

Semana 5: Machine Learning

Diego Duane Aliaga Calderon



1. Función de coste y retropropagación.



Función de Coste

Primero definamos algunas variables que necesitaremos usar:

- L = número total de capas en la red
- s_l = número de unidades (sin contar la unidad de sesgo) en la capa l
- K = número de unidades / clases de salida

Recuerde que en las redes neuronales, podemos tener muchos nodos de salida. Denotamos $h_{\Theta}(x)_k$ como una hipótesis que da como resultado k^{th} salida. Nuestra función de coste para redes neuronales será una generalización de la que usamos para la regresión logística.



Función de Costo

Recuerde que la función de coste para la regresión logística regularizada fue:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$



Función de Coste

Para las redes neuronales, será un poco más complicado:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K [y_k^{(i)} \log ((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)}) \log (1 - (h_\Theta(x^{(i)}))_k)] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

Hemos agregado algunas sumas anidadas para dar cuenta de nuestros múltiples nodos de salida. En la primera parte de la ecuación, antes de los corchetes, tenemos una suma adicional anidada que recorre el número de nodos de salida.



Función de Coste

En la parte de regularización, después de los corchetes, debemos tener en cuenta múltiples matrices theta. El número de columnas en nuestra matriz theta actual es igual al número de nodos en nuestra capa actual (incluida la unidad de sesgo). El número de filas en nuestra matriz theta actual es igual al número de nodos en la siguiente capa (excluyendo la unidad de sesgo). Como antes con la regresión logística, cuadramos cada término.

Nota:

- la suma doble simplemente suma los coste de regresión logística calculados para cada celda en la capa de salida.
- la suma triple simplemente suma los cuadrados de todos los individuos Θ_S en toda la red.
- la i en la suma triple no se refiere al ejemplo de entrenamiento i .



Algoritmo de retropropagación

"Backpropagation"

es la terminología de la red neuronal para minimizar nuestra función de coste, al igual que lo que estábamos haciendo con el descenso de gradiente en regresión logística y lineal. Nuestro objetivo es calcular:

$$\min_{\Theta} J(\Theta)$$



Algoritmo de retropropagación

Es decir, queremos minimizar nuestra función de coste J usando un conjunto óptimo de parámetros en θ .

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$$



Algoritmo de retropropagación

Back propagation Algorithm- Algoritmo de propagación hacia atrás:

-Conjunto de entrenamiento dado $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$

Conjunto $\Delta_{i,j}^{(l)} := 0$ para todo (l,i,j) , (por lo tanto terminas teniendo una matriz llena de ceros)

Por ejemplo de entrenamiento $t=1$ a m :

1- Conjunto $a^{(1)} := x^{(t)}$

2- Realizar propagación hacia adelante para calcular $a^{(l)}$ para $l=2,3,\dots,L$.



Algoritmo de retropropagación

3- Utilizando $y^{(t)}$, calcular $\delta^{(L)} = a^{(L)} - y^{(t)}$

Donde L es nuestro número total de capas y $a^{(L)}$ es el vector de salidas de las unidades de activación para la última capa. Entonces nuestro "valores de error" para la última capa son simplemente las diferencias de nuestros resultados reales en la última capa y las salidas correctas en y. Para obtener los valores delta de las capas antes de la última capa, podemos usar una ecuación que nos hace retroceder de derecha a izquierda:



Algoritmo de retropropagación

4- Calcular $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ utilizando

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}). * a^{(l)}. * (1 - a^{(l)})$$

Los valores delta de la capa l se calculan multiplicando los valores delta en la siguiente capa con la matriz theta de la capa l. Luego, en cuanto a elementos, multiplicamos eso con una función llamada g' , o g-prime, que es la derivada de la función de activación g evaluada con los valores de entrada dados por $z^{(l)}$.

