Intelligent Systems: Reasoning and Recognition

James L. Crowley

MoSIG M1 Winter Semester 2021
Lesson 10 11 March 2021

Network Programming Exercise:
Recognizing Handwritten Digits using Neural Networks

Outline:

The MNIST Digits Dataset 2

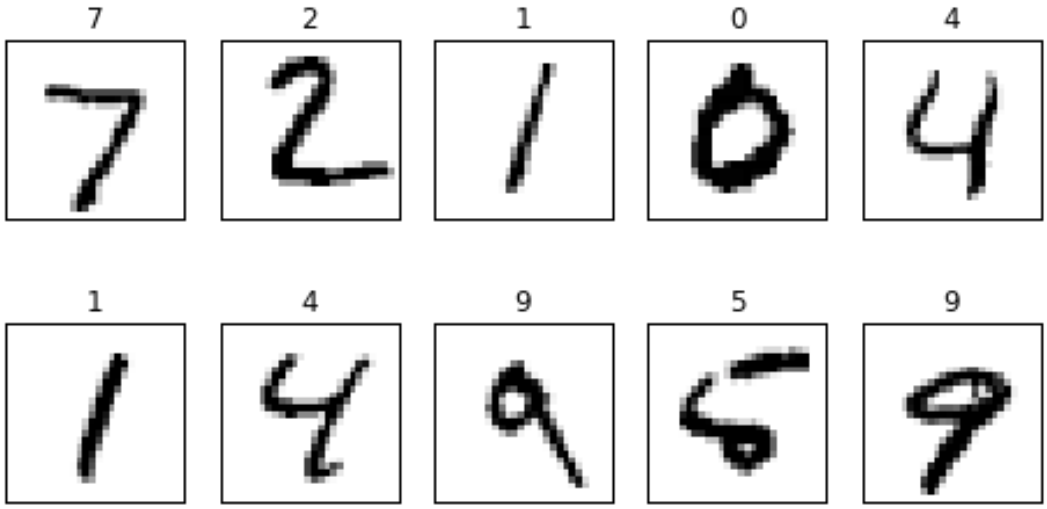
Installing a miniconda Programming Environment. 4

- Python 4
- Conda Python 4
- Installing MiniConda 5
- Installation of MiniConda on an Apple Macintosh 5
- Installation of MiniConda on Linux: 5
- Numpy 6
- Matplotlib 6
- Jupyter Notebooks 6

Keras 8

- Keras Code Examples 9
- A Simple Example of an MLP for MNIST digits 11
- Keras Example: a simple CNN for MNIST Digits 13

The MNIST Digits Dataset



The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of handwritten digits composed of 60,000 training images and 10,000 test images. The database was created by "re-mixing" samples of digits from NIST's original datasets taken from American Census Bureau employees and American high school students as part of the 1990 US Census. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced gray-scale levels.

Your task is to design and evaluate neural network architectures that can recognize hand-drawn digits using the gray-scale MNIST images. Your networks can consist of convolution layers, fully connected layers, an auto-encoder, or any combination of these or other neural network techniques. You may use weights that are pre-trained on a different dataset or trained from scratch. Your must provide objective performance evaluation, using metrics such as error rates, ROC curves, accuracy, precision and recall. It is recommended that you use Keras with the tensorflow backend libraries running in a minconda environment to help you in completing this exercise. However, you may use any software tools and framework that you wish.

This project should be performed by teams of 3 students, and should be described by a written report with descriptions and analysis of performance evaluation results for the techniques that are tested. Reports may be written in French or English. Programming teams are given freedom in their choice of techniques to evaluate. The following is an indicative barometer for grading. Actual grades will depend on a subjective appreciation for the amount of effort deployed and the depth of understanding displayed in the results, and the clarity of the report. Creativity is encouraged and will be rewarded! Reports are due on the Monday 12 April 2021.

Grade	Example of Criteria (Max grade is 20)
8	Construct and describe a fully connected multi-layer network to recognize MNIST digits. Divide the MNIST training data into a training set (80%), an evaluation set (10%) and a test set (10%). Use the training set to train your network over multiple epochs while displaying accuracy for your training set and your evaluation set. Stop training when the accuracy scores diverge. Present results for Accuracy, Precision, Recall, F1, AUC for ROC curves, using the test set.
+1 to +4	Using your initial network as a baseline technique, demonstrate the effects of changing the number of layers (+1) and the number of units per layer (+1), learning with different learning rates (+1) and using different optimization techniques, by providing Accuracy, Precision, Recall, F1, and AUC for ROC curves for each network.
8	Implement a multi-layer Convolutional network to recognize MNIST digits similar to the network provided as an example on the course web site, or similar to LeNet5. Use the training set to train your network over multiple epochs while displaying accuracy for your training set and your evaluation set. Stop training when the accuracy scores diverge. Present results for Accuracy, Precision, Recall, F1, AUC for ROC curves, using the test set.
+1 to +4	Using your initial convolution network as a baseline technique, demonstrate the effects of changing the number of convolutional layers (+1) and the number of filters per layer (+1), different types and sizes of pooling layers (+1), and number of fully connected layers (+1), by providing Accuracy, Precision, Recall, F1, and AUC for ROC curves for each network.
+1	Demonstrate the effects for different values of dropout.
+2	Document the performance of a published network (LeNet5, VGG, Yolo, etc) for recognizing MNIST Digits.
+2 to +6	Use your best network to build a real time system to recognize your own hand written digits.

