**Assignment Notes:**

Common steps for pre-processing a new dataset are:

* Figure out the dimensions and shapes of the problem (m\_train, m\_test, num\_px, ...)
* Reshape the datasets such that each example is now a vector of size (num\_px \* num\_px \* 3, 1)
* "Standardize" the data

The main steps for building a Neural Network are:

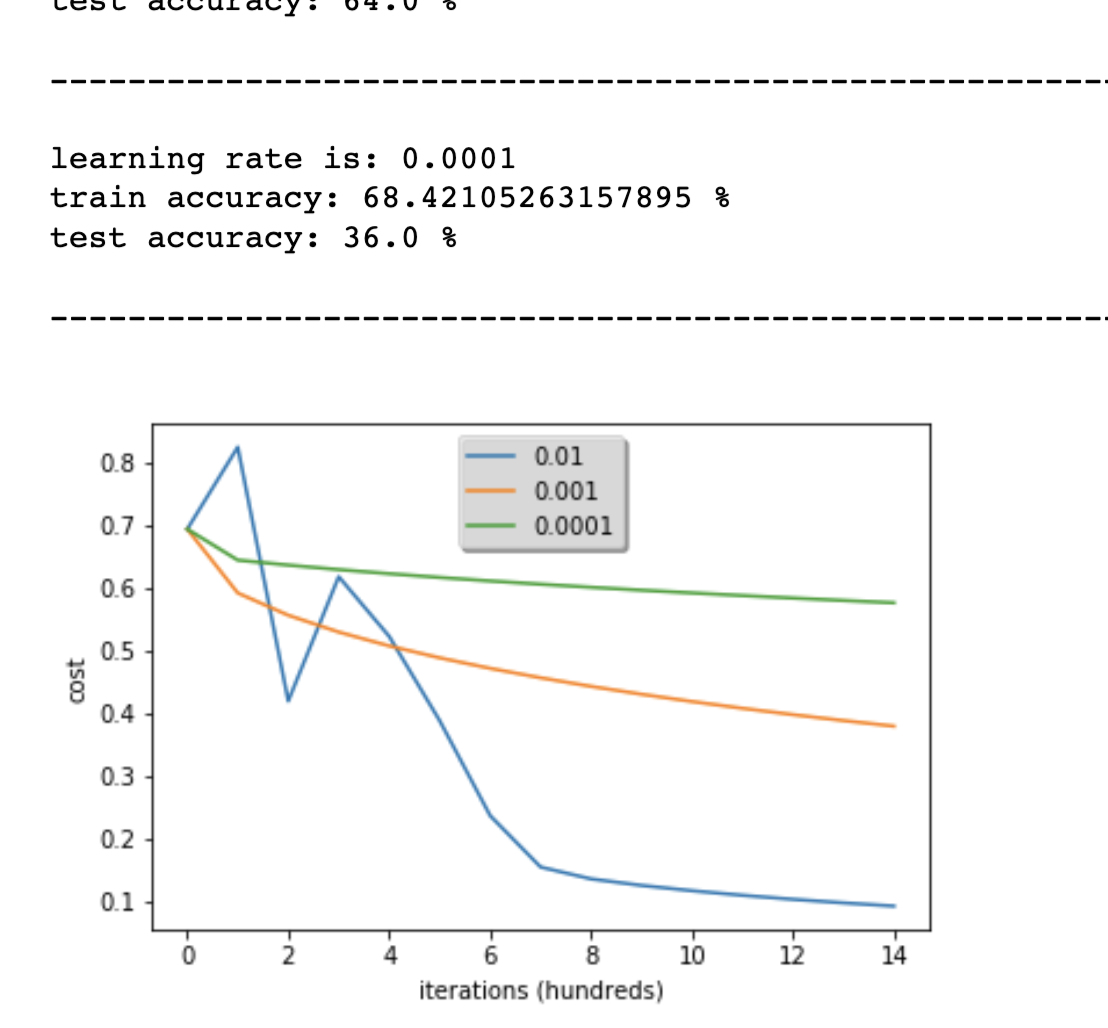
1. Define the model structure (such as number of input features)
2. Initialize the model's parameters
3. Loop:
   * Calculate current loss (forward propagation)
   * Calculate current gradient (backward propagation)
   * Update parameters (gradient descent)

You often build 1-3 separately and integrate them into one function we call model().

**Choice of learning rate**

**Reminder**: In order for Gradient Descent to work you must choose the learning rate wisely. The learning rate αα determines how rapidly we update the parameters. If the learning rate is too large we may "overshoot" the optimal value. Similarly, if it is too small we will need too many iterations to converge to the best values. That's why it is crucial to use a well-tuned learning rate.

