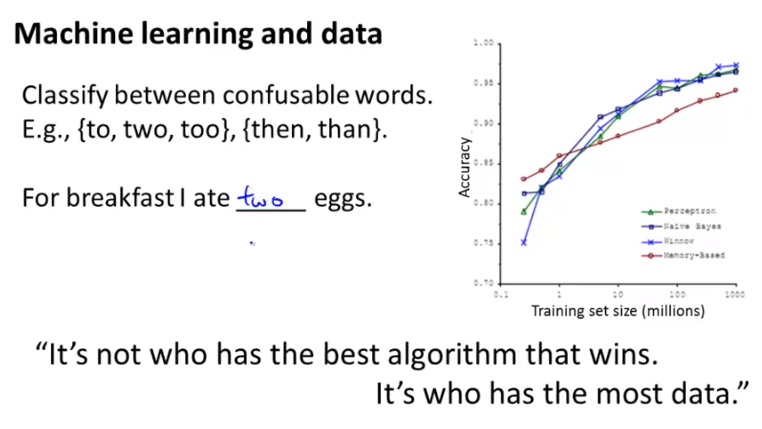
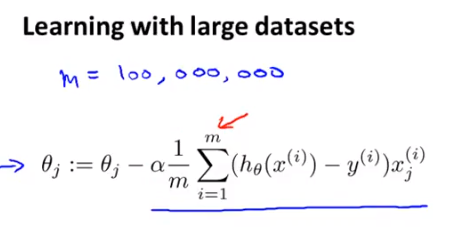
Machine learning y Big Data

En los próximos vídeos, vamos a hablar del aprendizaje automático a gran escala. Es decir, algoritmos que tratan grandes conjuntos de datos. Si miran hacia atrás, a la historia reciente del aprendizaje automático de hace 5 o 10 años, una de las razones por las que los algoritmos de aprendizaje funcionan mejor ahora que, incluso digamos, hace 5 años, es simplemente la enorme cantidad de datos que tenemos ahora y con los que podemos entrenar a nuestros algoritmos. En estos próximos vídeos, vamos a hablar de los algoritmos que vamos a usar cuando tenemos este tipo de conjuntos de datos masivos.   
  
Entonces, ¿por qué queremos utilizar estos grandes conjuntos de datos? Ya hemos visto que una de las mejores maneras de conseguir un sistema de aprendizaje automático de alto rendimiento, es que tomen un algoritmo de aprendizaje de baja oscilación y entrenen a éste sobre una gran cantidad de datos. Así que, uno de los primeros ejemplos que ya hemos visto fue este ejemplo para clasificar entre las palabras confundibles.



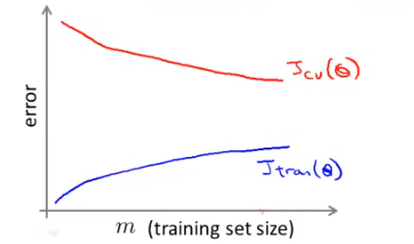
Así, «Para el desayuno, comí dos (DOS) huevos.» En este ejemplo vimos este tipo de resultados, en donde siempre y cuando alimenten al algoritmo con una gran cantidad de datos, parece que funciona muy bien. Así que son los resultados como estos lo que ha llevado al dicho en el aprendizaje automático de que a menudo, no es quién tiene el mejor algoritmo el que gana, es quien tiene la mayor cantidad de datos. De modo que desean aprender de grandes conjuntos de datos, por lo menos cuando podemos conseguir estos grandes conjuntos de datos.

Pero el aprendizaje con grandes conjuntos de datos viene con sus propios problemas singulares, en concreto, los problemas computacionales. Digamos que el tamaño de su conjunto de entrenamiento es «m» igual a 100 millones. Y esto es en realidad bastante realista para muchos conjuntos de datos modernos. Si se fijan en el conjunto de datos del Censo de EE.UU., hay, ya saben, 300 millones de personas en los EE.UU., por lo general pueden obtener cientos de millones de registros. Si observan la cantidad de tráfico que tienen los sitios web populares, fácilmente pueden obtener conjuntos de entrenamiento que son mucho más grandes que cientos de millones de ejemplos. Y digamos que quieren entrenar un modelo de regresión lineal, o tal vez un modelo de regresión logística, en cuyo caso esta es la regla del gradiente de descenso. Y si se fijan en lo que tienen que hacer para calcular el gradiente, que es este término por aquí,

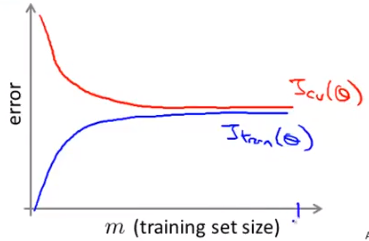


Entonces cuando «m» es cien millones, tienen que realizar una suma de más de cien millones de términos, a fin de calcular estos términos derivados y realizar un solo paso de gradiente descendente. Debido al costo computacional por sumar más de cien millones de entradas, a fin de calcular sólo un paso del gradiente de descenso, en los próximos vídeos hablaremos acerca de las técnicas ya sea para sustituir este algoritmo por algo más, o para encontrar formas más eficientes para calcular esta derivada. Al final de esta secuencia de videos sobre el aprendizaje automático a gran escala, sabrán cómo ajustar modelos, --regresión lineal, regresión logística, redes neuronales,etcétera-- inclusive hasta con conjuntos de datos de, digamos, cien millones de ejemplos. Por supuesto, antes de esforzarnos por entrenar un modelo con cien millones de ejemplos, también debemos preguntarnos, bueno, ¿por qué no utilizar solamente mil ejemplos?. Tal vez podamos escoger al azar los subconjuntos de un millar de ejemplos de cien millones de ejemplos y entrenar a nuestro algoritmo sólo con mil ejemplos. Así que antes de invertir esfuerzo en el desarrollo real y el software necesario para entrenar a estos modelos masivos, a menudo es una buena prueba de validez si el entrenamiento en sólo un millar de ejemplos pudiera resultar igual de bien.

La forma habitual para comprobar la validez del uso de un conjunto de entrenamiento mucho más pequeño que podría ser igual de bueno, es decir, si el uso de un conjunto de entrenamiento mucho más pequeño, «m» es igual a 1000, que pudiera hacer igual de bien, es trazar las curvas de aprendizaje; de manera que si fueran a trazar las curvas de aprendizaje, y si su objetivo de entrenamiento tuviera este aspecto, eso es «J entrena «theta»». Y si su objetivo de conjunto de validación cruzada «Jcv» de «theta», se viera de esta manera:



Entonces esto se parece a un algoritmo de aprendizaje de alta varianza, y estaremos más seguros de que la adición de ejemplos de entrenamiento adicionales mejoraría el desempeño, mientras que por el contrario, si fueran a trazar las curvas de aprendizaje, si su objetivo de entrenamiento se viera así, y si su objetivo de validación cruzada luciera de esa manera,



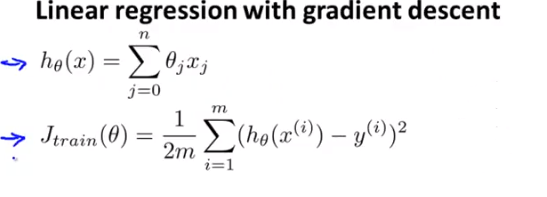
Entonces esto se parece al algoritmo de aprendizaje clásico de alta oscilación. Y en este último caso si tuvieran que trazar esto hasta, por ejemplo, «m» es igual a 1000, de manera que «m» es igual a 500, hasta «m» es igual a 1000, entonces parece poco probable que el aumento de «m» a cien millones vaya a ser mucho mejor, y entonces estaría bien que usar «m» igual a 1000, en lugar de invertir una gran cantidad de esfuerzo para encontrar la escala del algoritmo.

Por supuesto, si estuvieran en la situación que se muestra en esta última figura, entonces lo natural sería añadir variables adicionales, o añadir unidades ocultas adicionales a su red neuronal y así sucesivamente, de modo que terminen con una situación más cercana a la de la primera figura, en donde tal vez esto es hasta «m» es igual a 1000, y esto entonces les da más confianza que tratar de añadir infraestructura para cambiar el algoritmo para usar mucho más que miles de ejemplos que en realidad podría ser un buen uso de su tiempo. Así que en el aprendizaje automático a gran escala, nos gusta encontrar formas de cómputo razonables, o formas computacionalmente eficientes, para hacer frente a los conjuntos de datos muy grandes. En los próximos vídeos, veremos dos ideas principales. La primera se llama gradiente de descenso estocástico y la segunda se llama Map Reduce, para tratar con conjuntos de datos muy grandes. Y después de que hayan aprendido acerca de estos métodos, con suerte eso les permitirá ampliar sus algoritmos de aprendizaje para muchos datos, y les permitirá obtener un mejor rendimiento en muchas aplicaciones diferentes.

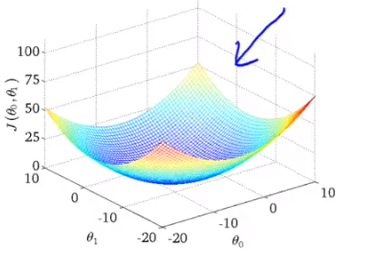
# Descenso del gradiente estocástico

Para muchos algoritmos de aprendizaje, entre ellos la regresión lineal, la regresión logística y las redes neuronales, la forma en que derivamos el algoritmo fue encontrando una función de costos, o encontrando un objetivo de optimización, y después usando un algoritmo como el gradiente de descenso para minimizar esa función de costos. Cuando tenemos un conjunto de entrenamiento muy grande, el gradiente de descenso se convierte en un procedimiento muy costoso computacionalmente. En este video, hablaremos de una modificación al algoritmo básico del gradiente de descenso llamado gradiente de descenso estocástico, que nos permitirá ampliar estos algoritmos para conjuntos de entrenamiento mucho mayores.

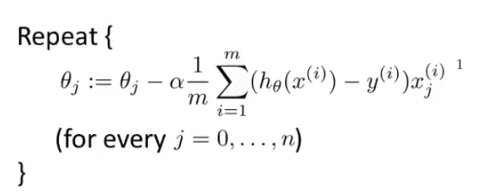
Supongamos que están entrenando un modelo de regresión lineal utilizando el gradiente de descenso. Como un resumen rápido, la hipótesis y la función de coste se verán así:



La función J, que es la mitad del promedio de error cuadrado de su hipótesis sobre sus ejemplos de entrenamiento «m», y la función de costos que ya hemos visto, se parece a este tipo de función en forma de arco.

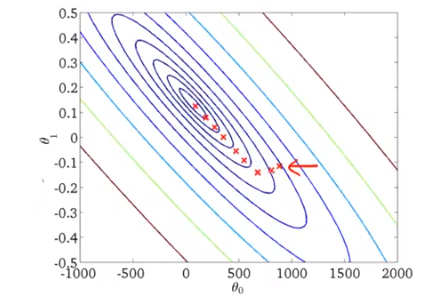


Así, trazada como la función de los parámetros «theta» 0 y «theta» 1, la función de costo «J» es un tipo de función en forma de arco. Y el gradiente de descenso, algoritmicamente se ve así:

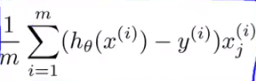


En donde en el bucle interno del gradiente de descenso ustedes actualizan repetidamente los parámetros «theta» usando esa expresión.

