4.0 Errors in (Computer) Science

October 5, 2018

1 Errors when dealing with data

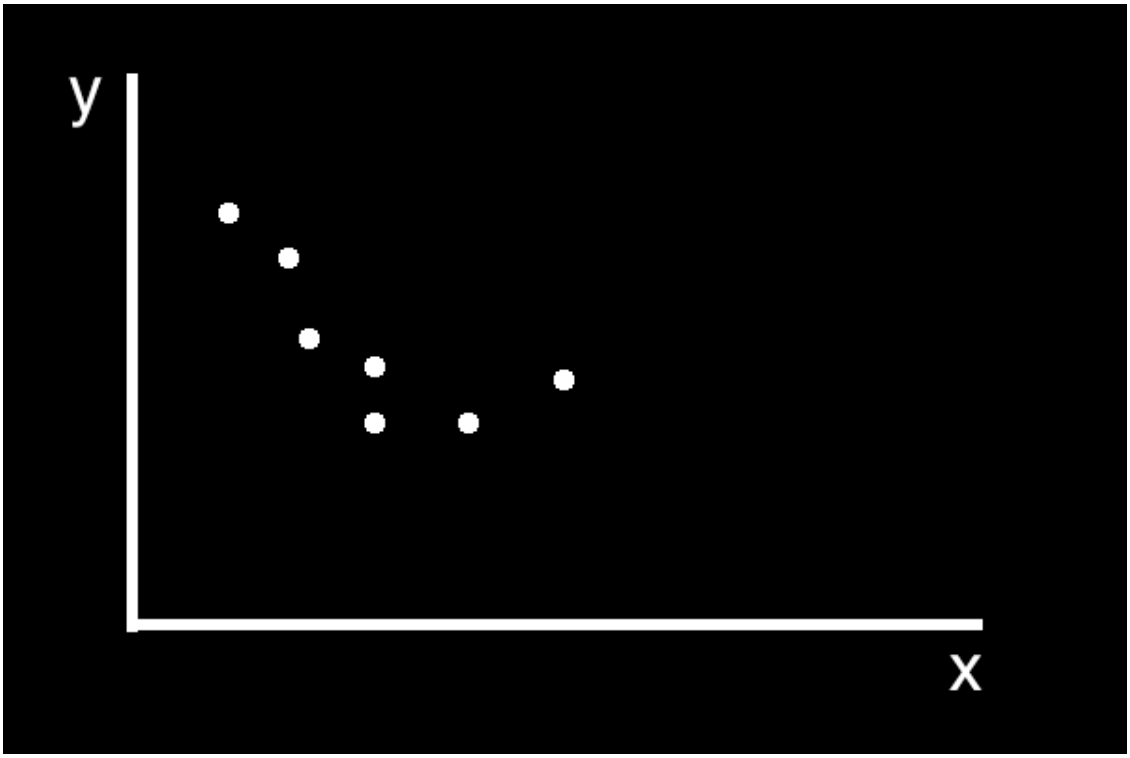
There are four main types I can think of: 1. Extrapolation 2. Over/under fitting the data 3. Under sampling the distribution 4. Systematics

Only one of these types (2) occurs just between you and your computer. People often worry a lot about these kinds of errors but miss huge problems due to other kinds of errors, don't make this mistake!