Let's compare the learning curve of our model with several choices of learning rates. Run the cell below. This should take about 1 minute. Feel free also to try different values than the three we have initialized the learning\_rates variable to contain, and see what happens.

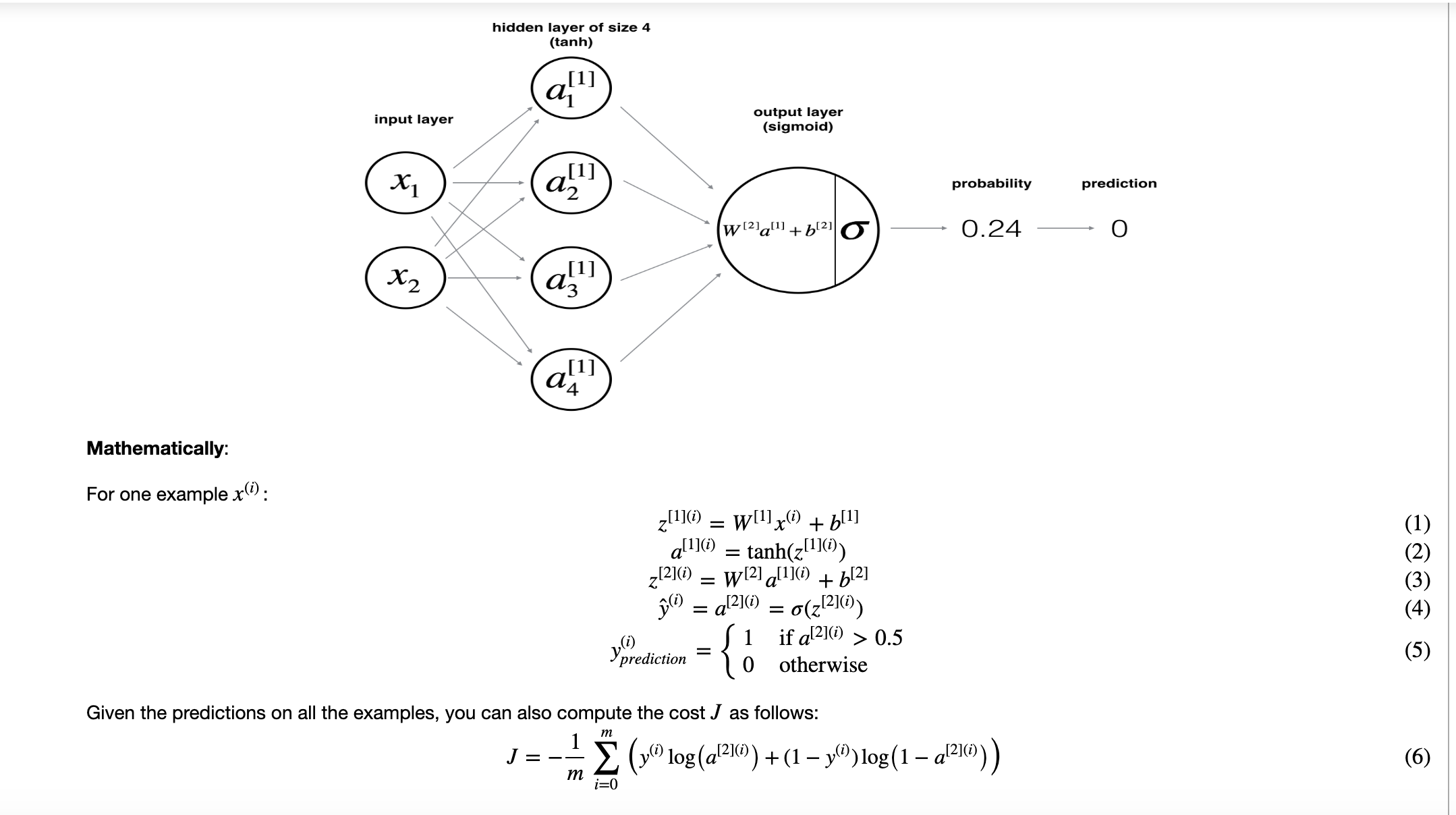


**Interpretation**:

* Different learning rates give different costs and thus different predictions results.
* If the learning rate is too large (0.01), the cost may oscillate up and down. It may even diverge (though in this example, using 0.01 still eventually ends up at a good value for the cost).
* A lower cost doesn't mean a better model. You have to check if there is possibly overfitting. It happens when the training accuracy is a lot higher than the test accuracy.
* In deep learning, we usually recommend that you:
  + Choose the learning rate that better minimizes the cost function.
  + If your model overfits, use other techniques to reduce overfitting. (We'll talk about this in later videos.)

