintern den eigenen Zustand (z.B. Lesefehler, Temperatur, Anlaufzeit, ...) überwacht. Werden bestimmte Grenzwerte überschritten, meldet die Festplatte selbst einen möglichen Fehler an das System, welcher dann auch als Syslog-Eintrag gespeichert wird. Diese Selbstüberwachung kann helfen Fehler zu erkennen, ist aber alleine nicht ausreichend [4].
In Abbildung 5 werden beispielhaft einige Syslog-Einträge gezeigt. Die letzte Nachricht mit dem 'Tag' 1 (höchste Priorität) zeigt z.B. einen Festplattenausfall. Für die SVM werden diese Daten aufbereitet. Das heißt, die Syslog-Einträge werden in Sequenzen der Länge k zusammengefasst. Anschließend werden die Häufigkeit der Tags pro Sequenz daraus extrahiert, so dass Paare von *tag:anzahl* entstehen. Aus dem Beispiel aus Abbildung 5, würde

Host	Facility	Level	Tag	Time	Message
198.129.8.6	local7	notice	189	1171061732	sysstat
198.129.8.6	kern	info	6	1171061732	kernel md: using maximum available idle IO bandwidth
198.129.8.6	cron	info	78	1171061733	crond 2500 (root) CMD (/usr/lib/sa/sa1 1 1)
198.129.8.6	auth	info	38	1171062445	rsh(pam_unix) 2215 session opened for user by (uid=0)
198.129.8.6	auth	info	38	1171062445	in.rshd 2216 root@hpcs2.cs.edu as root: cmd=/root/temps
198.129.8.6	daemon	info	30	1171062590	smartd 88 Device: /dev/twe0 SMART Prefailure Attribute
198.129.8.6	auth	notice	37	1171062590	sshd(pam_unix) 12430 auth failure; logname=el-fork-o
198.129.8.6	kern	info	6	1171062590	kernel md: using 512k, over a total of 12287936 blocks.
198.129.8.6	cron	info	78	1171062601	crond 2500 (root) CMD (/usr/lib/sa/fork-it 1 1)
198.129.8.6	kern	alert	1	1171062692	kernel raid5: Disk failure on sde1, disabling device

Abbildung 5: Syslog Auszug [8]

dies also so aussehen:

$(1:1\ ,\ 6:2\ ,\ 30:1\ ,\ 37:1\ ,\ 38:2\ ,\ 78:2\ ,\ 189:1)$

Dies sind die aggregierten Klassifizierungs-Merkmale. Des Weiteren werden nicht nur diese Merkmale, sondern auch die zeitliche Reihenfolge der Tags innerhalb einer Sequenz betrachtet. Dazu werden die Syslog-Einträge anhand des Tags in beispielsweise drei Prioritäts-Kategorien eingeteilt.

- $tag < 10$: hoch (0)
- $10 <= tag <= 140$: medium (1)
- $tag > 140$: niedrig (2)

Das Beispiel aus Abbildung 5 hätte somit die angepasste Sequenz:

$$(2, 0, 1, 1, 1, 1, 1, 0, 1, 0)$$

Mit $k = 10$, gibt es $3^{10} = 59.049$ mögliche angepasste Sequenzen, daher sollte k reduziert werden, um weniger mögliche angepasste Sequenzen zu erhalten. Anschließend kann jeder möglichen angepassten Sequenz eine ID zugeordnet werden um dann die Häufigkeit der angepassten Sequenz innerhalb einer Sequenz von Sequenzen zu bestimmen. Dies ist die Spektrum-Repräsentation, welche als Spektrum-Klassifizierer verwendet wurde.
Wenn in den Syslog-Einträgen ein Festplattenausfall auftritt, können dann die vorhergehenden Syslog-Einträge herangezogen und wie beschrieben aufbereitet werden. Die so entstandenen Vektoren können dann klassifiziert werden. Mit Hilfe dieser aufbereiteten Vektoren (Tag-Häufigkeit in einer Sequenz, Sequenz-Häufigkeit in einer Sequenz von Sequenzen) sowie der Klassifizierung $Fehler\ aufgetreten$, können dann SVMs trainiert werden. Während des SVM-Trainings sollten aber ebenso viele Sequenzen einbezogen werden, die keinen Fehler verursachen, anonsten fehlt die zweite Klassifizierung $kein\ Fehler\ aufgetreten$. Die SVMs versuchen dann eine Hyperebene zu finden, welche die Vektoren entsprechend der Klassifizierung trennt. Mit Hilfe der entstehenden SVMs, können dann zukünftige Syslog-Eintragssequenzen klassifiziert werden um Ausfälle frühzeitig zu erkennen.

