CHAPTER 1

Generative Modeling

This chapter is a general introduction to the field of generative modeling. We shall first look at what it means to say that a model is *generative* and learn how it differs from the more widely studied *discriminative* modeling. Then I will introduce the framework and core mathematical ideas that will allow us to structure our general approach to problems that require a generative solution.

With this in place, we will then build our first example of a generative model (Naive Bayes) that is probabilistic in nature. We shall see that this allows us to generate novel examples that are outside of our training dataset, but shall also explore the reasons why this type of model may fail as the size and complexity of the space of possible creations increases.

What Is Generative Modeling?

A generative model can be broadly defined as follows:

> A generative model describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new data.

Suppose we have a dataset containing images of horses. We may wish to build a model that can generate a new image of a horse that has never existed but still looks real because the model has learned the general rules that govern the appearance of a horse. This is the kind of problem that can be solved using generative modeling. A summary of a typical generative modeling process is shown in Figure 1-1.

First, we require a dataset consisting of many examples of the entity we are trying to generate. This is known as the *training data*, and one such data point is called an *observation*.

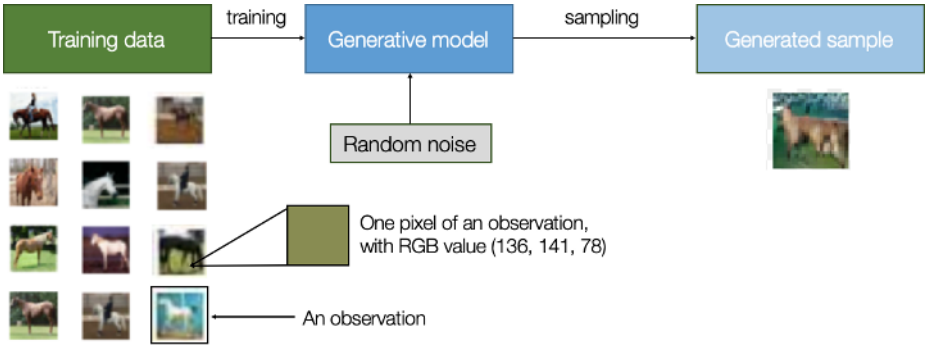

Figure 1-1. The generative modeling process

Each observation consists of many *features*—for an image generation problem, the features are usually the individual pixel values. It is our goal to build a model that can generate new sets of features that look as if they have been created using the same rules as the original data. Conceptually, for image generation this is an incredibly difficult task, considering the vast number of ways that individual pixel values can be assigned and the relatively tiny number of such arrangements that constitute an image of the entity we are trying to simulate.

A generative model must also be *probabilistic* rather than *deterministic*. If our model is merely a fixed calculation, such as taking the average value of each pixel in the dataset, it is not generative because the model produces the same output every time. The model must include a *stochastic* (random) element that influences the individual samples generated by the model.

In other words, we can imagine that there is some unknown probabilistic distribution that explains why some images are likely to be found in the training dataset and other images are not. It is our job to build a model that mimics this distribution as closely as possible and then sample from it to generate new, distinct observations that look as if they could have been included in the original training set.

Generative Versus Discriminative Modeling

In order to truly understand what generative modeling aims to achieve and why this is important, it is useful to compare it to its counterpart, *discriminative modeling*. If you have studied machine learning, most problems you will have faced will have most likely been discriminative in nature. To understand the difference, let's look at an example.

Suppose we have a dataset of paintings, some painted by Van Gogh and some by other artists. With enough data, we could train a discriminative model to predict if a given painting was painted by Van Gogh. Our model would learn that certain colors,

shapes, and textures are more likely to indicate that a painting is by the Dutch master, and for paintings with these features, the model would upweight its prediction accordingly. Figure 1-2 shows the discriminative modeling process—note how it differs from the generative modeling process shown in Figure 1-1.

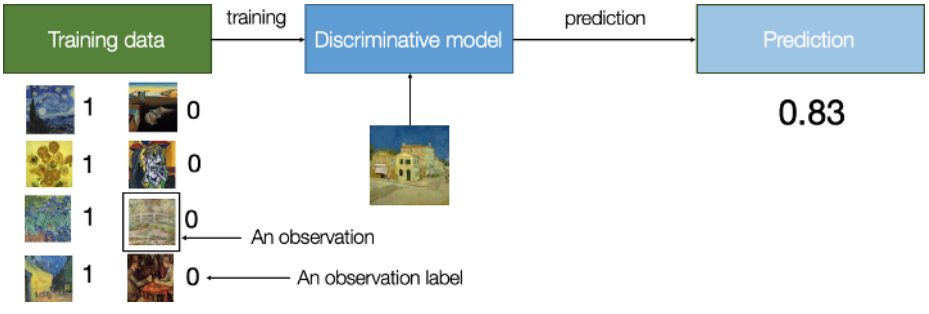

Figure 1-2. The discriminative modeling process

One key difference is that when performing discriminative modeling, each observation in the training data has a *label*. For a binary classification problem such as our artist discriminator, Van Gogh paintings would be labeled *1* and non–Van Gogh paintings labeled *0*. Our model then learns how to discriminate between these two groups and outputs the probability that a new observation has label *1*—i.e., that it was painted by Van Gogh.

For this reason, discriminative modeling is synonymous with *supervised learning*, or learning a function that maps an input to an output using a labeled dataset. Generative modeling is usually performed with an unlabeled dataset (that is, as a form of unsupervised learning), though it can also be applied to a labeled dataset to learn how to generate observations from each distinct class.

Let's take a look at some mathematical notation to describe the difference between generative and discriminative modeling.

Discriminative modeling estimates $p(y|\mathbf{x})$—the probability of a label y given observation $\mathbf{x}$.

Generative modeling estimates $p(\mathbf{x})$—the probability of observing observation $\mathbf{x}$.

If the dataset is labeled, we can also build a generative model that estimates the distribution $p(\mathbf{x}|y)$.

In other words, discriminative modeling attempts to estimate the probability that an observation **x** belongs to category *y*. Generative modeling doesn't care about labeling observations. Instead, it attempts to estimate the probability of seeing the observation at all.

The key point is that even if we were able to build a perfect discriminative model to identify Van Gogh paintings, it would still have no idea how to create a painting that looks like a Van Gogh. It can only output probabilities against existing images, as this is what it has been trained to do. We would instead need to train a generative model, which can output sets of pixels that have a high chance of belonging to the original training dataset.

Advances in Machine Learning

To understand why generative modeling can be considered the next frontier for machine learning, we must first look at why discriminative modeling has been the driving force behind most progress in machine learning methodology in the last two decades, both in academia and in industry.

From an academic perspective, progress in discriminative modeling is certainly easier to monitor, as we can measure performance metrics against certain high-profile classification tasks to determine the current best-in-class methodology. Generative models are often more difficult to evaluate, especially when the quality of the output is largely subjective. Therefore, much emphasis in recent years has been placed on training discriminative models to reach human or superhuman performance in a variety of image or text classification tasks.

