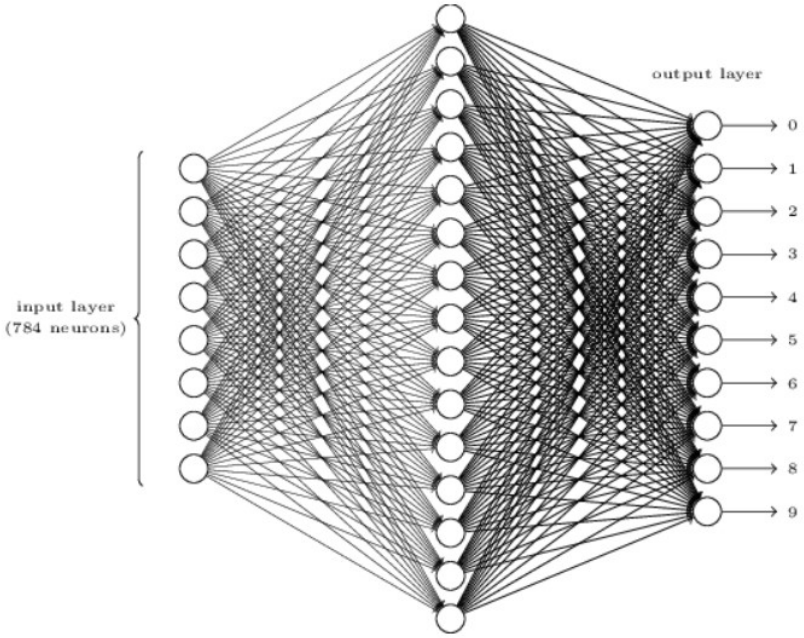
Sequence 5 exercises

Handwritten digit recognition with a neural network perceptron

In this exercise, we are going to deal with the problem of handwritten digit classification. The problem to solve is a 10-class problem for digits 0, 1, …, 9. In order to proceed to digit recognition, we are going to use a neural network with one hidden layer.



The available dataset is the MNIST database (http://yann.lecun.com/exdb/mnist/) with images of size 28x28 pixels. There are 60k images in the training set and 10k images in the testing set. See some samples in Fig.1

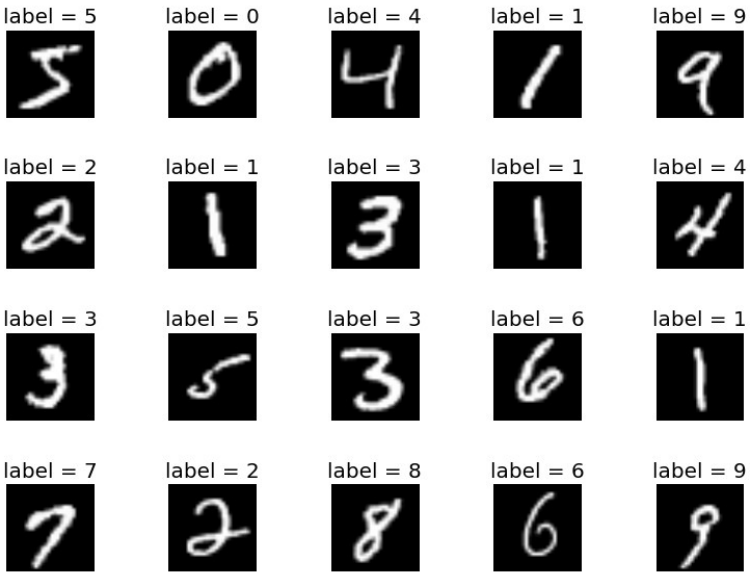


Fig.1 Labelled handwritten digits from the MNIST dataset

The input features are the 28x28=784 grey level pixels of the digit image and each image is associated to a label (supervised classification) belonging to the set: {0, 1, …, 9}. In order to fit with the output format of a neural network, the labels are re-defined with a vector of 10 components with zero values but the coefficient associated to the corresponding digit. For instance, an image with label digit 3 is associated to the label vector $\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$. That are 0 coefficient for bits 0, 1, 2, 4, 5, 6, 7, 8, 9 and 1 coefficient for bit 3.

