Tema 2.2

Optimización I: Función de pérdida y descenso por gradiente

Miguel Ángel Martínez del Amor

Deep Learning

Departamento Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Contenido

- Función de coste
- Descenso por gradiente
- Variantes del descenso por gradiente

Función de coste

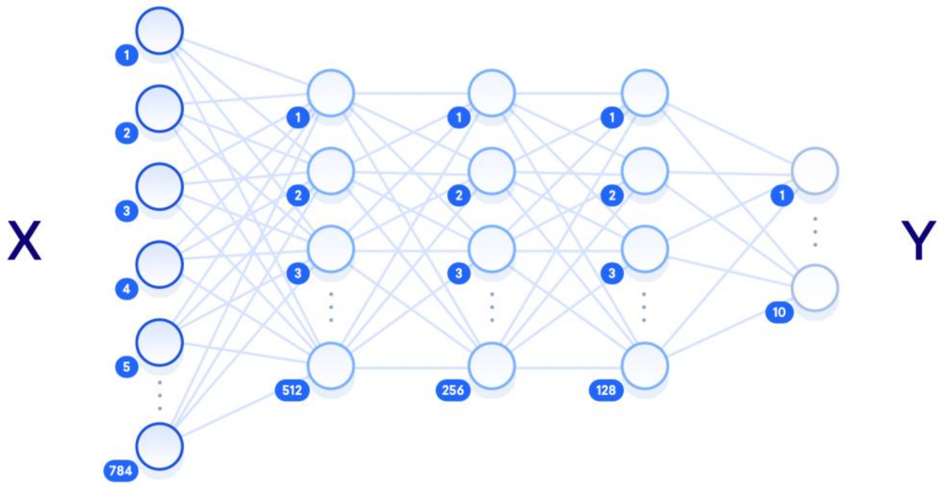
- **Problema de aprendizaje**: Dados un conjunto de datos de los que se conoce la salida deseada, proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Función de coste

- **Problema de aprendizaje**: Dados un conjunto de datos de los que se conoce la salida deseada, proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

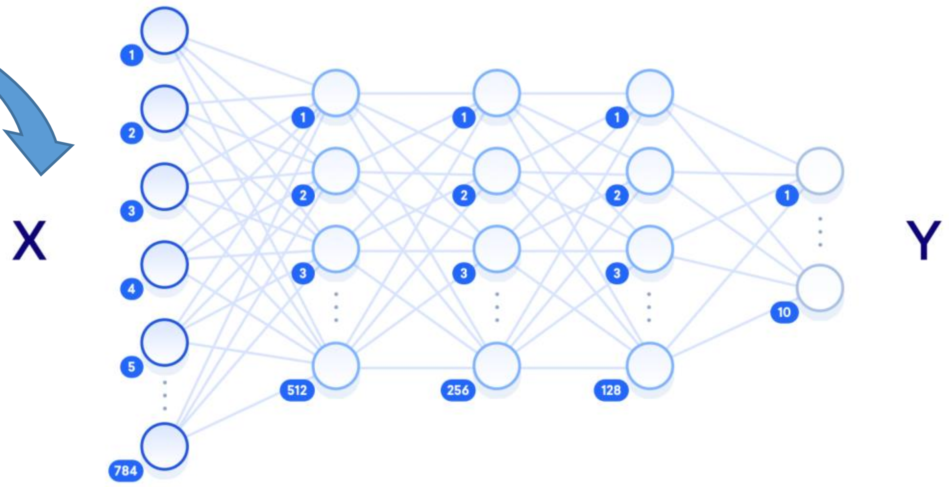
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa



Función de coste

- **Problema de aprendizaje**: Dados un conjunto de datos de los que se conoce la salida deseada, proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

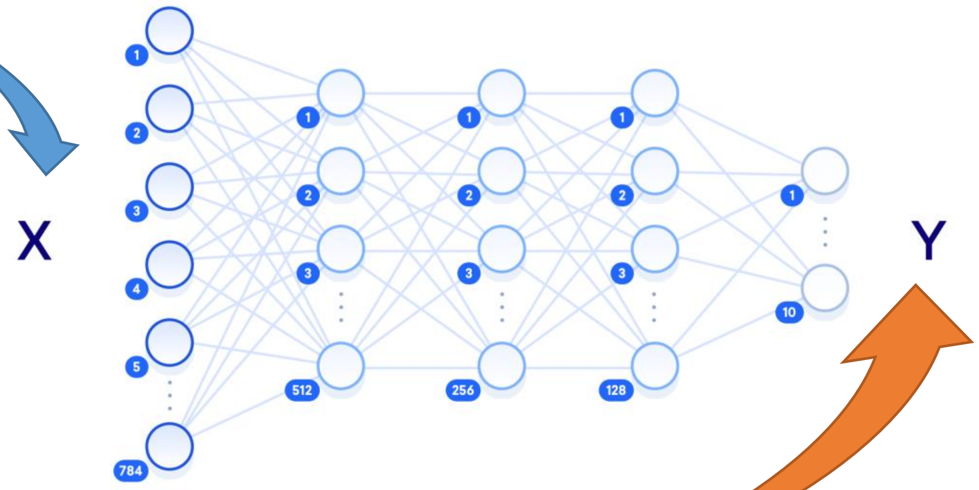
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa



Función de coste

- **Problema de aprendizaje**: Dados un conjunto de datos de los que se conoce la salida deseada, proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

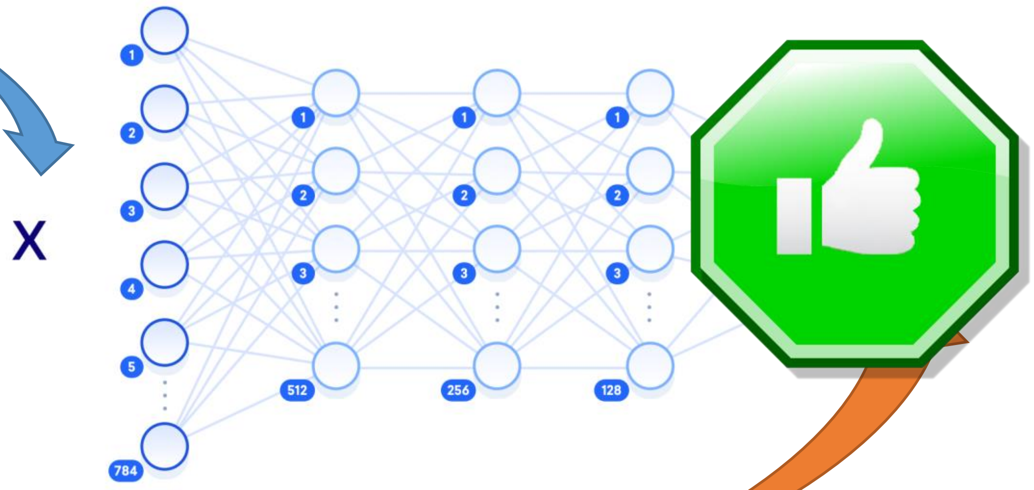
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa



Función de coste

- **Problema de aprendizaje**: Dados un conjunto de datos de los que se conoce la salida deseada, proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa



Función de coste

- Necesitaremos ajustar los parámetros del modelo (pesos) para que se comporte mejor con los datos.
- Por tanto, necesitamos **cuantificar** cuánto de "***buena***" es nuestra red para un ejemplo.
- En otras palabras, cuantificar cómo de buenos son nuestros pesos W
- Definiremos:
  - La **función de pérdida (loss):** para un ejemplo
  - La **función de coste** (**cost**): para un conjunto de ejemplos (dataset, batch)
  - La **función objetivo** a minimizar. La función de coste es una función objetivo.
- Estas funciones se suelen confundir

Función de coste: regresión lineal

- Recordemos, regresión lineal para valores continuos
- La función de coste es **MSE** (error cuadrático medio).
  - El dataset tiene *m* ejemplos
  - Sean θ los parámetros del modelo (p.ej. los pesos W).
  - $f_\theta(x^i)$ es la salida del modelo (con parámetros θ) para el ejemplo *i* con características x^i
  - y^i es la salida esperada (conocida, ground truth) para el ejemplo i.
  - *J(*θ*)* es la función de coste para los parámetros θ.

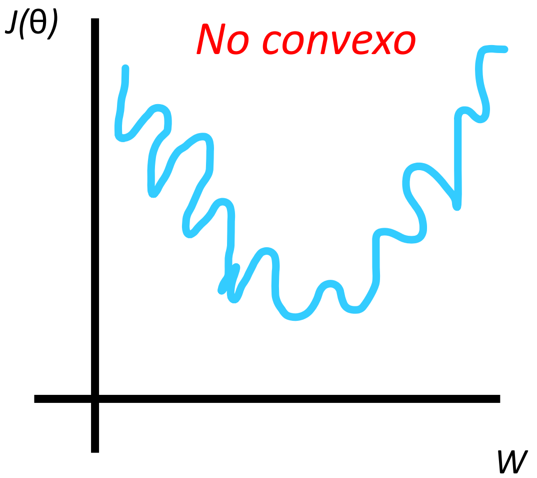
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(f_\theta\left(x^i\right) - y^i\right)^2$$

Función de coste: perceptrón

- En **clasificación binaria** mediante **regresión logística** (y=0 o 1)
  - O perceptrón con función sigmoide como activación
- La función de coste MSE no sería convexa en este caso.
- Queremos que lo sea para encontrar un mínimo.

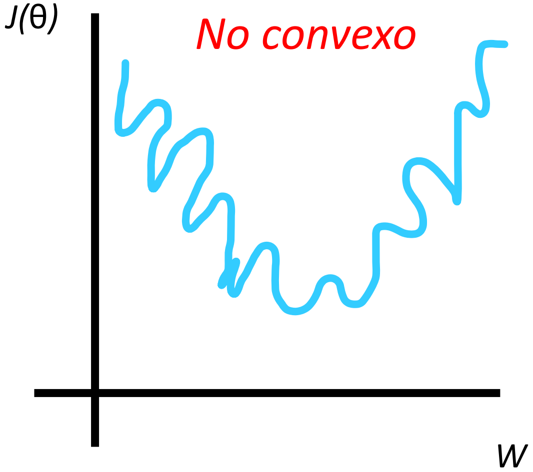
Función de coste: perceptrón

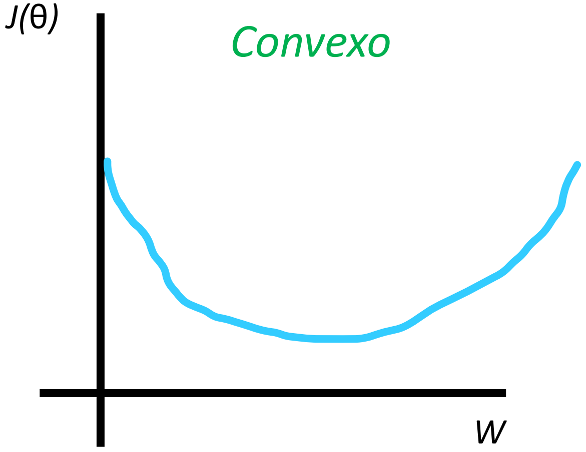
- En **clasificación binaria** mediante **regresión logística** (y=0 o 1)
  - O perceptrón con función sigmoide como activación
- La función de coste MSE no sería convexa en este caso.
- Queremos que lo sea para encontrar un mínimo.



Función de coste: perceptrón

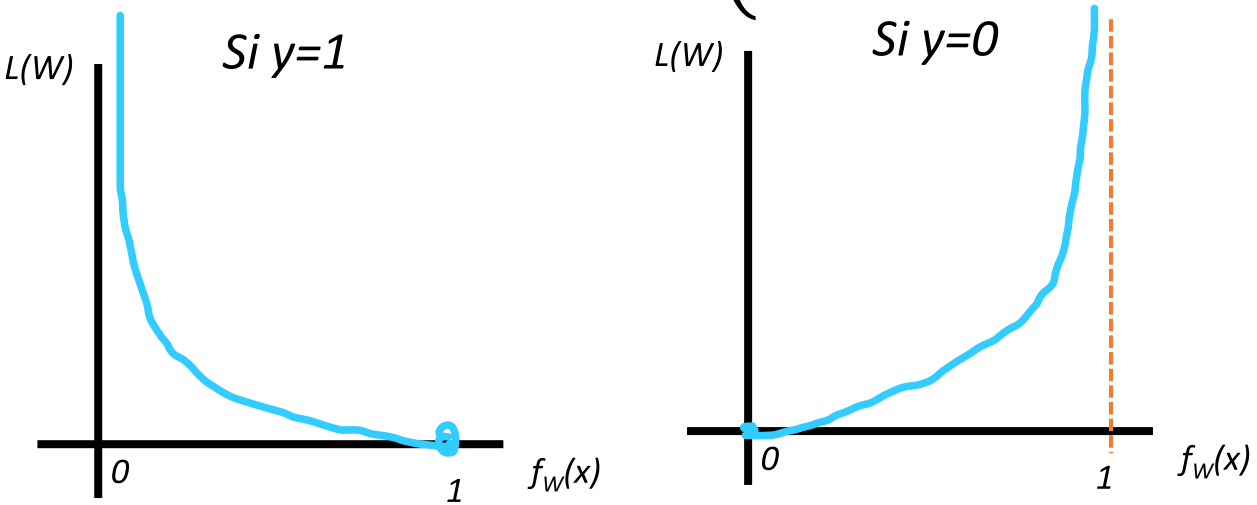
- En **clasificación binaria** mediante **regresión logística** (y=0 o 1)
  - O perceptrón con función sigmoide como activación
- La función de coste MSE no sería convexa en este caso.
- Queremos que lo sea para encontrar un mínimo.