For example, for image classification, the key breakthrough came in 2012 when a team led by Geoff Hinton at the University of Toronto won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a deep convolutional neural network. The competition involves classifying images into one of a thousand categories and is used as a benchmark to compare the latest state-of-the-art techniques. The deep learning model had an error rate of 16%—a massive improvement on the next best model, which only achieved a 26.2% error rate. This sparked a deep learning boom that has resulted in the error rate falling even further year after year. The 2015 winner achieved superhuman performance for the first time, with an error rate of 4%, and the current state-of-the-art model achieves an error rate of just 2%. Many would now consider the challenge a solved problem.

As well as it being easier to publish measurable results within an academic setting, discriminative modeling has historically been more readily applicable to business problems than generative modeling. Generally, in a business setting, we don't care *how* the data was generated, but instead want to know how a new example should be categorized or valued. For example:

- Given a satellite image, a government defense official would only care about the probability that it contains enemy units, not the probability that this particular image should appear.
- A customer relations manager would only be interested in knowing if the sentiment of an incoming email is positive or negative and wouldn't find much use in a generative model that could output examples of customer emails that don't yet exist.
- A doctor would want to know the chance that a given retinal image indicates glaucoma, rather than have access to a model that can generate novel pictures of the back of an eye.

As most solutions required by businesses are in the domain of discriminative modeling, there has been a rise in the number of *Machine-Learning-as-a-Service* (MLaaS) tools that aim to commoditize the use of discriminative modeling within industry, by largely automating the build, validation, and monitoring processes that are common to almost all discriminative modeling tasks.

REVIEWED
By Ivan at 5:32 am, May 29, 2020

The Rise of Generative Modeling

While discriminative modeling has so far provided the bulk of the impetus behind advances in machine learning, in the last three to five years many of the most interesting advancements in the field have come through novel applications of deep learning to generative modeling tasks.

In particular, there has been increased media attention on generative modeling projects such as StyleGAN from NVIDIA,[1] which is able to create hyper-realistic images of human faces, and the GPT-2 language model from OpenAI,[2] which is able to complete a passage of text given a short introductory paragraph.

Figure 1-3 shows the striking progress that has already been made in facial image generation since 2014.[3] There are clear positive applications here for industries such as game design and cinematography, and improvements in automatic music generation will also surely start to resonate within these domains. It remains to be seen whether we will one day read news articles or novels written by a generative model, but the recent progress in this area is staggering and it is certainly not outrageous to suggest that this one day may be the case. While exciting, this also raises ethical

1 Tero Karras, Samuli Laine, and Timo Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," 12 December 2018, *https://arxiv.org/abs/1812.04948*.

2 Alec Radford et al., "Language Models Are Unsupervised Multitask Learners," 2019, *https://paperswithcode.com/paper/language-models-are-unsupervised-multitask*.

3 Miles Brundage et al., "The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation," February 2018, *https://www.eff.org/files/2018/02/20/malicious_ai_report_final.pdf*.

questions around the proliferation of fake content on the internet and means it may become ever harder to trust what we see and read through public channels of communication.

Figure 1-3. Face generation using generative modeling has improved significantly in the last four years[4]

As well as the practical uses of generative modeling (many of which are yet to be discovered), there are three deeper reasons why generative modeling can be considered the key to unlocking a far more sophisticated form of artificial intelligence, that goes beyond what discriminative modeling alone can achieve.

First, purely from a theoretical point of view, we should not be content with only being able to excel at categorizing data but should also seek a more complete understanding of how the data was generated in the first place. This is undoubtedly a more difficult problem to solve, due to the high dimensionality of the space of feasible outputs and the relatively small number of creations that we would class as belonging to the dataset. However, as we shall see, many of the same techniques that have driven development in discriminative modeling, such as deep learning, can be utilized by generative models too.

Second, it is highly likely that generative modeling will be central to driving future developments in other fields of machine learning, such as reinforcement learning (the study of teaching agents to optimize a goal in an environment through trial and error). For example, we could use reinforcement learning to train a robot to walk across a given terrain. The general approach would be to build a computer simulation of the terrain and then run many experiments where the agent tries out different strategies. Over time the agent would learn which strategies are more successful than others and therefore gradually improve. A typical problem with this approach is that the physics of the environment is often highly complex and would need be calculated at each timestep in order to feed the information back to the agent to decide its next move. However, if the agent were able to simulate its environment through a generative model, it wouldn't need to test out the strategy in the computer simulation or in

4 Source: Brundage et al., 2018.

the real world, but instead could learn in its own *imaginary* environment. In Chapter 8 we shall see this idea in action, training a car to drive as fast as possible around a track by allowing it to learn directly from its own hallucinated environment.

Finally, if we are to truly say that we have built a machine that has acquired a form of intelligence that is comparable to a human's, generative modeling must surely be part of the solution. One of the finest examples of a generative model in the natural world is the person reading this book. Take a moment to consider what an incredible generative model you are. You can close your eyes and imagine what an elephant would look like from any possible angle. You can imagine a number of plausible different endings to your favorite TV show, and you can plan your week ahead by working through various futures in your mind's eye and taking action accordingly. Current neuroscientific theory suggests that our perception of reality is not a highly complex discriminative model operating on our sensory input to produce predictions of what we are experiencing, but is instead a generative model that is trained from birth to produce simulations of our surroundings that accurately match the future. Some theories even suggest that the output from this generative model is what we directly perceive as reality. Clearly, a deep understanding of how we can build machines to acquire this ability will be central to our continued understanding of the workings of the brain and general artificial intelligence.

With this in mind, let's begin our journey into the exciting world of generative modeling. To begin with we shall look at the simplest examples of generative models and some of the ideas that will help us to work through the more complex architectures that we will encounter later in the book.

The Generative Modeling Framework

Let's start by playing a generative modeling game in just two dimensions. I've chosen a rule that has been used to generate the set of points $\mathbf{X}$ in Figure 1-4. Let's call this rule p_{data}. Your challenge is to choose a different point $\mathbf{x} = (x_1, x_2)$ in the space that looks like it has been generated by the same rule.

