07 - “Backpropagation” – Fundamentos de Deep Learning – (Parte 7) - Transcript

En esta serie de vídeos hemos estudiado conceptos muy importantes de deep learning que nos van a permitir construir nuestro primer modelo de red neuronal.

En este vídeo presento un resumen breve del algoritmo de descenso por gradientes o gradient descent que es utilizado para actualizar los parámetros de nuestra red neuronal y así minimizar la función de pérdida.

Recordemos que estrictamente la función de pérdida representa que también funciona nuestro modelo, es decir, que tan buenos son nuestros parámetros para representar un ejemplo en particular, en este caso para identificar la clase a la que pertenece una imagen en particular.

La función de costo estrictamente es el promedio de las funciones de pérdida de todos los datos que agrupan o que forman nuestros datos de entrenamiento, nuestro set de entrenamiento.

Después voy a presentar cómo calcular el número de parámetros de una red neuronal el cual depende del número de entradas así como el número de neuronas en cada capa de la red neuronal.

Finalmente, con un ejemplo muy didáctico y sencillo voy a explicar el algoritmo de retro propagación o back propagation para obtener el gradiente de la función de pérdida o de costo con respecto a cada uno de los parámetros que integran nuestra red neuronal el cual en combinación con descenso por gradientes va a ser utilizado para entrenar nuestro modelo y que así nuestra red neuronal aprenda a clasificar correctamente las imágenes de entrada.

Vamos a empezar.

Previamente presentamos el algoritmo de descenso por gradientes o gradient descent el cual es utilizado para actualizar los parámetros de nuestro modelo de red neuronal y así minimizar el valor de la función de costo.

En este ejemplo sencillo donde la función de costo depende solamente de un parámetro W lo que hacemos es calcular la derivada de la función de costo con respecto por ejemplo inicialmente a este punto a este valor inicial de W y así la derivada o de forma más general el gradiente nos indica la dirección de mayor razón de cambio de mayor cambio de la función de costo con respecto a este parámetro W.

Vamos a llamarlo simplemente W1 por simplicidad.

Lo que nosotros queremos es lo que el algoritmo de descenso por gradientes realiza es que vamos a tomar un paso en el sentido contrario del gradiente ya que no queremos avanzar en el sentido de mayor razón de cambio queremos avanzar en el sentido contrario es decir para minimizar dicha función por lo tanto vamos a tomar un paso en el sentido contrario de gradiente que también está dado por la pendiente en este ejemplo sencillo y vamos a generar un nuevo valor W y este sería nuestro nuevo valor W.

Así con este nuevo valor volveremos a calcular nuestra función de costo la cual ahora tendrá un valor diferente en este punto y volvemos a calcular el gradiente de la función de costo con respecto al nuevo valor W y repetimos el proceso actualizando así de forma iterativa nuestros valores de nuestros parámetros W hasta llegar al valor que minimiza dicha función de costo por lo tanto para actualizar los parámetros se utiliza la fórmula donde W es igual al valor W actual menos un valor alfa o learning rate por la derivada de la función de costo con respecto a nuestro parámetro W.

Aquí estoy utilizando por simplicidad una notación de una sola variable de forma más general W va a ser una matriz por lo que usamos W mayúscula es igual al valor actual de todos estos parámetros menos el valor alfa por la derivada de j o la función de costo con respecto a W pero ya utilizaríamos la notación de gradiente donde dependemos de múltiples variables de forma similar se calcularía la actualización del valor de nuestros parámetros Bayas donde cada uno de estos Bayas o Offsets pertenecen a cada una de las neuronas en nuestro modelo el valor nuevo de Bayas de B estaría dado por B menos este valor alfa por la derivada de nuestra función de costo con respecto a W.

