Aprendizaje en Redes Neurales

José Castro

1 Introducción

En este ejercicio, usted implementará el algoritmo de retropropagación y lo aplicará a la tarea de reconocimiento de dígitos escritos a mano.

2 Archivos en esta tarea

Los archivos incluidos en esta tarea son:

- `ej4.m` Un script de Octave que le ayuda con la primera parte de la tarea.
- `ej3data1.txt` Set de entrenamiento de dígitos escritos a mano.
- `ej4pesos.mat` Pesos iniciales para el ejercicio de redes neurales.
- `despliegueDatos.m` Función para visualizar el set de datos.
- `fmincg.m` Función para minimizar (similar a fminunc).
- `sigmoide.m` Función sigmoide.
- `calculeGradienteNumerico.m` Calcula numéricamente el gradiete de Θ.
- `reviseGradienteRN.m` Función que ayuda a revisar los gradientes.
- `debutInicialicePesos.m` Función para inicializar los pesos (parámetros) de la red.
- `preciccion.m` Función de predicción de la red neural.

- [*] `gradienteSigmoide.m` Calcula el gradiente de la función sigmoide.
- [*] `inicializarPesosRand.m` Iinicializar los pesos aleatoriamente.
- [*] `funcionDeCostoRN.m` Función de costo de la red neural.

* indica un archivo que usted debe completar.

A través de este ejercicio usted estará utilizando los scripts `ex4.m`. Este prepara los datos para los problemas y llaman a las funciones que usted escribirá, no necesita modificarlo. Solo se requiere que modifique las funciones en otros archivos (los que tienen *).

3 Redes Neurales

En el ejercicio anterior usted implementó la propagación hacia adelante de la red. En este ejercicio usted utilizará retropropagación para *aprender* los parámetros de la red neural.

El script `ej4.m` le ayudará a seguir los pasos de este ejercicio.

3.1 Visualizando los Datos

En la promera parte de `ej4.m` se cargan y se despliegan los datos llamando a la función `despliegueDatos`.

Este es el mismo conjunto de datos que el ejercicio anterior. `ex3data1.mat` contiene 5000 ejemplos de entrenamiento de dígitos escritos a mano[1]

La extensión `.mat` significa que los datos han sido almacenados en el formato nativo para matrices de Octave/Matlab, y no en un archivo de texto (ASCII) tal como un archivo csv (Comma Separated Values). Estas matrices se pueden leer directamente en Octave con el comando `load`. Una vez cargadas, las matrices apareceran en la memoria. La matriz ya tendrá nombre, así que no necesita asignarles nombre.

```
>> load('ex3data1.mat');
```

Hay 5000 ejemplos de entrenamiento en `ex3data1.mat` donde cada ejemplo de entrenamiento es una imagen en escala de gris de 20 pixeles por 20

[1]Este es un subconjuto de la base de datos MNIST de dígitos escritos a mano (`http://yan.lecun.com/exdb/mnist/`)

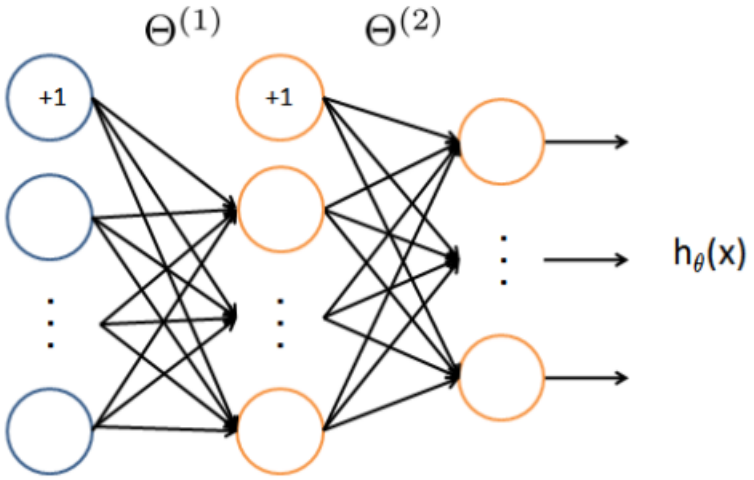
pixeles. Cada pixel es representado por un número de punto flotante que indica la intensidad de la escala de gris en esa ubicación. La grilla de pixeles de 20 por 20 viene "desenrollada" en un vector 400-dimensional. Cada ejemplo de entrenamiento se convierte en una fila de la matriz X. Esto nos da una matriz X de tamaño 5000 por 400 donde cada fila es un ejemplo para un dítigo escrito a mano.

$$\begin{bmatrix} — \left(x^{(1)}\right)^T — \\ — \left(x^{(2)}\right)^T — \\ \vdots \\ — \left(x^{(m)}\right)^T — \end{bmatrix}$$

Puede empezar visualizando los datos. En la parte 1 de `ex3.m` el código aleatoreamente selecciona 100 filas de `X` y los pasa por la función de `despliegueDatos`. Esta función mapea cada fila a una imagen en escala de gris de 20x20 pixeles y las despliega juntas. Se provee la función `despliegueDatos` que usted puede examinar. Después de correr este paso debería ver una figura como la siguiente:

3.1.1 Representación del modelo

Nuestra red neural es la siguiente:



$$a^{(1)} = x \qquad z^{(2)} = \Theta^{(1)} a^{(1)} \qquad z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$(+a_0^{(1)}) \qquad a^{(2)} = g(z^{(2)}) \qquad a^{(3)} = g(z^{(3)}) = h_\theta(x)$$
$$(+a_0^{(2)})$$

Tiene tres capas, una capa de entrada, una capa oculta, y una capa de salida. Recuerde que nuestra entrada son pixeles de imágenes de dígitos. Las imágenes son de tamaño 20×20, esto nos da una entrada de tamaño 400 (sin contar la neurona 0 de sesgo que siempre emite un $+1$). Los valores de entrenamiento se cargan en las variables `X` y `y` por el script `ej4.m`.

Se le prove un set de parámetros $(\Theta^{(1)}, \Theta^{(2)})$ que ya se encuentran entrenados. Estos están almacenados en `ej4pesos.mat` y serán cargados por `ej4.m` en las variables `Theta1` y `Theta2`. Los parámetros tienen la dimensión necesaria para una red con 25 neuronas en la segunda capa y 10 en la tercera (correspondiente a las 10 clases de los dígitos).

3.2 Propagación hacia adelante y función de costo

Ahora implemente la función de costo y el gradiente de la red neural. Primero complete el código de `funcionDeCostoRN.m` para que retorne el costo.

Tome en cuenta que la función de costo de la red neural (sin regular-

ización) es

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

Donde $h_\theta(x^{(i)})$ es calculado como se muestra en la figura de la red neural anterior. y $K = 10$ es el número de etiquetas posibles. Note que la activación de la útima capa es igual a la hipótesis, esti es $h_\theta(x^{(i)})_k = a_k^{(3)}$. También recuerde que se necesita recodificar los valores de y como vectores unitarios tal que

$$y = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \quad \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \quad \ldots \quad \text{o bien es} \quad \begin{bmatrix}\\0\\0\\\vdots\\1\end{bmatrix}$$

Usted debe implementar el cómputo de propagación hacia adelante que calcula $h_\theta(x^{(i)})$ para cada ejemplo i y sumar el costo sobre todos los ejemplos. **Su código debe funcionar para un set de datos de cualquier tamaño con cualquier número de etiquetas** (puede asumir que hay por lo menos $K \geq 3$ etiquetas).

Nota de implementación La matrix `X` contiene los ejemplos en las filas (por ejemplo `X(i,:)'` es el $i^{\text{ésimo}}$ ejemplo de entrenamiento, expresado como un vector de dimensión $n \times 1$). Cuando haga el código de `funcionDeCostoRN.m` necesitará agregar una columna de 1's a la matriz. Las matrices `Theta1` y `Theta2` contienen los parámetros de las neuronas por filas. Específicamente la primer fila de `Theta1` corresponde a la primer neurona oculta. En Octave cuando calcule $z^{(2)} = \Theta^{(1)}a^{(1)}$, asegúrese de que indexa `X` correctamente (y la transpone si es necesario) tal que pueda obtener $a^{(l)}$ como un vector columna. Puede utilizar un ciclo `for` para calcular el costo de los ejemplos.

