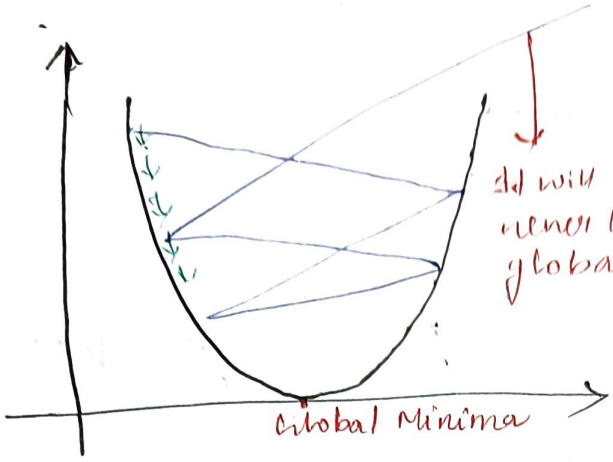
15-Jan-2023

Exploding Gradient Problem



> weight updation in such a way that it is jumping here and there.

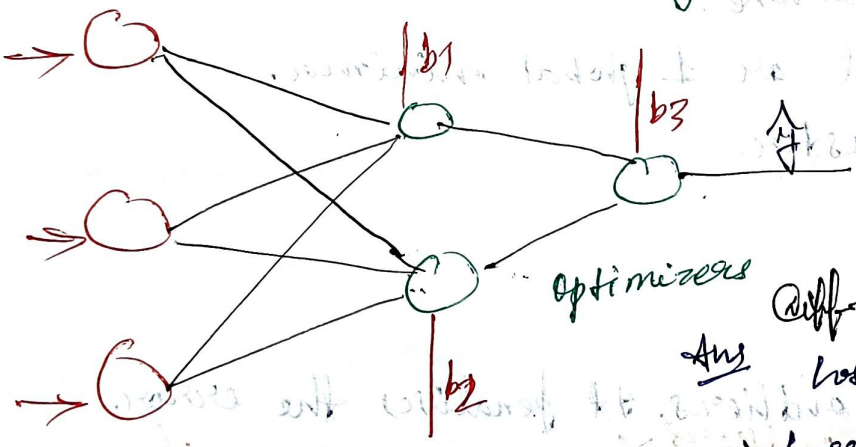
dd will never come to global minima.

Q. Why this jumping problem happen?

Ans Because of higher weight initialization.

Forward Propagation

Binary classification



Loss function
$= (y - \hat{y})^2$

Back Propagation

Diff b/w loss and cost fn?

Ans Cost fn $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

> Loss is for single data point

> Cost is for batch data points.

Loss Functions

ANN

1. Classification
2. Regression

Regression

1. MSE (Mean Squared Error)
2. MAE (Mean absolute Error)
3. Huber loss
4. RMSE (Root mean Squared Error)

① MSE (Mean Squared Error)

Loss $f^n = (y - \hat{y})^2$ Cost $J^n = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

↓

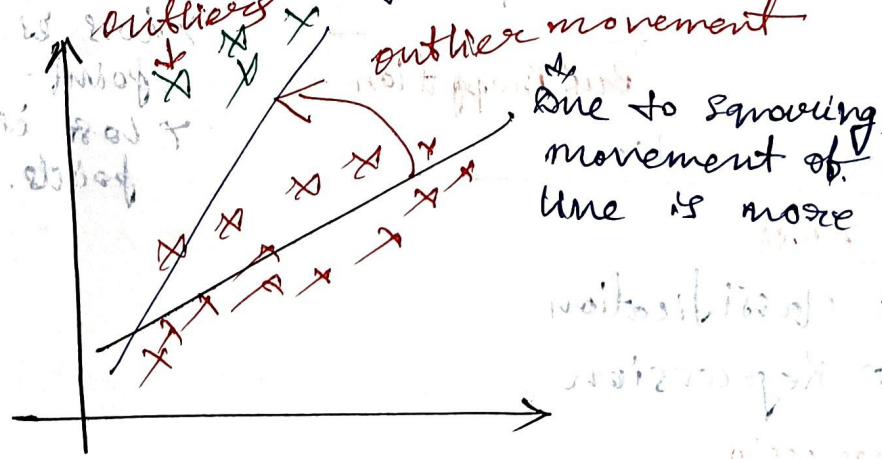
Quadratic Equation

Advantage

1. MSE is differentiable.
2. It has 1 local or 1 global minima.
3. It converges faster.

Disadvantage

1. Not robust to outliers. It penalises the error.



② MAE (Mean Absolute Error)

