In the previous exercises, you have seen how neural networks can be built by stacking basic building blocks (given that we have implemented forward and backward flows for each block) sequentially. While it is essential to understand how each building block works and how their gradients are computed to update the parameters, implementing those requires a fair amount of time and efforts. Fortunately, in practice, most of the low-level functions have been done for us by libraries. Some of the most commonly used deep learning libraries are:

- **Tensorflow** from Google
- **PyTorch** from Facebook
- **CNTK** from Microsoft
- **Mxnet** from Amazon
- **Caffe and Caffe2** from UC Berkeley/Facebook
- **Theano** from University of Montreal (discontinued)

In this exercise, we will use Tensorflow to build neural networks to classify hand-written digits. By default, Tensorflow uses static computational graphs to construct and run the networks. As this is not straightforward for beginners to understand and debug, we will use a high-level interface named Keras instead. We, however, encourage you to read about computational graph in Tensorflow/Pytorch, as understanding it will give you more customization on your models.

You need to install Tensorflow before running the notebooks.

1 Artificial Neural Network with Tensorflow

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, 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 a layer to the sequential model, use `model.add` function. Note that in Keras, fully connected layers are called *Dense* layers.
- Third, add an activation layer with *relu* function. You can specify this activation as a parameter of the above Dense function, or define it seperately
- 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 the batch size of 32 samples.

Step 4: Evaluate the model Fill in 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, we need to reshape the features, i.e. `x`, into the shape $\texttt{height} \times \texttt{width} \times \texttt{number\_of\_channel}$ to be compatible with Keras interface.

Step 2: Build a CNN model using Keras interface

- First, start with a base `Sequential` model as in the previous exercise
- Second, add the first 2D convolutional layer to the model, with: 32 filters, kernel size 3×3, stride 1, padding scheme *'same'*
- Third, add a activation layer with *relu* function (again, seperately or as a parameter of the above 2D convolutional layer).
- Continue with the second convolutional layer and activation layer as specified in the code comments
- Add a Dropout layer with rate 0.75, after the second activation layer
- Add a 2D max pooling layer with the pool size of 2×2 and padding scheme *'same'*, after the Dropout layer
- Add a Flatten layer after the max pooling layer. For each sample, this layer reshapes the 2D output from the pooling layer into a 1D vector, so that we can feed it to a fully connected layer in the next step.
- Continue with a fully connected layer 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. Adam and RMSProp are commonly known to give good performances