

Redes Densas: Parte 2

Orlando Ramos Flores

Contenido

- Estructura de red neuronal (continuación)
- Funciones de activación
- Funciones de pérdida y de costo
- Función softmax
- Retropropagación
 - Función de optimización



Estructura general de una red neuronal

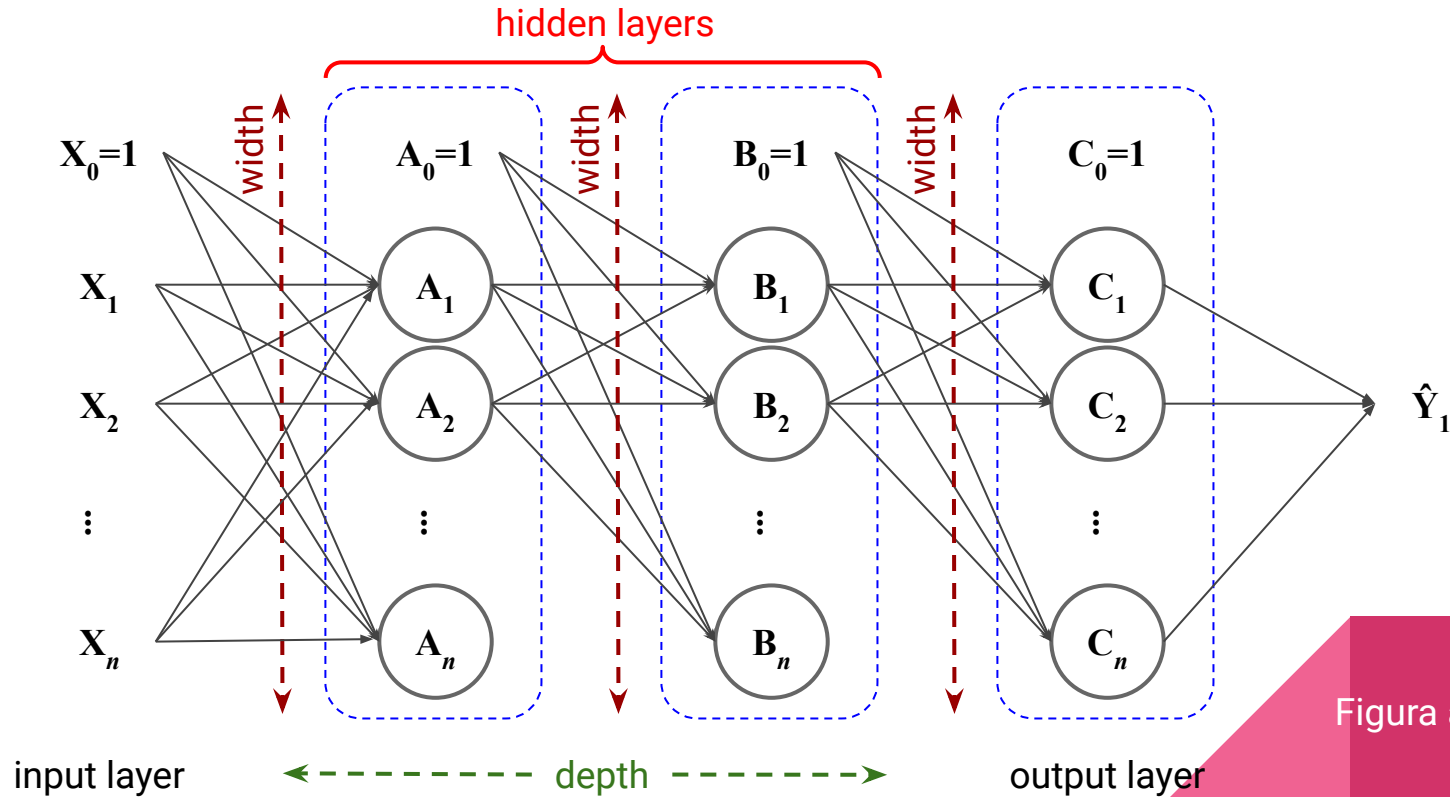


Figura adaptada de [1]

Puntos relevantes

1. Las unidades se organizan en capas, y cada capa contiene una o más unidades.
2. La última capa se denomina capa de salida.
3. Todas las capas antes de las capas de salida se denominan capas ocultas.
4. El número de unidades en una capa se conoce como el ancho (width) de la capa.
5. No es necesario que el ancho de cada capa sea el mismo, pero la dimensión debe ser alineada.
6. El número de capas se denomina profundidad (deep) de la red, de aquí es donde viene el nombre de **deep learning**.