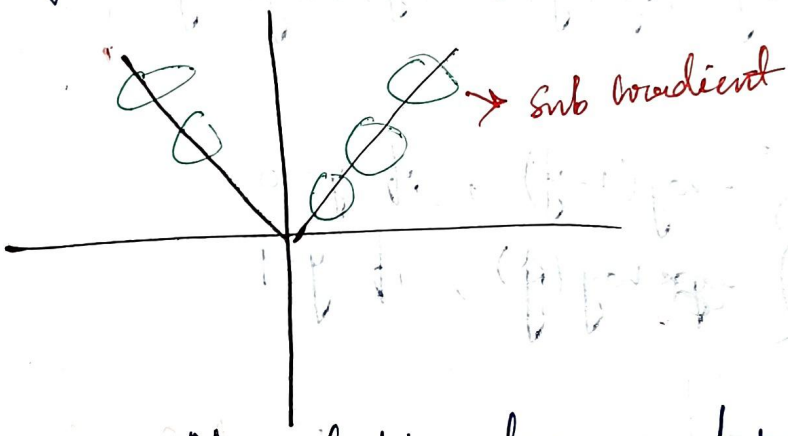
$$\text{Loss } f^n = |y - \hat{y}| \qquad \text{cost } f^n = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Advantage

① Robust to outliers, because movement of line is slow. We are not squaring here.

Disadvantage

① convergence is slow.



② It is not differentiable at every point.

③ Huber's Loss

> combination of MSE and MAE

Hyperparameter

$$\text{cost } f^n = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, & \text{if } |y-\hat{y}| \le \delta \\ \delta\,|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

④ RMSE

$$RMSE = \sqrt{MSE}$$

Loss or Cost fn for classification Problem

CROSS ENTROPY

1. BINARY CROSS ENTROPY (Binary class)
2. CATEGORICAL CROSS ENTROPY (Multi class)
3. SPARSE ~~CRO~~ CATEGORICAL ENTROPY (Multi class)

① Binary class Entropy (Binary classification)

$$\text{loss fn} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$ ← log loss

$$\text{loss fn} = \begin{cases} -\log(1-\hat{y}), & \text{if } y=0 \\ -\log(\hat{y}), & \text{if } y=1 \end{cases}$$

$$\hat{y} = \frac{1}{1+e^{-z}}$$ ← z = Activation fn (input * weight)

↓

Sigmoid Activation Function

② Category categorical cross Entropy (Multiclass classification)

	f_1	f_2	f_3
$i=1$	2	3	4
$i=2$	5	6	7
$i=3$	8	9	10

O/P	Good ($j=1$)	Bad ($j=2$)	Neutral ($j=3$)
Good	1	0	0
Bad	0	1	0
Neutral	0	0	1

$C=3$ = no. of category

> Categorical cross entropy will apply one-hot encoding.

$$\text{loss fn}(X_i, Y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y}_{ij})$$

Actual values = y_{ij} $\begin{bmatrix} y_{11} & y_{12} & y_{13} & \cdots & y_{1c} \\ y_{21} & y_{22} & y_{23} & \cdots & y_{2c} \end{bmatrix}$

$$y_{ij} = \begin{cases} 1, & \text{if the element is in the class} \\ 0, & \text{otherwise} \end{cases}$$

Prediction = $\hat{y}_{ij}$ ⇒ Softmax Activation = $\text{Softd}(z_i)$

$$= \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$= \frac{e^{z_i}}{e^1 + e^2 + e^3 + \cdots + e^K}$$

③ SPARSE Categorical Entropy

o/p = [0.2, 0.3, 0.5] → [2nd index]

highest

> It will give the highest value index as output.

Ex [0.2, 0.3, 0.1, 0.2, 0.2]

highest 1st index.

