TensorFlow

Before you begin: TensorFlow 2.0 and this course

Which TensorFlow version this bootcamp uses 2.0 or 1.x.

The answer is: Both!

We've designed the curriculum for the early modules in this course to have code that's generic enough that it works with both versions, so you are welcome to try it with the 2.0 alpha. If you are using the colabs, you'll use the latest version of TensorFlow that Google Colaboratory supports, which, at time of writing, is 1.13. You can replace this with 2.0 by adding a code block containing:

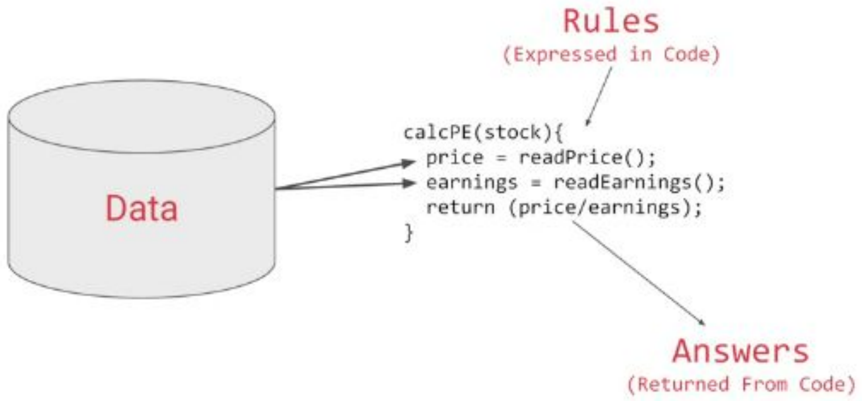
```
>>> !pip install tensorflow==2.5.0
```

... and you should be able to use TF2.0 alpha if you like!

Later modules may use specific versions of libraries, some of which may require TF2.0, and we will note them when you get there!

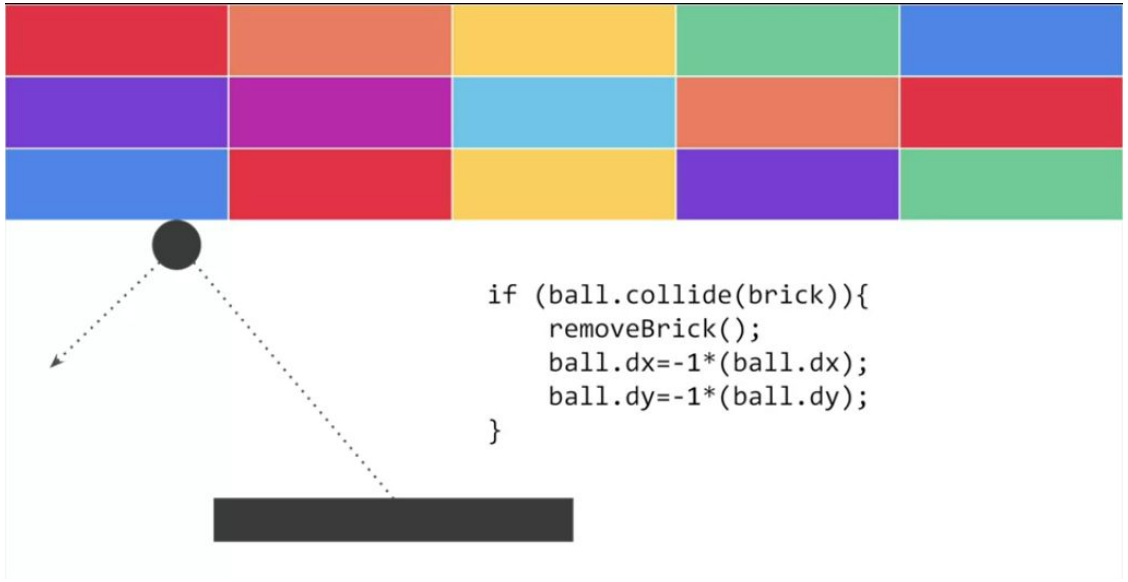
Welcome to TensorFlow, from basics to mastery.

Some of you may have taken deep learning or machine learning from me and learned about the amazing things you can now do with deep learning machine learning. One of the best tools you can use to implement these algorithms is ***TensorFlow.*** Learning algorithms can be quite   complicated, and today, programming frameworks like *TensorFlow, PyTorch, caffe,* and many others can save you a lot of time. These tools can be complicated and what this set of courses will do is teach you how to use *TensorFlow* effectively.



