2
Setup and Introduction to Deep Learning Frameworks

At this point, you are now familiar with **machine learning** (**ML**) and **deep learning** (**DL**) - this is great! You should feel ready to begin making the preparations for writing and running your own programs. This chapter helps you in the process of setting up TensorFlow and Keras, and introduces their usefulness and purpose in deep learning. Dopamine is presented as the new reinforcement learning framework that we will use later on. This chapter also briefly introduces other deep learning libraries that are important to know.

The topics that will be covered in this chapter are as follows:

- Introduction to Colaboratory
- Introduction and setup of TensorFlow
- Introduction and setup of Keras
- Introduction to PyTorch
- Introduction to Dopamine
- Other deep learning libraries

Introduction to Colaboratory

What is Colaboratory? Colaboratory is a web-based research tool for doing machine learning and deep learning. It is essentially like Jupyter Notebook. Colaboratory is becoming very popular these days as it requires no setup.

Throughout this book, we will be using Python 3 running on Colaboratory which will have installed all the libraries we may need.

Colaboratory is free to use and is compatible with most major browsers. The company in charge of the development of the Colaboratory tool is Google™. As opposed to Jupyter notebooks, in Colaboratory you are running everything on the cloud and not on your own computer. Here is the catch: you need a Google account since all the Colaboratory notebooks are saved into your personal Google Drive space. However, if you do not have a Google account, you can still continue reading to see how you can install every piece of Python library you will need to run things on your own. Still, I highly recommend you create a Google account, if only just to learn deep learning using the Colaboratory notebooks of this book.

When you run your code on Colaboratory, it runs on a dedicated virtual machine, and here is the fun part: you can have a GPU allocated to use! Or you can also use a CPU if you want. Whenever you are not running something, Colaboratory will deallocate resources (you know, because we all want to work), but you can reconnect them at any time.

If you are ready, go ahead and navigate to this link: `https://colab.research.google.com/`

If you are interested in more information and a further introduction to Colaboratory, search for *Welcome to Colaboratory!*. Now that you have accessed the previous link, let us get started with TensorFlow.

From now on, we will refer to **Colaboratory** as **Colab** for short. This is actually how people refer to it.

Introduction and setup of TensorFlow

TensorFlow (**TF**) has in its name the word *Tensor*, which is a synonym of vector. TF, thus, is a Python framework that is designed to excel at vectorial operations pertaining to the modeling of neural networks. It is the most popular library for machine learning.

As data scientists, we have a preference towards TF because it is free, opensource with a strong user base, and it uses state-of-the-art research on the graph-based execution of tensor operations.

Setup

Let us now begin with instructions to set up or verify that you have the proper setup:

1. To begin the installation of TF, run the following command in your Colaboratory:

   ```
   %tensorflow_version 2.x
   !pip install tensorflow
   ```

   This will install about 20 libraries that are required to run TF, including `numpy`, for example.

   Notice the exclamation mark (!) at the beginning of the command? This is how you will run shell commands on Colaboratory. For example, say that you want to remove a file named `model.h5`, then you would issue the command `!rm model.h5`.

2. If the execution of the installation ran properly, you will be able to run the following command, which will print the version of TF that is installed on your Colaboratory:

   ```
   import tensorflow as tf
   print(tf.__version__)
   ```

   This will produce the following output:

   ```
   2.1.0
   ```

3. This version of TF is the current version of TF at the time of writing this book. However, we all know that TF versions change frequently and it is likely that there will be a new version of TF when you are reading this book. If that is the case, you can install a specific version of TF as follows:

   ```
   !pip install tensorflow==2.1.0
   ```

   We are assuming that you are familiar with Python, thus, we will trust you with the responsibility of matching the proper libraries to the versions that we are using in this book. This is not difficult and can easily be done as shown previously, for example, using the == sign to specify the version. We will be showing the versions used as we continue.

TensorFlow with GPU support

Colaboratory, by default, has GPU support automatically enabled for TensorFlow. However, if you have access to your own system with a GPU and want to set up TensorFlow with GPU support, the installation is very simple. Just type the following command on your personal system:

```
$ pip install tensorflow-gpu
```

Notice, however, that this assumes that you have set up all the necessary drivers for your system to give access to the GPU. However, fear not, there is plenty of documentation about this process that can be searched on the internet, for example, `https://www.tensorflow.org/install/gpu`. If you run into any problems and you need to move forward, I highly recommend that you come back and do the work on Colaboratory, as it is the easiest way to learn.

Let us now address how TensorFlow works and how its graph paradigm makes it very robust.

Principles behind TensorFlow

This book is for absolute beginners in deep learning. As such, here is what we want you to know about how TF works. TF creates a graph that contains the execution from its input tensors, up to the highest level of abstraction of operations.