1.0.1 Extrapolation

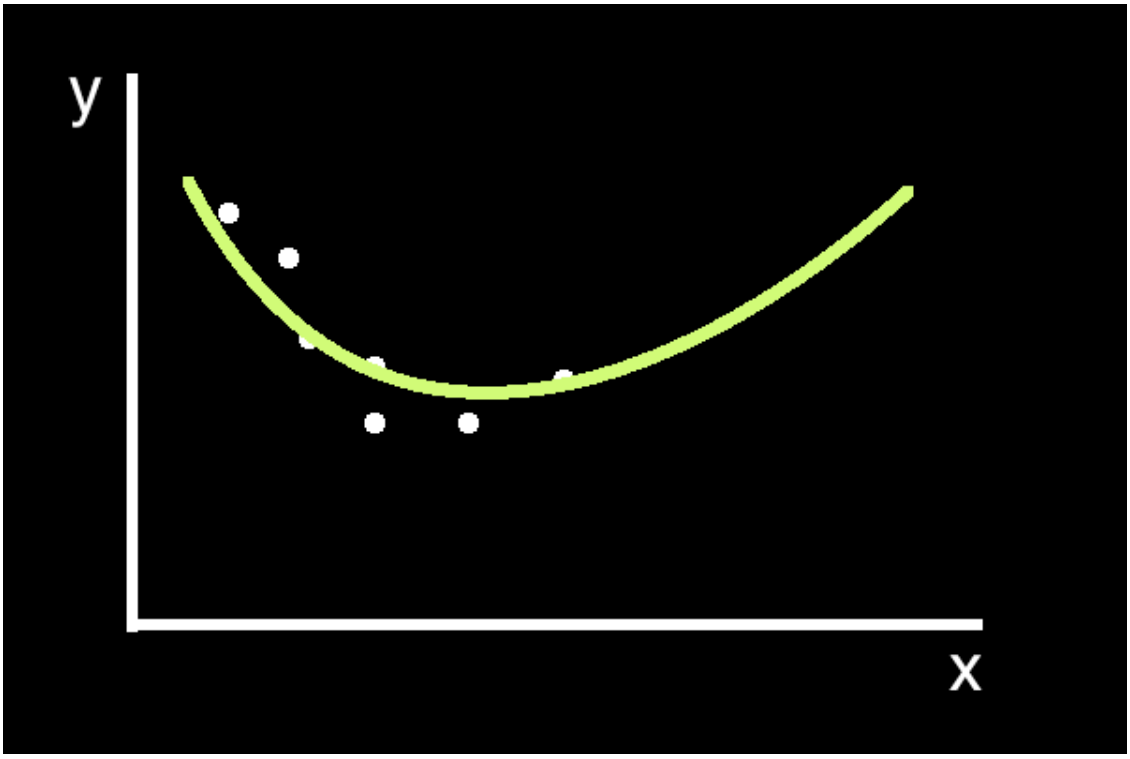
```
In [1]: from IPython.display import Image
        Image(filename='pics/data_0.png')
```

Out[1]:



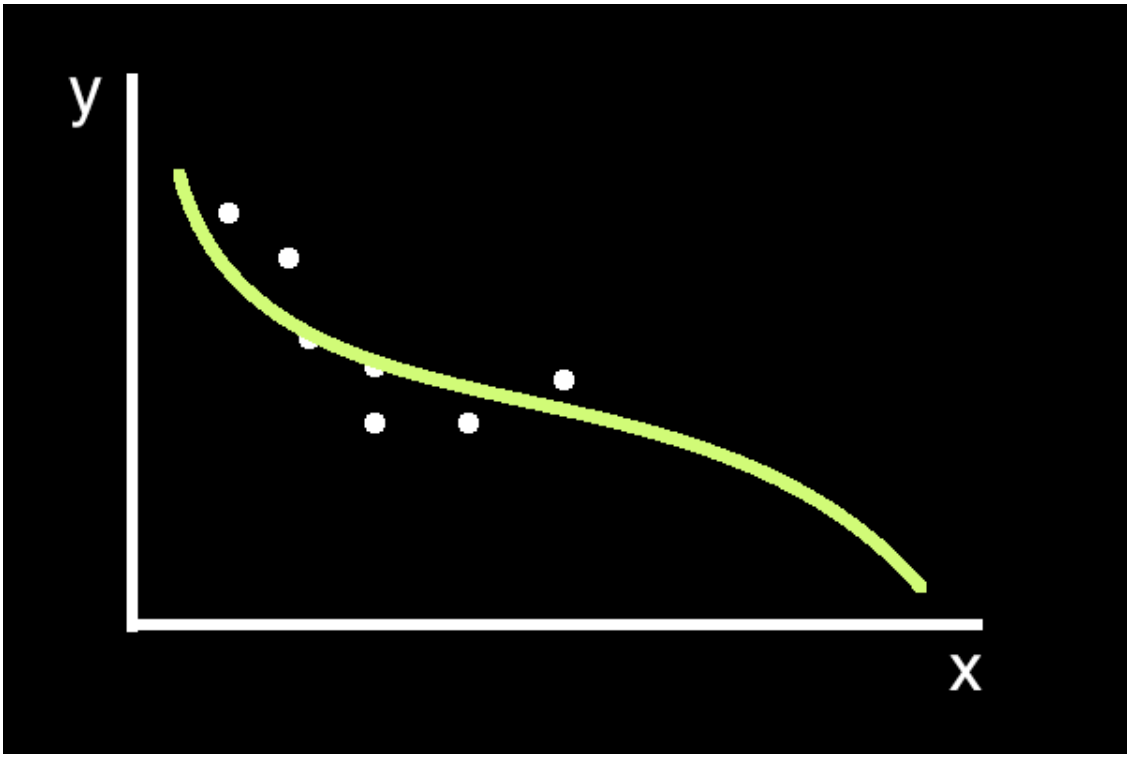
In [2]: Image(filename='pics/data_1.png')

