Regresión Logística

INF-396

Prof: Juan G. Pavez S.

Regresión Logística

- Se llama regresión pero es un modelo de **clasificación**.
- Es decir, una función que mapea un vector de entrada $x \in X$ a una etiqueta de salida $y \in \{1,...,C\}$
- Tiene cierta similitud con la **regresión lineal**.

Regresión Logística

- Generalizando la regresión lineal para clasificación

$$p(y|x,w) = Ber(y|\mu(x))$$

- Con $\mu(x) = E[y|x] = p(y=1|x)$

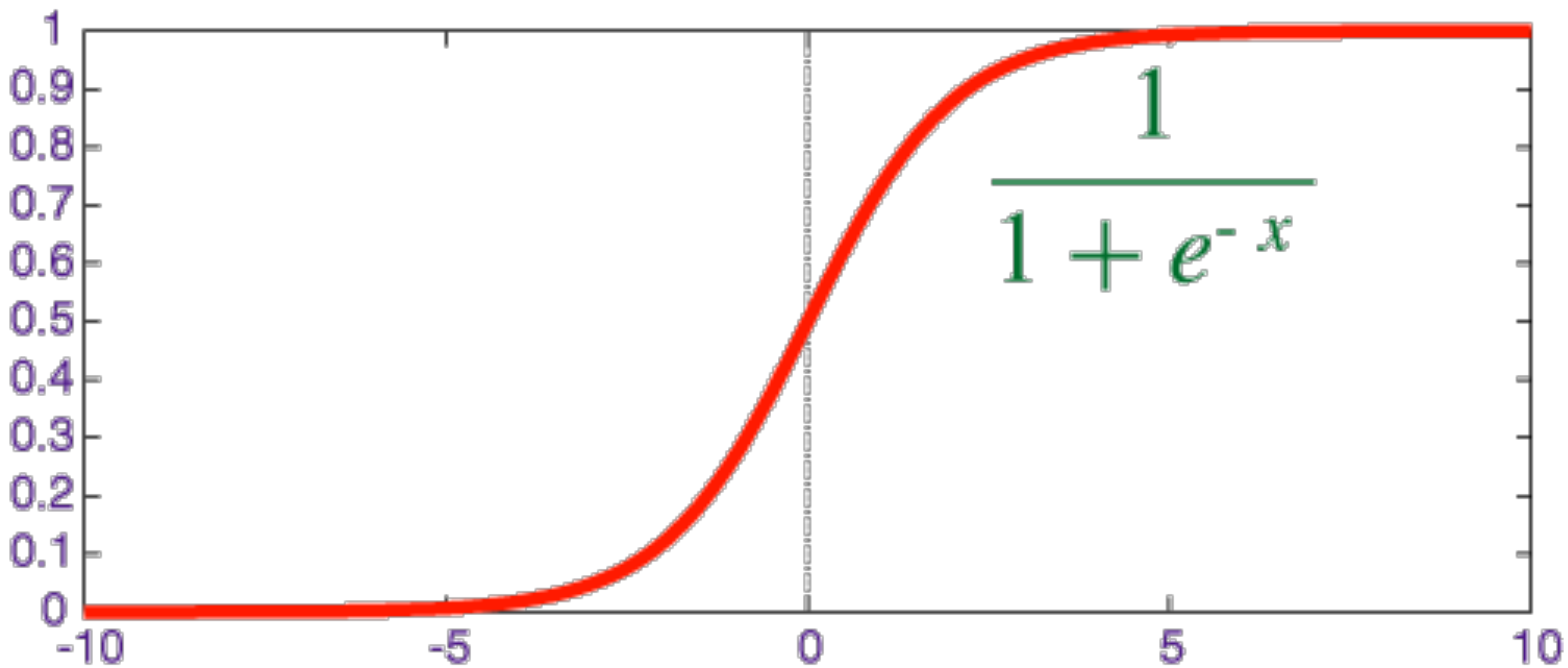
- Pero ahora

$$\mu(x) = sigm(w^T x)$$

- Donde sigm es la función sigmoidal y produce $0 \leq \mu(x) \leq 1$ así que puede ser interpretada como una probabilidad..

Regresión Logística

$$\mu(x) = sigm(w^T x) = \frac{1}{1 + e^{-w^T x}} = \frac{e^{w^T x}}{e^{w^T x} + 1}$$



Regresión Logística

- Generalizando la regresión lineal para clasificación

$$p(y \mid x, w) = Ber(y \mid sigm(w^T x))$$

- Y para obtener una regla de selección de etiqueta

$$\hat{y}(x) = 1 \iff p(y = 1 \mid x, w) > 0.5$$

- La verosimilitud negativa es definida como:

$$NLL(w) = -\sum_{i=1}^{N} \log[\mu_i^{I\{y_i=1\}}(1-\mu_i)^{I\{y_i=0\}}]$$

Regresión Logística

$$NLL(w) = -\sum_{i=1}^{N} \log[\mu_i^{I\{y_i=1\}}(1-\mu_i)^{I\{y_i=0\}}]$$

- Lo que es equivalente a

$$NLL(w) = -\sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$$ (Cross entropy error)

- En este caso el mínimo no puede ser derivado en forma analítica. Debido a eso, necesitamos usar un procedimiento de **optimización** para encontrar el mínimo.

