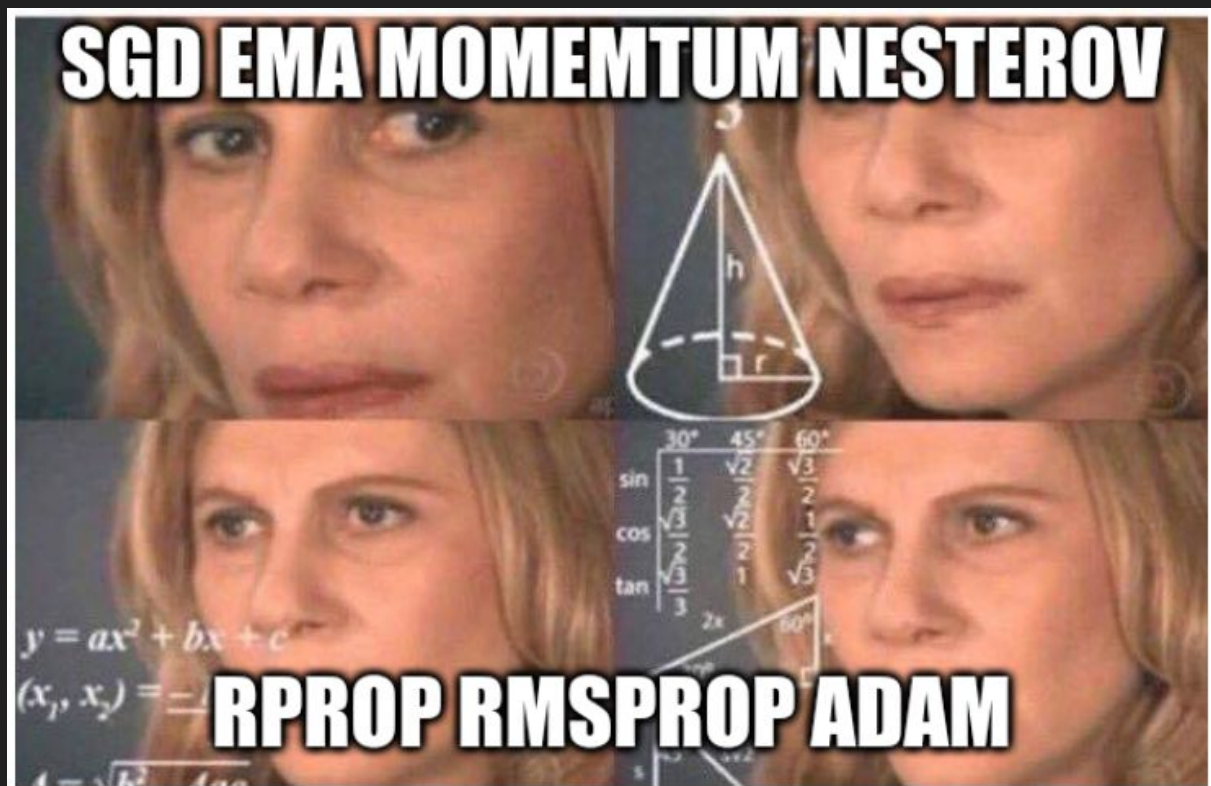


L07

Optimizers

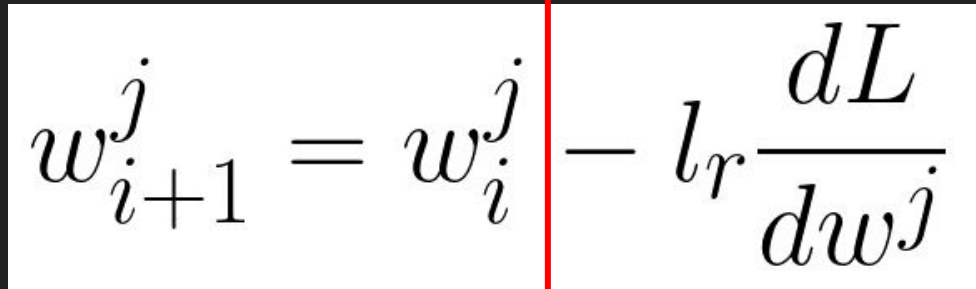
Intro



Loss Functions

They help model to “understand” to which direction it should move to.