Puntos relevantes

7. Cada capa toma como entrada la salida producida por la capa anterior, excepto la primera capa, que consume la entrada.
8. La salida de la última capa es la salida de la red y es la predicción generada en base a la entrada.
9. Una red neuronal puede verse como una función $f_{\theta}: \mathbf{x} \rightarrow \mathbf{y}$, que toma como entrada $\mathbf{x} \in \mathbb{R}^n$ y produce como salida $\mathbf{y} \in \mathbb{R}^m$ cuyo comportamiento es definido por $\theta \in \mathbb{R}^p$. Ahora podemos ser más precisos acerca de θ ; es simplemente una colección de todos los pesos \mathbf{w} para todas las unidades en la red.
10. El diseño de una red neuronal implica, entre otras cosas, definir la estructura general de la red, incluida la cantidad de capas y el ancho de estas capas.

Funciones de Activación

Función de activación: Propiedades

- En teoría, cuando una función de activación no es lineal, una red neuronal de dos capas puede aproximarse a cualquier función (dado un número suficiente de unidades en la capa oculta). Por lo tanto, buscamos funciones de activación no lineales en general.
- Una función que es **continuamente diferenciable** permite calcular gradientes y usar métodos basados en gradientes para encontrar los parámetros que minimizan nuestra función de pérdida sobre los datos. Si una función no es continuamente diferenciable, los métodos basados en gradientes no pueden progresar.
- Una función cuyo rango es finito (en lugar de infinito) conduce a un rendimiento más estable con los métodos basados en gradientes.



Función de activación

- La función de activación es la función no lineal que aplicamos sobre los datos de entrada que llegan a una neurona en particular, y la salida de la función se enviará a las neuronas presentes en la siguiente capa como entrada.



Función de activación: Unidad Lineal

La unidad lineal es la unidad más simple que transforma la entrada cómo:

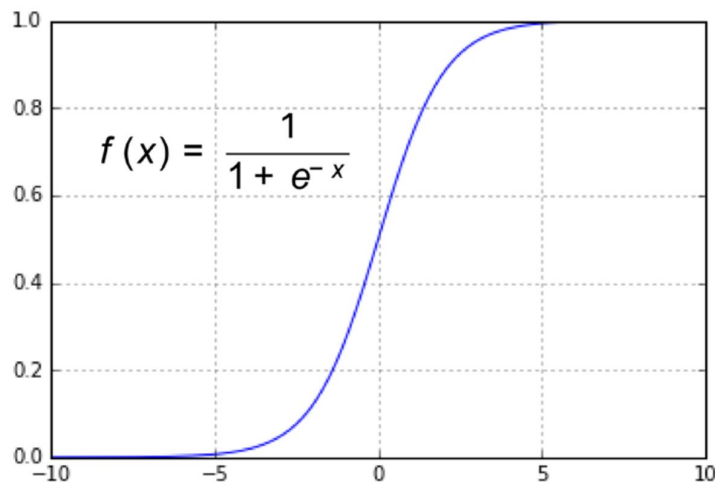
$$y = \mathbf{w} \cdot \mathbf{x} + \mathbf{b}.$$

