MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 111

Fall 2021

Take-Home Exam 3

REGULATIONS

Due date: 23:59, 16 January 2022 *(Not subject to postpone)*

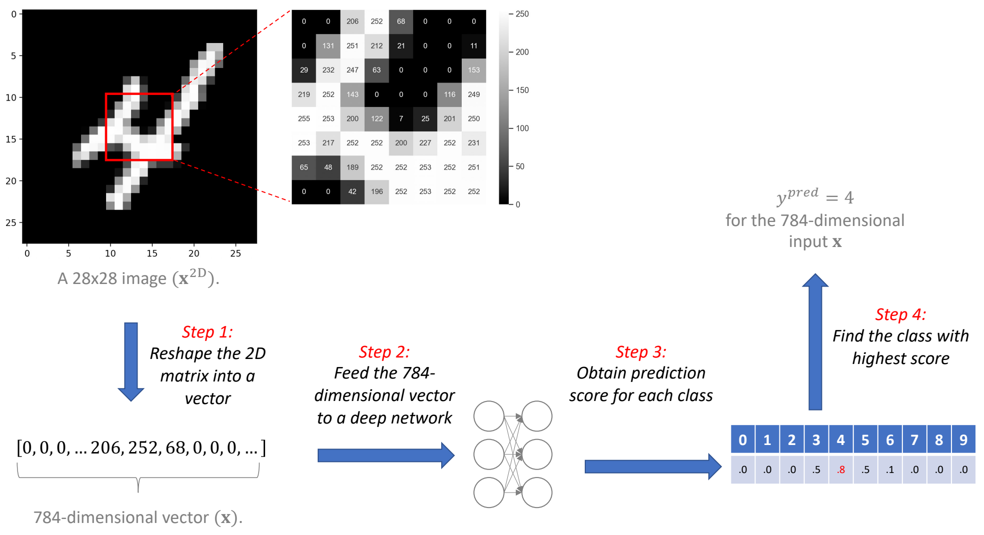
Submission: Electronically. You should save your program source code as a text file named `the3.py`. Check announcement on ODTUCLASS course page for the submission procedure.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

In this exam, you will have a chance to make inference (predictions) using a pre-trained deep network, namely a Multi-layer Perceptron (MLP). To be specific, we will look at the problem of image classification where we assign a label y (e.g. 'car', 'house', 'smiling', 'digit 0') to an image $\mathbf{x}$ provided as input. For example, the image can be the image of a digit (as shown in Figure 1), and the estimated label can be digit 4. To provide our input to an MLP, we will first transform the image into a 1D vector, as illustrated in Figure 1. For this, the image is scanned row by row and the concatenation of all the rows is taken as the vector representation of the image. Then, the network will provide us prediction score for each class. Finally, our estimated label, y^{pred}, will be the class with the highest score.



A 28x28 image ($\mathbf{x}^{2D}$).
Step 1: Reshape the 2D matrix into a vector
[0, 0, 0, ... 206, 252, 68, 0, 0, 0, ...]
784-dimensional vector ($\mathbf{x}$).
Step 2: Feed the 784-dimensional vector to a deep network
Step 3: Obtain prediction score for each class
Step 4: Find the class with highest score
$y^{pred} = 4$ for the 784-dimensional input $\mathbf{x}$
0 1 2 3 4 5 6 7 8 9
.0 .0 .0 .5 .8 .5 .1 .0 .0 .0

Figure 1: Image Classification with a Deep Network.

Formally, we denote a deep network as a parametric function $f(\,\cdot\,;\theta)$, where θ denotes the parameters (weights). We will provide you (i) the description of the network f, and its weights

θ, and (ii) a dataset of N samples, $\mathcal{D} = \{(\mathbf{x}_0, y_0), ..., (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ denotes an input and $y_i \in \mathcal{Y}$ is its label. We will provide you images as vectors.

