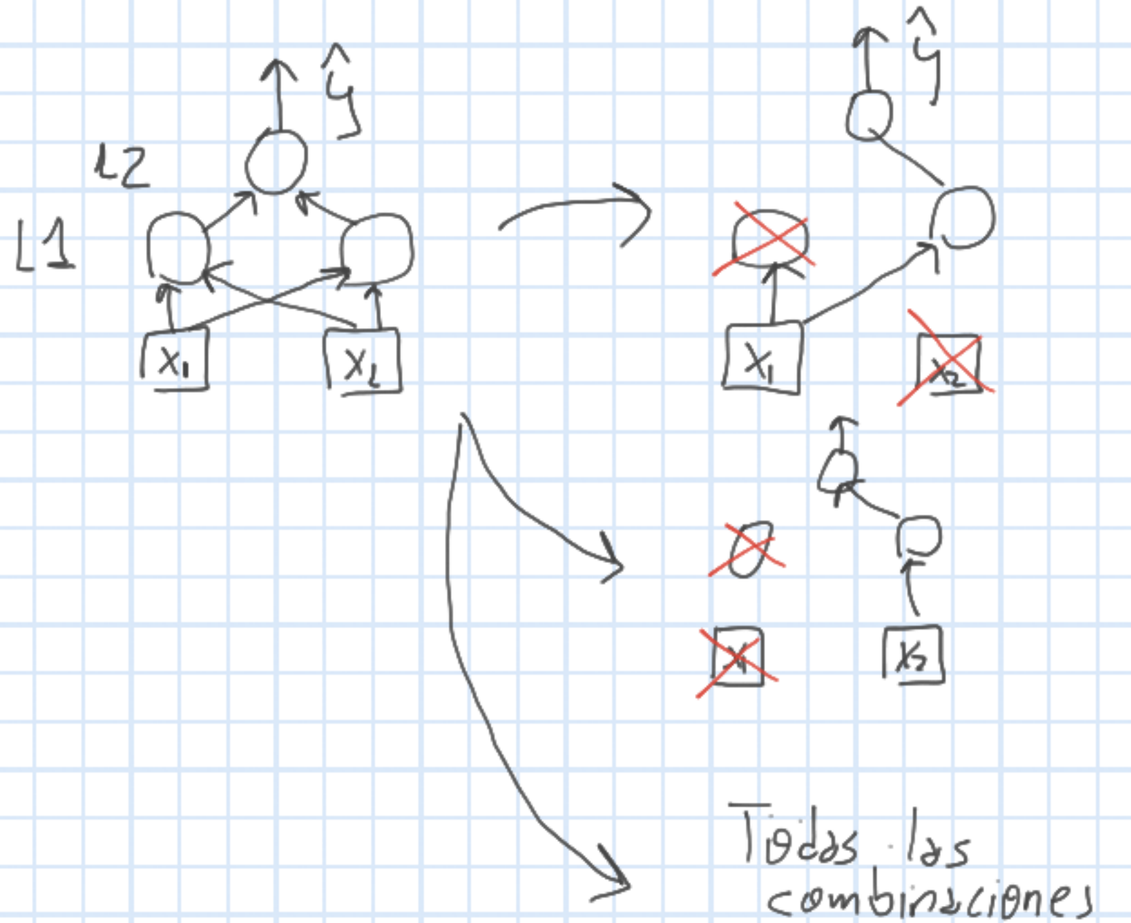
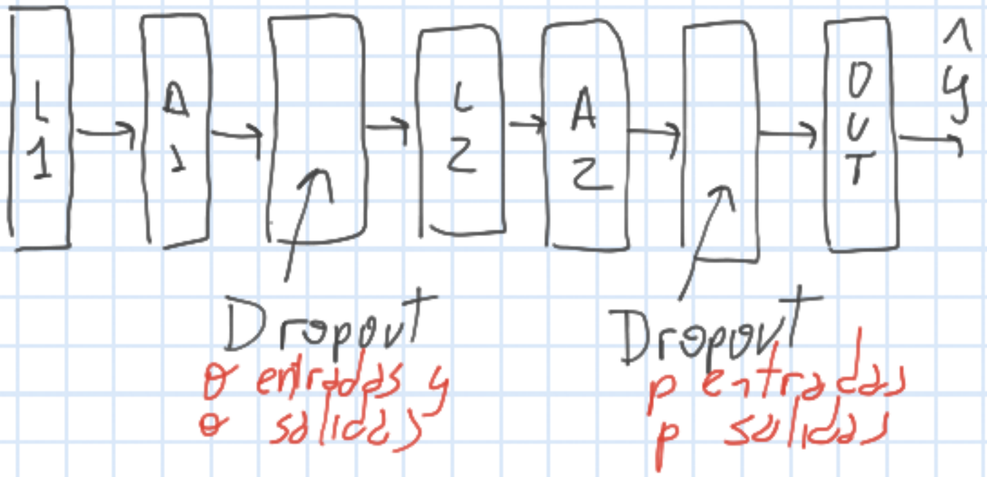
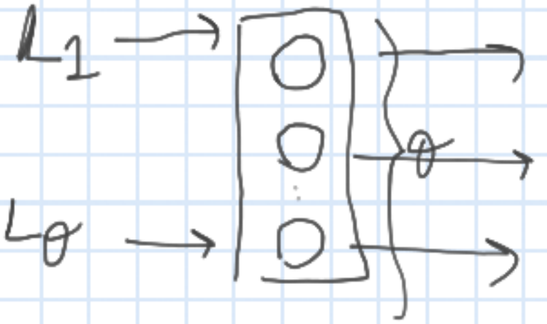
* Regularización Dropout