Función de coste: perceptrón

- En perceptrón (regresión logística). Con y=0 o 1.
- Función de pérdida: $\mathrm{L}(\theta) = \begin{cases} -\log\left(f_{\theta}(x)\right), si\ y = 1 \\ -\log\left(1 - f_{\theta}(x)\right), si\ y = 0 \end{cases}$



Función de coste: perceptrón

Función de coste: perceptrón

- Función de pérdida: $\mathrm{L}(\theta) = \begin{cases} -\log\left(f_{\theta}(x)\right), si\ y = 1 \\ -\log\left(1 - f_{\theta}(x)\right), si\ y = 0 \end{cases}$
- Se puede reescribir como (¡recordad que *y* es 0 o 1!):

$$L(\theta) = -y \cdot \log\left(f_{\theta}(x)\right) - (1 - y) \cdot \log(1 - f_{\theta}(x))$$

- Así, la función de coste se puede escribir como:

$$J(\theta) = \frac{1}{m}\left(\sum_{i=1}^{m} y^{i}\log(f_{\theta}\left(x^{i}\right)) + \left(1 - y^{i}\right)\log(1 - f_{\theta}\left(x^{i}\right))\right)$$

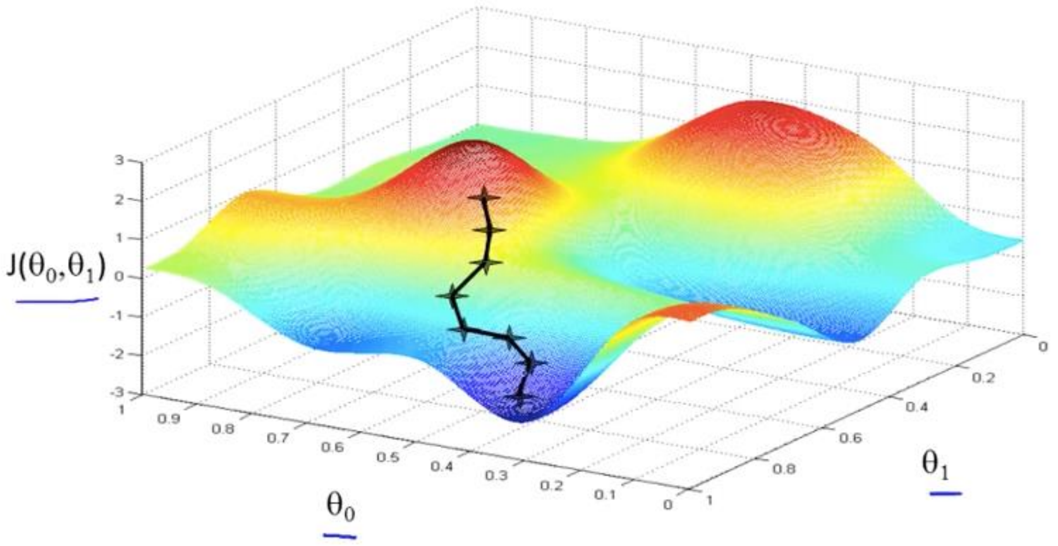
Función de coste: red neuronal

- La función de coste para una red neuronal para clasificación multiclase es una generalización de la del perceptrón
  - K clases
  - y_k^i es la salida esperada para la clase k para el ejemplo i
  - $f_\theta^k(x^i)$ es la salida para la clase k para el ejemplo i

$$J(\theta) = \frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log\left(f_\theta^k(x^i)\right) + \left(1 - y_k^i\right)\log(1 - f_\theta^k(x^i))\right)$$

Descenso por gradiente

- El objetivo es optimizar la función de coste para que valga lo mínimo posible, es decir $\min_{\theta} J(\theta)$



Descenso por gradiente

Descenso por gradiente

Descenso por gradiente

- Seguir el **gradiente** en negativo (la mayor pendiente)
- Es decir: calculamos el coste, su derivada, y actualizamos cada parámetro θ_j:

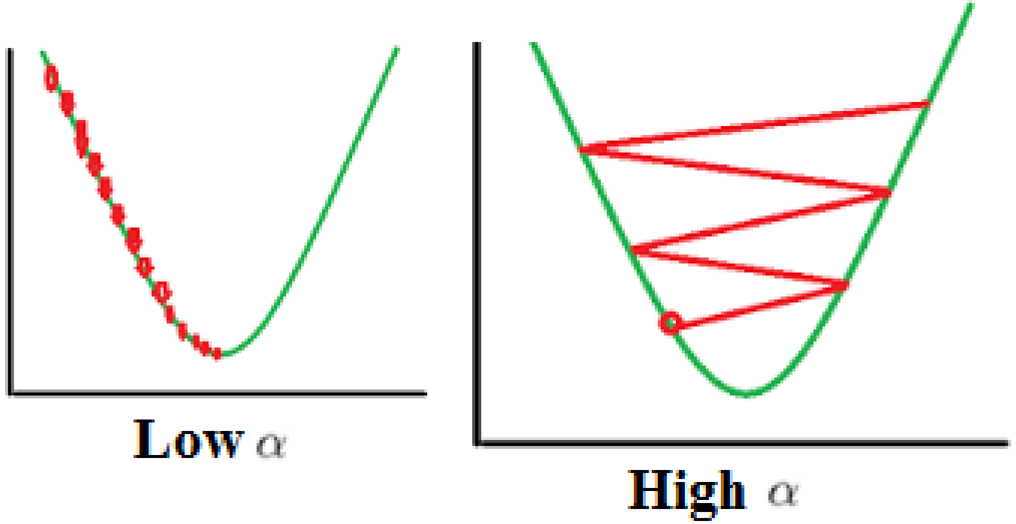
$$\theta_j = \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

