Deep Learning With Javascript

A Hacker's Guide to Getting Started with Neural Networks

Kevin Scott

Contents

Introduction i

1. What is Deep Learning 1

I Inference 12

2. Making Predictions 13

3. Data and Tensors 20

4. Predicting an Image 30

II Training 36

5. Training Your Neural Network 37

6. Training a Neural Network from Scratch 42

7. The Importance of Non-Linear Data 50

8. Structured Data 66

9. Recognizing Images 80

10. Transfer Learning with ImageNet 93

Resources 108

Appreciation 110

About the Author 111

Introduction

```
function login(username, password) {
  const user = User.get(username)

  if (!user) {
    throw new Error('Bad login attempt')
  } else if (!user.checkPassword(password)) {
    throw new Error('Bad login attempt')
  } else if (user.expired) {
    throw new Error('Your subscription has expired')
  }

  return user
}
```

You've probably seen code like this before.

This is how most software gets written today. Manually and line by line. You, the programmer, specify everything about how a program should operate, from how a user interacts to the way data is stored and retrieved.

Some have called this approach to code "explicit programming"[1], though a more straight forward description would simply be "programming".

When you're building a login form, or any sort of system where you want **this** action to cause **that** outcome, explicitly programming software makes a lot of sense. You can reason through code like this. There's little danger that the software wakes up one day on the wrong side of the bed and decides it's tired of this whole "login business" and why don't we try logging users *out* for a change? Your code will always do exactly what you've told it to.

But let's say you want software that can detect whether a photograph contains a human face or not. How might you approach that?

[1] https://www.forbes.com/sites/valleyvoices/2018/08/24/software-developer-revolution/#384796881e4a

```
function isThisAHumanFace(pixels) {
  if (
    hasEyes(pixels) &&
    hasEars(pixels) &&
    hasNose(pixels) &&
    hasMouth(pixels)
  ) {
    return true
  }

  return false
}
```

Perhaps you start by reasoning that a human head is made up of eyes, ears, a nose, and a mouth. What if the photograph is from above, and all we see is the top of the person's head? What about from below? What if the person is hanging upside down? What if half their face is hidden in shadow? What if they're wearing a floppy hat, or has long hair covering their eyes, or has a beard that covers their mouth? Trying to define every possibility in a scenario like this becomes very complicated, very quickly.

Deep Learning offers a fundamentally different approach to building software. Instead of explicitly enumerating every possible feature the software may encounter, you provide the computer with examples, and let *the computer* figure out how to get to the right answer.

That's the promise of Deep Learning. Tackle problems that traditionally were hard-to-impossible to solve with regular programming. And in the process, revolutionize how we build software.

* * *

I wanted a book like this one when I began studying Deep Learning. I'm a hacker at heart, and I learn by doing.

Most books on Deep Learning assume strong mathematics or statistics backgrounds. Traditionally this wasn't an issue, because anyone looking to enter the field would be expected to tackle problems that required these backgrounds.

Today, facilitated by better hardware, better frameworks, and new discoveries in research, that's starting to change. We're seeing a whole new range of use cases emerge that are relevant to programmers, designers, writers, artists, and poets, and over the next couple years I believe Neural Networks will see wide-scale deployment across all software domains.

The goal of this book is to demystify Neural Networks and prepare you for that future, through hacking and learning how to use them in real world projects.

To get the most out of this book, I assume you're familiar with modern Javascript, or at least confident enough to fake it. If you do need a primer on Javascript, there's a number of great recommendations in the Resources section.

How This Book is Organized

At the beginning of each chapter is a URL that provides a sandbox for writing your code, with everything set up and ready to go. You are of course welcome to run the examples locally if you'd prefer. If you do, make sure to install the following libraries via `npm`:

- `dljsbook` - This is the main set of suporting libraries for this book, including `datasets` and `UI`.
- `@tensorflow/tfjs` - This is the main Deep Learning framework we'll be using throughout the book.
- `@tensorflow/tfjs-vis` - (Optional) This is a visualization helper for displaying charts, images, and tables in the browser. We'll be using this to log out various items during our coding.

The `Tensorflow.js` documentation is top-notch and I'll avoid regurgitating it unless it's particularly relevant to the topic at hand.

The field of Deep Learning changes incredibly fast. Things evolve, and quickly. If anything is missing, confusing, or just plain wrong, please let me know (feedback@dljsbook.com) so I can fix it for the next version.

Happy hacking!

