Universität
Augsburg
University

Bachelorarbeit

Titel der Bachelorarbeit: Essen Erkennung

Name des Autors: TEMKENG Thibaut

Datum der Abgabe: Ende September

Betreuung: Name der Betreuer/ des Betreuers:Shou Liu

Fakultät für Embedded Intelligence for Health Care and Wellbeing

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

2 Grundlagen

2.1 Entwicklung von Künstlichen Neuronalen Netzen

3 Feedforward

4 Neuron

Das Neuron
- formalisierte Entsprechung der Nervenzelle besteht aus

- **Aktivierungszustand** (*Activation state*) Er gibt oder definiert den aktuellen Zustand eines Neurons.
- **Propagierungsfunktion** (*propagation function*) Sie bestimmt den gesamten Input(auch Netto-Input) des Neurons.Sie legt fest, wie die Eingabe des Neuron aufzuarbeiten ist Die häufigste verwendete Propagierungsfunktion ist die summe der gewichteten Eingaben, die das Neuron von anderen Neuronen erhält.
- **Aktivierungsfunktion** (*activation function*) Sie legt fest, wie der nächste Aktivierungszustand des Neurons aus dem Netto-Input und dem aktuellen Aktivierungszustand berechnet wird. Es gibt zahlreiche Aktivierungsfunktionen(zB. ELU, LeakyReLU, Sigmoid, Softmax, Maxout, tanh), die auch unterschiedliche Einflüsse auf die Ausgabe eines Neurons. Die verbreitetsten Aktivierungsfunktionen sind die Logistische(Sigmoid, Gleichung 4.2, Graph 6), **Re**ctified **l**inear **u**nit(ReLU, Gleichung 4.1, Graph10,) Relu und Softmax(Gleichung 4.3) Funktion.

$$ReLU = max(x, 0) \tag{4.1}$$

$$Sigmoid(x) = \frac{\mathrm{e^x}}{\mathrm{e^x} + 1} \tag{4.2}$$

$$Softmax(x_1, x_2, \cdots, x_n) = \frac{(e^{x_1}, ^{x_2}, \cdots, e^{x_n})}{\sum_{i=1}^{n} e^{x_i}} \tag{4.3}$$

Nach [**?**] sollten die Aktivierungsfunktionen vorgezogen werden, die mehr von folgenden Eigenschaften aufweisen:

Nichtlinearität: Bei einem mehrschichtigen Netz ist es nicht sinnvoll eine lineare Aktivierungsfunktion zu benutzen, denn ein solches Netz in ein einschichtiges Netz immer überführt werden kann und es ist bekannt, dass ein mehrschichtiges Netz mehr als ein einschichtiges Netz leisten kann.

Überall differenzierbar: Diese Eigenschaft ermöglicht, gradientenbasierende Optimierungsverfahren zu verwenden.

Wertebereich: Die gradientenbasierende Lernmethode ist stabiler, wenn der Wertebereich der Aktivierungsfunktion endlich ist und wenn es dagegen unendlich ist, ist das Lernen im Allgemeinen effektiver.

Monotonie: Wenn die Aktivierungsfunktion monoton ist, so ist die Fehleroberfläche eines einschichtigen Netz immer konvex,was bedeutet, dass es nur ein optimales Minimum gibt und das Optimierungsverfahren immer besser wird.

Identität in 0 ($f(x) \approx x$ wenn $x \approx 0$): Diese Eigenschaft ermöglicht einen schnellen Training, wenn die Gewichte zufällig initialisiert sind und wenn die Aktivierungsfunktion in der Nähe von Null nicht gegen die Identität konvergiert, muss bei der Initialisierung der Gewichte besonders sorgfältig vorgegangen werden.

Sigmoid-Aktivierungsfunktion Obwohl die Sigmoid-Funktion leicht anwendbar und differenzierbar ist, wird sie nach und nach auch nicht mehr verwendet, denn sie zum Beispiel das Problem des verschwindenden Gradienten(*vanishing gradient problem*) nicht löst, was die Leistung tiefer neuronaler Netze stark beeinträchtigt und sie konvergiert sehr langsam.

Ausgabefunktion(output function)

5 Layer in CNN

5.1 Convolution Layer

5.1.1 Standard Convolution Layer

5.1.2 Depthwise Separable Convolution Layer

Kombination von Depthwise und Pointwise Convolution. Verringerung der Anzahl von Parametern und Berechnungen.

5.2 Aktivierungsfunktion

Das neuronale Netzwerk wird während dem Training mit sehr vielen Daten gespeist und das sollte in der Lage sein, aus diesen Daten zwischen relevanten und irrelevanten Informationen Unterschied zu machen. Die Aktivierungsfunktion auch Transferfunktion oder Aktivitätsfunktion genannt, hilf dem NN bei der Durchführung dieser Trennung. Es gibt sehr viele Aktivierungsfunktionen und in folgenden werden wir sehen, dass eine Aktivierungsfunktion je nach zu lösende Aufgaben vorzuziehen ist.

$$\begin{cases} Y = f(\Sigma(Gewicht * Input + Bias)) \\ f := Aktivierungsfunktion \end{cases}$$

Binäre Treppenfunktion ist extrem einfach, siehe Abbildung 2, definiert als $f(x) = \begin{cases} 1, & \text{if } x \geq a \text{ (a:= Schwellenwert)} \\ 0, & sonst \end{cases}$. Sie ist für binäre Probleme geeignet, also Probleme

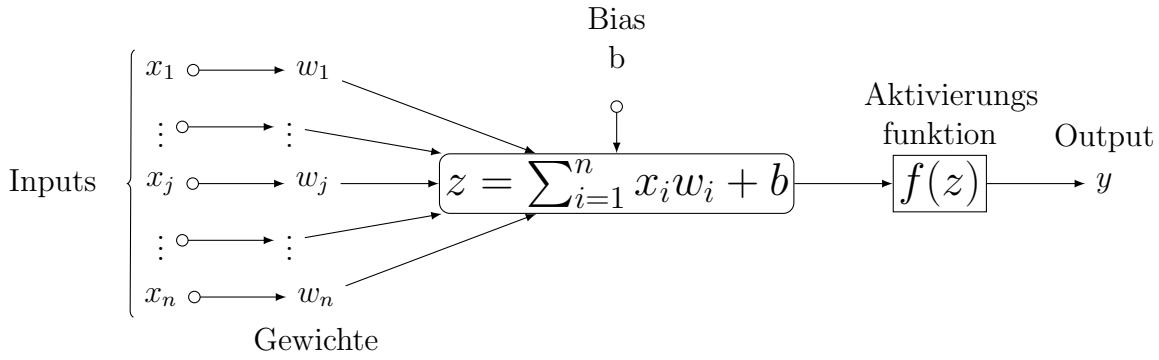
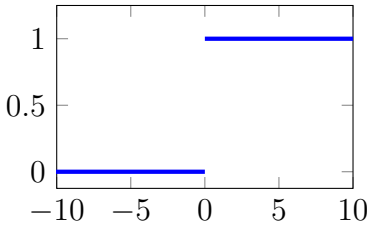


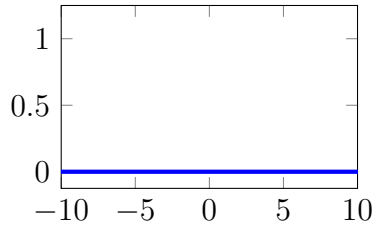
Abbildung 1: Funktionsweise von künstlichen Neuronen

wo man mit *ja* oder *nein* antworten sollte.Sie kann leider nicht mehr angewendet werden, wenn es mehr als zwei Klassen klassifiziert werden soll oder wenn das Optimierungsverfahren gradientenbasierend ist, denn Gradient immer null.

Abbildung 2: Binäre Treppenfunktion



(a) Binäre Treppenfunktion

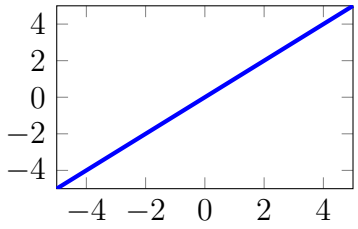


(b) Ableitung Binäre Treppenfunktion

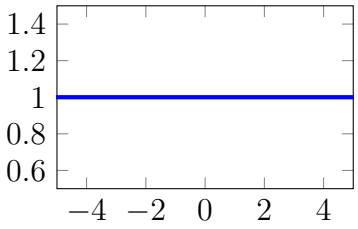
Lineare Funktion ist definiert als $f(x) = ax$, $f'(x) = a$, siehe Abbildung 4. Sie ist monoton, null zentriert und differenzierbar. Es ist jetzt möglich,nicht mehr nur binäre Probleme zu lösen und mit gradientenbasierenden Optimierungsverfahren während der Backpropagation Parameter anzupassen, denn Gradient nicht mehr null, also sie ist besser als binäre Funktion .Nutzt ein mehrschichtiges Netz die lineare Aktivierungsfunktion, so kann es auf ein einschichtiges Netz überführt werden und mit einem einschichtigen Netz können komplexe Probleme nicht gelöst werden.Außerdem ist der Gradient konstant. Der Netzfehler wird also nach einigen Epochen nicht mehr minimiert und das Netz wird immer das Gleiche vorhersagen.

