WPI Worcester Polytechnic Institute

Computer Science

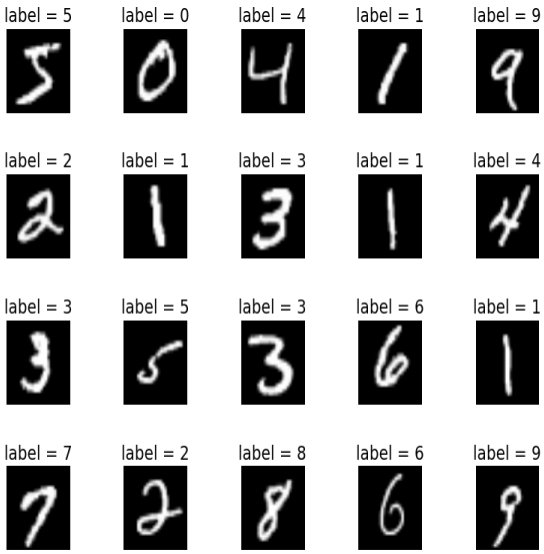
CS534 Artificial Intelligence
Homework 6 - Spring 2019

Michael Sokolovsky, Ahmedul Kabir and Prof. Carolina Ruiz

Due Date:
See homework webpage for due date and submission details

---

1. Project Description & Dataset



(Image taken from https://corochann.com/)

Primary Goal:
In this project you will build machine learning models for categorizing images. You will write programs that takes images like the hand-written numbers above and output what numbers the images represent. The two machine learning algorithms to be used are:

- Artificial Neural Network (ANN)
- Decision Tree (DT)

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.

Tasks:
In the project you will experiment with training machine learning models for identifying which digit is represented by a MNIST image. You will then submit a report describing your experimental methods and results. For Artificial Neural Network, you should use a package called **Keras** implemented in Python3. For Decision tree, you should use Python's Scikit-learn package. The input to your model will be an image, and the output will be a classification of the number, from 0-9. You'll get a chance to work with common machine learning packages used in state-of-the-art research. In addition, you will practice using the numerical computation package Numpy for preprocessing.

2. Project Requirements

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

Written Report

- ***Model & Training Procedure Description***
  For each classification algorithm, include sections describing the experiments that you performed varying hyper-parameters. For example, for ANN include the number of layers, number of neurons in each layer, number of epochs used for training, and batch size used for training. For decision trees, include parameter values you experimented with.
- ***Model Performance & Confusion Matrix***
  For each algorithm, describe the best performing model you obtained. Include a confusion matrix showing results of testing the model on the test set. The matrix should be a 10-by-10 grid showing which categories images were classified as. Use your confusion matrix to additionally report precision & recall for each of the 10 classes, as well as overall accuracy of your model.
- ***Plot***
  For your best performing ANN, include a plot showing how training accuracy and validation accuracy change over time during training. Graph number of training epochs (x-axis) versus training set and validation set accuracy (y-axis). Hence, your plot should contain two curves.
- ***Visualization***
  For each algorithm, include 3 visualizations of images that were misclassified by your best performing 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 28-by-28 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.

- ***Copy of Trained Model:***
  Turn in a copy of your best model saved as `trained_model.hw6.' Please use the following Keras methods for saving your model.

3. Project Preparatory Tasks and Guidelines

3.1 Artificial Neural Networks

Below are important guidelines to follow for implementing the project using ANNs. A model template is provided for you, and these guidelines follow the structure of the template.

1) Installing Software and Dependencies

template.py 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 following links:

- The **images.npy** file 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 28-by-28 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 create the three sets of data, use stratified sampling, so that each set contains the same relatively 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 a starting point for building your model. The template includes a sample first input layer and output layer. You must limit yourself to "Dense" layers - 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.

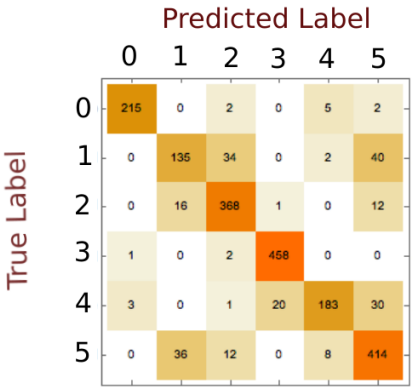
6) Reporting Your Results

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

fit() returns data about your training experiment. In the template 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.



3.2 Decision Tree Classification

Work with the same training, validation, and test sets you constructed earlier to make decision tree models for classifying the digits. You will focus on making three variations on decision trees and documenting the results and experiments in the report.

1) Creating a Baseline Model

Using the scikit-learn class DecisionTreeClassifier, train a decision tree on the raw pixel data just as you did with your neural networks, and report your results on the validation set. This is your baseline validation performance. Write about your experiments and include them in the **Description of Experiments** section.

2) Variation on the Baseline Model

Try changing the parameters of your tree, such as restricting tree depth or utilizing pruning to see if you can reduce generalization error. Write about your experiments and include them in the **Description of Experiments** section.

3) Creating Hand-Engineered Features

Try creating your own discriminatory features (between 4-10) from the raw pixel data and training a decision tree on those features. Talk about what features you chose to create in the **Description of Experiments** section. One example would be to include the average pixel intensity over the whole 28x28 image as a feature. You may want to consider working with the original 28x28 prior to come up with features. This is an opportunity to think creatively about what transformations to the raw data would help at discriminating between different numbers.

4) Reporting Results

Finally, run the above three final models on your **test set** and **report performances and confusion matrices.** How do your results compare to your neural network results?

4. Grading Rubric

Report

- Neural Networks
  - Description of Experiments – **10 pts**
  - Model & Training Procedure Description – **15 pts**
  - Plot – **10 pts**
  - Visualization – **5 pts**
  - Model Performance & Confusion Matrix – **5 pts**
- Decision Trees
  - Feature Extraction & Explanation – **10 pts**
  - Description of Experiments – **10 pts**
  - Model Performance & Confusion Matrix – **5 pts**

Code:

- Neural Network **- 20 pts**
- Decision Tree Code – **15 pts**

Model:

- Copy of Trained Model: **5 pts**

Total Points: 100 pts + 10 bonus pts