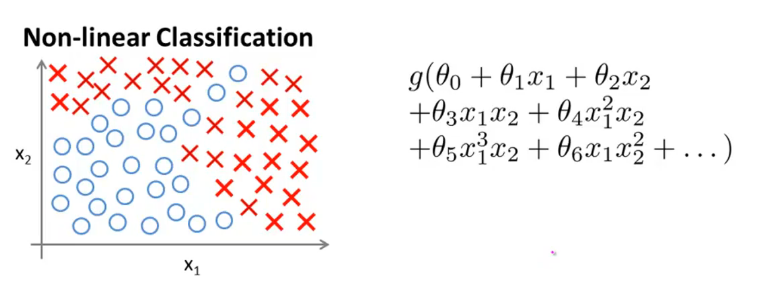
# Redes Neuronales

## Motivación

Entonces, ¿por qué necesitamos otro algoritmo de aprendizaje? Ya tenemos la regresión lineal y tenemos regresión logística, así que ¿por qué necesitamos las redes neuronales?

Con el fin de motivar la discusión en torno a las redes neuronales, voy a empezar por mostrarte algunos ejemplos de problemas de aprendizaje automático en los que necesitamos aprender hipótesis complejas no lineales.

Considera un problema de clasificación de aprendizaje supervisado en el que tienes un conjunto de entrenamiento como este:



Si quieres aplicar una regresión logística a este problema, una cosa que podrías hacer es aplicar una regresión logística con una gran cantidad de variables no lineales de este modo. Entonces aquí, como siempre, g es la función sigmoidea, y podemos incluir varios términos polinomiales como esos. Y, si incluyes suficientes términos polinomiales, quizás puedas obtener una hipótesis que separe los ejemplos positivos y negativos:

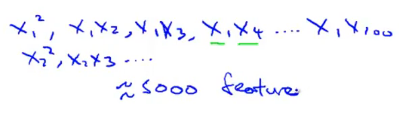


Este método particular funciona bien cuando sólo tienes, digamos, dos variables - x1 y x2 - porque entonces puedes incluir todos esos términos polinomiales de x1 y x2. Pero muchos problemas interesantes de aprendizaje automático tendrán muchas más variables.

Hemos estado hablando sobre una predicción de los precios de una vivienda, y supón que tienes un problema de clasificación de vivienda en lugar de un problema de regresión, quizás, como si tuvieras diferentes variables de una casa, y deseas predecir cuáles son las posibilidades de que tu casa se venda en los próximos seis meses, por lo que sería un problema de clasificación.

Y como vimos que podemos tener un gran número de variables, quizás cien variables diferentes de diferentes casas. Para un problema como este, si fueras a incluir todos los términos cuadráticos, todos esos, incluso todos los términos cuadráticos de segundo orden, habría muchos de ellos.

Habría términos como x1 al cuadrado, x1x2, x1x3, x1x4 hasta x1x100 y luego tienes 2x al cuadrado, x2x3 y así sucesivamente. Y sólo incluyendo los términos de segundo orden, esto es, los términos que son un producto de, ya sabes, dos de estos términos, x1 multiplicado por x1 y así sucesivamente, entonces, para el caso de n igual a 100, terminas con unas 5 mil variables.



Y, asintóticamente, el número de variables cuadráticas crece aproximadamente al orden de O(n2), en donde n es el número de variables originales, como 1x hasta x100, que teníamos anteriormente. Y esto, en realidad, está más cerca de .

Entonces, incluir todas las variables cuadráticas no parece una buena idea, porque son muchas variables y podrías terminar sobreajustando el conjunto de aprendizaje, y también puede ser computacionalmente costoso, trabajar con tantas variables.

Algo que puedes hacer es incluir sólo un subconjunto de éstos, entonces, si sólo incluyes las variables x1 al cuadrado, x2 al cuadrado, x3 al cuadrado, hasta, quizás, x100 al cuadrado, entonces el número de variables es mucho menor. Aquí sólo tienes 100 funciones cuadráticas como esta, pero ésta no tiene suficientes variables y, ciertamente, no te dejará ajustar el conjunto de datos como en la figura superior.

De hecho, si incluyes sólo estas funciones cuadráticas junto con el x1 original, y así sucesivamente, hasta x100 variables, entonces realmente podrás ajustar hipótesis muy interesantes como elipses, pero, ciertamente, no puedes ajustar un conjunto de datos más complejos como el que se muestra aquí.

Entonces, 5000 variables parecen demasiadas, si fueras a incluir las cúbicas o las conocidas de tercer orden de los demás:

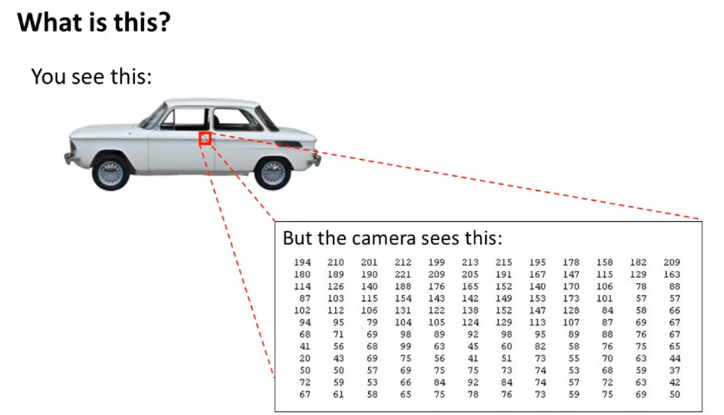


Puedes imaginar que habrá muchas de estas variables. De hecho, van a ser variables de orden O(n3), y para nuestro ejemplo de 100 variables, terminarás con el orden de aproximadamente unas 170,000 de estas variables cúbicas y, por lo tanto, incluyendo las variables auto-polinomiales más elevada, esto realmente aumenta de forma dramática tu espacio de variables y no parece una buena forma de proponer variables adicionales con las cuales construir clasificadores cuando n es grande.

Para muchos problemas de aprendizaje automático, n será bastante grande. Aquí hay un ejemplo:

Consideremos el problema de la visión por computadora. Y supongamos que quieres usar aprendizaje automático para entrenar a un clasificador para examinar una imagen y decirnos si la imagen es un auto o no. Mucha gente se pregunta por qué la visión por computadora puede ser complicada. Quiero decir, cuando tú y yo vemos una imagen, podemos fácilmente identificar lo que es. Te preguntas cómo es que un algoritmo de aprendizaje podría no saber lo que es esta imagen.

Para entender por qué la visión por computadora es complicada, hagamos un acercamiento a una pequeña parte de la imagen, como el área en la que está el pequeño rectángulo rojo:



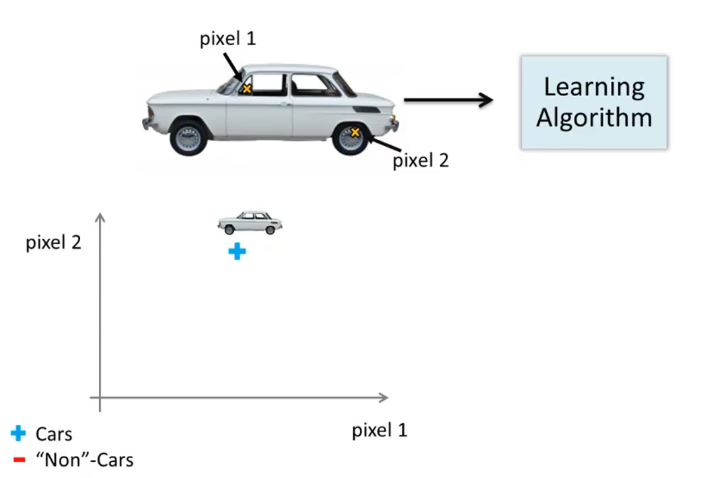
Resulta que cuando tú y yo vemos un auto, la computadora ve eso. Lo que ve es esta matriz, o esta cuadrícula, de valores de intensidades de pixeles que nos dice el brillo de cada pixel en la imagen. Entonces, el problema de la visión por computadora es ver esta matriz de valores de intensidad de pixeles y decirnos que esos números representan la manija de la puerta de un coche.

Concretamente, cuando utilizamos aprendizaje automático para construir un detector de autos, lo que hacemos es crear un conjunto de aprendizaje con valores asignados, con, digamos, algunos ejemplos de asignación de valores de autos y algunos ejemplos de asignación de valores de cosas que no son autos, entonces le damos nuestro conjunto de entrenamiento al algoritmo de aprendizaje al que se le enseñó un clasificador y, después, podemos probarlo y mostrar la nueva imagen y preguntar, "¿Qué es esta cosa nueva?".

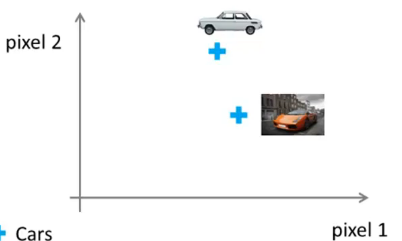


Y, con suerte, reconocerá que esto es un auto. Para entender por qué necesita hipótesis no lineales, vamos a ver algunas de las imágenes de autos y, no-autos que podamos introducir a nuestro algoritmo de aprendizaje.

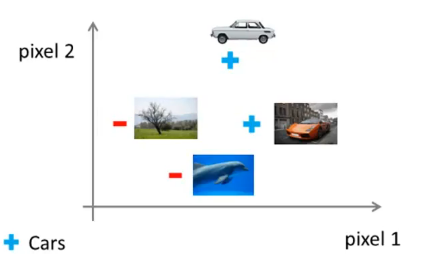
Vamos a elegir un par de ubicaciones de pixeles en nuestras imágenes, de forma que sea la ubicación del pixel uno y la ubicación del pixel dos, y vamos a dibujar este auto en la ubicación, en un determinado punto, dependiendo de las intensidades del pixel uno y del pixel dos.



