## Batch VS Epoch

| Example  Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.  This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples.  This also means that one epoch will involve 40 batches or 40 updates to the model.  With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process |
| --- |

You can have multiple updates for a single epoch. 1 epoch just means that it passed through the entire dataset once

Batch Gradient Descent

* You perform an update after going through a single batch (5 samples)

Stochastic Gradient Descent

* Perform an update after going through a single example

| In tensorflow you see this:  Epoch 1/20  468/469 [============================>.] - ETA: 0s - loss: 0.0012 - accuracy: 1.0000  And it means that for every epoch, there are 469 batches and there will be 469 updates performed  (That’s why you see the value of acc and loss changing even when its on the same epoch) |
| --- |

## One Hot Encoding

Tf one\_hot VS to\_categorical

To\_categorical

* Converts a class vector ([3, 7, 8, 9])

to a class matrix

[[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 1.]]

* Can say that the to\_categorical is fitted on the class vector and will output a one hot encoded matrix based on the values of the class vector (if num\_classes=4, doesnt mean it has to be 1,2,3,4!! It can be any 4 numbers, just that the lowest number will take the lowest index for the one hot encoded version!)

| tf.keras.utils.to\_categorical(y, num\_classes=None)  Num\_classes will be the depth of the one hot dimension  E.g if num\_classes = 3, then the one hot for 3 in [1,3,5] will be [0,1,0]  a = tf.keras.utils.to\_categorical([3, 7, 8, 9], num\_classes=4)  print(a)  tf.Tensor(  [[1. 0. 0. 0.]  [0. 1. 0. 0.]  [0. 0. 1. 0.]  [0. 0. 0. 1.]], shape=(4, 4), dtype=float32) |
| --- |

One\_hot

* Returns a one\_hot tensor (It takes nD tensor as input and returns n+1 D one hot encoded tensor.)

| tf.one\_hot(indices = [0, 1, 2], depth = 3)  [[1., 0., 0.],  [0., 1., 0.],  [0., 0., 1.]] |
| --- |

## Tensor

* An array of n Dimensions, typically where n > 3

## Numpy Array Dimensionality

| * X[0][0] is the same as x[0,0]      * This will index the first element of the first dimension (Which is the biggest outer bracket)        * This will return the 1st and 2nd arrays of the first dimension      * This will get the first 2 elements in the first dimension      * And then get the last element in the 2nd dimension      * And get the first element in the 3rd dimension     So, basically it works as such   * The outer most bracket will be the first dimension * As the brackets move inwards, your dimension increases * For the example above, [ [ [ ] ] ] where first dimension, second dimension, third dimension     With reference to image above   * If you index, you get the element itself (2D array) * If you slice, you get the elements which are put under one big bracket (3D array)        * Access the 1st element in the 1st dimension (So you look at the 2nd outermost brackets     and index the first 2nd outermost bracket)   * Access the 1st element in the 2nd dimension (So you look at the 3rd outermost brackets     and index the first 3rd outermost bracket)    What shape = (3,5,2) means is that there are:   * 3 elements in the first dimension (So there are three brackets in the outermost bracket) * 5 elements in the second dimension (So there are five brackets in each 2nd outer most bracket) * 2 elements in the third dimension (So there are 2 elements in each of the 3rd outer most brackets) |
| --- |

## 

## Difference between fit and predict

| When you fit, you pass in both the x and y values  When you predict, you pass in the x values only  **However, the format of x will be the same for both training and prediction!** (same size, shape, etc.) |
| --- |