Logistische Funktion ist definiert als $f(x) = \frac{1}{1+exp(-x)}$,$f'(x) = \frac{exp(x)}{(1+exp(x))^2}$, siehe Abbildung 6. Sie ist differenzierbar, monoton, nicht linear und nicht null zentriert(hier nur positive Werte).Zwischen $[-3, +3]$ ist der Gradient sehr hoch.Kleine Änderung in der Netzinput führt also zu einer großen Änderung der Netzausgabe. Diese Eigenschaft ist bei Klassifikationsproblemen sehr erwünscht.Die Ableitung ist glatt und von Netzinput abhängig. Parameter werden während der Backpropagation je nach Netzinput angepasst. Außerhalb von $[-3, 3]$ ist der Gradient fast gleich null, daher ist dort eine Verbesserung der Netzleistung fast nicht mehr möglich.Dieses Problem wird Verschwinden des

Abbildung 4: Lineare Funktion



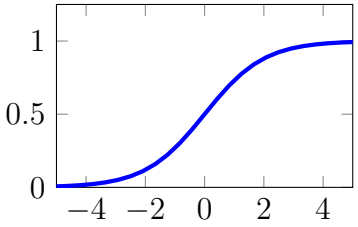
(a) Lineare Funktion: $f(x) = x$



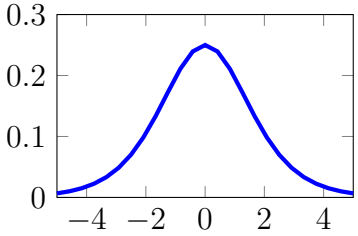
(b) Ableitung Lineare Funktion: $f'(x) = 1$

Gradienten *(vanishing gradient problem)* genannt.Außerdem konvergiert das Optimierungsverfahre sehr langsam und ist wegen wegen der exponentiellen (e^x) Berechnung rechenintensiv.

Abbildung 6: Logistische Aktivierungsfunktion:$sigmoid(x)$.



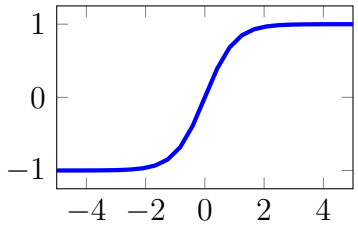
(a) Logistische Aktivierungsfunktion.



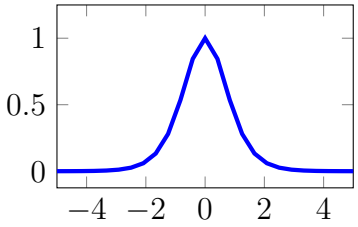
(b) Ableitung der Logistische Funktion.

Tangens Hyperbolicus ist definiert als $tanh := 2sigmoid(x) - 1$, siehe Abbildung 8.Außer dass sie null zentriert ist, hat sie die gleichen Vor- und Nachteile wie die Sigmoid Funktion.Sättigung fehl noch

Abbildung 8: Tangens Hyperbolicus.



(a) Tangens Hyperbolicus.

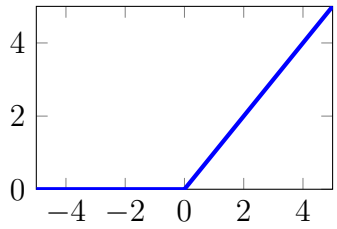


(b) Ableitung der Tangens Hyperbolicus.

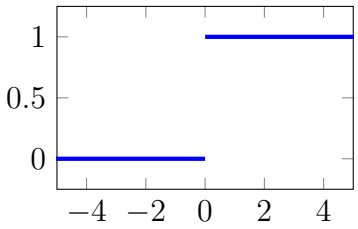
Rectified Linear Unit(ReLU) ist definiert als $f(x) = max(x, 0)$, siehe Abbildung 10. Sie ist sehr leicht zu berechnen. Es gibt keine Sättigung wie bei *Sigmoid* und *tanh*. Sie ist nicht linear,deshalb kann den Fehler schneller propagiert werden.Ein größter Vorteil der ReLU-Funktion ist, dass nicht alle Neurone gleichzeitig aktiviert sind, negative Eingangwerte werden zu null,daher hat die Ausgabe von Neuronen mit negativen Eingangwerten

keine Einfluss auf die Schichtausgabe, diese Neurone sind einfach nicht aktiv.Das Netz wird also spärlich und effizienter und wir haben eine Verbesserung der Rechenleistung. Es gibt keine Parameteranpassungen, wenn die Eingangwerte negative sind, denn der Gradient ist dort null.Je nachdem wie die Bias initialisiert sind, werden mehrere Neuron töten,also nie aktiviert und ReLU ist leider nicht null zentriert.

Abbildung 10: ReLU Aktivierungsfunktion



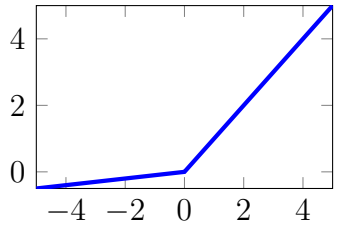
(a) ReLU Aktivierungsfunktion



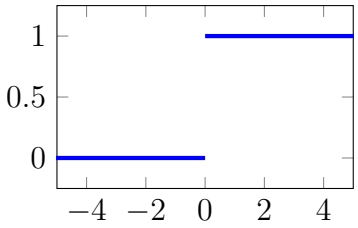
(b) Ableitung der ReLU Funktion

Leaky ReLU Funktion ist definiert als $f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & sonst \end{cases}$, siehe Abbildung 12. Sie funktioniert genauso wie die ReLU-Funktion, außer dass sie das Problem des toten Neurons und sie ist null zentriert.Es gibt somit immer eine Verbesserung der Netzleistung, solange das Netz trainiert wird.Wenn das Problem von Leaky ReLU nicht gut gelöst wird, wird empfohlen, die *Parametric ReLU* (PReLU) Aktivierungsfunktion zu verwenden, die während der Training selber lernt, Problem der toten Neurone zu lösen.

Abbildung 12: Leaky ReLU Funktion



(a) Leaky ReLU Funktion



(b) Ableitung der Leaky ReLU Funktion

Softmax ist definiert als $f(x_1, x_2, \cdots, x_n) = \frac{(e^{x_1}, x_2, \cdots, e^{x_n})}{\sum_{i=1}^{n} e^{x_i}}$.Die Softmax-Funktion würde die Ausgänge für jede Klasse zwischen null und eins zusammendrücken und auch durch die Summe der Ausgänge teilen. Dies gibt im Wesentlichen die Wahrscheinlichkeit an, dass sich der Input in einer bestimmten Klasse befindet.

In allgemein wird die ReLU aufgrund des Problems der toten Neurone nur in versteckte Schichten und die Softmax-Funktion bei Klassifikationsproblemen und Sigmoid-Funktion bei Regressionsproblemen in Ausgabeschicht verwenden.

5.3 Pooling Layer

Die Funktionsweise von Pooling-Schichten ist sehr ähnlich zu der von Convolutional-Layers. Pooling-Schichten besitzen einen Filter(*pool_size*) und eine Schrittgröße(*stride*). Der Filter gibt nur die Große der Blocks, der zusammengefasst wird und die Schrittgröße sagt wie der Filter über die Daten bewegen werden sollte.In die Praxis werden in den meisten Fällen einen 2 x 2 Filter und eine 2 x 2 Schrittgröße verwenden.

Durch die Pooling-Schichten wird die räumliche Dimension verringert und durch weniger räumliche Informationen haben wir nicht nur einen Gewinn an Rechenleistung, sondern auch eine wesentliche Netzparameterreduzierung, was die Wahrscheinlichkeit einer Überpassung(*Overfitting*) verkleinert.In allen Fällen hilft das Pooling, dass die Darstellung etwa invariant gegenüber kleinen Parallelverschiebung der Eingabe wird[4], d.h. Das Pooling-Layer gibt genau die gleiche Antwort unabhängig davon, wie seine Eingabe ein Bisschen verschoben wird.

Wir unterscheidet momentan vier Forme von Pooling-Schichten: Max-Pooling-Layer, Average-Pooling-Layer, Global-Average-Pooling-Layer und Global-Max-Pooling-Layer.

und somit auch die Anzahl von Netzparametern reduziert.Nach einem Convolutional-Layer haben wir sehr oft eine hohe Anzahl an Feature-Map, die das Netz versucht, zu lernen, aber all diese Merkmale sind für die Klassifizierung nicht immer wichtig. z.B Die Farbe eines Tellers mit Hähnchenflügel, wenn man Essen klassifizieren möchte.

5.3.1 Max-Pooling-Layer

Bei Max-Pooling-Layer lässt sich der Max(imum)-Operator auf den Filter anwenden, das heißt es wird nur der maximale Wert im Filter betrachtet.In Abbildung14 wird ein Beispiel der Funktionsweise des Max-Pooling-Layer.Im grünen Bereich liegt 16,3,5 und 10, deshalb steht $16 = max\{16, 3, 5, 10\}$ nach dem Max-Pooling-Layer im grünen Bereich.

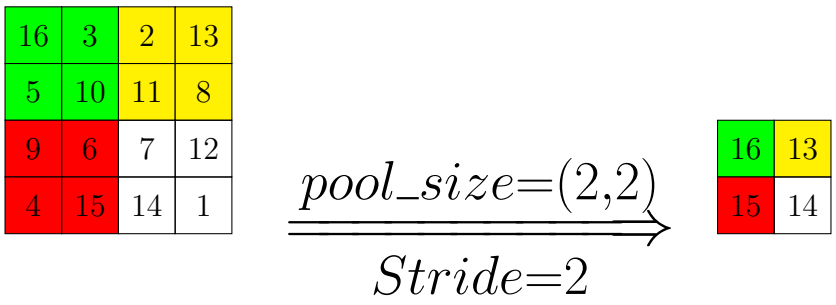


Abbildung 14: Funktionsweise eines Max-Pooling-Layer

5.3.2 Average-Pooling-Layer

Funktioniert so ähnlich wie das Max-Pooling-Layer, außer dass der Durchschnittswert anstatt das Maximum hier berechnet wird.Dies ist beispielhaft in Abbildung 15 dargestellt. Hier ergibt sich für den gelben Bereich folgende Rechnung: $\frac{9+6+4+15}{4} = 8.5$

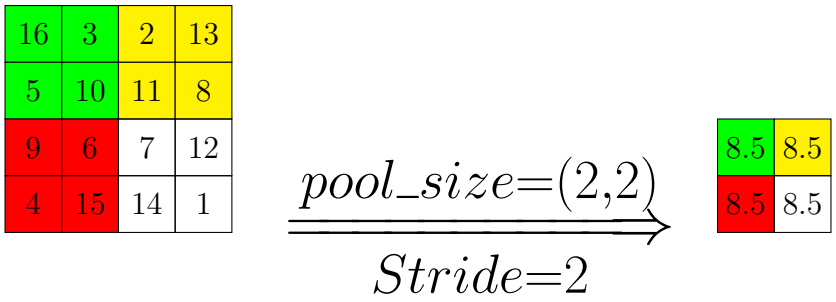


Abbildung 15: Funktionsweise eines Average-Pooling-Layer

5.3.3 Global Pooling Layer

Eine Variante des Max-Pooling-Layer bzw. Average-Pooling-Layer ist das Global-Max-Pooling-Layer bzw. Global-Average-Pooling-Layer.Für Global-Pooling-Layer wird die Berechnung des Maximums bzw. des Mittelwerts nicht Blockweise über die Daten durchgeführt, sondern über die gesamten Daten, das heißt es gilt immer pool_size= Stride=(Höhe,Breite)=räumliche Dimension der Daten. Wenn die oben ausgeführten Beispiele für Global-Pooling-Layer betrachtet, ergibt sich folgende Ergebnisse:

Global-Max-Pooling:16 Global-Average-Pooling:8.5

Das Global-Pooling-Layer wird sehr oft angewendet, um das Vorhandensein von Merkmale in Daten aggressiv zusammenzufassen. Es wird auch manchmal in Modellen als Alternative zur Verwendung einer vollständig verbundenen Schicht(Fully Connected Layer (FCL)) oder Flatten-Schicht beim Übergang von Feature-Maps zu einem verborgenen FCL oder einer Ausgabeschicht verwendet.

5.4 Fully Connected Layer

6 Backforward

6.1 Fehlerfunktion

Was wird hier gemacht

- Definition von Fehlerfunktion
- Warum wird die Fehlerfunktion berechnet.
- Anwendungsbeispiel.

Das Training von KNNs besteht darin, den vom NN begangenen Fehler zu korrigieren bzw. zu minimieren, daher wird es sehr oft als ein Optimierungsverfahren betrachtet. Wie gut die Vorhersage des neuronalen Netzes gerade ist, wird mit oder von der Fehlerfunktion auch Kostenfunktion genannt quantifiziert oder angegeben. Die Kostenfunktion bringt die Ausgabewerte des neuronalen Netzes mit den gewünschten Werten in Zusammenhang. Sie ist ein nicht-negativer Wert und je kleiner dieser Wert wird, desto besser

ist die Übereinstimmung desNNs. Der Gradient sagt wie die Netzparameter geeignet angepasst werden sollen und er wird durch die Berechnung aktueller Netzvorhersagen und der Fehlerfunktion berechnet. Basiert auf diesen Gradienten konvergiert die Kostenfunktion nach einigen Epochen gegen sein globales Minimum, sodass der Netzfehler bei Vorhersagen geringer ist.

Die meisten benutzten Kostenfunktionen sind die Kreuzentropie (*cross-entropy*, Gleichung 6.2)(CE) und die mittlere quadratische Fehler (*mean squared error*,Gleichung 6.1)(MSE). muss noch hinweisen wo die Formeln herkommen?

$Y := \{Y_1, \cdots, Y_n\}$:die tatsächlichen Werte

$\widehat{Y} := \{\widehat{Y}_1, \cdots, \widehat{Y}_n\}$:die Ausgabeweerte des neuronalen Netzes

$$MSE(Y, \widehat{Y}) = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2 \tag{6.1}$$

$$CE(Y, \widehat{Y}) = -\frac{1}{n} \sum_{i=1}^{n} (Y_i \log(\widehat{Y}_i) + (1 - Y_i) \log(1 - \widehat{Y}_i)) \tag{6.2}$$

Im Gegenteil zu CE Fehlerfunktionen,die sich nur auf Wahrscheinlichkeitsverteilungen anwenden lassen, können die MSE auf beliebige Werte angewendet werden. Nach Pavel et al.[7] ermöglicht die CE-Verlustfunktion besseres lokales Optimum zu finden als die MSE-Verlustfunktion und das soll daran liegen, dass das Training des MSE Systems schnell in einem schlechteren lokalen Optimum stecken bleibt, in dem der Gradient verschwand und somit keine weitere Reduzierung der Klassifizierungsfehler möglich ist.Im Allgemein ist die CE Kostenfunktion für die Klassifikationsprobleme und die MSE Fehlerfunktion für die lineare Regression-Probleme besser. kleine Experiment

6.2 Gradientenabstieg

Lernrate

Die Lernrate oder Schrittweite beim maschinellen Lernen ist ein Hyperparameter, der bestimmt, inwieweit neu gewonnene Informationen alte Informationen überschreiben[10]. Je nachdem wie die Lernrate gesetzt wird, werden bestimmte Verhalten beobachtet und sie nimmt sehr oft Werte zwischen 0.0001 und 0.4: Die Lernrate muss allerdings im Intervall $]0, 1[$ Werte annehmen, sonst ist das Verhalten des NN nicht vorhersehbar bzw. konvergiert das Verfahren einfach nicht. Für jeden Punkt x aus dem Parameterraum gibt es eine optimale Lernrate $\eta_{opt}(x)$, sodass das globale oder lokale Minimum sofort nach der Parameteranpassung erreicht wird und da $\eta_{opt}(x)$ am Trainingsanfang leider nicht bekannt ist, wird die Lernrate in die Praxis vom Anwender basiert auf seine Kenntnisse mit NNs oder einfach zufällig gesetzt.

- $\eta < \eta_{opt}$:So sind wir sichern, ein lokales oder globales Minimum zu erreichen.Aber die Anzahl der benötigten Iterationen bis zum Minimum steigt offensichtlich an und das Verfahren kann in einem unerwünschten lokalen Minimum stecken bleiben .

- $\eta > \eta_{opt}$:Hier wird die Anzahl der Iterationen zwar verringert, aber das Verfahren ist nicht stabil, denn in der Nähe vom Minimum oder es wird über das Minimum ständig hinausgegangen und es ist nicht mehr sichern, zum lokalen oder globalen Minimum zu gelangen.

In die Praxis gibt es Methoden und Funktionen, um die Lernrate während des Trainings anzupassen. z.B die *KERAS* Funktion *ReduceLROnPlateau*, die die Lernrate reduziert, wenn das NN nach einer bestimmten Anzahl von Epochen keine Verbesserung mehr aufweist. **Kleine Experiment** mit Größe und kleine Lernrate

- Definition von GA
- Funktionsweise von GA
  - Irgendwo anfangen
  - Richtung der steilsten Abstieg finden.
  - In Richtung der steilsten Abstieg laufen.
- Problem
  - Sattelpunkt ->Plateau
  - Ineffizient, wenn über den ganzen Trainingssatz(Berechnungszeit, Speicherplatzbedarf) -> lieber über Mini-Batch

6.3 Backpropagation: Optimizer

Das Ziel des Trainings tiefer neuronaler Netze ist, den Unterschied bzw. den Fehler zwischen Netzvorhersagen und tatsächlichen erwarteten Werten zu reduzieren ,also die Kostenfunktion zu minimieren und für jedes Maschine Lernen Problem gibt es Werte für Netzparameter(Gewichte und Bias) auch optimale Werte genannt, sodass das NN die Eingangdaten auf die gewünschten Werte korrekt abbildet. Mit korrekt wird gemeint, dass der Wert der Fehlerfunktion mit dieser optimalen Werten verschwindend klein oder sogar gleich null sei.