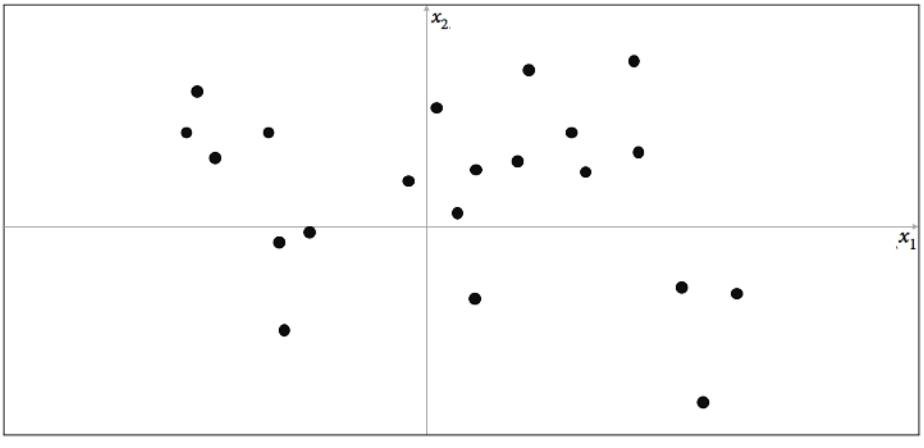

Figure 1-4. A set of points in two dimensions, generated by an unknown rule p_{data}

Where did you choose? You probably used your knowledge of the existing data points to construct a mental model, p_{model}, of whereabouts in the space the point is more likely to be found. In this respect, p_{model} is an *estimate* of p_{data}. Perhaps you decided that p_{model} should look like Figure 1-5—a rectangular box where points may be found, and an area outside of the box where there is no chance of finding any points. To generate a new observation, you can simply choose a point at random within the box, or more formally, *sample* from the distribution p_{model}. Congratulations, you have just devised your first generative model!

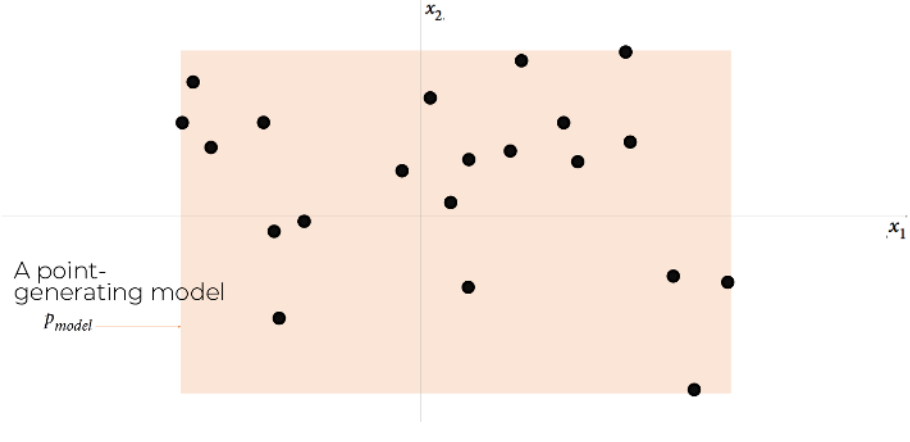

Figure 1-5. The orange box, p_{model}, is an estimate of the true data-generating distribution, p_{data}

While this isn't the most complex example, we can use it to understand what generative modeling is trying to achieve. The following framework sets out our motivations.

The Generative Modeling Framework

- We have a dataset of observations **X**.
- We assume that the observations have been generated according to some unknown distribution, p_{data}.
- A generative model p_{model} tries to mimic p_{data}. If we achieve this goal, we can sample from p_{model} to generate observations that appear to have been drawn from p_{data}.
- We are impressed by p_{model} if:
  - — Rule 1: It can generate examples that appear to have been drawn from p_{data}.
  - — Rule 2: It can generate examples that are suitably different from the observations in **X**. In other words, the model shouldn't simply reproduce things it has already seen.

Let's now reveal the true data-generating distribution, p_{data}, and see how the framework applies to this example.

As we can see from Figure 1-6, the data-generating rule is simply a uniform distribution over the land mass of the world, with no chance of finding a point in the sea.

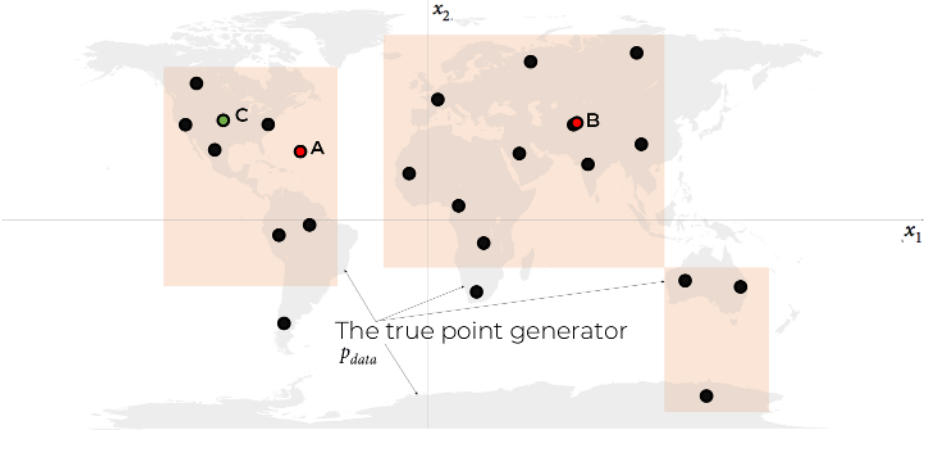

Figure 1-6. The orange box, p_{model}, is an estimate of the true data-generating distribution p_{data} (the gray area)

Clearly, our model p_{model} is an oversimplification of p_{data}. Points A, B, and C show three observations generated by p_{model} with varying degrees of success:

- **Point A** breaks Rule 1 of the Generative Modeling Framework—it does not appear to have been generated by p_{data} as it's in the middle of the sea.
- **Point B** is so close to a point in the dataset that we shouldn't be impressed that our model can generate such a point. If all the examples generated by the model were like this, it would break Rule 2 of the Generative Modeling Framework.
- **Point C** can be deemed a success because it could have been generated by p_{data} and is suitably different from any point in the original dataset.

The field of generative modeling is diverse and the problem definition can take a great variety of forms. However, in most scenarios the Generative Modeling Framework captures how we should broadly think about tackling the problem.

Let's now build our first nontrivial example of a generative model.

Probabilistic Generative Models

Firstly, if you have never studied probability, don't worry. To build and run many of the deep learning models that we shall see later in this book, it is not essential to have a deep understanding of statistical theory. However, to gain a full appreciation of the history of the task that we are trying to tackle, it's worth trying to build a generative model that doesn't rely on deep learning and instead is grounded purely in probabilistic theory. This way, you will have the foundations in place to understand all generative models, whether based on deep learning or not, from the same probabilistic standpoint.

If you already have a good understanding of probability, that's great and much of the next section may already be familiar to you. However, there is a fun example in the middle of this chapter, so be sure not to miss out on that!

As a first step, we shall define four key terms: *sample space*, *density function*, *parametric modeling*, and *maximum likelihood estimation*.

Sample Space

The *sample space* is the complete set of all values an observation $\mathbf{x}$ can take.

In our previous example, the sample space consists of all points of latitude and longitude $\mathbf{x} = (x_1, x_2)$ on the world map.

For example, $\mathbf{x}$ = (40.7306, –73.9352) is a point in the sample space (New York City).

Probability Density Function

A *probability density function* (or simply *density function*), $p(\mathbf{x})$, is a function that maps a point $\mathbf{x}$ in the sample space to a number between 0 and 1. The sum[5] of the density function over all points in the sample space must equal 1, so that it is a well-defined probability distribution.[6]

In the world map example, the density function of our model is 0 outside of the orange box and constant inside of the box.

