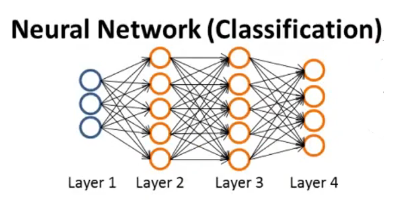
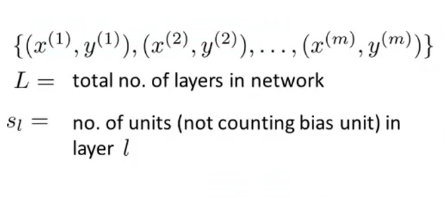
## Función de costes

Las redes neuronales son uno de los algoritmos de aprendizaje más potentes que tenemos hoy. En este video y en los siguientes, me gustaría comenzar a hablar sobre un algoritmo de aprendizaje para establecer los parámetros de la red neuronal dado un conjunto de entrenamiento. En cuanto a la discusión de la mayoría de los algoritmos de aprendizaje, vamos a comenzar hablando de la función de costo para ajustar los parámetros de la red. Voy a enfocarme en la aplicación de las redes neuronales para los problemas de clasificación.

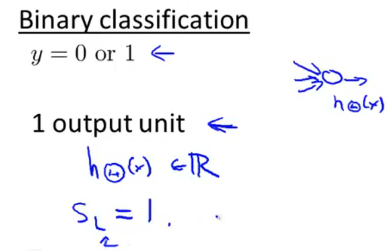
Entonces, supongamos que tenemos una red como la que se muestra:



Y supongamos que tenemos un conjunto de entrenamiento como este con pares xi y yi de ejemplos de entrenamiento igual a m. Voy a usar la L mayúscula para denotar el número total de capas en esta red. Así, para la red que se muestra, tendríamos que L mayúscula es igual a 4. Y voy a usar s subíndice l para denotar el número de unidades o neuronas, sin contar la unidad de oscilación en la capa L de la red. Entonces, por ejemplo, tendríamos un S1 que es la capa de entrada igual a 3, S2 en mi ejemplo son cinco unidades. Y la capa de salida S4, Que también es igual SL, porque la L mayúscula es igual a cuatro. La capa de salida en mi ejemplo tiene cuatro unidades.

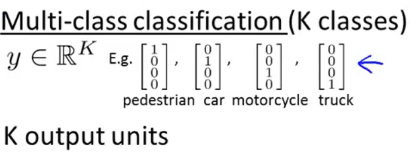


Vamos a considerar dos tipos de problemas de clasificación. La primera es la clasificación binaria, en donde las etiquetas "y" son cero o uno. En este caso, tendríamos una única unidad de salida. Entonces, esta red neuronal en la parte superior tiene cuatro unidades de salida, pero si tuviéramos una clasificación binaria, sólo tendríamos una unidad de salida que calculará h(x). Y las salidas de la red neuronal serían h(x), que sería un número real. Y, en este caso, el número de unidades de salida, SL, donde L nuevamente es el índice de la capa final, porque es el número de capas que tenemos en la red. Entonces, el número de unidades que tenemos en la capa de salida va a ser igual a uno.

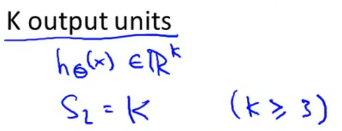


En este caso, para simplificar la notación más tarde, también voy a establecer k como el número de la unidad en la capa de salida que en este caso es igual a 1.

El segundo problema de tipo de clasificación que consideraremos será el problema de clasificación multiclase, en el que podemos tener k distintas clases. Entonces, el ejemplo anterior tenía esta representación para "y" cuando teníamos cuatro clases.

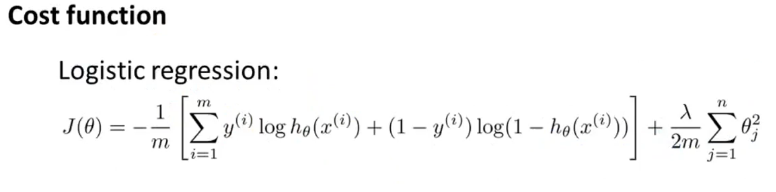


En este caso, tendremos K mayúscula unidades de salida y nuestras hipótesis mostrarán vectores de salida que tienen K dimensiones. Y el número de unidades de salida serán igual a K.

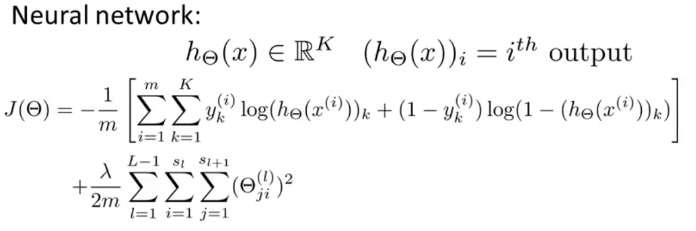


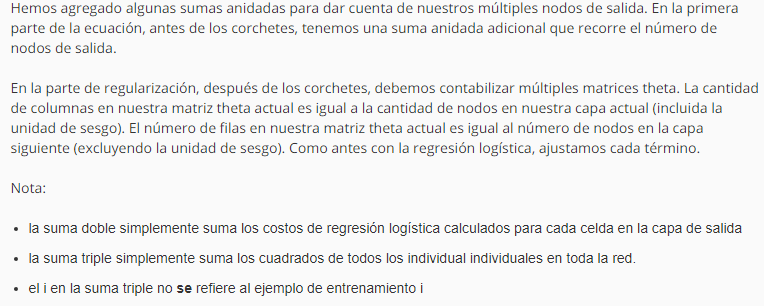
Por lo general, tendremos una K mayor o igual a tres en este caso, porque si tuviéramos dos clases, entonces no necesitaríamos utilizar el método de todos contra uno. Sólo debemos usar el método todos contra uno si tenemos una K mayor o igual a tres clases. Entonces, como sólo tenemos dos clases, sólo necesitamos utilizar una unidad de salida. Ahora, vamos a definir la función de costo para nuestra función de costo para nuestra red neuronal.

La función de costo que usaremos para la red neuronal será una generalización de la que usamos para la regresión logística. Para la regresión logística solíamos minimizar la función de costo j de «theta» que era menos 1 sobre m de esta función de costo, y después se suma este término de regularización adicional, que era la suma de j igual a 1 hasta n, porque no regularizamos el término de oscilación «theta» cero.



Para una red neuronal, nuestra función de costo va a ser una generalización de ésta. Donde, en lugar de tener básicamente sólo una unidad de salida de regresión logística, podemos tener K de éstas. Entonces, esta es nuestra función de costo.





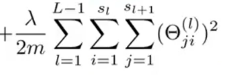
Ahora la red neuronal muestra vectores en RK en donde K podría ser igual a 1 si tenemos el problema de clasificación binaria.

Voy a utilizar la notación, para denotar la salida “i-ésima” . Eso es h de x es un vector de K dimensiones. Y así, el subíndice i sólo selecciona el elemento “i-enésimo” del vector mostrado por mi red neuronal.

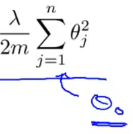
Mi función de costo, j de «theta» ahora va a ser la siguiente, es menos uno sobre m de la suma de un término similar a lo que teníamos en la regresión logística. Excepto que tenemos que esta suma desde K es igual a uno hasta K. La sumatoria es básicamente una suma sobre mi unidad de salida K. Entonces, si tengo cuatro unidades superiores, esto es que la capa final de mi red neuronal tiene cuatro unidades de salida, entonces, esta es una suma desde K igual a uno hasta cuatro de básicamente la función de costo de los algoritmos de regresión logística, pero sumando la función de costo sobre cada una de mis cuatro unidades de salida. Entonces, notarás que esto aplica particularmente a , porque básicamente estamos tomando la unidad de salida K y comparándola con el valor de



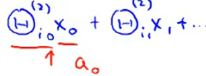
Y, por último, el segundo término es el término de regularización similar a lo que teníamos para la regresión logística.



Estas sumatorias de términos que se ven muy complicadas, y siempre es complicado hacer una suma con estos términos, «theta» j i l para todos los valores de i j y l. Excepto que no sumamos los términos correspondiente a los valores de oscilación como lo hicimos para la progresión logística. En concreto, no sumamos los términos correspondientes a cuando i es igual a cero.



Esto se debe a que cuando calculamos la activación de la neurona, tenemos términos como estos, como sabes «theta», i 0 más «theta», i 1, x1 más, y así sucesivamente, donde supongo que podríamos tener un 2 si esta es la primera capa oculta, de forma que los valores con el 0 ahí, que corresponden a algo que se multiplica en un x0 o un a0



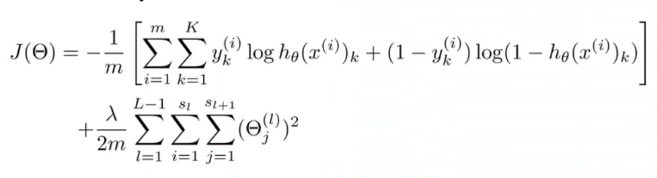
Y así, esto es como una unidad de oscilación y, por analogía con lo que hicimos para la progresión logística, no sumamos esos términos en nuestro término de regularización porque no queremos regularizarlos y encadenar los valores a 0. Pero esta sólo es una convención posible, e incluso si fueras a sumar sobre, tu sabes, i es igual a 0 hasta SL, funcionará más o menos igual y no haría una gran diferencia. Pero tal vez esta convención de no regularizar el término de oscilación sólo es ligeramente más común.

Entonces, esta es la función de costo que vamos a utilizar para llenar tu propia red. En el siguiente video empezaremos a hablar sobre un algoritmo para intentar optimizar la función de costo.

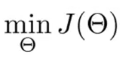
## Algoritmo Backpropagation

En el video anterior hablamos sobre una función de costo para la red neuronal. En este video, vamos a empezar a hablar de un algoritmo, para minimizar la función de costo. En particular, vamos a hablar sobre el algoritmo retropropagación.

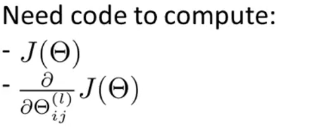
Aquí se encuentra la función de costo que anotamos en el video anterior:



Lo que nos gustaría hacer es tratar de encontrar los parámetros de «theta» para tratar de minimizar J de «theta». Con el fin de utilizar ya sea el descenso de gradiente o uno de los algoritmos de optimización avanzados.



Lo que tenemos que hacer, por lo tanto, es escribir el código que lleva esta entrada a los parámetros de «theta» y calcula j de «theta» y estos términos de la derivada parcial.

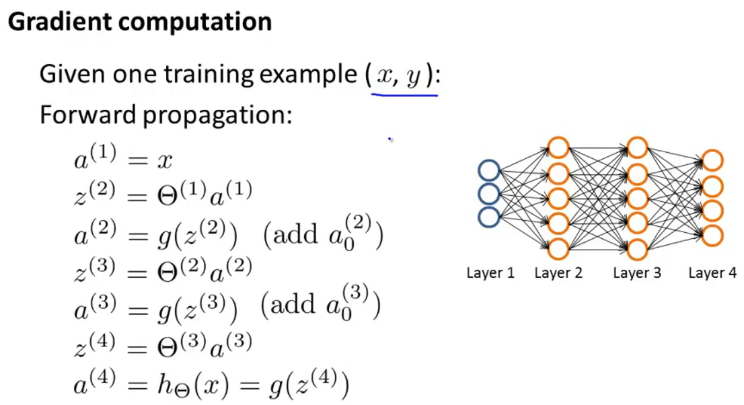


Recuerda que los parámetros en la red neuronal de estos elementos, en «theta» superíndice l subíndice ij, ese es el número real y por eso, estos son los términos de la derivada parcial que necesitamos calcular.

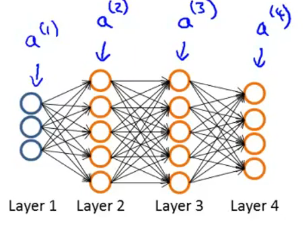


Con el fin de calcular la función de costo j de «theta» sólo tienes que utilizar la fórmula de aquí arriba y por eso, lo que quiero hacer en la mayor parte de este video es enfocarme en hablar acerca de cómo podemos calcular los términos de la derivada parcial.

Vamos a empezar por hablar de un caso cuando sólo tenemos un ejemplo de entrenamiento, así que imagínate, si quieres que todo nuestro conjunto de entrenamiento conste de un solo ejemplo de entrenamiento que sea un par xy. no voy a escribir x1y1, sólo escribe esto. Escribe un ejemplo de entrenamiento como xy y pasemos a la secuencia de cálculos que íbamos a hacer con este ejemplo de entrenamiento.