Gradientenverfahren

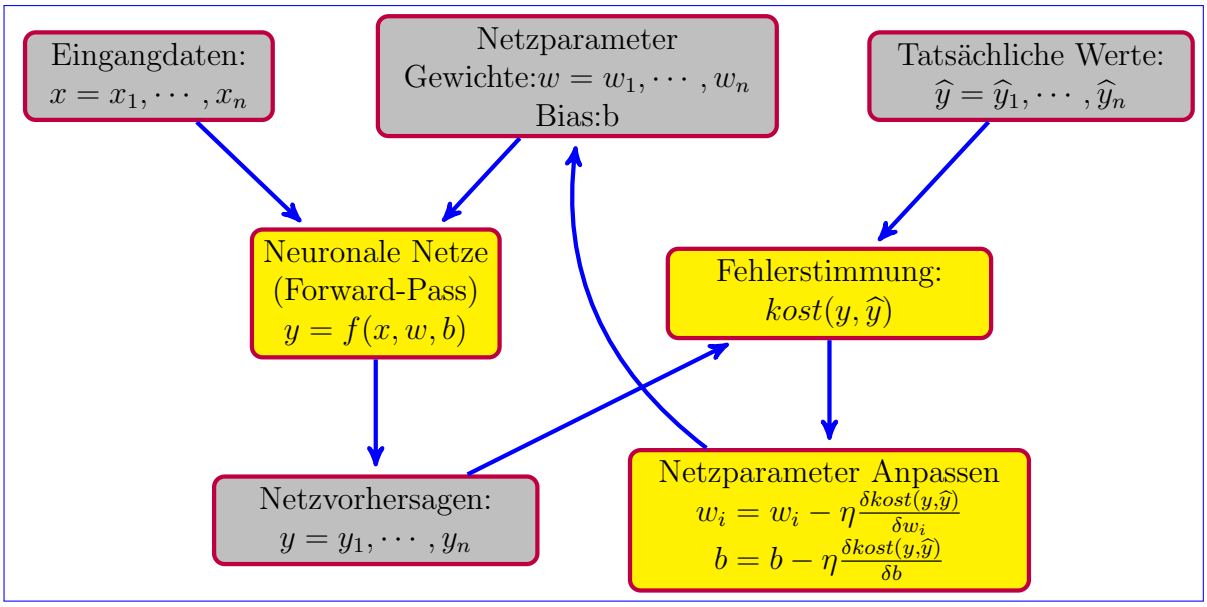
Warum wird Gradientenverfahren gebraucht

Zum Finden dieser optimalen Netzparameter können wir einfach mehrmals zufällige Werte versuchen. Aber dieser Lösungsansatz weist ein großes Problem auf: Angenommen soll das NN zehn optimale Netzparameter aus dem Intervall $]-1,1[$ finden und der Computer nur zwei Nachkommastelle (z.B 0.08) darstellen kann. Für den Wert eines Netzparameters bzw. die Werte der zehn Netzparameter gibt es also $2 * 10^2 - 1$ bzw. $(2 * 10^2 - 1)^{10}$ Möglichkeiten. also maximale $(2 * 10^2 - 1)^10$ Iterationen, um die optimalen Werte zu finden, was einfach zu viel und nicht akzeptabel ist. In die Praxis haben NNs Millionen von Parametern. Statt eines zufälligen Einsetzen von Netzparametern wird lieber ein Gradientenabstiegsverfahren(*Gradient Descent*) angewendet. Das Gradientenabstiegsverfahren ist ein Verfahren, bei dem die Richtung des Gradienten zu Nutze

gemacht wird,um der globale Extremwert einer ableitbaren Funktion zu erreichen.Im Maschinen Lernen wird Gradientenabstiegsverfahren verwendet, um an die optimalen Netzparameter anzunähern, besser gesagt die Fehlerfunktion zu minimieren.

Gradientenverfahren Funktionsweise

Das Gradientenabstiegverfahren funktioniert wie folgt:Es wird einen zufälligen Punkt aus dem Parameterraum ausgewählt, diese entspricht der Netzparameterinitialisierung am Trainingsanfang.Dann werden die Eingangdaten in das NN eingespeist(*Forwardpropogation*) und danach wird der Fehler zwischen den Netzvorhersagen und den korrekten Werten berechnet. Ein Fehler gibt es (fast) immer, denn die Initialisierung wird zufällig gemacht und die Wahrscheinlichkeit, dass wir von Anfang an die optimalen Werte finden, ist verschwindend klein.Der Gradient der Kostenfunktion wird in abhängig von den gegebenen Eingangdaten und den erwarteten Werten berechnet.wir orientieren die Netzparameter dem Gradienten entgegen(*Backward-Pass*), also in Richtung des Minimums, denn der Gradient zeigt immer in Richtung der höchsten Punkt einer Funktion an. Die Abbildung 16 stellt das Backpropagation-Verfahren bildlich dar.



$f(x, w, b)$: Netzfunktion. $kost(y, \widehat{y})$: Kostenfunktion.
η: Lernrate $\frac{\delta kost(y,\widehat{y})}{\delta w_i}$: Ableitung der Kostenfunktion abhängig von w_i.

Abbildung 16: Ablauf der Backpropagation

Pfeile in Abbildung 16 weisen nur den Prozessablauf hin.

Porblem

- Wahl der Lernrate: groß oder klein ?
- *Learning rate schedules:*Anpassung der Lernrate während des Trainings.Wahl der Hyperparameter vor Trainingsanfang

- gleiche Lernrate für alle Parameter. effizienter?
- in lokale Minima stecken bleiben

Type von Gradientenverfahren
Bisher existiert drei Variante des Gradientenabstiegsverfahren, die sich nur mit der Größe der Daten, die sie verwendet, um den Gradienten der Kostenfunktion berechnet, unterscheidet.Zum Aktualisierung der Netzwerkparameter nutzen sie jeweils die Gleichung (6.3)

$$\theta_{t+1} = \theta_t - \eta g_t, \quad g_t = \frac{\delta E}{\delta \theta_t} \tag{6.3}$$

η: Lernrate
θ_t: Netzwerkparameter zum Zeitpunkt t
E: Die Fehlerfunktion

- **Stochastic Gradient Descent:***SGD*
  Bei SGD wird jeweils ein Element bzw. Sample aus der Trainingsmenge durch das NN durchlaufen und den jeweiligen Gradienten berechnen, um die Netzwerkparameter zu aktualisieren.Diese Methode wird sehr oft *online training oder Verfahren* genannt,denn jedes Sample aktualisiert das Netzwerk. SGD verwendet geringer Speicherplatz und die Iterationsschritte sind schnell durchführbar.Zusätzlich kann die Konvergenz für großen Datensatz wegen der ständigen Aktualisierung der Netzwerkparameter beschleunigen.Diese ständigen Aktualisierung hat die Schwankung der Schritte in Richtung der Minima zur Folge, was die Anzahl der Iteration bis zum Erreichen des Minimums deutlich ansteigt und dabei helfen kann, aus einem unerwünschten lokalen Minimum zu entkommen.Ein großer Nachteil dieses Verfahren ist der Verlust der parallelen Ausführung, es kann jeweils nur ein Sample ins NN eingespeist werden.

- **Batch Gradient Descent:***BGD*
  BGD funktioniert genauso wie SGD, außer dass der ganze Datensatz statt jeweils ein Element aus dem Datensatz genutzt wird, um die Netzwerkparameter zu aktualisieren.Jetzt kann das Verfahren einfach parallel ausgeführt werden, was den Verarbeitungsprozess des Datensatzes stark beschleunigt. BGD weist weniger Schwankungen in Richtung der Minimum der Kostenfunktion als SGD auf, was das Gradientenabstiegverfahren stabiler macht.Außerdem ist das BGD recheneffizienter als das SGD, denn nicht alle Ressourcen werden für die Verarbeitung eines Samples, sondern für den ganzen Datensatz verwendet.BGD ist leider sehr langsam,denn die Verarbeitung des ganzen Datensatz kann lange dauern und es ist nicht immer anwendbar, denn sehr große Datensätze lassen sich nicht im Speicher einspeichern.

