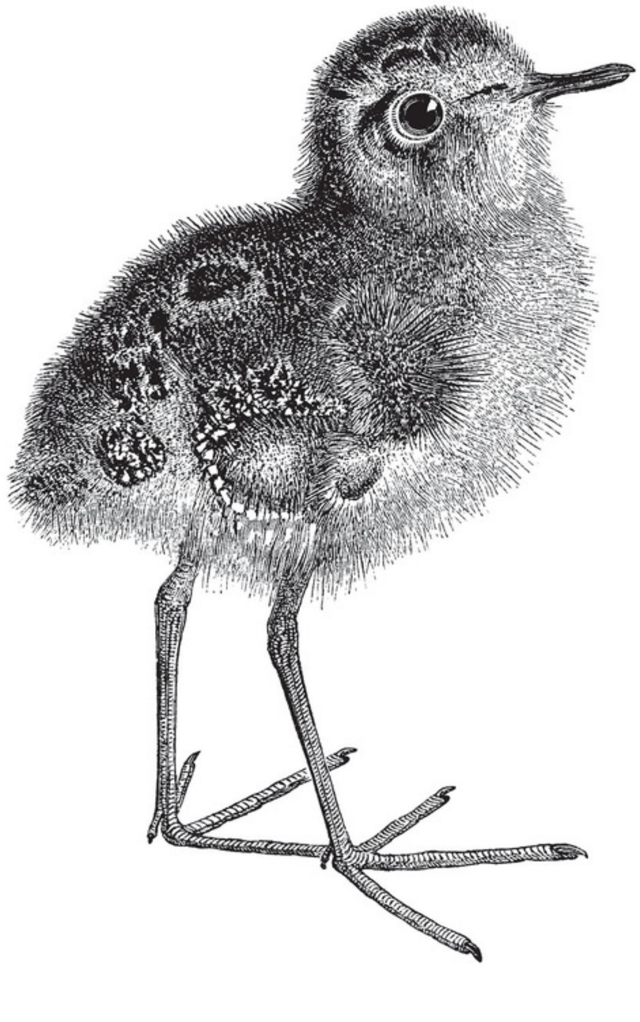
O'REILLY®

Practical AI on the Google Cloud Platform

Learn How to Use the Latest AI Cloud Services on the Google Cloud Platform



Early Release
RAW & UNEDITED

Micheal Lanham

Practical AI on the Google Cloud Platform

Learn How to Use the Latest AI Cloud Services on the Google Cloud Platform

Micheal Lanham

Practical AI on the Google Cloud Platform

by Micheal Lanham

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Jonathan Hassell

Development Editor: Angela Rufino

Production Editor: Deborah Baker

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Rebecca Demarest

December 2020: First Edition

Revision History for the Early Release

- 2020-01-30: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492075813 for release details.

978-1-492-07574-5

[LSI]

Preface

This book was developed for an audience that I regularly see at my local Deep Learning Meetup group. A group of young, very eager and very intelligent people wanting to understand and use the latest AI. Except, their dreams are often crushed or soured when they realize the mathematics or programming they took in school is not that math they need to use for AI. For those that learn the math and programming they then face the roadblock of building real working AI with often limited resources. While many companies see the value in investing in AI the amount they are currently willing to invest is very little. In fact, developing cutting edge AI can be quite expensive computationally and that equals money.

Google likely encountered the same audience I had seen at my Meetup groups. A group of very young and keen newbies eager to learn but missing certain resources when it decided to build its first AI cloud platform. However, what likely solidified that decision was seeing the same mentality expressed in corporations and businesses worldwide. Companies were essentially in the same boat as those trying to learn this new AI tech. By providing an entire AI platform with state of the art models and other training tools in the cloud was a no brainer. And so it was born, Google AI on the GCP.

It of course remains to be seen what the next wave of AI/machine learning will look like. Will it be a set of automation tools that make our life easier or will it be something else? While I agree automation tools like robot cleaners, self-driving cars and so on will be a part of it, it won't be the only part. What I see is the potential for a whole new wave of AI assistant or AI automation assistant apps. These could be smart assistants that do everything from the following:

> Self Help: everything from assistant to maintain physical training to just mangaging your weight, this will likely be a major part of what we use AI for in the future.
>
> Personal Automation: Google has already shown how personal assistants will be able to book your next hair appointment but this could be taken a step further. Perhaps automating a series of tasks to create a vacation or perhaps an adventure vacation.
>
> Financial Management: imagine being able to have Warren Buffet as your personal financial advisor, except instead of him you get an AI assistant that can and will manage all your money for you. That includes setting up and making investments and so on. An AI that would work 24/7 managing your money and making extra money for you won't be a hard app to sell.
>
> Care Assistants: as the world ages the number of physical care assistants to help aging needs to also increase. For the most part this will be physical care but this likely will include more mudane tasks like remembering medication or making appointments. Although, this could progress to real medical diagnosticians that are better able to track your whole medical history through life.

Teachers: perhaps an AI teacher that allowed you to better manage and track your learning goals. This AI agent could progress with you in life and manage your lifetime's education and even accreditations. Image hiring an AI bot that trained you but then also accredited your training to others?

Entertainment: people in the AI community have been writing or trying to write self-aware chat bots for decades. Examples of this have been the focus of several movies including the Oscar winner Her. Her, features a Siri like AI that becomes or is self-aware and falls in love with a human. The AI in Her was developed more as a general Siri like assistant and was not intended as a love bot, at least not portrayed that way in the movie. Except, perhaps the intentention should be to build love or companion AI in a variety of forms. You could have imaginary AI friends for the lonely child or perhaps replicas of old celebrities for those aging and so on. The possibilites are endless in this area.

Protection: imagine if you had an AI that could protect you in multiple ways from guarding your financies to just looking behind your back when you walk home late at night. These could be separate AI or perhaps combined.

Everything else: the list of these types of AI agents could go on and on and it remains to be seen as to what becomes practical in this brand new space.

While the goal of this book will be to teach you how to use the Google AI platform for a variety of reasons it is my hope that you think about how these services could be used to power the above type of AI assistants.

Of course, you may not be ready to pull all the resources together to build an AI assitant. Therefore, I have also made this book accessible for you by showing the base case for using the service or system and how best to use it. That way you will be able to use the system for your immediate use and then perhaps later move on to building a full working AI assistant. For those of you looking to just build business AI then I have also shown plenty of examples with that in mind.

If you have any comments or suggestions on the content in this book, please email me at cxbxmxcx@gmail.com.

Who Should Read this Book

You will enjoy this book if your eager to learn and enjoy working with or seeing plenty coding examples. I have tried to make this book accessible by minimizing the discussion of mathematics. However, you may find yourself needing some math refresher or tutorials to grasp certain core concepts. You should also be patient as training models can and will take time, so patience is a must. As well, you need to understand you will most certainly fail at what seems the most rudimentary of tasks. The important thing to remember is that AI is just hard, it takes thought and understanding, you just need more time understanding.

Why I Wrote this Book

I wanted to write this book because Google had developed all this cool AI no one seemed to be using. Then I realized I wanted to write this book to help those looking for their place in AI. That sounds really cheesy until you understand my involvement in AI. I first learned of AI as a young man in the 1980s. It wasn't until the later 1990s I took a serious look back at advanced neural networks only to again be disppointed by their lack of progress. Instead, I relied on some traditional data science and later ventured into various genetic learning strategies. All along the way over many years returning to deep learning and AI only to be disappointed time and again, until only recently.

It was through a recent project only a few short years ago that I took a look back at deep learning and realized how far it had come. However, unike many, I had a broad base of understanding of the concepts and mathematcis. Being exposed to deep learning many many years previously and keeping up with research gave me a significant advantage. Which allowed me to pick up the technology quicker than most. Then, when I sought the help of others it occured to me that most were eager to learn but were unable to do so because of the frustrations as to what was being taught. From those frustrations I spun out a group of introductory workshops aimed at providing a tutorial on deep learning. Through hosting those sessions and the very positive response I deciced I wanted a bigger voice and writing a book therefore made sense. So a big part of me writing this book is to help others bypass the same frustrations I and others have succumbed to over the years. Hoping that alleviating those frustrations give newcomers more energy to tackle tougher problems more successfully in the future.

Navigating this Book

This book is best read from start to finish but of course you likely are a technical professional that needs to find their way and get answers as quickly as possible. With that in mind use the following summary on where you think is a good place to start or perhaps just refresh your knowledge if you return to the book later as a reference:

> Chapter 1 - Data Science to Deep Learning: this chapter is the best place to start for anyone newish to data science, machine learning and/or deep learning. If you have only a basic understanding of any of those topics you will not want to miss this chapter. You really should only bypass this chapter if you consider yourself a master at deep learning.
>
> Chapter 2 - Google AI on the Cloud: this is where we do a gentle introduction to the available AI services on Google and how those services are structured. From there we will look at how to use Google Colab service to build our first working deep learing network.
>
> Chapter 3 - Image Analysis and Recognition of the Cloud: for this chapter we take a look at how deep learning networks perform image recognitions and the tools they use to do this. We will look at building a similar image classification network then we will move on to using the Google Vision AI service.
>
> Chapter 4 - Video Analysis on the Cloud: video analysis is really an extension of image analysis and in this chapter we look to how Google has developed a service for doing just that. Later, we will look to a variety of examples of using that service.

Chapter 5 - Understanding Language on the Cloud: natural language processing or NLP is one area deep learning systems are making unbelievable progress on. In this chapter we first understand how a deep learing system can process language and then we build a simple language processor.

Chapter 6 - Chatbots and Conversational AI: Google provides a service of the box called Dialogflow that does a variety of language processing tasks. In this chapter we look to use Dialogflow as a chat agent for a variety of tasks.

Chapter 7 - Building Agent Assistants: Google provides a whole SDK for building assistants on its Google Assistant platform. In this chapter we look at how to build an AI assistant with the SDK.

Chapter 8 - Building Advanded Agent Assistants: integrating other AI services with a Google assistant is our ultimate goal. In this chapter we look at how we can integrate the various AI services covered in previous chapters with the GA SDK.

Chapter 9 - Building Integration Cloud AI: not every interface may make sense as a assistant. So in this chapter we look at building a good old fashioned web app with a number of intergrated AI features. We will see how we can build a hosted web server on the GCP and how to intgrate that seemlessly with AI services.

Chapter 10 - What Else Can you Do?: for the last chapter we look at a broader overview of useful applications and services on GCP. These will include such things as Google Maps, BigData, search and recommendation systems to name a few.

A Note on the Google AI Platform

AI is currently progressing at a ferocious rate and many things are quickly getting dated and/or broken. It is quite likely that parts of this book could get broken, but not to fear. Many of the examples in this book are based on very common tasks and in many cases Google may have similar and more up to data examples. Therefore, if at any time through this book you encounter an issue do a quick Google search and you may see the answer that fixes your problem in a number of places.

You will likely find that many of the code examples early on this book use very common blocks of code and this is very much intentional. However, be sure you understand all the details of that code including any minor changes. Many times what appear to be just a minor differnce turns out to be the reason the code was failing. Be sure you pay attention to details and make sure you understand how the code works. Including understanding the inputs and outputs.

Another thing to note about the Google AI platform is that it is a mostly free but also a paid service. I will often recommend areas to be careful in order to avoid costly mistakes. However, it is still up to the reader to be wary of their own cloud usage as well as the security of that usage. Please be aware that costly mistakes can and often do happen because of a lack of awareness. Just be careful when using cloud services.

Things You Need for this Book

Working through the examples in the book will require you to have the following knowledge or resources:

Python: you should have a good grasp of the Python language and how to run scripts on your own.

Desktop Computer: all the examples are provided online and in cloud services but it is still recommended you do this on a wider screen desktop computer for better results. Of course, for those die hards using a phone will always be an option.

Mathematics: for best results you want to have an interest in math. You don't have to be a genius in mathematics but advanced at the high school or post secondary level is recommended.

Fortitude: you need the ability to persevere through extremes and developing AI will certainly challenge that. You will have many ups and downs learning and doing AI. Be prepared to be humble and get tough, you will need it.

Google: googling is a skill apparently not all people have. Marke sure to keep yours current and use Google to enhance your knowledge.

Conventrions Used in this Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This element signifies a tip or suggestion.

NOTE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Practical AI on the Google Cloud Platform* by Micheal Lanham (O'Reilly). Copyright 2020 Micheal Lanham, 978-1-492-07581-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

NOTE

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, conferences, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

> O'Reilly Media, Inc.
>
> 1005 Gravenstein Highway North
>
> Sebastopol, CA 95472
>
> 800-998-9938 (in the United States or Canada)
>
> 707-829-0515 (international or local)
>
> 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://www.oreilly.com/catalog/9781492075813*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I would really like to thank the team at O'Reilly for all their help and support in my journey writing this book. It has been a pleasure to write this book and the team at O'Reilly are certainly a big part of that. In particular many thanks to my Acquisitions Editor, Jonathan Hassell, who was a great help in narrowing the scope of this project and making it more accessible. As well, my Content Development Editors Angela Rufino and Melissa Potter have been a tremendous asset in keeping my content on track and focused.

As always I would like to thank my family for all their help and support. None of this extra curriculur writing would not be possible without their assistance and support. From them grudingly posing for photos or stressfully making graphics they have always helped out where I needed it and I can't thank them enough for that. A special thanks goes to my partner Rhonda, my angel who has taken care of me and is someone I think of while writing almost every word. To my children, all nine of them, I cherish the time I have spent with all of them and each of them has likewise inspired me to write in some way.

A big part of this work is the result of me giving talks for my Deep Learning Meetup in Calgary. It is from that group and through many of those sessions contents in this book were born. The Perceptron Game was developed for and as a result of me teaching free sessions on introductory deep learning for this group. It is from those teachers I feel that have been most valuable as insights in helping me write this group and I would like to thank all those that attend the Calgary Deep Learning Meetup.

Lastly, I want to thank the elephant in the room, Google. Google, on so many levels make our growth in technology and in particular AI possible. I don't think enough credit can and will be given to Google and to a big extent the vast army of people that work there. There are many what you might call AI rockstars working at Google but this acknowledgement is more for those "regular" people pushing their limits to make the world a better place. Of course, there is currently uncertainty if AI will make the world a better place. My hope is that with Google leading the charge we are on the right track.

Chapter 1. Data Science and Deep Learning

Only 20 years after entering the new millennium and man has thrust the bulk of his technological knowledge into the machine age. An age that is suggested to bring more change than our earlier discovery of electricity. A change so massive that it is believed that all life on our planet will be affected in some way, good or bad. Technologists refer to this change or revolution as machine learning or more recently as the dawn of artificial intelligence. While it remains to be seen how intelligent we can make machines, one thing is for sure this new wave of technology is now everywhere. Developers all over the globe are struggling to keep up with the changes and make sense of everything as well as try to benefit from new tools and/or techniques. Fortunately, companies like Google have realized the difficulties and expense of crafting this new powerful AI and are now commercializing powerful AI services on the Cloud. It is the goal of this book to guide the reader through the use of these growing new AI powered cloud services from Google.

NOTE

There are currently a growing list of other Cloud AI providers providing competition for Google and they include Microsoft, Amazon, IBM and an ever growing list of others.

In this chapter we introduce a number of base concepts about machine learning and data science as well as introduce the field of deep learning. Below is a list of topics we will cover in this chapter:

- What is Data Science?
- Classification and Regression
- Data Discovery and Preparation
- The Basics of Deep Learning
- Understanding How Networks Learn
- Building a Deep Learner

What is Data Science?

Data science is the practice of applying statistical methods to data in order to ascertain some further characteristics about said data. This could be for the purpose of predicting some future event or classifying some observation of an event. Anyone who has ever checked the weather for the next day has used data science in the form of predicting the weather. In fact, humans have been intuitively practicing data science for thousands of years and it all started when we learned to predict the weather for tomorrow given the weather from the past.

While we may have been practicing data science in many forms from weather prediction to engineering for thousands of years, it wasn’t until quite recently that the actual field of data science became well known and coveted. This was due primarily to the Big Data revolution which began about 10 years ago. From this spawned a whole broader outlook on computer aided learning about data which collectively became known as machine learning.

Since machine learning originates from the application of data science it only makes sense that they would share a common vocabulary and methodology. As such, we often recommend anyone seriously interested in developing advanced AI tech like deep learning learn some data science. Not only will this help them better grasp the terminology but also understand the origins or purpose for many techniques. We will address the primary topics in this book but it is suggested to the reader to learn more about data science on their own.

TIP

There are plenty of free courses available online. Just use your favorite search engine and search for “data science course free”.

Now that we understand what data science is and how it relates to machine learning and deep learning we will move on to look at how we make sense of data in the next section.

Classification and Regression

Data science has developed many ways of exploring and making sense of data and we often refer to this whole practice as learning. The greater area of machine learning encompasses all forms of learning from deep learning, reinforcement learning, unsupervised, semi-supervised and supervised learning to name just a few. Figure 1-1 shows an overview of the various forms of learning and how they relate with one another.

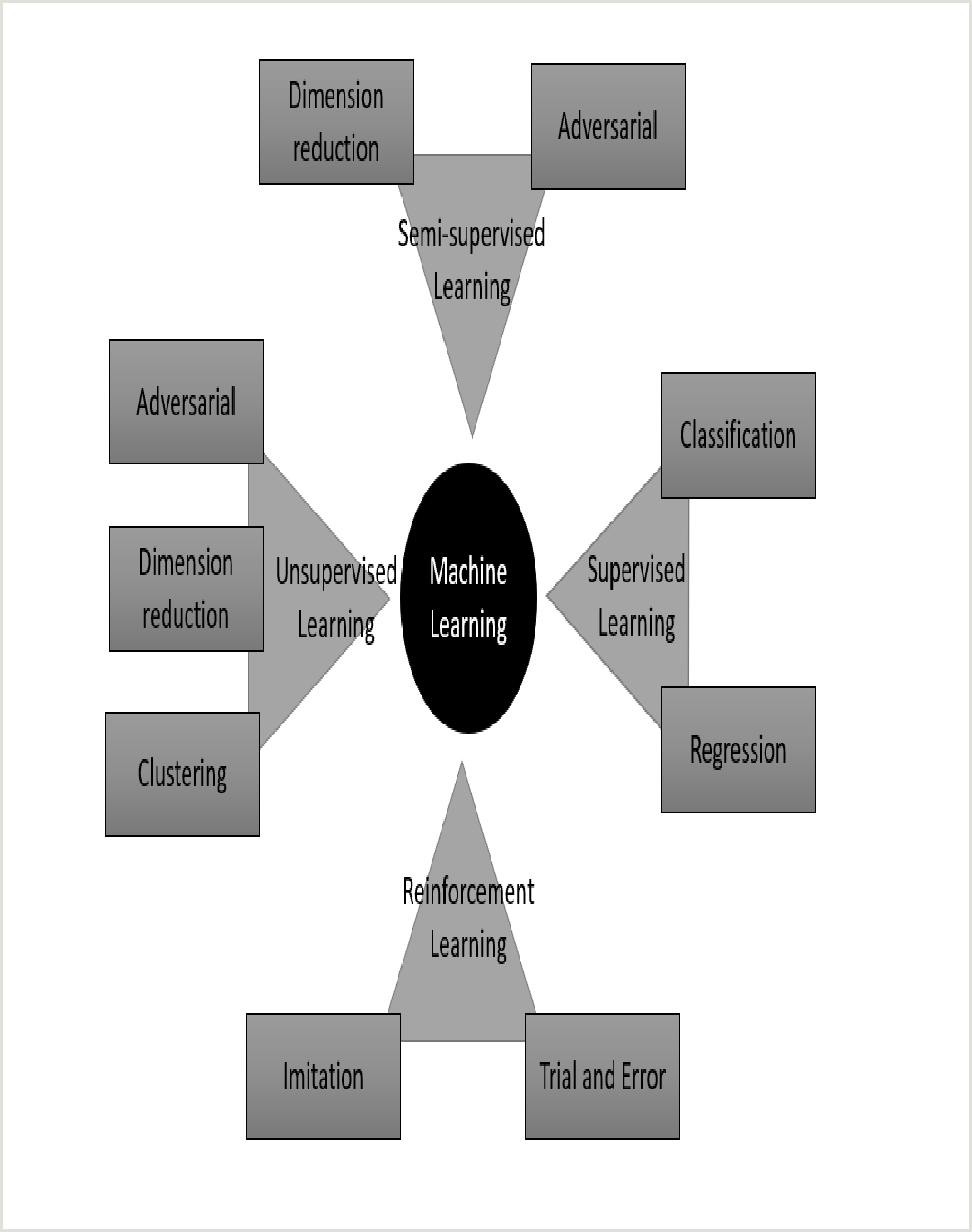


Figure 1-1. Forms of Learning that Encompass Machine Learning

As you can see from Figure 1-1 there is a diverse set of learning methodologies, the triangles, that encompass machine learning as a whole. Within each learning branch is also identified the key problem or task this learning attempts to tackle, shown with rectangles. Each of these sub problems or tasks derives numerous additional applications. We use the term Adversarial for both Semi-supervised and Unsupervised learning to denote the class of algorithms that self learn by training against itself or other similarly matched algorithms. The most famous form of adversarial learner is the GAN or generative adversarial network. We won't have much time to go into detail about the methods of unsupervised, semi-supervised or reinforcement learning in this book. However, after gaining the knowledge in this book and in particular this chapter you may want to explore those forms on your own later. At the middle right of Figure 1-1 is the area of supervised learning and it's various branches. This is the main branch we will focus on in this text and in particular the areas of regression and classification. Supervised learning itself is so named because it requires that the data be first labelled before being fed into the learning algorithm. An example is a data set showing the amount of accumulated rainfall in millimeters (30 millimeters = 1 inch) over the course of 12 months shown in Table 1-1:

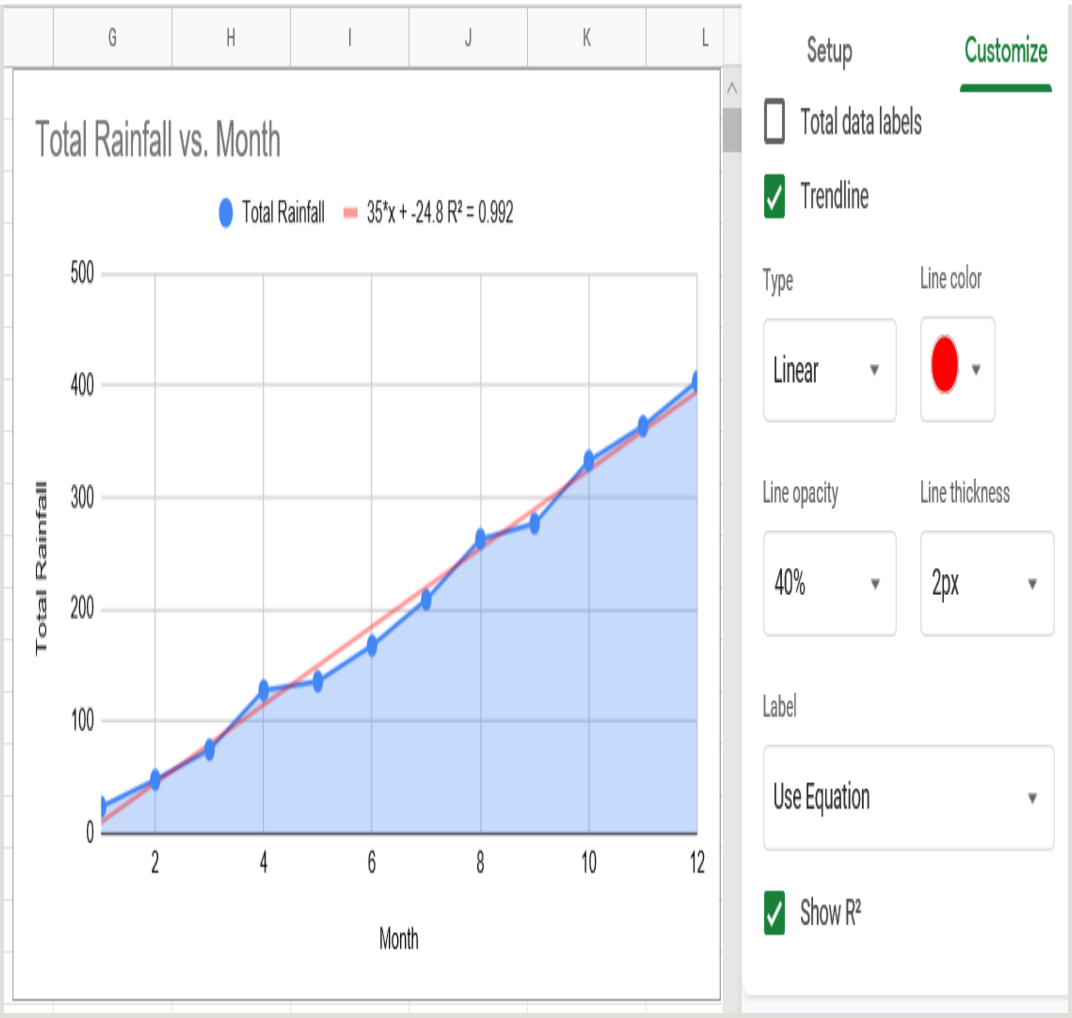
Table 1-1. Mythical Rainfall over Months

Month	Min Rainfall	Max Rainfall	Total Rainfall
1	22	30	24
2	22	25	48
3	25	27	75
4	49	54	128
5	8	8	136
6	29	47	168
7	40	41	209
8	35	63	263
9	14	25	277
10	45	57	333
11	20	39	364
12	39	51	404

The data shows monthly precipitation values from fictional ground stations in a mythical country or location. In order to keep things simple we are going to contrive our data for the first few examples. Over the course of the book though we will look at plenty of real data sets. As we can see the data is labeled with a number of attributes from month, minimum rainfall, maximum rainfall, and total accumulated rainfall. This data will work as an excellent example of labelled data we can use to perform supervised learning of regression and classification later in this chapter. Before that, let us take a close look at what regression is in the next section.

Regression

Regression is the process of finding the relationship between dependent variables and independent variables. The most common form of regression is linear regression. So named because it assumes a linear relationship between variables. Figure 1-2 below is an example of drawing a line of regression through that previous set of weather data shown in Table 1.1. Plotted was the independent variable month against the dependent last column total rainfall. For this simple example we only use 2 variables. The plot was generated with Google Sheets, which provides linear regression as a data analysis tool out of the box using the Trendline option under customize Series.



☐ Total data labels

✓ Trendline

Type: Linear

Line color: ●

Line opacity: 40%

Line thickness: 2px

Label: Use Equation

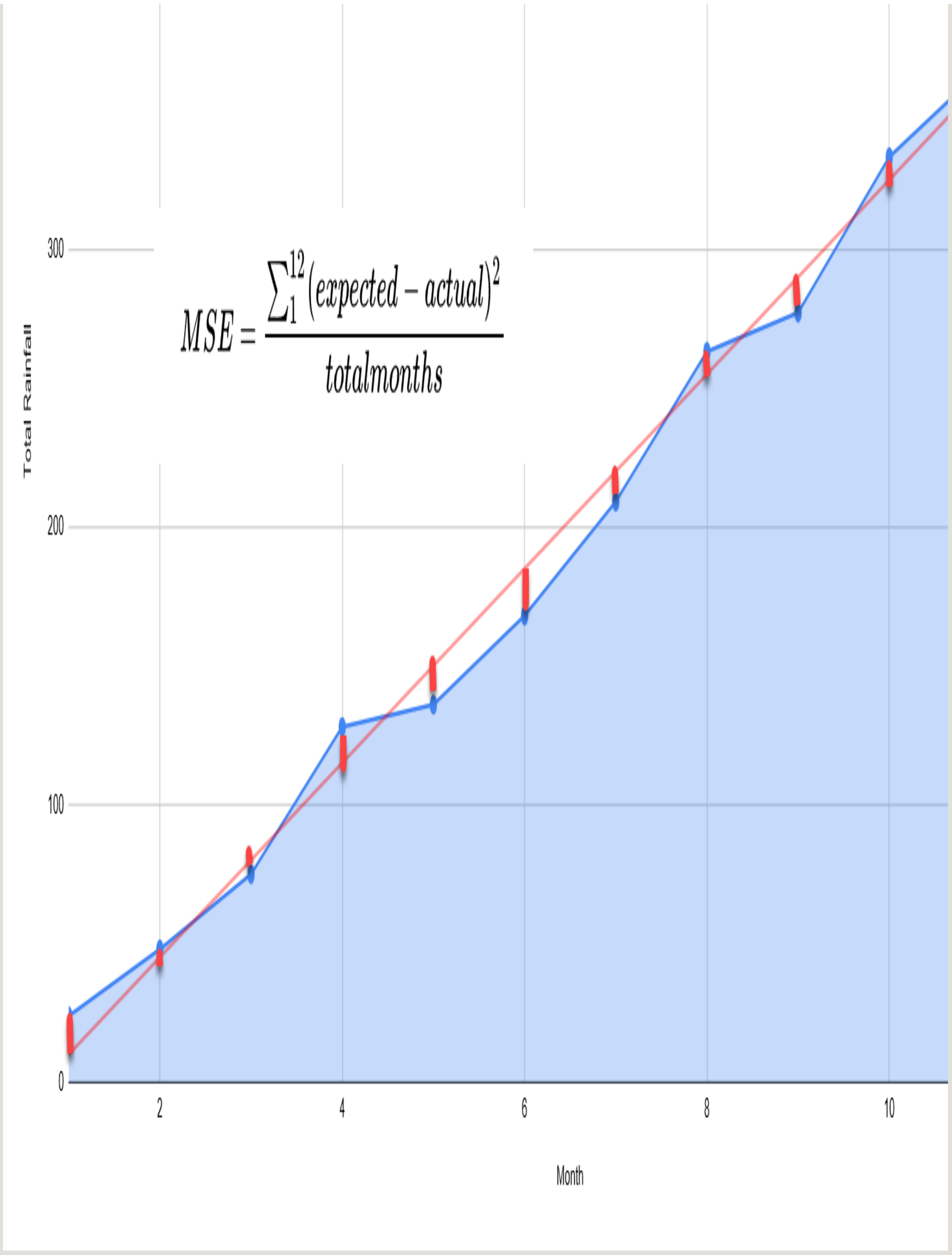
✓ Show R²

Figure 1-2. Linear Regression of Total Rainfall vs. Month

Placing a Trendline is the same as performing regression against our plotted data. In this case we plotted the month number against the accumulated total rainfall for the year. Using an accumulation of rain rather that an actual amount per month likewise simplifies our placement of a trendline. The regression we are using in the example is linear and Google Sheets also allows us to derive and show the equation of line. Check the charts legend and note the equation is in the form y = mx + b or in other words linear. You will also notice in the legend another value called R2 or what we call R squared. R squared is used as a goodness of fit (how well the predicted values from regression match the actual data) and because the value ranges to a maximum of 1.0 it often provides a good baseline measure. However, R squared is not our preferred method for determining goodness of fit and we will talk to better methods in the next section.

