

DAT300 CHAPTER 13  
Parallelizing Neural Network Training with  
TensorFlow

Jorid Holmen  
Applied Robotics, Norwegian University of Life Sciences

This chapter will move from the mathematical foundations of machine learning and deep learning, to focus on TensorFlow. TensorFlow is one of the most popular deep learning libraries currently available, and makes it possible to implement neural networks (NNs) efficiently.

## 1 TensorFlow and training performance

TensorFlow can speed up our machine learning tasks significantly.

### 1.1 Performance challenges

The performance of computer processors has been continuously improving in recent years, which allows us to train more powerful and complex learning systems.

Scikit-learn allows us to spread computations over multiple processing units, but Python is by default limited to execution on one core due to the **global interpreter lock (GIL)**. We also have to think about that most advanced desktop hardware rarely comes with more than eight or 16 cores.

Often we have tasks that are unfeasible for single processing units, and for that we have to use **graphic processing units (GPUs)**. The GPU is the main components on graphic cards. You can think of graphic cards as a small computer cluster inside your machine, and it is much cheaper than a CPU.

The problem is that it is challenging to write code to target GPUs, but that is what TensorFlow is for!

### 1.2 What is TensorFlow?

TensorFlow is a scalable and multiplatform programming interface for implementing and running machine learning algorithms, including convenience wrappers for deep learning. A convenience wrapper is a way of simplifying the writing of computer programs. TensorFlow is a Python-based, free, open-source machine learning platform, developed primarily by Google. Much like NumPy, the primary purpose of TensorFlow is to enable engineers and researchers to manipulate mathematical expressions over numerical tensors.

TensorFlow goes far beyond the scope of NumPy in the following ways:

- It can automatically compute the gradient of any differentiable expression, making it highly suitable for machine learning.
- It can run not only on CPU but also on GPUs and TPUs, highly-parallel hardware accelerators.
- Computation defined in TensorFlow can be easily distributed across many machines.
- TensorFlow programs can be exported to other runtimes, such as


- C++, JavaScript (for browser-based applications), or
- TFLite (for applications running on mobile devices or embedded devices), etc.
- This makes TensorFlow applications easy to deploy in practical settings.


TensorFlow is built around a computation graph composed of a set of nodes. Each node represents an operation that may have zero or more input or output. A computation graph is defined as a representation that combines signal-flow and precedence graphs to describe the computational properties of an algorithm, used for scheduling operations and deriving storage and communication requirements.

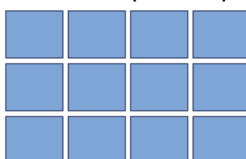
## 2 Data representation of neural networks

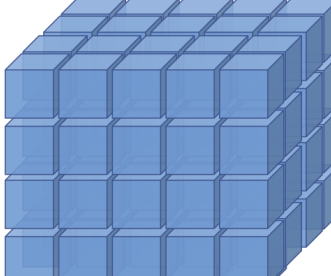
### 2.1 What is a tensor

All machine learning systems use tensors as their basic data structure. A **tensor** is created as a symbolic handle to refer to the input and outputs of these operations. Tensors are containers for (mostly numerical) data. Mathematically, tensors can be understood as a generalization of scalars, vectors, matrices, and so on, to arbitrary number of dimensions. A dimension is often called an axis in the context of tensors. A scalar is defined as rank-0 tensor, vector is rank-1 tensor, matrix is rank-2 tensor, and matrices stacked in third dimension is rank-3 tensor.

Rank 0:   
(scalar)

Rank 1:   
(vector)

Rank 2: (matrix)  


Rank 3: 

### 2.2 Vector data

Vector data is the most common case of data representation. In such datasets, each single data point can be encoded as a vector. Thus, a batch of data will

be encoded as 2D tensors (array of vectors). In a 2D tensor, the first axis is the samples axis and the second axis is the features axis.

## 2.3 Time series or sequence data

Whenever time matters in your data (or the notion of sequence order), it makes sense to store it in a 3D tensor with an **explicit time axis**. Each sample can be encoded as a sequence of vectors (a 2D tensor). Thus, a batch of data will be encoded as a 3D tensor.