Out[2]:



In [3]: Image(filename='pics/data_1a.png')

Out[3]:



In [4]:
```
# from metro.co.uk
Image(filename='pics/iphone_chinese.png')
```

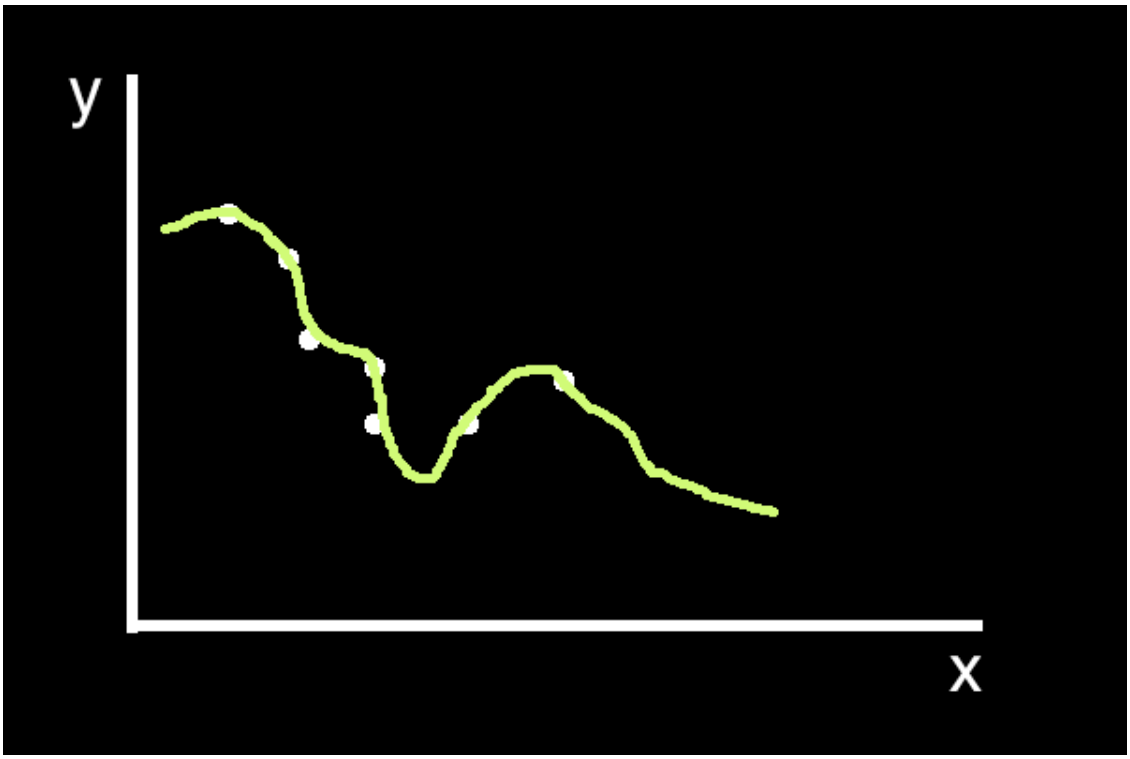
Out[4]:

This mother and son claim the iPhone X's facial recognition system cannot distiuignish between them

