KAPITEL 1

Einstieg in PyTorch

In diesem Kapitel richten wir alle erforderlichen Elemente ein, die wir für die Arbeit mit PyTorch benötigen. Jedes folgende Kapitel wird anschließend auf diesem anfänglichen Grundgerüst aufbauen, daher ist es wichtig, dass wir es richtig machen. Dies führt zu unserer ersten grundlegenden Frage: Sollten Sie besser einen maßgeschneiderten Computer für Deep Learning zusammenstellen oder einfach eine der vielen verfügbaren Cloud-basierten Ressourcen verwenden?

Zusammenbau eines maßgeschneiderten Deep-Learning-Rechners

Wenn man sich ins Deep Learning stürzt, verspürt man den Drang, sich ein Rechenmonstrum zusammenzustellen. Sie können Tage damit verbringen, sich verschiedene Arten von Grafikkarten anzusehen, die verschiedenen Speichertypen in Erfahrung zu bringen, die mit Ihrer CPU kompatibel sind, die beste Art von Speicher zu kaufen und zu durchdenken, wie groß ein SSD-Laufwerk im Rahmen Ihres Budgets sein mag, damit Sie so schnell wie möglich auf Ihre Festplatte zugreifen können. Auch ich war davor nicht gefeit; ich habe vor ein paar Jahren einen Monat damit verbracht, eine Liste mit Komponenten zu erstellen und einen neuen Computer auf meinem Esstisch zusammenzubauen.

Mein Rat – vor allem wenn Sie neu in die Welt des Deep Learning einsteigen – lautet: Tun Sie es nicht. Sie können leicht mehrere Tausend Euro in einen Rechner investieren, den Sie möglicherweise gar nicht so oft benutzen. Stattdessen empfehle ich Ihnen, dieses Buch mithilfe von Cloud-Ressourcen (entweder in Amazon Web Services, Google Cloud oder Microsoft Azure) durchzuarbeiten und erst dann über den Bau eines eigenen Rechners nachzudenken, wenn Sie das Gefühl haben, dass Sie einen eigenen Rechner für den 24/7-Betrieb benötigen. Sie müssen *in keiner Weise* große Investitionen in Hardware tätigen, um den Code in diesem Buch ausführen zu können.

Unter Umständen werden Sie nie in die Lage kommen, einen maßgeschneiderten Rechner für sich selbst bauen zu müssen. Es gibt sicherlich Fälle, in denen es billi-

ger sein kann, einen maßgeschneiderten Rechner zu bauen, insbesondere wenn Sie wissen, dass sich Ihre Berechnungen immer auf einen einzigen Rechner (mit bestenfalls einer Handvoll GPUs) beschränken werden. Wenn Ihre Berechnungen jedoch allmählich mehrere Rechner und GPUs erfordern, erscheint die Cloud-Lösung attraktiver. Angesichts der Kosten für die Zusammenstellung eines maßgeschneiderten Rechners würde ich vor einem möglichen Kauf lange und intensiv nachdenken.

Falls es mir nicht gelungen ist, Sie vom Bau Ihres eigenen Rechners abzubringen, finden Sie in den folgenden Absätzen Vorschläge dazu, was Sie dafür benötigen würden.

Grafikprozessor (GPU)

Das Herzstück jedes Deep-Learning-Rechners ist der Grafikprozessor bzw. die GPU. Sie ist die Grundlage für die Vielzahl an Berechnungen in PyTorch und wird die wahrscheinlich teuerste Komponente in Ihrem Computer sein. In den letzten Jahren sind die Preise für GPUs aufgrund ihres Einsatzes beim Mining von Krypto-Währungen wie dem Bitcoin gestiegen und die Vorräte spürbar gesunken. Glücklicherweise scheint diese Blase allmählich zu verschwinden, und das Angebot an GPUs ist wieder etwas größer geworden.

Zum Zeitpunkt des Verfassens dieses Buchs empfehle ich die Beschaffung der NVIDIA GeForce RTX 2080 Ti. Eine billigere Variante ist die GTX 1080 Ti (wenn Sie jedoch die Entscheidung für die 1080 Ti aus Budgetgründen abwägen, schlage ich erneut vor, stattdessen die Cloud-Optionen in Betracht zu ziehen). Obwohl es auch Grafikkarten von AMD gibt, ist ihre Unterstützung in PyTorch derzeit nicht gut genug, um etwas anderes als eine NVIDIA-Karte zu empfehlen. Aber behalten Sie ihre ROCm-Technologie im Auge, die sie allmählich zu einer ernst zu nehmenden Alternative im GPU-Bereich machen sollte.

Hauptprozessor (CPU) und Motherboard

Wahrscheinlich werden Sie sich für ein Motherboard der Z370-Serie entscheiden wollen. Viele Menschen werden Ihnen sagen, dass die CPU für das Deep Learning keine Rolle spielt und dass Sie mit einer CPU mit einer relativ geringen Rechengeschwindigkeit auskommen können, solange Sie eine leistungsstarke GPU haben. Meiner Erfahrung nach werden Sie überrascht sein, wie oft die CPU zum Engpass werden kann, insbesondere bei der Arbeit mit augmentierten Daten.

Arbeitsspeicher (RAM)

Mehr Arbeitsspeicher ist immer ratsam, da Sie dadurch mehr Daten im Speicher halten können, ohne auf den viel langsameren Festplattenspeicher zurückgreifen

zu müssen (besonders wichtig während des Trainings). Sie sollten mindestens 64 GB DDR4-Speicher für Ihren Rechner in Betracht ziehen.