El g-prime los términos derivados también se pueden escribir como:

$$g'(z^{(l)}) = a^{(l)}. * (1 - a^{(l)})$$



Algoritmo de retropropagación

5- $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ o con vectorización,
 $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

Por lo tanto, actualizamos nuestro nuevo Δ matriz.

- $D_{i,j}^{(l)} := \frac{1}{m}(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}), \text{ si } j \neq 0.$
- $D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)} \text{ si } j = 0.$

La matriz capital-delta D se usa como un "acumulador" para sumar nuestros valores a medida que avanzamos y eventualmente calcular nuestra derivada parcial. Así obtenemos $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$



Intuición de propagación hacia atrás

Backpropagation Intuition- Intuición de propagación hacia atrás:

Recuerde que la función de coste para una red neuronal es:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K [y_k^{(i)} \log ((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)}) \log (1 - (h_\Theta(x^{(i)}))_k)] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

Si consideramos la clasificación simple no multiclasa ($k = 1$) e ignorar la regularización, el coste se calcula con:

$$cost(t) = y^{(t)} \log (h_\Theta(x^{(t)})) + (1 - y^{(t)}) \log (1 - h_\Theta(x^{(t)}))$$



Intuición de propagación hacia atrás

Intuitivamente, $\delta_j^{(l)}$ es el "error" para $a_j^{(l)}$ (unidad j en la capa l). Más formalmente, los valores delta son en realidad la derivada de la función de costo:

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(t)$$

Recuerde que nuestra derivada es la pendiente de una línea tangente a la función de coste, así que cuanto más empinada es la pendiente, más incorrectos somos. Consideremos la siguiente red neuronal a continuación y veamos cómo podríamos calcular algunos $\delta_j^{(l)}$:



Intuición de propagación hacia atrás

Para calcular $\delta_2^{(2)}$, multiplicamos los pesos $\Theta_{12}^{(2)}$ y $\Theta_{22}^{(2)}$ por sus respectivos δ valores encontrados a la derecha de cada borde. Entonces obtenemos $\delta_2^{(2)} = \Theta_{12}^{(2)} * \delta_1^{(3)} + \Theta_{22}^{(2)} * \delta_2^{(3)}$. Para calcular todos los posibles $\delta_j^{(l)}$, podríamos comenzar desde la derecha de nuestro diagrama. Podemos pensar en nuestros bordes como nuestro Θ_{ij} . Yendo de derecha a izquierda, para calcular el valor de $\delta_j^{(l)}$, simplemente puede tomar la suma total de cada peso multiplicado por δ del que proviene. Por lo tanto, otro ejemplo sería $\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$.



2. Retropropagación en la práctica.



Comprobación de gradiente

La verificación de gradiente asegurará que nuestra retropropagación funcione según lo previsto. Podemos aproximar la derivada de nuestra función de coste con:

$$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2\epsilon}$$

Con múltiples matrices theta, podemos aproximar la derivada **con respecto a Θ_j** , como sigue:

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$



Comprobación de gradiente

Un pequeño valor para ϵ (epsilon) como $\epsilon = 10^{-4}$, garantiza que las matemáticas funcionen correctamente. Si el valor de ϵ es demasiado pequeño, podemos terminar con problemas numéricos.



Comprobación de gradiente

Por lo tanto, solo estamos sumando o restando épsilon a la matriz Θ_j . En octave podemos hacerlo de la siguiente manera:

```
1 epsilon = 1e-4;
2 for i = 1:n,
3   thetaPlus = theta;
4   thetaPlus(i) += epsilon;
5   thetaMinus = theta;
6   thetaMinus(i) -= epsilon;
7   gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
8 end;
```



Inicialización Aleatoria

Iniciar todos los pesos theta a cero no funciona con redes neuronales. Cuando volvemos a propagar, todos los nodos se actualizarán al mismo valor repetidamente. En cambio, podemos inicializar aleatoriamente nuestros pesos para nuestras matrices Θ utilizando lo siguiente:

Iniciar cada $\Theta_{i,j}^{(l)}$ a un valor aleatorio en $[-\epsilon, \epsilon]$.



