In machine learning, artificial neural network is a supervised learning method providing a robust approach to approximating real-valued, discrete-valued and vector-valued target functions. It is one of the most effective methods for certain types of problems such as learning to interpret complex realworld sensor data. Inspired by biological neural system in the human brain, basically neural network contains a densely interconnected set of simple units, where each unit takes a number of real-valued inputs and produces a single real-valued output (which may become the input to many other units).

In this lab session, we will focus on two types of neural network, namely the classical *Artificial Neural Network (ANN)* and the *Convolutional Neural Network (CNN)*, with toy application in digit recognition.

As implementing a efficient CNN model often requires a lot of efforts, in real projects, it is common and highly recommended to make use of existing libraries. There are a number of libraries for deep learning, released recently by big tech companies and laboratories, such as:

- **Tensorflow** from Google
- **Torch and PyTorch** from Facebook
- **CNTK** from Microsoft
- **Caffe** from UC Berkeley
- **Theano** from University of Montreal

All these libraries are written by highly-skilled programmers with excellent resources so they all have good performance. Nevertheless, Tensorflow seems to be the most popular one at the moment. As a result, we will use Tensorflow for the exercises today.

First, make sure you have finished installing all the necessary libraries as specified in the setup guide. It is not mandatory to follow all the step in the guide, as long as you have the required libraries ready.

If you are using Tensorflow with Python 3 (this will be the case if you are using Windows), you need to start the notebooks by first running:

ipython3 notebook

We will use Keras interface [1] of Tensorflow, as it is simple, easy to read

[1]https://www.tensorflow.org/api_docs/python/tf/keras

and to use.

1 Artificial Neural Network with Tensorflow

In this exercise, you will train a simple ANN to classify handwritten digits from the MNIST dataset. The code for this exercise is in the file *ANN.ipynb*.

Step 1: Load dataset Run the code to load the dataset, split train/test and convert the labels into one-hot representation.

Step 2: Build an ANN model using Keras interface In this step, you need to call the functions from the Keras interface to build an ANN model to classify the digit images.

- First, start with a base `Sequential` model [2], which is a linear stack of layers.
- Second, add the first fully connected layer to the model. You need to select and create a layer from Keras using `K.layers` with the suitable arguments. To add the layer to the sequential model, use `model.add` function. Note that in Keras, fully connected layers are called *Dense* layers [3].
- Third, add a activation layer with *relu* fucntion [4]
- Continue with the next layers as specified in the code comments
- Compile the model with *sgd* optimizer and *categorical_crossentropy* loss function.

Step 3: Train the model Train the model using `model.fit` function, with `x_train` and `y_train_onehot` as inputs, for 10 training epochs with batch size of 32 samples.

[2]https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
[3]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
[4]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation

Step 4: Evaluate the model Fill the missing code to perform prediction and evaluation.

Step 5: Visualize the classification result Run the code to visualize the classification results.

2 Convolutional Neural Network with Tensorflow

In this exercise, we will build another digit classifier using Convolutional Neural Network model. The code for this exercise is in the file *CNN.ipynb*.

Step 1: Load dataset In this step, you need to first load the data, including features and labels. To use CNN models with 2D convolutional filters, you need to reshape the features, i.e. `x`, into the shape $\mathtt{height} \times \mathtt{width} \times \mathtt{number\_of\_channel}$ to be compatible with Keras interface.

Step 2: Build a CNN model using Keras interface In this step, you need to call the functions from the Keras interface to build a CNN model to classify the digit images.

- First, start with a base `Sequential` model as in the previous exercise
- Second, add the first 2D convolutional layer [5] to the model, with: 32 filters, kernel size 3×3, stride 1, padding scheme *'same'*
- Third, add a activation layer with *relu* function
- Continue with the second convolutional layer and activation layer as specified in the code comments
- Add a Dropout layer [6], with rate 0.75, after the second activation layer
- Add a 2D max pooling layer [7], with pool size 2×2 and padding scheme *'same'*, after the Dropout layer

[5] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
[6] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout
[7] https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

- Add a Flatten layer [8] after the max pooling layer. This layer reshape the 2D outputs from the pooling layer into 1D vectors.
- Continue with the fully connected layers to make predictions from the outputs of the Flatten layer, as specified in the code comments
- Compile the model with *sgd* optimizer and *categorical_crossentropy* loss function.

Step 3: Train the model Train the model using `model.fit` function, with `x_train` and `y_train_onehot` as inputs, for 10 training epochs with batch size of 32 samples.

Step 4: Evaluate the model Fill the missing code to perform prediction and evaluation.

Step 5: Visualize the classification result Run the code to visualize the classification results.

Step 6: Parameter tuning Step 1 to 5 have built a basic CNN model for digit classification. Experiments with different configurations to improve the accuracy you have obtained. Some suggestions:

- Increase the number of layers to have deeper model
- Add stronger regularizer, e.g. stronger Dropout rates, more Dropout layers, weight decay, etc. to mitigate overfitting
- Train the model for more training epochs
- Employ more advanced optimization techniques
- Try other activation functions

[8] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten