R. Bhalley, *Deep Learning with Swift for TensorFlow*
https://doi.org/10.1007/978-1-4842-6330-3_1

1. Machine Learning Basics

Rahul Bhalley [1]

(1) Ludhiana, India

> *We're unquestionably in the business of forging the gods.* [1]
>
> —*Pamela McCorduck*

Nowadays, artificial intelligence (AI) is one of the most fascinating fields of computer science in addition to quantum computing (Preskill, 2018) and blockchain (Nakamoto, 2008). Hype since the mid-2000s in the industrial community has led to large amounts of investments for AI startups. Globally leading technology companies such as Apple, Google, Amazon, Facebook, and Microsoft, just to name a few, are quickly acquiring talented AI startups from all over the world to accelerate AI research and, in turn, improve their own products.

Consider a portable device like Apple Watch. It uses machine intelligence to analyze your real-time motion sensory data to track your steps, standing hours, swimming reps, sleep time, and more. It also calculates your heart rate from temporal blood color variations underneath your wrist's skin, alerts you about your heartbeat irregularities, performs electrocardiography (ECG), measures oxygen consumption in blood (VO max) during exercise, and much more. On the other hand, devices like iPhone and iPad use the LIDAR information from camera sensors to create depth map of surrounding instantly. This information is then combined with machine intelligence to deliver computational photography features such as bokeh effect with adjustable strength, immersive augmented reality (AR) features such as reflection and lighting of surrounding on AR objects, object occlusions when humans enter in the scene, and much more. Personal voice assistant like Siri understands your speech allowing you to do various tasks such as controlling your home accessories, playing music on HomePod, calling and texting people, and more. The machine intelligence technology becomes possible due to fast graphics processing unit (GPU). Nowadays GPU on portable devices are fast enough to process user's data without having to send it to the cloud servers. This approach helps in keeping the user's data private and hence secure from undesirable exposure and usage (Sharma and Bhalley, 2016). In fact, all the features mentioned above are made available with on-device machine intelligence.

It might surprise you that AI is not a new technology. It actually dates back to the 1940s, and it was not considered useful and cool at all. It had many ups and downs. The AI technology arose to popularity for mainly three times. It

1990s, it was known as “connectionism”; and since 2006, we know AI as “deep learning.”

At some point in the past, there was also a misconception, believed by many researchers, that if all the rules of the way everything in the universe works were programmed in a computing machine, then it would automatically become intelligent. But this idea is strongly challenged by the current state of AI because we now know there are simpler ways to make machines mimic human-like intelligence.

In earlier days of AI research, the data was sparsely available. The computational machines were also slow. These were one of the main factors that drowned the popularity of AI systems. But now we have the Internet, and a very large fraction of the population on Earth interacts with one another which generates humongous amounts of data quickly which are stored in servers of respective companies. (Raina et al., 2009) figured out a way to run the deep learning algorithms with faster speed. The combination of large datasets and high-performance computing (HPC) has led researchers to quickly advance the state-of-the-art deep learning algorithms. And this book is focused on introducing you to these advanced algorithms starting from the simpler concepts.

In this chapter, we will introduce the basic concepts of machine learning which remain valid for its successor, the deep learning field. Chapter 2 focuses on the mathematics required to clearly understand the deep learning algorithms. Because deep learning is an empirical subject, understanding only mathematical equations for deep learning algorithms is of no use if we cannot program them ourselves. Moreover, the computers were built to test theorems of mathematics by performing numerical computation (Turing, 1936). Chapter 3 introduces a powerful, compiled, and fast programming language for deep learning called Swift for TensorFlow which extends Apple’s Swift language (which is already capable of differentiable programming) to include deep learning–specific features with the TensorFlow library. TensorFlow is a deep learning–specific library and deserves the whole Chapter 4 dedicated to its introduction. Then we dive into the basics of neural networks in Chapter 5. Finally, we will program some advanced computer vision algorithms in Chapter 6.

But let us first differentiate between the terms artificial intelligence, machine learning, and deep learning because these are sometimes used interchangeably. Artificial intelligence, also called machine intelligence, represents a set of algorithms which can be used to make machines intelligent. AI systems usually contain hard-coded rules that a program follows to make some sense out of the data (Russell & Norvig, 2002), for instance, finding a noun in a sentence using hard-coded English grammar rules, preventing a robot from falling into a pit using if and else conditions, and so on. These systems are considered weakly intelligent nowadays. Another term is machine learning (ML) which unlike AI algorithms uses data to draw insights from it (Bishop, 2006), for instance, classifying an image using non-parametric algorithms like k-nearest neighbors, classifying a text using decision tree methods, and so on. ML uses data to learn and is also known to perform weaker than deep learning. Finally, the current state-of-the-art AI is deep learning. Deep learning (DL) also uses data for learning but in a hierarchical fashion (LeCun et al., 2015) taking inspiration from the brain. DL algorithms can learn the mapping of very complicated datasets easily without compromising accuracy, but they instead perform better than machine learning algorithms. If you’d draw a Venn diagram, shown in Figure 1-1, you’d see deep learning is a subset of machine learning, whereas the artificial intelligence field is a superset of both these fields.

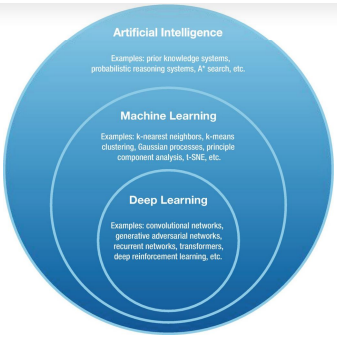

Figure 1-1 A Venn diagram representing the overlap (not precisely scaled) between artificial intelligence, machine learning, and deep learning algorithms. Each set gives a few examples of algorithms belonging to that field.

Now we can begin our journey with deep learning starting with simple machine learning concepts.

1.1 Machine Learning

A machine learning algorithm learns to perform some task by learning itself from the data while improving its performance. The widely accepted definition (Mitchell et al., 1997) of machine learning is as follows: "A computer program is said to learn from experience E with respect to some class of task T and performance measure P if its performance at task T, as measured by P, improves with experience E." The idea is to write a computer program that can update its state via some performance measure to perform the desired task with good performance by experiencing the available data. No human intervention should be required to make this program learn.

