CHAPTER 11

Things to remember, what’s next for you, and final words

Welcome to the end of the ride. Through the last ten chapters, we have discovered, explored, and learned about the exciting field of machine learning with TensorFlow.js. With over ten unique examples that included a broad diversity of networks and use cases, we predicted the number of steps taken, clustered data, posed, identified toxic content, generated images, and more. It’s been a lot.

It is for this reason that for the last chapter, I would like to close with a recap of some concepts we saw and even add more on them—we will call this section “Things to remember.” Moreover, I also want to present a few pointers on how you could further extend what you learned here and offer some references that might help you in your future machine and deep learning adventures.
Things to remember

Tensors

This book was all about tensors and how they flow. So, it is very appropriate to start with them. As we learned, a tensor is a generalization of matrices of \( N \) dimensions with an attribute rank, shape, and dtype. Tensors have a very important property I want to highlight: they are immutable. Once their values are set, you cannot change them. So, every time we used a tensor method like `tf.add()`, we actually created a new one.

Another important fact about tensors is that we can change their shape. Functions like `tf.reshape()` (shown in the following) can turn, for example, a tensor of rank 1 into rank 2. Other functions that also modify a tensor’s shape are `tf.squeeze()` and `tf.expandDims()`, to remove and add dimensions, respectively:

```javascript
> const tf = require('@tensorflow/tfjs-node');
> let x = tf.tensor1d([1, 2, 3, 4]);
> x = x.reshape([2, 2]);
> x.print();
> Tensor
>    [[1, 2],
>     [3, 4]]
> x = x.expandDims(2);
> x.shape
> [ 2, 2, 1 ]
> x.squeeze().shape
> [ 2, 2 ]
```

1To execute this (and some of the upcoming) code, launch Node.js interactive shell using `$ node` from a directory where tfjs-node is installed.
You can also perform mathematical operations on tensors. For example, on several occasions, we applied arithmetical functions such as `tf.add()` and `tf.div()` to add or divide the tensor’s values by the given argument, for instance:

```javascript
> const a = tf.tensor1d([1, 2, 3, 4]);
> a.add(2).print();
> Tensor [3, 4, 5, 6]
```

Similarly, it is possible to compute element-wise operations between two tensors:

```javascript
> const b = tf.tensor1d([5, 6, 7, 8]);
> a.add(b).print();
Tensor [6, 8, 10, 12]
```

Last, we cannot forget about how to convert tensors to arrays. Functions like `tf.dataSync()` and `tf.arraySync()` return the tensor as an array or a nested array.

**Memory management**

Having many tensors takes a toll on your memory, causing unexpected slowdowns and, ultimately, crashes. That is why it is good practice to manage the memory of the application when working with extensive networks or when the app runs in the WebGL backend because it does not automatically garbage collect the unused tensors. To handle this problem, TensorFlow.js provides several functions to control the memory usage. Some of these are `tf.dispose()` to dispose of any tensor, `tf.tidy()` to clean up the tensors allocated within the given function, and `tf.memory()`
to return the program’s memory information. One we did not use, but we ought to know, is tf.keep(), used to avoid disposing of a tensor created inside a tf.tidy():

```
> const tf = require('@tensorflow/tfjs-node');
> const y = tf.tidy(() => {
...   x = tf.keep(tf.tensor1d([1, 2, 3, 4]));
...
...   return x.add(1);
... })
> x.print();
Tensor
[1, 2, 3, 4]
> y.print()
Tensor
[2, 3, 4, 5]
```

But if you remove the tf.keep() function and try to use the tensor x, you will get a “Tensor is disposed” error.

**TensorFlow.js Visualization (tfjs-vis)**

A package we regularly used was tfjs-vis, short for TensorFlow.js Visualization. With the library, we frequently visualized the datasets we used and the training progress. But the library does more than this.

Other than drawing scatter plots, with tfjs-vis you can create bar charts (tfvis.render.barchart()), histograms (tfvis.render.histogram()), or line charts (tfvis.render.linechart()).

Besides visualizing data, you can also create more specialized graphs such as a prettier model summary (Figure 11-1) using

```
tfvis.show.modelSummary({ name: 'Model', tab: 'Summary'}, model);
```
Figure 11-1. Model summary produced with tfjs-vis

Or you can visualize the distribution of a tensor’s values (Figure 11-2) with
const a = tf.tensor1d([1, 2, 3, 4, 5, 6, 6]);
const surface = {name: 'Values Distribution', tab: 'Tensor'};
await tfvis.show.valuesDistribution(surface, a);

Figure 11-2. Distribution of a tensor’s values
In the earlier chapters, we used tfjs-vis through its “visor” window. However, in Chapter 9, we learned that you could also present the visualizations on a `<canvas>` by using the canvas id as the first argument to the function:

```javascript
const container = document.getElementById('canvas-training-tfvis');
await tfvis.show.valuesDistribution(container, tensor);
```

For the complete reference guide, visit [https://js.tensorflow.org/api_vis/latest/](https://js.tensorflow.org/api_vis/latest/).