Ahora bien, en el resto de este video, voy a seguir usando la regresión lineal como el ejemplo en ejecución. Pero las ideas aquí, las ideas del gradiente de descenso estocástico son completamente generales y también aplican a otros algoritmos de aprendizaje, como la regresión logística, las redes neuronales y otros algoritmos que se basan en entrenar el gradiente de descenso sobre un conjunto de entrenamiento específico. Así que aquí está una imagen de lo que hace el gradiente de descenso. Si los parámetros se inicializan hacia en el primer punto señalado con una flehca, entonces, a medida que ejecuten el gradiente de descenso, las diferentes iteraciones del gradiente de descenso tomarán los parámetros hasta el mínimo global. Así que tomen una trayectoria que se parezca a eso y que se dirige de manera muy directa al mínimo global.



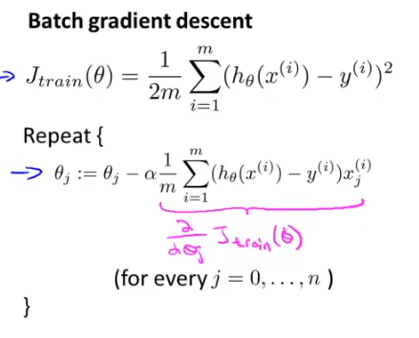
Ahora, el problema con el gradiente de descenso es que si «m» es grande, entonces el cálculo de este término derivado:



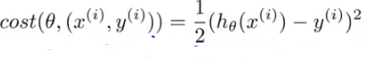
puede ser muy costoso ya que esto requiere sumar sobre todos los ejemplos «m». De modo que si «m» es 300 millones, bien, entonces en los Estados Unidos hay cerca de 300 millones de personas. Así que los datos del censo de los EE.UU. o Estados Unidos pueden estar en el orden de esa cantidad de registros, de modo que ustedes desean ajustar el modelo de regresión lineal a eso y luego sumar más de 300 millones de registros. Y eso es muy caro. Para darle un nombre al algoritmo, esta versión particular de gradiente de descenso también se llama gradiente de descenso por lotes (*“batch”*). Y el término lote se refiere al hecho de que estamos viendo todos los ejemplos de entrenamiento a la vez. Lo llamamos una especie de lote de todos los ejemplos de entrenamiento. Y tal vez no es en realidad el mejor nombre, pero esto es lo que la gente del aprendizaje automático llama a esta versión particular del gradiente de descenso. Y si realmente imaginan que tienen 300 millones de registros de los censos guardadas en el disco, la forma en que este algoritmo funciona es que necesitan leer en la memoria de la computadora los 300 millones de registros a fin de calcular este término derivativo. Necesitan transmitir todos estos registros a través de la computadora porque no pueden almacenar todos sus archivos en la memoria de la computadora. Así que tienen que leer a través de ellos y, poco a poco, ya saben, acumular la suma con el fin de calcular la derivada. Y, después de haber hecho todo ese trabajo, eso les permite tomar un paso del gradiente de descenso. Y ahora tienen que hacer todo de nuevo, ya saben, buscar a través de los 300 millones de registros, acumular estas sumas, y después de haber hecho todo ese trabajo, pueden tomar otro pequeño paso usando el gradiente de descenso. Y luego, hacerlo de nuevo; y después, toman un tercer paso, y así sucesivamente. De modo que les va a tomar mucho tiempo conseguir que el algoritmo converja.

En contraste con el gradiente de descenso por lotes, lo que vamos a hacer es encontrar un algoritmo diferente que no necesite que se vean todos los ejemplos de entrenamiento en cada iteración, sino que necesite observar solamente un solo ejemplo de entrenamiento en una iteración. Antes de pasar al nuevo algoritmo. A

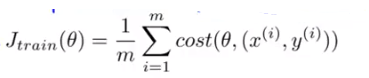
Aquí está sólo un algoritmo del gradiente de descenso por lotes escrito de nuevo, tomando de nuevo la función de costos y la actualización y, por supuesto, dentro de la actualización tenemos el término que se usa en la regla del gradiente de descenso, que es la derivada parcial con respecto a los parámetros «theta» «J» de nuestro objetivo de optimización, «J entrenamiento» de «theta».



Ahora, veamos el algoritmo más eficiente que se adapta mejor a los grandes conjuntos de datos. Con el fin de trabajar desde los algoritmos llamados gradientes de descenso estocásticos, escribamos la función de costos de una manera ligeramente diferente. Tenemos que encontrar el costo del parámetro «theta» con respecto a un ejemplo de entrenamiento «x(i), y(1)» para que sea igual a la mitad de las veces del error al cuadrado en la que incurre mi hipótesis en ese ejemplo, «x(i), y(i)». De modo que este término de la función de costos realmente mide qué tan bien está funcionando mi hipótesis sobre un solo ejemplo «x(i), y(i)».



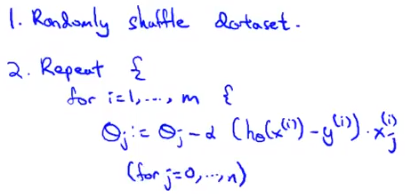
Ahora notan de que la función de costo «j trainr» general se puede escribir de esta forma equivalente.



Así que «j train» es sólo el promedio sobre mis ejemplos de entrenamiento «m» del costo de mi hipótesis en ese ejemplo «x(i), y(i)».

Armado con esta visión de la función de costos para la regresión lineal, permítanme escribir ahora lo que hace el gradiente de descenso estocástico.

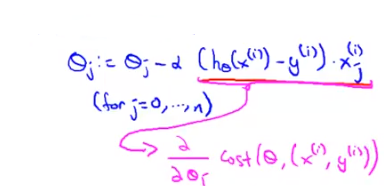
1. El primer paso del gradiente de descenso estocástico es mezclar al azar el conjunto de datos. Con esto sólo me refiero a mezclar de manera aleatoria o a reordenar sus ejemplos «m» de entrenamiento. Es una especie de paso de pre-procesamiento estándar; regresaré a esto en un minuto. Pero el trabajo principal del gradiente de descenso estocástico se hace a continuación en lo siguiente.
2. Vamos a repetir para «i» es igual a 1 hasta «m». Así que vamos a buscar repetidamente a través de mis ejemplos de entrenamiento y realizar la siguiente actualización. Voy a actualizar el parámetro «theta» «j» como «theta» «j» menos «alfa» veces «h» de «x(i)» menos «y(i)» veces «x(i)j». Y vamos a hacer esta actualización como de costumbre para todos los valores de «j».



Ahora, observamos que este término aquí es exactamente lo que teníamos dentro de la suma del gradiente de descenso por lotes.



De hecho, para aquellos de ustedes que estén familiarizados con cálculo, es posible demostrar que ese término es igual a la derivada parcial con respecto a mi parámetro «theta» «j» del costo de los parámetros «theta» sobre «x(i), y(i)». En donde el costo es, por supuesto, esto que se definió anteriormente.



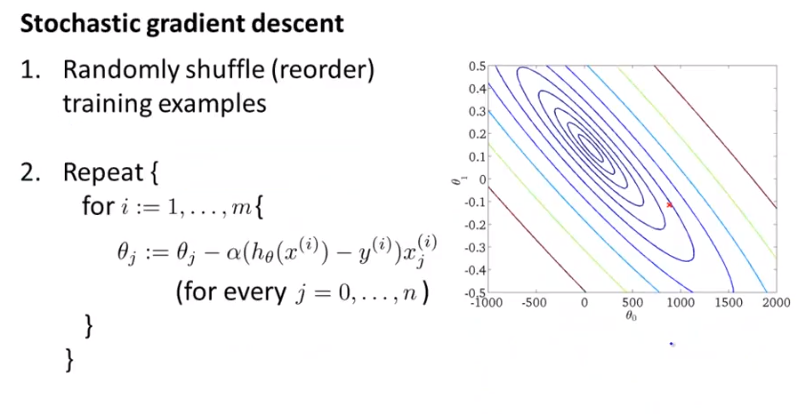
Así que lo que el gradiente de descenso estocástico está haciendo es en realidad escanear a través de los ejemplos de entrenamiento. Y primero va a ver mi primer ejemplo de entrenamiento «x(1), y(1)». Y después, viendo sólo este primer ejemplo, va a tomar básicamente como un paso pequeño del gradiente de descenso con respecto al costo de sólo este primer ejemplo de entrenamiento. De modo que, en otras palabras, vamos a ver el primer ejemplo y modificar los parámetros un poco para ajustar sólo el primer ejemplo de entrenamiento un poco mejor. Una vez hecho esto dentro de este bucle interior, se va a ir entonces al segundo ejemplo de entrenamiento. Y lo que va a hacer allí es dar otro pasito más en el espacio del parámetro, así que modificar los parámetros sólo un poco para ajustar solamente un segundo ejemplo de entrenamiento un poco mejor. Una vez hecho esto, se va a ir a mi tercer ejemplo de entrenamiento y modificar los parámetros para tratar de ajustar sólo el tercer ejemplo de entrenamiento un poco mejor, y así sucesivamente hasta que, ya saben, pasen a través de todo el conjunto de entrenamiento. Y después, este bucle exterior repetido puede ocasionar que pase múltiples veces sobre el conjunto de entrenamiento completo. Esta visión del gradiente de descenso estocástico también motiva la razón por la que deseamos empezar por mezclar de manera aleatoria el conjunto de datos. Esto nos asegura que cuando escaneamos a través del conjunto de entrenamiento aquí, terminamos visitando los ejemplos de entrenamiento en algún tipo de orden mezclado al azar. Dependiendo de si sus datos ya estaban ordenados al azar, o si venían mezclados originalmente en un orden extraño, en la práctica esto sólo agilizaría las conversiones al gradiente de descenso estocástico sólo un poco. Así que, en aras de la seguridad, por lo general es mejor mezclar aleatoriamente el conjunto de datos si no están seguros si llegó a ustedes en un orden mezclado al azar o no.

Pero más importante aún, otra vista del gradiente de descenso estocástico es que es muy parecido al gradiente de descenso de lote, pero en lugar de esperar a sumar todos los términos del gradiente sobre todos los ejemplos «m» de entrenamiento, lo que estamos haciendo es que estamos tomando este término del gradiente:

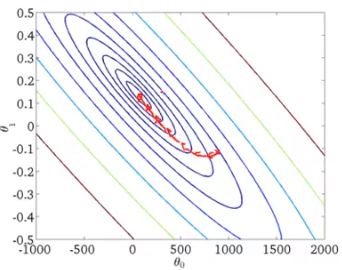


usando sólo un ejemplo de entrenamiento, y ya estamos empezando a hacer progreso para mejorar los parámetros. Así que en lugar de esperar hasta tomar una trayectoria a través de todos los 300,000 de registros de censo de Estados Unidos, es decir, en lugar de tener que buscar a través de todos los ejemplos de entrenamiento, antes de que podamos modificar los parámetros un poco y avanzar hacia un mínimo global, en vez de eso, para el gradiente de descenso estocástico, sólo tenemos que mirar a un solo ejemplo de entrenamiento y ya estamos empezando a realizar progresos en este caso de los parámetros, hacia mover los parámetros al mínimo global.