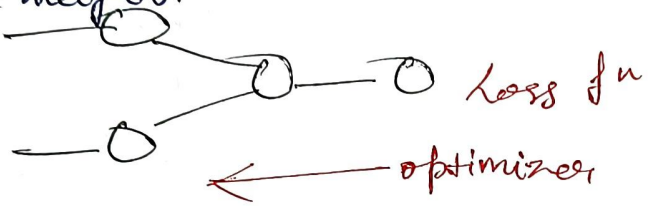
> losing other information.

Right combination of HL, O/P layer, Problem Statement & Loss fn

	Hidden layers	Op layer	Problem Statement	Loss function
①	Relu and its variants	Sigmoid	Binary classification	Binary cross entropy
②	Relu and its variants	Softmax	Multi class	Categorical or Sparse
③	Relu or its variants	Linear	Regression	MSE, MAE, Huber loss, RMSE

Optimizers ☆

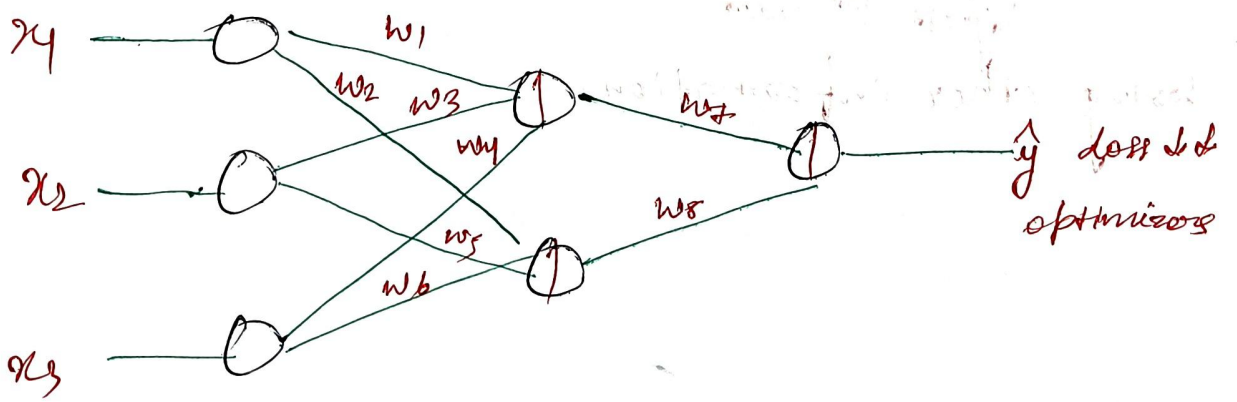
- To update weights.



① Gradient Descent
② SGD (Stochastic Gradient Descent)
③ SGD with momentum
④ Mini batch SGD
⑤ ~~Adadelta~~ ~~Adagrad~~ and RMSProp
⑥ Adam optimizers
⑦ Adagrad

① Gradient Descent optimizers

(all the records are taking together in B.I F.P & B.P)

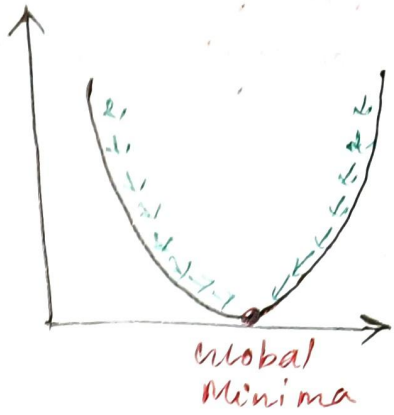


weight updation formula

$$W_{new} = W_{old} - \eta \left[\frac{\partial loss}{\partial w_{old}} \right]$$

↓ slope

Chain Rule of Derivative



MSE

$$\text{Loss fn} = (y - \hat{y})^2$$

$$\text{cost fn} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Epochs vs Iterations

Datapoints = 1000

Epochs 1 { 1000 datapoints → F.P; ← B.P (weight will get updated) } → $\hat{y}_i$, cost fn ↓.

Epochs 2 { → ; ← (weight will get updated) } → $\hat{y}_i$, cost fn ↓.

⋮

> This will continue till cost fn is minimized.

Example: cost fn decreased till 98th epochs & after that it is not decreasing, then stop at that point.

