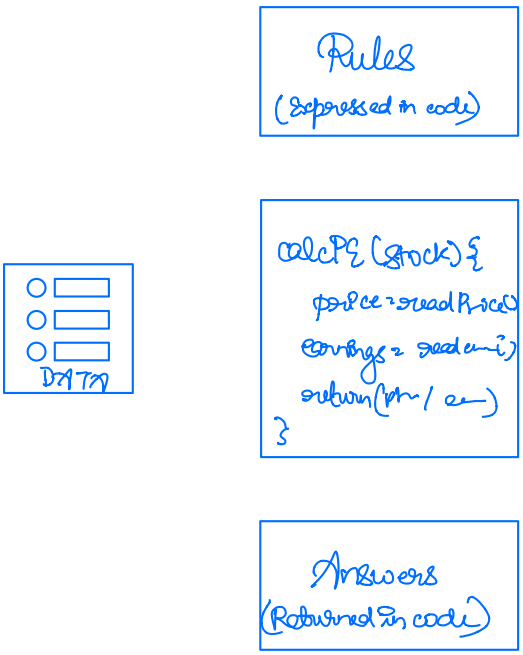
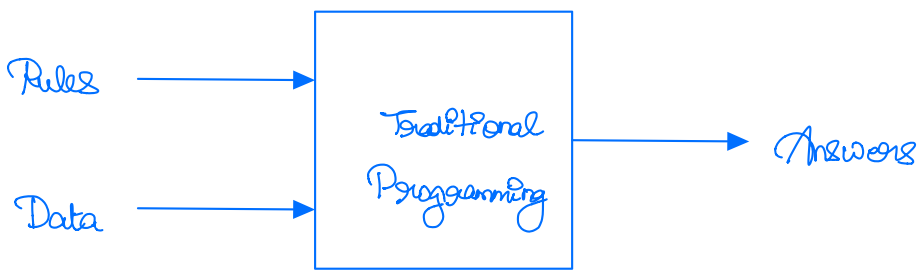
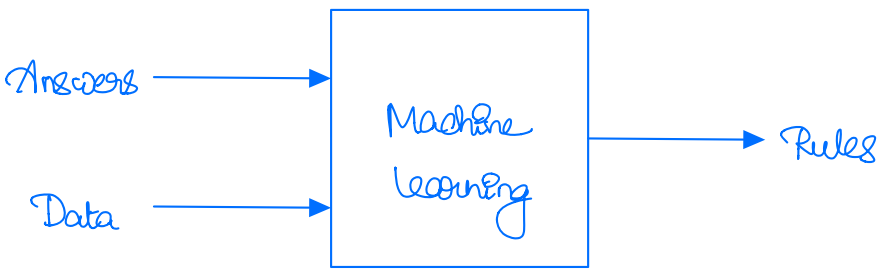
Traditional Programming





Machine Learning



Example: Activity Recognition

In traditional programming, you can tell if a person is walking by checking if their speed is less than 4, running if speed is > 4 but < 12, and cycling if more than 12.

But what in case of golfing? Similarly, the speed of a cycle can depend upon inclination and surface friction, and people walk at different paces.

```
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;
}
```

```
// Uh oh
```

What if we give our data to Machine learning?

It could look like this:

```
0101001010100101010
0100101010100101111
0101001010100101010
0010101001010100101
01010
 Label = WALKING
```

```
1010100101001010101
0101010010010010010
0100100111110101010
1111010100100111101
11
 Label = RUNNING
```

```
1001010011111010101
1110101011101010110
1010101011110101010
0111111110001111010
101
 Label = BIKING
```

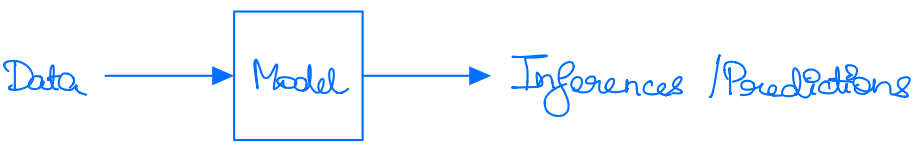
```
1111111111010011101
1001111101011111010
0101011101010101010
1101010101010100111
10
 Label = GOLFING
 (Sort of)
```

The ML algorithm can look for patterns in the data and assign them to labels, and predict future labels through those learnt patterns, and this can be more suitable than writing rules manually.