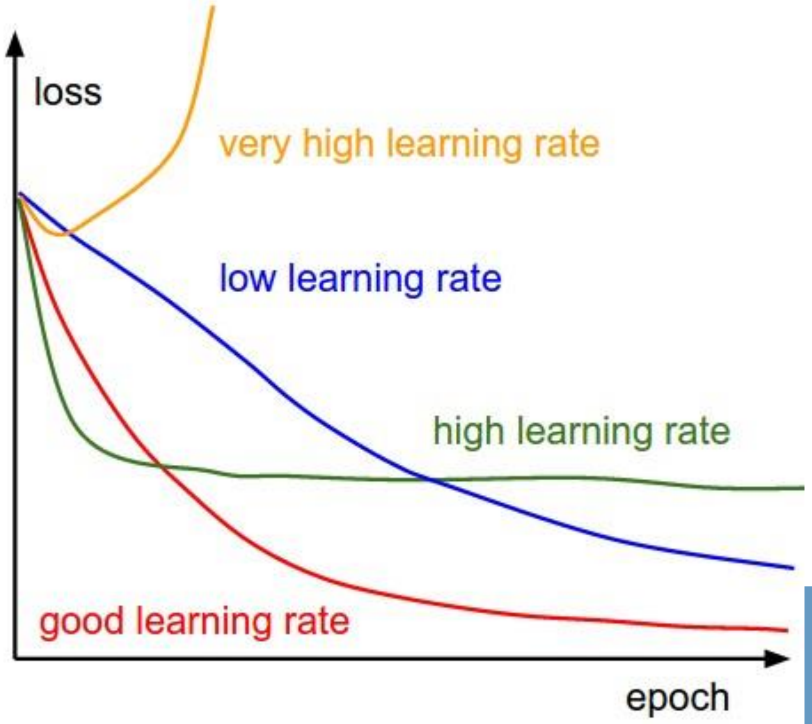
- La derivada de la función de coste es la misma tanto para regresión lineal como logística, así que la actualización quedaría como:

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} (f_\theta(x^i) - y^i)\, f'_\theta(x^i) x_j^i$$

Descenso por gradiente

- Cada iteración sobre los ejemplos del conjunto de entrenamiento se denomina **época (epoch)**
- α es un hiperparámetro: **factor de aprendizaje (learning rate)**
- ¿Qué valor es mejor?

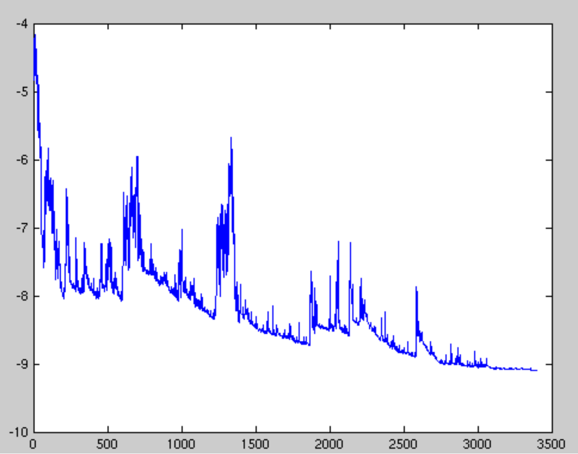




Descenso por gradiente: variantes

- **Descenso del gradiente en lote (batch)**
  - Aplicar descenso del gradiente en lote significa usar todos los ejemplos del conjunto de entrenamiento D.
  - **Ventajas**:
    - En cada iteración solo tenemos que calcular un gradiente.
    - Garantiza la convergencia al mínimo global (superficie del error es convexa) o a un mínimo local (si la superficie no es convexa).
  - **Inconvenientes**:
    - Si el conjunto de datos es muy grande no podremos cargarlo en memoria.
    - No es válido para casos en que los datos llegan en streaming (online learning)

Descenso por gradiente: variantes



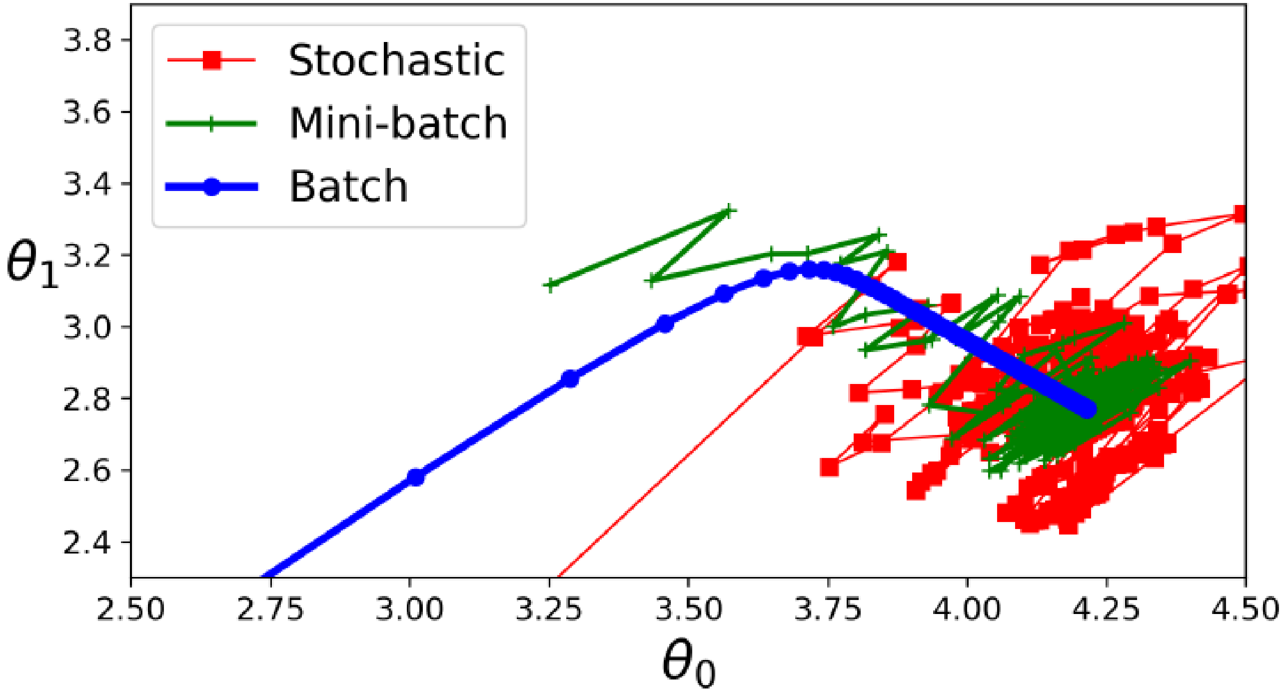
- **Descenso del gradiente estocástico (SGD)**
  - En cada iteración, se escoge un ejemplo de manera aleatoria (con o sin reemplazamiento).
  - Ventajas:
    - Ayuda al algoritmo a escaparse de mínimos locales
    - Tiene más probabilidad de alcanzar el mínimo global que el descenso por gradiente cuando superficie irregular.
  - Inconvenientes:
    - Puede realizar el calculo de gradientes redundantes al encontrar ejemplos muy similares.
    - Hay que realizar el cálculo de muchos más gradientes
    - Puede que nunca alcance un mínimo: variar el factor de aprendizaje, decrementando su valor en cada iteración.