Como su nombre lo indica, la unidad no tiene un comportamiento no lineal



Función de activación: Sigmoides o Logística

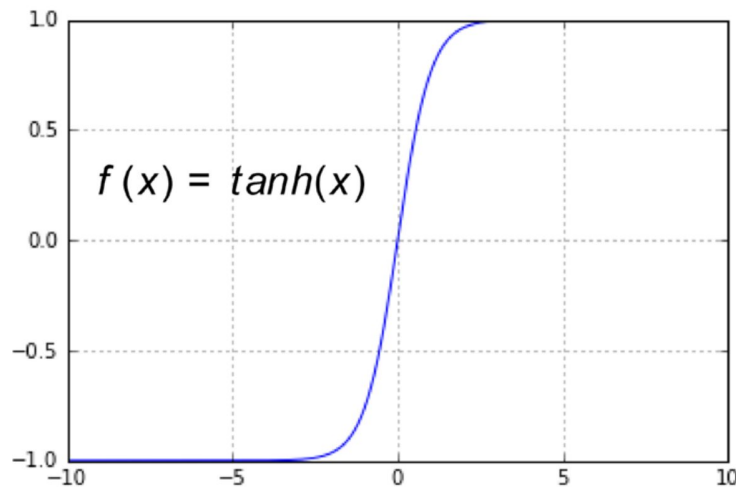
La función sigmoide es una función no lineal que toma un número de valor real como entrada y comprime todas sus salidas al rango de $[0,1]$ que puede interpretarse como una probabilidad. Su forma matemática y derivada:



$$f(x) = \frac{1}{1 + e^{-x}}$$
$$f'(x) = \frac{\partial f(x)}{\partial x} = f(x) * (1 - f(x))$$
$$y = \frac{1}{1 + e^{-(wx+b)}}$$

Función de activación: Tangente hiperbólica (\tanh)

\tanh es una función de activación no lineal que comprime todas sus entradas al rango $[-1, 1]$. La forma matemática de la función de activación de \tanh y su derivada se dan a continuación:

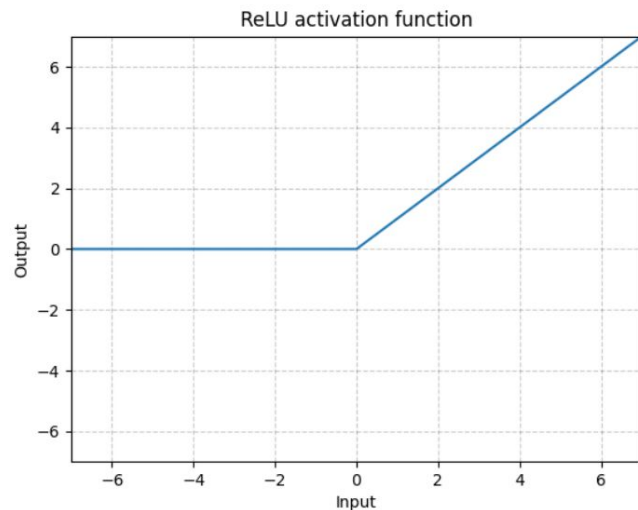


$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f'(x) = \frac{\partial f(x)}{\partial x} = (1 - (f(x))^2)$$

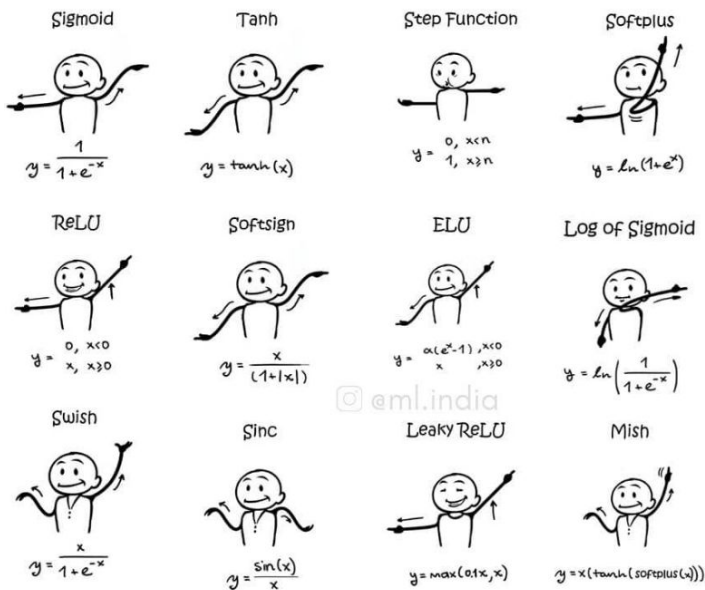
Función de activación: Unidad lineal rectificada (ReLU)

ReLU, una función de activación no lineal, se introdujo en el contexto de una red neuronal de convolución. ReLU no es una función centrada en cero, a diferencia de la función *tanh*. Su notación es $f(x) = \max(0, x)$.