1.0.2 Over/Under fitting Data

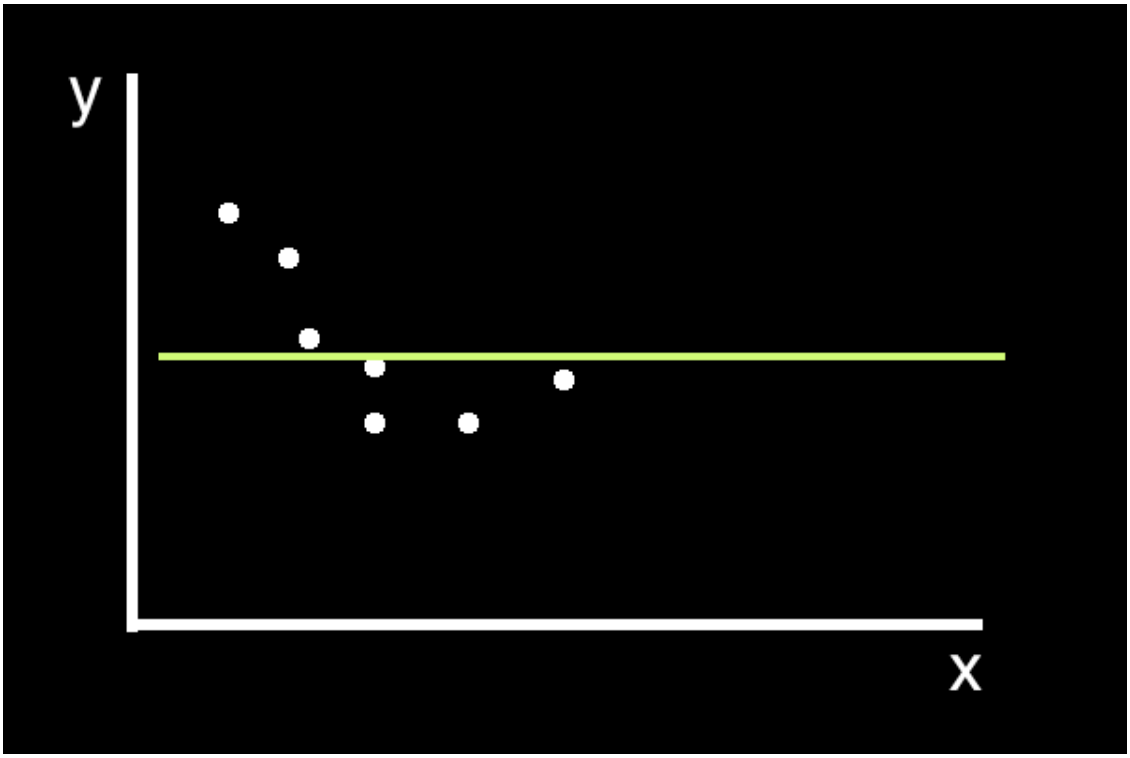
In [5]: Image(filename='pics/data_overfit.png')

Out[5]:



In [6]: Image(filename='pics/data_underfit.png')

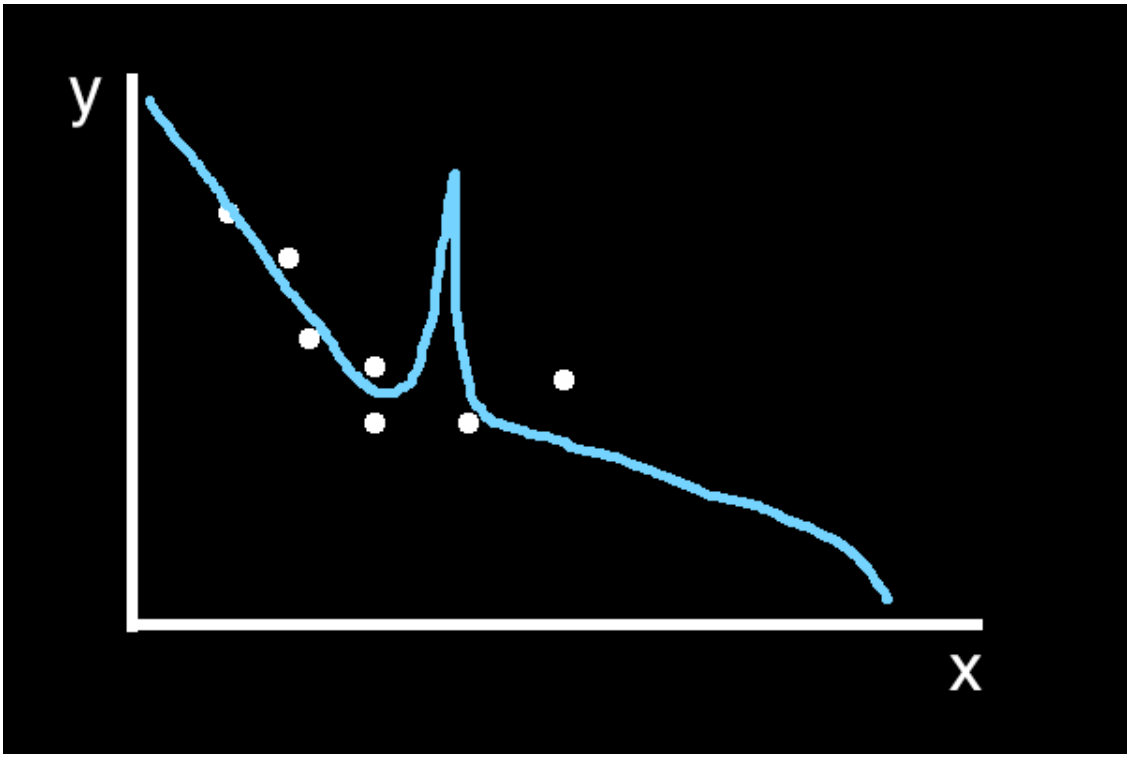
Out[6]:



1.0.3 Under sampling distribution

```
In [7]: Image(filename='pics/data_undersample.png')
```

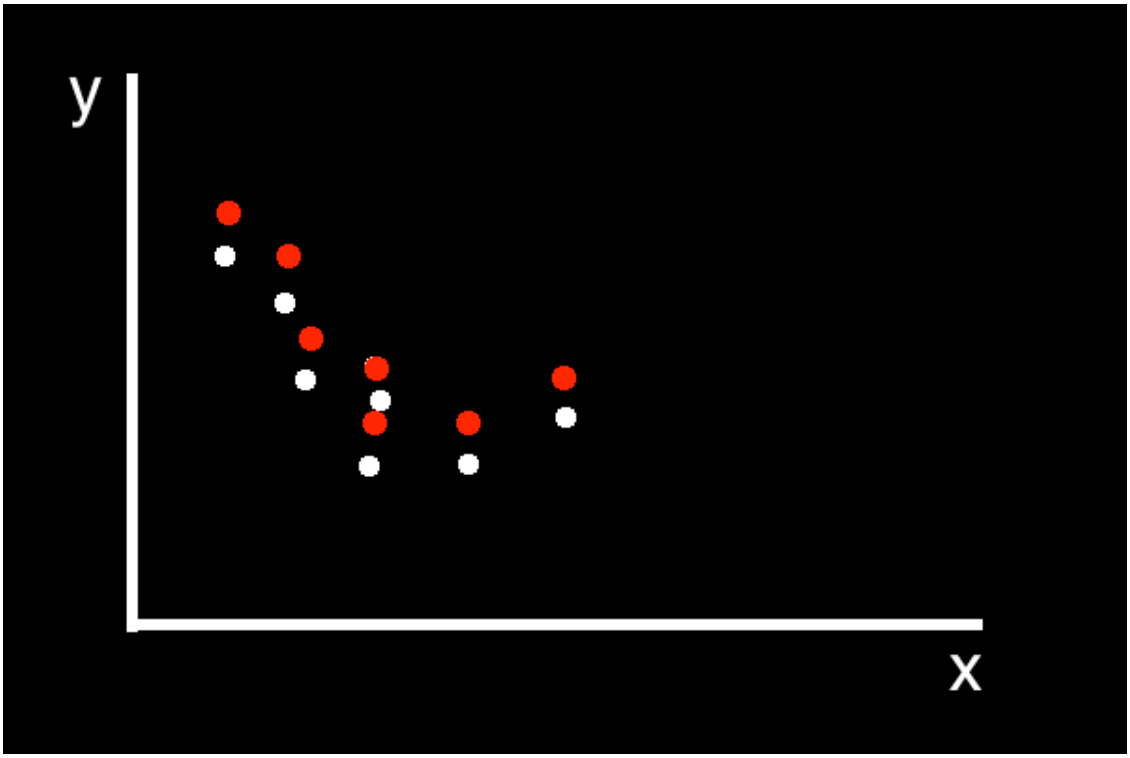
Out[7]:



1.0.4 Systematic Errors (and poisoning)

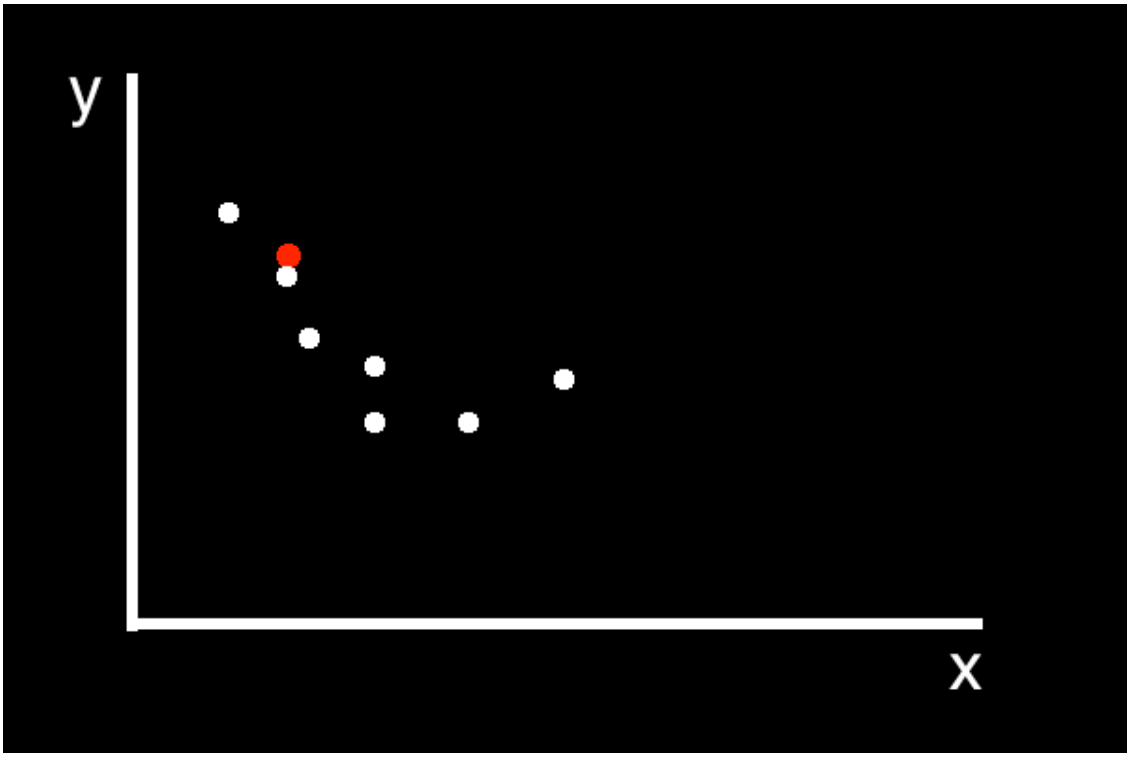
In [8]: Image(filename='pics/data_poison_all.png')

Out[8]:



In [9]: Image(filename='pics/data_poison_1.png')

Out[9]:



1.1 Quantifying Errors

How do we check if our model is working and generalising? It is not enough just to run it on our training data. Split your data into 3 parts:

1. Training data
2. Test data
3. Blind data

Training data Use to train your models.

Test data Use to evaluate your models - probably you look at this multiple times but it is never directly used for training.

Blind data Preferably you can't access until you're about to publish your results! Only look at once, and after all models are frozen.

1.1.1 Precision

Sometimes called purity. The fraction of the things you selected were actually what you thought you selected:

$$\text{precision} = \frac{\text{true}_s}{\text{true}_s + \text{other}_s}$$

$$s = \text{selected} u = \text{unselected} \text{true} = \text{what we were looking for} \text{other} = \text{anything else}$$

1.1.2 Recall

Sometimes called efficiency. The fraction of the things you wanted to select that you did select.

$$\text{efficiency} = \frac{\text{true}_s}{\text{true}_s + \text{true}_u}$$

1.1.3 What to maximise?

Do not just maximise precision or recall. Why? Instead maximise some version of the F score:

$$F_\beta = (1 + \beta^2) \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

If we want to give a higher weight to recall we choose a higher value for β, *e.g.* $\beta = 2$ and if we want to give a higher weight to precision we use a lower value for β, *e.g.* $\beta = 0.5$.

1.2 Neural Networks: Regularisation and Dropout

1.2.1 Regularisation

The idea is to add a penalty term to the loss function you are trying to minimise. This has the effect of penalising large weights.

$$J \to J + \lambda |w|^2$$

In Keras:

```
In [ ]: from keras import regularizers
        model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)))
```

1.2.2 Dropout

This is where you make the network randomly cope without parts of its brain. Sounds cruel but it means that it can't overly rely on any single neuron (so it shouldn't just rely on one pixel/feature *etc.*)

In Keras:

```
In [ ]: from keras.layers import Dropout
        dropout_rate = 0.1
        model.add(Dropout(dropout_rate))
```

The variable `dropout_rate` is how many of the weights to randomly set to zero during training.

```
In [ ]:
```