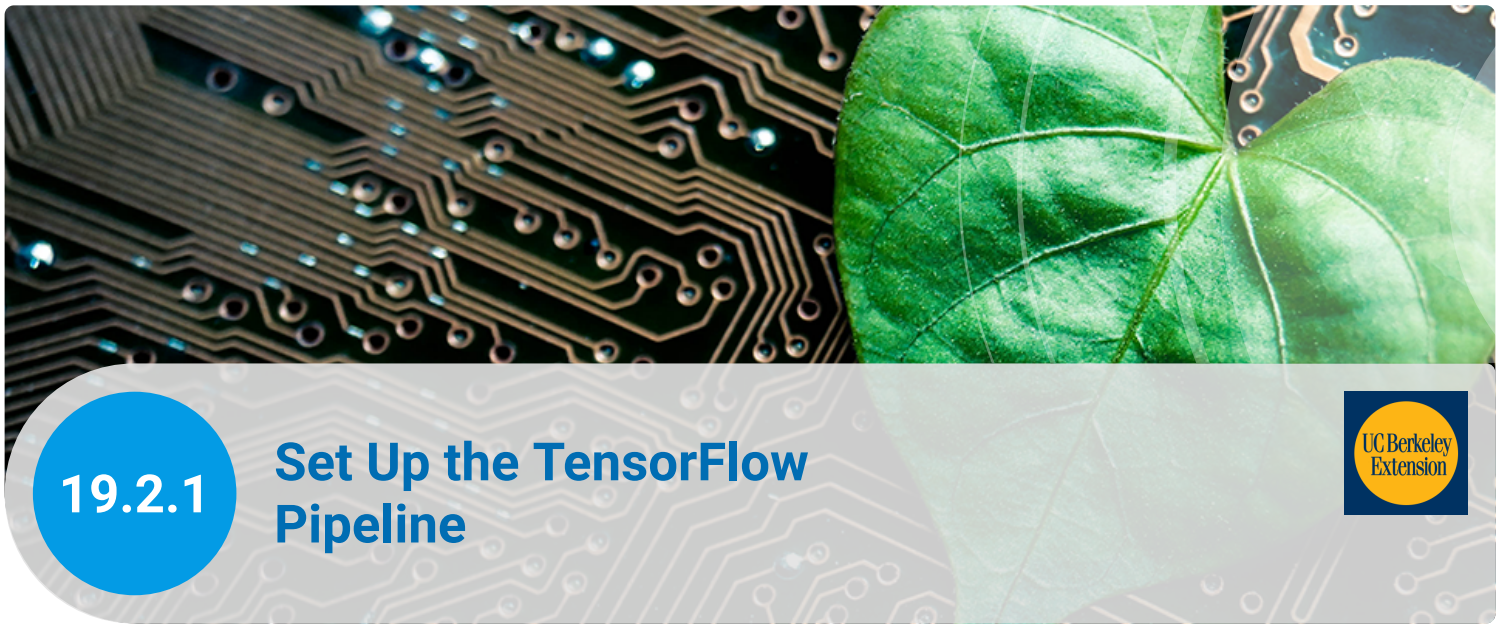


19.2.1 Set Up the TensorFlow Pipeline

Beks knows from her boot camp experience that there's nothing quite like hands-on experience when it comes to coding. So, now that she feels comfortable playing with TensorFlow, she's ready to start the REAL fun: building her first neural network!

To build our first neural networks in Python, we must first get acquainted with the TensorFlow 2.0 programming library. TensorFlow 2.0 provides users an easy-to-use application programming interface (API) that can design, train, export, and import neural network models without extensive machine learning knowledge. To install TensorFlow 2.0 into our Anaconda environment, we can use pip. Simply type the following command in your terminal (for Mac OS X users) or your Git Bash window (for Windows users):

```
# Installs latest version of TensorFlow 2.X
pip install --upgrade tensorflow
```

Run this code to check if your TensorFlow version is installed correctly. Did you get an error?

```
python -c "import tensorflow; print(tensorflow.__version__)"
```

- ○ Yes.
- ○ No.

Check Answer

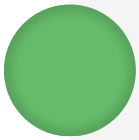
Finish ▸

There are a number of smaller modules within the TensorFlow library that make it even easier to build machine learning models. For our purposes, we'll use the Keras module to help build our basic neural networks. Keras contains multiple classes and objects that can be combined to design a variety of neural network types. These classes and objects are order-dependant, which means that depending on what Keras objects are used (and in what order), the behavior of the neural network model will change accordingly. For our basic neural network, we'll use two Keras classes:

- The **Sequential** class is a linear stack of neural network layers, where data flows from one layer to the next. This model is what we simulated in the TensorFlow Playground.
- The generalized **Dense** class allows us to add layers within the neural network.

With the Sequential model, we'll add multiple Dense layers that can act as our input, hidden, and output layers. For each Dense layer, we'll define the

number of neurons, as well as the activation function. Once we have completed our Sequential model design, we can apply the same Scikit-learn **model -> fit -> predict/transform** workflow as we used for other machine learning algorithms.



REWIND

The process of **model -> fit -> predict/transform** follows the same general steps across all of data science:

1. Decide on a model, and create a model instance.
2. Split into training and testing sets, and preprocess the data.
3. Train/fit the training data to the model. (Note that "train" and "fit" are used interchangeably in Python libraries as well as the data field.)
4. Use the model for predictions and transformations.