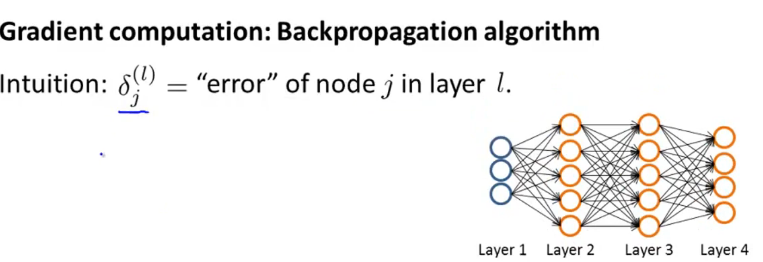


Lo primero que hacemos es aplicar la propagación hacia adelante para calcular si una hipótesis en realidad resulta dada esta entrada. En concreto, recordemos que, la llamamos a(1), que son los valores de activación de esta primera capa que era la entrada acá. Bien, voy a establecer eso para x y luego vamos a calcular z(2) igual a «theta»(1) a(1) y a(2) es igual a g, la función de activación sigmoidea aplicada a z(2) y esto nos daría nuestras activaciones para la primera capa interna. Eso es para la capa dos de la red y añadimos también estos términos de oscilación. A continuación aplicamos 2 pasos más de estos cuatro y la propagación para calcular a(3) y a(4) que es también la salida de una hipótesis h de x.



Así que ésta es nuestra implementación vectorizada de la propagación hacia adelante y que nos permite calcular los valores de activación para todas las neuronas en nuestra red neuronal.

A continuación, con el fin de calcular las derivadas, vamos a utilizar un algoritmo denominado Retropropagación.

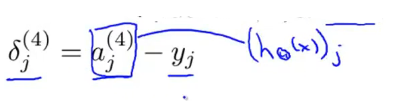


La intuición del algoritmo de retropropagación es que por cada observación vamos a calcular el término «delta» superíndice subíndice j y eso de algún modo va a representar el error de la observación j en la capa.

Bien, recuerda que superíndice subíndice j es la activación de de la unidad j en la capa así que, este término «delta» en cierto sentido va a capturar nuestro error en la activación de ese dúo neuronal. Entonces, ¿cómo podríamos desear que la activación de esa observación fuera un poco diferente? Pues simple, tomando el ejemplo de la red neuronal que tenemos a la derecha que tiene cuatro capas donde L mayúscula es igual a 4. Por cada unidad de salida, vamos a calcular este término «delta». Así, «delta» para j de la unidad en la cuarta capa es igual a justo la activación de esa unidad menos el que era el valor real de y en nuestro ejemplo de entrenamiento.



Bien, este término de aquí también se puede escribir como:



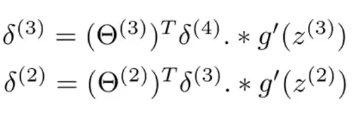
Y así este término «delta» es justo la diferencia entre lo que la hipótesis da por resultado y el que era valor de "y" en nuestro conjunto de entrenamiento, donde esta "y" subíndice j es j del elemento del valor del vector "y" en nuestro conjunto de entrenamiento con valor asignado.

Y, por cierto, si tú piensas en «delta» "a" y en "y" como vectores, entonces puedes también tomarlos y proponer desarrollar una implementación vectorizada de que es justo «delta» 4 que se establece como:



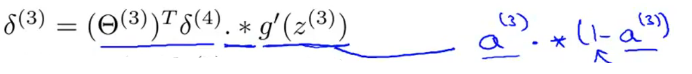
Y aquí, cada uno de estos «delta» 4 a4 y "y", cada uno de estos es un vector cuya dimensión es igual al número de unidades de salida de nuestra red.

Así que ahora hemos calculado los términos del error de «delta» 4 para nuestra red. Lo que debemos hacer después es calcular los términos «delta» para las capas anteriores de nuestra red:



He aquí una fórmula para calcular «delta» 3, y es «delta» 3 es igual a «theta» 3 transpuesta multiplicado por «delta» 4. Y ese punto asterisco, es la operación de multiplicación elemento a elemento que conocemos de MATLAB. Así que «delta» 3 transpuesta a «delta» 4, ése es un vector; g prima z3 ése también es un vector así que la operación con esos dos vectores es una multiplicación elemento a elemento.

El término g prima de z3, que formalmente es en realidad la derivada de la función de activación de g evaluada en los valores de entrada dados por z3. Si sabes cálculo, puedes tratar de resolverlo tú mismo y ver si puedes simplificarlo hasta el mismo resultado al que llegué. Pero yo sólo te voy a decir pragmáticamente lo que eso significa. Lo que tú haces para calcular esta g prima, en estos términos derivados es justo a3 punto veces 1 menos a3 en donde a3 es el vector de las activaciones. 1 es el vector de los unos y a3 es de nuevo la activación del vector de los valores de la activación para esa capa.



A continuación, aplicas una fórmula similar para calcular «delta» 2 en donde de nuevo ésta puede calcularse utilizando una fórmula similar. Sólo que ahora es a2 como tal y yo luego lo demuestro aquí, pero tú puedes en realidad, y es posible demostrarlo si sabes cálculo que esta expresión es igual a matemáticamente, la derivada de la función g, de la función de activación, que estoy señalando mediante g prima.



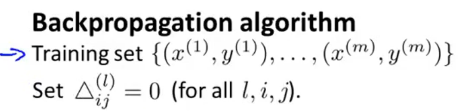
Y, finalmente, eso es todo y no hay ningún término «delta»1, porque la primera capa corresponde a la capa de entrada y esa es justo la variable que observamos en nuestros conjuntos de entrenamiento, que no tengan ningún error asociado, no queremos tratar de cambiar esos valores. Y así tenemos los términos de «delta» sólo para las capas 2, 3 y 4 para este ejemplo.

El nombre retropropagación proviene del hecho de que empezamos por calcular el término «delta» para la capa de salida y luego regresamos una capa y calculamos los términos «delta» para la tercera capa oculta y luego regresamos otro paso para calcular «delta» 2 y así sucesivamente, digamos que estamos retropropagando los errores de la capa de salida hacia la capa 3 y hacia la capa 2, de ahí el nombre de Retropropagación.

Finalmente, la derivación es muy complicada y está sorprendentemente implicada pero si realizas estos pocos pasos de cálculo es posible que demuestres, y es francamente algo viral, una complicada demostración matemática. Es posible demostrar que si tú ignoras la regularización entonces los términos de la derivada parcial que quieres están exactamente dados por las activaciones y estos términos «delta». Ignorando lambda o si lo quieres ver de otro modo, como si el término de regularización lambda fuese igual a 0.



Arreglaremos este detalle más adelante con respecto al término de regularización, aunque bueno, al realizar la retropropagación y al calcular estos términos de «delta», puedes, y calcular bastante rápido estos términos de la derivada parcial para todos tus parámetros. Así que estos son un montón de detalles. Vamos a reunir todo y a poner todo en perspectiva para hablar acerca de cómo implementar la retropropagación para calcular las derivadas con respecto a tus parámetros. Y en caso de que tengamos un gran conjunto de entrenamiento no sólo un conjunto de entrenamiento de un ejemplo, esto es lo que hacemos.

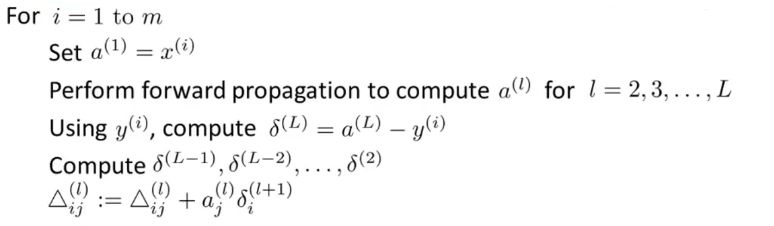


Vamos a suponer que tenemos un conjunto de entrenamiento de "m". Lo primero que vamos a hacer es que vamos a establecer estos «delta» l subíndice i j. ¿Y qué es este símbolo triangular? Esto de hecho es «delta» mayúscula en el alfabeto griego. El símbolo que teníamos en la diapositiva anterior era «delta» minúscula. De modo que el triángulo es la mayúscula de «delta». Vamos a igualar esto a cero para todos los valores de l, i, j. En algún momento, esta «delta» mayúscula l, i, j se utilizará para calcular el término de la derivada parcial, la derivada parcial con respecto a «theta» l, i, j de J de teta.



Bien, como veremos en un momento, estos «delta»s se van a utilizar como acumuladores que lentamente agregarán elementos para calcular las derivadas parciales.

A continuación, vamos a recorrer nuestro conjunto de entrenamiento:



Así que, vamos a decir que i es igual a 1 hasta m y entonces para la iteración de i, vamos a trabajar con el ejemplo de entrenamiento (xi, yi).

Entonces, lo primero que vamos a hacer es establecer a1, que son las activaciones de la capa de entrada, establecer que sea igual a xi, que son las entradas para nuestro ejemplo i de entrenamiento, y luego vamos a realizar la propagación hacia adelante para calcular las activaciones para la capa dos, la capa tres y así sucesivamente hasta la capa final (la capa L mayúscula). A continuación, vamos a utilizar el valor asignado de la salida "yi" de éste ejemplo específico que estamos viendo para calcular el término de error «delta» L para la salida de allí. Entonces, «delta» L es lo que resulta de la hipótesis menos el valor real observado.

Luego vamos a usar el algoritmo de retropropagación para calcular «delta» l menos 1, «delta» l menos 2, y así hasta llegar a «delta» 2 y una vez más ahí no está ahora «delta» 1 porque no asociamos un término de error con la capa de entrada ya que son las variables de entrada tal y como son.

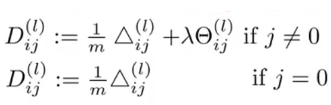
Y, por último, vamos a utilizar estos términos de «delta» mayúscula para acumular estos términos de la derivada parcial que anotamos anteriormente.

Y, por cierto, si tú miras esta expresión, es posible vectorizar esto también. Concretamente, si piensas en «delta» ij como una matriz, indexada mediante subíndice ij. Entonces, si «delta» l es una matriz, se puede reescribir esto como:



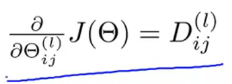
Así que esa es una implementación vectorizada de esto que automáticamente realiza esta actualización de todos los valores de "i" y "j".

Finalmente, después de ejecutar el conjunto de los cuatro ciclo fors salimos entonces del cuarto ciclo for y calculamos el siguiente:

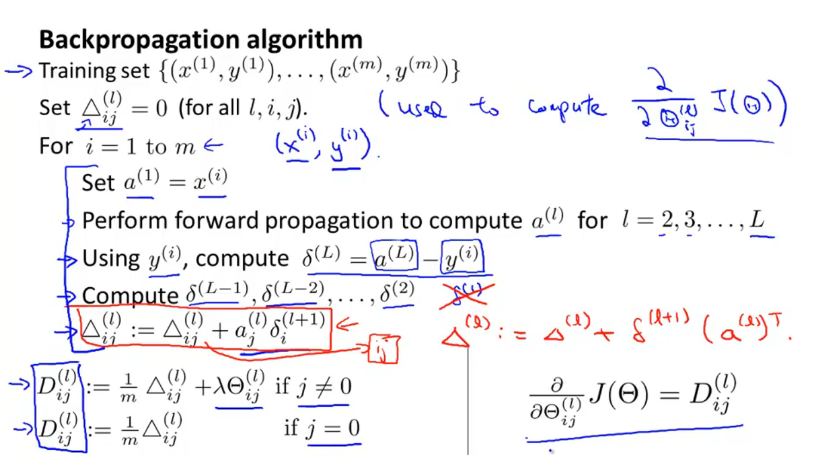


Calculamos D mayúscula como a continuación y tenemos dos casos separados para j igual a cero y j no igual a cero. El caso en el que j es igual a cero corresponde al término de sesgo, así que cuando j es igual a cero esa es la razón por la que nos falta este término de regularización extra.

Por último, a pesar de que la demostración formal es bastante complicada, lo que puedes mostrar es que una vez has calculado estos términos de D, esa es exactamente la derivada parcial de la función costo con respecto a cada uno de tus perímetros, así que puedes usarlos en cualquier descenso de gradiente o en uno de los algoritmos de optimización avanzada



Todo en una misma slide:



Así que ese es el algoritmo de la retropropagación y de cómo calcular las derivadas de tu función de costo para una red neuronal. Sé que esto parece como si estos fueran un montón de detalles y se tratara de un montón de pasos concatenados. Sin embargo, tanto en las tareas de programación escritas y, más tarde en este video, te daremos un resumen de todo ello de tal forma que podamos tener juntas todas las piezas del algoritmo para que sepas exactamente qué es lo que necesitas para poner en práctica, si así lo quieres, para implementar la retropropagación para calcular las derivadas de la función de costos de tu red neuronal con respecto a ésos parámetros

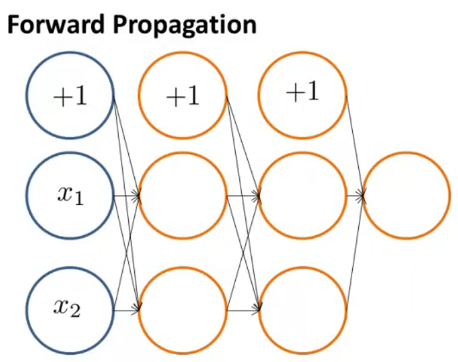
## Intuición de la retropropagación

En el video anterior, hablamos sobre el algoritmo de retropropagación. Para mucha gente que lo esté viendo por primera vez, la primera impresión a menudo es ¡Vaya!, este es un algoritmo en verdad muy complicado y mira todos estos diferentes pasos. No estoy muy seguro cómo encajan entre sí y es como si fuera una caja negra con todos estos pasos complicados. En caso de que así sea como tú te sientes respecto a la retropropagación, no te preocupes, está bien. La retropropagación quizás sea por desgracia un algoritmo matemáticamente menos limpio o menos simple en comparación con la regresión lineal o la regresión logística y yo en realidad he usado la retropropagación, ¿sabes?, con bastante éxito durante muchos años e incluso hoy en día, algunas veces todavía siento como que no tengo un sentido muy claro de qué es lo que está haciendo, en su mayoría es intuición, acerca de qué está haciendo la propagación de fondo. Para aquellos de ustedes que estén haciendo los ejercicios de programación que al menos mecánicamente los guiarán a través de los diferentes pasos sobre cómo implementar la retropropagación, pues bien, ustedes serán capaces de hacer que esto funcione para sí mismos.