Based on this definition, there are three fundamental ideas that help us in making machines learn, namely, experience, task, and performance measure. Each of these ideas is discussed in this section. In Section 1.4, we will see how these ideas are expressed mathematically such that a learning computer program can be written. Following Section 1.4, you will realize that this simple definition forms the basis for how machines learn and that each paradigm of machine learning discussed in the book can be implicitly expressed in terms of this definition.

Before we proceed further, it's important to clarify that a machine learning algorithm is made up of various basic components. Its learning component is called a model which is simply a mathematical function. Now let us move on to understand these fundamental ideas.

1.1.1 EXPERIENCE

The *experience* is multiple observations made by a model to learn to perform a task. These observations are samples from an available dataset. During learning, a model is always required to observe the data.

The data can be in various forms such as image, video, audio, text, tactile, and others. Each *sample*, also known as *example*, from data can be expressed in terms of its *features*. For example, features of an image sample are its pixels where each pixel consists of red, green, and blue color values. The different brightness value of all these colors together represents a single color in the visible range of the spectrum (which our eyes can perceive) of electromagnetic radiations.

In addition to the features, each sample sometimes might also contain a corresponding *label* vector, also known as *target* vector, which represents the class to which the sample belongs. For instance, a fish image sample might have a corresponding label vector that represents a fish. The label is usually expressed in *one-hot encoding* (also called 1-*of*-*k coding* where *k* is the number of classes), a representation where only a single index in a whole vector has a value of one and all others are set to zero. Each index is assumed to represent a certain class, and the index whose value is one is assumed to represent the class to which the sample belongs. For instance, assume the [1 0 0] vector represents a dog, whereas [0 1 0] and [0 0 1] vectors represent a fish and a bird, respectively. This

The features of samples we have listed previously are raw features, that is, these are not handpicked by humans. Sometimes, in machine learning, feature selection plays an important role in the performance of the model. For instance, a high-resolution image will be slower to process than its low-resolution counterpart for a task like face recognition. Because deep learning can work directly on raw data with great performance, we won't discuss feature selection in particular. But we will go through some preprocessing techniques as the need arises in code listings to get the data in correct format. We refer the interested readers to (Theodoridis and Koutroumbas, 2009) textbook to read about feature selection.

In deep learning, we may require to preprocess the data. *Preprocessing* is a sequence of functions applied on raw samples to transform them into a desired specific form. This desired form is usually decided based on the design of the model and the task at hand. For instance, a raw audio waveform sampled at 16 KHz has 16,384 samples per second expressed as a vector. For even a short audio recording, say 5 seconds, this vector's dimension size will become very large, that is, an 81,920 elements long vector! This will take longer to process by our model. This is where preprocessing becomes helpful. We can then preprocess each raw audio waveform sample with the fast Fourier transform (Heideman et al., 1985) function to transform it into a spectrogram representation. Now this image can be processed much faster than the previous lengthy raw audio waveform. There are different ways to preprocess the data, and the choice depends on the model design and the task at hand. We will cover some preprocessing steps in the book for different kinds of data, non-exhaustively, wherever the need occurs.

1.1.2 TASK

The *task* is an act of processing the sample features by the model to return the correct label for the sample. There are fundamentally two tasks for which machine learning models are designed, namely, regression and classification. There are more interesting tasks which we will introduce and program in later chapters and are simply the extension of these two basic tasks.

For instance, for a fish image, the model should return the [0 1 0] vector. Because here the image is being mapped to its label, this task is commonly known as image classification. This serves as a simple example for a classification task.

A good example of a regression task is object detection. We might want to detect the location of an object, say ball, in an image. Here, features are image pixels, and the labels are the coordinates of an object in the image. These coordinates represent a bounding box for the object, that is, the location where the object is present in a given image. Here, our goal is to train a model that takes image features as input and predicts the correct bonding box coordinates for an object. Because the prediction output is real-valued, object detection is considered as a regression task.

1.1.3 PERFORMANCE MEASURE

Once we have designed a model to perform a task, the next step is to make it learn and evaluate its performance on the given task. For evaluation, a performance measure (or metric) of some form is used. A performance metric can take various forms such as accuracy, F1 score, precision and recall, and others, to describe how well the model performs a task. Note that the same performance metric should be used to evaluate the model during both training and testing phases.

As a rule of thumb, one must try to select a single-number performance metric whenever possible. In our previous image classification example, one can easily use accuracy as a performance metric. *Accuracy* is defined as a fraction of the total number of images (or other samples) classified correctly by the model. As shown next, a multiple-number performance metric can also be used, but it makes it harder to decide which model performs best from a set of trained models.

Let us consider two image classifiers C_1 and C_2 whose task is to predict if an image contains a car or not. As shown in Table 1-1, if classifier C_1 has 0.92 accuracy and classifier C_2 has 0.99 accuracy, then it is obvious that C_2 performs better than C_1.

Classifier	Accuracy
C1	92%
C2	99%

Now let us consider precision and recall for these two classifiers which is a two-number evaluation metric. *Precision* and *recall* are defined as a fraction of *all* and *car* images in the test or validation set that the classifier correctly labeled as cars, respectively. For our arbitrary classifiers, these metric values are shown in Table 1-2.

Table 1-2 The precision and recall of classifiers C_1 and C_2 on an image recognition task.

Classifier	Precision	Recall
C1	98%	95%
C2	95%	90%

Now it seems unclear which model has a superior performance. We can instead turn precision and recall into a single-number metric. There are multiple ways to achieve this such as mean or F_1 score. Here, we will find its F_1 score. *F_1 score*, also called *F-measure* and *F-score* , is actually a harmonic mean between precision and recall and is calculated with the following formula:

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \tag{1.1}$$

Table 1-3 shows the F_1 score for each classifier by putting their precision and recall values in Equation 1.1.

From Table 1-3, by simply looking at the F_1 scores, we can easily conclude that classifier C_2 performs better than C_1. In practice, having a single-number metric for evaluation can be extremely helpful in determining the superiority of the trained models and accelerate your research or deployment process.