Speicher

Der Speicher für ein individuelles System sollte zwei Arten umfassen: zum einen eine Festplatte mit M2-Schnittstelle (SSD) – so groß wie möglich – für Ihre »heißen« Daten, damit Sie so schnell wie möglich auf diese zugreifen können, wenn Sie aktiv an einem Projekt arbeiten. Als zweites Speichermedium fügen Sie ein 4 TB (Terabyte) großes Serial-ATA-(SATA-)Laufwerk für Daten hinzu, an denen Sie nicht aktiv arbeiten; wechseln Sie je nach Bedarf zwischen »heißem« und »kaltem« Speicher.

Ich empfehle Ihnen, einen Blick auf die Webseite PCPartPicker (*https://pcpartpicker.com*) zu werfen, um einen Eindruck von den Deep-Learning-Rechnern anderer Leute zu bekommen. (Sie können sich auch all die schrägen und verrückten Ideen ansehen!) Sie erhalten ein Gefühl für die Auswahl an möglichen Computerteilen und den dazugehörigen Preisen, die insbesondere bei Grafikkarten sehr stark schwanken können.

Nachdem Sie nun die Möglichkeiten für Ihren lokalen physischen Rechner gesehen haben, ist es an der Zeit, auf die Cloud-Alternativen zu sprechen zu kommen.

Deep Learning in der Cloud

Okay, nun könnten Sie fragen, warum die Cloud-Variante besser sein sollte – besonders dann, wenn Sie sich das Preisschema von Amazon Web Services (AWS) angeschaut und herausgefunden haben, dass sich der Eigenbau eines leistungsfähigen Rechners innerhalb von sechs Monaten bezahlt macht. Denken Sie darüber nach: Wenn Sie gerade erst anfangen, werden Sie diesen Rechner in den besagten sechs Monaten nicht rund um die Uhr nutzen. Sie werden es schlichtweg nicht tun. Das bedeutet, dass Sie den Cloud-Rechner abschalten und in der Zwischenzeit lediglich ein paar Cents für die gespeicherten Daten bezahlen dürften.

Sie brauchen nicht gleich zu Beginn eine von NVIDIAs Profikarten wie die Tesla V100 zu verwenden, die ebenfalls in Ihrer Cloud-Instanz verfügbar sind. Sie können mit einer der wesentlich preisgünstigeren (manchmal sogar kostenlosen) K80-basierten Instanz beginnen und zu einer leistungsstärkeren Grafikkarte wechseln, sobald diese benötigt wird. Das ist ein wenig günstiger als der Kauf einer einfachen GPU-Karte und die Aufrüstung auf eine RTX 2080 Ti für Ihre eigene Rechnerumgebung. Wenn Sie acht V100-Karten zu einer einzelnen Instanz hinzufügen möchten, können Sie dies mit nur wenigen Klicks erledigen. Versuchen Sie das einmal mit Ihrer eigenen Hardware.

Ein weiterer Punkt ist die Wartung. Wenn Sie es sich zur Gewohnheit machen, Ihre Cloud-Instanzen regelmäßig neu zu erstellen (idealerweise jedes Mal, wenn Sie

wieder an Ihren Projekten arbeiten), werden Sie fast immer auf einem Rechner arbeiten, der auf dem neuesten Stand ist. Gehört der Rechner Ihnen, obliegt Ihnen auch die Aktualisierung. An dieser Stelle muss ich gestehen, dass ich meinen eigenen, speziell konzipierten Deep-Learning-Rechner habe und die Ubuntu-Installation darauf so lange ignoriert habe, bis sie nicht mehr unterstützt wurde. Dies führte dazu, dass ich schließlich einen Tag damit verbrachte, das System erneut an einen Punkt zu bringen, an dem es wieder Updates erhalten konnte. Peinlich.

Wie dem auch sei, nehmen wir an, Sie haben die Entscheidung getroffen, die Cloud zu nutzen. Hurra! Nun zur nächsten Frage: Welchen Anbieter sollten Sie wählen?

Google Colaboratory

Doch warten Sie, bevor wir uns die Anbieter ansehen: Was ist, wenn Sie überhaupt keine Lust haben, einen nennenswerten Aufwand zu treiben? Keine Lust auf das lästige Aufbauen eines Rechners oder die ganze Mühe, Instanzen in der Cloud einzurichten? Was wäre denn tatsächlich die bequemste Wahl? Dafür hat Google das Richtige für Sie. *Colaboratory* (oder *Colab*) (*https://colab.research.google.com*) ist eine weitgehend kostenlose, installationsfreie, benutzerdefinierte Jupyter-Notebook-Umgebung, für die Sie ein Google-Konto benötigen, um Ihre eigenen Notebooks einzurichten. Abbildung 1-1 zeigt einen Screenshot eines in Colab erstellten Notebooks.