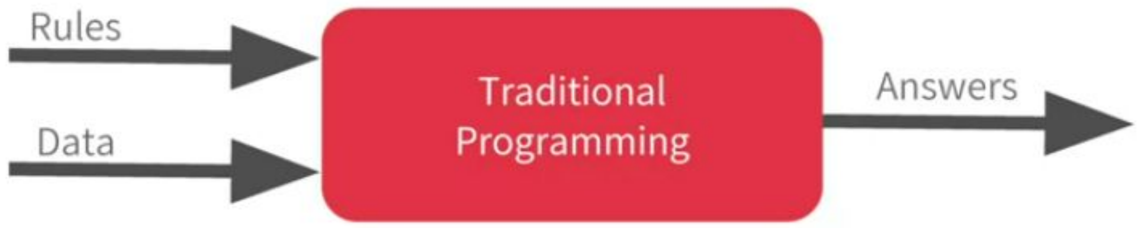
```
calcPE(stock){
  price = readPrice();
  earnings = readEarnings();
  return (price/earnings);
}
```

if we have to write an application that figures out a ***stock analytic***, maybe the price divided by the ratio, we can usually write code to get the values from a data source,do the calculation and then return the result.

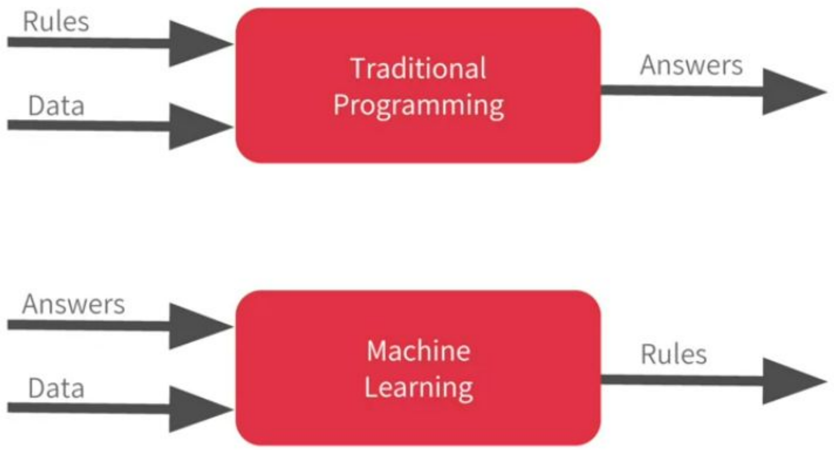


```
if (ball.collide(brick)){
    removeBrick();
    ball.dx=-1*(ball.dx);
    ball.dy=-1*(ball.dy);
}
```

If we're writing a game we can usually figure out the rules. For example, if the ball hits the brick then the brick should vanish and the ball should rebound. But if the ball falls off the bottom of the screen then maybe the player loses their life. We can represent that with this diagram. Rules and data go in answers come out.



Rules and data go in answers come out. ***Rules*** are expressed in a programming language and ***data*** can come from a variety of sources from local variables all the way up to databases.



Machine learning rearranges this diagram where we put ***answers in data in and then we get rules out***. So instead of us as developers figuring out the rules when should the brick be removed, when should the ***player's life end***, or what's the ***desired analytic*** for any other concept, what we will do is we can get a bunch of examples for what we want to see and then have the computer figure out the rules

Activity Recognition

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

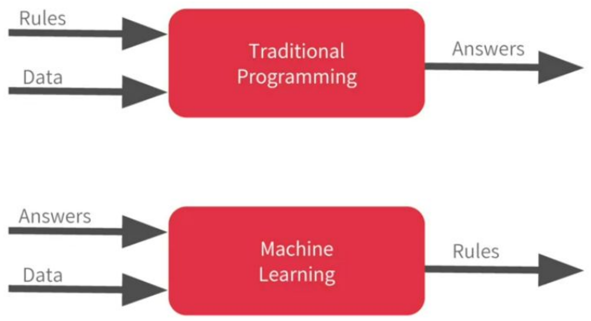
```
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