Goodness of Fit

The primary problem with R squared is that it actually does not measure goodness of fit. What we find is that the more varied the data, the larger the standard deviation, the lower the values of R squared. Thus, R squared generally indicates lower values over more diverse and larger datasets and this makes it useless in deep learning. Instead we apply an error function against our predicted and actual values taking the difference squaring and then averaging it. The result of this known as the mean or average squared error or MSE. Figure 1-3 shows how we would calculate the mean squared error from our last example. Inside the diagram is the equation which essentially means we take the expected, predicted value with regression, and subtract that from the actual. We square that number to make it positive and then sum all those values. After that we divide by our total samples to get an average amount of error.



$$MSE = \frac{\sum_{1}^{12}(expected - actual)^2}{totalmonths}$$

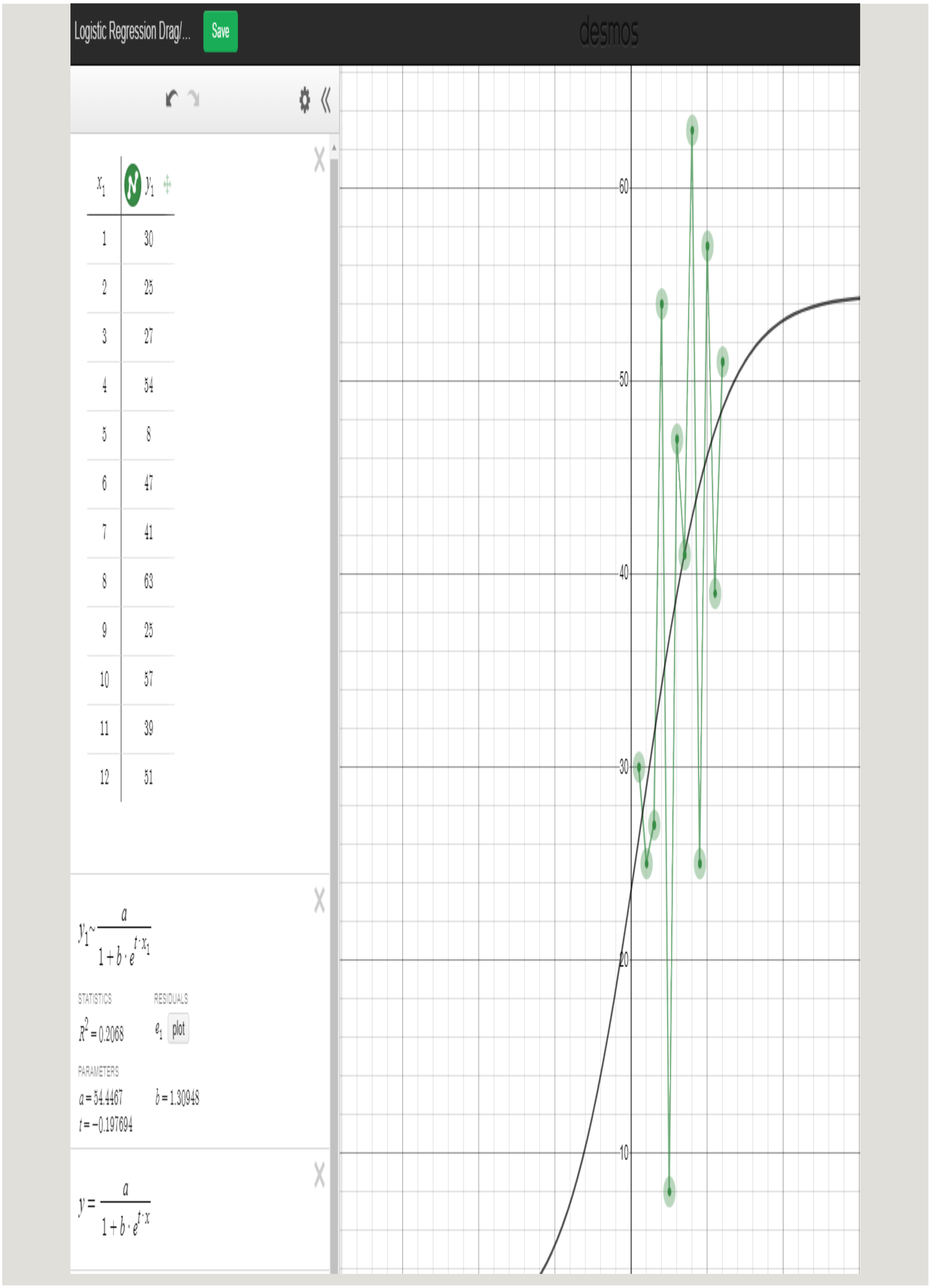
Figure 1-3. Calculating Mean Square Error for Regression

MSE is a relative measure of error and is often specific to your dataset. While MSE does not give us a general quality of fit like R squared it does give us a relative indication of goodness of fit. This means that lower values of MSE indicate a better goodness of fit. There are other similar measures we can use to determine how well a regression model fits to the data. These include Root Mean Squared Error or RMSE which is just the root of MSE; and Mean Absolute Error or MAE measures the independent difference between variables. Determining goodness of fit will ultimately determine the quality of our models and is something we will revisit often through the rest of the book.

In the next section we look at a different form of regression, logistic regression or what we commonly refer to as classification.

Classification with Logistic Regression

Aside from regression the next common problem we will look to solve is classifying data into discrete classes. This process is known as classification but in data science we refer to it as logistic regression. Logistic meaning logit or binary which makes this a binary form of regression. In fact, we often refer to this form of regression as regression with classes or binary regression. So named because the regression model does not predict specific values but rather a class boundary. You can think of this as the equation of regression being the line that separates the classes. An example of how this looks/works is shown in Figure 1-4.



Logistic Regression Drag/... Save

x_1	y_1
1	30
2	25
3	27
4	54
5	8
6	47
7	41
8	63
9	25
10	57
11	39
12	51

$$y_1 \sim \frac{a}{1 + b \cdot e^{t \cdot x_1}}$$

STATISTICS

$R^2 = 0.2068$

RESIDUALS

e_1 plot

PARAMETERS

$a = 54.4467$ $b = 1.30948$

$t = -0.197694$

$$y = \frac{a}{1 + b \cdot e^{t \cdot x}}$$

Figure 1-4. Example of Logistic Regression src: desmos.com

In Figure 1-4 we see our example rainfall data again but this time plotted on month and maximum rainfall for different year. Now the purpose of the plot is to classify the months by rainy (wet) or dry. The equation of regression in the diagram denotes the class break between the 2 class of months, wet or dry. With classification problems our measure of goodness of fit now becomes how well the model predicts an item is in the specific class. Goodness of fit for classification problems use a measure of accuracy or what we denote ACC as accuracy, with a value from 0 to 1.0 or 0% to 100% to denote the certainty/accuracy of data being within the class.

TIP

The source for Figure 1-4 was a free data science learning site called Desmos.com. Desmos is a great site where you can visualize many different machine learning algorithms. It is also highly recommended for anyone wanting to learn more fundamentals about data science and machine learning.

If we refer back to Figure 1-4 it is also worth mentioning that the logistic regression used here is a self-supervised method. That means we didn’t have to label the data to derive the equation but we can also use supervised learning or labelled data, to train classes as well. Table 1.2 shows a sample rainfall dataset with classes defined. A class of 1 indicates a wet month while a class of 0 denotes dry.

Table 1-2. Months Classified by Wet or Dry

Month	Wet/Dry (0 or 1)
1	1
2	0
3	0
4	1
5	0
6	1
7	0
8	1
9	0
10	1
11	0
12	1

It is easy to see from table 1.2 which months break into which classes, wet or dry. However, it is important to note how we define classes. Using a 0 or 1 to denote whether a data item is within a class or not will become a common technique we use later in many classification problems. Since we use accuracy to measure fit with classification it also makes this type of model more intuitive to train. Although, if your background is programming you may also realize that you could also classify our sample data far easier with a simple if statement. While that is true, especially for these simple examples of single dependent variable regression or classification, it is far from the case when we tackle problems with multiple dependent variables. We will cover multi-variable learning in the next section.

Multi-variant Regression and Classification

The example problem we just looked at was intended to be kept simple in order to convey the key concepts. In the real world however, data science and machine learning is far from simple and often needs to tackle far more complex data. In many cases data scientists looks at numerous independent variables or what is referred to as features. A single feature denotes a single independent variable we would use to describe our data. With the previous example we only looked at one independent variable, the month number for both problems of regression and classification. This allowed us to derive a relationship between that month number (a feature) and a dependent variable. For regression we used total monthly rainfall to determine linear relationship. Then for classification we used maximum monthly rainfall to determine the month’s class wet or dry. Except, in the real world we often need to consider multiple features that need to be reduced down to a single value using regression or classification.

NOTE

The data science algorithms we look at here for performing regression and classification were selected because they lead into the deep learning analogs we will look at later. There are numerous other data science methods that perform the same tasks using statistical methods that we will not spend time on this book. Interested readers may wish to explore a course, book or video on data science later.

In the real world data scientists will often deal with datasets that have dozens, hundreds or thousands of features. Dealing with this massive amount of data requires more complex algorithms, but the concepts for regression and classification are still the same. Therefore, we won’t have a need to explore finer details of using these more complex classic statistical methods. As it turns out, deep learning is especially well suited to learning data with multiple features. However, it is still important for us to understand various tips and tricks for exploring and preparing data for learning in the next section.

Data Discovery and Preparation

Machine learning, data science and deep learning models are often very much dependent on the data we have available to train or solve problems. Data itself can represent everything from tabular data, pictures, images, videos, document text, spoken text and computer interfaces as an example. With so much diversity of data it makes it difficult to establish well defined cross cutting rules that we can use for all datasets but in this section we look to a few important considerations you should remember when handling data for machine learning.

Preparing Data

One of the major hurdles data scientists and machine learners face is finding good quality data. There are of course plenty of nice free sample datasets to play with for learning but when it comes to the real world we often need to prepare your own data. It is therefore critical to understand what makes data good or bad.

Bad Data

One characteristic of bad data is that it is duplicated, incomplete or sparse. Meaning, it may have multiple duplicated values or it may be missing values for some or many features. Table 1-3 shows an example of our previous mythical rainfall data now with incomplete or bad data.

Table 1-3. Mythical Rainfall over Months (Missing Data)

Month	Min Rainfall	Max Rainfall	Total Rainfall
1	22	30	24
2	22	25	48
3	25		
4	49	54	128
5	8	8	136
6		47	168
7	40	41	209
8	35		
9	14		277
10	45	57	333
11			
12	39	51	404

Now, if we wanted to perform linear regression on the same dataset we would come across some issues. The primary one being the missing values on labelled dependent variable total rainfall. We could try and replace the missing values with 0 but that would just skew our data. Instead what we can do is just omit the data items with bad data. This reduces the previous dataset to the new values shown in table 1-4.

Table 1-4. Mythical Rainfall over Months (Cleaned Data)

Month	Min Rainfall	Max Rainfall	Total Rainfall
1	22	30	24
2	22	25	48
4	49	54	128
5	8	8	136
7	40	41	209
9	14		277
10	45	57	333
12	39	51	404

Plotting this data in Google Sheets and applying a trendline produces figure 1-5. As you can see in the figure the missing values are not much of an issue. You can clearly see now how removing the null values also shows us how well regression performs. Pay special attention to where the missing months should be and look at how well the trendline or regression equation is predicting these values. In fact, data scientists will often not only remove bad data with missing null values but also good data. The reason they remove good data is in order to validate their answer. For instance, we can go back to our full sample data as shown in table 1-1 and use some of those values in order to validate our regression. Take month 3, the accumulated value is 75. If we consider the predicted value for month 3 from Figure 1-5 we can see the value is predicting around 75. This practice of removing a small set of data for testing and validating your answers is fundamental to data science and something we will cover in the next section.

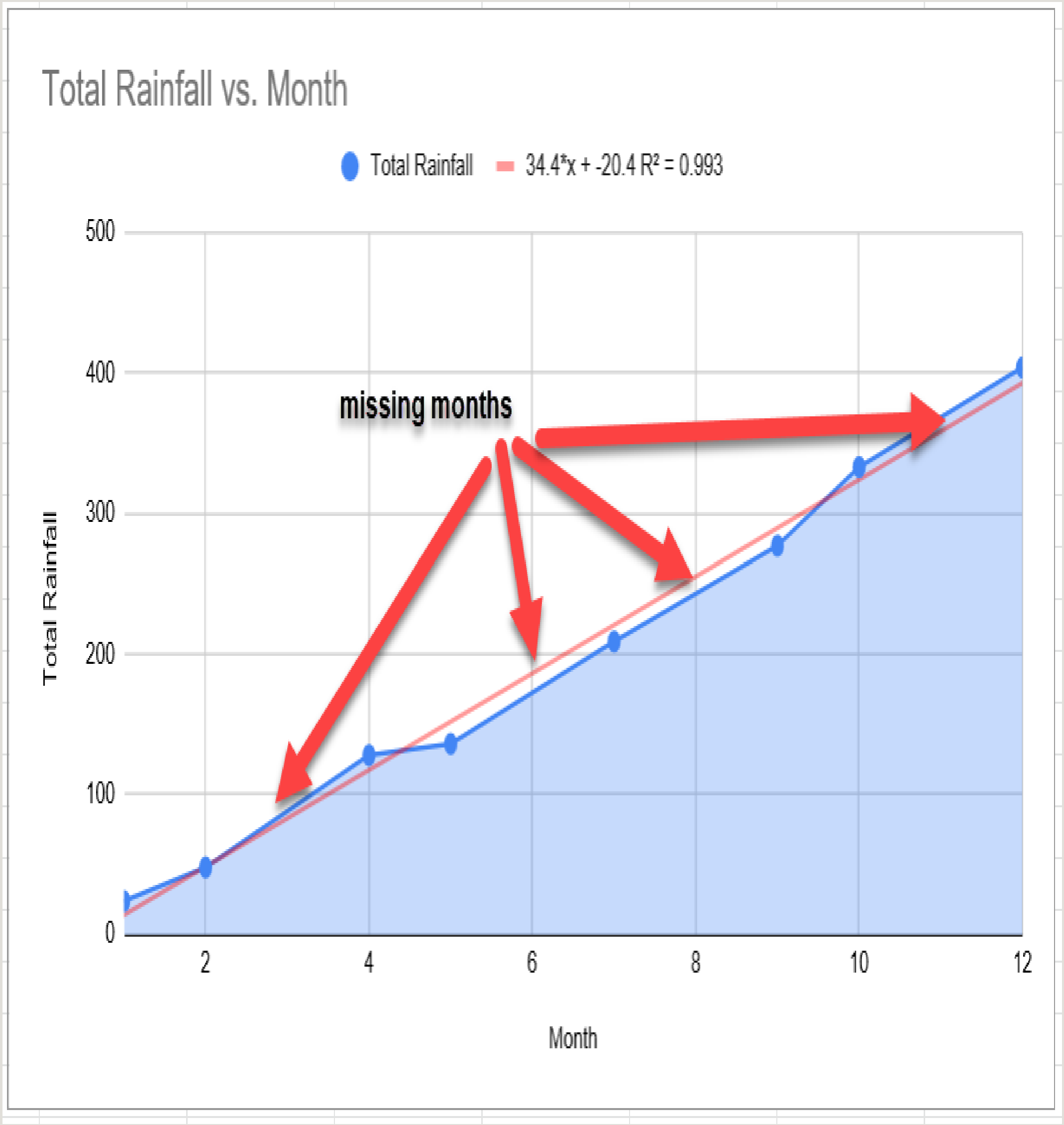


Figure 1-5. Example of Regression with Reduced Data

Training, Test and Validation Data

A fundamental concept in data science is breaking source data into three categories training, test and validation. We then set aside the bulk of the data for training often about 80%. Then, we break the remaining data down into 15% test and 5% validation. You may initially think this could compromise your experiment but as we saw when removing small amounts of data it increased the confidence in our model. Since model confidence is a key criteria for any successful machine learning model removing a small percentage for testing and validation is seen as trivial. As we will see setting aside data for testing and validation will be critical for evaluating our performance and baselining our models.

TIP

Another critical purpose for breaking out data into test and validation is in oder to confirm if the model is not over or under fitting. We will cover the concept of over and under fitting we when get to deep learning later in this chapter.

Good Data

Aside from the obvious missing, duplicate or null features characterizing data as good is subjecting to the machine learning technique. When using all classical data science methods we almost always want to verify the dependency between variables. We do this in order to make sure 2 dependent variables are not strongly dependent on one or the other or both. For instance, in our previous rainfall example the total accumulated rainfall per month would be heavily dependent of the maximum monthly rainfall. Therefore, most classic data science methods would discourage using both variables since they heavily depend on one another. Instead, those methods strongly encourage variable independence but again this is often not the ideal when it comes to real world. This is also where we see the true benefit of deep learning methods. Deep learning has the ability to work with independent, dependent and sparse or missing data well if not better than any other statistical method we have at our disposal. However, there is still some common rules we can use to prepare data for all types of learning in the next section.

Preparing Data

The type of preparation you need to perform on your data is quite dependent on the machine learning algorithm being employed and the type of data itself. Classical statistics based methods like the ones we used for regression and classification earlier often require more data preparation. As we saw, you need to be careful if the data has null, duplicate or missing values and in most cases you either eliminate those records or annotate them in some manner. For example, in our previous rainfall example we could have a used a number of methods to fill in those missing data values. However, when it comes to deep learning, as we will see shortly, we often throw everything at the learning network. In fact, deep learning often uses data sparsity to it's advantage and this strongly goes against most classic data science. If anything, deep learning suffers more from too similar data and duplicated data can be especially problematic. Therefore, when preparing data for deep learning we want to consider some basic rules which our outside the norm for data science:

1. Remove duplicated data - duplicated data is often an issue for deep learning and data science in general. Duplicates provide extra emphasis to the duplicated rows. The one exception to this will be time based data or where duplicate values have meaning.

2. Maintain data sparsity - avoid the temptation to fill in data gaps or remove data records due to missing values. Deep learning networks generalize data better when fed sparse data or when the network itself is made sparse. Making a network layer sparse is called Dropout and is a concept we will cover in later chapters.

3. Keep dependent variables - data scientists will often reduce the number of features in large datasets by removing highly dependent features. We saw this in our rainfall example where the total rainfall in a month was highly dependent on the maximum rainfall in a month. A data scientist would want to remove the dependent feature, where as a deep learner would likely keep it in. The reason for this is that while the feature is observed to be highly dependent it may still have some independent effect.

4. Increase data variability - in most data science problems we often want to constrain data variability in some manner. Reducing data variation allows a model to train quicker and with a better answer. However, the opposite is often the case with deep learning. Where we often want to expose the model to the biggest variation in order to encourage better generalization and avoid false positives. We will explore why this can be an issue in later chapters.

5. Normalize the data - normalizing the data is something we will cover in more detail as we go through the various examples. We do this in order to make features unitless and typically range in value from -1 to +1. In some cases you may

normalize data to 0 to 1. In any case, we will cover normalization when it pertains to the relevant sample later.

Aside from applying the above general rules to your data you will also want to understand what it is you want from your data and what your expectations are. We will cover this in more detail in the next section.

Questioning Your Data

One key observation we need to make with any dataset before applying a data science or machine learning algorithm against is determining how likely the expected answer. For example, if you are training an algorithm to guess the next roll on a 6 sided dice you know that at a minimum an algorithm should guess 1/6th or 1 out of 6 times correctly. Likewise, if your trained algorithm guessed 1 out of 10 times the correct answer on a 6 sided die then this would indicate very poor performance. Since even a random guess is likely 1 out of 6 times correct. Aside from understanding the baseline expectation here are some other helpful questions/rules, again skewed more towards deep learning:

1. Evaluate baseline expectation - determine how likely a random guess is to get the correct answer you are also asking.
2. Evaluate maximum expectation - how likely is it to get the best answer? Are you constraining your search to a very small space, so small that even finding it could be problematic? For example, assume we want to train a network to recognize cats. We feed it one picture of a cat and 10000 pictures of dogs, which we train the network to recognize. In that case our algorithm would have to identify 1 cat out of 10001 pictures correctly. However, with deep learning since our

network was only trained on one cat picture it will only recognize one exact, more or less, cat. The take away here is to make sure the data covers as much variety as possible, the more the better.

3. Evaluate least expectation - opposite of above but conversely how likely is it for your algorithm to get the wrong answer. In other words is the random guess, base expectation, very high to start? If the base expectation is above 50% then you should reconsider your problem in most cases.
4. Annotate the data - are you able to annotate or add to the data in some manner. For instance, if your dataset consists of dog pictures what if you horizontally flipped all pictures and added those. This would in essence duplicate your data and increase your variability. Flipping images and other methods will be explored later in relevant exercises.

Make sure to always review rules 1 to 3 from above. It is important to understand that the questions have the answers in your data and that the answers are obtainable. However, the opposite is also very true and you need to make sure that the answer is not so obvious. Conversely, unsupervised and semi-supervised learning methods are designed to find answers from the data on their own. In any case, when performing regression or classification with supervised learning you will always want to evaluate the expectations from your data.

A common practice now is to construct unsupervised and semi-supervised deep learning networks to extract the relevant features from the data and then train on those new features. These networks are able to learn, on their own, what features have relevancy. The practice is known as autoencoding and is one of the first type of networks we will learn later in this chapter.

The Basics of Deep Learning

Deep learning and the concept of connected learning systems that function similarly to a biological brain have been around since the 1950's. While deep learning is inspired by biology it in no way attempts to model a real biological neuron. In fact, we still understand very little of how we learn and strengthen the connections in any brain; however, we will need to understand in great detail, how the connections strengthen or weaken, how they learn, in the deep learning neural networks we build.

TIP

A fantastic book on the history and revolution of deep learning is called the "The Deep Learning Revolution" by Terrence J. Sejnowski. Sejnowski is considered a founding father of deep learning which make his tales about the history more entertaining.

In early 2020, state of the art deep learning systems can encompass millions of connections. Understanding how to train such megalithic systems is outside the scope of this book, but not using such systems. Google, and others now provide access to such powerful deep learning systems through a cloud interface. These cloud interfaces/services are simple to use, as we will see in later chapters. However, understanding the internals of a deep learning system will make it easier to identify when things go wrong and how to fix them. As well, understanding the simplicity of these system will likely take away any apprehension or intimidation you feel towards deep learning.

Therefore, we will start with the heart of the deep learning network the perceptron.

Central to a deep learning system is the perceptron. You can think of the perceptron as being analogous to the engine in a car. Except in a car there is a single engine in a deep learning system there may be thousands of perceptrons all connected in layers. Figure 1-6 shows a single perceptron with a number of input connections and a single output, controlled by an activation function.

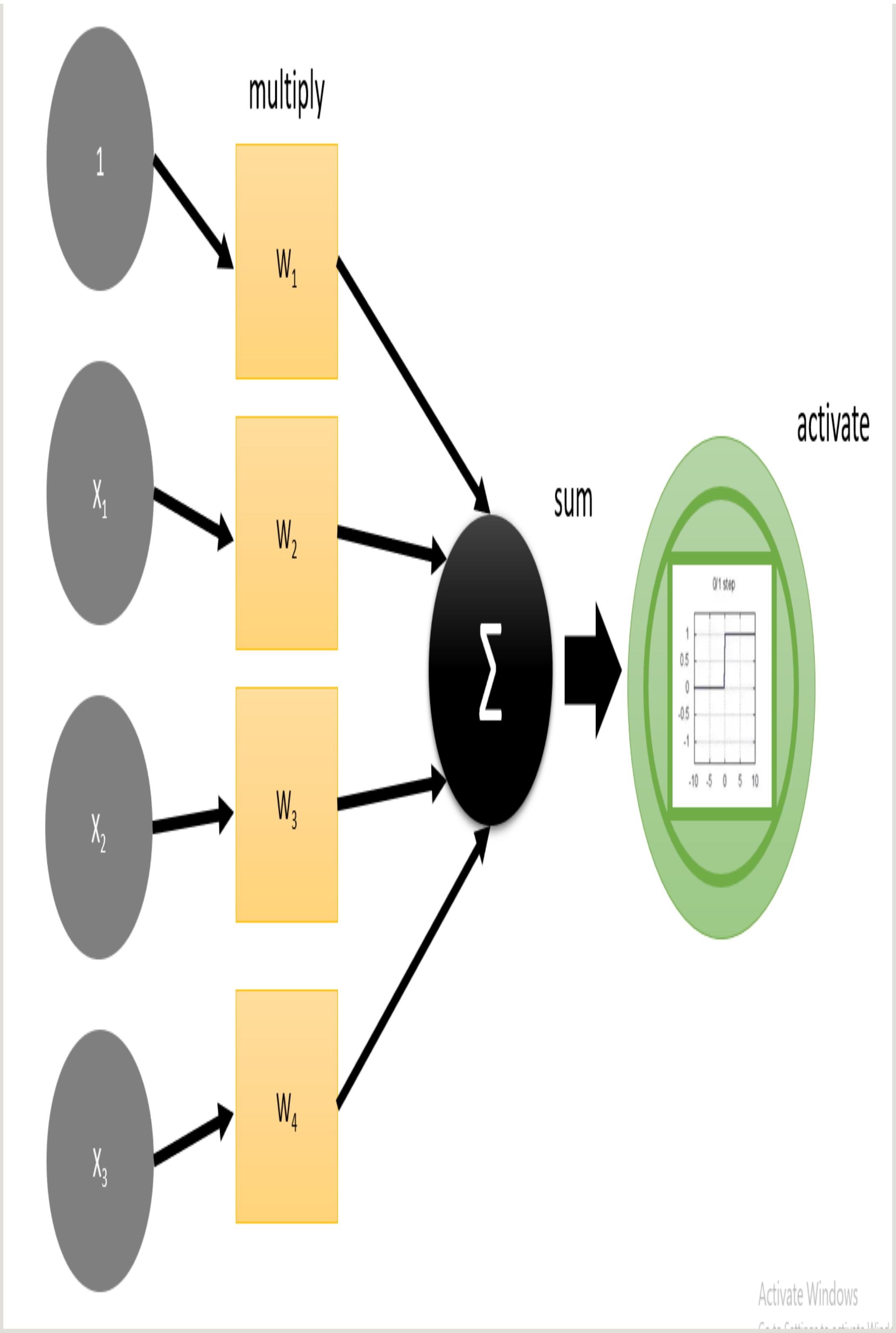


Figure 1-6. A Simple Model of the Perceptron

We can picture all activity flowing through figure 1-6 from the left to the right. Starting at the far left the inputs are labelled X_{1} to X_{3} to show 3 inputs. In a real network the number of inputs could be in the thousands or millions. Moving from the inputs, we then multiply by a weight for each input denoted W_{1} to W_{4}. The weights represent how strong the connection is, thus a higher value weight will have a stronger connection. Notice that we multiply the first weight by one, this is for the bias. After all the weights are multiplied by the inputs they are summed at the next step denoted by greek symbol \sum for summation. Finally, the total sum is pushed through an activation function and the result is output, the result dependent on the activation function. It may help to think that the activation functions controls how the perceptron fires and passes it's output along. This whole process is called a forward pass through the network and is called inference or how the network answers.

Generally, we will try and minimize the use of math in order to explain concepts in this book. However, math is a core element of this technology and it is sometimes easier and more relevant to express concepts in terms of math equations. Therefore we will start by showing how a perceptron fires mathematically as shown in Equation 1-1.

Equation 1-1. Equation 1-1

$$y = W_1 + \sum_{i=1}^{n} x_i * W_{i+1}$$

Where: y = output sent to activation function W = a weight x = an input

Equation 1-1 shows the summation part of the forward pass through a single perceptron. This is just where the weights are multiplied by the inputs and everything is added up. It can also be helpful to view how this looks in code. Listing 1-1 shows a function written in Python that performs the summation step in a perceptron.

Example 1-1. Syntax highlighting sample

```
def summation(inputs, weights):
        sum = weights[0]
        for i in range(len(inputs)-1):
                sum += weights[i + 1] * inputs[i]
        return sum
```

After summation the result is passed into an activation function. Activation functions are critical to deep learning and these functions control the perceptrons output. In the single perceptron example an activation function is less critical and in this simple example we will just use a linear function, shown in Listing 1-2. This is the simplest function as it just returns a straight mapping of the result to the output.

Example 1-2. Syntax highlighting sample

```
def act_linear(sum):
        return sum
```

Listing 1-3 shows an example of a step activation function. So named because the output steps to a value when the threshold is reached. In the listing the threshold >= 0.0 and the stepped output is 1.0. Thus, when a summed output is greater than or equal to zero than the perceptron outputs 1.0.

Example 1-3. Syntax highlighting sample

```python
def act_step(sum):
      return 1.0 if sum >= 0.0 else 0.0
```

NOTE

The code examples here are meant for demonstration only. While the code is syntactically correct and will run, don't expect much from the output. This is because the network weights still need to learn. Something we will cover later in this chapter.

Finally, we can put all this code together in Listing 1-4 where we have written a forward_pass function that combines summation and the earlier linear activation function.

Example 1-4. Syntax highlighting sample

```python
def forward_pass(inputs, weights):
      return act_linear(summation(inputs, weights))

print(forward_pass([2,3,4],[2,3,4,5])
```

Can you predict the output of Listing 1-4 and previous related listings? Try to predict the outcome without typing the code into a Python interpreter and running it. We will leave it as an exercise to the reader to find the answer on their own. While the code in the previous example may seem simple there are a number of subtle nuances that often trip up newcomers. Therefore, will reinforce the concept of the perceptron further in the next section by playing a game.