Descenso por gradiente: variantes

- **Descenso del gradiente estocástico con minibatch**
  - Solución intermedia
  - Realiza la actualización tomando B muestras aleatorias del conjunto de entrenamiento.
  - Por un lado reducimos la varianza vista anteriormente y conseguimos una convergencia más estable.
  - Al ser un pequeño subconjunto de ejemplos, nos permite emplear hardware paralelo con memoria limitada (GPUs).
  - Tamaños comunes de mini-batch: 32, 50, 64, 128, 256.
  - P.ej.: Krizhevsky ILSVRC ConvNet usaba 128 ejemplos por batch.

Descenso por gradiente: variantes

- Batch vs SGD vs Minibatch

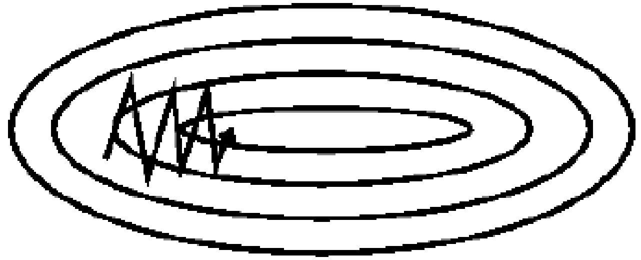


Descenso por gradiente: variantes

- **Momentum**
  - Ayuda a evitar oscilaciones agregando a la regla de actualización una fracción del gradiente usado en la ultima actualización

$$\mathrm{W}_j^{t+1} = \mathrm{W}_j^t - \alpha \frac{d}{d\mathrm{W}_j} J(W^t) + \boldsymbol{\beta \Delta W_j^t}$$

  - **Intuitivamente**: ganar más velocidad en más pendiente
    - Incrementa la actualización en aquellos parámetros cuyo gradiente apunta en la misma dirección y penaliza los que cambian.
  - Un valor común para es $\boldsymbol{\beta = 0,9}$ (también 0,5; 0,95; 0,99)



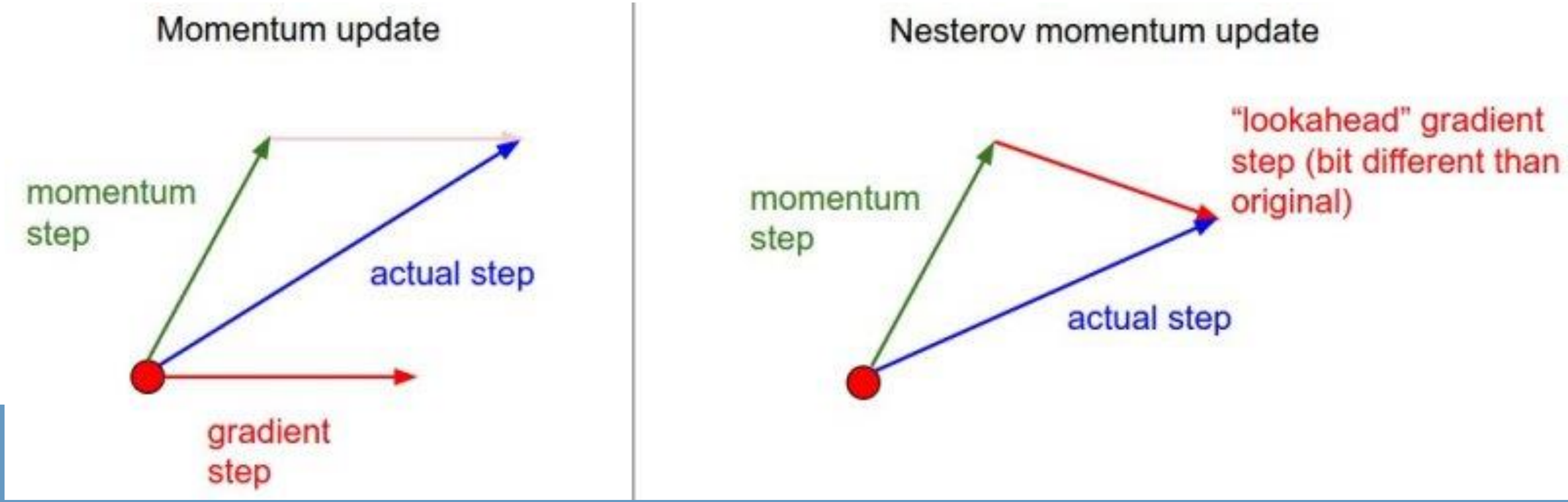
Descenso por gradiente: variantes

- **Aceleración de Nesterov (NAG)** (*o momentum Nesterov*)
  - Con la velocidad acumulada (momentum) podría pasarse del mínimo volviendo a subir por la otra cara de la pendiente: **Reducir la velocidad** cuando la **pendiente** cambie.
  - Predecir la siguiente posición para poder tomar decisiones de antemano.

$$\theta_j^{t+1} = \theta_j^t - \alpha \frac{d}{d\theta_j} J\left(\theta^t + \boldsymbol{\beta} \Delta\theta_{\boldsymbol{j}}^{\boldsymbol{t}}\right) + \boldsymbol{\beta} \Delta\theta_{\boldsymbol{j}}^{\boldsymbol{t}}$$

Descenso por gradiente: variantes

- **Aceleración de Nesterov (NAG)** (*o momentum Nesterov*)
  - Con la velocidad acumulada (momentum) podría pasarse del mínimo volviendo a subir por la otra cara de la pendiente: **Reducir la velocidad** cuando la **pendiente** cambie.
  - Predecir la siguiente posición para poder tomar decisiones de antemano.