```
// Oh crap
```

In previous example, ***activity recognition.***

- If I'm building a device that detects if somebody is say walking and I have data about their speed,I might write code like this and,
- if they're running well that's a faster speed so I could adapt my code to this and,
- if they're biking, well that's not too bad either. I can adapt my code like this.
- But, then I have to do ***golf recognition*** too, now my concept becomes broken.

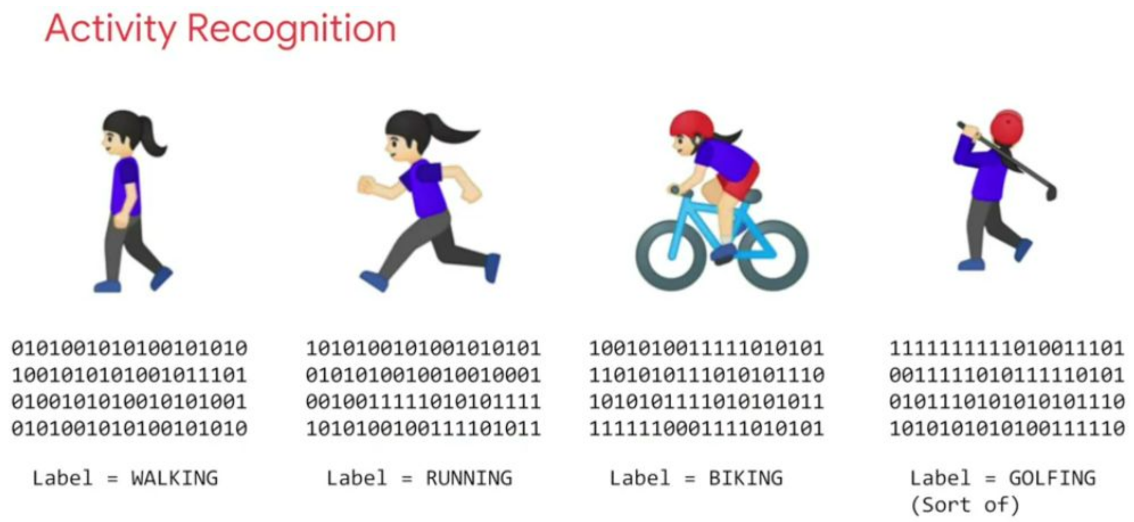
NOTE: But not only that, doing it by speed alone of course is quite naive. We walk and run at <u>different speeds uphill and downhill</u> and other people walk and run at different speeds to us.



How we will recognize this ?

Ultimately machine learning is very similar but we're just *flipping the axes.*

- So instead of me trying to express the problem as rules when often that isn't even possible, I'll have to compromise.
- The new paradigm is that I get lots and lots of examples and then I have labels on those examples and I use the data to say this is what walking looks like, this is what running looks like, this is what biking looks like and yes, even this is what golfing looks like.
- So, then it becomes answers and data in with rules being inferred by the machine.
- “ *A machine learning algorithm then figures out the specific patterns in each set of data that determines the distinctiveness of each*”.



<u>"A machine learning algorithm then figures out the specific patterns in each set of data that determines the distinctiveness of each</u>". That's what's so powerful and exciting about this programming paradigm. It's more than just a new way of doing the same old thing.

- It opens up new possibilities that were infeasible to do before.

Hello world to Deep learning

The basics of creating a neural network which is the workhorse of doing this type of pattern recognition. *A neural network is just a slightly more advanced implementation of machine learning and we call that deep learning*. But fortunately it's actually very easy to code. So, we're just going to jump straight into **deep learning.**

- Earlier we mentioned that machine learning is all about a computer learning the patterns that distinguish things. Like for activity recognition, it was the pattern of walking, running and biking that can be learned from various sensors on a device. To show how that works, let's take a look at a set of numbers and see if you can determine the pattern between them.

```
X = -1,  0, 1, 2, 3, 4
Y = -3, -1, 1, 3, 5, 7
```

- Well, the answer is Y equals 2X minus 1.Congratulations, you've just done the basics of machine learning in your head. So let's take a look at it in code now

Here's our first line of code. This is written using *Python* and *TensorFlow* and an API in TensorFlow called *keras*.Keras makes it really easy to define neural networks.

- “ A neural network is basically a set of functions which can learn patterns”.

```
>>> model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
```

- The simplest possible neural network is one that has only one neuron in it, and that's what this line of code does.
- In keras, you use the word dense to define a layer of connected neurons. There's only one dense here. So there's only one layer and there's only one unit in it, so it's a single neuron.
- Successive layers are defined in sequence, hence the word sequential.
- You define the shape of what's input to the neural network in the first and in this case the only layer, and you can see that our input shape is super simple. It's just one value.

You've probably seen that for machine learning, you need to know and use a lot of math, calculus probability and the like. It's really good to understand that as you want to optimize your models but the nice thing for now about TensorFlow and keras is that a lot of that math is implemented for you in functions. There are two function roles that you should be aware of though and these are loss functions and optimizers. This code defines them.

```
>>>  model.compile(optimizer='sgd', loss='mean_squared_error')
```

- The *neural network has no idea* of the relationship between X and Y, so it makes a guess.
- It will *then use the data* that it knows about,that's the set of Xs and Ys that we've already seen to measure how good or how bad its guess was.
- The ***loss function*** measures this and then gives the data to the optimizer which figures out the next guess.
- So the ***optimizer*** thinks about how good or how badly the guess was done using the data from the loss function & gradually after fine tuning it reaches approx 100% accuracy.
- In this case, the loss is mean squared error and the optimizer is SGD which stands for stochastic gradient descent.

Our next step is to represent the known data. These are the Xs and the Ys that you saw earlier. The np.array is using a Python library called numpy that makes data representation particularly enlists much easier. So here you can see we have one list for the Xs and another one for the Ys.