The Perceptron Game

Games and puzzles can be a fun, engaging and powerful way to teach abstract concepts. The Perceptron game was born out of frustration from teaching students the previous coding example. Only later to realize that 90% of the class often still missed major and important concepts. Of course, many other deep learners, including the godfather himself Dr. Geoff Hinton, has been said to use variations of a similar game. This version is meant to be played as a solitaire puzzle or as a group collaboration. It really depends on how many friends you want to play with. One thing to keep in mind before inviting the family over is that this game is still heavily math focused and may not be for everyone.

NOTE

You can find all the printable materials for the game from the books source code download for Chapter 1.

The play area for the Perceptron game is a perceptron or in this case a printed mat like that shown in Figure 1-7. This is the same figure we seen previously but this time annotated with some extra pieces. Aside from printing out the play area, the perceptron mat, you will need to find about 8 - 6 sided dice. You can use fewer dice, but the more the better. We will use the dice as numeric place holders. For the most part the value on the dice represents it’s respective value, except for 6 which takes 0.

Thus the value for each die face is: . = 1 . = 2 . = 3 . = 4 . = 5 . = 0

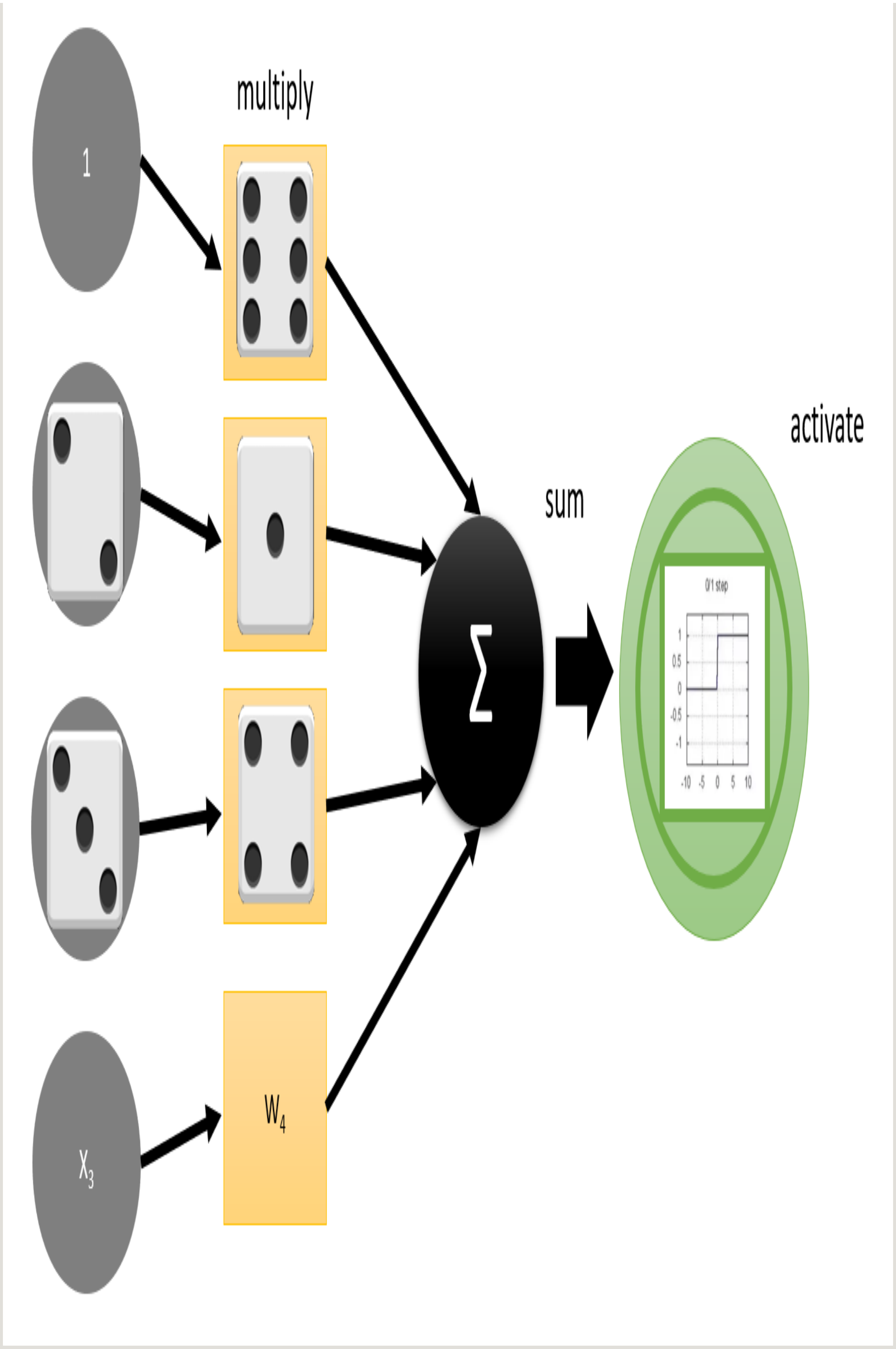


Figure 1-7. The Preceptron Game Play Area with Dice

Given the die positions on the mat shown in Figure 1-7. We can see there are 2 inputs represented by 2 and 3. Then inside the perceptron we have weights set to 0, 1, 4 (remember 6 = 0). Provided these inputs and weights we can calculate the total summation by:

1. bias = 1 x 0 = 0
2. input 2 = 2 x 1 = 2
3. input 3 = 3 x 4 = 12

Total Sum = 0 + 2 + 12 = 14

The total sum now needs to be output through an activation function. For simplicity we will say our current perceptron does not use an activation function. This means all the outputs will be linear or raw values. So 14 becomes the output value for the perceptron, except assume the real answer we want, the labelled answer, is 10. That means our perceptron has to learn, the weights need to be adjusted in order to provide the right output. Fortunately, there is a relatively simple equation, Equation 1-2, that can do that.

Equation 1-2. Equation 1-2

$$W_i = W_i + \alpha\,(L - O)$$

Where: L = the labeled value O = the output from the perceptron W = the weight to be adjusted \alpha = training constant

Equation 1-2 adjusts each weight by a factor controlled by alpha and which is a result of the difference in actual value and one predicted (forward pass) in the perceptron. Going back to our last example we can correct one of the sample weights shown by substituting values into Equation 1-2 and assuming a vlaue of .1 for alpha and the weight to be 4 (above example), we get Equation 1-3.

Equation 1-3. Equation 1-3

$$3.6 = 4 + .1(10 - 14)$$

Thus from Equation 1-3 we can see the new value for the weight would be 3.6. Now, if we put those values back into the equation the new output for the perceptron would be 12.8. However, the right answer is still 10. Which is okay and the reason for that is we don’t want to adjust a single weight to quickly. Remember, that this is only one input and we may need to adjust for thousands or millions of inputs. Hence the reason we only set alpha to a small value. By using only a small value we can then incrementally go through the inputs over and over again until the perceptron weights learn. Going back to the previous example with actual 10 and output 14, we can perform weight updates iteratively as shown in Table 1-5.

Table 1-5. Perceptron Learning

X1	X2	W1	W2	Label	Ouput	Error
2	3	1	4	10	14	4
2	3	.6	3.6	10	12	2
2	3	.4	3.4	10	11	1
2	3	.3	3.3	10	10.5	.5
2	3	.25	3.25	10	10.25	.25
2	3	.125	3.125	10	10.125	.125 …

By iteratively adjusting weights we can see how the perceptron converges to an answer for a single set of inputs. We of course want to look at far more complex problems and hence the reason for the game.

NOTE

Stop, before you throw this book down and yell, "Eureka! I know how neural networks learn!". Real networks use a far more complex method called backpropagation, which is coming up in this chapter.

The goal of the perceptron game is to find the weights that will solve for the correct outputs. In Table 1-6, there is a list of single inputs and the expected outputs. What you need to do is find the weights (weight 1 for the bias and weight 2 for the input) that will let the perceptron predict the correct output.

Table 1-6. Game 1

X1	Expected Ouput
4	14
3	11
2	8
1	5 …

Now you have a number of options to use in able to learn or set the weights. You can: * Guess - as a human you may be able to intuitively figure the answer out in your head. Try to guess what the weights are first. * Random - use the dice and roll random values. Then try those random values and see if those work. As a hint, the bias (weight 1) and input 1 (weight 2) weights are not the same value and not zero (6 on a die). * Equation 1-3 - use the equation we looked at earlier and use that to solve for the weights. If you get stuck this may be a good method to fall back on. * Programming - We will frown upon this option in this chapter, but just this chapter. Leave the programming for later.

TIP

Even if you guess the answer quickly try using the Random method as well. Understanding how different methods solve for the weights is the point of this exercise.

The answer to this problem and the others is provided at the end of the chapter. We didn’t want readers to spot the answers while doing the problem. When you are done check your answer at the back of the book and move on to the next Perceptron puzzle in Table 1-7 and 1-8.

Table 1-7. Game 2

X1	X2	Expected Ouput
4	2	8
3	1	5
2	0	2
1	3	7
0	4	8
5	5	15

Table 1-8. Game 3

X1	X2	X3	Expected Ouput
4	2	1	8
3	1	0	5
2	0	2	2
1	3	3	7
0	4	4	8
5	5	5	15

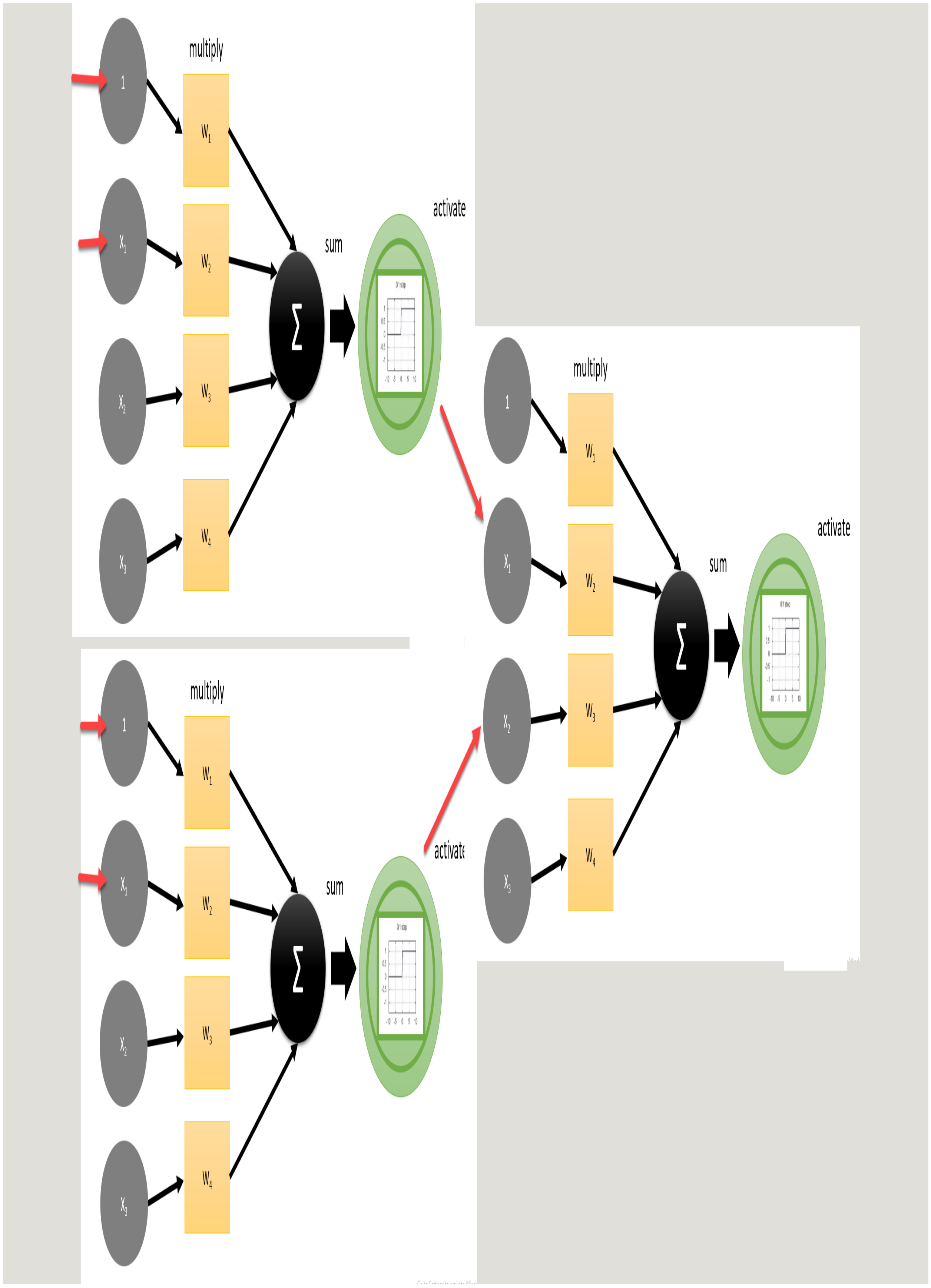
Now there can be multiple answers to the above games depending on how you solve it. The answers in the back are provided via guessing and may differ if you used Equation 1-3 for instance. Either way, if your perceptron is able to regress the right output and you understand how this is done you are well on your way.

With regression under our belt it is time to move on to classification. Now we are interested in classifying something as either in a class or not. That is a day is wet or dry, cold or hot, cloudy or sunny. However, in order to do this correctly we now have to step our output through an activation function. Using an activation function, particularly the step function, will allow us to better classify our output. Refer back to Listing 1-3 to review the step function but essentially if the output is less than zero nothing is output but 1.0 otherwise. Now, if we consider the game in Table 1-9 the output is shown as a class 0 or 1.

Table 1-9. Game 4

X1	Expected Ouput
4	0
3	0
2	1
1	1 …

Programmatically you could likely solve Game 4 in seconds but what weights would you need to solve the perceptron that could properly classify those outputs? Well, the problem is it can’t be done. Go ahead and try Equation 1-3, but it doesn’t work. Not for a single perceptron anyway. We can solve this however by adding a couple more perceptrons like that shown in Figure 1-8. In this figure we can see three perceptrons connected, 2 input and one output. We call each set of perceptrons a layer. Therefore, the figure has an input layer with 2 perceptrons and one output layer with a single perceptron. Inside these perceptrons there are 4 input weights (bias + input X 2) in the input layer and 2 weights in the output layer. Are you able to balance these weights now in order to provide the correct output? Give it a try.



multiply
1
X1
X2
X3
W1
W2
W3
W4
sum
∑
activate

Figure 1-8. Multilayer Preceptron Game 4

NOTE

In classroom settings we typically have students form groups and pretend each is a perceptron. They are then told to organize themselves in layers and solve the various weights in the problem.

Table 1-10. Game 5

X1	X2	X3	Y1	Y2
4	0	2	0	1
5	1	3	0	1
3	2	4	0	1
2	3	5	1	0
1	4	0	1	0
0	5	1	1	0

For the last game, we want to increase the number of outputs from one class to 2. This means we also need to put 2 perceptrons in the output layer. Likewise, we are adding another input and therefore it likely makes sense to also add another input perceptron. We then end up with a multi-layer perceptron network shown in Figure 1-9. The figure shows a network with 3 input perceptrons each taking 2 inputs for a total of 9 weights (2 input + bias X 3). Then in the second output layer of 2 perceptrons we have 8 weights (3 input + bias X 2). For a total of

17 weights. However, the game is far more simpler than it first appears and the trick is to follow the zeros.

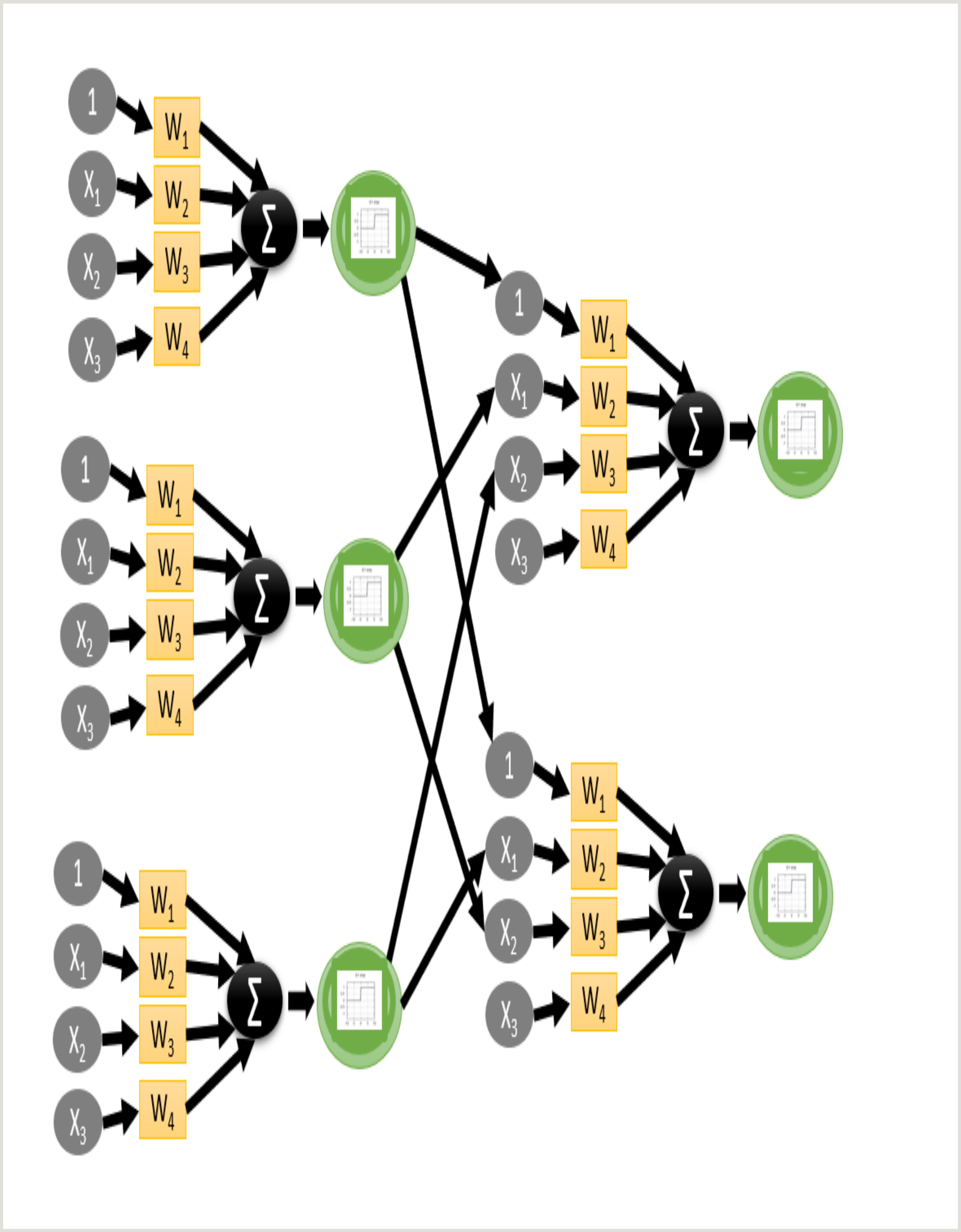


Figure 1-9. Multilayer Perceptron Game 5

After you solve each problem consult the back of the chapter for the answer. You may be able to solve the games completely in your head but it can also help to physically use a mat and dice to try to solve the games randomly, perhaps even more fun. However, you should also consider at this point how you might apply Equation 1-3, the perceptron learning equation to a multi-layer perceptron. The short answer is you can’t and we will look to why in the next section.

Understanding How Networks Learn

As we’ve seen by playing the Perceptron Game when we start to combine multiple perceptrons into layers things get complicated quickly. We call those multi-layer perceptron models neural networks or advanced neural networks or more recently deep learning systems. Whatever we call them, when we scale from a single perceptron to even just a few solving for the amount to update a single weight in the entire system becomes very complicated. You likely already realized that when playing the game but hopefully you also figured out that solving the weights becomes systematic. That is once you have one weight figured out you can move backwards and solve the rest. As it turns this systematic solving the weights by moving backwards is how we solve the weights in networks today. That system is called backpropagation and something we will delve into greater detail next.

WARNING

As you’ve already seen there are numerous ways to solve the weights in a network. Random was often a good solution before networks became far too complex. However, the preferred method is now backpropagation which may change in the future.

Backpropagation

While Equation 1-3 will work for updating or learning the weights in a single perceptron it is not able to find the updates across an entire network. In order to do this we fall back to calculus which is able to determine how much change or affect each weight has on the network output. By being able to determine this we can work backwards and determine how much each weight in network needs to updated or corrected. This system is called backpropagation and the complicated parts come from calculus but fortunately the whole system can be automated with a technique called automatic differentiation. However, it still is important to intuitively understand how this system works in the event something goes wrong. Problems will and do often happen and they are the result of something called diminishing or exploding gradients. Therefore, to understand if you have a diminishing or exploding gradient we will explore backpropagation in some detail.

In order to determine the amount of change of each weight we need to know to calculate the amount of change for the entire system. We can do this by taking the equation that gives us the forward answer, or prediction and differentiate it with calculus. Recall that calculus gives us the rate of change of an equation or system. For basic calculus with one variable this is elementary and Figure 1-10 shows how a function can be differentiated at a single point to find the gradient or change at that point.

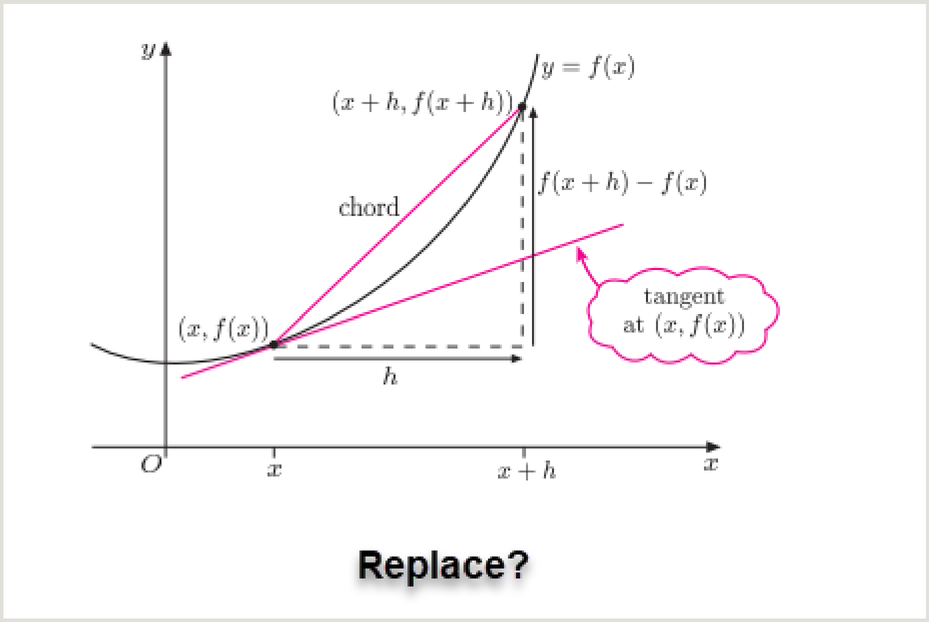


Figure 1-10. Finding Gradient of Point on Function

NOTE

If you have a foundational knowledge of calculus you should be able to understand the upcoming material. Even if it has been sometime since you practiced calculus. However, those readers with no knowledge of calculus are recommended to explore that material further on their own. There are plenty of free videos or courses online that can provide this knowledge.

Now that we understand calculus is essential we can move on to solving our equations. However, with a multilayer perceptron each perceptron has it's own weights, summation and activation functions so differentiating all of this sounds quite complex. The short answer is yes, it very much used to be. Except, we have found tricks to dramatically simplify the problem. If we consider that each layer of perceptrons uses the same activation function than we can treat the entire layer as linear system of equations. Thus reducing a single layer down to a single function such as f(). Incidentally, reducing a layer down to a linear system of equations or in other words a matrix, reduces the computational complexity immensely as well. This is internally how all deep learning systems work today and this is what makes processing a layer through a network so fast. It is also the reason that deep learning systems now surpass humans in terms of many tasks we previously thought we would never be surpassed on.

NOTE

Included in your required maths background is linear algebra. Linear algebra helps us solve linear system of equations and also do some cool 3D visuals like games. It is another mathematical tools that can help you understand deep learning and other system, likely more so than calculus.

By reducing an entire layer down to a system of equations we can then assume a single function f(). Each successive function then would apply itself to f, like so g(f()). Where the g function is a second or successive layer in a network. Remember that the output from the first layer feeds into the second layer or function and so on. We can solve this function by using the chain rule and demonstrated in Equation 1-5.

Equation 1-4. Equation 1-5

$$h(x) = g(f(x))$$

$$\frac{df}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

In Equation 1-5 the first equation is reduced down to equation h(x). From calculus we can use the chain rule which tells us that any equation in the first form can then be differentiated in the second form. This gives us a method to differentiate each of the layers and then using some more math magic we can derive the set of specific weight update equations shown in Figure 1-11.

$$x_{i,j}^{l} = \sum_{m}\sum_{n} w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b_{i,j}^{l}$$

$$o_{i,j}^{l} = f(x_{i,j}^{l})$$

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}}$$

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right)$$

$$\frac{\partial E}{\partial w_{m',n'}^{l}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta_{i,j}^{l} o_{i+m',j+n'}^{l-1}$$

Figure 1-11. Finding the Gradient Weight

This equation shows the calculus for deriving the gradient of change for each weight. Gradients represent an amount and direction of change. Thus, by finding the gradient, we can understand the amount the individual weight or parameter contributed to the output error. We can then reverse the gradient and adjust the weight in the opposite direction. Keep in mind that each time we change a weight we want to do the least amount possible. That is so a change in the weight doesn't cause another weight to get unbalanced. This is quite tricky when we train thousands or millions of weights so we introduce a learning rate called alpha. Recall, we used alpha in our single perceptron example to set the amount of change or improvement each iteration and the same applies here. Except in this case we need to make a alpha a much smaller value and in most cases the value is .001 or less.

Alpha = the learning rate of a network. Is a common parameter we will see over and over again and it is used to tune how fast a network trains. Setting it to to a low a value and the network learns very slowly but it may avoid certain training pitfalls. Setting alpha too high, and the network learns quickly but then will likely become unstable. Instead of converging to an answer it will likely give a wrong answer. These issues occur because the network may get stuck in some local minimum as shown in Figure 1-12. When the goal of any network is find to the global minimum or maximum value.

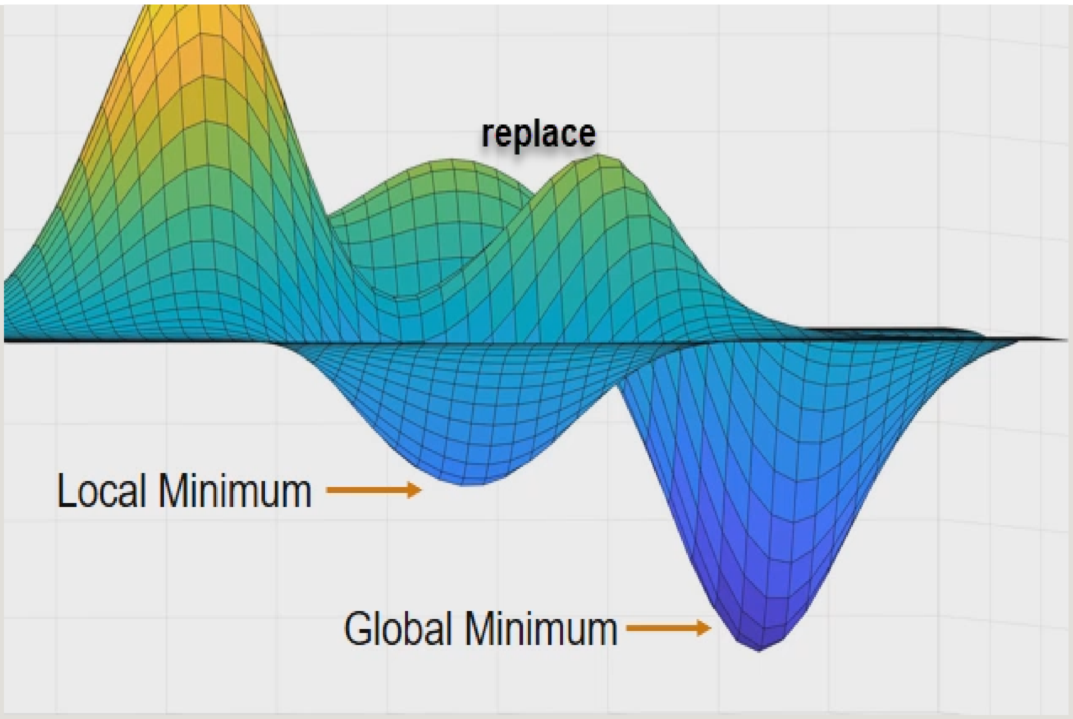


Figure 1-12. Local Minimum vs Global Minimum

Optimization and Gradient Descent

The whole process of backpropagation is further described as using gradient descent. So named because backpropagation finds the gradient that describes the impact of an individual weight. Then it reverses the direction of the gradient and uses that to find the global minimum of the solution. We call this entire process of optimizing a solution to a global minimum as optimization. So named because we reduce the total errors of a solution to a global minimum. Optimization itself is fundamental to data science and is used to describe the method that minimizes the errors of a method. Minimizing error is relative to the function being performed, either regression or classification for instance, and uses the specific function error metric to determine performance. For example with regression we may minimize on mean squared error (MSE) or in the case of classification accuracy.

Optimizers come in several variations but many of the ones we use for deep learning are all based on gradient descent, the backpropagation method. Here is list of optimizers we will cover throughout this book:

- Gradient Descent - this is the base algorithm and works as described in the section on backpropagation.
- Stochastic Gradient Descent (SGD) - this is an improved version that uses random batch sampling to improve on generalization. This is the actual standard and we will cover a whole section on this method below.
- Nesterov - this method introduces the concept of momentum. Momentum is like an additional speed control for SGD and allows it to converge quicker. Nesterov provides an additional speed boost to momentum as well.
- AdaGrad - is a form of gradient descent that adjusts to how frequent the data is. This, in turn, gives it an advantage when handling sparse data. That is data associated with infrequent features with higher value will benefit more when using AdaGrad. This method does suffer from diminishing learning rates however. This method also introduces the concept of adaptive learning rates.
- AdaDelta - is based on AdaGrad but improves on AdaGrad by not requiring an initial learning rate (alpha) as the algorithm will adjust it’s own. It also manages the diminishing learning rates better.
- RMSprop - was an independently developed version of AdaDelta by Geoff Hinton.
- Adaptive Moment Estimation (Adam) - is an extension to AdaDelta and RMSprop that allows finer control over the

momentum parameters. Adam is also one of the more popular optimizers you may encounter in recent papers.

