Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
Profesor: Dr. Pablo Alvarado Moya
CE-5506 Introducción al reconocimiento de patrones Entrega: **11 de octubre, 2017**
II Semestre 2017

Proyecto 1

1 Introducción

En este proyecto usted utilizará varios modelos de clasificación para resolver dos problemas: un problema *de juguete* en dos dimensiones que facilita la visualización de resultados, y otro problema de clasificación con datos reales, en donde se procura clasificar imágenes de dígitos escritos a mano, en particular la base de datos MNIST.

En la primera parte del proyecto usted deberá implementar la función de error asociada a una red neuronal, utilizar métodos de optimización para encontrar los mejores parámetros de la red neuronal usando un conjunto de entrenamiento, y luego visualizar el resultado de la clasificación. Esta parte del proyecto se realizará en GNU/Octave, utilizando funcionalidades del paquete `optim`. Usted deberá evaluar el desempeño de los optimizadores utilizados.

La segunda parte del proyecto utilizará la biblioteca Keras para Python, y con ella se implementarán varios tipos de clasificadores y se evaluará su desempeño con los datos datos de la base MNIST, que contiene alrededor de 70 000 imágenes de 28×28 píxeles ilustrando variantes de los dígitos del 0 al 9 escritos a mano (ver figura 1).

Figura 1: Ejemplos de los dígitos en la base de datos MNIST.

2 Objetivos

2.1 Objetivo general

Poner en acción los conceptos de clasificación revisados en clase hasta el momento y evaluar su desempeño.

2.2 Objetivos específicos

1. Implementar una función de error asociada a un problema de clasificación.
2. Utilizar optimizadores para minimizar la función de error considerando un conjunto de entrenamiento $\{(\underline{\mathbf{x}}^{(i)}, y^{(i)}) \mid i = 1, \ldots, m\}$.
3. Visualizar un espacio de entrada de dos dimensiones superpuesto con la salida de clasificación.
4. Programar y entrenar en la biblioteca Keras varios clasificadores vistos en clase y otros nuevos para clasificar la base MNIST.
5. Utilizar matrices de confusión para evaluar los resultados de clasificación

3 Procedimiento

3.1 Función de error

Una red neuronal se puede concebir como un grafo dirigido, en donde cada arista tiene un peso asociado y representa el producto de la entrada a esa arista con su peso, y en donde cada nodo realiza la suma de todas las salidas de las aristas y luego aplica una *función de activación* al resultado.

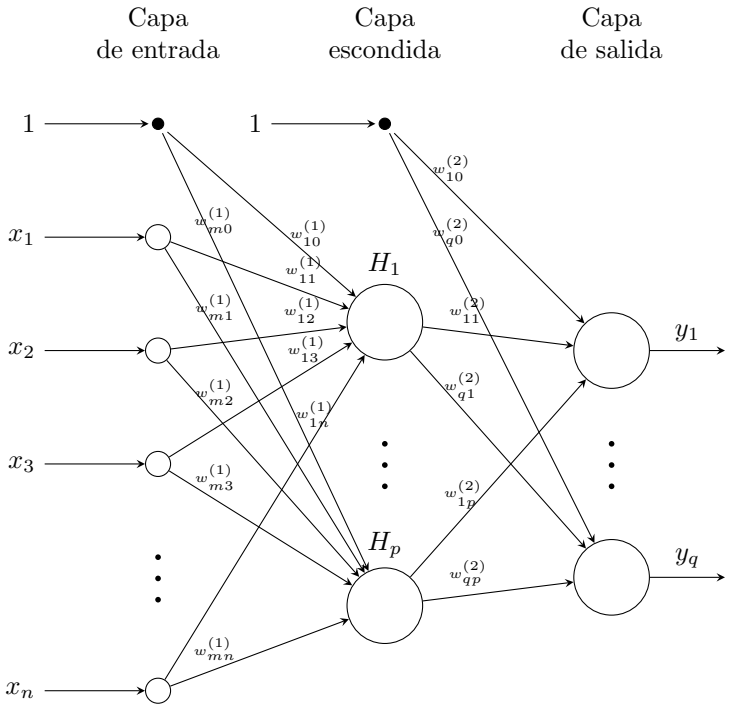

Figura 2: Red neuronal con una sola capa escondida

Por ejemplo, en la figura 2 si la neurona H_1 tiene función de activación g, entonces a su salida produce

$$y_{H_1} = g\left(\underline{\mathbf{w}}_{1\cdot}^T \begin{bmatrix} 1 \\ \underline{\mathbf{x}} \end{bmatrix}\right)$$

con el vector $\underline{\mathbf{w}}_{1\cdot}^T = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1n}^{(1)} \end{bmatrix}$.