- **Mini-batch Stochastic Gradient Descent:***MSGD*
  MSGD ist eine Mischung aus SGD und BGD.Dabei wird der Datensatz in kleine

Mengen (*Mini-Batch oder Batch*) möglicherweise gleicher Größe aufgeteilt.Je nachdem wie man die Batch-Große setzt, enthalten wir SGD oder BGD wieder. Das Training wird Batchweise durchgeführt, d.h. es wird jeweils ein Batch durch das NN propagiert, der Verlust jedes Sample im Batch wird berechnet und dann deren Durchschnitt benutzt, um die Netzwerkparameter zu anzupassen.MSGD verwendet den Speicherplatz effizienter und kann von Parallelen Ausführung profitieren. Noch dazu konvergiert MSGD schneller und ist stabiler. In die Praxis wird fast immer das MSGD Verfahren bevorzugt.

Zum besserer Anwendung der Gradientenabstiegsverfahren wurden mehrere Optimierte Lernverfahren entwickelt. Im folgenden wird ein kurzer Einblick über die bekanntesten Lernverfahren(*Optimizer*) gegeben.
Alle heutige Optimizer haben SGD als Vorfahren und der Hauptnachteil von SGD ist , dass es die gleiche Lernrate für die Anpassung aller Netzwerkparameter verwendet und diese Lernrate wird auch während des Trainings nie geändert.

6.3.1 Adaptive Gradient Algorithm :AdaGrad

AdaGrad bietet während des Netztrainings nicht nur die Möglichkeit, die Lernrate zu verändern, sondern auch für jeden Parameter eine geeignete Lernrate zu finden. Die AdaGrad-Aktualisierungsregel ergibt sich aus der folgenden Formel:

$$\begin{aligned} \alpha_t &= \sum_{i=1}^{t} (g_{i-1})^2 \quad \theta_{t+1} = \theta_t - \eta_t g_t \\ \eta_t &= \frac{\eta}{\sqrt{\alpha_t} + \epsilon} \end{aligned} \tag{6.4}$$

$$\text{Voreingestellte Parameter}(KERAS): \quad \alpha_0 = 0.0 \quad \eta = 0.001 \quad \epsilon = 10^{-7}$$

Dabei wird am Trainingsanfang eine Lernrate für jeden Parameter definiert und im Trainingsverlauf separat angepasst. Dieses Verfahren eignet sich gut für spärliche Daten, denn es gibt häufig auftretende Merkmale sehr niedrige Lernraten und seltene Merkmale hohe Lernraten, wobei die Intuition ist, dass jedes Mal, wenn eine seltene Eigenschaft gesehen wird, sollte der Lernende mehr aufpassen. Somit erleichtert die Anpassung das Auffinden und Identifizieren sehr voraussehbarer, aber vergleichsweise seltener Merkmale.[11].Wie in der Gleichung (6.4) festzustellen,nach einer bestimmten Anzahl von Iterationen haben wir keine Verbesserung der Netzleistung, denn je größer t wird, desto kleiner η_t wird und irgendwann wird η_t so klein, dass $\eta_t g_t$ fast gleich null ist.

6.3.2 Root Mean Square Propagation:RMSProp

RMSProp wie AdaGrad findet für jeden Parameter eine geeignete Lernrate und zur Anpassung der Netzparameter basiert der RMSProp Optimizer auf den Durchschnitt der aktuellen Größen der Gradienten statt auf der Summe der ersten Moment wie in

AdaGrad.Da $E[g^2]_t$ nicht schneller als $\alpha_t(6.4)$ ansteigt, wird die radikal sinkenden Lernraten von Adagrad deutlich verlangsamt.Die Parameteranpassungen richten sich nach der folgenden Gleichung:

$$\begin{aligned} E[g^2]_t &= \alpha E[g^2]_{t-1} + (1-\alpha) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t, \quad \epsilon \approx 0 \end{aligned} \tag{6.5}$$

Der RMSProp funktioniert besser bei Online- und nicht-stationären Problemen.

- Vorteile
  - moving average of gradient
  - Es reduziert die radikal sinkenden Lernraten von Adagrad
- Nachteile
  -
  -

6.3.3 Adam Adaptive Moment Estimation:Adam

Der Adam[12] Optimizer ist auch ein adaptiver Algorithmus,der die ersten und zweiten Momente der Gradienten schätzt, um individuelle adaptive Lernraten für verschiedene Parameter zu berechnen. Adam weist die Hauptvorteile von AdaGrad, das mit spärlichen Gradienten gut funktioniert, und RMSProp, das einige Probleme von AdaGrad löst und das für nicht-konvexe Optimierung geeignet ist,auf.Wie die Parameteranpassung von Adam Optimizer genau funktioniert, ergibt sich aus der folgenden Gleichung:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad \widehat{m}_t = \frac{m_t}{1-\beta_1^t} \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \quad \widehat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \epsilon} \widehat{m}_t \end{aligned} \tag{6.6}$$

$$\text{Voreingestellte Parameter}(KERAS): \quad \begin{matrix} \beta_1\text{: } 0.9 & \beta_2\text{: } 0.999 \\ \eta\text{: } 0.001 & \epsilon\text{: } 10^{-7} \end{matrix}$$

Zu weiteren Vorteile der Nutzung von Adam gehört auch seine Einfachheit zur Implementierung, effizienter Nutzung der Speicherplatz und seine Invarianz zur diagonalen Neuskalierung der Gradienten.
Kleines Experiment

7 Model mit geringerer Rechenzeit und Speicherplatzbedarf

7.1 AlexNet

7.2 SqueezeNet

7.3 Xception

7.4 MobileNet

8 Effizienter Nutzung tiefer neuronaler Netze

Die neueren maschinellen Lernmethoden verwenden immer tiefer neuronale Netze wie z.B *Xception(134 Layers),MobileNetV2(157 Layers), InceptionResNetV2(782 Layers)*, um Ergebnisse auf dem neuesten Stand der Technik in verschiedenen Bereichen zu erzielen. Aber die Verwendung von sehr tiefer NNs bringt mit sich nicht nur eine deutliche Verbesserung der Modellleistung, sondern auch einen bedeutenden Bedarf an Rechenleistung und an Speicherplatz, was der Einsatz solcher Modelle auf Echtzeitsystemen mit begrenzten Hardware-Ressourcen schwierig macht.Es wurden bisher mehrere Ansätze untersucht, um die dem NN zugewiesenen Ressourcen effizienter zu nutzen.Modellbeschneidung(*Network pruning*), die die redundanten und die nicht relevanten Netzparameter entfernt.Destillation von NNs, die ermöglicht, die großen Modellen in kleinen zu komprimieren.Quantisierung von NN, die für die Darstellung von einzelnem Netzparameter weniger als 32 Bits nutzt.Im folgenden werden nur die Beschneidung und Quantisierung von NN mehr eingegangen werden.

Neuronale Netze sind sowohl rechenintensiv als auch speicherintensiv, was ihre Bereitstellung auf eingebetteten Systemen mit begrenzten Hardware-Ressourcen erschwert[5]. Um dem Problem von Rechenzeit und Speicherplatzbedarf entgegenzuwirken, wird die tiefe Kompression(*Deep Compression*) Technik von [5, Han et al] eingeführt. Die Deep Compression Technik besteht aus drei Phasen:Netzwerkbereinigung (*Pruning Network*),Quantizierung(*Quantization*) und Huffman-Codierung(*Huffman Coding*)

8.1 Pruning Network

Wie oben schon erwähnt, wird beim *Pruning* neuronaler Netze versucht, unwichtige oder redundante Parameter oder komplette Neuronen aus dem Netz zu entfernen, um ein Netz mit möglichst geringer Komplexität zu erhalten. Mit unwichtigen Parametern werden die Parameter (Gewichte und Bias) gemeint, die fast null sind, denn Parameter mit Wert null beeinflussen das Output des Neurons nicht mehr und sind einfach überflüssig .Da fast keine Parameter nach Netztraining genau gleich null sind, wird einen Schwellenwert entweder fixiert oder bestimmt, um zu entscheiden, welche Parameter als null bzw. unwichtig betrachtet werden.Das Pruning reduziert die Anzahl der Parameter, was die

Netzkomplexität, die Rechenzeit und auch die Wahrscheinlichkeit des Overfitting reduziert.Für AlexNet und VGG-16 Modell wird durch Pruning die Anzahl der Parameter um 9 bzw. 13 mal reduziert[5].Zur effizienteren Speicherung des beschnitten Netzes kann ein CRS (*Compressed Row Storage*) oder CCS (*Compressed Column Storage*) Format verwendet werden, das $2a + n + 1$ statt $n * n$ Zahlen speichert, wobei a die Anzahl der Elemente ungleich Null und n die Anzahl der Zeilen oder Spalten der Matrix ist.