Descenso por gradiente: variantes

- **Adagrad** [Duchi et al., 2011]
  - Adagrad adapta **un learning rate para cada parámetro**:
    - Aplicar un mayor learning rate a aquellos atributos más dispersos (sparse), o menos frecuentes. P.ej. en NLP, las palabras menos frecuentes son más informativas.

$$\mathrm{W}_j^{t+1} = \mathrm{W}_j^{t} - \frac{\alpha}{\sqrt{\boldsymbol{E}[\boldsymbol{g}^2]_t + \boldsymbol{\varepsilon}}} \frac{d}{dW_j} J(W^t)$$

  - $E[g^2]_t$ es un vector que tiene mismas dimensiones que parámetros del modelo
    - se actualiza acumulando el gradiente al cuadrado $E[g^2]_t = E[g^2]_{t-1} + g_t^2$
  - ***Pro***: permite un learning rate constante (0,01).
  - ***Contra***: la acumulación de los gradientes puede hacerse infinitesimal y desvanecerse.

Descenso por gradiente: variantes

- **Adadelta** [Zeiler 2012]
  - Trata de solventar la agresiva reducción del learning rate vista en adagrad.
    - Limitando la suma de los gradientes a una ventana temporal.
  - Para evitar que los valores acumulados desvanezcan, introduce un suavizado:

$$\boldsymbol{E[g^2]_t = \rho E[g^2]_t + (1-\rho) g_t^2}$$

  - Si definimos $RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon}$, la regla de actualización es:

$$\mathrm{W}_j^{t+1} = \mathrm{W}_j^{t} - \frac{\boldsymbol{RMS[\Delta W]_{t-1}}}{\boldsymbol{RMS[g]_t}} \frac{d}{dW_j} J(W^t)$$

  - **¡no es necesario establecer un learning rate**! Y normalmente**,** $\boldsymbol{\rho = 0,9}$

Descenso por gradiente: variantes

- **RMSprop** [Tieleman and Hilton, 2012]
  - Es un algoritmo no publicado.
  - Desarrollado independientemente de adadelta y diseñado para resolver el mismo problema de adagrad.
  - Su regla de actualización es (con $\alpha = 0{,}001$):

$$\mathrm{W}_j^{t+1} = \mathrm{W}_j^{t} - \frac{\alpha}{\boldsymbol{RMS[g]_t}} \frac{d}{dW_j} J(W^t)$$

Descenso por gradiente: variantes

- **Adam** [Kingma and Ba, 2014]
  - Una combinación de las ventajas de AdaGrad y RMSProp:
    - **AdaGrad**: mejora el rendimiento con parámetros dispersos.
    - **RMSProp**: learning rate adaptado a los gradientes recientes. Se adapta bien en problemas on-line y no-estacionarios (por ejemplo con ruido)
  - Nuevos términos (unos valores buenos son $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  - Su regla de actualización es (un buen valor es $\alpha = 0{,}002$):

$$\mathrm{W}_j^{t+1} = \mathrm{W}_j^t - \frac{\alpha}{\sqrt{\boldsymbol{v_t}} + \boldsymbol{\varepsilon}} \boldsymbol{m_t}$$

Descenso por gradiente: variantes

- **Adam** [Kingma and Ba, 2014]
  - Una combinación de las ventajas de AdaGrad y RMSProp:
    - **AdaGrad**: mejora el rendimiento con parámetros dispersos.
    - **RMSProp**: learning rate adaptado a los gradientes recientes. Se adapta bien en problemas on-line y no-estacionarios (por ejemplo con ruido)
  - Nuevos términos (unos valores buenos son $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

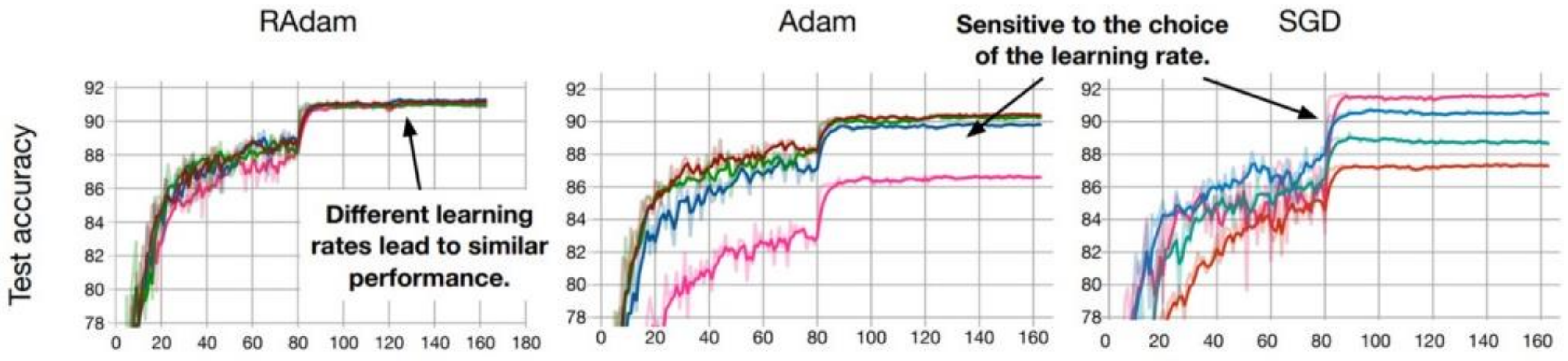
Momentum!! **RMSprop!!**

  - Su regla de actualización es (un buen valor es $\alpha = 0{,}002$):

$$W_j^{t+1} = W_j^t - \frac{\alpha}{\sqrt{\boldsymbol{v_t}} + \boldsymbol{\varepsilon}} \boldsymbol{m_t}$$

