|  |  |
| --- | --- |
| **AI Applications** | |
| Grundbegriffe | |
| Komponenten von Machine Learning | |
| Beim Machine Learning geht es darum, eine unbekannte, «reale» Zielfunktion mithilfe von bekannten Daten über eine Modellfunktion zu approximieren. | |
| Unbekannte Zielfunktion  Bekannte Daten | |
|  | Finales Modell |
| Modellfamilie   * Der Learning-Algorithmus beinhaltet Loss & Optimizer. * Die Modellfamilie wird auch als «Hypothesis Set» bezeichnet. | |
| Artificial Neural Networks | |
| Artificial Neurons | |
| Ein Artificial Neuron ist eine «einfache» Recheneinheit bestehend aus:   * Einem Inputvektor . * Einem Gewichtevektor . * Ein Bias / Intercept . * Eine nicht-lineare Aktvierungsfunktion .   Jedes Neuron berechnet das Skalarprodukt von und , addiert und sendet den Wert durch die Funktion .   * Das und sind die trainierbaren Parameter. | |
|  | |
| Aktivierungsfunktion | |
| Die Aktivierungsfunktion ist eine beliebige nicht-lineare mathematische Funktion, welche man «frei» wählen kann.   * Die Wahl von beeinflusst die Qualität des Modells. | |
| The Top 12 Activation Functions, a Short Guide for Choosing - Metacoder | |

|  |  |  |  |
| --- | --- | --- | --- |
| Grenzen von Neuronen | | | |
| Sieht man sich die Aktivierungsfunktionen genauer an, so fällt auf, dass ein einzelnes Neuron nur linear separierbare (binäre) Muster erkennen kann.   * Vergleiche «Sigmoid-Funktion» mit der linearen Regression. * Dies gilt auch bei Daten in höheren Dimensionen. * XOR ist ein bekanntes, nicht linear separierbares Problem. | | | |
| Solving XOR Problem using neural network – C# – Tech-Quantum | | | |
| Artificial Neural Networks (ANN) | | | |
| Die Probleme eines einzelnen Neurons lassen sich lösen, indem wir mehrere, «hintereinander-gestapelte» Neuronen verwenden, sogenannte «Layers».   * Wir sprechen dann auch von einem «Multi Layer Perceptron» * Ein «Perceptron» bezeichnet ein einzelnes Neuron. | | | |
|  | | | |
| Das Gebiet von Machine Learning wird aktuell von komplexen ANNs mit Millionen von Parametern dominiert. Wir sprechen daher oft auch von sogenannten «Deep Learning» Architekturen.   * Terminologie: Ein ANN implementiert eine Modellfamilie. * Einfachere Modelle sind oftmals weniger ausdrucksstark. | | | |
| **Backpropagation** | | | |
| Expression Trees | | | |
| Ein «Expression Tree» ist eine Datenstruktur für die Darstellung und Berechnung von mathematischen Ausdrücken.   * Die Implementation davon ist für dieses Modul nicht relevant. | | | |
| Grundbausteine | | | |
| Ein «Expression Tree» besteht immer aus den gleichen Grundbausteinen: | | | |
|  |  |  |  |
|  |  |  |  |
| * Wobei die «» Regel auch für , , und gilt. | | | |
| Erklärungsbeispiel | | | |
| Wir können einen Expression Tree für die Formel «» zeichnen.   * Wie wir sehen werden, sind Expression Trees nicht eindeutig. | | | |

|  |
| --- |
|  |
| Backpropagation |
| Der «Backpropagation»-Algorithmus ist eine effiziente Methode für die Berechnung der Gradienten einer Funktion.   * Die meisten «Optimizer» in ML sind Gradienten-basiert. * Ohne Backpropagation gäbe es also kein Machine Learning. |
| Erklärungsbeispiel |
| Betrachten wir ein Modell in der Form:  Wir verwenden nun den MSE als Loss und SGD als unser Optimizer.      Wie wir sehen, müssen wir in jedem Optimierungsschritt den Gradienten der Loss-Funktion bestimmen.   * Dazu müssen wir also jeweils alle Ableitungen berechnen. |
| Aufbau des Expression Trees |
| Rechnen wir die einzelnen Ableitungen im Gradienten aus, so erhalten wir:        Wir erkennen also, dass alle Ableitungen einen identischen Teil beinhalten.   * Wir können dies in einem Expression Tree darstellen. |
|  |
| Anwendung von Backpropagation |
| Wir können diesen Baum nun mithilfe von Backpropagation auswerten.   1. **Forward-Pass:** Als erstes Werten wir von «unten nach oben» alle Teilbäume bis zum benötigten Endresultat aus. 2. **Backward-Pass**: Anschliessen gehen wir «zurück» und passen die Parameter an.  * Backpropagation basiert dabei auf «Dynamic Programming». |

|  |  |  |
| --- | --- | --- |
| **Convolutional Neural Networks (CNN)** | | |
| Anwendungsgebiet | | |
| Die Bildverarbeitung ist eines der bekanntesten Einsatzgebiete von Neuralen Netzwerken. ANNs ermöglichen es z.B. beliebige Bilder zu klassifizieren. | | |
|  | | |
| Probleme von ANNs | | |
| Um ein Bild in einem «normalen» ANN verwenden zu können, müssen wir es in einen Vektor umwandeln.   * Diesen Prozess nennt man «Flattening». | | |
| Convolutional Neural Networks (CNN): Step 3 - Flattening - Blogs -  SuperDataScience | Machine Learning | AI | Data Science Career | Analytics  | Success | | |
| Besonders bei Bildern gehen bei diesem Prozess aber die «Muster» der Daten wie Formen und Strukturen verloren.   * Ein ANN kann diese Muster nur mit Mühe wiederherstellen. | | |
| Convolutional Feature Detectors (CFD) | | |
| Um dieses Problem zu lösen, versucht man anstelle von «Flattening» die Formen und Strukturen eines Bildes mithilfe von «Convolution» zu erkennen.   * Diese Idee ist wie so oft von unserem Hirn inspiriert. * Die erkannten Strukturen sind dann unsere «Features». | | |
