UiT
THE ARCTIC UNIVERSITY OF NORWAY

Mandatory Assignment

Mandatory Assignment in	**FYS-3033 - Deep Learning**
Hand-out:	**Monday February 13, 2023, 09:00**
Hand-in:	**Friday March 3, 2023, 10:00**

The Mandatory Assignment contains 5 pages including this cover page

Contact person:	**Michael Kampffmeyer**
Email:	**michael.c.kampffmeyer@uit.no**

Before You Start

Portfolio instructions

Your code should be submitted together with your report (see instructions below). **Further, please include a discussion of the results obtained and a discussion of the implementation in your report, which show that you understand what you are doing.** Please include your implementation of the missing functions also in the .pdf report.

The code should be commented in such a way that any person with programming knowledge should be able to understand how the program works. Like your report, the code must be your own individual work.

You are permitted to use standard built-in functions and/or packages (e.g.numpy in Python). However: make sure that the packages you use do not over-simplify your implementation.

Hand-in format

Please submit your solution to Canvas as *one* single `.zip` file that contains two folders, one called `doc` that contains your report, and another one called `src` containing the code. The file name of the `.zip` file should follow the format `mandatory_assignment_`*`candidate_name`*`.zip` (replace *`candidate_name`* with your name).

Please include the course name on the frontpage of your report.

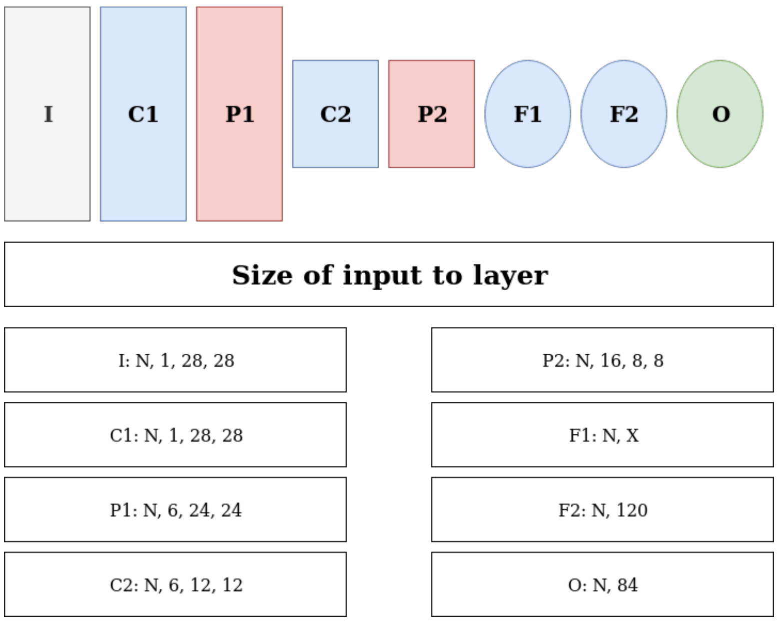
Problem 1

In this problem you will perform the task of hand-written digits recognition, i.e. a classification problem with 10 classes (C=10). You will train your model on the MNIST dataset, which consists of 28x28 grey-scale images of hand-written digits. In order to obtain the dataset and the pre-code download the `mandatory_assignment.zip` file. Once you have unzipped the archive enter the `data` folder and run `./get_mnist.sh` to obtain the dataset. Alternatively, you can download the files through the following links and place them in the `data` folder.

- Training dataset, consisting of 60 000 images. However, such a large number of images will be computationally demanding. Therefore, you will train on a subset consisting of 2000 images.

- Test dataset, consisting of 10 000 images. However, such a large number of images will be computationally demanding. Therefore, you will train on a subset consisting of 500 images.

You will perform this task using a model inspired by on of the earliest Convolutional Neural Network (CNN) architectures, namely the LeNet-5, which is illustrated in Figure 1. The LeNet-5 consist of two convolutional layers, two max-pooling layers and two fully connected layer, in addition to the input and output layer. The boxes at the bottom of Figure 1 indicate the size of the input to each layer, where N denotes the number of samples.



Size of input to layer

I: N, 1, 28, 28	P2: N, 16, 8, 8
C1: N, 1, 28, 28	F1: N, X
P1: N, 6, 24, 24	F2: N, 120
C2: N, 6, 12, 12	O: N, 84

Figure 1: *Illustration of the LeNet-5 architecture. Layers labeled with a 'C' refers to a convolutional layer, which performs a convolution operation followed by an activation function. Layers labeled with a 'P' refers to a pooling layer, which performs a max pooling operation. Layers labeled with a 'F' refers to a fully connected layer, which performs matrix multiplication followed by an activation function. The 'I' and 'O' layer indicate the input and the output layer, respectively. Below the architecture itself is a description of the size of the input to each layer.*

(1a) Assuming no padding in any of the layers and a stride of 1 for convolutional layers and 2 for pooling layers, what are the dimensions of the filters in the convolutional layers in the above architecture? Further, what is the input size to the first fully connected layer (F1) (marked with X)? Add your answer for the input size to the file `cnn.py` where "TODO: INSERT NUMBER HERE" is indicated.

(1b) For the above architecture, compute the number of weight (incl. bias) parameters.

(1c) Describe the benefits of convolutions when processing grid-structured data such as images and videos.

In the `mandatory_assignment.zip` file you will find a `problem1.py` file in order to run the pre-code for the first problem. In order to get the code to run you will have to modify a set of functions in the `layers.py` file in the `code` folder.

(1d) Fill in the forward and backward pass of the ReLU layer (`ReluLayer`). In order to allow you to check the correctness of your implementation, we have implemented a gradient-checker for you. Once you have implemented the forward and backward pass run the `problem1.py` file and inspect the relative error for your ReLU layer. If it is lower than 1e-6 your implementation is probably correct.

(1e) Fill in the forward and backward pass of the fully connected layer (`FullyConnectedLayer`). Again, you can run `problem1.py` to validate the correctness of your loss derivatives with respect to the bias and weight parameter.

(1f) Fill in the forward and backward pass of the convolutional layer (`ConvolutionalLayer`). Follow the same procedure to validate your implementation.

(1g) Fill in the forward and backward pass of the max-pooling layer (`MaxPoolingLayer`). Follow the same procedure to validate your implementation.

(1h) Train your network by running the `problem1.py` script contained in the mandatory assignment folder and report your results.

Problem 2

In this problem you will utilize a Long short-term memory (LSTM) network to perform the task of text generation. You will train the model on a dataset that consists of Shakespear quotes.

(2a) Describe the vanishing gradient problem for RNNs and how gated Recurrent Neural Networks such as LSTMs and GRU address this issue.

(2b) Describe the attention mechanism and how it can be used to address the sequence-to-sequence task (Neural Machine Translation).

In the `mandatory_assignment.zip` file you will find a `problem2.py` file in order to run the pre-code for the second problem. In order to get the code to run you will again have to modify a set of functions in the `layers.py` file in the `code` folder.

(2c) Fill in the forward and backward pass for a single time step of the lstm layer (`LSTMLayer`). A numerically stable version of the `sigmoid` activation function can be found in the `layers.py` file and you can use numpy's `tanh` function.

(2d) Train and run on text data and include a sample of the generated text in your report. You can choose between generating Shakespear quotes or Donald Trump tweets by changing the `dataset` variable in `problem2.py` from `shake` to `trump`.