Table 1-3 The precision, recall, and F_1 score of classifiers C_1 and C_2 on an image recognition task.

Classifier	Precision	Recall	F1 score
C1	98%	85%	91%
C2	95%	90%	92.4%

Having discussed the fundamental ideas of machine learning, we will now shift our focus toward different machine learning paradigms.

1.2 Machine Learning Paradigms

Machine learning is usually classified into four categories based on the kind of dataset experience a model is allowed

reinforcement learning (RL). We briefly discuss each of these machine learning paradigms.

1.2.1 SUPERVISED LEARNING

During training , when a model makes use of labeled data samples for learning to perform a task, then this type of machine learning is known as *supervised learning*. It is called "supervised" because each sample belonging to the dataset has a corresponding label. In supervised learning, during training, the goal of the machine learning model is to map from samples to their corresponding targets. During inference, the supervised model must predict the correct labels for any given samples including samples unseen during training.

We have already gone through an idea of an image classification task previously which is an example of SL. For example, you can search photos by typing the class (or category) of object present in the photo in Apple's Photos app. Another interesting SL task is automatic speech recognition (ASR) where a sequence of audio waveforms is transcribed by the model into a textual sequence representing the words spoken in the audio recording. For instance, Siri, Google Assistant, Cortana, and other personal voice assistants on portable devices all use speech recognition to convert your spoken words into text. At the time of writing, SL is the most successful and widely used machine learning in production .

1.2.2 UNSUPERVISED LEARNING

Unsupervised learning is a type of machine learning where a model is allowed to observe only sample features and not the labels. UL usually aims at learning some useful representation of a dataset in the hidden features of model. This learned representation can later be used to perform any desired task with this model. UL is of great interest to the deep learning community at the time of this writing.

As an example, UL can be used to reduce the dimensionality of high-dimensional data samples which, as we discussed earlier, can help in processing the data samples faster through the model. Another example is density estimation where the goal is to estimate the probability density of a dataset. After density estimation, the model can produce samples similar to those belonging to the dataset it was trained on. UL algorithms can be used to perform various interesting tasks as we shall see later.

It's very important to note that UL is called "unsupervised" because of the fact that labels aren't present in the dataset but we still require labels to be fed to the loss function (which is the fundamental requirement of maximum likelihood estimation discussed in Section 1.3) along with the prediction in order to train the model. In this situation we assume some appropriate labels for the samples ourselves. For example, in generative adversarial networks (Goodfellow et al., 2014), the label for datapoint generated from a generator is given a fake label (or 0), whereas a datapoint sampled from a dataset is given a real label (or 1). Another example is auto-encoder (Vincent et al., 2008) where labels are the corresponding sample images themselves .

1.2.3 SEMI-SUPERVISED LEARNING

Semi-supervised learning is concerned with training a model from a small set of labeled samples and the predictions (using the contemporarily semi-trained model) for unlabeled samples as *pseudo-targets* (also called *soft targets*) during training. From the perspective of the kind of data experienced during training, SSL is halfway between supervised and unsupervised learning because it observes both labeled and unlabeled samples. SSL is particularly useful when we have a large dataset containing only a handful of labeled samples (because they're laborious and hence costly to obtain) and a large number of unlabeled samples. Interestingly, an SSL technique for training the model can considerably boost its performance.

We do not cover semi-supervised learning in the book. For rigorous understanding of semi-supervised learning, we refer the interested readers to (Chapelle et al., 2006) textbook.

1.2.4 REINFORCEMENT LEARNING

Reinforcement learning is based on reward obtained by the agent interacting with the environment to maximize its cumulative reward (weighted average sequence of rewards, also known as return) over a number of trials (called episodes) to achieve its goal. RL is a paradigm of machine learning that involves a sequence of decision-making processes.

The agent acts on the world by taking some action, say it moves forward. Following this, the environment's state gets updated, and a reward is returned (or given) back to the agent by the environment. The reward is either positive or negative and can be, respectively, regarded as a good or bad response to the agent from the world in accordance to the behavioral science viewpoint. We are more interested in return instead of the current step reward because the goal of the agent is to maximize the return over the course of each episode. Here, an episode is a sequence of interactions between an agent and its environment from a start to an end. Examples of an episode are as follows: a gameplay by the agent where the game ends when a certain condition is met and an agent trying to stay alive in harsh environmental conditions until it dies due to some accident. See Figure 1-2 for a diagrammatic view of an interaction between the agent and its environment .

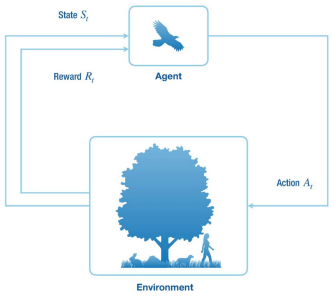

Figure 1-2 An interaction between a reinforcement learning agent and the environment.

The agent perceives the previous state of the environment S_t-1 and takes an action A_t on the environment whose state S_t changes and is returned to the agent. The agent also receives a scalar reward R_t from the environment describing how good is the current state for the agent. Although the state of the environment changes when the agent acts, it may also change by itself. In multi-agent reinforcement learning, there might also be other agents maximizing their own returns .

Reinforcement learning is a very interesting machine learning field and is being studied aggressively at the time of this writing. It is also considered to be more close to the way humans (or other mammals) learn by making behavioral modifications, that is, reinforcing an action based on the reward. A recent work (Minh et al., 2015) showed that a combination of deep learning and reinforcement learning called *deep reinforcement learning* can even surpass human-level gameplay capabilities.

Unfortunately, we don't discuss reinforcement learning in the book. Interested readers may refer (Sutton and Barto, 2018) textbook for the fundamentals of this field. For deep reinforcement learning advances, we suggest the works (Mnih et al., 2015; Schulman et al., 2017).

Now let us look at the basic idea known as maximum likelihood estimation that helps in constructing machine learning algorithms.

1.3 Maximum Likelihood Estimation

The setting described here is assumed throughout the machine learning literature. Constrained with this setting, the parameters of a model are estimated. There are two fundamental approaches to solve the parameters estimation problem, namely, *maximum likelihood estimation* and *Bayesian inference* . We will focus on maximum likelihood estimation because this is what we'll use to train

Chapter 2 of (Bishop, 1995) textbook. And for detailed notes on the origination of the maximum likelihood function, please refer (Akaike, 1973).