$P_i \sim \text{Bernoulli}_i(p)$

P_i son iid

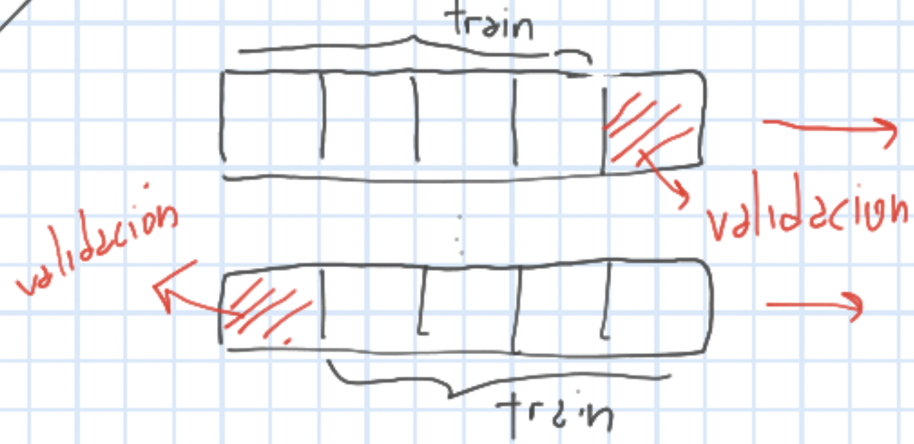
HP model 0,5

$$M_i = \begin{cases} L_i & \text{if } P_i = 1 \\ 0 & \text{if } P_i = 0 \end{cases}$$

* Optimizadores

(1) Dataset → redes neuronales

→ Cross-validation K-folds.