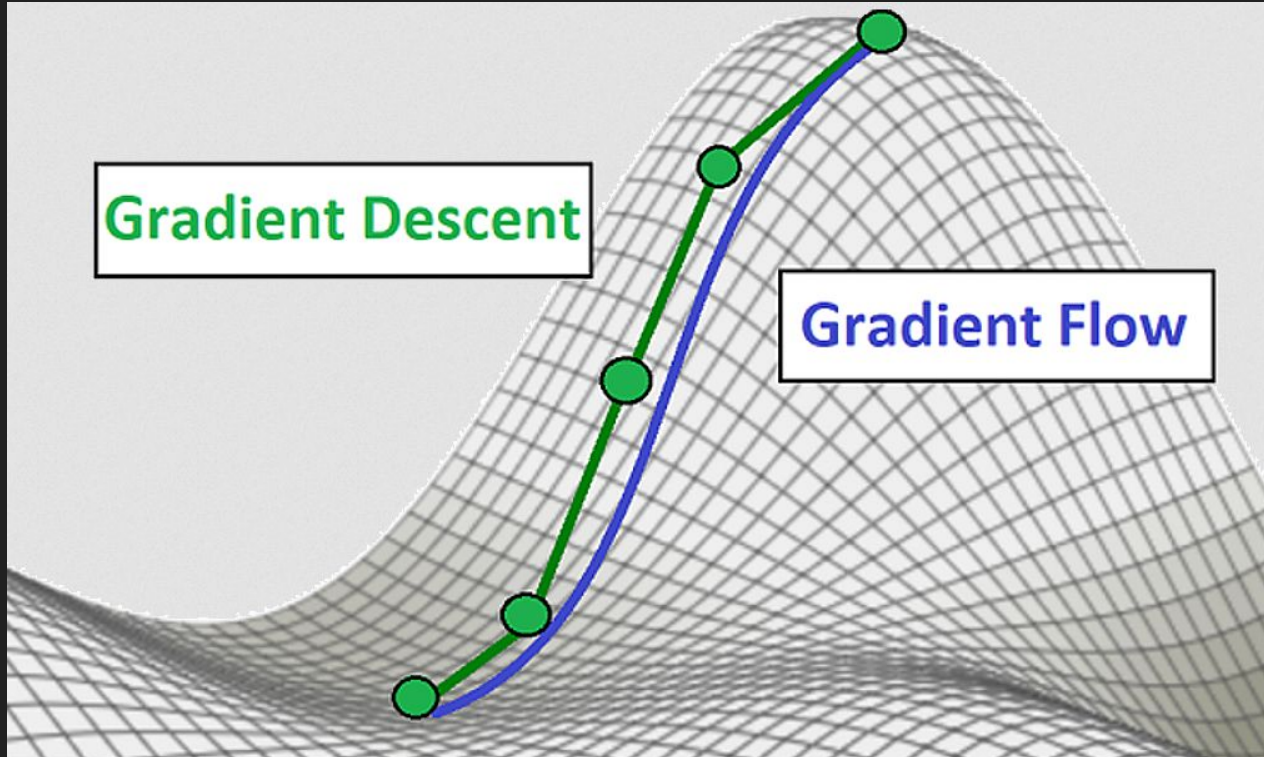
Optimizers are for making movement to the chosen direction.



The diagram shows the weight update equation $w_{i+1}^j = w_i^j - l_r \frac{dL}{dw_i^j}$ centered on a white background. A red rectangular box highlights the entire equation. Six red arrows point towards the equation from various angles: one from the top-left, one from the top-center, one from the top-right, one from the bottom-left, one from the bottom-center, and one from the bottom-right.

$$w_{i+1}^j = w_i^j - l_r \frac{dL}{dw_i^j}$$

Gradient Descent. Drawings.