1. What is Deep Learning

When I was growing up, I read a story about a man from Serbia named Kalfa Manojlo[1]. He was a blacksmith's apprentice who dreamt of becoming the first man in human history to fly.

He wasn't the first to try. People had been flapping their arms like birds for centuries with no apparent success, but Manojlo, with the confidence of somebody too young to know what they don't know, believed he could build a pair of wings better than any that had come before. On a chilly November day in 1841 Manojlo took his winged contraption, scaled the town Customs Office, and launched himself into the air above a crowd of bemused onlookers.

You've probably never heard of this guy unless you're Serbian, so you can probably guess what happened: Manojlo landed head first in a nearby snowbank to much amusement from the gathered crowd. (Pretty good entertainment in the 1840s.)

Many early attempts at flight were like this. People thought that to fly like a bird, you had to imitate a bird. It's not an unreasonable assumption; birds have been flying pretty darn well for a pretty long time. But to conquer flight on a human scale requires a fundamentally different approach.

Neural Networks, the engines that power Deep Learning, are inspired by the human brain, in particular its capacity to learn over time. Similar to biological brains, they are composed of cells connected together that change in response to exposure from stimulus. However, Neural Networks are really closer to statistics on steroids than they are to a human brain, though they share some similarities. The strategies we'll use to build and train them are fundamentally different than anything you'd see in the animal kingdom.

Though the concepts behind Neural Networks have been around since the '50s, it's only within the past decade that it's exploded in industry, largely because of advances in hardware, volume of data, and cutting-edge research. Even though we're still in the early days of this technology moving into the broader world of software, the current generation of tools have racked up some very impressive accomplishments. Let's take a look at why Deep Learning is, and promises to be, such a revolutionary technology.

[1] http://elevate.airserbia.com/Elevate_Specijal_jul2017/index.html#28/z

> Throughout this book we use the phrase **Deep Learning** instead of Artificial Intelligence. AI is an exceptionally broad term, and depending on the speaker, can mean anything from Logistic Regression to Skynet taking over the world. Because of its ambiguity, we'll instead use the terms **Machine Learning** and **Deep Learning** to describe the process for how Neural Networks learn and function. **Machine Learning** describes the general practice of teaching computers to learn things and make predictions, and **Deep Learning** is a subset of Machine Learning. The *Deep* part refers to the architecture of the network, specifically that it has many layers, or it is "deep". Machine Learning, when contrasted with Deep Learning, can also be implied shorthand for traditional statistical methods, in contrast to more recent Deep Neural Networks.

The Proliferation of Deep Learning

You've probably encountered Neural Networks in use through one of the popular virtual assistants, like Siri, Alexa, or Google Home, on the market today. When you use your face or fingerprint to unlock your phone, that's a Neural Network. There's a Neural Network running on your phone's keyboard, predicting which words you're likely to type next, or autosuggesting likely phrases in your email application. Perhaps you've been prompted to tag your friends in uploaded photos, with your friends' faces highlighted and their names autosuggested.

Deep Learning can recognize and classify images to a degree of accuracy that exceeds humans[2]. Deep Learning monitors your inbox for spam and monitors your purchases for fraud. An autonomous agent uses Deep Learning to decide which move to make in a game of Go. An autonomous car uses Deep Learning to decide whether to speed up, slow down, and turn right or left. Hedge funds use Deep Learning to predict what stocks to buy, and stores use it to forecast demand. Magazines and newspapers use Deep Learning to automatically generate summaries of sports, and doctors are using Deep Learning to identify cancerous cells, perform surgery, and sequence genomes. Researchers recently trained a Neural Network to diagnose early-stage Alzheimer's disease long before doctors could[3].

There's almost no industry that won't realize huge changes from Deep Learning, and these changes are coming not in decades, but *today* and over the next few years.

[2] https://arxiv.org/pdf/1706.06969.pdf
[3] https://www.ucsf.edu/news/2018/12/412946/artificial-intelligence-can-detect-alzheimers-disease-brain-scans-six-years

Andrew Ng, co-founder of Google Brain, often refers to this technology as "the new electricity"[4], a technology that will become so ubiquitous as to be embedded in every device, everywhere around us. This oncoming sea change has huge implications for how we approach the craft of software, and how we build applications. Andrej Karpathy, Director of AI at Tesla, refers to this as "Software 2.0":

> It turns out that a large portion of real-world problems have the property that it is significantly easier to collect the data (or more generally, identify a desirable behavior) than to explicitly write the program. In these cases, the programmers will split into two teams. The 2.0 programmers manually curate, maintain, massage, clean and label datasets; each labeled example literally programs the final system because the dataset gets compiled into Software 2.0 code via the optimization. Meanwhile, the 1.0 programmers maintain the surrounding tools, analytics, visualizations, labeling interfaces, infrastructure, and the training code. — Andrej Karpathy[5]

Intelligent Devices

In the recent past, Neural Networks have demanded such enormous computational power that it's only ever made sense to run them on servers. That's beginning to change.