While there is only one true density function p_{data} that is assumed to have generated the observable dataset, there are infinitely many density functions p_{model} that we can use to estimate p_{data}. In order to structure our approach to finding a suitable $p_{model}(\mathbf{X})$ we can use a technique known as *parametric modeling*.

Parametric Modeling

A *parametric model*, $p_\theta(\mathbf{x})$, is a family of density functions that can be described using a finite number of parameters, θ.

The family of all possible boxes you could draw on Figure 1-5 is an example of a parametric model. In this case, there are four parameters: the coordinates of the bottom-left (θ_1, θ_2) and top-right (θ_3, θ_4) corners of the box.

5 Or integral if the sample space is continuous.

6 If the sample space is discrete, $p(\mathbf{x})$ is simply the probability assigned to observing point $\mathbf{x}$.

Thus, each density function $p_\theta(\mathbf{x})$ in this parametric model (i.e., each box) can be uniquely represented by four numbers, $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$.

NOTE

Likelihood

The *likelihood* $\mathcal{L}(\theta \mid \mathbf{x})$ of a parameter set θ is a function that measures the plausibility of θ, given some observed point $\mathbf{x}$.

It is defined as follows:

$$\mathcal{L}(\theta|\mathbf{x}) = p_\theta(\mathbf{x})$$

That is, the likelihood of θ given some observed point $\mathbf{x}$ is defined to be the value of the density function parameterized by θ, at the point $\mathbf{x}$.

If we have a whole dataset $\mathbf{X}$ of independent observations then we can write:

$$\mathcal{L}(\theta|\mathbf{X}) = \prod_{\mathbf{x} \in \mathbf{X}} p_\theta(\mathbf{x})$$

Since this product can be quite computationally difficult to work with, we often use the *log-likelihood* ℓ instead:

$$\ell(\theta|\mathbf{X}) = \sum_{\mathbf{x} \in \mathbf{X}} \log p_\theta(\mathbf{x})$$

There are statistical reasons why the likelihood is defined in this way, but it is enough for us to understand why, intuitively, this makes sense. We are simply defining the likelihood of a set of parameters θ to be equal to the probability of seeing the data under the model parameterized by θ.

In the world map example, an orange box that only covered the left half of the map would have a likelihood of 0—it couldn't possibly have generated the dataset as we have observed points in the right half of the map. The orange box in Figure 1-5 has a positive likelihood as the density function is positive for all data points under this model.

It therefore makes intuitive sense that the focus of parametric modeling should be to find the optimal value $\hat{\theta}$ of the parameter set that maximizes the likelihood of observing the dataset $\mathbf{X}$. This technique is quite appropriately called *maximum likelihood estimation*.

Maximum Likelihood Estimation

Maximum likelihood estimation is the technique that allows us to estimate $\hat{\theta}$—the set of parameters θ of a density function, $p_\theta(\mathbf{x})$, that are most likely to explain some observed data $\mathbf{X}$.

More formally:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ \mathscr{L}(\theta|\mathbf{X})$$

$\hat{\theta}$ is also called the *maximum likelihood estimate* (MLE).

We now have all the necessary terminology to start describing how we can build a probabilistic generative model.

Most chapters in this book will contain a short story that helps to describe a particular technique. In this chapter, we shall start by taking a trip to planet Wrodl, where our first generative modeling assignment awaits…

Hello Wrodl!

The year is 2047 and you are delighted to have been appointed as the new Chief Fashion Officer (CFO) of Planet Wrodl. As CFO, it is your sole responsibility to create new and exciting fashion trends for the inhabitants of the planet to follow.

The Wrodlers are known to be quite particular when it comes to fashion, so your task is to generate new styles that are similar to those that already exist on the planet, but not identical.

On arrival, you are presented with a dataset featuring 50 observations of Wrodler fashion (Figure 1-7) and told that you have a day to come up with 10 new styles to present to the Fashion Police for inspection. You're allowed to play around with hairstyles, hair color, glasses, clothing type, and clothing color to create your masterpieces.

Figure 1-7. Headshots of 50 Wrodlers[7]

As you're a data scientist at heart, you decide to deploy a generative model to solve the problem. After a brief visit to the Intergalactic Library, you pick up a book called *Generative Deep Learning* and begin to read…

To be continued…

Your First Probabilistic Generative Model

Let's take a closer look at the Wrodl dataset. It consists of $N = 50$ observations of fashions currently seen on the planet. Each observation can be described by five features, (*accessoriesType*, *clothingColor*, *clothingType*, *hairColor*, *topType*), as shown in Table 1-1.

Table 1-1. The first 10 observations in the Wrodler face dataset

face_id	accessoriesType	clothingColor	clothingType	hairColor	topType
0	Round	White	ShirtScoopNeck	Red	ShortHairShortFlat
1	Round	White	Overall	SilverGray	ShortHairFrizzle
2	Sunglasses	White	ShirtScoopNeck	Blonde	ShortHairShortFlat
3	Round	White	ShirtScoopNeck	Red	LongHairStraight
4	Round	White	Overall	SilverGray	NoHair
5	Blank	White	Overall	Black	LongHairStraight
6	Sunglasses	White	Overall	SilverGray	LongHairStraight
7	Round	White	ShirtScoopNeck	SilverGray	ShortHairShortFlat

7 Images sourced from *https://getavataaars.com*.

face_id	accessoriesType	clothingColor	clothingType	hairColor	topType
8	Round	Pink	Hoodie	SilverGray	LongHairStraight
9	Round	PastelOrange	ShirtScoopNeck	Blonde	LongHairStraight

The possible values for each feature include:

- 7 different hairstyles (*topType*):
  - *NoHair*, *LongHairBun*, *LongHairCurly*, *LongHairStraight*, *ShortHairShortWaved*, *ShortHairShortFlat*, *ShortHairFrizzle*
- 6 different hair colors (*hairColor*):
  - *Black*, *Blonde*, *Brown*, *PastelPink*, *Red*, *SilverGray*
- 3 different kinds of glasses (*accessoriesType*):
  - *Blank*, *Round*, *Sunglasses*
- 4 different kinds of clothing (clothingType):
  - *Hoodie*, *Overall*, *ShirtScoopNeck*, *ShirtVNeck*
- 8 different clothing colors (*clothingColor*):
  - *Black*, *Blue01*, *Gray01*, *PastelGreen*, *PastelOrange*, *Pink*, *Red*, *White*

There are $7 \times 6 \times 3 \times 4 \times 8 = 4{,}032$ different combinations of these features, so there are 4,032 points in the sample space.

We can imagine that our dataset has been generated by some distribution p_{data} that favors some feature values over others. For example, we can see from the images in Figure 1-7 that white clothing seems to be a popular choice, as are silver-gray hair and scoop-neck T-shirts.

The problem is that we do not know p_{data} explicitly—all we have to work with is the sample of observations $\mathbf{X}$ generated by p_{data}. The goal of generative modeling is to use these observations to build a p_{model} that can accurately mimic the observations produced by p_{data}.

To achieve this, we could simply assign a probability to each possible combination of features, based on the data we have seen. Therefore, this parametric model would have $d = 4{,}031$ parameters—one for each point in the sample space of possibilities, minus one since the value of the last parameter would be forced so that the total sums to 1. Thus the parameters of the model that we are trying to estimate are $(\theta_1, \ldots, \theta_{4031})$.

This particular class of parametric model is known as a *multinomial distribution*, and the maximum likelihood estimate $\widehat{\theta}_j$ of each parameter is given by:

$$\widehat{\theta}_j = \frac{n_j}{N}$$

where n_j is the number of times that combination j was observed in the dataset and $N = 50$ is the total number of observations.

In other words, the estimate for each parameter is just the proportion of times that its corresponding combination was observed in the dataset.

For example, the following combination (let's call it combination 1) appears twice in the dataset:

- (*LongHairStraight, Red, Round, ShirtScoopNeck, White*)

Therefore:

$$\widehat{\theta}_1 = 2/50 = 0.04$$

As another example, the following combination (let's call it combination 2) doesn't appear at all in the dataset:

- (*LongHairStraight, Red, Round, ShirtScoopNeck, Blue01*)

Therefore:

$$\widehat{\theta}_2 = 0/50 = 0$$

We can calculate all of the $\widehat{\theta}_j$ values in this way, to define a distribution over our sample space. Since we can sample from this distribution, our list could potentially be called a generative model. However, it fails in one major respect: it can never generate anything that it hasn't already seen, since $\widehat{\theta}_j = 0$ for any combination that wasn't in the original dataset **X**.

To address this, we could assign an additional *pseudocount* of 1 to each possible combination of features. This is known as *additive smoothing*. Under this model, our MLE for the parameters would be:

$$\widehat{\theta}_j = \frac{n_j + 1}{N + d}$$

REVIEWED
By Ivan at 6:16 pm, May 30, 2020

Now, every single combination has a nonzero probability of being sampled, including those that were not in the original dataset. However, this still fails to be a satisfactory generative model, because the probability of observing a point not in the original dataset is just a constant. If we tried to use such a model to generate Picasso paintings, it would assign just as much weight to a random collection of colorful pixels as to a replica of a Picasso painting that differs only very slightly from a genuine painting.

We would ideally like our generative model to upweight areas of the sample space that it believes are more likely, due to some inherent structure learned from the data, rather than just placing all probabilistic weight on the points that are present in the dataset.

REVIEWED
By Ivan at 6:25 pm, May 30, 2020

To achieve this, we need to choose a different parametric model.

Naive Bayes

The Naive Bayes parametric model makes use of a simple assumption that drastically reduces the number of parameters we need to estimate.

We make the *naive* assumption that each feature x_j is *independent* of every other feature x_k.[8] Relating this to the Wrodl dataset, we are assuming that the choice of hair color has no impact on the choice of clothing type, and the type of glasses that someone wears has no impact on their hairstyle, for example. More formally, for all features x_j, x_k:

$$p(x_j \mid x_k) = p(x_j)$$

This is known as the *Naive Bayes* assumption. To apply this assumption, we first make use of the chain rule of probability to write the density function as a product of conditional probabilities:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, \ldots, x_K) \\ &= p(x_2, \ldots, x_K \mid x_1) p(x_1) \\ &= p(x_3, \ldots, x_K \mid x_1, x_2) p(x_2 \mid x_1) p(x_1) \\ &= \prod_{k=1}^{K} p(x_k \mid x_1, \ldots, x_{k-1}) \end{aligned}$$

8 When a response variable y is present, the Naive Bayes assumption states that there is *conditional* independence between each pair of features x_j, x_k given y.

where K is the total number of features (i.e., 5 for the Wrodl example).

We now apply the Naive Bayes assumption to simplify the last line:

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k)$$

This is the Naive Bayes model. The problem is reduced to estimating the parameters $\theta_{kl} = p(x_k = l)$ for each feature separately and multiplying these to find the probability for any possible combination.

How many parameters do we now need to estimate? For each feature, we need to estimate a parameter for each value that the feature can take. Therefore, in the Wrodl example, this model is defined by only $7 + 6 + 3 + 4 + 8 - 5 = 23$ parameters.[9]

The maximum likelihood estimates $\widehat{\theta_{kl}}$ are as follows:

$$\widehat{\theta_{kl}} = \frac{n_{kl}}{N}$$

- 7 different hairstyles (*topType*):
  - *NoHair, LongHairBun, LongHairCurly, LongHairStraight, ShortHairShortWaved, ShortHairShortFlat, ShortHairFrizzle*
- 6 different hair colors (*hairColor*):
  - *Black, Blonde, Brown, PastelPink, Red, SilverGray*
- 3 different kinds of glasses (*accessoriesType*):
  - *Blank, Round, Sunglasses*
- 4 different kinds of clothing (clothingType):
  - *Hoodie, Overall, ShirtScoopNeck, ShirtVNeck*
- 8 different clothing colors (*clothingColor*):
  - *Black, Blue01, Gray01, PastelGreen, PastelOrange, Pink, Red, White*

where $\widehat{\theta_{kl}}$ is the number of times that the feature k takes on the value l in the dataset and $N = 50$ is the total number of observations.

Table 1-2 shows the calculated parameters for the Wrodl dataset.

Table 1-2. The MLEs for the parameters under the Naive Bayes model

topType	n	$\hat{\theta}$
NoHair	7	0.14
LongHairBun	0	0.00
LongHairCurly	1	0.02
LongHairStraight	23	0.46
ShortHairShortWaved	1	0.02
ShortHairShortFlat	11	0.22
ShortHairFrizzle	7	0.14
Grand Total	*50*	*1.00*

hairColor	n	$\hat{\theta}$
Black	7	0.14
Blonde	6	0.12
Brown	2	0.04
PastelPink	3	0.06
Red	8	0.16
SilverGrey	24	0.48
Grand Total	*50*	*1.00*

clothingColor	n	$\hat{\theta}$
Black	0	0.00
Blue01	4	0.08
Grey01	10	0.20
PastelGreen	5	0.10
PastelOrange	2	0.04
Pink	4	0.08
Red	3	0.06
White	22	0.44
Grand Total	*50*	*1.00*

9 The –5 is due to the fact that the last parameter for each feature is forced to ensure that the sum of the parameters for this feature sums to 1.

accessoriesType	n	$\hat{\theta}$
Blank	11	0.22
Round	22	0.44
Sunglasses	17	0.34
Grand Total	*50*	*1.00*

clothingType	n	$\hat{\theta}$
Hoodie	7	0.14
Overall	18	0.36
ShirtScoopNeck	19	0.38
ShirtVNeck	6	0.12
Grand Total	*50*	*1.00*

To calculate the probability of the model generating some observation **x**, we simply multiply together the individual feature probabilities. For example:

$$
\begin{aligned}
&p(\textit{LongHairStraight, Red, Round, ShirtScoopNeck, White}) \\
&= p(\textit{LongHairStraight}) \times p(\textit{Red}) \times p(\textit{Round}) \times p(\textit{ShirtScoopNeck}) \times p(\textit{White}) \\
&= 0.46 \times 0.16 \times 0.44 \times 0.38 \times 0.44 \\
&= 0.0054
\end{aligned}
$$

Notice that this combination doesn't appear in the original dataset, but our model still allocates it a nonzero probability, so it is still able to be generated. Also, it has a higher probability of being sampled than, say, *(LongHairStraight, Red, Round, ShirtScoop-Neck, White)*, because white clothing appears more often than blue clothing in the dataset.