Advantage

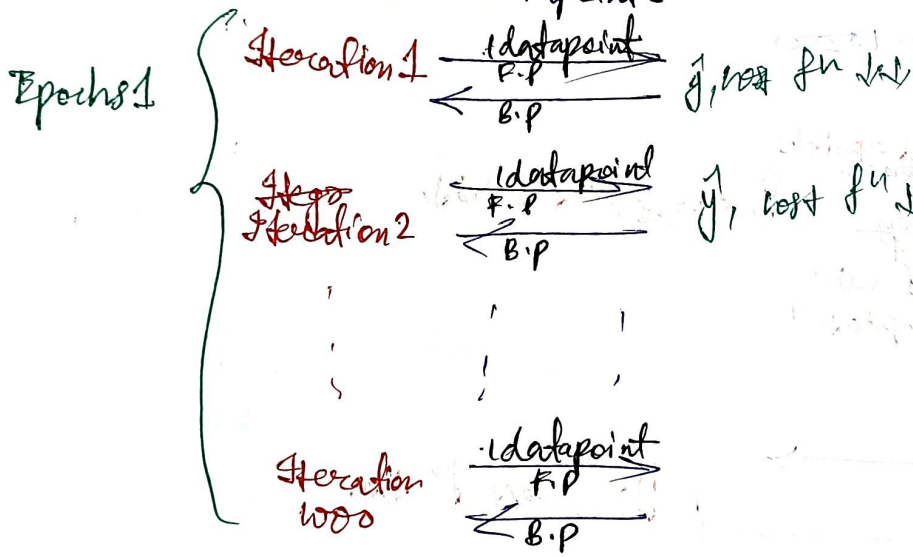
① . Convergence will happen.

Disadvantage

① It will require high RAM and CPU. It is resource intensive.

② Stochastic Gradient Descent

1000 datapoints



Epochs 2

Epochs 1000

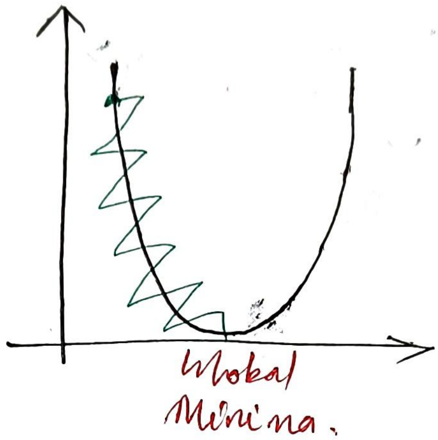
Advantage

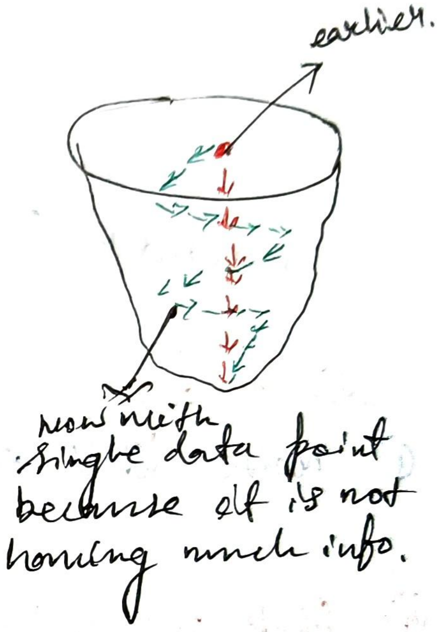
① Solved resource issue

Disadvantage

① Time complexity will be very high.

② Noise will get introduced because of single data point. Movement in the graph is called noise.





③ Mini Batch SGD

> Along with epochs and iterations, we give batch size.