| Bild  Kernel  Features | | |
| Funktionsweise | | |
| Die Convolution erkennt die Strukturen eines Bildes, indem mehrere «Kernel» über das Bild «geschoben» werden. An jeder Position werden dann die Features berechnet und in einer Map gespeichert.   * Kernel und Bias sind die trainierbaren Parameter. * Mehrere Layer können auch detaillierte Features erkennen. | | |
| Feature Maps | | |
| Wurden die Kernel sinnvoll trainiert, so lassen sich aus den «Feature Maps» die Muster der Daten ableiten.   * Eine Feature Map zeigt also die «Stärke» eines Features an. * Wir können pro Layer auch mehrere Feature Maps erstellen. | | |
|  | | |
| Berechnungen Kernel | | |
|  | | |
|  |  |  |
| * Die Summe der Produkte der einzelnen Komponenten. * Resultat vom Kernel, Inputdaten, Kernel | | |
| Ein Bild, das Tisch enthält.  Automatisch generierte BeschreibungBerechnungen Feature Map | | |
|  | | |
| * Nun werden die Kernels verschoben und erneut berechnet. * Normalerweise wenden wir hier die Aktivierungsfunktion an. * Feature an einer bestimmten Position | | |
| Max / Average Pooling | | |
| In vielen Fällen ist es sinnvoll, die berechneten Daten in einer Feature Map weiter zu reduzieren. Wir können dafür das sogenannte «Pooling» verwenden:   * Max-Pooling: Nimm den grössten Wert in einem definierten Bereich. * Average-Pooling: Nimm den Durchschnitt aller Werte in einem Bereich. * Andere Pooling-Mechanismen existieren, sind aber selten. * Wir können so die Anzahl trainierbaren Parameter reduzieren. | | |
| Max Pooling Explained | Papers With Code | | |
| Trotz der Datenreduktion bleibt die Performance des Modells meistens gleich wie bei einem Modell ohne «Pooling».   * Wir können also komplexere ANNs mit weniger Daten bauen. | | |
| Strides | | |
| Der sogenannte «Stride» definiert die Distanz, mit welcher ein Kernel über das Bild «geschoben» wird.   * Kann auch beim Pooling angegeben werden. Standard ist 1. | | |
|  | | |
| Vorteile von CFDs | | |
| Die CFDs haben diese Vorteile:   * Translation Invariance: Einzelne Features können überall im Bild erkannt werden. | | |

|  |  |
| --- | --- |
| * Geteilte Gewichte: Durch teilen der Kernels verringert sich der Berechnungsaufwand. * Parallelisierbar: Die Berechnung der Kernel kann komplett parallelisiert werden. * «Translation Invariant» bedeutet «Positionsunabhäng» | |
| Anwendungsbeispiele | |
| «Alex Net» | |
| Alex Net gilt als eine der einflussreichsten Arbeiten im Bereich von «Computer Vision» und hat seit seiner Vorstellung im Jahr 2012 zahlreiche wissenschaftliche Arbeiten angeregt.   * Art des ANNs: CNN * Anzahl zu klassifizierende Bilder: 1.2 Mil. * Anzahl der Klassen: 1000 * Anzahl der Parameter: 60 Mil. * Anzahl der Neuronen: 650’000 * Top-1 Error Rate: 37.5% * Top-5 Error Rate: 17.0% * Ein Bild zum Aufbau befindet sich im Anhang. * Für die Prüfung muss man evtl. den Aufbau verstehen. | |
| Voice Recognition | |
| Grundsätzlich können wir auch die Spracherkennung als ein einfaches Klassifizierungsproblem betrachten.   * Welches Wort (= Klasse) wurde im Input gesagt? | |
|  | |
| Wenn wir den Audio-Input nun in ein Spektrogramm (≈ Audiobild) umwandeln, so können wir diese Klassifizierung auch mit einem CNN lösen. | |
| **Deep Learning Architectures** | |
| Werkzeuge & Techniken | |
| Softmax-Funktion | |
| Besonders bei der Klassifizierung ist oftmals eine Wahrscheinlichkeit als Output eines ANNs gewünscht. Dies lässt sich mit der Softmax-Funktion erreichen:   * Wobei die Logits (numerische Outputs) des ANNs sind. * Die Softmax-Funktion ist ableitbar (s. Backpropagation). | |
| ANN  Logits  Klassen   * Merke: Die Summe über muss immer 1 sein. | |
| Dropout | |
| Die Regularisierungsmethode «Dropout» versucht, Overfitting zu reduzieren, indem es beim Trainieren eines ANNs «zufällig» eine bestimmte Anzahl Neuronen deaktiviert.   * Die Anzahl deaktivierte Neuronen nennt man Dropout Rate. * Dropout ähnelt von der Idee her den Ensemble Methoden. | |
|  | |
| Ohne Dropout\_\_ | \_\_\_Mit Dropout |
| * Die Dropout Rate ist ein Hyperparameter. * Dropout wird nur beim Training verwendet (s. Bild). | |
| Co-Adaptation | |
| Ohne Dropout kann eine Co-Adaptation zwischen den einzelnen Feature Maps auftreten. Das bedeutet, dass die Features nur noch in der Menge und nicht mehr allein sinnvolle Resultate liefern.   * Dropout kann diese Co-Adaptation aufbrechen. * Dies ist der Hauptgrund für die Verwendung von Dropout. | |
|  | |
| Data Augmentation | |
| Alle ML-Algorithmen (z.B. CNNs) können ausschliesslich mit Daten arbeiten, welche den Trainingsdaten ähneln.  «We cannot expect a network to do better than what it was trained for.»   * CNNs können z.B. keine Rotationen o. Skalierung erkennen. | |
| Probleme | |
| Besonders bei grossen ANNs werden viele Daten für das Trainieren der zahlreichen Parameter benötigt. Wir stossen dabei aber auf einige Schwierigkeiten:   * Daten zu sammeln & annotieren ist teuer. * Die Datenmengen sind oftmals zu klein. * Nicht alle Daten sind für ANNs geeignet. | |
| Datenmanipulation | |