Companies are investing huge sums in improving consumer-grade hardware to run Neural Networks. Apple's recently released NPU chip - a dedicated neural processing unit embedded in every new iPhone - saw an astounding *9x* increase over the previous year's model[6]. As Moore's Law slows down for traditional CPUs, manufacturers are finding that they can eke out huge gains focusing on more specialized domains, like Neural Networks. This means that, more and more, consumer-level hardware will feature powerful hardware capable of handling powerful Neural Networks on board.

And just in time, because there's compelling reasons to run Neural Networks directly on the device.

A big one is privacy. With an Neural Network on-device, data never needs to go to the cloud. All the processing happens locally. This is compelling for consumers concerned about who is using their data and how. There's also plenty of use cases that demand privacy, especially as smart devices expand into more intimate spaces like our homes.

[4] https://www.theguardian.com/future-focused-it/2018/nov/12/is-ai-the-new-electricity
[5] https://medium.com/@karpathy/software-2-0-a64152b37c35
[6] https://www.apple.com/in/iphone-xs/a12-bionic/

Another reason is latency. If you're in a self-driving car, you can't be reliant on a cloud connection to detect whether pedestrians are in front of you. Even with a decent connection, it's hard to do real time analysis on a 60 FPS video or audio stream if you're processing it on the server; that goes double for cutting-edge AR and VR applications. Doing the processing directly on the device lets us avoid the round trip.

Consumer devices have direct access to a wide array of sensors - proximity sensors, motion sensors, ambient light sensors, moisure sensors - that Neural Networks can hook into, and the devices in turn provide a rich surface for enabling direct interactivity with Neural Networks in ways that servers can't match.

As more companies face the decision of whether to deploy their Neural Network on a server or directly on the device, I believe increasingly the answer will be the device. It just makes sense.

> I believe we're currently at an inflection point, and mobile devices are soon going to dominate inference. The growth of the entire AI ecosystem is going to be fueled by mobile inference capabilities. Thanks to the scale of the mobile ecosystem, the world will become filled with amazing experiences enabled by mainstream AI technology … Tasks that were once only practical in the cloud are now being pushed to the edge as mobile devices become more powerful. — Dan Abdinoor, CTO of Fritz.ai

Javascript and Neural Networks

While there's no shortage of domain-specific languages for on-device Neural Networks, I think Javascript is well positioned to garner market share for *on-device* Neural Networks going forward.

Javascript boasts a chameleon-like ability to exist on every platform, everywhere. It starts out with an enormous installed base of every browser on every phone and desktop, but it's also a popular environment for building native mobile application development (with React Native), native desktop development (Electron), or server-side development (Node.js).

> Any application that can be written in JavaScript, will eventually be written in JavaScript. —Jeff Atwood, Cofounder of StackOverflow

In most AI frameworks (for instance, Tensorflow) the framework serves as a translation layer between high level abstractions and the actual mathematical operations being farmed out to the GPU. Since the underlying mathematical layer is C++, this gives software developers freedom to build the abstractions in whatever language they are most comfortable.

Javascript is ideal for our purposes in this book, because you already have it installed (through your web browser) and as a language it excels in handling rich interactive experiences. Though all the Networks we'll write in this book are designed to run in a browser, you may wish to tackle larger datasets requiring more computation in the future; if so, all examples can be ported to run in Node.js and can take advantage of whatever serverside GPUs you have available.

The biggest drawback I see for using Javascript for Deep Learning is the nascent ecosystem. `npm` still lags Python in the breadth and depth of AI-specific packages, and a huge amount of resources, tutorials, and books demonstrate AI concepts in Python or R. To me, this presents an opportunity as a community to step up and contribute the next generation of tools. Javascript is a wily language and I fully expect developers to fill the gaps in short order.

Neural Networks

We've talked about why Neural Networks are important, but what exactly *is* a Neural Network?

A Neural Network is basically layers of mathematical functions that transform numeric input and return some other numeric output.

The fundamental building block of a Neural Network is called a **Neuron**.

A lone Neuron

The Neuron is responsible for transforming the data that moves through it. The nature of this transformation is specified by you, the programmer, based upon the problem you're tackling and the type of Network you want to build.

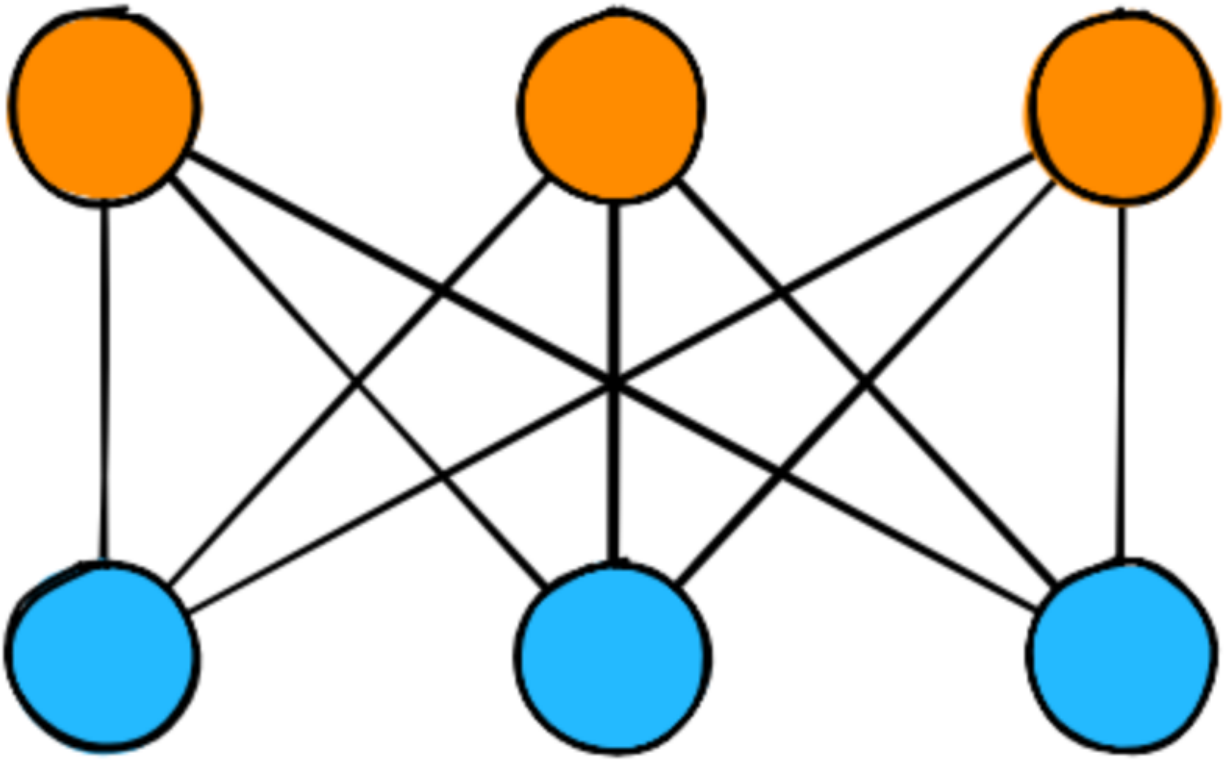
In code, Neurons are commonly referred to as **units**.

A collection of Neurons comprise a **Layer**.

A Layer of Neurons

Since a layer is a collection of neurons acting in concert, it is capable of much richer computation than a single Neuron.

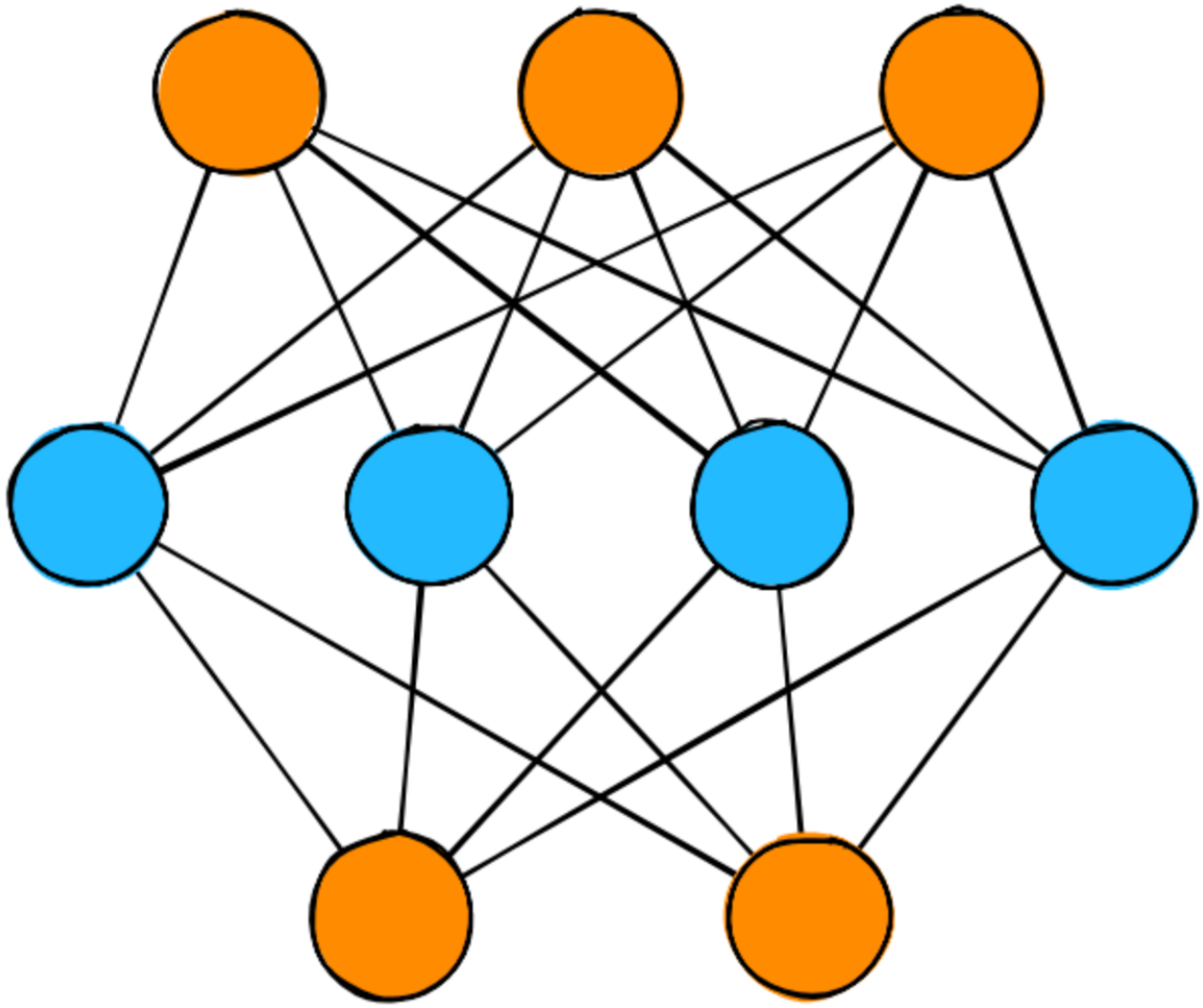
Neurons are connected to other Neurons via **Weights**.



Neurons connected by Weights

Weights describe the strength of the connection between a given pair of Neurons. That is, a stronger weight implies a stronger connection between one Neuron and another. A higher weight will lead to a higher likelihood that a particular Neuron will influence the final prediction.