Regresión Logística

$$NLL(w) = -\sum_{i=1}^{N} [y_i \log \mu_i + (1 - y_i)\log(1 - \mu_i)]$$ (Cross entropy error)

- Para encontrar el mínimo necesitamos el gradiente y el hessiano:

$$g(w) = f'(w) = \sum_{i=1}^{N} (\mu_i - y_i)x_i = x^T(\mu - y)$$

$$\begin{aligned} H(w) &= g'(w) = (\nabla_w \mu_i)x_i^T \\ &= X^T S X \end{aligned}$$

- Con $S = diag(\mu_i(1 - \mu_i))$

Regresión Logística

- Demostración para g:

  -Sea $\mu = \dfrac{1}{1 + e^{-z}}$ y $z = w^T x$

  - $J(w) = -\left(y \log \mu + (1 - y)\log(1 - \mu)\right)$

  - $\dfrac{\partial J}{\partial w} =$

Regresión Logística

- Demostración para g:

  -Sea $\mu = \dfrac{1}{1+e^{-z}}$ y $z = w^T x$

  - $J_i(w) = -\left(y \log \mu + (1-y)\log(1-\mu)\right)$

  - $\dfrac{\partial J}{w} = \dfrac{\partial J}{\partial u}\dfrac{\partial u}{\partial z}\dfrac{\partial z}{\partial w}$

$$\frac{\partial u}{\partial z} = (1+e^{-z})^{-2}e^{-z} = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} = u(1-u)$$

Regresión Logística

- Demostración para g:

-Sea $\mu = \dfrac{1}{1+e^{-z}}$ y $z = w^T x$

- $J_i(w) = -(y \log \mu + (1-y)\log(1-\mu))$

- $\dfrac{\partial J}{w} = \dfrac{\partial J}{\partial u}\dfrac{\partial u}{\partial z}\dfrac{\partial z}{\partial w}$

$$\frac{\partial u}{\partial z} = (1+e^{-z})^{-2}e^{-z} = u(1-u) \qquad \frac{\partial z}{\partial w} = x$$

$$\frac{\partial J}{\partial w} = \frac{u-y}{u(1-u)}u(1-u)x = x(u-y)$$

Regresión Logística

- Para realizar clasificación multiclases, se puede usar la regresión logística multinomial (también conocido como **clasificador de máxima entropía**).