Installing a miniconda Programming Environment.

This lab exercise can be done in any programming environment. However, we recommend using Jupyter Notebooks with Keras in a MiniConda installation of Python.

Python

Python is an interpreted, high-level programming language that is widely used in machine learning research. Python was created in the late 1980s by Guido van Rossum at the CWI reseach center in the Netherlands as a language that emphasizes code readability. Its language constructs and object-oriented approach are intended to help programmers write clear, logical code for small and large-scale projects. Python is ideal for rapid protyping of software.

Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible with Python 2.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation does not have semantic meaning.

You can find many on-line tutorials and MOOCs on the web.
For example, https://www.python.org/about/gettingstarted/

Conda Python

Conda is an open source environment and package management system that runs on Windows, Apple macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda can easily be used to create, save, load and switch between environments on your a computer. It was created for Python programs, but can package and distribute software for any language including C and HTML.

As a package manager, conda makes it easy to find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run a different version of Python, while continuing to run your usual version of Python in your normal environment.

In its default configuration, conda can install and manage the thousands of packages available at repo.anaconda.com that are built, reviewed and maintained by Anaconda.

We will use a simple minimal version referred to as miniconda.

Installing MiniConda

MiniConda install packages are available Linux, Apple MacOS, or MS Windows.
Installer packages for full anaconda can be found at
https://www.anaconda.com/download/
The installer packages for miniconda are at (https://conda.io/miniconda.html)

The following is my personal installation guide for 2020, that is adapted for computer vision using OpenCV.

Installation of MiniConda on an Apple Macintosh

Miniconda can be installed from a .pkg file
(Miniconda3-latest-MacOSX-x86_64.pkg)
or using bash with a .sh file
(Miniconda3-latest-MacOSX-x86_64.sh).
Both are available at https://conda.io/miniconda.html

For a bash installation,
1) navigate to https://docs.conda.io/en/latest/miniconda.html
2) download Miniconda3-latest-MacOSX-x86_64.sh
3) open a terminal and navigate to the directory where the .sh may found and run:
$ bash Miniconda3-latest-MacOSX-x86_64.sh

Installation of MiniConda on Linux:

For installation on a Linux system.
1) navigate to https://docs.conda.io/en/latest/miniconda.html
2) download Miniconda3-latest-Linux-x86_64.sh to and run:
3) open a terminal and navigate to the directory you downloaded to and run:
$ bash Miniconda3-latest-Linux-x86_64.sh

After installation, create and activate a Python environment. I use Python 3.7 for compatibility with OpenCV. If you are not using OpenCV you may use the latest Python release.

Create and activate a Python conda environment. For example, the following creates an environment named SIRR

$ conda create -n SIRR python=3.7
$ source activate SIRR

Numpy

Install the Numpy library (http://www.numpy.org/)

NumPy is an open source project intended to enable numerical computing with Python. Numpy was created in 2005, building on the existing Numeric and Numarray libraries. Numpy is extremely useful and widely used for machine learning.

The latest version is numpy=1.20. For OpenCV it is useful to use an older (1.16.4) version of Numpy to avoid warnings of impending changes in syntax.

Numpy is directly available in MiniConda.

$ conda install numpy=1.16.4

Matplotlib

Matplotlib (https://matplotlib.org/) is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib is directly available in MiniConda. Type:

$ conda install matplotlib

Jupyter Notebooks.

Install Jupyter Notebooks (http://jupyter.org/)

Jupyter notebooks are widely used for collaborative machine learning.

A Jupyter Notebook is an open-source web application that allows creation and sharing of documents that contain live code, equations, visualizations, HTML markups and narrative text. Jupyter notebooks provide a browser-based tool for interactive authoring of documents that may combine explanatory text, mathematics, computations and their rich media output.

Jupyter notebooks provide:

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- The ability to execute code from the browser, with the results of computations attached to the code which generated them.

- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc.
- In-browser editing for rich text using the Markdown markup language, which can provide commentary for the code, is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using LaTeX, and rendered natively by MathJax.

To install Jupyter Notebooks with miniconda, type:

$ conda install jupyter notebook

Keras

Keras is a deep learning API written in Python, created by Francois Chollet at Google for running experiments with the Google machine learning platform TensorFlow. Keras offers consistent and simple APIs. Keras minimizes the number of user actions required for common use cases, and attempts to provide clear and actionable feedback for user errors.

TensorFlow is an end-to-end, open-source machine-learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities:

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices
- Exporting programs to external runtimes such as servers, browsers, mobile and embedded devices.