Como función de activación g usualmente se utiliza la función logística o sigmoidal:

$$g(x) = \frac{1}{1+e^{-x}}$$

que tiende a cero para $x \to -\infty$ y tiende a uno para $x \to \infty$.

Si definimos que la aplicación de la función de activación a un vector equivale a aplicar la función de activación escalar a cada uno de los componentes, es decir:

$$g\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{bmatrix}$$

y si además agrupamos todos los pesos de la i-ésima capa en una matriz $\mathbf{W}^{(i)}$ (aquí n_i es el número de entradas a la i-ésima capa y p_i el número de neuronas de la i-ésima capa)

$$\mathbf{W}^{(i)} = \begin{bmatrix} w_{10}^{(i)} & w_{11}^{(i)} & w_{12}^{(i)} & \cdots & w_{1n_i}^{(i)} \\ w_{20}^{(i)} & w_{21}^{(i)} & w_{22}^{(i)} & \cdots & w_{2n_i}^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{p_i0}^{(i)} & w_{p_i1}^{(i)} & w_{p_i2}^{(i)} & \cdots & w_{p_in_i}^{(i)} \end{bmatrix}$$

entonces la salida $\underline{\mathbf{y}}^{(i)}$ de dicha capa ante la entrada $\underline{\mathbf{x}}^{(i)}$ es

$$\underline{\mathbf{y}}^{(i)} = g\left(\mathbf{W}^{(i)} \begin{bmatrix} 1 \\ \underline{\mathbf{x}}^{(i)} \end{bmatrix}\right) \tag{1}$$

Observe que la k-ésima fila de $\mathbf{W}^{(i)}$ contiene todos los pesos asociados a todas las aristas que *llegan* a la k-ésima neurona, mientras que la l-ésima columna de esa matriz contiene todos los pesos asociados a las aristas que *salen* de la l-ésima componente de la entrada.

De este modo, considerando que la salida de la capa i es la entrada de la capa $i+1$, la salida total predicha por la red neuronal en la figura 2 está dada por:

$$\underline{\hat{\mathbf{y}}}\left(\underline{\mathbf{x}}; \mathbf{W}^{(1)}, \mathbf{W}^{(2)}\right) = g\left(\mathbf{W}^{(2)} \begin{bmatrix} 1 \\ g\left(\mathbf{W}^{(1)} \begin{bmatrix} 1 \\ \underline{\mathbf{x}} \end{bmatrix}\right) \end{bmatrix}\right) \tag{2}$$

Dado un conjunto de m datos de entrenamiento $\{(\underline{\mathbf{x}}^{(j)}, y^{(j)}); j = 1 \ldots m\}$ podemos definir la función de error en términos de todos los pesos de todas las capas como aquella que

mide la diferencia entre lo que la red predice y lo que el conjunto de entrenamiento indica:

$$J(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) = \frac{1}{2} \sum_{j=1}^{M} \left(y^{(j)} - \underline{\hat{\mathbf{y}}}(\underline{\mathbf{x}}^{(j)}; \mathbf{W}^{(1)}, \mathbf{W}^{(2)}) \right)^2 \tag{3}$$

Queremos encontrar los pesos que producen el menor valor posible de J. Vimos en clase que a esto se le conoce como el problema de mínimos cuadrados ordinarios. Si este problema se resuelve por medio de descenso de gradiente recibe el nombre de algoritmo de retropropagación de error.