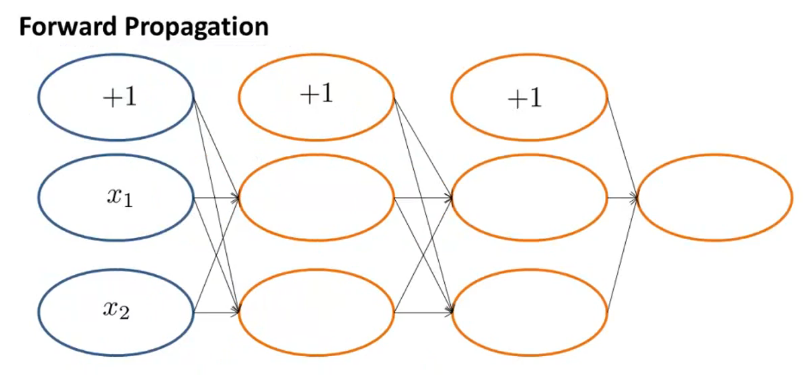
Y lo que quiero hacer en este video es poner un poco más de atención sobre los pasos mecánicos de la retropropagación y tratar de darles una intuición un poco mejor acerca de qué es lo que están haciendo la retropropagación para con suerte convencerlos de que, al menos de qué es un algoritmo razonable.

En caso de que aún después de este video, la retropropagación todavía te parezca que es una caja muy negra y, que bueno, ya sabes, sean pasos demasiado complicados e incluso un tanto mágicos para ti, no te preocupes, todo está bien. Y aun cuando, ya sabes, he utilizado la retropropagación durante muchos años, a veces es un algoritmo difícil de entender. Pero espero que este video te ayude un poco.

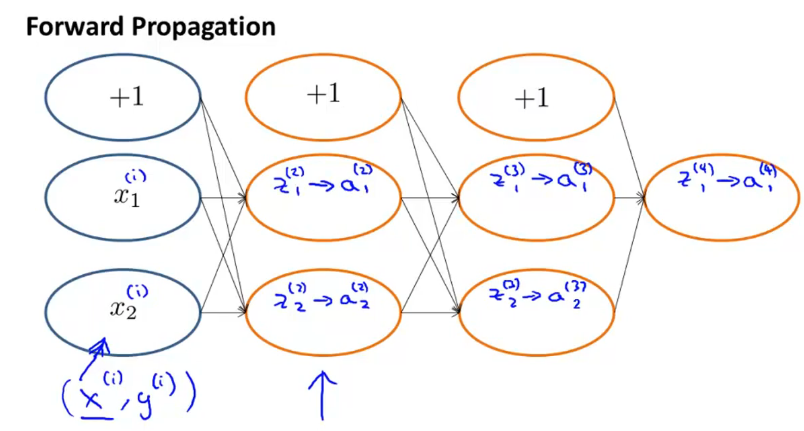
Con el fin de comprender mejor la retropropagación, echemos otro vistazo a qué está haciendo la propagación hacia adelante. Aquí se encuentra una red neuronal con dos unidades de entrada (sin tener en cuenta la unidad de oscilación), y dos unidades ocultas en esta capa y dos unidades ocultas en la siguiente capa y luego tendríamos una unidad de salida. Y otra vez, estas cuentan 2, 2, 2 no cuentan estas unidades de oscilación en la parte superior.



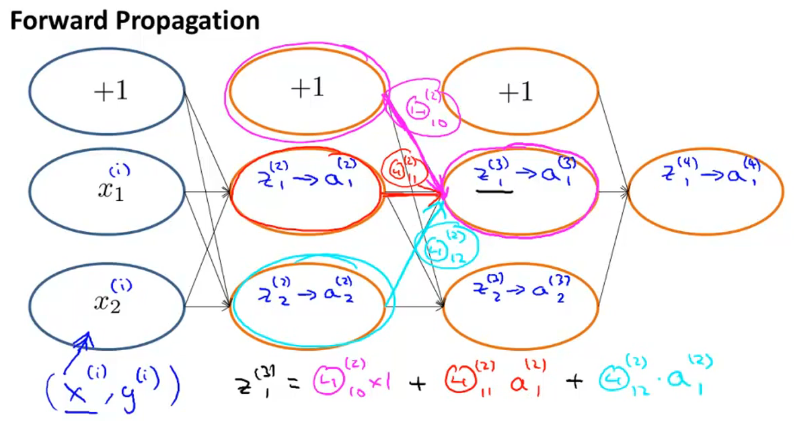
Con el fin de ilustrar la propagación hacia adelante, voy a dibujar esta red un poco diferente. Y en particular, voy a dibujar esta red neuronal con los nodos dibujados como estas elipses muy gordas, de modo que pueda escribir texto en ellas.



Cuando se realiza la propagación hacia adelante, puede que tengamos algún ejemplo concreto, digamos algún ejemplo con (x(i) ,y(i)) y será esta x(i) que introdujimos en la capa de entrada, para que esto pueda ser, x(i)1 y x(i)2 (i) que son los valores con los que establecemos la capa de entrada y cuando los propagamos hacia adelante en dirección a la primera capa oculta, lo que que hacemos es calcular z(2) 1 z(2) 2, y así, éstos son la suma ponderada de las entradas de las unidades de entrada y luego aplicamos el sigmoide de la función logística y la función de activación sigmoidea aplicada al valor de z, y nos da estos valores de activación Bien, esto nos da a(2)1 y a(2) 2 y luego propagamos hacia adelante otra vez para obtener, ya sabes, aquí, z(3) 1, aplicamos el sigmoideo de la función logística, la función de activación a esto, para obtener 3,1 y asimismo, de este modo, hasta que lleguemos a z(4) 1, aplicamos la la función de activación a esto y nos da a(4) 1 que es el mejor valor de salida de la red neuronal.



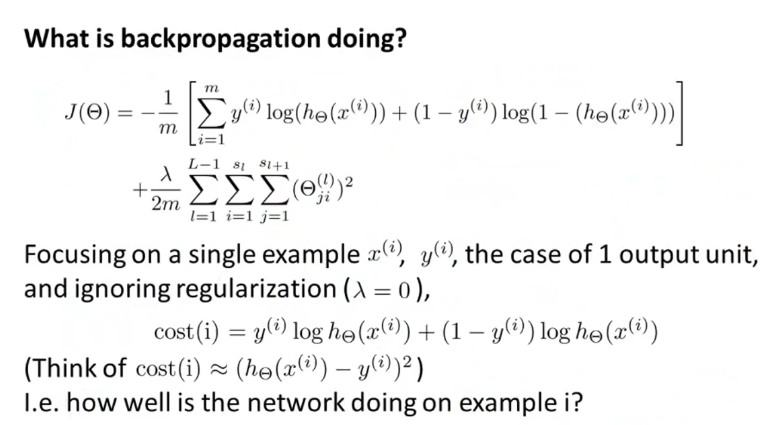
Y si miras lo que éste cálculo realmente está haciendo, enfocándonos en esta unidad oculta digamos que tenemos este peso, que se muestra en magenta, es mi peso «theta» 2(1)0 la indexación no es importante, y de esta manera aquí, que supongo estoy resaltando en rojo, eso es «theta» 2(1) 1 y este peso de aquí, que estoy dibujando en verde, en turquesa, es «theta» 2(1) 2, entonces, la forma en la que se calcula el valor de z(3) 1 es z(3) 1 que es igual a este peso en magenta, tantas veces este valor, así que «theta» 2(1) 0 veces 1, y entonces + este peso en rojo por veces este valor, y así, eso es «theta» 2(1) 1 veces a(2) 1, y finalmente este turquesa rojo tantas veces este valor, que es, por lo tanto, + «theta» 2(1) 2 veces a(2) 1.



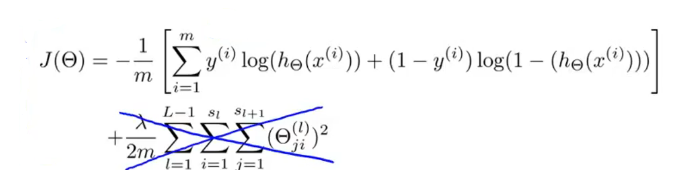
Y bien, eso es la propagación hacia adelante.

Y resulta que, como veremos más adelante en este video, lo que la retropropagación está haciendo es, realizar un proceso muy similar a esto, excepto que en lugar de los cálculos fluyan de izquierda a derecha en esta red, el flujo de cálculos va de derecha a izquierda en la red y utiliza un cálculo muy similar a este, y diré en dos diapositivas más exactamente lo que quiero decir con esto.

Para entender mejor lo que está haciendo la retropropagación, vamos ver la función de costo.



Sólo es la función de costo que teníamos para cuando tuviéramos sólo una unidad de salida Si tenemos más de una unidad de salida, tendríamos un sumatorio, ya sabes, mayor a las unidades de salida indexadas mediante k y no solo i, pero si sólo hay una unidad de salida entonces esta es una función de costos y hacemos una propagación hacia adelante y una retropropagación en un ejemplo a la vez. Así que, vamos a centrarnos en un único ejemplo x(i)y(i) y enfoquémonos en el caso en el que se tiene una unidad de salida como y(i) aquí que sólo es un número real, e ignoremos la regularización, y así lambda que sea igual a cero y el término final de regularización desaparece:



Ahora, si observamos esta suma total, descubrirás que el término del costo asociado con el ejemplo de entrenamiento “i-nésimo”, es decir el costo asociado con el ejemplo de entrenamiento x(i)y(i), que va a ser dado por esta expresión, en el que el costo es, digamos, del ejemplo de entrenamiento i que está escrito así.

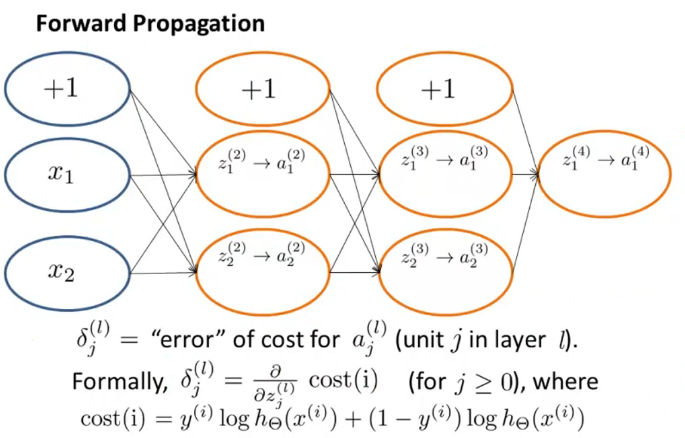


Y lo que esta función de costo hace es que juega un papel similar al error cuadrático. Bueno, en lugar de ver esta complicada expresión, si quieres puedes considerar el coste de i para que sea aproximadamente, la diferencia cuadrada entre o las salidas de la red neuronal versus lo que es el valor real:



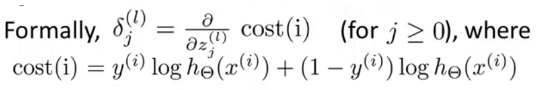
Al igual que en la regresión logística en realidad podemos preferir usar esta función de costo ligeramente más complicada utilizando el registro, aunque para el propósito de la intuición, tómate la libertad de pensar en la función de costos como una clase de error cuadrático de la función de costo, y entonces este coste de i mide cuán bien lo está haciendo la red para predecir correctamente el ejemplo i. ¿Cuán cerca está la salida al valor real asignado a y(i) ?

Echemos un vistazo a lo que está haciendo la retropropagación.



Una intuición útil es que la retropropagación está calculando éstos términos «delta» superíndice l subíndice j, y podemos pensar en ellos como el "error" del valor de activación que obtuvimos para la unidad j en la capa, en la capa “l-nésima”.

De manera más formal, y esto quizá es sólo para aquellos de ustedes que están familiarizados con el cálculo, lo que los términos de «delta» son en realidad es lo siguiente:



Son una derivada parcial con respecto a zj (l), es decir esta suma ponderada de las entradas que estamos calculando para los términos de z, la derivada parcial con respecto a estos elementos de la función de costo.

Bien, en concreto, la función de costo es una función del valor asignado a "y" y del valor, de la salida h de x mediante nuestra red neuronal. Y si pudiéramos ir hacia adentro de la red neuronal y simplemente cambiar estos valores de zj(l) j un poco, entonces eso afectaría a estos valores que salen de la red neuronal Y así, eso terminará por cambiar la función de costo.