For example, let's say that we have tensors x and w that are known input vectors, and that we have a known constant b, and say that you want to perform this operation:

$$\mathbf{w}^T\mathbf{x} + b$$

If we create this operation by declaring and assigning tensors, the graph will look like the one in *Figure 2.1*:

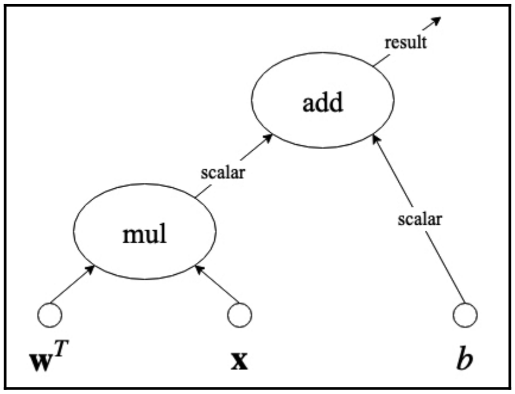

Figure 2.1 - Example of a tensor multiplication and addition operation

In this figure, there is a tensor multiplication operation, *mul*, whose result is a scalar and needs to be added, *add*, with another scalar, *b*. Note that this might be an intermediate result and, in real computing graphs, the outcome of this goes up higher in the execution tree. For more detailed information on how TF uses graphs, please refer to this paper (Abadi, M., et.al., 2016).

In a nutshell, TF finds the best way to execute tensor operations delegating specific parts to GPUs if available, or otherwise parallelizing operations on the CPU cores if available. It is open source with a growing community of users around the world. Most deep learning professionals know about TF.

Now let us discuss how to set up Keras and how it abstracts TensorFlow functionalities.

Introduction and setup of Keras

If you search on the internet for sample TensorFlow code, you will find that it may not be super easy to understand or follow. You can find tutorials for beginners but, in reality, things can get complicated very easily and editing someone else's code can be very difficult. Keras comes as an API solution to develop deep learning Tensorflow model prototypes with relative ease. In fact, Keras supports running not only on top of TensorFlow, but also over CNTK and Theano.

We can think of Keras as an abstraction to actual TensorFlow models and methods. This symbiotic relationship has become so popular that TensorFlow now unofficially encourages its use for those who are beginning to use TensorFlow. Keras is very user friendly, it is easy to follow in Python, and it is easy to learn in a general sense.

Setup

To set up Keras on your Colab, do the following:

1. Run the following command:

```
!pip install keras
```

2. The system will proceed to install the necessary libraries and dependencies. Once finished, type and run the following code snippet:

```
import keras
print(keras.__version__)
```

This outputs a confirmation message of it using TensorFlow as the backend as well as the latest version of Keras, which at the time of writing this book is 2.2.4. Thus, the output looks like this:

```
Using TensorFlow backend.
2.2.4
```

Principles behind Keras

There are two major ways in which Keras provides functionality to its users: a sequential model and the Functional API.

These can be summarized as follows:

- **Sequential model**: This refers to a way of using Keras that allows you to linearly (or sequentially) stack layer instances. A layer instance, in this case, has the same meaning as in our previous discussions in `Chapter 1`, *Introduction to Machine Learning*. That is, a layer has some type of input, some type of behavior or main model operation, and some type of output.
- **Functional API**: This is the best way to go deeper in defining more complex models, such as merge models, models with multiple outputs, models with multiple shared layers, and many other possibilities. Don't worry, these are advanced topics that will become clear in further chapters. The Functional API paradigm gives the coder more freedom to do different innovative things.

We can think of the sequential model as an easy way of starting with Keras, and the Functional API as the way to go for more complex problems.

Remember the shallow neural network from `Chapter 1`, *Introduction to Machine Learning*? Well, this is how you would do that model using the sequential model paradigm in Keras:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(10, input_shape=(10,)),
    Activation('relu'),
    Dense(8),
    Activation('relu'),
    Dense(4),
    Activation('softmax'),
])
```

The first two lines of code import the `Sequential` model and the `Dense` and `Activation` layers, respectively. A `Dense` layer is a fully connected neural network, whereas an `Activation` layer is a very specific way of invoking a rich set of activation functions, such as ReLU and SoftMax, as in the previous example (these will be explained in detail later).

Alternatively, you could do the same model, but using the `add()` method:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(10, input_dim=10))
model.add(Activation('relu'))
model.add(Dense(8))
model.add(Activation('relu'))
model.add(Dense(4))
model.add(Activation('softmax'))
```

This second way of writing the code for the neural model looks more linear, while the first one looks more like a Pythonic way to do so with a list of items. It is really the same thing and you will probably develop a preference for one way or the other. However, remember, both of the previous examples use the Keras sequential model.