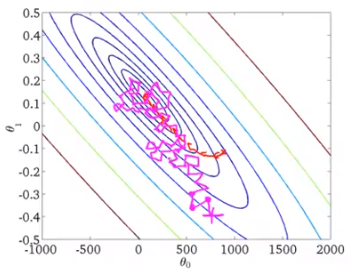
Por lo tanto, aquí está el algoritmo escrito de nuevo en el que el primer paso es mezclar aleatoriamente los datos, y el segundo paso es en el que se hace el trabajo real, en donde está esa actualización con respecto a un único ejemplo de entrenamiento «x(i), y(i)».



Por lo tanto, vamos a ver lo que este algoritmo hace a los parámetros. Anteriormente, vimos que cuando estamos usando gradiente de descenso por lotes, este es el algoritmo que analiza todos los ejemplos de entrenamiento a la vez. El gradiente de descenso por lotes tenderá a tomar una trayectoria de línea razonablemente recta para llegar al mínimo global de esta manera.

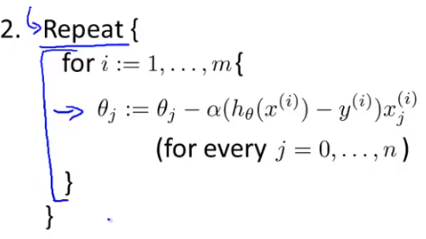


En contraste con el gradiente de descenso estocástico, cada iteración va a ser mucho más rápida porque no necesitamos sumar sobre todos los ejemplos de entrenamiento, pero cada iteración sólo está tratando de ajustar mejor el único ejemplo de entrenamiento. Así que, si empezáramos el gradiente de descenso estocástico, vamos a iniciar el gradiente de descenso estocástico en un punto. La primera iteración, ya saben, puede tomar los parámetros en esa dirección y tal vez la segunda iteración, que ve sólo el segundo ejemplo, tal vez sólo por casualidad, no tengamos suerte y en realidad nos dirigimos en una mala dirección con los parámetros de esta manera. En la tercera iteración en la que tratamos de modificar los parámetros para ajustar mejor sólo los terceros ejemplos de entrenamiento, tal vez terminemos dirigiéndonos en esa dirección. Y luego vamos a ver el cuarto ejemplo de entrenamiento y haremos eso. El quinto ejemplo, sexto ejemplo, séptimo y así sucesivamente. Y a medida que ejecutan el gradiente de descenso estocástico, lo que encuentran es que generalmente moverá los parámetros en la dirección del mínimo global, pero no siempre. Así que tomará trayectorias más aleatorias, tortuosas hacia el mínimo global.

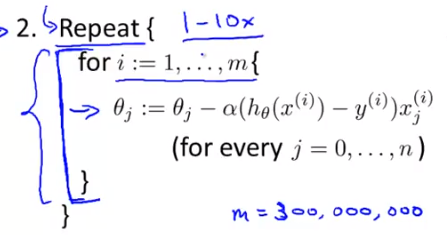


Y, de hecho, a medida que ejecutan el gradiente de descenso estocástico, éste no converge en realidad en el mismo mismo sentido en el que lo hace el gradiente de descenso por lotes y lo que termina haciendo es dar vueltas continuamente en alguna región que está en alguna región cerca del mínimo global, pero no sólo llega al mínimo global y se queda allí. Pero en la práctica esto no es un problema porque, ya saben, siempre y cuando los parámetros terminen en alguna región allí, tal vez están bastante cerca del mínimo global. Así que siempre y cuando los parámetros terminen muy cerca del mínimo global, esa será una muy buena hipótesis, y por lo general, al ejecutar el gradiente de descenso estocástico, obtenemos un parámetro cerca del mínimo global y eso es suficientemente bueno para esencialmente la mayoría de propósitos prácticos.

Sólo un detalle final. En el gradiente de descenso estocástico tuvimos esta repetición del bucle exterior que nos dice que hagamos este bucle interior varias veces. Así que, ¿cuántas veces repetimos este bucle externo?



Dependiendo del tamaño del conjunto de entrenamiento, podría ser suficiente si hacemos este bucle una sola vez. Y hasta, ya saben, tal vez 10 veces podría ser lo usual, de modo que podríamos terminar repitiendo este bucle interno de una a diez veces. Así que si tenemos un conjunto de datos verdaderamente masivo como este censo de los EE.UU. que nos dio ese ejemplo del que he estado hablando, con 300 millones de ejemplos, es posible que para cuando haya tomado un solo pase a través de su conjunto de entrenamiento. Así que, esto es para «i» es igual a 1 hasta 300 millones. Es posible que para cuando hayan tomado un solo pase a través de su conjunto de datos, ya tengan un hipótesis perfectamente buena, en cuyo caso puede que sólo tengan que hacer este bucle interno una sola vez si «m» es muy, muy grande. Pero, en general, tomando cualquier cantidad de entre 1 a 10 pases a través de su conjunto de datos puede ser bastante común,



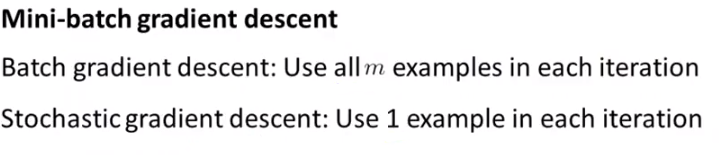
Pero en realidad depende del tamaño de su conjunto de entrenamiento. Y si contrasta esto con el gradiente de descenso por lotes, con el gradiente de descenso por lotes, después de tomar un pase a través de todo el conjunto de entrenamiento, habrían tomado un solo paso del gradiente de descenso, así que uno de estos pequeños pasos de bebé del gradiente de descenso en el que toman sólo un pequeño paso del gradiente de descenso, y esta es la razón por la que el gradiente de descenso estocástico puede ser mucho más rápido.

Por lo tanto, ese fue el algoritmo del gradiente de descenso estocástico. Y si lo implementan, les permitirá ampliar muchos de sus algoritmos de aprendizaje a conjuntos de datos mucho más grandes y obtener mucho más rendimiento de esa manera.

# Descenso del gradiente *“mini batch”*

En el video anterior, hablamos del gradiente de descenso estocástico y de cómo éste puede ser mucho más rápido que el gradiente de descenso en lotes. En este vídeo vamos a hablar de otra variación sobre estas ideas que se llama gradiente de descenso de mini lote (*“mini batch”*), que puede trabajar a veces incluso un poco más rápido que el gradiente de descenso estocástico.

Para resumir el algoritmo del que hemos hablado hasta ahora, en el gradiente de descenso por lotes vamos a utilizar todos los ejemplos «m» en cada iteración, mientras que en el gradiente de descenso estocástico vamos a utilizar un solo ejemplo en cada iteración.



Lo que hace el gradiente de descenso de mini lotes es algo en un punto intermedio.

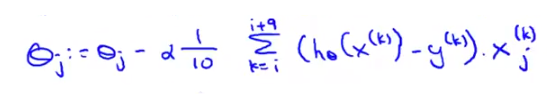


Específicamente con este algoritmo, vamos a utilizar ejemplos «b» en cada iteración, en donde «b» es un parámetro llamado "tamaño de mini lote", de modo que la idea es que está de alguna manera entre el gradiente de descenso por lotes y el gradiente de descenso estocástico. Esto es igual que el gradiente de descenso por lotes, excepto que voy a utilizar un tamaño de lote mucho más pequeño. Una elección típica para el valor de «b» podría ser que «b» es igual a 10, digamos, y un rango típico realmente podría estar en cualquier lugar desde «b» es igual a 2, hasta «b» es igual a 100. Así que ese será un rango de valores bastante típico para el lote de mini lotes.

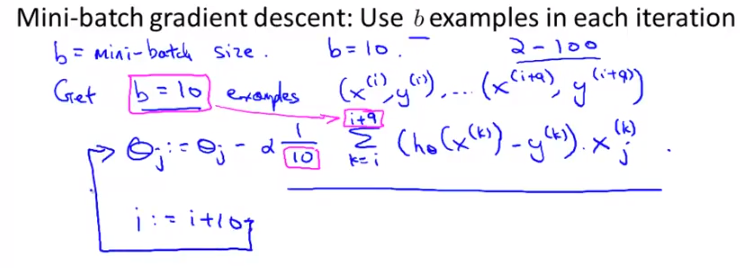
La idea es que, en lugar de utilizar un ejemplo a la vez, o «m» ejemplos a la vez, usaremos «b» ejemplos a la vez. De modo que escribiré esto de manera informal, vamos a obtener, digamos, «b». Para este ejemplo, digamos que «b» es igual a 10. Así que vamos a obtener los próximos 10 ejemplos de mi conjunto de entrenamiento, de modo ese pueda ser algún conjunto de ejemplos «xi, yi».



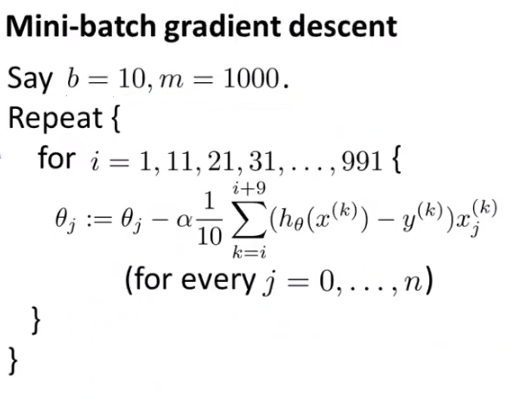
Si se trata de 10 ejemplos entonces la indexación será hasta «x(i+9), y(i+9)», de modo que son 10 ejemplos en total y luego vamos a realizar básicamente una actualización del gradiente de descenso usando estos 10 ejemplos.



De modo que en esta expresión, estamos sumando los términos del gradiente sobre mis diez ejemplos. Así que, el número diez, este es mi tamaño de mini lote y en «i+9» el 9 proviene de la elección del parámetro «b» y después de esto, aumentaremos «i» por el décimo. Vamos a pasar a los próximos diez ejemplos y luego nos seguiremos moviendo de esta manera.



Sólo para escribir el algoritmo en su totalidad, a fin de simplificar la indexación para esto voy a suponer que tenemos un tamaño de mini lote de diez y un tamaño de conjunto de entrenamiento de mil.



Lo que vamos a hacer es tener este tipo de fórmula: para «i» es igual a 1, 11, 21, así que escalonando en pasos de 10 porque observamos 10 ejemplos a la vez; y luego realizamos este tipo de actualización del gradiente de descenso usando diez ejemplos a la vez, de modo que este 10 y este i+9, esos son la consecuencia de haber elegido que mi mini lote sea 10. Si tengo 1000 muestras de entrenamiento, entonces necesito 100 pasos de tamaño 10 a fin de pasar a través de mi conjunto de entrenamiento.

Así que este es el gradiente de descenso de mini lote. Comparado con el gradiente de descenso por lotes, éste también nos permite hacer progresos mucho más rápido. Así que tenemos de nuevo nuestro ejemplo de ejecución de, ya saben, los datos del censo de EE.UU. con 300 millones de ejemplos de entrenamiento, entonces lo que estamos diciendo después de ver tan sólo los 10 primeros ejemplos, podemos empezar a hacer progresos para mejorar los parámetros «theta» así que no tenemos que buscar a través de todo el conjunto de entrenamiento. Sólo tenemos que mirar a los primeros 10 ejemplos y esto nos permitirá empezar a hacer progresos y después podemos ver los segundos diez ejemplos y modificar los parámetros un poco de nuevo, y así sucesivamente.