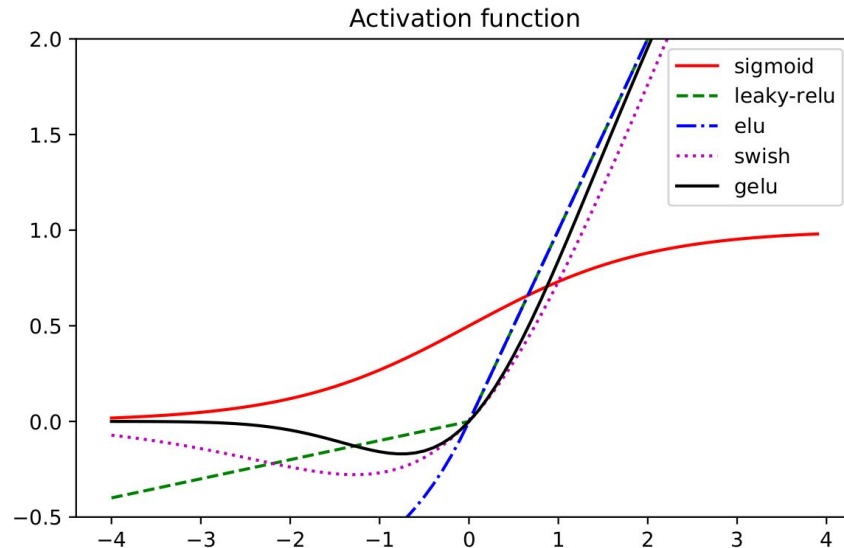


Funciones de activación

Deep Learning Activation Functions



Source: <https://sefiks.com/2020/02/02/dance-moves-of-deep-learning-activation-functions/>



Funciones de Pérdida y Costo


Función de pérdida

Se asume que los datos tienen la forma $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ donde $x \in \mathbb{R}^n$ y $y \in \{0, 1\}$, que es el objetivo (target) de interés (en este caso es un objetivo binario). Para un solo punto de datos, podemos calcular la salida de la red neuronal, que denotamos como \hat{y} .

Ahora necesitamos calcular qué tan buena es la predicción de nuestra red neuronal \hat{y} en comparación con y . Aquí viene la noción de una **función de pérdida**.

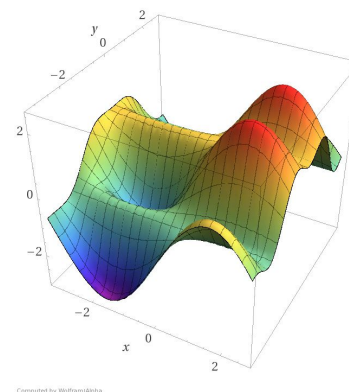
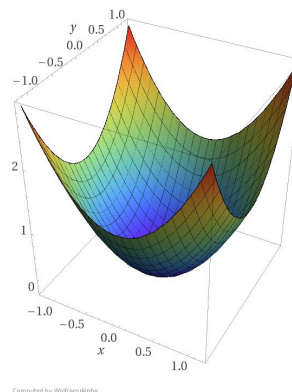
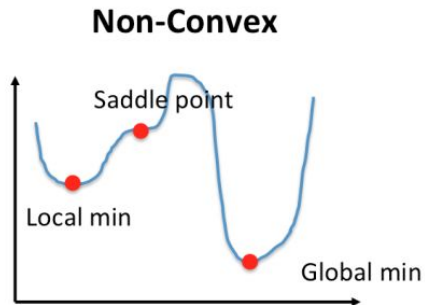
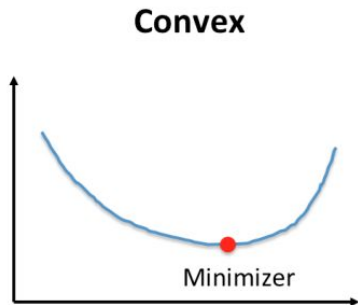
Una función de pérdida mide la discrepancia entre \hat{y} e y que denotamos por ℓ .