**What to remember from this assignment:**

1. Preprocessing the dataset is important.
2. You implemented each function separately: initialize(), propagate(), optimize(). Then you built a model().
3. Tuning the learning rate (which is an example of a "hyperparameter") can make a big difference to the algorithm. You will see more examples of this later in this course!



**Reminder**: The general methodology to build a Neural Network is to:

1. Define the neural network structure ( # of input units, # of hidden units, etc).

2. Initialize the model's parameters

3. Loop:

- Implement forward propagation

- Compute loss

- Implement backward propagation to get the gradients

- Update parameters (gradient descent)

You often build helper functions to compute steps 1-3 and then merge them into one function we call nn\_model(). Once you've built nn\_model() and learnt the right parameters, you can make predictions on new data.

**4.1 - Defining the neural network structure**

**Exercise**: Define three variables:

- n\_x: the size of the input layer

- n\_h: the size of the hidden layer (set this to 4)

- n\_y: the size of the output layer

**Hint**: Use shapes of X and Y to find n\_x and n\_y. Also, hard code the hidden layer size to be 4.

**4.2 - Initialize the model's parameters**

**Exercise**: Implement the function initialize\_parameters().

**Instructions**:

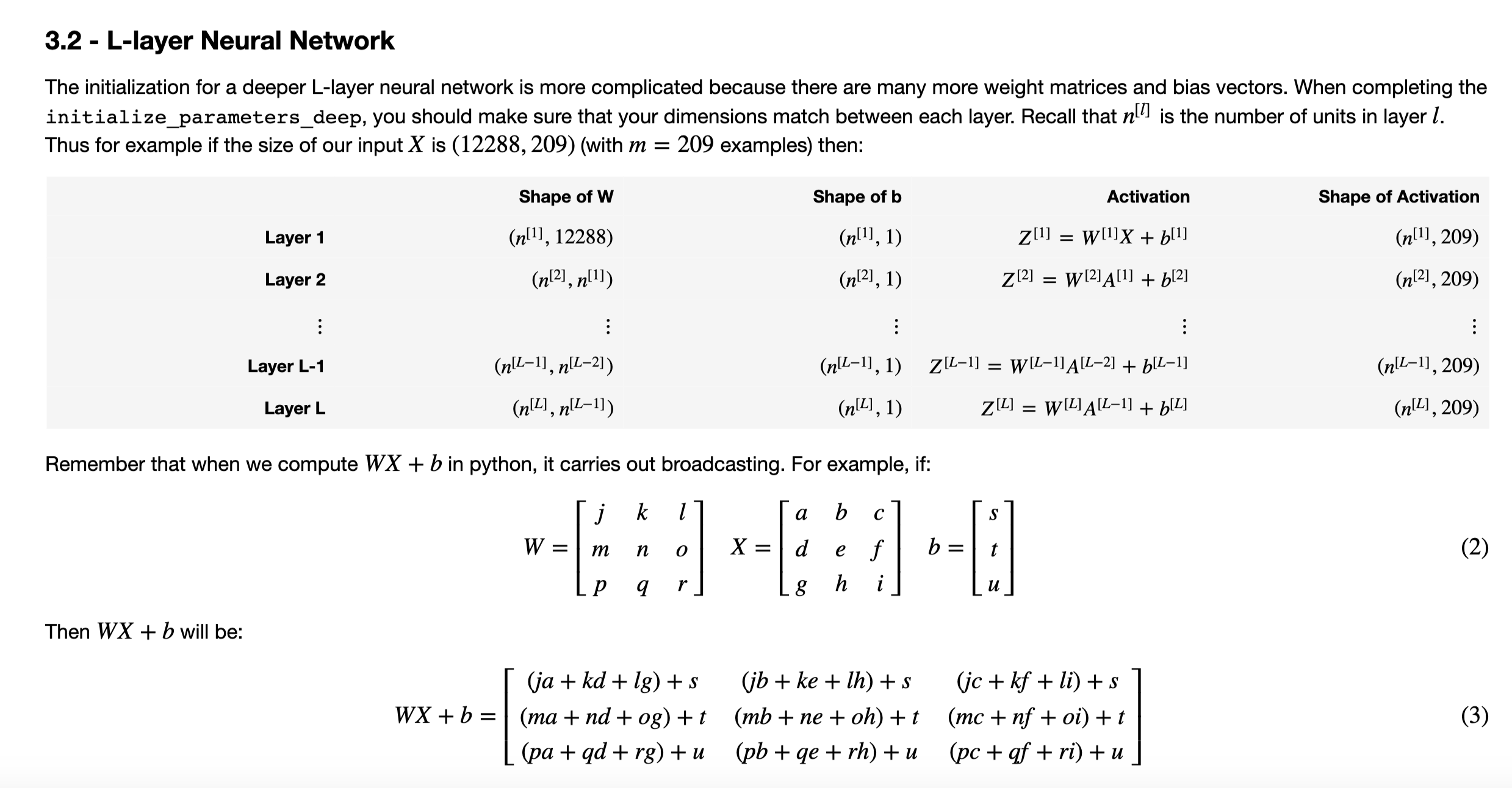
* Make sure your parameters' sizes are right. Refer to the neural network figure above if needed.
* You will initialize the weights matrices with random values.
  + Use: np.random.randn(a,b) \* 0.01 to randomly initialize a matrix of shape (a,b).
* You will initialize the bias vectors as zeros.
  + Use: np.zeros((a,b)) to initialize a matrix of shape (a,b) with zeros.