Por lo tanto, esa es la razón por la que el gradiente de descenso de mini lote puede ser más rápido que el gradiente de descenso por lotes, a saber, pueden empezar a hacer progresos en la modificación de los parámetros después de mirar sólo diez ejemplos, en lugar de tener que esperar hasta que hayan buscado cada ejemplo individual de los 300 millones de ellos.

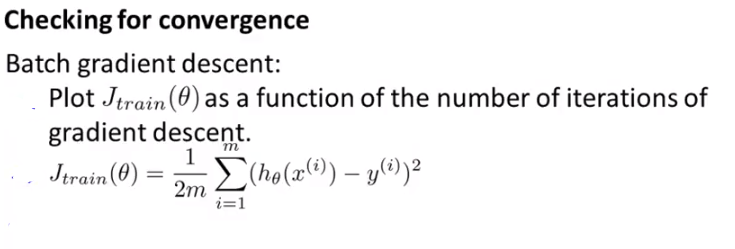
Así que, ¿qué hay del gradiente de descenso de mini lote, en comparación con el gradiente de descenso estocástico?. Así que, ¿por qué queremos observar los ejemplos «b» a la vez, en lugar de mirar un solo ejemplo a la vez, como en el gradiente de descenso estocástico? La respuesta está en la vectorización. En particular, es posible que el gradiente de descenso de mini lote supere el desempeño del gradiente de descenso estocástico sólo si tienen una buena implementación vectorizada. En ese caso, la suma sobre 10 ejemplos se puede realizar de una manera más vectorizada, que le permitirá paralelizar parcialmente sus cálculos en los diez ejemplos. Así que, en otras palabras, usando la vectorización apropiada para calcular los términos derivativos, algunas veces pueden usar parcialmente las buenas bibliotecas de álgebra lineal numérica y paralelizar sus cálculos del gradiente en los ejemplos «b», mientras que, si están viendo un solo ejemplo a la vez con el gradiente de descenso estocástico entonces, ya saben, sólo mirando un ejemplo a la vez, no hay mucho para paralelizar. Al menos, hay menos para paralelizar. Una desventaja del gradiente de descenso de mini lote es que ahora existe este parámetro extra «b», el tamaño de mini lote con el que quizás tengan que jugar y que, por tanto, podría llevar tiempo. Pero si tienen una buena implementación vectorizada, esto se puede ejecutar aún más rápido que el gradiente de descenso estocástico.

Así que este fue el gradiente de descenso de mini lote que es un algoritmo que, en cierto sentido, hace algo que de alguna manera está entre lo que hace el gradiente de descenso estocástico y lo que hace el gradiente de descenso por lotes. Y si eligen su valor razonable de «b», yo suelo usar «b» es igual a 10, pero, ya saben, otros valores, desde por ejemplo 2 hasta 100, serían razonablemente comunes. Así que si elegimos un valor de «b», y si usan una buena implementación vectorizada, a veces pudiera ser más rápido que el gradiente de descenso estocástico, y más rápido que el gradiente de descenso por lotes.

# Convergencia del gradiente estocástico

Ustedes ahora ya saben acerca del algoritmo de gradiente de descenso estocástico, pero cuando están ejecutando el algoritmo, ¿cómo se aseguran de que está completamente depurado y que está convergiendo bien? Igualmente importante, ¿cómo sintonizan el índice de aprendizaje «alfa» con el gradiente de descenso estocástico?. En este vídeo hablaremos de algunas técnicas para hacer esto, para asegurarse de que está convergiendo, y para captar el índice de aprendizaje «alfa».

Antes, cuando usamos el gradiente de descenso por lotes, nuestra forma estándar para asegurarnos de que el gradiente de descenso estaba convergiendo, fue que trazamos la función de costos de optimización como una función del número de iteraciones. Así que esta fue la función de costos y nos debemos asegurar de que esta función de costos disminuye en cada iteración.



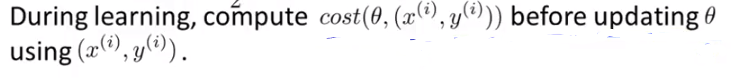
Cuando los tamaños de los conjuntos de entrenamiento eran pequeños, podíamos hacer eso porque podíamos calcular la suma de manera eficaz. Pero cuando se tiene un tamaño enorme de conjunto de entrenamiento, entonces usted no desea tener que pausar su algoritmo de manera periódica, no quiere pausar el gradiente de descenso estocástico periódicamente a fin de calcular la función de costos ya que requiere una suma del tamaño del conjunto de entrenamiento completo y todo el punto del gradiente estocástico fue que deseaban empezar a hacer progresos después de ver sólo un ejemplo, sin necesidad de revisar de vez en cuando a través de todo el conjunto de entrenamiento justo a la mitad del algoritmo, sólo para calcular cosas como la función de costos de todo el conjunto de entrenamiento. Así que para el gradiente de descenso estocástico, a fin de verificar que el algoritmo está convergiendo, esto es lo que podemos hacer en su lugar.

Tomemos la definición del costo que teníamos con anterioridad,



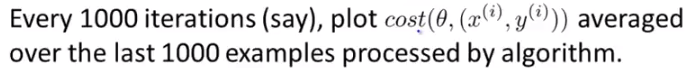
De modo que el costo de los parámetros «theta» con respecto a un solo ejemplo de entrenamiento es sólo una mitad del error al cuadrado en ese ejemplo de entrenamiento. Después, mientras que el gradiente de descenso estocástico está aprendiendo, justo antes de que entrenemos en un ejemplo concreto, en el gradiente de descenso estocástico vamos a mirar los ejemplos «xi», «yi» en orden, y después tomamos una pequeña actualización con respecto a este ejemplo, y pasamos al siguiente ejemplo, «xi» más 1, «yi» más 1, y así sucesivamente, ¿correcto? Eso es lo que hace el gradiente de descenso estocástico.

Así que, mientras el algoritmo está mirando el ejemplo «xi, yi», pero antes de que haya actualizado los parámetros «theta» usando ese ejemplo, vamos a calcular el costo de ese ejemplo. Sólo para decir lo mismo otra vez, pero usando palabras ligeramente diferentes, un gradiente de descenso estocástico está buscando a través de nuestro conjunto de entrenamiento justo antes de que actualicemos «theta», utilizando un ejemplo de entrenamiento específico «x(i)» coma «y(i)»; vamos a calcular qué tan bien está funcionando nuestra hipótesis en ese ejemplo de entrenamiento.

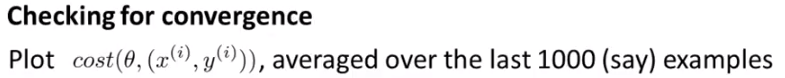


Y queremos hacer esto antes de actualizar «theta» porque si sólo hemos actualizado «theta» utilizando ese ejemplo de entrenamiento, ya saben, eso podría estar funcionando mejor en ese ejemplo de lo que sería representativo.

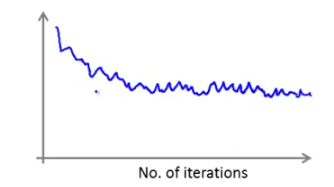
Por último, a fin de comprobar la convergencia del gradiente de descenso estocástico, lo que podemos hacer es cada, digamos, cada mil iteraciones, podemos trazar estos costos que hemos estado calculando en el paso anterior.



Podemos trazar esos costos promedio sobre, digamos, los últimos mil ejemplos procesados ​​por el algoritmo. Y si lo hacen así, es posible que les dé un cálculo sobre qué tan bien está funcionando el algoritmo sobre, ya saben, los últimos 1000 ejemplos de entrenamiento que su algoritmo ha observado. Así, en lugar de calcular «J train» periódicamente, lo que haría necesario revisar a través de todo el conjunto de entrenamiento, con este otro procedimiento, como parte del gradiente de descenso estocástico, no cuesta mucho calcular estos costos también, justo antes de actualizar al parámetro «theta». Y todo lo que estamos haciendo es, cada mil iteraciones más o menos, simplemente promediamos los últimos 1000 costos que hemos calculado y trazamos eso. Y al ver esos gráficos, esto nos permitirá comprobar si el gradiente de descenso estocástico está convergiendo. Así que aquí están algunos ejemplos de cómo pudieran verse estos gráficos.

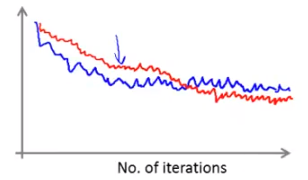


Supongan que han trazado el costo promedio sobre los últimos mil ejemplos, debido a que éstos se promedian sobre sólo un millar de ejemplos, van a ser un poco ruidosos, así que pudieran no disminuir en cada iteración individual. Entonces, si obtienen una figura como esta, de manera que el gráfico es ruidoso porque su promedio está sobre sólo un pequeño subconjunto, por decir, un millar de ejemplos de entrenamiento, si obtiene una figura que se vea así,



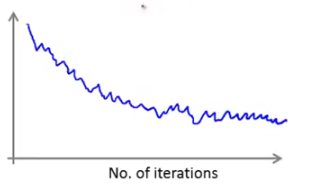
Ya saben, eso sería una ejecución bastante decente con el algoritmo tal vez, en donde parece que el costo ha bajado y entonces esta meseta que parece un poco aplanada, ya saben, empezando alrededor de ese punto. Así que parece que así es como se ven sus costos; entonces tal vez su algoritmo de aprendizaje ha convergido.

Si desean probar el uso de una frecuencia de aprendizaje más pequeña, algo que pueden ver es que el algoritmo puede aprender más lentamente al inicio, por lo que el costo se reduce más lentamente, pero luego, eventualmente, tienen una frecuencia de aprendizaje más pequeña. En realidad, es posible que el algoritmo termine en una solución ligeramente mejor. De modo que la línea roja puede representar el comportamiento del gradiente en descenso estocástico usando una frecuencia de aprendizaje menor, más lenta.

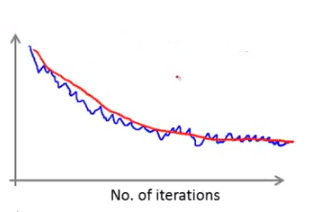


Y la razón de que este es el caso se debe a que, si recuerdan, el gradiente de descenso estocástico no sólo converge en el mínimo global, es que lo que hace es que los parámetros oscilarán un poco alrededor del mínimo global. Así que, mediante el uso de un índice de aprendizaje más pequeño, terminarán con oscilaciones más pequeñas. Y a veces, esta pequeña diferencia será insignificante, y algunas veces con el más pequeño pueden obtener un valor ligeramente mejor para los parámetros.

Aquí hay algunas otras cosas que podrían ocurrir. Digamos que ejecutan el gradiente de descenso estocástico y tienen un promedio de más de mil ejemplos cuando trazan estos costos. Así que aquí podría estar el resultado de otro de estos gráficos.

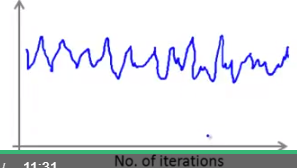


De nuevo, parece que ha convergido. Si fueran a tomar este número, mil, y aumentar a un promedio de más de 5000 ejemplos, entonces es posible que pudieran obtener una curva más suave que se parece más a esto. Y al promediar sobre estos, digamos 5000 ejemplos en lugar de 1000, podrían ser capaces de obtener una curva más suave como esta. Y eso es el efecto de aumentar el número de ejemplos sobre los cuales promediar.

