

HANSER



Leseprobe

zu

„Maschinelles Lernen“

von Jörg Frochte

ISBN (Buch): 978-3-446-46144-4

ISBN (E-Book): 978-3-446-46355-4

Weitere Informationen und Bestellungen unter

<https://www.hanser-fachbuch.de/978344646144-4>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

1	Einleitung	9
2	Maschinelles Lernen – Überblick und Abgrenzung	14
2.1	Lernen, was bedeutet das eigentlich?	14
2.2	Künstliche Intelligenz, Data Mining und Knowledge Discovery in Databases	15
2.3	Strukturierte und unstrukturierte Daten in Big und Small	18
2.4	Überwachtes, unüberwachtes und bestärkendes Lernen	21
2.5	Werkzeuge und Ressourcen	27
2.6	Anforderungen im praktischen Einsatz	28
3	Python, NumPy, SciPy und Matplotlib – in a nutshell	38
3.1	Installation mittels Anaconda und die Spyder-IDE	38
3.2	Python-Grundlagen	41
3.3	Matrizen und Arrays in NumPy	51
3.4	Interpolation und Extrapolation von Funktionen mit SciPy	63
3.5	Daten aus Textdateien laden und speichern	69
3.6	Visualisieren mit der Matplotlib	70
3.7	Performance-Probleme und Vektorisierung	74
4	Statistische Grundlagen und Bayes-Klassifikator	78
4.1	Einige Grundbegriffe der Statistik	78
4.2	Satz von Bayes und Skalenniveaus	80
4.3	Bayes-Klassifikator, Verteilungen und Unabhängigkeit	86
5	Lineare Modelle und Lazy Learning	100
5.1	Vektorräume, Metriken und Normen	100
5.2	Methode der kleinsten Quadrate zur Regression	114
5.3	Der Fluch der Dimensionalität	121
5.4	k-Nearest-Neighbor-Algorithmus	122

6	Entscheidungsbäume	129
6.1	Bäume als Datenstruktur	129
6.2	Klassifikationsbäume für nominale Merkmale mit dem ID3-Algorithmus	134
6.3	Klassifikations- und Regressionsbäume für quantitative Merkmale	148
6.4	Overfitting und Pruning	162
7	Ein- und mehrschichtige Feedforward-Netze	168
7.1	Einlagiges Perzeptron und Hebbsche Lernregel	169
7.2	Multilayer Perceptron und Gradientenverfahren	176
7.3	Klassifikation und One-Hot-Codierung	196
7.4	Auslegung, Lernsteuerung und Overfitting	198
8	Deep Neural Networks mit Keras	219
8.1	Sequential Model von Keras	220
8.2	Verschwindender Gradient und weitere Aktivierungsfunktionen	226
8.3	Initialisierung und Batch Normalization	229
8.4	Loss-Function und Optimierungsalgorithmen	238
8.5	Overfitting und Regularisierungstechniken	255
9	Feature-Engineering und Datenanalyse	264
9.1	Pandas in a Nutshell	264
9.2	Aufbereitung von Daten und Imputer	274
9.3	Featureauswahl	289
9.4	Hauptkomponentenanalyse (PCA)	302
9.5	Autoencoder	313
9.6	Aleatorische und epistemische Unsicherheiten	319
9.7	Umgang mit unbalancierten Datenbeständen	325
10	Ensemble Learning mittels Bagging und Boosting	329
10.1	Bagging und Random Forest	329
10.2	Feature Importance mittels Random Forest	335
10.3	Gradient Boosting	342
11	Convolutional Neural Networks mit Keras	352
11.1	Grundlagen und eindimensionale Convolutional Neural Networks	353
11.2	Convolutional Neural Networks für Bilder	365
11.3	Data Augmentation und Flow-Verarbeitung	378
11.4	Class Activation Maps und Grad-CAM	383
11.5	Transfer Learning	393
11.6	Ausblicke Continual Learning und Object Detection	401

12	Support Vector Machines	405
12.1	Optimale Separation	405
12.2	Soft-Margin für nicht-linear separierbare Klassen	411
12.3	Kernel-Ansätze	412
12.4	SVM in scikit-learn	418
13	Clustering-Verfahren	425
13.1	k-Means und k-Means++	429
13.2	Fuzzy-C-Means	434
13.3	Dichte-basierte Cluster-Analyse mit DBSCAN	438
13.4	Hierarchische Clusteranalyse	445
13.5	Evaluierung von Clustern und Praxisbeispiel Clustern von Ländern	452
13.6	Schlecht gestellte Probleme und Clusterverfahren	469
14	Grundlagen des bestärkenden Lernens	481
14.1	Software-Agenten und ihre Umgebung	481
14.2	Markow-Entscheidungsproblem	484
14.3	Q-Learning	492
14.4	Unvollständige Informationen und Softmax	506
14.5	Der SARSA-Algorithmus	514
15	Fortgeschrittene Themen des bestärkenden Lernens	519
15.1	Experience Replay und Batch Reinforcement Learning.....	522
15.2	Q-Learning mit neuronalen Netzen	538
15.3	Double Q-Learning.....	545
15.4	Credit Assignment und Belohnungen in endlichen Spielen	552
15.5	Inverse Reinforcement Learning	559
15.6	Deep Q-Learning	561
15.7	Hierarchical Reinforcement Learning	577
15.8	Model-based Reinforcement Learning	582
15.9	Multi-Agenten-Szenarien	586
	Literatur	591
	Index	601

1

Einleitung

Manche sagen, maschinelles Lernen sei ein Teilgebiet der künstlichen Intelligenz, andere ein Hilfsmittel in Disziplinen wie Data Mining oder Information Retrieval. Es hängt von der Sichtweise ab. Für mich ist es das Gebiet, das die interessantesten Aspekte zusammenbringt, nämlich Informatik, Mathematik und Anwendungen, die sehr vielfältig sind. Man kann hier mit Menschen zusammenarbeiten, denen es um autonomes Fahren oder um lernende Roboter in Industrie und Haushalt geht. Andere Anwendungsfelder sind beispielsweise das Verhalten von Menschen in Online-Shops oder Prognosen über Kreditwürdigkeit.

Das Feld ist aber deutlich breiter: Ein Kollege von mir setzt zum Beispiel in einem Projekt maschinelles Lernen ein, um Stimmen bedrohter Vögel in Neuseeland – <http://avianz.massey.ac.nz/> bzw. [PMC18] – zu erkennen und auch um zu ermitteln, wie viele Vögel wirklich auf einer Aufnahme zu hören sind. Das ist nicht leicht, denn es könnte dreimal derselbe Vogel sein, der ruft, oder eben drei verschiedene. Das ist maschinelles Lernen in der Ökologie. Andere Kollegen arbeiten mit Satellitendaten unterschiedlicher Qualität und versuchen so, Aussagen zu treffen, um die ausgebrachte Wasser- und Düngermenge zu optimieren. Das sind zwar alles sehr ernsthafte Fälle. Man darf aber auch einfach Spaß haben und versuchen, die KI in einem Computerspiel besser und unterhaltsamer zu machen. Das maschinelle Lernen ist also eine Art Schweizer Taschenmesser, welches man in den unterschiedlichsten Situationen sinnvoll und unterhaltsam einsetzen kann.

Ich versuche in diesem Buch, die meiner Ansicht nach wichtigsten Techniken und Ansätze darzustellen. Es enthält aber zunächst nur eines von diesen mitteldicken Schweizer Taschenmessern. Wenn Sie das Buch durchgearbeitet haben, schauen Sie mal nach folgenden Begriffen, die es nicht in die dritte Auflage geschafft haben: Outlier Detection, Radiale-Basisfunktionen-Netze, Self Organizing Maps, Recurrent Neural Networks ...

Es gibt viele interessante Themen in diesem Feld. Am Ende des Buches haben Sie sich bestimmt die Grundlagen – und auch etwas mehr – angeeignet, um sich schnell in neue Bereiche einarbeiten zu können. Um die ganzen Werkzeuge in diesem Schweizer Taschenmesser sinnvoll nutzen zu können, sollte man einen breiten Überblick über das Gebiet haben und sich nicht auf eine Technik beschränken. Aktuell sind z. B. Convolutional Neural Networks sehr in Mode und natürlich kann man auch diese irgendwie benutzen, ohne verstanden zu haben, wie neuronale Netze überhaupt funktionieren. Ein Keras-Tutorial dazu kann man schnell abtippen und sehen, wie der eigene Computer Ziffern mit hoher Genauigkeit erkennt. Man muss zwar nicht immer alles durchdringen, aber wenn man sich dafür interessiert, will man dort nicht stehen bleiben, oder?

Ich verstehe zum Beispiel nicht viel von meinem Auto, weil es mich nicht interessiert. Meine Ignoranz funktioniert hervorragend, weil das Produkt recht gut entworfen ist und ich nicht den Wunsch habe, an ihm herumzuschrauben. Als reiner User fahre ich bei Problemen in eine Werkstatt. Ich gehe aber davon aus, dass Sie mit dem maschinellen Lernen mehr machen wollen als sich hineinzusetzen und loszufahren. Sie sind Designer von Lösungen, kreativer Kopf hinter neuen Anwendungen oder die Werkstatt, die sich um die rote Warnlampe kümmert.