```
>>> xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)

>>> ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
```

The training takes place in the fit command. Here we're asking the model to figure out how to fit the X values to the Y values.

```
>>> model.fit(xs, ys, epochs=500)
```

The epochs equals 500 value means that it will go through the *training loop 500 times*. This training loop is what we described earlier. *Make a guess, measure how good or how bad the guesses with the loss function, then use the optimizer and the data to make another guess and repeat this*

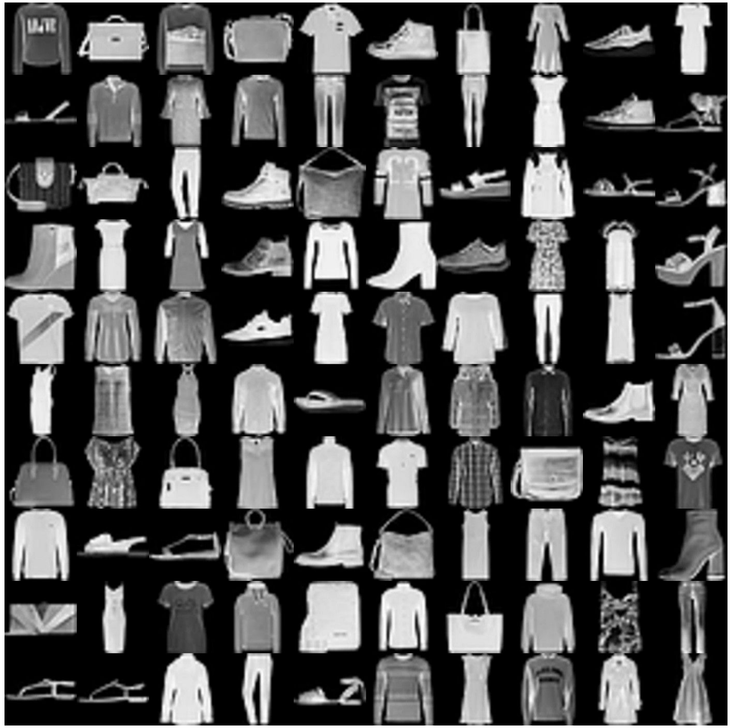
When the model has finished training, it will then give you back values using the predict method. So it hasn't previously seen 10, and what do you think it will return when you pass it a 10?

An Introduction to Computer Vision

- You learned what the machine learning paradigm is and how you use data and labels and have a computer in fair the rules between them for you. You looked at a very simple example where it figured out the relationship between two sets of numbers. Let's now take this to the next level by solving a real problem, computer vision
- Computer vision is the field of having a computer understand and label what is present in an image

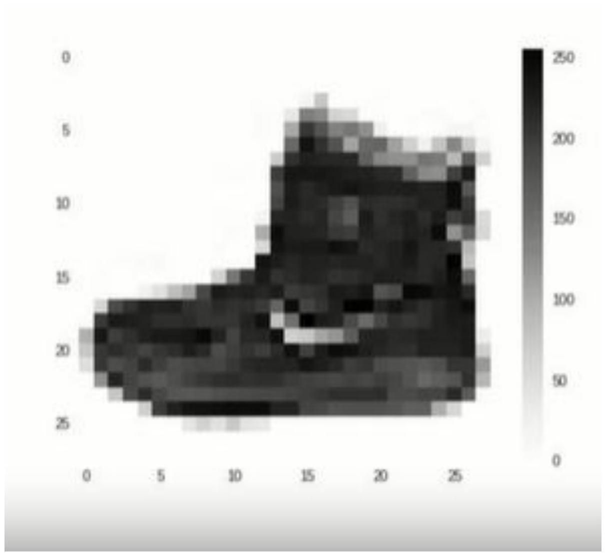
FASHION MNIST

- 70k Images
- 10 Categories
- Images are 28 x 28
- Can train a neural net



FASHION MNIST

- 70k Images
- 10 Categories
- Images are 28 x 28
- Can train a neural net
- Gray scale Images.
- 1 pixel have values from 0 to 255.
- 1 pixel per byte



RECAP

- In Last Example we had simply trained an NN using small dataset that has only six values.
- Now, We’ll gonna dig deeper and analyze 70k images of Fashion MNIST dataset.
- Fortunately, it's still quite simple because Fashion-MNIST is available as a data set with an API call in TensorFlow