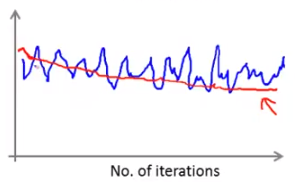


La desventaja de hacer esto demasiado grande, es que, por supuesto, ahora tienen un punto de datos sólo cada 5,000 ejemplos. Así que la retroalimentación que obtienen sobre lo bien que está funcionando su algoritmo de aprendizaje es tal vez más retrasada debido a que obtuvieron un punto de datos en su gráfico sólo cada 5,000 ejemplos, en vez de cada 1,000 ejemplos.

Siguiendo una línea de pensamiento similar, algunas veces pudieran ejecutar un gradiente de descenso estocástico y terminar con un gráfico que se parece a esto.

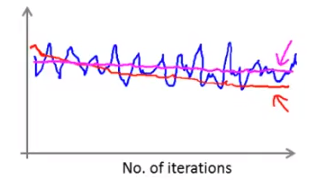


Y con un gráfico que se parezca a esto, ya saben, parece que el costo simplemente no disminuye en absoluto, parece que el algoritmo sencillamente no está aprendiendo. Sólo parece que aquí hay una curva plana y que el costo no está disminuyendo. Pero, de nuevo, si aumentaran esto para promediar sobre un mayor número de ejemplos, es posible que observen algo como esta línea roja,



Parece que el costo realmente está disminuyendo, es sólo que la línea azul con un promedio de más de 2, 3 ejemplos, la línea azul era demasiado ruidosa, así que no se podía ver la tendencia real del costo disminuyendo en realidad y posiblemente promediar más de 5,000 ejemplos, en lugar de 1,000, pudiera ayudar.

Por supuesto, cuando promediamos sobre un número de ejemplos más grande, si promediamos sobre 5,000 ejemplos, sólo estoy utilizando un color diferente, también es posible que vean una curva de aprendizaje que termina viéndose así.



Que sigue siendo plana, incluso cuando promedian sobre un número mayor de ejemplos. Y si obtienen eso, entonces eso es una verificación más firme de que, desafortunadamente, el algoritmo no está aprendiendo mucho por alguna razón. Y necesitan cambiar el índice de aprendizaje, o cambiar las variables, o cambiar algo más sobre el algoritmo.

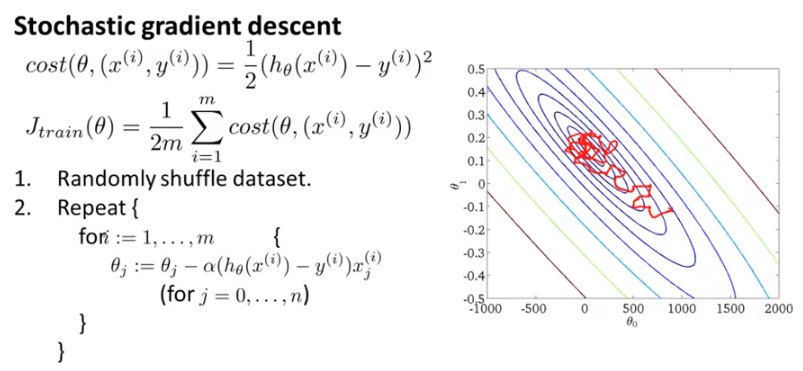
Finalmente, una última cosa que pudieran observar si tuvieran que trazar estas curvas y ven una curva que se ve así,



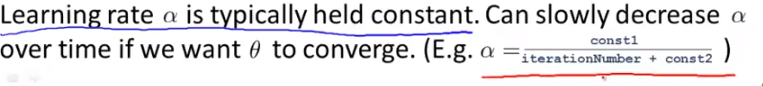
En donde realmente parece que está aumentando, y si ese es el caso, entonces esto es una señal de que el algoritmo es divergente. Y lo que realmente deben hacer es utilizar un valor menor de la «alfa» del índice de aprendizaje.

Así que espero que esto les dé un sentido de los fenómenos que pudieran ver cuando trazan este promedio de costos sobre algún rango de ejemplos, al igual que sugerir el tipo de cosas que pudieran tratar de hacer cuando ven diferentes gráficos. De modo que si los gráficos parecen demasiado ruidosos, o si se menea demasiado hacia arriba y hacia abajo, entonces traten de aumentar el número de ejemplos sobre los que están promediando, de modo que puedan ver mejor la tendencia general en el gráfico. Y si ven que los errores están en realidad aumentando, los costos están aumentando, traten de utilizar un valor de «alfa» más pequeño.

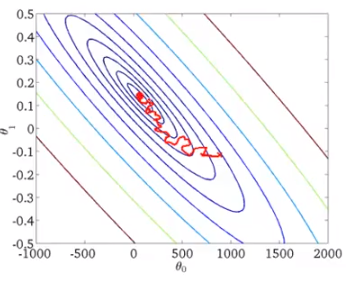
Por último, vale la pena examinar la cuestión del índice de aprendizaje un poco más. Vimos que cuando ejecutamos el gradiente de descenso estocástico, el algoritmo comienza aquí y hará un tipo de serpenteo hacia el mínimo, y entonces realmente no convergerá y, en su lugar, va a deambular alrededor de la mínima por siempre. Así que terminan con un valor de parámetro que con suerte está cerca del global mínimo pero que no estará exactamente en el mínimo global.



En las implementaciones más típicas del gradiente de descenso estocástico, el índice de aprendizaje «alfa» suele mantenerse constante. Así que con lo que terminamos, es una imagen como esta. Si desean que el gradiente de descenso estocástico converja realmente al mínimo global, hay una cosa que pueden hacer, que es que pueden disminuir lentamente el índice de aprendizaje «alfa» a través del tiempo. Por lo tanto, una forma muy típica de hacerlo sería establecer «alfa» es igual a cierta constante 1, dividida por el número de iteración, más la constante 2.



De modo que el número de iteración es el número de iteraciones que han ejecutado del gradiente de descenso estocástico, en realidad es el número de ejemplos de entrenamiento que han visto, y la const 1 y la const 2 son los parámetros adicionales del algoritmo con los que tuviera que jugar un poco a fin de obtener un buen desempeño. Una de las razones por la que las personas tienden a no hacer esto es porque terminan teniendo que invertir tiempo jugando con estos 2 parámetros adicionales, la constante 1 y la constante 2, y esto hace que el algoritmo sea más meticuloso. Ya saben, son sólo más parámetros con los que se puede juguetear a fin de hacer que el algoritmo funcione bien. Pero si consiguen sintonizar bien los parámetros, entonces la imagen que pueden obtener es que el algoritmo realmente serpenteará alrededor hacia el mínimo, pero a medida que se acerca debido a que están disminuyendo el índice de aprendizaje, el serpenteo se hará cada vez más pequeño, hasta que prácticamente simplemente converge hacia el mínimo global.



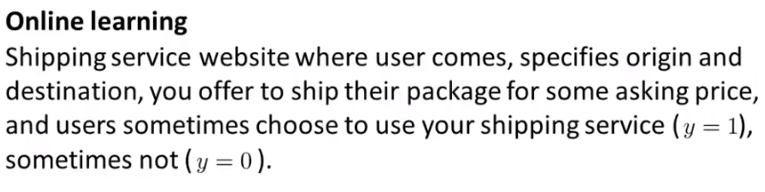
Espero que esto tenga sentido, ¿cierto? Y la razón por la que esta fórmula tiene sentido es que, a medida que se ejecuta el algoritmo, el número de iteración se hace grande, de manera que «alfa» se hará lentamente pequeño, así que tomarán pasos más y más pequeños hasta que, con suerte, converjan hacia el global mínimo. Así que si disminuyen lentamente «alfa» a cero, pueden terminar con una hipótesis ligeramente mejor. Pero debido al trabajo extra que se necesita para juguetear con las constantes, y porque, francamente, generalmente nos ponemos muy contentos con cualquier valor de parámetro que esté, ya saben, muy cerca del mínimo global, normalmente, este proceso de disminución lenta de «alfa» generalmente no se hace y mantener constante el índice de aprendizaje «alfa» es la aplicación más común del gradiente de descenso estocástico, aunque verán personas que usan cualquiera de las versiones.

En resumen, en este video hablamos de una manera para monitorear de manera aproximada cómo está funcionando el gradiente de descenso estocástico en términos de la optimización de la función de costos. Y este es un método que no requiere de exploración en todo el conjunto de entrenamiento de manera periódica para calcular la función de costos en todo el conjunto de entrenamiento. Pero en vez de eso, mira, por decir, sólo los últimos mil ejemplos más o menos. Y pueden utilizar este método ya sea para asegurarse de que el gradiente de descenso estocástico está bien y que está convergiendo, o utilizarlo para ajustar el índice de aprendizaje «alfa».

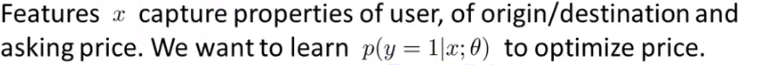
# Temas avanzados

## Online learning

En este vídeo me gustaría hablar acerca de una nueva configuración del aprendizaje automático a gran escala que se llama configuración de aprendizaje en línea. La configuración de aprendizaje en línea nos permite modelar problemas en donde tenemos una inundación continua o un flujo continuo de datos entrantes y nos gustaría un algoritmo que aprenda a partir de eso. Hoy en día, muchos de los sitios web más grandes, o muchas de las empresas de sitios web más grandes utilizan diferentes versiones de algoritmos de aprendizaje en línea para que aprendan de la avalancha de usuarios que siguen llegando, de regreso al sitio web. Específicamente, si tienen un flujo continuo de datos generado por un flujo continuo de usuarios que visitan su sitio web, lo que puede hacer es usar, algunas veces, un algoritmo de aprendizaje en línea para conocer las preferencias de los usuarios a partir del flujo de datos y usar eso para optimizar algunas de las decisiones sobre su sitio web.   
  
Supongamos que dirigen un servicio de envíos. Los usuarios vienen y piden ayuda para enviar sus paquetes de la ubicación de A a la ubicación B, y supongamos que dirige un sitio web, en donde los usuarios repetidas veces entran y les dicen desde dónde quieren enviar el paquete y a dónde desean enviarlo (así que el origen y el destino), y su sitio web ofrece enviar el paquete por algún precio de venta, así que yo voy a enviar el paquete por $ 50, yo lo enviaré por $20. Y con base en el precio que ustedes les ofrecen a los usuarios, los usuarios algunas veces eligen utilizar su servicio de envíos; eso es un ejemplo positivo y algunas veces se van y deciden no usar su servicio de envíos.