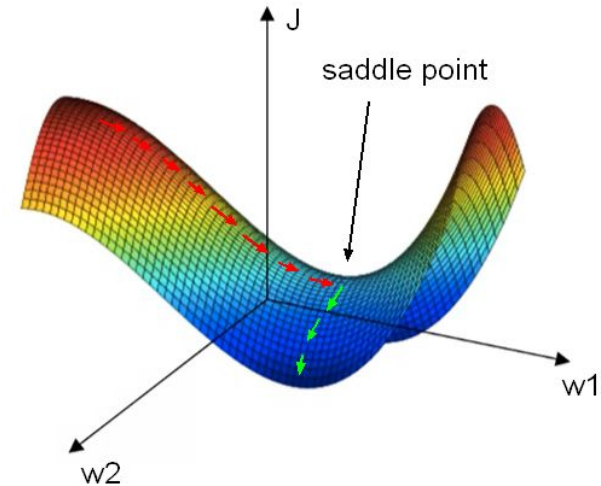
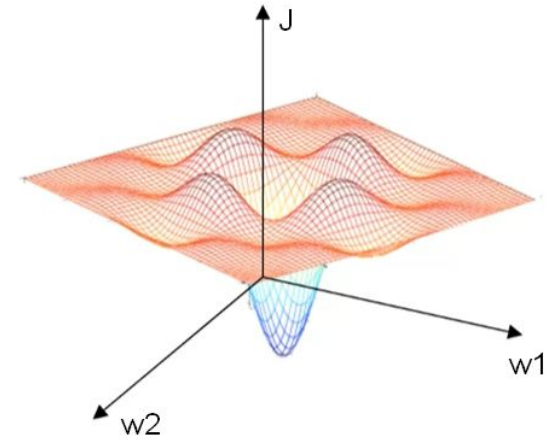
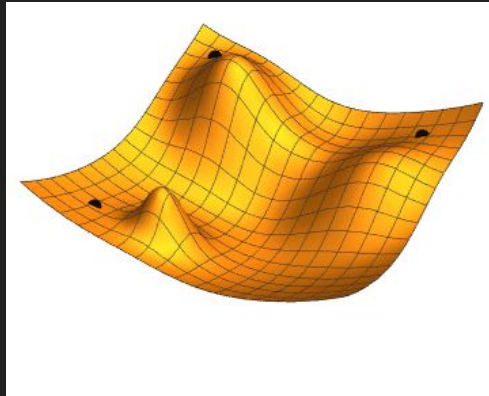
Gradient Descent

Pros:

- Can find minima

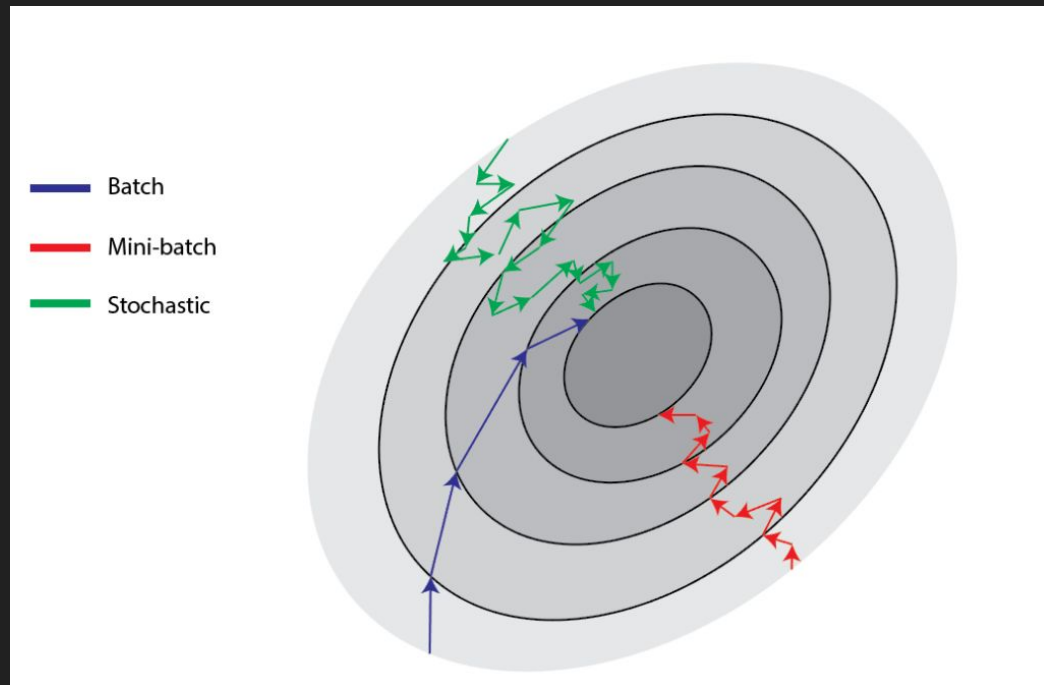
Cons:

- We have to calculate gradient on the whole data set
- Can stuck on plateau or shallow local minima



Batch Training

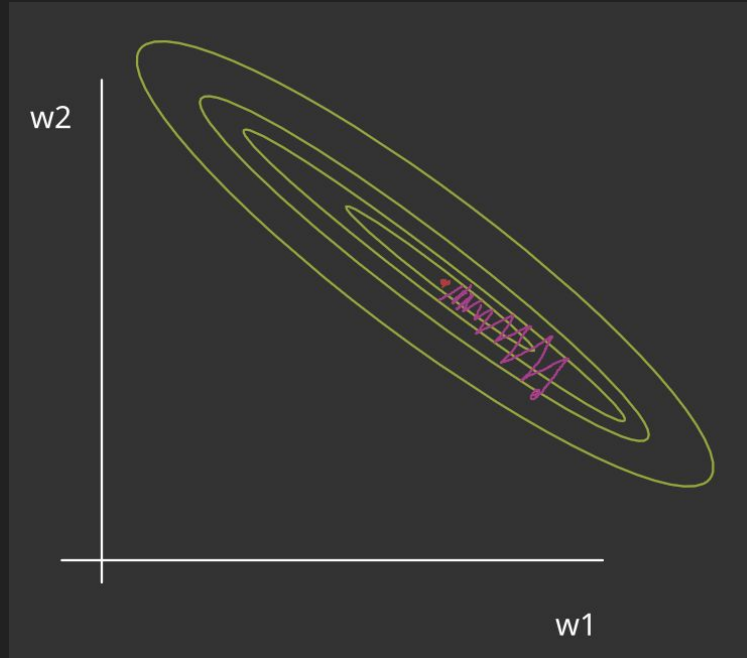
- **Batch Gradient Descent.**
Batch Size = Size of Training Set
- **Stochastic Gradient Descent.**
Batch Size = 1
- **Mini-Batch Gradient Descent.**
 $1 \ll \text{Batch Size} \ll \text{Size of Training Set}$