Datapoints = 10000
batch size = 100
Epoch = $\frac{10000}{100}$ = 100

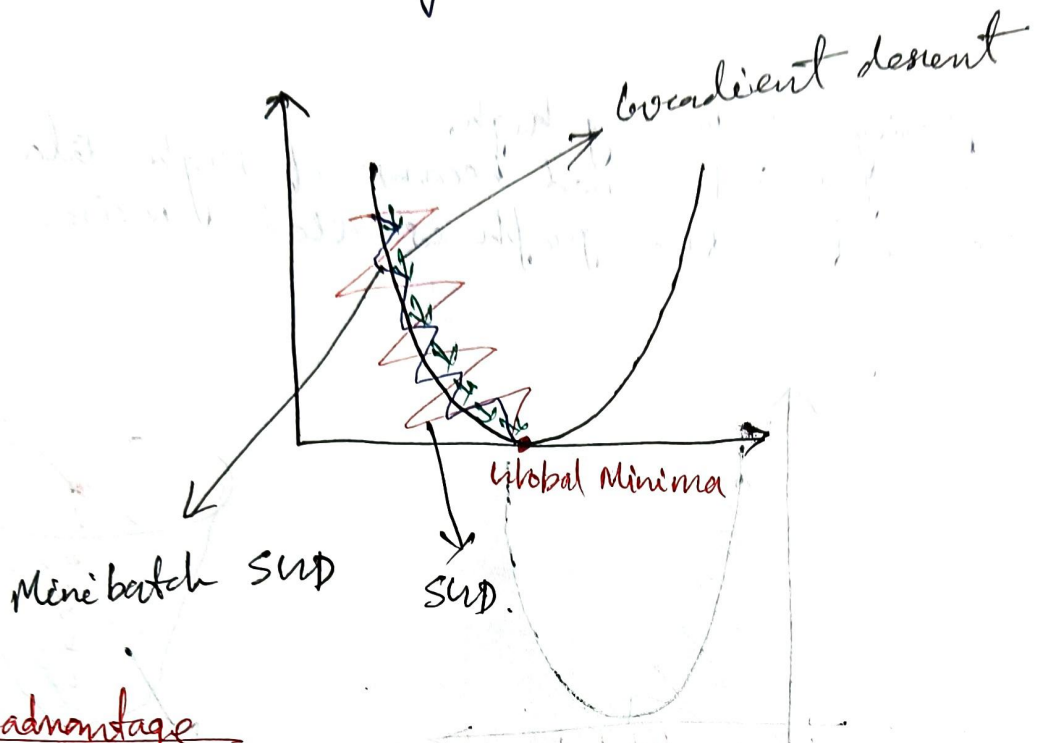
100 datapoints as batch

Epoch 1 { Iterat —F.P→ ŷ, loss fn …
Iterat ←B.P—

Epoch 100

Advantages

① Solved resource issue, with convergence speed increases
② Noise will get reduced, when compared to SGD.
③ Efficient resource usage.



Disadvantage

① Noise still exist.

④ SGD with momentum

$$W_{new} = W_{old} - \eta \frac{\partial loss}{\partial W_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial loss}{\partial b_{old}}$$

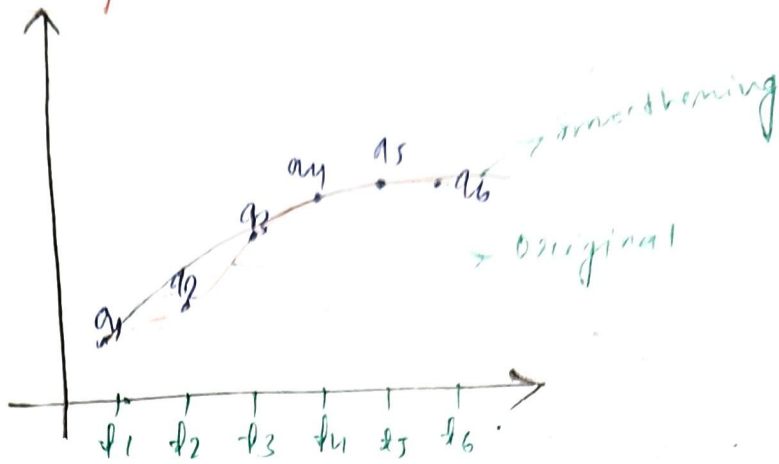
$$w_t = w_{t-1} - \eta \frac{\partial loss}{\partial w_{t-1}}$$ ⇒ Using this to smoothen

Exponential Weight Average (Smoothening)