Therefore, a Naive Bayes model is able to learn some structure from the data and use this to generate new examples that were not seen in the original dataset. The model has estimated the probability of seeing each feature value independently, so that under the Naive Bayes assumption we can multiply these probabilities to build our full density function, $p_\theta(\mathbf{x})$.

Figure 1-8 shows 10 observations sampled from the model.

Figure 1-8. Ten new Wrodl styles, generated using the Naive Bayes model

For this simple problem, the Naive Bayes assumption that each feature is independent of every other feature is reasonable and therefore produces a good generative model.

Now let's see what happens when this assumption breaks down

Hello Wrodl! Continued

You feel a certain sense of pride as you look upon the 10 new creations generated by your Naive Bayes model. Glowing with success, you turn your attention to another planet's fashion dilemma—but this time the problem isn't quite as simple.

On the conveniently named Planet Pixel, the dataset you are provided with doesn't consist of the five high-level features that you saw on Wrodl (*hairColor, accessoriesType,* etc.), but instead contains just the values of the 32 × 32 pixels that make up each image. Thus each observation now has 32 × 32 = 1,024 features and each feature can take any of 256 values (the individual colors in the palette).

Images from the new dataset are shown in Figure 1-9, and a sample of the pixel values for the first 10 observations appears in Table 1-3.

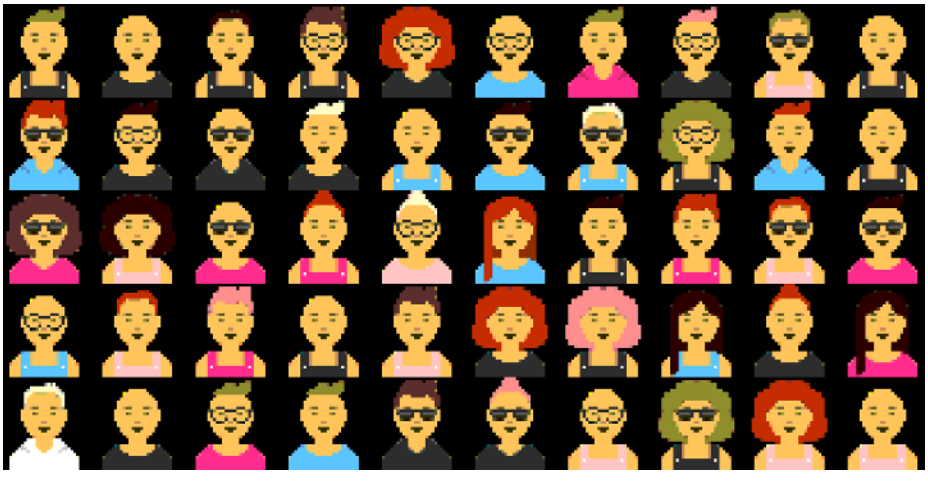

Figure 1-9. Fashions on Planet Pixel

Table 1-3. The values of pixels 458–467 from the first 10 observations on Planet Pixel

face_id	px_458	px_459	px_460	px_461	px_462	px_463	px_464	px_465	px_466	px_467
0	49	14	14	19	7	5	5	12	19	14
1	43	10	10	17	9	3	3	18	17	10
2	37	12	12	14	11	4	4	6	14	12
3	54	9	9	14	10	4	4	16	14	9
4	2	2	5	2	4	4	4	4	2	5
5	44	15	15	21	14	3	3	4	21	15
6	12	9	2	31	16	3	3	16	31	2
7	36	9	9	13	11	4	4	12	13	9
8	54	11	11	16	10	4	4	19	16	11
9	49	17	17	19	12	6	6	22	19	17

You decide to try your trusty Naive Bayes model once more, this time trained on the pixel dataset. The model will estimate the maximum likelihood parameters that govern the distribution of the color of each pixel so that you are able to sample from this distribution to generate new observations. However, when you do so, it is clear that something has gone very wrong.

Rather than producing novel fashions, the model outputs 10 very similar images that have no distinguishable accessories or clear blocks of hair or clothing color (Figure 1-10). Why is this?

Figure 1-10. Ten new Planet Pixel styles, generated by the Naive Bayes model

The Challenges of Generative Modeling

First, since the Naive Bayes model is sampling pixels independently, it has no way of knowing that two adjacent pixels are probably quite similar in shade, as they are part of the same item of clothing, for example. The model can generate the facial color and mouth, as all of these pixels in the training set are roughly the same shade in each observation; however for the T-shirt pixels, each pixel is sampled at random from a variety of different colors in the training set, with no regard to the colors that have been sampled in neighboring pixels. Additionally, there is no mechanism for pixels near the eyes to form circular glasses shapes, or red pixels near the top of the image to exhibit a wavy pattern to represent a particular hairstyle, for example.

Second, there are now an incomprehensibly vast number of possible observations in the sample space. Only a tiny proportion of these are recognizable faces, and an even smaller subset are faces that adhere to the fashion rules on Planet Pixel. Therefore, if our Naive Bayes model is working directly with the highly correlated pixel values, the chance of it finding a satisfying combination of values is incredibly small.

In summary, on Planet Wrodl individual features are independent and the sample space is relatively small, so Naive Bayes works well. On Planet Pixel, the assumption that every pixel value is independent of every other pixel value clearly doesn't hold. Pixel values are highly correlated and the sample space is vast, so finding a valid face by sampling pixels independently is almost impossible. This explains why Naive Bayes models cannot be expected to work well on raw image data.

This example highlights the two key challenges that a generative model must overcome in order to be successful.

Generative Modeling Challenges

- How does the model cope with the high degree of conditional dependence between features?
- How does the model find one of the tiny proportion of satisfying possible generated observations among a high-dimensional sample space?

Deep learning is the key to solving both of these challenges.

We need a model that can infer relevant structure from the data, rather than being told which assumptions to make in advance. This is exactly where deep learning excels and is one of the key reasons why the technique has driven the major recent advances in generative modeling.

The fact that deep learning can form its own features in a lower-dimensional space means that it is a form of *representation learning*. It is important to understand the key concepts of representation learning before we tackle deep learning in the next chapter.

Representation Learning

The core idea behind representation learning is that instead of trying to model the high-dimensional sample space directly, we should instead describe each observation in the training set using some low-dimensional *latent* space and then learn a mapping function that can take a point in the latent space and map it to a point in the original domain. In other words, each point in the latent space is the *representation* of some high-dimensional image.

