Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Machine Learning in Datenkompressionsalgorithmen

vorgelegt von

Kevin Malinowski

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Seminar Effiziente Programmierung

Studiengang: Informatik
Matrikelnummer: 6420285

Betreuer: Anastasiia Novikova

Hamburg, 07-03-2018

Inhaltsverzeichnis

1 Einleitung

Maschinelles Lernen wird heutzutage in vielen verschiedenen Anwendungsbereichen verwendet. Sei es zur Spracherkennung bei Smartphones, bei Suchmaschinen wie Google oder sogar zur Erkennung von Kreditkartenbetrug.
In dieser Hausarbeit werde ich den Werdegang von PAQ, einem auf maschinelles Lernen basierten Kompressionsprogramm vorstellen.
Dazu werde ich zunächst Datenkompression grundsätzlich beschreiben.
Es stellen sich Schwierigkeiten, wie zum Beispiel, dass Lernen welches nicht zur Laufzeit geschieht unpraktisch ist und dass in der Realität meist neuere Information höhere Relevanz als ältere haben. Ich werde zeigen, dass Kompression ein durchaus sinnvolles Anwendungsgebiet von maschinellem Lernen ist.
Ich werde Unterschiede zwischen den Lernansätzen überwachtes und unüberwachtes Lernen nennen und durch Beispiele, wie dem künstlichen neuronalen Netzwerk und dem K-Means Algorithmus verdeutlichen. Zum Schluss werde ich noch einen Benchmark auflisten um zu zeigen wie gut PAQ sich gegen andere Komprimierungsprogramme schlägt, und aufzeigen, dass PAQ obwohl langsamer die beste Komprimierungsrate aufweist.

2 Machine Learning in Datenkompression

2.1 Kompression

Datenkompression oder Datenkomprimierung ist der Vorgang die Anzahl von Bits zu verdichten oder reduzieren, die benötigt werden um digitale Daten zu speichern oder versenden.
Dabei unterscheidet man zwischen verlustfreier und verlustbehafteter Kompression.
Verlustfreie Datenkompression findet statt, wenn aus dem komprimierten Datensatz das Original komplett wiederhergestellt werden kann, wie zum Beispiel wenn man etwas mit "7zip" packt und dann wieder entpackt.
Folglich ist verlustbehaftete Kompression, wenn wir beim Komprimieren Daten verlieren, die nicht wiederhergestellt werden können, beispielsweise bei Bild-, Musik- oder Videodateien wo entsprechend JPEG, MP3 und H.265/HEVC die Datensätze reduzieren, um heutzutage vor allem bei Streamingdiensten und Internettelefonie die benötigte Bandbreite gering zu halten und möglichst nur unwichtige Daten zu vernichten, wie Dinge die ein Mensch nicht wahrnehmen kann.
Verlustbehaftete Kompression wird stets in Form einer Umwandlung eines gegebenen Datensatzes ausgeführt. Hierbei wird der wichtige vom unwichtigen Teil getrennt. Der unwichtige Teil wird vernichtet und der wichtige Teil wird danach verlustfrei komprimiert.

2.1.1 Kompression bedeutet Verstehen

Um einen Datensatz zu komprimieren, ist es notwendig voraussagen zu können ob das nächste Bit 0 oder 1 ist. Wenn wir eine Sprache verstehen, ist es uns möglich vorauszusagen welches Wort als nächstes in einem Absatz stehen könnte, indem wir durch analysieren der vorherigen Wörter den Satz verstehen.
Analog dazu ist es auch bei Daten möglich das nächste Bit vorauszusagen, wenn man

vergangene Bitsequenzen versteht.
„Die beste Zukunftsvorhersagung ist jene Theorie, welche die Vergangenheit am einfachsten oder am kürzesten erklärt.“ [1]