Wer hier nur ein Werkzeug kennt, läuft in eine Falle, die als *Law of the instrument* oder auch **Maslows Hammer** nach dem Ausspruch *If all you have is a hammer, everything looks like a nail* bekannt ist. Um hier für unterschiedliche Probleme mit großen und kleinen Datenmengen ebenso wie für unterschiedliche Ressourcenlagen Lösungen anbieten zu können, versuchen wir es mit vielen Werkzeugen. Das Ziel ist es, beim Kennenlernen dieser Werkzeuge zwischen den beiden Monstern Skylla (Theorielastigkeit) und Charybdis (flache Unbedeutendheit) möglichst unbeschadet hindurch zu kommen. Das bedeutet, ich möchte, dass Sie am Ende des Buches die mathematischen Hintergründe dieser Technik im Wesentlichen kennen, aber nicht notwendigerweise in der trockenen Form aus Definition, Satz und Beweis. Ich bin überzeugt, dass ein guter Mittelweg darin besteht, möglichst viele Algorithmen aus dem Bereich einmal selbst umzusetzen. Benutzt man nur fertige Bibliotheken, ist es etwa so, als würde jemand glauben, vom Zusehen schwimmen gelernt zu haben. Ich möchte also mit Ihnen gemeinsam wirklich *schwimmen* und Algorithmen basierend auf Verständnis und Theorie umsetzen. Dabei ist es in Ordnung, wenn unsere Umsetzungen nicht das Rennen bzgl. der Performance gewinnen. Es geht darum, die Prinzipien und theoretischen Grundlagen einmal ausprobiert zu haben. Allerdings soll das kein fundamentalistisches Dogma sein. Wer glaubt, er habe den Ansatz gefunden, der immer und für jeden funktioniert, ist im besten Fall eine Gefahr für sich und im schlimmsten für andere Menschen.

Es gibt ein paar Stellen, an denen wir mit der Idee, die Dinge from-scratch umzusetzen, nicht weiterkommen würden: tiefe neuronale Netze (Deep Learning) und Support Vector Machines. Beide benötigen mehr Wissen und Software rund um Optimierung und Co. als in dieses Buch passt. Gleichzeitig sind sie sehr spannend, sodass man sie nicht einfach weglassen sollte, nur weil die Umsetzung nicht gut auf ein paar Buchseiten passt. Wir setzen daher die neuronalen Netze in ihrer klassischen Form zu Fuß um und gehen dann für die tiefen dazu über, Keras als Bibliothek zu verwenden. Im Zusammenhang mit klassischen Netzen handeln wir fast alle Fallen und Begriffe ab und gehen dann mit Keras zu Anwendungen über, die mehr Leistung oder bessere Optimierung brauchen. Dasselbe gilt in gewisser Weise für die Support Vector Machines, die ebenfalls ein Modul aus der quadratischen Optimierung benötigen; nur weglassen sollte man sie nicht. Hier schauen wir uns erst etwas theoretischer die Grundlagen an und greifen dann zum Ausprobieren zur fertigen Umsetzung aus scikit-learn.

Weil wir abseits dieser beiden Fälle tendenziell über Algorithmen gehen, versuchen wir vorzugsweise, einen eher algorithmischen statt einen statistischen Zugang zu nehmen. Das liegt auch daran, für welche Zielgruppe ich gewöhnlich maschinelles Lernen aufbereite. Ich selbst unterrichte das Fach für Ingenieure oder angewandte bzw. technische Informatiker. Die Ingenieure in der Praxis oder an der Hochschule sind oft mit MATLAB vertraut und haben dort ggf. bereits etwas mit maschinellem Lernen versucht. Die Informatiker in den Studiengängen hatten Java, C und ggf. MATLAB. Das meiner Meinung nach beste und kostengünstigste Ökosystem zum maschinellen Lernen findet man jedoch bei Python oder R. Letzteres ist gerade bei Statistikern sehr beliebt. Für Ingenieure ziehe ich Python R vor; u. a. wegen dessen besserer Einbindung, auch in Robotik-Projekten, und wegen des leichten Umstiegs. Der Sprung von MATLAB zu Python ist gering, muss aber immer gemacht werden. Ziemlich genau diese Sprunghöhe, die jemand schaffen muss, der von MATLAB bzw. GNU/Octave zu Python wechseln möchte, ist auch im Buch eingebaut. Es funktioniert aber auch, wenn jemand von Java oder C++ kommt.

Daneben sind Ingenieure oder Ingenieurinformatiker oft sehr gut ausgebildet in linearer Algebra und Analysis, dafür fehlt die Statistik in der Ausbildung. Daher habe ich auch die Statistik

in der Darstellung des maschinellen Lernens möglichst knapp gehalten und den wirklich notwendigen Teil der Mathematik ins Buch integriert.

Die Analysis und lineare Algebra – in Kapitel 5 – sind in weiten Teilen eingebettet, wenn auch teilweise auf dem Niveau einer Erinnerung.

Im Buch baue ich die Quelltexte immer (fast) vollständig ein. Sie können die Quellen natürlich auch von meiner Seite <https://joerg.frochte.de> downloaden, aber mir ist es wichtig, dass der Python-Code wirklich ins Buch eingebunden ist. Wenn man einen algorithmischen Zugang versucht, sind die Algorithmen eben wesentlich. Außerdem möchte ich gerne, dass man das Buch sowohl vor dem Computer benutzen kann – direkt alles mitmachen und ausprobieren – als auch in einem Park sitzend und nur lesend. Ich selbst lese gerne auch gedruckte Fachbücher als Entspannung, wenn ich gerade keinen Bildschirm sehen will ... und ich vermute, ich bin damit nicht allein. Man kann Algorithmen in Python tatsächlich sehr kompakt notieren und auf die wesentlichen Ideen beschränken, was Python ebenfalls für den Abdruck interessant macht. Bezüglich des Codes wird jemandem, der Python schon länger benutzt, etwas auffallen: Ich benutze kein `snake_case`, was in Python ungewöhnlich ist. Gründe sind sicherlich, dass meine Kursteilnehmer von C, MATLAB oder Java kommen, ich selbst auch oft zwischen diesen Programmiersprachen wechsele und mir dabei eine Art Crossover-Stil angewöhnt habe. Ich hoffe, es stört niemanden, der an sauberes PEP 8 gewöhnt ist, und bitte falls doch um Nachsicht. Der Stil für die Variablennamen ist hier aber nicht so wichtig. Wir haben es die meiste Zeit mit sehr kurzen Variablen zu tun, wie X für die Trainingsmenge und Y für die Menge der Ziele usw. Da wölbt sich keine Schlange und macht kein Kamel einen Höcker.

In den Python-Codes, die einen wichtigen Teil des Buches ausmachen, müssen wir ohne Pfeile oder Fettdruck für Vektoren und Matrizen auskommen. Entsprechend lassen wir dies auch für die Formeln weg und bemühen uns, so zu klären, was was ist. Formeln haben immer dann eine Nummer, wenn auf diese später noch einmal verwiesen wird. Gleichheitszeichen im Pseudocode sind i. d. R. als *gleichgesetzt*, also als Zuweisung, zu lesen.

Im Buch sind drei Arten von „Boxen“ eingebaut:



Diese hat etwas damit zu tun, wo Sie in **scikit-learn** den Algorithmus, den wir gerade umgesetzt haben, finden können. Es ist lediglich ein Verweis auf professionelle Umsetzungen. **Keras** und **scikit-learn** sind für mich zwei sehr wichtige Stützpfeiler, um schnell und gut etwas in Python umsetzen zu können. Ich habe die Verweise auf diese beiden Bibliotheken beschränkt.



Die zweite Textbox ist ein allgemeines *Achtung*. Ich setze diese Box nicht so oft ein, weil irgendwie alles oder nichts wichtig ist. Aber manche Dinge sind doch ärgerlicher als andere und kosten sehr viel Zeit, wenn man sie überliest. Wenn ich einen dieser Fälle bereits erlebt habe, taucht diese Box auf.



Die wichtigste Gruppe von Boxen erkennt man an dem Schraubenschlüssel. Hier gibt es Anregungen, was Sie selbst anschließend ausprobieren könntnen. Keine von diesen ist nötig, um dem Rest des Buches zu folgen. Es sind einfach Vorschläge, da-

mit Sie selbst etwas mit dem neu Gelernten anfangen können, ohne dass die Lösung sofort danebensteht. Oft gibt es nicht DIE Lösung, sondern eben sehr viele Ansätze.

Wenn ich eine Variable aus einem Quellcode im Fließtext verwende, wird diese als Schreibmaschinenschrift gesetzt. Dinge, die fettgedruckt sind, tauchen im Index hinten auf. Der Sinn liegt darin, dass Sie etwas hinten im Index suchen und dann auf der Seite sofort sehen, wo es steht. In der Regel wird etwas nur beim ersten Auftauchen in den Index aufgenommen. Ansonsten bedeutet ein kursiver Druck etwas Ähnliches wie *Eigennamen* oder *In-Anführungszeichen*. Der Einstieg in Python ist in Kapitel 3 konzentriert. Wer Python zusammen mit NumPy & Co. schon beherrscht, kann das Kapitel überspringen. Alle anderen erhalten in Kapitel 3 einen schnellen praktischen Einstieg, wie mit Matrizen und Arrays in NumPy gearbeitet wird. Im Unterschied zu Python als Grundlage habe ich beim Aufbau des Buches versucht, auch die Einarbeitung der mathematischen Grundlagen über mehrere Kapitel zu verteilen und nicht in einem Einstiegskapitel oder Anhang zu bündeln; einfach damit es nicht einen langen trockenen Teil gibt und dann viele Passagen, die quasi primär aus Pseudo- bzw. Quellcode bestehen. In Kapitel 4 folgt zusammen mit dem ersten Klassifikator ein guter Teil der minimalisierten statistischen Grundlagen, die wir brauchen. In Kapitel 5 gehen wir noch einmal tiefer auf Vektorräume, Normen und Metriken ein. Diese sind u. a. notwendig, wenn man hinterher, wie in Kapitel 13, versucht, Grade von Ähnlichkeiten zwischen Objekten zu ändern, und zwar durch die Art, wie Abstände definiert werden. In den Kapiteln 7 und 8 ist wiederum etwas Optimierung eingebaut, welche nötig ist, um neuronale Netze zu trainieren.