3.1.2 Ergebnisse

Über einen Zeitraum von 24 Monaten wurden auf einem 1024-node Linux-Cluster Syslog-Einträge gesammelt. Dabei traten 100 eindeutige Festplatten-Ausfälle auf. Die jeweils 1200 Syslog-Einträge die vor einem Fehler auftraten, wurden dann aufbereitet und als Input einer SVM herangezogen. Um beide Seiten der Klassifizierung zu erhalten wurden ebensoviele Syslog-Einträge herangezogen, die zu keinem Fehler führten. Die Hälfte der gesamten Syslog-Einträge wurde fürs Training der SVM verwendet, die andere Hälfte für Tests der trainierten SVM.
Um eine bestmögliche Trainigskonfiguration zu finden, wurden mehrere Trainingsdurchläufe durchgeführt, die jeweils einen Teil M der 1200 Einträge verwendeten. Das Subset startet dabei immer beim ersten Eintrag der 1200 Einträge. M lag zwischen 400 und 1100. Bei $M = 1100$ ver-

bleiben z.B. nur noch 100 Einträge (durchschnittlich ca. 30 Stunden) bis der Fehler auftritt. Außerdem wurden sowohl ausschließlich aggregierte Klassifizierungs-Merkmale als auch eine Kombination aus aggregierten sowie auch Spektrum-Klassifizierungsmerkmalen verwendet.

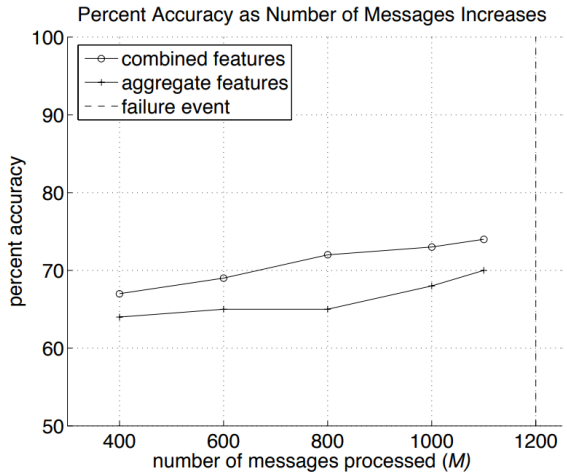

Abbildung 6: Vorhersagegenauigkeit der SVMs [8]

Abbildung 6 zeigt die Genauigkeit, mit der die SVMs Festplattenausfälle vorhersagen konnten. Es zeigt sich, dass bei $M = 1100$ die Genauigkeit am Besten ist, was aber zu erwarten war, da mehr Daten zu besseren Modellen führen. Außerdem wird gezeigt, dass eine Kombination aus aggregierten sowie auch Spektrum-Klassifizierungsmerkmalen besser ist, als nur die aggregierten Klassifizierungsmerkmale. Insgesamt konnte das System bei $M = 1100$ und einer Kombination der Merkmale eine Genauigkeit von 74% erreichen, bei $M = 1000$ eine Genauigkeit von 73%. Das heißt das 100 Syslog-Nachrichten bevor ein Festplatten-Ausfall auftritt, die SVM zu 74% dieses Ereignis vorhersagt.
Abbildung 7 zeigt, dass die SVMs auch gute Sicherheit bezüglich falscher Fehlermeldungen bietet. Die Kurven wölben sich alle nach oben links. Der Punkt (0/1) oben links würde eine 100% Trefferquote bedeuten. Die Fläche unter der Kurve (AUC) wäre dann genau $AUC = 1$, was dem besten Fall entsprechen würde. $AUC = 0,5$ würde einer zufälligen Klassifizierung entsprechen. Bei kombinierten Merkmalen und $M = 1000$, liegt dieser Wert jedoch schon bei $AUC = 0,79$.