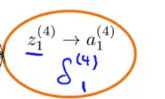
Y de nuevo, esto en realidad sólo es para aquellos de ustedes que sean expertos en cálculo.

Si estás familiarizado, o si te sientes cómodo con las derivadas parciales. ¿Qué son estos términos de «delta»?, son, resulta que son la derivada parcial de la función de costos con respecto a estos términos intermedios que estamos calculando.

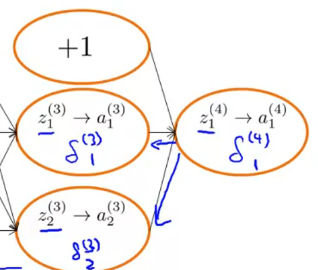
Así que, ellos son la medición de cuánto nos gustaría cambiar los pesos de la red neuronal para afectar estos valores intermedios del cálculo, de modo que la afectación de la salida final de la red neuronal "h" de "x" afecta, por lo tanto, los costos totales. En el caso de esta última parte de esta intuición de la derivada parcial, en caso de que esto no haya tenido sentido, no te preocupes por eso, el resto podemos hacerlo sin realmente hablar de derivadas parciales pero vamos a ver con más detalle qué está haciendo la retropropagación.

Para la capa de salida, primero se define este término de «delta», decimos «delta» 4(1), como y(i) si es que estamos haciendo una propagación hacia adelante y una retropropagación de este ejemplo de entrenamiento. Dice que es y(i) menos a(4) 1, así que es realmente el error, es la diferencia entre el valor real de "y" menos lo que era el valor predicho. Así que, vamos a calcular «delta» 4(1) o algo similar.

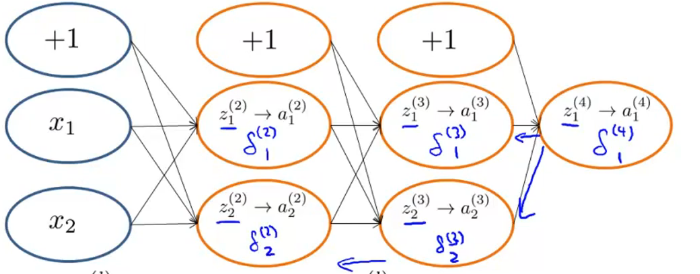




A continuación vamos a propagar estos valores hacia atrás. Explico esto en un momento y termino de calcular los términos «delta» de la capa anterior. Vamos a concluir con «delta» 3(1); «delta» 3(2);

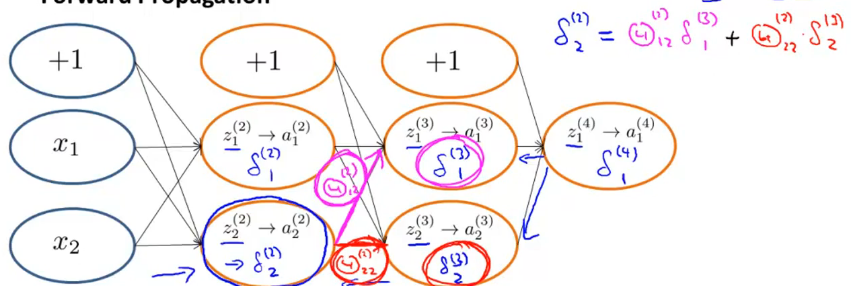


Y luego vamos a propagar esto más hacia atrás y a terminar de calcular «delta» 2(1) y «delta» 2(2).



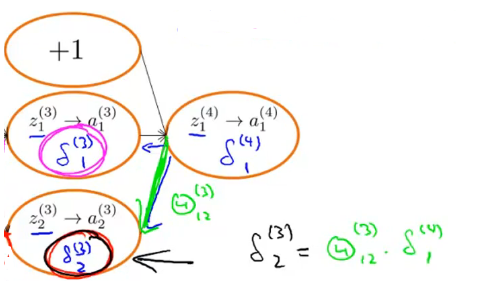
Ahora, el cálculo de la retropropagación es como hacer correr el algoritmo de propagación hacia adelante, pero a la inversa.

Bien, esto es lo que quiero decir. Echemos un vistazo a cómo hemos llegado a el valor de «delta» 2(2) Y entonces, tenemos que «delta»2(2) es similar a la propagación hacia adelante, déjenme asignar valores a un par de los pesos. Así que este peso debería estar en turquesa, digamos ese peso es «theta» 2 de 1, 2 y este peso aquí abajo, permítanme resaltar esto en rojo. Esto va a ser, digamos, «theta» 2 de 2,2. Así que si vemos cómo se calcula «delta» 2(2). ¿Cómo se calcula para esta notación? Pues resulta que lo que vamos a hacer es que vamos a tomar este valor y a multiplicarlo por este peso y a agregarlo a este valor multiplicado por ese peso. Bien, en realidad es una suma ponderada de las nuevos, de estos valores nuevos de «delta», ponderados por la fuerza del contorno correspondiente. Bueno, en concreto, Esta «delta» 2,2 va a ser igual a «theta» 2(1)2, que es ese peso en magenta, veces «delta» (3)1 + y entonces lo que tengo en rojo es «theta» 2(2) 2 veces «delta» 3(2) Así que, en realidad es, literalmente este peso en rojo veces este valor + este peso en magenta veces este valor y así es cómo terminamos con ese valor de «delta».



Y nada más como otro ejemplo, echemos un vistazo a este valor. ¿Cómo hemos llegado a delta2(3)? Bueno, es un proceso similar,

Si este peso, que voy a resaltar en verde, es igual a, digamos, «delta» 3(1) 2, entonces tenemos que «delta» 3(2) será igual a ese peso en verde, «theta» 3(1)2 veces «delta» 4(1).



Por cierto, hasta ahora he estado escribiendo sólo los valores de «delta» para las unidades ocultas y, aunque excluyo las unidades de oscilación. Dependiendo de cómo definas el algoritmo de retropropagación o dependiendo de cómo lo implementes, ya sabes, puedes terminar implementando algo que calcule los valores de «delta» para estas unidades de oscilación también.

La unidad de oscilación son siempre los valores de salida igual a 1 y son justo lo que son y no hay forma de que cambiemos su valor, así que, dependiendo de en tu implementación de la retropropagación, la manera en la que yo usualmente lo pongo en práctica, lo concluyo calculando estos valores de «delta», aunque sólo los desechamos y no los utilizamos, porque no terminan siendo parte del cálculo necesario para calcular las derivadas. Bien, pues ojalá esto te dé un poco de intuición acerca de lo que hace la

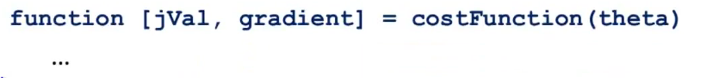
En caso de que todo esto todavía te parezca muy mágico y una especie de caja negra, en un video posterior, en el video de resumen, voy a tratar de dar un poco más de intuición sobre lo está haciendo la retropropagación. Pero, por desgracia, este es, ya sabes un algoritmo que es difícil tratar de visualizarlo y de comprender qué es lo que realmente está haciendo. Aunque por fortuna, ya sabes, a menudo, supongo claro, muchas personas lo han estado usando con mucho éxito durante muchos años y si infieres el algoritmo, puedes tener un algoritmo de aprendizaje muy eficaz, incluso si resulta difícil visualizar las funciones internas que expliquen exactamente cómo funciona.

## Retropropagación en la práctica

## Desenrollando parámetros

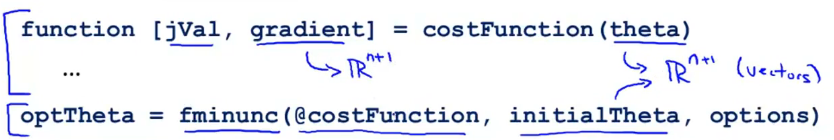
En el video anterior, hablamos acerca de cómo utilizar la retropropagación para calcular las derivadas de tu función de costo. En este video quiero decirte rápidamente un detalle para implementar el desarrollo de tus parámetros a partir de las matrices hacia vectores, las cuales necesitamos para poder utilizar las rutinas de optimización avanzada.

Específicamente, digamos que has implementado una función de costo que toma esta entrada; es decir, los parámetros de «theta» y retorna la función de costo (jVal) y retorna las derivadas (gradient).

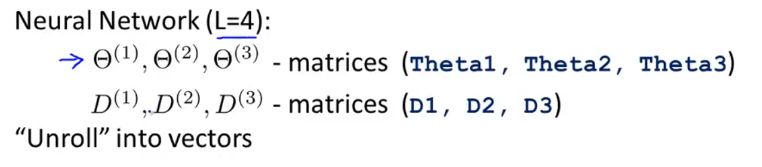


Después puedes pasar esto hacia un algoritmo de optimización avanzada mediante fminunc y fminunc no es el único, por cierto. También existen otros algoritmos de optimización avanzada.

 Pero todo lo que ellos hacen es tomar esta entrada, una función explícita para costos y algún valor inicial de «theta». Estas rutinas suponen que «theta», y el valor inicial de «theta», que son vectores del parámetro, tal vez Rn o Rn + 1. Aunque estos son vectores y también se asume que, bueno ya sabes, tu función de costo retornará como un segundo valor de retorno. Este gradiente, que también es Rn y Rn + 1 que también es un vector.



Esto funcionó bien cuando estábamos usando una regresión logística pero ahora que estamos usando una red neuronal nuestros parámetros ya no son vectores, pero en vez de ello son estas matrices:

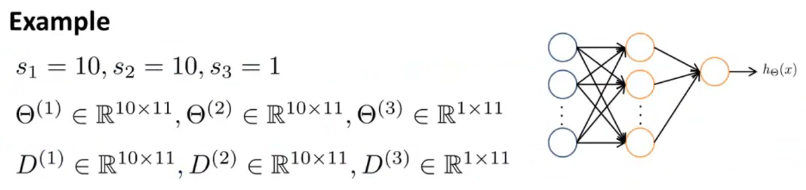


Las cuales para una red neuronal completa tendríamos que son matrices de parámetro para «theta» 1, «theta» 2, «theta» 3 que podríamos representar en Octave como estas matrices «theta»1, «theta»2, «theta» 3. Y del mismo modo los términos de gradiente que se espera que retornen.

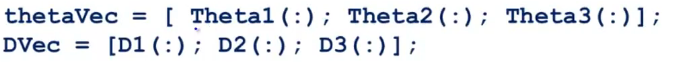
Bien, en el video anterior, mostramos cómo calcular estas matrices de gradiente, que eran D mayúscula 1, D mayúscula 2, D mayúscula 3, que podríamos representar en Octave como las matrices D1, D2, D3.

En este video quiero contarte rápidamente algo acerca de la idea de cómo tomar estas matrices y desarrollarlas en vectores. Entonces, terminan por estar en un formato adecuado para pasarlas a «theta», aquí, o para obtener un gradiente acá.

Digamos, en concreto, que tenemos una red neuronal con una capa de entrada con diez unidades, una capa oculta con diez unidades y una capa de salida con sólo una unidad:

 Entonces s1 es el número de unidades en la capa uno y s2 es el número de unidades en la capa dos y s3 es un número de unidades en la capa tres. En este caso, la dimensión de tus matrices «theta» y D va a ser dada mediante estas expresiones. Por ejemplo, «theta» uno va hacia una matriz de 10 por 11 y así sucesivamente.

Entonces, en Octave si quieres convertirlas estas matrices y vectores. Lo que puedes hacer es tomar tus «theta» 1, «theta» 2, «theta» 3 y escribir esta parte del código:



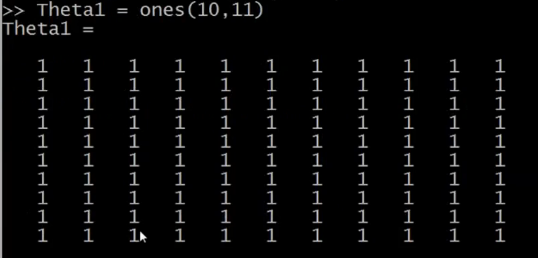
Y éste tomará todos los elementos de tus tres matrices «theta» y tomará todos los elementos de «theta» uno, todos los elementos de «theta» 2, todos los elementos de «theta» 3, desarrollarlos y poner todos los elementos en un enorme y largo vector. Que es thetaVec y asimismo el segundo comando tomaría todas tus matrices D y las desarrolla en una enorme y largo vector y lo llamamos DVec.

Para finalizar, si quieres regresar de las representaciones vectoriales a las representaciones de la matriz. ¿Qué haces para regresar a «theta» uno?, digamos que, pues tomas thetaVec y lo extraes de los primeros 110 elementos. Y así, «theta» 1 tiene 110 elementos porque es es una matriz de 10 por 11, de modo que extraes los primeros 110 elementos después usted puede, utilizar el comando “*reshape*” para remodelar esos regresos hacia «theta» 1.



Y del mismo modo, para conseguir que regrese «theta» 2 extraes los siguientes 110 elementos y los remodelas. En cuanto a «theta» 3, extraes lo once elementos finales y ejecutas la remodelación para recuperar «theta» 3.