$$p(y = c \,|\, x, W) = \frac{exp(W_c^T x)}{\sum_{c'=1}^{C} exp(W_{c'}^T x)}$$

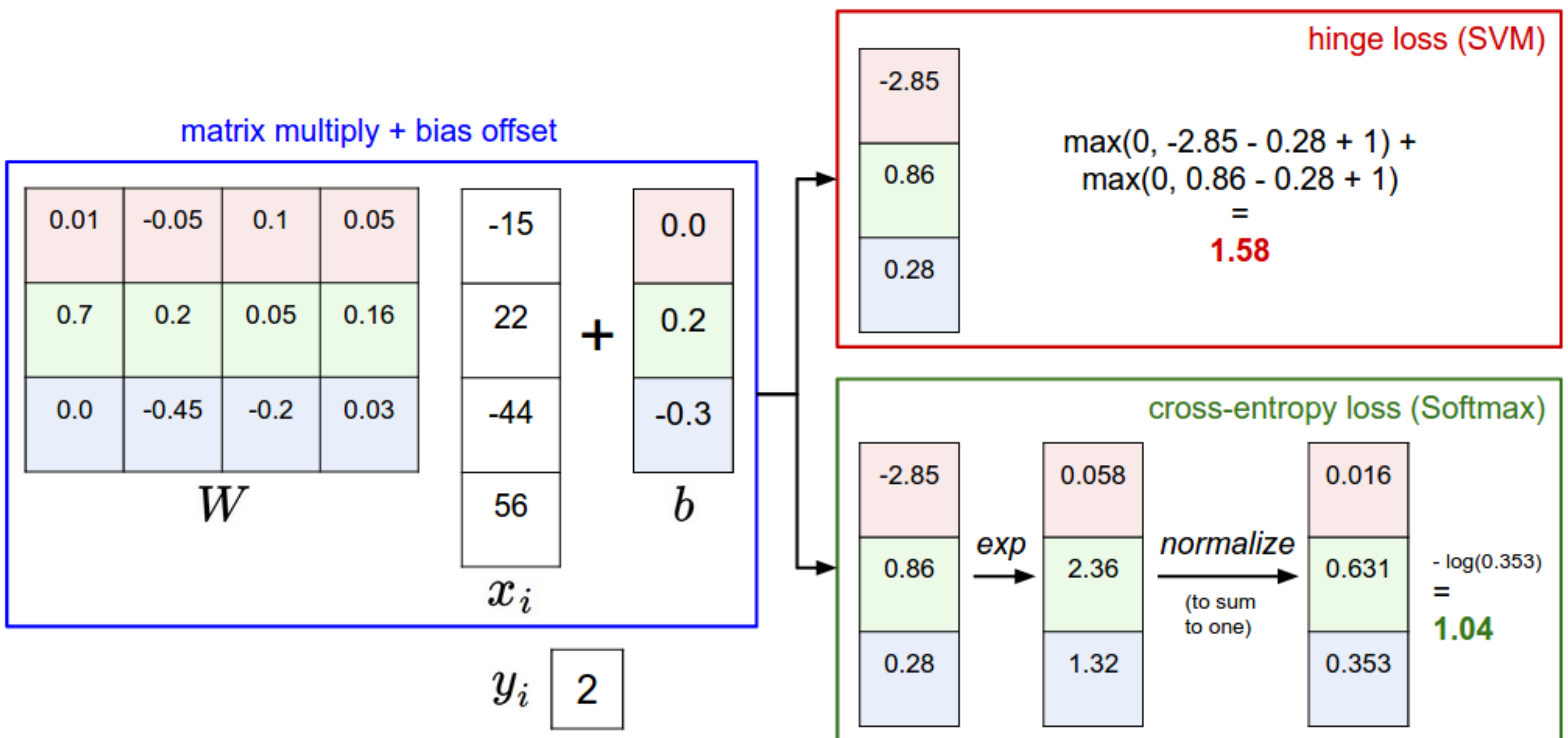
- sea

$$\mu_{ic} = p(y_i = c \,|\, x_i, W) = S(\eta)_c$$

Regresión Logística

- Para realizar clasificación multiclases, se puede usar la regresión logística multinomial (también conocido como **clasificador de máxima entropía**).

$$p(y = c \mid x, W) = \frac{exp(W_c^T x)}{\sum_{c'=1}^{C} exp(W_{c'}^T x)}$$

- 

Regresión Logística

- La función de **softmax**:

$$S(\eta)_c = \frac{e^{\eta_c}}{\sum_{c'=1}^{C} e^{\eta_{c'}}}$$

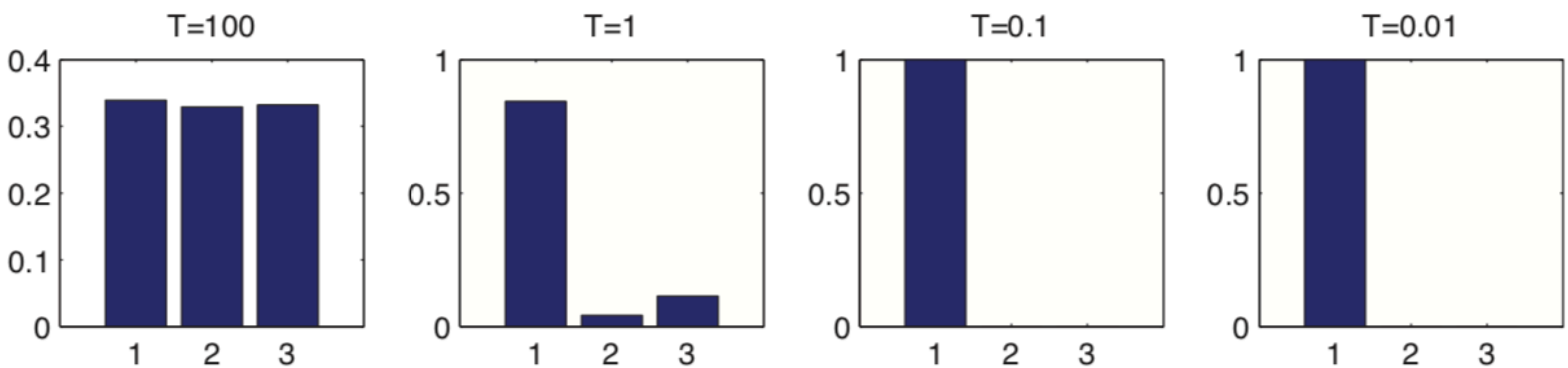
- Que actúa como una función de **máximo** suavizada.

- Uno puede agregar un parámetro T de temperatura para regular la 'suavidad' de la función.

$$S(\eta/T)_c$$

Regresión Logística

- $$S(\eta/T)_c = \frac{e^{\eta_c/T}}{\sum_{c'=1}^{C} e^{\eta_{c'}/T}} \qquad \eta = (3,0,1)$$



Regresión Logística

- Sea $\eta_i = w_T x_i$ un vector Cx1 y $y_{ic} = I(y_i = c)$
  La log-verosimilitud es

$$l(w) = log \prod_{i=1}^{N} \prod_{c=1}^{C} \mu_{ic}^{y_{ic}} = \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

$$= \sum_{i=1}^{N} \left[\left(\sum_{c=1}^{C} y_{ic} W_c^T x_i \right) - \log\left(\sum_{c'=1}^{C} exp(w_{c'}^T x_i) \right) \right]$$

- Y el gradiente es $g(w) = \sum_{i=1}^{N} (\mu_i - y_i) \otimes x_i$
  donde:

$$y_i = [I(y_i = 1), \ldots, I(y_i = c)]$$

$$\mu_i(w) = [p(y_i = 1 \mid x_i, W), \ldots, p(y_i = c \mid x_i, W)]$$

Regresión Logística

- Sea $\eta_i = w_T x_i$ un vector Cx1 y $y_{ic} = I(y_i = c)$
  La log-verosimilitud es

$$l(w) = log \prod_{i=1}^{N} \prod_{c=1}^{C} \mu_{ic}^{y_{ic}} = \sum_{i=1}^{C} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

$$= \sum_{i=1}^{N} \left[\left(\sum_{c=1}^{C} y_{ic} W_c^T x_i \right) - \log\left(\sum_{c'=1}^{C} exp(w_{c'}^T x_i) \right) \right]$$