**4.3 - The Loop**

**Question**: Implement forward\_propagation().

**Instructions**:

* Look above at the mathematical representation of your classifier.
* You can use the function sigmoid(). It is built-in (imported) in the notebook.
* You can use the function np.tanh(). It is part of the numpy library.
* The steps you have to implement are:
  1. Retrieve each parameter from the dictionary "parameters" (which is the output of initialize\_parameters()) by using parameters[".."].
  2. Implement Forward Propagation. Compute Z[1],A[1],Z[2]Z[1],A[1],Z[2] and A[2]A[2] (the vector of all your predictions on all the examples in the training set).
* Values needed in the backpropagation are stored in "cache". The cache will be given as an input to the backpropagation function.



**Questions- Concept Related:**

1. Why is the loss function convex?
2. GPU and CPU
3. Why are images represented in RGB scale?
4. Why are the images so blurr?
5. Why normalize images by subtracting dataset's image mean, instead of the current image mean in deep learning?

<https://stats.stackexchange.com/questions/211436/why-normalize-images-by-subtracting-datasets-image-mean-instead-of-the-current>

Subtracting the dataset mean serves to "center" the data. Additionally, you ideally would like to divide by the sttdev of that feature or pixel as well if you want to normalize each feature value to a z-score.

The reason we do both of those things is because in the process of training our network, we're going to be multiplying (weights) and adding to (biases) these initial inputs in order to cause activations that we then backpropogate with the gradients to train the model.

We'd like in this process for each feature to have a similar range so that our gradients don't go out of control (and that we only need one global learning rate multiplier).

Another way you can think about it is deep learning networks traditionally share many parameters - if you didn't scale your inputs in a way that resulted in similarly-ranged feature values (ie: over the whole dataset by subtracting mean) sharing wouldn't happen very easily because to one part of the image weight w is a lot and to another it's too small.

You will see in some CNN models that per-image whitening is used, which is more along the lines of your thinking.