## 2.4 Image data

Images typically have three dimensions: height, width and colour depth. Grayscale images have only a single color, and can therefore be stored in a 2D tensor, but image tensors are always 3D. For grayscale images there will only be a one-dimensional color channel for grayscale images. Several images are stored in a 4D, since it includes the number of images (samples).

## 2.5 Video data

Video data is one of the few types of real-world data for which you will need 5D tensors. A video can be understood as a sequence of frames, each frame being a color image. Because each frame can be stored in a 3D tensor (height, width, color depth), a sequence of frames can be stored as a 4D tensor (frames, height, width, color depth). A batch of several videos can be stored in a 5D tensor (samples, frames, height, width, color depth).

# 3 TensorFlow

## 3.1 Commands in TensorFlow

### 3.1.1 Installing Tensorflow

`pip install tensorflow-gpu` to use GPU (recommended), otherwise it is just `pip install tensorflow`.

### 3.1.2 Creating tensors in TensorFlow

You can simply create a tensor from a list or NumPy array using:  
`tf.convert_to_tensor()`.

### 3.1.3 Manipulating the data type and shape of a tensor

You can manipulate tensor data types and shapes via several TensorFlow functions that cast, reshape, transpose and squeeze:

- `tf.cast(x, dtype, name=None)` - casts a tensor to a new data type.

- `tf.transpose(a, perm=None, conjugate=False, name='transpose')`  
- permutes the dimensions of a tensor. In simpler terms, it rearranges the order of the axes of the tensor according to a specified pattern.
- `tf.reshape(tensor, shape, name=None)` - used to change the shape of a tensor without altering its data.
- `tf.squeeze(input, axis=None, name=None)` - removes dimensions of size 1 from a tensor's shape.

#### 3.1.4 Applying mathematical operations to tensors

Applying mathematical operations, in particular linear algebra operations, is necessary for building most machine learning algorithms.

- The element-wise product of two tensors `tf.multiply(t1, t2).numpy()`
- `tf.math.reduce_mean()`
- `tf.math.reduce_sum()`
- `tf.math.reduce_std()`
- the matrix-matrix product between two tensors `tf.linalg.matmul()`
- `tf.norm()`

#### 3.1.5 Split, stack and concatenate tensors

This section is about splitting tensors, or stacking and concatenating multiple tensors into a single tensor.

To split we use `tf.split()` with the argument `num_or_size_splits`. If the argument is 3, the tensor will be split in three equal parts. If the tensor is 5 and the argument is `num_or_size_splits[3,2]`, the tensor will be split into two tensor with size 3 and 2.

To stack or concatenate we use `tf.concat` or `tf.stack`.

#### 3.1.6 Creating a TensorFlow Dataset from exisisting tensors

We can create a dataset from a tensor, lists or NumPy arrays using

`tf.data.Dataset.from_tensor_slices()`.

To create batches we use `dataset.batch()`.

#### 3.1.7 Combining two tensors into a joint dataset

We could for example have one tensor for features and one tensor for labels. We need to build a dataset that combines these two together.

```
ds_x = tf.data.Dataset.from_tensor_slices(t_x)
ds_y = tf.data.Dataset.from_tensor_slices(t_y)
ds_joint = tf.data.Dataset.zip((ds_x, ds_y))
```

A common source of error could be that the element-wise correspondence between the original features ( $x$ ) and labels ( $y$ ) might be lost.

### 3.1.8 Shuffle, batch and repeat

It is important to feed training data randomly shuffled batches, when using stochastic gradient descent. You have already seen how to create batches, but you can also shuffle and reiterate over the batches.

