In Supervised ML/DL, we have a training set of form:

$$D = \{ z_n = (x_n, y_n) \}_{i=1}^{N} \quad : \text{Data}$$

where $z_n = (x_n, y_n) \sim P_{x \times y}$ and $P_{x \times y}$ is an unknown underlying distribution.

We fix a Hypothesis class, H, to choose our model from, e.g., linear models, neural networks, etc.

We then define the Empirical Risk as:

$$\hat{R}(h) = R_D(h) = \frac{1}{N} \sum_{n=1}^{N} \ell(h; z_n) \quad : \text{Loss/Objective}$$

Empirical Risk Minimization (ERM) can be written as:

$$h_D^* = \underset{h \in H}{\operatorname{argmin}} \, R_D(h)$$

Stochastic Gradient Descent (SGD)

- Our goal is to minimize the empirical risk:

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} \ell(w, z_n)$$

- We know that in GD, our updates are:

$$w^{(t+1)} = w^{(t)} - \varepsilon \cdot \nabla_w L(w^{(t)})$$

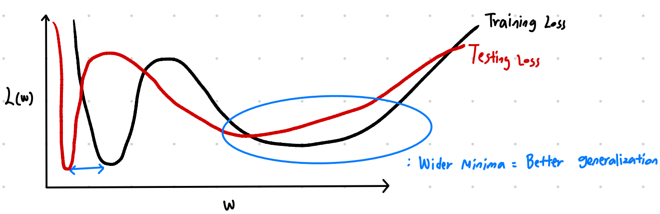
- In Stochastic GD, we use:

$$w^{(t+1)} = w^{(t)} - \varepsilon \nabla_w \hat{L}_k(w^{(t)}), \quad \text{where} \quad \hat{L}_k(w) = \underbrace{\frac{1}{b}}_{\text{Batch size}} \sum_{n \in \underbrace{\Omega_k}_{\text{Batch}}} \ell(w, z_n)$$

What are the benefits of SGD?

1. Faster optimization
2. Better generalization

↳ Why? : 'Wide Minima' Phenomenon. "The wider the minimum, the better the performance on the test set"



Finding Wider Minima

: We can think of each gradient calculated from SGD as a noisy version of the actual GD :

$$U_K(w) = \left[\nabla L(w) - \nabla \hat{L}_K(w)\right] = \frac{1}{b}\sum_{n\in\Omega_K}\left[\nabla L(w) - \nabla \ell(w, z_n)\right]$$

Stochastic Gradient Noise
: Difference between $\nabla_w L(w)$ and $\nabla \hat{L}_K(w)$

Zero-mean and i.i.d. RV

CLT states that the sum (or average) of large number of i.i.d. random variables will be approximately normally distributed.

If we assume that $U_K(w)$ has finite variance then, from Central Limit Theorem (CLT) we have that:

$$U_K(w) \sim N(0, \sigma^2 I)$$

Hence, with the Gaussian assumption we have:

$$W(t+1) = W(t) - \varepsilon\nabla L(W(t)) + \sqrt{\varepsilon}\sqrt{\varepsilon\sigma^2}\, Z(t)$$

Standard Gaussian random variable

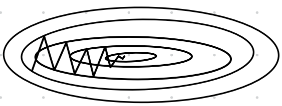
For small steps, we can write the discrete update above as a Stochastic Differential Equation (SDE):

$$dw_t = -\varepsilon\nabla L(w_t) + \sqrt{\varepsilon\sigma^2}\, dB_t$$

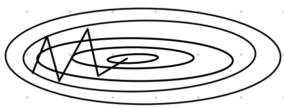
Brownian motion

What else can we do to get a flatter minima? : Momentum

Momentum Update



Plain gradient Update



With momentum

· The momentum method maintains a running average of all gradients until the current step.