What does this mean in practice? Let's suppose we have a training set consisting of grayscale images of biscuit tins (Figure 1-11).

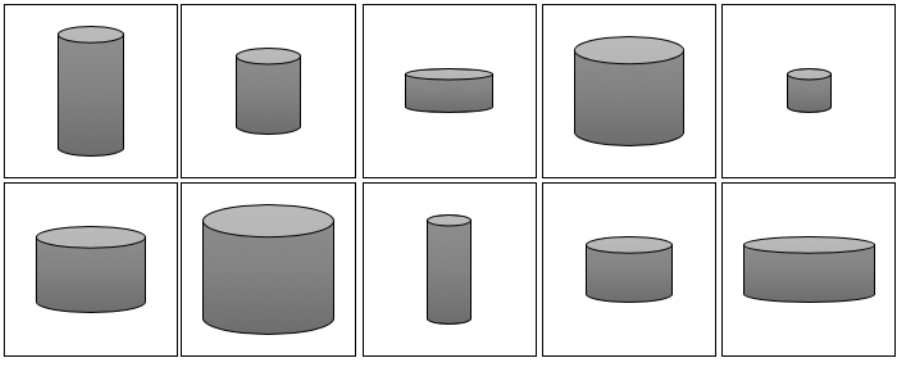

Figure 1-11. The biscuit tin dataset

To us, it is obvious that there are two features that can uniquely represent each of these tins: the height and width of the tin. Given a height and width, we could draw the corresponding tin, even if its image wasn't in the training set. However, this is not so easy for a machine—it would first need to establish that height and width are the two latent space dimensions that best describe this dataset, then learn the mapping function, f, that can take a point in this space and map it to a grayscale biscuit tin image. The resulting latent space of biscuit tins and generation process are shown in Figure 1-12.

Deep learning gives us the ability to learn the often highly complex mapping function f in a variety of ways. We shall explore some of the most important techniques in later chapters of this book. For now, it is enough to understand at a high level what representation learning is trying to achieve.

One of the advantages of using representation learning is that we can perform operations within the more manageable latent space that affect high-level properties of the image. It is not obvious how to adjust the shading of every single pixel to make a given biscuit tin image *taller*. However, in the latent space, it's simply a case of adding 1 to the *height* latent dimension, then applying the mapping function to return to the image domain. We shall see an explicit example of this in the next chapter, applied not to biscuit tins but to faces.

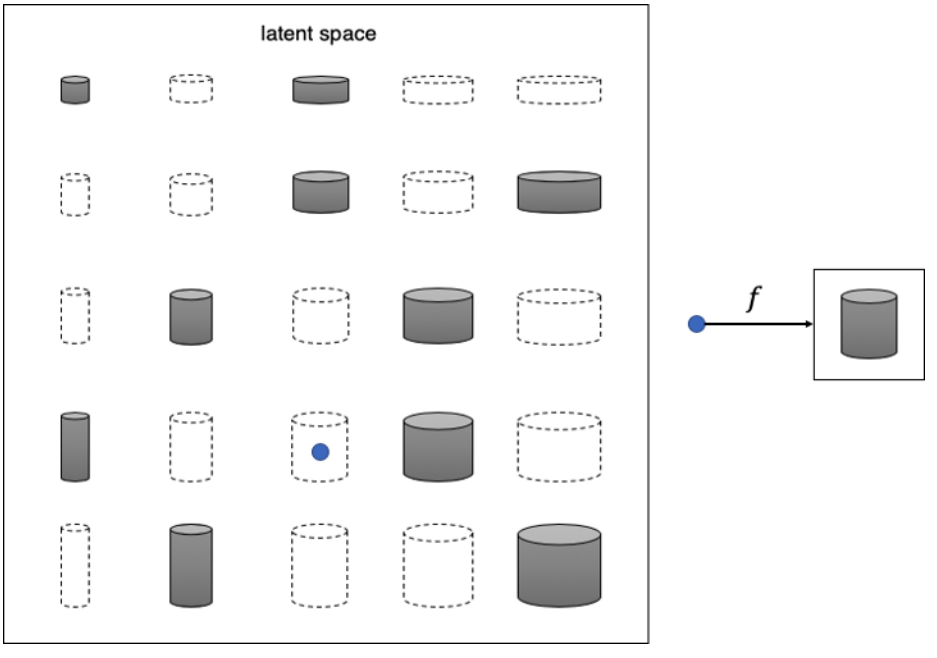

Figure 1-12. The latent space of biscuit tins and the function f that maps a point in the latent space to the original image domain

REVIEWED
By Ivan at 3:08 pm, Jun 06, 2020

Representation learning comes so naturally to us as humans that you may never have stopped to think just how amazing it is that we can do it so effortlessly. Suppose you wanted to describe your appearance to someone who was looking for you in a crowd of people and didn't know what you looked like. You wouldn't start by stating the color of pixel 1 of your hair, then pixel 2, then pixel 3, etc. Instead, you would make the reasonable assumption that the other person has a general idea of what an average human looks like, then amend this baseline with features that describe groups of pixels, such as *I have very blonde hair* or *I wear glasses*. With no more than 10 or so of these statements, the person would be able to map the description back into pixels to generate an image of you in their head. The image wouldn't be perfect, but it would be a close enough likeness to your actual appearance for them to find you among possibly hundreds of other people, even if they've never seen you before.

Note that representation learning doesn't just assign values to a given set of features such as *blondeness of hair*, *height*, etc., for some given image. The power of representation learning is that it actually learns which features are most important for it to describe the given observations and how to generate these features from the raw data. Mathematically speaking, it tries to find the highly nonlinear *manifold* on which the data lies and then establish the dimensions required to fully describe this space. This is shown in Figure 1-13.

