# lab5_solution

December 10, 2018

## 1 COMP3222/6246 Machine Learning Technologies (2018/19)

## 2 Week 10 – Perceptrons, Deep Net, and Convolutional Neural Net

In this lab, we introduce how to implement a perceptron, a deep neural network and also a convolutional neural network (DNN). Though it is not a good practise, we use all the data to train and test our model for the purpose of demonstration. We also present you with a code that is working, but yields poor results. We expect you to spot these issues and improve the code. Exercises are also provided at the end of each section to improve your technical skill.

### 2.1 Setup

*Make sure that the following code is executed before every other sections of this lab*

```
In [0]: # To support both python 2 and 3
        from __future__ import division, print_function, unicode_literals

        # Common imports
        import os
        import numpy as np
        import tensorflow as tf

        # To plot nice figures
        %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt
        plt.rcParams['axes.labelsize'] = 14
        plt.rcParams['xtick.labelsize'] = 12
        plt.rcParams['ytick.labelsize'] = 12

        # Clear tensorflow's and reset seed
        def reset_graph(seed=None):
            tf.reset_default_graph()
            tf.set_random_seed(seed)
            np.random.seed(seed)
```

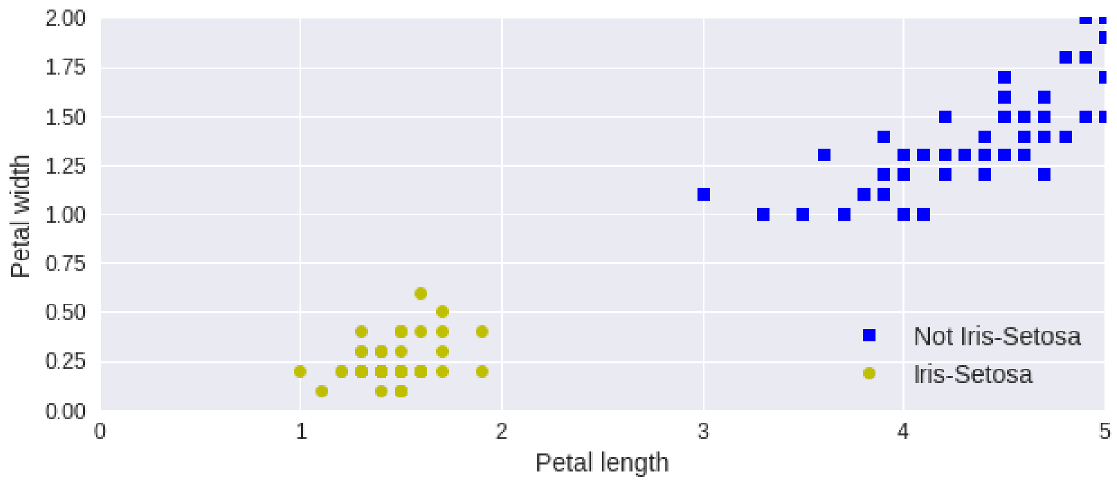
## 2.2 A Perceptron

In this section, we will use an artificial neuron (aka *perceptron*) to perform binary classification on linearly separable data. Specifically, we will use a portion of the Iris dataset; the description of this dataset can be found at http://scikit-learn.org/stable/datasets/index.html#iris-dataset.

```
In [0]: from sklearn.datasets import load_iris

        # get dataset
        iris = load_iris()
        X = iris.data[:, (2, 3)]  # use only petal length and petal width
        y = (iris.target == 0).astype(np.int) # classify them as either setosa or not setosa

        # visualise the data
        axes = [0, 5, 0, 2]
        plt.figure(figsize=(10, 4))
        plt.plot(X[y==0, 0], X[y==0, 1], "bs", label="Not Iris-Setosa")
        plt.plot(X[y==1, 0], X[y==1, 1], "yo", label="Iris-Setosa")
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
        plt.legend(loc="lower right", fontsize=14)
        plt.axis(axes)
        plt.show()
```



Clearly, this task can be easily done by using a linear classifier. Could you visualise the linear decision boundary on the figure above? Where should it be?