Recordemos que el valor de este learning rate representa qué tan grande es el paso que vamos a tomar para ir minimizando nuestra función de costo es un valor crítico ya que un valor muy pequeño nos haría converger al mínimo de manera muy muy lenta y un valor muy grande podría resultar en un paso que no lleva al mínimo sino que en algún momento la función de costo no se minimice sino al contrario explote y se vaya haciendo más grande cada vez entonces es muy importante sintonizar bien este valor volviendo a nuestro ejemplo sencillo con el que hemos trabajado ya frecuentemente nos damos cuenta que en este caso tenemos valores de pesos sinápticos asociados a cada neurona en este ejemplo tenemos simplemente una capa de tres neuronas pero cada neurona tiene cuatro pesos sinápticos asociados por lo tanto tenemos un total de tres valores que son para correspondientes a cada neurona cuatro pesos sinápticos para cada neurona por lo que tendríamos tres por cuatro igual a 12 pesos sinápticos en total en esta red neuronal que como mostramos en un ejemplo anterior representaríamos en una matriz de tres por cuatro no olvidemos que además cada neurona tiene asociado un bayas por lo que además de estos pesos sinápticos tenemos que sumarle un parámetro más que vamos a entrenar que es el este parámetro B por lo que tendríamos en total 12 pesos sinápticos más en este caso tres bayases dándonos un total de 15 parámetros que nuestra red neuronal va a entrenar por lo que regresando a la simplificación anterior nos damos cuenta que aún en un modelo sumamente sencillo como el que tenemos en este ejemplo nuestra función de costo que esté aquí a la salida o nuestra función de pérdida si solamente hablamos de una muestra que estamos procesando a través de nuestra red neuronal en este caso una sola imagen depende de 15 parámetros por lo tanto estamos hablando de un espacio multidimensional en el cual queremos encontrar el mínimo un espacio mucho más grande que el espacio que estamos mostrando aquí en este ágeno donde solo depende de un parámetro sin embargo para fines ilustrativos el proceso es muy similar este modelo por ejemplo es un modelo más grande de red neuronal y también sigue siendo un modelo pequeño donde tenemos solamente dos capas este es una capa, este es otra capa, estas se llaman capas ocultas y una capa de salida un modelo que en total tiene tres capas más las entradas que en este caso son ocho elementos de entrada pueden ser los píxeles de un imagen si seguimos con el formato de ejemplo que hemos venido manejando algo que vale la pena notar y que es importante considerar es la cantidad de parámetros que nuestro modelo tiene por ejemplo en este modelo aunque se ve mucho más complejo que los modelos anteriores que el modelo anterior que veníamos trabajando de una sola capa podemos ver que tenemos entonces ocho elementos de entrada vamos a decir que son ocho píxeles y doce neuronas en cada capa oculta y la capa de salida tiene solamente tres neuronas que corresponderían a las clases que vamos a reconocer que en nuestro ejemplo han sido la clase gato, perro y ave si quisiéramos calcular el número de parámetros que este modelo tiene simplemente tenemos que pensar en cuántas conexiones hay asociadas en cada capa de nuestra red normal por ejemplo si pensamos en la primera capa tenemos que son ocho elementos de entrada por lo tanto cada neurona va a tener asociados ocho pesos sinápticos uno correspondiente a cada elemento de entrada y son doce neuronas por lo tanto nuestra capa de entrada vamos a llamar que nuestra capa de entrada uno vamos a llamar la W1 o nuestra matriz de parámetros tendría dimensiones si pusiéramos por ejemplo un shape para ver las dimensiones tendría dimensiones en este caso de doce neuronas, doce por ocho y en total estaríamos hablando de que tenemos esta cantidad de parámetros que serían 96 pesos sinápticos pero no olvidemos que cada neurona además tiene asociado un bayas cada una de estas neuronas tiene un bayas que también es un parámetro que vamos a aprender por lo tanto no solamente tenemos los pesos sinápticos sino también tenemos ese elemento de bayas que en este caso sería uno para cada neurona lo que nos daría un total de 108 parámetros que la red que nuestro modelo tiene que aprender que tenemos que entrenar simplemente para la primera capa en la segunda capa vamos a llamarle W2 si hacemos el mismo procedimiento y calcularamos la forma o las dimensiones de esta matriz de W2 tendríamos que esta capa tiene doce neuronas y ahora cada neurona tiene asociado un peso sináptico para cada elemento de entrada a dicha neurona y ahora los elementos de entrada ya no se tratan de los píxeles de nuestra imagen sino se trata de los elementos de las activaciones de la capa previa y la capa previa tiene doce neuronas por lo tanto nuestra segunda capa tendría en total doce neuronas por doce conexiones a cada neurona, doce pesos sinápticos además también no olvidemos el bayas que también está asociado a esta capa a esta capa, a cada neurona de esta capa por lo que tendríamos que sumarle doce elementos para el bayas y este resultado nos va a dar un total de 156 parámetros que la red tiene que aprender para la segunda capa oculta y finalmente la capa de salida si llamamos esto W3 tendría una forma, aquí solamente tenemos tres neuronas de salida pero cada neurona tiene asociado una conexión para cada