NN : Optimizers for NNs-1

Weight Initialization

Why need different Optimizer techniques ?

Ans: When training Deep NN, the model tends to have immense training time due to
- Large number of layers in NN
- Large number of neurons in NN

This makes the parameter (weight matrix) of the NN to be large in size.

What are the issues with Deep NN ?

1. Dead Neuron: When the weight $W = 0$ and bias $b = 0$ for all the layers of the NN

- As activation function is Relu which means :
$$\frac{\partial relu(z)}{\partial x} = 1 \text{ if } Z > 0 \text{ and } \frac{\partial relu(z)}{\partial x} = 0 \text{ if } Z \leq 0$$
- Thus making weight updation to be zero, meaning the NN doesn't learn during training time
- Similarly, when $W = k$ and bias $b = k$ for all the layers of the NN, where k is a constant
- The model acts as a single neuron NN, inspite of there being N number of Neurons.

2. Exploding Gradients
- If the Deep NN uses linear activation function for all of its L layers
$$a = g(z) = z$$
- Then the Weight Matrix = [W^1, W^2, W^3,...... W^{L-1}, W^L], and the final layer output is:
$$\hat{y} = g(W^L \times a^{L-1})$$

As Activation being linear: $g(W^L a^{L-1}) = W^L a^{L-1}$ and $a^{L-1} = g(W^{L-1} a^{L-2})$:
$$\hat{y} = W^L \times g(W^{L-1} \times a^{L-2})$$

Since the NN has layers $\in [1, L]$:
$$\hat{y} = W^L W^{L-1} ..... \times W^2 W^1 X$$

- Now for a Deep NN, L is a large value, which makes $\prod_{i=1}^{L} W^i$ will be a very large value

- Therefore the gradient values becomes exponentially high

How to avoid these issues ?
Weight Initialization strategies

1. Uniform Distribution: We initialize the weights as:

$$w^{k}_{ij} \sim uniform\,[\frac{-1}{\sqrt{fan_{in}}}, \frac{1}{\sqrt{fan_{out}}}\,]$$

- Where fan_{in} is the number of input to a neuron while fan_{out} is the number of output of the neuron

2. Glorot/Xavier init:
   a. Normal Distribution

$$w^{k}_{ij} \sim N(0, \sigma_{ij}),\ where\ \sigma_{ij} \ = \ \sqrt{\frac{2}{fan_{in} + fan_{out}}}]$$

   b. Uniform Distribution

$$w^{k}_{ij} \sim uniform\,[\frac{-\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}\,]$$

   Note: Used when activation function is tanh

3. He Init:
   a. Normal Distribution

$$w^{k}_{ij} \sim N(0, \sigma_{ij}),\ where\ \sigma_{ij} \ = \ \sqrt{\frac{2}{fan_{in}}}]$$

   b. Uniform Distribution

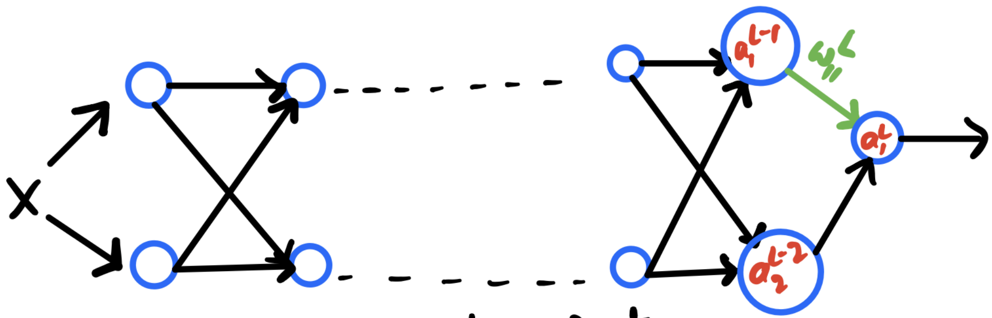
$$w^{k}_{ij} \sim uniform\,[\frac{-\sqrt{6}}{\sqrt{fan_{in}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}}}\,]$$

   Note: Used when activation function is ReLU

Why need to initialize the weight based on input and output of the neuron ?