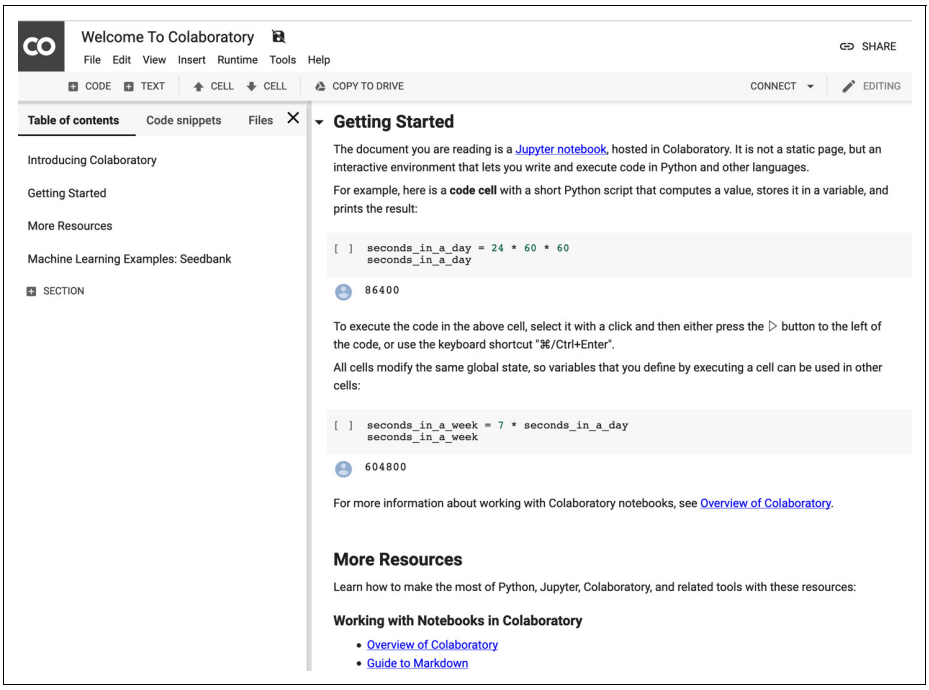

Abbildung 1-1: Google Colab(oratory)

Colab ist ein großartiges Tool, um ins Deep Learning einzutauchen, da es bereits vorinstallierte Versionen von TensorFlow und PyTorch bereithält. Dementsprechend müssen Sie nichts weiter eingeben als den `import torch`-Befehl und auch keine weiteren Einstellungen vornehmen. Jeder Benutzer erhält einen freien Zugang zu einer NVIDIA-T4-Grafikkarte mit bis zu zwölf Stunden ununterbrochener Laufzeit. Kostenlos. Um das einzuordnen: Empirische Untersuchungen haben ergeben, dass Sie etwa die Hälfte der Geschwindigkeit einer 1080 Ti für Ihr Training erhalten, aber zusätzlich noch 5 GB Speicher, sodass Sie größere Modelle speichern können. Es besteht zudem die kostenpflichtige Möglichkeit, eine Verbindung zu neueren GPUs oder Googles eigener TPU-Hardware herzustellen. Aber Sie können so ziemlich jedes Beispiel aus diesem Buch mit Colab ausführen, ohne dass irgendwelche Kosten für Sie anfallen. Aus diesem Grund empfehle ich, Colab zunächst parallel zu diesem Buch zu verwenden, um dann bei Bedarf zu entscheiden, ob Sie auf dedizierte Cloud-Instanzen und/oder Ihren eigenen persönlichen Deep-Learning-Server ausweichen möchten.

Wenn Sie zum ersten Mal mit Google Colab arbeiten, werden Sie feststellen, dass es verschiedene Möglichkeiten gibt, die Dateien des GitHub-Repositorys dieses Buchs einzubinden. Eine gute Einführung bietet Ihnen das Willkommens-Notebook, das sich beim Aufruf von Google Colab im Hintergrund lädt, und hier insbesondere der Abschnitt »Weitere Ressourcen«.

Colab ist der Ansatz, mit dem am wenigsten Aufwand verbunden ist. Sie möchten aber vielleicht ein wenig mehr Kontrolle darüber haben, wie Dinge installiert werden oder wie Sie einen Secure-Shell-(SSH-)Zugang zu Ihrer Instanz in der Cloud erhalten. Werfen wir am besten einen Blick auf das Angebot der wichtigsten Cloud-Anbieter.

Cloud-Anbieter

Jeder der drei großen Cloud-Anbieter (Amazon Web Services, Google Cloud Platform und Microsofts Azure) bietet GPU-basierte Instanzen (auch als *virtuelle Maschinen* oder *VMs* bezeichnet) und offizielle Images, die auf diesen Instanzen bereitgestellt werden. Sie haben alles, was Sie brauchen, um loszulegen, ohne selbst Treiber oder Python-Bibliotheken installieren zu müssen. Gehen wir die Angebote der einzelnen Anbieter einmal durch.

Amazon Web Services

AWS, das Schwergewicht auf dem Cloud-Markt, erfüllt nur zu gern Ihre GPU-Anforderungen und bietet P2- und P3-Instanzarten an, um Sie zu unterstützen. (Der G3-Instanztyp wird eher in tatsächlich grafikbasierten Anwendungen wie der Videocodierung verwendet, daher werden wir hier nicht weiter darauf eingehen). Die P2-Instanzen verwenden die älteren NVIDIA-K80-Karten (maximal 16 können an eine Instanz angeschlossen werden), und die P3-Instanzen setzen die extrem

schnellen NVIDIA-V100-Karten ein (Sie können acht davon auf eine Instanz legen, wenn Sie sich trauen).

Sollten Sie sich für AWS entscheiden, empfehle ich, für dieses Buch die Klasse `p2.xlarge` zu verwenden. Das kostet Sie zum Zeitpunkt des Verfassens dieses Buchs rund einen US-Dollar pro Stunde und bietet Ihnen ausreichend Leistung, um die Beispiele problemlos durchzuarbeiten. Sollten Sie mit diversen anspruchsvollen Kaggle-Wettbewerben beginnen, bietet es sich gegebenenfalls an, auf P3-Instanzen aufzustocken.