2.2 Machine Learning

Maschinelles Lernen ist ein Gebiet der Informatik, welches einem Computersystem ermöglicht von Daten zu lernen und Vorhersagen treffen zu können. Man unterscheidet üblicherweise zwischen Lernphase und Vorhersagephasen.
In der Lernphase verbessert sich das System [es lernt], indem es versucht Muster in einem Datensatz zu erkennen und zu verstehen um in der Vorhersagephase mit einer gewissen Genauigkeit bestimmen zu können, was die Bedeutung von unbekannten Daten ist.
Tendenziell wird die Vorhersage besser, je länger die Lernphase ist. Dabei muss man einerseits genug lernen um Aussagekräftig genug zu werden, andererseits muss man aufpassen, dass das Computersystem bei zu viel Lernen der Datensatz auswendig gelernt wird, anstatt Gesetzmäßigkeiten darin zu erkennen. Dies führt dazu, dass das System schlechter darin wird unbekannte Daten vorherzusagen. Man nennt dieses Phänomen Overfitting(Überanpassung).
Generell unterscheiden wir beim Maschinellen Lernen zwischen supervised und unsupervised Learning, also überwachtes und unüberwachtes Learnen.

2.2.1 Supervised Learning

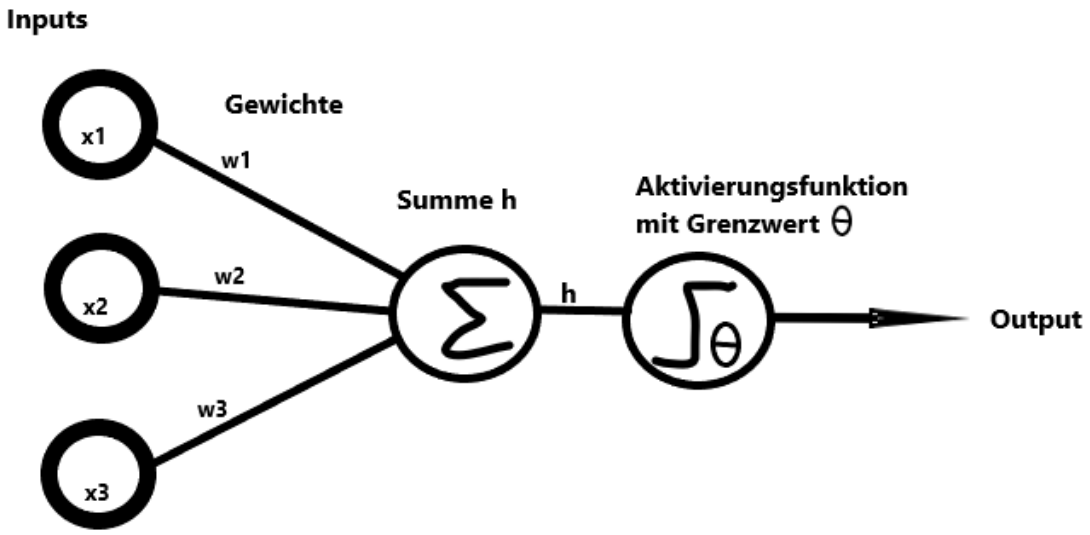
Beim überwachten Lernen beobachtet das System Testexemplare bestehend aus Input-Output Paaren, um eine Funktion zu finden, die zu einem Input den richtigen Output findet.
Nehmen wir als Beispiel einen Fahrschüler. Seine Inputs wären einerseits seine Wahrnehmungen, beispielsweise sieht er eine rote Ampel und die Outputs sind vom Fahrlehrer(dem supervisor) gegebene Anweisungen wie "Bremse!" oder "Links abbiegen." Weiter können auch Fotos unsere Inputs sein wobei die Outputs wiederum vom Fahrlehrer wären in Form von "Das ist ein Bus" oder "Das ist ein Vorfahrtschild." [2]

[1] [DCE], Section 1.4
[2] [AIA], Section 18.1

Es gibt viele verschiedene Algorithmen, die supervised Learning benutzen, welche unterschiedliche Stärken und Schwächen haben. Im folgenden stelle ich künstliche neuronale Netzwerke vor, da diese auch von PAQ benutzt werden.