Gradient Descent Modifications

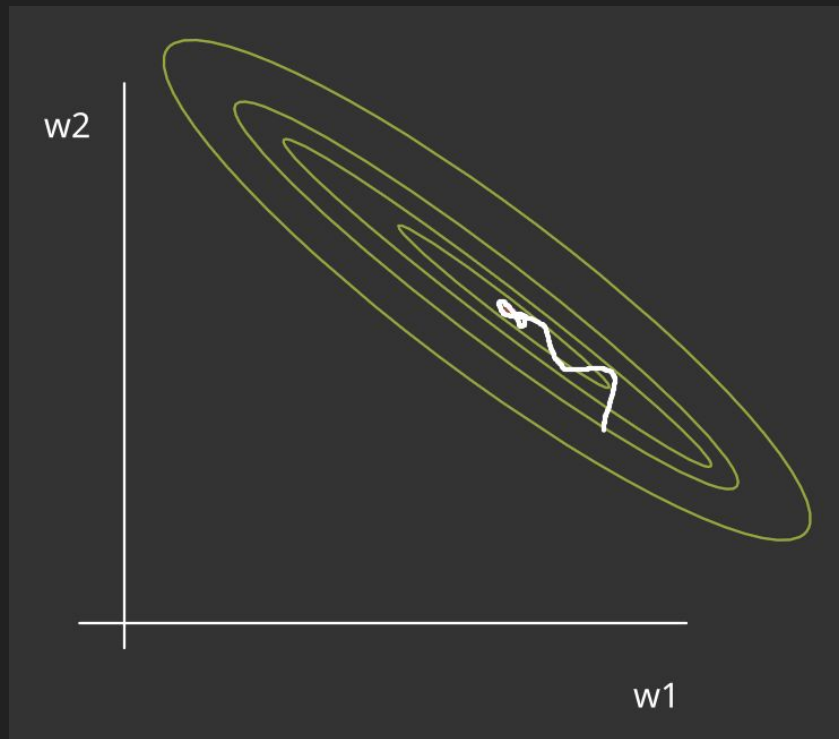


SGD



Ref: <https://mathformachines.com/posts/visualizing-the-loss-landscape/>

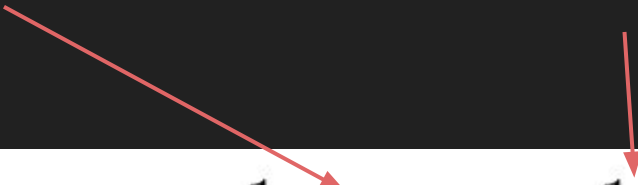
SGD with Momentum. Rolling Ball



SGD with Momentum. Rolling Ball. Newton Law.

Potential energy / Loss

Friction force


$$\begin{cases} \frac{\partial v}{\partial t} = \frac{1}{m}(F + F_{friction}) = -\frac{1}{m} \nabla f - \frac{1}{m} \eta v \\ \frac{\partial x}{\partial t} = v \end{cases}$$

Solving Equations. Finite difference method

$$\begin{aligned}\frac{\partial f}{\partial x} &\approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \Rightarrow \\ \frac{\partial v}{\partial t} &= \frac{v_{t+1} - v_t}{\Delta t} \\ \frac{\partial x}{\partial t} &= \frac{x_{t+1} - x_t}{\Delta t}\end{aligned}$$

Reference: https://en.wikipedia.org/wiki/Finite_difference_method

Solving Equations. Finite difference method

$$\begin{cases} v_{t+1} = v_t - \frac{\Delta t}{m} \nabla f - \frac{\Delta t}{m} \eta v_t \\ x_{t+1} = x_t + \Delta t v_t \end{cases}$$

$$\frac{\Delta t}{m} = 1$$

$$\Delta t = \alpha$$

$$v_t \left(1 - \frac{\Delta t}{m} \eta\right) = \beta$$

Solving Equations. Finite difference method. Solution

Stochastic Gradient Descent with Momentum

$$\begin{cases} x_{t+1} = x_t + \alpha v_t \\ v_{t+1} = \beta v_t - \nabla f \end{cases}$$

$$\begin{aligned} x_{t+1} &= x_t + \alpha(\beta v_{t-1} + \nabla f) = \\ &= x_t - \alpha \nabla f + \alpha \beta v_{t-1} \end{aligned}$$