Time	t_1	t_2	t_3	t_4	...	t_n
value	a_1	a_2	a_3	a_4	...	a_n

$V_{t_1} = a_1$

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$ → exponential smoothening



Let's $\beta = 0.95$

$V_{t_2} = 0.95 a_1 + 0.05 a_2$

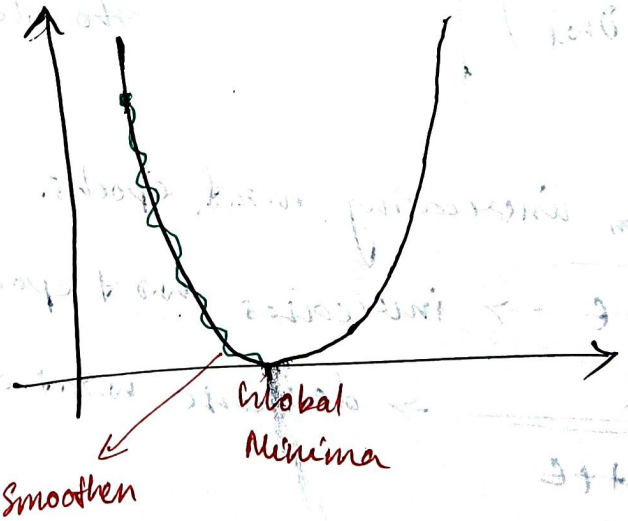
↓
has more control

→ In exponential smoothening, value at $t-1$ will control the value of t.

$V_{t_3} = \beta * V_{t_2} + (1-\beta) V_{t_3}$

$= 0.95 a_2 + 0.05 a_3$

After Smoothening



→ Noise will get reduced.

Advantages

① It reduces the noise.
② It smoothen the noise
③ Quick convergance.

⑤

⑤ Adagrad (Adaptive Gradient Descent)

$W_t \geq W_{t-1} + \cdots$

$$W_t = W_{t-1} - \eta \frac{\partial loss}{\partial W_{t-1}}$$

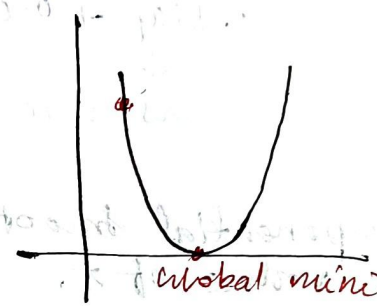
η = fixed

$$W_t = W_{t-1} - \eta' \frac{\partial loss}{\partial W_{t-1}}$$ → Updated

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$ → Dynamic Learning Rate

→ Epsilon = small value to prevent the zero condition for α_t

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial loss}{\partial w_t}\right)^2$$



> Convergance may happen at the constant rate.
> But we want convergance should be faster at the begining and after that it should be slow.

> α_t will keep on increasing w.r.t epochs.

$\sqrt{\alpha_t + \epsilon}$ → increases w.r.t epochs

$\frac{\eta}{\sqrt{\alpha_t + \epsilon}}$ → decrease w.r.t epochs.

Disadvantages

① If α_t becomes large in very large neural network

$w_t \approx w_{t-1}$

Vanishing Gradient Problem can occur.

② η' = Possibility to become a small value ≈ 0.

⑥ Adadelta and RMS Prop (Bringing exponential weighted average in learning rate)

$$\eta' = \frac{\eta}{\sqrt{S_{dw_t} + \epsilon}}$$ → Dynamic Learning Rate

S_{dw_t} → Exponential Weight Average

Exponential Weighted Average

$S_{dw_t} = 0$

$$S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial loss}{\partial w_t}\right)^2$$

⑦ Adam Optimizer

> combination of SGD with momentum + RMS Prop (Dynamic Learning Rate)

$$w_t = w_{t-1} - \eta' V_{dw}$$

$$V_{dw_t} = \beta * V_{dw_{t-1}} - (1-\beta)\frac{\partial loss}{\partial w_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{S_{dw_t} + \epsilon}}$$

Biased weight updation

$$b_t = b_{t-1} - \eta' V_{db}$$

$$V_{db_t} = \beta * V_{db_{t-1}}$$

$$V_{db_t} = \beta * V_{db_{t-1}} - (1-\beta) \frac{\partial loss}{\partial b_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{S_{db_t} + \epsilon}}$$