Künstliche Neuronale Netzwerke

Künstliche neuronale Netzwerke(KNN) werden benutzt, um Computersystemen ohne feste Programmierung oder Vorwissen in Anlehnung an ein menschliches Gehirn etwas zu trainieren. Dabei werden Neuronen künstlich simuliert. Jedes künstliche Neuron hat ein oder mehr gewichtete Inputs und entscheidet durch eine Summenfunktion, ob es feuern soll oder nicht. In Realität haben solche Netzwerke allerdings mehrere Schichten. Erst wenn die letzte sogenannte Outputschicht feuert, erhalten wir eine Ausgabe vom KNN. Zur Veranschaulichung das von Frank Rosenblatt vorgestellte Perceptron, ein einfaches KNN.



Wir haben unsere Inputs x1,x2,x3 die jeweils ein Gewicht w1,w2 und w3 haben. Die Summe h ergibt sich aus: x1w1+x2w2+x3w3. Ist h größer als ein Grenzwert θ feuert das Neuron. Trainiert wird ein solches Neuron zum Beispiel durch Backpropagation. Da wir unser Trainingsset kennen, können wir sagen, ob das Netz richtige Ergebnisse liefert oder nicht. Nun gehen wir Rückwärts durch das Netz und passen die Gewichte so an, dass wir eine möglichst geringe Fehlerrate haben.

2.2.2 Unsupervised Learning

Unüberwachtes Lernen hat das Ziel, ohne jegliches Vorwissen über den Datensatz Gemeinsamkeiten zu erkennen, um so eine Information über die Daten zu erhalten. Man unterteilt üblicherweise die Daten dazu in Cluster.
Dabei sollen Objekte innerhalb eines Clusters möglichst ähnlich sein und Objekte verschiedener Clusters sollten möglichst unähnlich sein. Dabei sind Probleme, die sich stellen: Wie setzen wir unsere Schwellenwerte, um "ähnlich" zu definieren. Und wie erkennen wir sogenannte Ausreißer, also Daten die nicht zum Rest des Datensatzes passen. Diese können zum Beispiel bei einem Datensatz Messfehler aus einem Experiment sein. Im Folgenden erkläre ich den K-Means Algorithmus, da er für mich am besten das Prinzip von unsupervised Learning verdeutlicht.

K-Means

Nehmen wir an wir wollen Punkte in einem Koordinatensystem mit K-Means Clustern. Wir haben die Werte (3,4),(3,6),(3,8),(4,5),(4,7),(5,1),(7,3),(7,4) und (8,5) und wollen nun zwei Cluster finden, die sie möglichst gut aufteilen. Dazu wählen wir zunächst zwei beliebige Punkte als unsere Centroiden.(1)

Ich wähle (0,0) für Centroid 1 und (10,10) für Centroid 2.
Nun rechnen wir zu jedem Punkt den Abstand zu den Centroiden aus und fügen den Punkt, dem Cluster mit dem jeweils näheren Centroiden hinzu.(2)

Der Einfachheit halber benutze ich als Distanzfunktion: d(x,y)=|x1-y1|+|x2-y2| Cluster 1 enthält nun: (3,4),(3,6),(4,5),(5,1),(7,3) und Cluster 2:(3,8),(4,7),(7,4),(8,5). Nun rechnen wir von jedem Cluster den Mittelpunkt aus um so ein neuen Centroiden zu bekommen.(3)

Unsere Centroiden sind dann (4.4,3.8) und (5.5,6). Nun wiederholen wir Schritt 2 und 3 solange bis sich unsere Cluster nicht mehr ändern. Cluster 1: (3,4),(4,5),(5,1),(7,3),(7,4) ;Cluster 2: (3,6),(3,8),(4,7),(8,5). Centroid 1: (4,3.4) ; Centroid 2: (4.5,6.5). Cluster 1: (3,4),(4,5),(5,1),(7,3),(7,4) Cluster 2: (3,6),(3,8),(4,7),(8,5). Mit diesem Schritt sind die Cluster gleich geblieben, damit terminiert der Algorithmus.