- Y el gradiente es $g(w) = \sum_{i=1}^{N} (\mu_i - y_i) \otimes x_i$
  donde:

$$y_i = [I(y_i = 1), \ldots, I(y_i = c)]$$

$$\mu_i(w) = [p(y_i = 1 \,|\, x_i, W), \ldots, p(y_i = c \,|\, x_i, W)]$$

Dejaremos la derivación para tarea

Algoritmos de optimización

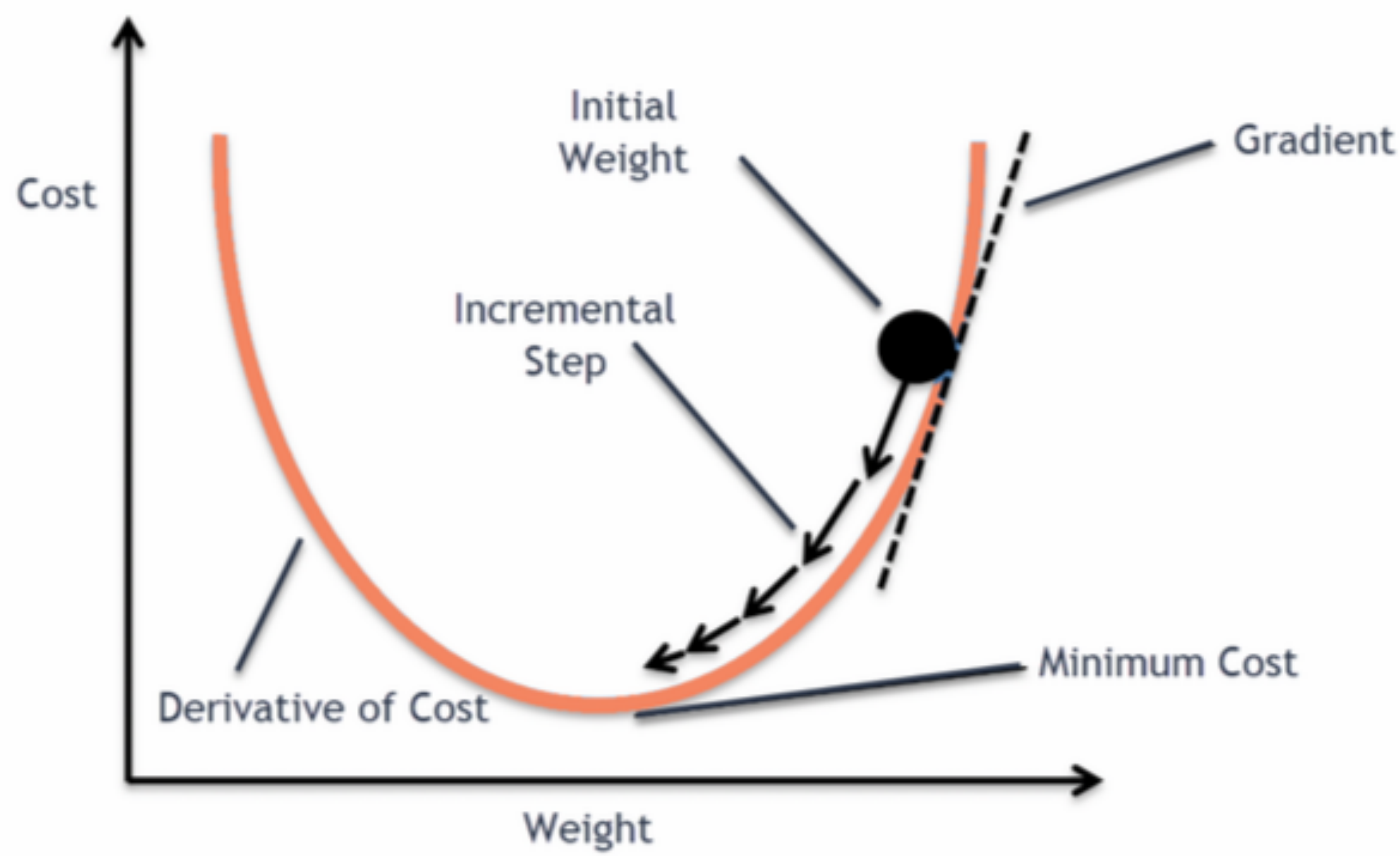
- **Algoritmos de optimización: Gradiente descendiente**
  Debido a que no podemos obtener el mínimo de manera analítica, utilizaremos algoritmos de optimización.
  La idea es moverse iterativamente hacia el mínimo de la función de perdida.

  Para hacer eso, el procedimiento de gradiente descendiente usa información local para moverse en la dirección opuesta al gradiente

$$\theta_{k+1} = \theta_k - \eta_k g_k$$

  Con $g_k(\theta) = \frac{d}{d\theta} f(x)$ y η_k es el tamaño del paso o **learning rate** que controla el tamaño de los pasos que damos.