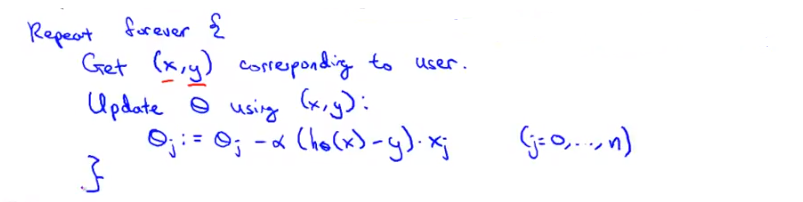
Así que digamos que queremos un algoritmo de aprendizaje para que nos ayude a optimizar lo que es el precio de venta que queremos ofrecer a nuestros usuarios. Y específicamente, digamos que nos encontramos con algún conjunto de variables que capturen las propiedades de los usuarios. Si conocemos algo de las variables demográficas, estas capturan el origen y el destino del paquete, a dónde quieren enviar el paquete y cuál es el precio que les ofrecemos para enviar el paquete, y lo que queremos hacer es conocer cuál es la la probabilidad de que elijan enviar el paquete usando nuestro servicio de envío dadas estas variables,



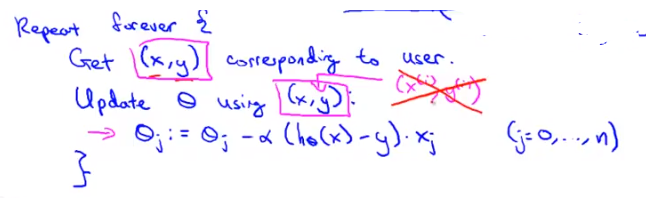
Y nuevamente sólo como recordatorio, estas variables «X» también capturan el precio que estamos pidiendo. De modo que si pudiéramos estimar la probabilidad de que vayan a estar de acuerdo en usar nuestro servicio por cualquier precio dado; entonces podemos tratar de elegir un precio para que tengan una probabilidad bastante alta de elegir nuestro sitio web mientras que, de manera simultánea, con suerte nos ofrezcan una retribución justa, ofreciéndonos una ganancia justa por enviar su paquete.

Así que si podemos conocer esta propiedad de «y» es igual a 1 dado cualquier precio y dadas las otras variables, realmente podríamos usar esto para elegir los precios adecuados a medida que tenemos usuarios nuevos. Así que para modelar la probabilidad de que «y» es igual a 1, lo que podemos hacer es usar la regresión logística o la red neuronal o algún otro algoritmo como ese. Pero vamos a empezar con la regresión logística.   
Ahora, si tienen un sitio web que sólo funciona de forma continua, esto es lo que un algoritmo de aprendizaje en línea haría. Voy a escribir «repetir por siempre.» Esto sólo significa que nuestro sitio web va a seguir manteniéndose en funcionamiento. Lo que sucede en el sitio web es que, de vez en cuando, un usuario vendrá, y para el usuario que viene vamos a obtener algún par «x,y» que corresponde a un cliente o a un usuario en el sitio web. De modo que las variables «x» son, ya saben, el origen y el destino especificado por este usuario y el precio que le ofrecimos en esta ocasión, y «y» es ya sea uno o cero, dependiendo de si eligieron o no usar nuestro servicio de envíos.

Ahora, una vez que tenemos este par {x, y}, lo que hace un algoritmo de aprendizaje en línea es entonces actualizar los parámetros «theta» usando sólo este ejemplo «x,y» y, en particular, actualizaremos los parámetros «theta» ya que «theta» j se actualiza como «theta» j menos el índice de aprendizaje «alfa» multiplicado por mi regla de gradiente de descenso habitual para la regresión logística. Así que hacemos esto para «j» es igual a cero hasta n, y ese es mi corchete cerrado.

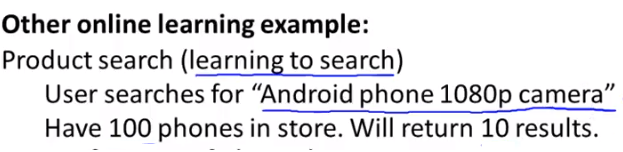


Así que para otros algoritmos de aprendizaje, en lugar de escribir «X-Y», cierto, escribí cosas como «Xi, Yi» pero en esta configuración de aprendizaje en línea en realidad estamos desechando la noción de la existencia de un conjunto de entrenamiento fijo, en su lugar tenemos un algoritmo. Ahora, lo que ocurre a medida que obtenemos un ejemplo y luego aprendemos a usar ese ejemplo de esta manera, y luego tiramos ese ejemplo, descartamos ese ejemplo y nunca lo usamos de nuevo, de modo que sólo miramos a un ejemplo a la vez; aprendemos de ese ejemplo, lo descartamos, que es la razón por la que, ya saben, también estamos acabando con esta noción de la existencia de este tipo de conjunto de entrenamiento fijo indexado por «i».

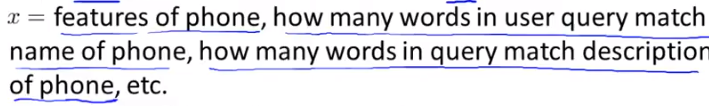


Y, si realmente dirigen un sitio web importante en donde en realidad tienen un flujo continuo de usuarios que lo visitan, entonces este tipo de algoritmo de aprendizaje en línea es en realidad un algoritmo bastante razonable, porque los datos están esencialmente gratis, si tienen demasiados datos, esos datos son esencialmente ilimitados, entonces en realidad tal vez no hay necesidad de mirar un ejemplo de entrenamiento más de una vez. Por supuesto, si sólo tuviéramos un número reducido de usuarios, entonces en lugar de utilizar un algoritmo de aprendizaje en línea como éste, quizás sería mejor que guardaran todos sus datos en un conjunto de entrenamiento fijo y después ejecutaran algún algoritmo sobre ese conjunto de entrenamiento. Pero si realmente tienen un flujo continuo de datos, entonces un algoritmo de aprendizaje en línea puede ser muy eficaz. También debo mencionar que un efecto interesante de este tipo de algoritmo de aprendizaje en línea es que puede adaptarse a las preferencias cambiantes del usuario.   
  
Y, en particular, si al paso tiempo, debido a cambios en la la economía, tal vez los usuarios se empiezan a hacer sensibles al precio y están dispuestos a pagar, ya saben, menos dispuestos a pagar precios más altos; O si se vuelven menos sensibles a los precios y están dispuestos a pagar precios más altos, O si diferentes asuntos llegan a ser más importantes para los usuarios; si empiezan a tener nuevos tipos de usuarios que visitan su sitio web; este tipo de algoritmo de aprendizaje en línea también se puede adaptar a las preferencias cambiantes del usuario y realizar un seguimiento de lo que su cambiante población de usuarios podría estar dispuesta a pagar. Y lo hace porque si su grupo de usuarios cambia, entonces estas actualizaciones a sus parámetros «theta» simplemente se adaptarán lentamente a sus parámetros para lo que sea que se parezca su último grupo de usuarios.

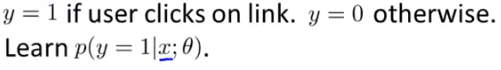
Aquí hay otro ejemplo de un tipo de aplicación a la que pueden aplicar el aprendizaje en línea. Esta es una aplicación en la búsqueda de producto en la que queremos aplicar el algoritmo de aprendizaje en línea para aprender a dar buenos listados de búsqueda a un usuario. Digamos que dirigen una tienda en línea que vende teléfonos - que vende teléfonos móviles o vende teléfonos celulares. Y tienen una interfaz de usuario en donde el usuario puede visitar su sitio web y escribir una consulta como "teléfono Android 1080p cámara". Así que 1080p es un tipo de una especificación para una cámara de vídeo que podrían tener en un teléfono, un teléfono celular, un teléfono móvil. Supongamos que tenemos un centenar de teléfonos en nuestra tienda. Y debido a la manera en la que nuestro sitio web está diseñado, cuando un usuario escribe una consulta, si se trata de una consulta de búsqueda, nos gustaría encontrar un elección de diez teléfonos diferentes para mostrar qué ofrecer al usuario. Lo que nos gustaría hacer es tener un algoritmo de aprendizaje que nos ayude a averiguar cuáles son los diez teléfonos, de los 100, que deberíamos regresar al usuario en respuesta a la consulta de búsqueda del usuario.



Así es cómo podemos abordar el problema. Para cada teléfono y dada una consulta específica de un usuario, podemos construir un vector de dirección “X”. De modo que el vector de dirección «X» puede capturar diferentes propiedades del teléfono. Puede capturar cosas como, cuán similar es la consulta de búsqueda del usuario con los teléfonos; Capturamos cosas como cuántas palabras en la consulta de búsqueda del usuario coinciden con el nombre del teléfono, cuántas palabras en la consulta de búsqueda del usuario coinciden con la descripción del teléfono, y así sucesivamente. De modo que las variables «x» capturan las propiedades del teléfono y captura cosas acerca de cuán similar o lo bien que el teléfono coincide con la consulta del usuario a lo largo de diferentes dimensiones.



Lo que nos gustaría hacer, es calcular la probabilidad de que un usuario haga clic en el enlace para un teléfono específico, porque le queremos mostrar al usuario los teléfonos que probablemente desee comprar, le queremos mostrar al usuario los teléfonos que tienen alta probabilidad de ser seleccionados con un clic en el navegador web. Así que voy a definir «y» es igual a uno si el usuario hace clic en el enlace para un teléfono, y «y» es igual a cero en caso contrario; y lo que me gustaría hacer es conocer la probabilidad de que el usuario hará clic en un teléfono determinado dadas, ya saben, las variables «x», que capturan las propiedades del teléfono y lo bien que la consulta coincide con el teléfono.



Para dar a este problema un nombre en el idioma de las personas que dirigen los sitios web como éste, el problema de aprender esto, se llama en realidad el problema de conocer el porcentaje de clics predicho, el CTR predicho. Simplemente significa conocer la probabilidad de que el usuario haga clic en el enlace específico que le están ofreciendo, así que CTR es una abreviatura para el porcentaje de clics. Y si pueden estimar el porcentaje de clics predicho para cualquier teléfono, lo que podemos hacer es usar esto para mostrarle al usuario los diez teléfonos sobre los cuales es más probable que hagan clic, porque de los cien teléfonos, podemos calcular esto para cada uno de los 100 teléfonos y sólo seleccionar los 10 teléfonos en los que es más probable que el usuario haga clic,



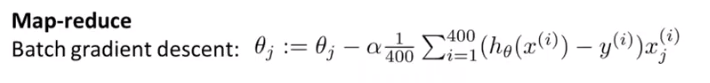
Y esto será una manera bastante razonable de decidir cuáles son los diez resultados para mostrar al usuario. Para que quede claro, supongamos que cada vez que un usuario realiza una búsqueda, regresamos diez resultados; lo que esto va a hacer es que, en realidad, nos dará diez pares «x, y», esto realmente nos da diez ejemplos de entrenamiento cada vez que un usuario entra a nuestro sitio web debido a los diez teléfonos que decidimos enseñarle al usuario, por cada uno de los 10 teléfonos obtenemos un vector de dirección «X» y para cada uno de esos 10 teléfonos, que le mostramos al usuario, también obtendremos un valor para «y», también observaremos el valor de «y», dependiendo de si hicimos clic o no en ese URL, así que una manera de dirigir un sitio web como este sería mostrarle de forma continua al usuario, ya saben, sus diez mejores estimaciones para otros teléfonos que les pudieran agradar así que, cada vez que un usuario los visite, tendrían diez ejemplos, diez pares «x, y» y después usarían un algoritmo de aprendizaje en línea para actualizar los parámetros utilizando esencialmente 10 pasos del gradiente de descenso sobre estos 10 ejemplos, y luego pueden eliminar los datos, y si realmente tienen un flujo continuo de usuarios que visitan su sitio web, esto sería una manera bastante razonable para conocer los parámetros para su algoritmo, de modo que podamos mostrar los diez teléfonos a sus usuarios que pudieran ser más prometedores y los más probables para que hagan clic en ellos.