Normalmente ℓ calcula la discrepancia entre \hat{y} e y sobre una serie de puntos de datos en lugar de un solo punto de datos.



Funciones de pérdida

La función de pérdida debe de ser convexa



Función de costo

- Anteriormente, denotamos a θ como la colección de todos los pesos y términos de sesgo de todas las capas de la red. Supongamos que θ se ha inicializado con valores aleatorios. Denotamos por f_{NN} la función general que representa la red neuronal.
- Recordar que \hat{y} toma un solo punto de datos y calcula la salida de la red neuronal. También podemos calcular el desacuerdo con la salida actual y usando la función de pérdida $\ell(\hat{y}, y)$ que es $\ell(f_{\text{NN}}(\mathbf{x}, \theta), y)$
- Se calcula el gradiente de esta función de pérdida, denotada por: $\nabla \ell(f_{\text{NN}}(\mathbf{x}, \theta), y)$
- Se actualiza θ usando el descenso más pronunciado como $\theta_s = \theta_{s-1} - \alpha \cdot \nabla \ell(f_{\text{NN}}(\mathbf{x}, \theta), y)$, donde s denota un solo paso (podemos tomar muchos de estos pasos sobre diferentes puntos de datos en nuestro conjunto de entrenamiento una y otra vez hasta que tengamos un valor razonablemente bueno para $\ell(f_{\text{NN}}(\mathbf{x}, \theta), y)$).

¿La Función de pérdida y función de costo son iguales?



Función de pérdida VS Función de costo

Sí, la función de costo y la función de pérdida son sinónimos y se usan indistintamente, pero son "diferentes".

- Una función que se define en una única instancia (punto) de datos se denomina función de pérdida. Una **función de pérdida/función de error** es para un solo ejemplo/entrada de entrenamiento.

$$\textit{Absolute Loss} = |y - \hat{y}|$$

- Una función que se define en una instancia de datos completa se denomina función de costo. Es la pérdida promedio sobre todo el conjunto de datos de entrenamiento.

$$\textit{Mean Absolute Error} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Funciones de pérdida

- **Regresión.** Las tareas de regresión tratan con datos continuos. Por ejemplo, predecir el precio del valor inmobiliario, salarios, o precios de acciones, etc.
 - Mean Absolute Error
 - Mean Squared Error
 - Root Mean Squared Error
 - Root Mean Squared Logarithmic Error
- **Clasificación.** Por ejemplo, análisis de sentimientos, clasificación de imágenes, etc.
 - Binary Cross Entropy
 - Categorical Cross Entropy (Multiclass classification)
 - Kullback Leibler Divergence Loss
 - Negative Log Likelihood

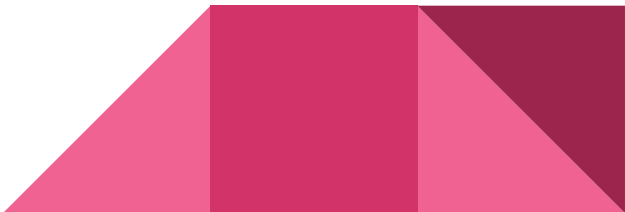


Funciones de pérdida

Entropía Binaria Cruzada (EBC)

$$ECB(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^N \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Entropía Cruzada Categórica

$$\frac{1}{N} \sum_{i=1}^N (\hat{y}^{(i)} - y^{(i)})$$


Función Softmax

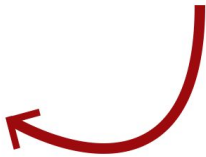
Función Softmax

La función Softmax es una forma de regresión logística que normaliza un valor de entrada en un vector de valores que sigue una distribución de probabilidad cuya suma total es 1. Los valores de salida están entre el rango $[0,1]$, lo cual es bueno porque podemos evitar la clasificación binaria y acomodar tantas clases o dimensiones en nuestro modelo de red neuronal. Esta es la razón por la cual softmax a veces se denomina regresión logística multinomial.