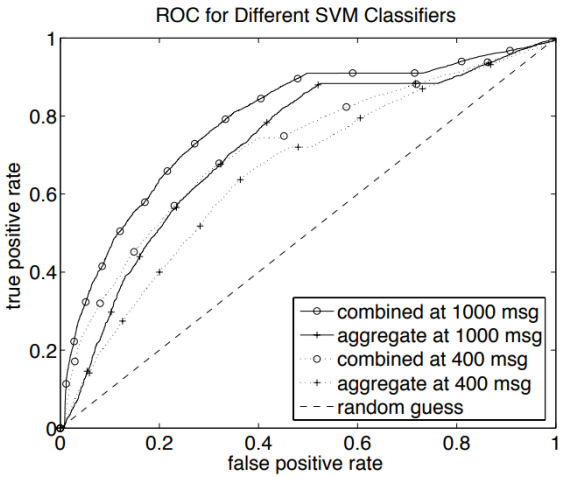

Abbildung 7: ROC Kurve der SVMs [8]

Insgesamt ist dieses Verfahren vielversprechend und kann helfen Festplatten-Ausfälle rechtzeitig vorherzusagen. Das System könnte noch erweitert werden, indem noch mehr Merkmale zur Klassifizierung hinzugezogen werden. Außerdem kann dieses Verfahren nicht nur auf Festplatten angewandt werden, sondern überall wo Daten gesammelt werden.

3.2 Energieoptimierung

Wie eingangs erwähnt, kann maschinelles Lernen verwendet werden um Rechenzentren in Hinsicht auf die Energieeffizienz zu optimieren. Um die Effizienz eines Rechenzentrums zu bewerten, wird von Google, aber auch von vielen anderen Institutionen das Power usage effectiveness (PUE) Verhältnis verwendet [6]. Diese Zahl ist das Verhältnis von insgesamt im Rechenzentrum verbrauchter Energie zur Energieaufnahme der Rechner. Umso näher der PUE-Wert sich 1 nährt, umso effektiver ist das Rechenzentrum.

$$PUE = \frac{gesamte\ Energieaufnahme}{Energieaufnahme\ der\ Rechner}$$

Ein hoher PUE-Wert bedeutet, dass viel Energie für Wärme und Wärmeabfuhr verbraucht wird.

Google gelang es mit Hilfe von maschinellem Lernen, den PUE eines Rechenzentrums um 15% zu senken, was einer Energieeinsparung bei der Kühlung von 40% entspricht [7]. Dies wurde durch das Training eines neuronalen Netzwerks ermöglicht. Trainingsdaten hierbei waren Aufzeichnungen von Kühlwassertemperaturen, Energieverbrauch, Pumpengeschwindigkeit, Anzahl und Art der Kühlsysteme, Wetterbedingungen, ... Mit Hilfe dieser Trainigsdaten konnte das Netzwerk den zu erwartenden PUE mit einer Genauigkeit von 0,4% bestimmen. Außerdem wurden zwei weitere neuronale Netz trainiert, um die zukünftige Temperatur sowie Auslastung des Rechenzentrums zu bestimmen. Anhand der PUE-Vorhersage konnten dann

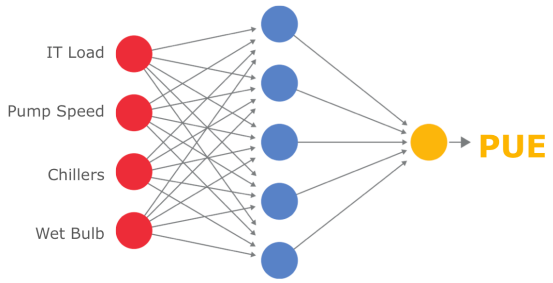

Abbildung 8: Vereinfachtes Neuronales Netzwerk zur Ermittlung des erwarteten PUE [10]

Änderungen am Rechenzentrum vorgenommen werden, die zu einer besseren Nutzung der Ressourcen führt. Mit Hilfe der Temperatur- und Auslastungsvorhersage kann außerdem sichergestellt werden, dass die Änderungen im anwendbaren Rahmen stattfinden, ohne das Hardware überstrapaziert wird. Des Weiteren ist die Möglichkeit der Rechenzentren-Simulation ein Anwendungszweck. So können Parameter modifiziert werden, um die dadurch folgenden Auswirkungen zu bestimmen, denn im realen Betrieb kann nicht einfach die Konfiguration beliebig modifiziert werden.

