2.12: Introduction to Robotics
Lab 10:
Machine Learning*

Spring 2020

Instructions:

1. **You will be working on this lab alone on your individual computer, but if you face any issues, contact a TA. Feel free to work alongside a fellow classmate, but turn in your own work.**
2. **Submit a PDF of the collection of screenshots and answers to the questions that are asked for this lab to Stellar.**

1 Introduction

In this lab, you will experiment with machine learning techniques. Specifically, you will:

1. Use support vector machine (SVM) to differentiate between two linearly separable classes
2. Use SVM to differentiate between two nonlinearly separable classes
3. Use a neural network (NN) for image classification
4. Use a convolutional neural network (CNN) for image segmentation

We will use `scikit-learn` for the SVM section, and TensorFlow for the neural network sections. For the first two sections, you will be using data created specifically for this lab. In the last two sections, you will be using standard data sets that are available online. The scripts included will download each data set.

2 Setting up

2.1 Scikit-Learn

For the SVM section of this lab, you will be using Scikit-learn.

```
pip install scikit-learn
```

*

1. Version 1 - 2020: Jerry Ng, Rachel Hoffman-Bice, Steven Yeung and Kamal Youcef-Toumi

2.2 TensorFlow

For the neural network section of this lab, you will be using TensorFlow.

```
pip install tensorflow
```

2.3 Lab code

The github repository can be found here

```
https://github.com/mit212/Lab10_2020.git
```

Please clone or download the zip file into a known location.

3 1. Support Vector Machine

As you may recall from lecture, SVMs is a suprevised learning method that can be used to separate data based on features. A support vector is a vector that essentially draws the boundaries between these classes based training using known, labeled data points.

3.1 Linearly Separable Case

First, navigate to the SVM folder and open p1abc.py in your favorite text editor. At the top you will notice three boolean variables, p1a, p1b, p1c. For now, please set p1b and pic to false. Go ahead and run the code. You should see a figure pop up with a bunch of red data and a bunch of blue data in distinct groups. This is the known and classified data that we will use to train our first SVM.

Now, make p1b true.. This is where we actually train our data.

Within the p1b IF statement starting around line 46, you will see the following command:

```
clf=svm.LinearSCV()
clf.fig(data,val)
```

the first command makes clf an instance of the LinearSCV class and the second command uses the fit method to generate a support vector that separates the(x,y) data points based on known their value/classification. In this case, if you see in `data_a.xlsx` the red points in the bottom left corner are classified as a "0" and the blue points are classified as a "1".

After the data has been fitted, the svm is used to predict the classification of two additional test points, plotted with + signs. You should see that they both appear blue, meaning the svm classified those data parts as most likely belonging to the "blue/1" label.

Now, change the p1c boolean to True at the top and run the code. Basically the same plot should appear but now there is a black line running through the middle of the graph. This black line is the decision boundary determined by the SVM!

Question 1 *What do you think might happen if a data point were to fall exactly on the decision boundary?*

3.2 Nonlinear SVM

Now open p1def.py in your favorite text editor. At the top you will notice three boolean variables, p1d, p1e, p1f. For now, please set p1e and pif to false. Go ahead and run the code. You should see a figure pop up with a bunch of red data and a bunch of blue data in distinct groups. Now here it can be fairly obvious that a simple line will not separate the data. This is where using different kernels comes in to play!

change p1e to True and run the code again. Inthis section, instead of using the svm.LinearSVC method, we are using the svm.SVC method. By default, the svm.SVC method uses a radial basis function as its kernel. Here you will see an additional data point indicated by a "+". Although the point appears to be directly between the four clusters of data, the SVM has classified it as "blue". why?

The answer can be revealed by plotting the decision boundaries! Now change p2f to true and run the code again. Here you should see the same plot but with the decision boundaries dictated by a solid black line, the margins dictated by dashed lines. If you recall, the goal of SVM is to find a decision boundary that maximizes the margins separating the two data sets. You'll also notice that some points are circled with green lines. These points are the support vectors, basically the points that have the most influence on determining the location of the decision boundary. In this case, the decision boundary connects the two blue sections in the middle, while cutting off the red sections from each other.