Aquí está una demostración rápida en Octave de ese proceso. Entonces, para este ejemplo pongamos «theta» 1 igual para que sean "unos" de 10 por 11, así que es una matriz de puros unos



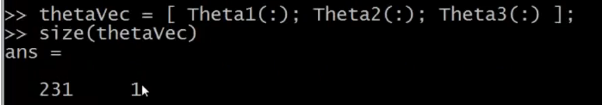
Y para hacer esto más fácil, veamos, pongamos que sean 2 veces unos, 10 por 11:



y entonces digamos que «theta» 3 es igual a 3 veces los unos de 1 por 11. Así que esto es 3 matrices por separado: «theta» 1, «theta» 2, «theta» 3.



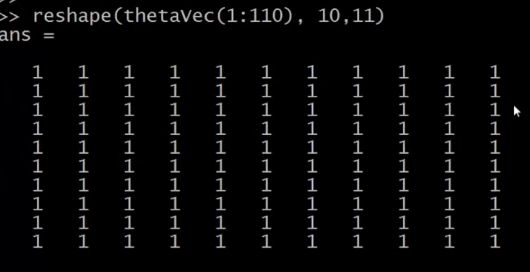
Queremos poner todo esto como un vector. thetaVec es igual a «theta» 1; «theta» 2. «theta» 3. ¿Correcto? Eso es, punto y coma, y dos puntos en medio, así y ahora thetavec va a ser un vector muy largo. De 231 elementos.



Si lo despliego, descubro que es un vector muy largo con todos los elementos de la primera matriz, todos los elementos de la segunda matriz, y todos los elementos de la tercera matriz en un vector columna.



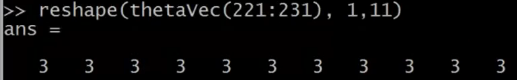
Y si quiero regresar a mis matrices originales, puedo remodelar thetaVec. Vamos a extraer los primeros 110 elementos y a remodelarlos en una matriz de 10 por 11. Esto me da de vuelta «theta» 1.



Y si luego extraigo los siguientes 110 elementos. Entonces, eso se indexa de 111 a 220. Recupero todos mis 2.

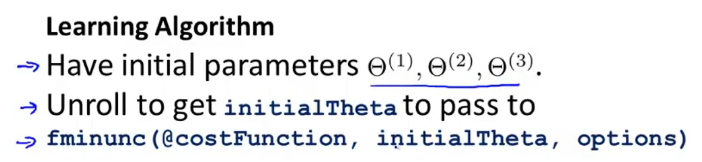


Y si voydesde 221 hasta el último elemento, que es el elemento 231 y lo remodelo 1 por 11, recupero «theta» 3.

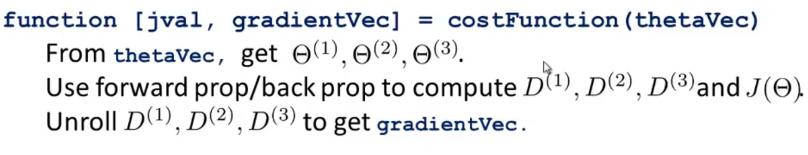


Para hacer este proceso muy concreto, he aquí cómo usamos el desarrollo de la idea para implementar nuestro algoritmo de aprendizaje.

Digamos que tienes algún valor inicial de los parámetros «theta»1, «theta» 2, «theta» 3. Lo que haremos es tomar estos y desarrollarlos en un vector largo vamos a recuperar el «theta» inicial pasarlo a fminunc como esta configuración inicial de los parámetros de «theta».

 El otro asunto que necesitamos poner en marcha es implementar la función de costo. Aquí está mi implementación de la función de costo.

La función de costo nos va a dar la entrada, thetaVec, que va a ser todos los vectores de mi parámetro que, están en, la forma que ha sido desarrollada en un vector.



Entonces, lo primero que voy a hacer es utilizar thetaVec y voy a usar las funciones de remodelación. Así podré extraer los elementos de thetaVec y utilizar el remodelado para recuperar mis matrices originales de parámetro, «theta» 1, «theta» 2, «theta» 3. Así que, estas van a ser las matrices que voy a obtener. Bien, esto me da una forma más conveniente en la cual utilizar estas matrices, de modo que, puedo ejecutar la propagación hacia adelante y la retropropagación para calcular mis derivadas, y para calcular mi función de costo j de «theta».

Y finalmente, puedo entonces tomar mis derivadas y desarrollarlas para mantener los elementos en el mismo orden como lo hice cuando desarrollé mis «theta»s. Pero, voy a desarrollar D1, D2, D3, para obtener gradientVec que es ahora a donde puede retornar mi función de costo. Puede retornar un vector de estas derivadas.

Pues bien, espero que ahora tengan un sentido claro de cómo convertir de ida y de vuelta entre la representación matricial de los parámetros en comparación con la representación del vector de los parámetros.

La ventaja de la representación de la matriz es que cuando tus parámetros se almacenan como matrices es más conveniente cuando estás haciendo una propagación hacia adelante y una retropropagación y es más fácil cuando tus parámetros se almacenan como matrices que puedes aprovechar, en cierto modo, de las implementaciones vectorizadas.

Mientras que en contraste, la ventaja de la representación vectorial, cuando tienes, por ejemplo, thetaVec o DVec es que cuando estás utilizando los algoritmos de optimización avanzada, esos algoritmos tienden a suponer que tienes todos tus parámetros desarrollados en un enorme y largo vector. Así que, sólo con lo que acabamos de ver esperamos que ahora tú puedas rápidamente hacer conversiones entre estas dos formas según lo necesites.

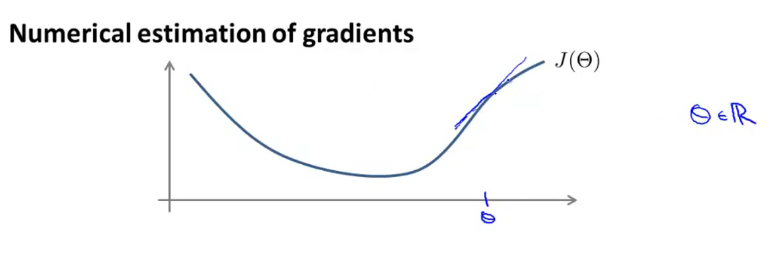
## Comprobando el gradiente

En los últimos videos, hemos hablado acerca de cómo realizar una propagación hacia delante y una retropropagación en una red neuronal con el fin de procesar los derivados.

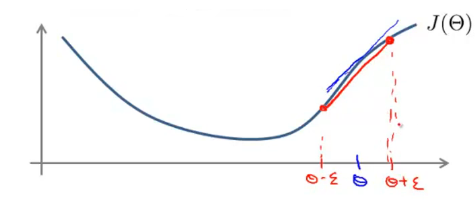
Sin embargo la retropropagación como un algoritmo tiene muchos detalles y, como saben, puede ser un poco difícil de implementar. Y una propiedad desafortunada es que existen muchas formas de tener errores imperceptibles en la retropropagación de forma que si los ejecutan con el gradiente de descenso o algún otro algoritmo de optimización, podría parecer que en realidad está funcionando. Y, como saben, la función de costos J de «theta» puede terminar disminuyendo en cada iteración en el gradiente de descenso, pero esto podría llevarse a cabo a pesar de que puede haber algún error en la implementación de la retropropagación. Así que parece que J de «theta» está disminuyendo, sin embargo podrían terminar con una red neuronal que tiene un nivel más alto de errores de lo que tendrían con una implementación libre de errores y simplemente no sabrían que este error imperceptible estuvo ocasionándoles este desempeño.

Entonces, ¿qué podemos hacer al respecto? Existe un concepto llamado comprobación de gradiente la cual elimina casi todos estos problemas. Entonces hoy, cada vez que implemente una retropropagación o un algoritmo de gradiente de descenso similar en la red neuronal o en cualquier otro modelo de una complejidad razonable, siempre implementaré la comprobación de gradiente. Y si ustedes lo llevan a cabo les ayudará a comprobar y en cierto modo a tener mayor confianza de que la implementación de la propagación hacia delante y la retropropagación o lo que sea, es 100% correcta. Y de acuerdo a lo que he visto esto elimina prácticamente todos los problemas relacionados con el tipo de implementación errónea de la retropropagación. Y en los videos anteriores, en cierta forma les pido que tengan confianza en que las fórmulas que les he dado para procesar los «delta»s, y las "D" y así sucesivamente, les pido que tengan confianza en que realmente procesan las gradientes de la función de costos, pero una vez que ustedes implementan la comprobación del gradiente numéricamente, que es el tema de este video, podrán ser capaces de verificar por sí mismos que el código que escriban en efecto está procesando la derivada de la función de costos de "J".

Entonces, este es el concepto. Consideren el siguiente ejemplo. Supongan que tengo la función J de «theta», y tengo algún valor, «theta», y para este ejemplo, voy a asumir que «theta» es sólo un número real. Y digamos que deseo calcular la derivada de esta función en este punto. Y así la derivada es, como saben, igual a la pendiente de ese tipo de línea de la tangente.

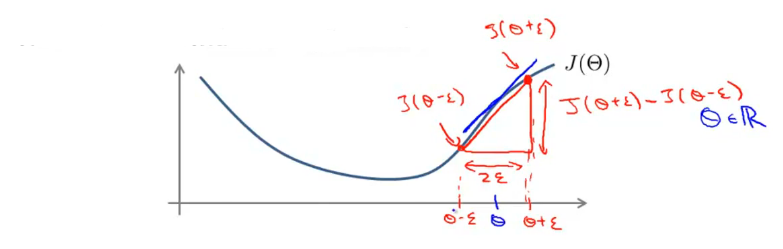


Así es como voy a aproximar en forma numérica la derivada, o más bien este es un procedimiento para aproximar en forma numérica la derivada: Voy a procesar «theta» más «épsilon», de forma que el valor está un poco a la derecha. Y vamos a procesar «theta» menos «épsilon». Y voy a, observen estos dos puntos y conéctenlos por medio de una línea recta.



Y voy a conectar estos dos puntos por medio de una línea recta y voy a utilizar la pendiente de esta pequeña línea roja como mi aproximación a la derivada que es, la verdadera derivada es la pendiente de esa línea azul ahí. De forma que, como saben, parece que sería una muy buena aproximación.

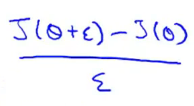
Matemáticamente, la pendiente de esta línea roja es esta altura vertical, dividida por esta anchura horizontal, así que este punto en la parte superior es J de «theta» más «épsilon» Este punto aquí es J de «theta» menos «épsilon». De forma que esta diferencia vertical es J de «theta» más «épsilon», menos J de «theta», menos «épsilon» y esta distancia horizontal es sólo 2 «épsilon».



Así que, mi aproximación va a ser la derivada, con respecto a «theta» de J de «theta»-- añadan este valor de «theta»-- que eso es aproximadamente J de «theta» más «épsilon», menos J de «theta», menos «épsilon», sobre 2 «épsilon».

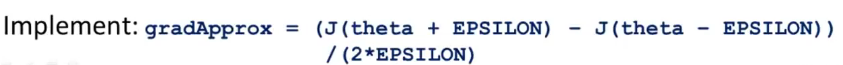


Por lo general, utilizo un valor muy pequeño para «épsilon» y establecemos que «épsilon» tal vez se encuentre en el orden de 10 a menos 4. Por lo general existe un rango amplio de distintos valores para «épsilon» que funcionan muy bien. De hecho, si ustedes dejan que «épsilon» sea demasiado pequeño entonces, matemáticamente, este término de aquí de hecho, matemáticamente, como saben, se convierte en la derivada, se vuelve con exactitud en la pendiente de la función en este punto. Es sólo que no queremos utilizar «épsilon», que es demasiado, demasiado pequeño porque entonces podríamos encontrarnos con problemas numéricos. Entonces, como saben, por lo general utilizo «épsilon» alrededor de 10 a menos 4, digamos. Y por cierto que algunos de ustedes pueden haber visto una fórmula alternativa para estimar la derivada que es esta fórmula.



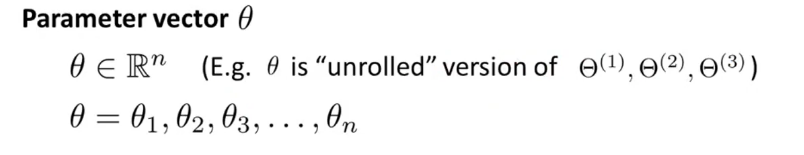
Se llama de diferencia unilateral. Mientras que la fórmula de la izquierda se llama de diferencia bilateral. La diferencia bilateral nos proporciona una estimación ligeramente más precisa, de forma que por lo general utilizo esa en lugar de sólo esta estimación de diferencia unilateral.

