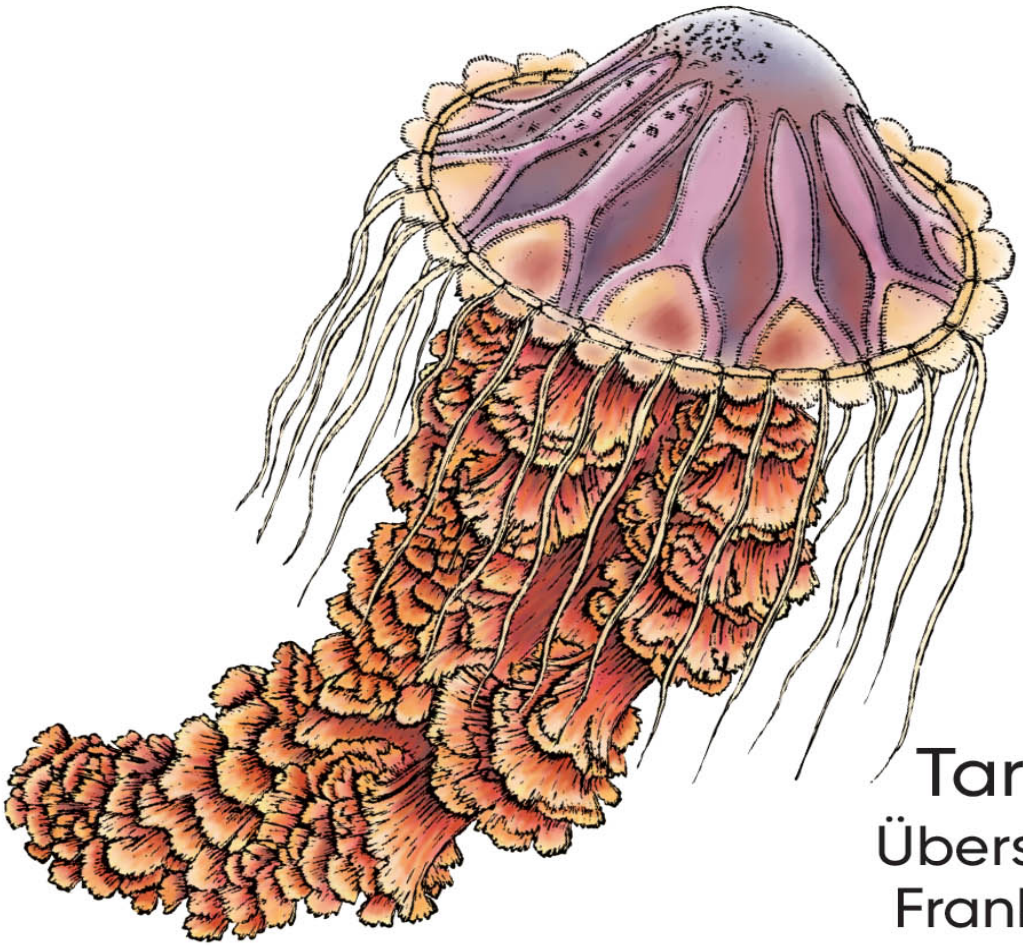
O'REILLY®

GANs mit PyTorch selbst programmieren

Ein verständlicher Einstieg in Generative Adversarial Networks



Tariq Rashid
Übersetzung von
Frank Langenau

GANs mit PyTorch selbst programmieren

Ein verständlicher Einstieg in Generative Adversarial Networks

Tariq Rashid

Deutsche Übersetzung von Frank Langenau

O'REILLY®

Tariq Rashid

Lektorat: Alexandra Follenius
Übersetzung: Frank Langenau
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-147-9
PDF 978-3-96010-393-6
ePub 978-3-96010-394-3
mobi 978-3-96010-395-0

1. Auflage 2020

Title of the English original: *Make Your First GAN With PyTorch*
ISBN 979-8624728158

Wieblinger Weg 17
69123 Heidelberg

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint
»O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: kommentar@oreilly.de.

5 4 3 2 1 0

Inhalt

Teil III Komplexere GANs

9 Convolutional GANs

10 Konditionierte GANs

Fazit

Anhänge

A Ideale Verlustwerte

Einführung

Künstliche Intelligenz explodiert!

Maschinelles Lernen und künstliche Intelligenz (KI) sind in den letzten Jahren regelrecht explodiert, und alle paar Monate vermelden die Nachrichten außergewöhnliche Leistungen.

Nicht nur, dass Ihr Smartphone Sie versteht, wenn Sie mit ihm sprechen, es kann auch recht gut zwischen verschiedenen menschlichen Sprachen übersetzen. Selbstfahrende Autos können inzwischen so sicher fahren wie Menschen. Und Maschinen sind heute in der Lage, manche Krankheiten genauer und früher zu diagnostizieren als erfahrene Ärzte.