2.3 Neural Network Data Compressor

1996 wollten Jürgen Schmidhuber und Stefan Heil zeigen, dass Neuronale Netzwerke vielversprechend für verlustfreie Datenkompression sind. Dazu entwickelten sie einen Kompressor, der Textdateien mit einer Kombination aus einem neuronalen Netzwerk und Kodierungsalgorithmen komprimiert.
Das neuronale Netzwerk ist dazu da, um mit Wissen vorheriger Zeichen, die bedingte Wahrscheinlichkeitsverteilung des nächsten Zeichen zu approximieren. Dabei werden Zeichen die wahrscheinlicher sind mit kürzeren Kodierungen kodiert als unwahrscheinliche. Dieses neuronale Netzwerk arbeitet offline. Das bedeutet, dass es eine explizite Lernphase hat, in der es mit einem separatem Trainingssatz trainiert wird. Gelernt wurde, dadurch dass der Trainingsdatensatz bekannt ist, mit überwachtem Lernen durch Backpropagation. Ist das Training vorbei, werden die Gewichte eingefroren und das trainierte Netzwerk nur noch zur Vorhersage verwendet.
Traning mit einer 10000 Wörter Trainingsdatei hat auf ihrer HP700 Workstation Tage gedauert. Und auch die Vorhersagephase lief 1000 mal langsamer als gewöhnliche Komprimiermethoden.

Matt Mahoney machte im Jahr 2000 einige Änderungen an Schmidhuber und Heils Datenkompressor, um das Programm zu beschleunigen. Das neuronale Netzwerk sagt nun ein Bit anstatt eines Zeichens voraus. Das führt dazu, dass mehr als nur Textdateien komprimiert werden können und das Training erfolgt nun zur Laufzeit. Dieses neuronale Netz enthält 258.000 Inputs, die in 4 Kontexte eingeteilt sind:

- Die letzten 8 gelesenen Bits konkateniert mit der Position(0-7) im aktuellen Byte.
- Die letzten 16 Bits Input
- Die letzten 24 Bits Input, die mit einer Polynomfunktion modulo 16 gehashed werden.
- die letzten 32 Bits gehashed in ein 17 Bit Wert.

Diese Änderungen führen dazu, dass nur noch wenige der Millionen Neuronen gleichzeitig aktiv sind.

2.3.1 PAQ

Mit dem Komprimierungsprogramm "PAQ1" versuchte Mahoney 2002, das Problem zu lösen, dass in Realität Datensätze oft nicht stationär sind. Das heißt, dass neue Daten relevanter sind als alte. Somit stimmt die Annahme nicht, dass jedes Bit gleich relevant zu betrachten sei.
Als Beispiel nimmt er einen Input von "...can...can...can...cat...cat". Wären alle Daten gleich relevant würde man davon ausgehen, dass nächstes mal wenn ca gelesen wird die Wahrscheinlichkeit für ein "n" höher ist als für ein "t", da "can" 3 mal auftritt und "cat" nur 2 mal. Doch in Realität würde man davon ausgehen, dass es wahrscheinlich cat sein wird, da in der Inputfolge das letzte "can" länger her ist als "cat".
Um dies in die Wahrscheinlichkeitsrechnung zu integrieren, verwendet PAQ1 einen 8-Bit Zähler, welcher die Anzahl zuletzt gelesener Nullen und Einsen zählt.
Dies ist möglich, da beide Zähler nicht gleichzeitig groß sein können und weil große Zähler approximiert werden können ohne signifikanten Kompressionsverlust zu bekommen. Dabei werden die Anzahlen 0-10 exakt dargestellt. Größere Anzahlen haben größeren Abstand je größer sie sind: 12, 14, 16, 20, 24, 28, 32, 48, 64, 96, 128, 256 und 512. Wird ein großer Zähler inkrementiert, springt der Zähler mit Wahrscheinlichkeit $1/(b-a)$. Zum Beispiel wenn wir 14 Einsen gelesen haben und jetzt noch eine gelesen wird, ist unser Zähler nicht 15, sondern mit Wahrscheinlichkeit $1/(16-14)$ wird er 16 oder bleibt bei 14.
„Ein Problem, dass dadurch entsteht neue Bits höher zu gewichten, ist das inkorrekte Ergebnisse auftreten, wenn der Datensatz tatsächlich unabhängig ist. Dazu benutzt PAQ1 eine Ad-hoc Lösung. Einerseits bevorziehen wir jüngere Daten, doch verhalten uns für Zähler <=5 wie ein stationäres Modell.“[3]