Now, just for comparison purposes, this is how you would code the exact same neural network architecture, but using the Keras Functional API paradigm:

```
from keras.layers import Input, Dense
from keras.models import Model

inputs = Input(shape=(10,))

x = Dense(10, activation='relu')(inputs)
x = Dense(8, activation='relu')(x)
y = Dense(4, activation='softmax')(x)

model = Model(inputs=inputs, outputs=y)
```

If you are an experienced programmer, you will notice that the Functional API style allows more flexibility. It allows you to define input tensors to use them as input to different pieces of the model, if needed. However, using the Functional API does assume that you are familiar with the sequential model. Therefore, in this book, we will start with the sequential model and move forward with the Functional API paradigm as we make progress toward more complex neural models.

Just like Keras, there are other Python libraries and frameworks that allow us to do machine learning with relatively low difficulty. At the time of writing this book, the most popular is Keras and the second most popular is PyTorch.

Introduction to PyTorch

At the time of writing this book, PyTorch is the third most popular overall deep learning framework. Its popularity has been increasing in spite of being relatively new in the world compared to TensorFlow. One of the interesting things about PyTorch is that it allows some customizations that TensorFlow does not. Furthermore, PyTorch has the support of Facebook™.

Although this book covers TensorFlow and Keras, I think it is important for all of us to remember that PyTorch is a good alternative and it looks very similar to Keras. As a mere reference, here is how the exact same shallow neural network we showed earlier would look if coded in PyTorch:

```
import torch

device = torch.device('cpu')

model = torch.nn.Sequential(
          torch.nn.Linear(10, 10),
          torch.nn.ReLU(),
          torch.nn.Linear(10, 8),
          torch.nn.ReLU(),
          torch.nn.Linear(8, 2),
          torch.nn.Softmax(2)
        ).to(device)
```

The similarities are many. Also, the transition from Keras to PyTorch should not be too difficult for the motivated reader, and it could be a nice skill to have in the future. However, for now, most of the interest of the community is on TensorFlow and all its derivatives, especially Keras. If you want to know more about the beginnings and basic principles of PyTorch, you might find this reading useful (Paszke, A., et.al., 2017).

Introduction to Dopamine

An interesting recent development in the world of deep reinforcement learning is Dopamine. Dopamine is a framework for the fast prototyping of deep reinforcement learning algorithms. This book will deal very briefly with reinforcement learning, but you need to know how to install it.

Dopamine is known for being easy to use for new users in the world of reinforcement learning. Also, although it is not an official product of Google, most of its developers are Googlers. In its current state, at the time of writing this book, the framework is very compact and provides ready-to-use algorithms.

To install Dopamine, you can run the following command:

```
!pip install dopamine-rl
```

You can test the correct installation of Dopamine by simply executing the following command:

```
import dopamine
```

This provides no output, unless there are errors. Usually, Dopamine will make use of a lot of libraries outside of it to allow doing many more interesting things. Right now, some of the most interesting things one can do with reinforcement learning is to train agents with reward policies, which has direct applications in gaming.

As an example, see *Figure 2.2*, which displays a time snapshot of a video game as it learns, using policies that reinforce desired behavior depending on the actions taken by an agent:

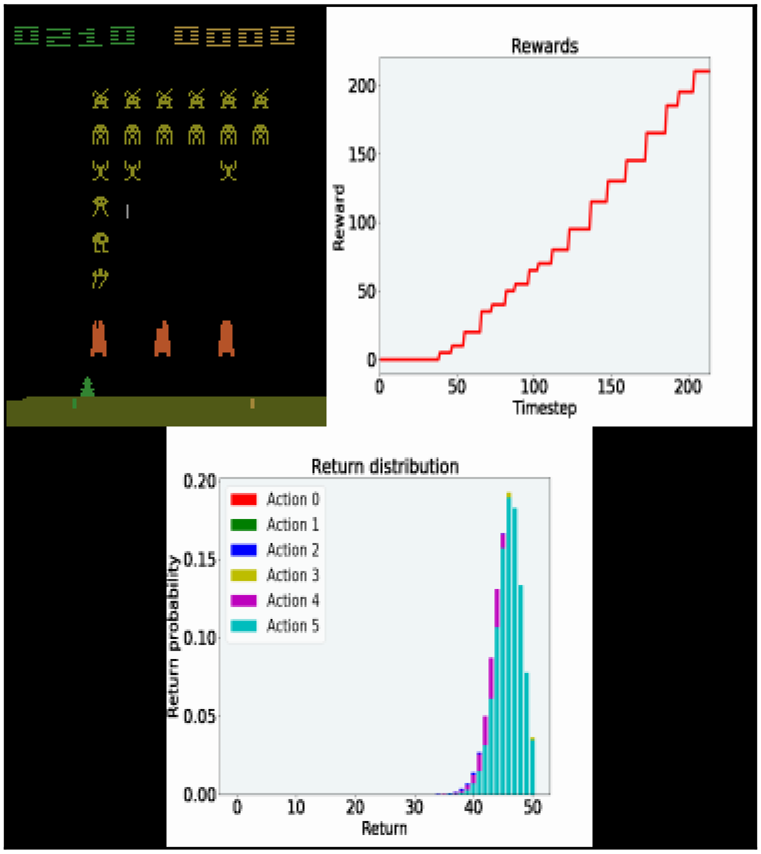

Figure 2.2 - Sample visualization of Dopamine's agent in a reinforcement learning problem in gaming