$$\Delta W^{(k)} = \beta\Delta W^{(k-1)} - \varepsilon\nabla_w Loss\left(W^{(k-1)}\right)^T$$

maintains a running average

- Typical β value is 0.9

Ex)

$$\Delta W^{(1)} = \beta\Delta W^{(0)} - \varepsilon\nabla_w L(W^{(0)})^T$$

$$\Delta W^{(2)} = \beta\Delta W^{(1)} - \varepsilon\nabla_w L(W^{(1)})^T$$

$$= \beta\left(\beta\Delta W^{(0)} - \varepsilon\nabla_w L(W^{(0)})^T\right) - \varepsilon\nabla_w L(W^{(1)})^T$$

$$= \beta^2\Delta W^{(0)} - \beta\varepsilon\nabla_w L(W^{(0)})^T - \varepsilon\nabla_w L(W^{(1)})^T$$

$$\Delta W^{(3)} = \beta\Delta W^{(2)} - \varepsilon\nabla_w L(W^{(2)})^T$$

$$= \beta\left(\beta\left(\beta\Delta W^{(0)} - \varepsilon\nabla_w L(W^{(0)})^T\right) - \varepsilon\nabla_w L(W^{(1)})^T\right) - \varepsilon\nabla_w L(W^{(2)})^T$$

$$= \beta\left(\beta^2\Delta W^{(0)} - \beta\varepsilon\nabla_w L(W^{(0)})^T - \varepsilon\nabla_w L(W^{(1)})^T\right) - \varepsilon\nabla_w L(W(2))^T$$

$$= \beta^3\Delta W(0) - \beta^2\varepsilon\nabla_w L(W(0))^T - \beta\varepsilon\nabla_w L(W(1))^T - \varepsilon\nabla_w L(W(2))^T$$

And usually $\beta = 0.9 < 1$

Vanila GD : $W(t+1) = W(t) - \varepsilon\nabla_w L(W(t))$

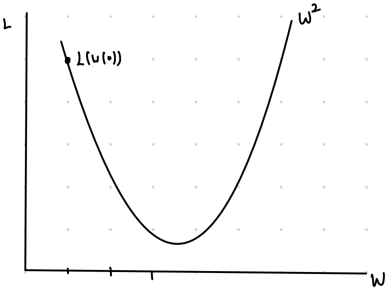
$$W(1) = W(0) - \varepsilon\nabla_w L(W(0))$$

$$W(2) = W(1) - \boxed{\varepsilon\nabla_w L(W(1))}$$

$$= \boxed{W(0) - \varepsilon\nabla_w L(W(0))} - \varepsilon\nabla_w L(W(1))$$

$$W(3) = W(2) - \varepsilon\nabla_w L(W(2))$$

$$= \boxed{W(0) - \varepsilon\nabla_w L(W(0)) - \varepsilon\nabla_w L(W(1))} - \varepsilon\nabla_w L(W(2))$$



Let $L(w(0)) = L(-4) = 16$, $\beta = 0.9$, $\varepsilon = 0.01$

Momentum

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \varepsilon \nabla_w Loss\left(W^{(k-1)}\right)^T$$

↳ maintains a running average

$\delta w(0) = -4$ →

helps moving fast

$\Delta w(1) = \beta \Delta w(0) - \varepsilon \nabla_w L(w(0)) = -4\beta + 8\varepsilon = -3.52$

$\Delta w(2) = \beta \Delta w(1) - \varepsilon \nabla_w L(w(1)) = -3.088$

$\Delta w(3) = -2.69$

$\Delta w(4) = -2.03$

$\Delta w(5) = -1.75$

⋮

$\delta w(18) = -0.0005$

Vanila GD

$w(0) = -4$

$w(1) = w(0) - \varepsilon \nabla_w L(w(0)) = -4 + 8\varepsilon = -3.992$

$w(2) = w(1) - \varepsilon \nabla_w L(w(1)) = -3.984$

⋮

$w(500) = w(499) - \varepsilon \nabla_w L(w(499)) = -0.079$

GD w/ Momentum converges much faster than Vanila GD.