To solve a machine learning problem, we require a dataset $\mathbb{D}$ containing a set of N datapoints (or samples), that is, $\mathbb{D}_{\mathcal{X}} = \left\{\left(\mathbf{x}^{(i)}, t^{(i)}\right)\right\}_{i=1}^{N}$. Assume that each datapoint is identical and sampled independently from a joint data-generating distribution $P_d(\mathbf{x}, t)$ which is a probability density function (PDF) meaning that data sample $\mathbf{x}$ and corresponding target t are continuous random variables. Note that the distribution of target random variable t is actually dependent on the task performed by the model, that is, the target's distribution is, respectively, continuous and discrete for regression and classification tasks. We will also call $P_d(\mathbf{x}, t)$ as a *data probability density function* (or data PDF). We only have access to the dataset $\mathbb{D}$ sampled from the data PDF and not the distribution itself. Therefore, we cannot access more datapoints than what are available to us.

Because the dataset $\mathbb{D}$ is sampled from the data PDF, it statistically describes the data PDF itself. Our goal in parametric machine learning is to approximate the mapping of the data PDF by estimating the parameters of a parameterized probability density function $P_m(\mathbf{x}, y|\theta)$ (or simply $P_m(\mathbf{x}, y)$) of our own choice which we will call a *model probability density function* (or model PDF), and we will also frequently refer to it simply as a *model* throughout the book. Here, θ represents parameters, and the random variables $\mathbf{x}$ and $\mathbf{y}$ are the data sample and the corresponding predicted value given $\mathbf{x}$ as input. We will actually minimize the distance, also called loss or error, between prediction variable $\mathbf{y}$ and target variable $\mathbf{t}$ by updating the parameter values using the maximum likelihood function.

The model and data PDFs are completely different distributions such that their samples are also statistically different. But we want the predictions $\mathbf{y}$ from the model PDF to resemble corresponding targets $\mathbf{t}$ from the data PDF for a given data sample $\mathbf{x}$. As pointed out earlier, we don't have access to the parameters of the data PDF so we cannot simply copy its parameter values to those of the model function. But we do have dataset $\mathcal{X}$ at our disposal which we can exploit to approximate the data PDF because it's its statistical description.

Now we will describe *maximum likelihood estimation* to approximate the data PDF with our parameterized model PDF. Because the datapoints in $\mathcal{X}$ are independent and identically distributed, their joint probability is given by

$$P_m(\mathcal{X}|\theta) = \prod_{i=1}^{N} P_m\left(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}|\theta\right) = \mathcal{L}(\theta|\mathcal{X}) \tag{1.2}$$

Here, $P_m(\mathbb{D}|\theta)$ is the conditional model PDF and is read as "joint probability of a dataset given the parameters." The function $\mathcal{L}(\theta|\mathbb{D})$ is a *likelihood function* of parameters θ for a given fixed and finite dataset $\mathbb{D}$. You may also interpret this function as "likely a good approximation of the data PDF from which the dataset $\mathbb{D}$ is sampled." To reduce the clutter, we will implicitly assume the parameters conditioning in both the model and log-likelihood function. We will use the Bayes theorem to convert joint data probability distribution into a conditional distribution we care about.

$$\mathcal{L} = \prod_{i=1}^{N} P_m\left(\mathbf{x}^{(i)}, \mathbf{t}^{(i)}\right) = \prod_{i=1}^{N} P_m\left(\mathbf{t}^{(i)}|\mathbf{x}^{(i)}\right) P_m\left(\mathbf{x}^{(i)}\right) \tag{1.3}$$

The data PDF is approximated by maximizing the likelihood function which requires updating the parameter values of the model PDF. More specifically, the parameters are updated by an iterative method as briefly described in Section 1.4 and at great length in Section 5.1. Numerically, it is more convenient to first take the negative logarithm of the likelihood and then minimize it. This is equivalent to maximizing the likelihood function.

$$L = -\ln \mathcal{L} = -\sum_{i=1}^{N} \ln P_m\left(\mathbf{t}^{(i)}|\mathbf{x}^{(i)}\right) - \sum_{i=1}^{N} \ln P_m\left(\mathbf{x}^{(i)}\right) \tag{1.4}$$

The logarithm function plays a very important role here. It turns the products into summations which helps in stabilizing the numerical computation. This is because the products of values closer to zero are much smaller values which may get rounded off due to limited-precision representational capacity of computational devices. This is *why machine learning frequently uses the logarithm function.*

minimize the loss function by updating its parameter values such that our parameterized model PDF approximates the data PDF using the available fixed and finite data samples from the data PDF. Note that the second term in this equation doesn't contribute in the parameters estimation of the model PDF because it doesn't depend on the model's parameters and is simply a negative additive term which can be ignored for maximizing the likelihood function. The loss function can now be simply described by the following equation:

$$L = -\sum_{i=1}^{N} \ln P_m\left(\mathbf{t}^{(i)} \middle| \mathbf{x}^{(i)}\right) \tag{1.5}$$

Minimizing the loss function is equivalent to minimizing the negative log-likelihood function which further can be considered as maximizing the log-likelihood function, hence the term *maximum likelihood estimation*. Here, our model PDF represents a conditional distribution of targets given the data samples $P_m(\mathbf{t}|\mathbf{x})$. We shall see later that a neural network is a framework for modeling this conditional distribution. And also based on the distribution of the target random variable, different loss functions arise using this equation.

As a sidenote, hypothetically speaking, if the likelihood function perfectly approximates the data PDF, then the available dataset $\mathcal{X}$ and other identical datasets can be sampled from it. But in practice, it is not possible to perfectly approximate the data PDF, but we can instead closely approximate it. This is all what we do in machine learning.

Now let us look at the elements of a machine learning algorithm and make the vague definition of machine learning given by (Mitchell et al., 1997) mathematically concrete.