Das alte chinesische Spiel Go wird seit 3.000 Jahren gespielt, und obwohl die Regeln einfacher sind als beim Schach, ist das Spiel selbst viel komplexer und verlangt längerfristige Strategien. Erst vor Kurzem waren Entwickler in der Lage, mit einem System für maschinelles Lernen erstmals gegen einen Weltmeister zu spielen und zu gewinnen. Dieses System für maschinelles Lernen hat

zudem neue Spielstrategien entdeckt, von denen wir glauben, dass Menschen sie in diesen 3.000 Jahren nicht hätten finden können!

Jenseits dessen, dass KI-Systeme lernen können, Aufgabe ausführen, ist die Entdeckung neuer Strategien ein immenser Erfolg auf dem Gebiet des maschinellen Lernens.

Kreative KI

Im Oktober 2018 hat das renommierte Aktionshaus Christies ein Porträt für 432.500 Dollar verkauft. Dieses Gemälde wurde nicht von einer Person gemalt, sondern von einem neuronalen Netz generiert. Ein mit KI gemaltes Porträt, das für fast eine halbe Million Dollar verkauft wurde, ist ein Meilenstein in der Geschichte der Kunst.

Dieses neuronale Netz wurde mit einer neuen und spannenden Technik trainiert, dem sogenannten *Adversarial Training* (von engl. adversarial – gegnerisch). Die Architektur wird als *Generative Adversarial Networks*, kurz GANs, bezeichnet (etwa »erzeugende gegnerische Netzwerke«).

GANs stoßen auf großes Interesse, speziell im Bereich der kreativen Technik, weil sie Bilder erzeugen können, die sehr plausibel aussehen. Diese Bilder entstehen weder durch einfaches Kopieren und Einfügen von Teilen der Trainingsbeispiele noch durch unscharf gemittelte Trainingsdaten. Das ist es, was GANs von den meisten anderen Formen des maschinellen Lernens unterscheidet. GANs lernen, Bilder auf einer Ebene zu erzeugen, die weit über dem bloßen Replizieren oder Mitteln von Trainingsdaten liegt.

Yann LeCun, einer der weltweit führenden Forscher auf dem Gebiet der neuronalen Netze, hat GANs als »die coolste Idee des Deep Learning in den letzten 20 Jahren« bezeichnet.

GANs sind neu

Im Vergleich zu den jahrzehntelangen Forschungen und Verfeinerungen auf dem Gebiet der neuronalen Netze haben GANs erst 2014 mit dem inzwischen bahnbrechenden Paper von Ian Goodfellow die Aufmerksamkeit auf sich gezogen.

Das heißt, GANs sind ziemlich neu, und man beginnt gerade erst, die kreativen Möglichkeiten zu erforschen. Das bedeutet aber auch, dass wir noch nicht ganz verstehen, wie sie sich so effektiv trainieren lassen, wie wir es bei herkömmlichen Netzen bereits können. Wenn sie funktionieren, dann spektakulär gut. Aber viel zu oft scheitern sie. Die Arbeitsweise von GANs und die Gründe, die dazu führen können, dass sie scheitern, werden derzeit sehr intensiv erforscht.

Für wen dieses Buch gedacht ist

Dieses Buch richtet sich an alle, die erste Schritte unternehmen wollen, um das Wesen von GANs und deren Arbeitsweise zu verstehen. Es eignet sich auch für jeden, der lernen möchte, wie man sie mit Industriestandardtools tatsächlich erstellt.

In dieser Einführung wird versucht, eine verständliche, einfache Sprache zu verwenden und anhand zahlreicher

Bilder Ideen visuell zu erklären. Ich werde unnötige Fachausdrücke vermeiden und mathematische Gleichungen nur wenn unbedingt nötig verwenden.

Mein Ziel ist es, möglichst vielen Lesern mit ganz unterschiedlichem Hintergrund GANs nahezubringen und sie dafür zu begeistern, eigene GANs zu erstellen.

Dieses Buch versucht nicht, jedes Thema erschöpfend zu behandeln oder eine Enzyklopädie für GANs zu sein. Es deckt absichtlich nur das Minimum ab, damit Sie eine solide Grundlage bekommen, auf der Sie zu weiterführenden Erkundungstouren aufbrechen können.

Teilnehmer an Kursen für maschinelles Lernen werden in diesem Buch eine gute Basis für das weitere Studium von GANs finden.

Neuronale Netze

GANs bestehen aus neuronalen Netzen. Auch wenn dieser Leitfaden hierzu eine kleine Auffrischung bringt, ist mein vorheriges Buch *Neuronale Netze selbst programmieren* (O'Reilly 2017) als behutsamste Einführung in neuronale Netze und ihre Arbeitsweise zu empfehlen. Es bietet auch eine Einführung in die Analysis und den Gradientenabstieg, was für die bevorstehende Reise in GANs nützlich ist.