3.2.1 Vorgehensweise

Im folgenden Text wird näher darauf eingegangen, wie Google dieses neuronale Netzwerk entwickelt hat, siehe dazu auch [6].
Die Merkmale, die das neuronale Netzwerk zur Ermittlung des PUE als Input-Vektor erhält, sind so ausgewählt, dass möglichst wenig linear abhängige Merkmale vorhanden sind. Dies reduziert die Trainingszeit und reduziert die Chancen von Overfitting (zu starke Anpassung des Kantengewichts).
Der Trainingsprozess des neuronalen Netzwerks kann in effektiv vier Schritte aufgeteilt werden:
1. zufällige Initialisierung
Die Gewichte der Kanten werden zufällig initialisiert. Wichtig hierbei ist, dass die Katen nicht mit 0 initialisiert werden. Würden alle Kanten mit 0 initialisiert werden, hätte dies zur Folge, dass die nachfolgenden Schritte keine Anpassungschance erhalten, da alle Werte immer mit 0 multipliziert werden würden. Aus diesem Grund werden die Kanten mit zufälligen Werten im Bereich $[-1, 1]$ initialisiert.
2. Forward Propagation - Algorithmus anwenden
In diesem Schritt wird schichtweise der Netzinput jedes Neurons berechnet, da jede Schicht auf den Outputs der vorhergehenden Schicht basiert. Es wird dabei eine sigmoide Übertragungsfunktion g(z) implementiert, die das Feuern eines biologischen Neurons immitiert.

$$g(z) = \frac{1}{1 + e^{-z}}$$

Nachdem alle Schichten berechnet wurden, enthalten die Output-Neuronen konkrete Werte.
3. Kostenfunktion berechnen
Die Kostenfunktion ist üblicherweise der quadratische Fehler zwischen vorhergesagtem und tatsächlichem Output. Diese Größe sollte mit jeder Iteration kleiner werden.
4. Back Propagation - Algorithmus anwenden
Der Wert der Kostenfunktion wird nun schichtweise rückwärts durch das Netzwerk geleitet. Dabei wird jedes Kantengewicht anhand des mitverschuldeten

Fehlers angepasst.

5. Iteration

Die Schritte 2-4 müssen solange wiederholt werden, bis das neuronale Netzwerk konvergiert oder die maximale Anzahl an Iterationen erreicht ist.

Googles Neuronales Netzwerk enthält fünf versteckte Schichten mit jeweils 50 Neuronen. Als Regulierung wird ein Error-Grenzwert von 0,001 festgelegt. Die Input-Datensätze enthalten 19 Merkmale, der Output-Vektor nur eines, den PUE. Es stehen 185.435 Datensätze zur Verfügung, von denen 70% zu Trainings- und 30% zu Validierungszwecken verwendet werden. Da die Input-Merkmale in ihrem Wertebereich sehr unterschiedlich sind, werden diese zuvor auf einen Wertebereich zwischen $[-1, 1]$ normalisiert.

$$z_{NORM} = \frac{z - MEAN(z)}{MAX(z) - MIN(z)}$$

z ist der Input-Vektor und z_{NORM} der normalisierte Input-Vektor.

3.2.2 Ergebnisse

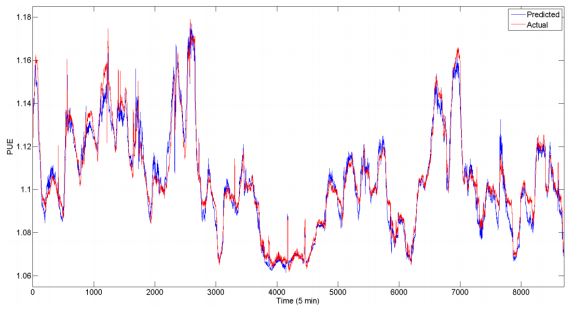

Abbildung 9: Vorhergesagter und tatsächlicher PUE [6]

Abbildung 9 zeigt in blau den vom neuronalen Netzwerk vorhergesagten PUE im Vergleich zum roten tatsächlichen PUE eines Google Rechenzentrums. Der mittlere absolute Fehler liegt bei 0,004 und die Standardabweichung bei 0,005. Generell ist der Fehler höher bei PUE-Werten größer als 1,14.