1.4 Elements of a Machine Learning Algorithm

In this section, we describe the fundamental elements of a machine learning algorithm that apply to all paradigms of machine learning briefly discussed in Section 1.2. There are four crucial components of a machine learning algorithm, namely, data, model, loss function, and optimization. Optionally, one can also use regularization techniques to improve the generalization of the model and balance the bias and variance trade-off (see Section 1.5).

Remember that in machine learning, our primary goal is to train a model that should perform well on unseen data because our training dataset will not contain the data generated by users in the future.

1.4.1 DATA

The data present in the dataset serves as an experience for the machine learning model. When the model is first initialized with random parameter values, it has no knowledge of how to perform well on the certain task. This knowledge is gained by iteratively exposing the model to data (usually in small sample counts, also known as *mini-batches*). As the model experiences more samples, it gradually learns to perform the task more accurately.

A *dataset* is simply a structured, usually a tabular (or matrix), arrangement of datapoints. Table 1-4 shows an arbitrary example of a dataset. This dataset contains some characteristics (or features) of people where each row contains the features for a single person. Each row contains the height (in centimeters), age (in years), and weight (in kilograms) values for a certain person. Note that features and their corresponding targets can be tensor values of any dimensions (0 dimensions, i.e., scalar variables, here).

Table 1-4 Dataset of people with their heights, ages, and weights represented by x_1, x_2, and t scalars, respectively.

Height (x_1)	Age (x_2)	Weight (t)
151.7	25	47.8

Height (x_1)	Age (x_2)	Weight (t)
139.7	20	36.4
136.5	18	31.8
156.8	28	53.0

Given the dataset, we must decide the features and targets for a task. In other words, selection of the correct features for a task is dependent solely upon our decision. For instance, we can use this dataset to perform a regression task of predicting the weight given the height and age of a person. In this setting, our *feature vector* $\mathbf{x}$ contains height $x1$ and age $x2$, that is, $\mathbf{x} = [x1\ x2]$, for each person (called sample), whereas the target t is the weight of a person. For example, $\mathbf{x} = [136.5\ 18]$ and $t = 31.8$ are the feature vector and target scalar for the third person in the given dataset.

In machine learning literature, datapoint input to the model is also known as *example* or *sample*, and its features are also known as *attributes*. For instance, features of a car can be color, tire size, top speed, and so on. Another example can be a drug whose features may include chemical formula, proportion of each chemical element, and so on. The target, also known as *label* or *hard target* (when differentiating from soft target), is the desired output corresponding to a given sample. For instance, given the features of an image (pixel values), the target can be a chair, table, and so on, represented as a vector. Also note that all targets always remain the same for a given sample in a dataset. In rare cases, we generate *soft targets* for unlabeled samples in semi-supervised learning, whereas the targets are *always* assumed immutable in supervised learning.

1.4.1.1 Design Matrix

The most popular way of representing a dataset is design matrix as described in Table 1-4. *Design matrix* is a collection of samples and targets where each row contains a single sample (and each of its columns contains a feature describing it) and its corresponding target. The target values are used for supervised learning tasks. It is *only* the sample that contains the features and not the target. The target is the desired output we expect for our model to predict for a given sample's features.

1.4.1.2 Independent and Identical Samples

A good dataset should statistically represent the overall distribution of samples in a data-generating probability density function. Remember that we never have access to this PDF but the dataset only. Since a dataset statistically represents a probability distribution, there are always some patterns in the way samples are distributed. And the machine learning algorithms aim to discover such patterns in datasets to perform the desired tasks. We can only intuitively judge the patterns easily in low dimensions (at most three dimensions) by staring at a set of samples, but machine learning algorithms can understand the patterns in higher dimensions (even billions). For instance, we can say "If the height of a person lies in range [110,160], then their weight will lie in range [30,80]" and so on. But if we add features such as workout time, nutrition intake, and so on, then the sample has much more complex detailed patterns which can become difficult for us to process collectively, but the machine learning algorithm can analyze it for us and even more accurately.

For any machine learning algorithm to perform well on the task, we need two constraints on the dataset for discovering patterns between samples. First, the value of each sample should be independent of another sample, that is, samples should not be correlated, although the individual features in each sample should be correlated.

That being said, an imaginary dataset of photographs of nature conforming to such constraints might contain images of rivers, vehicles, skies, underwater lives, plants, humans, animals, and so on, where each sample is a photograph and its features are pixel values as a combination of red, green, and blue colors. This suffices the identical requirement because, for instance, the landscape image will always contain the sky and the ground. Furthermore, this nature dataset also conforms to the independent constraint because none of the images affect the other image's feature (pixel) values. One great example of such dataset is ImageNet (Russakovsky, et al., 2015).

1.4.1.3 Dataset Splits

Now that we know what a dataset is and how it is represented, the next step is to use it to learn its hierarchical representation in a model. But before model training, the dataset must be split into multiple subsets. It is a very important step to follow before training a machine learning model for reasons discussed in the following.

In practice, the model typically has a large number of parameters and easily overfits the whole dataset. By *overfitting* , we mean that the model performs very well on the dataset but performs poorly on the unseen examples which it will certainly encounter in the real world. Another term is *underfitting* which means that the model doesn't perform well on either the dataset it was trained on or the unseen dataset. Balancing between both underfitting and overfitting is a problem that plagues the whole machine learning landscape and is also known as the *bias and variance trade-off* (see Section 1.5).

We split our whole dataset into mainly three subsets, namely, training set, test set, and validation set. If the complete dataset contains examples which are i.i.d, then each subset will contain i.i.d examples also. Each of these subsets plays a crucial role in building a good machine learning model and are explained as follows.

The *training set* contains the examples and targets used to tune the model's learnable parameters. The model is trained to minimize the error between the predicted output and the desired target value for a given input features sample, iteratively. As a prerequisite for the model to perform well on unseen examples, it should initially perform well on the training set which it is allowed to experience during the training process.

The *test set* contains the examples which the model is not allowed to experience during the training process. It is *only* used for testing the performance of the model. In the real world, the test set is curated to contain examples from the dataset which are usually difficult to perform well on. This is done so as to choose a better machine learning model. The test set might also contain examples which are likely to be experienced by the machine learning model in reality. We are usually more concerned with the real-world data we might expect from users, for instance, photographs taken by a smartphone than the cartoon character images. So our test set should contain smartphone-clicked photographs. The end result we want is our model to produce good predictions on unseen examples that are closer to true targets. If a given model performs well on unseen examples, it is said to have a good *generalization* property – otherwise, bad.