1. Perceptron implementation from scratch (Homework)

1.1 Perceptron: how does that work?

The directory ***Perceptron from scratch*** contains all the files related to the implementation of the successive steps of a perceptron neural network. Look at ***main.py***, ***network.py*** and ***function.py*** and answer the following questions:

- What is the number of neurons of the input layer, the hidden layer and the output layer? Draw the network architecture with the corresponding inputs, weights, biases and outputs names on the figure.
- How are the weights and biases values initialized?
- Which activation function is used for the hidden layer and what is its role?
- How is the output computed (give the corresponding expression)? What is the role of the softmax function?
- The loss function used here is the cross entropy function whose role is to compute the similarity between the predicted probability vector and the right label vector. Look at ***https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451*** in order to have the definition of the cross entropy and explain how it works?

The considered perceptron is dependent of the following hyper-parameters:

- ✓ the learning rate of the gradient descent process
- ✓ the number of neurons in the hidden layer
- ✓ the number of iterations over the training set
- ✓ the batch size when using Stochastic Gradient Descent

What are the hyper parameter values in the provided code?

In the following, we are going to study the influence of the two first hyper parameters, the two last will be studied in section 2.

1.2 Influence of the learning rate value

Keep the default hyper-parameters and run the algorithm with the following values of the learning rate: 0.005 and 0.5. Gather your results in a Table and comment the results the results.

Learning rate	*Training Acc (%)*	*Testing Acc (%)*	*Training time in s*
0.005			
0.5			

1.3 Changing the number of neurons

Let's consider the classifier with the best learning rate value and the default other hyper parameters values. In this part, we are going to change the number of neurons of the hidden layer. Let's remember that there is no method to learn the optimal neuron number. Test the following numbers of neurons in the hidden layer 50, 20, 5. Gather the results in a Table and comment your results.

Neuron number	*Training Acc (%)*	*Testing Acc (%)*	*Training time (s)*
50			
20			
5			

2. Perceptron implementation using tensorflow.keras library (Classwork)

The previous perceptron implementation from scratch helps understanding each step of a neural network design and optimization but in practice, ones is using the keras library (https://keras.io/) in order to design a neural network. This library contains embedded functions that yield to a very quick implementation of any neural network once you are familiar with those functions.

The aim of this part is twofold:

- ✓ Implementing a perceptron of handwritten digit classification using the keras library
- ✓ Going on studying the influence of hyper parameters, in particular the influence of the batch size and the number of epochs.

2.1 Developing Perceptron with keras

By looking at https://victorzhou.com/blog/keras-neural-network-tutorial/#the-full-code , your job is to develop a python program that implement the same perceptron architecture as in section 1 with the best values as learning rate and number of hidden layer neurons (cf. section 1 results). Run it. What are the training and testing accuracies?

2.2 Batch size / number of epochs influence

In this subsection, the number of neurons in the hidden layer is tuned to 20, the used optimizer is the Adam optimizer with default values.

Define what the batch size and the number of epoch are. Run the algorithm with the following values of the batch sizes (200 and 2000) by keeping the number of epochs to 10. Gather your results in the Table and explain what it is going on.

(Batch size, epoch)	*Testing Acc (%)*
(200, 10)	
(20000, 10)	
(20000, 100)	

Considering batch size = 20000, increase the number of epoch to 100. What do you notice? Explain.

As a conclusion to this subpart, is it better to have a low batch_size with a low number of iterations or a high batch_size with a high number of iterations?

2.3 Monitoring the training process

As a matter of fact, it is of great importance to monitor the evolution of the loss and the accuracy functions during the training process. Usually both the training and validation loss and accuracy functions are monitored along the epochs. To do so, there are two main ways: using the *.history* function or using *TensorBoard*. You can choose the way that seems the most convenient to you.

2.3.1 Monitoring with the *.history()* function

To monitor the evolution of accuracy and/or loss functions, use the following piece of code available in the *code for curve monitoring.py* file:

```
# Train the model.
training_history = model.fit(

    blablabla

)
print('\n\n')



# plotting the metrics: accuracy curves and model loss curves
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(training_history.history['accuracy'])
plt.plot(training_history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(training_history.history['loss'])
plt.plot(training_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.tight_layout()
```

2.3.2 Monitoring with TensorBoard

Here, we are going to use TensorBoard. Adapt the program accordingly (look at the *piece of code.py* file showing how to adapt your current code to use TensorBoard). Then when running your code, information to be monitored are saved in the logs directory.

```
# Preparing for Tensorboard use
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

# Train the model.
training_history = model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=250,
  batch_size=300,
  validation_data = (test_images, to_categorical(test_labels)),
  callbacks = [tensorboard_callback]
)
```

