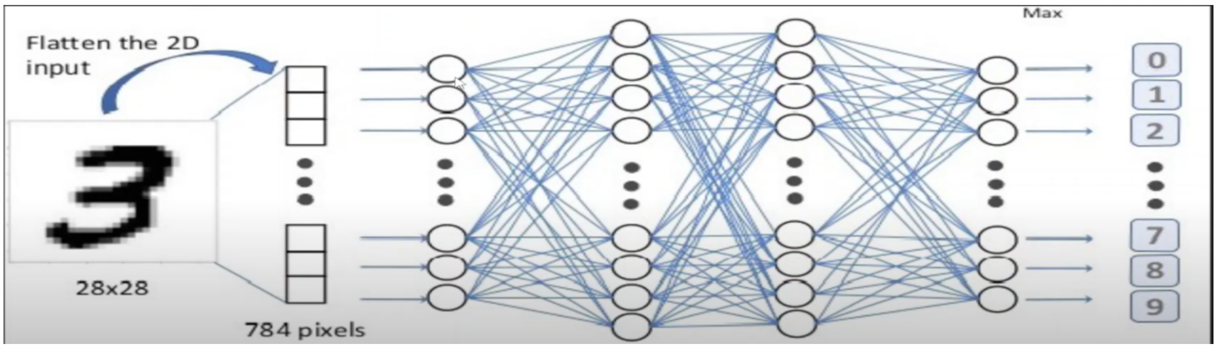
## Question 1 – Practical Part:



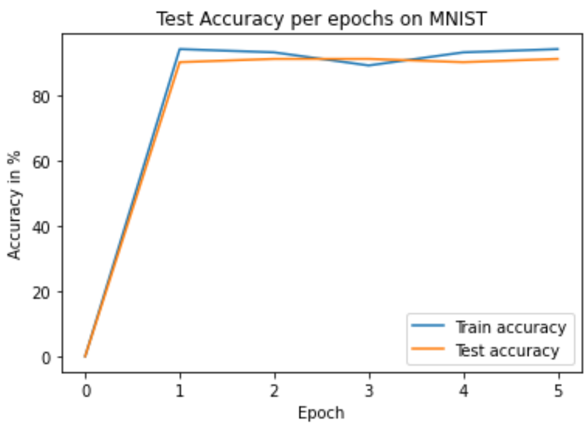
Let’s recall our hyperparameter for the model we are trying to implement:

```
# Define Hyper-parameters
input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

We have an input layer (vector of 28x28 pixels -> 784 pixels), one hidden layer of size of 32 and output size of a vector of 10 entries (one for each number). I used the tanh activation function .

*Training procedure:*

- For each epoch (and batch size): Train the model on 600 iterations with a learning rate of 0.001. Then update the parameters of the network. Calculate the accuracy on the batch size and store it in a list.
- To improve, the accuracy, several test were made by hand for the hyperparameters:
    - Number of iterations
    - Numbers of epochs
    - Learning rates
- I tested our network during the training procedure for each batch in an epoch. As I do for the train, I calculated the accuracy on the batch size and store it in a list.



```
Accuracy of the network on the 10000 test images: 92.45 %
```

## Question 2 – Practical Part:

Let's now build an in-built neural network using the torch functions

```
# hyperparameters
input_size = 784
output_size = 10
hidden_size = 200
batch_size = 128  -- like the want us to do in the exercise
learning_rate = 0.001
```

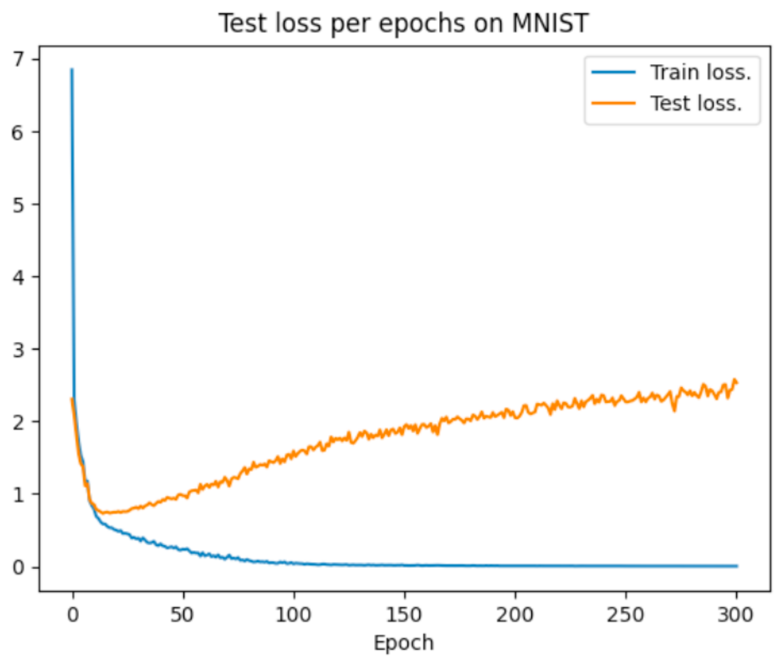
We used the ReLu activation function this time.

The goal to this exercise is demonstrate overfitting for neural network so the objective is to approach as much as possible the 0 value for the loss on only the training set (the label are given randomly for each image so the error on the test need to be increasing with the epoch).

The output will be the cross-entropy function loss:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

The model trained on this random label is saved as *modelRandom.pt* and we can test it on the file eval.py



This is the result we tried to aim for, the result was predictable as the epoch goes our model overfit his learning rules ( his weight ) but derive from the reality ( Bernoulli Distribution ).

```
The loss on the Train : 0.002442731987684965
The loss on the Test  : 2.532808542251587
```