Die Auswahl von Merkmalen und ihre Reduktion sind das Thema des Kapitels 9. Hier brauchen wir noch einen Nachschlag bzgl. der Statistik und dazu Grundlagen zu Eigenwerten und Eigenvektoren. Diese sind nötig, um die Principal Component Analysis richtig einzuordnen. Vielleicht etwas ungewöhnlich ist die Lage des Kapitels 9 innerhalb des Buches. Man könnte eigentlich annehmen, dass die Diskussion über die Daten weiter vorne kommen sollte. Jedoch wollte ich für einige Demonstrationen schon Lernverfahren zur Verfügung haben, wozu neuronale Netze gehören, da diese z. B. andere Ansprüche haben als ein Entscheidungsbaum. Neu in der dritten Auflage ist, dass ich versuche, in dieses Kapitel Pandas als wohl wichtigste Bibliothek im Umgang mit strukturierten Daten einzubinden.

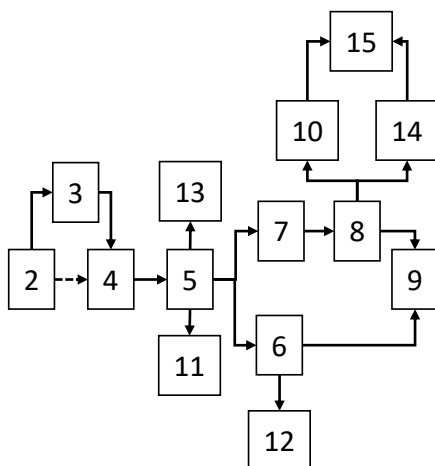


Abbildung 1.1 Abhängigkeiten zwischen den Kapitel des Buches

Wer das Buch in einer Vorlesung einsetzen will und kürzen möchte bzw. muss, für den habe ich die Abbildung 1.1 eingebaut. Sie gibt die wesentlichen Abhängigkeiten wieder. Es kann immer mal irgendwo ein Verweis auch außerhalb dieser Abhängigkeiten auftauchen. Daneben gibt es in jedem Kapitel viele Möglichkeiten zu kürzen, abhängig von den Vorkenntnissen der Teilnehmerinnen und Teilnehmer oder den Zielen zu kürzen. Wer z. B. eigentlich bestärkendes Lernen mit neuronalen Netzen machen möchte, der kann sich im Kapitel 8, auf die ersten zwei beiden Abschnitte konzentrieren. Generell kann man dann vieles auf dem Weg anpassen oder weglassen; dafür sind Dozentinnen und Dozenten eben wichtig.

Jetzt sollten wir aber anfangen, nachdem ich mich bei einigen Menschen bedankt habe. Das sind im Vergleich zur letzten Ausgabe einige mehr geworden. Ich möchte mich wirklich bei Lesern bedanken, die mir Anregungen schicken oder Stellen nennen, bei denen die Erklärungen im Buch nicht rundlaufen. Diese haben mich – ebenso wie die Arbeit zusammen mit Herrn Kaufmann am Projekt Weiterbildung AI (<https://we-ai.de>) – sehr weitergebracht bzgl. der Frage wo man noch was ändern oder ergänzen müsste. Neben den Leserinnen und Lesern, die mir konstruktive Rückmeldungen per E-Mail geben, haben mir auch viele Kolleginnen und Kollegen geholfen. Ich vergesse bestimmt jemanden und möchte mich dafür entschuldigen, jedoch unter anderem möchte ich mich bedanken bei Sabine Weidauer, Benno Stein, Matthias Rottmann, Peter Gerwinski, Herbert Schmidt, Michael Knorrenschild, Peter Beater, Christof Kaufmann, Henrik Blunck, Marco Schmidt, Stefan Müller-Schneiders, René Schoesau, Stefan Bader, Roland Schroth, Annabel Lindner, Julia Sudhoff und Gernot Kucera.

Wenn man ein Buch schreibt, kostet das immer viel zusätzliche Zeit neben dem normalen Beruf. Daher vielen Dank, dass ich mir diese nehmen durfte, an meinen Sohn Laurin und meine Frau Barbara. Letzterer genau wie den Korrektor und Lektorin des Hanser-Verlags vielen Dank für die Anregungen und Überarbeitungen.

Auch für die Zukunft freue ich mich auf jede Rückmeldung an joerg@frochte.de – und nun los!

wenn Sie das Buch lesen, schon nicht mehr. Durch die Anwendung von `argmax` stellen wir fest, welche Klasse die höchste Wahrscheinlichkeit hat und entsprechend als Vorhersage gilt.

```
70 testGenerator.reset()
71 yP = CNN.predict_generator(testGenerator, steps=len(testGenerator), verbose=True)
72 yPClass = np.argmax(yP,axis=1)
```

`testGenerator.classes` enthält die Labels gemäß den Verzeichnissen. Hiermit finden wir heraus viele Hunde und Katzen denn genau in unserer Testmenge sind und abschließend schauen wir danach, wie oft es mit der Klassifikation nicht funktioniert hat.

```
73 cats = np.sum(testGenerator.classes == 0)
74 dogs = np.sum(testGenerator.classes == 1)
75 catsAsDogs = np.sum( np.abs(yPClass[testGenerator.classes == 0] -0) )
76 dogsAsCats = np.sum( np.abs(yPClass[testGenerator.classes == 1] -1) )
77 confMatrix = np.array([[cats-catsAsDogs)/cats, catsAsDogs/cats],
78                       [dogsAsCats/dogs, (dogs-dogsAsCats)/dogs]])
79 print(confMatrix)
```

Als Konfusionsmatrix erhalte ich:

	y_P Cat	y_P Dog
y_T Cat	85.08	14.92
y_T Dog	06.07	93.93

Es kann sein, dass Sie wegen der zufälligen Initialisierung eine andere Verteilung bekommen. Fall Sie genau mit einem Netz weiterarbeiten wollen, laden Sie sich das ZIP-File von einer Webseite. Das `hd5` dieses Netzes ist ebenfalls enthalten.

Der Klassifikator ist also wesentlich besser, wenn es um Hunde geht als um Katzen. Oder um genauer zu sein: Im Zweifelsfall tendiert er dazu etwas als Hund zu klassifizieren. Im nächsten Abschnitt versuchen wir unseren Klassifikator noch etwas besser zu verstehen.

■ 11.4 Class Activation Maps und Grad-CAM

Wenn man ein CNN verwendet, um zum Beispiel eine Klassifizierung vorzunehmen, geht schon mal was schief. Manchmal sind es Dinge, die man algorithmisch lösen kann. Also zum Beispiel, indem man die Architektur des Netzes verändert oder die Optimierer. Nicht selten sind es aber einfach die Daten, auf die man aufsetzt. Ein schönes Beispiel stammt aus dem Paper [RSG16]. Hier sollten u. a. Wölfe und Huskys unterschieden werden. Die Bilder der Wölfe hatten jedoch immer Schnee im Hintergrund, während die Bilder der Huskys keinen Schnee im Hintergrund hatten. Auf einem Testset, welches nach dem gleichen Prinzip aufgebaut ist, funktioniert die Klassifikation hervorragend. Wird jedoch ein Husky mit Schnee im Hintergrund gezeigt, so wird dieser, sogar mit einer hohen durch das Netz angegebenen Genauigkeit, als Wolf klassifiziert. Das Netz hat durch einen Bias in den Daten zwar hohe Genauigkeiten, ist aber für viele Einsätze sinnlos; ein Dackel im Schnee ist weitab von einem Wolf.

Während es sich bei [RSG16] um ein bewusstes Experiment handelte, kommt das Problem auch in wesentlich kritischeren Anwendungen vor. Das Problem wird klarer, wenn man sich

die Arbeiten um das Paper [EKN⁺17] vor Augen führt. Hier geht es darum, Flecken auf der Haut als Hautkrebs – sogar verschiedene Arten – und harmlose Artefakte zu identifizieren. Das Paper hat einen sehr großen Eindruck gemacht und wurde tausendfach zitiert. Die Aussage war vereinfacht, dass die KI bzw. AI besser war als die meisten Ärzte und sich bald vermutlich jeder einfach sein Smartphone schnappen könnte, um damit schnell herauszufinden, ob es ein harmloses Muttermal oder Hautkrebs ist. Die Begeisterung macht es schwer, im Netz bzw. den Medien die eher kritischen Stimmen ausfindig zu machen. Der Datensatz weist nämlich einige Probleme auf, wie u. a. in [NKS⁺18] thematisiert wird. Die Bilder des Trainingssets enthalten teilweise Markierungen, um die Größen der Abbildung besser einschätzen zu können. Das Problem, auf das auch in [NKS⁺18] hingewiesen wird, ist, dass in dem Trainingsdatensatz Bilder mit Größenmarkierungen eher bösartige Hautveränderungen beinhalten, sodass es nah liegt, dass der Algorithmus unbeabsichtigt lernt, dass die Markierungen für bösartige Veränderungen stehen.

Häufig können Artefakte in der Datenerfassung wie der Schnee oder die Markierungen auf medizinischen Bildern unerwünschte Korrelationen hervorrufen, die die Klassifikatoren während des Trainings aufgreifen und nicht im Sinne der späteren Anwendung nutzen.

Diese Probleme können allein durch die Betrachtung der Rohdaten und Vorhersagen sehr schwer zu identifizieren sein. Das gilt besonders, weil man es hier mit unstrukturierten Daten zu tun hat, bei denen der Klassifikator selber auch die Merkmale generiert. Die Forschung auf dem Gebiet ist sehr aktiv und es gibt vielversprechende Ansätze. Ich möchte hier ein Beispiel für eine Methode präsentieren, die in ihrer Leistungsfähigkeit zwar begrenzt ist, aber einen ersten Einblick liefert. Außerdem erlaubt uns die Beschäftigung noch einmal mehr über CNN zu lernen. Man geht bei CNN davon aus, dass die erzeugten Merkmale an Abstraktion zunehmen. Was bedeutet das? Man weiß, dass der erste Layer im Wesentlichen recht einfache Muster auf der Basis von Kanten oder Gradienten erkennt. Es sind eher einfache Filter wie in Abbildung 11.11 auf Seite 367, welche die Voraussetzung für das weitere Lernen abstrakter Dinge schaffen. Die Hoffnung ist, dass die nächsten Layer sich dann von den Pixeln wegbewegen und eben ein Konzept wie z. B. Auge oder Schwanz erkennen. Optimalerweise gibt es am Ende nach unserer letzten Faltung dann Merkmale, die quasi für *Katzenaugen* oder *Pfote ohne Krallen* stehen. Es ist aber keineswegs sichergestellt, dass diese abstrakten Konzepte irgendetwas mit dem zu tun haben, was wir Menschen als ein abstraktes Konzept für ein Tier oder ein Ding verstehen. Sie sind mit Sicherheit in dem Sinne abstrakter als sie nicht mehr direkt auf der Pixelebene basieren.

