Forward propagation ANN

En esta práctica vamos a implementar una red de neuronas simple. Para ello vamos a leer un dataset de 5000 imágenes de entrenamiento de dígitos escritos a mano. Están en formato Octave/Matlab y para leerlo necesitamos usar scipy.io.loadmat [1].

```
from scipy.io import loadmat, savemat
data = loadmat('data/ex3data1.mat', squeeze_me=True)
y = data['y']
X = data['X']
```

Cada ejemplo de entrenamiento es una imagen en escala de grises de 20x20 píxeles . Cada píxel está representado por un número de coma flotante que indica la intensidad de la escala de grises en ese lugar. La cuadrícula de 20 por 20 píxeles está aplanada en forma de vector de 400 dimensiones. Esto nos da una matriz X de 5000 por 400 en la que cada fila es una imagen manuscrita.

La segunda parte del conjunto de entrenamiento es un vector de 5000 dimensiones y que contiene etiquetas para el conjunto de entrenamiento etiquetadas de 0 a 9.

Figura 1: Ejemplo del dataset de entrenamiento.

Con estos datos de entrenamiento vamos a implementar una red de neuronas,

[1] This is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/).

utilizando los parámetros de una red ya entrenada.

El objetivo de esta parte es programar el algoritmo de forward propagation, utilizando los pesos en la predicción. La red se muestra en la Figura 2.

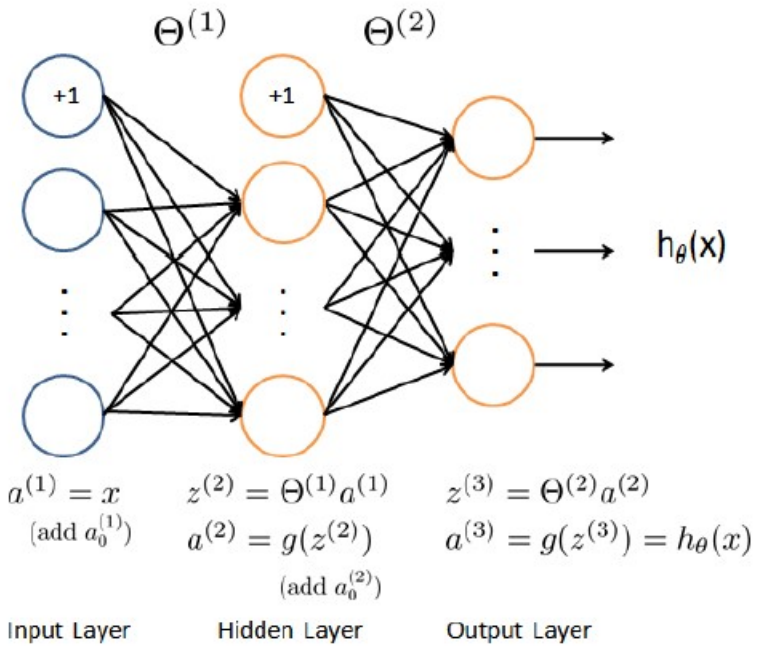

Figura 2: Modelo de NN a crear.

Tiene 3 capas: una capa de entrada, una capa oculta y una capa de salida. Recordemos que nuestras entradas tienen un tamaño de 400 unidades(excluyendo la unidad de sesgo adicional que siempre da como resultado +1).

Los parámetros pre-entrenados tienen 25 unidades en la segunda capa y 10 unidades de salida (correspondientes a las 10 clases de dígitos). Estos parámetros se pueden leer utilizando la función scipy.io.loadmat, como se muestra a continuación.

```
weights = loadmat('data/ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']
```

Ejercicio 1:

Completa el código de la función ANN.predict para obtener la predicción de la red neuronal. Debes implementar el cálculo feedforward que calcula $h(x(i))$ para cada ejemplo i y devuelve las predicciones asociadas. La predicción de la red neuronal será la etiqueta que tenga la mayor salida $(h(x))_k$.

Sugerencia: Para obtener una solución vectorizada del problema, es conveniente añadir una columna de 1 a la matriz X antes de calcular la propagación de la red neuronal (simula los valores de los umbrales que recordemos pueden modelarse como una entrada siempre asignada a 1) usando la función np.hstack que permite apilar matrices por columnas.

https://numpy.org/doc/stable/reference/generated/numpy.hstack.html

```
m = X.shape[0]
X1s = np.hstack([np.ones((m, 1)), X])


def predict(theta1, theta2, X):
    """
    Predict the label of an input given a trained neural network.

    Parameters
    ----------
    theta1 : array_like
        Weights for the first layer in the neural network.
        It has shape (2nd hidden layer size x input size)

    theta2: array_like
        Weights for the second layer in the neural network.
        It has shape (output layer size x 2nd hidden layer size)

    X : array_like
        The image inputs having shape (number of examples x image dimensions).

    Return
    ------
    p : array_like
        Predictions vector containing the predicted label for each example.
        It has a length equal to the number of examples.
    """

    return p
```

Aplicado a los parámetros proporcionados, Theta1 y Theta2, y a los ejemplos de entrenamiento en X se debería obtener una precisión (accuracy) de alrededor del **97,5%**.

