CS4341 Introduction to Artificial Intelligence
Project 3 - A Term 2017

By Michael Sokolovsky, Ahmedul Kabir and Prof. Carolina Ruiz

Project Description and Dataset

Primary Goal:
In this project, you will build a neural network model for categorizing images. You will write a program that takes images like the hand-written numbers above and output what numbers the images represent (e.g., "4" for the first image, "3" for the second image, and so on).

Dataset:
MNIST is a dataset composed of handwritten numbers and their labels. It is a famous dataset that has been used for testing new machine learning algorithm's performance. Each MNIST image is a 28x28 grey-scale image. Data is provided as 28x28 matrices containing numbers ranging from 0 (corresponding to white pixels) to 255 (corresponding to black pixels). Labels for each image are also provided with integer values ranging from 0 to 9, corresponding to the actual value in the image. There are 6500 images in our version of the database and 6500 corresponding labels. A link to this dataset is provided on this same project webpage.

Tasks:
In this project, your will experiment with training a Neural Network model for identifying which digit is represent by a MNIST image. Your task is to build a digit classifier using the artificial neural network package called Keras, implemented in Python3. The input to your model will be an image, and the output will be a classification of the number, from 0-9. You will get a chance to work with a common Neural Network package used in state-of-the-art research. In addition, you will practice using the numerical computation package Numpy for preprocessing.

Your goal will be to create a model that successfully identifies the digits represented by images in MNIST with a level of accuracy as high as you can but of at least 70%, and describe and analyze your results in a written report.

Project Requirements

Projects will include turning in a written report, code for training and visualization, and a model. Details of what to include are listed below:

Written Report

- ***Set of Experiments Performed***
  Include a section describing the set of experiments that you performed, what structures you experimented with (i.e., number of layers, number of neurons in each layer), what parameters you varied (e.g., number of epochs of training, batch size and any other parameter values, weight initialization scheme, activation function), and what accuracies you obtained on each of these experiments.
- ***Model & Training Procedure Description***
  Include a section describing in more detail the most accurate model you were able to obtain: the structure of your model, including number of layers, number of neurons in each layer, weight initialization scheme, activation function, number of epochs used for training, and batch size used for training.
- ***Graph***
  Include a graph showing how training accuracy and validation accuracy change over time during the training of your best model. Graph epochs versus training set and validation set accuracy.
- ***Model Performance & Confusion Matrix***
  Include a confusion matrix showing results of your best model reported on the test set. The matrix should be a 10x10 grid showing which categories images were classified as. Use your confusion matrix to additionally report precision and recall for each of the 10 classes, as well as overall accuracy of your model.
- ***Visualization***
  Include 3 visualizations of images that were misclassified by your best model and any observations about why you think these images were misclassified. You will have to create or use a visualization program that takes a 28x28 matrix input and translate it into a black-and-white image.

Code:

- ***Model Code***
  Please turn in your preprocessing, model creation, model training, graphing and confusion matrix code.

Model

- ***Copy of Trained Model***
  Turn in a copy of your best model saved as 'trained_model.h5.' Please use the following Keras methods for saving your model.

Project Preparatory Tasks and Guidelines

Below are import guidelines to follow for implementing the project. **A model template is provided for you on this project webpage, and these guidelines follow the structure of the template.**

1) Installing Software and Dependencies

template.py (available on the project webpage) is written with the Keras API in a python3 script. You will use this template to build and train a model. To do so, you will need to implement the project in Python3 and install Keras and its dependencies. *Please make sure you have a working version of Python3 and Keras as soon as possible, as these programs are necessary for completing the project.*

2) Downloading Data

Raw data is provided and can be downloaded from the project webpage. Images are provided for you in the **images.npy** file, which contains 6500 images from the MNIST dataset. **labels.npy** contains the 6500 corresponding labels for the image data.

3) Preprocessing Data

All data is provided as NumPy .npy files. To load and preprocess data, use python's NumPy package

Image data is provided as 28x28 matrices of integer pixel values. However the input to the network will be a flat vector of length 28*28 = 784. You will have to flatten each matrix to be a vector, as illustrated by the toy example below:

$$\begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \\ p & q & r & s & t \\ u & v & w & x & y \end{bmatrix} \rightarrow \begin{bmatrix} a & b & c & d & e & f & g & h & \dots & w & x & y \end{bmatrix}$$

The label for each image is provided as an integer in the range of 0 to 9. However the output of the network should be structured as a "one-hot vector" of length 10 encoded as follows:

$$0 \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$1 \rightarrow \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$2 \rightarrow \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$etc...$$

To preprocess data, use NumPy functions like reshape for changing matrices into vectors. You can also use Keras's to_categorical function for converting label numbers into one-hot encodings.