Keras is the high-level API of TensorFlow 2: an approchable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

Keras allows engineers and researchers to take advantage of the scalability and cross-platform capabilities of TensorFlow 2. Keras can be run on TPU or on large clusters of GPUs, and it is possible to export networks trained with Keras to run in the browser or on a mobile device. See: https://keras.io/about/

The core data structures of Keras are layers and models. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, you can use the Keras functional API, which allows to build arbitrary graphs of layers, or write models entirely from scratch via subclasssing.

Keras is available directly in miniconda. To install keras, type:

$ conda install keras

This should also install tensorflow

It is strongly recommended that you review the tutorial at
https://keras.io/getting_started/intro_to_keras_for_engineers/

Keras Code Examples

You can create a simple fully connected network in Keras as a set of layers using constructors from the Keras "sequential model".
The MNIST data set is directly available using the mnist library.

```
import numpy as np
import tensorflow as tf
from tensorflow import  keras
from keras.models import Sequential
from keras.layers import Dense
from keras.datasets import mnist
```

The following creates a network with 2 layers.
The first layer has 784 units, with relu activation.
The second layer has 10 units and uses softmax activation.

```
model = Sequential([
        Dense(32, input_shape=(784,),activation='relu')
        Dense(10,activation='softmax')
])
```

You can compile this model, using categorical crossentropy loss function and the adam (adaptive moment) optimisation with

```
# Compile with Adam optimizer and categorical Cross entropy

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

We can train a model using model.fit.

```
history=model.fit(trainingSet, trainingLabel, validation_split=0.30, epochs=5,
batch_size=128)

# to list all data in history
print(history.history.keys())
```

Assigning the results to history allows us to plot the evolution of the model as it is trained over 5 epochs.

Keras offers a variety of activation functions, loss functions and optimizers.
See the Keras documentation for descriptions and examples.

A number of classic CNN architectures can be found in the Keras Libraries.
These include VGG16, VGG19 and ResNet50.

For example, VGG16 is a classic CNN architecture that is very useful for transfer learning. (we will study this next lecture).

To obtain a VGG 16 network that has been pre-trained with the imagenet dataset:

From keras.applications.vgg16 import VGG16.

Model=VGG16(weights='imagenet', include_top=True).

Keras also contains a library for data augmentation to increase the size of a training set by deforming the training data with affine transformations or color transformation.

Some useful links:
Miniconda: https://docs.conda.io/en/latest/miniconda.html
Keras: https://anaconda.org/conda-forge/keras
Keras documentation: https://keras.io/
MNIST Dataset: http://yann.lecun.com/exdb/mnist/

A Simple Example of an MLP for MNIST digits

This is a simple MLP for recognizing MNIST Digits that was provided by one of the MOSIG M1 teams from the 2019/2020 academic year. .
The jupyter notebook for this example can be downloaded from the course web site.

```
# import environmental variables.

import tensorflow as tf
from keras import layers
from keras.datasets import mnist
from keras.models import Sequential

import matplotlib.pyplot as plt
import math
from keras.layers import Dense
import tensorflow.keras.backend as K
from keras.utils import to_categorical

import os
import keras
from keras.callbacks import Callback
import matplotlib.pyplot as plt
import numpy as np
# from scikitplot.metrics import plot_confusion_matrix, plot_roc

# for history
from glob import glob

# Load the MNIST Digits
# Keras provide a command to directly load the MNIST digits

(trainingSet, trainingLabel), (testSet, testLabel) = mnist.load_data()

# Define a 3 layer MLP
# Flattent the 28x28 MNIST digits to a 784 coefficient vector.
# add 2 dense layers with 784 units and relu activation,
# followed by an output layer with 10 units and softmax.

model = Sequential()
model.add(layers.Dense(784, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(784, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))

# show the model

model.summary()

# Compile with Adam optimizer and categorical Cross entropy

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

```
trainingSet = trainingSet.reshape((60000, 28 * 28))
trainingSet = trainingSet.astype('float32') / 255
testSet = testSet.reshape((10000, 28 * 28))
testSet = testSet.astype('float32') / 255

# Train the model and save teh history
trainingLabel = to_categorical(trainingLabel)
testLabel = to_categorical(testLabel)
history=model.fit(trainingSet, trainingLabel, validation_split=0.30, epochs=5,
batch_size=128)

test_loss, test_acc = model.evaluate(testSet, testLabel)
print('test_acc:', test_acc, 'test_loss', test_loss)

# list all data in history
print(history.history.keys())

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Keras Example: a simple CNN for MNIST Digits

Here is a simple example of Keras code to load the MNIST Digits, and train a model.

```
# Make sure images have shape (28, 28, 1)

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# A model for a 2 layer CNN with 3x3 kernels followed by 2x2 pooling

model = keras.Sequential(
   [
      keras.Input(shape=input_shape),
      layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Flatten(),
      layers.Dropout(0.5),
      layers.Dense(num_classes, activation="softmax"),
   ]
)

model.summary()

batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```