Algoritmos de optimización

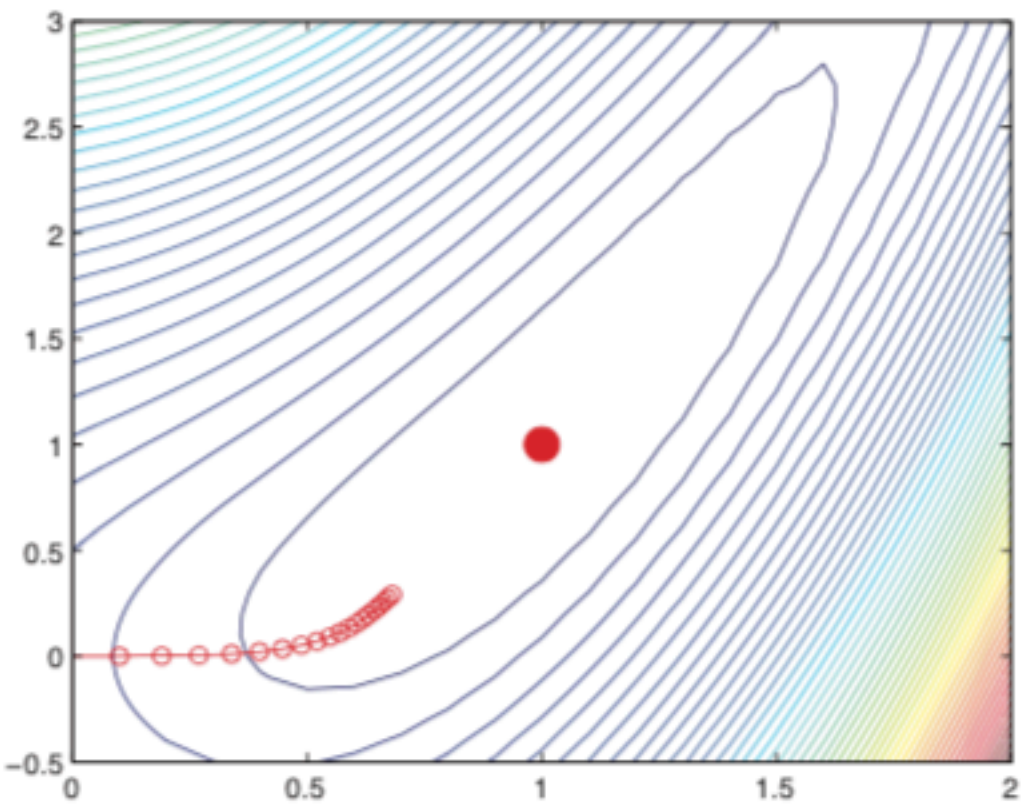


$$g_k(\theta) = \frac{d}{d\theta} f(x)$$

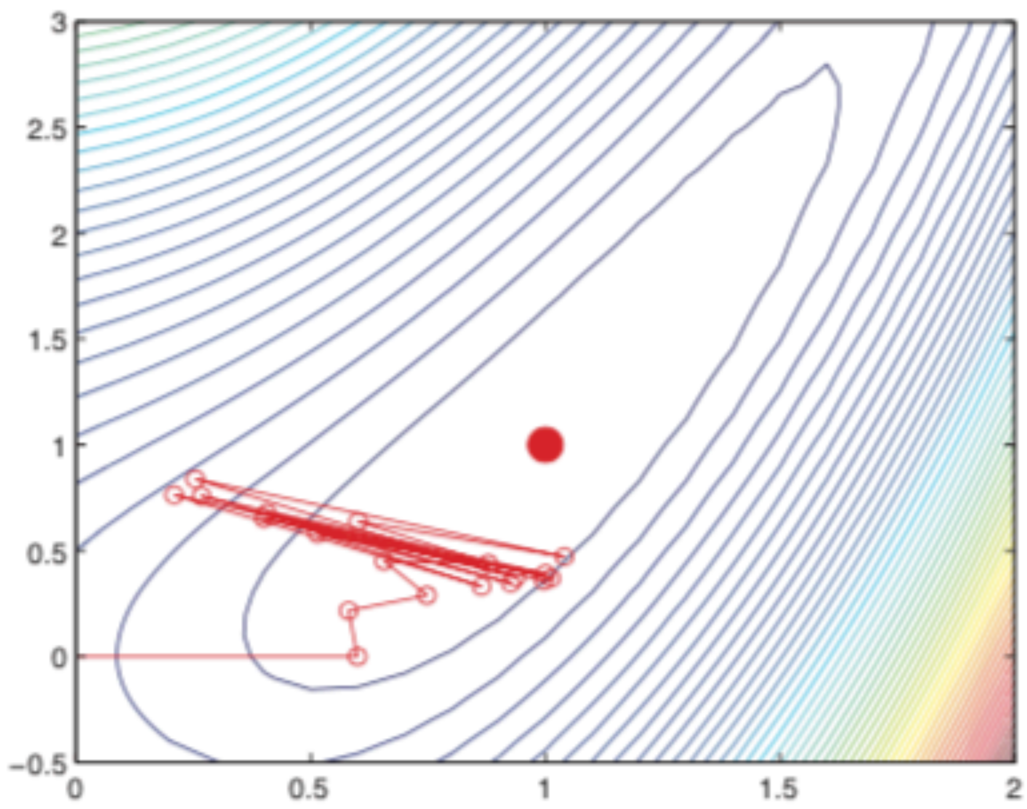
$$\theta_{k+1} = \theta_k - \eta_k g_k$$