An agent in reinforcement learning is the piece that decides what action to take next. The agent accomplishes this by observing the world and the rules of the world. The more defined the rules are, the more constrained the result will be. If the rules are too loose, the agent may not make good decisions on what actions to take.

Although this book does not dive a great deal into reinforcement learning, we will cover an interesting gaming application in the last chapter of the book. For now, you can read the following white paper for more information about Dopamine (Castro, P. S., et.al., 2018).

Other deep learning libraries

Besides the big two, TensorFlow and Keras, there are other competitors that are making their way in the world of deep learning. We already discussed PyTorch, but there are more. Here we talk about them briefly.

Caffe

Caffe is also a popular framework developed at UC Berkeley (Jia, Y., et.al. 2014). It became very popular in 2015-2016. A few employers still demand this skillset and scholarly articles still mention its usage. However, its usage is in decay in part due to the major success of TF and the accessibility of Keras.

For more information about Caffe, visit: `https://caffe.berkeleyvision.org`.

Note also the existence of Caffe2, which is developed by Facebook and is open source. It was built based on Caffe, but now Facebook has its new champion, PyTorch.

Theano

Theano was developed by Yoshua Bengio's group at the University of Montreal in 2007 (Al-Rfou, R., *et.al.* 2016). Theano has a relatively old user base that probably saw the rise of TF. The latest major release was made in late 2017 and, although there are no clear plans of new major releases, updates are still being made by the community.

For more information about Theano, please visit: `http://deeplearning.net/software/theano/`

Honorable mentions

There are other alternatives out there that may not be as popular, for a variety of reasons, but are worth mentioning here in case their future changes. These are as follows:

Name	**Developed by**	**More information**
MXNET	Apache	`https://mxnet.apache.org/`
CNTK	Microsoft	`https://cntk.ai`
Deeplearning4J	Skymind	`https://deeplearning4j.org/`
Chainer	Preferred Networks	`https://chainer.org/`
FastAI	Jeremy Howard	`https://www.fast.ai/`

Summary

This introductory chapter showed how to set up the necessary libraries to run TensorFlow, Keras, and Dopamine. Hopefully, you will use Colabs to make things easier for you to learn. You also learned the basic mindset and design concept behind these frameworks. Although such frameworks are the most popular at the time of writing this book, there are other competitors out there, which we also introduced briefly.

At this point, you are all set to begin the journey to mastering deep learning. Our first milestone is to know how to prepare data for deep learning applications. This item is crucial for the success of the model. No matter how good the models are and how deep they are, if the data is not properly formatted or treated, it can lead to catastrophic performance results. For that reason, we will now go to `Chapter 3`, *Preparing Data*. In that chapter, you will learn how to take a dataset and prepare it for the specific task you are trying to solve with a specific type of deep learning model. However, before you go there, please try to quiz yourself with the following questions.

Questions and answers

1. **Does Colab run on my personal computer?**

   No, it runs in the cloud, but with some skill and setup, you could connect it to your own personal cloud.

2. **Does Keras use GPUs?**

   Yes. Since Keras runs on TensorFlow (in the setup of this book) and TensorFlow uses GPUs, then Keras also does.

3. **What are the two main coding paradigms in Keras?**

   (A) Sequential model; (B) Functional API.

4. **Why do we care about Dopamine?**

   Because there are only a few reinforcement learning frameworks you can trust out there, and Dopamine is one of them.

References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., and Kudlur, M. (2016). *Tensorflow: A system for large-scale machine learning.* In *12th {USENIX} Symposium on Operating Systems Design and Implementation* ({OSDI} 16) (pp. 265-283).
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. (2017). *Automatic differentiation in pytorch.*
- Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. (2018). *Dopamine: A research framework for deep reinforcement learning*. arXiv preprint arXiv:1812.06110.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014, November). *Caffe: Convolutional architecture for fast feature embedding.* In *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 675-678). *ACM.*
- Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A. and Bengio, Y. (2016). *Theano: A Python framework for fast computation of mathematical expressions*. arXiv preprint arXiv:1605.02688.