1. What is Partial Derivative?

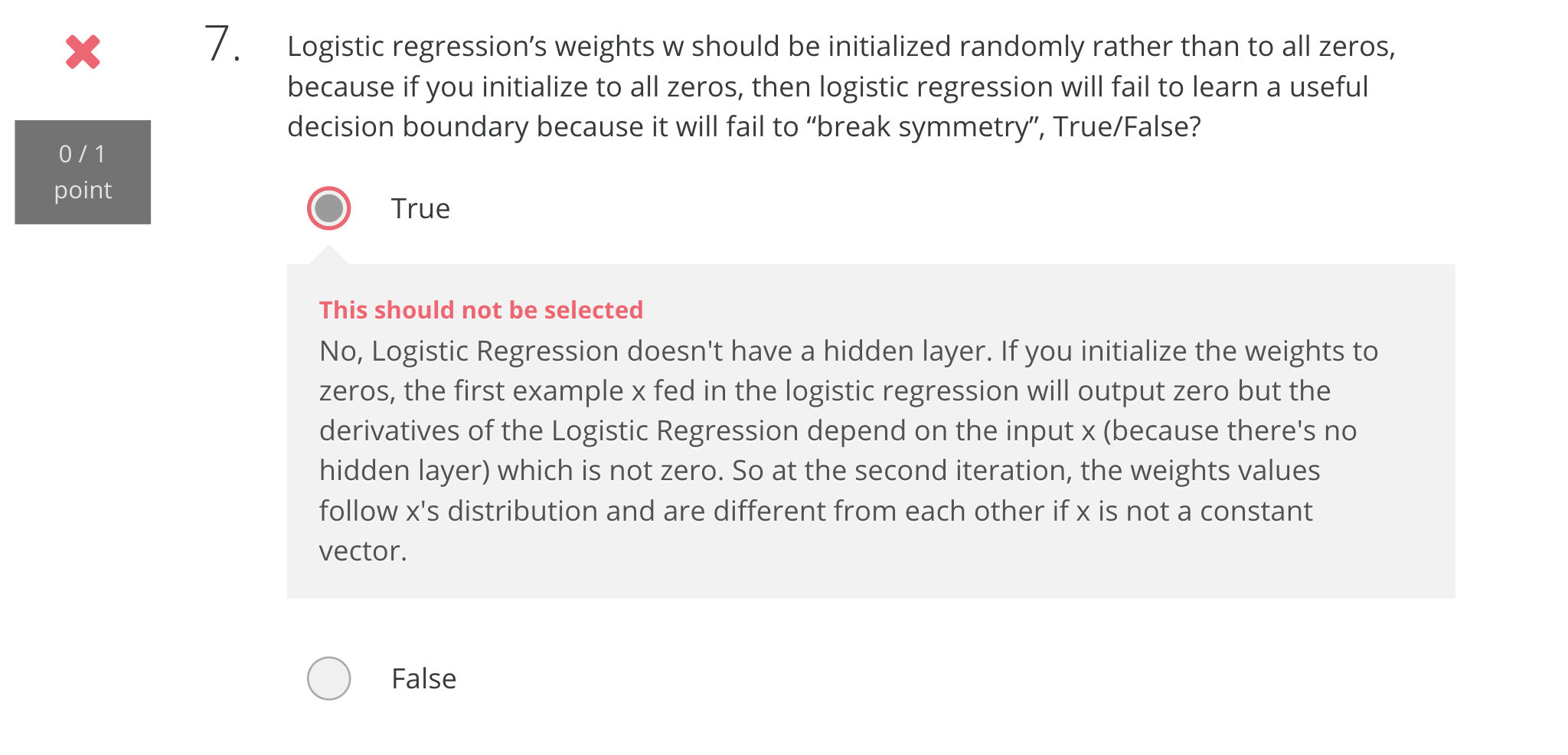
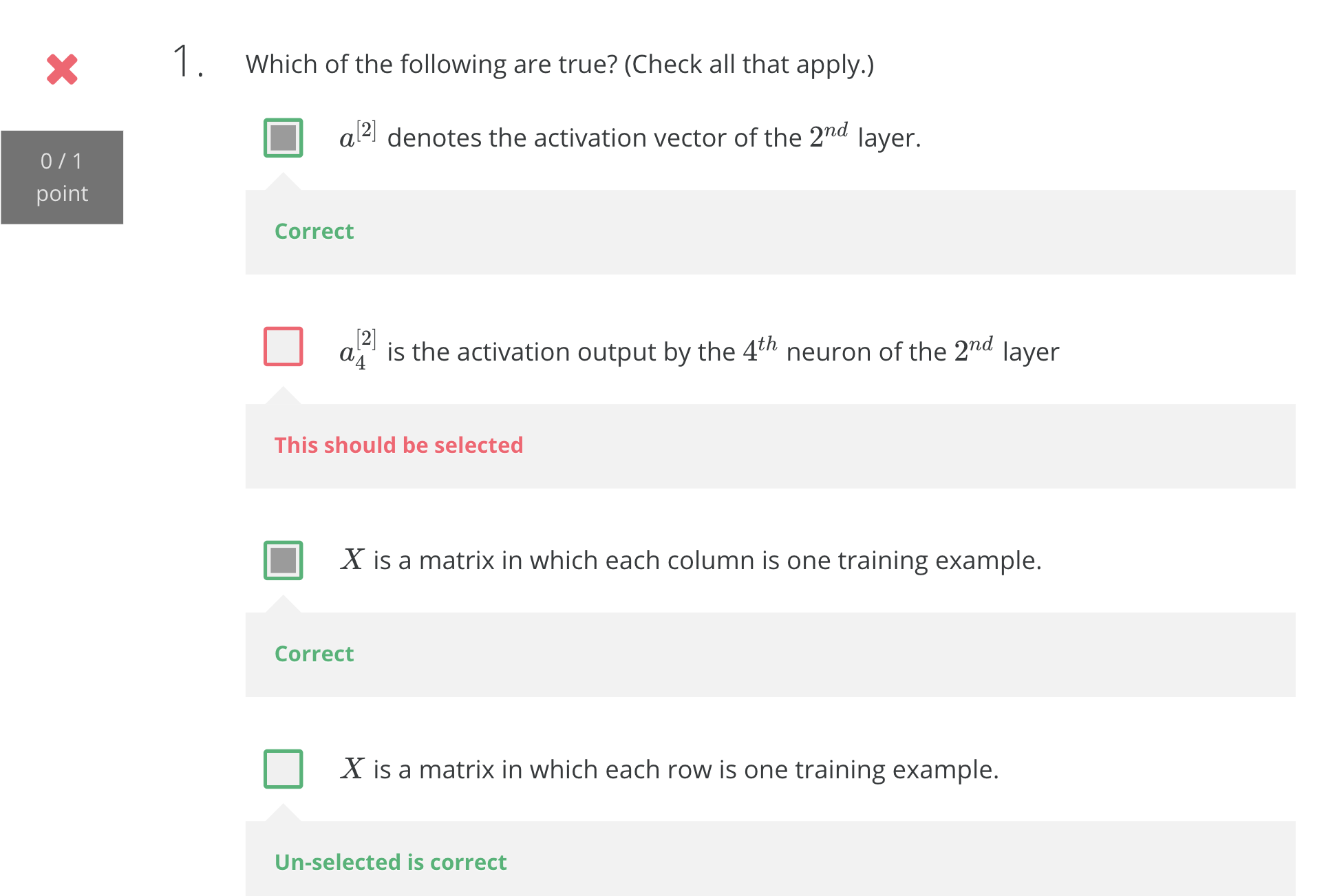
In [mathematics](https://en.wikipedia.org/wiki/Mathematics), a **partial derivative** of a [function of several variables](https://en.wikipedia.org/wiki/Function_(mathematics)#MULTIVARIATE_FUNCTION) is its [derivative](https://en.wikipedia.org/wiki/Derivative) with respect to one of those variables, with the others held constant