Nimmt man an, dass der eine Filter am Schluss primär für einen Hunde- oder Katzenschwanz bzw. Katzenaugen oder Pfoten ohne Klauen steht, dann kommt man schnell hinter die Grundidee aus dem Paper [ZKL⁺16]. In diesem Paper wird ein Ansatz mit dem Namen **Global Average Pooling**, kurz **GAP**, propagiert. Der Name ist Programm und der bedeutet nichts anderes als Mittelwertbildung jeder Feature-Map. Statt eines Übergangs mittels Flatten in ein nachgelagertes dichtes Netz wird ein **GAP-Layer** verwendet; eben der Ansatz, den wir schon als Alternative zum Max-Pooling kennengelernt haben. Nimmt man an, dass ein Netz 14 Feature Maps hat, besteht der GAP-Layer aus deren Mittelwerten. Das bedeutet aber auch, dass man, um diese Technik anwenden zu können, eine spezielle Netzarchitektur benötigt. Hinter der Merkmalsgenerierung durch Faltung und Pooling darf nur noch ein GAP-Layer und ein vollverbundener Output-Layer folgen. Es erfolgt kein Flattening. Das Netz muss mit der in Abbildung 11.17 dargestellten Architektur trainiert werden, bevor man diese nutzen kann, um zu visualisieren, welche Aspekte im ursprünglichen Bild am meisten zu einer Klassifizierung bei-

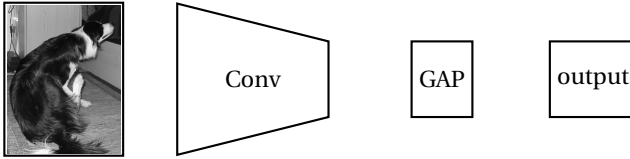


Abbildung 11.17 Netzwerkarchitektur mit GAP-Layer

getragen haben. Diese Darstellung, die wir erhalten wollen, nennt man **Class Activation Maps**. Es ist eine skalare Bitmap mit Werten, die den Grad der Aktivierung angeben.

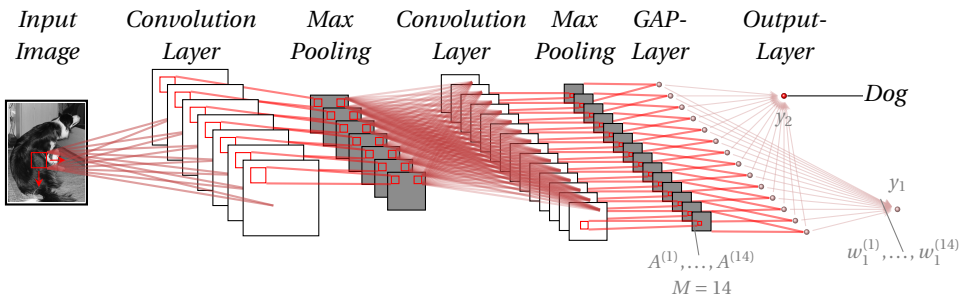


Abbildung 11.18 Detail-Netzwerkarchitektur mit GAP-Layer

Der erste Schritt ist das Bilden des Mittelwertes über eine Feature Map $A^{(m)}$:

$$\text{Mittelwert} = \sum_{i,j} A_{i,j}^{(m)} \tag{11.9}$$

Anschließend werden diese mit den Gewichten $w_c^{(m)}$ multipliziert. c ist dabei der Index für die Verbundene Klasse, während m für die Nummer der Feature Map steht. Die Entscheidung für eine Klasse geschieht dann gemäß der Gleichung (11.10).

$$y_c = \sum_{m=1}^M \left(\frac{1}{I \cdot J} \sum_{i,j} A_{i,j}^{(m)} \right) \cdot w_c^{(m)} \tag{11.10}$$

Das ist bis jetzt nur eine etwas andere Art, die Klassifikation durchzuführen. Wie hängt dies nun mit der Visualisierung der Aktivierung zusammen. Dazu stellen wir die Gleichung (11.10) ein wenig um.

$$y_c = \frac{1}{I \cdot J} \sum_{i,j} \underbrace{\left(\sum_{m=1}^M A_{i,j}^{(m)} \cdot w_c^{(m)} \right)}_{\text{CAM}} = \frac{1}{I \cdot J} \sum_{i,j} \underbrace{\left(\sum_{m=1}^M A^{(m)} \cdot w_c^{(m)} \right)}_{\text{CAM}} \frac{1}{I \cdot J} \sum_{i,j} \text{CAM}_{i,j} \tag{11.11}$$

Hierbei vertauschen wir die Summen über die Dimensionen I und J der Feature Map und die Summe über alle Feature Maps. In der inneren Klammer von Gleichung (11.11) erhalten wir nun eine Matrix, welche an den Stellen – über alle Feature Maps hinweg gemittelt – große Einträge hat, wo eine starke Aktivierung für die Entscheidung Klasse c vorliegt. Der Ausdruck *gemittelt* ergibt sich wegen der Division durch $I \cdot J$, also der Größe der Matrix. Die Darstellung

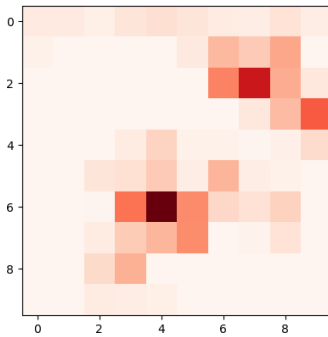


Abbildung 11.19 Beispielhafte Darstellung einer Class Activation Map (CAM)

$A^{(m)}$ soll deutlich machen, dass es eben Matrizen sind, die mit $w_c^{(m)}$ gewichtet und anschließend durch die Summation überlagert werden.

Diese CAM-Matrix können wir nun verwenden, um im ursprünglichen Bild diese Stellen für einen Menschen sichtbar zu machen. Dazu muss man sich aber vor Augen führen, dass die Feature Map durch die Pooling-Schritte und den Verschnitt am Rand – es sei denn, es wurde durchgängig ein Padding angewendet – deutlich kleiner geworden ist. Daher ist es üblich, diese eine Class Activation Map, wie sie beispielhaft in Abbildung 11.19 gezeigt wird, hochzuskalieren und diese skalierten Werte mit dem ursprünglichen Bild zu überlagern.

Das Problem ist, dass man mit dem GAP-Ansatz in der Regel schlechtere Ergebnisse erzielt als mit einem dichten Netz. Wir würden also bei gleich komplexen Modellen mehr Transparenz gegen weniger Genauigkeit tauschen. Zum Glück wurde ein Jahr nach dem GAP-Ansatz noch ein weiteres Paper publiziert. In [SCD⁺17] wird ein Ansatz vorgestellt, mit dessen Hilfe man versucht, die alten Architekturen weiterverwenden zu können und gleichzeitig doch mehr Transparenz zu erhalten. Das Ergebnis ist insofern sehr angenehm, als es auf viele Netze angewendet werden kann. Die Zuordnung zu den Aktivierungen für die Klassenentscheidung ist aber weniger gradlinig und an einigen Stellen eher heuristisch motiviert.

Als Erstes konzentrieren wir uns auf den Term $A^{(m)} \cdot w_c^{(m)}$ in Gleichung (11.11). Die Feature Maps existieren natürlich auch bei der klassischen CNN-Architektur, aber statt $w_c^{(m)}$ haben wir einen wesentlich komplexeren Zusammenhang, der durch ein neuronales Netz gegeben ist. Hier nimmt man eine Anleihe bei der Backpropagation, bei der man auch für die Gewichtsänderung versucht, die Einflüsse durch die Gradienten zu motivieren. Allerdings ist in diesem Fall nicht die ganze Feature Map mit dem Output-Neuron y_c verknüpft, sondern einzeln Einträge der Feature Map, welche in einen Vektor durch Flattening umformatiert werden.

$$A^{(m)} \cdot w_c^{(m)} \rightsquigarrow A^{(m)} \cdot \underbrace{\left(\sum_{i,j} \frac{\partial y_c}{\partial A_{i,j}^{(m)}} \right)}_{\text{Summierter Gradient}}$$

Nimmt man an, dass ein Netz vorliegt, in dem die Zusammenhänge linear sind und zuvor ein Average Pooling durchgeführt wurde, ist damit der Ansatz über einen GAP-Layer ein Spezialfall dieser Art, Gewichte zu berechnen.

Die gesamte Formel aus [SCD⁺17] lautet:

$$\text{ReLU} \left(\underbrace{\sum_{m=1}^M \left(\frac{1}{I \cdot J} \sum_{i,j}^{I,J} \frac{\partial y_c}{\partial A_{i,j}^{(m)}} \right)}_{\text{Gemittelter Gradient}} \cdot A^{(m)} \right) = \text{ReLU} \left(\sum_{m=1}^M w_c^{(m)} \cdot A^{(m)} \right) \quad (11.12)$$

In (11.12) tritt der gemittelte Gradient an die Stelle des Gewichtes für die Feature Map beim GAP-Ansatz. Anschließend verhindert die ReLU-Funktion, dass negative Werte auftreten können. Das ist einer der Aspekte, den ich mit *heuristisch* vorab meinte. Es liegt die Idee zugrunde, dass negative Gewichte beim ursprünglichen GAP-Verfahren eher ausdrücken wollen; *gehört nicht zu der Klasse* und positive *gehört zu der Klasse*. Daneben haben die Autoren des Grad-CAM rumprobiert – und was positiv ist diese Tests auch in die Veröffentlichung aufgenommen – und bei Ihnen klappte die Darstellung mit ReLU besser. Das ist aber kein mathematischer Beweis, nicht einmal für einen Spezialfall.