Das Pruning-Verfahren kann per Hand(also durch festlegen von Hyperparametern vor Trainingsbeginn) oder automatisch(die Hyperparameter werden während des Trainings gelernt) durchgeführt werden. Erstmals von [5, Han et al.] vorgeschlagen, wird das Pruning-Verfahren per Hand durchgeführt und dabei wird vor dem Netztraining einen Schwellenwert(*Threshold*) für alle Layers fixiert.Obwohl dieses Vorgehen gute Ergebnisse aufweist, hat es Nachteile, die nicht unberücksichtigt lassen werden können:Einerseits muss das Netz mehrmals erneut trainiert werden, nur um den Schwellenwert anzupassen und anderseits ist die Anzahl diese Iterationen eingeschränkt.In Abbildung 17 ist der Ablauf des Pruning-Verfahrens dargestellt. Dieses Verfahren kann also zu einer nicht optimalen Konfiguration führen. Um dieser Probleme entgegenzuwirken, haben [6, Manessi et al.] das Pruning-Verfahren automatisiert. Manessi et al.[6] haben die Beschnei-

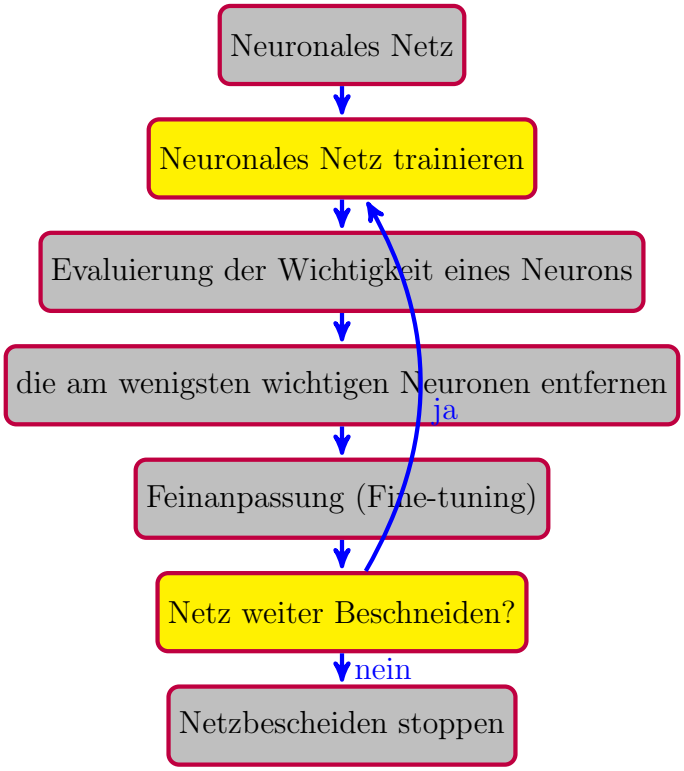


Abbildung 17: Ablauf der Netzbeschneidung (*Pruning Network*)

dungsmethode verbessert, indem sie das Verfahren in Bezug auf die Schwellenparameter differenzierbar gemacht haben, in anderen Worten sind die Schwellenwerte wie Gewichte und Bias lernbare Netzparameter. Dadurch können während der Lernphase automatisch

die besten Schwellenparameter zusammen mit den Netzwerkgewichten geschätzt werden, was die Trainingszeit stark verkürzt.Mit dieser Verbesserung kann mehr Verbindungen oder komplette Neuronen entfernt, denn statt eines globalen Schwellenwerts wird einen Schwellenwert für jede Schicht berechnet.[6] muss ich hier die Formeln eingeben?

8.2 Quantisierung von NN

Die Quantisierung [13] bezieht sich auf den Prozess der Reduzierung der Anzahl der Bits,die für Darstellung einer Zahl notwendig sind. Im Bereich des *Deep Learning* ist das Standard numerische Format für Forschung und Einsatz bisher 32-Bit Fließkommazahlen oder FP32, denn es bietet ein bessere Genauigkeit, aber die anderen Formaten wie 8- ,4-,2- oder 1-Bits werden auch verwendet, obwohl sie mehr oder weniger ein Verlust an Genauigkeit aufweisen.

Die Verwendung von weniger genauen numerischen Formaten hat nicht nur einen kleinen Verlust der Netzleistung zur Folge, sondern auch die Verwendung von deutlich reduzierter Bandbreite und Speicherplatz. Noch dazu beschleunigt die Quantisierung die Berechnungen, denn die ganzzahlige Berechnung zum Beispiel ist schneller als die Fließkommaberechnung.

Die Quantisierung ist eigentlich nur die Abbildung eines großen Bereiches auf einen kleinen und dazu werden zwei Hauptwerte benötigt: der dynamische Bereich des Tensors und ein Skalierungsfaktor. Angenommen haben wir einen dynamischen Bereich $[0, 500]$ und einen Skalierungsfaktor:5, dann ergibt sich der neue Bereich $[0, 100]$, es wird also Werte zwischen $[5k, 5(k+1)]$ oder $[5k-0.5, 5k+0.5]$ auf $5k$ abgebildet.Es ist sinnvoller, die Skalierungsfaktors unter Berücksichtigung der Anzahl und der Verteilung der Werte in dynamischen Bereich des Tensors auszuwählen.

Im Allgemein wird einen Skalierungsfaktor für jeden Tensor jeder Schicht berechnet und diese kann offline oder online gemacht werden.Bei der *Offline* Berechnung werden vor der Bereitstellung des Modells einigen Statistiken gesammelt, entweder während des Trainings oder durch die Ausführung einiger Epochen auf dem trainierten FP32-Modell und basierend auf diesen Statistiken werden die verschiedenen Skalierungsfaktoren berechnet und nach der Bereitstellung des Modells festgelegt.Durch die Anwendung dieser Methode lauft man die Gefahr, dass zur Laufzeit die Werte, die außerhalb der zuvor beobachteten Bereiche auftreten, abgeschnitten werden, was zu einer Verschlechterung der Genauigkeit führen kann.Bei der *online* werden die *Min/Max*-Werte für jeden Tensor dynamisch zur Laufzeit berechnet. Bei dieser Methode kann es nicht zu einer Beschneidung kommen, jedoch können die zusätzlichen Rechenressourcen, die zur Berechnung der Min/Max-Werte zur Laufzeit benötigt werden, unerschwinglich.[13]

Es gibt zwei Arten und Weisen, wie die Quantisierung durchgeführt wird.Die erste ist das vollständige Training eines Modells mit einer gewünschten niedrigeren Bit-Genauigkeit(kleiner als 32 Bits).Die Quantisierung mit sehr geringer Genauigkeit ermöglicht ein potenziell schnelles Training und Inferenz, aber der Hauptproblem mit diesem Ansatz ist, dass Netzparameter nur bestimmte Werte annehmen können, so ist

die Aktualisierung der Netzparameter bzw. das Backpropagation nicht mehr wohldefiniert. Das zweite Szenario quantisiert ein trainiertes FP32-Netzwerks mit einer geringeren Bit-Genauigkeit ohne vollständiges Training.Eine aggressive Quantisierung hat im Allgemein einen negativen Einfluss auf die Netzleistung und um diesen Leistungsabfall zu überwinden, wird sehr oft auf Methoden wie das erneuerte Training des Netz nach der Quantisierung, die gleichzeitige Verwendung von verschiedenen Formaten oder die uneinheitliche Quantisierung zurückgegriffen.

Vor kurzem haben [14, Yoni et al] einen neue Ansatz für die Quantisierung vorgeschlagen,der das lineare Quantisierungsproblem als *Minimum Mean Squared Error* (MMSE) Problem löst und der nicht nur die 4-Bit(INT4) Quantisierung von schon trainierten Modellen ohne ein neues Training des Modells, sondern auch die Einsparung von Chipfläche (*chip area*) ermöglicht

- die vorgeschlagene Methode liefert Ergebnisse auf dem neuesten Stand der Technik bei minimalem Verlust der Genauigkeit der Aufgaben.

[14]

8.3 Huffman Coding

9 overfitting in neuronale Netzwerke

9.1 Was ist Overfitting

Wenn ein von der Maschine gelerntes Modell zu gut auf die Trainingsdaten abgestimmt ist und sehr schlechte Vorhersagen über Daten macht, die es bisher nicht gesehen hat, wird gesagt, dass das Modell an Überanpassung(Overfitting) leidet, anders gesagt, das Modell war nicht in der Lage, die relevanten Merkmale aus den Trainingsdaten zu verallgemeinern, sondern die ganzen Trainingsdaten auswendig zu lernen.Die irrelevanten Informationen aus den Trainingsdaten können z.B die Position des Tellers, wenn man Essen klassifizieren sollte, sein. Im folgenden werden einige Mittels vorgestellt, um mit Overfitting umzugehen.