The *validation set* , also known as the *development set* , is used to select over a set of possible hyper-parameter configurations, model architectures, and so on for a machine learning algorithm. Unlike the test set, the validation set is used during the training process to evaluate the model's performance. But reusing the same validation set can lead the machine learning algorithm to overfit it. To overcome this problem, researchers sometimes use multiple cross-validation sets. Then the machine learning algorithm is evaluated on multiple validation sets one by one to evaluate its accuracy on unseen examples.

As a rule of thumb, the dataset should be split into 70% training set and 30% test and validation sets. But if the dataset has a large number of samples, then you might not require to follow this criteria. The approach should be to split the dataset such that it represents a good estimate of misclassified examples in terms of accuracy. For more details, we refer the reader to (Ng, 2018) for guidelines on

whereas in this book our aim is to introduce you to advanced deep learning algorithms and program them.

1.4.2 MODELS

We discussed about the dataset in the previous section. The goal is to utilize the dataset in a machine learning algorithm that learns to make predictions on the unseen samples. To accomplish this, we need a machine learning model.

In machine learning, the *model* is a mathematical function with learnable coefficients. In this context, the coefficients of the function are termed as *parameters*. Before training, the machine learning model is initialized with small random parameter values. These parameters are slowly changed during the *optimization phase* , also known as the *training phase* . The aim is to have a machine learning model that performs well on the unseen examples. During the *inference phase* , the machine learning model is used in real-world applications where it encounters the examples not seen during the training phase.

There are two kinds of models that are widely used to approximate the data probability density function, namely, parametric and non-parametric models. We briefly discuss each of these models in the following text.

1.4.2.1 Non-parametric Models

Non-parametric models use a whole dataset to predict the label for a given test sample's features. They use a kernel function that measures the similarity between samples. This function is iteratively called for all training examples and a test example. The choice of kernel function varies between different kernel methods. For instance, the radial basis kernel method uses the radial basis function; k-nearest neighbor regression and classification methods use functions such as Manhattan, Euclidean distance, and others. The kernel function can also be easily extended to give rise to a neural network.

Deeper explanation of non-parametric models lies outside the scope of this book. We refer you to (Bishop, 2006) and (Murphy, 2012) textbooks in this literature.

1.4.2.2 Parametric Models

A *parametric model* is a function that contains tunable coefficients (also known as parameters). The parametric model learns by updating its parameter values to perform a task. Unlike non-parametric models, which always use the whole dataset to make predictions, the parametric model learns the representation of a dataset once and makes prediction using the knowledge present in its parameters. It is difficult to interpret the knowledge present in parametric models and this is an active area of research. Please refer (Carter et al., 2019; Olah et al., 2017, 2018) for in-depth visualization of neural network's features. Non-parametric models merge the training and testing concepts. The parametric models are sometimes slow to train but fast for inference, that is, they are good candidates for real-time deployment for users of either on-device or on-cloud services.

The main focus of this book are parametric models introduced in Chapter 5 . In Section 1.1, we didn't discuss how a model learns. In the following text, we will explain the learning (or training) process.

1.4.3 LOSS FUNCTION

The *loss function* computes the distance between prediction and target values. It measures how good the model is at predicting the correct output for a given input by finding the distance between prediction and target values where the notion of distance is defined by various loss functions (see Section 5.5) based on the prediction distribution. Note that the loss function is also called the *error function* , *cost function* , and *objective function* in other textbooks. In this book, we prefer the term loss function.

During training, the loss function guides the learnable parameters of the model toward the values such that the model predicts the desired outputs for the corresponding inputs. Note that the loss function is not to be used as a

For a regression task, the most common choice of loss function is sum of squared errors defined by the following equation:

$$L(\chi;\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2. \tag{1.6}$$

where $L(\mathcal{X};\theta)$ is the loss function, $y^{(i)}$ is the predicted value for an input sample $x^{(i)}$, $t^{(i)}$ is its corresponding target value, and θ are learnable parameters. In total, there are N number of samples in the dataset. The square in this loss function ensures that the overall loss remains non-negative. The fraction term in the loss function has its own importance in that it simplifies the derivative of the loss function as follows:

$$\nabla_\theta L = \frac{dL}{dy^{(i)}} = y^{(i)} - t^{(i)} \tag{1.7}$$

Note that, once our model is trained, we denote the learnable parameters as θ^* and the loss function as $L(\mathcal{X};\theta^*)$ or simply $L(\theta^*)$ leaving it abstract for which subset of the dataset is the loss function being used.

Now that we have our dataset for experience and model for learning the task, the last component of a machine learning algorithm is an optimizer (and, optionally, regularizer) which is used to make the model learn from data and is discussed next .

1.4.4 OPTIMIZER

Optimization , also known as *training* , is a process of updating the learnable parameters of a model using the loss function to minimize the error between targets and predictions. During training, we optimize our machine learning model which is a two-step process, namely, calculation of the gradient of the error with respect to each learnable parameter of the model and updating the parameter values.

In the first step, we compute the gradient of the loss function with respect to each learnable parameter of the machine learning model. To accomplish this task, we use an efficient algorithm called *error backpropagation* (Rumelhart et al., 1986) (also known as *backpropagation*, or simply *backprop*). This algorithm is simply a successive application of the chain rule of calculus (see Section 2.3) to compute the gradients. A more general algorithm to compute gradients is automatic differentiation (see Section 3.3) of which error backpropagation is a special case.