| Mit «Data Augmentation» wollen wir dieses Problem lösen, indem wir aus den vorhandenen Daten neue generieren.   * Die Idee ist, dass die Modelle so besser generalisieren. | |
|  | |
| **Autoencoder** | |
| Anwendungsgebiet | |
| Autoencoder haben das Ziel, die eigenen Inputdaten über ein ANN zu reproduzieren. Somit lassen sich z.B. «Denoising»-Aufgaben lösen.   * «Denoising»: Unregelmässigkeiten in z.B. Bildern entfernen * Ziel des Autoencoders ist: | |
|  | |
| Sinn & Zweck | |
| Bei Autoencoder zwingen wir das Netzwerk dazu, so viele Information wie nur möglich durch einen «Bottleneck»-Layer zu übertragen. Der Autoencoder muss also eine enorm kompakte Repräsentation des Inputs (einen «Code») lernen.   * Der Code ist unser «Feature Vector» im «Latent Space». * Im «Latent Space» sind ähnliche Elemente nahe beieinander. * Autoencoder sind also mehr als eine Identitätsfunktion. | |
|  | |
| Anwendungsgebiet Pretraining | |
| Da Autoencoders keine Labels benötigen, können wir damit Modelle ohne Labels trainieren und dann die gelernten Features (den Encoder) in einem anderen ANN wiederverwenden.   * Wir nennen dieses Konzept «Transfer Learning» * Autoencoder sind also «Unsupervised / Self-Supervised» | |
|  | |
| Anwendungsgebiet Denoising | |
| Der Code vom «Bottleneck»-Layer ist eine Repräsentation der wichtigsten Features über alle Inputs hinweg. Da einzelne, isolierte Aspekte («Noise») nicht Teil vom Code sind, können wir diese mit einem Autoencoder entfernen. | |
|  | |
| **Recursive Neural Networks (RNN)** | |
| Anwendungsgebiet | |
| Durch die Verwendung von RNNs lassen sich sequenzielle Daten, wie z.B. Aktienkurse, Texteingaben oder Wetterbilder, sinnvoll verarbeiten.   * Sinnvoll bedeutet, dass die Sequenzmuster beachtet werden. * z.B. Kleidervorhersage basierend auf den letzten n Tagen. | |
| ? | |
| Sequenzielle Daten | |
| Bis anhin sind wir bei unseren ANNs immer von «statischen» Daten ausgegangen. Tatsächlich handelt es sich aber bei den meisten Dingen in der realen Welt um «Sequenzen» von Daten.   * Wörter = Zeichensequenzen, Texte = Wortsequenzen, etc. * Auch Dinge wie Aktienkurse o. Wetterdaten sind Sequenzen. | |
| Wetterdaten - Fraunhofer IBP | |
| Oftmals können wir anhand der Sequenz mehr über die Daten schliessen.   * z.B. «Ich lese aktuell eine …… Zusammenfassung.» | |
| Unterscheidung von ANNs | |
| Mit dieser Idee können wir nun unsere ANNs basierend auf den Ein- und Ausgabedaten in **4** Kategorien unterteilen. | |
| **1.** One-to-One | |
| Die Applikation nimmt einen Input und erstellt daraus einen Output.    Bild  Cat   * Die Daten sind wie bisher «statisch». (z.B. Bildklassifizierung) | |
| **2.** Many-to-One | |
| Die Applikation nimmt eine Sequenz von Inputdaten und erstellt einen Output.    Great  Movie  Positiv   * z.B. Sentiment-Analyse von einem Text | |
| **3.** One-to-Many | |
| Die Applikation nimmt einen Input und erstellt eine Sequenz von Outputdaten.    Bild  Gray  Cat   * z.B. Automatische Beschriftung von Bildern | |
| **4.** Many-to-Many | |
| Die Applikation nimmt eine Sequenz von Inputdaten und erstellt daraus eine Sequenz von Outputdaten.    Hello  World  Hallo  Welt   * z.B. Automatische Übersetzung von Texten | |

|  |  |  |
| --- | --- | --- |
| Probleme mit ANNs und CNNs | | |
| ANNs | | |
| Könnte man nicht einfach die Inputsequenz in einen Vektor codieren und durch ein reguläres ANN senden?   * Wir codieren eine Sequenz einfach in ein One-to-One ANN. * Warum also überhaupt diese Unterteilungen? | | |
| Hello  World  … |  | Hallo  Welt  … |
| Das Problem hierbei ist, dass normale ANNs die Reihenfolge der Eingabedaten nicht beachten. Jedes Element im Vektor wird also unabhängig von allen anderen Elementen betrachtet.   * Genau diese Abhängigkeiten wollen wir aber miteinbeziehen. * ANNs sind daher nicht geeignet für unsere Problemstellung. | | |
| CNNs | | |
| Könnte man nicht einfach eine Sequenz mit Elementen als Bild interpretieren und ein CNN darauf anwenden? | | |
|  | | |
| Tatsächlich können wir mit den CFD wiederkehrende Muster in Sequenzen erkennen. Jedoch können auch CNNs keine Informationen über die Reihenfolge der Eingabedaten speichern.   * CNNs können also Muster wie «Gut» o. «Schlecht» erkennen. * Wir können so z.B. Textsequenzen mit CNNs klassifizieren. * Für andere Aufgaben sind CNNs aber ungeeignet. | | |
| Recursive Neural Networks (RNN) | | |
| Funktionsweise | | |
| Wir können die Probleme von ANNs und CNNs lösen, indem wir die Eingabedaten nacheinander in ein RNN einfügen. In jedem Schritt speichern wir nun einen Teil der Berechnungen ab, welchen wir dann im nächsten Schritt wiederverwenden können. | | |
|  | | |
|  | | |
| * Beachte, dass es sich bei auch um einen Vektor handelt. | | |
| Unfolding | | |
| In der obigen Darstellung haben wir das RNN «entfalten», also über die Sequenz her ausgespannt. In dieser Form können wir das Netzwerk auch als ein reguläres ANN mit vielen Layers betrachten. Oft werden RNNs aber als sogenannte «Memory Cells» dargestellt.   * Wir nennen dieses «Entfalten» auf Englisch «Unfolding». * «Unfolding» wird auch beim Trainieren von RNNs angewendet. | | |