[3][PAQ], Section 3

2.3.2 Secondary Symbol Estimation

Beginnend mit Version PAQ2, die im Mai 2003 von Serge Osnach veröffentlicht wurde, verwendet das Komprimierungsprogramm eine sogenannte Secondary Symbol Estimation(SSE).
Diese soll die Vorhersagen des Kompressors verbessern. Sie ist eine Map, die nachdem PAQ1 eine Vorhersage trifft, eben diese Vorhersage und einen 10-Bit Kontext als Input nimmt. Sie gibt eine verbesserte Vorhersage des nächsten Bits aus. Dabei werden Vorhersagen auf 64 Werten einer nichtlinearen Skala quantisiert, wobei die Werte nahe 0 und 1 feiner aufgelöst sind. Nach jeder Vorhersage passt die SSE den Wert an, der am dichtesten getroffen wurde, um den Fehler der Vorhersage zu minimieren.
Dabei können mehrere SSE-Komponenten in aufsteigender Kontextgröße hintereinander geschaltet werden. Es ist auch möglich sie parallel zu schalten und den Durchschnitt der Ergebnisse zu nehmen.

September 2003 hat Mahoney die SSE überarbeitet, sodass nur noch 32 Werte in der Map stehen, aber sowohl der Wert über und der Wert unter der Vorhersage angepasst werden.

2.3.3 Adaptive Model Weighting

Mit PAQ4 wurde das Programm umgeschrieben, sodass Modelle, die das nächste Bit genauer Vorhersagen können, eine höhere Gewichtung bekommen. Wir haben nun 19 Modelle, also 19 neuronale Netzwerke zufällig initialisierte, die unabhängige Voraussagen treffen und kombinieren diese wie folgt:
$S_0 = \epsilon + \sum_i w_i n_{0i}$ =Anzahl der gezählten Nullen
$S_1 = \epsilon + \sum_i w_i n_{1i}$ =Anzahl der gezählten Einsen
$S = S_0 + S_1$ =Anzahl gezählter Bits
$p_0 = S_0/S$ =Wahrscheinlichkeit, dass nächstes bit eine 0 ist
$p_1 = S_1/S$ =Wahrscheinlichkeit, dass nächstes bit eine 0 ist
Nehmen wir an, dass alle Modelle gleich gewichtet werden, müssen wir unsere jeweiligen w_i=1 setzen, sodass wir nur die reinen Anzahlen der 1/0 benutzen. Da wir die Gewichte aber nun anpassen wollen, addieren wir sie mit der Partiellen Ableitung ihrer Kodierkosten.[4]

[4][Ada], Section 2.1

Wir erhalten:
$w_i = max[0, w_i + (x - p_1)(Sn_{1i} - S_1 n_i)/S_0 S_1]$
wobei $x - p_1$ unser Vorhersagefehler ist, x das zuletzt kodierte Bit ist und n_i die Anzahl der gezählten Bit vom Modell i ist. Dabei werden die Gewichte zunächst mit 0 initialisiert, um schnelles anfängliches Training zu garantieren.