metrica_val 1

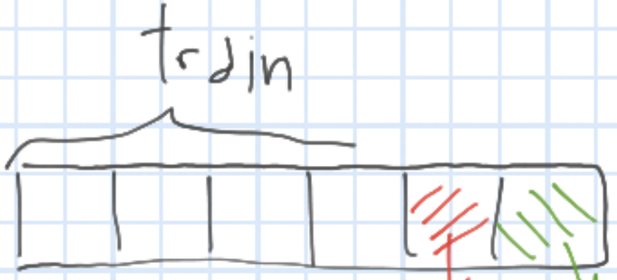
metrica_val K

$$metrica\_val = \frac{1}{K}\sum_{i=1}^{K} metrica\_val\_i$$

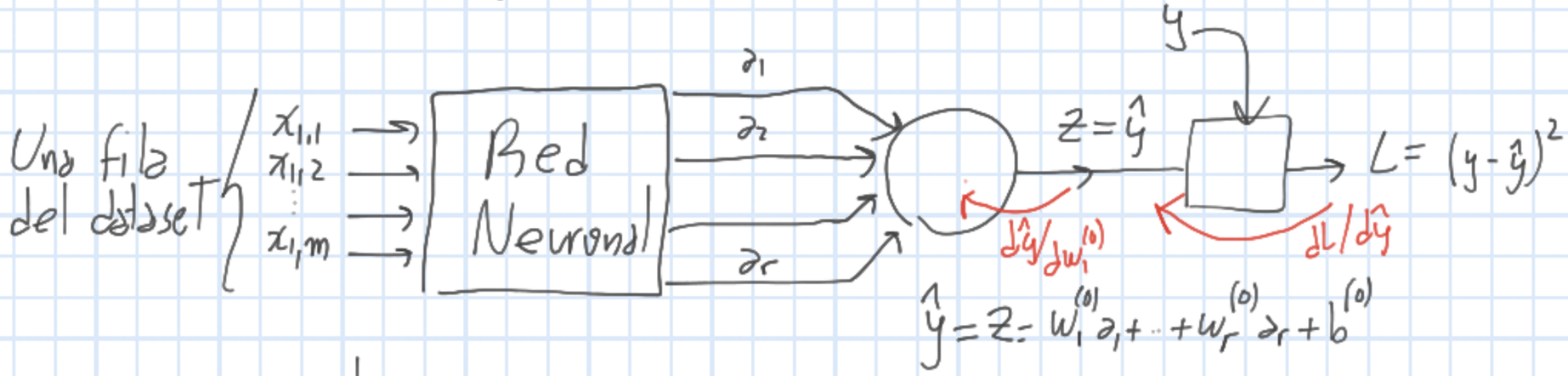
→ seleccionar HP modelo

→ me quedo con el mejor HP → test → metrica final

En DL:



* (2) Data normalization



* No normalizar entrada afecta velocidad de entrenamiento

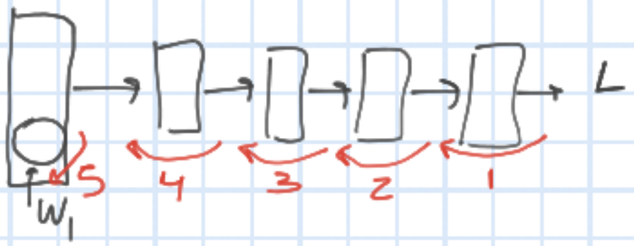
* No normalizar la salida puede generar gradientes que exploten

$$\frac{dL}{dw_1^{(0)}} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dw_1^{(0)}} = -2(y - \hat{y}) \; a_1$$