|  |
| --- |
| Memory Cell |
| ⚪  ⚪  ⚪  ⚪  ⚪  ⚪  ⚪⚪  ⚪  ⚪  ⚪  ⚪  ⚪  ⚪  ⚪⚪   * , und sind die trainierbaren Parameter. * Diese 3 Parameter werden von allen Elementen geteilt. * ist eine beliebige, nicht-lineare Aktvierungsfunktion. |
| Varianten von RNNs |
| Stacked RNNs |
| Wie mit den «Layers» können wir auch mehrere Memory Cells von RNNs hintereinanderstellen und trainieren.   * Wir sprechen dann auch von «Deep RNN» |
|  |
|  |
|  |
| * Der Output von einem RNN wird zum Input des nächsten. * Das Endresultat aller RNNs befindet sich dann in . |
| Bidirectional RNNs |
| In vielen Anwendungsfällen kann es vorkommen, dass ein Element eines RNN von der ganzen Sequenz abhängt.   * Die Spracherkennung ist ein dominantes Beispiel dafür. |
|  |
| Bidirektionale RNNs kombinieren nun RNNs die «vorwärts» durch die Sequenz laufen mit RNNs die «rückwärts» durch die Sequenzen laufen.   * So können wir bidirektionale Anhängigkeiten abbilden. |
|  |
|  |
| * Wobei eine für dieses Modul nicht relevante Funktion ist. * Beachte, dass Gewichte und Bias **nicht** geteilt werden. |
| Trainieren von RNNs |
| Wir können RNNs wie gewohnt mithilfe von «Backpropagation» trainieren.   1. Initialsiere die Gewichte 2. **Forward-Pass:** Laufe «vorwärts» über alle Elemente der Sequenz und berechne für jedes Element den jeweiligen Loss. 3. **Backward-Pass:** Laufe nun «rückwärts» über alle Elemente der Sequenz und passe für alle Elemente die Gewichte an. 4. Wiederhole, bis der Loss minimal wird.  * Wir verwenden dazu das «Unfolding». |
| * Unfolding + Backprop. = «Backpropagation through Time» |
| Loss |
| Die Loss-Funktion hängt wie so oft vom jeweiligen Problem ab. Oft werden aber diese Loss-Funktionen verwenden:   * Binary Cross Entropy (Binärklassifizierung) * Categorical Cross Entropy (n-Klassifizierung) * Root Mean Square Error (Regression) |
| Optimizer |
| In den meisten Fällen verwenden wir einen Gradienten-basierten Optimizer wie z.B. «Stochastic Gradient Descent». |
| Probleme von RNNs |
| Vanishing & Exploding Gradient |
| Beim «Backward-Pass» vom Backpropagation-Algorithmus kann es vorkommen, dass die Gradienten der Loss-Funktion nach einigen Iterationen enorm klein oder enorm gross werden.   * Wir nennen dies «Vanishing / Exploding Gradient Problem» * Alle Gradienten-basierten Algorithmen haben dieses Problem. * Wegen dem «Unfolding» ist es bei RNN besonders prävalent. |
| Vanishing Gradient |
| Probleme von Vanishing Gradient sind:   * Die Gewichte ändern sich nur noch minimal. * Das Modell wird nicht mehr verbessert. * Besonders «ältere» Elemente der Sequenz verlieren ihre Wichtigkeit. * Das Modell lernt nur noch kurzfristige Zusammenhänge. |
| Exploding Gradient |
| Probleme von Exploding Gradient sind:   * Die Gewichte ändern sich enorm. * Das Modell wird sehr instabil. * Die Werte können den gültigen Wertebereich übersteigen. («Value Overflow») |
| Lösungen |
| Tatsächlich gibt es zahlreiche Ansätze, wie sich das Problem lösen lässt:   * Aktivierungsfunktionen wie «ReLU» oder «Leaky ReLU» verhindern das Problem. * Mit «Gradient Clipping» können wir die Gradienten in einem definierten Bereich halten. * Eine gute Wahl der Initialparameter wirkt sich ebenfalls positiv auf das Problem aus. * Alle diese Ansätze können für RNN verwendet werden. * Die nachfolgenden Ansätze sind aber besonders relevant. |
| Skip Connections |
| Anstatt wie bisher für die Berechnung von und den Wert zu verwenden, können wir auch einen beliebigen wert an der Stelle nehmen.   * Wir können so dem «Vanishing Gradient» entgegenwirken. * Wir nennen das auch den «Delay». |
| **Gated RNN** |
| Anwendungsgebiet |
| Gated RNNs sind eine Erweiterung der normalen RNNs. Sie sind in der Lage, auch langfristige Zusammenhänge innerhalb einer Sequenz zu merken.   * Sie basieren auf der Idee von «Leaky Units» |
| Long Short-Term Memory (LSTM) |
| LSTMs funktionieren ähnlich wie normale RNNs, besitzen aber einen komplexeren Aufbau der «Memory Cell» und einen zusätzlichen «Cell State».   * LSTMs können so auch langfristige Zusammenhänge lernen. |
|  |
| Gates |
| Die «Gates» erlauben es einem LSTM, Informationen zu merken und bei Bedarf auch wieder zu vergessen. LSTMs bestehen dabei aus **4** Gates.   * LSTMs lernen also nur die wichtigen Informationen. * Im Grunde beinhalten Gates lediglich eine Sigmoidfunktion. |
| **1.** Forget Gate |
| Das «Forget Gate» vergisst irrelevante Informationen aus den vorherigen Teilschritten. Wir berechnen also:   * Wobei die Sigmoidfunktion ist. * Wennnahe bei 0 ist, werden die Werte «vergessen». |
|  |
| **2.** Compute Gate |
| Das «Compute Gate» berechnet den neuen «Cell State». Die Sigmoidfunktion soll dabei die relevanten und irrelevanten Informationen «aussortieren».   * Wird auch «Input Gate» oder «Store Gate» genannt. |
|  |
| **3.** Update Gate |
| Das «Update Gate» verbinden den alten mit dem neuen «Cell State». Dieses Resultat wird anschliessen an die nächste «Cell» weitergereicht.   * Siehe die vorherigen Berechnungen für , und . |
|  |
| **4.** Output Gate |