Diese Formel werden wir nun einmal versuchen umzusetzen. Dazu müssen wir Gradienten berechnen, und das wollen wir gerne automatisch berechnen lassen. Hierzu bietet TensorFlow – nicht Keras – eine geeignete Funktionalität. Entsprechend müssen wir TensorFlow selbst einbinden. Darüber hinaus soll der Gradient nicht über das ganze Modell berechnet werden, sondern nur von den Feature Maps bis zu den Output-Neuronen. Hierzu müssen wir quasi ein Teilmodell bilden. Um dieses Teilmodell zu erzeugen, nutzen wir ausnahmsweise einen Aspekt der **Model Class API** von Keras statt der `Sequential`. Entsprechend importieren wir die Klasse **Model**.

```

80
81 import tensorflow as tf
82 from tensorflow.keras.models import Model
83 def heatmap(img, model):
84     for l in model.layers:
85         if isinstance(l, Conv2D): lastConvLayer = l
86     calcFeaturesAndPred = Model([model.input], [lastConvLayer.output, model.output])

```

Als Erstes Suchen wir in Zeile 85 den letzten Layer vom Typ `Conv2D`, also die letzte Faltung. Liegt dahinter noch ein `Pooling`, wird dies in die Gradientenbildung einbezogen. In Zeile 86 bauen wir uns mit der `Model`-API ein neues Modell auf der Basis unseres bereits trainierten Modells. Der Unterschied ist hier lediglich, dass wir zwei Outputs festlegen, nämlich die bekannten Output des `Softmax` und zusätzlich den Output des letzten Faltungslayers `lastConvLayer.output`.

Nun benötigen wir nach langer Zeit das in Abschnitt 3.5 erwähnte `with`-Statement. Es ist die in TensorFlow vorgesehene Methode, um bei der parallelen Ausführung mit Ressourcenkonflikten und der Bereinigung umzugehen. Mit diesem `with`-Statement verknüpft ist hier `tf.GradientTape`. Es ist Teil der TensorFlow API zur automatischen Differenzierung, also der Berechnung des Gradienten einer Funktion in Bezug auf die Eingabevariablen. Die Bezeichnung `tape` kommt daher, dass alle Operationen, die im Kontext eines `tf.GradientTape` ausgeführt werden quasi auf ein *Band* (eng. `Tape`) aufgezeichnet. TensorFlow verwendet anschließend dieses Band bzw. `Tape`, um die Gradienten, die mit jeder aufgezeichneten Operation verbunden sind zu berechnen. Das funktioniert aber lediglich mit TensorFlow-Variablen. Entsprechend wandeln wir in Zeile 87 unser Bild in eine solche Variable um. Mit diesem Typ kann man fast genauso arbeiten wie mit `NumPy`-Arrays. In Zeile 89 müssen wir wieder eine leere Dimension hinzufügen, da wir eben nur ein Bild durch die Verarbeitung verfolgen wollen. Die Rückga-

bewerte unserer konstruierten Funktion sind jetzt wie gewünscht Feature Maps und die Vorhersagen. Wir sind nur daran interessiert, warum sich das Netz für die vorhergesagte Klasse entschieden hat, daher ermitteln wir deren Index (Zeile 90) und nutzen diesen anschließend, um den zugehörigen Outputwert zu ermitteln. Die Feature Maps haben vier Achsen. Die erste ist unsere künstliche Dimension, die wir hinzugefügt haben, da wir nur ein Bild haben. Die nächsten beiden stehen für die Größe der Map und die letzte für die Anzahl der Feature Maps.

```

87     img = tf.Variable(img);
88     with tf.GradientTape() as tape:
89         featureMaps, predictions = calcFeaturesAndPred(img[np.newaxis,...])
90         maxActivation = np.argmax(predictions[0])
91         predictedClassEntry = predictions[:, maxActivation]
```

Nachdem die Berechnungen ausgeführt sind, nutzen wir nun in Zeile 93 die auf dem Tape aufgezeichneten Daten, um die Gradienten aus Gleichung (11.12) zu berechnen. Die Syntax ist so, dass das erste Argument die Variable ist, die differenziert werden soll, und das zweite die, die nach der differenziert wird. In Zeile 94 bilden wir den Mittelwert für jede Feature Map. Hierbei ist es hilfreich, das axis-Argument gleich einem Tuple zu setzen. Die Werte (0, 1, 2) bedeuten, dass der Mittelwert über eben diese Achsen gebildet wird. Der Gradient hat die Dimensionen: (*Anzahl von Datensätzen, x-Auflösung der Feature Map, y-Auflösung der Feature Map, Anzahl der Feature Maps*). Die erste Dimension ist bekanntlich unsere künstliche, da wir nur ein Bild haben. Wird also über die ersten drei Achsen gemittelt, erhalten wir so viele Werte, wie wir Feature Maps haben, bzw. so viele Werte, wie die letzte Dimension angibt.

```

92
93     grads = tape.gradient(predictedClassEntry, featureMaps)
94     pooledGrads = np.mean(grads, axis=(0, 1, 2))
```

Nun multiplizieren wir gemäß Gleichung (11.12) die Feature Maps mit den Gewichten in Zeile 95 und erhalten $w_c^{(m)} \cdot A^{(m)}$. Anschließend bilden wir in Zeile 96 die Summe über alle Feature Maps.

```

95     weightedFeatures = featureMaps * pooledGrads
96     heatmapImg = np.sum(weightedFeatures, axis=-1).squeeze()
```

Am Schluss wenden wir ReLU an, was faktisch durch `np.maximum` geschieht. In den letzten zwei Zeilen der Funktionen geben wir die Werte zurück. Da wir aber die Heatmap als Bild weiterverarbeiten wollen, normieren wir diese zuvor vor. Sinnvolle Ansätze sind da auf den Bereich 0 bis 1 oder bis 255. Durch den Einsatz von Tape ist `predictions` eine TensorFlow-Variable. Mit der Methode `numpy` wandeln wir diese wieder um.

```

98     heatmapImg = np.maximum(heatmapImg, 0)
99     if np.max(heatmapImg) > 0 : heatmapImg /= np.max(heatmapImg)
100    return heatmapImg, predictions.numpy()
```

Jetzt haben wir eine Class Activation Map, müssen diese aber noch über unser ursprüngliches Bild legen. Hierzu müssen wir das Bild skalieren und ein paar weitere Anpassungen vornehmen. Da dies jetzt eher technisch ist, versuche ich kurz alles zusammenzufassen. `cm Reds` wandelt als Klasse skalare Daten in eine RGBA-Darstellung gemäß der `ColorMap` um. Rottöne passen hier schön zu dem Namen `Heatmap`, aber natürlich geht alles, was sequenziell ist,

und Ihren Geschmack trifft. Den Alpha-Kanal brauchen wir nicht, deshalb unterdrücken wir diesen mithilfe von [...; :3]. In PIL gibt es eine optisch schöne Methode zum Skalieren. Um diese nutzen zu können konvertieren wir etwas hin und her. Die Methoden dazu kommen aus der **image**-Klasse von Keras.

```
102 from matplotlib import cm
103 from PIL import Image as PILImage
104 from tensorflow.keras.preprocessing import image
105
106 def overlayHeatmap(img, heatmapImg):
107     heatmapImg = cm.Red(s(heatmapImg)[..., :3])
108     heatmapImg = image.array_to_img(heatmapImg)
109     heatmapImg = heatmapImg.resize(img.shape[:-1], resample=PILImage.BICUBIC)
110     heatmapImg = image.img_to_array(heatmapImg)
```

Die nächsten drei Zeilen sind nur da, falls Sie die gleiche Ausgabe haben wollen wie im Buch. Eigentlich ist es in bunt natürlich etwas hübscher.

```
111
112     imGray = 0.2989*img[:, :, 0] + 0.5870*img[:, :, 1] + 0.1140*img[:, :, 2]
113     imGray = cm.gray(imGray)[..., :3]
```

Nun geht es daran, die Heat bzw. Class Activation Map mit dem Bild zu überlagern. Das passiert einfach per Addition. Die Gewichte sorgen dafür, dass die Class Activation Map dominant ist und das Bild nur leicht im Hintergrund sichtbar ist. Wenn Sie es gerne anders haben wollen, ändern Sie es halt. Die beiden Zeilen bewirken, dass beide Bilder gleich skaliert sind, auch wenn sich das Verhalten der Konvertierungsmethoden mal ändert oder nicht gleich ist.

```
114
115     if np.max(imGray) <= 1: imGray = 255*imGray
116     if np.max(heatmapImg) <= 1: heatmapImg = 255*heatmapImg
117     superimposedImg = np.minimum(heatmapImg * 0.6 + 0.2*imGray, 255).astype(np.uint8)
118     return superimposedImg
```

Nun wollen wir das alles auch mal ausprobieren. Dazu greifen wir auf ein paar Bilder aus meinem privaten Fundus zurück. Diese sind im ZIP-File auf der Webseite enthalten. Sie können aber auch Bilder aus der Testmenge auswählen. Diese enthält aber noch eine kleine Gemeinheit, die ich mit Ihnen besprechen möchte. Ich nehme nämlich auch drei Bilder, die weder Katzen noch Hunde enthalten. Darüber hinaus noch eine freigestellte Katze, auf die wir später kommen.

```
119
120 import matplotlib.pyplot as plt
121 plt.rcParams.update({'figure.max_open_warning': 0})
122 imageList = ['hund1.jpg', 'hund2.jpg', 'hund3.jpg', 'hund4.jpg',
123             'katze1.jpg', 'katze2.jpg', 'katze3.jpg', 'katze4.jpg',
124             'ratte.jpg', 'luchs.jpg', 'wolf.jpg', 'katze1freigestellt.jpg']
```

Zunächst laden wir in einer Schleife alle Bilder und skalieren diese direkt beim Laden auf die Größe, die das CNN verarbeiten kann. Unser Netz ist trainiert, Voraussagen für Daten zu treffen, die auf den Bereich 0 bis 1 normiert wurden, entsprechend normieren wir das Bild bzgl. der Wert für die weiteren Schritte.