Así que este es un problema de búsqueda de productos o un ejemplo de aprendizaje para clasificar teléfonos, aprender a buscar teléfonos. Así que voy a mencionar rápidamente algunos más. Uno es, si tienen un sitio web y están tratando de decidir, ya saben, cuál oferta especial mostrarle al usuario, esto es muy similar a los teléfonos, o si tienen un sitio web y le muestran a diferentes usuarios diferentes noticias; así que si son un sitio web que agrega noticias, entonces pueden nuevamente usar un sistema similar para seleccionar, para mostrarle al usuario, ya saben, cuáles son las noticias en las que sea más probable que estén interesados y cuáles son las noticias sobre las que es más probable que hagan clic. En estrecha relación con las ofertas especiales, tenemos recomendaciones de producto. Y de hecho, si tienen un sistema de filtrado colaborativo, pueden hasta imaginar un sistema de filtrado colaborativo que les proporcione variables adicionales para alimentar en un clasificador de regresión logística para tratar de predecir el índice de clic para los diferentes productos que pudieran recomendar a un usuario. Desde luego, debo decir que cualquiera de estos problemas también se podrían haber formulado como un problema de aprendizaje automático estándar, en el que tienen un conjunto de entrenamiento fijo. Tal vez podrían dirigir su sitio web durante unos días y después guardar un conjunto de entrenamiento, un conjunto de entrenamiento fijo y ejecutar un algoritmo de aprendizaje sobre eso. Pero estos son el tipo real de problemas, en donde observan a las grandes empresas obtener tanta cantidad de datos, que en realidad pudiera no haber necesidad de guardar un conjunto de entrenamiento fijo, sino en lugar de esto, pueden utilizar un algoritmo de aprendizaje en línea para simplemente aprender de manera continua a partir de los datos que los usuarios están generando en su sitio web.   
  
Así que, esa fue la configuración de aprendizaje en línea y como vimos, el algoritmo que aplicamos a ésta es realmente muy similar al algoritmo del gradiente de descenso estocástico, sólo que en lugar de buscar a través de un conjunto de entrenamiento fijo, estamos obteniendo un ejemplo de un usuario, aprendiendo de ese ejemplo, y luego descartándolo y siguiendo adelante. Y si tienen un un flujo continuo de datos para alguna aplicación, valdría la pena tomar en cuenta este tipo de algoritmo para su aplicación. Y, por supuesto, una ventaja del aprendizaje en línea es que también, si tienen un grupo cambiante de usuarios, o si las cosas que están tratando de predecir están cambiando lentamente, si el gusto de sus usuarios está cambiando poco a poco, el algoritmo de aprendizaje en línea puede adaptarse poco a poco a sus hipótesis aprendidas para cualesquiera que sean los últimos conjuntos de comportamientos de los usuarios también.

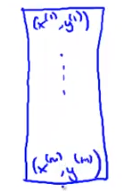
## Map reduce y paralelismo de datos

En los últimos videos, hemos hablado acerca del gradiente de descenso estocástico y, ya saben, otras variaciones del algoritmo de gradiente de descenso estocástico, incluidas las adaptaciones para el aprendizaje en línea, pero todos estos algoritmos pueden ejecutarse en una máquina, se pueden ejecutar en una computadora. Y algunos de los problemas del aprendizaje automático son demasiado grandes para que los ejecuten en una máquina, a veces simplemente tienen tantos datos, que simplemente no quieren volver a ejecutar todos esos datos a través de una sola computadora, sin importa qué algoritmo usarían en esa computadora. Así que este video me gustaría hablar de un enfoque diferente para el aprendizaje automático a gran escala que se llama el enfoque de MapReduce. Y aunque tenemos bastantes vídeos sobre el gradiente de descenso estocástico, y vamos a pasar relativamente menos tiempo con el MapReduce -no juzguen la importancia relativa del MapReduce contra el gradiente de descenso basándose en la cantidad de tiempo que paso en estas ideas. En particular, mucha gente dirá que el MapReduce es al menos igualmente importante, y algunos que es una idea aún más importante, en comparación con el gradiente de descenso, sólo que es relativamente más fácil de explicar, por lo que voy a pasar menos tiempo en él, pero usando estas ideas podrán ser capaces de ampliar los algoritmos de aprendizaje para problemas mucho más grandes de lo que es posible cuando usan el gradiente de descenso estocástico. Esta es la idea.

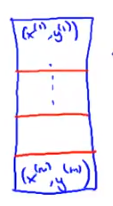
Digamos que queremos ajustar un modelo de regresión lineal o un modelo de regresión logística o alguno de éstos; y vamos a empezar de nuevo con el gradiente de descenso por lotes, así que esa es nuestra regla de aprendizaje con el gradiente de descenso por lotes.



Y para mantener la escritura de esta diapositiva manejable, voy a suponer en todo, que tenemos «m» es igual a 400 ejemplos. Desde luego, por nuestros estándares, en términos del aprendizaje automático a gran escala, ustedes saben que «m» puede ser muy pequeña y por eso, esto se podría aplicar más comúnmente a problemas en los que podrían tener tal vez cerca de 400 millones de ejemplos, o algo como eso, pero sólo para hacer que la escritura en la diapositiva sea más sencilla, voy a fingir que tenemos 400 ejemplos. En este caso, la regla de aprendizaje con el gradiente de descenso por lotes tiene este 1 sobre 400 y la suma de “i” es igual a 1 a través de 400, mis 400 ejemplos aquí, y si «m» es grande, entonces este es un paso costoso computacionalmente.   
  
Así que lo que hace la idea del MapReduce es lo siguiente, y debo decir que la idea del MapReduce se debe a Dos investigadores, Jeff Dean y Sanjay Gimawat. Jeff Dean, por cierto, es uno de los ingenieros más legendarios en todo Silicon Valley y el construyó una gran porción de la infraestructura arquitectónica sobre la que funciona Google en la actualidad.   
  
Pero aquí está la idea de MapReduce. Así que digamos que tengo algunos conjuntos de entrenamiento, si deseamos indicarlo mediante este cuadro aquí de pares «X Y»,



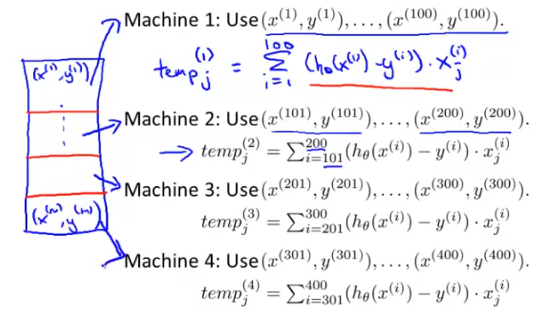
donde está «X1», «Y1», bajando a mis 400 ejemplos «Xm», «Ym». Así que ese es mi conjunto de entrenamiento con 400 ejemplos de entrenamiento.   
  
En la idea del MapReduce, una manera de hacerlo es dividir este conjunto de entrenamiento en diferentes subconjuntos.   
  
Voy a suponer para este ejemplo que tengo 4 computadoras, o 4 máquinas para correr en paralelo en mi conjunto de entrenamiento, que es por lo que estoy dividiendo esto en 4 máquinas.



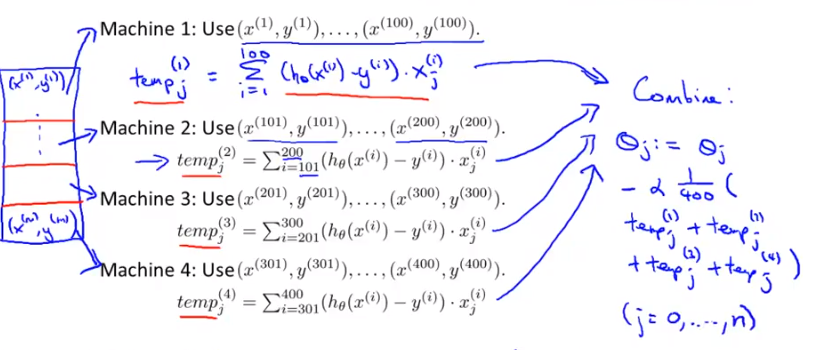
Si tienen 10 máquinas, o 100 máquinas, entonces dividirán su conjunto de entrenamiento en 10 piezas, o 100 piezas, o lo que sea.   
  
Y lo que debe hacer la primera de mis 4 máquinas es, por decir, utilizar sólo el primer cuarto de mi conjunto de entrenamiento -así que usar sólo los primeros 100 ejemplos de entrenamiento. Y, en particular, lo que va a hacer es mirar esta suma y ​​calcular esa suma sólo para los primeros 100 ejemplos de entrenamiento.



Así que permítanme escribir eso. Voy a calcular una variable temp 1 al superíndice 1, que indica la primera máquina, «j» igual a la suma de “i” es igual a 1 hasta 100. Y luego, de manera similar, voy a tomar el segundo cuarto de mis datos y los voy a enviar a mi segunda máquina, y mi segunda máquina usará los ejemplos de entrenamiento 101 a 200 y calcularán las variables similares temp 2 "j", eso es la misma suma para el índice de los ejemplos 101 a 200. Y del mismo modo, las máquinas 3 y 4 usarán el tercer cuarto y el último cuarto de mi conjunto de entrenamiento.



Así que ahora cada máquina tiene que sumar sobre 100, en lugar de sobre 400 ejemplos, así que sólo tiene que hacer un cuarto del trabajo y, por tanto, presuntamente, podría hacerlo aproximadamente cuatro veces más rápido. Finalmente, después de que todas estas máquinas han realizado este trabajo, voy a tomar estas variables temp y las voy a unir de nuevo. De modo que tomo estas variables y las envío al, ya saben, el servidor maestro centralizado y lo que hará el servidor maestro es combinar estos resultados juntos y, en particular, actualizará mis parámetros «theta» «j» de acuerdo con «theta» «j» se actualiza como «theta» «j» menos el índice de aprendizaje «alfa» multiplicado por uno sobre 400 veces temp, 1, «j», más temp «2j», más temp «3j», más temp «4j» y, desde luego, tenemos que hacer esto por separado para «j» es igual a 0 hasta n donde n es el número de variables.