**Designing a model**

Designing a model is not a simple task. Creating one requires comprehending the dataset at hand, understanding every detail of the job one wishes to solve, and knowing the different types of layers available in the framework. This last point is not simple. If, at some point, you felt overwhelmed by the layers, and their peculiarities, let me assure you that that is completely fine. Over time, after designing many models, you will start to understand their differences and functionalities.

In TensorFlow.js, there are two principal ways of creating models. One is using the **Sequential** (“stack of pancakes”) approach and the other, the **Functional** (“pancakes on the table”) approach. Most of our exercises used the Sequential model, a model topology that describes a stack of layers where the output of one is the input of the next one. The Functional or `tf.model()` way is the graph-based alternative where you define the topology by specifying its input and output and connecting the layers using `tf.apply()`. Regardless of the type of layer, you must always declare the shape of the input tensor in the input layer. With a Sequential model, you can specify the shape using `inputShape` in the first layer. Or, if you are using a `tf.model()`, then you need to define the shape in `tf.input()`.
After designing the architecture, the next step is compiling and fitting the model. Before training a model, you always have to compile it to set the loss function, optimizer, and metrics attributes necessary for training. Then comes `model.fit()` to fit it. The parameters of `model.fit()` are the training data, the targets, or, in the case of `model.fitDataset()`, a `tf.data.Dataset` object. In both cases, you also need an object to configure the hyperparameters. Of these hyperparameters, you should at least define the batch size and number of epochs. Otherwise, the model will use the default values. Other important attributes are the callbacks, validation data, and shuffle.

For more information about the layers, visit the documentation at https://js.tensorflow.org/api/latest/#Layers.

**Hyperparameters and attributes**

If designing a network’s architecture is complicated, deciding its hyperparameters’ values is arguably even more. There is no right or wrong set of attributes for a network. After all, each dataset and use case is different. However, there are best practices and guidelines that serve as a starting point. Remember the DCGAN from Chapter 10? The learning rate of 0.0002 we used came from the paper that describes the model. My recommendation is that, when creating a model, check the literature, projects, or tutorials to find configurations that have worked for others. Try these configurations. Study how they influence the model’s performance. Doing so will give you an idea of how to approach the problem.

Another essential attribute to consider is the loss function. Unlike hyperparameters, the loss function is directly bound to the model’s task. So, choosing one is more straightforward than selecting, let’s say, the number of epochs. For example, if the task at hand is a regression problem, the mean squared error is an optimal option. If the problem is
classification, binary cross-entropy performs well for binary classification, just like categorical cross-entropy works for multiclass classification. Table 11-1 summarizes this information.

**Table 11-1. Loss functions suitable for certain tasks**

| Task               | Target                   | Loss Function          | Last Layer's Activation Function |
|--------------------|--------------------------|------------------------|---------------------------------|
| Regression         | Continuous value         | Mean squared error     | -                               |
| Classification     | Binary classification    | Binary cross-entropy   | Sigmoid                         |
| Classification     | Multiclass classification | Categorical cross-entropy | Softmax                        |

**Testing**

In software development, testing is essential, and yet, in the book, we did not perform any kind of test. The reason I omitted the test cases was to simplify the code as much as possible. But behind the scenes, I performed various tests that mostly involved confirming the shape of tensors or their values. To test statements in TensorFlow.js, the API provides a function, `tf.util.assert()`, that asserts if an expression is true. To illustrate it, consider the following example where we test the shape of a tensor:

```javascript
> const tf = require('@tensorflow/tfjs-node');
> const a = tf.tensor2d([1, 2, 3, 4], [2, 2]);
> tf.util.assert(JSON.stringify(a.shape) == JSON.stringify([2, 2]), 'shape is not [2,2]');
> tf.util.assert(JSON.stringify(a.shape) == JSON.stringify([2, 3]), 'shape is not [2,3]');
Uncaught Error: shape is not [2,3]
    at Object.assert (/.../node_modules/@tensorflow/tfjs-core/dist/util.js:105:15)
```
In the first example, we are testing if the shape of the tensor \( a \) is \([2, 2]\). If true (like here), nothing happens. Note that for comparing the shapes (which are arrays), we had to convert them to strings and compare those. If the expression is not true (the second test case), it returns the message provided in the second argument. A second common testing scenario is asserting the range of a tensor’s values, like this:

```javascript
> const b = tf.tensor1d([-1, 0, 1]);
> tf.util.assert(
...   b.min().dataSync()[0] >= -1 && b.max().dataSync()[0] <= 1,
...   'values outside range',
...);
```

In this example, we used `tf.min()` and `tf.max()` to check if the values of the tensor are between -1 and 1, a scenario we saw in Chapter 10. The indexing is used because `dataSync()` returns the min or max value in an array of length 1.