Poniendo todo junto

Primero, elija una arquitectura de red; elija el diseño de su red neuronal, incluidas cuántas unidades ocultas en cada capa y cuántas capas en total desea tener.

- Número de unidades de entrada = dimensión de características $x^{(i)}$
- Número de unidades de salida
- Número de unidades ocultas por capa = generalmente (debe equilibrarse con el coste de cálculo a medida que aumenta con más unidades ocultas).
- Valores predeterminados: 1 capa oculta. Si tiene más de 1 capa oculta, entonces se recomienda que tenga la misma cantidad de unidades en cada capa oculta.



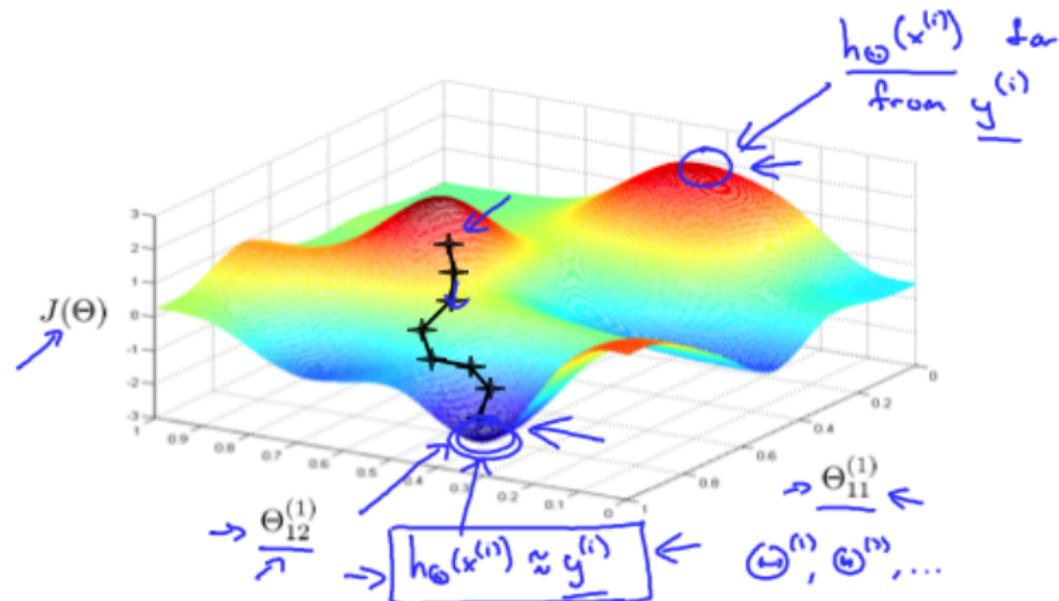
Entrenando una red neuronal

1. Inicializa aleatoriamente los pesos
2. Implemente la propagación hacia adelante para obtener $h_{\Theta}(x^{(i)})$ para cualquier $x^{(i)}$
3. Implementar la función de coste
4. Implemente la retropropagación para calcular derivadas parciales
5. Utilice la comprobación de gradiente para confirmar que su propagación hacia atrás funciona. Luego deshabilite la comprobación de gradiente.
6. Utilice el descenso de gradiente o una función de optimización incorporada para minimizar la función de coste con los pesos en theta.



Poniendo todo junto

La siguiente imagen nos da una idea de lo que está sucediendo mientras implementamos nuestra red neuronal:



Andrew Ng



Poniendo todo junto

Esto minimizará nuestra función de coste. Sin embargo, tenga en cuenta que $J(\Theta)$ no es convexo y, por lo tanto, podemos terminar en un mínimo local.



3. Aplicacion de redes neuronales.





Neural Network-Based Autonomous Driving

23 November 1992



















Bibliografía

- <https://www.coursera.org/learn/machine-learning/home/week/5>