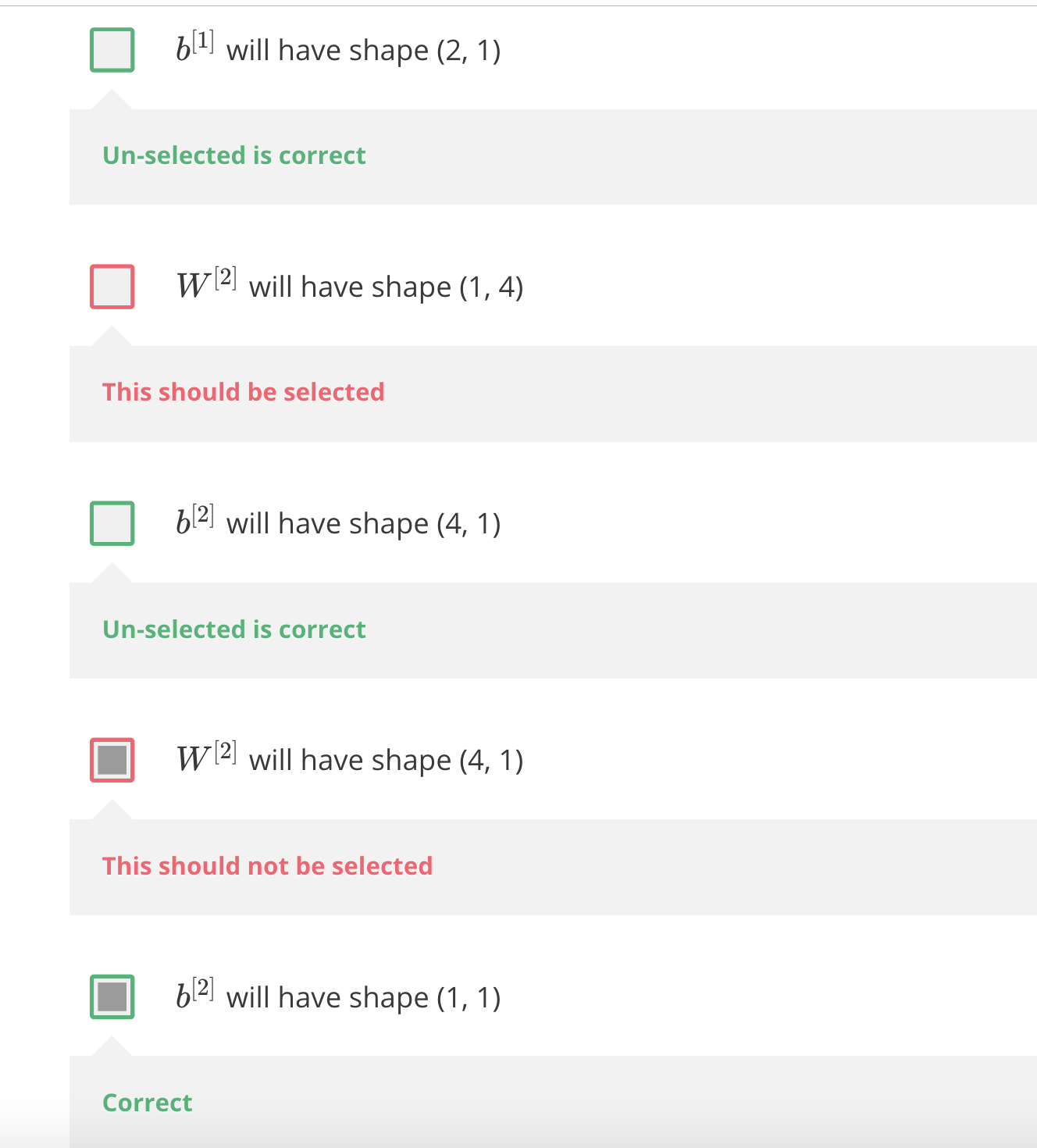
1. Why tanh activation function is better than sigmoid function and why ReLu is superior to all other functions?

