# Convolution and Recurrent Networks

# Assignments: Neural Networks
# [Marks = 15]

## Instructions to follow: Please perform your assignments in Google colab

### Main codes for Grading

1. Download the "nn.py" file and copy the content into your new python Notebook.
2. Next, please name your Notebook as "nn". Don't alter the names, it has to be "nn"
3. Follow the scripts /instruction and complete the functions with your code. You have to replace ## YOUR CODE HERE ## with your actual codes.
4. Once all the codes are written, next download this notebook as .ipynb file and .html file and .py file ( .ipynb file, .html file for grading and .py file for testing your codes). ***These are your solution files.***
   Note: The evaluation is solely dependent on this notebook for all correct code implementations. No marks are allocated for testing. The testing process is included for educational purposes to illustrate how codes are validated in industries to ensure their correctness

### Implementation of convolution and recurrent networks

5. Open the provided "test_nn.ipyb" file in Google colab environment.
   a. Upload the "nn.py" file (your solution file downloaded in 4) into this Notebook.
      File → Upload Notebook → Upload → Browse and select your file
   b. Create a new folder inside the notebook as it acts as a local directory. Name it as "data", don't alter the names.
      Left side of your Colab Notebook A folder like icon is visible which is a file menu. You must create a new folder there.
      File → New Folder → Name it as 'data'
   c. Upload all the provided datas inside "data" folder created in 5.b
      i. mnist.hdf5
      ii. Youtube01-Psy
      iii. Youtube02-KatyPerry
      iv. Youtube03-LMFAO
      v. Youtube04-Eminem
      vi. Youtube05-Shakira

vii. youtube-comments.hdf5

d. Now just run the codes.
Don't change any codes

e. Download this Notebook as .py file (i.e., as "test_nn.py"). ***This will be further used in testing setup.***

## Running Tests on the codes using the pytest command

6. Open the provided Notebook " convolution_and_recurrent_networks.ipynb" in Google colab.
7. Also upload "nn.py" file (your solution file downloaded in 4) and "test_nn.py" (downloaded in 5.e) into this Notebook.
8. Create a new folder inside the notebook as it acts as a local directory. Name it as "data", don't alter the names.
   Left side of your Colab Notebook A folder like icon is visible which is a file menu. You must create a new folder there.
   File → New Folder → Name it as 'data'
9. Upload all the provided datas inside "data" folder created above in 5.b
   i. mnist.hdf5
   ii. Youtube01-Psy
   iii. Youtube02-KatyPerry
   iv. Youtube03-LMFAO
   v. Youtube04-Eminem
   vi. Youtube05-Shakira
   vii. youtube-comments.hdf5
10. Mount the Google Drive to access files stored in Google Drive within the Colab environment. (code already provided)
11. Now, run all the codes as given in the notebook.
    Please don't change any codes.
12. Run these tests using the !pytest command. Check how many of your codes are written correctly.
13. Download this file as an .ipynb file after all the testing's and submit.

## Files to be submitted for evaluation

- Your Solution file "nn.ipynb" and "nn.html"for grading (downloaded in 4)
- Your Solution file downloaded as "nn.py" (downloaded in 4)
- Testing file "convolution_and_recurrent_networks.ipynb" for pytest results (downloaded in 13)

## Gradings:

Note: Grading is solely based on your codes submitted in the "nn.ipynb" file. No marks are assigned for testing your codes.

```
1. def create_toy_rnn(input_shape: tuple, n_outputs: int) \
        -> Tuple[tensorflow.keras.models.Model, Dict]:
    """Creates a recurrent neural network for a toy problem.

    The network will take as input a sequence of number pairs, (x_{t},
y_{t}),
```

```
    where t is the time step. It must learn to produce x_{t-3} - y{t}
as the
    output of time step t.

    This method does not call Model.fit, but the dictionary it returns
alongside
    the model will be passed as extra arguments whenever Model.fit is
called.
    This can be used to, for example, set the batch size or use early
stopping.

    :param input_shape: The shape of the inputs to the model.
    :param n_outputs: The number of outputs from the model.
    :return: A tuple of (neural network, Model.fit keyword arguments)
    """
```

**[Marks = 3]**

```
2. def create_mnist_cnn(input_shape: tuple, n_outputs: int) \
        -> Tuple[tensorflow.keras.models.Model, Dict]:
    """Creates a convolutional neural network for digit classification.

    The network will take as input a 28x28 grayscale image, and produce
as
    output one of the digits 0 through 9. The network will be trained
and tested
    on a fraction of the MNIST data: http://yann.lecun.com/exdb/mnist/

    This method does not call Model.fit, but the dictionary it returns
alongside
    the model will be passed as extra arguments whenever Model.fit is
called.
    This can be used to, for example, set the batch size or use early
stopping.

    :param input_shape: The shape of the inputs to the model.
    :param n_outputs: The number of outputs from the model.
    :return: A tuple of (neural network, Model.fit keyword arguments)
    """
```

**[Marks = 4]**

```
3. def create_youtube_comment_rnn(vocabulary: List[str], n_outputs:
   int) \
        -> Tuple[tensorflow.keras.models.Model, Dict]:
    """Creates a recurrent neural network for spam classification.

    This network will take as input a YouTube comment, and produce as
output
```

```
    either 1, for spam, or 0, for ham (non-spam). The network will be
trained
    and tested on data from:
    https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection

    Each comment is represented as a series of tokens, with each token
    represented by a number, which is its index in the vocabulary. Note
that
    comments may be of variable length, so in the input matrix,
comments with
    fewer tokens than the matrix width will be right-padded with zeros.

    This method does not call Model.fit, but the dictionary it returns
alongside
    the model will be passed as extra arguments whenever Model.fit is
called.
    This can be used to, for example, set the batch size or use early
stopping.

    :param vocabulary: The vocabulary defining token indexes.
    :param n_outputs: The number of outputs from the model.
    :return: A tuple of (neural network, Model.fit keyword arguments)
    """
```

**[Marks = 4]**

```
4. def create_youtube_comment_cnn(vocabulary: List[str], n_outputs:
   int) \
        -> Tuple[tensorflow.keras.models.Model, Dict]:
    """Creates a convolutional neural network for spam classification.

    This network will take as input a YouTube comment, and produce as
output
    either 1, for spam, or 0, for ham (non-spam). The network will be
trained
    and tested on data from:
    https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection

    Each comment is represented as a series of tokens, with each token
    represented by a number, which is its index in the vocabulary. Note
that
    comments may be of variable length, so in the input matrix,
comments with
    fewer tokens than the matrix width will be right-padded with zeros.

    This method does not call Model.fit, but the dictionary it returns
alongside
    the model will be passed as extra arguments whenever Model.fit is
called.
```

```
    This can be used to, for example, set the batch size or use early
stopping.

    :param vocabulary: The vocabulary defining token indexes.
    :param n_outputs: The number of outputs from the model.
    :return: A tuple of (neural network, Model.fit keyword arguments)
    """
```

**[Marks = 4]**