Your task will be to write a function that performs forward-pass for a given $\mathbf{x}_i$ through the network and obtain the prediction scores, i.e. $f(\mathbf{x}_i; \theta)$.

Forward-pass in a Deep Network

A deep network is composed of a sequence of *layers*, l_1, .., l_L, and during forward pass, $\mathbf{x}_i$ is passed through those layers in sequence:

$$\mathbf{x}_i\ (\mathbf{a}_i^0) \xrightarrow{\text{layer } l_1} \text{activations } \mathbf{a}_i^1 \xrightarrow{\text{layer } l_2} \text{activations } \mathbf{a}_i^2 \ldots \xrightarrow{\text{layer } l_L} \text{prediction scores } \mathbf{s}_i, \tag{1}$$

where we use $\mathbf{a}_i^j$ to denote the output (activations) of applying layer l_j on the previous layer's values, $\mathbf{a}_i^{j-1}$. We will use M_j to denote the number of values (neurons) in layer l_j. In our case $\mathbf{a}_i^0$ is nothing else but $\mathbf{x}_i$, the image expressed as vector.

In a compact form, we can write forward pass as follows:

$$\mathbf{s}_i = l_L(l_{L-1}(\ldots l_2(l_1(\mathbf{x}_i)))). \tag{2}$$

Then, the prediction of the network, y_i^{pred}, is the class that has the highest score in $\mathbf{s}_i$ – as illustrated in Figure 1.

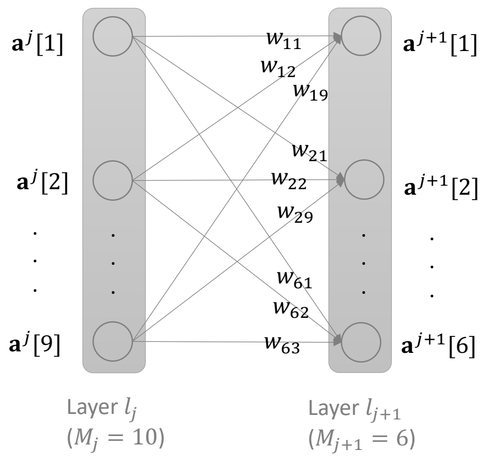


Figure 2: Illustration of a linear layer.

In this exam, we will only consider the following layers:

- **Linear layer**: A linear layer is one of the most essential elements in a deep network. A linear layer between $\mathbf{a}^j \in \mathbb{R}^{M_j}$ and $\mathbf{a}^{j+1} \in \mathbb{R}^{M_{j+1}}$ describes a linear mapping from $\mathbf{a}^j$ to $\mathbf{a}^{j+1}$ as follows:

$$\mathbf{a}^{j+1}[k] = \sum_{m=1}^{M_j} w_{km} \times \mathbf{a}^j[m], \tag{3}$$

  where we use w_{km} to denote the weight (parameter) connecting *neuron* $\mathbf{a}^j[m]$ to $\mathbf{a}^{j+1}[k]$. See also Figure 2 for an illustration.

  The set of w_{km} for layer l_{j+1} can be represented as a list of vectors, W_{j+1}, which has M_{j+1} vectors and each vector has M_j elements.

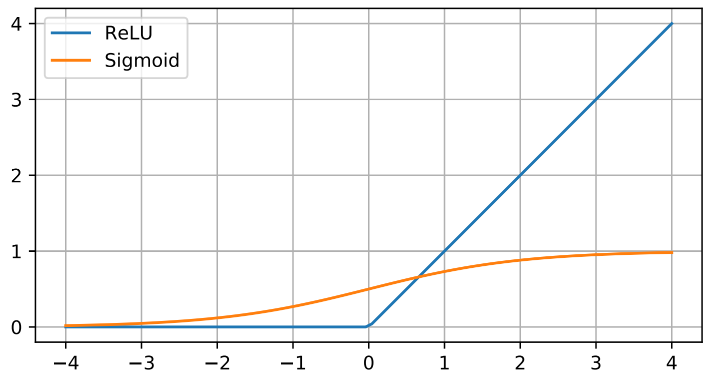


Figure 3: The ReLU and Sigmoid activation functions.