Así, en concreto, lo que implementan en Octave es la implementación de lo siguiente:

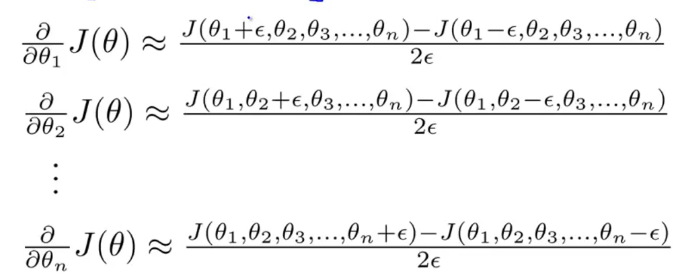


Implementan el proceso de aproximación de la gradiente que va a ser tan sólo una aproximación a la derivada, como saben, en esta fórmula: J de «theta» más «épsilon», menos J de «theta», menos «épsilon», dividido por dos veces «épsilon». Y esto nos proporcionará una estimación numérica de la gradiente en ese punto. Y en este ejemplo parece ser una estimación bastante buena.

Ahora, en el ejemplo anterior consideramos el caso de cuando «theta» es un número real. Ahora veamos un caso más general donde «theta» es un vector de parámetros. Así que digamos que  y pudiera ser una versión “desenrrolada” de los parámetros de nuestra red neuronal.



De modo que «theta» es un vector que tiene "n" elementos, «theta» 1 hasta «theta» n. Entonces podemos utilizar un concepto similar para aproximar todos los términos de la derivada parcial:

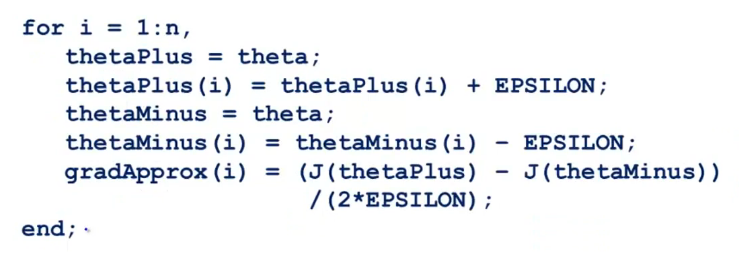


En concreto, la derivada parcial de la función de costos con respecto al primer parámetro de «theta» 1, esa puede obtenerse al tomar J e incrementar «theta» 1. Entonces tienen J de «theta» 1 más «épsilon» y así sucesivamente menos J de esta «theta» 1 menos «épsilon» y dividida por 2 «épsilon».

La derivada parcial respecto al segundo parámetro «theta» 2, es de nuevo esto, excepto que ustedes toman J de-- aquí están incrementando «theta» 2 por «épsilon». Y aquí ustedes están disminuyendo «theta» 2 por «épsilon». Y así hasta llegar a la derivada parcial con respecto a «theta» n. Lo que sería si ustedes incrementan y disminuyen «theta» n por «épsilon» allí.

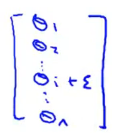
Así, estas ecuaciones les proporcionan una forma para aproximar en forma numérica la derivada parcial de "J" con respecto a cualquiera de los parámetros que se derivan.

En concreto, lo que implementarán es por lo tanto, lo siguiente. Implementamos lo siguiente en Octave para ejecutar en forma numérica las derivadas:



Decimos que i es igual a 1 hasta n donde "n" es la dimensión de nuestro parámetro vector «theta». Y por lo general hago esto con la versión desarrollada de los parámetros. De forma que «theta» es sólo una lista larga de todos mis parámetros en mis redes neuronales.

Voy a establecer que «thetaPlus» es igual a «theta», a continuación incrementen «theta» más el elemento «i-nésimo» por ««épsilon»». Y esto es básicamente, «thetaPlus» que es igual a «theta» excepto para «thetaPlus» i, que ahora está incrementada por «épsilon». De forma que si «thetaPlus» es igual «theta» 1, «theta» 2 y así sucesivamente y luego «theta» i tiene un «épsilon» añadido, y luego sigue hasta «theta» n. Así que en esto consiste «thetaPlus».

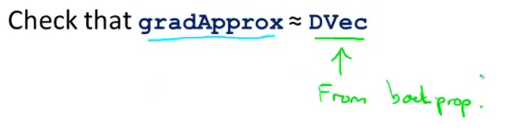


Y del mismo modo estas dos líneas establecen a «thetaMinus» en algo similar excepto que esto, en lugar de «theta» i más «épsilon», esto ahora se convierte en «theta» i menos «épsilon».



Y entonces finalmente, implementamos esta aproximación de la gradiente i, y esto les dará la aproximación a la derivada parcial con respecto a «theta» i de J de «theta».

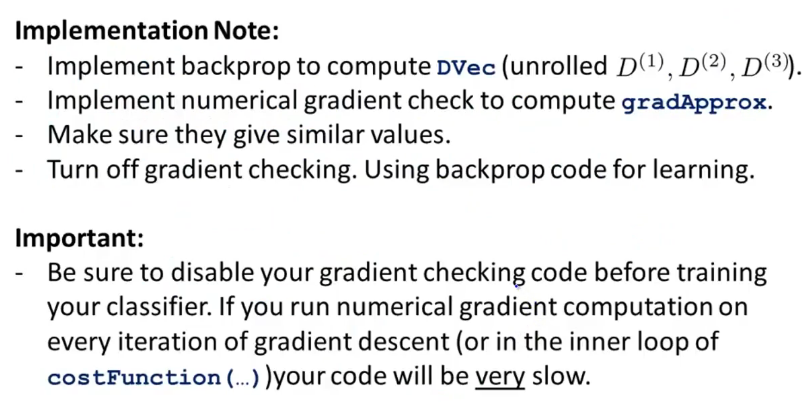
Y la forma en que utilizamos esto en nuestra complementación de la red neuronal es que queremos poner en práctica esto, implementar estos cuatro bucles para procesar, como saben, la derivada parcial de la función de costos con respecto a cada parámetro en nuestra red. Y podemos tomar el gradiente que obtuvimos de la retropropagación. Así, los vectores DVec fueron las derivadas que obtuvimos de la retropropagación.



Así la retropropagación, es una forma relativamente eficiente para procesar las derivadas o las derivadas parciales de una función de costos con respecto a todos nuestros parámetros. Y lo que por lo general hago es que tomo mi derivada procesada en forma numérica, que es esta aproximación del gradiente que acabamos de obtener de aquí arriba y nos aseguramos de que esa sea igual o aproximadamente igual a, como saben, a los valores pequeños del redondeo numérico que están bastante cercanos a los vectores DVec que obtuve de la retropropagación. Y si estas dos formas de procesar la derivada me dan la misma respuesta o al menos me dan respuestas muy parecidas, como saben, hasta unas plazas decimales, Entonces tengo confianza de que es correcta mi implementación de retropropagación.

Y cuando conecto estos vectores DVec a la gradiente de descenso o a algún algoritmo de optimización avanzada, puedo entonces estar mucho más seguro de que estoy calculando las derivadas correctamente y por lo tanto, con suerte mis códigos funcionarán correctamente y harán un buen trabajo optimizando J de «theta».

Finalmente, deseo resumir todo y decirles cómo implementar esta comprobación de la gradiente numérica. Esto es lo que suelo hacer.



La primer cosa que hago es implementar la retropropagación para procesar Dvec. También, este es un procedimiento del cual hablamos en un video anterior para procesar el DVec que puede ser nuestra versión desarrollada de estas matrices.

Entonces, seguidamente lo que hago es implementar una comprobación de la gradiente numérica para calcular la aproximación de la gradiente. Así que esto es lo que he descrito antes en este video, en la diapositiva anterior.

Después deberán asegurarse de que el DVec y la aproximación de la gradiente den valores similares, como ya saben, digamos que hasta unos cuantas plazas decimales.

Y finalmente, y este es el paso importante, entre más empiecen a utilizar su código para el aprendizaje, para entrenar en serio su red, es importante desactivar la comprobación de gradiente. Y para no procesar ya esta aproximación de la gradiente utilizando las fórmulas de las derivadas numéricas de las que hablamos anteriormente en este video. Y la razón de esto es que-- el código numérico, el código de la comprobación de gradiente, las cosas de las que hablamos en este video-- es muy caro computacionalmente, esa es una forma muy lenta de tratar de aproximar la derivada.

En contraste con el algoritmo de retropropagación del que hablamos antes, que es de lo que hemos hablado antes para procesar, como saben, D1, D2, D3 o para DVec. La retropropagación es una forma mucho más eficiente computacionalmente de procesar las derivadas. De forma que una vez que hayan verificado que la implementación de su retropropagación es correcta, deberán desconectar la comprobación de la gradiente y sólo dejar de utilizarla.

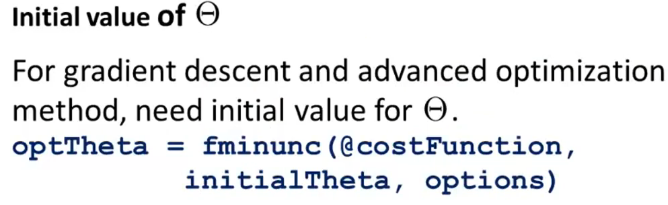
Sólo para recapitular, deberán estar seguros de deshabilitar el código de comprobación de la gradiente antes de ejecutar su algoritmo para muchas iteraciones de la gradiente de descenso, o para muchas iteraciones de los algoritmos de optimización avanzada con el fin de entrenar su clasificador. Específicamente, si fueran a ejecutar la comprobación de la gradiente numérica en cada iteración de la gradiente de descenso, o si estuvieran en el bucle interno de la función de costos, entonces el código sería muy lento. Porque el código de la comprobación de la gradiente numérica es mucho más lento que el algoritmo de retropropagación, que el método de retropropagación donde si recuerdan estuvimos procesando «delta» 4, «delta» 2, «delta» 2 y así sucesivamente. Ese fue el algoritmo de retropropagación. Esa es una forma mucho más rápida de procesar las derivadas de la comprobación de la gradiente. Entonces cuando estén listos, una vez que han verificado que la implementación de la retropropagación es correcta, asegúrense de desactivar o deshabilitar el código de comprobación de la gradiente mientras entrenan su algoritmo o de lo contrario el código funcionará de manera muy lenta.

Entonces así es como obtienen las gradientes en forma numérica Y así es cómo ustedes pueden verificar que la implementación de la retropropagación es correcta. Cuando implemento la retropropagación o un algoritmo de gradiente de descenso similar para un modelo complicado, siempre utilizo la comprobación de gradiente. Esto realmente me ayuda a estar seguro de que mi código es correcto.

## Random initialization

En los videos anteriores, reunimos casi todas las piezas que ustedes necesitan para implementar y entrenar una red neuronal. Esta es una última idea que tengo que compartir con ustedes, la cual es la idea de la inicialización aleatoria.

Cuando están ejecutando un algoritmo como el gradiente de descenso o también los algoritmos de optimización avanzada es necesario que elijamos un valor inicial para los parámetros «theta». Así que para el algoritmo de optimización avanzada, como saben, se asume que ustedes pasarán algún valor inicial para los parámetros «theta».

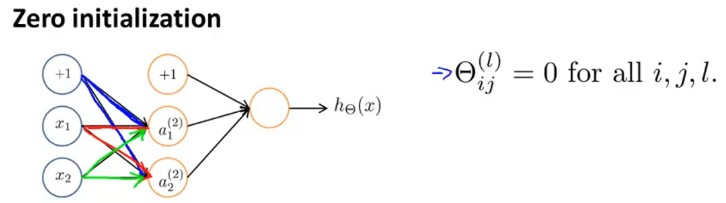


Ahora consideremos el gradiente de descenso. Para eso, como saben, también tenemos que inicializar «theta» con algo. Y entonces podemos gradualmente seguir los pasos para ir hacia abajo, utilizando el gradiente de descenso, para ir hacia abajo a fin de minimizar la función J de «theta».

Entonces, ¿Cómo podemos establecer el valor inicial de «theta»? Es posible establecer el valor inicial de «theta» al vector de todos cero.

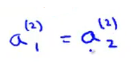
Considerando que esto funcionó bien cuando estuvimos utilizando la regresión logística. Inicializar todos los parámetros en cero en realidad no funciona cuando se está entrenando una red neuronal.

Consideren el entrenamiento de la siguiente red neuronal. Y digamos que hemos inicializado todos los parámetros de la red a cero.



Y si lo llevan a cabo entonces lo que eso significa es que en la inicialización este peso que estoy marcando en azul va a ser igual a ese peso. Por lo tanto, ambos son cero. Y este peso que estoy marcando en rojo, es igual a ese peso. Están en rojo. Y también este peso, el cual estoy marcando con verde va a ser igual al valor de ese peso. Y lo que eso significa es que ambas unidades ocultas: a1 y a2 estarán ejecutando la misma función de sus entradas.

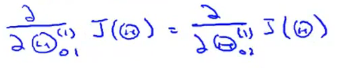
Y así, terminarán en cada uno de sus ejemplos de entrenamiento terminarán con a (2)1 que es igual a a(2)2.



Y además no voy a mostrar esto en mucho detalle, sin embargo debido a que estos pesos salientes son iguales, ustedes pueden demostrar que los valores «delta» también serán los mismos. Así concretamente, terminarán con «delta» 11, «delta» 2 1, que es igual a «delta» 2 2.



Y si trabajan con el mapa más a fondo, lo que pueden demostrar es que las derivadas parciales con respecto a sus parámetros cumplirán con lo siguiente. Que la derivada parcial de la función de costo con respecto a( Estoy anotando las derivadas respecto a estos pesos en azul en sus redes). Descubrirán que estas dos derivadas parciales serán iguales entre sí.