Mit Hilfe des erstellten Neuronalen Netzwerks zur PUE-Vorhersage konnten diverse Simulationen durchgeführt werden um einzelne Parameter zu optimieren. Ein Beispiel hierfür ist die Erhöhung des Temperatur-Schwellwerts des verlassenden Kühlwassers um ca. 1,6 Grad Celsius, was zu einem um durchschnittlich 0,005 reduzierten PUE führte.
Die PUE-Vorhersage kann auch zum aufspüren von Fehlerquellen im laufenden Betrieb genutzt werden, indem Anomalien aufgezeigt werden. Beispielsweise hat Google im Jahr 2011 Gas als Energiequelle zum Betreiben eines Rechenzentrums hinzugezogen. Einige Sensoren meldeten den Gaszufluss mit Impulsen alle 1000 Gaseinheiten, andere alle 100 oder sogar bei jeder Gaseinheit. Die Rechenzentrum-Betreiber konnten diesen Fehler feststellen, weil der reale PUE im Vergleich zum vorhergesagten PUE um bis zu 0,1 höher war, sobald Gas als Energiequelle verwendet wurde.
Ein weiteres Beispiel der Nützlichkeit von PUE-Simulationen hat sich gezeigt, als Google ein Datenzentrum erweiterte und dadurch während der Arbeiten ein Teil des Server-Traffics umgeleitet werden musste. Durch PUE-Simulationen konnte eine Konfiguration für diese Situation gefunden werden, die den PUE um 0,02 senkte.
Viele derartige Optimierungen summieren sich schnell auf und resultieren in einer großen Einsparung von Energie und somit Geld und nicht zuletzt auch einer besseren Umweltbilanz.
Anhand dieser Beispiele lässt sich schon erkennen, dass Maschinelles Lernen zu Optimierungszwecken sehr nützlich sein kann. Allerdings müssen die ausgewählten Input-Merkmale sorgfältig ausgewählt werden. Außerdem müssen viele Datensätze zur Verfügung stehen um ein präzises Modell zu erhalten.

3.3 Scheduling

Die bisherigen Anwendungszwecke von maschinellem Lernen zur Optimierung von HPC bezogen sich mehr auf die Hardware. Maschinelles Lernen kann aber auch zur Optimierung der Software verwendet werden. An der Universität von Barcelona wurde dazu im Bereich CPU-Scheduling maschinelles

Lernen verwendet [9]. Scheduling bestimmt, zu welcher Zeit und auf welchem Rechner ein Task läuft. Um das Scheduling so weit wie möglich auf energiesparende Taskzuteilung zu optimieren, ist es von Vorteil, wenn der Ressourcenverbrauch des Tasks bereits vor Ausführung bekannt ist. Außerdem ist in dieser Hinsicht wichtig, dass alle Rahmenbedingungen der Ausführung eingehalten werden und der Task innerhalb der vorgegebenen Zeit beendet wird und dauerhaft genügend Ressourcen zur Verfügung hat (Service Level Agreement SLA).
Beide dieser Parameter sind vor Ende der Taskausführung nicht bekannt, werden aber benötigt um ein optimales energiesparendes Scheduling erreichen zu können. Deshalb wurde an der Universität von Barcelona mittels vorhandener Daten ein Maschinenlern-System entwickelt, welches genau diese zwei Größen anhand von Taskparametern vorhersagt. Das System verwendet an dieser Stelle den M5P-Lernalgorithmus, welcher eine abschnittsweise lineare Funktion erzeugt. Mittels dieser Vorhersagen werden dann bekannte Scheduling-Algorithmen verwendet um Tasks auf dem bestmöglichen System zu starten oder einen Task auch auf ein anderes System zu verschieben.
Dieses Scheduling wurde dann mit demselben Scheduling-Algorithmus verglichen, der allerdings vom Benutzer vorbestimmte Parameter anstatt der Vorhersagen verwendete. Es hat sich gezeigt, das bei vielen unterschiedlichen Tasks, der Scheduler mit Vorhersagen vom Maschinenlern-System ca. 24% weniger Rechen-Nodes benötigte um die selben Tasks zu verteilen. Bei vielen sehr ähnlichen Tasks, hat der Scheduler mit vom Benutzer bestimmten Parametern besser abgeschnitten und ca. 30% weniger Rechen-Nodes benötigt um dieselben Tasks zu verteilen [9]. Allerdings war dieses Ergebnis zu erwarten, da homogene Tasks sehr gut verteilt werden können und Vorhersagen erst nützlich werden, wenn die Tasks sehr unterschiedlich sind.