In the second step, we update the parameters of the model using the gradient information obtained in the first step of optimization. Since the gradient of the loss function gives a direction in which its output increases the most, we iteratively update the parameters with small steps in the direction negative to the gradient because our goal is to minimize the loss function. The learning algorithm used for this parameter update step of optimization is called *gradient descent* as described by the following equation:

$$\theta_{(\tau+1)} \leftarrow \theta_{(\tau)} - \eta\,\nabla_{\theta(\tau)} L \tag{1.8}$$

Here, τ denotes the time step of the optimization process, $\theta(\tau)$ denotes the value of a parameter at time step τ, and similarly $\theta(\tau+1)$ is for the parameter values at time step $\tau + 1$. The term $\nabla\theta(\tau)L$ denotes the gradient computed in the preceding text with algorithmic differentiation at time step τ with respect to the weight parameter $\theta(\tau)$. Since the gradient values are usually large, we must take small steps with a hyper-parameter, denoted by term η, known as *step size* (or *learning rate*), to minimize the loss function where $\eta \in (0,1]$. To take small steps, the learning rate is multiplied by the negative gradient of the loss function with respect to each parameter of the model. The optimization is a sequential iterative process, and at each step of iteration, the parameter $\theta(\tau)$ is added with a small step update of $-\eta\nabla\theta(\tau)L$ toward lowering the error between predictions and targets.

In a nutshell, a *single training step* involves alternately forward propagating the input features signal and then backward propagating the error signal computed and emitted by the loss function. The backward propagation computes the gradient which is then utilized to update the parameters of the model. This vanilla gradient descent technique might not always give best results. But there exist various modifications to it for more robust optimization as described at length in Section 5.6.

Note that the models can suffer from overfitting and underfitting problems discussed in Section 1.5. The model

are trained on the same dataset. Another solution is to use regularization techniques while training the model as briefly discussed next and at length in Section 5.7.

1.4.5 REGULARIZER

Although regularization maybe considered as an optional element of a machine learning algorithm, it plays an important role in training a much more generalized model. *Regularization* is any modification made to the data, model, loss function, or optimizer so as to reduce the model's generalization error (i.e., the model should have low bias and variance).

Here, we describe the widely used regularization term for the loss function known as L^2 *norm penalty*, or *ridge regression* (Hoerl & Kennard, 1970), which modifies the loss function as follows:

$$L(\mathbf{x};\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2 + \frac{\lambda}{2}\|\theta\|_2^2 \tag{1.9}$$

where $\|\theta\|_2$ computes the L^2 norm of parameters and λ denotes the weighting of the regularization term which is usually set equal to 1. Training the model with this new loss function helps in reducing the generalization error we care about, that is, the model will perform better on unseen samples.

In Section 5.7, we will discuss various other regularization techniques. For now, let us shift our attention from training models to the concept of bias and variance .

1.5 Bias and Variance Trade-Off

Bias and variance are the characteristics of the model performance on different datasets. *Bias* is concerned with model performance on training set, whereas *variance* is concerned with model performance on validation sets. We desire to find a model that has low bias and variance. But, in practice, we usually trade one for the other. Furthermore, bias and variance trade-off not only affects the traditional machine learning methods but plagues the whole machine learning landscape.

Before discussing bias and variance trade-off, let us introduce the term generalization. The model is said to have a good *generalization* property if it performs very well on the unseen datapoints. We aim to find such model in machine learning.

There are mainly three cases related to bias and variance of a model. Although the fourth case (not mentioned but inferable from other three cases) might be desirable, it is *not* possible statistically, even in theory; otherwise, we might never require to train models.

In the first case, when the model performs well on the training set but poorly on multiple validation sets, it is said to have *low bias and high variance*, respectively. This means the model has a sufficient number of parameters to perform well on the training set, but it overfits the training set (because the number of parameters is more than the requirement to discover the underlying data PDF) or *memorized* the mapping of the training set and has not approximated the underlying data-generating PDF. This is because if it has approximated the data distribution, it should also be able to perform well on the validation sets because, like the training set, validation sets also statistically represent the data PDF.

In the second case, when the model performs well on both training and multiple validation sets, it is said to have *low bias and low variance* , respectively. Here, the model has approximated the underlying data PDF from which all datasets were sampled for both training and testing purposes. In practice, we strive to search for this model.

In the third case, when the model does not perform well on both training and multiple validation sets, it is said to have *high bias and high variance*, respectively. Here, the model does not have a sufficient number of parameters to approximate the training set and, therefore, doesn't also perform well on multiple validation sets. Because the model has low capacity and does not perform well even on the training set, it is said to *underfit*.

1.6 Why Deep Learning?

We discuss various issues with traditional machine learning methods which can be easily tackled with the deep learning approach. This motivates us to study and explore deep learning methods as we shall see in the following.

1.6.1 CURSE OF DIMENSIONALITY

In practice, we usually have samples of dimensions sometimes up to even thousands or even millions. Processing such samples with traditional machine learning methods such as the k-nearest neighbor regressor or classifier, decision trees, and others becomes very inefficient or sometimes even intractable.

Consider a problem of designing a machine learning algorithm that aims to approximate the mapping of an available dataset. Now assume that the available data sample x has a dimension of 1 and is a scalar variable, that is, $x \in \mathbb{R}$. We can start by partitioning the input one-dimensional vector space by a definite integral interval. And doing so, we actually divide the input vector space into M such one-dimensional cells (see Figure 1-3 (a)). We can plot each data sample from the training set on this line. We require to have at least one training point in each cell to make good predictions. Since we have label information for each sample, when we require to predict the label for a new unseen sample, we can simply plot it on this line and assign it to that label which has a maximum count in that cell, for a classification task. In the case of regression, we can take the average of all real-valued labels of training points and assign that value to this test sample.

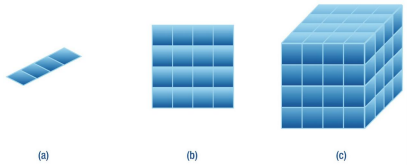

Figure 1-3 The curse of dimensionally for (a) one-, (b) two-, and (c) three-dimensional datapoints. There is an exponential requirement M^d of samples for predicting the label of a new sample where $M = 4$ is the number of sections along each dimension and d is the number of dimensions.