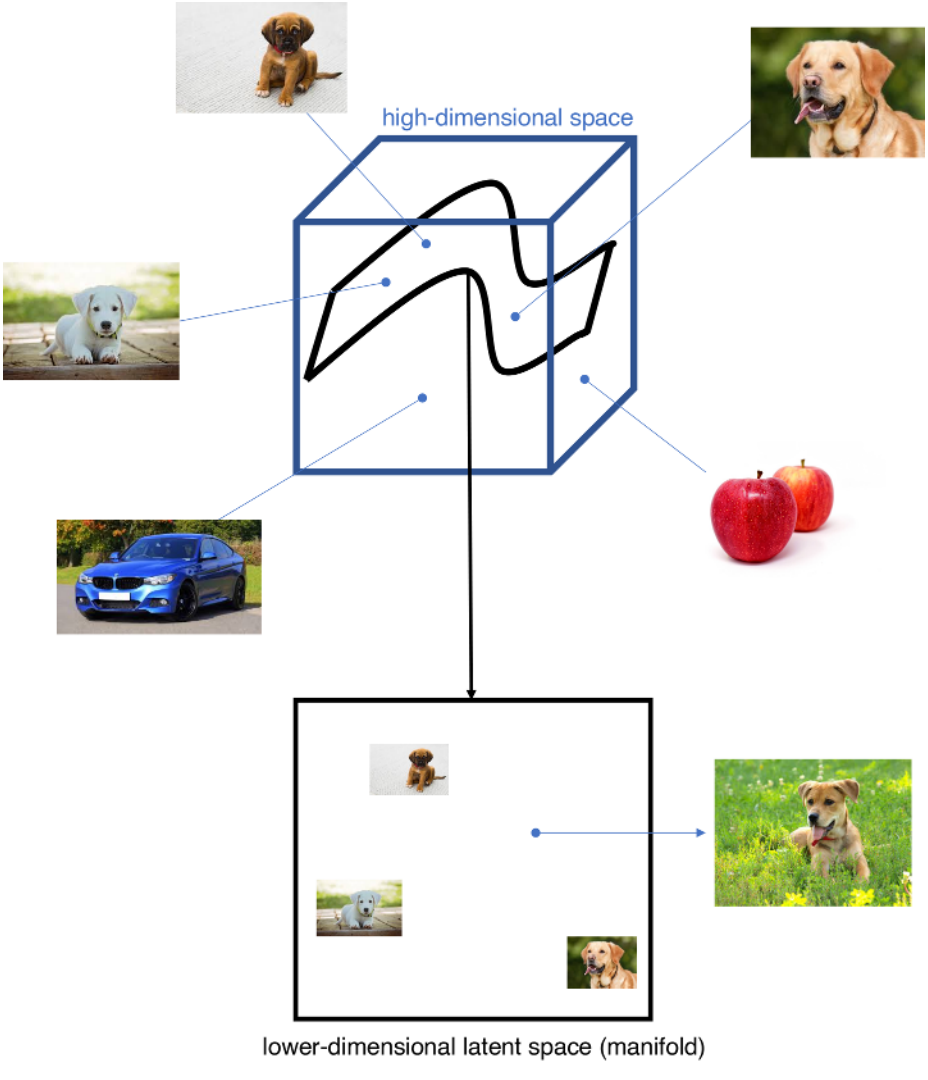

Figure 1-13. The cube represents the extremely high-dimensional space of all images; representation learning tries to find the lower-dimensional latent subspace or manifold on which particular kinds of image lie (for example, the dog manifold)

In summary, representation learning establishes the most relevant high-level features that describe how groups of pixels are displayed so that is it likely that any point in the latent space is the representation of a well-formed image. By tweaking the values of features in the latent space we can produce novel representations that, when

mapped back to the original image domain, have a much better chance of looking *real* than if we'd tried to work directly with the individual raw pixels.

Now that you have an understanding of representation learning, which forms the backbone of many of the generative deep learning examples in this book, all that remains is to set up your environment so that you can begin building generative deep learning models of your own.

REVIEWED
By Ivan at 3:18 pm, Jun 06, 2020

Setting Up Your Environment

Throughout this book, there are many worked examples of how to build the models that we will be discussing in the text.

To get access to these examples, you'll need to clone the Git repository that accompanies this book. Git is an open source version control system and will allow you to copy the code locally so that you can run the notebooks on your own machine, or perhaps in a cloud-based environment. You may already have this installed, but if not, follow the instructions relevant to your operating system.

To clone the repository for this book, navigate to the folder where you would like to store the files and type the following into your terminal:

```
git clone https://github.com/davidADSP/GDL_code.git
```

Always make sure that you have the most up-to-date version of the codebase by running the following command:

```
git pull
```

You should now be able to see the files in a folder on your machine.

Next, you need to set up a virtual environment. This is simply a folder into which you'll install a fresh copy of Python and all of the packages that we will be using in this book. This way, you can be sure that your system version of Python isn't affected by any of the libraries that we will be using.

If you are using Anaconda, you can set up a virtual environment as follows:

```
conda create -n generative python=3.6 ipykernel
```

If not, you can install *virtualenv* and *virtualenvwrapper* with the command:[10]

```
pip install virtualenv virtualenvwrapper
```

10 For full instructions on installing *virtualenvwrapper*, consult the documentation.

You will also need to add the following lines to your shell startup script (e.g., *.bash_profile*):

```
export WORKON_HOME=$HOME/.virtualenvs ❶
export VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3 ❷
source /usr/local/bin/virtualenvwrapper.sh ❸
```

❶ The location where your virtual environments will be stored

❷ The default version of Python to use when a virtual environment is created—make sure this points at Python 3, rather than Python 2.

❸ Reloads the *virtualenvwrapper* initialization script

To create a virtual environment called *generative*, simply enter the following into your terminal:

```
mkvirtualenv generative
```

You'll know that you're inside the virtual environment because your terminal will show (`generative`) at the start of the prompt.

Now you can go ahead and install all the packages that we'll be using in this book with the following command:

```
pip install -r requirements.txt
```

Throughout this book, we will use Python 3. The *requirements.txt* file contains the names and version numbers of all the packages that you will need to run the examples.

To check everything works as expected, from inside your virtual environment type `python` into your terminal and then try to import Keras (a deep learning library that we will be using extensively in this book). You should see a Python 3 prompt, with Keras reporting that it is using the TensorFlow backend as shown in Figure 1-14.

```
Python 3.6.5 (default, Oct  6 2018, 09:49:35)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import keras
Using TensorFlow backend.
>>> keras.__version__
'2.2.4'
```

Figure 1-14. Setting up your environment

Finally, you will need to ensure you are set up to access your virtual environment through Jupyter notebooks on your machine. Jupyter is a way to interactively code in

Python through your browser and is a great option for developing new ideas and sharing code. Most of the examples in this book are written using Jupyter notebooks.

To do this, run the following command from your terminal inside your virtual environment:

```
python -m ipykernel install --user --name generative ❶
```

❶ This gives you access to the virtual environment that you've just set up (`generative`) inside Jupyter notebooks.

To check that it has installed correctly, navigate in your terminal to the folder where you have cloned the book repository and type:

```
jupyter notebook
```

A window should open in your browser showing a screen similar to Figure 1-15. Click the notebook you wish to run and, from the *Kernel* → *Change kernel* dropdown, select the `generative` virtual environment.

Figure 1-15. Jupyter notebook

You are now ready to start building generative deep neural networks.

Summary

This chapter introduced the field of generative modeling, an important branch of machine learning that complements the more widely studied discriminative modeling. Our first basic example of a generative model utilized the Naive Bayes assumption to produce a probability distribution that was able to represent inherent structure in the data and generate examples outside of the training set. We also saw how these kinds of basic models can fail as the complexity of the generative task grows, and analyzed the general challenges associated with generative modeling. Finally, we took our first look at representation learning, an important concept that forms the core of many generative models.

In Chapter 2, we will begin our exploration of deep learning and see how to use Keras to build models that can perform discriminative modeling tasks. This will give us the necessary foundations to go on to tackle generative deep learning in later chapters.