Una vez hecha `funcionDeCostoRN` debería poder observar un costo cercano a 0.287629.

3.3 Función de costo regularizado

La función de costo regularizada esta dada por la ecuación:

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] \\ &+ \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}\left(\Theta_{j,k}^{(1)}\right)^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}\left(\Theta_{j,k}^{(2)}\right)^2\right] \end{aligned}$$

Puede asumir que la red neural tiene solo tres capas – una capa de entrada, una capa oculta, y una capa de salida. Sin embargo, su código debe funcionar para cualquier cantidad de neuronas de entrada, neuronas en la capa oculta, y neuronas de salida. Aunque se ha puesto la ecuación de arriba con los valores del ejercicio actual, **su código debería de funcionar para cualquier tamaño de $\Theta^{(1)}$ y de $\Theta^{(2)}$**.

Note que no debe regularizar los valores correspondientes a la neurona de sesgo. Para las matrices `Theta1` y `Theta2` esto corresponde a la primera columna de la matriz.

Debe agregar ahora regularización a `funcionDeCostoRN.m`, una vez hecho esto para los parámetros cargados de `Theta1` y `Theta2` con un $\lambda = 1$, debe ver un costo cercato a 0.383770.

4 Retropropagación

En esta parte del ejercicio, debe implementar el algoritmo de retropropagación para calcular el gradiente de la función de costo. Usted necesita implementar `funcionDeCostoRN.m` tal que retorne un valor aproximado para el gradiente `grad`. Una vez calculado el gradiente, usted será capaz de entrenar la red minimizando la función de costo $J(\Theta)$ utilizando un optimizador avanzado como `fmincg` (minimización por gradiente conjugado). Su primera implementación calculará los gradientes para la versión no regularizada de la red neural. Una vez que haya verificado que su cómputo sin regularización es correcto, implementará el gradiente con regularización.

4.1 Gradiente de la función Sigmoide

Para ayudarle a empezar con esta parte del ejercicio, primero debe implementar el gradiente de la función sigmoide, el cual puede ser calculado como:

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1 - g(z))$$

Donde

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

Cuando termine, intente probarla con varios valores llamando a `gradienteSigmoide(z)` en la linea de comandos de Octave. Para valores grandes, tanto positivos como negativos, el gradiente debe ser cercano a 0. Cuando `z = 0`, el gradiente debe ser igual a 0.25. Su código debería trabajar bien tanto con escalares como con vectores y matrices. Para una matriz debe calcular el gradiente de cada elemento.

4.2 Inicialización aleatoria

Cuando se entrena una red neural es importante inicializar los parámetros (pesos) para romper la simetr´ia. Una estrategia efectiva para inicializar los valores es seleccionar valores para $\Theta^{(l)}$ uniformemente en el rango $[-\epsilon_{init}, \epsilon_{init}]$. Debe utilizar en este caso $\epsilon_{init} = 0.12$. [2]. este rango asegura que los parámetros se mantienen pequeños y hace el aprendizaje más eficiente.

Su trabajo es completar `inicializarPesos.m` para inicializar los pesos Θ, debe midificar el archivo y agregar el código:

```
epsilon_init = 0.12;
W = rand(L_out, 1+L_in) * 2 * epsilon_init - epsilon_init;
```

4.3 Retropropagación

Ahora debe implementar el algoritmo de retropropagación. Recuerde que la intuición es la siguiente. Dada un ejemplo de entrenamiento $(x^{(t)}, y^{(t)})$, primero corremos un paso "hacia adelnate" para calcular las activaciones en la red, incluyendo las de la hipótesis de salida $h_\Theta(x)$. Después para cada

[2]Una estrategia efectiva para escoger a ϵ_{init} es basarlos en la cantidad de unidades en la red. Una buena eleccion es $\epsilon_{init} = \frac{\sqrt{6}}{\sqrt{L_{in}+L_{out}}}$.

nodo j de la capa l, se debe calcular el término de "error" $\delta_j^{(l)}$ que mide que tanto es responsable ese nodo por errores en la salida.