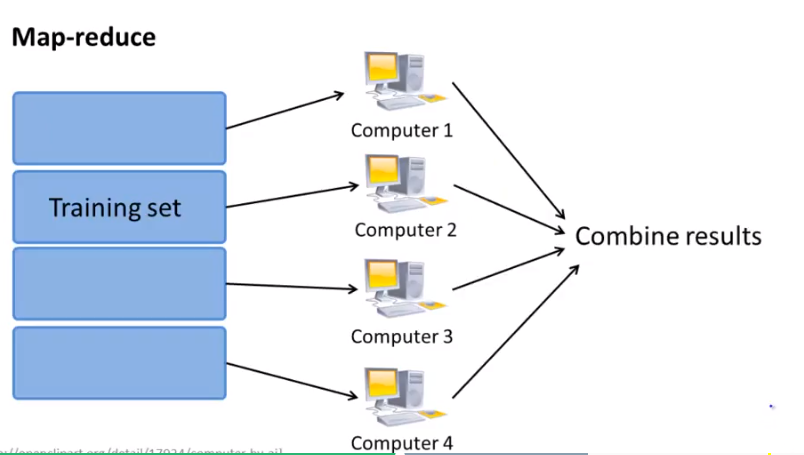


Discúlpenme por desglosar esta ecuación en múltiples líneas pero espero que quede claro. Así que lo que esta ecuación está haciendo es exactamente lo mismo que:



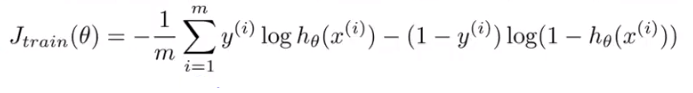
Cuando tienen un servidor maestro centralizado que toma los resultados, los temp 1«j», los temp 2«j», temp 3«j» y temp 4«j» y los suma y, por supuesto, la suma de estas cuatro cosas, ¿cierto? eso es sólo la suma de esto, más la suma de esto, más la suma de esto, más la suma de eso, y esas cuatro cosas sencillamente se suman para ser iguales a esta suma que estábamos calculando originalmente con el descenso de corriente por lotes. Y luego tenemos «alfa» multiplicado por 1 de 400, «alfa» multiplicado por 1 de 100, y esto es exactamente equivalente al algoritmo de gradiente de descenso por lotes, sólo que en lugar de tener que sumar sobre todos los cuatrocientos ejemplos de entrenamiento en una sola máquina, podemos dividir la carga de trabajo en cuatro máquinas.   
  
Así que aquí está la imagen general de cómo se ve la técnica de MapReduce.   
  
Tenemos algunos conjuntos de entrenamiento, y si queremos paralelizar a través de cuatro máquinas, vamos a tomar el conjunto de entrenamiento y lo vamos a dividir, ya saben, por igual, dividirlo tan uniformemente como sea posible en cuatro subgrupos.

Después, vamos a tomar los 4 subconjuntos de los datos de entrenamiento y a enviarlos a 4 computadoras diferentes. Y cada una de las 4 computadoras puede calcular una suma sobre sólo un cuarto del conjunto de aprendizaje y luego, finalmente, cada una de las computadoras toma los resultados, los envía a un servidor centralizado, que entonces combina los resultados juntos.

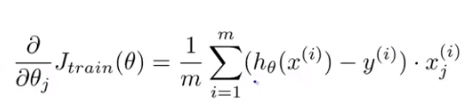


Así que en la línea anterior en ese ejemplo, el grueso del trabajo en el gradiente de descenso fue calcular la suma de «i» es igual a 1 a 400 de algo. Así, de manera más general, la suma de «i» igual a 1 hasta «m» de esa fórmula para el gradiente de descenso. Y ahora, debido a que cada una de las cuatro computadoras puede hacer sólo una cuarta parte del trabajo, potencialmente pueden conseguir hasta un aceleramiento de hasta 4x.   
  
En particular, si no existieran latencias de la red y ningún costo de las comunicaciones de la red para enviar los datos de ida y vuelta, podrían conseguir potencialmente un aceleramiento de hasta 4x. Por supuesto, en la práctica, debido a las latencias de la red, la sobrecarga por combinar los resultados después y otros factores, en la práctica obtienen un aceleramiento de poco menos de 4x. Pero, sin embargo, este tipo de enfoque de MapReduce nos ofrece una manera para procesar conjuntos de datos mucho más grandes de lo que es posible utilizando una sola computadora.

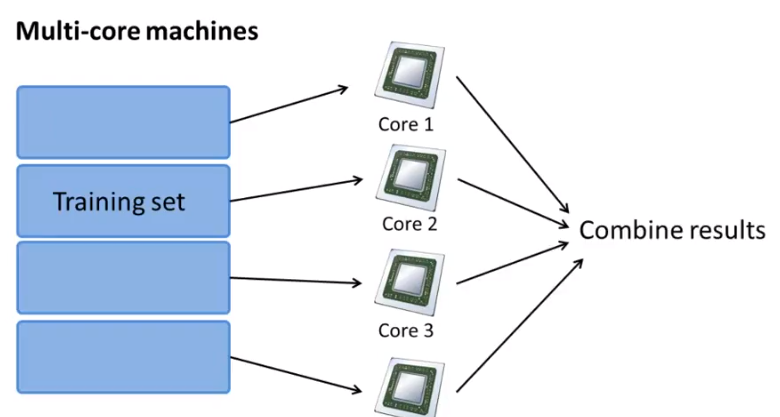
Si están pensando en aplicar el MapReduce a algunos algoritmos de aprendizaje, a fin de acelerar este proceso por medio de paralelizar el cálculo sobre diferentes computadoras, la pregunta clave que deben hacerse es, ¿puede su algoritmo de aprendizaje expresarse como una suma sobre el conjunto de entrenamiento? Y resulta que muchos algoritmos de aprendizaje se pueden expresar en realidad como el cálculo de sumas de funciones sobre el conjunto de entrenamiento y el costo computacional de ejecutarlas en grandes conjuntos de datos es porque tienen que sumar sobre un conjunto de entrenamiento muy grande. De modo que, siempre que su algoritmo de aprendizaje se pueda expresar como una suma sobre el conjunto de entrenamiento, y siempre que el grueso del trabajo del algoritmo de aprendizaje se pueda expresar como la suma del conjunto de entrenamiento, entonces MapReduce puede ser un buen candidato para ampliar sus algoritmos de aprendizaje a través de conjuntos de datos muy grandes.   
  
Veamos un ejemplo más.   
  
Digamos que queremos usar uno de los algoritmos de optimización avanzada. Asi que cosas como LBFGS, la constante gradiente, etcétera. Y digamos que queremos entrenar un algoritmo de aprendizaje de regresión logística. Para eso, tenemos que calcular dos cantidades principales. Una es para los algoritmos de optimización avanzada como, ya saben, LBFGS y el gradiente constante. Tenemos que proporcionarle una rutina para calcular la función de costos del objetivo de optimización, así que para la regresión logística, recuerdan que una función de costos tiene esta especie de suma sobre el conjunto de entrenamiento, así que si están paralelizando sobre diez máquinas, dividirían el conjunto de entrenamiento en diez máquinas y hacen que cada una de las diez máquinas calcule la suma de esta cantidad sobre sólo un décimo de los datos dimensiones.



Entonces, algo más que necesita el algoritmo de optimización avanzada es una rutina para calcular estos términos derivados parciales. Una vez más, estos términos derivados, para la regresión logística, se pueden expresar como una suma sobre el conjunto de entrenamiento, así que una vez más, similar a nuestro ejemplo anterior, harían que cada máquina calculara esa suma sobre solamente una pequeña fracción de sus datos de entrenamiento.



Y, por último, después de haber calculado todas estas cosas, podrían entonces enviar sus resultados a un servidor centralizado, que puede entonces sumar las sumas parciales. Esto corresponde a sumar esas variables temp «i» o temp «ij», que se calcularon localmente en la máquina número i, y así el servidor centralizado puede sumar estas cosas y obtener la función de costos general y obtener la derivada parcial general, la cual pueden pasar entonces a través del algoritmo de optimización avanzada.   
Así que, en general, al tomar otros algoritmos de aprendizaje y expresarlos en una especie de forma sumatoria, o por medio de expresarlos en términos de calcular sumas de funciones sobre el conjunto de entrenamiento, pueden utilizar la técnica de MapReduce para paralelizar otros algoritmos de aprendizaje también, y ampliarlos a grandes conjuntos de entrenamiento.   
  
Por último, como un último comentario, hasta ahora hemos estado analizando los algoritmos de MapReduce que les permiten paralelizar sobre varias computadoras, tal vez múltiples computadoras en un clúster de computadoras, o sobre múltiples computadoras en el centro de datos.   
  
Resulta que a veces incluso si tienen sólo una computadora, MapReduce también puede ser aplicable. En particular, en muchas computadoras individuales actualmente, pueden tener múltiples núcleos de procesamiento. Puede tener varias CPU, y dentro de cada CPU pueden tener múltiples núcleos de procesamiento. Si tienen un gran conjunto de entrenamiento, lo que pueden hacer si, por ejemplo, tienen una computadora con 4 núcleos de procesamiento, lo que pueden hacer es, incluso en una sola computadora, pueden dividir los conjuntos de entrenamiento en pedazos y enviar el conjunto de entrenamiento a diferentes núcleos dentro de una sola caja, como dentro de una sola computadora de escritorio, o un solo servidor, y usar MapReduce de esta forma para dividir la carga de trabajo. Cada uno de los núcleos puede entonces realizar la suma sobre, por decir, un cuarto de su conjunto de entrenamiento y luego puede tomar las sumas parciales y combinarlas a fin de obtener la suma sobre todo el conjunto de entrenamiento.



La ventaja de pensar en MapReduce de esta manera, como paralelizando sobre los núcleos dentro de una sola máquina, en lugar de paralelizar sobre múltiples máquinas es que, de esta manera, no tienen que preocuparse por la latencia de la red, ya que toda la comunicación, todo el envío de las variables temp j de ida y vuelta, todo eso sucede dentro de una sola máquina. Y así la latencia de red se convierte en un problema mucho menor, si lo comparan con el hecho de usar esto para paralelizar sobre diferentes ordenadores dentro del centro de datos.

Finalmente, un último detalle sobre la paralelización dentro de una máquina con múltiples núcleos. Dependiendo de los detalles de su implementación, si tienen una máquina de múltiples núcleos y si tienen ciertas bibliotecas de álgebra lineal numérica. Resulta que la suma de las bibliotecas de álgebra lineal numérica puede paralelizar automáticamente sus operaciones de álgebra lineal a través de múltiples núcleos dentro de la máquina.   
  
Así que si tienen la suerte de usar una de esas bibliotecas de álgebra lineal numérica, y ciertamente esto no aplica a todas y cada una de las bibliotecas, si están usando una de esas bibliotecas y si tienen una muy buena implementación de vectorización del algoritmo de aprendizaje, algunas veces simplemente pueden implementar su algoritmo estándar de aprendizaje de una forma vectorizada y no preocuparse por la paralelización, y las bibliotecas de álgebra lineal numérica pueden hacerse cargo de algunos de ellos por ustedes. Así que no necesitan implementar MapReduce pero, para otros problemas de aprendizaje, aprovecharse de este tipo de aplicación Map Reduce, encontrar y usar estas formulas de MapReduce y paralelizar a través de los núcleos expresamente por sí mismos, podría ser una buena idea también y le podría permitir acelerar su algoritmo de aprendizaje.   
  
En este vídeo hablamos acerca del enfoque MapReduce para paralelizar el aprendizaje automático por medio de tomar los datos y extenderlos a través de muchas computadoras en el centro de datos. Aunque estas ideas son aplicables para paralelizar a través de múltiples núcleos dentro de una sola computadora. Actualmente existen muchas buenas implementaciones de código abierto de MapReduce, así que hay muchos usuarios en el sistema de código abierto llamado Hadoop, y usando ya sea su propia implementación, o usando la implementación de código abierto de alguien más, pueden usar estas ideas para paralelizar los algoritmos de aprendizaje y conseguir que se ejecuten sobre conjuntos de datos mucho más grandes posible utilizando una sola máquina.