Redes Neuronales

Fernanda Sobrino

6/20/2021

Introducción

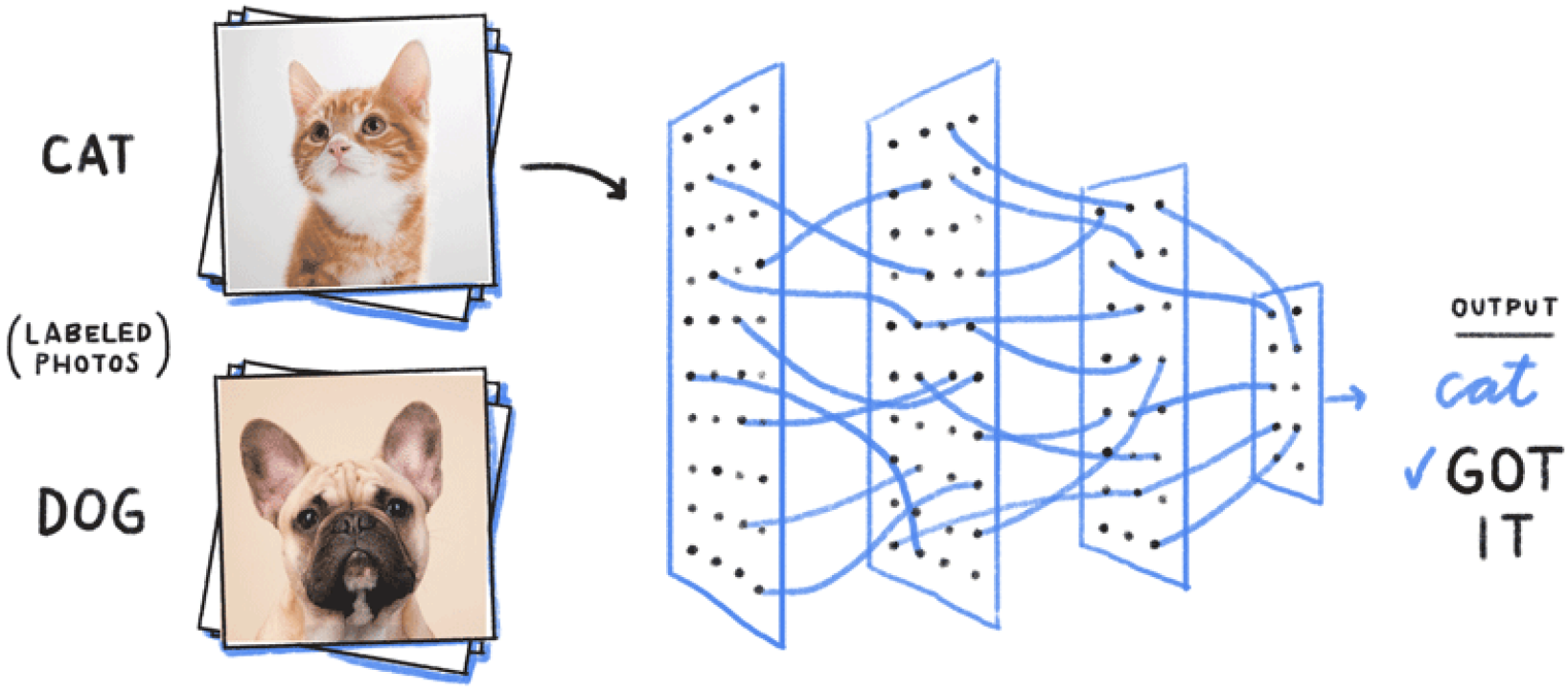
Ejemplos de cosas que pueden hacer

Ejemplos de cosas que pueden hacer

Por qué son una buena idea?

- capaces de describir relaciones no lineales y complejas
- son generalizables, predicen en datos que no han visto
- no impone ninguna restricción en las variables de entrada, capaces de modelar datos altamente volátiles y con varianzas variables
- diseñadas para resolver problemas fáciles para un humano pero complejos para una máquina

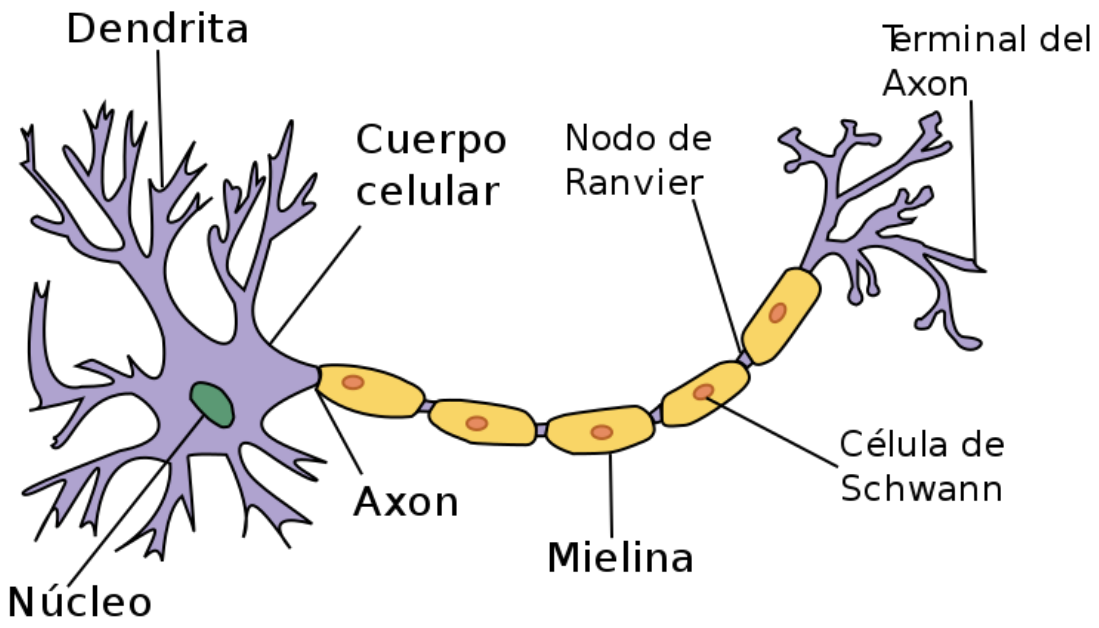
Por qué son una buena idea?



De dónde salieron?

- 1943 McCulloch y Pitts describen el primer modelo matemático de una neurona
- implementadas por primera vez en los 90s
- inspiradas en el funcionamiento biológico de las neuronas

Neuronas biológicas

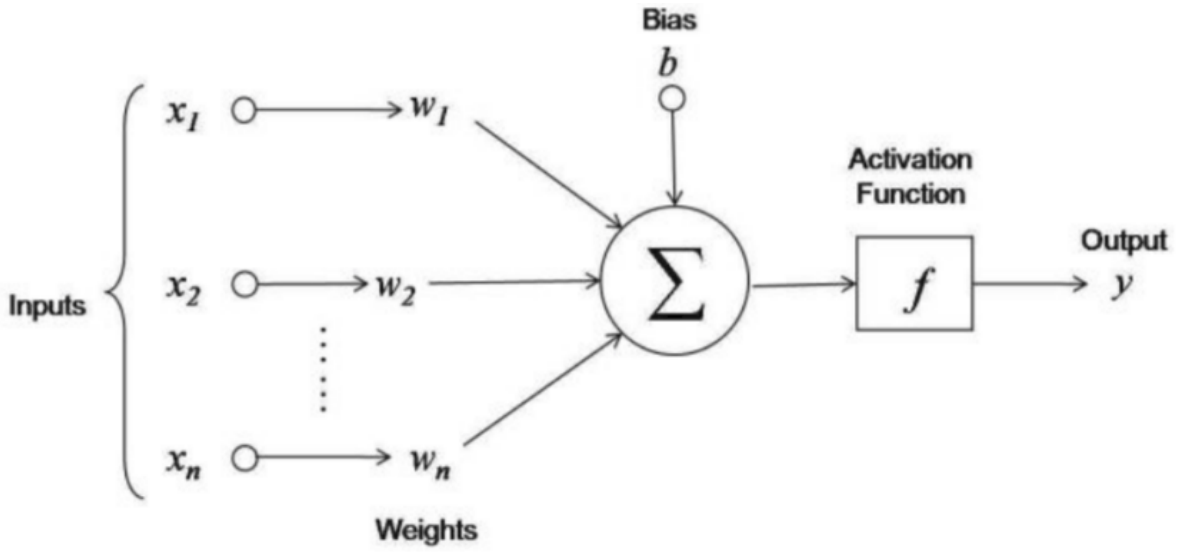


Neuronas biológicas

- las dendritas reciben señales de entrada a través de procesos bioquímicos
- el impulso es ponderado de acuerdo a su importancia o frecuencia
- el cuerpo de la célula acumula las señales de entrada y si se alcanza cierto umbral, se transmite una señal de salida
- dicha señal es transmitida por el axón
- la sinapsis es el punto de interacción entre neuronas

Neuronas Artificiales

Neuronas artificiales



Qué son?

- conjunto de unidades de input/output conectadas
- cada conexión tiene un peso asociada a ella
- durante el aprendizaje se ajustan estos pesos
- las conexiones emulan la sinapsis de las neuronas biológicas

Componentes de las redes neuronales:

- función de activación: transforma la información de entrada en una señal transmitida al resto de la red
- topología de la red: número de neuronas/nodos, capas y como están conectadas
- entrenamiento: como se fijan los pesos de conexión entre neuronas

Funciones de Activación

Definición

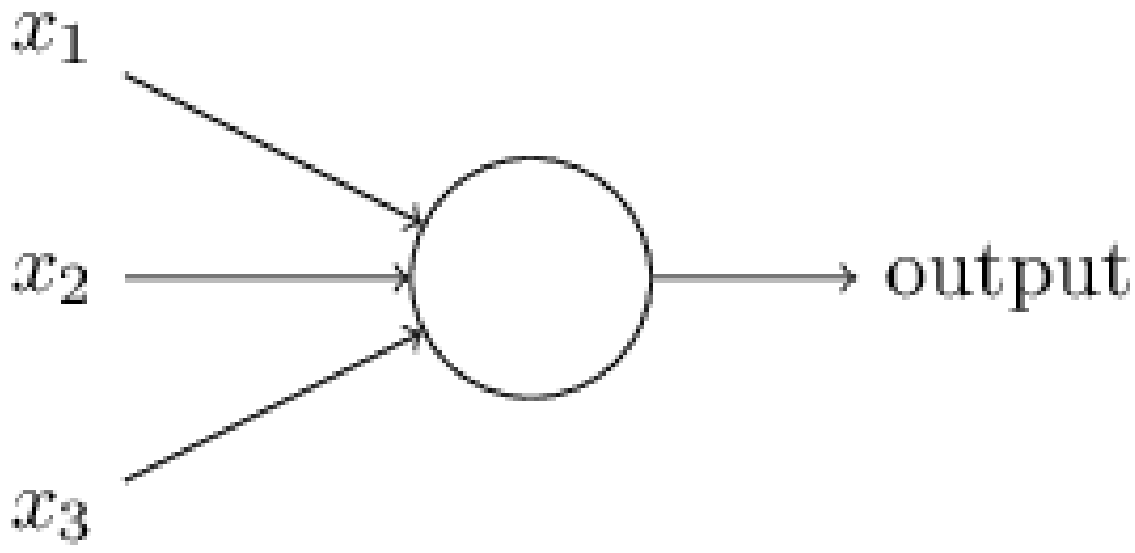
- asociadas a un nodo
- define la salida de un nodo dada una entrada o conjuntos de entradas
- $\phi(v) = f(v)$
- donde f puede ser casi cualquier función (por lo general querremos que sea continuamente diferenciable y que el rango sea finito)

Tipos de funciones de activación

- perceptron
- sigmoidal logística
- linear
- linear saturada
- hiperbólica
- gaussiana

Perceptrons

- clasificación binario lineal
- no tan usado pero útil para entender como funcionan las redes neuronales
- inputs $x_1, x_2, ..., x_n$ outputs $\{0, 1\}$



Perceptrons: Ejemplo

- inputs x_1, x_2, x_3
- cada input tiene un peso asociado w_1, w_2, w_3
- pesos que tan importante es cada uno de los inputs para el ouput
- el output está dado por:
  *

$$output = \begin{cases} 0 & \sum_j w_j x_j \leq threshold \\ 1 & \sum_j w_j x_j > threshold \end{cases}$$

- intuitivamente: el perceptron toma las decisiones dandole diferentes pesos a la evidencia
- un único perceptron es la red neuronal hacia adelante más sencilla

Perceptron forma vectorial

- El input es el vector x de dimensiones $nx1$

$$output = \begin{cases} 0 & w \cdot x + b \leq 0 \\ 1 & w \cdot x + b > 0 \end{cases}$$

- donde b es el bias del perceptron $b \equiv -threshold$
- *Intuitivamente* podemos pensar en el bias como en que tan fácil es que el perceptron sea 1
- son utiles para calcular operaciones logicas basicas como AND, OR y NAND

Ejemplo AND

- La pregunta es cuales son los pesos y el bias para que el perceptron representa un AND

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Algorítmo sencillo de entrenamiento

- este es un ejemplo de aprendizaje para un solo perceptron
- definiciones:
  - r tasa de aprendizaje $\in (0, 1)$
  - $y = f(z)$ output
  - $D = \{(x_1, d_1), ..., (x_s, d_s)\}$ conjunto de entrenamiento, donde d es el output deseado
  - w_i pesos

Algorítmo sencillo de entrenamiento

1. inicializar los pesos
2. para cada ejemplo j en el conjunto D
   a. calcular el output actual $y_j(t) = f(w \cdot x_j)$
   b. actualizar los pesos con $w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}$

Ejemplo AND

- pesos iniciales: $w_1 = 0.9$ y $w_2 = 0.9$ bias es -0.5
- para cada ejemplo checar el output con el perceptron actual
  - $x_1 = x_2 = 0 \implies \cdot w + b = .9 * 0 + .9 * 0 - 0.5 = -0.5$ $\implies$ $output = 0$ Correcto
  - $x_i = 1, x_j = 0$ implica que $x \cdot w + b = 0.4 > 0 \implies output = 1$ Incorrecto

Ejemplo AND

- actualizar pesos usando:
  - tasa de aprendizaje: $\eta = 0.5$
  - $\epsilon = \mathit{actual} - \mathit{prediction} = 0 - 1 = -1$
  - nuevo peso dado por $w_i' = w_i + \eta * \epsilon$
  - $w_i = 0.9 + 0.5 * (-1) = 0.4$
- $\implies$ $x_i = 1, x_j = 0$ ahora
  $x \cdot w + b = -0.1 \leq 0 \implies \mathit{output} = 0$ Correcto
- comprobamos que funcione para el ultimo caso: $x_i = 1$,
  $x \cdot w + b = 0.3 > 0 \implies \mathit{output} = 1$ Correcto
- El perceptron con $w_i = 0.4$, $b_i = 0.5$ describe perfectamente el caso de AND

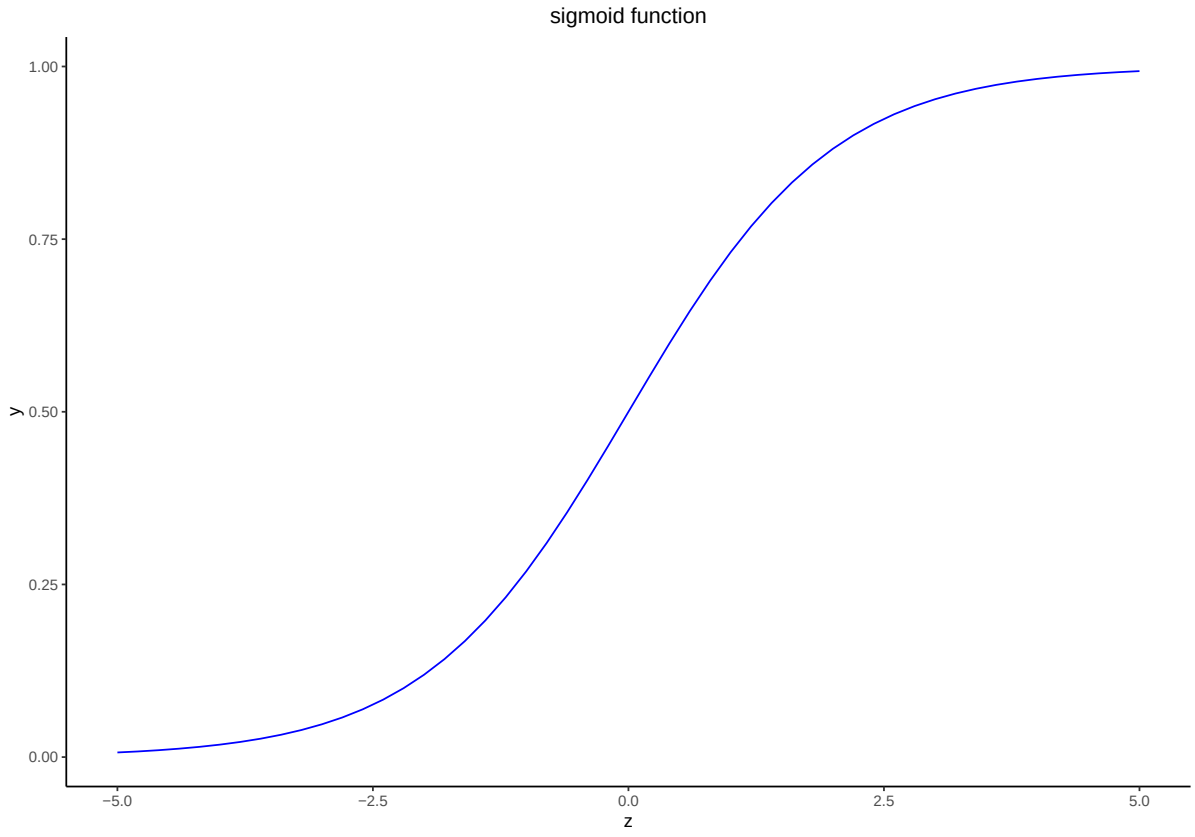
Sigmoidal

- función continua con outcome $\in [0, 1]$
- continuamente diferenciable
- una de las funciones mas populares usadas para redes neuronales
- $\sigma(w \cdot x + b)$

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + exp(-\sum_j w_j + b)}$$

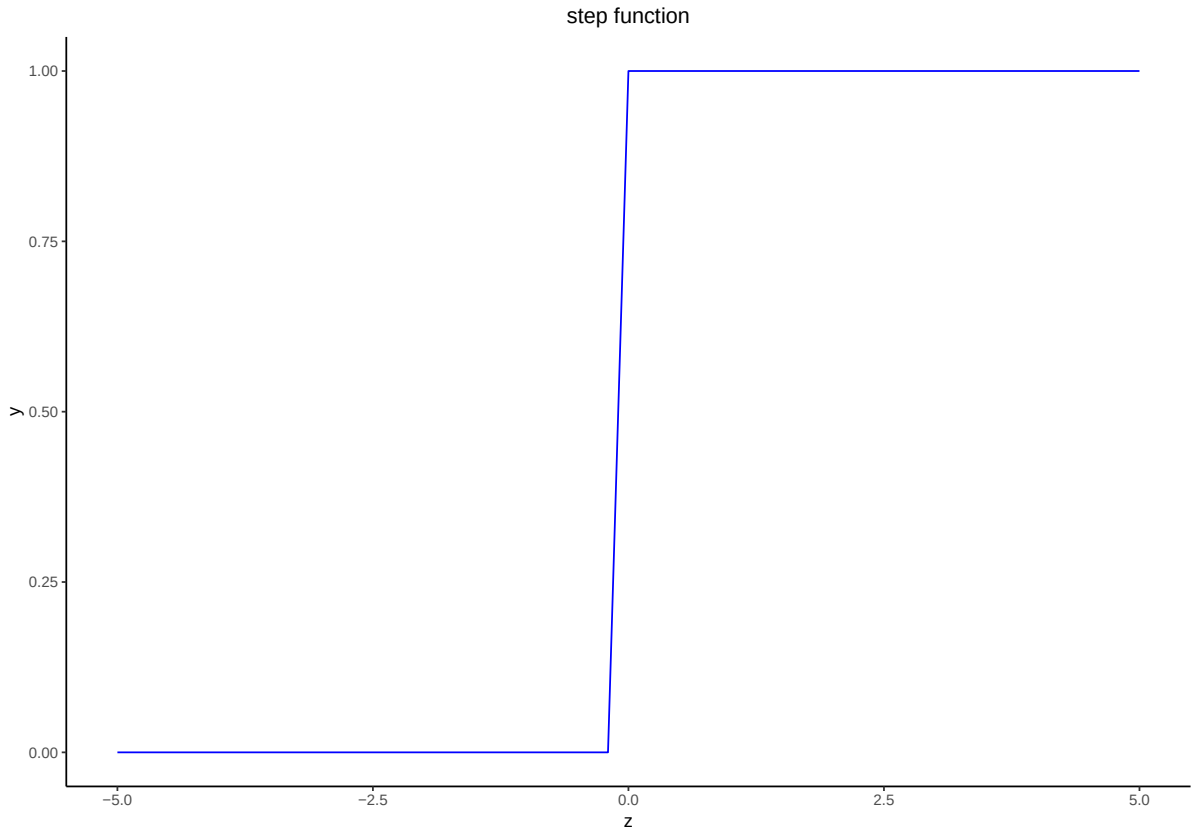
Sigmoidal vs Perceptron

- $z \equiv w \cdot x + b \approx \infty$ entonces $e^{-z} \to 0$ y $\sigma(z) \to 1$
- $z \equiv w \cdot x + b \approx -\infty$ entonces $e^{-z} \to$ inf y $\sigma(z) \to 0$
- muy similar al perceptron



Sigmoidal vs preceptron

- representacion grafica del preceptron



- la funcion Sigmoidal es una versión infinitamente diferenciable del perceptron

Vetajas de la funcion Sigmoidal

- diferenciable
- Δw_j y Δb producen un pequeño cambio en $\Delta output$

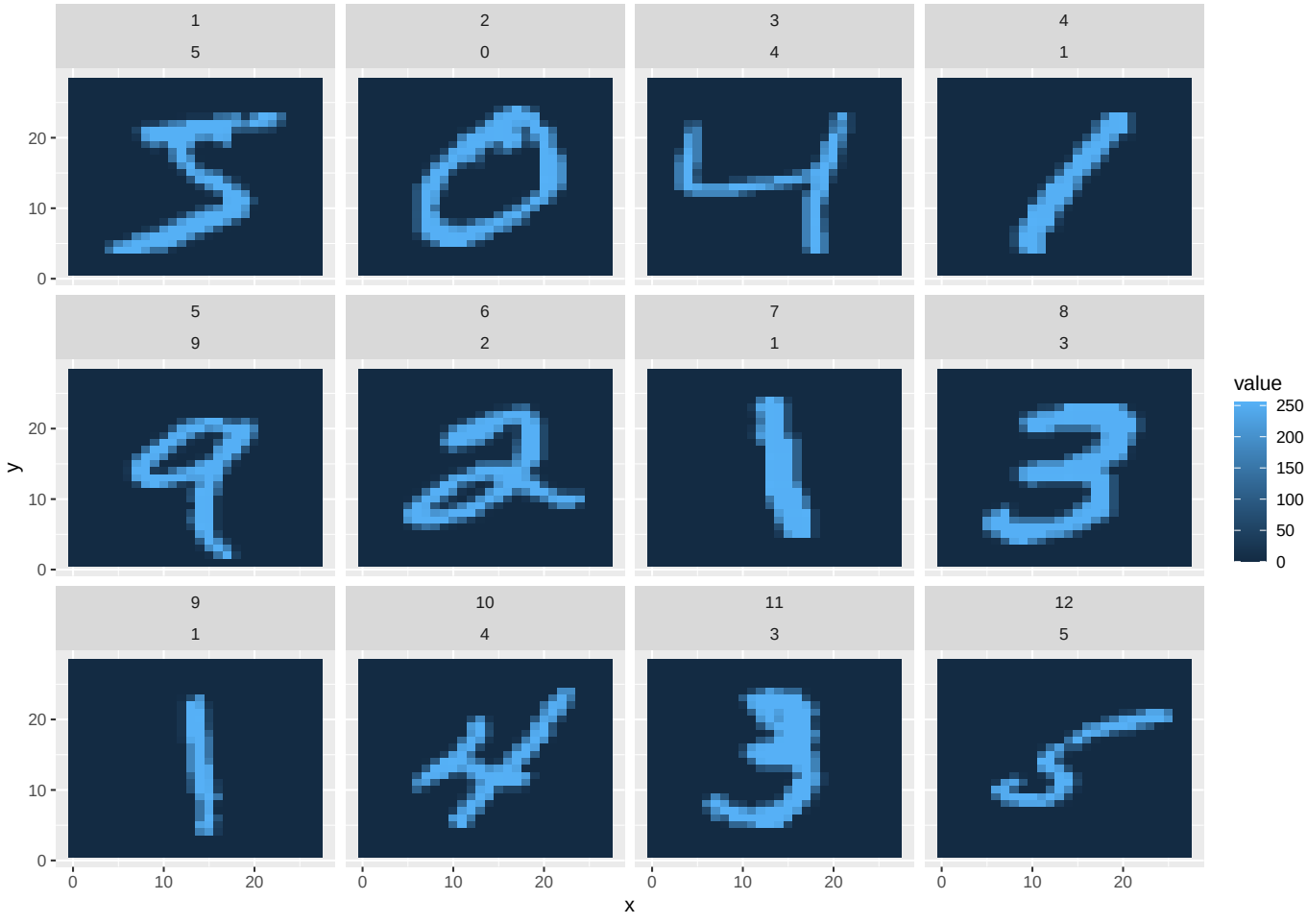
$$\Delta output \equiv \sum_j \frac{\delta output}{\delta w_j} \Delta w_j + \frac{\delta output}{\delta b} \Delta b$$

- $\Delta output$ es una función linear de los cambios Δw_j y Δb
- intuitivamente: pequeños cambios en los pesos o el bias generan un pequeño cambio en el output Muy bueno para el aprendizaje

Vetajas de la funcion Sigmoidal

- σ simplifica el álgebra de las parciales (e^x es una función linda)
- el resultado es un numero entre $[0, 1]$
- interpretación: la probabilidad de pertenecer a cierta categoría

Problema



Problema:

- clasificar dígitos escritos a manos
- *Humano*: fácil no?
- entonces debería de ser sencillo escribir un algoritmo que lo haga
- *Computadora*: difícil

Solución:

- redes neuronales
- en vez de escribir un conjunto de reglas (muchas) para diferenciar entre un 9 y un 0
- usar muchos ejemplos de números escritos a mano ya clasificados para entrenar la red
- la red neuronal usa los ejemplos para automaticamente inferir las reglas de que es un 9 o un 0
- $\uparrow \#$ ejemplos $\implies \uparrow$ precisión

Topología de la red

Topología de la red

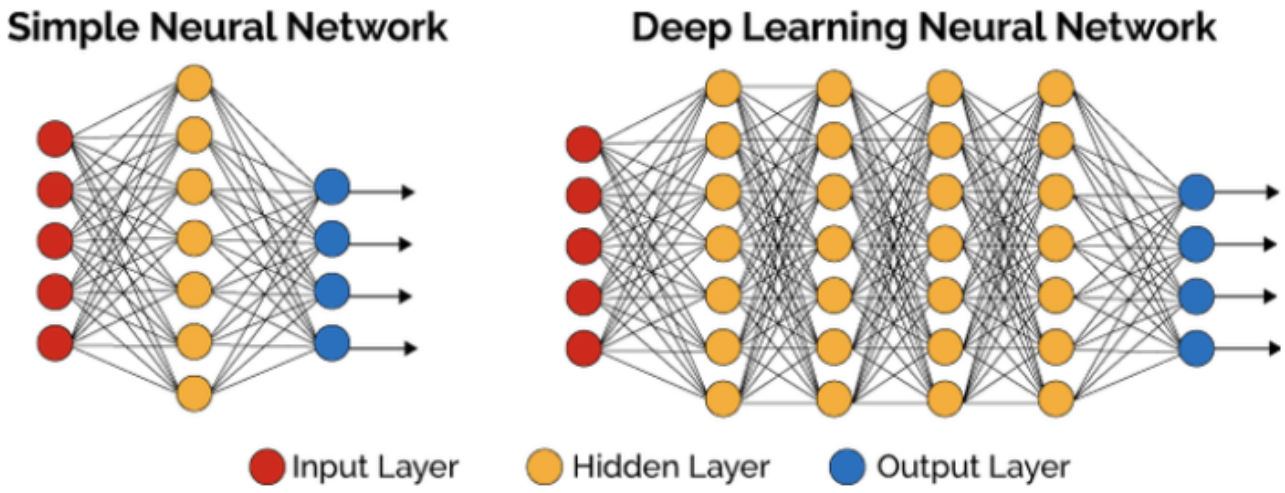
- conjunto de patrones y estructuras de capas interconectadas, 3 características principales
  1. número de capas
  2. tipo de conexión entre las capas (dirección de la información)
  3. número de nodos en cada capa

Número de capas

- capa de entrada: recibe las señales no procesadas, aplica la activación y pasa una nueva señal a la capa siguiente
- capas intermedias (hidden layers)
  - 1 sola (shallow NN)
  - > 1 (deep NN)
- capa de salida

Capas intermedias

- no es trivial determinar la cantidad que necesitas (heuristicas)
- trade-off entre la cantidad de capas y que tanto tarda en correr



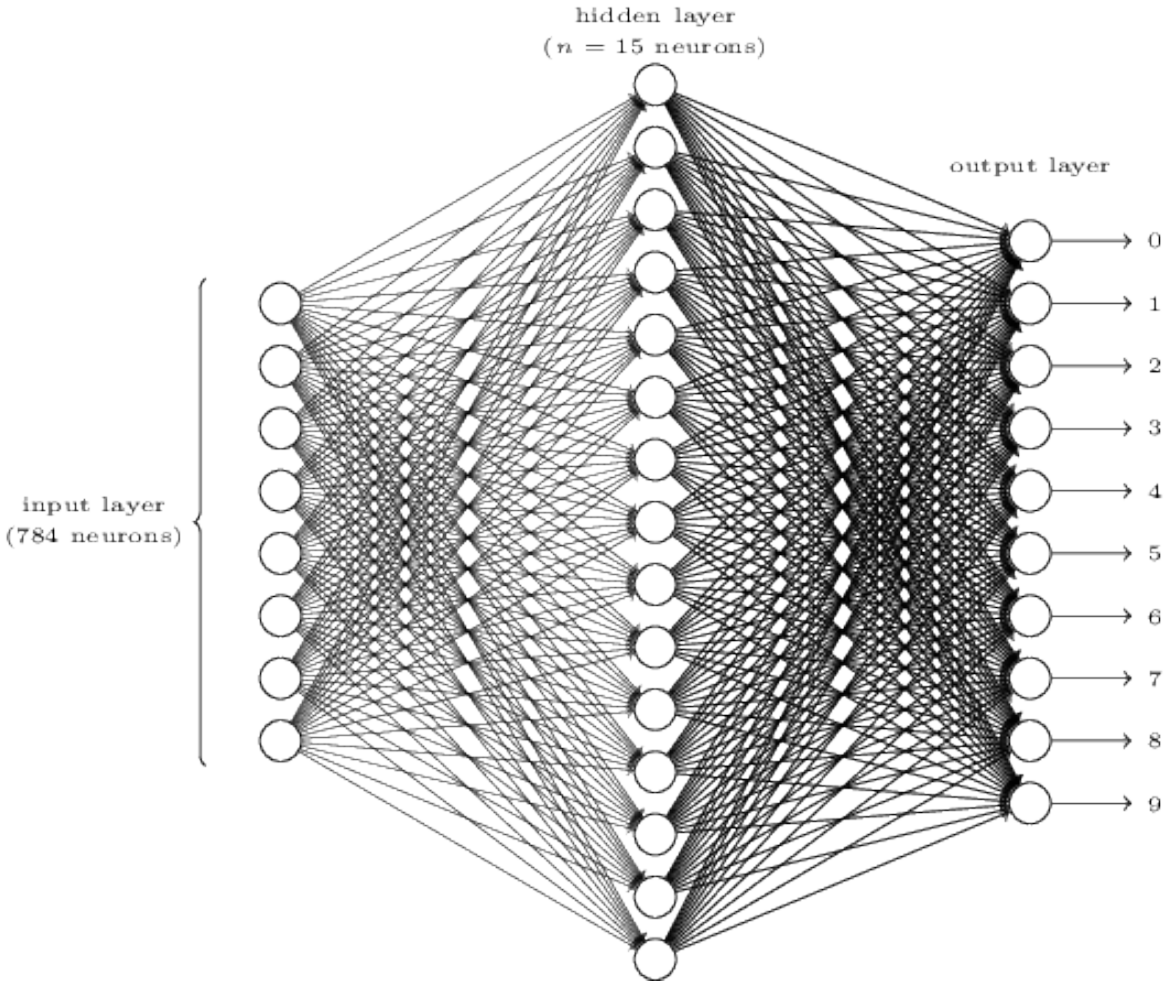
Dirección del viaje de la información

- *feedforward networks*: la información siempre viaja hacía adelante
  - los nodos de entrada $\rightarrow$ intermedios $\rightarrow$ salida
- *recurrentes*: la información puede viajar en ambas direcciones
  - nodos de entrada $\leftrightarrow$ intermedias $\leftrightarrow$ salida
  - incorporan memoria de corto plazo
  - parecidas a una cascada de información
- hay mas tipos de redes, volveremos a esto más adelante

Número de nodos por capa

- entrada: predeterminado por el número de características en los datos
- intermedios:
  - determinados antes de entrenar el algoritmo
  - determinados por medio de heurísticas
  - no hay una regla de como hacerlo
- salida: depende del problema en específico

Clasificar números escritos a mano



Clasificar números escritos a mano

- nodos de entrada:
  - número de pixeles de nuestros ejemplos 28x28 $\Longrightarrow$ 784 nodos de entrada en escala de grises
  - escala de grises quiere decir 0 $\equiv$ pixel todo blanco y 1 $\equiv$ pixel todo negro
- nodos intermedios:
  - por ahora 1 sola capa
  - n diferentes nodos
- nodos de salida:
  - 10 uno por dígito
  - probabilidad de que sea ese dígito
  - el nodo con el valor más alto será la clasificación

Clasificar números escritos a mano

- podemos diseñar una red con menos nodos de salida que describa lo mismo: SI
- cómo? podemos usar solo 4 nodos de salida
  - cada nodo como binario $\{0, 1\}$
  - esto es cierto porque $2^4 = 16 > 10$
- entonces por qué usamos una con 10 nodos de salida?
  - evidencia empírica: si corremos las dos la de 10 nodos clasifica mejor que la de 4
- no existe un método teórico que nos diga que es mejor, siempre hay que probar

Aprendizaje

Clasificar números escritos a mano: datos

- base de datos MNIST
- contiene 60,000 imágenes de números escritos a mano ya clasificados
- imágenes en escala de grises de 28x28 pixeles, recuerden 784 nodos de entrada
- x va a ser cada una de las unidades de entrenamiento
- x es un vector de 784x1 cada entrada es un pixel en escala de grises
- $y = y(x)$ donde y es un vector de 10x1
- $y(x) = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)^T$ implica que el número es un 6

Como aprende la Red Neuronal?

- queremos w_i y b_i que hagan que los resultados de nuestra RD se acerquen a $y(x)$
- para esto necesitaremos:
  1. función de costos, por ejemplo cuadrática

$$C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a||^2$$

  2. algoritmo para reducir el error

Función de costos

- medida de que tan 'bien' funciona nuestra RD
- también conocida como función de perdida u objetivo
- por lo general usaremos funciones diferenciables y continuas para facilitarnos la vida

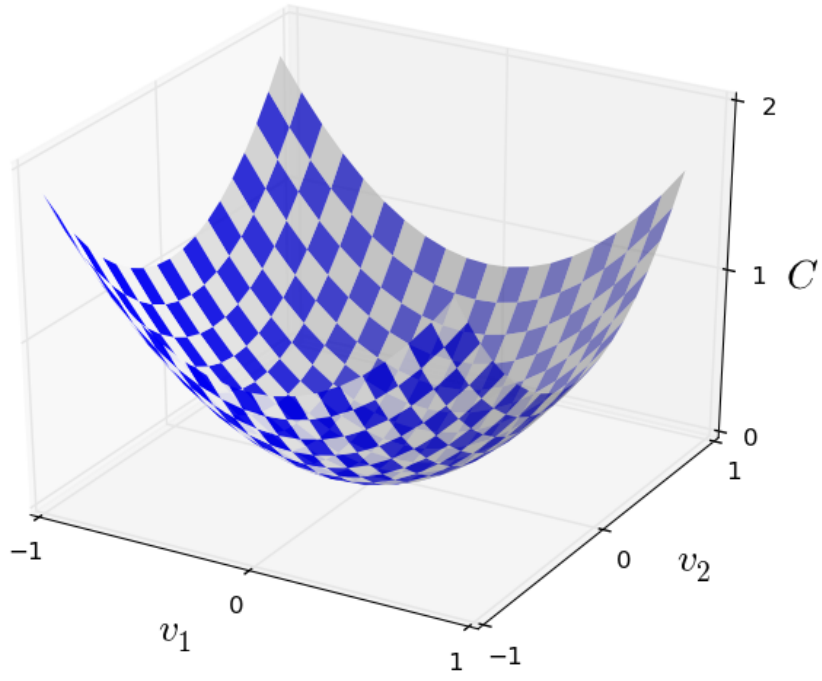
Gradient Descent

Gradient Descent

- $min \quad C(v)$ con $v = v_1, v_2, ...,$
- cómo encontramos este mínimo? 2 opciones
  - calculo: derivar, igualar a cero y revisar condiciones de segundo orden. Fácil si tu función es sencilla y tienes pocas variables
  - numéricamente

Gradient Descent: Ejemplo en 3d

- $C = C(v_1, v_2)$



Gradient Descent: Ejemplo en 3d

- empezar en un punto aleatorio de la gráfica
- $\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$
- el gradiente de C es: $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$
- entonces $\Delta C \approx \nabla C \cdot \Delta v$
- queremos que ΔC sea tal que vayamos lo más rápido posible al mínimo
- definimos: $\Delta v = -\eta \nabla C$ donde η es la velocidad de aprendizaje

Gradinet Descent

- $\Delta C \approx -\eta ||\nabla C||^2$
- usamos esto para actualizar nuestras v's y acercarnos más al mínimo
- $v \rightarrow v' = v - \eta \nabla C$
- si seguimos haciendo esto, eventualmente alcanzaremos el mínimo global
- intuitivamente el gradient descent es tomar pequeños pasos en la dirección que mas rápidamente disminuye C

Cómo usamos esto en una Red Neuronal?

- para encontrar los pesos y bias que minimicen la función de costos

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}$$

Problemas al aplicar el gradient descent

- función de costos: $C = \frac{1}{n} \sum_x C_x$
- es el promedio de distintos ejemplos de entrenamiento $C_x \equiv \frac{||y(x)-a||^2}{2}$
- para cada ejemplo tenemos que calcular ∇C_x, su gradiente
- esto puede ser muy lento
- cómo solucionamos esto? SGD (stochastic gradient descent)

Stochastic Gradient Descent

- estimar ∇C usando un subconjunto de ejemplos de entrenamiento
- el promedio de estos debería de acercarse al verdadero gradiente ∇C
- SGD escoge aleatoriamente m ejemplos de entrenamiento $x_1, ..., X_m$ mini-batch
- si m es lo suficientemente grande $\nabla C_{x_j} \rightarrow \nabla C$

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

Stochastic Gradient Descent

- escoges aleatoriamente un mini-batch de los ejemplos de entrenamiento y entrenas
- haces lo mismo pero con un subconjunto distinto de ejemplos
- cuando se hayan utilizado todos los ejemplos concluimos una época (epoch) de entrenamiento
- se puede repetir el proceso en una nueva época
- si tienes 60,000 ejemplos de entrenamiento y escoges mini-batches de tamaño 10 implica SGD es 6000 veces mas rápido que el GD

Backpropagation

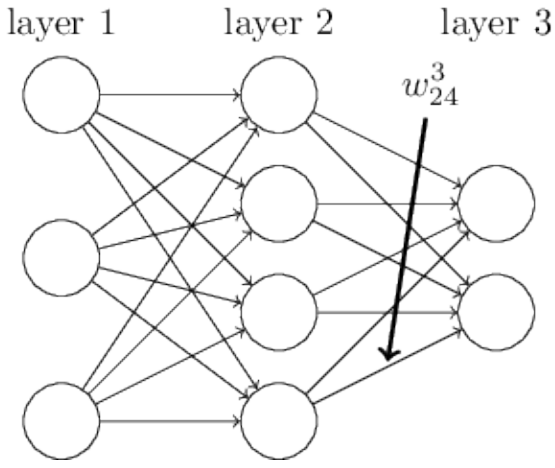
Abriendo la caja negra?

- qué hacen las capas intermedias?
- intuitivamente: pensar como le diríamos a la maquina como distinguir si algo es o no un perro
- cómo? buscando ciertas características en la foto
  - si hay ojos o no?
  - en que parte hay una lengua
  - etc..
- el problema:
  - estas sencillas reglas pueden salir mal
  - fácil distinguir un perro de una bicicleta
  - pero un perro de un cerdito?
  - video acá

Abriendo la caja negra?

- podemos resolver estos problemas dividiendo el problema en distintas redes neuronales
- sub-redes que responden preguntas para cada uno de los pixeles de una imagen
- al final tenemos una red que puede contestar si es un perro o no pero con muchas capas
- *deep learning*

Backpropagation



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

Backpropagation

- algoritmo rápido usado para aprender en RN
- Notación: w^l_{jk} es el peso de la conexión del nodo k en la capa (l-1) al nodo j de la capa l
- b^l_j es el bias del nodo j de la capa l
- a^l_j la activación de la neurona j de la capa l

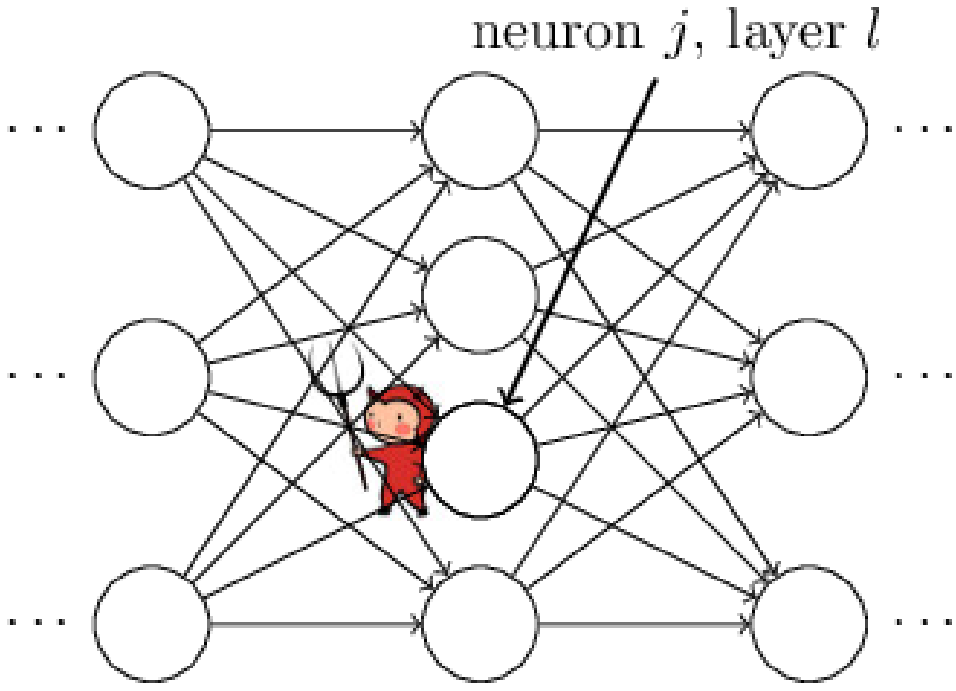
Backpropagation

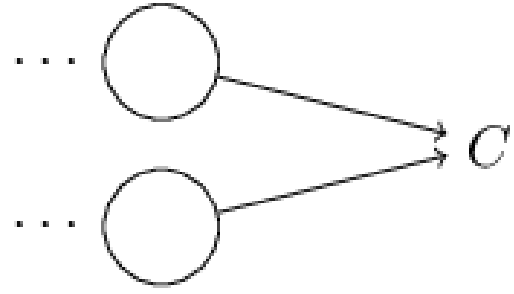
- la activación esta dada por: $a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j \right)$
- en forma vectorial como: $a^l = \sigma(w^l \cdot a^{ll-1} + b^l)$
- notación: $z^l \equiv w^l \cdot a^{l-1} + b^l$
- el objetivo de la backpropagation es calcular las derivadas parciales $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$

Backpropagation

- backpropagation necesita que la función de costos cumpla con:
  1. se pueda escribir como el promedio de las funciones de costo de los ejemplos individuales $C = \frac{1}{n} \sum_x C_x$
  2. se puede escribir como una función de los outputs de la red, es decir $C = C(a^L)$ donde a^L
- muchas funciones cumplen esto, la cuadrática $C_x = \frac{1}{2}||y - a^L||^2$ y $C = \frac{1}{2} \sum_j (y_j - a_j^L)^2$

Backrpopagation





Backpropagation

- definición: δ_j^l error en la neurona j en la capa l
- bakcpropagation: método para obtener δ_j^l y las derivadas parciales
- para entender como se define la delta supongamos que hay un diablito en alguna parte de la red en la neurona j de la capa l
- cuando viene el input al nodo el diablito lo altera agregando Δz_j^l
- el nodo procesa $z_j^l + \Delta z_j^l$
- y crea la activación $\sigma(z_j^l + \Delta z_j^l)$ en vez de solo $\sigma(z_j^l)$
- este pequeño cambio se propagara a todas las demás capas

Backrpopagation

- costo de esta alteración es de $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$
- el diablito quiere encontrar el Δz_j^l que reduzca C
- si $\frac{\partial C}{\partial z_j^l}$ es grande entonces el diablito puede disminuir mucho el costo escogiendo Δz_j^l con el signo contrario a la parcial
- si $\frac{\partial C}{\partial z_j^l}$ esta muy cerca de cero nada de lo que haga el diablito va a afectar mucho el costo
- intuitivamente: $\frac{\partial C}{\partial z_j^l}$ es una medida del error de ese nodo
- si el diablito decidiera cambiar a_j^l en vez de z_j^l todo funciona de la misma manera pero el álgebra es un poco peor

Backrpopagation: en resumen

1. ecuación 1: $\delta^L = \nabla_a C \odot \sigma'(z^L)$
   - si la C es cuadrática $\delta^L = (a^L - y) \odot \sigma'(z^L)$
   - $\odot$ es el producto elemento por elemento de los dos vectores
2. ecuación 2: $\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$
   - el error al nivel del output de la capa l
3. ecuación 3: $\frac{\partial C}{\partial b^l_j} = \delta^l_j$
   - tasa de cambio del costo con respecto al bias
4. ecuación 4: $\frac{\delta C}{\delta w^l_{jk}} = a^{l-1}_k \delta^l_j = a_{in} \delta_{out}$
   - tasa de cambio del costo con respecto a los pesos

Backrpopagation: en la práctica

1. input x: activación para la capa de entrada a^1
2. feedforward: para cada capa $l = 1, ..., L$ calcular $z^l = w^l a^{l-1} + b^l$ y $a^l = \sigma(z^l)$
3. error de salida: calcular $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. error de propagación hacia atrás: para cada $l = L-1, L-2, L-3, ..., 2$ calcular $\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma(z^l)$
5. resultado: el gradiente de la función de costos es $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta_j$ y $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

Backrpopagation: en la práctica

1. inputs: ejemplos de entrenamiento
2. para cada elemento del conjunto de entrenamiento x, encontrar $a^{1,1}$ y
   - feedforward: $l = 2, 3, ..., L$ calcular $z^{x,l} = w^l a^{x,l-1} + b^l$ y $a^{x,l} = \sigma(z^{x,l}$
   - error de salida: $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$
   - propagación hacia atrás del error: calcular $l = L-1, ..., 2$ los errores $\delta^{x,l} = (w^{l+1})^T \delta^{x,l+1} \odot \sigma'(z^{x,l})$
3. gradient descent: para cada $l = L, L-1, ..., 2$ actualiza los pesos usando la regla $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$ y $b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

- Para hacerlo stochastic necesitamos un loop afuera que calcule los mini-batches del conjunto de entrenamiento

Qué tan buena es una red neuronal comparada con otras técnicas de ML para nuestro ejemplo

- Si usamos otras técnicas para nuestro problema de clasificar dígitos
  - adivinar aleatoriamente nos da una accurancy de 10%
  - si usamos que tan obscura es la imagen de cada dígito y luego comparamos dígitos nuevos con la media y los asignamos tenemos accurancy de 22.25%
  - un SVM tiene una accurancy de 94.35%
  - la mejor RD a la fecha para este problema tiene una accurancy de 99.79%, casi tan bueno como un humano

Técnicas que mejoran el aprendizaje

Técnicas que mejoran el aprendizaje

1. otras funciones de costos
2. otras funciones de activación
3. regularización
4. hiper parámetros

Otras funciones de costos: Cross entropy

$$C = -\frac{1}{n}\sum_x [y * ln(a) + (1-y)ln(1-a)]$$

* aprende mas rápido que la función cuadrática * evita el problema de saturación de la cuadrática * si $\sigma(x) \to 0||1$ entonces $(a-y)\sigma'(z)x$ es muy pequeńa * $\implies$ aprendizaje muy lento en estos puntos * si $y \approx a$ entonces $C \approx 0$

Otras funciones de costos: Cross entropy

- por qué aprende mas rápido?
- $\implies \frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j(\sigma(z) - y)$
- entonces si $\sigma(z) - y$ es grande
- Esto quiere decir que entre peor calcule la red $\sigma(z) - y$ mas eápido aprende aprende

Otras funciones de activación (softmax)

- tipo de capa de salida
- $a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$
- distribución de probabilidad
- la función sigmoidial no puede ser interpretada así
- intuitivamente: softmax escala las z_j y las agrupa como una distribución de probabilidad

Overfitting

- la red funciona muy bien en el conjunto de aprendizaje pero no generaliza bien en datos fuera de estos
- si la precisión no aumenta en los ejemplos nuevos probablemente hay overfitting
- solución: conjunto de validación
  - calcular la precisión del conjunto de validación en cada ephoch
  - detener el entrenamiento cuando la precisión ya no aumente mas
- manera mas 'facil' de reducir el overfitting es tener mas ejemplos de entrenamiento
- el test de validacion va a ser usado para entrenar a los hiper parametros

Regularización

- técnica para reducir el overfitting sin tener mas ejemplos de entrenamineto y sin cambiar la estructura de la red
- Existen varias técnicas de regularización :
  - Cambiar la función de costos (strong)
  - Modificar la manera de muestreo (weak)
  - Modificar el algoritmo de entrenamiento (strong)

Regularizacion L2

- agregar un elemento extra a la función de costos
- $C = -\frac{1}{n}\sum_{x_j}[y_j * ln(a_j^L) + (1 - y_j)ln(1 - a_j^L) + \frac{\lambda}{2n}\sum_w w^2]$
- el termino de regularización (el nuevo en la cross-entropy) es una suma ponderada de todos los pesos de la red, no incluye los biases
- Este termino hace que la red prefiera aprender pesos pequeños a grandes

Por qué la regularización reduce el overfitting?

- pesos mas pequeños
- difícil aprender desviaciones pequeñas (ruido) de los datos
- la red es forzada a aprender cosas mas generales de los ejemplos
- esto hace que la red regularizada difícil aprender desviaciones pequeñas (ruido) de los datos
- Ojo esto no quiere decir que siempre queremos el modelo mas sencillo o que siempre el modelo mas simple es mejor

Hiper parámetros

- cómo vamos a escoger cosas como: la tasa de aprendizaje, el parámetro de regularización, la cantidad de nodos, etc
  - manualmente
  - grid search
  - búsqueda aleatoria
  - optimización bayesiana