- *https://www.oreilly.de/buecher/12892/9783960090434-neuronale-netze-selbst-programmieren.html*

Darüber hinaus enthält es eine Einführung in die Programmierung mit Python, die gerade genug abdeckt, um einfache neuronale Netze in eigener Regie aufzubauen.

Wie Sie dieses Buch verwenden

Ein Konzept lernt und versteht man am besten, wenn man es praktisch anwendet – Learning by Doing. Deshalb entwickelt dieses Buch Ideen und Theorien rund um eine praktische Schritt-für-Schritt-Reise.

Dieser Leitfaden wird Sie begleiten auf dieser Reise, auf der wir manchmal scheitern, bevor wir eine Lösung finden. Zu scheitern und sich durch die Abhilfemaßnahmen durchzuarbeiten, ist eine echte Erfahrung, und das ist viel wertvoller als die bloße Lektüre theoretischer Anleitungen zu GANs.

Freie und Open-Source-Software

Alle hier vorgestellten Tools und Dienste, mit denen Sie Ihre eigenen GANs aufbauen können, sind entweder kostenlos oder Open Source und kostenlos. Dies ist wichtig, um möglichst wenige Menschen davon auszuschließen, etwas über neuronale Netze und GANs zu lernen und diese selbst zu erstellen.

Python ist eine der beliebtesten und am leichtesten zu erlernenden Programmiersprachen und hat sich als Standard im maschinellen Lernen und der KI etabliert. Zudem gibt es eine lebhafte globale Community und ein gesundes Ökosystem von Bibliotheken.

Google bietet derzeit eine kostenlose webbasierte Python-Umgebung namens *Google Colab* an, was bedeutet, dass Sie weder Python noch irgendeine andere Software

installieren müssen. Sie können leistungsstarke neuronale Netz vollständig in der Infrastruktur von Google entwickeln und ausführen, wofür Sie lediglich einen modernen Webbrowser benötigen, der auf einem recht bescheidenen Computer oder Laptop läuft.

PyTorch ist eine Erweiterung von Python, mit der sich Modelle für maschinelles Lernen einfach entwerfen, erstellen und ausführen lassen. Neben TensorFlow gehört es zu den beliebtesten Frameworks für maschinelles Lernen.

Alle diese Tools zählen auch zum Industriestandard, sodass Sie wertvolle wiederverwendbare Fähigkeiten erwerben.

Anmerkung des Autors

Ich sehe meine Mission als gescheitert an, wenn irgendein Leser Schwierigkeiten hat, zu verstehen, was GANs sind und wie sie trainiert werden. Ich hätte auch versagt, wenn Sie nicht in der Lage sind, Ihr eigenes einfaches GAN zu erstellen.

Der Inhalt dieses Buchs ist mit einer Reihe von Studenten und Entwicklern getestet worden. Wenn also irgendetwas nicht schlüssig erscheint, kontaktieren Sie mich bitte über *twitter@myoneuralnet*, per E-Mail unter *makeyourownneuralnetwork@gmail.com* oder auf GitHub:

- *https://github.com/makeyourownneuralnetwork/gan*

Zusätzliche Erörterungen und Antworten auf interessante Leserfragen finden Sie im Blog, der dieses und das vorherige Buch begleitet:

- *https://makeyourownneuralnetwork.blogspot.com*

Abschließend möchte ich Ihnen wärmstens empfehlen, sich Ihrer lokalen Community für maschinelles Lernen oder algorithmische Kunst anzuschließen. Lernen in einer Gruppe ist viel effektiver, denn es macht Spaß, seine Arbeit mit anderen zu teilen und sich von dem inspirieren zu lassen, was andere sich ausdenken.

Viel Spaß dabei!

TEIL I

PyTorch und neuronale Netze

Zuerst lernen Sie PyTorch kennen und setzen es dann in der Praxis ein, indem Sie einen einfachen Klassifizierer für Bilder erstellen und Ihr Wissen zu neuronalen Netzen auffrischen.

KAPITEL 1

Grundlagen von PyTorch

In meinem letzten Buch *Neuronale Netze selbst programmieren* haben wir einfache, aber effektive neuronale Netze erstellt, und zwar ausschließlich mit Python und der Bibliothek NumPy für das Verarbeiten von Datenarrays.

Auf beliebte Frameworks wie PyTorch und TensorFlow für das Erstellen von neuronalen Netzen haben wir verzichtet, weil es wichtig war, die Netze von Grund auf neu aufzubauen, um ihre Funktionsweise wirklich zu verstehen.

Diese ganze Arbeit, die wir zu Fuß erledigen mussten, macht deutlich, dass der Aufbau größerer Netzwerke eine mühsame Aufgabe werden könnte. Einer der aufwendigsten Bereiche ist die Berechnung der Beziehung zwischen dem Fehler, der durch Backpropagation zurückgegeben wurde, und den Gewichten in unserem Netz. Wenn wir das Netz verändern, müssen wir möglicherweise die gesamte Arbeit noch einmal absolvieren.

