[A1]

- Keeping labels as directly numbers is ok but it will make the model think that 9 has most value while 0 has least value, 0 to 9 numbers are not equally likely. Making a array with index as their actual label and the value of that index as binary is good as it represents if its there or not , it dont give any label priority.

[A2]

- Operations on numpy array are very very fast as in numpy arr, operations are performed at once (vectorised operations) , while in python lists elementwise operatios are performed .
- Also the numpy lib provides a lot of func to apply on a numpy array.
- Numpy also has built in n-d array support making it helpful for matrices than list

[A3]

- To remove the other chanels (maybe gb and alpha). Because we need to identify shape of the number (1/2/3...) , and we dont have any work from color grading so grayscale works too and dimension reduces from 28x28x4 to 28x28x1 making our work easier.

[A4]

- We are indirectly flattening each image, converting images from a 2d arr to a 1d arr so we can deploy neural network on it (as told in 3B1B video).

[A5]

- Learning rate is a *kind* of weight of gradient descent derivative. It controls how much to change (subtract or add) the weight while doing backpropagation. It controls the power of derivative of loss with respect to weight (or bias).

[A6]

- Dimensions of weight is such because there are i neurons in input layer , h in hidden , for each neuron in hidden layer , we will need i weights coming from each feature. So total weights required for hidden layer woudl be i times h. Now we will get a weight matrix wih , whose ath (a eth) row will have all weights required for the ath neuron in hidden layer , so each rows shape would be (1,i). And there would be h such rows because there are h neurons in hidden layer.
  So final dim of wih is (h, i).

Now for biases , each neuron in hidden layer will have a bias which will be added to the sigma(xi * wi). So bias matrix would be of dimension (h,1).

[A7]
What is broadcasting.

- Broadcasting is converting a matrix to appropriate size (depends on the other matrix) so an operand can be applied on it with the other matrix. Like for adding , [9,8,7] to [[0,0,0], [1,2,3], [4,5,6]] , the dimensions are not matched and '+' can operate with same dimensions so both matrix need to be of 3,3. Numpy do broadcasting internally, by itself , it copies the 987 matrix for two other rows to make it [987,987,987] (where 987 = [9,8,7]) so now each element by element can be added.

Broadcasting in bias.

- its not required/used when dealing with just one image/example as sigma w * x matrix comes out of dim 16,1 where 16 is no of neuron in hidden layer. And bias matrix is also of dim 16,1 so no broadcasting reqd. But for m examples , its required as feature matrix gets of dimension (784,m) and w * x get of (16,m) and bias is of 16,1 so it spans itself verticall to get of 16,m size.

[A8]
np.random.randn(x,y) gives random integer np array of shape (x,y). weights are initialised that way to noramlise them.

[A9]
The first reason to use activation functions is to introduce complexity (or non linearity). If we wont use any activation func in any layer then at last our func will just be of the form Z = WX + b. which is linear and is nothing other than a straight line so the model wont be able to predict complex functions.
Also if we wont keep the output of one layer in some limits it will just explode at one point or vanish because z1 = w1x0 + b1, z2 = w2*z1 + b2 --> z2 = w2*(w1x0 + b1) + b1 and it will keep on increasing or decreasing. To keep it within some limits we intorduce activation functions like sigmoid which clamp the inputs (output of previous layers) into range of 0 to 1. Because of this reason its mostly used in the last layer of binary classification also bcoz it gives output in range of 0 to 1 which can be taken as probability of that label.

Relu is used to remove negative activation neurons. It is so bcoz to introduce non linearity and avoid vanishing gradients. It saves computational power and time also. And it dont saturate outputs (just like sigmoid does ,which can also lead to vainishing outputs).

Leaky relu was introduced bcoz in relu , if input of a neuron is mostly negative then that neuron is always dead and never ables to learn anything and just becomes a dead neurons. To remove dead neuron problem, leaky relu was made so negative neurons' gradients also update but by a small factor.

[A10] **Recheck**
Softmax is a activation function mainly used at the last layer of neural nets. It turns list of numbers (here neurons) into probabilites which sum up to 1. If the problem is a binarry classification problem , we generally use sigmoid as it gives prob of 1st class and we get prob of second by subtracting 1 from it. Softmax will also give same result for multiple classes.

[A11]
Loss functions are used to calculate loss (or say inaccuracy) between predicted labels by the model and the actual model. The higher the loss , the more the model is predicting wrong labels so we want our loss (output of loss func) to be as minimum as possible. To do this , we take derivative of loss with our variables (parameters (w,b)). For maximum loss wrt to some particular values of weights and biases , derivative of loss wrt to w or b will be zero , so we update our params by subtracting learning rate times dl/dw (or dl/db) from the param to reduce the value of params so that at one point our dl/dw or dl/db gets close to zero.

[A12]
Cross entropy loss , also called log loss , is used for classification problems. Its of two types , binary and multi class. Basically binary = multi class but no of class = 2. For it , the yhat we give to it ,they should be output after applying softmax (or sigmoid for binaryclassification).

[A13]
To normalise inputs. Generally inputs are randomly spread over the planes but we normalize them by bringing them near to origin so their mean becomes 0 and training becomes stable. Initially inputs may be very very large and during forward prop, z = wx + b can explode so we normalise them first.

[A14]

Because it give probab of each label and sum of probab of all label comes out to be 1. One more reason is that combination of cross entropy loss function and softmax as activation gives very simpilified result when calculating derivative (i.e. yhat-y).

[A15]
We can but it dont give intermediate values like a1, z1 etc requird for backprop. I have made a forward_modified which can be used.

[A16]
We generally divide our data into 3 parts, train, validation, and test. On train data, we do training. (lmao). anyways , on train data, we predict output , calculate loss in predicted and real , then we update our parameters acc to that to minimise loss.

On validation data, we see how much loss is there between our predicted values and actual values. We then improve our model , on training data only, by hypertuning parameters like epochs (which we set a high number first and then reduce by early stopping {there are other methods to but this one is my first goto method}) , learning rate, etc

It has never been trained on our val data so we use it to measure how our model is **generalised** for any data. When we feel our model is nicely generalised then at last we test it on the very new data i.e. test data.

[A17]
Parameters are weights and biases which change over time for good results.
(epochs, learning rate etc are hyperparameters and not parameters)

[A18]

argmax gives the index of that col (if axis = 0) or row (if axis = 1) which has max value. in this case the index is label so it give label which has max probablity.