Con el archivo `create_data.m` usted tiene a disposición una función con interfaz

```
function y=create_data(dim,numClasses,shape)

# usage [X,Y] = create_data(numSamples,numClasses=3)
#
# This function creates random examples in the two-dimensional
# space from -1 to 1 in each coordinate.  The output Y is arranged
# in numClases outputs, such that they can be used as outputs of
# artificial neurons (or units) directly.
#
# Inputs:
#   numSamples: total number of samples in the training set
#   numClasses: total number of classes in the training set
#   shape: distribution of the samples in the input space:
#          'radial' rings of clases,
#          'pie' angular pieces for each class
#          'vertical' vertical bands
#          'horizontal' horizontal bands
#
# Outputs:
#   X: Design matrix, with only the two coordinates on each row
#      (no 1 prepended).  Its size is numSamples x 2
#   Y: Corresponding class to each training sample.  Its size is
#      numSamples x numClases

# ...

endfunction;
```

para crear conjuntos de datos en diferentes distribuciones (ver figura 3). Adicionalmente a la matriz de datos de entrada X, la matriz Y contendrá los valores deseados en cada neurona de salida: la primera columna corresponde a las salidas de la primera neurona, la segunda columna corresponde a las salidas deseadas de la segunda neurona, y así

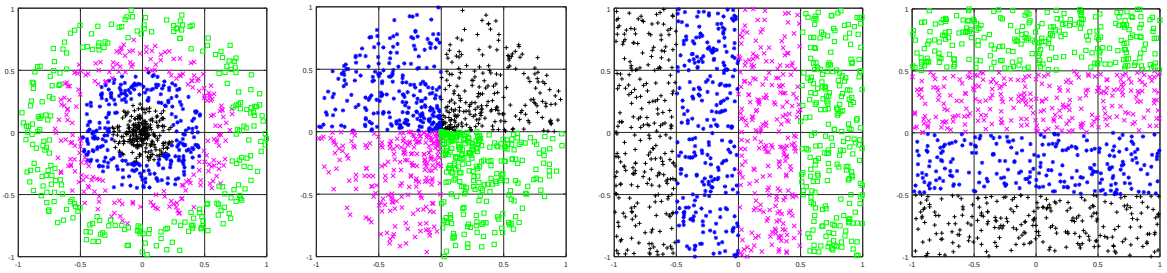

Figura 3: Distribuciones de datos de entrenamiento para primera parte del proyecto: (a) radial, (b) pie, (c) bandas verticales, (d) bandas horizontales. Los ejemplos usan 1000 datos con 4 clases.

sucesivamente. La salida de una neurona debe ser "1" si los datos de entrada corresponden a la clase que dicha neurona representa, o "0" en otro caso. La función `plot_data.m` le permite visualizar los datos generados, tal y como los muestra la figura 3.

En este proyecto usaremos por motivos de visualización únicamente tres clases a la salida, y un espacio de entrada de dos dimensiones.

1. Programe una función predictora que implemente (2) con la siguiente interfaz:

```
function y=predict(W1,W2,X)

# usage predict(W1,W2,X)
#
# This function propagates the input X on the neural network to
# predict the output vector y, given the weight matrices W1 and W2 for
# a two-layered artificial neural network.
#
# W1: weights matrix between input and hidden layer
# W2: weights matrix between the hidden and the output layer
# X:  Input vector, extended at its end with a 1

# PONGA SU CODIGO AQUÍ
endfunction;
```

   Usted debe implementar esta función de forma que si el usuario entrega varias entradas como filas de una matrix `X` de tamaño $m \times 2$, entonces produzca la salida `y` de $m \times$ filas$(\mathbf{W}^{(2)})$, es decir, el usuario puede indicar varias entradas, cada una en una fila de `X`, y la función debe entonces producir un vector con la salida correspondiente a cada fila de `X`.

   Note que para resolver este punto usted debe implementar la función sigmoide de forma apropiada, y considerar las entradas de sesgo en cada capa (marcadas como entrada '1' en la figura 2).

2. Programe una función denominada `target` con la siguiente interfaz:

```
function y=target(W1,W2,X,Y)

# usage target(W1,W2,X,Y)
#
# This function evaluates the sum of squares error for the
# training set X,Y given the weight matrices W1 and W2 for
# a two-layered artificial neural network.
#
# W1: weights matrix between input and hidden layer
# W2: weights matrix between the hidden and the output layer
# X:  training set holding on the rows the input data, plus a final column
#     equal to 1
# Y:  labels of the training set

# PONGA SU CODIGO AQUÍ
endfunction;
```

   Esta función debe evaluar (3). Observe que usted puede utilizar la función implementada en el punto anterior.