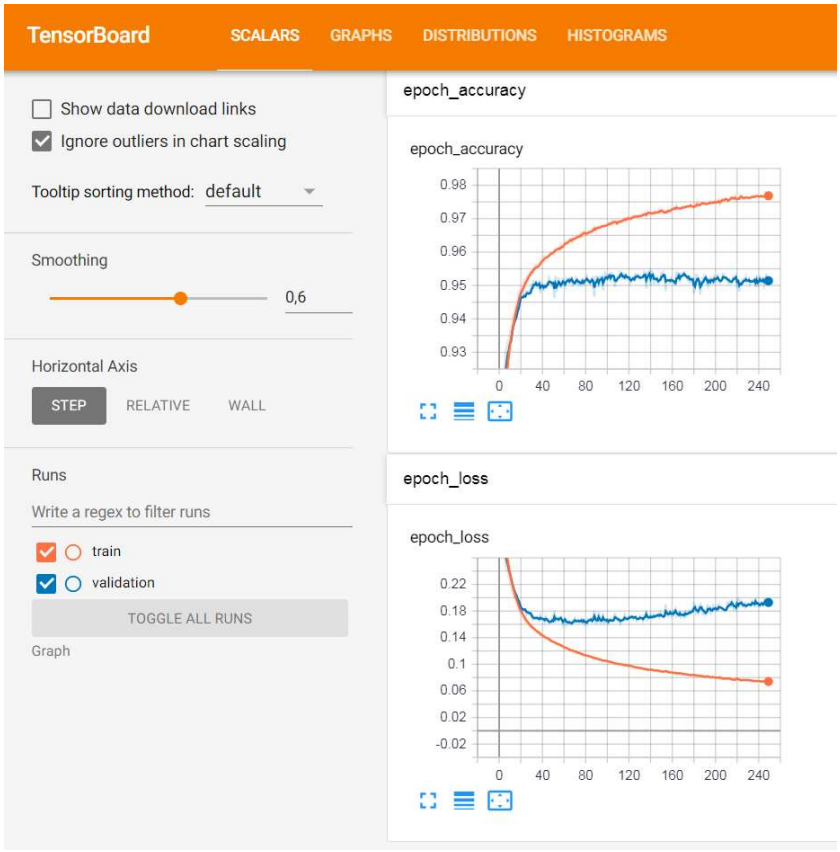
Run your code and then launch an anaconda prompt command window. Go to the directory where your python code is executed and type the following command:

tensorboard --logdir=logs --host localhost --port 8080

Rmk: the port number is 6060 when using Linux. If necessary, look at https://www.datacamp.com/community/tutorials/tensorboard-tutorial or https://www.tensorflow.org/tensorboard/get_started

After typing the tensorboard command, wait for the following message: Press CTRL+C to quit.

Then go to a browser and type: http://localhost:8080 to launch the tensorboard interface. This kind of interface in which accuracy and loss evolutions are drawn might appear:



Rmk :

- Do not quit the prompt command window until you have finished using tensorboard interface.

- When running the code again, first quit the anaconda prompt window. And restart tensorboard again after running completed.

2.3.3 Applying curve monitoring

Display the loss and accuracy curve evolutions in the 3 following cases:

- Batch_size = 500 and epoch = 20
- Batch_size = 30000 and epoch = 20
- Batch_size = 200 and epoch = 250

Comment the obtained curves evolution.

2.4 Using the learned model

The final goal of a trained model is to be used in order to predict new answers. Use your model to predict the digit of the five first images of the test set and compare the predicted labels with the ground truth (look at *model_name.predict()* function).

When a model has been trained, it is most of the time necessary to save its parameters in a h5-format file so that it will be possible to load the model and use it to predict answers on new data. Save your model (look at the *piece of code.py* file for help).

2.5 Recognizing your own handwritten digits

Use your smartphone in order to generate some new samples of handwritten digit images and use the learned classifier in order to predict the written digits. For sake of efficiency, it is better if you write your own digits on an entirely white paper.

To do so, create a new py file in which first you download the trained model, then download the new images to be tested in the right format (normalized images and images structured as a 3D matrix I[img_number, 28, 28] even if you have only one image to test).