### Asynchronous code

While this point is not only related to TensorFlow.js, it is worth mentioning that when developing web applications, you should consider using asynchronous code to avoid blocking the app. Potential scenarios include calls to the functions `tf.Sequential.fitDataset()` and `tf.loadLayersModel()` or, in general, when executing a function that might cause a noticeable pause in the app. In our examples, we used the synchronous version—`tf.dataSync()` and `tf.arraySync()`—of the tensors-to-array functions since we were not dealing with large tensors and the response was immediate. However, if necessary, consider using the asynchronous variants `tf.data()` and `tf.array()`.
What’s next for you?

So, you finished the book. Does that mean you are done learning deep learning? Never! Deep learning, and the discipline of data, is a vast field that grows with every passing day. Fortunately, the applications and lessons we learned served as preparation to get you started and start exploring on your own. If you ask me where you should start, I would recommend exploring several of the TensorFlow.js pre-trained models we did not see, for example, the body segmentation model and the speech command recognition. After them, you could study other deep learning and machine learning concepts like embeddings, autoencoders, image segmentation, attention-based models, and dimensionality reduction algorithms. Regarding the latter, you can find an implementation of t-Distributed Stochastic Neighbor Embedding (t-SNE) in the official TensorFlow.js repository. However, at the time of writing, the library has not been updated to the latest version of TF.js. Another option is returning to the models we created to improve them. Add (or remove) more layers, use different data, tweak the hyperparameters, create a fun app, write a Chrome extension, and more.

If you want to take the models even further, try deploying them in other platforms, for example, mobile, cloud, or even on a desktop application using Electron. By deploying them somewhere else, not only will you get the chance to interact with them from a fresh perspective but also the opportunity of learning about a new framework. Additionally, try to experiment with TensorFlow (Python). Even better, once there, alter the models you already have (transfer learning, maybe?) to experience the similarity between the TensorFlow and TensorFlow.js frameworks.

---

2https://github.com/tensorflow/tfjs-tsne
And lastly, and above all, have fun. Train, deploy, make, break, visualize, design, question, and enjoy—that is, in my opinion, the best way of learning.

**External resources**

The following is a list of external resources to guide you through your TensorFlow.js adventures:

- The official API documentation at [https://js.tensorflow.org/api/latest/](https://js.tensorflow.org/api/latest/): This is the library’s guide. In it, you will find documentation about every single function and object TensorFlow.js has to offer.

- StackOverflow’s TensorFlow.js tag at [https://stackoverflow.com/questions/tagged/tensorflow.js](https://stackoverflow.com/questions/tagged/tensorflow.js): Use this tag to ask questions or find answers (now you should be able to answer some!).

- The TensorFlow.js’ GitHub issues tab at [https://github.com/tensorflow/tfjs/issues](https://github.com/tensorflow/tfjs/issues): This is mostly for reporting bugs and issues with the library, and not for asking help. Notwithstanding, you might find some answers here, especially when they are related to bugs.

- For finding datasets, try *Kaggle’s Dataset* ([www.kaggle.com/datasets](http://www.kaggle.com/datasets)) collection or *Google Dataset Search* ([https://datasetsearch.research.google.com/](https://datasetsearch.research.google.com/)).

- For the theory and mathematics behind the concepts here introduced, I highly recommend the book *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
The topics we covered here belong mostly to the subfield of deep learning. But machine learning is more about networks and tensors. If you wish to learn more about machine learning in general, I recommend reading *Hands-on Scikit-Learn for Machine Learning Applications* by David Paper (Apress) and *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* by Aurélien Géron (O’Reilly).

**Thank you**

As cliché as it sounds, I want to end by saying thank you. This book has been not only a journey for you but for me, too. Every single line of code, concept, and piece of information you learned here was also a lesson for me. I am grateful for that.

As said in the beginning of this book, I wrote this book during my backpacking adventures. In fact, as I am writing this line, I am stuck in New Zealand in the middle of the Coronavirus crisis. So, behind every page you read, there is a new city, a different company, an exciting tale, or even a terrible Internet connection. But even though I was exposed to those changes, the only constant thing was the desire to bring a great book about TensorFlow.js and machine learning to you.

I truly hope that through the last 11 chapters, you have gained a whole world of knowledge that will follow you in your future endeavors. If you have questions, find an error, or want to show me your newest app, please contact me on Twitter: [https://twitter.com/jdiossantos](https://twitter.com/jdiossantos). I would love to hear from you.

Stay cool!

*Juan :)*

*April 7, 2020, Christchurch, New Zealand*