- AdaMax - is an improvement to Adam that updates the momentum parameters.
- Nadam- is a combination of Nesterov with RMSprop which is like super charging the momentum on RMSprop.
- AMSGrad - is a new Gradient Descent algorithm with momentum that intends to improve on methods like Adam where it is shown just SGD with momentum works as well or better. This method is becoming the goto method when Adam does not perform as well may be expected.

The above list has doubled since about 2012 and it is quite likely it could again double in another few short years.

NOTE

You may find yourself generally sticking to a few standard optimizers for various classes of problems. A lot of this depends on the problem you are trying to solve and the data you are using. We will of course explore further details about optimizers in later chapters as we solve particular problems.

Vanishing or Exploding Gradients

Now generally, the whole system of backpropagation (gradient descent) and finding the partial derivative with respect to each weight works automagically. That is, most deep learning libraries like Keras, TensorFlow and Pytorch provide automatic differentiation of the partial derivative of an network out of the box. While this is incredibly powerful and for something we used to do by hand a blessing, it still has some problems. While we generally won't encounter these issues until we look at larger and more complex networks, it is worth mentioning here.

What happens occasionally and for a variety of reasons the gradient descent optimization algorithm may start calculating what we call an exploding or diminishing gradients. This may happen for the various optimizers we covered earlier. Remember, a gradient denotes the amount and direction a weight contributes to the network. What can happen, is that an optimizer may start to calculate an incredibly large value for a gradient, called an exploding gradient, or conversely very small or diminishing gradients. In the case of exploding gradients the network will start to generally overpredict. While in the case of diminishing gradients the network will just stop learning and freeze. In order to help you diagnose these issues early use the following guide:

1. Network does not improve after x number of iterations
2. Network is unstable and you see large changes in error moving from positive to negative
3. Network appears to go backward in learning

The best way to diagnose these issues is by watching and monitoring how your network trains. In most cases you will want to closely observe your network training for the first several thousand iterations. In the next section we talk to one further optimization when training networks.

SGD and Batching Samples

One problem we come across when training thousands of data through a network is it can take a long time and isn't general enough. That is, if we update our network for each individual weight we may find elements that cancel each other out. This can further be compounded if the data is pulled from the same order. In order to alleviate these problems we introduce a random batching approach to updating our network. By batching the data into groups of data and then applying changes averaged across those groups better generalizes the network, which is generally a good thing. Furthermore, we randomize this batching process so that no 2 batches are alike and data is further processed randomly. This whole technique is called stochastic gradient descent when used with backpropagation in order to train a deep learning network.

We use the term stochastic to mean random, since we are now pulling random groups of samples. The gradient descent part is the heart of backpropagation optimization as we already learned. Stochastic Gradient Descent or SGD is the standard optimizer as we've seen earlier. There are plenty of variations to SGD that are more powerful and we will explore those as well. The important thing to remember about SGD and other optimizers is that they use batches of data and no longer individual samples. As it turns out, since we are using linear systems of equations this also becomes more computationally efficient.

- batch_size = the batch size used to determine updates to the network. Typical batch sizes are 32 - 256 for large dataset sizes. The batch size is a deceptive parameter that may or may not have an incredible impact on network training. It generally will be one of the first parameters you tune in enhance a network.

Another improvement to batching is mini-batching. That is when we break up and batch the batches into smaller batches. These smaller also random batches have been shown to increase data variance further, which is a good thing. This in turns leads to better generalization and, of course, training.

There is, a third option or should we say the original option. Originally, data was just batched and the method was called Batch Gradient Descent. The major problem with this was the batches were always the same and this reduced the data variance. Which, as we now know, led to decreased training performance and learning. Batch gradient descent is an option but not one you will choose very often.

Batch Normalization and Regularization

Batch normalizing is an additional process we may perform as the inputs flow through the network. By normalizing the data in a batch or after it processes through a layer allows for more stable networks by avoiding vanishing and exploding gradients. Regularization is the same process but typically involves balancing internal networks weights using the L1 or L2 norm. We use the term norm to refer to a normalization of the vector space or as we perform in linear algebra, normalizing a vector. The L1 or L2 refer to the distance used to calculate the vectors magnitude. In calculating the L1 norm we use what is referred to as the taxi cab or block distance while the L2 norm is the more typical euclidean distance. An example of calculating the L1 and L2 norm is shown in Equation 1-6. Notice the subtle but important difference between the 2 calculations.

Equation 1-5. Equation 1-6

$$||X||_1 = |3| + |4| = 7$$

$$||X||_1 = |3|^2 + |4|^2 = \sqrt{9 + 16)} = \sqrt{25)} = 5$$

Normalization and regularization can be important way to optimize deep learning as we will see when we start building networks.

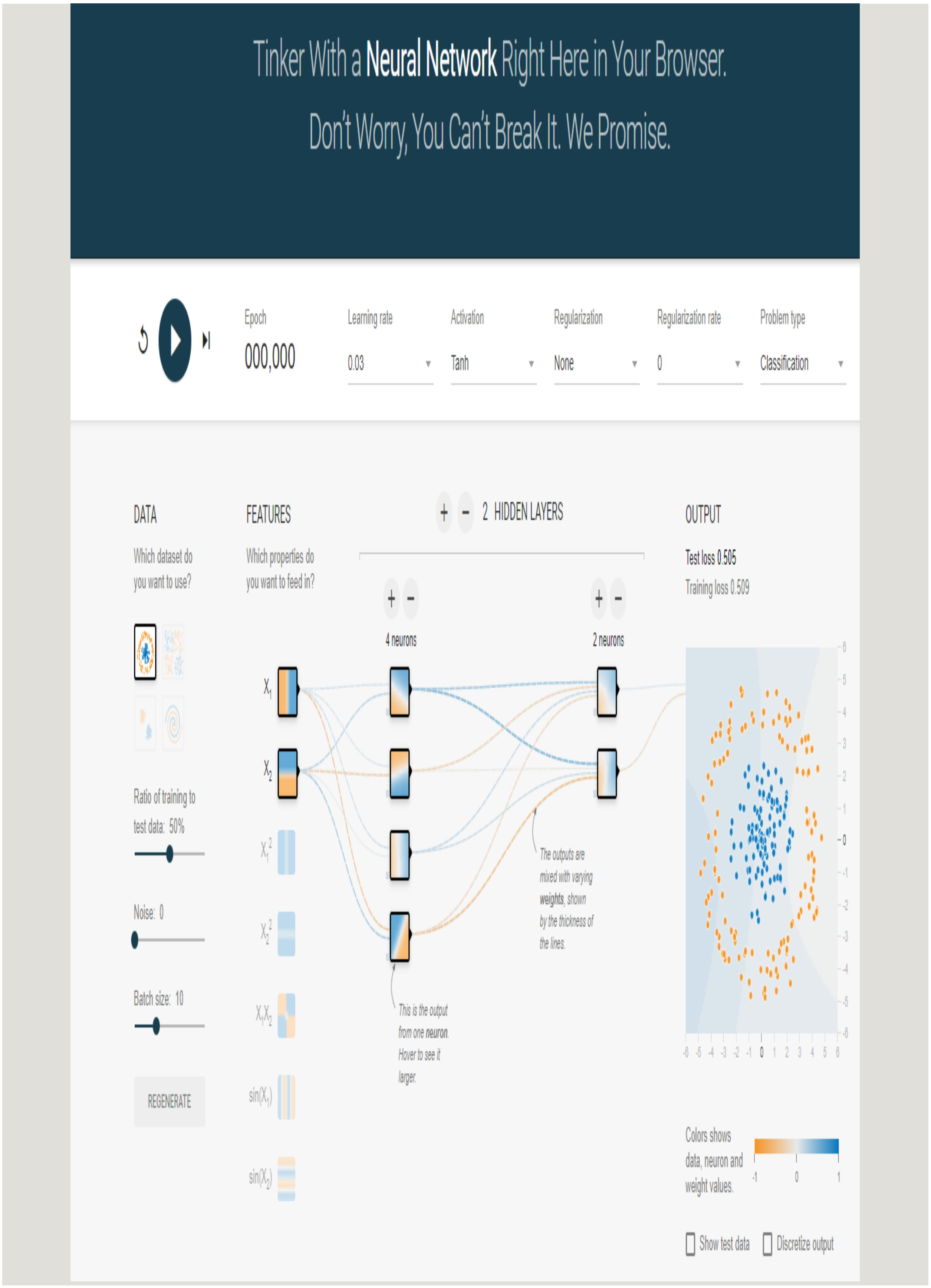
Activation Functions

We already covered the absence of an activation function or just straight linear output of a network. As well, we looked at the step activation function. Which essentially steps the output and is very useful in classification problems. We will use a variety of activations functions that are specific to regression or classification. However, in many cases we may use broader functions to work between hidden layers of networks that will work on either problem. Much like optimizers there are a variety of activation functions we find more useful for certain problems and data types. There is often no hard or fast rule and a lot your experience working with these functions will be by experience. Another option is to digest several papers and take recommendations from those. While that can work and is quite useful anyway, you often have to be careful that problems and network design align well with your own problem. The following is the more common activation functions you may come across and we will likely use in this book:

- Linear - essentially no function. The output from summation is sent direct to output. This as we've seen, is a perfect function for regression problems.

- Step - we’ve seen a basic implementation of a step function in Listing 1-3. Use this one for classification problems.
- Sigmoid or Logistic activation - also called the squishification function. Called that because it squishes the output to a value between 0.0 and 1.0. It was the first common activation function because it was so easy to differentiate when calculating backprop by hand. Figure 1-13 shows the sigmoid function and how it resembles logistic regression. This method is used for classification.
- Tanh or hyperbolic tangent function - squishes the output to between -1 and +1. Used best for classification problems.
- ReLU (Rectified Linear Unit) - is a combination of the step function and linear. This function is used for regression and quite often in between hidden layers.
- Leaky ReLU - exactly like ReLU but with a leaky step function. This function has been found to quite effective in controlling vanishing or exploding gradients. This one works best between layers.
- Parametric ReLU (PReLU) - is a leaky ReLU function that provides further control with parameters. Again best used between layers but also works for regression.
- ELU (Exponential Linear Unit) - provides an exponential rather than linear response. This method works for regression problems. Works well between layers and for regression.
- Softmax - is for classification problems where the output is represented by a probability vector that denotes how well an output, in range 0.0 to 1.0, fits within a set of classes. The total sum of the output of all classes equals 1.0 or 100%. If we go back to the Perceptron game and review the classification problems, in that case we needed 2 output

neurons to denote our separate classes. Softmax, would allow us to reduce our output to one neuron that can output a vector of the classes and probabilities of being within each class.



Tinker With a **Neural Network** Right Here in Your Browser.

Don't Worry, You Can't Break It. We Promise.

Epoch 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

2 HIDDEN LAYERS

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

Test loss 0.505

Training loss 0.509

The outputs are mixed with varying ***weights****, shown by the thickness of the lines.*

This is the output from one ***neuron****. Hover to see it larger.*

Colors shows data, neuron and weight values.

Figure 1-13. Activation Functions

Figure 1-13 has also almost doubled in just under a decade. That means if you are training deep learning networks you need to keep current with the best activation and optimization functions and so on. Fortunately, by using AI (deep learning) services provided by Google, the cloud services manage most of that away or as we will see provide us help along the way.

Loss Functions

As we've seen when we talked about Goodness of Fit, earlier in this chapter our methods or what we call models need a way to determine the amount of error. We may also use the terms loss or cost to denote the total amount of error. Recall, that our goal is to minimize this error, loss or cost using some variation of gradient descent optimization. Loss or error functions as we've seen with MSE or mean squared error also are differentiated by the goal of the network be it regression or classification. Below is a quick preview of loss functions you will likely encounter in this book or on your own exploring deep learning solutions:

- MSE (mean squared error) - we already covered this method earlier when we looked at regression. MSE represents the mean average error distance squared.
- RMSE (root mean squared error) - the square root of MSE. This variation is useful when trying to better understand the variance of your model.
- MSLE (mean squared logarithmic error) - denotes the MSE on a logarithmic scale. This method is useful for large ranges

of numbers. That is when values range from zero to billions or more.

- MAE, (mean absolute error) - measures the error distance between two variables or features. Thus in a 2D (x,y) regression plot, error would be measured on the x and y axis both vertically and horizontally. Where in MSE, the measure of error is only the vertical difference on the Y axis.
- Binary Classification functions - there is a whole list of base error functions that measure classification on outputs. That is they determine the amount of error an input is within a class (1) or not within a class (0). This works well for binary problems that is if they are in or out.
- Binary Cross-Entropy Loss - what we find with classification problems is that mathematically it works better to classify in terms of probability within a class rather than to just binary classify as above. That means this becomes the preferred method and one we will discuss at length when we get to those chapters later.
- Hinge Loss - is a binary classification loss function similar to cross entropy. It differs in that it allows classification to range in values from [-1,1]. Standard cross entropy uses values in the range [0,1].
- Squared Hinge Loss - an extension to Hinge Loss.
- Multi-class Classifier - this is useful for when you want to class an input into multiple classes. For example a picture of a dog fed into a network could be identified as being a dog, perhaps a specific dog breed and perhaps color.
- Multi-class Cross-Entropy Loss - is the same approach we use in binary cross-entropy except for multiple class problems. It is the preferred and standard approach.

- Spare Multi-class Cross-Entropy Loss - deals with the problem of identifying data over a large number of classes. Datasets as large as 11000 classes have been released in just the last year. Even with 18 million images fed into a classifier with that many classes still only leaves about 1600 images per class.
- Kullback Leibler Divergence Loss (KL Divergence) - this function is an advanced function that determines the amount of error between distributions of data. Which makes it not well suited to multi-class classification problems but does well for adversarial training.

Use the list of above loss functions as a reference. We will explore the more important loss functions more closely when we get hands on in examples. In the next section we look to building a simple multi-layer perceptron network.

NOTE

Adversarial training is a form of network training we find in autoencoders and generative adversarial networks or GANs. They are so named because they pit networks against each other.

Building a Deep Learner

We already understand quite well on how to build a network but doing the backpropagation with automatic differentiation and all the other parts it really makes more sense to use a library like Keras, TensorFlow, Pytorch or others. All of these libraries are available for ease of use on a local machine which is sometimes required for data privacy concerns. Otherwise, for this book anyway we will use the cloud to build all our networks. However, it can be useful to look at code examples of how deep learning networks are built with other libraries in Python. Example 1-5 shows an example of a simple classifier network, one that could be used to solve our Perceptron Game 5 problem.

Example 1-5. Syntax highlighting sample

```
model = Sequential()
model.add(Dense(3, input_dim=2, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs=1000, batch_size=10)
```

The example Keras code shown in Example 1-5 builds a deep learning network with 3 input nodes and a 2nd output layer with 2 output nodes. We refer to perceptrons as nodes here to makes things more generic. Not all nodes in a future network may follow a perceptron. The code starts by denoting a variable called model of type Sequential, the model denotes the entire deep learning network which in this case is denoted as Sequential. Sequential here just means continually connected. After that each layer is added with an add statement, the first layer being 3 nodes with an input dimension of 3 and activation function called rectified linear unit or ReLU. Don't worry too much about the new activation functions just now, we will cover that later. Next the output layer is added with a sigmoid function. Then the entire model is compiled, which means set up for backpropagation. Finally, the model calls fit, which means it will iterate through the data for 1000 epochs or iterations, batching the learning process in batches of size 10.

Example 1-5 is an example of how accessible and powerful this technology has become. What can be done in 6 lines of Python code using the Keras library likely took hundreds of lines of code just a few years ago. However, as accessible as this technology is it still requires an immense amount of data and processing power to be effective. While data can be accessed free or available possibly from your organization. Computational processing is often another matter entirely and this why we focus on using cloud resources for all our networks in this book.

TIP

Keras is a great library that can get you quickly programming deep learning models. Be sure to check out the Keras website as keras.io for more info and tutorials to get you started.

Fortunately, there is a free tool available from Google that will allow us to setup a multilayer network quickly and train it in minutes, yes minutes. Open a browser and search for *tensorflow playground*. The search result will likely be the first but an example of the site is shown in Figure 1-14.

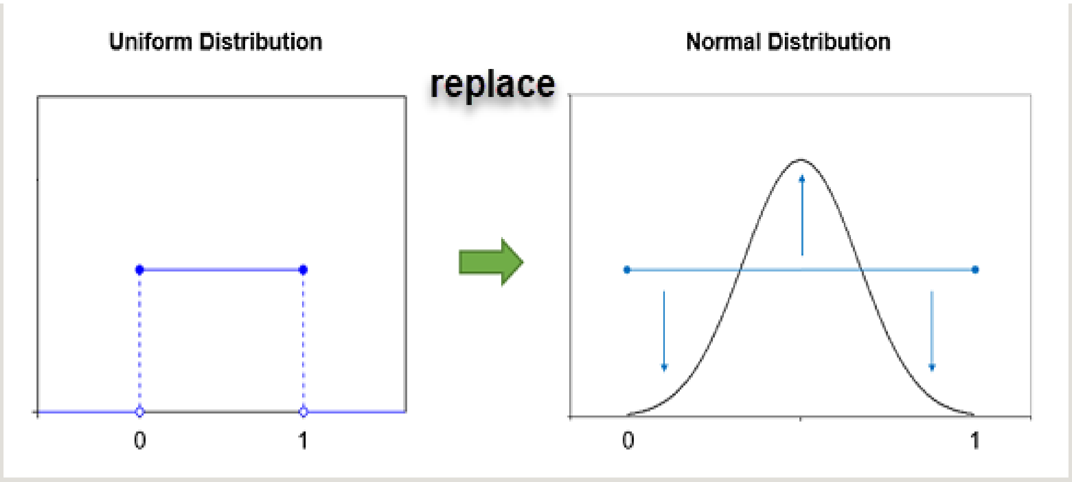


Figure 1-14. TensorFlow Playground

As soon as you open that site, Figure 1-14, you will see there are 2 inputs denoted X1 and X2 shown as 2 shaded boxes. These boxes represent distributions of data. You can think of a distribution as an endless box of data. Each time you reach in the box and pull a sample, at random, the value of the sample is determined by the distribution. This is an important concept and further explained in Figure 1-15. In the figure we can see 2 distributions. If we guess a value of .5 (x) and apply it to each distribution we get a value of .5 for uniform and perhaps 1.7 for normal. This is because the data is skewed by the shape of the distribution. This is an important concept to grasp and one we will revisit later.

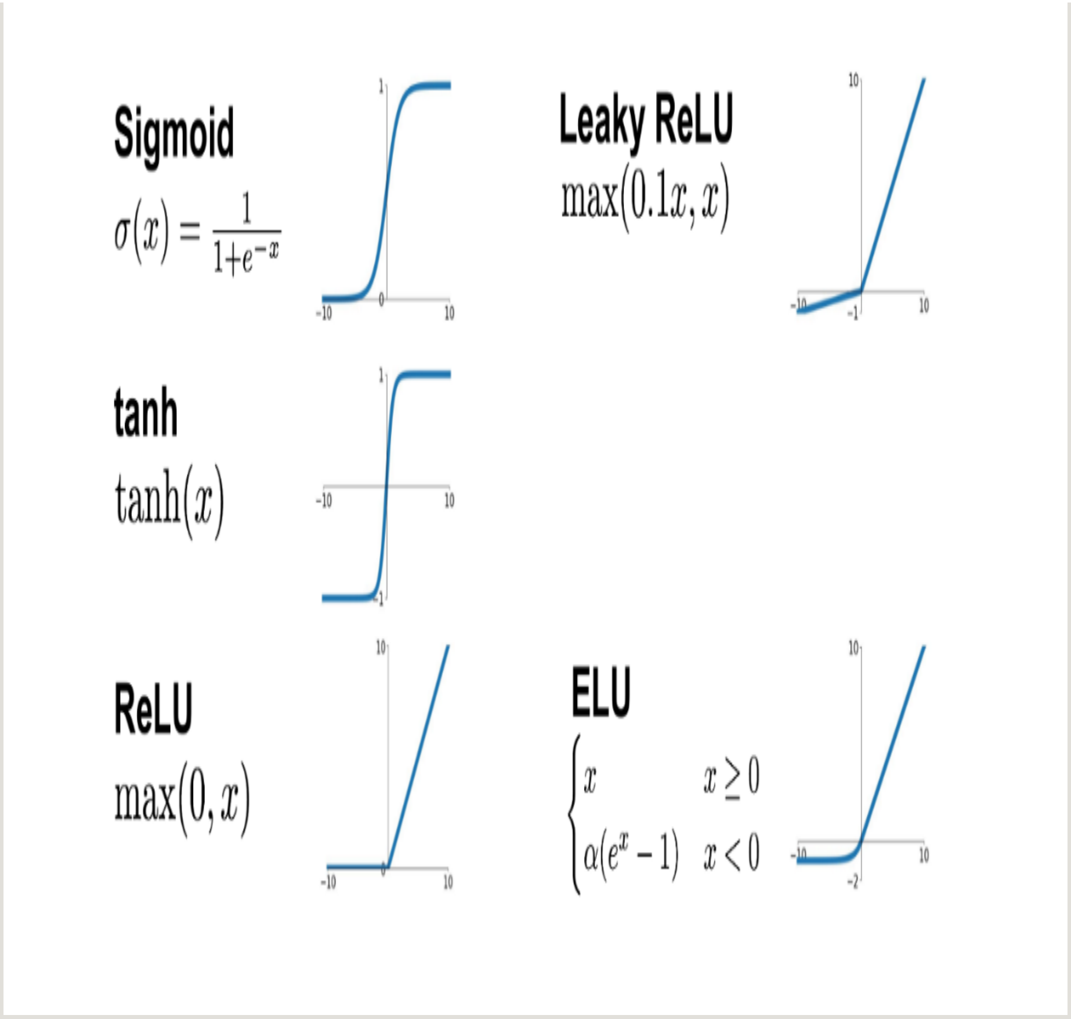


Figure 1-15. Comparison of Normal and Uniform Distributions

NOTE

Being able to understand distributions and probability are fundamentally important to data science and in turn deep learning. If you find you lack some knowledge in statistics or probability theory then you likely should brush up. Again, there are plenty of free materials online.

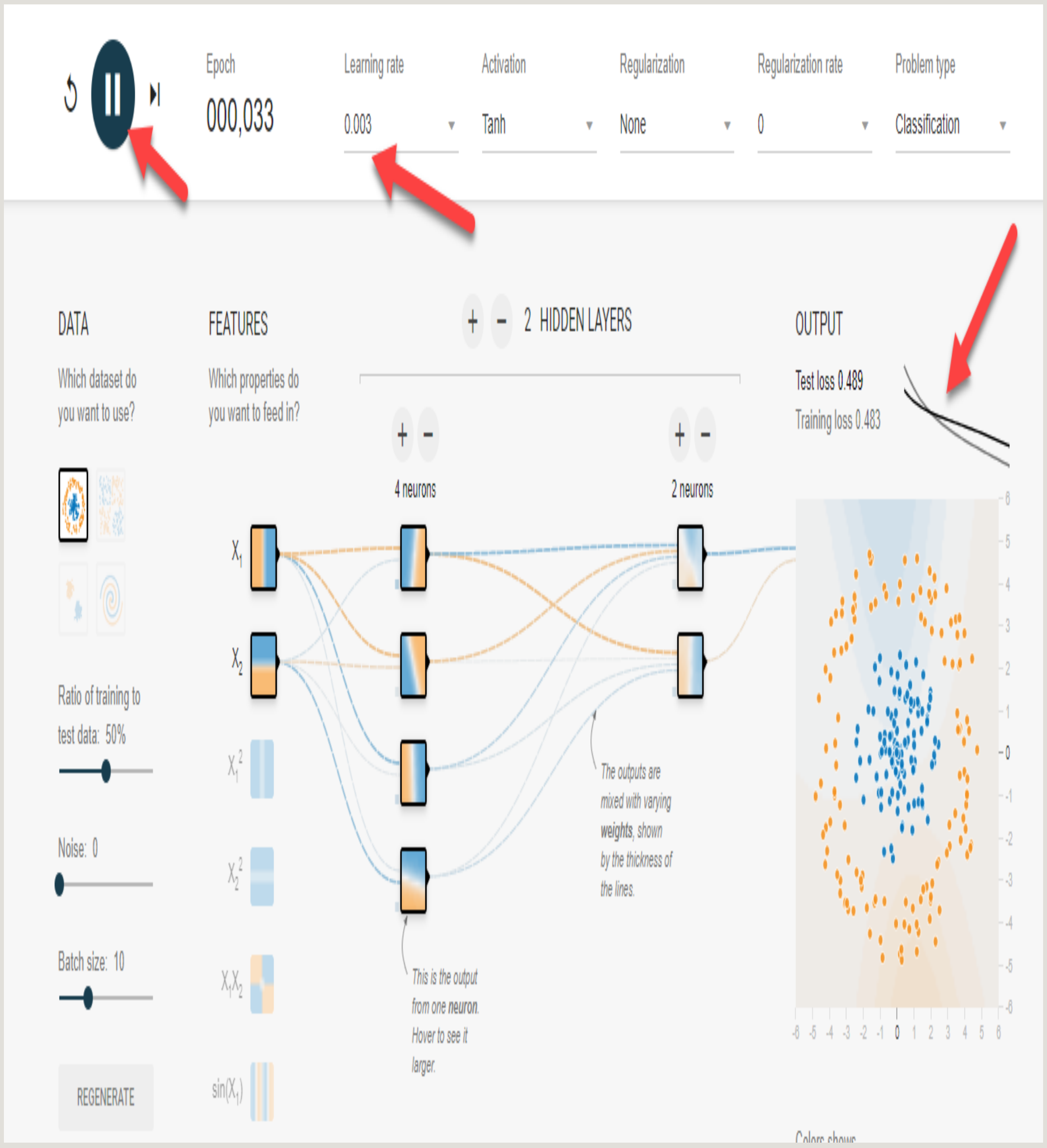
Getting back to Figure 1-13 and TensorFlow Playground we can see that inside the network their are 2 hidden layers with an input layer of 4 neurons and output layer of 2 neurons. Pressing the Play button in the left will start the network training and you will see how the network classifies the output as the number of epochs progress. In the end the loss is minimized to a small level and the fit looks quite nice. However, at this point, and always, we always want to understand if we can optimize the network in some manner.

OPTIMIZING A DEEP LEARNING NETWORK

After we have our inputs flowing through the network and can see an output this training effectively our next step is always to optimize a network. We do this step before running any test our validation data through as well. Recall, that we always want to break our input data into three sets of data, a training, test and validation set. Before doing that though there are few simple tricks we can apply to this model and in general as simple first step optimization tricks to any network:

1. Learning Rate - alpha = determine what effect adjusting the learning rate up or down has on the network. Adjust the Learning Rate to values .01 and replay the sample. Then adjust the rate to .1. Which learned faster?
2. Activation function - Tanh - try various activation functions. Tanh and Sigmoid are complimentary to classification, while ReLU and Linear are applicable to regression.
3. Regularization and Regularization rate - regularizing data is a form of normalizing data between layers. We this in order to avoid those exploding or diminishing gradients which can happen if a weight gets too large or small. . Hidden layers - increase the number of hidden layers and thus neurons in the network. Determine the effect this has on the network. Neurons - increase or decrease the number of neurons on each

layer of the network. Monitor the training performance of the network and watch for over or under fitting.


Epoch 000,033

Learning rate 0.003

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
4 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.489
Training loss 0.483
Colors shows

Figure 1-16. Training a network in Tensorflow Playground

Figure 1-16 shows a network being training with a modified learning rate. Play with the network and try to optimize the network to the fewest neurons that is still able to learn to classify the outputs effectively. Make sure that you do not overfit or underfit to the the data.

You are unable to control the loss function aside from setting the problem type as regression or classification. Be sure to switch between the two problem types and see what effect that has on the output as well. In the next section we look at what can happen if your network design has too few or too many layers or neurons.

Overfitting and Underfitting

One of our primary goals in optimizing a network is to build the smallest and most concise network for the task at hand. It can be very easy to throw endless layers, neurons and weights at a problem. The problem with this approach is that deep learning networks can actually memorize data. That is, they can learn data so well that they just remember the answer to a specific question rather than generalize an answer. This is the reason we withhold a percentage of data for both test and validation. Typically after optimizing a network to set of training data you then evaluate the trained network on the test data set. If the network predicts comparative results we say it has generalized to the training data. In some cases running the test set may generate very bad predictions and this often indicates the network has been overtrained or overfitted to the data.

Over or under fitting is a critical element to building successful networks so it is a topic we will revisit over and over again throughout this book. Although, is is easy to see how we can over and under fit using TensorFlow playground. Figure 1-17 shows the result of over and under fitting the neural network. Add or remove layers and neurons to see if you can create the same over and under fit patterns. You may also have to alter the learning rate, activation function and/or the number of epochs you run. On the bottom left side of the interface there is also options to set the ratio of training to test data as well as noise and setting batching size.