Descenso por gradiente: variantes

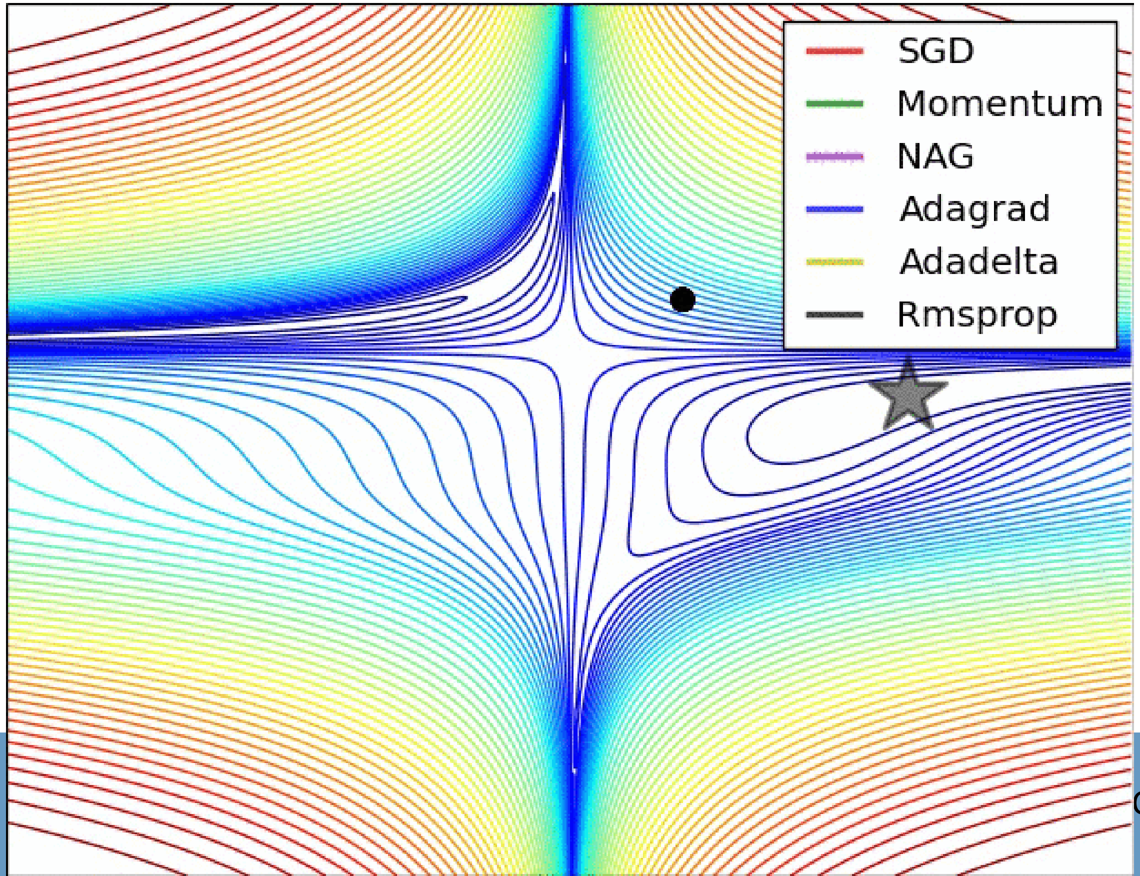
- **Radam** (Rectified Adam): [Liu, Jiang et al 2019]
  - Introducido en Agosto de 2019.
  - Variación del optimizador Adam
  - Provee un ajuste dinámico y automático al learning rate adaptativo.
  - Basado en los efectos de la varianza y momentum durante el entrenamiento.



Descenso por gradiente: variantes

- Comparación variantes

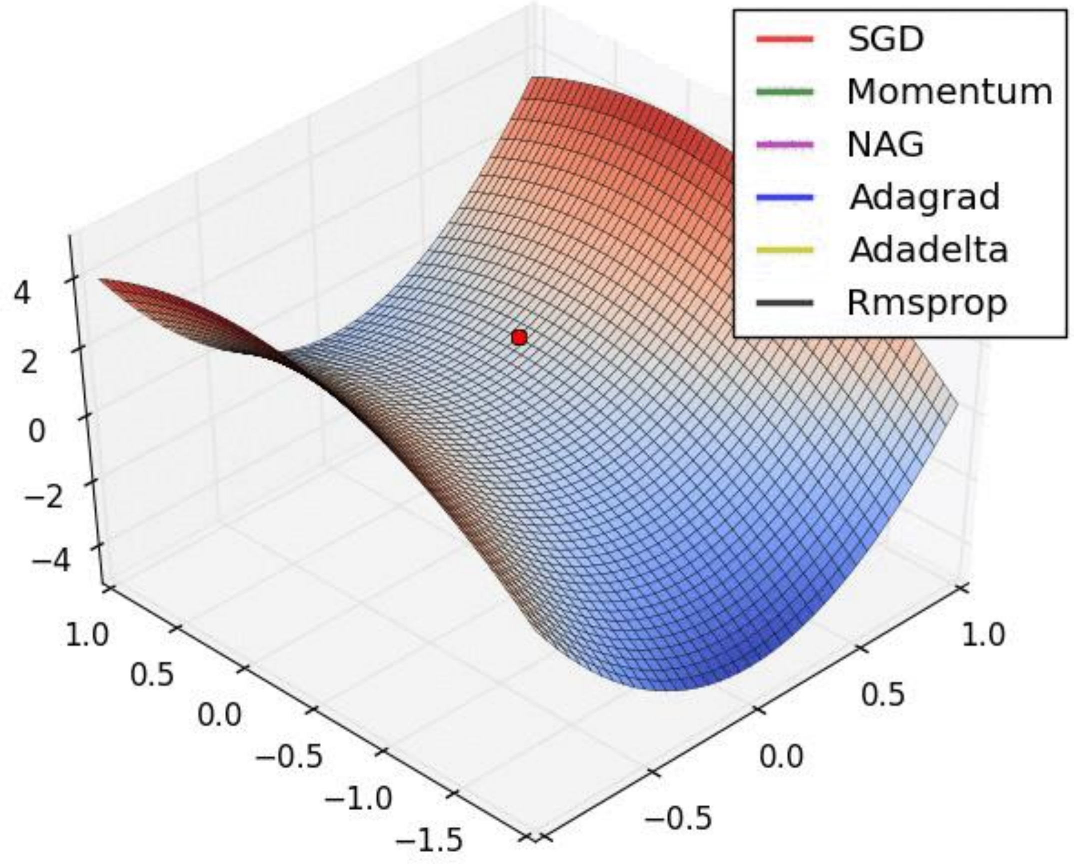
http://cs231n.github.io/neural-networks-3/