Die Vorhersage muss immer als Pipeline gedacht werden. Daten werden geladen, vorverarbeitet und dann die Prognose durchgeführt. Wenn die Quelle wechselt ist das eine ganz unangenehme Fehlerquelle. Vielleicht waren vorher Daten immer schon auf $[0,1]$ normiert und es erfolgte daher keine Normierung im Code. Wechselt die Quelle, bekommt man auf einmal z. B. Daten im Bereich $[0,255]$. Da die Bilddimensionen stimmen, gibt es keine Fehlermeldung, aber völlig sinnlose Vorhersagen. Solche Fehler zu finden, kostet manchmal Stunden!

```
125 for imageFile in imageList:
126     imgSize = CNN.input_shape[1:-1]
127     img = image.load_img(imageFile, target_size=imgSize)
128     img = image.img_to_array(img)/255.0
129     plt.figure(); plt.imshow(img)
```

Nun erstellen wir mit unserer Funktion die Heatmap und lassen uns gleichzeitig über die Vorhersage informieren.

```
130     hm, predictions = heatmap(img, CNN)
131     plt.figure(); plt.imshow(hm, cmap=cm.Red)
132     if np.argmax(predictions[0]) == 0: classtring = 'cat'
133     else: classtring = 'dog'
134     print(imageFile, ' predicted as ', classtring, ' with ', predictions)
```

Anschließend überlagern wir die Heatmap mit dem Original und speichern das Ergebnis als PNG, um es z. B. in einem Buch abdrucken zu können.

```
135     fusion = overlayHeatmap(img, hm)
136     plt.figure(); plt.imshow(fusion)
137     name = imageFile.split('.')[0]
138     plt.title(name+str(predictions))
139     name = 'heat'+name+'.png'
140     image.array_to_img(fusion).save(name, 'PNG')
```



Den Code oben kann man schlecht wiederverwenden. Kopieren Sie die beiden Funktionen `heatmap` und `overlayHeatmap` inklusive aller benötigten Einbindungen einmal in eine Datei `gradCAM.py`. Dann können wir diese später noch benutzen.

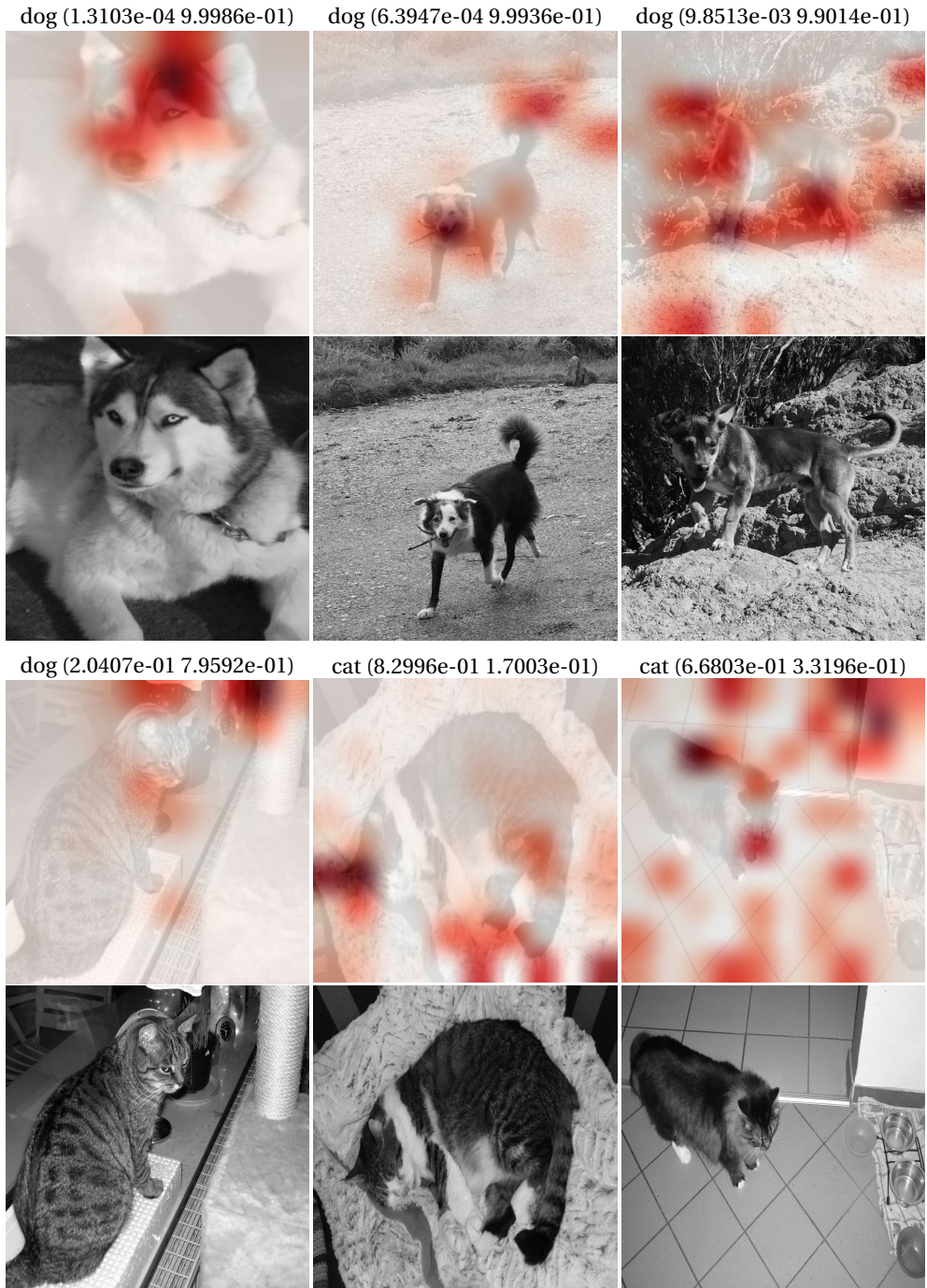


Abbildung 11.20 Heatmaps und Vorhersagen für Bilder mit Hunden und Katzen

Nun schauen wir mal, was wir rausbekommen haben. Wir wussten aus Abschnitt 11.3 schon, dass unser Netz ein Hundefreund ist und im Zweifel eher dazu tendiert, eine Katze zum Hund zu machen. Abbildung 11.20 zeigt die Prognosen und die Heatmaps für drei Hundebilder. Alle drei sind richtig als Hund klassifiziert und das auch mit einer hohen Sicherheit von Seiten des CNN. Man sieht, dass sich das Netz auf Schnauzen und Schwänze zu konzentrieren scheint. Das dritte Bild ist aber verdächtig. Es wurden viele Bereiche im Bild berücksichtigt, auf denen kein Hund zu sehen ist. Entweder sind mehr Hundebilder in freier Wildbahn im Archiv oder das Netz lässt sich von Strukturen im Hintergrund leicht ablenken. Solchen Klassifizierungen sollten wir mit etwas Vorsicht begegnen. Mal sehen, wie es bei den Katzen weitergeht.

Wie man aus Abschnitt 11.3 vermuten konnte, treten die Fehler eher bei den Katzen auf. Die erste Katze wurde mit beinahe 80% als Hund klassifiziert. Wir sehen aber auch, dass ähnlich wie bei der dritten Katze für die Klassifizierung eher der Hintergrund den Ausschlag gegeben hat.

cat (9.0749e-01 0.9250e-01)

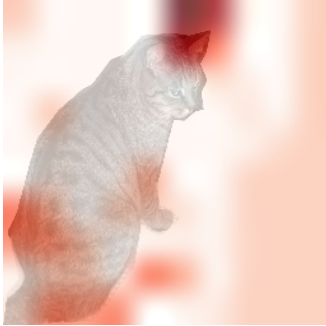


Abbildung 11.21 Heatmap & Vorhersage für freigestellte Katze

Das kann legitim sein, weil man z. B. ein Kamel eher nicht in der Antarktis vermutet, aber hier erscheint es weniger verständlich. Der Hintergrund lenkt scheinbar ab. Wie wäre es wohl ausgegangen, wenn die erste Katze ohne Hintergrund hätte eingeordnet werden sollen? Für den Test habe ich diese einmal unfachmännisch freigestellt und erneut klassifizieren lassen. Abbildung 11.21 zeigt eindrucksvoll, dass sich das Netz nun wieder auf Ohren und Fellaspekte konzentriert. Die Katze wird nun auch zu 90% als Katze erkannt.

Wie sieht es mit den eingebrachten Bildern einer Farbratte, eines Luchses und eines Wolfes aus? Biologisch wäre es plausibel, wenn sich das Netz beim Luchs für Katze, beim Wolf für Hund und bzgl. der Ratte sich das Netz als knappes Ergebnis für Katze oder Hund entscheiden würde. Wie man in Abbildung 11.22 deutlich sieht, geht unser Netz aber nicht nach der Biologie, sondern nach Mustern. Beim Wolf ist noch alles, wie wir es erwarten. Es schaut auf Gesicht und Körper und sagt *Hund*. Beim Luchs lässt es sich wieder vom Hintergrund ablenken und klassifiziert ihn ebenfalls als *Hund*. Entgegen der generellen Tendenz unseres Netzes wird die Farbratte mit großer Sicherheit zur Katze. Vermutlich weil der Spielplatz der beiden Haustiere sich ähnelt. Die Ratte jedenfalls kommt in der Heatmap kaum vor.