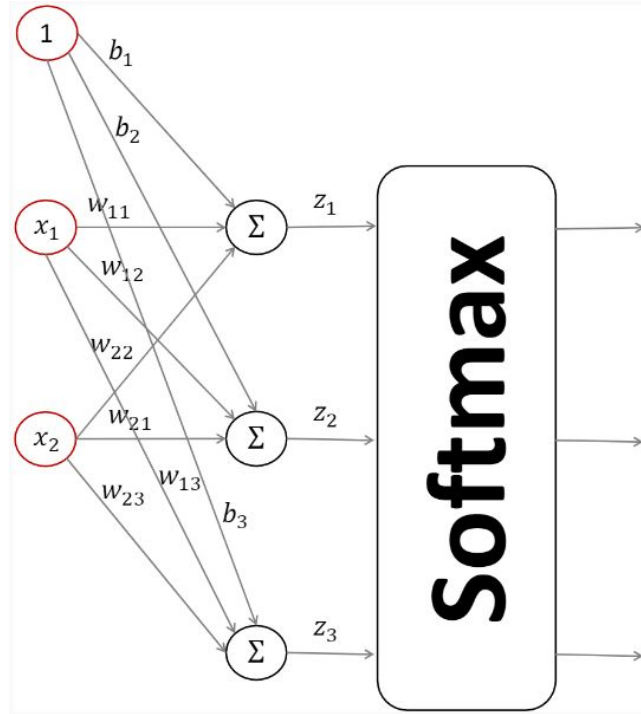
$$P(y=j \mid \theta^{(i)}) = \frac{e^{\theta_j^{(i)}}}{\sum_{k=0}^k e^{\theta_k^{(i)}}}$$

where $\theta = w_0x_0 + w_1x_1 + \dots + w_kx_k = \sum_{i=0}^k w_i x_i = \mathbf{w}^T \mathbf{x}$

Softmax function



Función Softmax



La función Softmax se emplea entre capas

Alternativas a Softmax

- Taylor softmax
- Soft-margin softmax
- SM-Taylor softmax
- Softmax Networks
- Hyperspherical Prototype networks



Retropropagación

Algoritmo de retropropagación (backtracking)

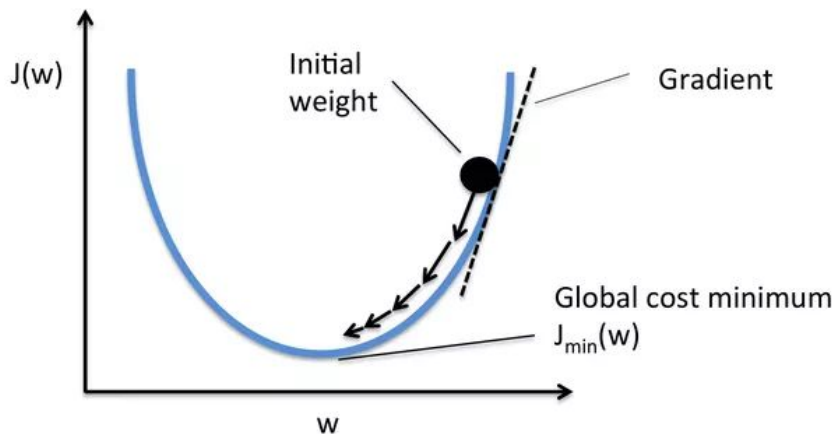
El algoritmo de retropropagación permite que la información del costo fluya hacia atrás a través de la red, para calcular el gradiente.

Para predicciones precisas, es necesario minimizar el error calculado. En una red neuronal, esto se hace usando retropropagación. El error actual generalmente se propaga hacia atrás a una capa anterior, donde se usa para modificar los pesos y el sesgo de tal manera que se minimice el error. Los pesos se modifican mediante una función denominada **Función de Optimización**.