derivative of Z^{L}_{1} w.r.t $w_{11}^{\ L}$ is defined as:

$$\frac{\partial Z^{L}_{1}}{\partial w_{11}^{\ L}} \ = a^{L-1}_{\quad 1} = \ activation(Z^{L-1}_{\quad 1})$$



$$\frac{\partial Loss}{\partial w_{11}^L} = \frac{\partial Loss}{\partial a_1^L} \, \frac{\partial a_1^L}{\partial z_1^L} \, \underbrace{\frac{\partial z_1^L}{\partial w_{11}^L}}$$

$$\Downarrow$$

$$\left(\frac{\partial \left(w_{11}^L a_1^{L-1} + w_{21}^L \, a_2^{L-1} \right)}{\partial w_{11}^L} \right)$$

$$\Downarrow$$

$$\left(a_1^{L-1} \right)$$

And observe Z^{L-1}_1 is nothing but a function of weights = [W^1, W^2,...... W^{L-1}]

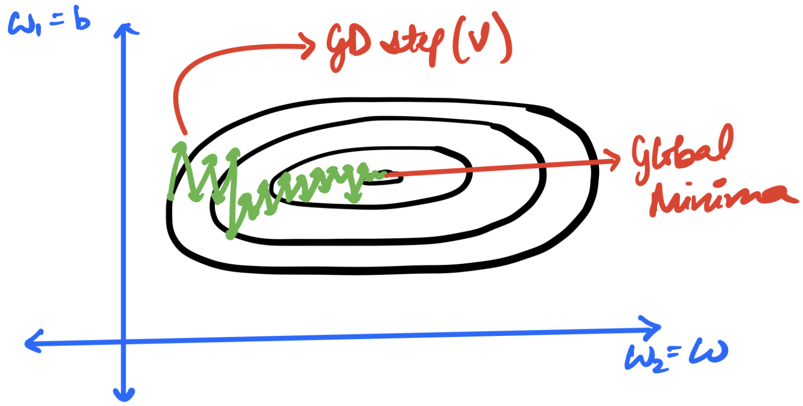
- Hence if Z^{L-1}_1 has greater number of inputs, the value for each weight value is influenced drastically leading to exploding gradient

Optimizer

Why does SGD and Mini Batch Gradient Descent (GD), take so many epochs while training Deep NN ?

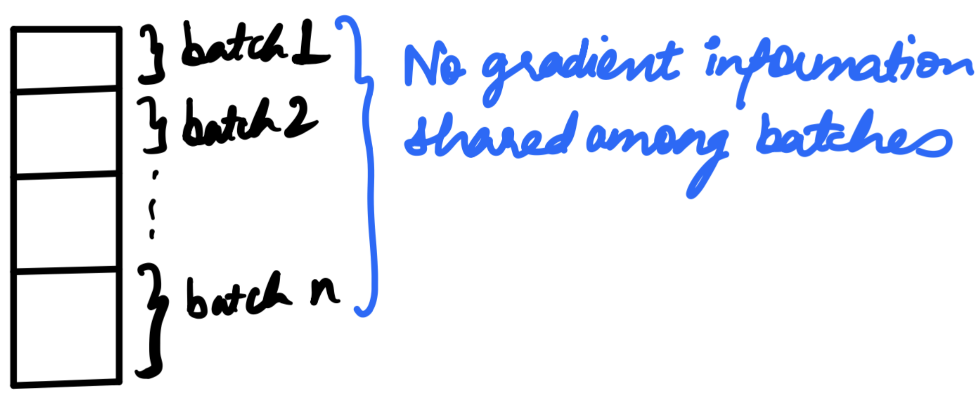
Ans: Mini-batch GD takes steps (V) where the GD tends to:

- Move in direction where it will never reach minima
- Hence due to all these noisy steps, the GD takes so many epochs



Why does mini-Batch GD have noisy steps ?

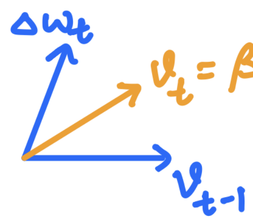
Ans: Because, training data is divided into batches
- And for some batch the model has very small loss
- while for a few batch, the loss is quite high
- Making the gradients of weights fluctuate



How to reduce these noisy steps ?