The Machine Learning Paradigm

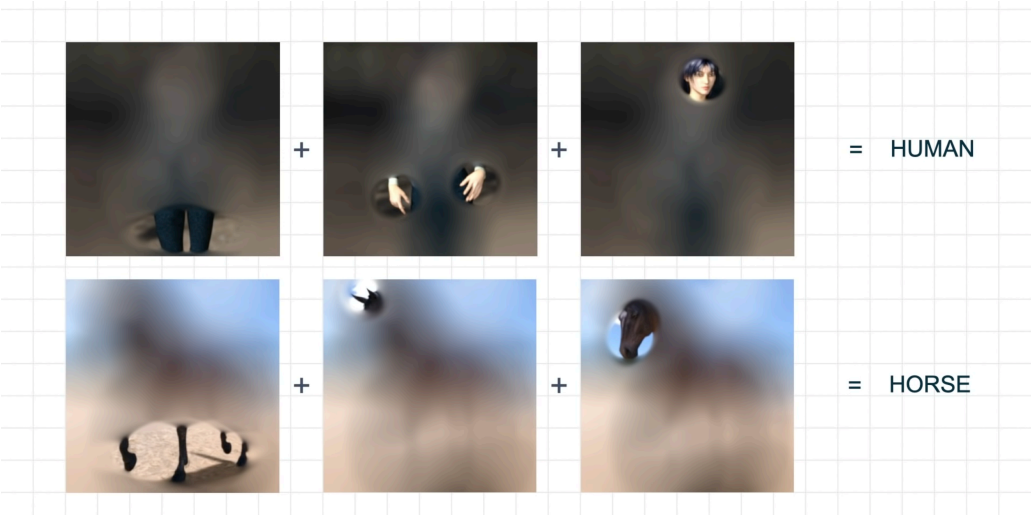
Repeat

When we do this, we get a model



Computer Vision

Images are just a bunch of numbers. We can add filters to images that a computer can detect that helps a computer understand the image.



If the computer can figure out which filters help extract features that determine between answer of human or horse, then it can differentiate and in future predict human or horse.

In summary,:

- Filters extract features like hands or ears.
- Filters can then be combined with labels to make a prediction of image contents.
- Filters that match the labels are learnt overtime. These filters are randomly initialized but overtime, using loop earlier, their values can be measured for accuracy and optimized.

In terms of code:

In tensorflow, we can define a model as a number of layers, where each layer is a piece of functionality that define what we want a machine to learn.

```
model = tf.keras.models.sequential( [
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
                           input_shape=(300,300,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation-'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation-'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid'),
])
```

Conv2D stands for 2D Convolution, which is another word for a filter. Since this is a sophisticated scenario with detailed images, we have to apply filters multiple times.

Max Pooling is a way of compressing the image while enhancing features. It is optional but highly useful, as it means that the next steps have less pixels to work with, while still having features.

After all of the filters are learnt, then we use the dense neural network that matches filters to labels, so human legs get matched with humans, and horse legs with horse, and so on.

The second dense is the output of the neural network that is also a neuron. As we only have 2 classes here (horse & human), we can do it with a single neuron here, where if the value is closer to 0, then it's a horse, and if value is closer to 1, then it is a human.

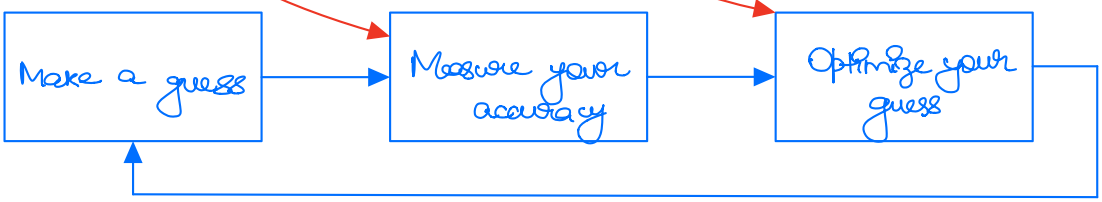
Once a model is defined, we have to define how it works with training.

```
optimiser = tf.keras.optimizers.RMSprop(lr=0.001)
model.compile(loss='binary_crossentropy',
              optimizer = optimizer,
              metrics=['accuracy'])
```

RMSprop stands for Root Mean square propagation and is an optimizer.

(1) loss function (using Binary Crossentropy as there are 2 classes

(2)



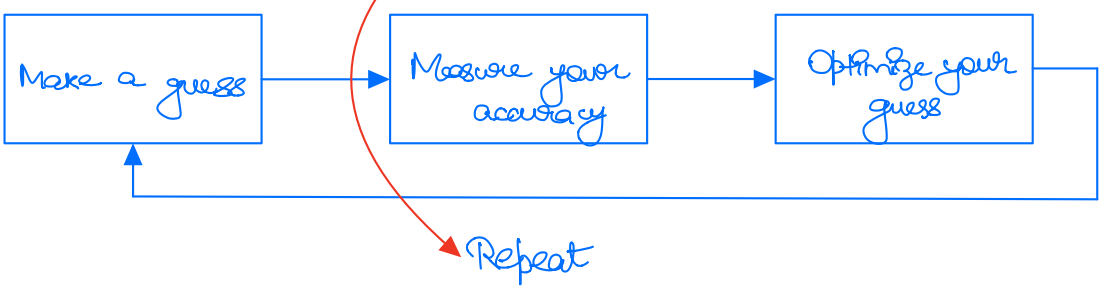
Repeat

Optimizers takes in the (randomly initialised) parameters accuracy, and update the parameters with a new guess.

Next step is to fit the answers to the labels. In order to figure out the rules:

```
model.fit(train_generator, epochs=12, ...)
```

Defines how many times to repeat the process.



As the model gets better at matching the data to the label, the process convergence is used.

It takes time to figure out how many epochs, what model architecture, what parameters etc. will lead to convergence. A skilled machine learning developer should be able to reach convergence quickly.