elemento de la capa previa por lo que serían tres por doce y además cada neurona tiene asociado suballas que en este caso serían solamente tres por lo que tendríamos un total de tres por doce, treinta y seis más tres, treinta y nueve treinta y nueve elementos o parámetros que se van a entrenar simplemente para la capa de salida, para la última capa en total tendríamos 108 más 156 más 39 parámetros dándonos un total de 303 elementos entrenables en esta red neuronal y 303 es un gran número de parámetros para visualizarlo como un espacio multidimensional donde la función de pérdida para un determinado ejemplo depende de todos esos parámetros por lo que es importante entrenar o actualizar cada uno de estos parámetros utilizando el algoritmo de descenso por gradientes así, aunque es un modelo muy sencillo para el estándar del número de parámetros en redes neuronales actuales, es un modelo mucho más complejo que el ejemplo con el que venimos trabajando y la forma de abordar este modelo cuando lo entrenemos mediante el uso de software como Python o algún framework de deep learning nosotros simplemente vamos a actualizar todos los parámetros de un solo paso como mencionamos en la diapositiva anterior dada por la ecuación W igual al valor actual de W menos este alfa por la derivada de la función de pérdida con respecto a W, donde W es una matriz que contiene todos los parámetros ahora, para entender mejor cómo se aplica este proceso de aprendizaje es muy útil considerar el modelo mostrado en la pantalla donde vamos a abordar solamente una neurona a la vez entonces, imaginemos que esta neurona ahora viéndola como un nodo en una gráfica computacional que realiza dos operaciones, estrictamente un nodo en una gráfica computacional realizaría la multiplicación y luego la suma sin embargo, para fines didácticos vamos a asumir que este nodo realiza la multiplicación y luego a ese resultado de la multiplicación le suma un valor que en este caso sería nuestro valor de vallas imaginemos entonces que esta neurona está en la capa de entrada donde una de las entradas a este nodo va a ser uno de los valores de los píxeles X en el caso de que estemos hablando de una imagen y la otra entrada va a ser el parámetro W asociado a ese píxel para esta neurona veamos entonces que aunque usualmente representamos que el parámetro está asociado en esta conexión vamos a manejarlo como si fueran dos entradas de tal forma que la salida en este punto en esta neurona pues estaría dado por la multiplicación de X o W por X, vamos a decir, más el vallas este es la salida en este valor luego este valor se envía y se vuelve en la entrada vamos a llamar que esto es z y esto se vuelve la entrada para la siguiente neurona donde aquí también tiene una entrada que es una W, otra vamos a decirle W2 simplemente por simplicidad y tiene asociado también un peso sináptico B2 de esta forma aquí tenemos una salida z2 y esta salida z2 se va a conectar a otras neuronas donde este proceso se va a repetir en algún momento nuestro proceso de cómputo va a terminar arrojando vamos a decirlo así, vamos a... estos puntos significa que pasa por otras capas, pasa por otras neuronas pero eventualmente se va a obtener una función de pérdida asociada a esta imagen de entrada entonces esta función de pérdida va a ser la salida en nuestro modelo completo y lo que queremos hacer, voy a utilizar otro color para hacer esta explicación es luego obtener el gradiente de la función de pérdida con los elementos de las capas previas de tal forma que eventualmente a esta neurona que tenemos aquí le va a llegar el gradiente de la función de pérdida, de esta función de pérdida con respecto a este valor z2 ya que el proceso que se sigue es un proceso que se llama backpropagation el cual consiste en tomar el gradiente a la salida con respecto al último valor que se tenía para generar esta salida e irlo transmitiendo hacia atrás, es decir, retropropagándolo por eso este esquema de aprendizaje se conoce como backpropagation entonces en algún momento va a llegar el gradiente de la función de pérdida con respecto a este valor z2 y este gradiente va a llegar siguiendo el proceso de retropropagación y lo que queremos es calcular cuál es el gradiente de la función de pérdida con respecto a los parámetros w2 el gradiente de la función de pérdida con respecto a b2 es decir, con respecto a este parámetro que son los parámetros que nosotros vamos a aprender que nuestra red neuronal va a aprender y asimismo con respecto a los otros parámetros w vamos a decirle w1 para identificarlos vamos a decir que este es w1 y este es b1 y el gradiente de la función de pérdida con respecto a b1 al primer vallas para hacer esto lo que vamos a hacer es uso de la regla de la cadena entonces en este punto podemos decir que la derivada de z2 o voy a utilizar estrictamente como estamos hablando solamente de valores que dependen de una sola variable voy a utilizar la notación tradicional de derivada y no la de gradiente entonces la derivada de z2 con respecto a este parámetro w2 podemos calcular directamente en este punto y nos damos cuenta que la derivada aquí por ejemplo la derivada repito voy a usar la notación normal la derivada de la función de pérdida con respecto a w2 estaría dada por la aplicación de la regla de la cadena donde tomamos la derivada de la función de pérdida con respecto a z2 por la derivada