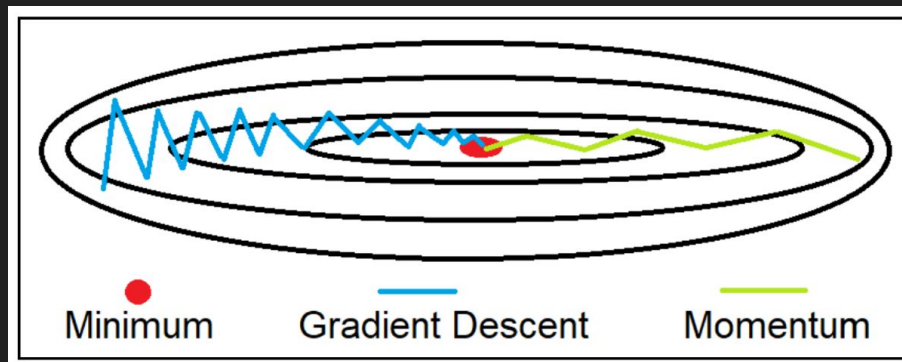
alpha - learning rate

beta - momentum factor

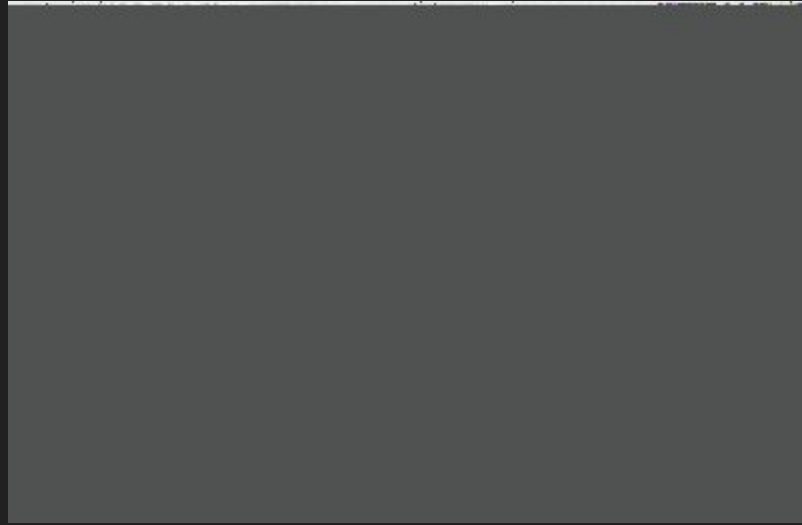
We want some kind of 'moving' average which would 'denoise' the data and bring it closer to the original function. Exponentially weighted averages can do the trick.

SGD with Momentum

- Gradient descent with momentum converges faster than standard gradient descent
- Momentum reduces oscillation
- We can set a higher learning rate
- Local minima can be an escape and reach global minima due to the momentum involved