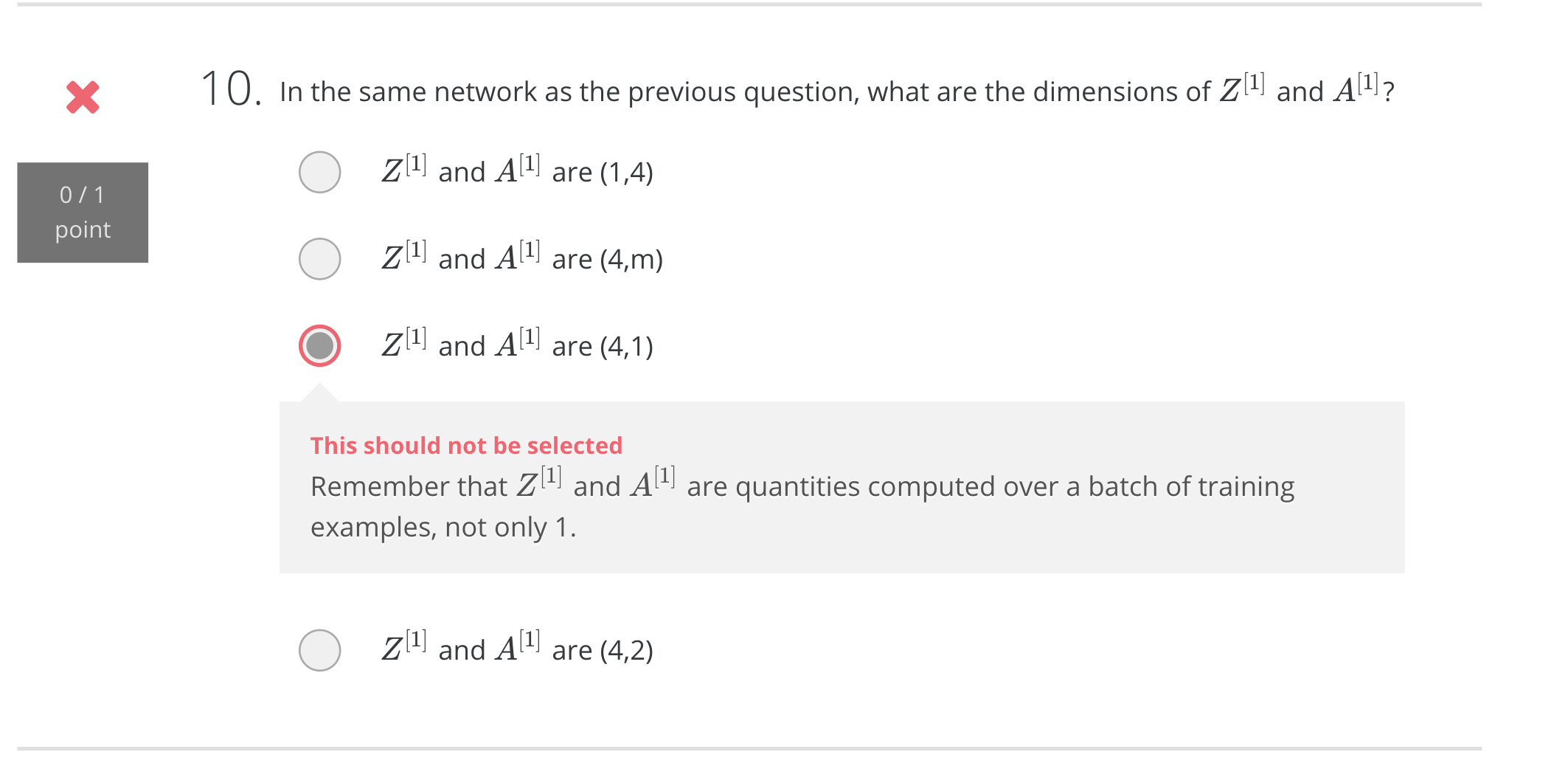
tanh is superior to sigmoid because it is centered to 0. tanh and sigmoid both have derivate close to 0 when value of activation function is either very large or very small. In ReLu when z>0 then derivative is 1 which helps in training neural network faster.