Descenso por gradiente: variantes

- Comparación variantes

http://cs231n.github.io/neural-networks-3/

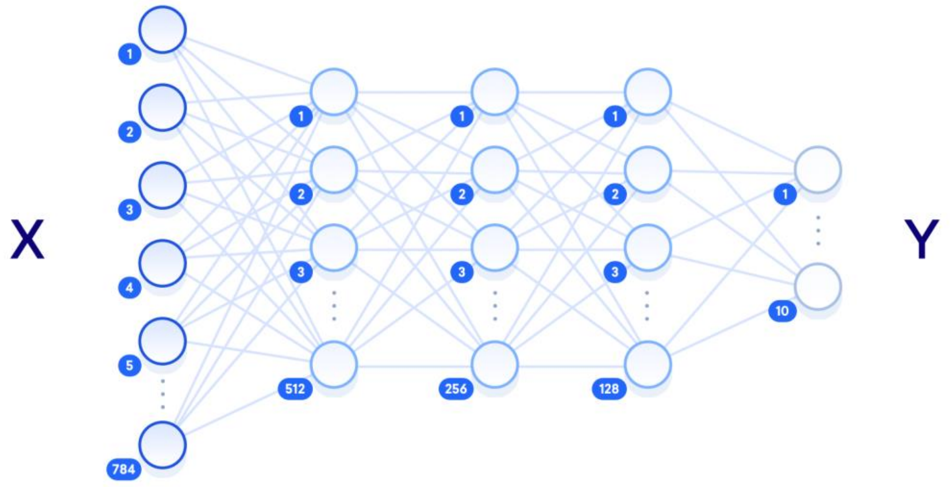


Descenso por gradiente: pautas prácticas

- De "An overview of gradient descent optimization algorithms":
  - *RMSProp, Adadelta y Adam son algoritmos muy parecidos que funcionan bien en circunstancias similares. [...] la corrección del sesgo llevada a cabo por Adam ayuda a mejorar el rendimiento de RMSProp al final de la optimización cuando los gradientes son dispersos. Por tanto* ***Adam*** *podría ser la mejor opción.*
- De Andrej Karpathy, en el curso Stanford CS231n:
  - *En la práctica,* ***Adam*** *es recomendado como el optimizador por defecto, a menudo funciona algo mejor que RMSProp. Sin embargo, en recomendable intentar SGD+Nesterov como una alternativa.*

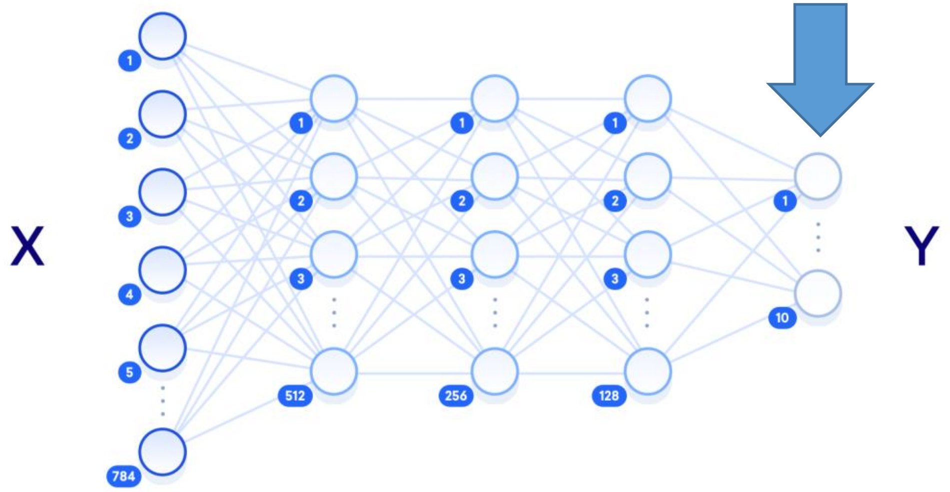
Descenso por gradiente

- Las actualizaciones se pueden aplicar directamente para actualizar pesos en modelos como regresión lineal y logística (perceptrón con función de activación sigmoide).
- ¿Cómo proceder con red neuronal multicapa?



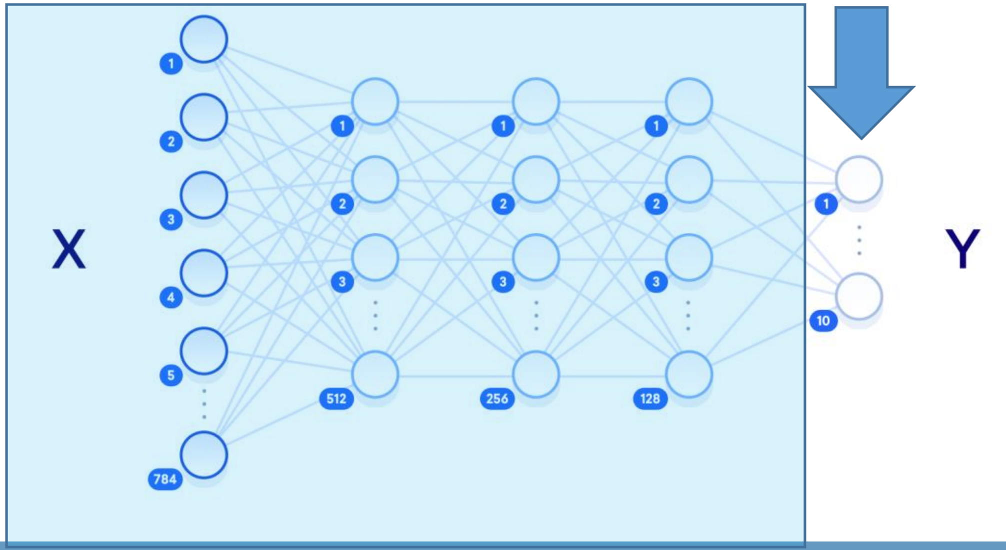
Descenso por gradiente

- Las actualizaciones se pueden aplicar directamente para actualizar pesos en modelos como regresión lineal y logística (perceptrón con función de activación sigmoide).
- ¿Cómo proceder con red neuronal multicapa?



Descenso por gradiente

- Las actualizaciones se pueden aplicar directamente para actualizar pesos en modelos como regresión lineal y logística (perceptrón con función de activación sigmoide).
- ¿Cómo proceder con red neuronal multicapa?



Recapitulación

- **Función de coste/pérdida**: cómo de buena es la red
- Una **función de coste** para regresión lineal, logística y multi-clase
- **Descenso por gradiente**: buscamos un mínimo óptimo para la función de coste. Para ello calculamos gradientes (derivadas) por dimensión y actualizamos los pesos.
- **Variantes**: SGD, Momentum, NAG, Adagrad, AdaDelta, RMSprop, Adam, RAdam