Funciones de optimización

Las funciones de optimización suelen calcular el gradiente, es decir, la **derivada parcial** de la **función de pérdida** con respecto a los pesos, y los pesos se modifican en la dirección opuesta al gradiente calculado. Este ciclo se repite hasta llegar a los mínimos de la función de pérdida.



$$J(w) = p - \hat{p} \text{ función de pérdida}$$

Funciones de optimización

Los algoritmos de optimización de la función de costos intentan encontrar los valores óptimos para los parámetros del modelo al encontrar los mínimos globales de las funciones de costos. Algunos son:

- Gradient Descent
- RMS Prop
- Adam



Funciones de optimización: Gradient Descent

El algoritmo de descenso de gradiente hace uso de gradientes de la función de costo para encontrar el valor óptimo para los parámetros. El descenso de gradiente es un algoritmo iterativo. Intenta encontrar un mínimo global. En cada iteración:

- Se encuentra el costo de los datos
- Se calcula la diferenciación (derivada) parcial de la función de costo con respecto a los pesos y el sesgo.
- Luego, los pesos y el sesgo se actualizan haciendo uso de los gradientes de la función de costo y la **tasa de aprendizaje** (learning rate) α . El valor de α puede variar de 0.0 a 1.0. Cuanto mayor sea el valor de α , mayor es el número de pasos que se toman para encontrar el mínimo global de la función de costo
- Se repiten los pasos mencionados anteriormente hasta que se complete un número específico de iteraciones o cuando se alcance un mínimo global.

Funciones de optimización: RMS Prop

Root Mean Squared Prop (RMS Prop) es un algoritmo de optimización que es muy similar a Gradient Descent pero los gradientes se suavizan y elevan al cuadrado y luego se actualizan para alcanzar pronto el mínimo global de la función de costo. En cada iteración:

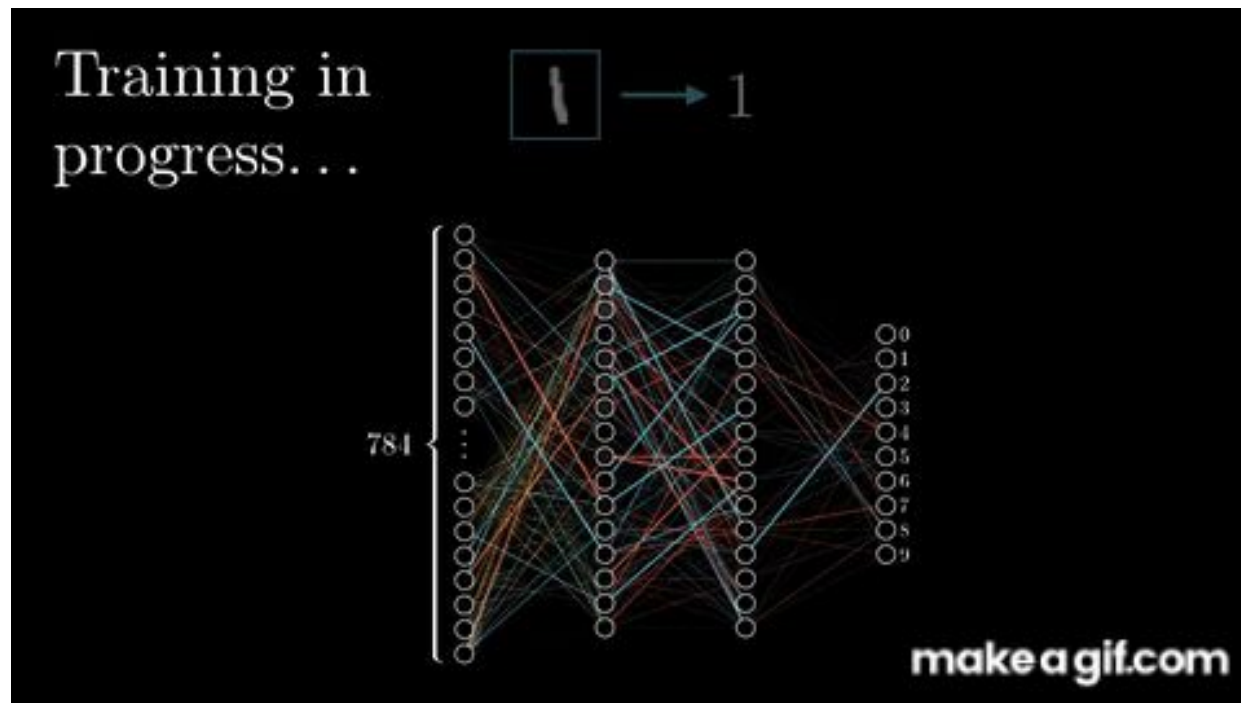
- Se encuentra el costo de los datos
- Se calcula la diferenciación parcial de la función de costo con respecto a los pesos y el sesgo
- Los parámetros de pesado y sesgo se suavizan y luego se actualizan mediante el uso de gradientes de la función de costo y α (tasa de aprendizaje)
- Repetir los pasos mencionados anteriormente hasta que se complete un número específico de iteraciones o cuando se alcance un mínimo global