Hier werden wir *PyTorch* einsetzen, weil uns diese Bibliothek eine Menge Routinearbeiten abnimmt, sodass wir uns auf den Entwurf unserer Netze konzentrieren können.

Zu den leistungsfähigsten und komfortabelsten Features von PyTorch gehört, dass die Bibliothek sämtliche Berechnungen für uns erledigt, egal welche Gestalt oder Größe das Netz hat, das wir uns ausdenken. Und wenn wir das Design unseres Netzes verändern, passt PyTorch die Berechnungen automatisch an, ohne dass wir Bleistift und Papier auspacken müssen, um die Gradienten erneut zu berechnen.

Außerdem hat man sich bei PyTorch wirklich sehr darum bemüht, dem Look-and-feel von normalem Python zu entsprechen. Das bedeutet, es ist leicht zu erlernen, wenn Sie Python bereits kennen, und es gibt weniger Überraschungen, wenn Sie damit arbeiten.

Google Colab

Wir haben im Buch *Neuronale Netze selbst programmieren* Code mithilfe der webbasierten Python-Notebooks geschrieben, die auf unserem eigenen Computer gelaufen sind. Jetzt verwenden wir Python-Notebooks, die der kostenlose Dienst *Colab* von Google bereitstellt und die unseren Code auf den Google-eigenen Computern ausführen.

Der Zugriff auf Colab-Dienste von Google erfolgt gänzlich über einen Webbrowser. Es ist nicht erforderlich, irgendwelche Software auf dem eigenen Computer oder Laptop zu installieren.

Bevor wir loslegen, sollten Sie sich mit einem Google-Konto anmelden. Wenn Sie über ein Gmail- oder YouTube-Konto verfügen, ist dies Ihr Google-Konto. Haben Sie noch kein

Google-Konto eingerichtet, können Sie eines über den folgenden Link erstellen:

- *https://accounts.google.com/signup*

Sobald Sie angemeldet sind, aktivieren Sie den Colab-Dienst von Google, indem Sie den diesen Link besuchen:

- *https://colab.research.google.com*

Von dieser Seite aus gelangen Sie zu einem Beispiel-Python-Notebook. Wählen Sie im Menü *File* den Eintrag *New Python 3 notebook,* um ein neues Notebook anzulegen (siehe Abbildung 1-1).

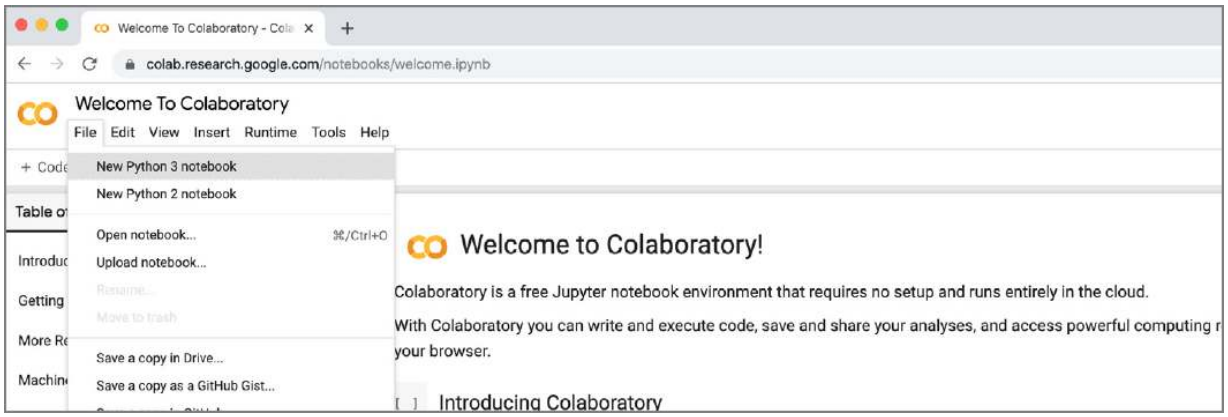


Abbildung 1-1: Die Startseite von Google Colab

Es erscheint ein leeres Python-Notebook (siehe Abbildung 1-2), das wir sofort verwenden können.