| Das «Output Gate» berechnet nun den finalen Output der aktuellen «Cell».   * Siehe die vorherigen Berechnungen für . |
|  |
| Varianten von LSTMs |
| Die Grundidee von LSTMs lässt sich in verschiedenen Varianten finden:   * Peephole Variation * Coupled Gate Variation * Depth Gated / Clockwork RNNs |
| Probleme mit LSTMs |
| LSTMs haben enorm viele trainierbare Parameter, etwa 4-mal so viele wie wir in normalen RNNs finden.   * Dieses Problem lässt sich leider nur schwer lösen. * Architekturen wie «GRU» sollen diese Anzahl aber verringern. |
| Gated Recurrent Units (GRU) |
| GRUs funktionieren vom Prinzip her ähnlich wie LSTMs, benötigen aber weniger trainierbare Parameter.   * GRUs besitzen ausserdem keinen «Cell State» * Somit sind alle Parameter von «aussen» Sichtbar. |
|  |
| Gates |
| GRUs bestehen aus **3** Gates.   * Sie haben also 3-Mal so viele Parameter wie RNNs. |
| **1.** Reset Gate |
| Das «Reset Gate» soll die kurzfristigen Zusammenhänge merken. Das Resultat dieses Gates gibt an, welche Informationen vergessen werden sollen. |
|  |
| **2.** Update Gate |
| Das «Update Gate» soll die langfristigen Zusammenhänge merken. |
|  |
| **3.** Output Gate |
| Der finale Output der aktuellen «Cell» wird nun wiefolgt berechnet:   * Siehe die vorherigen Berechnungen für und . * Der «Reset» wird also direkt auf angewandt. |
|  |
| LSTM vs. GRU |
| Die beiden Arten von RNN unterscheiden sich grundsätzlich in ihrem Aufbau und der Komplexität. Es gilt:   * **GRU:** Sind einfacher u. kompakter, wodurch das Training schneller wird. Sie können aber nicht so lange Sequenzen lernen. * **LSTM:** Sind komplexer u. schwerfälliger, können aber auch lange Sequenzen lernen. Das Training dauert normalerweise länger. * GRUs haben auch ein Gate weniger als LSTM. * Grob gilt: Wenige Daten = GRU, Viele Daten = LSTM. * Wie immer hängt dies aber von der Anwendung ab. |

|  |  |  |
| --- | --- | --- |
| **Reinforcement Learning (RL)** | | |
| Grundkonzepte | | |
| Reinforcement Learning bildet neben Supervised und Unsupervised-Learning eine eigene Klasse von AI-Algorithmen basierend auf dem Konzept von «Trial-and-Error» Learning. | | |
|  | | |
| Was bedeutet «Learning»? | | |
| Generell betrachtet beschreibt «Lernen» einen Prozess der Aneignung von neuen Informationen. Für Menschen beinhaltet dies insbesondere das Ändern des Verhaltens basierend auf Erfahrungen.   * Wir sprechen in diesem Fall von «Behavioral Learning». * Wir können diese Idee auch in Machine Learning anwenden. | | |
| Grundidee | | |
| In RL lässt man ein «Agent» mit einem «Environment» interagieren. Das Ziel vom «Agent» ist es dabei, einen bestimmten «Reward» zu maximieren.   * Wir verwenden dabei das «Trial-and-Error» Prinzip. * Wir lernen also **nicht** von vorgesammelten Daten. * Stattdessen «generiert» ein Agent seine eigenen Daten. | | |
| Schwierigkeiten | | |
| Auch wenn sich Reinforcement Learning konzeptionell gut anhört, müssen wir dabei folgende Punkte beachten:   * Sample Efficiency: RL-Algorithmen brauchen Millionen von Iterationen, tausende Simulationsstunden und enorme Rechenleistung. * Reward Definition: Die Definition vom Reward kann schwieriger sein als man denkt. * Erfolgsgeschichten: AlphaGo, Gran Turismo Sophy, etc. * Misserfolge: Antworten von TayTweets, etc. | | |
| **!** | | |
| Grundbausteine | | |
| Grundsätzlich besteht Reinforcement Learning immer aus **2** Elementen:   * Environment: Beinhaltet die States, Actions, Rewards und Transitions. * Agent: Versucht ein RL-Problem durch Interaktionen mit dem Environment zu lösen. | | |
| Markov Decision Process (MDP) | | |
| Wir können das Konzept des Environments auch in einem «Markov Decision Process» formalisieren. Ein MDP ist ein generelles, mathematisches Framework bestehend aus **4** Elementen:   1. Eine Menge von States 2. Eine Menge von Actions 3. Die Rewards 4. Die Transitions oder  * MDPs werden auch in vielen anderen Bereichen verwendet. * Wir können ein MDP auch in einem Diagramm darstellen. | | |
| * In diesem Falls sind , , und die «Terminal-States». * Terminal-States beenden eine sogenannte «Episode». * Wir erhalten den Reward **bei der Ankunft** in einem State. | | |
| Funktionsweise | | |
| Bei Reinforcement Learning interagiert ein Agent nun mit unserem definierten Environment. Das bedeutet konkret:   1. Der Agent nimmt eine Action . 2. Er sieht eine Veränderung vom State . 3. Er erhält einen Reward   Dieser Prozess wird nun unendlich lange oder bis zum Erreichen eines  Terminal-States wiederholt.   * Der Agent will dabei den Reward maximieren (= hedonisch). * Um dies zu erreichen, muss er ein optimales Verhalten lernen. | | |
| * Jede Action hat (potenziell) einen Einfluss auf das Environment. * Das Environment wechselt also in einen neuen State . | | |
| Policy | | |
| Das Verhalten eines Agents wird vollständig durch seine Policy definiert. Die Policy ist eine Funktion, welche States und Actions auf Wahrscheinlichkeiten abbildet.   * Wir schreiben z.B.: * «Die Wahrscheinlichkeit in nach zu gehen ist 76%.» | | |
| Ziel des Agents | | |