Para un nodo de salida podemos medir directamente la diferencia entre el valor objetivo y el obtenido y utilizar esto para definir $\delta_j^{(3)}$ (la red tiene 3 capas). Para unidades ocultas, se debe calcular un promedio ponderado de los errores de la siguiente capa $(l+1)$.

En detalle, aquí esta el algoritmo de retropropagación. Concretamente debe implementar un loop `for t = 1:m` y llevar a cabo los pasos 1-4 siguientes, con la $t^{\text{ésima}}$ iteración calculando los valores para el t^{esimo} ejemplo $(x^{(t)}, y^{(t)})$. La etapa 5 divide los gradientes acumulados por m para obtener el gradiente de la función de costo.

1. Ponga los valores de entrada $(a^{(1)})$ al t-ésimo ejemplo $x^{(t)}$. Haga propagación hacia adelante calculando las activaciones $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$ para las capas 2 y 3. Note que debe agregar el término `+1` para asegurarse que las activaciones para las capas $a^{(1)}$ y $a^{(2)}$ también incluyen la unidad de sesgo.

2. Para cada uidad de salida k de la tercer capa, haga

$$\delta_k^{(3)} = (a_k^{(3)} - y_k)$$

   Donde $y_k \in \{0, 1\}$ indica si el actual ejemplo de entrenamiento pertenece a la clase k $(y_k = 1)$, o pertenece a otra clase $(y_k = 0)$. Puede que le sirva utilizar arrays lógicos (explicados en la tarea anterior).

3. Pra la capa oculta $l = 2$ haga:

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

4. Acumule el gradiente de este ejemplo utilizando la siguiente fórmula. Note que debe eliminar o brincarse el $\delta_0^{(2)}$ (`delta_2 = delta_2(2:end)`)

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Obtener el gradiente (sin regularización) para la red neural dividiendo los valores entre m

$$\frac{\partial}{\partial \Theta_{ij}^{(i)}} J(\Theta) = D_{ij}^{(i} = \frac{1}{m}\Delta_{ij}^{(l)}$$

4.4 Revisión de Gradiente

En su red usted esta minimizando la función de costo $J(\Theta)$. Para revisar los gradientes de sus parámetros, puede imaginarse que "desenrolla" los parámetros Θ_1 y Θ_2 en un vector largo θ. Al hacerlo, puede pensar que la función de costo es $J(\theta)$ y que utiliza el siguiente procedimiento de revisión.

Suponga que tiene una funci´on $f_i(\theta)$ que supuestamente calcula $\frac{\partial}{\partial\theta_i}J(\theta)$; y usted quiere revisar si f_i calcula el valor correcto, Sea

$$\theta^{(i+)} = \theta + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix} \quad \text{y} \quad \theta^{(i-)} = \theta - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix}$$

Así $\theta^{(i+)}$ es igual a θ excepto que la iésima entrada ha sido incrementada por el valor de ϵ, de manera similar $\theta^{(i-)}$ es decrementada. Se puede verificar $f_i(\theta)$ revisando que:

$$f_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon}$$

El grado en que estos dos alores coincides depende en los detalles de J. Pero asumiendo en este caso que $\epsilon = 10^{-4}$, encontrará que estos valores deben coincidir en por lo menos 4 dígitos significativos.

4.5 Red Regularizada

Después de haber implementado la retropropagación. Se puede agregar la regularización. Debe agregar el término de regularización. Específicamente, se debe calcular

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)}$$

Para $j = 0$, y

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\delta}{m}\Theta_{ij}^{(l)}$$

Para $j \geq 1$. Note que no debe regularizar el término 0.

4.6 Aprendiendo los parámetros utilizando `fmincg`

Después de haber aprendido correctamente su entrenamiento debería reportar una precisión cercana al 95.3%. Puede obtener mejor entrenamiento variando la cantidad de iteraciones y el parámetro λ. Con los valores apropiados, es posible lograr que la red neural haga clasificación perfecta en estos datos.

5 Evaluación

- 45 pts – `funcionDeCostoLR.m`
- 5 pts – `gradienteSidmoide.m`
- 50 pts – `Retropropagación`