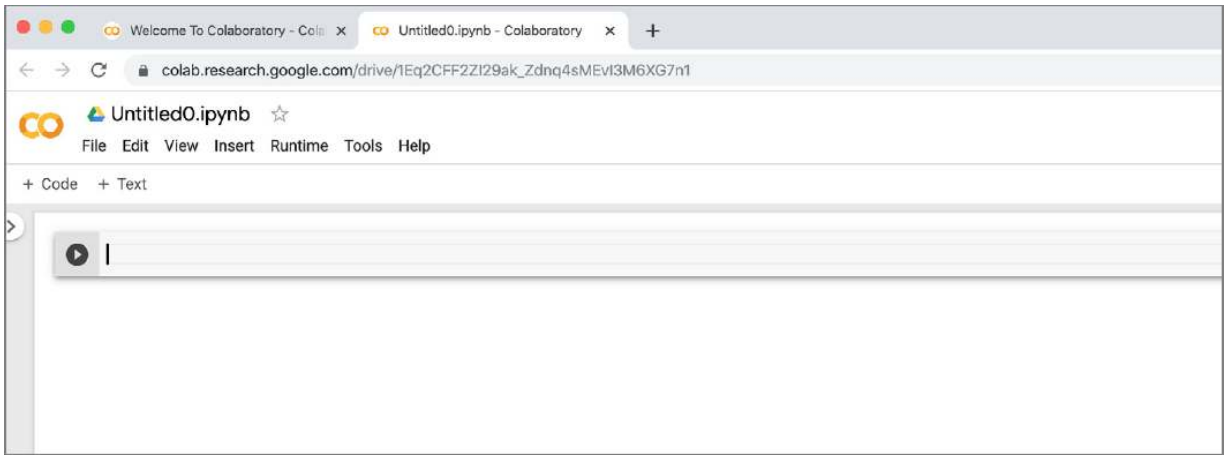


Abbildung 1-2: Ein neues leeres Python-Notebook

Wenn Sie sich auf einem separaten Browser-Tab den Google-Dateispeicher *Drive* ansehen, finden Sie einen neuen Ordner namens *Colab Notebooks*. Dies ist der Ordner, in dem neue Python-Notebooks standardmäßig gespeichert werden.

Abbildung 1-3 zeigt ein neues Notebook mit dem Namen *Untitled0.ipynb*.

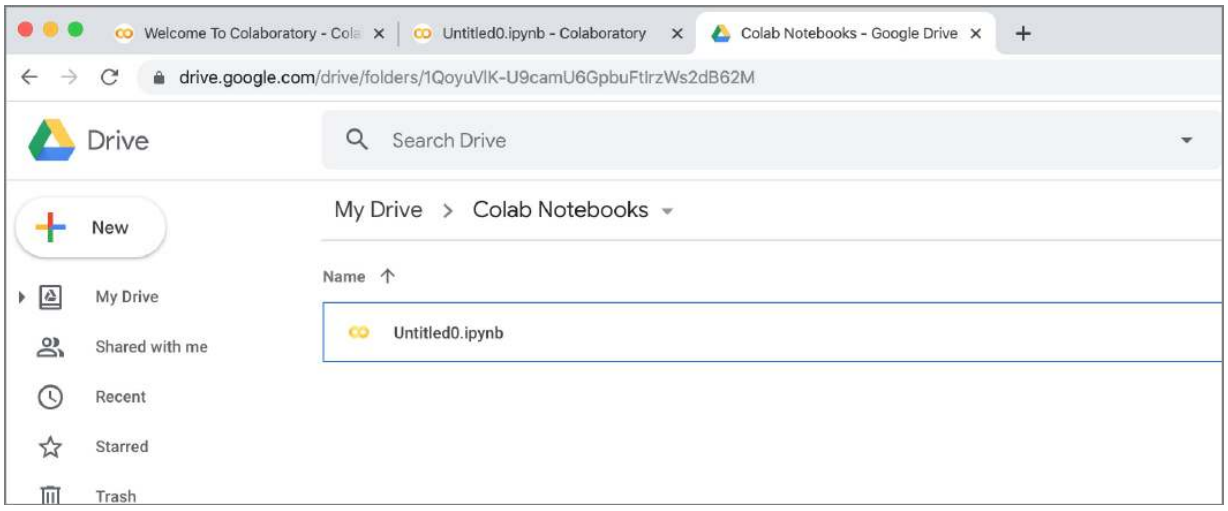


Abbildung 1-3: Ein neues Notebook mit dem Namen »Untitled0.ipynb«

Wir überprüfen nun, ob wir Python-Code ausführen können. Tippen Sie in die erste Zelle den folgenden

einfachen Code ein:

```
2 + 3
```

Klicken Sie auf die Schaltfläche mit dem Abspielen-Symbol links neben der Zelle, um den Code auszuführen. Wenn Sie den Colab-Dienst bisher noch nicht genutzt haben, kann es eine Weile dauern, bis die erste Python-Anweisung ausgeführt wird, da Google einen Moment braucht, um eine virtuelle Maschine zu starten und Ihr Notebook mit ihr zu verbinden.

Schließlich sollte die Antwort 5 erscheinen, wie Abbildung 1-4 zeigt.