Optimization Algorithms

- **Algoritmos de optimización: Gradiente descendiente**



$\mu = 0.1$



$\mu = 0.6$

https://www.benfrederickson.com/numerical-optimization/

Optimization Algorithms

- **Algoritmos de optimización: Gradiente descendiente**

  Búsqueda lineal: En cada paso busca el η_k que minimice.

$$\phi(\eta) = f(\theta_k + \eta_k d_k)$$

So

$$\eta_k = argmin_{n>0}\phi(\eta)$$

Entonces $\phi'(\eta) = 0$ es una condición necesaria donde

$$\phi'(\eta) = d^T f'(\theta + \eta d)$$

Optimization Algorithms

- **Optimization algorithms: Gradient Descent**

  Line Search:
  Entonces $\phi'(\eta) = 0$ es una condición necesaria.

- Esto pasa si:

  - $f'(\theta + \eta d) = 0$ es un punto estacionario.

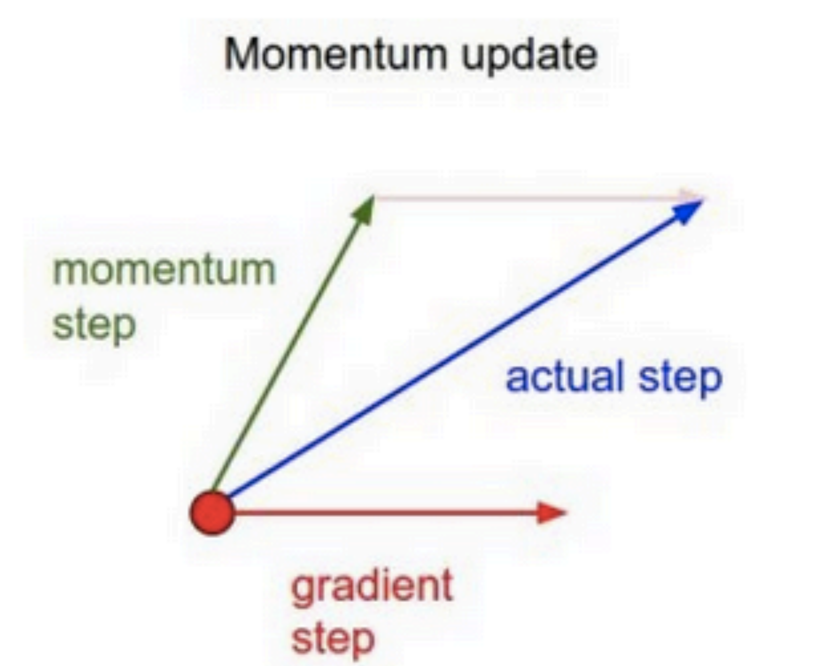
  - $f'(\theta + \eta d) \perp d$ la búsqueda se detiene donde el gradiente es perpendicular a la dirección de la búsqueda: **zig-zag behavior**.

Optimization Algorithms

- **Algoritmo de optimización: Momentum**
  Se puede ver cómo una **bola pesada** bajando por una colina.
  Da ‘memoria’ al gradiente descendiente.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

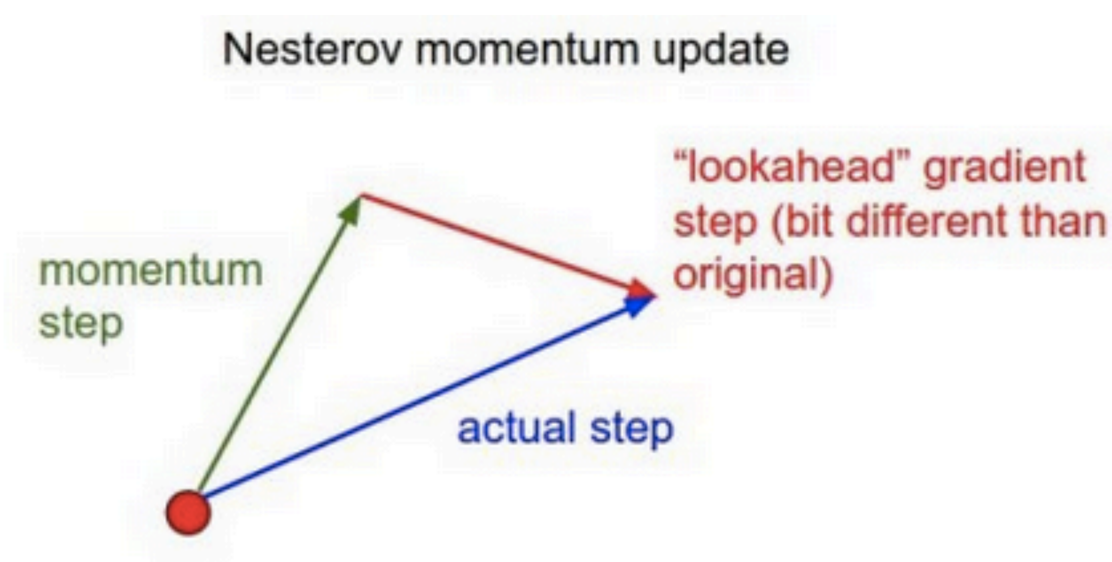


Optimization Algorithms

- **Algoritmo de optimización: Momentum Nesterov**
  Ahora el gradiente se calcula en donde se daría el nuevo paso.
  Corrige luego de hacer el error.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta_{t+1} = \theta_t - v_t$$