| Ein RL-Agent versucht also immer, eine optimale Policy zu finden, welche den totalen Reward maximiert.   * Wir können die Policy also als «Modell» interpretieren. | | |
| **Temporal Difference Learning (TDL)** | | |
| Ausgangslage | | |
| Nehmen wir an, wir haben ein Environment vollständig anhand des MDP spezifiziert. Wir können nun die einzelnen Elemente in einer Tabelle darstellen. | | |
| * Lese: Von nehme ich und lande in mit Reward 4. * Diese Tabelle ist deterministisch (Mit lande ich immer in ) * Oft sind solche Tabellen stochastisch (60% in , 40% in ) | | |
| Sicht des Agents | | |
| In den meisten Fällen kennt der Agent die Definitionen des Environments nicht, sondern muss diese durch Interaktionen mit dem Environment herausfinden. | | |
| * Lese: Ich war in , nahm und bin nun in mit Reward 9. * Der Agent muss sich nun für die nächste Aktion entscheiden. * Nach und nach lernt der Agent die besten Aktionen. | | |
| State Value Function | | |
| Die State Value Function gibt uns den erwarteten, totalen Reward eines States unter Verwendung von Policy an. Wir schreiben dies als:  Wir können verwenden, um die Qualität einer Policy zu evaluieren und somit die optimale Policy zu finden.   * Ausgeschrieben gilt also: * Denke: Der totale Reward, wenn ich von der Policy folge. * Anmerkung: Die Notation kann zum Teil etwas abweichen. | | |
| Anwendungsbeispiel | | |
| Nehmen wir an, dass der Agent die Werte T, R und kennt. Wir können nun anhand der State Value Function die optimale Aktion ablesen.   * Im Normalfall müssen T, R und zuerst gelernt werden. | | |
| * Lese: In wähle Aktion da am grössten ist. | | |
| Linearität von Erwartungswerten | | |
| Das «» in der -Formel beschreibt einen Erwartungswert, also einen Wert, der einer bestimmten Wahrscheinlichkeit unterliegt. Allgemein gilt dabei:  Wir können also daraus schliessen:  Woraus nun ausserdem gilt:   * Achtung: Der Erwartungswert ist nur eine «Erwartung». * Der reale Wert kann abweichen, es gilt also . * Aus diesem Grund müssen wir ein «» schreiben. | | |
|  | | |
| Differenz | | |
| Wir können den Unterschied zwischen realem und erwartetem Wert mit einer Differenz ausdrücken: | | |
| Reward Prediction Error (RPE) | | |
| Basierend auf diesen Überlegungen können wir nun unsere Formel mit der Differenz ergänzen:  Diese Differenz bezeichnen wir nun auch als den Reward Prediction Error:   * Wir initialisieren also mit «inkorrekten» Werten (z.B. 0) * Unser RL-Algorithmus soll nun die Differenz minimieren. * Konvergieren die Werte von , wird im Durchschnitt 0. | | |
| Discount Factor | | |
| Wir können in der Formel oben erkennen, dass wir alle zukünftigen Rewards grundsätzlich gleichbehandeln. Oftmals macht es jedoch Sinn, dass wir nähere Rewards den entfernteren vorziehen.   * Hätte ich lieber 10.- jetzt oder 50.- in 100 Jahren? * Lange Zeitspannen kommen immer mit einem Risiko. * Denke: «Lebe ich in 100 Jahren noch?» | | |
| Ein Bild, das Text, Gerät, Messanzeige enthält.  Automatisch generierte Beschreibung | | |
| Der Discount Factor implementiert diese Idee des «Vernachlässigens» durch einen konstanten Wert . Daraus folgt:  Woraus wir nun ableiten:   * Setzen wir , so erhalten wir die ursprüngliche Formel. * Anstatt vom Reward sprechen wir dann oft vom «Return ». | | |
| Temporal Difference | | |
| Die rechte Seite der Gleichung nennen wir auch die «Temporal Difference».   * Daher kommt der Name «Temporal Difference Learning» | | |
| Rechenbeispiel | | |
| Berechne den Wert , wobei gilt   und   * Das bezeichnet die «Trajectory», also den definierten Pfad. * Da der Pfad vollständig definiert ist, können wir die Erwartungswerte komplett ausrechnen. Daher gilt . | | |
| Grundformel:  Idee: Berechnung von «Hinten» nach «Vorne»   * Bei Terminal-States gilt immer . | | |
| Policy Evaluation | | |
| Um nun die «realen» Werte für  und somit anschliessend eine optimale Policy zu finden, wenden wir nun iterativ einen Optimierungsschritt an:   * Wobei wir die Differenz hier direkt als RPE bezeichnen. * Der Hyperparameter ist die «Learning Rate». | | |
| Pseudocode | | |
|  | | |
| Anmerkung | | |
| In diesem Optimierungsschritt lernen wir noch keine optimale Policy , sondern berechnen lediglich die Werte für eine gegebene Policy .   * Wir «evaluieren» also eine gegebene Policy . * Aus diesem Grund nennen wir es auch «Policy Evaluation». | | |
| **Policy Improvement** | | |
| Ausgangslage | | |
| Bisher haben wir gesehen, wie wir die Qualität einer Policy ermitteln können. Wie finden wir nun aber die optimale Policy mit dem grössten Reward?   * Das ist schliesslich unser Ziel in Reinforcement Learning. | | |
| Exploration-Exploitation Dilemma | | |
| Am Anfang von RL kennt unser Agent die Rewards im Environment noch nicht. Wir müssen diese zuerst herausfinden.   * Wir müssen also zuerst die Werte ermitteln. * Im Normalfall müssen wir sogar die Transitions T lernen. | | |
|  | | |
| Nach einer bestimmten Zeit soll der RL-Agent aber damit beginnen, dem vielversprechendsten Pfad zu folgen. Er soll sein Wissen also ausnutzen.   * Doch wann soll dieser Wechsel stattfinden? * Wir müssen Erkunden und Ausnutzen irgendwie balancieren. * Dies nennen wir das «Exploration-Exploitation Dilemma». | | |
