QUESTION: How do you construct a network with multiple layers? Is the order of function calls important when constructing the network?

Using the convenient structure of ***sequential*** model, we can pile up **hierarchicaly** the layers in the correct order, according to the desired *depth* and goals.

QUESTION: What happens when you call the module "Linear" with "bias = False"?

The module *linear* introduces a layer as a series of linear transformations ($w^T x + b$). Setting "bias=False" it means that we choose $b = 0$ or better say $w_0 = 0$ (not learning a bias).

QUESTION: How can we add non-linear activation function to the intermediate layers?

In the context of "*Sequential*" structure , we can always use a non linear activation function σ, by adding a line of (e.g.)

```
nn.ReLU()
```

if we desire a Rectified linear unit function, **right after each corresponding "linear" fully-connected layer** It will apply this nonlinearity to all units within the previous layer.

QUESTION: Why do we normalize the input images?

Normilizing the pixels of images according to the idea of a Gaussian curve centered at zero, allows the algorithm to converge faster. Usualy for RGB we devide by 256 so every pixel has a range of [0,1] .

QUESTION: How do you construct a network with multiple layers? Is the order of function calls important when constructing the network?

Using the convenient structure of ***sequential*** model, we can pile up **hierarchicaly** the layers in the correct order, according to the desired *depth* and goals.

QUESTION: What happens when you call the module "Linear" with "bias = False"?

The module *linear* introduces a layer as a series of linear transformations ($w^T x + b$). Setting "bias=False" it means that we choose $b = 0$ or better say $w_0 = 0$ (not learning a bias).

QUESTION: In dropout regularization, why do we scale the values by a factor of 1/p?

When we apply Dropout regularization during training in favour of the robustness of our model, we want to pay back for the imbalance we create to the models' weights towards an accurate evaluation (in testing phase we don't apply dropouts).

QUESTION: When training a neural network, we often have to construct the network from multiple components. Which are the components of the network in the case of the shallow model with a single layer? Which are the different functions for constructing these components, adding the dataset, and training the network?

Mathematicaly: For a single layer we're focusing on all the different neurons within that layer. In practice we're working with the actual weights that correspond to each specific feature. The main construction we care about is the W matrix,which for a single layer can be simplified by a single vector of size $d+1$ (if we consider d the feature dimensions) and contains all the representantive weight parameters, absorbing also the bias as a w_0 component. The layer in general can be a typical **dense connection layer,** or a shared-parameters layer with **sparse connections** (as the **convolution layer**).

Pytorch (not far away from Tensorflow/Keras idea):

- for fully connected layers we call

```
nn.Linear(in_features, out_features, bias=True)
```

- for a convolution layer

```
nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  bias=True, padding_mode='zeros')
```

where *kernel_size* is the size of the filter, *stride* is the # of pixels to be slided, and *padding* the number of pixels to surround our image-grid.

- for "pooling" (shrinking the size of feuture map)

```
nn.MaxPool2d(kernel_size, ...)
nn.AvgPool2d(kernel_size, ...)
```

(MaxPool2d--> center pixel corresponds to the maximum of all pixels in a pooling-frame whereas AvgPool2d-->center pixel corresponds to the average of all pixels in a pooling-frame)

- for output we use activation of **softmax** (for multiclass probabilities) of simply **sigmoid** function (for binary classification)

```
nn.Softmax
nn.Sigmoid
```

QUESTION: What is the purpose of the module "Flatten"?

```
nn.Flatten(start_dim=1, end_dim=- 1)
```

This layer repares the data to become the input of a fully-connected network (if eg. we have a pixel-volume of 3D "flatten" layer **shrinks that to a 1D tensor**).