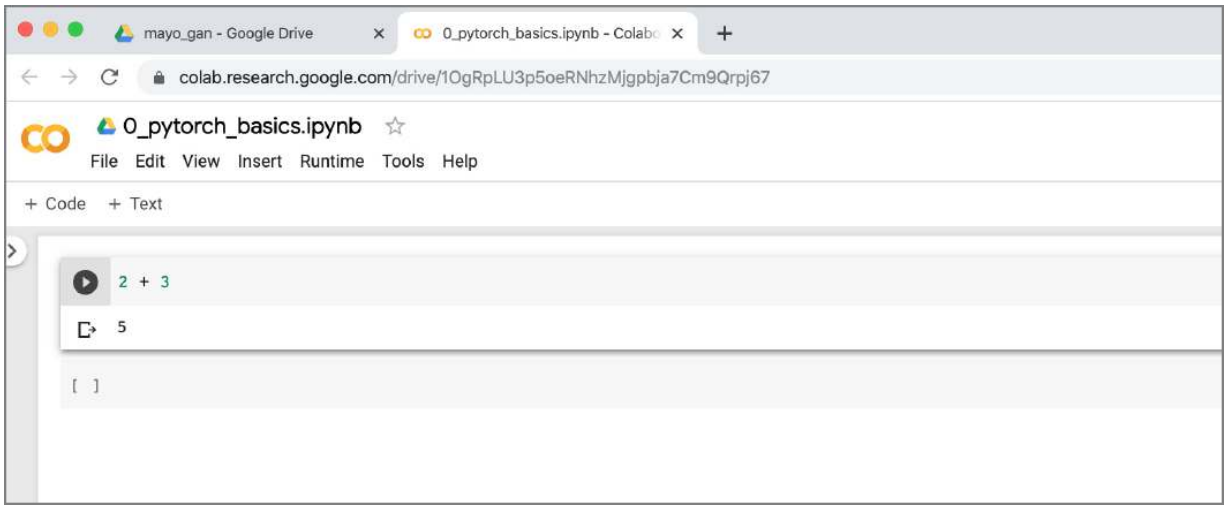


Abbildung 1-4: Das Ergebnis des ausgeführten Codes

Großartig! Alles funktioniert, und wir sind bereit, mehr über PyTorch zu erfahren.

PyTorch-Tensoren

Bevor wir PyTorch verwenden können, müssen wir das Python-Modul torch importieren. Erfreulicherweise hält der Colab-Service von Google viele dieser beliebten Bibliotheken für maschinelles Lernen für uns bereit, PyTorch eingeschlossen. Wir müssen die Bibliotheken nur noch importieren, um sie unmittelbar einsetzen zu können. Es ist nicht erforderlich, einen komplizierten Installationsprozess zu durchlaufen.

Geben Sie den folgenden Code in die erste Zelle ein und führen Sie ihn aus:

```
import torch
```

Um PyTorch zu verstehen, bietet es sich an, seine grundlegenden Informationseinheiten mit reinem Python zu vergleichen. In reinem Python speichern wir Zahlen in Variablen. Diese Variablen können wir wie mathematische Symbole verwenden, um neue Werte zu berechnen und diese in neuen Variablen zu speichern, wenn wir das wünschen.

Sehen Sie sich dazu den folgenden einfachen Python-Code an:

```
# Normale Python-Variablen

x = 3.5

y = x*x + 2

print(x, y)
```

Wir erzeugen eine Variable `x` und geben ihr den Wert `3.5`. Dann erzeugen wir eine neue Variable `y` und geben ihr einen Wert, der aus dem Ausdruck `x*x + 2` berechnet wird,

was `(3.5*3.5) + 2` oder `14.25` ist. Schließlich geben wir die Werte von `x` und `y` aus.

Tippen Sie den Code in eine neue Zelle ein und führen Sie ihn aus. Abbildung 1-5 zeigt, wie das Ergebnis aussehen sollte.

Abbildung 1-5: Die Verwendung einer Variablen

PyTorch hat eine eigene Form von Variablen, um Zahlen zu speichern – die sogenannten *PyTorch-Tensoren*. Mit den folgenden Anweisungen erstellen Sie einen sehr einfachen Tensor:

```
# Einfacher PyTorch-Tensor

x = torch.tensor(3.5)

print(x)
```

Wir erzeugen hier etwas, das `x` genannt wird. Dieses `x` ist ein PyTorch-Tensor, der mit dem Wert `3.5` initialisiert wird.

Geben Sie den Code ein und führen Sie ihn aus, um zu sehen, was die Ausgabe von `x` bewirkt.

Abbildung 1-6: Einen PyTorch-Tensor erzeugen und ausgeben

Die Ausgabe zeigt, dass der numerische Wert `3.5000` beträgt, aber auch, dass er in einem PyTorch-tensor enthalten ist. Es ist nützlich zu wissen, in welcher Art von Container diese Zahl gespeichert ist.

Führen wir nun einige einfache arithmetische Berechnungen mit diesem Tensor aus. Geben Sie in die nächste Zelle den folgenden Code ein:

```
# Einfache Arithmetik mit Tensoren

y = x + 3

print(y)
```

Hier erzeugen wir aus dem Ausdruck `x + 3` eine neue Variable `y`. Eben haben wir `x` als PyTorch-Tensor mit dem Wert `3.5` erzeugt. Welchen Wert wird also y haben?