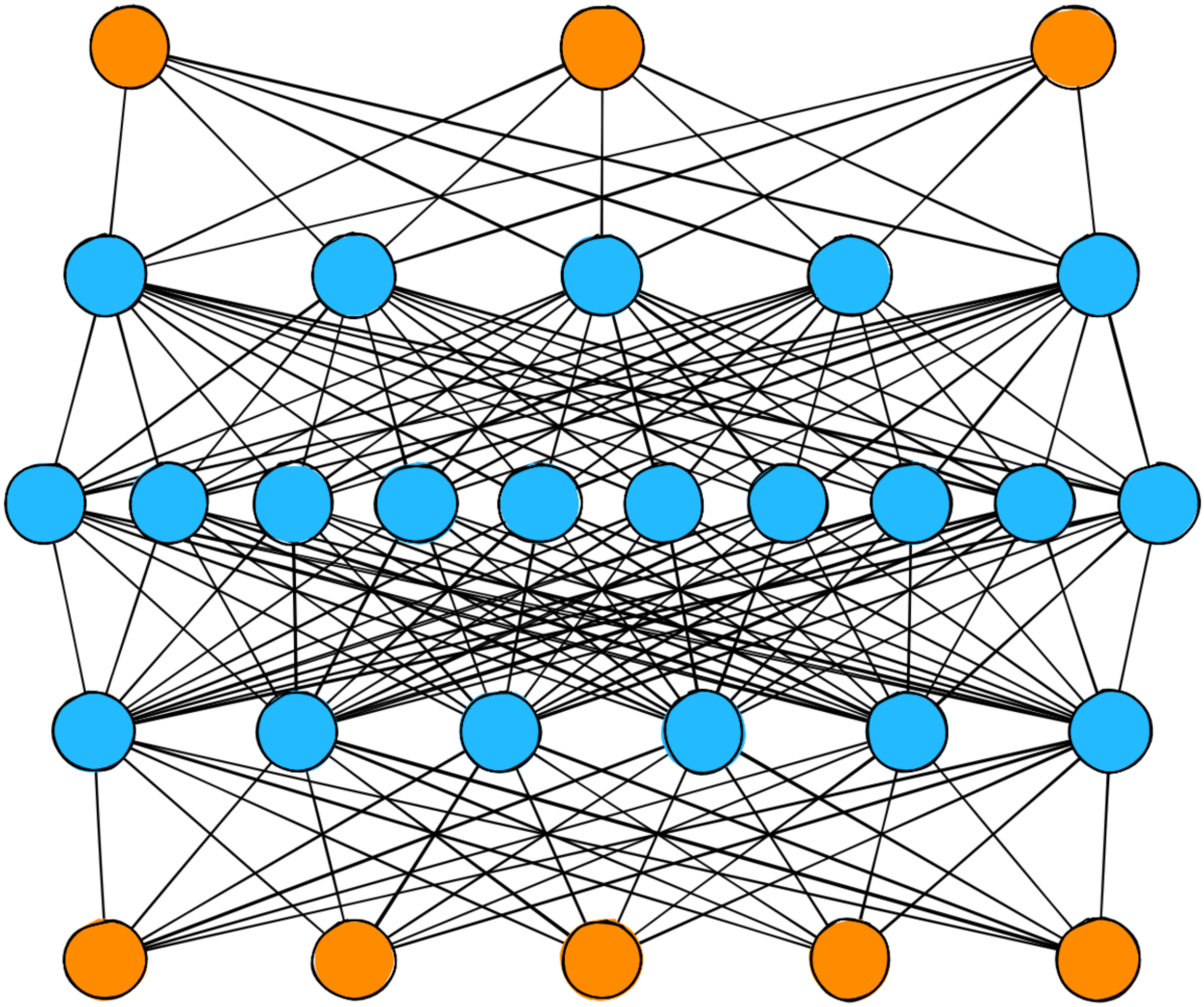
A Neural Network is made up of some number of layers.



Layers in a Neural Network

This diagram shows a three-layer Neural Network. The first layer is the **input layer**. This is where data enters the Network. The last layer is the **output layer**, responsible for emitting the transformed values from the Network.

The layer in the middle is called a **hidden layer**. Hidden layers make up the bulk of Neural Networks, and it's Networks with a lot of hidden layers that give rise to the phrase "Deep Learning": those Networks are "Deep". You can have as many hidden layers as your computer can handle.



A complicated Neural Network

A Neural Network exists in one of two phases: **Inference** or **Training**.

Inference describes the flow of data moving forward in one direction through the Network, from the input layer to the output layer. You can think of this as the network *predicting* the expected value, based on the given input values. For instance, you might feed it a picture of an animal, and the Network might answer "dog".

At an abstract level, specific Neurons fire based on the presence or absence of certain characteristics: Does it have fur? Does it have floppy ears? Is its tongue hanging out at an odd angle? Based on the answers to these questions, the Network might answer "Yes, I think this is a dog!".

I believe this is probably a dog and I am 99.9% sure

Inference is usually (though not always) how your users will interact with your Neural Network.

Training describes the flow of data forward *and* backward through the network. Based on the accuracy of the Network's predictions, changes ripple backwards, adjusting weights so that the network can improve and produce more accurate predictions in the future. You might feed the network a hundred photos of dogs, allowing the network to figure out - on its own - that all dogs tend to have fur and oddly angled tongues.

You may see the terms **forward propagation** and **backpropagation**. These are formal terms for describing Inference and an element of Training, respectively.

You often approach these two phases - Inference and Training - separately, and in this book we'll tackle them separately. We'll start with **Inference**: understanding how to interact with pretrained Neural Networks, passing them data and getting back predictions. Second, we'll discuss **Training**: how to build Neural Networks from scratch and train them to give us accurate predictions.