| Random Policy Explore | | |
| Die «Random Policy» wählt die nächste Aktion jeweils **zufällig**. Durch dieses Verhalten erkundet die Random Policy immer das gesamte Environment.   * Wir können so also die Werte sinnvoll approximieren. | | |
|  | | |
|  | |  |
|  | | |
| Greedy Policy Exploit | | |
| Die «Greedy Policy» wählt immer die Aktion, welche den **grössten** Reward bringt. Durch dieses Verhalten maximiert die Policy den Reward, findet aber keine potenziell besseren Pfade mehr.   * Wir könnten z.B. zuerst Random und dann Greedy verwenden. | | |
| Wird nie gefunden! | | |
|  | |  |
|  | | |
| * Dieses Diagramm basiert auf dem vorherigen. * Da wird der Pfad nicht mehr erkundet. * Somit ändert sich nicht mehr, obwohl wäre. | | |
| Epsilon-Greedy Policy (EGP) | | |
| Um nun die beste Kombination von «Random» und «Greedy» Policy zu bekommen, müssen wir die beiden irgendwie balancieren. Die EGP erreicht dies durch einen neuen Parameter .   * Mit Wahrscheinlichkeit : Greedy Action * Mit Wahrscheinlichkeit : Random Action * Die Wahl des Hyperparameters ist dabei offen. * Das «» kann z.B. fix sein oder sich über die Zeit verändern. * Oft sieht man oder (von 0.95 nach 0.05) | | |
| Policy Iteration | | |
| Wir können nun die Idee von «Policy Evaluation» und «Policy Improvement» miteinander kombinieren. | | |
| **Policy Evaluation**  Berechne alle Werte für die Policy | | |
| **Policy Improvement**  Passe nun die Policy basierend auf den Werten an. | | |
| * Die Policy Evaluation ist fertig, wenn die Werte konvergieren. * Die Policy Improvement soll weiterhin erkunden (z.B. EGP) * Natürlich sind hier auch Varianten möglich. | | |
| Anmerkung | | |
| Der Begriff «Policy Improvement» ist in diesem Fall etwas irreführend, da die Policy an sich nicht «verbessert» wird.   * Wir wählen bloss die Aktion und verbessern so die Werte. * Am Ende verwenden wir **immer** die Greedy-Policy. * Wir werden aber noch andere Algorithmen kennen lernen. | | |
| **Q-Learning / SARSA** | | |
| Ausgangslage | | |
| Wir haben nun gesehen, wie wir die Werte für das Finden der optimalen Policy verwenden können. Jedoch haben wir dabei ein wenig gemogelt:   * Wir haben die «beste» Aktion immer basierend auf dem Reward und dem Wert ausgewählt. * Normallerweise kennen wir aber nur den State , den Reward und die Actions . * Allgemein kennt der Agent die nächsten States nicht. * Denke z.B. AlphaGo: Der nächste State hängt vom Zug des Agents **und** vom Zug des Menschen ab. | | |
|  | |  |
| Bis anhin | | Normalfall |
| State-Action Values | | |
| Die State-Action Values (Q-Values) lösen dieses Problem, indem sie den zu erwartenden Reward nicht den States, sondern den Actions zuordnen.   * Nehme ich in die Action , erwarte ich Reward . * Q-Values sind wie die Werte von der Policy abhängig. | | |
|  |  | |
| Wähle Aktion da | Wähle Aktion da | |
| Policy Evaluation | | |
| Grundsätzlich gilt dabei die exakt selbe Formel wie bei den Werten:  Woraus wir nun den RPE sowie den Optimierungsschritt ableiten können: | | |
| Evaluationsmethoden | | |
| Es gibt verschiedene Methoden für die Policy Evaluation mit Q-Values. Die Grundidee ist, dass wir im State eine Aktion nehmen und dann im nächsten Schritt die alten Werte anpassen.   * Im Modul haben wir SARSA und Q-Learning angeschaut. | | |
| SARSA | | |
| Diese Methode funktioniert wiefolgt:   1. Wähle in eine Aktion 2. Führe die Aktion aus und lande in 3. Wähle in eine Aktion 4. Passe nun die Werte von mittels des Werts und an.  * Wiederhole dies nun iterativ bis zum Ende der Evaluation. * SARSA steht dabei für «State-Action-Reward-State-Action» * Die gewählte Aktion kommt in vor. («On-Policy») | | |
| Pseudocode | | |
| * Wobei und gilt. | | |
| Q-Learning | | |
| Diese Methode funktioniert wiefolgt:   1. Wähle in eine Aktion 2. Führe die Aktion aus und lande in 3. Wähle in eine Aktion 4. Ermittle in nun die «beste» Aktion 5. Passe nun die Werte von mittels des Werts und an.  * Wiederhole dies nun iterativ bis zum Ende der Evaluation. * Die beste Aktion ist die Aktion mit dem grössten Wert. * Die gewählte Aktion kommt nicht in vor. («Off-Policy») | | |
| Pseudocode | | |
| Ein Bild, das Text enthält.  Automatisch generierte Beschreibung   * Wobei und gilt. | | |
| **Deep Reinforcement Learning (DRL)** | | |
| Ausgangslage | | |
| Probleme von Q-Values | | |
| Denken wir an komplexe Videospiele (z.B. Dota), so stossen wir mit den Q-Values schnell an die Grenzen:   * Videospiele haben enorm viele States . * Videospiele haben viele Actions . * Wir haben also enorm viele Q-Values . * Wir müssen alle diese Q-Values lernen! * Die Q-Values skalieren also enorm schlecht. * Wir können dies in einer Tabelle visualisieren. | | |
| * Was wäre mit 1'000 Actions und 1'000'000 States? | | |
| Q-Values haben ausserdem ein Problem mit der Generalisierung. Oftmals können wir nämlich in «ähnlichen» States auch ähnliche Aktionen anwenden.   * z.B. Ist die Veränderung von einem Pixel meist nicht relevant. * Die Aktion mit Pixel und Pixel ist also dieselbe. * Q-Values können diesen Zusammenhang aber nicht erlernen. | | |