Machen Sie sich doch mal einen Spaß daraus und schicken Sie Bilder von Tieren mit und ohne Hintergrund durch das Netz und sehen Sie sich die Ausgaben genau an. Vielleicht bekommen Sie so ein besseres Gefühl für das Verhalten des CNN.

cat (8.5684e-01 1.4315e-01) dog (4.1199e-02 9.5880e-01) dog (4.0606e-02 9.5939e-01)



Abbildung 11.22 Heatmaps und Vorhersagen für Bilder mit nicht berücksichtigten Tieren

Nachdem die Idee einmal in der Welt war, wurde natürlich an unterschiedlichen Orten geschaut, ob man da nicht noch mehr rausholen kann. Ein Problem ist, dass die Größe der Feature Maps am Schluss automatisch die Genauigkeit der Lokalisation beeinflusst. Hätten wir noch ein paar mehr Poolings eingebaut, wäre noch weniger über gewesen, um herauszufinden, was interessant war. Ein anderes ist, dass wir händisch vorgehen müssen. Man muss Bilder als Mensch stichpunktartig analysieren und schauen, ob es einem plausibel vorkommt. Varianten wie Grad-CAM++, siehe [CSHB18], aus dem Jahr 2018 lösen diese Probleme noch nicht. Es geht eher um punktuelle Verbesserungen, um den Preis von mehr Heuristik bzw. weniger leicht nachvollziehbaren Formeln. Ich hoffe, dass Sie durch das Grad-CAM- bzw. GAP-Verfahren einen Einblick in die Möglichkeiten und Grenzen bekommen haben. Nun werden wir im nächsten Kapitel sehen, ob wir bzgl. unserer Hunde und Katzen nicht noch besser werden können und ggf. auch die starke Sensitivität für den Hintergrund etwas verringern können.

■ 11.5 Transfer Learning

Wir haben im Abschnitt 11.3 unseren Datenbestand durch Data Augmentation aufgewertet. Dies kann auch ein wenig helfen, wenn man nur auf vergleichsweise wenig Daten zum Training zurückgreifen kann bzw. wenn es darum geht, die Erkennung unempfindlicher gegenüber Rotationen zu machen. Wie wir schon diskutiert haben, kann man hiermit jedoch nur beschränkt neue Informationen dem Netz zuführen. Ein anderer Ansatz, um ggf. mit vergleichsweise ge-

Index

A

ablation study 576
Ablationsstudie 576
Accuracy 239
Action-Value-Funktion → Q-Function
activation function → Aktivierungsfunktion
Actor-Critic-Methods 552
Adagrad 246
Adam 246
Agent
– Actuators 482
– Condition-action rules 482
– Critic-Modul 484
– goal-based agents 481
– kognitiv 482
– learning agents 481
– Learning Element 484
– model-based reflex agent 481
– Performance Element 484
– Problem Generator 484
– robust 481
– Sensors 482
– simple reflex agents 481
– sozial 482
– State 483
– utility-based agents 481
Agglomerative Clusterverfahren 445
Aktivierungsfunktion 169
– linear 180
– Rectifier Linear Unit 226
– Sigmoidfunktion 178
– Tangens Hyperbolicus 178
aleatorische Unsicherheit 320
Anaconda Prompt 520
Anonymisierung 31
AutoML 470
Average Pooling 357
Average-Linkage 447
Axiome von Kolmogorow 79

B

Backpropagation 185
Bagging 329
Basis 106
– kanonisch 106
Basiswechsel 107
Batch Reinforcement Learning 535
Batch-Learning 188
Baumhöhe 136
bedingte Wahrscheinlichkeiten 80
Bellmansches Optimalitätsprinzip 492
bestärkenden Lernens 24
Bestärkendes Lernen 481
– Batch Reinforcement Learning 522
– Belohnung 487
– Environment 482
– Growing Batch Reinforcement Learnings
522
– Kumuliertiver Reward 489
– modellbasiert 492
– modellfrei 492
– Offline 522
– Off-Policy-Learning 515
– Online 522
– On-Policy-Learning 515
– optimale Aktion 491
– optimale Policy 490
– Policy 487
– Reward 487
– Strategie 487
– Value Function 489
Bias 94
Bias-Neuron 172
Big Data 19
Binäre Kreuzentropie 243
Binary Cross-Entropy 243
Boltzmann Policy 509
Boltzmann-Funktion 509
Boosting 329, 342
Bootstrap-Sample 330

C

C++ -Buildtools 520
Catastrophic Forgetting 401
Cauchy-Kriterium 504
Centroid-Method 447
Class Activation Maps 385
Classification Error 239
Cleverbot 16
Clipped Double Q-Learning 552
CNN 352
– Bias-Neuron 370
Complete-Linkage 447
Continual Learning 401
ConvNet 352
Convolutional Layer 355
Convolutional Neural Networks 352
Covariate Shift 231
CPython 75
Credit Assignment Problem 553
Cross-Correlation 365
Cross-Entropy → Kreuzentropie
Cross-Entropy-Error 240
Curse of Dimensionality 20, 121
CVXOPT 405

D

Data Augmentation 379
Dataset
– Acute Inflammations Data Set 83
– Bevölkerungsentwicklung der BRD 205
– Bike Sharing Data Set 158
– Boston Housing Dataset 114, 216, 220
– CIFAR-10 371
– Fisher's Iris Data Set 69
– MNIST 73, 252
– Mouse Dataset 426
– Two Moons 124
Datensatz 18
Datenschutz 31
Datensicherheit 31
DBSCAN 438
Deep Autoencoder 314
Deep Networks 181
Deep Q-Learning 561
Dendrogramm 445
Dichteverbundenheit 439
Dimension des Vektorraums 106

discount factor 489
Diskontierungsfaktor → Discount Factor
Divisive Clusterverfahren 445
Dropout 261
Dual Target Tests 92
Duale Formulierung 410
Duales Problem 410
Duck-Typing 45
Dunn Index 453
Dying ReLU 227
Dyna-Q 583

E

Eager Learner 24
Eager Learning 122
Early Stopping 208, 214
Einlagiges Perzeptron 172
ELU 228
EMNIST 401
Ensemble Learning 329
Ensemble Learnings 342
Entropie 139
Entscheidungsbaum
– Aufteilung 135
Entwicklungsknoten 585
Episode 524
epistemische Unsicherheit 320
Epoche 524
equivariant representation 368, 370
Ereignis 79
Ereignisraum 79
Ergebnisraum 78
Ergebnisse 79
Erwartungswert 93
Erzeugendensystem 105
Experience Replay 523
– Mini-Batch 524
External Path Length 136
Extrapolation 68, 205
eXtreme Gradient Boosting 349

F

Feature 18
Feature Importance 336, 337
Feature Map 354
Feedforward-Netze 181
Fehler

- statistisch 93
- systematische 93
- Feudal Reinforcement Learning 581
- Flattening 357
- Fluch der Dimensionalität → Curse of Dimensionality
- Formel von Lance und Williams 448
- Freiheitsgrade ein neuronales Netz 199
- Fully-connected Layer 220, 357
- Fully-connected Neural Network 220
- Fuzzifizier 435

- G**
- GAP 384
- GAP-Layer 384
- Gauß-Newton-Verfahren 114
- Gaußschen Glockenkurve 93
- Generalisierung 205
- Generative Adversarial Networks 313
- Geometrische Reihe 489
- GeoPandas 455
- Gewichtete Pfadlängensumme 136
- Gini Importance 337
- Gini Impurity 148
- Global Average Pooling 384
- GMRES 120
- GNU Octave 54
- Goodharts Gesetz 547
- Goodhart's Law 547
- Gradient 183
- Gradient Boosting 343
- Gramsche Matrix 415
- ϵ -greedy 495
 - Exploitation 495
 - Exploration 495
- Greedy-Algorithmus 297
- Grid World 496
- Growing Batch Reinforcement Learning 535
- Gut gestelltes Problem 251, 469

- H**
- Hard Sigmoid 227
- Hauptkomponentenanalyse 302
- Hebbsche Lernregel 173
- Hidden-Layer 179
- Hierarchie von Lernebenen 580
- Histogramm 95
- Hitchbot 16
- Huber-Loss-Function 243
- Hyperebene 115

- I**
- imbalanced-learn toolbox 326
- Imputation 281
- Incremental Learning 188
- Information Gain 141
- Informationsgehalt 138
- Input 354
- Input-Layer 179
- Intervallskala 83
- Inverse Pendel 531
- IPython 40

- J**
- Joint Action Learning 587
- Jupyter-Notebook 39

- K**
- Kaggle 28
- Kardinalskala 84
- Karush-Kuhn-Tucker-Bedingungen 408
- Kategoriale Merkmale 196
- Kategoriale Zielwerte 196
- Kausalität 87
- kd-Baum 125
- Keras 11, 219
 - AveragePooling2D 373
 - Callbacks 223
 - Conv2D 373
 - Early Stopping 222
 - EarlyStopping 223
 - evaluate 244, 254
 - flow_from_directory 381
 - glorot_normal 230
 - glorot_uniform 230
 - he_normal 231
 - he_uniform 231
 - image 389
 - ImageDataGenerator 378
 - Initializer 231
 - Keras function 362
 - lecun_uniform 230
 - load_model 222
 - Model 387