Probieren Sie es aus.

Abbildung 1-7: Arithmetik mit einem Tensor

Wie Abbildung 1-7 zeigt, hat `y` den Wert `6.5`, was Sinn ergibt, denn `3.5 + 3 = 6.5`. Außerdem sehen wir, dass `y` ebenfalls ein PyTorch-Tensor ist.

Sicherlich erinnern Sie sich daran, dass auch NumPy-Arrays in der gleichen Weise funktionieren. Diese Vertrautheit kommt uns entgegen, und durch die Übereinstimmung mit NumPy ist es zudem einfacher, PyTorch zu erlernen.

Automatische Gradienten mit PyTorch

Sehen wir uns nun an, wie sich PyTorch von reinem Python und NumPy abhebt und es so besonders macht. Der folgende Code erzeugt genau wie zuvor einen Tensor x, dieses Mal aber geben wir PyTorch eine zusätzliche Option requires_grad= True mit. Wir werden bald sehen, was diese Option bewirkt.

```
# PyTorch-Tensor

x = torch.tensor(3.5, requires_grad=True)

print(x)
```

Führen Sie den Code aus und sehen Sie sich an, was für x ausgegeben wird (siehe Abbildung 1-8).

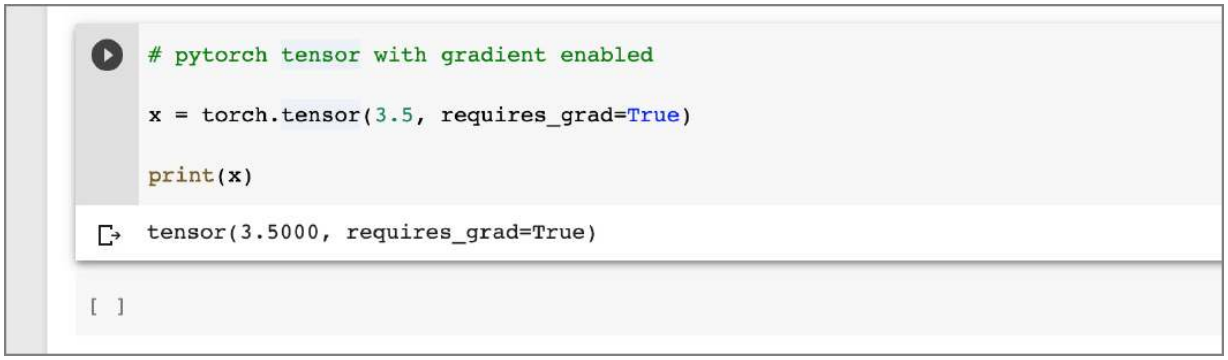


Abbildung 1-8: Eine Tensoroperation mit einer zusätzlichen Option

Wie Abbildung 1-8 zeigt, hat `x` den Wert `3.5000` und ist vom Typ tensor. Aus der Ausgabe geht auch hervor, dass für den Tensor `x` die Option requires_grad auf True gesetzt ist.

Wir erzeugen nun wie zuvor eine neue Variable y aus x, dieses Mal aber mit einem anderen Ausdruck:

```
# y wird als Funktion von x definiert

y = (x-1) * (x-2) * (x-3)

print(y)
```

Der Code berechnet `y` aus dem Ausdruck `(x-1) * (x-2) * (x-3)`. Führen Sie den Code aus.

```
[34] # pytorch tensor with gradient enabled

     x = torch.tensor(3.5, requires_grad=True)

     print(x)

     tensor(3.5000, requires_grad=True)

     # y is defined as a function of x

     y = (x-1) * (x-2) * (x-3)

     print(y)

     tensor(1.8750, grad_fn=<MulBackward0>)

[ ]
```

Abbildung 1-9: Eine Variable als Funktion definieren

Wie Abbildung 1-9 zeigt, beträgt der Wert von y wie erwartet `1.8750`. Das ergibt sich daraus, dass `x` gleich `3.5` ist und somit `(3.5-1) * (3.5-2) * (3.5-3)` das Ergebnis `1.8750` liefert.

Abbildung 1-10 mit dem Graphen der Funktion y = (x-1) * (x-2) * (x-3) veranschaulicht, was wir eben berechnet haben.