Es ist unglaublich einfach, eine Deep-Learning-Umgebung auf AWS zu erstellen:

1. Melden Sie sich an der AWS-Konsole an.
2. Wählen Sie *EC2* und klicken Sie auf *Launch a virtual machine*.
3. Suchen Sie nach der Option *Deep Learning AMI (Ubuntu)* und wählen Sie diese aus.
4. Wählen Sie `p2.xlarge` als Ihren Instanztyp.
5. Starten Sie die Instanz, indem Sie entweder ein neues Schlüsselpaar erstellen oder ein vorhandenes Schlüsselpaar wiederverwenden.
6. Stellen Sie mittels Ihrer Kommandozeile eine Verbindung zur Instanz her, indem Sie SSH verwenden und Port 8888 auf Ihrem lokalen Rechner auf die Instanz umleiten:

```
ssh -L localhost:8888:localhost:8888 \
-i /path/my-key-pair.pem my-instance-user-name@my-instance-public-dns-name
```

7. Starten Sie Jupyter Notebook, indem Sie **`jupyter notebook`** eingeben. Kopieren Sie die erzeugte URL und fügen Sie sie in Ihren Browser ein, um auf Jupyter zuzugreifen.

Vergessen Sie nicht, Ihre Instanz abzuschalten, wenn Sie sie nicht benutzen! Sie können das tun, indem Sie mit der rechten Maustaste auf die Instanz in der Weboberfläche klicken und unter *Instance State* die Option *Stop* wählen. Dadurch wird die Instanz heruntergefahren, und es entstehen Ihnen keine Kosten für die Instanz, solange sie nicht läuft. Allerdings wird Ihnen der Speicherplatz, den Sie der Instanz zugewiesen haben, auch dann in Rechnung gestellt, wenn die Instanz ausgeschaltet ist; seien Sie sich dessen bewusst. Um die Instanz und den Speicherplatz vollständig zu löschen, wählen Sie stattdessen die Option *Terminate*.

Azure

Wie AWS bietet auch Azure eine Mischung aus günstigen K80-basierten Instanzen und teureren Tesla-V100-Instanzen an. Azure stellt außerdem Instanzen zur Verfügung, die auf der älteren P100-Hardware basieren und als Kompromiss zwischen den beiden anderen Instanzen gelten. Für dieses Buch empfehle ich, als Instanztyp eine einzelne K80 (NC6) zu verwenden, die 90 Cent pro Stunde kostet. Wechseln

Sie anschließend je nach Bedarf auf andere Typen wie NC, NCv2 (P100) oder NCv3 (V100).

So richten Sie die VM in Azure ein:

1. Melden Sie sich im Azure-Portal an und suchen Sie nach dem Image der *Data Science Virtual Machine*.
2. Klicken Sie auf die Schaltfläche *Get It Now*.
3. Füllen Sie die Angaben zur VM aus (geben Sie ihr einen Namen, wählen Sie die SSD- statt einer HDD-Festplatte, einen SSH-Benutzernamen sowie ein SSH-Passwort, das der Rechnung der Instanz zugrunde liegende Abonnement und legen Sie den Standort fest, der Ihnen am nächsten liegt und den NC-Instanztyp anbietet).
4. Klicken Sie auf die Option *Create*. Die Instanz dürfte in etwa fünf Minuten bereitgestellt werden.
5. Sie können SSH mit dem Benutzername-Passwort-Paar verwenden, das Sie dem öffentlichen DNS-Namen (*Domain Name System*) dieser Instanz gegeben haben.
6. Jupyter Notebook sollte laufen, sobald die Instanz bereitgestellt wurde; navigieren Sie zu *http://dns_name_of_instance:8000* und verwenden Sie die Benutzername-Passwort-Kombination, die Sie für SSH zur Anmeldung verwendet haben.

Google Cloud Platform

Zusätzlich zu den von Amazon und Azure unterstützten Instanzen K80, P100 und V100 bietet die Google Cloud Platform (GCP) die zuvor erwähnten TPUs für diejenigen an, die immense Daten- und Rechenanforderungen haben. Sie benötigen keine TPUs für dieses Buch, zumal diese sehr kostspielig sind. Sie werden seit PyTorch Version 1.0 unterstützt. Denken Sie also nicht, dass Sie TensorFlow verwenden müssen, um die Vorteile dieser Technologie zu nutzen, wenn Sie ein Projekt haben, das ihre Verwendung erfordert.

Der Einstieg in die Google Cloud ist ebenfalls recht einfach:

1. Suchen Sie nach der Deep-Learning-VM auf dem GCP Marketplace.
2. Klicken Sie auf *Starten*.
3. Geben Sie der Instanz einen Namen und ordnen Sie sie der Region zu, die sich am nächsten bei Ihnen befindet.
4. Wählen Sie als Rechnertyp *8 vCPUs*.
5. Wählen Sie eine GPU des Typs K80 aus.
6. Stellen Sie sicher, dass im Abschnitt *Framework* PyTorch ausgewählt ist.
7. Aktivieren Sie das Kontrollkästchen *Install NVIDIA-GPU automatically on first startup*.

8. Setzen Sie die Bootfestplatte auf *SSD Persistent Disk*.
9. Klicken Sie auf die Option *Bereitstellen*. Die vollständige Bereitstellung der VM dauert etwa fünf Minuten.
10. Um auf der Instanz eine Verbindung zu Jupyter herzustellen, vergewissern Sie sich zunächst, dass Sie im richtigen Projekt in `gcloud` eingeloggt sind, und geben Sie den folgenden Befehl ein:

```
gcloud compute ssh _INSTANCE_NAME_ -- -L 8080:localhost:8080
```

Die Gebühren für Google Cloud sollten sich auf etwa 45 US-Cent pro Stunde belaufen, was das Produkt zum günstigsten der drei großen Cloud-Anbieter macht.