1. How to choose activation function?

If output is binary use sigmoid otherwise use ReLu (Rectified linear Unit)

1. Why do you need non-linear activation function?
2. 





1. What is planar data?

**Questions- Programming/Software Related:**

* + 1. What is Numpy?

**NumPy** (pronounced [/ˈnʌmpaɪ/](https://en.wikipedia.org/wiki/Help:IPA/English) ([*NUM-py*](https://en.wikipedia.org/wiki/Help:Pronunciation_respelling_key)) or sometimes [/ˈnʌmpi/](https://en.wikipedia.org/wiki/Help:IPA/English)[[2]](https://en.wikipedia.org/wiki/NumPy#cite_note-2)[[3]](https://en.wikipedia.org/wiki/NumPy#cite_note-3) ([*NUM-pee*](https://en.wikipedia.org/wiki/Help:Pronunciation_respelling_key))) is a library for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)), adding support for large, multi-dimensional [arrays](https://en.wikipedia.org/wiki/Array_data_structure) and [matrices](https://en.wikipedia.org/wiki/Matrix_(math)), along with a large collection of [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [mathematical](https://en.wikipedia.org/wiki/Mathematics) [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) to operate on these arrays.

* + 1. How to read images in numpy?
    2. What does numpy ndarray shape do?

For 1D array, return a *shape tuple* with only *1* element (i.e. (n,))  
For 2D array, return a *shape tuple* with only *2* elements (i.e. (n,m))  
For 3D array, return a *shape tuple* with only *3* elements (i.e. (n,m,k))  
For 4D array, return a *shape tuple* with only *4* elements (i.e. (n,m,k,j))

* + 1. What is an H5 file?

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data. H5 files are commonly used in aerospace, physics, engineering, finance, academic research, genomics, astronomy, electronics instruments, and medical fields.

* + 1. How to create hierarchal dataset from images?
    2. Why do we set seed in python and R?

The use of a seed function of a random module to initialize the pseudo-random number generator to **generate the deterministic random data** you want. Random number or data generated by **Python’s random module is not truly random, it is pseudo-random**(it is PRNG), i.e. deterministic. It produces the numbers from some value. This value is nothingbut a seed value. Numbers generated by the random module depend on the seed value.

* The **random module uses the seed value as a base to**[generate a random number](https://pynative.com/python-random-module/). If seed value is not present, it takes the system’s current time.
* If you don’t initialize the pseudorandom number generator using a random.seed (), internally seed function use current system current time value as the seed value. That’s why whenever we execute random.random() we always get a different value.
* The seed value is very significant in the field of computer security to pseudorandomly generate a strong secret encryption key.

**TO DO LIST:**

Explore h5y package

Explore other file type formats for images

Read more about loss

**Important links:**

Numpy: <https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html>

Numpy.zeroes: <https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html>

* X.shape is used to get the shape (dimension) of a matrix/vector X.
* X.reshape(...) is used to reshape X into some other dimension.

Broadcasting: <https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html>

Implementing a neural network from scratch in Python: <http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/>

Demystifying Deep CNN: <http://scs.ryerson.ca/~aharley/neural-networks/>

Loss function: <https://isaacchanghau.github.io/post/loss_functions/>