After preprocessing, you will need to take your data and randomly split it into Training, Validation, and Test Sets. In order to create the three sets of data, use stratified sampling, so that each set maintains the same relative frequency of the ten classes.

You are given 6500 images and labels. The training set should contain ~60% of the data, the validation set should contain ~15% of the data, and the test set should contain ~25% of the data.

Example Stratified Sampling Procedure:

- Take data and separate it into 10 classes, one for each digit
- From each class:
  - take 60% at random and put into the Training Set,
  - take 15% at random and put into the Validation Set,
  - take the remaining 25% and put into the Test Set

3) Building a Model

```
# Model Template

model = Sequential() # declare model
model.add(Dense(10, input_shape=(28*28, ), kernel_initializer='he_normal')) # first layer
model.add(Activation('relu'))
#
#
#
# Fill in Model Here
#
#
model.add(Dense(10, kernel_initializer='he_normal')) # last layer
model.add(Activation('softmax'))
```

In Keras, Models are instantiations of the class Sequential. A Keras model template written with the Sequential Model API is provided which can be used as starting point for building your model. The template includes a sample first input layer and output layer. You must limit yourself to "Dense" layers, which are Keras' version of traditional neural network layers. This portion of the project will involve experimentation.

Good guidelines for model creation are:

- Initialize weights randomly for every layer, try different initialization schemes.
- Experiment with using ReLu Activation Units, as well as SeLu and Tanh.
- Experiment with number of layers and number of neurons in each layer, including the first layer.

Leave the final layer as it appears in the template with a softmax activation unit.

4) Compiling a Model

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

Prior to training a model, you must specify what your loss function for the model is and what your gradient descent method is. Please use the standard categorical cross-entropy and stochastic gradient descent ('sgd') when compiling your model (as provided in the template).

5) Training a Model

```
# Train Model
history = model.fit(x_train, y_train,
                    validation_data = (x_val, y_val),
                    epochs=10,
                    batch_size=512)
```

You have the option of changing how many epochs to train your model for and how large your mini-batch size is. Experiment to see what works best. Also remember to include your validation data in the fit() method.

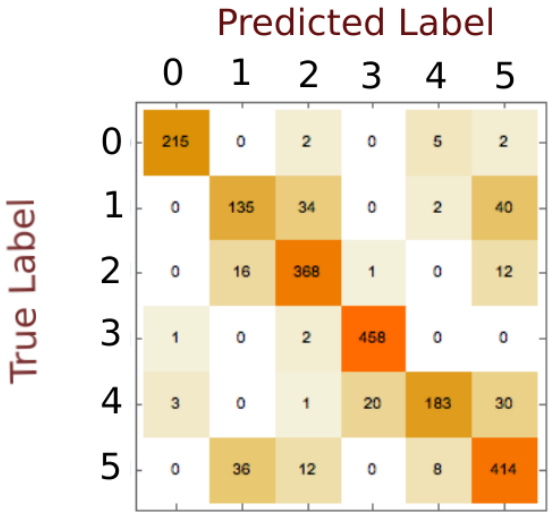
6) Reporting Your Results

```
print(history.history)
```

fit() returns data about your training experiment. In the template.py this is stored in the "history" variable. Use this information to construct your graph that shows how validation and training accuracy change after every epoch of training.

```
model.predict()
```

Use the predict() method on model to evaluate what labels your model predicts on test set. Use these and the true labels to construct your confusion matrix, like the toy example below, although you do not need to create a fancy visualization. Your matrix should have 10 rows and 10 columns.



Grading Rubric

Report

- Set of Experiments Performed – **20 pts**
- Model & Training Procedure Description – **10 pts**
- Graph – **10 pts**
- Model Performance & Confusion Matrix – **10 pts**
- Visualization – **10 pts**

Code:

- Model Code – **30 pts**

Model

- Copy of Trained Model – **10 pts**

Total Points: 100 pts