Welchen Cloud-Anbieter sollte ich wählen?

Wenn Sie in keine Richtung tendieren, empfehle ich Ihnen, die Google Cloud Platform (GCP) zu nutzen. Sie ist die preiswerteste Variante und kann bei Bedarf sogar mithilfe von TPUs skaliert werden. Außerdem bietet sie bedeutend mehr Flexibilität als AWS oder Azure. Wenn Sie jedoch bereits Erfahrungen mit einer der beiden anderen Plattformen gemacht haben, können Sie auch in diesen Umgebungen absolut problemlos arbeiten.

Sobald Ihre Cloud-Instanz läuft, können Sie sich bei deren Kopie von Jupyter Notebook anmelden. Schauen wir uns das also als Nächstes an.

Verwendung von Jupyter Notebook

Falls Sie bislang noch nicht auf Jupyter Notebook gestoßen sind, hier die Kurzfassung: Diese browserbasierte Umgebung ermöglicht Ihnen, Livecode mit Text, Bildern und Visualisierungen zu kombinieren, und ist zu einem der wichtigsten Tools von Datenwissenschaftlern auf der ganzen Welt geworden. In Jupyter erstellte Notebooks können leicht gemeinsam genutzt werden; dementsprechend sind auch alle Notebooks in diesem Buch (*https://oreil.ly/pytorch-github*) leicht zugänglich. Abbildung 1-2 zeigt einen Screenshot von Jupyter Notebook in Betrieb.

Wir werden in diesem Buch keine fortgeschrittenen Funktionen von Jupyter einsetzen; alles, was Sie wissen müssen, ist, wie man ein neues Notebook erstellt und dass Sie den Inhalt einer Zelle durch gleichzeitiges Drücken der Umschalt- und der Enter-Taste ausführen (und vielleicht, dass Kommandozeilenbefehle direkt mit einem vorangestellten *!* ausgeführt werden können). Sollten Sie es noch nie benutzt haben, schlage ich vor, die Jupyter-Dokumentation (*https://oreil.ly/-Yhff*) zu lesen, bevor Sie zu Kapitel 2 übergehen.

Bevor wir uns mit der Benutzung von PyTorch beschäftigen, widmen wir uns noch einer letzten Sache: wie man alles manuell installiert.

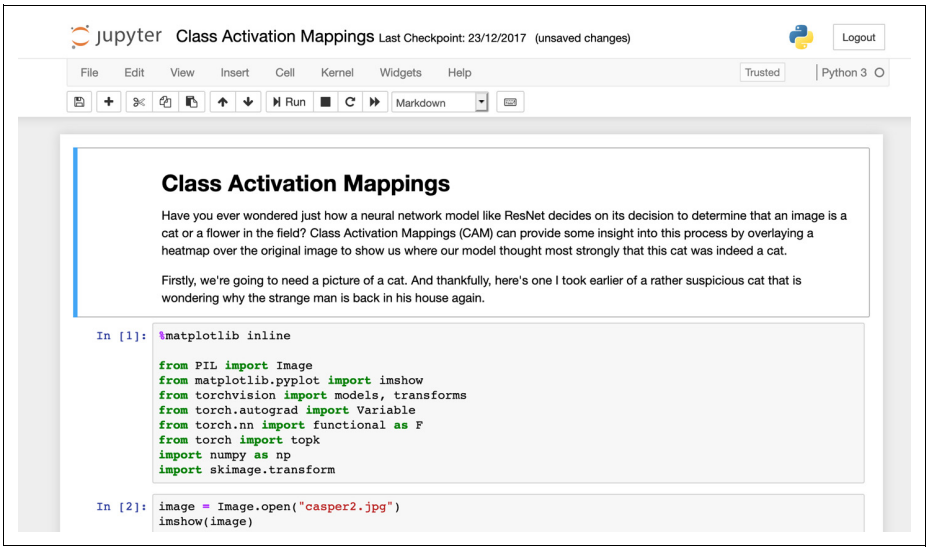

Abbildung 1-2: Jupyter Notebook

PyTorch selbst installieren

Vielleicht möchten Sie etwas mehr Kontrolle über Ihre Programme behalten als mit einem der eben beschriebenen von der Cloud bereitgestellten Images. Oder Sie benötigen für Ihren Code eine bestimmte Version von PyTorch. Oder aber Sie möchten die Programme tatsächlich – all meinen kritischen Anmerkungen zum Trotz – auf Ihrem System zu Hause ausführen. Schauen wir uns an, wie man PyTorch im Allgemeinen auf einem Linux-Server installiert.

Sie können PyTorch mit Python 2.*x* verwenden, aber ich empfehle dringend, das nicht zu tun. Während die Upgrade-Saga von Python 2.*x* auf 3.*x* nun schon seit über einem Jahrzehnt andauert, beginnen immer mehr Pakete, die Unterstützung für Python 2.*x* einzustellen. Wenn Sie also keinen besonderen Grund haben, stellen Sie sicher, dass Ihr System mit Python 3 läuft.