Funciones de optimización: Adam

Adaptive Moment Estimation (Adam) es un algoritmo que surgió al combinar Gradient Descent con impulso y RMS Prop. En cada iteración:

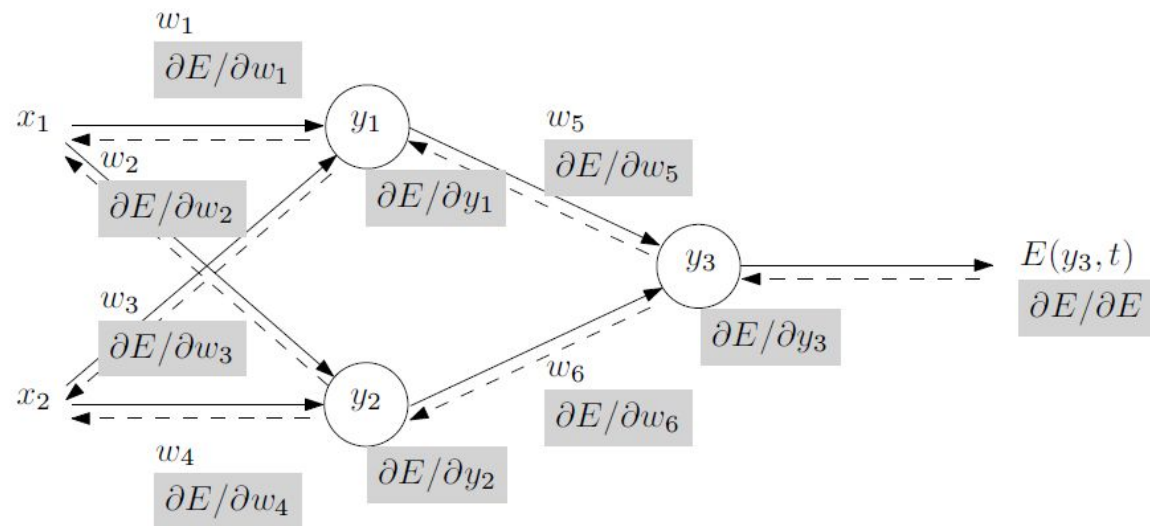
- Se encuentra el costo de los datos
- Se calcula la diferenciación parcial de la función de costo con respecto a los pesos y el sesgo
- Los pesos y sesgos se suavizan con la técnica utilizada en RMS Prop y Gradient Descent con impulso (momentum) y luego los pesos y sesgos se actualizan utilizando gradientes de función de costo y α (tasa de aprendizaje)
- Repetir los pasos mencionados anteriormente hasta que se complete un número específico de iteraciones o cuando se alcance un mínimo global

Retropropagación



Retropropagación (Diferenciación automática)

(a) Forward pass



(b) Backward pass