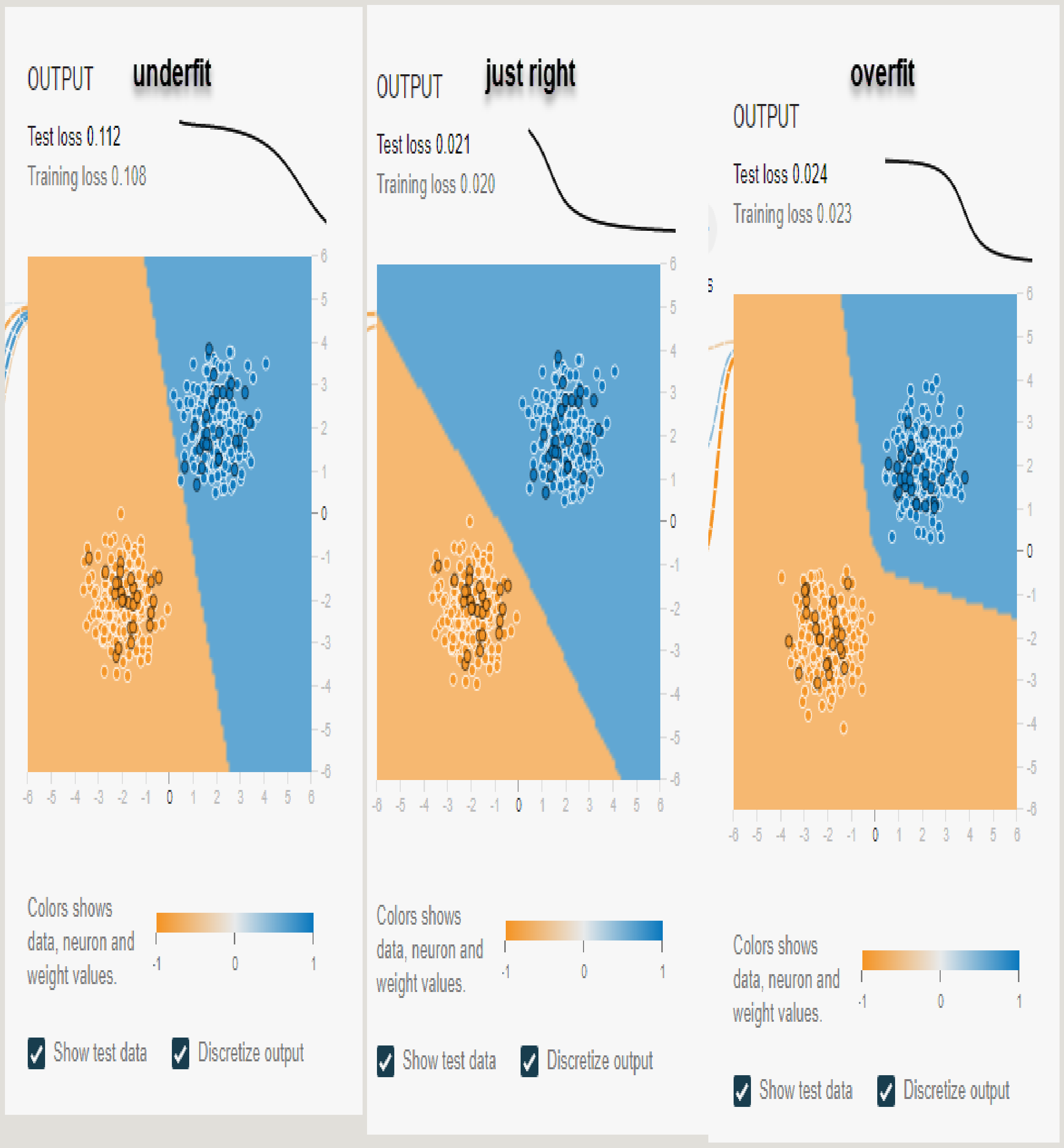


Figure 1-17. Example of over and under fitting

Network Capacity

The capacity of a network, that is the number of neurons and weights in a network describes its capacity to learn the required data. If you network is small, with only a couple layers and neurons you could not expect such a network to learn a large dataset. Likewise, too large of a network, one with lots of layers and neurons, and the network becomes able to memorize the data. Again this is the reason we break out test and validating datasets to confirm how well a network performs after training.

Hopefully you can appreciate now how simultaneously easy and difficult building deep learning networks can be “now”. On the surface stacking layers and designing networks is like combining lego blocks. Although lego blocks that wrap around complex systems of equations. Building deep learning models takes attention to detail and patience, lots of patience. Fortunately, using Google Cloud Platform will wrap many of the complex details and provide a performant platform that should reduce training times. Thus allowing you to conduct more experiments in the same time.

Conclusion

Deep learning has become the cornerstone of the new wave of AI tech that is embracing the globe. The thing we need to keenly remind ourselves though is that the cornerstone of this AI is still based on old tech like data science. This means we still need to understand the tenants of data science in order to be successful AI practitioners. That in turn means that understanding the data is also a requirement for anyone looking to be a successful. Sadly, this fact is often lost on eager newcomers looking to build cool AI only to find nothing they try works. Almost always this speaks to a lack of fundamentals and understanding of the data. Hopefully, you can appreicate the importance of data science and keep that in mind as we move into deeper AI. In the next chapter we will being exploring AI on the Google Cloud Platform.

Chapter 2. AI on the Google Cloud Platform

Since the implementation of the first self-driving cars Google has a predominant hand in shaping the AI landscape. The company has embraced all aspects of AI development from text and speech recognition and translation to deep reinforcement learning. Including aquiring or recruiting some of the best and brightest minds in the AI space. From this Google is now providing it’s expertise in AI as a complete suite of services. While others are competing to do the same Google is certainly the first place you should look when building any AI application in the future.

In this chapter we overview the many powerful AI services Google currently provides on it’s Cloud Platform. From here we will look at how you can sign up and start working with these services. Then we will dive in and work hands-on with Google Colab and build a couple deep learning examples with Keras. After that, we will look to expand on those examples by using the AutoML Tables service. Using this service to then upload and work with our own sample datasets. Finally, we will review some Google Cloud Shell and look at other utilities we will need over the course of the book.

In this chapter we start coding and working with cloud services. As such, you may find it helpful to have a computing device with a larger screen and keyboard in order to work with the coding examples.

Here is a brief summary of the main content we will cover in this chapter:

- AI Services on GCP
- Google Colab Notebooks
- AutoML Tables
- The Cloud Shell
- Managing Cloud Data

AI Services on GCP

The Google Cloud Platform (GCP) was launched in spring of 2008 with the App Engine platform. Since then the platform has grown to encompass several services from computing, storage, networking, security, Big Data, IoT and of course AI. While Google is often considered just behind Amazon with respect to general cloud services that can certainly not be said for it's AI services. Google has devoted considerable infrastructure and development efforts to developing the next wave of AI and has been doing so since it's investment in 2006 with self-driving cars. Hands down, the AI teams at Google, consisting of Google Brain, DeepMind and others are considered to be the best of the best. It therefore makes sense that Google will and has set the standards for AI services. Although, even with these high standards there are still many other challengers producing comparable products. In the end, you may find yourself looking to other services but in this book we will stick with Google for now.

Figure 2-1 outlines the structure of how the AI services within the GCP are organized. At the outside is the AI Hub, this is essentially the container for everything AI and GCP. Inside the hub are the main components the building blocks and platform. The AI Building Blocks consist of the various pre-developed, trained and tuned AI models for everything from Vision to language, speech, recommendations and more. These blocks are consumed by the platform services in the AI Platform and this is denoted in Figure 2-1 by the arrow. Throughout this book we will look at how to integrate those AI Building Blocks into the AI Platform and we will start with the main hub next.

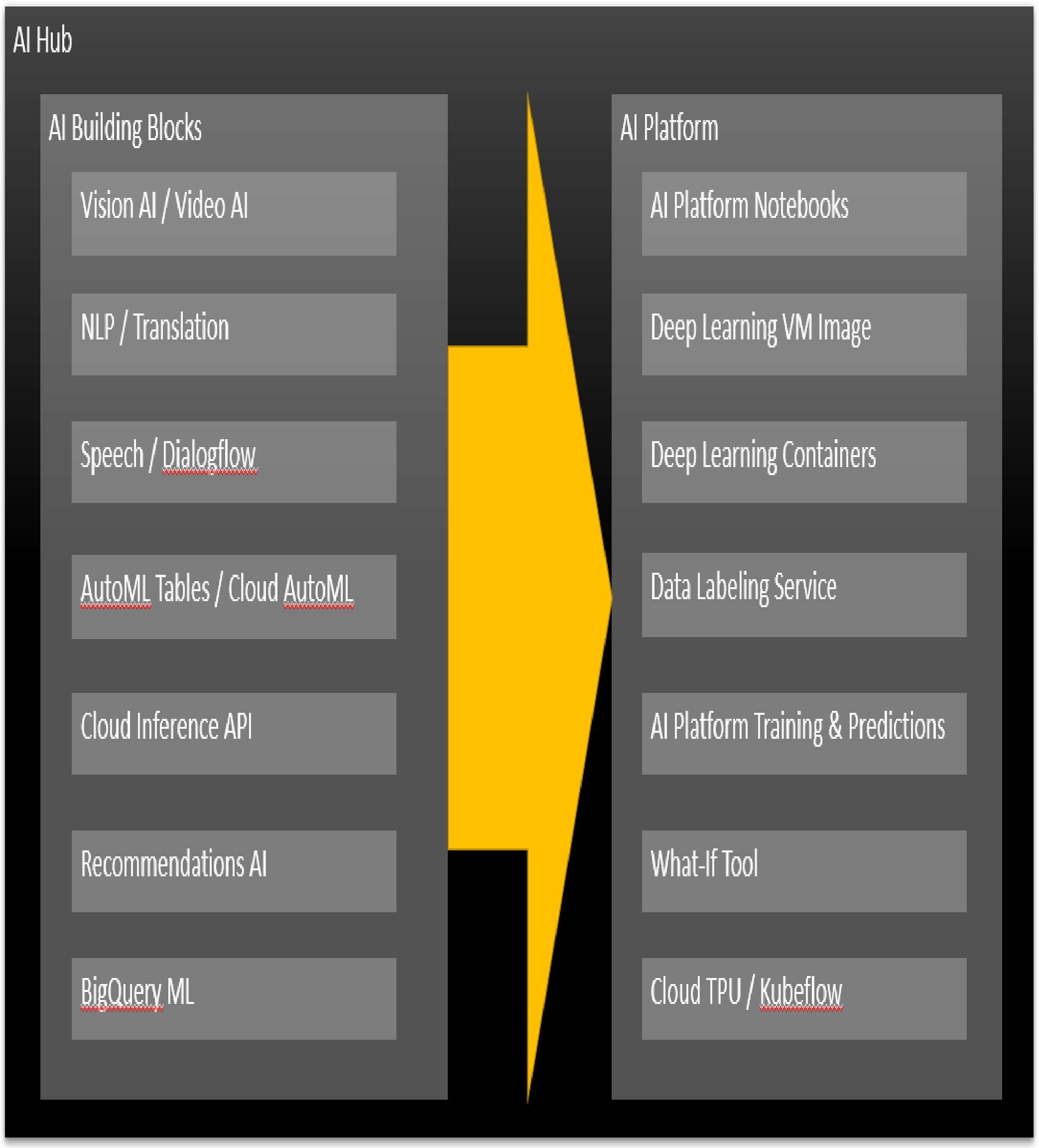
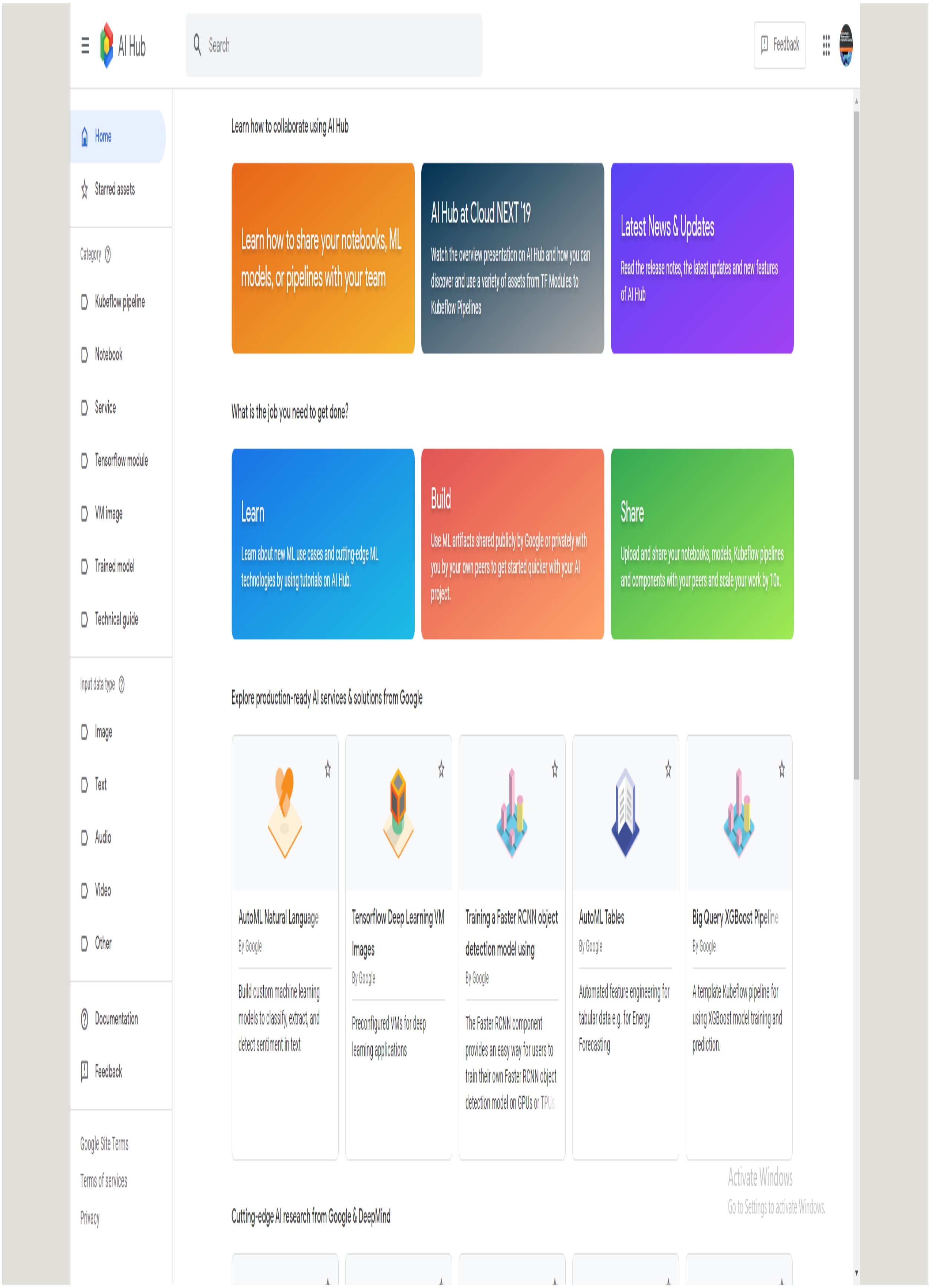


Figure 2-1. Google AI Hub, Building Blocks and Platform

THE AI HUB

Your first gateway to accessing the AI services on GCP is through the AI Hub. You can go to the hub at the following URL aihub.cloud.google.com or you just Google "ai hub". When you do go the page, you will be required to login with a Google account. If you do not have one, you will need to create one. After you sign in to your Google account you will be directed to the AI Hub page as shown in

Figure 2-2. This dashboard page will be our main entry point into the many services we use throughout this book. Before jumping into those services however we will look at the main navigation items that will be of interest.



AI Hub

Search

Feedback

Home

Starred assets

Category

Kubeflow pipeline

Notebook

Service

Tensorflow module

VM image

Trained model

Technical guide

Input data type

Image

Text

Audio

Video

Other

Documentation

Feedback

Google Site Terms

Terms of services

Privacy

Learn how to collaborate using AI Hub

Learn how to share your notebooks, ML models, or pipelines with your team

AI Hub at Cloud NEXT '19

Watch the overview presentation on AI Hub and how you can discover and use a variety of assets from TF Modules to Kubeflow Pipelines

Latest News & Updates

Read the release notes, the latest updates and new features of AI Hub

What is the job you need to get done?

Learn

Learn about new ML use cases and cutting-edge ML technologies by using tutorials on AI Hub.

Build

Use ML artifacts shared publicly by Google or privately with you by your own peers to get started quicker with your AI project.

Share

Upload and share your notebooks, models, Kubeflow pipelines and components with your peers and scale your work by 10x.

Explore production-ready AI services & solutions from Google

AutoML Natural Language

By Google

Build custom machine learning models to classify, extract, and detect sentiment in text

Tensorflow Deep Learning VM Images

By Google

Preconfigured VMs for deep learning applications

Training a Faster RCNN object detection model using

By Google

The Faster RCNN component provides an easy way for users to train their own Faster RCNN object detection model on GPUs or TPUs

AutoML Tables

By Google

Automated feature engineering for tabular data e.g. for Energy Forecasting

Big Query XGBoost Pipeline

By Google

A template Kubeflow pipeline for using XGBoost model training and prediction.

Activate Windows

Go to Settings to activate Windows.

Cutting-edge AI research from Google & DeepMind

Figure 2-2. The Google AI Hub

In Figure 2-2 on the navigation bar on the left we see 2 main headings; Category and Input data type. The Category heading represents the platform category with which you want to run an AI service or building block on. Where as the Input data type, area represents a point of entry into importing various forms of data (image, text, audio, video and other) on the cloud platform. You can then use this data in the various platform AI services or building blocks you use or develop on your own.

NOTE

Using many of the AI building blocks and platform services require authorizing a payment account. This is because these various services may be billed for use. Google does provide a rather generous amount of credits in order to test development however.

AI PLATFORM

The AI Platform represents the backend services, models or containers that host your AI Building Blocks or process data. As such, when starting to build any AI service your first task is to decide on what platform will host your service. Of course, services and platforms are interchangeable and you may transfer an AI service developed on one platform to another. However, you do still need to select a development or test platform first. We will “be” primarily be working with the Notebook platform for most coding exercises but it will be useful to cover a broad overview of all platforms below:

- Kubeflow Pipeline - Kubeflow itself is an open source project from Google. The pipeline part is about deploying apps within Kubernetes container engine. Container engines like this simplify deployment immensely.
- Notebook - represents the Google Colab notebook platform which is a derivative of Jupyter Notebooks. The Google Colab notebook will be our primary coding surface extensively for most development projects.
- Service - services come in many different forms from AI Building blocks like vision to the human data labeling service. Including the ability to enroll in potential Workshop experiments. Service in this case is perhaps misnamed in that it does not represent hosting AI services.
- Tensorflow Module - in this case Tensorflow represents the container for the model for processing data. This platform module represents state of the art Tensorflow models for image classification to natural language processing.
- VM Image - allows you to run your code, service or module on your own VM server. This is useful for when finer access control on performance and data management. However, a running VM image will consume services and therefore money. This will likely be your most expensive option for deploying AI services.
- Trained Model - this platform allows you to run a trained model, usually Tensorflow (Keras) or Pytorch in a container. This limits the model to just be used as needed and is often the most economical for fulltime production services.

While platform is a bit of a misnomer for all the functions within the AI Platform it is still helpful to understand all the potentials you can use with AI on the GCP. In the next section we look at the higher level building blocks we will focus on throughout this book.

AI BUILDING BLOCKS

Our primary focus for the rest of this book will be to look in detail at the robust commercial set of AI services Google is currently providing as blocks. We will spend considerable time on each block in detail as well as look at how to integrate multiple blocks together. Below is a summary of the various blocks, what they can be used for and in particular what section or chapter we will focus that block on:

- Vision AI - in Chapter 3 - Image Analysis and Recognition on the Cloud we will we look at how to use the Vision block to various forms of image recognition and search tasks.
- Video AI - in Chapter 4 - Video Analysis on the Cloud, we look to using the Video block to index and detect various tasks or items within video.
- NLP / Translation - in Chapter 5 - Understanding Language on the Cloud and Chapter 6 - Chatbots and Conversational AI will spend plenty of time looking at all service blocks with respect to language and speech.
- AutomML Tables - later in this chapter we being to explore the power of AutoML and how well it can build and train models effectively for simpler problems.
- Cloud Inference API - in Chapter 8 - Building Advanced AI Agents, we look to using the inference API in some advanced examples.
- Recommendation AI - in Chapter 9 - Building Integrated Cloud AI Applications we include recommendations as part

of an all inclusive sample app.

- BigQuery ML - we will address various data queries that use this block in various sections throughout the book. Starting with this chapter we will look at how to move data with BigQuery.

These building blocks will be essential elements to the many applications and models we build throughout this book. They also provide us with the greatest path to success in building real world AI applications. AI applications we can put in place to power your business or commercial applications. With that basic understanding of the platform and building blocks we can move on to working with the first platform component in the next section.

Google Colab Notebooks

The Notebook platform is a derivation of a popular open source project we often use for data science and deep learning called Jupyter Notebooks. You typically run Jupyter as a service on a development machine and then open the interface in a web browser. The version Google ported to the GCP is called Colab. Colab is essentially Jupyter Notebooks on GCP running in your browser. Except, being GCP focused it provides so much more. It allows you to seemlessly integrate with GCP data services with cloud AI services. It will be an essential driver in many of the projects we work on in this book. Therefore, we will do a few walk through exercises that show you how to build deep learning models with Keras on Colab.

BUILDING A REGRESSION MODEL WITH COLAB

The best way to learn is by doing and this is certainly the case with deep learning and AI in general. Therefore, in this section we walk through building a regression model using Keras. Keras is a high level deep learning Python library that will allow us to build our own models quickly. While we won't build many deep learning models at this level of detail it is important to understand this foundational process. That way, when something does go invariably wrong you will be able to diagnose the issue on your own. Thus avoiding technical frustrations and errors in potentially critical applications.

For this practical exercise we look at developing a Keras deep learning model for performing regression analysis on Google Colab. Notebooks are meant to be developed in stages and this will be perfect for this and many other exercises. Likewise, we will break the entire regression exercise into sub-exercises that you can reuse on your own later:

Example 2-1. Setting up the Environment

- Open a web browser and point it to colab.research.google.com. If you have not signed up for GCP and the AI services you will need to do so now.
- Figure 2-3 shows the dialog you will face when going directly to Colab. Click the New Python 3 Notebook button to create a new notebook.

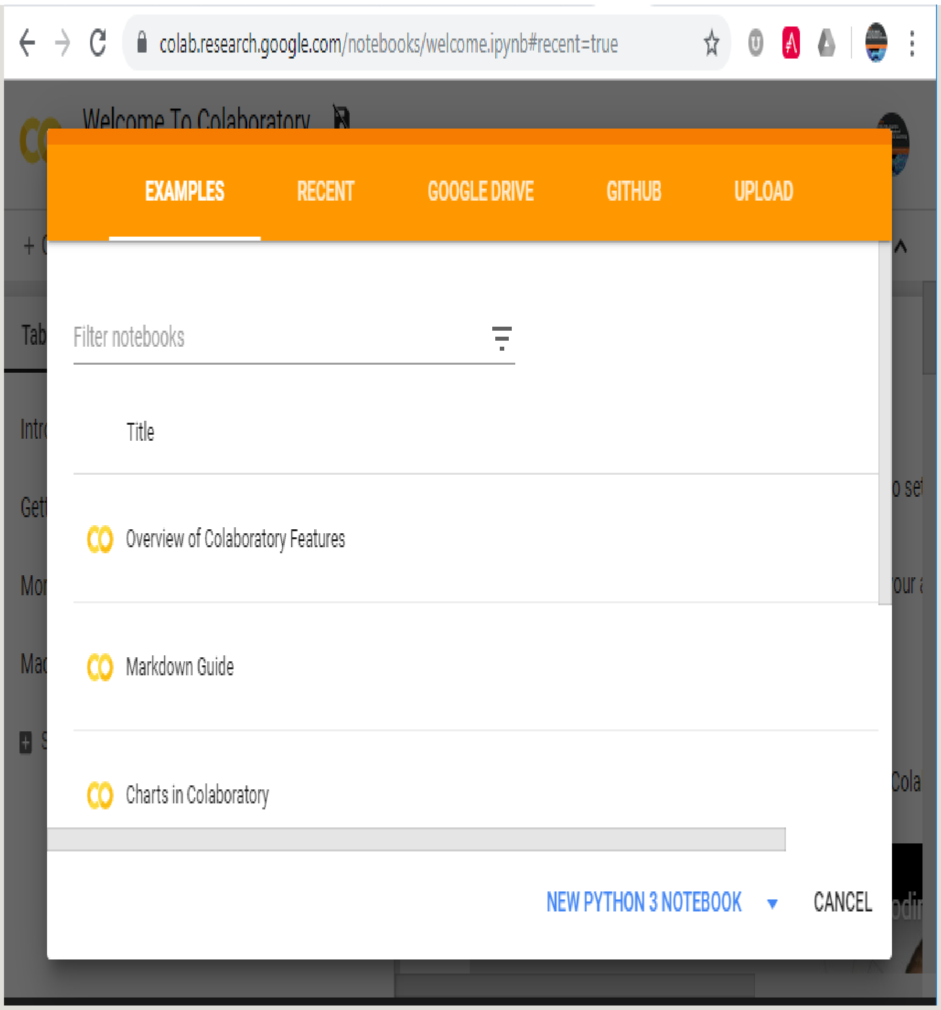


Figure 2-3. Starting Google Colab

- Inside this empty notebook will be one code cell. Each cell can contain code or text. Enter the following code in the cell:

Example 2-2.

```
In [1]: !pip install seaborn
```

- The ! before a command routes that command to the shell. Thus, this command does a pip install of the seaborn library for plotting.
- After the code is entered click the black arrow beside the cell in order to execute the code. You should see the command

execute and the seaborn library get installed on the shell.

- Next, click the + Code button in the header to add a new code cell and enter the following setup code:

Example 2-3.

```
from __future__ import absolute_import, division, print_function,
unicode_literals

import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

- The above code just does a number of imports and then sets up the Tensorflow backend. Tensorflow is the deep learning engine that powers Keras. You can build models directly in Tensorflow but that requires a greater understanding of the basic math than we previously covered. As such, we will stick with Keras.

- Make sure the cell is error free and run it by pressing the black arrow. The only output from this cell is the version of Tensorflow you just installed.

You may find yourself using the first sub-exercise over and over again as you work your way through building and training models. Now with the environment setup we want to move on the next sub-exercise, importing data.

Example 2-4. Importing Data

- Continuing from the last exercise notebook Exercise 2-1 create a new cell and enter the following code:

Example 2-5.

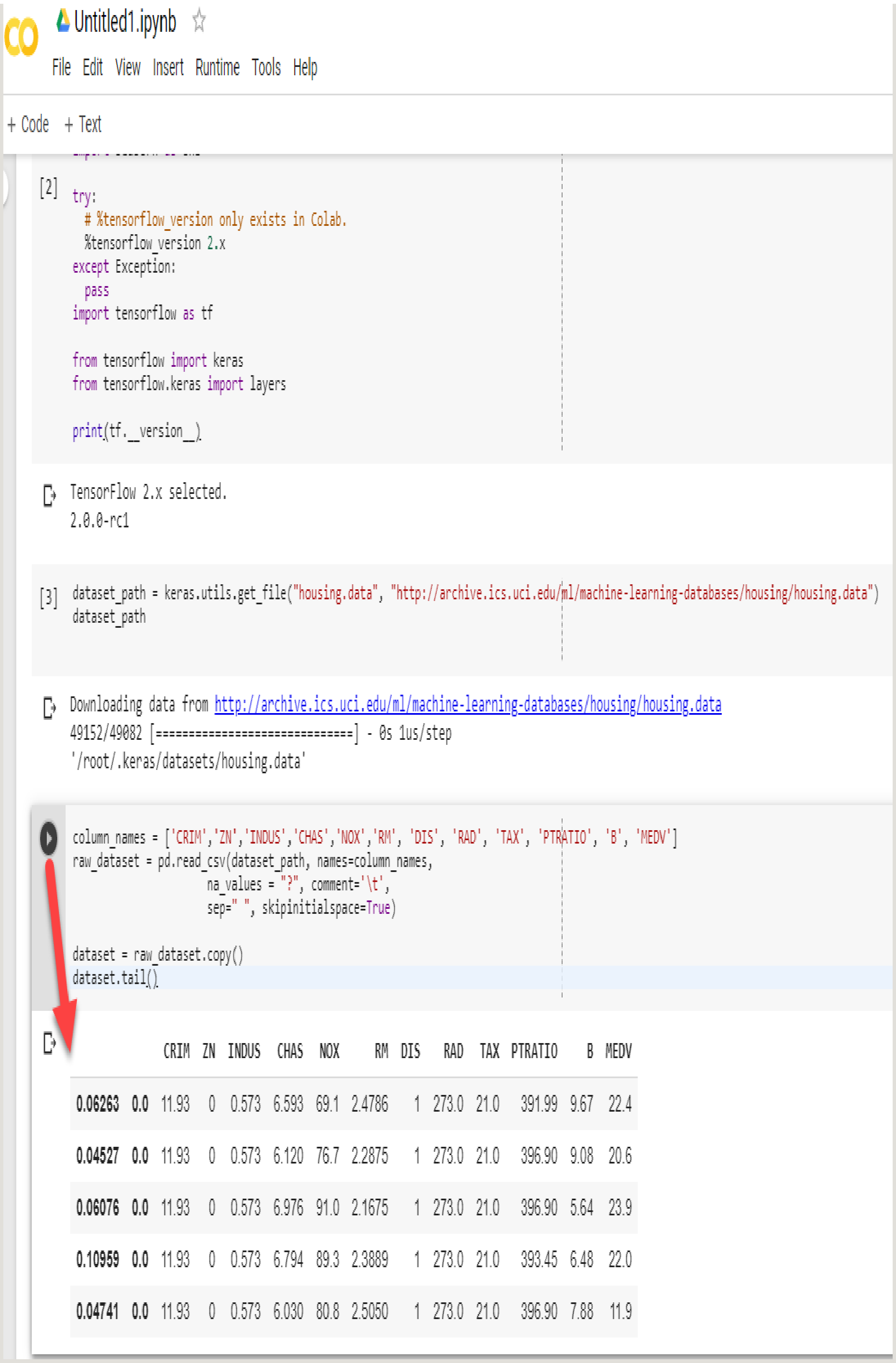
```
dataset_path = keras.utils.get_file("housing.data",
"http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data")
dataset_path
```

- The above code sets a variable with the dataset_path. The path is where a Keras utility will get the file from an sample data archive and download it for our regression experiment.
- Run the code by pressing the black arrow. You will see the utility download the file and then print the path where the file is stored local to the notebook.
- We next need to load our data into a data container called a data frame. The Pandas library we imported in setup will allow us to load and parse the data we just downloaded into a data frame. In the next section of code we label the columns and parse the data into a data frame:

Example 2-6.

```
column_names = ['CRIM','ZN','INDUS','CHAS','NOX','RM', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'MEDV']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
```

- The first line of code setup the column names. These names were obtained by a sister file in the same housing folder on the archive site.
- Next, Pandas pd uses read_csv to parse the downloaded dataset given the rules in the function call.
- Finally, the code completes by doing a copy and then outputs the tail of the data frame.
- Execute the code by pressing the black arrow. The code will run and you should see the output as shown in Figure 2-4.