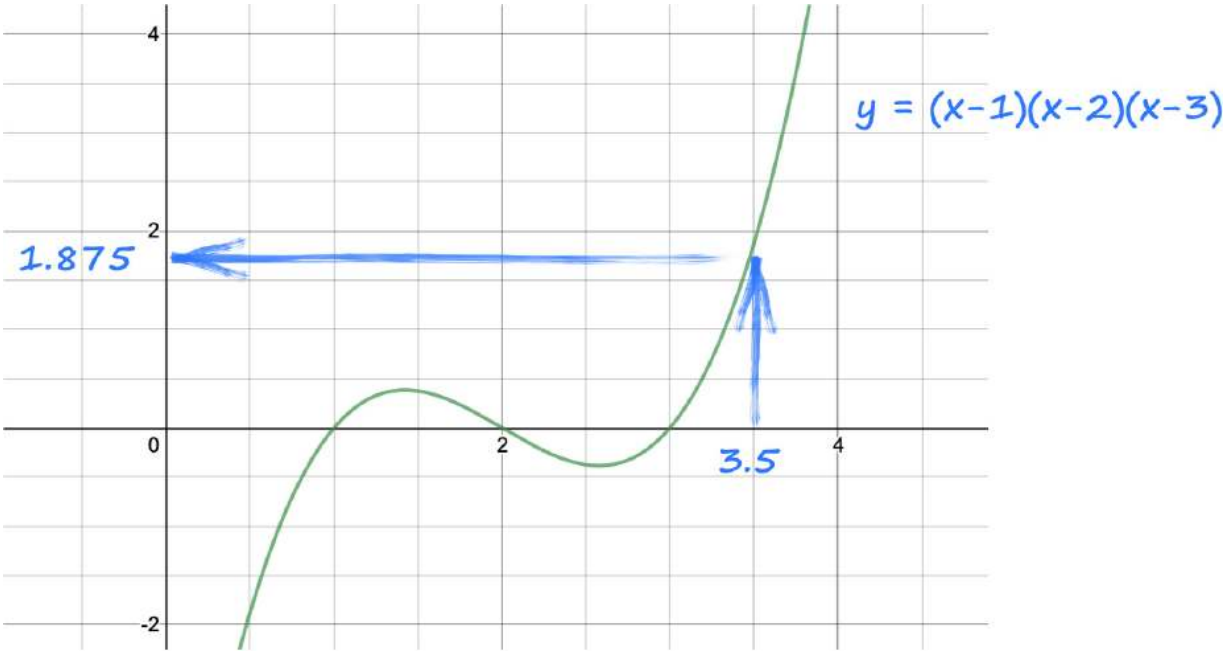


*Abbildung 1-10: Der Graph der Funktion y = (x-1) * (x-2) * (x-3)*

Bislang erscheint alles normal und vertraut.

Tatsächlich hat PyTorch aber zusätzliche Arbeit geleistet, die wir nicht gesehen haben. Denn PyTorch hat nicht einfach den Wert `1.8750` berechnet und in einen Tensor namens `y` gestellt. Vielmehr hat sich PyTorch tatsächlich daran erinnert, dass `y` mathematisch in Form von `x` definiert ist.

Wären `x` und `y` normale Python-Variablen oder sogar NumPy-Arrays, käme Python nicht auf die Idee, dass `y` von `x` kommt. Das ist auch nicht notwendig. Nachdem der Wert von `y` berechnet ist, und zwar aus `x`, ist nur noch dieser Wert an sich wichtig und dass er in `y` steht. Fertig.

PyTorch-Tensoren funktionieren anders. Sie merken sich, aus welchen anderen Tensoren sie berechnet werden und wie. Im Beispiel erinnert sich PyTorch daran, dass `y` von `x` gekommen ist.

Weshalb ist das nützlich? Schauen wir mal.

Sie werden sich erinnern, dass es bei den Berechnungen zum Trainieren eines neuronalen Netzes erforderlich ist, den Fehlergradienten per Analysis zu bestimmen, das heißt die Rate, mit der sich der Ausgabefehler infolge veränderter Gewichte für die Netzverknüpfungen ändert.

Die Ausgabe eines neuronalen Netzes wird von den Verknüpfungsgewichten bestimmt. Diese Ausgabe hängt von den Gewichten genau so ab, wie `y` von `x` abhängt. Sehen wir uns also an, wie PyTorch die Änderungsrate von `y` ermitteln kann, wenn sich `x` verändert.

Wir berechnen den Gradienten von `y` bei `x = 3.5`, das heißt, `dy/dx` bei `x = 3.5`.

Hierfür muss PyTorch für `y` feststellen, von welchen Tensoren es abhängt und wie die mathematische Form dieser Abhängigkeit aussieht. Dann kann es `dy/dx` berechnen.

```
# Gradienten berechnen

y.backward()
```

Diese einzelne Anweisung erledigt das alles. PyTorch betrachtet `y`, sieht, dass es von `(x-1) * (x-2) * (x-3)` kommt, und ermittelt automatisch den Gradienten `dy/dx`, der – wenn Sie es auflösen – den Ausdruck $3x^2 - 12x + 11$ ergibt.

Diese Anweisung berechnet auch den numerischen Wert dieses Gradienten und setzt ihn in den Tensor `x` neben den eigentlichen Wert von `x`. Da `x` gleich `3.5` ist, wird der Gradient zu `3*(3.5*3.5) - 12*(3.5) + 11 = 5.75`.

Der Graph in Abbildung 1-11 veranschaulicht, wo wir diesen Gradienten berechnet haben.