Proyecto 6

Ruben Cuadra

Dr. Victor de la Cuva

October 30, 2017

Red Neuronal BPN

Entrenamiento

El código consta de una librería con 21 funciones que el objetivo final es entrenar una red neuronal de múltiples capas para que logre clasificar imágenes de cuales quiera dimesiones en cualquier cantidad de etiquetas.

Para lograr el entrenamiento de la red se requieren 3 pasos:

1. Obtener datos de entrenamiento:

Debemos invocar la función *getDataFromFile*, la función recibe como parámetro el nombre de un archivo (o su *path*), debe ser un archivo de texto donde cada renglón representara un ejemplo de entrenamiento.

Cada renglón posee N valores, los cuales están separados por espacios, el ultimo valor será la etiqueta correspondiente para ese ejemplo. Ej.

Las fotos de arriba son parte del archivo de entrenamiento en el ejemplo, en este caso cada linea representaría una imagen de 20x20 con un número escrito a mano, entonces la linea posee 400 + 1 números separados por espacios, los primeros 400 números representan un color en la escala de grises donde -1 es blanco y 1 es negro, mientras que el ultimo será la etiqueta relacionada a la foto (cuadrado rojo), en nuestro archivo, el primer renglón tiene una etiqueta de 0, diciendo que esa foto representa un 0 escrito a mano.

La función lee el archivo y nos regresa todos los ejemplos en una matrix de Xs (mxn) y todas las etiquetas en un vector de Ys (mx1). m = numEjemplos, n = numVars.

Para este paso ejecutamos la linea:

```
xExamples,tags = getDataFromFile("digitos.txt")
```

2. Definir el número de capas a usar en la red

Cada capa proviene de la clase *NNLayer* y recibe 2 parámetros, el numero de neuronas en ella y el tipo de activación que posee la capa (Todas las neuronas tendrán la misma activación), la activación es un *enum* que proviene de la clase *activaciones* debemos crear un arreglo básico con múltiples objetos de esta clase, donde el elemento pero representara la capa 1 y así consecutivamente.

```
l    = [
        NNLayer(25,activaciones.LINEAL),
        NNLayer(5,activaciones.SIGMOIDAL)
    ]
```

3. Entrenar la red

Consta de mandar a llamar la función *entrenaRN* la función recibe 2 parámetros obligatorios y opcionalmente se pueden modificar las demas, se debe pasar los ejemplosX y las etiquetas obtenidas en el **paso 1**, los parámetros opcionales son:

hidden_layers : Recibe un arreglo de *NNLayer*, basándonos en el **paso 2** podemos pasarle la variable *l* . Default es un arreglo vacío

iters: Maximo de iteraciones antes de finalizar el aprendizaje, default es 1000

e : Error Maximo permitido, mientras menor sea mejor serán las predicciones de esa red, tardara mucho mas en ser entrenada con menores e. Default 0.001

alpha : razón de aprendizaje, mientras mayor sea es mas probable es que oscile y que no llegue a una solución, default 0.001

activacionFinal : recibe un *enum* de la clase *activaciones* y representa la función de activación en la capa Final, es importante mencionar que la capa final tendrá K neuronas, donde K es el numero de diferentes etiquetas en el vector etiquetas pasado a esta misma función, default *a*ctivaciones.SIGMOIDAL

Una vez ejecutada la función el código iterara hasta que el error converge (cuando el *costo* sea menor que *e*) o hasta que se acaban las iteraciones, lo que suceda primero. La función devuelve un diccionario p , el diccionario siempre contiene las llaves:

W1: Matriz que representa los pesos para cada neurona de la capa 1

b1 : Vector que representa los pesos para cada neurona en la capa 1

l : Numero de capas en total

Cuando la función recibe un arreglo con N capas vía *hidden_layers* el diccionario tendrá W1,W2,W… hasta WN+1, donde WN+1 representa la capa de salida

4. **Guardar los pesos aprendidos**

Para guardar los resultados para usarlos en una predicción a futuro usamos el comando `np.save('network.npy',p)` que viene de la librería de numpy y nos genera un archivo "network.npy" con los valores de nuestra red.

Predicción

Para hacer las predicciones necesitamos un archivo "network.npy", el repositorio contiene una red ya entrenada llamada "network038.npy" la cual se podria usar para probar, los pasos serian:

1. **Obtener pesos desde el archivo "network.npy"**

Usamos la linea `W,b = getWeightsFromFile("network038.npy")`

2. **Obtener vector de predicciones**

   _Y = prediceRNYaEntrenada(xExamples,W,b)

   La función **prediceRNYaEntrenada** recibe como parámetro 1 o varios ejemplos X, un arreglo W, donde el elemento W[0] representa W1, la matriz de pesos para la capa 1 y un arreglo b, donde el elemento b[0] representa b1 que es un vector para la capa 1, estas variables se obtienen en el **paso 1** de la **predicción.** La función nos devuelve un vector _Y, donde _Y[0] es la predicción para el ejemplo X[0]. (En este ejemplo xExamples son TODOS los ejemplos usados para entrenar la red, se obtienen en el **paso 1 de entrenamiento**)

3. **Imprimir error**

   error = getErrorPercentage(tags,_Y)

   la función **getErrorPercentage** recibe el vector _Y calculado en el paso anterior y un vector tags que representa los valores reales, en este caso usamos tags que se obtuvo en el **paso 1 de entrenamiento** debido a que la predicción se uso con todas las X de ese mismo paso.

Extras

Si leemos los ejemplos con el paso 1 de predicción

```
xExamples,tags = getDataFromFile(“digitos.txt")
```

Podemos obtener cualquier elemento de xExamples y tags

```
example = 3014 #Menor a 5000
x,tag = xExamples[example],[tags[example]]
```

Y podemos graficar ese numero

```
graficarNumero( x )
```

Si tenemos duda de su valor real podemos imprimir la tag correspondiente al numero

```
print tag
```

Abajo se ve un ejemplo del numero

Proyecto 6