$$w_1^{(0)} \leftarrow w_1^{(0)} - \alpha \, dL/dw_1^{(0)}$$

$$w_1^{(0)} \leftarrow w_1^{(0)} + \alpha \, 2 (y - \hat{y}) \, a_1$$

*(3) Vanishing gradients

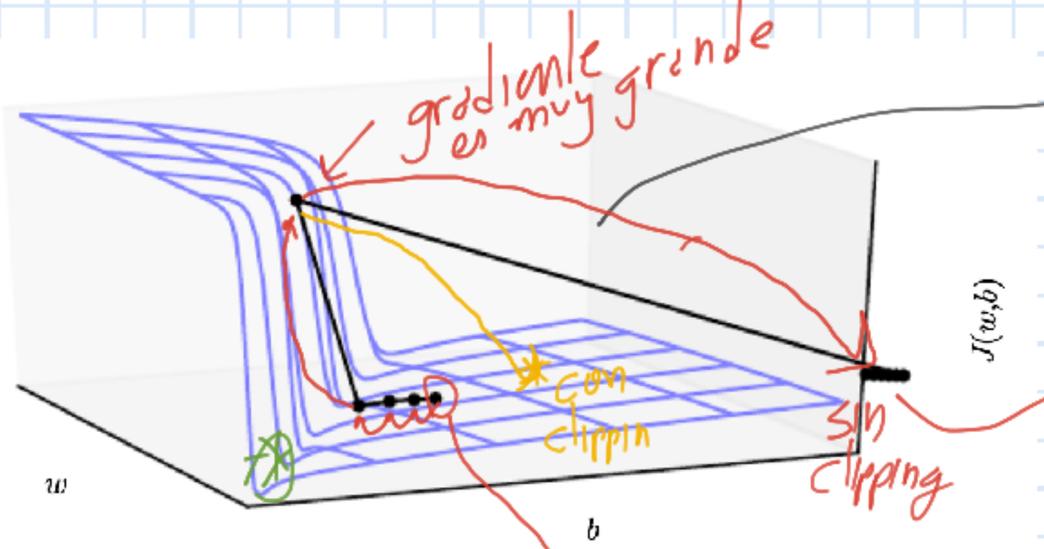


$$\frac{dL}{dw_1} = (1)(2)(3)(4)(5)$$

usando como activación $\Gamma(\cdot)$

- tiende a cero → aprox 0
  - ↳ $w_1 \leftarrow w_1 - \alpha \frac{dL}{dw_1} \longrightarrow w_1 \leftarrow w_1$ (no aprende)
- ¿Cómo se resuelve? → ReLu

*(4) Exploding gradients



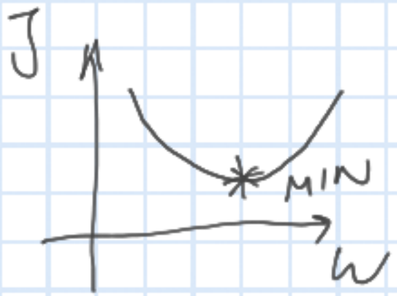
→ gráfico $J(w, b)$

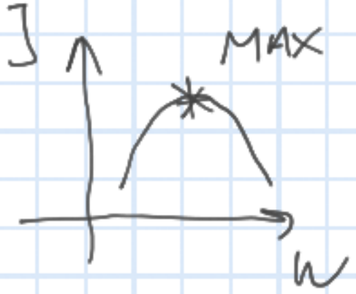
b_0 y w_0

perdí toda la optimización

↳ gradient clipping (limita el máx valor que puede tomar)

Optimización de funciones





(1D) $\frac{dJ}{dw} = 0$ (n dim) $\nabla_w(J) = \begin{bmatrix} \partial J/\partial w_1 \\ \vdots \\ \partial J/\partial w_0 \end{bmatrix} = \bar{0}$

- modelos lineales → $\bar{\nabla}(J) = \bar{0}$ tiene forma cerrada
- modelos más complejos → $\bar{\nabla}(J) = \bar{0}$ no tiene forma cerrada

necesito algoritmos de optimización

$\bar{w} \leftarrow \bar{w} + \alpha \bar{\nabla}(J$ (SGD)

$L: \mathbb{R}^1 \rightarrow \mathbb{R}^k$ k params desconocidos

$\bar{\nabla}(L) = \begin{bmatrix} \partial L/\partial w_1 \\ \vdots \\ \partial L/\partial w_k \end{bmatrix} \in \mathbb{R}^{k \times 1}$

$H(L) = \begin{bmatrix} \partial^2 L/\partial w_1^2 & \cdots & \partial^2 L/\partial w_1 \partial w_k \\ \vdots & & \\ \partial^2 L/\partial w_k \partial w_1 & \cdots & \partial^2 L/\partial w_k^2 \end{bmatrix}$

Punto: $\bar{w}_0$, $J(\bar{w}_0)$ dado

$$J(\bar{w}) \approx J(\bar{w}_0) + (\bar{w}-\bar{w}_0)^T \nabla J + \frac{1}{2}(\bar{w}-\bar{w}_0)^T H (w-\bar{w}_0) + \dots$$