| Lösungsidee | | |
| Wie wir bereits wissen, sind Q-Values nichts Weiteres als eine Funktion.   * Die Funktion bildet States und Actions auf einen Reward ab. | | |
|  | | |
| Aus der Mathematik wissen wir, dass wir jede Funktion irgendwie approximieren können. Wieso also nicht auch diese? | | |
| Deep Reinforcement Learning | | |
| In Deep Reinforcement Learning kom-binieren wir Methoden aus dem Deep Learning (wie ANNs) mit RL. Wir können so völlig neue Problemstellungen lösen.   * Wir verwenden z.B. CNNs, um die Q-Values zu approximieren. | | |
|  | | |
| Da wir in diesem Fall die Q-Values mithilfe von Deep Learning approximieren, sprechen wir auch von einem sogenannten Deep Q-Network (DQN). | | |
| **Continuous-Action Environments** | | |
| Ausgangslage | | |
| Bis anhin haben wir mit unseren Algorithmen jeweils die / Werte gelernt und daraus die optimale Policy abgeleitet. Wir wollen nun Algorithmen anschauen, welche direkt die Policy verbessern.   * Wir lernen also keine V-Values oder Q-Values mehr. * Somit können wir auch mit kontinuierlichen Aktionen arbeiten. | | |
| Continuous-Actions | | |
| In vielen Anwendungsfällen kann es  vorkommen, dass ein Agent aus einer «unendlichen» Menge von Actions auswählen kann. Wir reden dann von sogenannten kontinuierlichen Aktionen.   * z.B. Drehe das Lenkrad um 1, 0.1, 0.01, 0.001, … Grad. * Wie würden wir das mit z.B. Q-Learning anstellen? * Wir müssten eine unendliche Menge von Q-Values lernen. | | |
|  | | |
| Gaussian Policy (GP) | | |
| Eine Gaussian Policy ist eine Policy, bei der die Actions basierend auf einer Normalverteilung ausgewählt werden.   * Bei kontinuierlichen Aktionen nehmen wir eine GP an. * Lese: «Action is sampled from a gaussian distribution» * Wir erkennen, dass jeder State eigene Werte und hat. | | |
| Normalverteilung – Wikipedia   * Eine hohe Varianz bedeutet «explore», eine tiefe «exploit». * Erkennbar ist dies durch die «Spitze» der Verteilung. | | |
| Das Ziel unseres RL-Algorithmus ist es nun, die optimalen Werte und zu finden, welche einen bestimmten Reward maximieren. Wir schreiben daher auch:   * Wobei der trainierbare Parameter ist. * Oftmals können wir auch fixieren und nur lernen. | | |
| Policy Gradient | | |
| Die Policy Gradient Methode ermöglicht es uns nun, den optimalen Parameter zu finden. Sie verwendet dazu verschiedene mathematische Konzepte. | | |
| **1.** Objective Function | | |
| Die Objective Function beschreibt den zu erwartenden totalen Reward einer Policy bei definiertem Parameter .     * Diesen Wert wollen wir also maximieren. * Dazu müssen wir den optimalen Parameter finden. | | |
| **2.** Gradient Ascent | | |
| Wir können nun dank der Funktion den optimalen Parameter mithilfe der «Gradient Ascent» Methode finden.     * Achtung: «Gradent **Ascent**», nicht «Gradient **Descent**». * ist ein Erwartungswert. Wie finden wir den Gradienten? * Wir verwenden dazu das «Policy Gradient Theorem». | | |
| **3.** Policy Gradient Theorem | | |
| Dieses Theorem erlaubt es uns, den Gradienten der Funktion zu bestimmen, obwohl es sich dabei um einen Erwartungswert handelt. Es gilt:     * Die Herleitung davon ist für dieses Modul nicht relevant. | | |
| **4.** Monte-Carlo Approximation | | |
| Um nun die «Gradient Ascent» Methode verwenden zu können, müssen wir nun konkrete Werte für den Gradienten der Funktion bestimmen.    Da aber weiterhin Erwartungswerte beinhaltet, müssen wir diese irgendwie umwandeln. Wir verwenden dazu die Monte-Carlo Approximation.    Als Vergleich dazu:     * Die M.C.-Approximation findet konkrete Werte für , indem sie zufällig Trajectories der Länge auswertet. * Dies entspricht also wieder der Idee von «Trial-and-Error». | | |

|  |
| --- |
| REINFORCE |
| Der «REINFORCE» Algorithmus gehört zu den einfachsten Policy Gradient Methoden. Er verwendet die Monte-Carlo Approximation. |
| Funktionsweise |
|  |
| **Actor-Critic** |
| Ausgangslage |
| Bei Actor-Critic verbinden wir nun die Funktionsweise von / Werten mit der Idee von Policy Gradient.   * Actor: Der Actor lernt eine optimale Policy, indem er fortlaufend Aktionen auswählt. * Critic: Der Critic lernt die / Werte und evaluiert die Aktionen des Actors, indem er den erhalten Reward mit dem erwartetem Reward vergleicht. * Wichtig: Im Gegensatz zu Policy Gradient wird der Actor fortlaufend verbessert und wertet nicht zuerst eine Trajectory aus. |
|  |
| Funktionsweise |
|  |

|  |
| --- |
| **Berechnungen in Keras** |
| Definitionen:   * : Eingabedaten im Format * : Anzahl der trainierbaren Parameter * : Ausgabedaten im Format |
| Convolution |
| layers.Conv2D(  filters = k, kernel\_size = (n,m),  strides = (u,v), activation='…') |
| * Standardwerte: |
| Max / Average Pooling |
| layers.MaxPooling2D(  pool\_size = (n,m), strides = (u,v)) |
| * Standardwerte: , , * Gleiches gilt auch für layers.AveragePooling2D(…) |
| Flatten |
| layers.Flatten() |
|  |
| Dense |
| layers.Dense( units = k ) |
|  |
| Dense |
| layers.SimpleRNN( units = k ) |
| * Beachte: Anzahl Zeitschritte, Anzahl Features |