Working in one-dimensional input space seems very easy. But when we jump to higher-dimensional spaces, we start to see the problem arising quickly. Now assume a two-dimensional input vector space for variable $\mathbf{x} \in \mathbb{R}^2$ composed of variables $x1$ and $x2$ representing x and y axes, respectively. Each training point can now be plotted as a point in the matrix. We can again divide each of the x and y axes for $x1$ and $x2$ variables into M sections which gives us M^2 cells (see Figure 1-3 (b)). We can again simply predict the label for this test sample using the process described previously. As we move to a third-dimensional vector space $\mathbf{x} \in \mathbb{R}^3$, where x, y, and z axes are represented by $x1$, $x2$, and $x3$ variables, respectively, then we get M^3 sections (see Figure 1-3 (c)) for which we again need at least one training sample in each cell, that is, at least M^3 labeled samples. Following this trend of increasing dimensions, we can see that the number of labeled samples required in a training set increases exponentially with a linear increase in dimensions. We require a total of M^d samples at least to approximate the dataset mapping where d is the dimension of the sample features space. This exponential relation between dimensions and number of samples required is known as *curse of dimensionality* (Bellman, 2015).

Fortunately, we can tackle the curse of dimensionality with deep neural network models as we shall see later in the book. Now we discuss another problem with traditional machine learning algorithms known as the smoothness assumption.

1.6.2 INVALID SMOOTHNESS ASSUMPTION

This is known as the *smoothness assumption*. This means that for any two similar input values (for instance, similar-looking images), the predicted label should always be same as the true label. Intuitively, this also holds true for our vision system because we can tell the difference when two images are similar or not. This assumption is helpful when we have sparse training data that doesn't fill up all the cells; then we can simply interpolate the input value closer to the available training sample and assign the same label to our test sample.

This view has been largely assumed valid for non-linear methods also until the work by (Szegedy et al., 2013). The authors showed that we can intentionally construct input images which look similar but are classified wrongly by deep neural networks. Geometrically, we can say that this assumption is valid but only for linear models, whereas for non-linear models, this doesn't hold true. But since we are interested in processing high-dimensional data because it carries important information and due to the weak representational capacity of traditional machine learning methods, we cannot achieve good accuracy. This motivates us to adopt deep learning methods.

Next, we look at some of the advantages offered by deep learning over traditional machine learning methods.

1.6.3 DEEP LEARNING ADVANTAGES

As we have discussed earlier, traditional machine learning methods have various problems. This hinders us from analyzing high-dimensional data. There are various advantages of deep learning methods over traditional machine learning methods as discussed here.

Unlike traditional machine learning methods, deep learning is known to give highly accurate results, sometimes even surpassing human-level performance on some tasks. The idea of deep learning is loosely inspired by the way the brain's neural system processes raw data directly obtained from our world. Analogous to the fact that our sensory organs like eyes and ears preprocess the data before sending to the brain, we also sometimes preprocess the data before using it with a deep learning model, but this preprocessing uses entirely a different sequence of functions. But note that there are many things we do not know about the brain itself. Deep learning methods use raw data as input samples and process them to learn abstract information in internal hierarchical high-dimensional spaces. We do not need to select important features for the model. Because the selection process is opinionated among humans, some will find a certain set of features important for the task, while others will prefer some other set of features. Deep learning solves this *feature handcrafting problem* by simply processing the raw data itself. Models in deep learning have multiple layers of neuron-like functions where each layer usually has a different dimension. The high-dimensional input enters the model and gets transformed into different dimensions in different layers until the last layer where the prediction output is generated. Assume a task of image classification. Here, the initial layers (closer to the input) might learn fine-scale details from the image dataset like edges, color gradients, and so on, whereas the intermediate layers (known as hidden layers) might learn more coarse-scale details like shapes such as circle, rectangle, and so on; the layers closer to last layer (known as the prediction layer) might learn the features like eyeball, body shape, and so on; and then finally the prediction layer will predict the label for a given image sample. These models in deep learning are known by many names such as deep learning models, artificial neural networks, deep neural networks, or simply neural networks, and the list of names goes on.

Chapter 5 is dedicated to the introduction of neural networks and various successful techniques to train them.

We also notice that traditional methods are sometimes intractable from the random access memory (RAM) and computational speed viewpoint when we have a large number of samples and each of them is representable in large dimensions. Deep learning models exploit the fact that parallel processing is way faster than sequential processing. These models process the features of samples in parallel. Although machine learning was revived in 2006 (Bengio et al., 2007; Hinton et al., 2006; Ranzato et al., 2007), around 2009, it became clear that graphics processing units (GPUs) can speed up deep learning

like Nvidia have put extreme efforts [2] into building faster GPUs with ever-evolving architectural design considerations. Google has also put efforts in building faster parallel processing devices which they call tensor processing units (TPUs) . TPUs are available on the cloud and even edge devices. At the time of this writing, famous platforms as a service (PaaS) such as Google Colaboratory and Kaggle offer free access to GPU and TPU devices on the cloud for deep learning. This has helped in immensely accelerating the deep learning research. All the codes written in this book are executable on the Google Colaboratory platform.

Deep learning methods can be considered as successors to traditional machine learning methods because of their various interesting advantages. Nearly every idea and concept discussed in this chapter for traditional methods (except non-parametric methods) ports without nearly any modifications to the deep learning framework. Fundamentally, the definition remains the same for deep learning: an idea of maximum likelihood estimation is well suited because deep learning models are essentially parametric models, deep learning also requires all the elements of learning algorithms as discussed in Section 1.4, and every type of machine learning can be performed (and much better) with deep learning methods. We will dive deeply into deep learning starting from Chapter 5 with simple neural networks.

1.7 Summary

This chapter introduced the fundamental concepts related to machine learning. We discussed various paradigms of machine learning and introduced the concept of maximum likelihood estimation that helps in training the machine learning models. Then we dived into various fundamental elements of a machine learning algorithm. We also introduced the idea of bias and variance to understand the generalization property of a model. Finally, we shed some light on the advantages of deep learning methods over traditional machine learning methods.

We will study deep learning starting from basic concepts in the following chapters. But before going that far, in the next chapter, we will study various topics from different branches of mathematics which are essential to understand deep learning.

Footnotes

1 Quoted from (McCorduck, 2004).

2 Nvidia Tesla V100 Performance Guide, 2018. Available at www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/v100-application-performance-guide.pdf (http://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/v100-application-performance-guide.pdf)

PREV Front Matter

NEXT 2. Essential Math