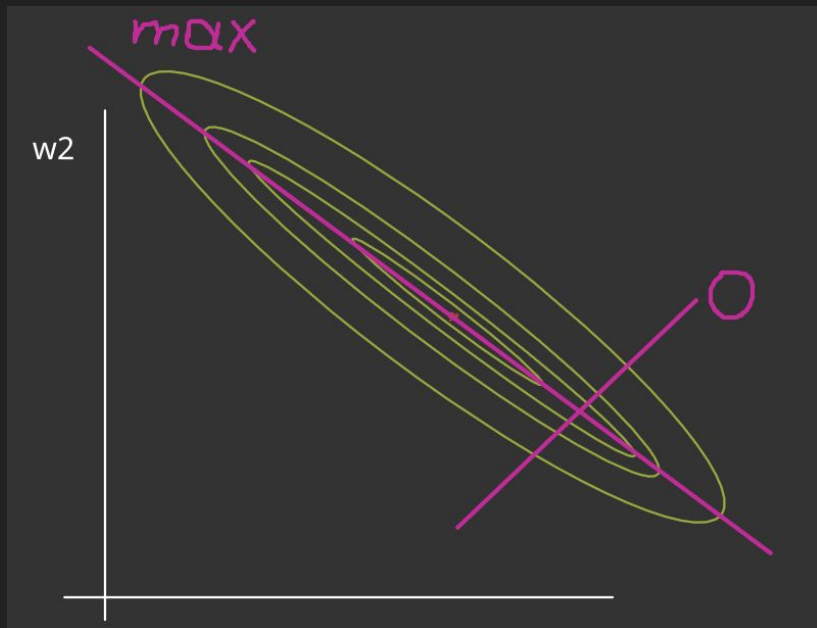
SGD with Momentum



Reference: <https://paperswithcode.com/method/sgd-with-momentum>

SGD with Exponential Moving Average

Almost the same as SGD with momentum

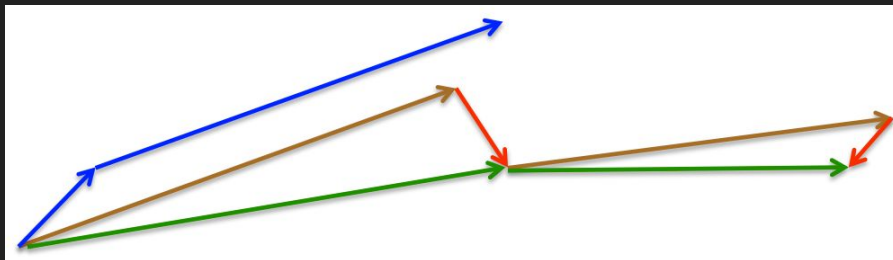


$$w_{t+1} = w_t - \alpha \text{EMA}(\nabla f)_t$$

$$\text{EMA}(f)_t = (1 - \beta)f_t + \text{EMA}(f)_{t-1}$$

Nesterov's Accelerated Gradient (Look Ahead)

The idea behind Nesterov's momentum is that instead of calculating the gradient at the current position, we calculate the gradient at a position that we know our momentum is about to take us, called as “look ahead” position. From physical perspective, it makes sense to make judgements about our final position based on the position that we know we are going to be in a short while.



- **First** make a big jump in the direction of the previous accumulated gradient.
- **Then** measure the gradient where you end up and make a correction.

brown vector = jump, **red** vector = correction, **green** vector = accumulated gradient **blue** vectors = standard momentum

Rprop

Takes into account not a gradient, but sign of gradient **sign**($\nabla \mathbf{f}$).

Learning rates are individual for each parameter.

$$w_{t+1}^i = w_t^i - \alpha_t^i \text{sign}(\nabla f^i(w_t))$$
$$\alpha_{t+}^i = \begin{cases} 1.2\alpha_t & \text{if } \text{sign}(\nabla f^i(w_t) \cdot \nabla f^i(w_{t-1})) > 0 \\ 0.6\alpha_t & \text{if } \text{sign}(\nabla f^i(w_t) \cdot \nabla f^i(w_{t-1})) \leq 0 \end{cases}$$

Works bad with batches

RMSProp

Intuition:

- If for some direction **derivative is high**, we want to **slow down** in this direction
- If for some direction **derivative is low**, we want to **speed up** in this direction

$$w_{t+1} = w_t - \alpha \frac{\nabla f(w_t)}{\sqrt{\text{EMA}(\nabla f^2)_t}}$$

$$\nabla f^2 = \left[\left(\frac{\partial f}{\partial w_0} \right)^2 \left(\frac{\partial f}{\partial w_1} \right)^2 \dots \left(\frac{\partial f}{\partial w_n} \right)^2 \right]$$

RMSProp + SGD with Momentum = Adam

$$w_{t+1} = w_t - \alpha \frac{\text{EMA}_{\beta_1}(\nabla f(w_t))}{\sqrt{\text{EMA}_{\beta_2}(\nabla f^2)_t} + \epsilon}$$

$$\nabla f^2 = \left[\left(\frac{\partial f}{\partial w_0} \right)^2 \left(\frac{\partial f}{\partial w_1} \right)^2 \dots \left(\frac{\partial f}{\partial w_n} \right)^2 \right]$$

$$\alpha = 3 \cdot 10^{-4}$$

$$\beta_1 = 0.9$$

$$\beta_2 = 0.999$$



Summary

<https://pytorch.org/docs/stable/optim.html>

<https://pytorch-optimizer.readthedocs.io/en/latest/index.html>

Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelata	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

HW

Dataset collection for multiclass classification, 1000 images per class. Options:

- Dogs, wolves, hienas
- Jaguar, pantera, lion
- Chihuahua, golden retriever, borzoi
- Coca-Cola, fanta, sprite
- Cars of 1930's, cars of 1960's, cars of 2020's

Addons (**not tested**):

- Firefox: [link](#)
- Google Chrome: [link](#)