Let's learn how to load a Neural Network in our browser and start using it.

I Inference

2. Making Predictions

https://dljsbook.com/i/inference

> It may be surprising to the academic community to know that only a tiny fraction of the code in many machine learning systems is actually doing "machine learning". When we recognize that a mature system might end up being (at most) 5% machine learning code and (at least) 95% glue code, reimplementation rather than reuse of a clumsy API looks like a much better strategy. — D. Sculley et all[1]

Your boss strolls in with your first assignment at the cutting-edge photo sharing website (at which you currently work for the purposes of this example). "It's hard to search our website," the boss says. "Users don't tagging their photos with relevant text. People have no idea what to search for so nobody can find anything. It's a disaster!"

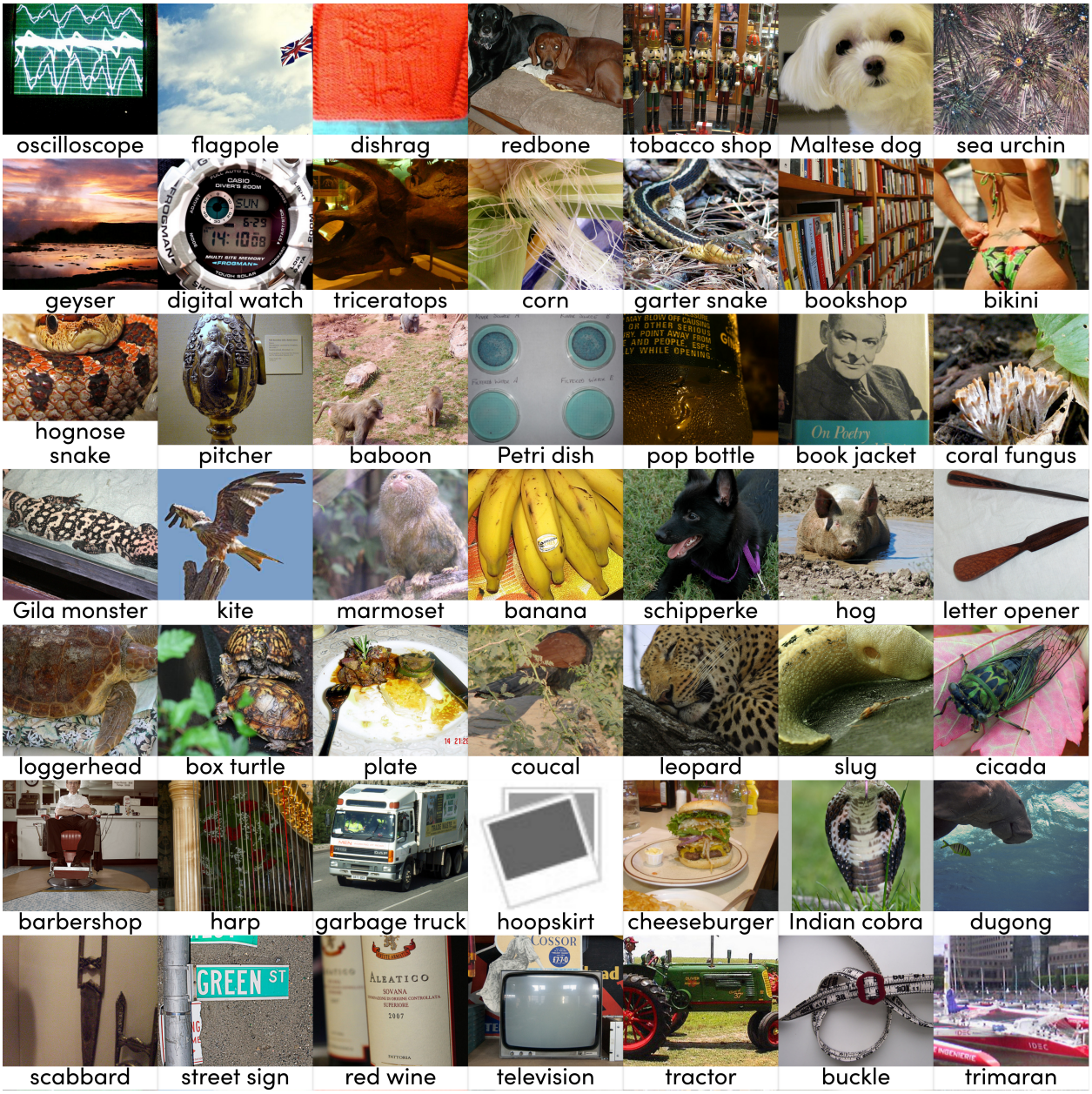
You, being the bright person you are, suggest a system that automatically suggests tags to the user based on the contents of the photo.