```
[2]  try:
       # %tensorflow_version only exists in Colab.
       %tensorflow_version 2.x
     except Exception:
       pass
     import tensorflow as tf

     from tensorflow import keras
     from tensorflow.keras import layers

     print(tf.__version__)
```

```
TensorFlow 2.x selected.
2.0.0-rc1
```

```
[3]  dataset_path = keras.utils.get_file("housing.data", "http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data")
     dataset_path
```

```
Downloading data from http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data
49152/49082 [==============================] - 0s 1us/step
'/root/.keras/datasets/housing.data'
```

```
column_names = ['CRIM','ZN','INDUS','CHAS','NOX','RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MEDV']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
```

```
                      CRIM  ZN  INDUS  CHAS   NOX    RM   DIS  RAD    TAX  PTRATIO      B  MEDV
0.06263  0.0  11.93  0  0.573  6.593  69.1  2.4786  1  273.0  21.0  391.99  9.67  22.4
0.04527  0.0  11.93  0  0.573  6.120  76.7  2.2875  1  273.0  21.0  396.90  9.08  20.6
0.06076  0.0  11.93  0  0.573  6.976  91.0  2.1675  1  273.0  21.0  396.90  5.64  23.9
0.10959  0.0  11.93  0  0.573  6.794  89.3  2.3889  1  273.0  21.0  393.45  6.48  22.0
0.04741  0.0  11.93  0  0.573  6.030  80.8  2.5050  1  273.0  21.0  396.90  7.88  11.9
```

Figure 2-4. Executing code in Colab

The archive we pulled the data from has plenty of sample datasets you can revisit on your own or as part of other learning. Code for the above import method is just one way of getting data into Colab, we will review a few other methods throughout this chapter.

After importing the data our next step as a data scientist would be to analyze and clean the data. Even as AI practitioners of deep learning we are still bound by rules of expectation and probability to understand our data. While deep learning purists prefer to consume any and all data, our current method will be to analyze and understand the data. We also do this because the dataset is relatively small, as we will see. In the next exercise we look at techniques for cleaning, splitting and inspecting the data.

Example 2-7. Cleaning, Splitting and Inspecting Data

- Since our dataset is quite small we will want to clean any null or not a number values from it. We can do this by creating a new cell and entering the following code:

Example 2-8.

```
dataset.isna().sum()
```

- This will identify the number of null or NaN values in the dataset. Assuming there are some we can remove them by running the following code:

Example 2-9.

```
dataset = dataset.dropna()
```

- After cleaning the data we need to move on to splitting the data into a training and test set. Recall from the first chapter we split our data up like this in order to reaffirm our results are not over or under-fitting.

- Create a new cell and enter the following code:

Example 2-10.

```
train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

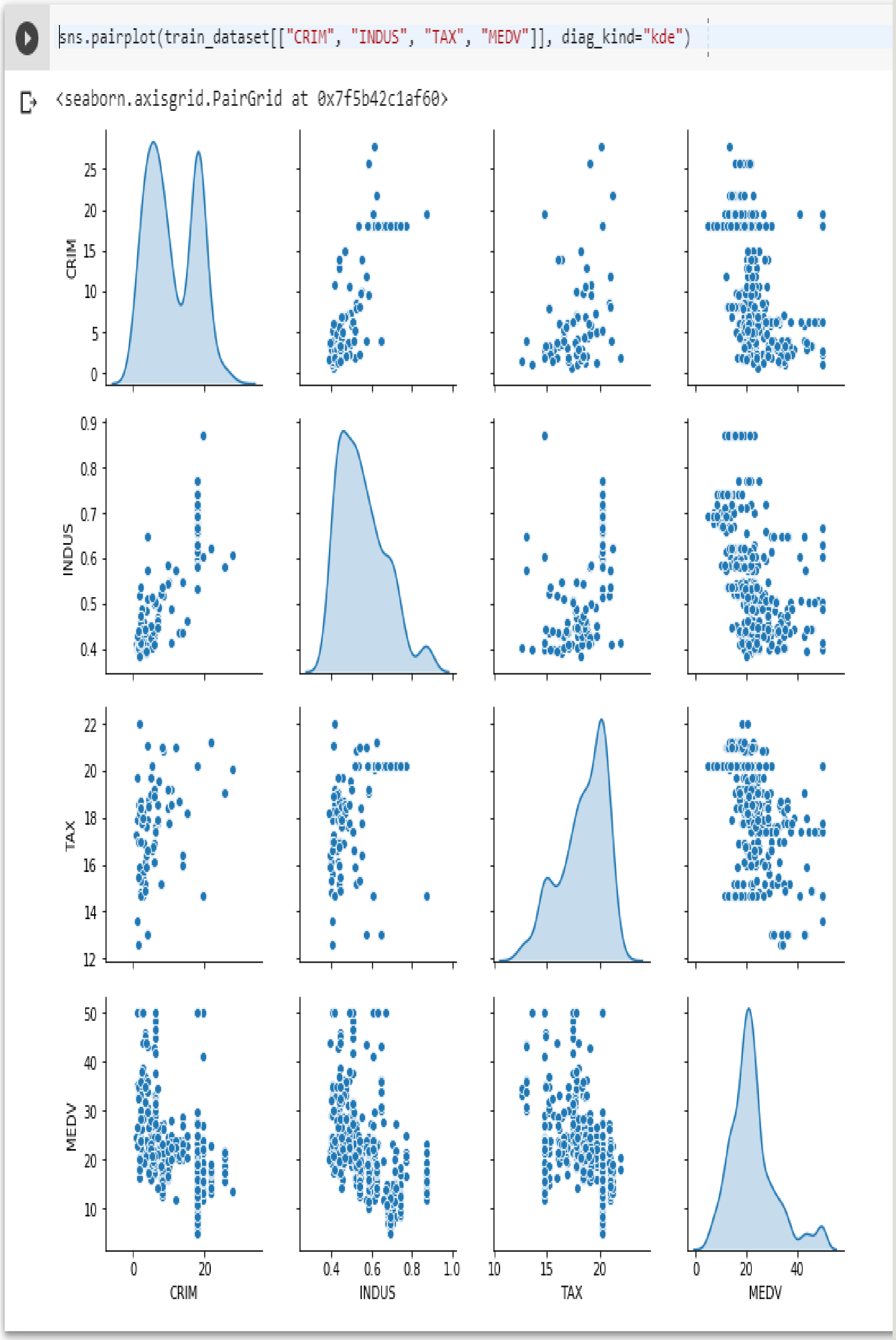
- This code splits up the dataset randomly into a training section, consisting of .8 or 80% of the data. With the remainder of the data going to the test_dataset. Run this cell to break up the data in test and training sets.

- We can now inspect the data for any obvious characteristics we should be aware of. Run the following code in a new code cell:

Example 2-11.

```
sns.pairplot(train_dataset[["CRIM", "INDUS", "TAX", "MEDV"]], diag_kind="kde")
```

- Running the last cell will generate the correlation cross plot in Figure 2-5. A correlation cross plot shows the relationship between various features in a dataset. In data science we often remove features that show little or no relationship with the target feature or sibling features.



```
sns.pairplot(train_dataset[["CRIM", "INDUS", "TAX", "MEDV"]], diag_kind="kde")
```

```
<seaborn.axisgrid.PairGrid at 0x7f5b42c1af60>
```

Figure 2-5. Correlation Cross Plot

- Next, we want to review the basic statistics of each feature using the following code:

Example 2-12.

```
train_stats = train_dataset.describe()
train_stats.pop("MEDV")
train_stats = train_stats.transpose()
train_stats
```

- Run the last cell and the basic descriptive statistics describing each feature will be output. This is something all data scientists will perform as a basic analysis step. Deep learners may perform this step in order to understand the variability of the data. However, a deep learning purist may still ignore these values.

NOTE

Deep learning purists will prefer to consume all data even with the realization that not all data may be relevant. Why? Well purists believe that we need to eliminate all human bias in any AI experiment. Now the purists still clean the data but they use other deep learning embedding and encoding techniques. These techniques learn the relevance from the data itself and are thus becoming more powerful all the time. Readers interested in this subject should search for autoencoding with deep learning. Keep in mind though that even a purist would only consider this method given a sufficient size of variable data.

With the data cleaned, split and inspected we can move on to preparing the data. That is, we want to identify the parts of the data that represent our target value. In our regression problem we are selecting one column or feature as our target. We then need to split this value off into labels. Let’s see how that is done in the next Exercise 2-4.

TIP

If at any time you find that cells are not running correctly it may be because your code is out of order. Meaning you may have run cells before you should have. It is often better to rerun previous cells before running new cells. In fact, if the notebook seems to start failing just go back to the beginning and start again.

Example 2-13. Preparing Data

- Since we are using supervised learning, this requires us to label the data. We label and split the target columns from the original test and training data sets with the following code:

Example 2-14.

```
train_labels = train_dataset.pop('MEDV')
test_labels = test_dataset.pop('MEDV')
```

- The target feature in this regression problem will be MEDV and we essentially pop it off the data frame and into a single column data frame with the above code.
- Run the code and the new variables will set.
- Next we want to normalize the data. We normalize a feature by subtracting from the mean and then dividing by the standard deviation. The end result produces data within one standard deviation of the mean. We do this so that not a single feature overrides other features. There are many other forms of normalization we will use on other examples.
- We will normalize the data by creating a new function and then executing that in the same cell, with the following code:

Example 2-15.

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

- You can confirm this step was taken by outputting the tail from either test or train data frames using the following code:

Example 2-16.

```
normed_train_data.tail()
normed_test_data.tail()
```

- Running that code will output the tail values from the respective data frame.

With the data now prepared we can move onto building the model or deep learning network in Keras that we will train to solve our problem. In the next Exercise 2-5, we build a neural network model that we feel is suited to solving our problem.

Example 2-17. Building the Model

- In order to explain the construction of the next function well we are going to break this cell up into code sections starting first with the function definition:

Example 2-18.

```
def build_model():
```

- Simple enough, then the first line will build the entire model with this code:

Example 2-19.

```
    model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=
[len(train_dataset.keys())]),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])
```

- Make sure this line is indented as it needs to be part of the function. This line creates an entire Sequential fully connected network that has 64 input neurons of input_shape = length of our input parameters. That is followed by a second layer of 64 neurons, a hidden middle layer. The third and final reduces to one output neuron, that will output our learned result.

- Next, we setup the optimizer, in this case we use the RMSProp optimizer with the following code:

Example 2-20.

```
    optimizer = tf.keras.optimizers.RMSprop(0.001)
```

- Again, make sure the line is indented. We could of course used a variety of other optimizers on this problem.

- The last step is to compile the model with the optimizer and a set of metrics using the following code:

Example 2-21.

```
    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mae', 'mse'])
```

- Compiling is where a model is setup for the backpropagation process. This is where the automatic differentiation calculates the required functions for learning.

- Finally, after compiling we return the model with the following:

Example 2-22.

```
    return model
```

- Finish entering the code and then execute the cell. With the function built we can now create a model with the following code:

Example 2-23.

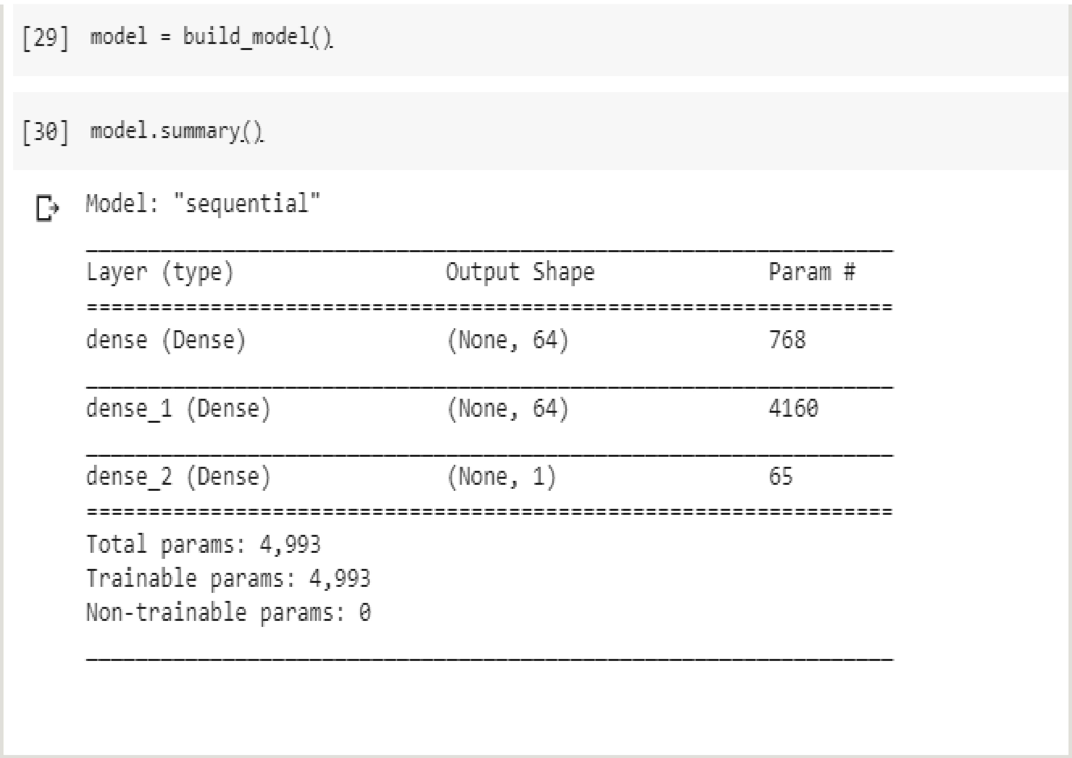
```
model = build_model()
```

- Run and execute that last line of code in order to build and set the model.
- After the model is set you can output a summary of the model by running the following code in a new cell:

Example 2-24.

```
model.summary()
```

- The summary output of this model is shown in Figure 2-6. In the figure you can see each layer in order of processing as well as the size and type. The Param # represents the number of weights in each layer. Notice that for this problem we are using almost 5000 weights.



```
[29] model = build_model()

[30] model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                768
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 4,993
Trainable params: 4,993
Non-trainable params: 0
_________________________________________________________________
```

Figure 2-6. Summary of Sequential Model

With the network model built we can move on to training it. Recall we train a network by showing it batches of labeled samples. The goal of the network will be minimize the error on the loss function, which in this case is MSE or mean squared error. In the next exercise we train the model to learn from the data.

Example 2-25. Training a Network

- Before we start training we want to do a simple test and run a number of samples through the predict function of the model with the following code:

Example 2-26.

```
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result
```

- That code extracts 10 rows of data and pushed that through the predict function. Then the results are output. Run the cell and look at the results. The results will not make much sense yet because the model is yet to be trained.
- We will be able to train the model in one line of code but we will want to also see the training progress. In order to do that we want to create a new small helper class to output progress.
- Enter the following code in a new cell and execute it:

Example 2-27.

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')
```

- Next, we define the number of epochs or training iterations we want to train the model on. Recall that one training iteration is one full pass of all data through the network. Create a new cell and execute it with the following code:

Example 2-28.

```
EPOCHS = 10000
```

- The important thing to understand here is that one EPOCH is one full pass of all training data. If you find that training is taking to long you can reduce this number to 1000.
- We can now train the model by entering the following code in a new cell executing it:

Example 2-29.

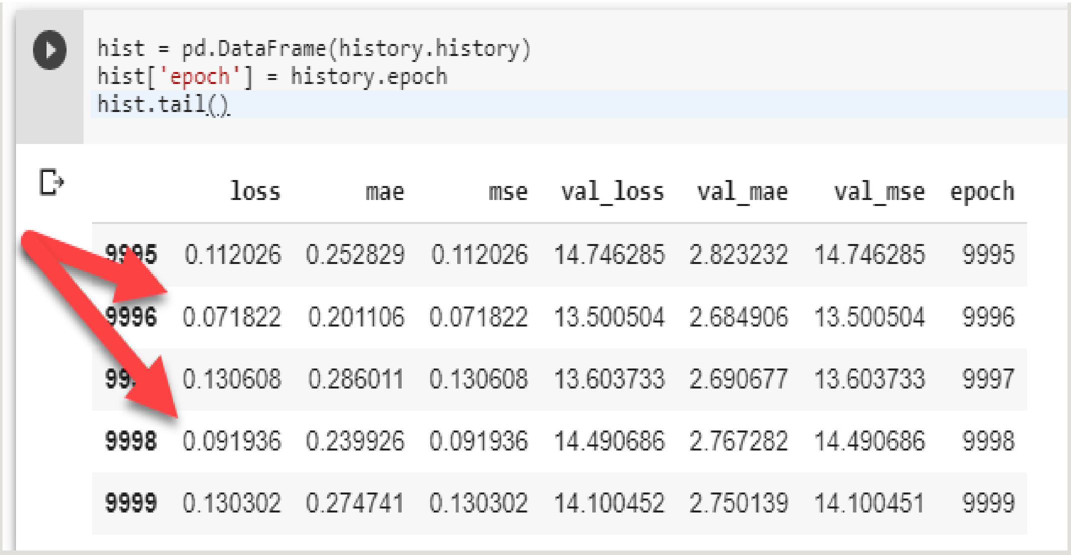
```
history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])
```

- This code sets up the model for training and feeds it the data normed_train_data and train_labels labels. We can also see that the model is running with a validation split. That provides us with another step in tracking our network progress.
- As the cell runs you will see a dot print for each 100 epochs, refer back to the PrintDot class if you need to understand this.
- At the end of this training session we want to review some of the history by running code in a new cell:

Example 2-30.

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

- Figure 2-7 shows the tail output from the training history. The goal of any training is to see our loss function converge. However, the output in Figure 2-7 shows that the loss function is not converging. Instead it is wobbling about which is likely being caused by a vanishing gradient of some type.



```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

```
          loss       mae       mse   val_loss   val_mae    val_mse  epoch
9995  0.112026  0.252829  0.112026  14.746285  2.823232  14.746285   9995
9996  0.071822  0.201106  0.071822  13.500504  2.684906  13.500504   9996
9997  0.130608  0.286011  0.130608  13.603733  2.690677  13.603733   9997
9998  0.091936  0.239926  0.091936  14.490686  2.767282  14.490686   9998
9999  0.130302  0.274741  0.130302  14.100452  2.750139  14.100451   9999
```

Figure 2-7. Output of Training History

We trained our model, but as we've seen there is a hint that the model wasn't converging well. What we need now is a way to explore the training history through a plot. Fortunately, we have already setup a plotting library in this example and can do just that in the next exercise.

Example 2-31. Training a Network

- Create a new cell and enter the function code below:

Example 2-32.

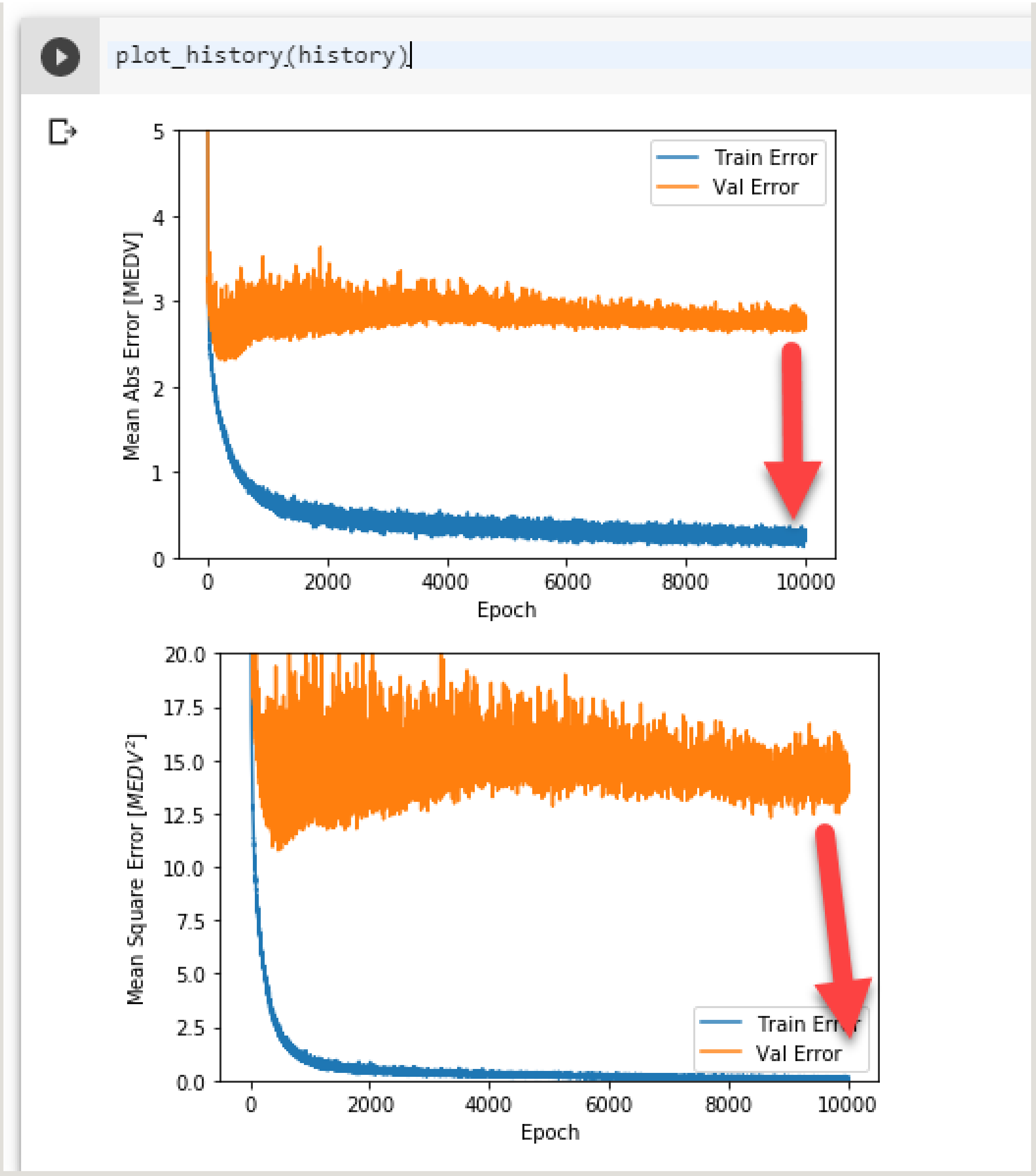
```python
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [MEDV]')
  plt.plot(hist['epoch'], hist['mae'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mae'],
           label = 'Val Error')
  plt.ylim([0,5])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$MEDV^2$]')
  plt.plot(hist['epoch'], hist['mse'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mse'],
           label = 'Val Error')
  plt.ylim([0,20])
  plt.legend()
  plt.show()
```

- This large block of code just outputs some plots that will display how well the model fits to training and an internal validation set. Recall, that we also set a .2 or 20% rate on internal validation when building the model. That means 20% of the data will be withheld for auto validating the model.
- Run the function code and then create a new cell and execute it with the following code:

Example 2-33.

```
plot_history(history)
```

- Running that block will output Figure 2-8 which shows that our model does indeed have a problem although likely not the one we may have expected.



```
plot_history(history)
```

Figure 2-8. Plot of Training History Errors

We can see now that our model does indeed have issues with respect to over-fitting on the data. Notice in Figure 2-8 how the errors on the training data, the bottom line, are much less than the errors on the validation line. Ideally, we want both lines to occupy the same space. Meaning they both have similar errors which means the model is not over or under-fitting. Fixing the model in this case just takes experience understanding how networks fail. Experience that can take months or years which is the reason we want to use those fully developed models later. The next exercise will show a few techniques to fix our current problem.

Example 2-34. Tuning a Network

- From the menu select Runtime → Run All. This will rerun all the cells and output the results again. This is also an excellent way to reset your environment if you get lost on which cell you were executing.
- Locate the cell in the notebook with the function to build the network model. Alter the function to the following code:

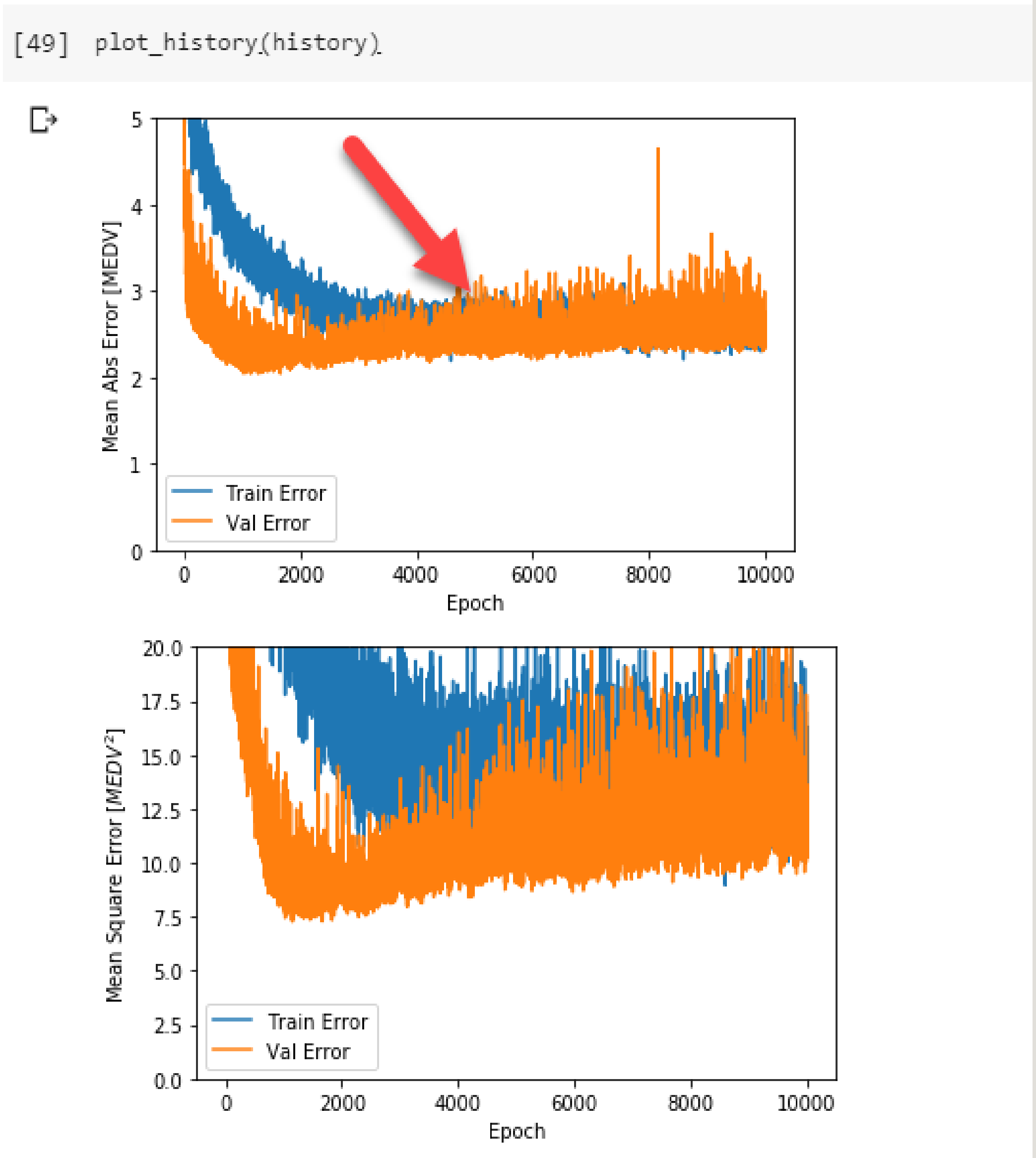
Example 2-35.

```
def build_model():
  model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=
[len(train_dataset.keys())]),
    layers.Dropout(.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(.5),
    layers.Dense(32, activation='relu'),
    layers.Dropout(.5),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model
```

- The code has been modified in 2 ways:

    - First the layer architecture itself was modified into a more funnel shape. As the top layer goes from 128 neurons down to the last hidden layer of 32 neurons before being output to 1.
    - Second, we added a special kind of layer called Dropout. Dropout layers, dropout or randomly turn off a percentage of neurons set by the dropout rate. In this setup we add dropout to every layer connection and set it to .5 or 50%. This means that half the neurons will fail to fire during training. The end effect is that the network better generalizes.

- After you update that code it is best to execute the entire notebook again using Runtime → Run All menu command.
- Running the notebook will produce Figure 2-9 and you can now see how the training and validation errors converge but then diverge again after that. Notice how the validation set now is better predicted at least initially. This is likely an indication of too much dropout. Can you fix it?



```
[49] plot_history(history)
```

Figure 2-9. Updated Plot of Training History Errors

In the last exercise we've seen how the convergence of training errors can differ based on how the model generalized. Eventually the model did reach some convergence but then deviated from over training. We can fix this in the next exercise by introducing an auto stopping mechanism.

Example 2-36. Auto-stopping Training

- Since most of our models will be using cloud resources it is important to make sure we avoid over training. Over training is lost resources and lost resources also mean lost money. This can certainly be the case when working with more complex data sets and AI tasks.