CUDA downloaden

Obwohl PyTorch durchaus auf einer CPU ausgeführt werden kann, ist für den praktischen Gebrauch in den meisten Fällen der Einsatz einer GPU erforderlich. Dementsprechend benötigt PyTorch eine GPU-Unterstützung. Dies lässt sich ziemlich einfach umsetzen – vorausgesetzt, Sie haben eine NVIDIA-Karte. Die GPU-Unterstützung wird von deren CUDA-Schnittstelle (*Compute Unified Device Architecture*) zur Verfügung gestellt. Laden Sie für Ihre Linux-Variante das passende Paketformat herunter (*https://oreil.ly/Gx_q2*) und installieren Sie das Paket.

Für Red Hat Enterprise Linux (RHEL) 7:

```
sudo rpm -i cuda-repo-rhel7-10-0local-10.0.130-410.48-1.0-1.x86_64.rpm
sudo yum clean all
sudo yum install cuda
```

Für Ubuntu 18.04:

```
sudo dpkg -i cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb
sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub
sudo apt-get update
sudo apt-get install cuda
```

Anaconda

Python beinhaltet eine Vielzahl von Paketverwaltungssystemen, die allesamt gute und weniger gute Eigenschaften besitzen. Wie die Entwickler von PyTorch empfehle ich Ihnen, Anaconda zu installieren. Anaconda ist ein Paketierungssystem, das darauf ausgerichtet ist, die optimale Zusammenstellung von Paketen für Datenwissenschaftler zu gewährleisten. Wie CUDA ist es ziemlich einfach zu installieren.

Gehen Sie auf die Webseite von Anaconda (*https://oreil.ly/9hAxg*) und suchen Sie die Installationsdatei für Ihren Rechner heraus. Da es sich um ein riesiges Archiv handelt, das über ein Shell-Skript auf Ihrem System ausgeführt wird, empfehle ich Ihnen, die heruntergeladene Datei *md5sum* auszuführen und sie mit der Liste der Signaturen (*https://oreil.ly/anuhu*) zu vergleichen. Prüfen Sie, ob die Signatur auf Ihrem Rechner mit der auf der Webseite übereinstimmt, bevor Sie den Kommandozeilenbefehl `bash Anaconda3-VERSION-Linux-x86_64.sh` ausführen. Dies stellt sicher, dass die heruntergeladene Datei nicht manipuliert wurde und auf Ihrem System sicher ausgeführt werden kann. Das Skript fordert Sie mehrmals auf, das Installationsverzeichnis anzugeben; wenn es keine Gründe dagegen gibt, akzeptieren Sie einfach die Standardeinstellungen.

Sie fragen sich vielleicht: »Kann ich das auch auf meinem MacBook bewerkstelligen?« Leider sind die meisten Macs heutzutage entweder mit Intel- oder AMD-Grafikprozessoren ausgestattet und bieten nicht wirklich eine Unterstützung für die Ausführung von PyTorch im GPU-beschleunigten Modus. Ich empfehle daher die Verwendung von Colab oder eines Cloud-Providers, anstatt zu versuchen, Ihren Mac lokal zu nutzen.

Zu guter Letzt – PyTorch (und Jupyter Notebook)

Jetzt, da Sie Anaconda installiert haben, ist die Einrichtung von PyTorch denkbar einfach:

```
conda install pytorch torchvision -c pytorch
```

Hierdurch werden PyTorch und das `torchvision`-Paket installiert, die wir in einigen nachfolgenden Kapiteln verwenden, um Deep-Learning-Architekturen zu erstel-

len, die mit Bildern arbeiten. Mit der Anaconda-Distribution wird automatisch auch Jupyter Notebook für uns installiert. Wir können also sofort loslegen:

```
jupyter notebook
```

Geben Sie *http://YOUR-IP-ADDRESS:8888* in Ihren Browser ein (eigentlich sollten Sie direkt darauf umgeleitet werden), erstellen Sie ein neues Notebook und geben Sie Folgendes ein:

```
import torch
print(torch.cuda.is_available())
print(torch.rand(2,2))
```

Dies sollte zu einer ähnlichen Ausgabe wie der folgenden führen:

```
True
 0.6040  0.6647
 0.9286  0.4210
[torch.FloatTensor of size 2x2]
```

Falls der Befehl `cuda.is_available()` als Ausgabe `False` zurückgibt, müssen Sie Ihre CUDA-Installation debuggen, damit PyTorch Ihre Grafikkarte erkennen kann. Die Werte des Tensors werden auf Ihrer Instanz etwas anders sein.

Aber was ist eigentlich dieser Tensor? Tensoren sind das Herzstück von fast allem in PyTorch. Deshalb sollten Sie wissen, was ein Tensor ist und was er für Sie leisten kann.

Tensoren

Ein *Tensor* ist sowohl ein Container für Zahlen als auch ein Regelsatz, der Transformationen zwischen Tensoren definiert, die neue Tensoren erzeugen. Es ist wahrscheinlich am einfachsten für uns, *Tensoren* als mehrdimensionale Arrays aufzufassen. Jeder Tensor hat einen *Rang*, der seinem Dimensionsraum entspricht. Ein einfacher Skalar (z.B. 1) kann als Tensor mit dem Rang 0 dargestellt werden, ein Vektor besitzt den Rang 1, eine $n \times n$-Matrix den Rang 2 und so weiter. Im vorigen Beispiel haben wir einen Tensor mit Zufallswerten vom Rang 2 durch die Verwendung von `torch.rand()` erstellt. Wir können sie auch aus Listen erzeugen:

```
x = torch.Tensor([[0,0,1],[1,1,1],[0,0,0]])
x
>tensor([[0, 0, 1],
    [1, 1, 1],
    [0, 0, 0]])
```

Wir können ein Element in einem Tensor ändern, indem wir die gewöhnliche Python-Indizierung verwenden:

```
x[0][0] = 5
>tensor([[5, 0, 1],
    [1, 1, 1],
    [0, 0, 0]])
```

Sie können spezielle vordefinierte Funktionen einsetzen, um bestimmte Arten von Tensoren zu erzeugen. Insbesondere die Funktionen `ones()` und `zeroes()` erzeugen Tensoren, die mit Einsen bzw. mit Nullen gefüllt sind:

```
torch.zeros(2,2)
> tensor([[0., 0.],
    [0., 0.]])
```

Auch mathematische Standardoperationen können mit Tensoren durchgeführt werden (z. B. können zwei Tensoren addiert werden):

```
torch.ones(1,2) + torch.ones(1,2)
> tensor([[2., 2.]])
```

Und wenn Sie einen Tensor vom Rang 0 haben, können Sie sich den Wert des Elements mit der Funktion `item()` ausgeben lassen:

```
torch.rand(1).item()
> 0.34106671810150146
```

Tensoren können sowohl auf der CPU als auch auf der GPU eingesetzt und mit der Funktion `to()` zwischen den Komponenten kopiert werden:

```
cpu_tensor = tensor.rand(2)
cpu_tensor.device
> device(type='cpu')

gpu_tensor = cpu_tensor.to("cuda")
gpu_tensor.device
> device(type='cuda', index=0)
```

Tensoroperationen

Wenn Sie sich die PyTorch-Dokumentation (*https://oreil.ly/1Ev0-*) anschauen, werden Sie feststellen, dass es eine *Vielzahl* an Funktionen gibt, die Sie auf Tensoren anwenden können – alles vom Finden des Maximalwerts bis zur Anwendung einer Fourier-Transformation. Im vorliegenden Buch müssen Sie nicht alle Funktionen kennen, um die Bilder, Texte und Töne in Tensoren zu verwandeln und sie für unsere Zwecke zu manipulieren. Sie werden dennoch einige benötigen. Ich empfehle Ihnen auf jeden Fall, spätestens aber nach dem Lesen dieses Buchs, einen Blick in die Dokumentation zu werfen. Gehen wir nun alle Funktionen durch, die in den kommenden Kapiteln verwendet werden.

Zum einen müssen wir häufig das Element mit dem Maximalwert in einem Tensor sowie den korrespondierenden *Index* des Elements finden (da dieses oft der Kategorie angehört, für die sich das neuronale Netz in seiner endgültigen Vorhersage entschieden hat). Dies können wir mit den Funktionen `max()` und `argmax()` vornehmen. Ebenso können wir die Funktion `item()` verwenden, um einen Wert aus einem eindimensionalen Tensor zu extrahieren.

```
torch.rand(2,2).max()
> tensor(0.4726)
```

```
torch.rand(2,2).max().item()
> 0.8649941086769104
```

In manchen Situationen möchten wir den Typ eines Tensors ändern, zum Beispiel von einem `LongTensor` in einen `FloatTensor`. Dies können wir mit der Funktion `to()` erreichen:

```
long_tensor = torch.tensor([[0,0,1],[1,1,1],[0,0,0]])
long_tensor.type()
> 'torch.LongTensor'
float_tensor = torch.tensor([[0,0,1],[1,1,1],[0,0,0]]).to(dtype=torch.float32)
float_tensor.type()
> 'torch.FloatTensor'
```

Die meisten Funktionen, die auf einem Tensor operieren und einen Tensor zurückgeben, erzeugen gleichzeitig einen neuen Tensor, um das Ergebnis zu speichern. Wenn Sie jedoch Arbeitsspeicher sparen wollen, vergewissern Sie sich, ob eine *In-Place*-Funktion definiert ist. Diese besitzt den gleichen Namen wie die ursprüngliche Funktion, lediglich mit einem angehängten Unterstrich (_).

```
random_tensor = torch.rand(2,2)
random_tensor.log2()
>tensor([[-1.9001, -1.5013],
        [-1.8836, -0.5320]])
random_tensor.log2_()
> tensor([[-1.9001, -1.5013],
        [-1.8836, -0.5320]])
```

Eine weitere gängige Operation ist die *Umformung* eines Tensors. Dies kann oft vonnöten sein, weil Ihre neuronale Netzwerkschicht eine etwas andere Form der Eingabe erfordert als die, die Sie gerade einspeisen. Zum Beispiel ist der Datensatz des *Modified National Institute of Standards and Technology* (MNIST) mit handgeschriebenen Ziffern eine Sammlung von 28 × 28 Pixel großen Bildern. Er besteht jedoch aus Arrays der Länge 784. Um die Netzwerke, die wir aufbauen, zu verwenden, müssen wir diese wieder in 1 × 28 × 28-Tensoren umwandeln (die führende 1 entspricht der Anzahl der Farbkanäle – für gewöhnlich Rot, Grün und Blau –, da die MNIST-Ziffern aber nur in Graustufen gehalten sind, haben wir lediglich einen Kanal). Die Umformung können wir entweder mit der Funktion `view()` oder mit `reshape()` durchführen:

```
flat_tensor = torch.rand(784)
viewed_tensor = flat_tensor.view(1,28,28)
viewed_tensor.shape
> torch.Size([1, 28, 28])
reshaped_tensor = flat_tensor.reshape(1,28,28)
reshaped_tensor.shape
> torch.Size([1, 28, 28])
```