9.2 Strategie gegen Overfitting

9.2.1 Data Augmentation

Ein großer Datensatz ist entscheidend für die Leistung tiefer neuronaler Netze.Dass ein Datensatz groß oder ausreichend für das Training eines neuronalen Netzwerk ist, hängt nur von der Größe des neuronalen Netzes ab und da die NNs, die die besten Leistungen aufweisen, Millionen von Parametern haben, ist fast unmöglich für jedes ML-Problem ausreichende Daten zu finden.Anstatt immer neue Daten zur Verbesserung der Netzleistung zu sammeln, können wir die Leistung des Modells verbessern, indem wir die bereits vorhandenen Daten erweitern.
Die populären Techniken oder Transformationen zur Vermehrung des Datensatzes sind

die horizontalen und vertikalen Spiegelungen, Drehungen, Skalierungen, Zuschneiden, Parallelverschiebungen und die Gauß'sches Rauschen. Für diese Arbeit habe ich zwei Ansätze im Gebrauch gehabt, um den Datensatz zu erhöhen: Der erste Ansatz besteht darin vor dem Training neue Daten zu erzeugen. Dabei werden die oben erwähnten Techniken vor dem Training angewendet, um zum Trainingszeitpunkt und zur Testzeit einen großen Datensatz zu haben und die originalen Daten werden zur Validierung verwendet.Der zweite Ansatz besteht darin, zum Trainingszeitpunkt und zur Testzeit die neuen Daten zu erzeugen. Hier haben wir keinen Datensatz größer als den originalen, aber die Daten, die ins Netzwerk eingespeist werden, ändern sich ständig.Angenommen wir die Möglichkeit haben, alle diese Transformationen durchzuführen, dann kann es vorkommen, dass eine Photo während der ersten Epoche horizontal gespiegelt wird und während der zweiten um 20 Grad gedreht wird, umso weiter.Zur Implementierung des zweiten Ansatzes bietet *KERAS* Framework die Funktion *ImageDataGeneerator*. Das interessanteste an *ImageDataGeneerator* ist, dass es mehrere Transformationen gleichzeitig anwenden

Imagedatagenarator

Neuronale Netzwerke können umso effektiver sein, je mehr Daten sie zur Verfügung

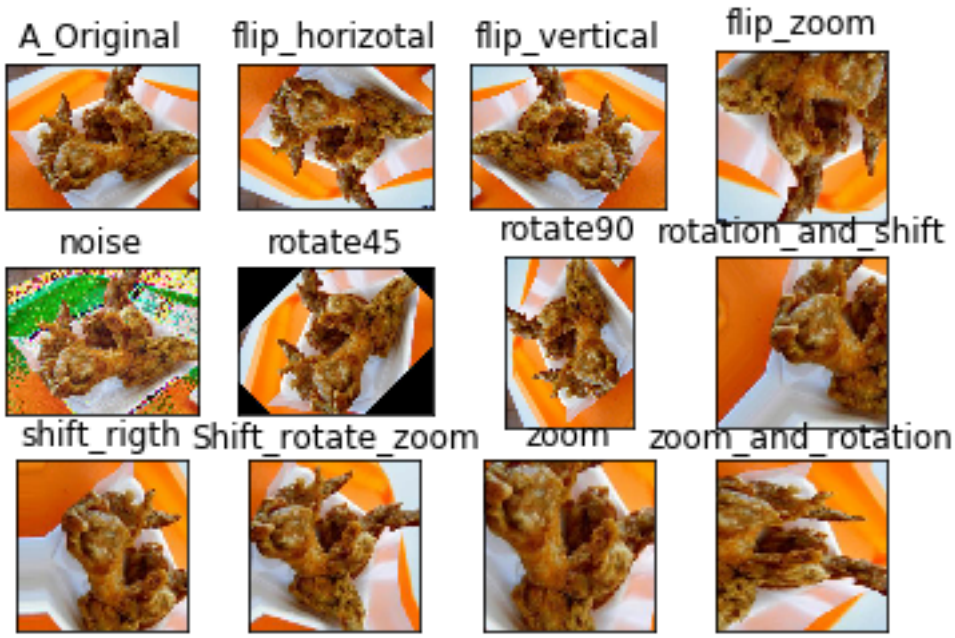


Abbildung 18: Anwendung von *ImageDataAugmentation*

haben. Es ist also mehr als wichtig über eine große Datenmenge zu verfügen

Die Datenvermehrung ist eine Strategie, die es Praktikern ermöglicht, die Vielfalt der für Trainingsmodelle verfügbaren Daten deutlich zu erhöhen, ohne tatsächlich neue Daten zu sammeln.

Selbst wenn die Daten von geringerer Qualität sind, können Algorithmen tatsächlich besser funktionieren, solange nützliche Daten durch das Modell aus dem ursprünglichen Datensatz extrahiert werden können.
Der erste Ansatz ist die Generierung erweiterter Daten vor dem Training des Klassifikators. Aber bestehende Datensätze reichen vielleicht nicht aus, um ein tiefgehendes Lernnetzwerk zu trainieren. Aufbau eines leistungsfähigen Klassifikators aus unzureichenden Daten, Datenerweiterungsmethoden sind nützlich.
Natürlich, wenn Sie viele Parameter haben, müssten Sie Ihrem Modell des maschinellen Lernens eine proportionale Anzahl von Beispielen zeigen, um eine gute Leistung zu erzielen. Außerdem ist die Anzahl der benötigten Parameter proportional zur Komplexität der Aufgabe, die Ihr Modell zu erfüllen hat.
Naturellement, si vous avez beaucoup de paramètres, vous aurez besoin de montrer à votre modèle d'apprentissage machine un nombre proportionnel d'exemples, pour obtenir de bonnes performances. De plus, le nombre de paramètres dont vous avez besoin est proportionnel à la complexité de la tâche que votre modèle doit effectuer.
Dies ist im Wesentlichen die Voraussetzung für die Datenvermehrung. Im realen Szenario haben wir möglicherweise einen Datensatz von Bildern, die unter einer begrenzten Anzahl von Bedingungen aufgenommen wurden. Unsere Zielanwendung kann jedoch unter einer Vielzahl von Bedingungen existieren, wie z.B. unterschiedliche Ausrichtung, Position, Maßstab, Helligkeit usw. Wir tragen diesen Situationen Rechnung, indem wir unser neuronales Netzwerk mit zusätzlichen synthetisch modifizierten Daten trainieren.

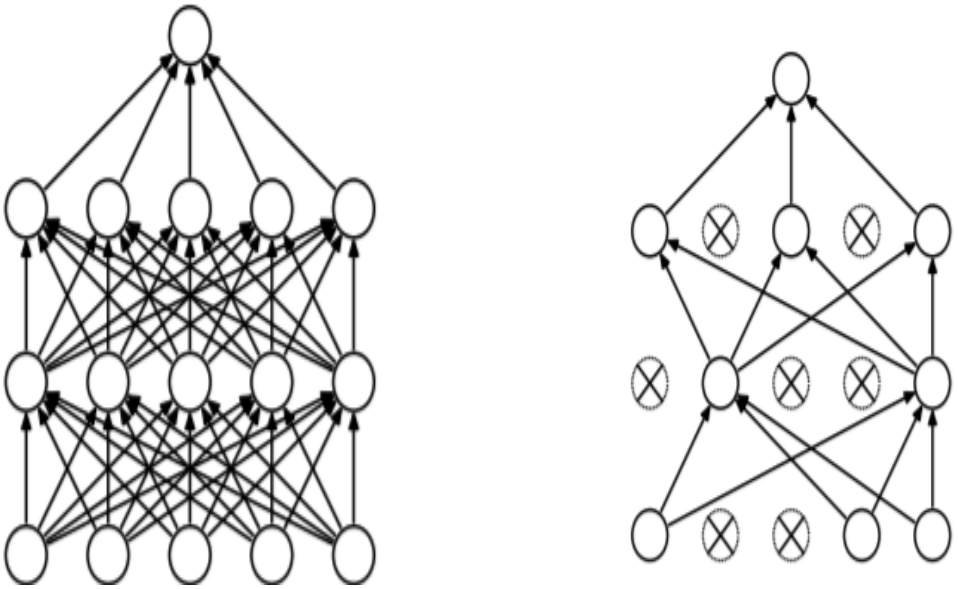
9.2.2 Dropout

Künstliche Neurone sind von biologischen Neuronen inspiriert, aber die Beiden unterscheidet sich sehr voneinander und einer der wichtigen Unterschiede ist, dass biologische Neuronen unvollkommene Maschine sind, die sehr oft nicht richtig funktioniert und das ist a priori nie den Fall bei biologischen Neuronen. Wir könnten also glauben, dass Künstliche neuronale Netze (KNN) die biologische übertreffen könnten.Es sei denn, dass diese Funktionsstörung von biologischen Neuronen nicht eine Schwäche ist, sondern eher eine Stärke ist.Eine der verblüffenden Entdeckungen in Künstliche Intelligenz (KI) Bereich ist, dass es wünschenswert ist, künstliche Neuronen von Zeit zu Zeit zu Fehlfunktionen zu bringen[1]. Jetzt können wir uns fragen, wie Dysfunktion von Neuronen die Performances neuronaler Netze verbessern kann.Die zufällige Hinzufügen von Dysfunktionen in einer Schicht der KNN wird *Dropout* benannt und wurde von [2, Geoffrey E. et al] eingeführt.