3. Programe ahora una función que calcule el gradiente de (3) con respecto a todos los pesos, evaluado sobre $\mathbf{W}^{(1)}$ y $\mathbf{W}^{(2)}$. Para ello puede utilizar métodos de

diferenciación numérica, para derivar con respecto a cada peso, o puede calcular analíticamente qué valor debe tener el gradiente, lo que se obtiene con relativa facilidad con la regla de la cadena y conociendo que para la función logística se sabe que $g'(x) = g(x)(1 - g(x))$:

```
function [gW1,gW2]=gradtarget(W1,W2,X,Y)

# usage gradtarget(W1,W2,X,Y)
#
# This function evaluates the gradient of the target function on W1 and W2.
#
# W1: weights matrix between input and hidden layer
# W2: weights matrix between the hidden and the output layer
# X:  training set holding on the rows the input data, plus a final column
#     equal to 1
# Y:  labels of the training set

# PONGA SU CODIGO AQUÍ
endfunction;
```

En el Internet puede encontrar bastantes fuentes de cómo calcular este gradiente analíticamente bajo el nombre *error backpropagation*.

Las dos matrices de salida de esta función tienen los mismos tamaños que las dos matrices de pesos a la entrada, y cada elemento de cada matriz contiene el valor de la derivada de la función `target` con respecto al peso correspondiente:

$$\texttt{gW1} = \begin{bmatrix} \frac{\partial J}{\partial w_{10}^{(1)}} & \frac{\partial J}{\partial w_{11}^{(1)}} & \frac{\partial J}{\partial w_{12}^{(1)}} & \cdots & \frac{\partial J}{\partial w_{1n}^{(1)}} \\ \frac{\partial J}{\partial w_{20}^{(1)}} & \frac{\partial J}{\partial w_{21}^{(1)}} & \frac{\partial J}{\partial w_{22}^{(1)}} & \cdots & \frac{\partial J}{\partial w_{2n}^{(1)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial w_{p0}^{(1)}} & \frac{\partial J}{\partial w_{p1}^{(1)}} & \frac{\partial J}{\partial w_{p2}^{(1)}} & \cdots & \frac{\partial J}{\partial w_{pn}^{(1)}} \end{bmatrix}$$

3.2 Entrenamiento de la red

Usted utilizará dos estrategias para entrenar su red.

Para poder simplificar la manipulación de los datos, usted debe empaquetar todos los pesos en $\mathbf{W}^{(1)}$ y $\mathbf{W}^{(2)}$ en un solo vector $\underline{\boldsymbol{\omega}}$ en una función denominada `packweights`. También implemente una función `unpackweights` que desempaquete el vector en las dos matrices correspondientes.

1. Primero usted entrenará su red utilizando el descenso de gradiente. Para ello implemente un programa iterativo que actualice los pesos:

   $$\underline{\boldsymbol{\omega}} \leftarrow \underline{\boldsymbol{\omega}} - \lambda \nabla_{\underline{\boldsymbol{\omega}}} J(\underline{\boldsymbol{\omega}})$$

   Usted debe decidir cuándo terminar las iteraciones.

   Pruebe varios valores de λ y evalúe cuántas iteraciones se requieren para converger.

2. El segundo método que va a utilizar es el de gradientes conjugados. Revise cómo utilizar el método `cg_min` en el paquete `optim` de octave.

3. Cuente el número de evaluaciones de la función de error para este método también.

3.3 Visualización de los resultados

Para visualizar los resultados de la clasificación usted va a generar una imagen que representa con colores la clasificación en cada punto de dicha imagen. La imagen es cuadrada y representa el rango vertical y horizontal de los datos de entrenamiento, es decir, de -1 a 1 en ambas direcciones.