However, since Momentum has running average aspect it sometimes overshoots.

- The running average step:
  - Get longer in directions where gradient retains the same sign
  - Become shorter in directions where the sign keeps flipping
- At any iteration, to compute the current step:
  - First compute the gradient step at the current location
  - Then add in the scaled previous step. (which is actually a running average)

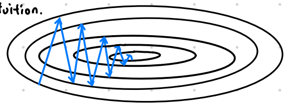
Nestorov's Accelerated Gradient : Momentum: Gradient step → extend previous step
Nesterov : extend previous step → then compute gradient

: It change the order of operations.

- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient step at the resultant position
  - Add the two to obtain the final step.
- Nestorov's Accelerated Gradient converges much faster.

Smoothing the trajectory : Normalizing steps by second moment.

Intuition.



: Huge oscilations in the vertical direction even while it's trying to make progress in the horizontal direction

Hence, we need optimization that goes vertically slow and horizontally fast.

Two popular methods that embody this principle : Adam and RMS Prop

RMSProp : Updates are by parameters

· Notation

- · Derivative of loss w.r.t any individual parameter w is shown $\partial_w D$
- · The squared derivative is $\partial_w^2 D = (\partial_w D)^2$. Not the second derivative
- · The mean squared derivative is running estimate of the average squared derivative. $\mathbb{E}[\partial_w^2 D]$

Modified Update rule:

1. Scale down updates with large mean squared derivatives
2. Scale up updates with small mean squared derivatives

RMSProp is a variant on the basic mini batch SGD algorithm.

Procedure:

- Maintain a running estimate of the mean squared value of derivatives for each parameter
- Scale update of the parameter by the inverse of the root mean squared derivative

$\mathbb{E}[\partial_w^2 D]_k = \gamma \cdot \mathbb{E}[\partial_w^2 D]_{k-1} + (1-\gamma)(\partial_w^2 D)_k$: running estimate of the mean squared value of derivatives for each parameter

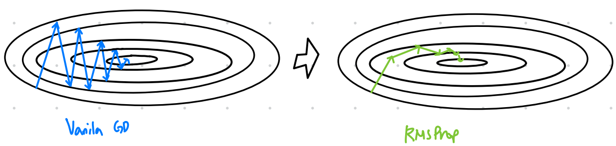
$$W_{k+1} = W_k - \frac{\eta}{\sqrt{\mathbb{E}[\partial_w^2 D]_k + \varepsilon}} \partial_w D$$

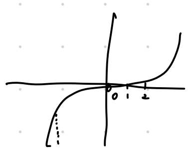
Preventing $\sqrt{\mathbb{E}[\partial_w^2 D]_k}$ not goes to 0.

where η : learning rate

Hence, if $\sqrt{\mathbb{E}[\partial_w^2 D]_k + \varepsilon}$: $\sqrt{\text{running average}}$ is small, scale ↑ : fastening

large, scale ↓ : slowing down





Adam : RMSProp + Momentum

: RMS Prop only considers a second-moment normalized version of the current gradient
ADAM utilizes a smooth version of the momentum-augmented gradient
↳ Considers both first and second moment

Procedure:

- Maintain a running estimate of the mean derivative for each parameter
- Maintain a running estimate of the mean squared value of derivatives for each parameter
- Scale update of the parameter by the inverse of root mean squared derivative

$$m_k = \delta m_{k-1} + (1-\delta)(\partial_w D)_k$$

$$v_k = \gamma v_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

$$\hat{m}_k = \frac{m_k}{1-\delta_k}, \quad \hat{v}_k = \frac{v_k}{1-\gamma_k}$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{v}_k + \varepsilon}} \hat{m}_k$$

Typical Params:

- RMSProp: $\eta: 0.001$, $\gamma = 0.9$
- ADAM: $\eta: 0.001$, $\delta = 0.9$, $\gamma = 0.999$