Multi-layer perceptrons with Keras

In this demo, we will train and test a multi-layer perceptron model on the MNIST handwritten digits dataset using Keras.

A similar (but more complicated) example is

https://keras.io/examples/vision/mnist_convnet/

1. Load dataset

Like sklearn, Keras also provides an API to download and load the MNIST dataset. Download the data, load it into memory, and convert pixel values to [0, 1].

```
from keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
```

2. Flatten the inputs into vectors (Group 1)

When we load the data with Keras, the images are not flattened. So we flatten them into vectors (i.e., the input is of size n x 784) to train with an MLP model.

Several ways:

- Use unknown: `X_train_flat = X_train.reshape((60000,-1))`
- Use known value: `X_train_flat = X_train.reshape((60000,784))`
- Use index of shape: `X_train_flat = X_train.reshape((60000,X_train.shape[1)*X_train.shape[2]))`
- Use multiply: `X_train_flat = X_train.reshape((60000,28*28))`

3. Convert label vectors into one-hot encodings (Group 2)

When using Keras for classification, the labels have to be converted into one-hot encoding vectors. We can do this using the to_categorial method.

```
from tensorflow import keras
Y_train=keras.utils.to_categorical(Y_train, num_classes=10, dtype=float)
Y_test=keras.utils.to_categorical(Y_test, num_classes=10, dtype=float)
```

4. Define the MLP model (Group 3)

We can define an MLP model using a Sequential model and the Dense layers. In most cases, we will define a model as a Sequential model, and then add layers to it one-by-one.

Let's create a network with 2 hidden layers, 50 nodes each, with your choices of activation functions.

```
from keras import Sequential
from keras.layers import Dense


MLP = Sequential()
MLP.add(Dense(50, activation='relu'))
MLP.add(Dense(50, activation='relu'))
MLP.add(Dense(10, activation='softmax'))model.compile(optimizer='sgd',
loss='categorical_crossentropy', metrics=['accuracy'])
```

5. Compile the model (Group 4)

Before training a Keras model, we need to compile it to set up all the options for training, such as loss function, optimizer, and evaluation metrics. Here we will use cross entropy loss and the SGD optimizer. Our evaluation metric will be accuracy.

```
model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])
```

6-7. Train the model and evaluate the trained model on test set (everyone)

Now we can train the model using the fit(...) method. We can specify the number of epochs and batch size for training. Finally, we can compute the model accuracy on the test set.

```
history = model.fit(x=X_train_flat, y=Y_train_one_hot, epochs=30)
print(history.history)
```

```
score = model.evaluate(X_test, Y_test, verbose=0)

print("Test loss:", score[0])

print("Test accuracy:", score[1])
```