Mit Version PAQ7 wurden logistische Gewichte der Modelle eingeführt, da diese eine bessere Kompressionrate gewährleisten. Dazu haben wir als Inputs jetzt jeweilige Wahrscheinlichkeiten aus den einzelnen Modellen, ob das nächste Bit eine 1 ist. Dann wäre die kombinierte Vorhersage: $p = squash(\sum_i w_i strech(p_i))$ wobei p_i die jeweiligen Vorhersagewahrscheinlichkeiten der Modelle ist.
strech(p)= ln(p/1-p) und squash ist die Umkehrfunktion von strech also 1/1(1-$e^{-}x$). Wieder passen sich die Gewichte mit Hilfe der partiellen Ableitung der Kostenfunktion an. $w_i := w_i + \lambda(y - p)strech(p_i)$ Dabei ist lambda unsere empirisch entnommene Lernrate und y-p unsere Fehlerrate. Es fällt auf das unsere Gewichte diesmal negativ sein können. Da Gleitkommazahlen eine gewisse Ungenauigkeit haben entnimmt PAQ strech und squash aus einer fest implementierten Wertetabelle.

3 Ausblick und Benchmark

Noch heute wird PAQ unter dem Namen ZPAQ weiterentwickelt. Der Namenswandel kommt davon, dass mit ZPAQ die erste PAQ Version erschienen ist, mit der man Pakete älterer Versionen entpacken kann. Dies geschieht, indem ZPAQ eine Beschreibung des benutzten Komprimieralgorithmus in die gepackte Datei legt.
Um die Arbeit abzuschließen, werde ich nun die einzelnen PAQ Versionen untereinander und dann mit anderen Komprimierprogrammen vergleichen.

Compressor		Calgary	Seconds	Memory	Date	Author	Major changes
P5		992,902	6.1*	256 KB	2000	Mahoney	64K x 1 neural network
P6		841,717	7.4*	16 MB	2000	Mahoney	1M neurons
P12		831,341	7.5*	16 MB	2000	Mahoney	Word context model
PAQ1		716,704	13*	48 MB	2002	Mahoney	Linear mixing with fixed weights
PAQ2		702,382	18*	48 MB	May 2003	Osnach	SSE
PAQ3		696,616	15*	48 MB	Sep 2003	Mahoney	Interpolated SSE
PAQ3N		684,580	30*	80 MB	Oct 2003	Osnach	Sparse models
PAQ4		672,134	43*	84 MB	Nov 2003	Mahoney	Adaptive mixer weights, record models
PAQ5		661,811	70*	186 MB	Dec 2003	Mahoney	Models for text, audio, images, runs, 2 mixers
PAQ6	-6	648,892	99*	202 MB	Jan 2004	Mahoney	Models for PIC, x86
PAQAR 4.0	-6	604,254	408*	230 MB	Jul 2004	Rhatushnyak	Many mixers and SSE chains
PAQ7	-5	611,684	142*	525 MB	Dec 2005	Mahoney	Logistic mixing, image models
PAQ8A	-4	610,624	152*	115 MB	Jan 2006	Mahoney	E8E9 preprocessor
PASQDA 4.4	-7 D	571,011	283*	470 MB	Jan 2006	Skibinski	PAQ7 + external dictionary
PAQAR 4.5	-5 D	570,374	299*	191 MB	Feb 2006	Rhatushnyak	PAQAR + external dictionary
PAQ8F	-6	605,650	161*	435 MB	Feb 2006	Mahoney	Bytewise indirect model, memory tuning
PAQ8L	-6	595,586	368	435 MB	Mar 2007	Mahoney	DMC model
PAQ8PX_V67	-6	598,969	469	421 MB	Jan 2010	Ondrus	Improved JPEG, TIFF, BMP, WAV models

Figure 3.1: Benchmark der PAQ Versionen. [DCE],4.3.6

Es wurde hierbei der sogenannte Calgary Corpus komprimiert. Dieser besteht aus 14 Dateien die als .tar Datei 3.152.896 Bytes groß sind. Enthalten sind bibliografische Referenzen, komplette Bücher als ASCII text, Computerprogramme, Bilder und Quellcodes um möglichst breit zu testen.
Getestet wurde auf einem 750 MHz Duron Prozessor und mit einem 2,0 GHz T3200. Die mit "*" markierten Zeiten sind auf dem Duron getestet wurden. Sie wurden angepasst an den T3200 mit Annahme, dass dieser 5.21 mal schneller wäre. Sofort zu erkennen ist, dass mit jeder neuen Version die komprimierte Bytegröße(die zweite Spalte) kleiner wurde. Ausnahme ist die Version PAQAR 4.0, die allerdings auch deutlich zeitintensiver

ist. Die mit D markierten Größen von PASQDA 4.4 und PAQAR 4.5 sind künstlich klein gehalten, da sie bei der Dekomprimierung ein englisches Wörterbuch enthalten müssen, was die Größe wiederum außerordentlich erhöhen würde. Die Tendenz ist auch, dass mit jeder Version die Komprimierzeit steigt. Von PAQ2 zu PAQ3 sinkt die Zeit, weil jetzt 2 statt nur einem Wert in der SSE angepasst werden.

Kompressor	Optionen	Komprimierte Größe in Bytes
Unkomprimiert		3.152.896
compress		1.319.521
Info-ZIP 2.32	-9	1.023.042
gzip 1.3.5	-9	1,022,810
bzip2 1.0.3	-9	860.097
7-zip 9.12b		824,573
ppmd Jr1	-m256 -o16	754,243
ppmonstr J		669,497
paq6	-8	647,110

Wieder wurde der Calgary Corpus komprimiert. Wir können auf ersten Blick sehen, dass bereits PAQ6 den Calgary Corpus stärker komprimiert als alle anderen gängigen Kompressoren.
Dies hat allerdings den Nachteil, dass es deutlich länger braucht. So hat der gleiche Rechner den Corpus mit PAQ6 140 Sekunden lang komprimieren müssen, wo beispielsweise 7zip nur 1.5 Sekunden gebraucht hat. Mittlerweile haben sich die Zeiten deutlich gebessert, so braucht der Rechner mit ZPAQ 1.10 nur 19.9 Sekunden, doch auch dies kommt nicht an traditionelle Kompressoren heran.

Quellenverzeichnis

[Ada] *Adaptive Weighing of Context Models for Lossless Data Compression.* – `http://mattmahoney.net/dc/cs200516.pdf`. Eingesehen am 28.02.2018

[AIA] *Artificial Intelligence A Modern Approach (3rd Edition).* `http://web.cecs.pdx.edu/~mperkows/CLASS_479/2017_ZZ_00/02__GOOD_Russel=Norvig=Artificial%20Intelligence%20A%20Modern%20Approach%20(3rd%20Edition).pdf`. – Eingesehen am 26.02.2018

[DCE] *Data Compression Explained.* `http://mattmahoney.net/dc/dce.html`. – Eingesehen am 21.02.2018

[DMC] *Data Compression Using Dynamic Markov Modelling.* `http://webhome.cs.uvic.ca/~nigelh/Publications/DMC.pdf`. – Eingesehen am 28.02.2018

[PAQ] *The PAQ1 Data Compression Program.* `http://mattmahoney.net/dc/paq1.pdf`. – Eingesehen am 28.02.2018

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Unterschrift

Veröffentlichung

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik eingestellt wird.

Ort, Datum	Unterschrift