Now, lets try changing the kernel and see what happens. Go to the line of code in the p1e IF statement (for me it is line 47) and change it to the following

```
clf = svm.SVC(kernel='poly')
```

The following kernels are available for use: 'linear', 'poly', 'rbf', 'sigmoid'.

Question 2 *How well does each kernel appear to classify the data? Show some screenshots of the different kernels being used.*

The full definintion of the SVC method with its default values is as follows and can be found here https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale',
coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=-1,
decision_function_shape='ovr', break_ties=False,
random_state=None)
```

You can change many of these variables if you wish, but for now lets just see what happens if we play with the polynomial kernel. By default, it is set to degree=3. Lets use a higher degree and see if it helps.

```
clf = svm.SVC(kernel='poly', degree=4)
```

Question 3 *Does changing the polynomial kernel degree help? Which one appears to be the best? Is there a disadvantage to using higher degree polynomial functions?*

4 Neural Network

This is the classical "hello world" example of neural networks used to classify handwritten images of numbers to the number that's written. Run the following command:

```
cd nn
python classifier.py  #or python3 classifier.py
```

An window will show up, there's a plot of the loss value and the accuracy over time as there are 5 sets of training called "epochs". As the epoch proceeds, we can see that the accuracies in both the training set and the test set increase as expected. Notice this is an ideal case. Over-fitting could happen if the epoch number is set too high and under-fitting could happen is the number is too low. Try go into the source code and give it a higher epoch number.

Now take a closer look into the code itself. First we installed the Tensorflow library. It is an open source library developed by Google for convenient and efficient deployment of common machine learning techniques. Keras is the neural network library that is built on Tensorflow. An alternative library is Pytorch, developed by Microsoft and Facebook, feel free to implement with both libraries and make a comparison. In 2.12 we will stick with Tensorflow.

In this lab we use the MNIST data set, which is a set of handwritten images of numbers that are correctly labelled. Each image contains 28×28 pixels. In this script, we use 60,000 image to train our network and 10,000 to test the network. Several steps to getting the input data to the write format. The 28×28 pixels are converted to a single array. Then they are fed through the net, where the trained y value is the number corresponding to the image.

Here we use two layers of neurons, with a 'sigmoid' and a 'softmax' activation function. There are a lot other activation functions, such as 'relu' and 'tanh'. Give it a try and see how that changes the result. Here, we use stochastic gradient decent to find our global minimum. Alternative optimizers such as 'adam' and 'adagrad' are also included in Keras. A cool gif of their performance can be found here: :`https://mlfromscratch.com/optimizers-explained/#/`

Eventually, we give each image sample 10 scores based on the output of the last layer of neuron. These 10 scores are the probabilities, corresponding to the numbers 0-9, that are evaluated with a mathematical technique called cross-entropy. The index with the highest probability is the number predicted by the image.

Question 4 *Why do we convert the 2-dimensional 28-28 input matrix into a 784 x 1 array? How is that different from convolutional neural network?*

Question 5 *Do you notice any interesting correlation between sigmoid and softmax as activation functions?*

5 Convolutional Neural Network

You will first need to download the dataset:

```
pip install tensorflow_datasets  #pip3 if python 3
```

And then run the following code, it might take a while:

```
python image_seg_ex.py  #python3 if python 3
```

This is an example of more advanced machine learning that would be likely necessary to detect something like an transparent water bottle. In this example, we will segment the images and classify them. We use the Oxford IIIT Pet dataset, where 37 categories of pets are correctly segmented from the image and classified. This specific example is using a modified version of a U-Net, which has several down sampling and then up sampling layers. The down-sampling encoder

is called MobileNetV2, details can be found in `https://arxiv.org/pdf/1801.04381.pdf`. The up-sampling decoder is the pix2pix package from the Tensorflow example.

The code is done in several parts, the first is displaying the image, and the mask. The mask is the segmented image that we want to produce with the model, and the image is of an animal. The next part is the construction of the net and the fitting of the system. It requires a lot more training than the previous example, 20 epochs instead of 5. After each epoch, there is a callback written to save the weights at the end of the training and also a callback to display the predicted image after training. The last section is the training loss and validation loss after each epoch. There is also a saved model in the folder.

Question 6 *At which stage does 'convolution' comes in?*