Now, let's move on to implementing a perceptron by using Scikit-learn.

```
In [0]: from sklearn.linear_model import Perceptron

        # initialise and train a perceptron
        pct = Perceptron(max_iter=100, random_state=None)
        pct.fit(X, y)
```

```
Out[0]: Perceptron(alpha=0.0001, class_weight=None, eta0=1.0, fit_intercept=True,
              max_iter=100, n_iter=None, n_jobs=1, penalty=None, random_state=None,
              shuffle=True, tol=None, verbose=0, warm_start=False)
```

Notice that there are many parameters that you can tweak later on. You can have a look at the description of each parameter in the Scikit-Learn's documentation http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

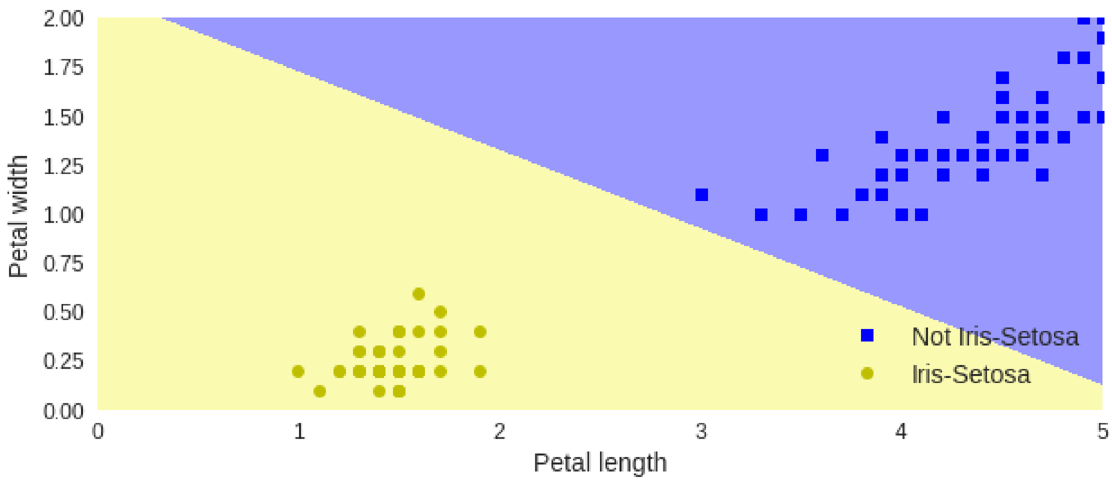
Next, we will extract the decision boundary from the model. Below we show a general way of extracting a decision boundary with any model. Note that it can be very computationally expensive if the feature space is large.

```
In [0]: # sampling and predict the whole space of features
        x0, x1 = np.meshgrid(
                np.linspace(axes[0], axes[1], 10).reshape(-1, 1),
                np.linspace(axes[2], axes[3], 10).reshape(-1, 1),
            )
        X_new = np.c_[x0.ravel(), x1.ravel()]
        y_predict = pct.predict(X_new)
        zz = y_predict.reshape(x0.shape)

        # plot the datapoints again
        plt.figure(figsize=(10, 4))
        plt.plot(X[y==0, 0], X[y==0, 1], "bs", label="Not Iris-Setosa")
        plt.plot(X[y==1, 0], X[y==1, 1], "yo", label="Iris-Setosa")

        # get a nice color
        from matplotlib.colors import ListedColormap
        custom_cmap = ListedColormap(['#9898ff', '#fafab0'])

        # plot the predicted samples of feature space
        plt.contourf(x0, x1, zz, cmap=custom_cmap)
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
        plt.legend(loc="lower right", fontsize=14)
        plt.axis(axes)
        plt.show()
```



***Exercise 1*** 1. The decision boundary of a single perceptron is a single straight line, but the above plot shows differently! Fix this plot. (*Hint*: you need to sample the feature space more)

*Solution*: Change the parameter of np.linspace above to generate more points. Have a look at the documentation: https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html.

2. Try running the code in [3] and [4] multiple times; two snippets above where a network is initialised, trained, and plotted. Do you always get the same decision boundary? Why?

   *Solution*: No, it's not the same boundary because we are using Stochastic Gradient Descent to train the perceptron and the random seed is also not fixed.

3. A single perceptron is not different from a linear classifier, which can be described by a straight line equation. Retrieve the formula for it. Verify that this is correct by comparing it with the plot above. (*Hint*: have a look in the list of attribute on the online documentation)

   *Solution*: Basically, a perceptron multiplies each feature with weight (some value), sum them up including the bias (or the intercept), then passes to heaviside function (as specified below). Therefore, the formula can be defined accordingly as a sum of weighted inputs plus bias equating to zero. Now, you need to recover the weights from the trained perceptron; coef_ and intercept_. Then with linear algebra, you will get a straight line equation that you need to confirm with your plot.

## 2.3 Activation Functions

There are many activation functions that can be used in a perceptron. Different functions result in different behaviours, and consequently different pros & cons. Though we will not go into details, it is beneficial for you to know some popular activation functions.

$$\text{heaviside}(z) = \begin{cases} 1 & \text{if } z >= 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{logit}(z) = \frac{1}{1 + e^{-z}}$$