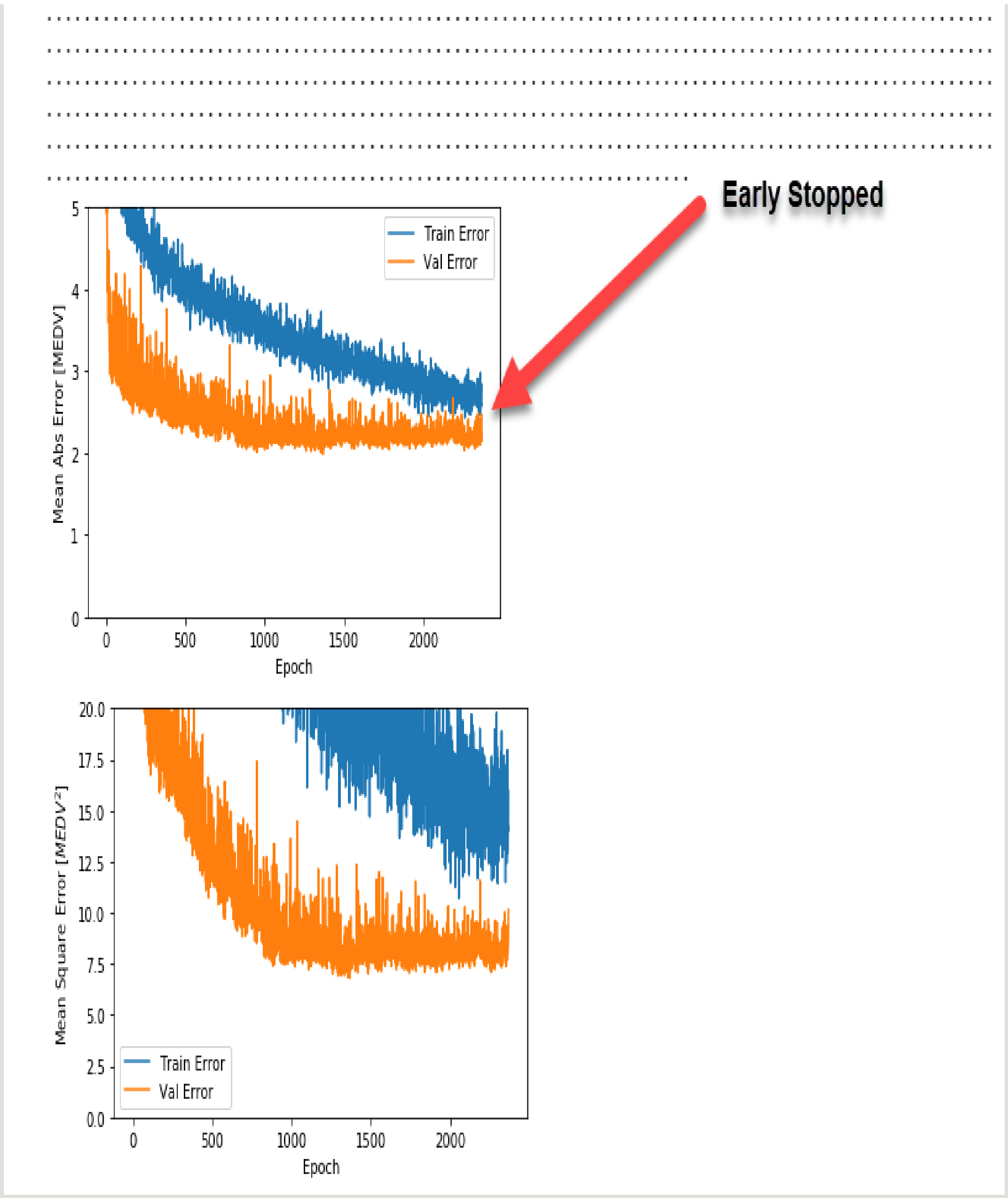
- Enter the following code in a new cell at the bottom of the notebook:

Example 2-37.

```
model = build_model()
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=1000)
history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split = 0.2, verbose=0, callbacks=[early_stop,
PrintDot()])
plot_history(history)
```

- The above code rebuilds the model and then using keras.callbacks uses a function call EarlyStopping to put a watch in place. The callback watch will monitor the training every number of iterations set by the patience parameter. In this case it is set to 1000.

- We then fit the model again using the new callback early_stop and then plot the history as shown in Figure 2-10.



```
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................
```

Figure 2-10. Example of Early Training Stopping

- Run the code and you should see the output generated in Figure 2-10.

With our model corrected and not over training we can move on to doing predictions with our reserved test data set. Running another test against the model this way assures us that the model is robust enough for the real-world. In the next exercise we look at how to make predictions and do a final evaluation of the trained model.

[[Exercise 2-10, Evaluating a Trained Model]]

Example 2-38.

- When we are satisfied with training the model it is time to do a proper evaluation and final assessment to make sure it is ready for the real world. We can do that by running evaluate on the model with the following code:

Example 2-39.

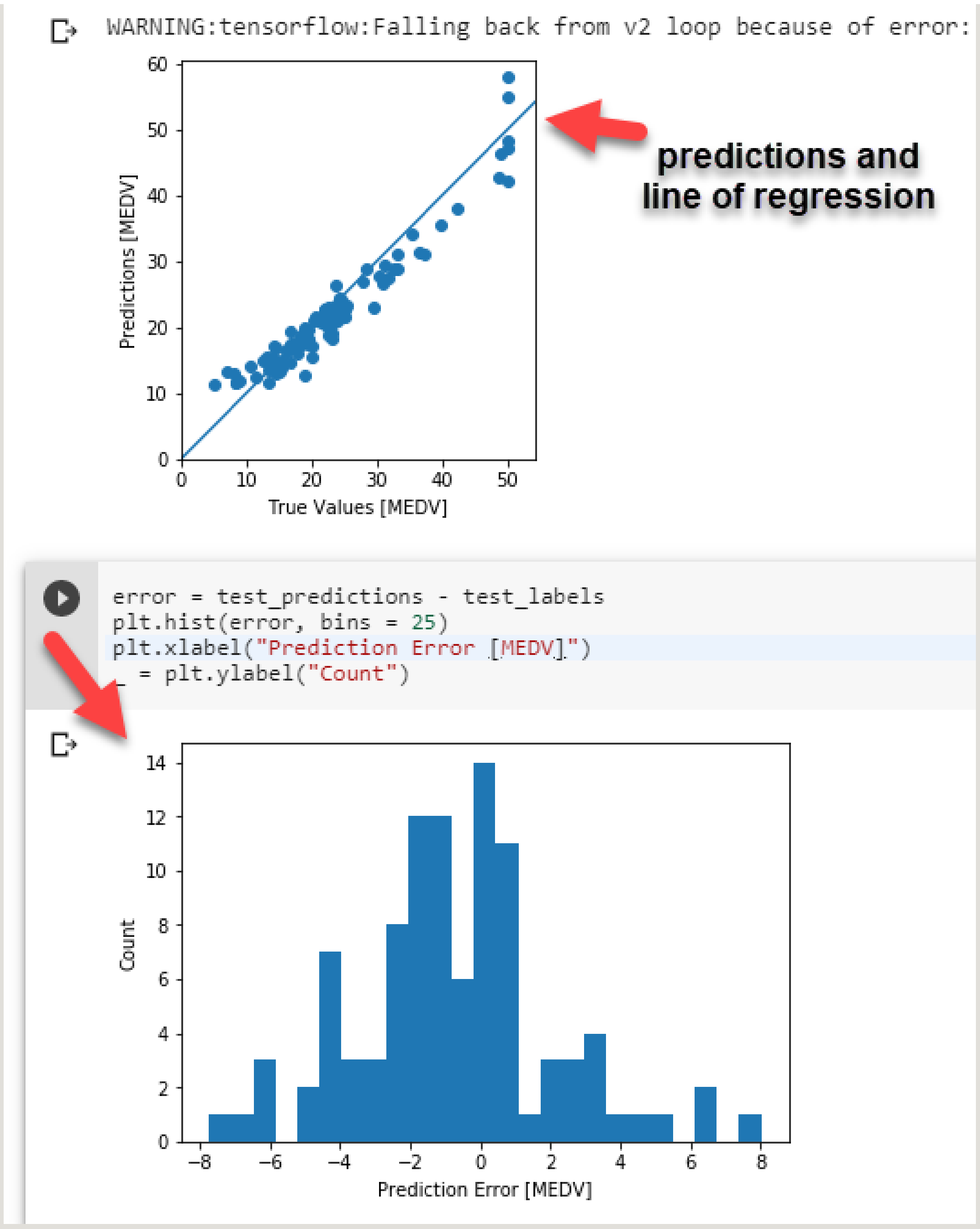
```
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)

print("Testing set Mean Abs Error: {:5.2f} MEDV".format(mae))
```

- Run that code in a new cell at the end of the notebook. Notice that we run evaluate against the test data and labels. The output will show the MAE of the training error in MEDV or medium household value against the test data.
- We can go a step further and plot those predictions by running the test data through the model and then plotting the results. This can simply be done with the following code:

Example 2-40.

```
test_predictions = model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MEDV]')
plt.ylabel('Predictions [MEDV]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```

- The first line in the above code is where model.predict is called against the normed_test_data. Then the data is plotted with the plot code. The top plot in Figure 2-11 shows the predicted output with a regression line through it.



```
WARNING:tensorflow:Falling back from v2 loop because of error:
```

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MEDV]")
_ = plt.ylabel("Count")
```

Figure 2-11. Evaluating the Model

- Finally, we want to get a sense of the distribution of the errors. Not every model we develop will be perfect and it is important we understand how well a model predicts. We can do that by generating a histogram of errors with the following code:

Example 2-41.

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MEDV]")
_ = plt.ylabel("Count")
```

- Enter that code in a new cell and run it. You should see the results as shown in the bottom of Figure 2-11. The output is not as smooth as we like because of the small dataset.

At this point your decision comes down is the model good enough for what I or my organization needs to do. If so, you make the journey to commercialize your new model. Since the model is already on the cloud that should be fairly straightforward. Of course if the model is not good enough then it is back to training and evaluation.

For this section we learned to build and tune a model. In order to tune our model we first altered the network architecture and then used a special layer called Dropout. Dropout layers randomly dropout neurons between activations and this allows for better generalization across data. Tuning networks can be time consuming, tedious and many times it feels like a black art. There have even been deep learning support groups formed to discuss these problems and alleviate frustrations. Of course this is nothing new to Google as having worked in this space for over a decade. Which is why Google has invested so heavily into AutoML and it is the first AI building block we will cover in the next section.

AutoML Tables

Anyone who has worked developing models, knows that there are countless variations and ways to solve a problem. As we have seen the plethora of network architecture and training options can not only intimidate developers, it can leave them indecisive. Unable to decide on the best network and just going back and forth between configurations for weeks or even months. In order to address this problem a new form of meta or training the trainer was introduced, called AutoNL. AutoML is essentially machine learning on top of machine learning. Which is great because it allows us to try various configurations automatically.

NOTE

AutoML Tables on GCP currently run at about $20 per hour. When training AutoML you must train in increments of an hour and the range is from 1-72 (1 hour to 3 days). Since the AutoML service is expensive to run it is suggested you only run the exercises as necessary in this section.

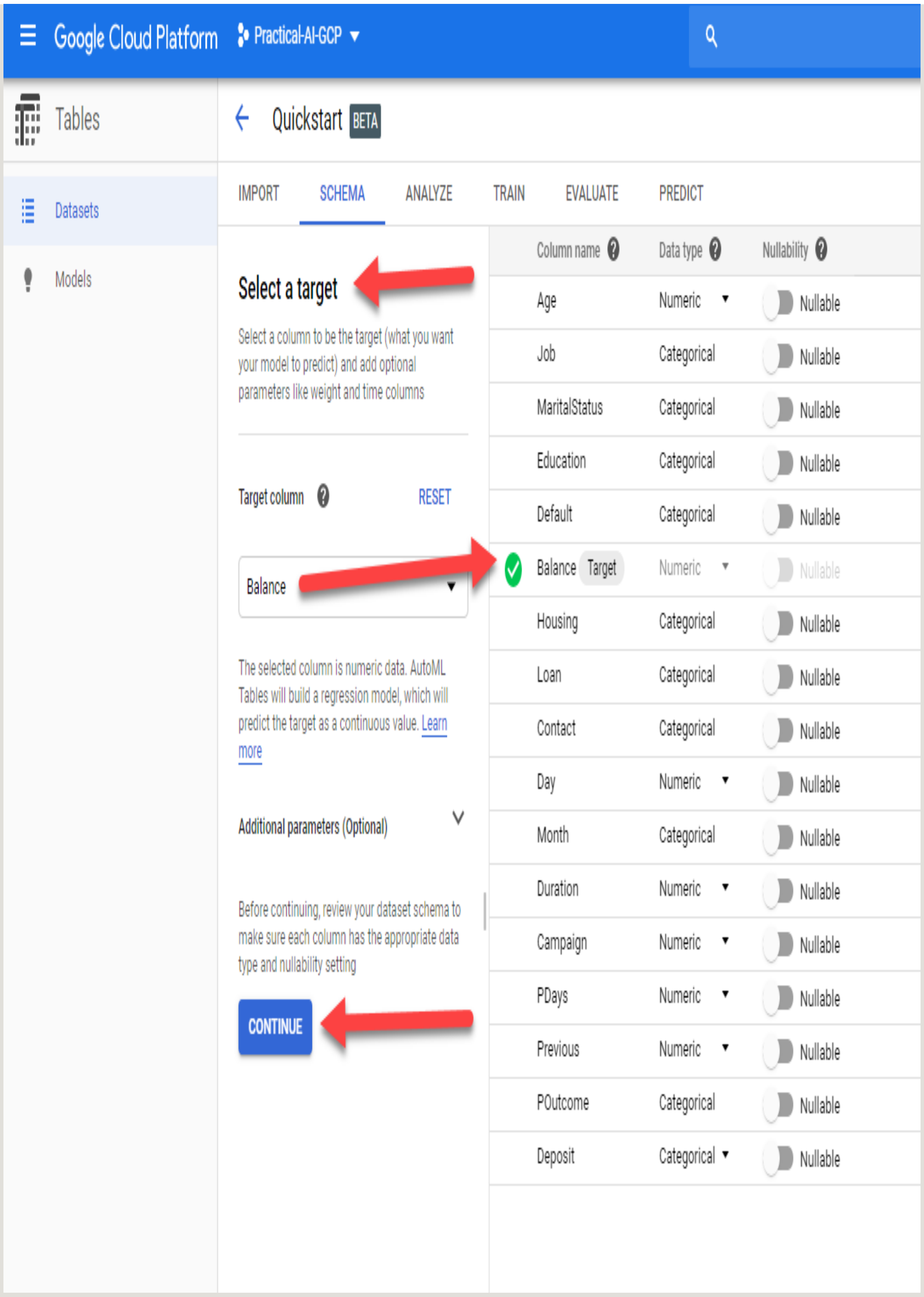
At this time AutoML Tables is designed to process tabular data for either building models automatically to perform regression or classification. In the next exercise we see how to setup and use AutoML Tables to perform regression.

Example 2-42. Developing an AutoML Model

- Navigate your browser to *https://console.cloud.google.com/automl-tables* or you can find a link on the main dashboard page.
- The start of an AutoML workflow requires you to pick or import a prepared dataset. We will learn how to prepare data

later but for now just use the Quickstart dataset. This is likely the only dataset showing on the interface.

- After you click on the Quickstart dataset a page will load showing Figure 2-12, the data schema. The Schema tab allows you to review the features in the dataset and determine what target to choose.



Select a target

Select a column to be the target (what you want your model to predict) and add optional parameters like weight and time columns

Target column RESET

Balance

The selected column is numeric data. AutoML Tables will build a regression model, which will predict the target as a continuous value. Learn more

Additional parameters (Optional)

Before continuing, review your dataset schema to make sure each column has the appropriate data type and nullability setting

Column name	Data type	Nullability
Age	Numeric	Nullable
Job	Categorical	Nullable
MaritalStatus	Categorical	Nullable
Education	Categorical	Nullable
Default	Categorical	Nullable
Balance Target	Numeric	Nullable
Housing	Categorical	Nullable
Loan	Categorical	Nullable
Contact	Categorical	Nullable
Day	Numeric	Nullable
Month	Categorical	Nullable
Duration	Numeric	Nullable
Campaign	Numeric	Nullable
PDays	Numeric	Nullable
Previous	Numeric	Nullable
POutcome	Categorical	Nullable
Deposit	Categorical	Nullable

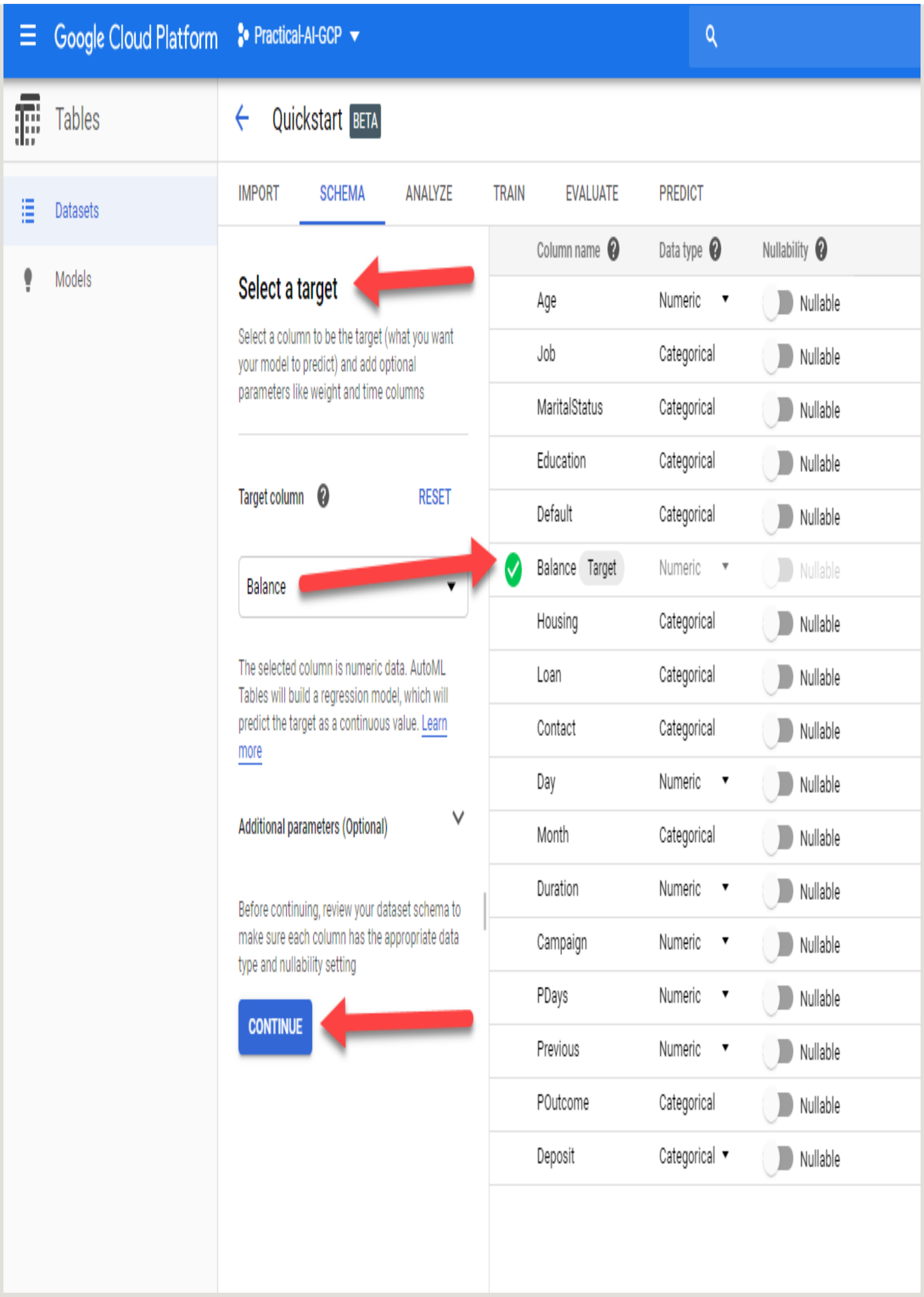
Figure 2-12. Examining the Schema

- Select the Balance as the Target as shown in Figure 2-12 and then press Continue.

- This will open the Analyze tab and run a basic descriptive statistics on all the column features. Notice that in this case some of the features are categorical. Categorical features are based on classes or categories.
- Clicking on any categorical feature will display a summary of the categories it consists of. Normally, we need to break down categorical values into numeric values in some manner. This is all taken care of us with AutoML tables.
- Select the Train tab and click the TRAIN MODEL button. This will prompt you to enter a few additional parameters. For this example we will set the parameters as follows:
    - Model name = *Regression_Balance* - minus the quotes, is the name of our model. Use a unique descriptive name for this value.
    - Training budget = 1 - for 1 hour of training time. AutoML training, at time of writing, was going for $20 US per hour. Not to worry though, training this example will only take minutes and unless you have been reckless you should have plenty of Google credits remaining.
    - Input feature selection - this allows you to turn off features/columns you many not want to train this model on. This is sometimes useful when comparing multiple models. Leave this as is, the features we have will all work for this model.
    - Advanced options - expand this section to view advanced parameters.
        - Early stopping - ON, be sure this option is on. This makes sure the model will stop if it has training all that it feels it can. This

option should be on by default but just be sure.

- When all options are set click the Train Model button. This will start training the model. Since we set training for an hour, expect to wait an hour for the results to return. Yep, an hour.
- Figure 2-13 shows the results returned from one hour of training on the example data. Your results may vary slightly but likely not by much.



Select a target

Select a column to be the target (what you want your model to predict) and add optional parameters like weight and time columns

Target column: Balance

The selected column is numeric data. AutoML Tables will build a regression model, which will predict the target as a continuous value. Learn more

Additional parameters (Optional)

Before continuing, review your dataset schema to make sure each column has the appropriate data type and nullability setting

Column name	Data type	Nullability
Age	Numeric	Nullable
Job	Categorical	Nullable
MaritalStatus	Categorical	Nullable
Education	Categorical	Nullable
Default	Categorical	Nullable
Balance Target	Numeric	Nullable
Housing	Categorical	Nullable
Loan	Categorical	Nullable
Contact	Categorical	Nullable
Day	Numeric	Nullable
Month	Categorical	Nullable
Duration	Numeric	Nullable
Campaign	Numeric	Nullable
PDays	Numeric	Nullable
Previous	Numeric	Nullable
POutcome	Categorical	Nullable
Deposit	Categorical	Nullable

Figure 2-13. Example Generated Model

- In Figure 2-13 we can see 3 measures of error, MAE, RMSE and R^2 or R squared. We learned earlier that R squared can be deceptive in some cases, is this such a case? MAE seems quite low but consider the RMSE of around 3000, or around 900 000 MSE. Remember RMSE is the root of the MSE.

- Click back to the Analyze tab and review the basic stats for the Balance feature again. Find the Standard deviation column and review the value. The value for Balance is around 3 million. Which means our error of around 9 hundred thousand or a million is less than 1/3 of a standard deviation. Which roughly equates to an error around 20%.

TIP

There are a number of methods for breaking down categorical features into numeric values. For instance, if a feature is a category like male or female we would assign a value of 1 for male and perhaps 0 for female.

In most cases we will aim to reduce our expected error to less than 5%. For some problems this could be relaxed and in other cases the expectation could require much less acceptable error. The definition of what is an acceptable error will be up to you and/or your organization. If the organization you are working for does not have these standards it should also consider looking to put them in place for when legal comes knocking.

NOTE

You could certainly expect any industrial or human interactable AI to require much higher error tolerances. However, it depends on the situation and in many ways the interaction itself. A fraud detection system may have a relatively low error threshold compared to a medical application for instance.

After the model is trained we can then evaluate the training process itself. For the next exercise we look to evaluating the generated model and look at predictions.

Example 2-43. Evaluating an AutoML Model

- After training we will generally want to evaluate the results of our model. By evaluating the model we can get a further sense of important and not so important features.
- Click Evaluate tab to open the page. The top portion reiterates the earlier error metrics but at the bottom we can see a Feature importance plot as shown in Figure 2-14. This plot indicates to us the importance of each feature, how important it is and how important it is with respect to other features.

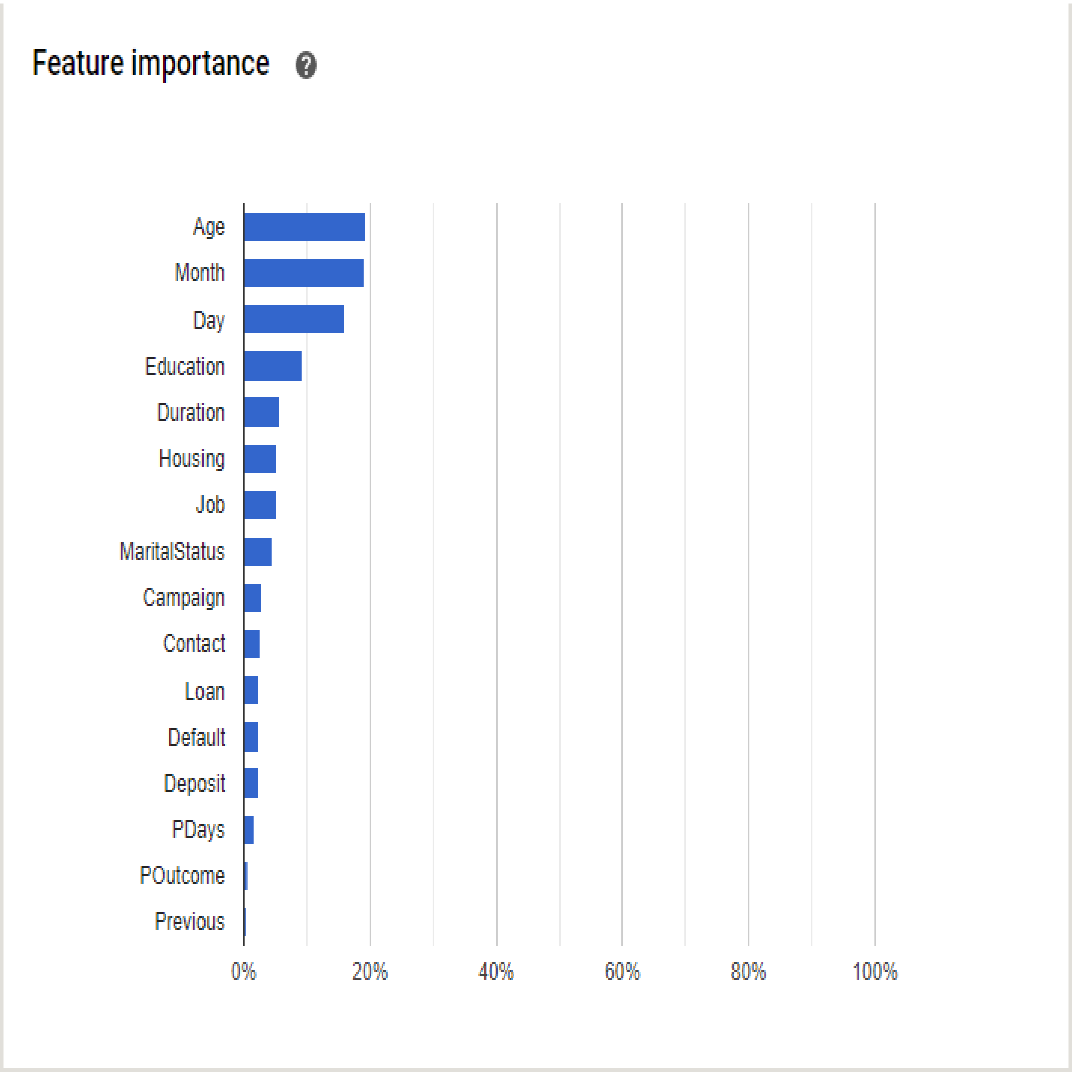
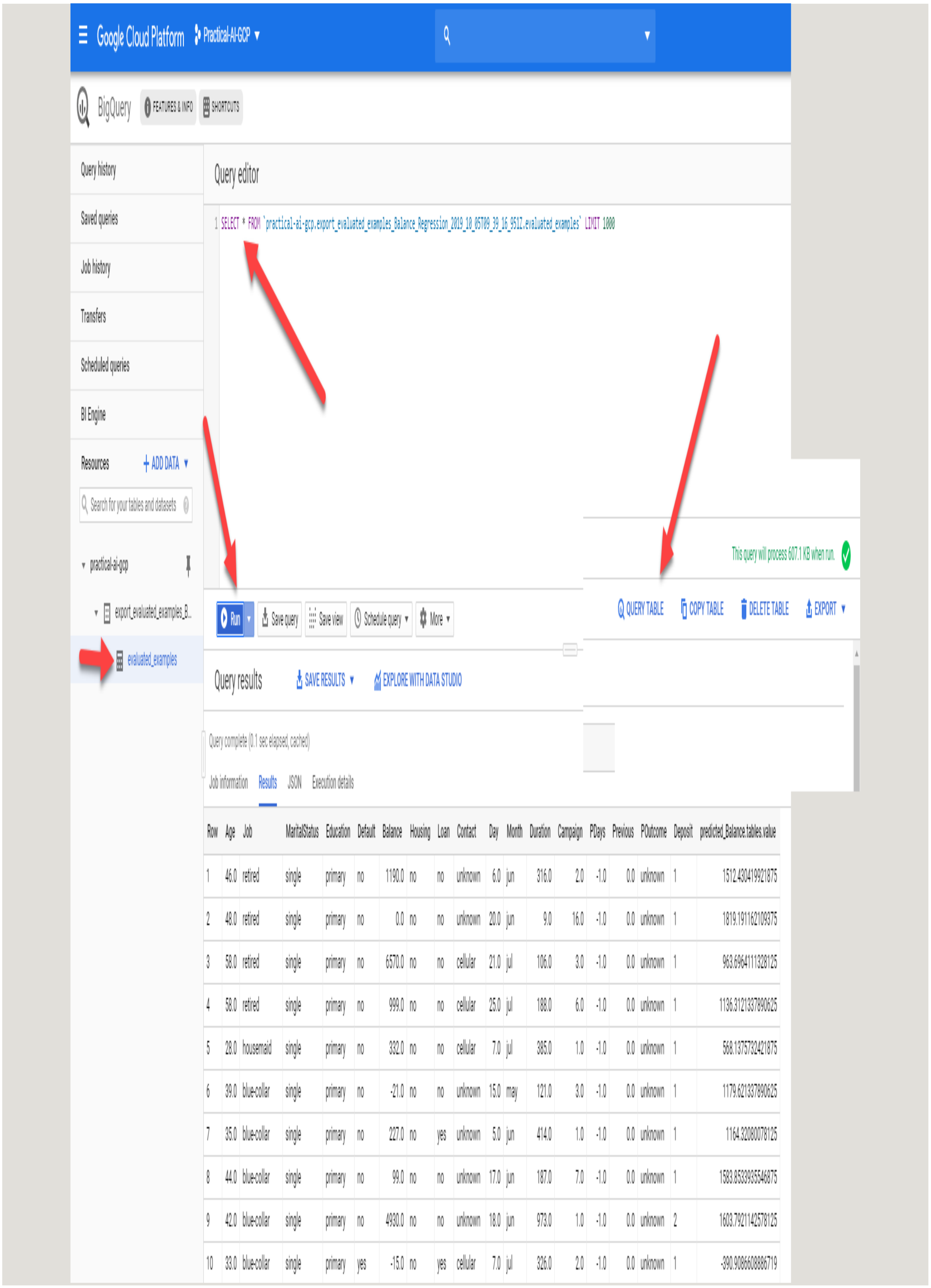


Figure 2-14. Feature importance plot of model

- In Figure 2-14 you can see the Age feature is the most important and the Previous feature the least.
- At this point we would often go back and eliminate/remove some less important features such as Previous, POutcome and PDays, and then retrain the model and re-evaluate. You may even repeat that process in order to tune the features further perhaps even adding other features in.
- Click the VIEW YOUR EVALUATION RESULTS IN BIGQUERY link in the middle of the page. This will direct

you to the query page and allow you to view the test or evaluation results.

- Next open the project data source and expand the results as shown in Figure 2-15.