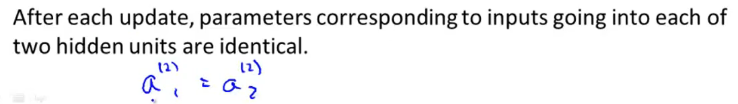


Y entonces, lo que esto significa es que incluso después de una actualización de gradiente de descenso. Ustedes actualizarán, digamos que el primer peso en azul con, como saben, los ciclos de velocidad de aprendizaje, que es esto. Y actualizarán el segundo peso en azul para la suma de los ciclos de velocidad de aprendizaje, esto. Pero lo que esto significa es que incluso después de la actualización de un gradiente de descenso, esos dos pesos en azul, esos dos parámetros de color azul terminarán siendo iguales entre sí.



Así que ahora serán algo distintos al valor cero, pero los valores serán iguales. Y del mismo modo, incluso después de la actualización del gradiente de descenso. Y para los otros casos pasrá algo parecido. Habrá algunos valores distintos a cero. Sólo que los dos valores en rojo serán iguales entre sí. Y de forma similar los dos pesos en verde, cambiarán de valor pero ambos terminarán teniendo el mismo valor entre sí.

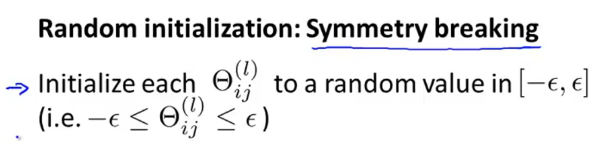
Así que después de cada actualización, los parámetros que corresponden a las entradas que van a cada de las dos unidades ocultas serán idénticos. Es decir que los dos pesos en verde siguen siendo iguales, los dos pesos en rojo siguen siendo iguales, los dos pesos en azul siguen siendo iguales y lo que eso significa es que aún después de una iteración de, digamos, la gradiente de descenso, descubrirán que las dos unidades ocultas siguen ejecutando exactamente las mismas funciones que la entrada. Así que todavía tienen a(1)2 que es igual a a (2)2. Así que han vuelto a este caso. Y como siguen ejecutando la gradiente de descenso



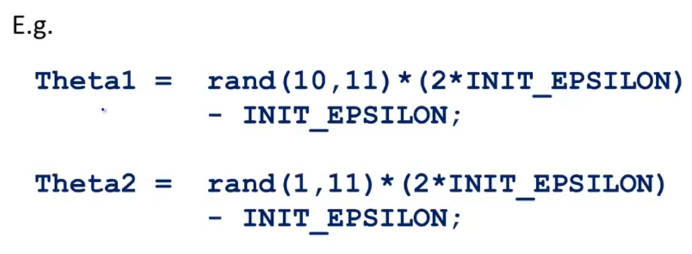
Los pesos en azul, los pesos en azul permanecerán iguales entre sí. Los dos pesos en rojo permanecerán iguales entre sí. Los dos pesos en verde permanecerán iguales entre sí. Y lo que esto significa es que su red neuronal en realidad no puede ejecutar funciones muy interesantes. Imaginen que no sólo tienen dos unidades ocultas sino que tienen muchas, muchas unidades ocultas. Entonces lo que esto nos indica es que todas sus unidades ocultas están ejecutando exactamente la misma variable todas sus unidades ocultas están ejecutando exactamente la misma función de entrada. Y esta es una representación altamente redundante. Porque eso significa que la unidad de regresión logística final, saben, en realidad sólo llega a ver una variable. Debido a que todas estas son iguales y esto impide que la red neuronal aprenda algo interesante.

Con el fin de evitar este problema, por lo tanto la forma en que inicializamos los parámetros de una red neuronal es con la inicialización aleatoria. En concreto, el problema que vimos en la diapositiva anterior es algunas veces llamado el problema de los pesos simétricos, es decir, si todos los pesos son iguales. Y así, esta inicialización aleatoria es la forma en que realizamos la ruptura de la simetría.

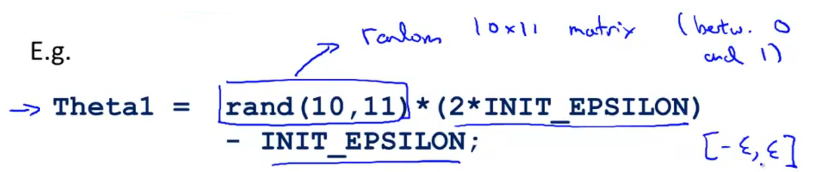
De forma que lo que hacemos es inicializar cada valor de «theta» a un número aleatorio entre menos «épsilon» y «épsilon». Entonces esta es una anotación para referirme a los números entre menos «épsilon» y mayor a «épsilon». De forma que todos los pesos para mis parámetros se van a inicializar aleatoriamente entre menos «épsilon» y mayor a «épsilon».



La forma en que escribo el código para hacer esto es en Octave,

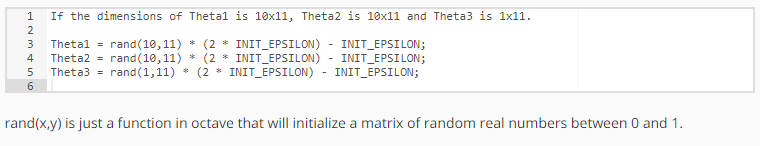


es decir, que «theta» 1 sea igual a este. Y este aleatorio 10 por 11. Así es como ejecutan una matriz dimensional aleatoria de 10 por 11, y todos los valores están entre 0 y 1. De forma que estos van a ser números reales que adquieren cualquier valor continuo entre 0 y 1. Y así, si ustedes toman un número entre 0 y 1, multiplíquenlo por 2 veces un «épsilon» y menos «épsilon», entonces terminarán con un número que se encuentra entre menos «épsilon» y «épsilon».



Y a propósito, este «épsilon» aquí no tiene nada que ver con el «épsilon» que estábamos utilizando cuando hicimos la comprobación de gradiente. De forma que cuando hicimos la comprobación numérica de gradiente, allí añadimos algunos valores de «épsilon» a «theta». Esto es, como saben, un valor no relacionado de «épsilon». Razón por la que lo estoy indicando como “init\_epsilon”, sólo para distinguirlo del valor de «épsilon» que estábamos utilizando en la comprobación de gradiente.

Y del mismo modo si desean inicializar «theta» 2 a una matriz aleatoria de 1 por 11, ustedes pueden hacerlo a través de esta pieza de código.



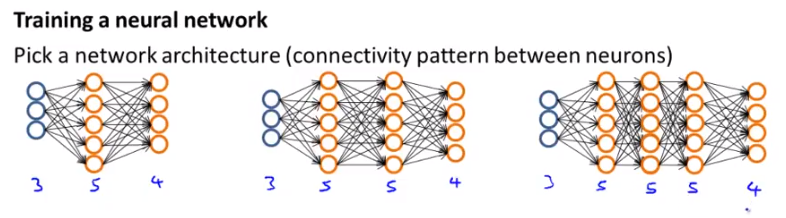
Así que, para resumir, para entrenar una red neuronal, lo que ustedes deben hacer es inicializar de forma aleatoria los pesos con, ya saben, con valores pequeños cercanos a 0, entre menos «épsilon» y mayor a «épsilon», digamos, y luego implementar la retropropagación; hacer la comprobación de gradiente y utilizar ya sea un gradiente de descenso o uno de los optimización avanzados para tratar de minimizar J de «theta» como una función de los parámetros «theta» empezando únicamente a partir de valores iniciales elegidos al aleatoriamente para los parámetros.

Y al hacer la ruptura de simetría, que es este proceso. Con suerte, la gradiente de descenso o los algoritmos de optimización avanzados serán capaces de encontrar un buen valor de «theta».

## Uniendo todo el proceso

En este video, lo que me gustaría hacer es tratar de unir todas las piezas para dar un resumen general o un panorama más amplio de cómo todas las piezas encajan y del proceso global de cómo implementar un algoritmo de aprendizaje de redes neuronales.

Al entrenar una red neuronal, lo primero que debemos hacer es elegir una arquitectura de red y por arquitectura solamente quiero decir un patrón de conectividad entre las neuronas. Así, podemos elegir entre, digamos, una red neuronal con tres unidades de entrada y cinco unidades ocultas y cuatro unidades de salida contra una de 3, 5 ocultas, 5 ocultas, 4 de salida y aquí son 3, 5, 5, 5 unidades en cada una de las tres capas ocultas y cuatro unidades abiertas, y así estas opciones de cuántas unidades ocultas en cada capa y cuántas capas ocultas, son opciones de arquitectura. Entonces, ¿cómo hacemos estas elecciones?



Bueno, primero, el número de unidades de entrada está muy bien definido. Y una vez que decidan la configuración específica de variables de «x», el número de unidades de entrada, simplemente será, digamos, la dimensión de sus variables de «x(i)» sería determinado por ello. Y si están haciendo clasificaciones multiclase, el número de salidas de esto será determinado por el número de clases en su problema de clasificación. Y sólo como recordatorio, si tienen una clasificación multiclase en donde «y» toma valores de, digamos, entre 1 y 10 para que tengan diez clases posibles. Entonces, recuerden escribir sus salidas «y» como este tipo de vectores.



Entonces, en lugar de clase uno, la recodifican como un vector de ese tipo, o para la segunda clase la recodifican como un vector de ese tipo. Así que si uno de los ejemplos toma el valor de la quinta clase, digamos, «y» es igual a 5, entonces lo que están mostrando a su red neuronal no es en realidad un valor de «y» igual a 5, sino que en la capa superior que tendría diez unidades de salida, entonces alimentarán al vector con un uno en la quinta posición. Entonces, la elección del número de unidades de entrada y el número de unidades de salida es, quizá, algo bastante sencillo.

Y en cuanto al número de unidades ocultas y al número de capas ocultas, un estándar razonable es usar una única capa oculta y así, este tipo de red neuronal mostrada a la izquierda con sólo una capa oculta es probablemente lo más común.

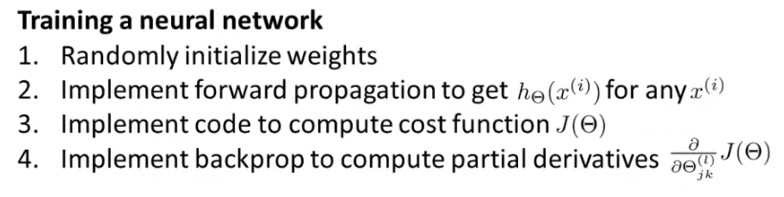
O si usan más de una capa oculta, igualmente, un estándar razonable será tener el mismo número de unidades ocultas en cada una de las capas

Y en cuanto al número de unidades ocultas, normalmente, entre más unidades ocultas haya, es mejor; sólo que, si tienen muchas unidades ocultas, puede llegar a ser computacionalmente más costoso, pero a menudo, el tener más unidades ocultas es bueno.

Normalmente, el número de unidades ocultas en cada capa será tal vez, comparable a la dimensión de «x», comparable al número de variables o podría ubicarse en cualquier punto a partir del mismo número de unidades ocultas como variables de entrada, hasta tal vez el doble, o hasta tres o cuatro veces ese valor. Así que tener un número de unidades ocultas es comparable, varias veces, o en cierto modo mayor al número de variables de entrada que normalmente es bueno.

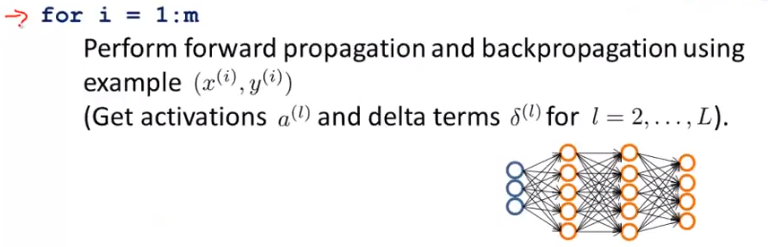
Así que, esperamos que esto les dé un juego de opciones estándar de elección para la arquitectura de la red y si siguen estos alineamientos, probablemente, conseguirán algo que funcione bien, pero en otros videos en los que más adelante hablaré específicamente de consejos sobre cómo aplicar algoritmos de aprendizaje, trataré más a fondo el cómo elegir una arquitectura de red neuronal. De hecho, explicaré mucho más sobre cómo hacer buenas elecciones para el número de unidades ocultas, número de capas ocultas y demás.

A continuación, esto es lo que necesitamos implementar para practicar una red neuronal:



En realidad, son seis los pasos que utilizo, tengo cuatro en esta diapositiva y dos más en la siguiente.