Optimization Algorithms

- **Algoritmo de optimización: Método de Newton**
  Considera la curvatura del espacio

$$\theta_{k+1} = \theta_k + \eta_k H_k^{-1} g_k$$

- Puede ser derivado de la aproximación de Taylor segundo orden:

$$f_{quad}(\theta) = f_k + g_k(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H_k(\theta - \theta_k)$$

Optimization Algorithms

- **Algoritmo de optimización método de newtown**

$$f_{quad}(\theta) = f_k + g_k(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H_k(\theta - \theta_k)$$
$$= \theta^T A\theta + b^T\theta + c$$

- Donde

$$A = \frac{1}{2}H_k, b = g_k - H_k\theta_k, c = f_k - g_k^T\theta_k + \frac{1}{2}\theta_k^T H_k\theta_k$$

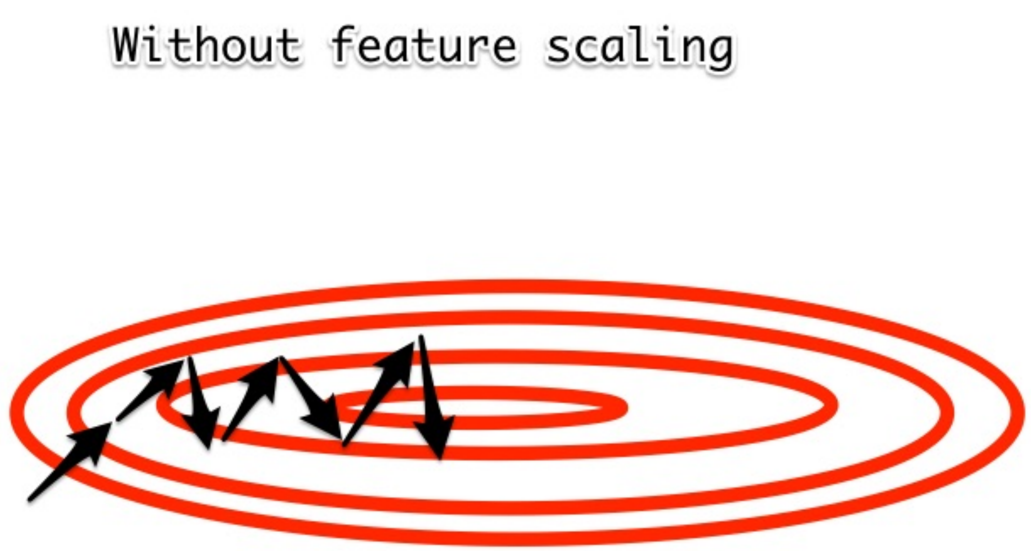
- Y el mínimo está en

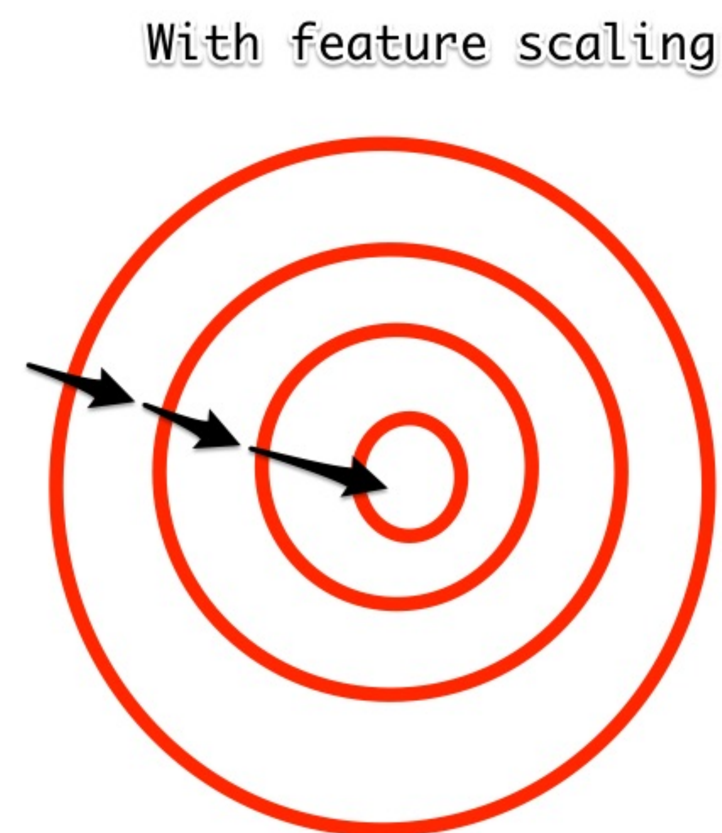
$$\theta = \frac{1}{2}A^{-1}b = \theta_k - H_k^{-1}g_k$$

Optimization Algorithms

- **Algoritmos de optimización: Escalamiento de features**

- Considerar estas dos features para el problema de predición de precios de casas :
  x1 = Tamaño (0 - 2000 metros cuadrados)
  x2 = Número de camas (1-5)





Optimization Algorithms

- **Algoritmos de optimización: Escalamiento de features**
- **MinMax Scaling**:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- xmin y xmax pueden ser conocidos de antes o calculados en el dataset

Optimization Algorithms

- **Optimization algorithms: Feature scaling**

- **Standarization**:

$$x' = \frac{x - \bar{x}}{\sigma}$$

- Convierte los datos en media zero y varianza unitaria.