Ejercicios 2:

Calcula la matriz de confusión para la predicción del 0. Es decir, asumimos que la clase positiva es 0 y el resto de clases son negativas. Calcula tambien precission, recall y F1-Score.

NOTA: En utils disponéis de una función que permite mostrar las imágenes que

podéis utilizar.

English version

In this practice we are going to implement a simple neural network. For this we are going to read a dataset of 5000 training examples images of handwritten digits. They are in Octave/Matlab format and to read them we need to use scipy.io.loadmat.[2].

```
from scipy.io import loadmat, savemat
data = loadmat('data/ex3data1.mat', squeeze_me=True)
y = data['y']
X = data['X']
```

Each training example is a 20x20 pixel grayscale image. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 pixel grid is flattened into a 400 dimensional vector. This gives us a 5000 by 400 matrix X in which each row is a handwritten image.

The second part of the training set is a 5000 dimensional vector containing labels for the training set labelled 0 to 9

Figure 1: Example of the training dataset.

[2]This is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/).

With this training data we are going to implement a neural network, using the parameters of an already trained network.

The objective of this part is to program the forward propagation algorithm, using the weights in the prediction. The network is shown in Figure 2.

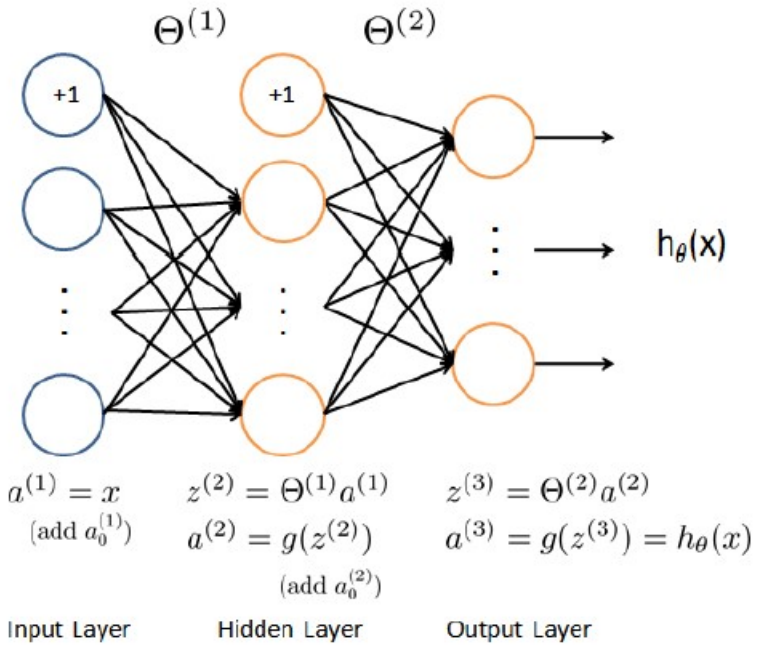

Figure 2: NN model to be created.

It has 3 layers: an input layer, a hidden layer and an output layer. Recall that our inputs have a size of 400 units (excluding the additional bias unit which always results in +1).

The pre-entered parameters have 25 units in the second layer and 10 output units (corresponding to the 10 digit classes). These parameters can be read using the scipy.io.loadmat function, as shown below.

```
weights = loadmat('data/ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']
```

Exercise 1:

Complete the code for the ANN.predict function to obtain the neural network prediction. You must implement feedforward computation that calculates $h(x(i))$ for each example i and returns the associated predictions. The neural network prediction will be the label with the highest output $(h(x))_k$.

Hint: To obtain a vectorised solution to the problem, it is convenient to add a column of 1 to the matrix X before calculating the neural network propagation (simulates the values of the thresholds which, remember, can be modelled as an input always assigned to 1) using the np.hstack function which allows stacking matrices by columns.

https://numpy.org/doc/stable/reference/generated/numpy.hstack.html

```
m = X.shape[0]
X1s = np.hstack([np.ones((m, 1)), X])


def predict(theta1, theta2, X):
    """
    Predict the label of an input given a trained neural network.

    Parameters
    ----------
    theta1 : array_like
        Weights for the first layer in the neural network.
        It has shape (2nd hidden layer size x input size)

    theta2: array_like
        Weights for the second layer in the neural network.
        It has shape (output layer size x 2nd hidden layer size)

    X : array_like
        The image inputs having shape (number of examples x image dimensions).

    Return
    ------
    p : array_like
        Predictions vector containing the predicted label for each example.
        It has a length equal to the number of examples.
    """

    return p
```

Applied to the parameters provided, Theta1 and Theta2, and to the training examples in X, an accuracy of about **97.5%** should be obtained.

Exercise 2:

Calculate the confusion matrix for the prediction of 0. That is, we assume that the positive class is 0 and all other classes are negative. Calculate also precission, recall and F1-Score.

NOTE: In utils you have a function that allows you to display the images you can use.