This is an example of an **Image Classification** problem. You want a Neural Network to predict the contents of an image and *classify* it, automatically.

For this example and subsequent chapters, we'll use a Neural Network called **MobileNet**. (Not all Neural Networks have names, but this one does). MobileNet is a Neural Network developed by Google. It's designed to be small and efficient, perfect for usage in a Browser.

All Neural Networks are trained on some collection of data, and if you have access to the original dataset, you can expect the Network to perform at or near its theoretical best. MobileNet was trained on a dataset called **ImageNet**. Its breadth and quality of classification (1000 categories to choose from), along with its size (an average of 500 images per category) has established it as a sort of "gold standard" for establishing Image Classification benchmarks [2].

[1] https://ai.google/research/pubs/pub43146
[2] http://www.image-net.org/challenges/LSVRC/



Samples of ImageNet

Let's start jumping into code and see what things look like. The `dljsbook` provides a handy interface for loading images from ImageNet. You can load an image with:

```
import { ImageNet } from '@dljsbook/data'

const imageNet = new ImageNet()

imageNet.getImage().then(data => data.print())
```

This will print an image to tfvis if it's available, or your browser console.

otterhound, otter hound

Printing an image from ImageNet to your browser

> Any call to `print` from `@dljsbook` accepts an optional HTML element as the first argument, like `print(document.getElementById('root'))`.

* * *

Now that you've seen the kind of data we're dealing with, let's see an example of **Inference** in action. We'll load MobileNet and feed it a random image, and see how accurate it is. The code to perform Inference with MobileNet is:

```
import * as tf from '@tensorflow/tfjs'
import { ImageNet } from '@dljsbook/data'
import { MobileNet } from '@dljsbook/models'

const imageNet = new ImageNet()

imageNet.ready().then(() => {

  // load the model
  return tf.loadModel(MobileNet.url).then(neuralNetwork => {

    // get a random image
    return imageNet.getImage().then(data => {

      // make a prediction
      const prediction = neuralNetwork.predict(data.image)

      // print the image
      data.print()

      // print the prediction
      imageNet.translatePrediction(prediction)
    })
  })
}).catch(err => {
  console.error(err)
})
```

If you open up tfvis or your console log, you should see an image and its associated category. Run the code again to load a new, random image. We'll step through this code line by line.

The first line loads a Neural Network from the web into your browser with a single line of code:

```
tf.loadModel('path_to_a_network');
```

`model` is often used interchangeably to refer to a Neural Network.

`tf.loadModel` returns a promise that resolves with the Neural Network. Once you've got your Network, you can load a random image from the ImageNet dataset:

```
const image = ImageNet.getRandomImage();
```

And send this random image through the Neural Network you loaded:

```
const prediction = neuralNetwork.predict(data.image);
imageNet.print(prediction)
```

And you've got your category. Let's next look at our `prediction` variable. To translate `prediction` into a normal array, you can write:

```
const pred = prediction.dataSync();
```

This will return 1000 numbers, corresponding to a thousand categories.

If you write `console.log(prediction)` directly, you'll see some strange output. This is because `prediction` is a **Tensor**, not a number. To see the output of `prediction` directly, write `prediction.print()`.

Tensors are a type of data container we'll cover in depth in Chapter 3.

You can get the most confident category with this snippet:

```
const pred = prediction.as1D().argMax().dataSync()[0]
```

This will pull the highest number, which corresponds to the category the Network is most confident matches the image. Then, you can map the value of prediction to a string matching the category:

```
console.log(ImageNet.classes[pred]);
> Dog
```

You might be wondering why the Network returns a number instead of the name of the category directly?

During the previous chapter, we discussed how Inference was like the Network looking at a picture of a dog and asking: "Does it have floppy ears? A tongue? It's a dog!" In reality, it's more accurate to say that a Network receives data as a series of numbers - 11010 - each representing a pixel value, and the Network emits a number - 61, for example - that you, the developer, interpet as "dog" or not. The emitted number is the result of some number of calculations that the Network has been trained to make internally.

Neural Networks deal entirely in numbers, and all the data you send through and collect from the Network needs to be converted appropriately. Generally, converting data *after* prediction is pretty straight forward. It's preparaing and converting the data *before* it goes into the model that's trickier.

Let's see how to do that.