- La desviación estándar y la media pueden ser obtenidos desde el conjunto de datos. Notar que estos valores deben ser guardados para uso posterior.

Optimization Algorithms

- **Optimization algorithms: Feature scaling**
- **Standarization**:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Recuerda escalar tus características cuando sea necesario!

- Convierte los datos en media zero y varianza unitaria.
- La desviación estándar y la media pueden ser obtenidos desde el conjunto de datos. Notar que estos valores deben ser guardados para uso posterior.

Optimization Algorithms

- **Considerar la sigmoidal - error cross entropy**

$$g(w) = \sum_{i=1}^{N} (\mu_i - y_i) x_i$$

- **Gradiente descendente:**

$$\theta_{k+1} = \theta_k - \eta \sum_{i=1}^{N} (\mu_i - y_i) x_i$$

Optimization Algorithms

- **Considerar la sigmoidal - error cross entropy**

$$g(w) = \sum_{i=1}^{N} (\mu_i - y_i)x_i$$

- **Gradiente descendente:**

$$\theta_{k+1} = \theta_k - \eta \sum_{i=1}^{N} (\mu_i - y_i)x_i$$

- **Gradiente descendente online (Stochastic Gradient Descent):**

$$\theta_{k+1} = \theta_k - \eta(\mu_k - y_k)x_k$$

Optimization Algorithms

- **Considerar la sigmoidal - error cross entropy**

$$g(w) = \sum_{i=1}^{N} (\mu_i - y_i)x_i$$

- **Gradiente descendiente en Mini-batch:**

$$\theta_{k+1} = \theta_k - \eta \sum_{i \in batch} (\mu_i - y_i)x_i$$

- Donde el mini batch es una muestra aleatoria de tamaño B del conjunto de datos (normalmente 32, 64, 128, or 256).

Optimization Algorithms

[3,4,2,3] [2,4,6,2] [1,4,2,3] [2,1,4,0] -> epoch

- **Gradiente descendente en mini-batch**

$$\theta_{k+1} = \theta_k - \eta \sum_{i \in batch} (\mu_i - y_i) x_i$$

- Podemos simular un stream de datos muestreando sin reemplazo del conjunto de datos randomizado.

- Se hacen varias **epochs** de entrenamiento en el conjunto de datos.

- Muchas veces el gradiente se puede estimar muy bien de una muestra de datos -> **Mini-batch gradient descent es considerablemente más eficiente**(considerar un conjunto de datos duplicados).

- El **ruido añadido por el muestreo** puede ayudar a escapar mínimos locales -> Actúa como **regularizador**.

Optimization Algorithms

- **Learning rate decay:**
  Una buena idea es comenzar con un learning rate grande y reducirlo lentamente a medida que el entrenamiento avanza.

- **Step decay**: Reduce por un factor definido cada ciertas epochs. Por ejemplo multiplica por 0.5(0.1) cada 5(20)epoch. También uno puede reducirlo cuando la performance no mejora

- **Exponential decay:** $\alpha = \alpha_0 e^{-k \times epoch}$

  Donde α_0 y k son parámetros

- **1/epochs decay:** $\alpha = \dfrac{1}{1 + k \times epoch}\alpha_0$

Optimization Algorithms

- **Tamaño de paso por parámetro**
  Algunas features tienen diferentes curvaturas de la superficie de error.
  Algunas feature pueden ser muy infrecuentes.

- **Adagrad:** Adapta el learning rate separado para cada característica.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\tau_0 + \sqrt{s_i(t)}} g_{t,i}$$

$$s_i(t) = \sum_{j=1}^{t} g_{ji}^2 \qquad s_i(t) = s_i(t-1) + g_{ti}^2$$

Optimization Algorithms

- **Adadelta:** Con adagrad, la suma de cuadrados del gradiente hace que las actualizaciones decaigan monotónicamente Para resolver eso se propone usar una estimación del segundo momento mediante medias móviles.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\tau_0 + \sqrt{E(g_i^2)}} g_{t,i}$$

Donde

$$E[g_i^2]_t = \gamma E[g_i^2]_{t-1} + (1-\gamma) g_{i,t}^2$$

Optimization Algorithms

- **Adam:** También mantiene una estimación por decaimiento exponencial de los gradiente y corrige las estimaciones sesgadas

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\tau_0 + \sqrt{\hat{v}_t}}\hat{m}_t$$

Donde

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_{i,t}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_{i,t}^2$$

- Y dado que son sesgadas hacia los valores iniciales, asumiendo valores iniciales de 0 la corrección es

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Optimization Algorithms

- **Adam:** También mantiene una estimación por decaimiento exponencial de los gradiente y corrige las estimaciones sesgadas

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\tau_0 + \sqrt{\hat{v}_t}}\hat{m}_t$$

Este es el recomendado (aunque cambia rápido)

Donde

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_{i,t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_{i,t}^2$$

- Y dado que son sesgadas hacia los valores iniciales, asumiendo valores iniciales de 0 la corrección es

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$