- **Rectified Linear Unit (ReLU) activation**: ReLU (proposed by Nair & Hinton in 2010) is a simple non-linear function: $\mathrm{relu}(x) = \max(0, x)$ for a scalar x – see Figure 3. The ReLU layer between $\mathbf{a}^j \in \mathbb{R}^{M_j}$ and $\mathbf{a}^{j+1} \in \mathbb{R}^{M_{j+1}}$ is defined then as follows:

$$\mathbf{a}^{j+1}[k] = \mathrm{relu}(\mathbf{a}^j[k]). \tag{4}$$

  In other words, a ReLU layer does not change dimensionality (i.e. $M_j = M_{j+1}$) and it does not have any parameters.

- **Sigmoid activation**: Sigmoid is a more smooth non-linear function than ReLU and defined as follows for a scalar x – see Figure 3:

$$\text{sigmoid-naive}(x) = \frac{1}{1 + e^{-x}}, \tag{5}$$

  which will lead to numerical issues in Python because of the exponent. Therefore, we will use the following clipped version:

$$\text{sigmoid-clipped}(x) = \begin{cases} 0, & \text{if } -700 \leq x \\ \frac{1}{1+e^{-x}}, & \text{if } -700 < x < 700 \\ 1, & \text{if } \quad 700 \leq x \end{cases} \tag{6}$$

  The Sigmoid layer between $\mathbf{a}^j \in \mathbb{R}^{M_j}$ and $\mathbf{a}^{j+1} \in \mathbb{R}^{M_{j+1}}$ is then defined as follows:

$$\mathbf{a}^{j+1}[k] = \text{sigmoid-clipped}(\mathbf{a}^j[k]). \tag{7}$$

  In other words, a Sigmoid layer does not change dimensionality (i.e. $M_j = M_{j+1}$) and it does not have any parameters.

PROBLEM & SPECIFICATIONS

You will be provided a network as follows:
`Network = [layer1, layer2, .., layerL]`,
where each `layer` is either of the following:

- `['linear_I', Weights]`: A linear layer with weights `Weights`, which is a list. `I` is an integer representing the order of this layer among all linear layers. If this layer is a mapping from M_j to M_{j+1}, `Weights` has M_{j+1} elements where each element has M_j numbers for $\mathbf{a}^j$.

- `'relu_I'`: The non-linear layer in Eqn. 4. `I` is an integer representing the order of this layer among all ReLU layers.
- `'sigmoid_I'`: The non-linear layer in Eqn. 7. `I` is an integer representing the order of this layer among all Sigmoid layers.

Your task is to write a function called `forward_pass(Network, X)`, where `Network` is as defined above and `X` is a 784-dimensional list representing an image. The output of the `forward_pass` function is the output of the last layer in the `Network`. We will arrange the `Network` such that its last layer will have $C = 10$ many numbers (i.e. prediction scores) if there are $C = 10$ classes. `forward_pass()` should not select the class with the highest score; that will be the job of other functions.

SPECIFICATIONS

- A zip file is provided for you with the following contents:
  - `mnist.pkl`: A file containing 1000 digit images and their labels.
  - `network_3layer.pkl`: A file for a pre-trained MLP with 3 linear layers.
  - `network_2layer.pkl`: A file for a pre-trained MLP with 2 linear layers.
  - `utils.py`: A Python module with a set of utilities that you will use.
- The dataset will be provided to you such that each image is a list with size 784. Though, you are recommended to make your code not depend on number 784.
- We will test your function with different MLPs, which will differ in their depth (the number of layers) and the types of layers (Linear, ReLU or Sigmoid). Do not make any assumptions in your code related to the sample networks provided.
- You can define as many functions as necessary.
- You cannot import any modules, except for the `math` library (for using the `exp` function).
- You can use iteration or recursion, though, it will be easier to use iteration. Using lists is sufficient as a container, though you can use dictionaries if you like.
- To consider two floating point numbers, `f1` and `f2`, to be equal, we will use `abs(f1-f2) < 1e-6`.