de z2 con respecto a w2 entonces este resultado de esta aplicación de la regla de la cadena nos va a dar el resultado de la función nos va a dar la derivada de la función de pérdida con respecto a w2 lo que va a pasar es que ese resultado de gradiente es el que se propaga de regreso en este sentido y nos va a permitir actualizar los valores de nuestros parámetros w2 en este caso se propaga la derivada de L o la función de pérdida con respecto a w2 de forma similar aplicando continuamente la regla de la cadena vamos a transmitir aquí el gradiente de la función de pérdida ahora con respecto a b2 y en este punto se transmite el gradiente de la función de pérdida con respecto a z vamos a llamar este z1 para identificar que es no es este z aquí con respecto a z1 por lo tanto también se calcula el gradiente en esta unidad para la entrada a dicha neurona y así mismo entonces ya tenemos en este punto que se está regresando el gradiente de la función de pérdida con respecto a z1 aquí entonces podemos calcular el gradiente local el cual estaría dado por la derivada de z1 con respecto a x y aquí sería también el gradiente local sería la derivada de z1 con respecto a w1 y aquí también tendríamos un gradiente local que sería la derivada de z1 con respecto a b1 y así de esta forma derivada de z1 con respecto a b1 nos lleva a que la derivada de la función de pérdida que es la que queremos encontrar con respecto a b1 por ejemplo estaría dado por la derivada de la función de pérdida con respecto a este z1 por la derivada de la función z1 con respecto a b1 y ese resultado es el que retro propagamos que sería igual vamos a decirlo de l con respecto a db1 y es el que vamos a utilizar para actualizar nuestros valores b1 y recordemos los vamos a actualizar utilizando el esquema de de senso por gradientes donde b1 es igual al valor de b1 actual menos alfa por la derivada de la función de pérdida con respecto a b1 y aquí pido una disculpa porque dije que iba a utilizar la notación de cálculo de una sola variable sin embargo por costumbre estoy utilizando la notación de cálculo multivariable en realidad la notación de cálculo multivariable es la notación correcta ya que todos estos casos aunque aquí estamos trabajando con neuronas muy muy puntuales donde solamente tienen una entrada en los casos reales vamos a tener que las entradas son vectores por lo tanto necesitamos calcular el gradiente el cual está dado por un vector que es la razón de cambio de una función en este caso la función de pérdida con respecto a un vector de entrada es decir que tanto cambia la función de pérdida l en este caso cuando ajustamos vamos a decir w1 w2 o w3 es decir con respecto a un vector de esta forma así vamos ajustando nuestros parámetros de la red normal finalmente podemos decir que aquí fluye un gradiente de regreso para w1 donde el cálculo sería el gradiente que estaría dado por la derivada de la función de pérdida con respecto a w1 que sería la derivada de la función de pérdida con respecto a z1 es decir la salida de esa unidad por la derivada de z1 con respecto a w1 es decir el gradiente local el gradiente fluye de regreso y aquí lo que hacemos es actualizar w1 como el valor actual de w1 menos alpha o el learning rate por la derivada de la función de pérdida con respecto a w1 notemos aquí como se trata de la primera entrada es decir el pixel no tiene caso hacer una retro propagación al entrada a los píxeles la imagen sin embargo se podría calcular a diferencia de la neurona con la que empezamos a trabajar donde si tenía sentido hacer la retro propagación con el valor de la entrada ya que este esta valor de entrada z1 para la siguiente neurona es la salida de esta neurona este proceso de retro propagación o back propagation no es otra cosa más que una aplicación de la regla de la cadena en cálculo así es como vamos a ir calculando los gradientes de la función de pérdida con respecto a cada uno de los parámetros no obstante no lo vamos a hacer neurona por neurona vamos a hacer mediante matrices que representan todos los pesos sinápticos pero es importante entender como es el proceso y creo que este ejemplo muy sencillo ilustra muy bien dicho esquema de aprendizaje en este vídeo con un ejemplo muy sencillo presentamos como el algoritmo de retro propagación en combinación con descenso por gradientes es utilizado para actualizar nuestros parámetros w y b es decir nuestros pesos sinápticos y los vayases para minimizar nuestra función de costo y así mejorar el desempeño de nuestra red neuronal en el siguiente vídeo vamos a presentar un ejemplo muy sencillo de un paso completo del modelo de red neuronal que hemos venido trabajando es decir vamos a tener vayases vamos a tener pesos sinápticos vamos a tener píxeles los cuatro píxeles que estamos asumiendo hasta el momento vamos a hacer el forward pass de nuestro modelo y vamos a tener la función de pérdida para una entrada en particular y con este valor voy a mostrar de forma didáctica como podemos calcular el gradiente de esta función de pérdida con respecto a todos los parámetros que integran nuestra red neuronal y así actualizaremos nuestros parámetros y luego volveremos a hacer otro forward pass para demostrar que con estos nuevos parámetros ya actualizados nuestra función de pérdida en realidad disminuye lo cual es lo que queremos hacer entonces vamos a seguir ¡Adiós!

[Música] [MÚSICA]