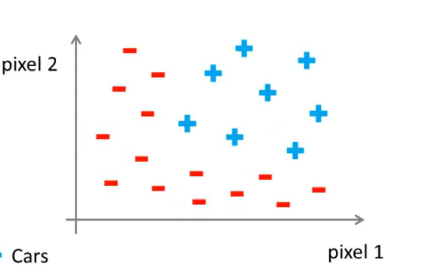
Y vamos a hacer esto con algunas otras imágenes. Ahora, tomemos un ejemplo diferente del auto, y veamos las mismas ubicaciones de los dos pixeles:

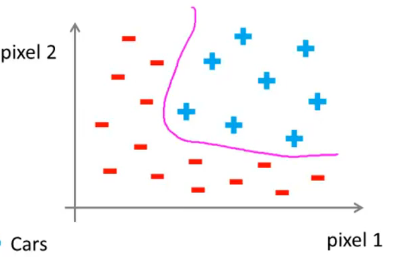


Y esa imagen tiene una diferente intensidad para el pixel uno y una diferente intensidad para el pixel dos. Entonces, termina siendo una ubicación diferente en la figura. Ahora vamos a trazar algunos ejemplos negativos también:

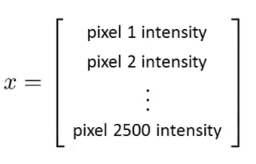


Y si hacemos esto para más y más ejemplos usando los positivos para denotar a los autos y los negativos para denotar a los no-autos, encontraremos que los autos y no-autos terminan en diferentes regiones del espacio, y lo que necesitamos, por lo tanto, es algún tipo de hipótesis no lineal para intentar separar las dos clases:





¿Cuál es la dimensión del espacio de variables? Supongamos que fuéramos a usar sólo imágenes de 50 por 50 pixeles tendríamos 2500 pixeles,



Así, la dimensión de nuestro tamaño de variable será , donde nuestro vector de variable x es una lista de todos los pixeles, brillo del pixel 1, el brillo del pixel dos, y así sucesivamente hasta el brillo del último píxel donde, en una representación computacional típica, cada uno de estos pueden ser valores entre, digamos, 0 a 255, si nos da el valor en escala de grises. Si estuviéramos usando imágenes en RGB, con valores separados para rojo, verde y azul, tendríamos n = 7500 ya que serían como 3 capas en lugar de una con la escala de grises, ya que cada pixel puede tomar 3 colores diferentes a la vez con distintos brillos.

Ahora, si quisiéramos tratar de aprender una hipótesis no lineal incluyendo todas las variables cuadráticas, esto es, todos los términos de la forma, , mientras que con los 2500 pixeles terminaríamos con un total de 3 millones de variables. Y esto es demasiado grande para ser razonable; el cálculo sería demasiado costoso de hacer para estos tres millones de variables en el conjunto de entrenamiento.

Entonces, una simple regresión logística junto con añadir, quizás, las variables cuadráticas o cúbicas - no es una buena manera de aprender hipótesis no lineales complejas cuando n es grande porque acabas con demasiadas variables.

En las siguientes secciones, vamos a hablar sobre las redes neuronales, que resulta ser una forma mucho mejor para aprender hipótesis complejas, hipótesis complejas no lineales, incluso cuando n es grande.

## Las neuronas y el cerebro

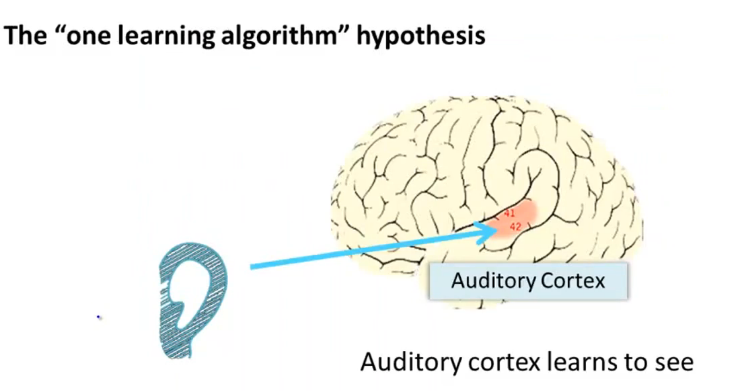
Las redes neuronales son un algoritmo bastante viejo que fue originalmente motivado por el objetivo de hacer máquinas que pudieran simular el cerebro. Ahora, en esta clase, por supuesto, voy a enseñarles sobre las redes neuronales porque funcionan muy bien para diferentes problemas de aprendizaje.

En esta sección me gustaría darles algunos de los antecedentes de las redes neuronales. Para que tengas una idea de lo que podemos esperar que hagan. Tanto en el sentido de aplicarlas a problemas de maquinaria actuales, así como para aquellos que te puedan interesar.

El origen de las redes neuronales fueron algoritmos que intentaban imitar al cerebro y de ahí viene el sentido de que, si queremos construir sistemas de aprendizaje, por qué no imitar la que es, quizás, la máquina de aprendizaje más asombrosa que conocemos, que posiblemente es el cerebro. Las redes neuronales se utilizaron ampliamente en los 1980’s y 1990’s y, por diversas razones, su popularidad disminuyó a finales de los de los 90’s. Pero, más recientemente, las redes neuronales han tenido un importante resurgimiento.

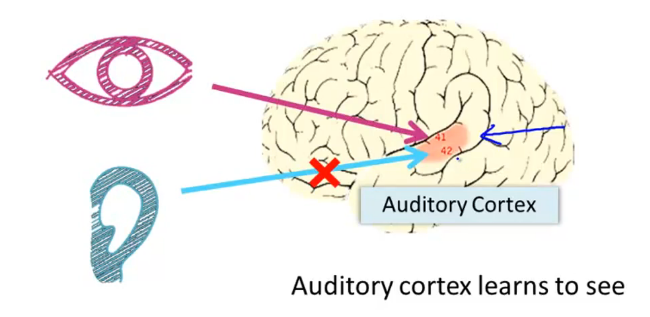
Una de las razones de este resurgimiento es que las redes neuronales son, computacionalmente, algunos de los algoritmos más costosos y recientemente las computadoras se volvieron lo suficientemente rápidas para realmente operar redes neuronales de gran escala y, por esto, así como por otras razones técnicas de las que hablaremos después, las redes neuronales modernas son la técnica más vanguardista para muchas aplicaciones.

Así que, cuando pensamos en imitar el cerebro, mientras que el cerebro humano puede hacer muchas cosas asombrosas, ¿cierto? El cerebro puede aprender a ver, procesar imágenes, y a escuchar, aprender a procesar el sentido del tacto. Podemos aprender a hacer operaciones matemáticas, aprender a hacer cálculos, y el cerebro hace muchas cosas diferentes y sorprendentes. Parece que si quieres imitar al cerebro, pareciera que debes escribir muchas partes diferentes de software para imitar todas estas cosas fascinantes y asombrosas que el cerebro hace, pero existe esta fascinante hipótesis de que la forma en la que el cerebro hace todas estas cosas diferentes no requiere miles de programas diferentes, sino que, en cambio, la forma en la que el cerebro lo hace requiere sólo un único algoritmo de aprendizaje. Esta es sólo una hipótesis pero déjame compartirte algunas evidencias de esto:



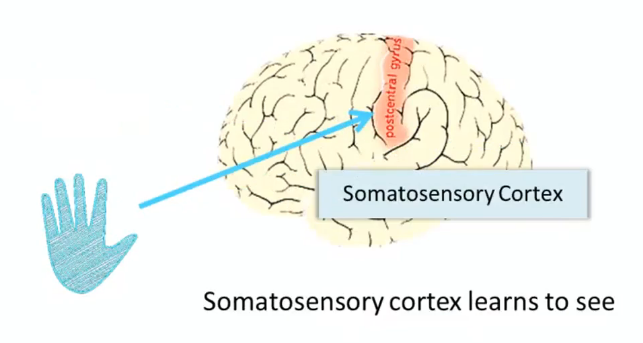
Esta parte del cerebro, esta pequeña parte roja del cerebro, es tu corteza auditiva y la forma en la que estás comprendiendo mi voz ahora es que tu oído está captando la señal auditiva y enrutando la señal auditiva a tu corteza auditiva, y eso es lo que te permite entender mis palabras.

Los neurocientíficos han hecho los siguientes experimentos fascinantes en los que se corta la conexión desde los oídos a la corteza auditiva, y se vuelve a conectar, en este caso, en el cerebro de un animal, de forma que la señal desde los ojos hacia el nervio óptico eventualmente se enrute hacia la corteza auditiva.

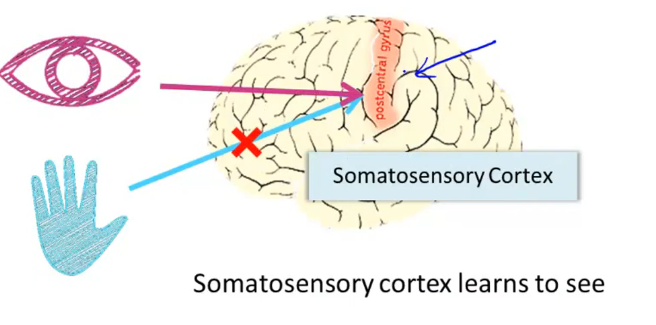


Si haces esto, resulta que la corteza auditiva aprenderá a ver. Y esto es, en toda la extensión de la palabra, ver tal y como lo conocemos. Entonces, si le haces esto a los animales, los animales pueden efectuar una tarea de discriminación visual, y entonces pueden ver imágenes y tomar decisiones apropiadas con base en las imágenes, y lo están haciendo con ese pedazo de tejido cerebral.

Aquí hay otro ejemplo:



Ese pedazo rojo de tejido cerebral es tu corteza somatosensorial. Ahí es donde procesas tu sentido del tacto. Si haces un proceso de reconexión similar entonces la corteza somatosensorial aprenderá a ver:

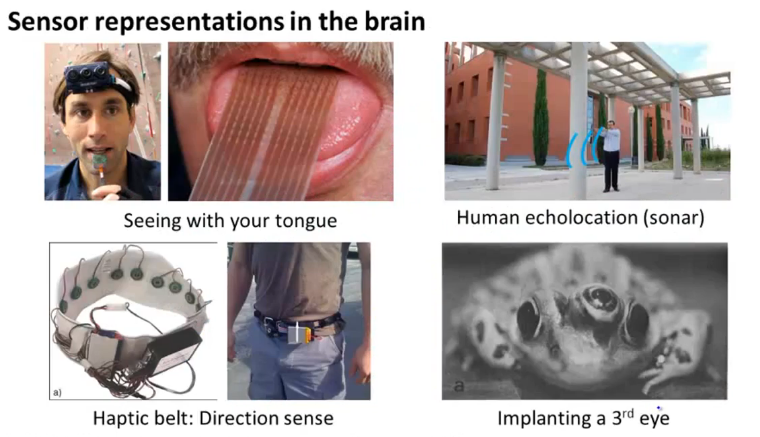


Debido a éste y a otros experimentos similares, se les llama experimentos de recableado neuronal.

Existe esta sensación de que, si la misma parte física de tejido cerebral puede procesar la vista o el sonido o el tacto, entonces quizás haya un algoritmo de aprendizaje que pueda procesar la vista o el sonido o el tacto. Y, en lugar de que sea necesario implementar un millar de programas diferentes o un millar de algoritmos diferentes para hacer, ya sabes, las miles de cosas maravillosas que hace el cerebro, quizás lo que necesitamos es encontrar una aproximación a cualquiera que sea el algoritmo de aprendizaje del cerebro e implementarlo, y que el cerebro aprenda por sí mismo cómo procesar estos diferentes tipos de datos.

En gran medida, sorprendentemente, pareciera que podemos conectar casi cualquier sensor a casi cualquier parte del cerebro y, dentro de lo razonable, el cerebro aprenderá a manejarlo.

Aquí hay algunos ejemplos más:



En la parte superior izquierda hay un ejemplo de aprendizaje pare ver con tu lengua. El modo de funcionamiento es ( esto es un sistema llamado BrainPort que está siendo sometido a actualmente a pruebas de la FDA para ayudar a ver a las personas ciegas) pero funciona con una cámara en escala de grises atada a tu frente, viendo hacia delante, que toma una imagen en escala de grises de baja resolución de lo que está frente a ti y, a continuación, se conecta un cable a una matriz de electrodos que colocas en tu lengua para que cada pixel sea mapeado en una ubicación en tu lengua, donde tal vez un alto voltaje corresponde a un pixel oscuro y un voltaje bajo corresponde a un pixel brillante y, a pesar de la forma en la que lo hace actualmente, con este tipo de sistemas tú y yo podremos aprender a ver en unos cuantos minutos con nuestras lenguas.

He aquí un segundo ejemplo de ecolocación humana, o de sonar humano.

Entonces, hay dos formas en las que puedes hacer esto. Puedes chasquear los dedos o chasquear la lengua. Yo no puedo hacerlo muy bien. Pero hay personas ciegas actualmente a las que se les está enseñando en las escuelas a hacer esto y a aprender a interpretar el patrón de sonidos que rebotan en su entorno - eso es un sonar. Así que, si después buscas en en YouTube, hay videos de este asombroso niño al que, trágicamente, debido al cáncer, fue necesario retirarle los globos oculares, así que este es un niño sin ojos. Pero, chasqueando los dedos, puede desplazarse sin chocar nunca con nada. Puede andar en patineta. Puede lanzar una pelota de baloncesto hacia el aro, y este es un niño sin globos oculares.

El Tercer ejemplo es el cinturón háptico, con el que, con una banda alrededor de su cintura, suenan zumbadores y siempre zumban para indicar la posición más al norte. Le puede dar a un humano un sentido de dirección similar al que quizás tienen las aves, ya sabes, el sentido de dónde está el norte. Y, algunos son ejemplos bizarros, pero si le conectas un tercer ojo a una rana, la rana aprenderá a usar ese ojo también.

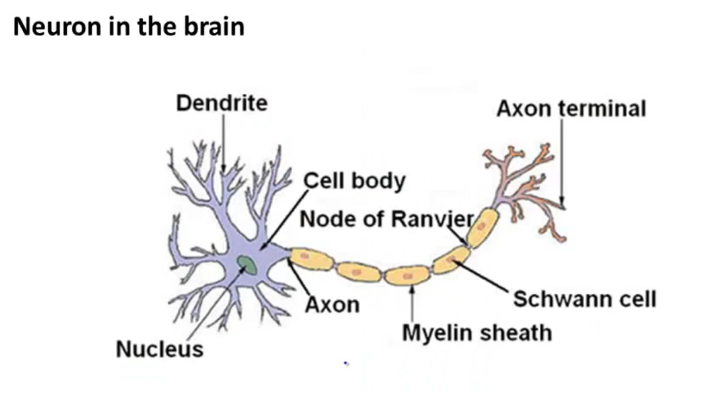
Entonces, es bastante increíble hasta qué medida es posible conectar casi cualquier sensor al cerebro y el algoritmo de aprendizaje del cerebro encontrará la forma de aprender a partir de esos datos y de manejar esos datos.

Y de ahí viene la idea de que, si podemos descubrir cuál es el algoritmo de aprendizaje del cerebro e implementarlo o implementar alguna aproximación a ese algoritmo en una computadora, quizás esa sería nuestra mejor oportunidad para hacer un progreso real hacia la IA, el sueño de la inteligencia artificial de algún día construir máquinas verdaderamente inteligentes.

Ahora, desde luego, no enseño sobre redes neuronales sólo porque puedan darnos una ventana hacia este lejano sueño de la IA, incluso a pesar de que, personalmente, es es una de las cosas en las que trabajo personalmente en mi vida de investigación. Pero la razón principal por la que estoy enseñando sobre redes neuronales en esta clase, es porque son realmente una técnica vanguardista y muy efectiva para las aplicaciones actuales de aprendizaje automático. Entonces, en los siguientes videos, comenzaremos a abordar los detalles técnicos de las redes neuronales, de forma que puedas aplicarlas a aplicaciones actuales de aprendizaje automático y hacerlas funcionar bien en los problemas. Pero para mí, ya sabes, una de las razones que más me emocionan es que, quizás, nos den esta ventana hacia lo que podríamos hacer si también pensamos en qué algoritmos algún día podrían ser capaces de aprender de una forma similar a la del ser humano.

## Representación del modelo

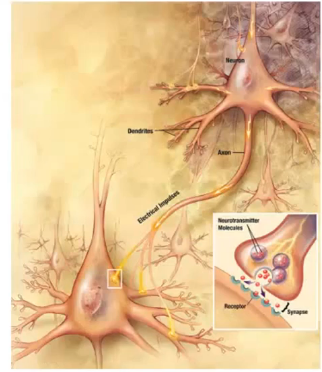
En esta sección quiero comenzar hablando sobre cómo representamos las redes neuronales, en otras palabras, cómo representamos nuestras hipótesis o cómo representamos nuestro modelo cuando utilizamos redes neuronales. Las redes neuronales fueron desarrolladas como simulación de las neuronas o de las redes de neuronas en el cerebro. Así que, para explicar la representación de la hipótesis, vamos a empezar por observar cómo se ve una sola neurona en el cerebro:



Tú cerebro y el mío están saturados de neuronas como estas y las neuronas son células en el cerebro, y las dos cosas que llaman la atención son primero que la neurona tiene un cuerpo celular como este y por otra parte, que la neurona tiene un número de cables de entrada llamados dendritas, que son como cabes de entrada y reciben entradas de otras ubicaciones, y la neurona también tiene un cable de salida llamado axón. Este cable de salida es lo que utiliza para enviar señales o mensajes a otras neuronas.

Así, en un nivel simple, una neurona es una unidad computacional que tiene un número de entradas a través de sus cables de entrada, realiza algunos cálculos, y luego envía resultados, mediante su axón a otros nodos o a otras neuronas en el cerebro.

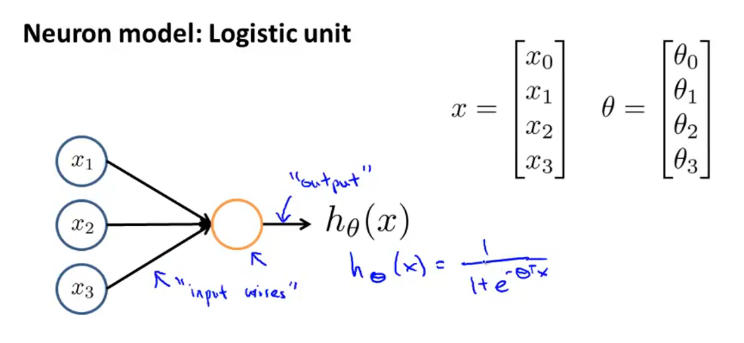
Aquí hay una ilustración de un grupo de neuronas:



La forma en que las neuronas se comunican unas con otras es con pequeñas pulsaciones eléctricas. También se llaman picos, pero se refieren a pequeños pulsos de electricidad.

Así que, vemos una neurona y lo que hace si quiere mandar un mensaje, es mandar un pequeño pulso de electricidad a través de su axón a diferentes neuronas. Tiene ese cable abierto, que se conecta al cable de entrada o se conecta a la dendrita de la segunda neurona, la cual entonces acepta este mensaje entrante, hace algunos cálculos y puede a su vez decidir enviar su mensaje en su axón a otras neuronas. Y este es el proceso por el cual todo pensamiento humano ocurre mientras estas neuronas hacen cálculos y pasan mensajes a otras neuronas como resultado de lo que reciben en otras entradas. Y por cierto, así es cómo nuestros sentidos y nuestros músculos trabajan también. Si deseas mover uno de tus músculos, esto funciona gracias a que la neurona envía estas pulsaciones eléctricas a tus músculos y hace que tus músculos se contraigan o que cuando algunos sensores como tus ojos quieren enviar un mensaje a tu cerebro, lo que hacen es mandar pulsaciones de electricidad a una neurona en tu cerebro.

En una red neuronal, o en una red neuronal artificial que implementemos en una computadora, vamos a utiliza un modelo muy simple de lo que una neurona hace. Vamos a modelar una neurona como una unidad logística. Así que cuando yo dibujo un círculo amarillo, deberías pensar que es como jugar una función análoga, tal vez el cuerpo de una neurona, y cuando se alimenta la neurona con entradas mediante sus dendritas o sus cables de entrada, la neurona hace algunos cálculos y da como resultado algún valor con su cable de salida que en una neurona biológica es esa especie de axón y cada vez que dibujo un diagrama como este, representa un cálculo de la función sigmoidea donde, como siempre, x y «theta» son nuestros vectores de parámetros:



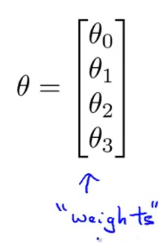
Así que esto es un muy simple, o un enormemente simplificado modelo de los cálculos que una neurona hace, donde obtiene el número de salidas, x1, x2, x3 y da como resultado algunos valores calculados como la función sigmoidea.

Cuando dibujo una red neuronal, generalmente sólo dibujo los nodos de salida x1, x2, x3, a veces cuando es útil hacerlo, dibujo un nodo extra para x0.

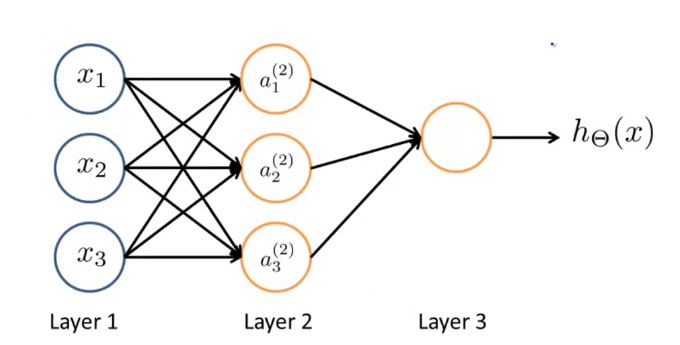
Este nodo x0 es llamado en ocasiones unidad de oscilación (“*bias unit*”) o la neurona de oscilación porque x0 es igual a 1. A veces, dibujo el sesgo, a veces no, dependiendo de si es más conveniente en teoría para ese ejemplo.

Finalmente, un poco de terminología, cuando hablamos acerca de redes neuronales, a veces diremos que esta es una neurona, una neurona artificial con una función de activación sigmoidea o logística. Así que esta función de activación en la terminología de las redes neuronales es tan solo otro término para la función o hipótesis. Continuaré usando esa terminología para conjugar a los parámetros.

En la literatura de las redes neuronales puedes escuchar a la gente hablar de los pesos de un modelo y pesos significa exactamente lo mismo que parámetros del modelo. Utilizaremos el término parámetros en estos videos, pero a veces puedes oír a otros utilizan el término pesos.



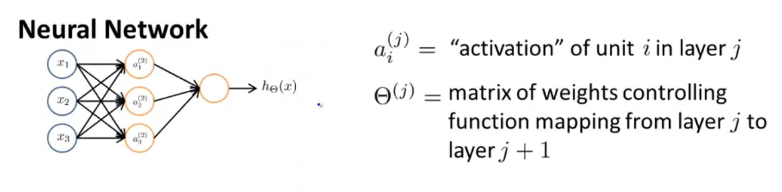
Entonces, este pequeño diagrama representa una sola neurona:

Lo que una red neuronal, simplemente es, un grupo de diferentes neuronas fuertemente unidas.

Específicamente, aquí tenemos unidades de entrada x1, x2 y x3, y una vez más, a veces puedo dibujar el nodo extra x0 y a veces no. Y aquí tenemos tres neuronas, que he escrito como, sabes, a(2)1, a(2)2 y a(2)3 donde una vez más, podemos si lo deseamos, añadir a esto a(2)0y agregar una unidad adicional de oscilación que siempre da como resultado el valor de 1 y finalmente tenemos un tercer nodo en la capa final, y un tercer nodo que arroja el valor final para la hipótesis h(x).

Para introducir un poco más de terminología en una red neuronal, la primera capa *(“first layer”),* también es llamada capa de entrada (“Input layer”) porque aquí es donde introducimos nuestras variables x1, x2, x3. La capa final también es llamada capa de salida (“Outpout layer”) porque esa capa tiene las neuronas que dan como resultado el valor final calculado por una hipótesis y luego la capa dos (“layer two”) en el medio, es llamada la capa oculta (“hidden layer”). El término capa oculta no es un buen término, pero la intuición indica que, como sabes, en el aprendizaje supervisado donde puedes ver las entradas, y podrás ver los resultados correctos. Considerando que la capa oculta tienen valores que no se logran observar cuando entrenamos el modelo. No es X y no es Y, por lo que les llamamos ocultas. Y más adelante veremos redes neuronales con más de una capa oculta, pero en este ejemplo tenemos una capa de entrada, la capa 1; una capa oculta, la capa 2; y una capa de salida, la capa 3. Pero básicamente cualquier cosa que no sea una capa de entrada y que no sea una capa de salida será llamada capa oculta.

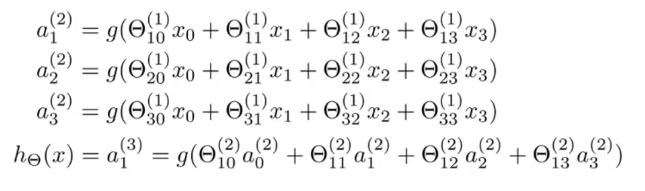
Así es que quiero ser muy claro acerca de lo que esta red neuronal está haciendo. Retrocedamos a través de los pasos computacionales que están incorporados por esto, representados en este diagrama:



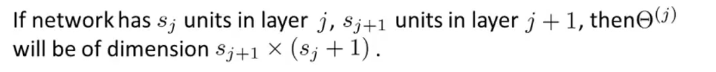
Para explicar los cálculos específicos representados por una red neuronal, Voy a usar "a" superíndice "j" subíndice "i" para denotar la activación de la neurona "i" o de la unidad "i" en la capa "j". Entonces, en concreto, hace referencia a la activación de la primera unidad en la capa 2, en nuestra capa oculta. Y por activación, me refiero, como sabes, al valor que es calculado por la neurona y que es dado como resultado específicamente.

Además, nuestra red neuronal es parametrizada por estas matrices, «theta» superíndice "j" donde nuestra «theta» "j" va a ser una matriz de onda controlando el mapeo de la función de una capa, tal vez de la primera capa a la segunda capa o de la segunda capa a la tercer capa.

Así que, aquí están los cálculos que son representados por este diagrama.



La primera unidad oculta tiene su valor calculado como la función sigmoidal o la función de activación sigmoidal también llamada función de activación logística, aplicada a una combinación lineal de sus entradas. Y del mismo modo, para la segunda y tercera unidad oculta, calculada mediante la fórmula.

Así que aquí tenemos tres unidades de entrada y tres unidades ocultas. Y también la dimensión de «theta»1 la cual es la matriz de parámetros que rigen nuestro mapeo desde las tres unidades de entrada, sobre las tres unidades ocultas «theta»1 va a ser una matriz de 3x4 (contando con x0) y más en general, si una red tiene Sj unidades en su capa j y Sj + 1 unidades en su capa j + 1 entonces la matriz «theta» j que rige el mapeo de la función de la capa j a la capa j+1 va a tener dimensión . 

\*Solo para aclarar esta notación, esto es S subíndice j + 1 y eso es S subíndice j y luego todo esto, + 1. De todo esto, eso es j + 1, ¿de acuerdo? Así que eso es S subíndice j + 1, por tanto, eso es S subíndice j + 1 multiplicado por Sj + 1 donde + 1 no es parte del subíndice. 

El +1 proviene de la adición en  de los "nodos de sesgo" x0 y  . En otras palabras, los nodos de salida no incluirán los nodos de polarización mientras que las entradas lo harán

Ejemplo: si la capa 1 tiene 2 nodos de entrada y la capa 2 tiene 4 nodos de activación. La dimensión de  va a ser donde 

Así es que, hemos hablado de lo que las tres unidades ocultas hacen para calcular sus valores.

Finalmente, este último, en la capa de salida, tenemos una unidad más que calcula h de x, y puede escribirse también como y es igual a:



Y si te das cuenta tengo escrito theta con un superíndice 2 aquí porque «theta» superíndice 2 es la matriz de parámetros, o la matriz de pesos que controla la función que mapea las unidades ocultas, que son las unidades de la capa 2, a la unidad de la capa 3 que es la unidad de salida.

Para resumir, lo que hemos hecho es mostrar como una imagen como esta de aquí define una red neuronal artificial que define una función h que mapea tus valores de entrada x que con suerte en algunos espacios y disposiciones y. Y estas hipótesis después parametrizadas por parámetros que estoy denotando con «theta» mayúscula variará para obtener diferentes hipótesis. Así es que llegamos a diferentes funciones de mapeo digamos de x a y.

Entonces, esto nos da una definición matemática de cómo representar la hipótesis en la red neuronal. En algunos de los siguientes videos, lo que me gustaría hacer es darte un mejor entendimiento sobre lo que estas representaciones de hipótesis hacen, y también hablar de algunos ejemplos y de cómo calcularlos eficientemente.

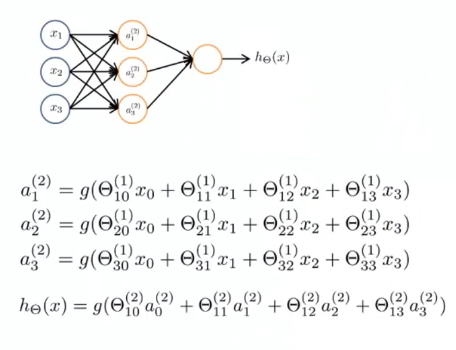
## Representación del modelo segunda parte

En la anterior sección vimos una definición matemática sobre cómo representar o cómo calcular las hipótesis utilizadas por las redes neuronales.

En esta sección, me gustaría mostrarte cómo realizar ese cálculo de forma eficiente, y eso es, mostrarte la implementación vectorial.

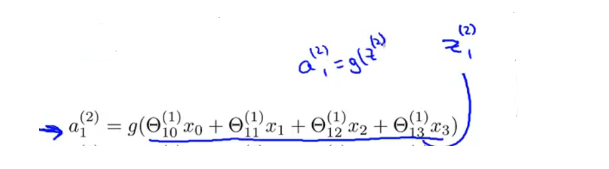
En segundo lugar, y más importante, quiero empezar a darte una intuición sobre por qué estas representaciones en redes neuronales pueden ser una buena idea y cómo pueden ayudarnos a aprender hipótesis complejas no lineales.

Considera esta red neuronal gráficamente y matemáticamente:



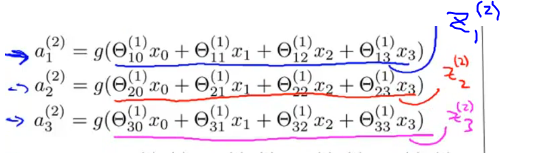
Anteriormente, dije que la secuencia de pasos que necesitamos para calcular la salida de una hipótesis en estas ecuaciones, en las que calculamos los valores de activación de las tres unidades ocultas (a) y después los utilizamos para calcular la salida final de nuestras hipótesis (h(x)).

Ahora, voy a definir algunos términos adicionales. Entonces, este término que estoy subrayando aquí, voy a definirlo como z superíndice 2 subíndice 1. De forma que tengamos que a(2)1, que es este término, sea igual a g de z a 1.



Por cierto, estos superíndices 2 paréntesis, significa que éstos son valores asociados con la capa 2, esto es, con la capa oculta en la red neuronal.

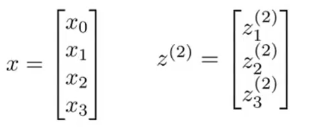
Haciendo lo mismo con los siguientes términos:



Con z solo nos hemos buscado una forma de resumir esa combinación lineal ponderada de los valores de entrada que entran en una neurona particular.

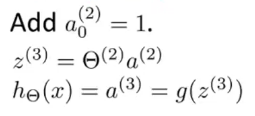
Ahora, si observamos ese bloque de números dentro de las funciones g(), puedes notar que ese bloque de números corresponde de forma sospechosamente similar a la operación de vector matriz, la multiplicación de vector matriz de x1 veces el vector x: . Mediante esta observación, Vamos a ser capaces de vectorizar este cálculo de la red neuronal.

Específicamente, vamos a definir el vector x de la variable de la forma usual, como el vector de x0, x1, x2, x3 donde x0, como siempre, es siempre igual a 1 y eso define z2 como el vector de estos valores z, z(2)1 z(2)2, z(2)3:



Y nota que, ahí, z2 es un vector de tres dimensiones. Ahora podemos vectorizar el cálculo de a(2)1, a(2)2, a(2)3 de la siguiente manera. Podemos escribir esto en sólo dos pasos. Podemos calcular; y  y, sólo para ser claro, z2 es un vector de tres dimensiones y a2 también es un vector de tres. Esto aplica la función sigmoidea en torno al elemento a cada uno de los elementos de z2. Y por cierto, para hacer nuestra notación un poco más coherente con lo que haremos más adelante, en esta capa de entrada tenemos las entradas x, pero también podemos pensar en éstas como en activaciones de las primeras capas. Entonces, si se define que a1 es igual a , Entonces, a1 es un vector, y puedo reescribir 

Ahora, con lo que he escrito hasta ahora he obtenido yo mismo los valores de , , . Pero necesito un valor más, lo que quiere decir que también quiero y que corresponda con una unidad de oscilación en la capa oculta que va a la capa de salida. Por supuesto, también habría que añadir una unidad de oscilación en la capa de entrada, que como ya sabes, no la coloqué en este ejemplo, pero, para resolver esta unidad de oscilación adicional, lo que vamos a hacer es sumar un a0 superíndice 2 adicional, que es igual a uno:



Y después de este paso, ahora tenemos que a (2) va a ser un vector de variables de cuatro dimensiones porque acabamos de sumar este adicional que es igual a 1 correspondiente a la unidad de oscilación en la capa oculta:

Y, finalmente, para calcular el valor real de salida de nuestras hipótesis, simplemente tenemos que calcular z (3). entonces, z (3) es igual a este término que estoy subrayando:



Este término interior es z3. Y z3 se indica  es decir, la función de mapeo 2 que tiene los pesos de la capa oculta a la capa de salida, multiplicado por el valor arrojado por las neuronas.

Y, por último, la salida de mi hipótesis h(x), que es a3, es la activación de mi única unidad en la capa de salida:



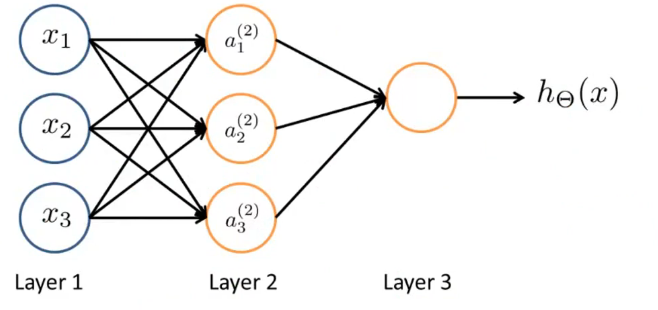
Entonces, este sólo es el número real. Lo puedes escribir como a3 o como, y eso es g(z3). A este proceso de calcular h(x) también se llama propagación hacia adelante, y se llama así porque comenzamos con las activaciones de las unidades de entrada y luego es como si hubiera una propagación hacia adelante de esto hacia la capa oculta y se calcularan las activaciones de la capa oculta y luego se propagara de nuevo hacia delante y se calcularan las activaciones de la capa de salida, pero este proceso de calcular las activaciones desde la entrada, entonces la capa oculta y luego la capa de salida, y eso también se llama propagación hacia adelante.

Y lo que acabamos de hacer fue trabajar en la implementación de este procedimiento en el sentido de vectores. Entonces, si lo implementas usando estas nuevas ecuaciones que tenemos:

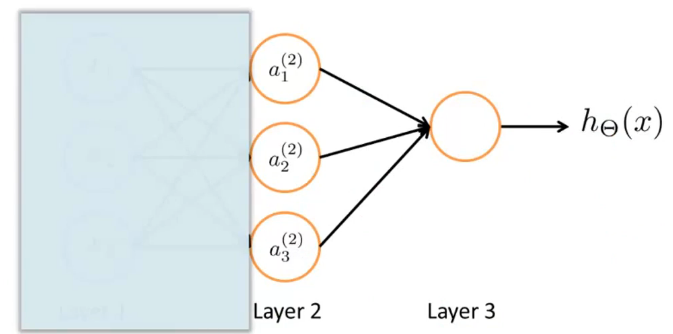
te darán una forma eficiente o ambas formas eficientes para calcular .

Esta vista de la propagación hacia adelante también nos ayuda a comprender lo que las redes neuronales podrían estar haciendo y por qué podrían ayudarnos a aprender interesantes hipótesis no lineales.

Considera la siguiente red neuronal:



Y supongamos que cubro la parte izquierda de esta imagen por ahora:

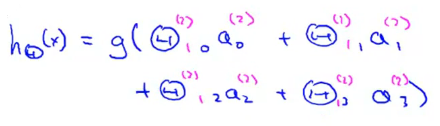


Si miras lo que queda en esta imagen, se ve muy parecido a una regresión logística en donde lo que estamos haciendo es usar a(3), esa es justo la unidad de la regresión logística y la estamos usando para hacer una predicción h(x). Y, en concreto, lo que las hipótesis están mostrando es que h(x) va a ser igual a g, lo que es mi función sigmoidea de activación multiplicada por «theta» 0 por a0, que es igual a 1 más «theta» 1 más «theta» 2 por a2 más «theta» 2 por a3:



Siendo que los valores a1, a2 y a3 son aquellos dados por las tres unidades , , .

Ahora, para ser realmente consistente a mi notación previa, de hecho, necesitamos, ya sabes, completar estos superíndices 2 en todas partes, y también tengo estos subíndices 1 en la función de mapeo porque sólo tengo una unidad de salida:

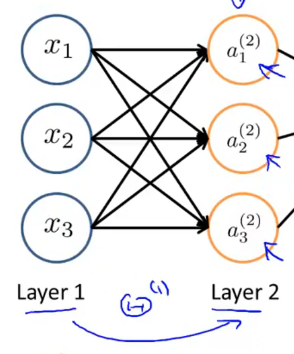


Pero si te enfocas en las partes azules de la notación. Esto es como el modelo de la regresión logística estándar, excepto que ahora tengo una «theta» mayúscula en lugar de una «theta» minúscula.

Y lo que esto está haciendo sólo es la regresión logística, pero donde las variables que se introducen en la regresión logística son los valores calculados por la capa oculta *(“layer 2”*).

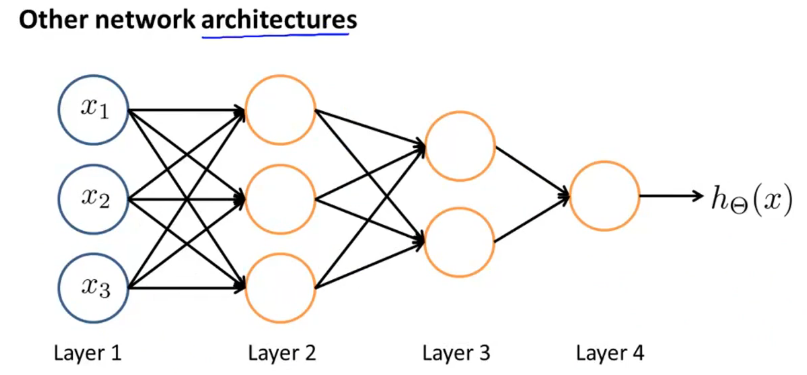
Sólo para repetirlo, lo que esta red neuronal está haciendo es como una regresión logística, excepto que en lugar de utilizar las variables originales x1, x2, x3, está utilizando nuevas variables , , .

Y lo bueno de esto, es que las variables , , , fueron aprendidas como funciones de la entrada. Concretamente, el mapeo de la función desde la capa 1 a la capa 2, que está determinada por algún otro conjunto de parámetros, «theta» 1:



Entonces, es como si la red neuronal, en lugar de estar obligada a alimentar a la regresión logística con las variables x1, x2 y x3 llega a aprender sus propias variables, , y , para introducirlas en la regresión logística y, como puedes imaginarte, dependiendo de los parámetros que elija para tu red neuronal puede aprender algunas variables bastante interesantes y complejas y, por lo tanto, puedes terminar con una mejores hipótesis que si estuvieras obligado a usar las variables en bruto x1, x2 o x3 , o si te vas a limitar a, digamos, elegir los términos polinomiales, ya sabes, x1x2, x1x3 … y así sucesivamente. Pero, en cambio, este algoritmo tiene la flexibilidad para tratar de aprender cualquier número de variables a la vez, usando estos , y para introducirlos en la última unidad.

Me di cuenta de que este ejemplo se describe como con un nivel un poco elevado, por lo que no estoy seguro si esta intuición de la red neuronal, ya sabes, con variables más complejas ya tendrá sentido, pero si todavía no es así, en los siguientes dos videos, voy a explicar un ejemplo específico de cómo una red neuronal puede usar esta capa oculta para calcular variables más complejas para introducirlas en esta capa final de salida, y cómo eso puede aprender hipótesis más complejas. Entonces, si lo que estoy diciendo aquí no tiene mucho sentido, acompáñame en los siguientes dos videos y, con suerte, trabajando con esos ejemplos, esta explicación tendrá un poco más de sentido. Pero sólo el punto O. Tú puedes tener redes neuronales con otros tipos de diagramas también, y la forma en la que las redes neuronales están conectadas, a eso se le llama la arquitectura. Así que el término "arquitectura" se refiere a la forma en la que las diferentes neuronas están conectadas entre sí. Este es un ejemplo de una arquitectura diferente de una red neuronal



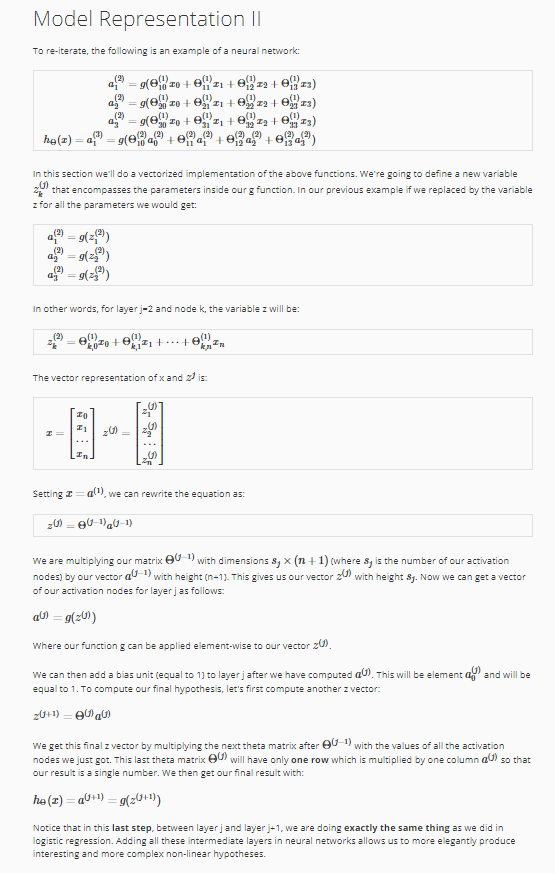
Y, nuevamente, en la segunda capa, aquí tenemos tres unidades de dirección que están calculando alguna función compleja, quizás de la capa de entrada, y luego la tercera capa que puede tomar las variables de la segunda capa y calcular variables todavía más complejas en la capa tres, de forma que, para el momento en el que llegues a la capa de salida, la capa cuatro, puedes tener variables aún más complejas de lo que podrías calcular en la capa tres y así obtener hipótesis no lineales muy interesantes.

Por cierto, en una red como esta, a la capa uno se le llama capa de entrada. La capa cuatro sigue siendo nuestra capa de salida, y esta red tiene dos capas ocultas. Así que, cualquier cosa que no sea una capa de entrada o una capa de salida, es llamado capa oculta.

Entonces, espero que de este video hayan obtenido un sentido de cómo el paso de propagación hacia adelante en una red neuronal trabaja donde comienzas desde las activaciones de la capa de entrada y propagando eso hacia adelante hacia la primera capa oculta, después la segunda capa oculta y luego, finalmente, la capa de salida. Y también vimos cómo podemos vectorizar ese cálculo.

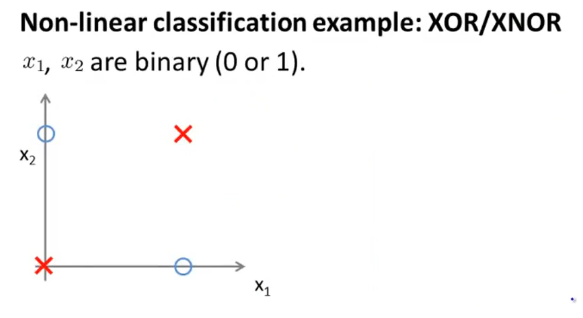
Me di cuenta de que algunas de las intuiciones en este video sobre cómo, ya sabes, otras capas están calculando variables complejas de las capas anteriores. Me di cuenta de que esa intuición podría seguir siendo un poco abstracta y de un nivel un tanto elevado. Entonces, lo que me gustaría hacer en los dos videos es trabajar con un ejemplo detallado de cómo una red neuronal puede utilizarse para calcular funciones no lineales de la entrada, y espero que eso te dé una buena idea del tipo de hipótesis complejas no lineales complejas que podemos obtener de las redes neuronales.

### Resumen de la sección:

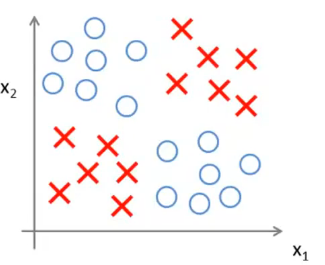


## Ejemplos e intuiciones I

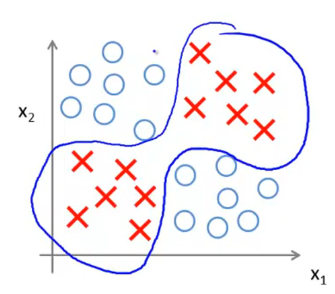
Considere el siguiente problema en el que tenemos variables de entrada x1 y x2 que son valores binarios, así que son cero o uno. Entonces x1 y x2 pueden tomar sólo uno de dos valores posibles.



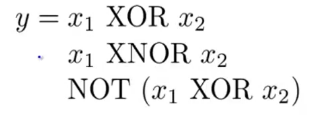
En este ejemplo dibujé sólo dos ejemplos positivos y dos ejemplos negativos, pero puedes considerar esto como una versión simplificada de un problema de aprendizaje más complejo en el que puede haber varios ejemplos positivos en la parte superior derecha y en la parte inferior izquierda y varios ejemplos negativos denotados por los círculos:



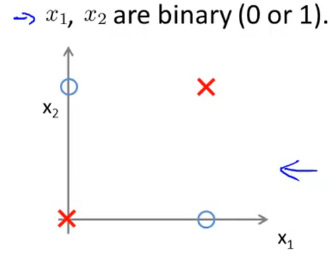
Lo que nos gustaría hacer es aprender un límite de decisión no lineal en el que debemos separar los ejemplos positivos y negativos:



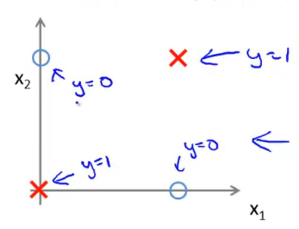
Entonces, ¿cómo puede una red neuronal hacer esto? Y, en lugar de usar el ejemplo con muchas observaciones, voy a usar el más sencillo. Específicamente, lo que esto está calculando realmente es la etiqueta "y" es igual a x1 XOR x2.



* La compuerta lógica XOR realiza una comparación de las entradas siendo el resultado 0 si las entradas son iguales o 1 cuando son diferentes.
* "XNOR", es llamada compuerta lógica de EQUIVALENCIA, porque su salida es "1" cuando las entradas se encuentran en el mismo estado.

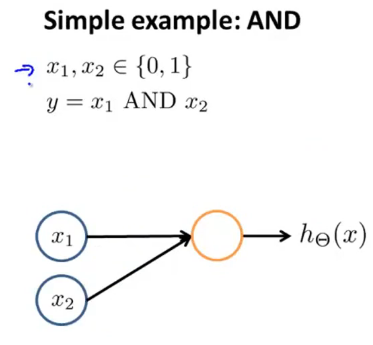


Entonces x1, XOR x2 - sólo es verdadero si exactamente una de las variables es igual a 1. Sucede que el ejemplo específico que voy a utilizar funciona un poco mejor si usamos el ejemplo de XNOR. Ambos son lo mismo, desde luego. Esto significa NOT x1 XOR x2, así que vamos a tener ejemplos positivos si ambos son verdaderos o si ambos son falsos ; y vamos a tener que "y"es igual a 1, "y" es igual a 1 y vamos a tener que "y" es igual a 0 si sólo uno de ellos es verdadero:



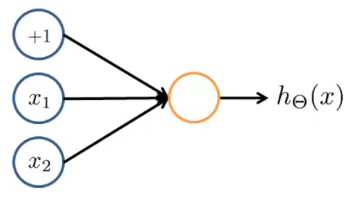
Y queremos saber si podemos hacer que una red neuronal ajuste este tipo de conjuntos de aprendizaje.

Con el fin de construir una red que ajuste el ejemplo XNOR, vamos a comenzar con uno ligeramente más simple y mostrar una red que se ajuste a la función AND. Concretamente, digamos que tenemos las entradas x1 y x2 que, de nuevo, son binarias. Entonces, son cero o uno. Y digamos que nuestras etiquetas destino "y" son iguales a x1 y x2. Éste es un AND lógico:

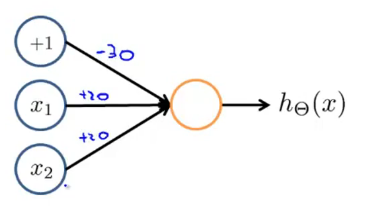


Entonces, ¿podemos hacer que una red de una unidad calcule esta función lógica AND?

Para hacer esto, voy, de hecho, a dibujar las unidades de oscilación también:



Ahora, vamos a asignar algunos valores a los pesos o los parámetros de esta red. Voy a escribir los parámetros en este diagrama:



Y esto significa que estoy asignando un valor de menos de treinta al valor asociado a x0. Y un valor de +20 al parámetro que multiplica x1, y un valor de +20 para el parámetro que multiplica en x2.

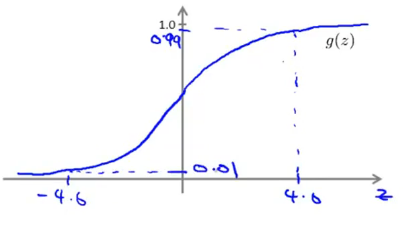
Así, concretamente, esto está diciendo que mi hipótesis “h” de x es igual a



Y matricialmente:

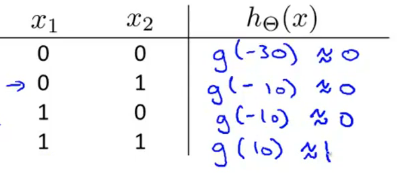


Veamos lo que calculará esta red con una sola neurona. Sólo para recordarte, la función de activación sigmoidea g de z se ve así. Comienza en 0, se eleva lentamente, pasa por 0.5 y tiene su asíntota en uno. Y para darte un punto de referencia, si el valor en el eje horizontal es igual a 4.6, entonces la función sigmoidea es igual a 0.99. Esto está muy cerca de 1 y como es casi simétrica, cuando vale -4.6, entonces la función sigmoidea es igual a 0.01, que está muy cerca de 0:



Veamos los cuatro posibles valores de entrada para x1 y x2 y veamos los resultados para la hipótesis en cada caso.

1. Si ambas, x1 y x2 son iguales a 0 - si ves esto, si x1 y x2 son iguales a 0 entonces las hipótesis del punto g de -30. Entonces, está muy hacia la izquierda de este diagrama. Esto estará muy cerca de 0.
2. Si x1 es igual a 0 y x2 es igual a 1, entonces esta fórmula aquí evalúa a g, por lo tanto la función sigmoidea aplicó a -10 y, otra vez, está en el extremo izquierdo de este diagrama y, por lo tanto, está nuevamente muy cerca de 0.
3. Si intercambiamos los valores otra vez tenemos g de -10.
4. Y, finalmente, si x1 es igual a 1, x2 es igual a 1, entonces tienes g de -30 +20 +20, y eso es g de +10, que, por lo tanto, está muy cerca de 1.

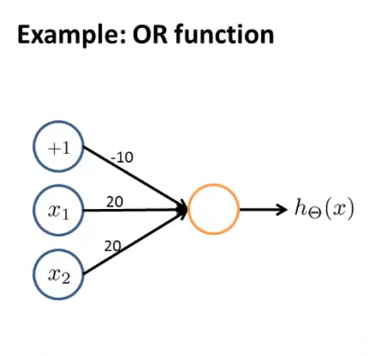


Y si se observa esta columna, esta es exactamente la función lógica AND. Entonces, esto es calcular h de x, que es aproximadamente x1 y x2.

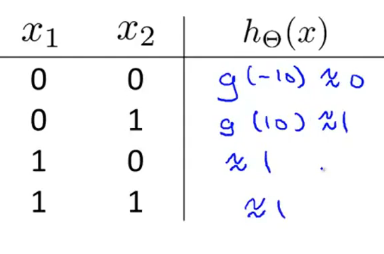


En otras palabras, muestra 1 si y sólo si x1 y x2 son iguales a 1. Entonces, haciendo nuestra pequeña tabla de verdad de esta manera, podemos averiguar cuál es la función lógica que nuestra red neuronal calcula.

Vamos con otro ejemplo:



La red que se muestra calcula la función OR, sólo para mostrarte cómo lo resolví, si escribes la hipótesis, encontrarás que está calculando . Y si completas estos valores encontrarás que g de -10, que es aproximadamente 0, g de 10 lo que es aproximadamente 1, y así sucesivamente:



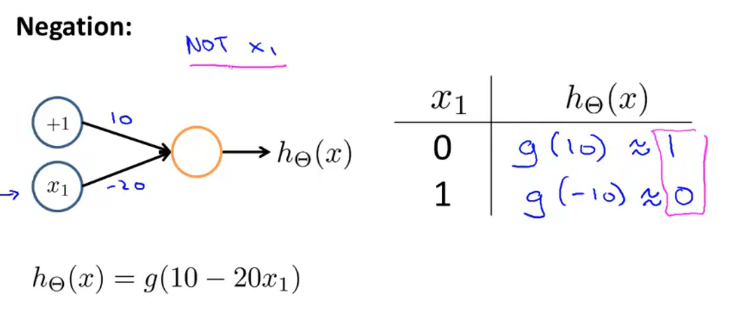
Estos son aproximadamente 1, y aproximadamente 1, y estos números son esencialmente, la función lógica OR.

Entonces, espero que con esto ahora comprendas cómo neuronas individuales en una red neuronal pueden utilizarse para calcular funciones lógicas como AND y OR y así sucesivamente. En el siguiente video continuaremos avanzando con estos ejemplos y trabajaremos con un ejemplo más complejo. Vamos a llegar a mostrarte cómo una red neuronal, ahora con múltiples capas de unidades, puede usarse para calcular funciones más complejas como la función XOR o la función XNOR.

## Ejemplos e intuiciones 2

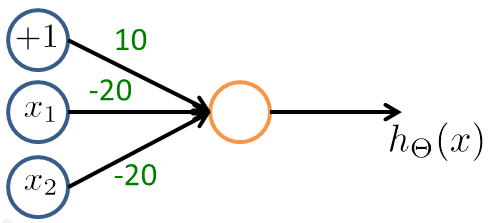
En este video quiero trabajar en nuestro ejemplo para mostrar cómo una red neuronal puede calcular hipótesis complejas no lineales.

En el último video vimos cómo una red neuronal puede usarse para calcular las funciones x1 y x2 y la función x1 o x2 cuando x1 y x2 son binarias. Es decir, cuando toman los valores 0 y 1. También podemos hacer que una red calcule la negación, lo que significa calcular la función "not x1". Voy a escribir las formas asociadas con esta red. Sólo tenemos una variable de entrada, x1 En este caso, y la unidad de oscilación más 1 y, si asocio esto con los pesos más 10 y menos 20, entonces mi hipótesis está calculando esto. H de x es igual a sigmoide de 10 menos 20 veces x1. Entonces, cuando x1 es igual a 0, mi hipótesis estará calculando g de 10 menos 20 veces 0 que es exactamente 10. Y eso es aproximadamente 1, y cuando x es igual a 1, esto será g de menos 10, que es aproximadamente igual a 0. Y, si observas lo que son estos valores, esta es esencialmente la función "not x1".

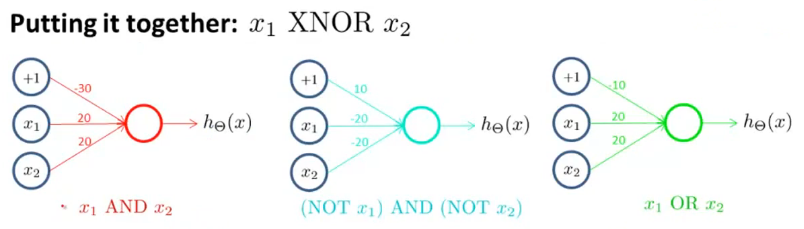


Así que, para incluir negaciones, la idea general es poner un gran peso negativo delante de la variable que desea anular. Así que, si es -20, multiplicado por x1 y esa es la idea general de cómo se llega a la negación de x1.

Otro ejemplo (ni x1 ni x2):

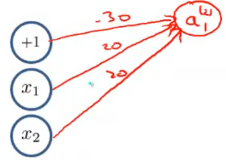


Ahora, tomando las tres piezas que hemos colocado juntas, la red para calcular x1 y x2 y la red para calcular not x1 y not x2 y una última red para calcular x1 or x2, deberíamos ser capaces de reunir estas tres piezas para calcular esta función x1, XNOR x2.

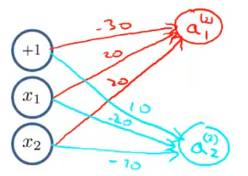


Y, como recordatorio, si fuera x1, x2, la función que queremos calcular tendría ejemplos negativos en , y tendríamos ejemplos positivos ,. Por lo que, claramente, necesitaríamos un límite de decisión no lineal para separar los ejemplos positivos y negativos.

Dibujemos la red. Voy a tomar mi entrada +1, x1, x2, y crear mi primera unidad oculta aquí. Voy a llamarla a(2)1 porque es mi primera unidad oculta. Y voy a copiar los pesos de la red roja; las redes x1 y x2. Entonces, ahora menos 30, 20, 20:



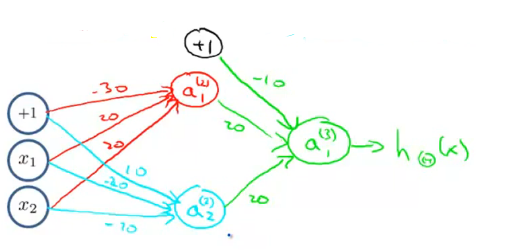
A continuación, voy a crear una segunda unidad oculta, a la que voy a llamar a(2)2, y que es la segunda unidad oculta de la capa dos. Y voy a copiarla desde la red azul en el medio, así que voy a tener los pesos 10, menos 20, menos 20.



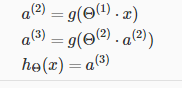
Entonces, tomemos algunos valores de la tabla de verdad. para la red roja, sabemos que estaba calculando x1 y x2. Y entonces esto es aproximadamente 0, 0, 0, 1, dependiendo de los valores de x1 y x2. Y para a(2)2, que es la red azul, bien, conocemos la función not x1 y not x2, entonces la salida es 1, 0, 0, 0 para los 4 los valores de x1 y x2.



Finalmente, voy a crear mi nota de salida, mi unidad de salida es a(3)1. Esta es una salida h de x más y voy a copiarla a la red OR y, para eso, voy a necesitar una unidad de oscilación más uno aquí. Entonces, colocamos eso y voy a copiar desde los pesos de las redes verdes. Entonces, es menos 10, 20, 20 y sabemos desde antes que esto calcula la función OR.

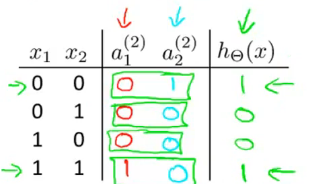






Entonces, pasemos a las entradas de la tabla de verdad.

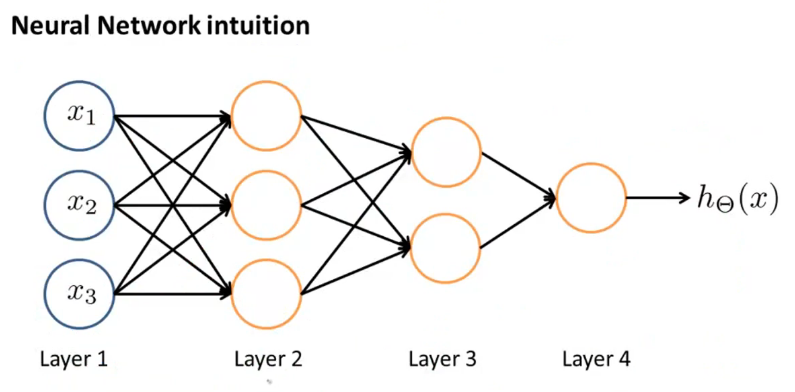
Para la primera entrada es 0 o 1, que va a ser 1, después el siguiente 0, o 0, que es 0, 0, o 0, que es 0, 1 o 0, y todo esto es a 1 y, por lo tanto, h de x es igual a 1 cuando x1 y x2 son 0 o cuando tanto x1 como x2 son 1.



Y concretamente, h de x muestra 1 exactamente en estas dos ubicaciones y se muestra 0 de lo contrario y, por lo tanto, con esta red neuronal, que tiene una capa de entrada, una capa oculta y una capa de salida, terminamos con un límite de decisión no lineal que calcula esta función XNOR.

Y la intuición más general es que en la capa de entrada, sólo tuvimos nuestras entradas en bruto y después tuvimos una capa oculta, que calculó algunas funciones un poco más complejas de las entradas que se muestran aquí; estas son funciones un poco más complejas, y después, añadiendo aún otra capa, terminamos con una función no lineal aún más compleja.

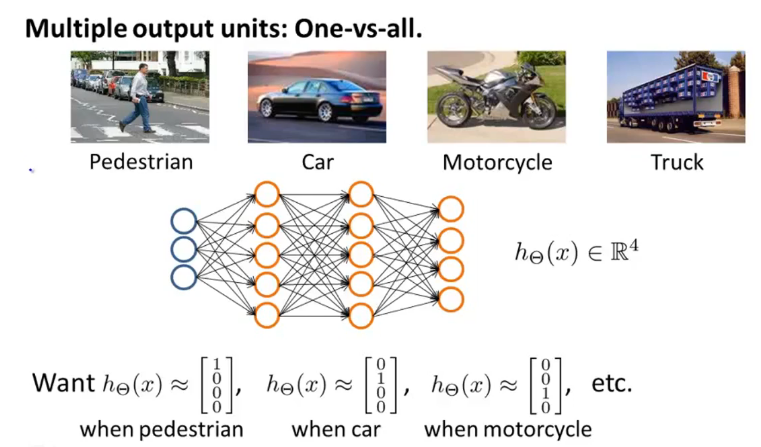
Y este es el tipo de intuición acerca de por qué las redes neuronales pueden calcular funciones bastante complicadas que, cuando se tienen varias capas, se tiene, ya sabes, una función relativamente simple de las entradas y la segunda capa, Pero la tercera capa puede trabajar a partir de eso para calcular funciones aún más complejas y, después, La capa después de esta puede calcular funciones todavía más complejas.



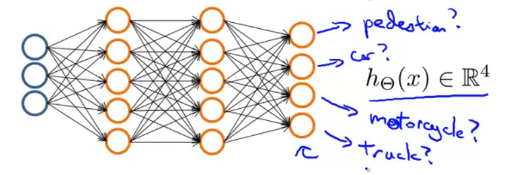
## Clasificación multiclase

En este video quiero hablarte sobre cómo utilizar la redes neuronales para hacer una clasificación multiclase en donde podemos tener más de una categoría y queremos que se distingan entre sí. En la última parte del último video, en donde teníamos el problema del reconocimiento de dígitos escritos a mano, que, de hecho, era un problema de clasificación multiclase porque habían diez categorías posibles para reconocer los dígitos desde Del 0 al 9 y si quieres que te llenemos con los detalles sobre cómo hacerlo. La forma en la que hacemos la clasificación multiclase en una red neuronal es esencialmente una extensión del método de todos contra uno.

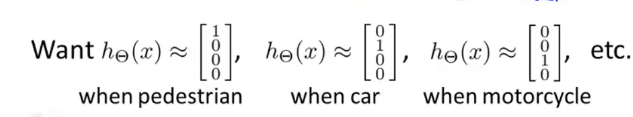
Entonces, digamos que tenemos un ejemplo de visión por computadora, donde en vez de sólo tratar de reconocer autos como en el ejemplo original con el que empecé, digamos que estamos tratando de reconocer cuatro categorías de objetos y, tomando una imagen, queremos decidir si se trata de un peatón, un auto, una moto o un camión. Si ese es el caso, lo que podemos hacer es construir una red neuronal con cuatro unidades de salida, de forma que nuestra red neuronal ahora muestre un vector de cuatro números.



De este modo, ahora la salida necesita ser un vector de cuatro números, y lo que vamos a intentar es obtener la primera unidad de salida para clasificar: es la imagen de un peatón, sí o no. La segunda unidad para clasificar: es la imagen de un auto, sí o no. Esta unidad para clasificar: es la imagen de una motocicleta, sí o no, y esto clasificaría: es la imagen de un camión, sí o no.

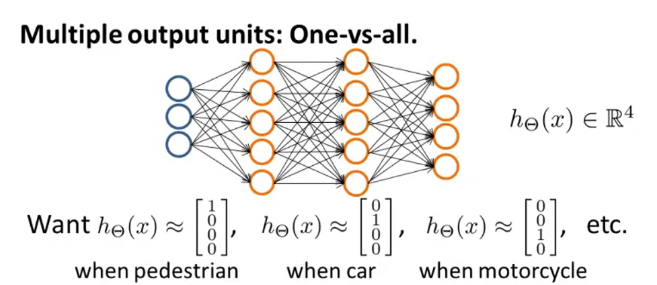


Y así, cuando la imagen es de un peatón, idealmente queremos que la red muestre 1, 0, 0, 0, cuando se trata de un auto, queremos que muestre 0, 1, 0, 0, cuando es una motocicleta, queremos que muestre 0, 0, 1, 0 y así sucesivamente.

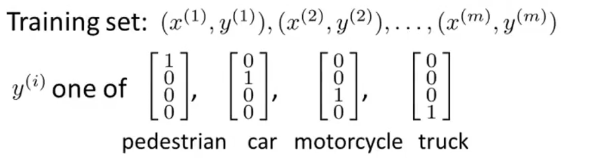


Entonces, éste es justo como el método de “todos contra uno” del que hablamos cuando describimos la regresión logística, y aquí tenemos esencialmente cuatro clasificadores de regresión logística, cada uno de los cuales está tratando de reconocer una de las cuatro clases que queremos distinguir entre ellas.

Así que, si reorganizamos la diapositiva un poco, aquí está nuestra red neuronal con cuatro unidades de salida en las que queremos que sean h(x) cuando tenemos diferentes imágenes:



Y la forma en la que vamos a representar el conjunto de entrenamiento en estos ajustes es como sigue:



Entonces, cuando tenemos un conjunto de entrenamiento con distintas imágenes de peatones, autos, motocicletas y camiones, lo que vamos a hacer en este ejemplo es que, mientras que anteriormente habíamos escrito las etiquetas como "y" siendo un entero de 1, 2, 3 o 4, en lugar de representar "y" de esta forma, vamos a representar "y" de la siguiente manera: específicacment, Yi será 1, 0, 0, 0 o 0, 1, 0, 0 o 0, 0, 1, 0 o 0, 0, 0, 1 dependiendo de cuál sea la imagen Xi correspondiente. Así, un ejemplo de entrenamiento sería un par Xi : Yi En donde Xi es una imagen con uno de los cuatro objetos y Yi será uno de estos vectores.

Con suerte, podemos encontrar una forma para lograr que nuestras redes neuronales arrojen algún valor. Entonces, la h de x es aproximadamente "y" y h de x y Yi ambas van a ser, en nuestro ejemplo, vectores de cuatro dimensiones cuando tenemos cuatro clases.

## Resumen gráfico red neuronal