SAMPLE RUN

Loading the Dataset and the Network

You can load the provided dataset and the networks as follows:

```
>>> import utils
>>> dataset = utils.load_dataset("mnist.pkl")
>>> network1 = utils.load_network("network_3layer.pkl")
>>> network2 = utils.load_network("network_2layer.pkl")
```

The networks can be visualized as follows:

```
>>> utils.display_network(network1)
['linear_1: 784->100', 'relu_1', 'linear_2: 100->50', 'relu_2', 'linear_3: 50->10']
```

For the first network, we see that there are three linear layers with different sizes (`784->100` means that that linear layer is a mapping from a 784-dimensional vector to a 100-dimensional vector) and two ReLU layers in the network. These five layers are at indexes 0, 1, 2, 3 and 4. `network[0][1]` would be the weights of `'linear_1'`; `network[2][1]` the weights of `'linear_2'`, and `network[4][1]` the weights of `'linear_3'`.

The second network looks slightly different:

```
>>> utils.display_network(network2)
['linear_1: 784->20', 'sigmoid_1', 'linear_2: 20->10']
```

Image samples can be visualized or inspected as follows:

```
>>> X_sample = dataset['X_test'][100] # An image of digit 6
>>> utils.display_image(X_sample) # Visualize the image using ASCII chars
............................
............................
............@@..............
............@@..............
...........@@...............
...........@................
..........@@................
..........@.................
..........@.................
.........@@.................
.........@@.......@@@.......
.........@......@@@@@@......
.........@.....@@.....@.....
.........@....@@......@.....
.........@....@.......@.....
.........@....@.......@.....
.........@@...@.......@.....
.........@@...@.......@.....
..........@@..@@@....@......
..........@@@......@@.......
...........@@@@..@@@........
.............@@@@@..........
............................
............................
............................
............................
............................
............................
>>> print(dataset['y_test'][100]) # Let us see the correct label for this image
6
```

Testing the Network

We will use and test your solution as follows:

```
>>> from the3 import *
>>> forward_pass(network1, X_sample)
```

```
[-23.150019341068386, -544.3884165004889, 102.3173543266603, -621.083942074961,
-349.55474935029247, -283.5770896553512, 1522.4062872150532, -937.1164874720137,
-460.4310917263387, -970.2596681084673]
```

For the second network, the output should be:

```
>>> forward_pass(network2, X_sample)
[0.08618086576461792, -1.6271762549877167, -0.01971033774316311, -2.9853801876306534,
-2.0900632441043854, -2.0362057238817215, 5.234064280986786, -3.0017894506454468,
-1.9272374510765076, -3.553870514035225]
```

Our sample has label 6 and we see that the networks were able to provide the highest score for digit 6 (note that the classes are in the following order 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

If you like, you can also calculate the accuracy of networks as follows (as the function makes N many passes over the network if there are N images in the dataset, this calculation might take a while):

```
>>> utils.calculate_accuracy(dataset, network1, predictor=forward_pass)
95.8
>>> utils.calculate_accuracy(dataset, network2, predictor=forward_pass)
90.7
```

NOTES

- Be careful about indexing. The mathematical descriptions start with index 1.
- Don't make any assumptions about layer ordering or sizes.
- Be careful about the aliasing problem. Make sure that a layer does not change the values of the previous layer.
- Your program will be graded through an automated process. Do not print anything on screen. Comply with the specifications.
- Your program will be tested with multiple data (a distinct run for each data). Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall THE3 grade in the range of [0,30].
- A program based on randomness will be graded zero.
- The glass-box test grade is not open to discussion nor explanation.
- Your code will be evaluated using Python 3.8.10 in the Linux environment on inek machines. Make sure that you test your code on inek machines before submission.