En GNU/Octave las imagenes son matrices $M \times N \times 3$ con M el número de filas, N el número de columnas y los tres canales representan las componentes rojo, verde y azul, con valores entre 0 a 1. El canal rojo representará la salida de la primera neurona representando a la clase 1, el canal verde representará la salida de la segunda neurona, representando la clase 2 y el canal azul representará la salida de la tercera neurona, representando la clase 3.

Note que como las salidas de las neuronas pueden ser máximo 1 y mínimo 0, entonces pueden utilizarse directamente como los valores de los canales de la imagen. Usted debe realizar el mapeo entre las coordenadas de la imagen y el espacio entrada de la red neuronal (de -1 a 1). Su imagen deberá tener un tamaño que permita visibilizar con detalle el resultado de la clasificación, como por ejemplo, 512×512. Usted debe recorrer entonces todos los píxeles de la imagen, calcular los valores de entrada correspondinetes, clasificar dicha entrada y utilizar los valores a las neuronas de salida como los valores de los canales rojo, verde y azul del pixel correspondiente. Con la función `imshow` muestre su imagen y sobrepóngale los puntos de entrada.

Usted deberá además salvar en un archivo sus matrices de datos para utilizarlas en los siguientes puntos. Debe investigar qué método puede utilizar para salvar estas matrices de modo que se simplifique su lectura desde Python.

3.4 Evaluación de los resultados

1. Para evaluar el desempeño de su algoritmo, genere otro conjunto de datos (distinto del conjunto de entrenamiento) con la misma distribución de puntos que los datos de entrenamiento.

2. Investigue qué es una matriz de confusión.

3. Utilice su función de predicción para predecir cada punto en el conjunto de prueba y cree la matriz de confusión correspondiente, donde las filas deben tener las clases reales y las columnas las clases predichas.

4. Investigue qué métricas de evaluación de clasificación se pueden derivar de la matriz de confusión, en particular la sensitividad y la precisión, y calcúlelas para sus datos.

4 Python, Scikit-Learn y Keras

En esta segunda parte se utilizará la biblioteca de Python Scikit-Learn para clasificar los datos generados en Octave en los puntos anteriores, y la biblioteca Keras, para clasificar un problema más complejo.

4.1 Scikit Learn y SVM

1. Instale la biblioteca Scikit-Learn con todas sus dependencias en Python. Para ello puede utilizar `pip`, Anaconda, o cualquier otro manejador de paquetes de su elección.
2. Busque en el Internet algún ejemplo sencillo de uso de las máquinas de soporte vectorial con Scikit-Learn. Investigue cómo puede importar los datos utilizados en GNU/Octave en la sección anterior para ser utilizados por Scikit-Learn
3. Repita el problema de clasificación de los puntos de prueba pero utilizando las SVM con distintos tipos de kernel. Calcule para cada caso la matriz de confusión y los valores derivados de esta.
4. Analice sus resultados.

4.2 Keras y Deep Learning

1. Instale la biblioteca Keras con todas sus dependencias en su versión de Python. Para ello puede utilizar `pip`, Anaconda, o cualquier otro manejador de paquetes moderno.
2. Busque en el Internet algún ejemplo sencillo de red neuronal (profunda) para clasificar la base de datos MNIST(dígitos escritos a mano). Usted debe poner en marcha el ejemplo que encuentre y modificarlo para:
   2.1. Clasificar alguna imagen arbitraria de dígitos. Usted puede introducir el nombre de un archivo, o puede diseñar una interfaz sencilla en Python que le permita al usuario dibujar el dígito a ser reconocido, y mostrar los resultados.
   2.2. Calcular la matriz de confusión para un conjunto de datos de prueba
   2.3. Utilizar el método *evaluate* del modelo para indicar diversas estadísticas.

   Observe que la base de datos MNISTya trae datos para entrenamiento y datos para prueba. Asegúrse de usarlos correctamente.
3. Utilice los datos de entrenamiento de MNISTpara entrenar una SVM y realice las pruebas de clasificación con los datos de prueba.
4. Analice los resultados y compárelos con la red neuronal.

5 Entregables

El código fuente y el artículo científico deben ser entregados según lo estipulado en el programa del curso.

Incluya un archivo de texto README con las instrucciones para la ejecución de los programas.