Dabei ist es wichtig, dass Sie den Tensor in eine Form überführen, bei der er die gleiche Anzahl an Gesamtelementen wie sein Original umfasst. Wenn Sie es mit `flat_tensor.reshape(3,28,28)` versuchen, werden Sie einen Fehler wie diesen sehen:

```
RuntimeError Traceback (most recent call last)
<ipython-input-26-774c70ba5c08> in <module>()
----> 1 flat_tensor.reshape(3,28,28)

RuntimeError: shape '[3, 28, 28]' is invalid for input of size 784
```

Sie fragen sich jetzt vielleicht, worin der Unterschied zwischen `view()` und `reshape()` besteht. Die Antwort ist, dass `view()` als ein Abbild des ursprünglichen Tensors funktioniert – wenn also die zugrunde liegenden Daten geändert werden, wird sich auch das Abbild ändern (und umgekehrt). Allerdings kann `view()` Fehler verursachen, wenn das erforderliche Abbild nicht *kontinuierlich* ist; d. h., wenn der Tensor nicht denselben Speicherblock teilt, den er belegen würde, wenn ein neuer Tensor mit der erforderlichen Form von Grund auf erstellt werden würde. Wenn dies geschieht, müssen Sie `tensor.contiguous()` aufrufen, bevor Sie `view()` verwenden können. Da `reshape()` jedoch all das hinter den Kulissen erledigt, empfehle ich im Allgemeinen die Verwendung von `reshape()` anstelle von `view()`.

Schließlich müssen Sie möglicherweise die Dimensionen eines Tensors neu anordnen. Dies wird Ihnen wahrscheinlich bei Bildern begegnen, da diese oft in Form von `[Höhe, Breite, Kanal]`-Tensoren gespeichert sind. PyTorch sieht jedoch die Anordnung `[Kanal, Höhe, Breite]` vor, um diese zu verarbeiten. Sie können die Funktion `permute()` nutzen, um diesem Problem auf eine ziemlich einfache Art und Weise zu begegnen:

```
hwc_tensor = torch.rand(640, 480, 3)
chw_tensor = hwc_tensor.permute(2,0,1)
chw_tensor.shape
> torch.Size([3, 640, 480])
```

Hier haben wir gerade die Funktion `permute` auf einen `[640,480,3]`-Tensor angewendet, wobei die Argumente den Indizes der Dimensionen des Tensors entsprechen. Dadurch haben wir erreicht, dass die letzte Dimension (2, da der Index bei 0 beginnt) an der ersten Stelle unseres Tensors steht, gefolgt von den restlichen zwei Dimensionen in ihrer ursprünglichen Reihenfolge.

Tensor-Broadcasting

An NumPy angelehnt, ermöglicht *Broadcasting* die Durchführung von Operationen zwischen einem Tensor und einem kleineren Tensor. Rückwärts ausgehend von den letzten Dimensionen, können Sie über zwei Tensoren broadcasten, wenn:

- die beiden Dimensionen identisch sind,
- eine der Dimensionen 1 ist.

Hier ein Beispiel:

```
a = torch.ones(1,2,2)

> tensor([[[1., 1.],
          [1., 1.]]])
```

```
b = torch.ones(1,2)

> tensor([[1., 1.]])

c = a + b

> tensor([[[2., 2.],
          [2., 2.]]])

c.shape
> torch.Size([1, 2, 2])
```

Wenn wir Tensor `b` zu Tensor `a` addieren, erhalten wir als Ergebnis Tensor `c`. Dabei wurde der kleinere Tensor so erweitert, dass er die Dimensionen des größeren Tensors abdeckt. Würden wir versuchen, einen `[2,2]`-Tensor zu einem `[3,3]`-Tensor zu addieren, erhielten wir diese Fehlermeldung:

```
The size of tensor a (2) must match the size of
tensor b (3) at non-singleton dimension 1
```

Wir könnten jedoch problemlos einen `[1,3]`-Tensor zu einem `[3,3]`-Tensor addieren. Das Broadcasting ist ein praktisches kleines Feature, das den Code kürzer macht und oft schneller ist, als den Tensor selbst manuell zu erweitern.

Das ist alles, was Sie zunächst in puncto Tensoren benötigen, um anzufangen! Wir werden später im Verlauf des Buchs noch weitere Operationen aufgreifen. Widmen wir uns nun Kapitel 2.

Zusammenfassung

Ob in der Cloud oder auf Ihrem lokalen Rechner, Sie sollten jetzt PyTorch installiert haben. Ich habe den grundlegenden Baustein der Bibliothek, den *Tensor*, vorgestellt, und Sie haben einen kurzen Blick auf Jupyter Notebook geworfen. Das ist alles, was Sie zum Einstieg benötigen! Im nächsten Kapitel verwenden Sie alles, was Sie bisher gesehen haben, um mit dem Aufbau neuronaler Netze und der Klassifizierung von Bildern zu beginnen. Vergewissern Sie sich also, dass Sie mit Tensoren und Jupyter vertraut sind, bevor Sie weitermachen.

Weiterführende Literatur

- Dokumentation zum Jupyter-Projekt (*https://jupyter.org/documentation*)
- PyTorch-Dokumentation (*https://pytorch.org/docs/stable*)
- AWS Deep Learning AMIs (*https://oreil.ly/G9Ldx*)
- Azure Data Science Virtual Machines (*https://oreil.ly/YjzVB*)
- Google Deep Learning VM Image (*https://oreil.ly/NFpeG*)
- PyTorch-Ökosystem (*https://pytorch.org/ecosystem/*)
- PyTorch-Ressourcen (*https://pytorch.org/resources/*)