4 Zusammenfassung

Nachdem nun mehrere Anwendungszwecke für maschinelles Lernen im HPC vorgestellt wurden, lässt sich sagen, dass maschinelles Lernen in diesem Bereich sehr hilfreich sein kann. In allen aufgezeigten Bereichen konnte ein Nutzen erzielt werden, welcher HPC effektiver macht. Dies kann die Vorhersage von Ausfällen sein, was hilft, fehlerhafte Komponenten rechtzeitig auszutauschen um die Verfügbarkeit zu erhöhen. Aber auch die Rechenzentren-Konfiguration bzw. -Steuerung kann optimiert werden um die vorhandenen Ressourcen besser zu nutzen. Auch die Task-Verteilung innerhalb des Systems kann verbessert werden, um effektiv mehr Rechenleistung zu erhalten.
Mit Hilfe von maschinellem Lernen können *Stellschrauben* angepasst werden, an denen mit menschlicher Intuition oder auch mit Formel-basierter Ingenieursarbeit evtl. niemals etwas verändert werden würde. Maschinelles Lernen hilft dabei, tiefere Verbindungen zwischen Merkmalen zu entdecken. Allerdings funktioniert dieser Prozess nicht von alleine. Die zu verwendenden Trainings-Merkmale müssen sorgfältig ausgewählt und auf einander abgestimmt werden, um brauchbare Resultate zu erhalten.
Bei sorgfältiger Anwendung sind jedoch noch viele weitere Optimierungen der vorgestellten Anwendungszwecke möglich. Auch viele weitere Anwendungen sind denkbar, da maschinelles Lernen überall angewandt werden kann, wo Daten vorhanden sind, aus denen man lernen kann.

Literatur

[1] Overview of Data Science, 2016. http://beta.cambridgespark.com/courses/jpm/01-module.html, Technischer Report, abgerufen Dezember 2017.

[2] W. Bibel, A. Scherer, and R. Kruse. *Neuronale Netze: Grundlagen und Anwendungen.* Computational Intelligence. Vieweg+Teubner Verlag, 2013.

[3] Dustin Boswell. Introduction to support vector machines. August 2002. Artikel.

[4] Wolf-Dietrich Weber Eduardo Pinheiro and Luiz André Barroso. Failure trends in a large disk drive population. *5th USENIX Conference on File and Storage Technologies*, 2007.

[5] Thorsten Schmidt David Fuchs. Neuronale Netze Das biologische Vorbild. http://www.geosimulation.de/methoden/neuronale_netze.htm, Website, abgerufen Dezember 2017.

[6] Jim Gao. Machine learning applications for data center optimization. 2014. https://static.googleusercontent.com/media/www.google.com/en//about/datacenters/efficiency/internal/assets/machine-learning-applicationsfor-datacenter-optimization-finalv2.pdf, Artikel.

[7] Richard Evans Jim Gao. DeepMind AI Reduces Google Data Centre Cooling Bill by 40%, 2016. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/, Technischer Report, abgerufen Dezember 2017.

[8] Errin W. Fulp Glenn A. Fink Jereme N. Haack. Predicting computer system failures using support vector machines. *USENIX Workshop on the Analysis of System Logs*, 2008. http://static.usenix.org/legacy/events/wasl08/tech/full_papers/fulp/fulp.pdf.

[9] Josep Ll. Berral Íñigo Goiri Ramón Nou Ferran Julià Jordi Guitart Ricard Gavaldà Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, 2010. https://upcommons.upc.edu/bitstream/handle/2117/7182/eenergy10.pdf.

[10] Joe Kava. Better data centers through machine learning. https://googleblog.blogspot.de/2014/05/better-data-centers-through-machine.html, abgerufen Dezember 2017.

[11] Ron Bekkerman Mikhail Bilenko John Langford. *Scaling Up Machine Learning - Parallel and Distributed Approaches*. Cambridge University Press, 2012.

[12] Uli Middelberg. Künstliche Neuronale Netze, 1995. http://www2.inf.uos.de/papers_html/dipl_95_uli/bp2.html, Diplomarbeit, abgerufen Dezember 2017.

[13] G.D. Rey and K.F. Wender. *Neuronale Netze: eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Aus dem Programm Huber: Psychologie-Lehrbuch. Huber, 2011.