- Model Class API 387
- ModelCheckpoint 225
- Monitor-CallBack 224
- RandomNormal 230
- RandomUniform 230
- save 222
- Sequential Model 220, 221
- strides 366
- to_categorical 243, 253
- TruncatedNormal 230
- Kernel 354, 415
- Kernel Matrix 415
- Kernel Methods 405, 415
- Klassifikation 22
- Klassifizierungsproblem 23
- kmeans++ 434
- k-Nearest-Neighbor-Algorithmus 122
- K-Nearest-Neighbor-Consistency 454
- k-NN 122
- k-NN Classification 122
- k-NN Regression 122
- kNN-Multivalued Function 472
- Knowledge Discovery in Databases 16
- Konfusionsmatrix 91
- Konsistenz 453
- Koordinatenform 115
- Korrelation 87
- Korrelationskoeffizient →
 - Pearson-Korrelationskoeffizient
- Korrelationsmatrix 293
- Kovarianz 289, 292
- Kovarianzmatrix 303
- Kovariate Verschiebung 231
- Kreuzentropie 240
- Kreuzvalidierung 97
 - Holdout 98
 - k-fach 98
 - Monte Carlo-Wiederholungen 98
- Künstliche Intelligenz 15
 - schwache 15
 - starke 15

L

- L1-Regularisierung 255
- L2-Regularisierung 255
- Lagrange-Multiplikatoren 408
- LAPACK 120

- Lazy Learner 24
- Lazy Learning 122
- Least Squares Method 118
- Lernhierarchie 580
 - Aufgabe 578
 - Plan 578
 - Reaktion 579
 - Reflexe 579
 - Task 578
- Lernrate 189
- Linear unabhängig 106
- Lineare Separierbarkeit 171
- Linearer Kernel 415
- Linearkombination 103
- Lloyd-Algorithmus 429

M

- Mahalanobis-Distanz 111
- MajorClust 438
- Manhattan-Metrik 107
- Manhattan-Norm 107
- Margin 408
- Markov Decision Process 487
 - deterministisch 487
- Maslows Hammer 10
- MATLAB 54
- Matplotlib 63
 - add_subplot 71
 - array 64
 - Axes3D 72
 - axis-Objekt 71
 - cla 66
 - clf 66
 - cmap 73
 - fill_between 280
 - imshow 74
 - pcolormesh 76
 - plt 66
 - pyplot 66
 - scatter 71
 - set_xlabel 71
 - set_ylabel 71
 - tight_layout 71
 - twinx 280
- Max-Pooling 357, 370
- Mean Decrease in Impurity 336
- Mean Squared Error 240

Median 276, 277
 Mehrklassenklassifikation 203
 Mehrlagiges Perzeptron 181
 Mengen
 – disjunkt 81
 Merkmal 18
 Merkmalsausprägung 89
 Metrik 110
 Mittelwert der Stichprobe 94
 Model-based Reinforcement Learning 582
 Monte Carlo Tree Search 585
 Multiclass Classification 203
 Multi-Label Classification 203
 Multilayer Perceptron → Mehrlagiges Perzeptron
 Multiples Testen 338

N

Nash Q-Learning 588
 Natürlicher Nullpunkt 84
 Near Miss Algorithm 325
 Neural Fitted Q Iteration 539
 Neuronales Netz
 – fully connected 181
 – Initialisierung der Gewichte 190
 – Sättigung 190
 – vollvermascht 181
 Nominalskala 83
 Norm 108
 Normierung 276
 Numerische Stabilität 511
 NumPy 51
 – arange 54
 – argpartition 62, 125
 – argsort 62
 – array 52
 – Array Broadcasting 60
 – Array Slicing 55
 – choice 58
 – Deep Copy 56
 – delete 58
 – Erzeugen von Arrays 54
 – Erzeugen von Vektoren 54
 – flatnonzero 57
 – ix_ 57
 – linalg 62
 – linalg.norm 62

– loadtext 70
 – maximum 226
 – meshgrid 75
 – partition 62
 – poly1d 65
 – rand 58
 – randint 58
 – ravel 76
 – reshape 55
 – seed 59
 – set_printoptions 59
 – size 57
 – sort 62
 – View 56
 – Zufallszahlen 58

O

Object Detection 401
 Ockhams Rasiermesser 64
 Offline-Learning 189
 Offset 115
 One-Hot Encoding 197
 One-Hot-Codierung 197
 One-Hot-Encoding 239
 Online-Learning 189
 On-Neuron 172
 OpenAI Gym 519
 OpenAI-Gym 495
 OpenCV 27
 OPTICS 438
 Ordinalskalar 83
 Outlier Detection 26
 Out-of-Bag-Error 331
 Output-Layer 179
 Overfitting 86, 209
 Oversampling 325

P

Pandas
 – concat 276
 – corr 294
 – DataFrame 264, 267
 – describe 273, 275
 – drop 265
 – factorize 274
 – head 268
 – hist 271

- iloc 265
- isna 282
- loc 266
- merge 270
- notna 282
- replace 273
- select_dtypes 276
- Series 264
- sum 282
- tail 268
- unique 272
- Parameter Sharing 368, 369
- Paris-Metrik 111
- Partially Observable Markov Decision Process 507
- Partielle Ableitung 183
- Pearson-Korrelationskoeffizient 87, 292
- Percentiles 275
- Pfadlängensumme 136
- pickle 525
- p-Normen 108
- Polygonzug 64
- POMDP → Partially Observable Markov Decision Process
- Pooling 356, 370
- Post-Pruning 164, 165
- Predictive Maintenance 359
- Pre-Pruning 164
- Primärhypothese 338
- Principal Component Analysis 302
- Pruning 164, 165
- Pseudonymisierung 31
- Pygame 561
- Python
 - Ausnahmebehandlung 49
 - Built-in Types 42
 - def 44
 - deque 44
 - Dictionaries 42
 - docstrings 45
 - dynamische Typisierung 45
 - enumerate 49
 - except 49
 - Exception 49
 - finally 49
 - import 46
 - importlib 47

- Introspection 40
- Lists 42
- None 45
- pretty printer 59
- Private Methode 50
- PYTHONPATH 47
- Referenzen 42
- return 45
- str 45
- Strings 42
- try 49
- Tuples 42, 44
- type 41
- Unpacking 46
- with 70

Q

- Q-Function 492
- Quantil 275, 277

R

- Radial Basis Function → Radiale Basisfunktionen
- Radiale Basisfunktionen 415
- RainbowDQN 576
- Random Forest 330
- Rationalskala 84
- Record 18
- Rectifier Linear Unit 226
- Recurrent Neural Networks 244
- Reduced-Error-Ansatz 165
- Regressionsproblem 23
- Reinforcement Learning → Bestärkendes Lernen
- Rekurrentes neuronales Netz 244
- ReLU
 - see Rectifier Linear Unit 226
- Residual Blocks 395
- Residuenquadratsumme 117
- RMSProp 246
- RoboCup Simulation League 28
- Runges Phänomen 67

S

- Satz von Bayes 80
- Satz von der totalen Wahrscheinlichkeit 81
- SavedModel 222

Scatter Plots 71
 Schlecht gestelltes Problem 251, 469
 Schlupfvariablen 411
 scikit-learn 11, 164, 346
 – DecisionTreeClassifier 162
 – DecisionTreeRegressor 162
 – GaussianNB 99
 – Imputer 289
 – KNeighborsClassifier 128
 – KNeighborsRegressor 128
 – LinearRegression 120
 – LinearSVC 418
 – MLPRegressor 218
 – PCA 312
 – RFE 301
 – SVC 418
 SciPy 51
 – dendrogram 449
 – interpolate 66
 – lagrange 66
 – linkage 448
 – pdist 448
 Seaborn 27
 Semantische Segmentierung 403
 Semi-überwachtes Lernen 404
 Separation 453
 Sequential Backwards Selection 298
 Sequential Forward Selection 300
 sequenzielle Rückwärtsauswahl 298
 sequenzielle Vorwärtsauswahl 300
 Sigmoidfunktion 178
 Sign Language MNIST 398
 Significant (ASL) Sign Language Alphabet
 Dataset 398
 Simulation Data Mining 20
 Single-Linkage 446
 Skalenniveaus 83
 slack variables → Schlupfvariablen
 Softmax 240
 Softmax-Funktion 509
 Softplus 228
 Softwarepatenten 261
 Sparse Interactions 368
 sparse interactions 355
 Spyder 39
 – %matplotlib 74
 – Command History 40

– Introspection 40
 – Tab Completion 40
 – Variable Explorer 39
 Standardabweichung 93
 – empirische 94
 Standardisierung 278
 Stichprobenvarianz 94
 Stochastic Gradient Descent 189
 Strategie
 – deterministisch 487
 Strukturierte Daten 18
 Studentisierung 278
 Stützstellen 64
 Subagging 330
 Summe der Fehlerquadrate 66, 117
 Support Vector Machines 405
 – one-vs-all 416
 – one-vs-one 416
 Support Vectors 408
 SVM → Support Vector Machines
 SymPy 27

T

Tangens Hyperbolicus 178
 TensorFlow 219
 TensorFlow Agents 521
 Testmenge 85, 209
 Theano 219
 time serie → Zeitreihen
 Tit for Tat 586
 Toeplitz-Matrizen 367
 TORCS 27
 tqdm 535
 Tractable Tree Search 586
 Trainingsmenge 85, 208
 Transfer Learning 394
 Transparenz 29
 Trustworthy AI 30
 Turing-Test 16

U

Überanpassung → Overfitting
 Überwachtes Lernen 21
 UCI Machine Learning Repository 28
 Undersampling 325
 Unstrukturierte Daten 18

U

– affinen 405

Unüberwachtes Lernen 25

V

Validierungsmenge 85, 164, 208

Vektor von Bias-Neuronen 220

Verhältnisskala 84

Verzerrung 94

W

Wahrscheinlichkeit

– gemessene 79

Weighted External Path Length 136

X

XBBoost 342

XGBoost 349

Y

YOLO 403

Z

Zeitreihen 352

Zero Padding 366

Zufallsbeobachtung 78

Zufallsexperiment 78

Zurücklegen 330

Zyklus 524