```
SELECT * FROM `practical-ai-gcp.export_evaluated_examples_Balance_Regression_2019_10_05T09_39_16_951Z.evaluated_examples` LIMIT 1000
```

This query will process 607.1 KB when run.

Query complete (0.1 sec elapsed, cached)

Row	Age	Job	MaritalStatus	Education	Default	Balance	Housing	Loan	Contact	Day	Month	Duration	Campaign	PDays	Previous	POutcome	Deposit	predicted_Balance.tables.value
1	46.0	retired	single	primary	no	1190.0	no	no	unknown	6.0	jun	316.0	2.0	-1.0	0.0	unknown	1	1512.430419921875
2	48.0	retired	single	primary	no	0.0	no	no	unknown	20.0	jun	9.0	16.0	-1.0	0.0	unknown	1	1819.191162109375
3	58.0	retired	single	primary	no	6570.0	no	no	cellular	21.0	jul	106.0	3.0	-1.0	0.0	unknown	1	963.6964111328125
4	58.0	retired	single	primary	no	999.0	no	no	cellular	25.0	jul	188.0	6.0	-1.0	0.0	unknown	1	1136.3121337890625
5	28.0	housemaid	single	primary	no	332.0	no	no	cellular	7.0	jul	385.0	1.0	-1.0	0.0	unknown	1	568.1375732421875
6	39.0	blue-collar	single	primary	no	-21.0	no	no	unknown	15.0	may	121.0	3.0	-1.0	0.0	unknown	1	1179.621337890625
7	35.0	blue-collar	single	primary	no	227.0	no	yes	unknown	5.0	jun	414.0	1.0	-1.0	0.0	unknown	1	1164.3208007812 5
8	44.0	blue-collar	single	primary	no	99.0	no	no	unknown	17.0	jun	187.0	7.0	-1.0	0.0	unknown	1	1583.8533935546875
9	42.0	blue-collar	single	primary	no	4930.0	no	no	unknown	18.0	jun	973.0	1.0	-1.0	0.0	unknown	2	1603.7921142578125
10	33.0	blue-collar	single	primary	yes	-15.0	no	yes	cellular	7.0	jul	326.0	2.0	-1.0	0.0	unknown	1	-390.9086608886719

Figure 2-15. Evaluating results in BigQuery

- Click QUERY TABLE, far right on Figure 2-25. This button will not show until you click the dataset, previously. You should notice the SQL query get generated in the Query editor window.
- Type an * symbol character after the SELECT word. Again, denoted in Figure 2-15. This is the same as asking for all columns in SQL. If you are unfamiliar with SQL queries you can get help with that by consulting any good texts on relational databases.
- Click the Run button, noted in Figure 2-15 and view the results.
- You can also explore the data with Google Data Studio by clicking the EXPLORE WITH DATA STUDIO link in the middle of the page. Data Studio is a great product for building cool visualizations you can share anywhere.

AutoML Tables is a great tool that you can use to quickly evaluate deep learning models on tabular data. If you are a data scientist or someone that needs to make sense of tabular data then it may very well be the tool you need. However, it is not without expense and although that expense may be justified. It still will depend on your experience and your available resources. AutoML is a big tool in Google's arsenal and as we will see all the main building blocks provide some AutoML variation.

While this chapter is meant as an introduction to the AI Hub and Platforms. It is also intended to prepare you for working through the examples in the rest of this book. As such, in the section we cover the Google Cloud Shell a tool we will use frequently later.

The Cloud Shell

If you have any significant experience with the GCP then you likely already know your way around the shell and may want to miss this section. For those other readers, the shell will be an essential tool we use for later more complex exercises. Understanding how, why and when to use the shell will only make you a stronger AI GCP developer.

As a way of introducing the shell and the power it can provide us we are going to create a simple web site on the shell. Well, perhaps not an entire site but a page should do the trick. This is a technique we will use later in the book to make acutal full working sites featureing AI components we build. For now, open a browser and follow the exercise below:

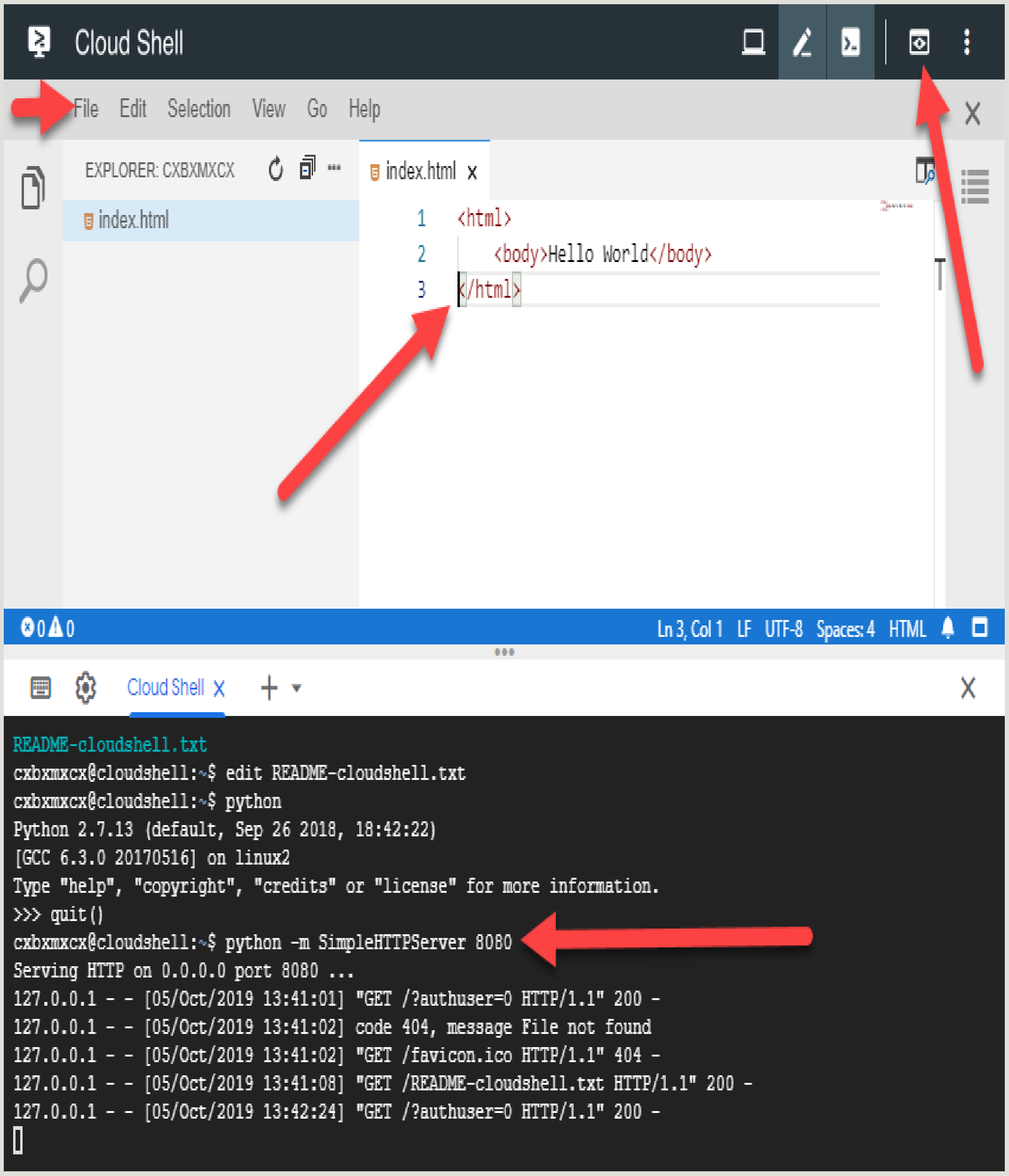
Example 2-44. Opening the Cloud Shell

- Point your browser to *https://console.cloud.google.com* and log in as well as choose your current project if you need to do so.
- Click the > icon at the top right of the window. The tool tip text reads Activate Cloud Shell.
- A shell window will open at the bottom of the browser. You should see your user email and the project you are currently working on.
- Enter into the console window the following command:

Example 2-45.

```
$ cloudshell edit
```

- This will open a blank editor window as shown in Figure 2-16.
- From the menu select File → New, to create a new file and name it index.html.


Cloud Shell
File Edit Selection View Go Help
EXPLORER: CXBXMXCX
index.html
Ln 3, Col 1 LF UTF-8 Spaces: 4 HTML
Cloud Shell

```
<html>
    <body>Hello World</body>
</html>
```

```
README-cloudshell.txt
cxbxmxcx@cloudshell:~$ edit README-cloudshell.txt
cxbxmxcx@cloudshell:~$ python
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
cxbxmxcx@cloudshell:~$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
127.0.0.1 - - [05/Oct/2019 13:41:01] "GET /?authuser=0 HTTP/1.1" 200 -
127.0.0.1 - - [05/Oct/2019 13:41:02] code 404, message File not found
127.0.0.1 - - [05/Oct/2019 13:41:02] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [05/Oct/2019 13:41:08] "GET /README-cloudshell.txt HTTP/1.1" 200 -
127.0.0.1 - - [05/Oct/2019 13:42:24] "GET /?authuser=0 HTTP/1.1" 200 -
```

Figure 2-16. Creating index.html and launching web server

- Enter the HTML text as shown in Figure 2-16 and and shown below. Save the file after with File → Save.

Example 2-46.

```
<html>
    <body>Hello World!</body>
</html>
```

- Now start up a self hosted web server on the console with the following:

Example 2-47.

```
python -m SimpleHTTPServer 8080
```

- That will launch a SimpleHTTPServer on the server running the console on port 8080.
- Go to the web page by clicking the Web Preview button at the top right of the window and as denoted in Figure 2-16.
- This will open a web page with your Hello World! message or whatever you put on your index.html page created ealier.

Being able to edit and run code through a console window in a web browser, you have to admit, quite cool. Except, we can actually do better than that and install a Visual Studio Code server and run Visual Studio Code in the browser. Including any additional modules or installation packages.

NOTE

Visual Studio Code is an open source project from Microsoft. It is quickly becoming a standard in Python development and other languages. The instructions for this example were modified from a blog post originally presented by Christiaan Hees on Medium.

Example 2-48. Installing Visual Studio Code Server

- From the console use the right side menu, the three dots, to Enable Boost Mode. This will require you to restart the instance and an authroization on your part.
- Install the code server with the following commands:

Example 2-49.

```
export VERSION=`curl -s https://api.github.com/repos/cdr/code-
server/releases/latest | grep -oP '"tag_name": "\K(.*)(?=")'`
```

```
wget https://github.com/cdr/code-server/releases/download/$VERSION/code-
server$VERSION-linux-x64.tar.gz
```

```
tar -xvzf code-server$VERSION-linux-x64.tar.gz
```

```
cd code-server$VERSION-linux-x64
```

- The first command searches for the latest version of the code server using curl and grep and saves it to VERSION. Then wget is used to download this version. Next tar is used to unpack the package and finally we change directories to the installation folder.
- From the installation folder we can launch the server with the following command:

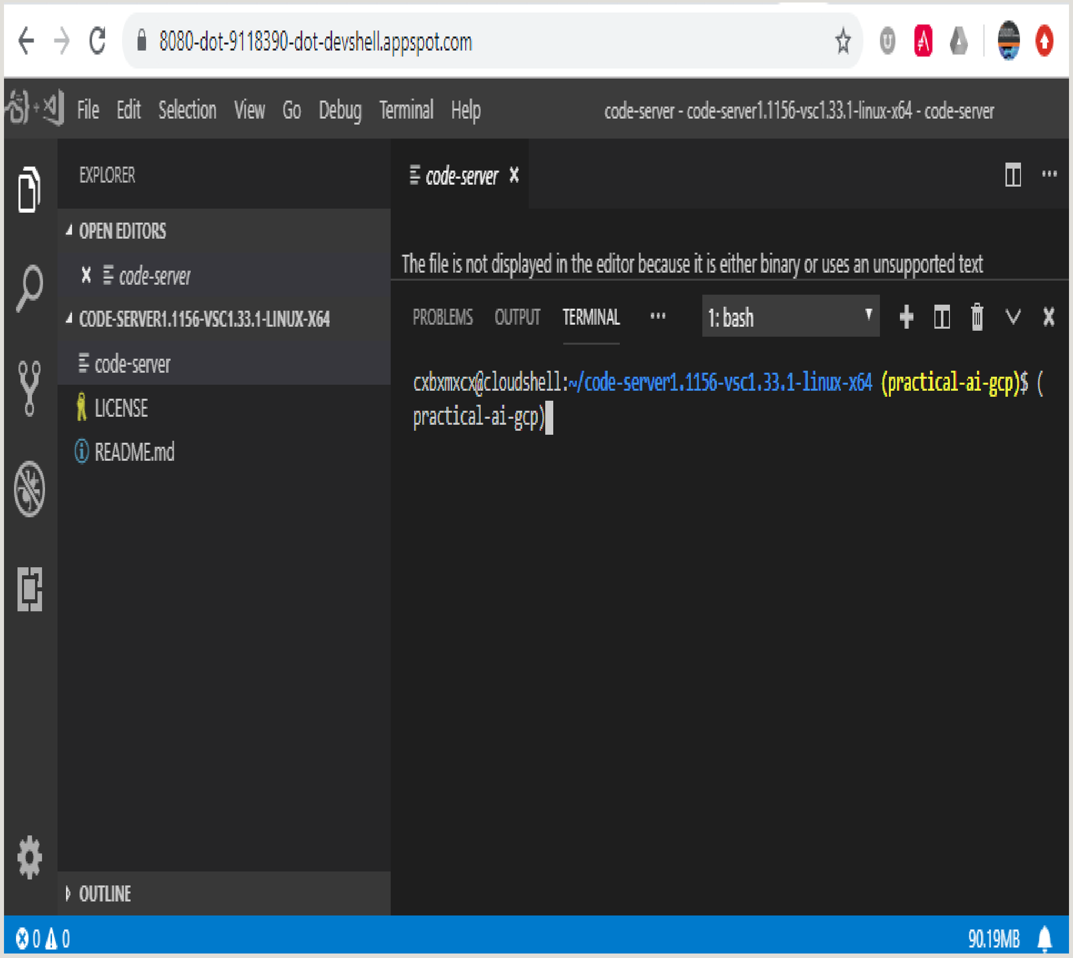
Example 2-50.

- Then use the Web Preview feature again to launch the page in a new browser tab. You likely will receive a 404 error, if you do just remove the ending ?authuser=0 as shown below:

Example 2-51.

```
https://8080-dot-YOURID-dot-devshell.appspot.com/?authuser=0
chnage to
https://8080-dot-YOURID-dot-devshell.appspot.com/
```

- Refresh the page again and you will see the Visual Studio Code IDE running in your browser as shown in Figure 2-17.


8080-dot-9118390-dot-devshell.appspot.com
File
Edit
Selection
View
Go
Debug
Terminal
Help
code-server - code-server1.1156-vsc1.33.1-linux-x64 - code-server
EXPLORER
OPEN EDITORS
code-server
CODE-SERVER1.1156-VSC1.33.1-LINUX-X64
code-server
LICENSE
README.md
OUTLINE
PROBLEMS
OUTPUT
TERMINAL
1: bash
0
0
90.19MB

The file is not displayed in the editor because it is either binary or uses an unsupported text

```
cxbxmxcx@cloudshell:~/code-server1.1156-vsc1.33.1-linux-x64 (practical-ai-gcp)$ (
practical-ai-gcp)
```

Figure 2-17. Visual Studio Code in the Browser

NOTE

Running a console session boosted. That is enabling a session to use boosted mode, is the same as instantiating a VM instance to run a self contained console. Running a VM instance uses credits and therefore you will be billed time for using this method.

One of the great things about the shell is that the session will remain on and active so that you can return to it. Which in turn allows us for a fully disconnected and cloud based development environment. Developing on and in the cloud also assures us that our environment is as close to production as possible. No more deploying from preconfigured development machines only to find you missed some archaic dependency. Instead, we are going to setup our development machine on the Google Console. In order to do that we need to install the required Python extensions in the next exercise.

Example 2-52. Installing Extensions to VS Code

- VS Code supports a number of extensions for development in many languages and just general helper tools like linters or search tools. Installing an extension is accomplished through the interface.

NOTE

A linter is a programming tool that analyzes code for syntax and style issues. It is always helpful to have a good standard linter for your chosen language.

- From the menu select View → Extensions.

- This will open the extensions menu and you can see all the available extensions for VS Code. Unfortunately, the Python extensions don't work, but that is okay as we don't really want to be developing Python code that way anyway. Instead, we will use the VS Code server platform for only when we need to diagnose special issues or inspect services we developed.
- You can search for any extension you need in the search bar at the top. Type in a search term such as data, cloud or python and the appropriate extensions will be listed.
- Type in *rest* to the search box and hit enter.
- A number of entries will come up but we likely want the first one called REST Client. This extension will allow us to use REST APIs from the cloud shell. Which in turn will ease our development efforts later.
- Select the REST Client extension and then click the Install button. This will install the extension.
- We can test the extension by creating a new File with File → New File from the menu. Name the file rest.txt, just as a test file.
- Enter the following text anywhere on the blank document:

Example 2-53.

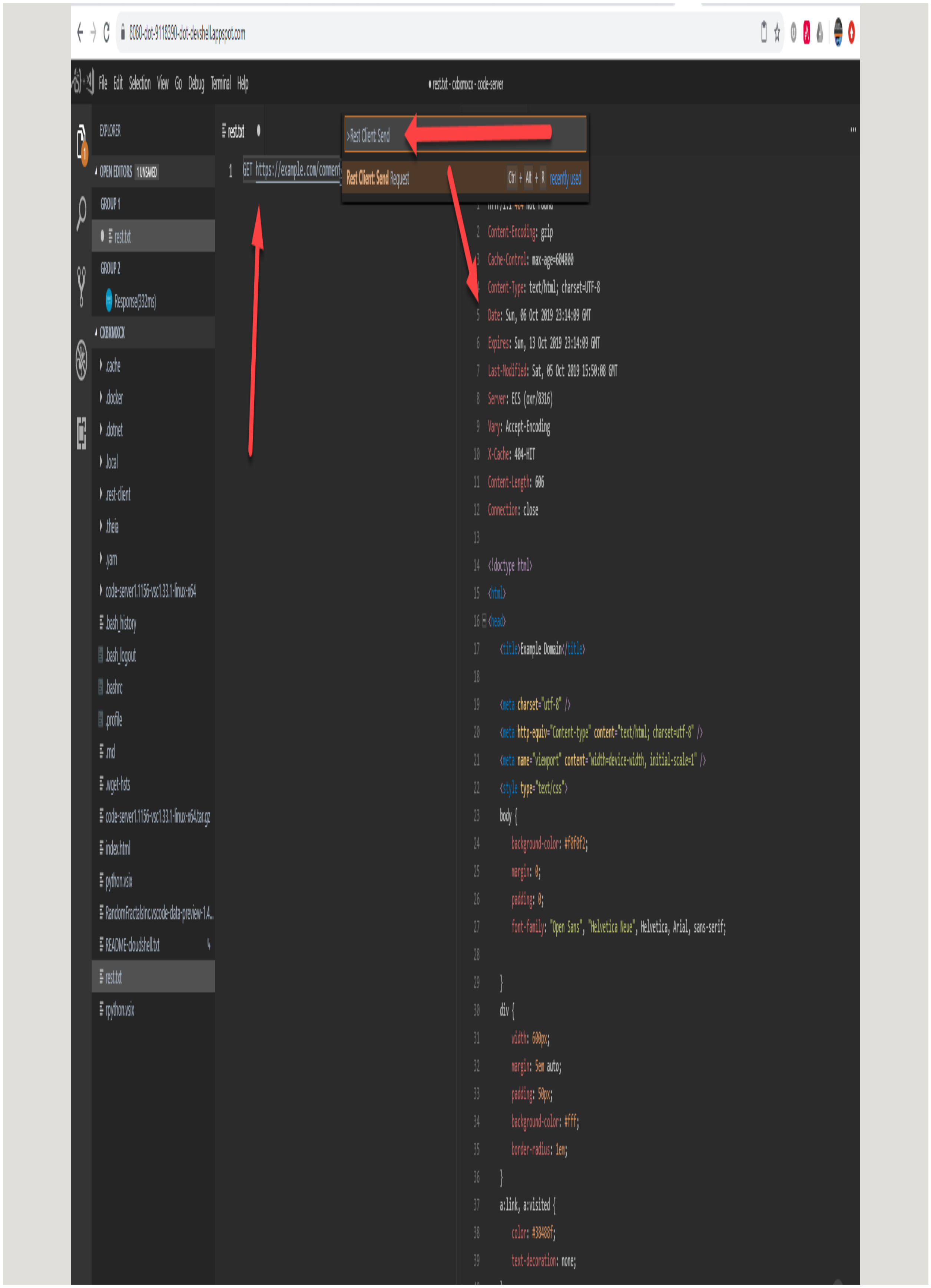
```
GET https://example.com/comments
```

- Keep your cursor on the line of text and type F1 to open the command window at the top.
- In the command window, and as shown in Figure 2-18, enter the following text to issue the command:

Example 2-54.

```
Rest Client: Send Request
```

- You don’t even have to finish typing the text and you can select the command from the filtered list, again as shown in Figure 2-18.



```
GET https://example.com/comment
```

```
HTTP/1.1 404 Not Found
Content-Encoding: gzip
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Sun, 06 Oct 2019 23:14:09 GMT
Expires: Sun, 13 Oct 2019 23:14:09 GMT
Last-Modified: Sat, 05 Oct 2019 15:50:08 GMT
Server: ECS (oxr/8316)
Vary: Accept-Encoding
X-Cache: 404-HIT
Content-Length: 606
Connection: close

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 50px;
        background-color: #fff;
        border-radius: 1em;
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
```

Figure 2-18. Getting a REST Request

- After the request is made the response will open in a new pane beside your request window. This can be very useful for diagnosing issues and developing services.

TIP

There are indeed many variations of REST clients you can use to test HTTP requests. We will use the one shown here both online and offline in order to test our API's later. If you have a preference using another REST client such as Postman then by all means use that, provided you can replicate the same procedures we will later.

From running a code server to making REST requests the Google Cloud Shell has many uses as we will see. One of the primary uses we can use the shell for is managing data, uploading or downloading files and that is what we will cover in the next section.

Managing Cloud Data

Up until now we have used the data made available on the GCP by various public sources or Google itself. Except, our goal will be to develop AI that can process our own data. In order to do that we will need to move and manage data on the GCP. For this section we introduce a collection of exercises/recipes you can use to move data for later projects in this book. Moving around data will be a common task and you will likely quickly memorize these recipes.

NOTE

If you have already worked on the GCP for any amount of time you likely already know how to move and transfer files around. Consider this section optional or as a refresher.

The first exercise we need to perform is to build a bucket or storage container on the GCP. This will be a place for us to move data around. By default you are provided with 5 GB of free storage. In all these exercises we will show how to use the console but Google also provides an interface for doing many of the same things.

Example 2-55. Creating/Managing a Cloud Bucket

- Open your Google Console Shell from *https://console.cloud.google.com*
- Enter the command:

Example 2-56.

```
gsutil mb gs://practicalai/
```

- This will create a new bucket called practicalai in your storage account.
- Next we want to list the buckets in our project with the following:

Example 2-57.

```
gsutil ls
```

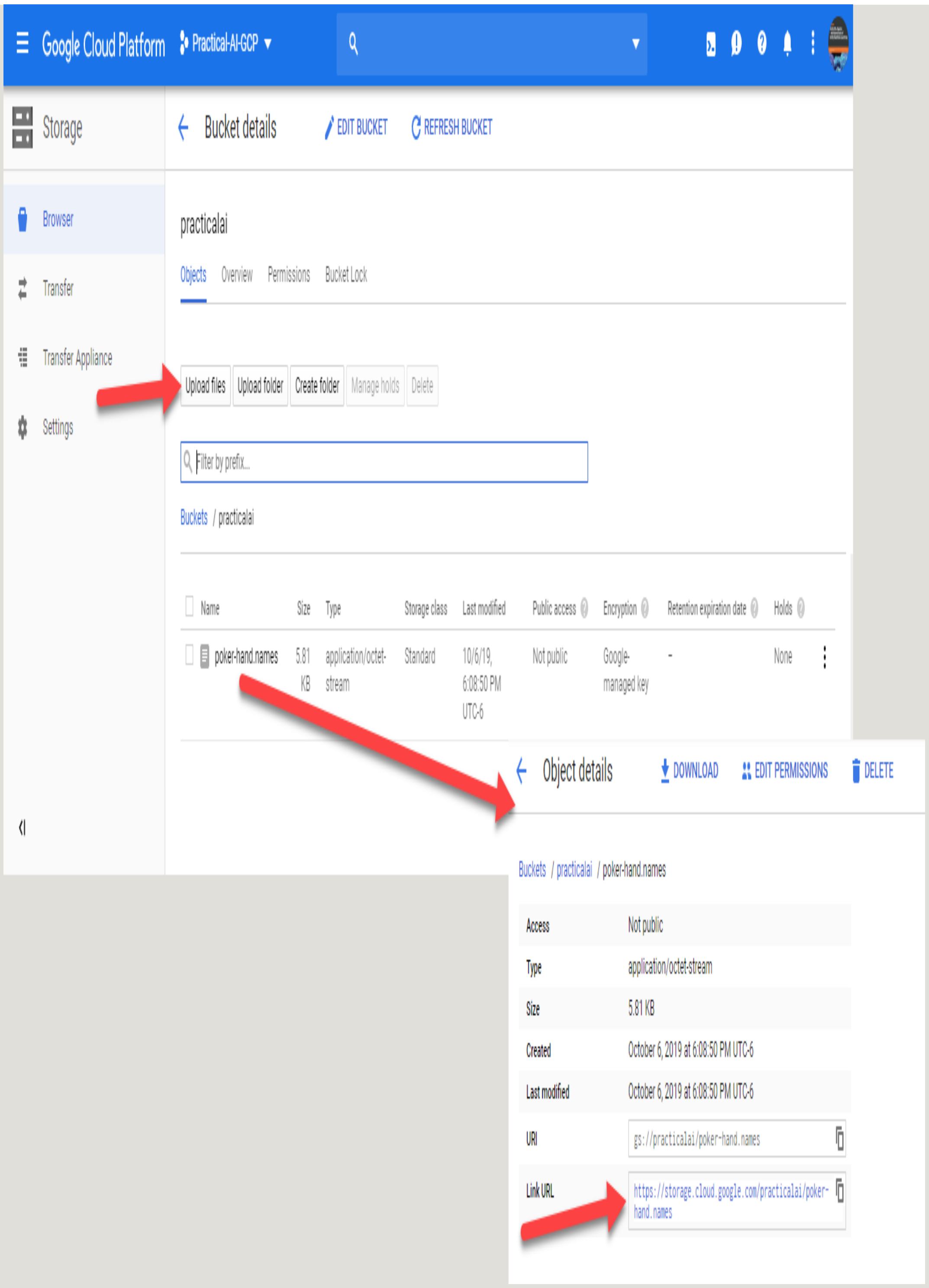
- You should see the new bucket in this list. Note the url starting with gs, for Google storage.

- Now we can upload a file into a bucket using the following template:

Example 2-58.

```
gsutil cp [OBJECT_LOCATION] gs://[DESTINATION_BUCKET_NAME]/
```

- You can also use the Google Storage interface to easily upload files to your bucket. GCS is at *https://console.cloud.google.com/storage*
- Open the storage interface in a browser. You can upload files into your bucket using the Upload files button as shown in Figure 2-19.



Name	Size	Type	Storage class	Last modified	Public access	Encryption	Retention expiration date	Holds
poker-hand.names	5.81 KB	application/octet-stream	Standard	10/6/19, 6:08:50 PM UTC-6	Not public	Google-managed key	-	None

Access	Not public
Type	application/octet-stream
Size	5.81 KB
Created	October 6, 2019 at 6:08:50 PM UTC-6
Last modified	October 6, 2019 at 6:08:50 PM UTC-6
URI	gs://practicalai/poker-hand.names
Link URL	https://storage.cloud.google.com/practicalai/poker-hand.names

Figure 2-19. Uploading files to your storage bucket

- Upload any file from your local computer to the GCS bucket. After you do you will see the file listed in the bucket. You can also list bucket contents from the console with the following command:

Example 2-59.

```
gsutil ls -r gs://[BUCKET_NAME]/**
```

- Once the file is in a GCS bucket it can moved a number of ways to the console or a notebook. One simple way we can transfer files is with wget using the following template:

Example 2-60.

```
wget url
```

- You can simply then insert the link url to your GCS file and wget will download it into your console session.

We will explore variations to the above methods for managing data in GCS buckets in the various exercises we explore later in the book. The important concept to grasp here is that all of our data needs to be first processed through GCS buckets before we can process if for AI. At least for the bulk training tasks. When we get to applications that use inference or an actual trained model we will look at managing data requests through a REST API.

Conclusion

Google has invested a considerable amount of resources into building an elite AI platform. Google boasts many of the top professionals working in various research and development groups spread out globally. On top of all this, the company has been very generous in providing free online resources like Google Colab. Along with plenty of free credits to spend on the entire GCP for building or learning about AI. While we just briefly touched on what is possible with AI on the GCP we have the whole rest of this book to explore the rest.

While we are still learning our way through the complex AI landscape in the next chapter we finally jump in and start using our first Google AI Building Block, Vision.

About the Author

Micheal Lanham is a proven software and tech innovator with 20 years of experience. During that time, he has developed a broad range of software applications. Including games, graphic, web, desktop, engineering, artificial intelligence, GIS, and machine learning applications for a variety of industries as an R&D developer. He currently resides in Calgary, Alberta, Canada, with his family.