Wie funktioniert Dropout

Genauer gesagt,Dropout bezeichnet die zeitliche zufällige Ausschaltung von Neuronen in einer (sichtbaren oder versteckten)Schicht der KNN. Wie die Abbildung 19 zeigt, wenn ein Neuron zufällig aus dem NN entfernt wird, werden auch all seine ein- und ausgehenden Verbindungen entfernt. In einer Dropout-Schicht wird ein Neuron n unabhängig von andren Neuronen mit einer Wahrscheinlichkeit p zurückgehalten, d.h n wird mit einer Wahrscheinlichkeit von p entfernt.Wie eine NN mit Dropout ausgewertet wird, wird in

Abbildung 20 veranschaulicht.[3]



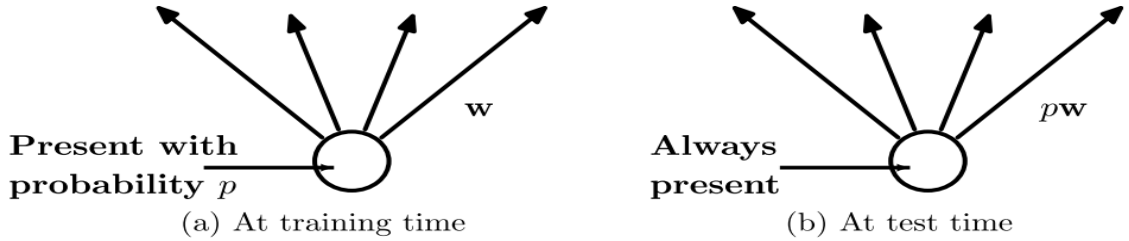
(a) Standard neuronale Netze (b) Netze nach Dropout

Abbildung 19: Neuronales Netz mit Dropout [3]

9.2.3 Vor- und Nachteile von Dropout

Automatische Erhöhung von Training Data

Die Ausschaltung von Neuronen führt nicht nur dazu, dass jedes Neuron unabhängig von Anderen Mustern korrekt lernt, sondern auch dazu, dass neue Trainingsdaten automatisch erzeugt werden.sollte ich auch die mathematische Formel Hinzufügen?.Nehmen wir an, dass wir eine Dropout-Schicht D zwischen zwei ConvLs $C1$ und $C2$, so wird die Outputs O_{c1} von $C1$ zuerst von D umgewandelt, bevor sie in $C2$ einmünden.Die von D ausgeführte Transformation ergibt $O_d = O'_{c1} + R$, wobei O'_{c1} Teilinformationen aus O_{c1} und R Informationen, die als Rauschen interpretieren werden können, bezeichnet. $C2$ soll also aus O'_{c1} O_{c1} trotz R gut interpretieren können. Für ein Netz mit n versteckten Einheiten, von denen jede fallen gelassen werden kann, haben wir durch Dropout 2^n mögliche Modelle. Das am Ende der Trainingszeit enthaltene Netz ist also eine durchschnittliche Approximation der 2^n mögliche verschiedene Modellen. In der Testphase wird das gesamte Netzwerk betrachtet und jede Aktivierung um einen Faktor p reduziert. Durch die Anwendung von Dropout wird die Trainingszeit für jede Epoche geringer, aber die Anzahl der Iterationen, die für die Konvergenz notwendig sind, wird leider mindestens verdoppelt. Dropout wendet also automatisch eine robuste Optimierung der neuronalen Netzwerkparameter an, die auch der Implementierung des *Ockhams* Rasiermessers entspricht.



(a) At training time (b) At test time

Abbildung 20: **Links**: Ein Neuron zur Trainingszeit, die mit Wahrscheinlichkeit p vorhanden ist und mit Neuronen in der nächsten Schicht mit Gewichten w verbunden ist. **Recht**: Zur Testzeit ist das Neuron immer vorhanden und die Gewichte werden mit p multipliziert. Die Ausgabe zur Testzeit ist identisch mit der erwarteten Ausgabe zur Trainingszeit[3].

9.2.4 Vergleich Ergebnisse

9.3 Batch-Normalisierung

Das Training tiefer neuronaler Netze ist sehr kompliziert und ein Grund dafür ist zum Beispiel die Tatsache, dass die Parameter einer Schicht während des Trainings tiefer neuronaler Netze immer unter der Annahme, dass sich die Parameter anderer Schichten nicht ändern, aktualisiert werden und da alle Schichten während des Updates geändert werden, verfolgt das Optimierungsverfahren ein Minimum, das sich ständig bewegt. Ein anderer Grund dafür ist die ständigen Veränderungen im Laufe des Trainings in die Verteilung des Netzinputs, diese Veränderung wird von [9] als interne kovariate Verschiebung(*Internal Covariate Shift*) genannt.Zur Lösung dieser Probleme schlagen *LeCunn et al*[**?**] vor dem Training das Netzinput zu normalisieren. Aber dieser Ansatz bringt nicht so viel, wenn das NN wirklich tief ist, denn nur der Netzinput profitiert von der Normalisierung und die kleinen Veränderungen in versteckte Schichten werden sich immer mehr verstärken, je tiefer man das Netz durchläuft. Mit der Ausbreitung tiefer NNs dehnt Batch-Normalisierung(BN)[9] diese Idee der Datennormalisierung auf versteckte Schichten tiefer NNs aus. Bei der BN werden die Eingaben in einem Netzwerk standardisiert, die entweder auf die Aktivierungen einer vorherigen Schicht oder auf direkte Eingaben angewendet wird, so standardisiert, dass der Mittelwert in der Nähe von null liegt und die Standardabweichung in der Nähe von eins liegt. Die BN wird über Mini-Batches und nicht über den gesamten Trainingssatz durchgeführt, daher enthalten wir nur Näherungen an tatsächliche Werte der Standardabweichung und des Mittelwerts über das Trainingssatzes, aber wir gewinnen an Geschwindigkeit und an Speicherplatzverbrauch.Die Gleichung (9.1) gibt die formale Beschreibung des BN Algorithmus an.

Batch-Normalisierungstransformation, angewendet auf Aktivierung x über einen Mini-Batch

Input: Werte von x über einer Mini-Batch: $B = \{x_{1...m}\}$
Lernbare Parameter β, γ
Output: $\{y_i = BN_{\beta,\gamma}(x_i)\}$

$$\text{Mini-Batch Mittelwert} : \mu_\beta = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{9.1a}$$

$$\text{Mini-Batch Standardabweichung} : \sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2 \tag{9.1b}$$

$$\text{Normalisierung}: \widehat{x_i} = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \tag{9.1c}$$

$$\text{Skalierung und Verschiebung} : y_i = \gamma\widehat{x_i} + \beta \equiv BN_{\gamma,\beta} \tag{9.1d}$$

Wenn $\gamma = \sqrt{\sigma_\beta^2 + \epsilon}$ und $\beta = \mu_\beta$, bekommen wir die gleiche Verteilung wie vor der Batch-Normalisierung, d.h die Eingabe war also schon normalisiert. Interessanterweise kann das Netz während des Trainings eine bessere Verteilung als die erwünschte finden, denn γ und β sind lernbare Parameter.

Durch die BN kann zum einen eine hohe Lernrate verwendet,was in tiefer NNs ohne BN dazu führen kann, dass die Gradienten explodieren oder verschwinden und in schlechten lokalen Minima stecken bleiben.Die Verwendung einer höhere Lernrate ermöglicht einer schnellere Konvergenz. Zum anderen wird die interne kovariate Verschiebung geringer, was das Training beschleunigt, in einigen Fällen durch Halbierung der Epochen oder besser. Noch dazu wird das Netz durch die BN in gewissem Maße reguliert, daher wird die Verwendung von Dropout bzw. Regulierungstechnik reduziert oder sogar überflüssig und somit eine Verbesserung der Verallgemeinerungsgenauigkeit.

10 Experiment

11 Abkürzungsverzeichnis

KNN Künstliche neuronale Netze

CNN Convolutional Neural Network

KI Künstliche Intelligenz

NN neuronale Netze

ConvL Convolutional Layer

FCL Fully Connected Layer

Literatur

[1] P. Kerlirzin, and F. Vallet Robustness in Multilayer Perceptrons

[2] Geoffrey E. Hinton and Nitish Srivastava and Alex Krizhevsky and Ilya Sutskever and Ruslan R. Salakhutdinov Improving neural networks by preventing co-adaptation of feature detectors

[3] Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville Adaptive Computation and Machine Learning series Page 342

[5] Song Han, Huizi Mao, William J. Dally Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

[6] Franco Manessi, Alessandro Rozza, Simone Bianco, Paolo Napoletano, Raimondo Schettini Automated Pruning for Deep Neural Network Compression

[7] Pavel Golik , Patrick Doetsch, Hermann Ney Cross-Entropy vs. Squared Error Training:a Theoretical and Experimental Comparison

[8] Neuronale Netze:Eine Einführung

[9] Sergey Ioffe, Christian Szegedy Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift

[10] Learning Rate

[11] John Duchi,Elad Hazan, Yoram Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization

[12] Diederik P. Kingma, Jimmy Ba Adam: A Method for Stochastic Optimization

[13] Compressing Models:Quantization

[14] Yoni Choukroun, Eli Kravchik, Fan Yang, Pavel Kisilev: Low-bit Quantization of Neural Networks for Efficient Inference

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, sowie die Satzung der Universität Augsburg zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Ort, den Datum