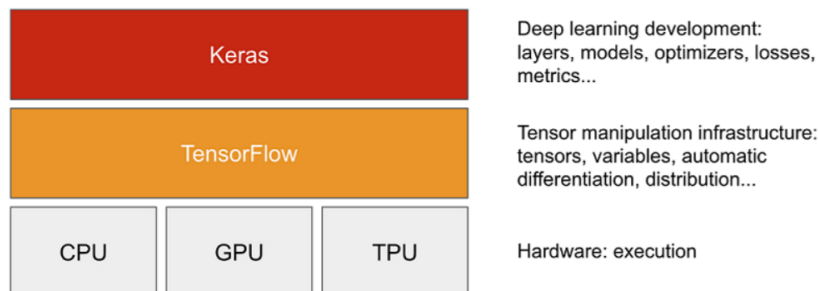
```
tf.random.set_seed(1)
ds = ds_joint.shuffle(buffer_size=len(t_x))
```

The rows are shuffled without losing the one-to-one correspondence between the entries in  $x$  and  $y$ . `buffer_size` determines how many elements in the dataset are grouped together before shuffling.

If the dataset is small, a small `buffer_size` can negatively affect the predictive performance of the NNs, as the dataset may not be completely randomized. The size usually does not have a noticeable effect on larger datasets. It is common to choose a buffer size that is equal to the number of training examples.

## 3.2 TensorFlow Keras API (tf.keras)

Keras is a high-level NN API, and a deep learning framework, and was originally developed to run on top of other libraries such as TensorFlow and Theano. Keras provides a user-friendly and modular programming interface that allows easy prototyping and the building of complex models (including deep learning models) in just a few lines of code. TensorFlow is a low-level tensor computing platform, Keras is a high-level deep learning API.



When we are training a deep NN model, we usually train the model incrementally using an iterative optimization algorithm such as stochastic gradient descent. Keras API is a wrapper around TensorFlow for building NN models. The Keras API provides a method, `.fit()` for training the models.

In cases where the training dataset is rather small and can be loaded as a tensor into the memory, TensorFlow models that are built with the Keras API can directly use this tensor via the `.fit()` method for training. When the dataset is too large for the computer memory, the data needs to be loaded to the device in chunks, batch by batch.

Keras is tightly integrated with TensorFlow and accessed through `tf.keras`. The Keras API makes building NN models extremely easy. The most common approach is through `tf.keras.Sequential()`, which allows stacking layers to form a network. A stack of layers can be given in a Python list, alternatively, the layers can be added one by one using the `.add()` method.

Furthermore Keras makes it possible to define a model by subclassing using `tf.keras.Model`. This gives us more control over the forward pass by defining the `call()` method for our model class to specify the forward pass explicitly.

Models built using the `tf.keras` API can be compiled and trained via the `.compile()` and `.fit()` arguments.

### 3.3 The Keras workflow

The general workflow of a Keras model is constructed like this:

1. Define your training data: input tensors and target tensors.
2. Define a network of layers (or model) that maps your input targets.
3. Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor, using the `compile()` method
4. Iterate on your training data by calling the `fit()` method of your model.
5. Evaluate the model.
6. Predict.

There are two ways to define a Keras model. Using the **Sequential** class (only for linear stacks of layers, which is the most common) and using the **functional** API (for directed acyclic graphs of layers, which lets you build completely arbitrary architectures). Once your model architecture is defined, the learning process is configured in the compilation step, where you specify:

- Loss function (objective function): the quantity that will be minimized during training. It represents a measure of success for the task at hand.
- Optimizer: determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD)

- Metrics: The measures of success you want to monitor during training and validation, such as classification accuracy. Unlike the loss, training will not optimize directly for these metrics. As such, metrics don't need to be differentiable.

Finally, the learning process consists of passing Numpy arrays of input data (and the corresponding target data) to the model via the `.fit()` method, similar to what you would do in scikit-learn and several other machine-learning libraries.

## 4 Usefull pages and videos

**Tensorflow in 100 Seconds - Fireship**

**What is Keras and Tensorflow — Keras vs Tensorflow**

**What is TensorFlow? - IBM Technology**

**Tensors for Neural Networks, Clearly Explained!!! - StatQuest with Josh Starmer**

**Dive into deep learning - online book**