↑ Taylor

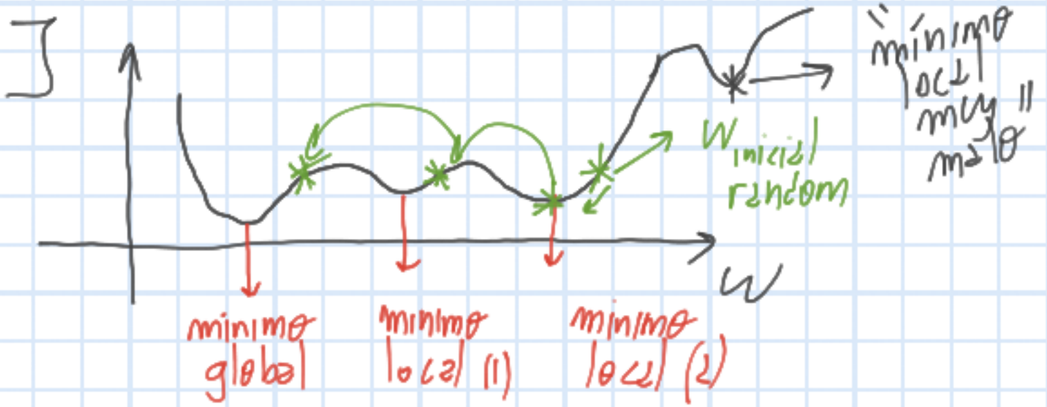
$$J(\underbrace{\bar{w}-\alpha \nabla J}_{\text{eq act pesos}}) \approx J(\bar{w}_0) - \alpha \nabla J^T \nabla J + \frac{1}{2}\alpha^2 \nabla J^T H \nabla J + \dots$$

↳ Ejemplo = $J = \frac{1}{2} w^2$ ↗ $\partial J/\partial w = x$

$\partial^2 J/\partial^2 w = 1$

Mínimo global vs mínimo local

DL → la función J es no convexa



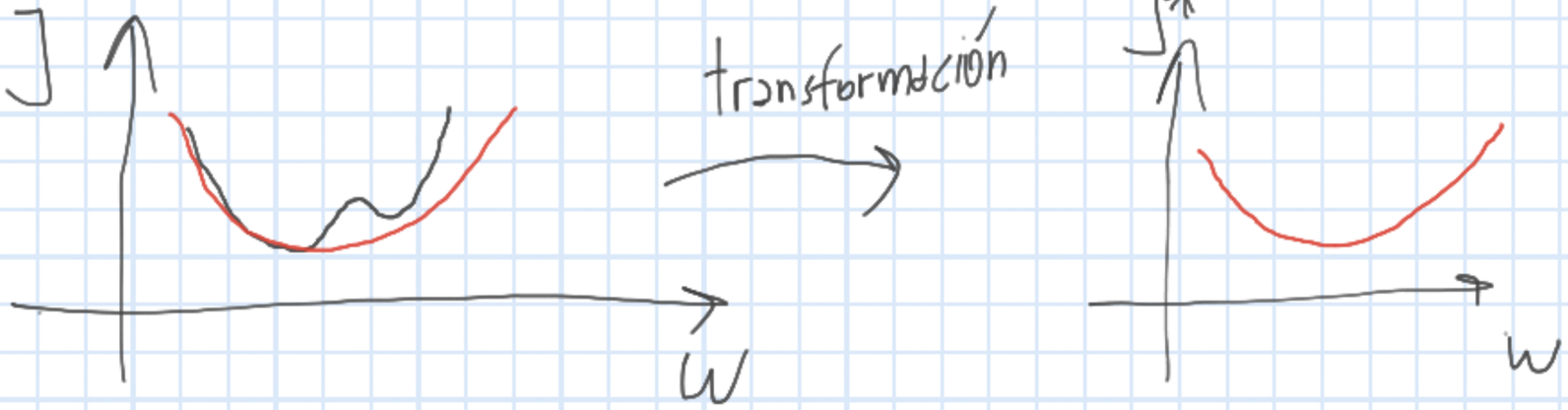
Pesos de la red neuronal

inician o con dist $\mathcal{U}$ o $\mathcal{N}$

$$W_i \leftarrow \mathcal{U}\left(-\frac{1}{\sqrt{m}}, +\frac{1}{\sqrt{m}}\right)$$

donde m es la cant. entradas

uniforme



* Optimizadores
  - Mini-batch SGD
  - SGD decaimiento lineal
  - momento de primer orden
  - AdaGrad
  - RMSProp
  - Adam (2014)