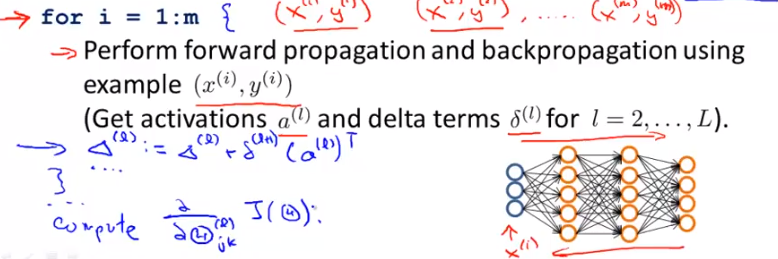
1. El primer paso es configurar la red neuronal e iniciar aleatoriamente los valores de los pesos. Y normalmente iniciamos los pesos con valores bajos, cercanos a cero.
2. Entonces implementamos una propagación hacia adelante para que podamos fijar cualquier entrada «x» en una red neuronal y hacer el cálculo de h(x) que es el vector de salida de los valores «y».
3. Entonces, también implementamos un código para hacer el cálculo de la función de costos de J de «theta».
4. Y después implementamos la retropropagación o su algoritmo, para calcular estos términos de la derivada parcial la derivada parcial de J de «theta» con respecto a los parámetros. En concreto, para implementar la retropropagación. Normalmente haremos esos con un ciclo for en los ejemplos de entrenamiento.



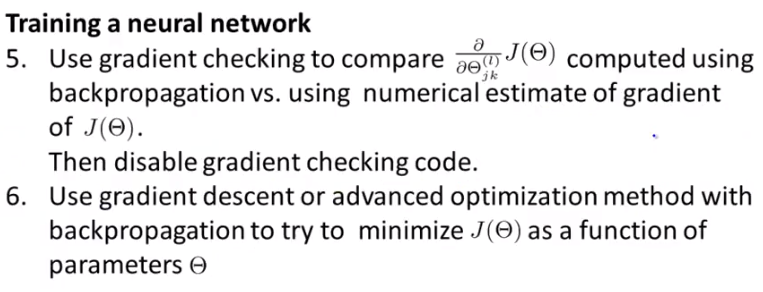
Puede que algunos de ustedes hayan escuchado sobre los métodos de factorización avanzada, francamente, muy avanzada en donde ustedes no tienen un ciclo for en los ejemplos de entrenamiento «m» la primera vez que implementen la retropropagación casi siempre debería haber un ciclo for en su código en donde estén repitiendo los ejemplos, es decir, («x1», «y1») entonces hacen la propagación hacia adelante y la retropropagación en el primer ejemplo, y entonces en la segunda repetición del ciclo for, hacen la propagación hacia adelante y la retropropagación en el segundo ejemplo («x2», «y2»), y así sucesivamente. Hasta que lleguen al ejemplo final («xm», «ym»). Entonces, debe haber un ciclo for en su implementación de propagación hacia adelante, el menos la primera vez que lo estén implementando. Y, francamente, hay maneras más complicadas de hacer esto sin un ciclo for, pero definitivamente, no recomiendo que intenten hacer la versión mucho más complicada la primera vez que implementen la retropropagación

Entonces, en concreto, tenemos un ciclo for en mis ejemplos de entrenamiento «m» y dentro del ciclo de for vamos a llevar a cabo la propagación hacia adelante y la retropropagación usando únicamente este ejemplo. Y lo que significa es que vamos a tomar «x(i)» y a alimentar eso en mi capa de entrada hacer la propagación hacia adelante y la retropropagación y eso me dará todas estas activaciones y todos estos términos «delta» para todas las capas de todas mis unidades en la red.

Vamos a calcular los términos de «delta» que son, aquí está la fórmula que dimos antes, más «delta» a la «l + 1» multiplicado por a matriz traspuesta. Y, finalmente, fuera del ciclo for, habiendo calculado estos términos de «delta», estos términos acumulados entonces tendríamos otro código y eso nos permitirá calcular la derivada parcial, y esta derivada parcial tiene que tomar en cuenta también el término lambda de regularización.



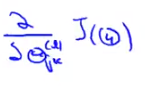
Estas fórmulas se dan en el video anterior



El siguiente es el paso cinco, lo que hago es usar la verificación de la gradiente para comparar los términos de la derivada parcial que fueron calculados. Entonces, he comparado las versiones calculadas usando la retropropagación contra la derivada parcial calculada usando las estimaciones numéricas de la derivada. Entonces, hagan la verificación de la gradiente para asegurarnos que ambas les den valores muy similares.

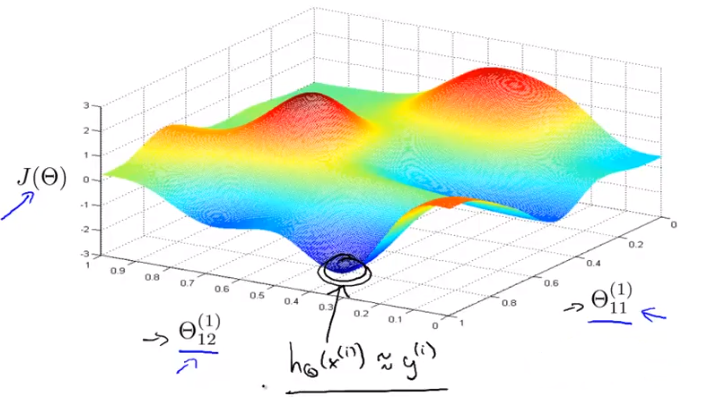
El haber hecho la verificación de la gradiente nos asegura que nuestra implementación de la retropropagación es correcta, y entonces es muy importante que deshabilitemos la verificación de la gradiente porque el código de la verificación de la gradiente es computacionalmente muy lento.

Finalmente, usamos un algoritmo de optimización como la pendiente de la gradiente, o uno de los métodos de optimización como como LB de GS, gradiente de contrato expresada dentro del mínimo de la función multivariable no restringida «fminunc» Usamos éstas junto con la retropropagación, de modo que ésta es lo que calcula esta derivada parcial por nosotros.



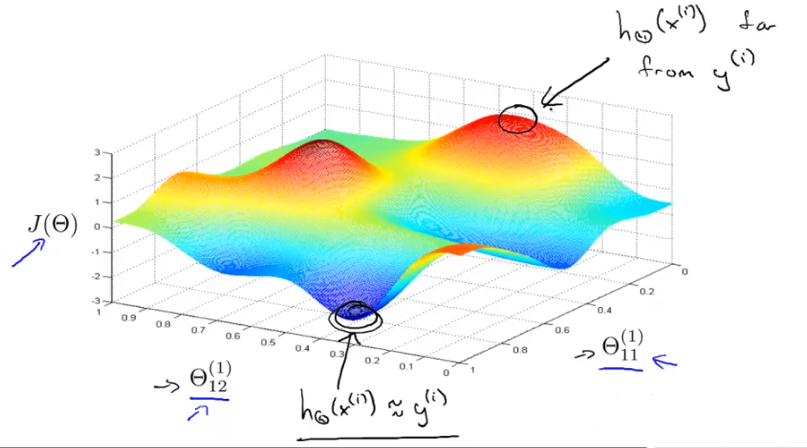
Y así, ya sabemos cómo calcular la función de costos, sabemos cómo calcular la derivada parcial usando la retropropagación, de modo que podemos usar uno de estos métodos de optimización para intentar minimizar J de «theta» como una función de los parámetros de «theta». Y, por cierto, para redes neuronales, esta función de costos J de «theta» no es convexa, o es no convexa y así puede ser susceptible, teóricamente, a los mínimos locales, y de hecho, los algoritmos como la gradiente de descenso y y los métodos de optimización avanzada en teoría, se atoran en las óptimas locales pero resulta que en la práctica, esto no representa normalmente un gran problema y aun cuando no podemos garantizar que estos algoritmos encontrarán una óptimo global, normalmente los algoritmos como la gradiente de descenso harán un muy buen trabajo minimizando esta función de costos J de «theta» y conseguirá un mínimo local muy bueno, incluso si no llega al óptimo global.

Finalmente, la gradiente de descenso para una red neuronal podría aun parecer un poco mágica. Entonces, sólo permítanme enseñar una figura más para intentar obtener esa intuición sobre lo que está haciendo la gradiente de descenso para una red neuronal.



De hecho, esta fue similar a la figura que usé antes para explicar la gradiente de descenso. De esta manera, tenemos alguna función de costos y tenemos una cantidad de parámetros en nuestra red neuronal. ¿correcto? Aquí sólo he puesto dos de los valores parámetro. En realidad, por supuesto, en la red neuronal podemos tener muchos parámetros con éstos. «theta» uno, «theta» dos, todas estas son matrices, ¿cierto? Así, podemos tener parámetros dimensionales altos, pero debido a las limitaciones de la fuente de partes que podemos dibujar. Supongamos que tenemos sólo dos parámetros en esta red neuronal. Aunque obviamente tengamos muchos más en la práctica

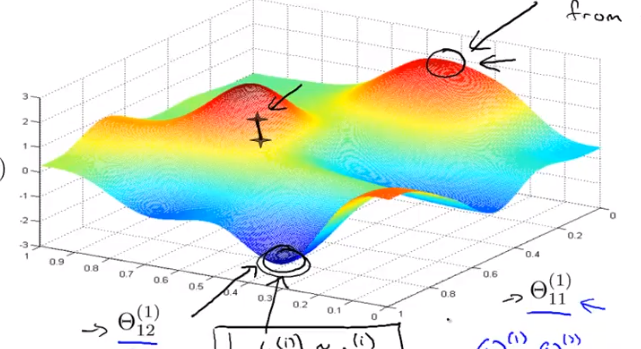
Ahora, esta función de costos J de «theta» mide qué tan bien se ajusta la red neuronal con los datos de entrenamiento. Así que si tomamos un punto como este de aquí abajo ese es el punto en donde J de «theta» es bastante bajo, y así, esto corresponde a una configuración de los parámetros. Hay una configuración de los parámetros de «theta» en donde para la mayoría de los ejemplos de entrenamiento, la salida de mi hipótesis, que puede estar muy cerca de «y(i)» y si esto es correcto entonces eso es lo que causa que mi función de costos sea bastante baja.



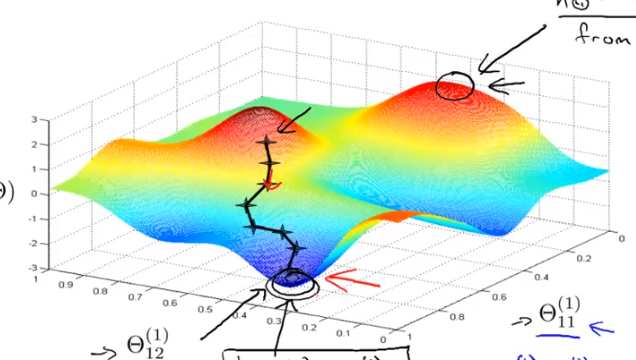
Por el contrario, si fueran a tomar un valor como ese, un punto así corresponde a, en donde en muchos ejemplos de entrenamiento, la salida de mi red neuronal está lejos del valor real de «y(i).

Así que, puntos como este de la derecha corresponden a lo que la hipótesis en donde la red neuronal está sacando valores que, en el conjunto de entrenamiento, están lejos de «y(i)». Así que no está ajustando el conjunto de entrenamiento bien, mientras que puntos así con bajos valores de la función de costos corresponden donde “J” de «theta» es bajo y por lo tanto corresponde a donde la red neuronal ajusta el conjunto de entrenamiento de manera adecuada, porque me refiero a que esto es lo que se necesita que esté correcto para que J de «theta» sea pequeño.

Así que lo que la gradiente de descenso hace es que se iniciará desde un punto inicial aleatorio como ese de aquí, y de forma repetitiva ira hacia abajo.



Y lo que la retropropagación hace es calcular la dirección del gradiente y lo que el gradiente de descenso hace es que está dando pequeños pasos hacia abajo hasta que, esperemos que en este caso, llegue a un óptimo local muy bueno.



Entonces, cuando implementen la retropropagación y usen la gradiente de descenso o uno de los métodos de optimización avanzados, este diagrama explica lo que hace el algoritmo. Está intentando encontrar un valor de los parámetros en donde los valores de salida en la red neuronal concuerdan estrechamente con los valores de las «y(i)» observadas en su conjunto de entrenamiento.

Espero que esto les haya dado una mejor concepción de cómo los distintos componentes del aprendizaje de redes neuronales se ajustan cuando se interactúan juntos.

En caso de que aún después de este video, todavía sientan que hay muchos componentes y no queda muy claro lo que algunos de esos componentes hacen o cómo interactúan estos componentes, no hay mucho problema. El aprendizaje de redes neuronales y retropropagación es un algoritmo complicado. Y aun cuando he visto las matemáticas detrás de la retropropagación durante muchos años y la he usado exitosamente por muchos años, al día de hoy aún siento que no siempre tengo la suma comprensión de lo que la retropropagación hace exactamente en algunos casos. Y cómo el proceso de optmización es al minimizar J de «theta». Y este es un algoritmo mucho más complicado de entender, como si lo manejara con menor entendimiento sobre lo que hace comparado con, digamos, la regresión lineal o la logística. las cuales eran matemática y conceptualmente mucho más simples y con algoritmos menos complicados.

Pero si se sienten de la misma manera, ya saben que es perfectamente normal, pero si implementan la retroprogpagación, esperemos que lo que encuentren es que es uno de los más poderosos algoritmos de aprendizaje y si implementan estos algoritmos, implementen la retropropagación, implementen uno de estos métodos de optimización y encuentren que la retropropagación será capaz de ajustarse de manera muy compleja, poderosa, con funciones no lineares con sus datos, y este es uno de los algoritmos de aprendizaje más efectivos que tenemos hoy en día.