$W \leftarrow W - \alpha \nabla J$

siempre usando la derivada primera

distintas estrategias para los momentos

Optimizadores que usan derivada segunda
- Newton
- BFGS
- L-BFGS

→ dependen de H

(1) Mini-Batch

For e in epochs:

For b in batch:

* forward...

$$G = \nabla_{\bar{w}} \left(\frac{1}{b} \sum_{i=1}^{b} \ell(f(\bar{x}_i, \bar{w}), y) \right)$$

$$\bar{w} \leftarrow \bar{w} - \alpha \bar{G}$$

es el gradiente

(2) Mini-batch con decaimiento lineal

For $k=0$ e in epoch:

for b in batch:

* forward...

$$\alpha_k = \left(1 - \frac{k}{T}\right)\alpha_0 + \frac{k}{T}\alpha_T$$

$$\bar{G} = \nabla_w \left(\frac{1}{b} \Sigma \ell(\hat{y}, y) \right)$$

$$\bar{w} \leftarrow \bar{w} - \alpha_k \bar{G}$$

$k = k+1$

$k=0 \rightarrow \alpha_k = \alpha_0$

$k=T \rightarrow \alpha_k = \alpha_T$

* First order momentum

for e in epoch:

for b in batch:

forward ... $\hat{y}_i = f(\bar{x}_i, \bar{w})$

$$\bar{G} = \nabla\left(\frac{1}{b}\sum_{i=1}^{b} \ell(\hat{y}_i, y_i)\right)$$

$$\bar{v} \leftarrow \gamma \bar{v} + \alpha \bar{G}$$

$$\bar{w} \leftarrow \bar{w} - \bar{v}$$

$\gamma = 0 \rightarrow$ vanila mini-batch

$\gamma \neq 0$

* Momento de segundo orden: AdaGrad

for e in epoch:

for b in batch:

forward

$$\bar{G} = \nabla\left(\frac{1}{b}\sum_{i=1}^{b} \ell(\hat{y}, y)\right)$$

$$\bar{r} \leftarrow \bar{r} + \bar{G} \odot \bar{G}$$

HP (element wise multiplication)

$$\bar{G} \odot \bar{G} = \begin{bmatrix} \partial J/\partial w_1 & \partial J/\partial w_1 \\ \partial J/\partial w_n & \partial J/\partial w_n \end{bmatrix}$$

$$\bar{\Delta} \leftarrow -\frac{\alpha}{\sqrt{\bar{r}}} \odot \bar{G}$$

element wise

$$\bar{w} \leftarrow \bar{w} + \bar{\Delta} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} - \begin{bmatrix} \alpha_1 & \partial L/\partial w_1 \\ \vdots & \vdots \\ \alpha_n & \partial L/\partial w_2 \end{bmatrix}$$

RMSProp

for e in epoch:
  for b in batch:
    forward

$$G = \nabla\left(\frac{1}{b}\sum_{i=1}^{b} l(\hat{y}_i, y_i)\right)$$

$$\bar{r} = p\,\bar{r} + (1-p)\,\bar{G}\odot\bar{G}$$

$$\bar{\Delta} = -\alpha/\sqrt{\bar{r}}\odot\bar{G}$$

$$\bar{w} \leftarrow \bar{w} + \bar{\Delta}$$

$p=0 \rightarrow$ no tiene memoria

$p=1 \rightarrow$ usa mucha memoria

HPs
- epoch ✓
- batch ✓
- p ✓
- α ✓

Adam (2014)

for e in epoch:

for b in batch:

forward

$$\bar{G} = \nabla\left(\frac{1}{b}\sum_{i=1}^{b} \ell(\hat{y}_i, y_i)\right)$$

momento 1er order

$$\bar{V} \leftarrow \rho_1 \bar{V} + (1-\rho_1)\bar{G}$$

momento 2do orden

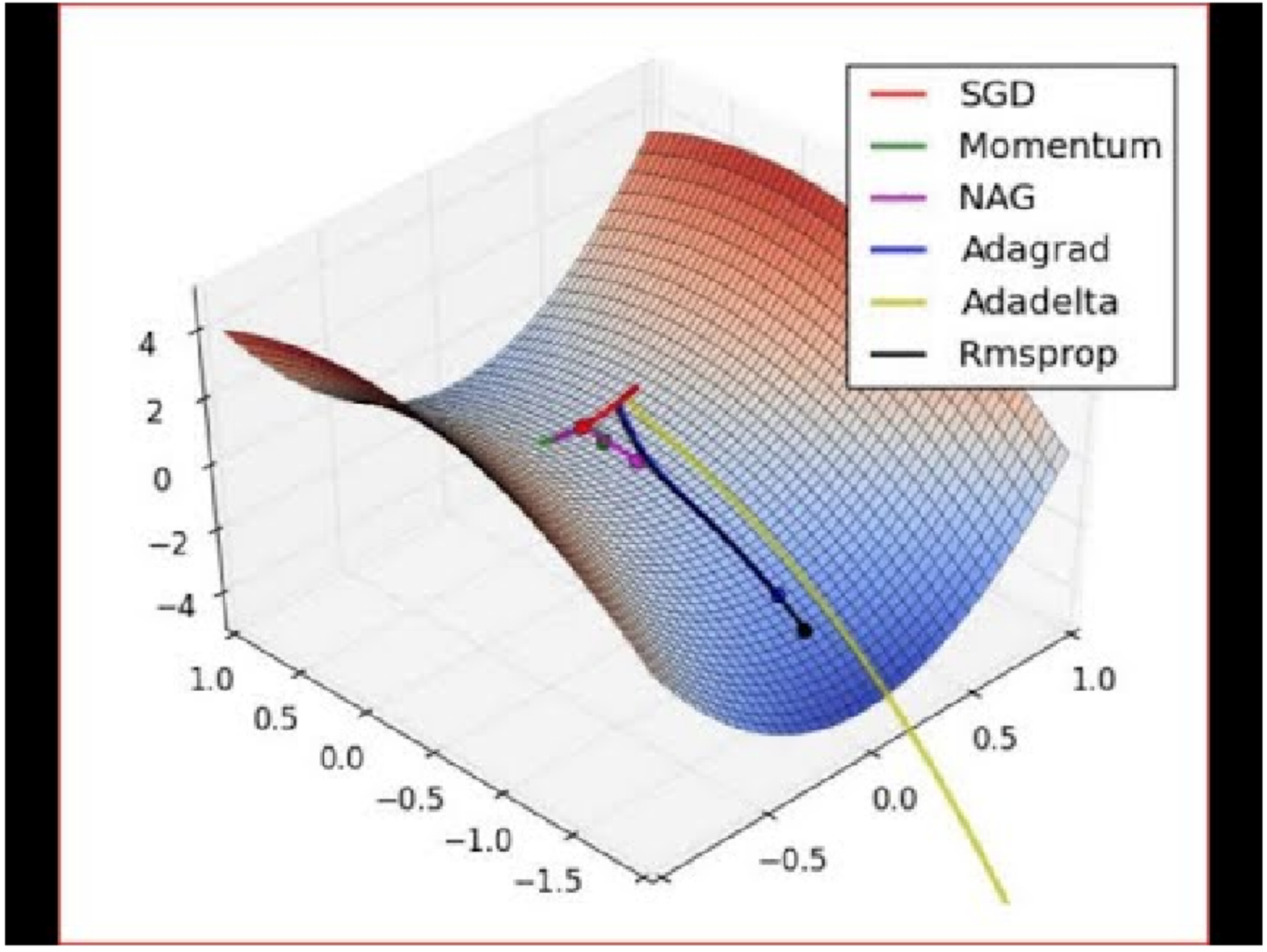
$$\bar{r} \leftarrow \rho_2 \bar{r} + (1-\rho_2)\bar{G} \odot \bar{G}$$

$$\bar{\Delta} \leftarrow -\alpha/\sqrt{r}\,\bar{V}$$

$$\bar{w} \leftarrow \bar{w} + \bar{\Delta}$$

HPs

- epoch
- batch
- ρ_1
- ρ_2
- α


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.5
0.0
0.5
1.0