Ans: By taking some weighted average (β) from the previous Optimizer Step (V_{t-1}) along with the current gradient (Δw) , hence:

$$V_t = \beta V_{t-1} + (1 - \beta)\Delta w_t$$



$$v_t = \beta\, v_{t-1} + (1-\beta)\, \Delta w_t$$

current gradient

previous gradient step

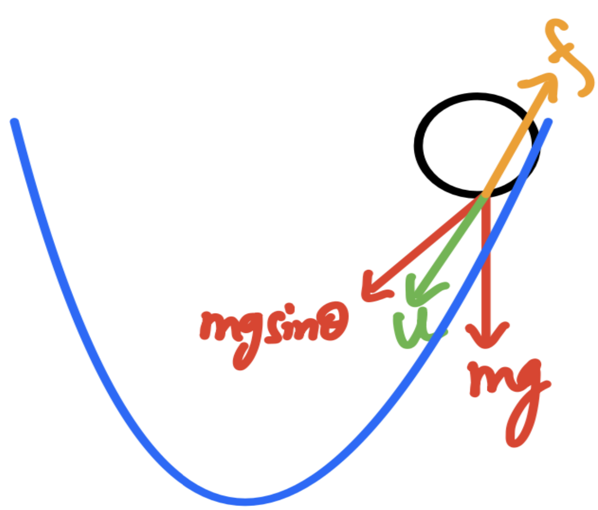
Note: t denotes t^{th} iteration, where 1 iteration = Forward + Backward Propagation

With the weightage (β) being introduced,

- The direction of V_3 will tend to be influenced by all previous gradients ΔW_1, ΔW_2 along with the current gradient ΔW_3
- Thus making the optimizer to take a step in the direction such that it avoids the noisy step
- This is known as Exponential Moving Average

This idea of Exponential Moving Average can be considered as a ball moving down a hill thus:

- $\beta \rightarrow$ Friction
- $V_{t-1} \rightarrow$ Velocity/Momentum
- $\Delta w_t \rightarrow$ Acceleration



How does Gradient Descent implement Exponential Moving Average ?

Ans: for some iteration t and layer k of the NN :

- We find the gradients dw^k, db^k

the exponential moving average is introduced as:

$$V_{dw^k} = \beta \times V_{dw^k} + (1 - \beta) \times dw^k$$

Similarly:

$$V_{db^k} = \beta \times V_{db^k} + (1 - \beta) \times db^k$$

Hence Weight updation with learning rate α becomes:

$$w^k = w^k - \alpha \times V_{dw^k}$$

$$b^k = b^k - \alpha \times V_{db^k}$$

Note: This Optimizer is called Gradient Descent with Momentum

How to further reduce the oscillations of Gradient Descent ?

Ans: optimizer tends to move in direction (oscillations) when gradient of one weight is greater than the other

- Meaning Δb >>> Δw

Hence to reduce this moving direction:

$$V_{dw^k} = \beta V_{dw^k} + (1 - \beta)(dw^k)^2$$

$$V_{db^k} = \beta V_{db^k} + (1 - \beta)(db^k)^2$$

And weight updation becomes:

$$w^k = w^k - \alpha \times \frac{dw^k}{\sqrt{V_{dw^k} + \epsilon}}$$

$$b^k = b^k - \alpha \times \frac{db^k}{\sqrt{V_{db^k} + \epsilon}}$$

where ϵ is a very small value = 10^{-8}

How is squaring useful?
Ans: as gradients in which the optimizer moves is higher then:

- the square of the gradient will be much high
- thus making $V_{db^k} > V_{dw^k}$ and $\frac{1}{V_{db^k}} < \frac{1}{V_{dw^k}}$
- Therefore after weight updation, w^k reaches optimal value faster

Note: This approach is known as RMSprop

Is there a way to combine both RMSprop's decreased oscillation and Momentum fast convergence?
Ans: This is done by Adam which defines

- Momentum:

$$V_{dw^k} = \beta_1 V_{dw^k} + (1 - \beta_1)\, dw^k$$
$$V_{db^k} = \beta_1 V_{db^k} + (1 - \beta_1)\, db^k$$

- RMSprop:

$$S_{dw^k} = \beta_2 S_{dw^k} + (1 - \beta_2)\, (dw^k)^2$$
$$S_{db^k} = \beta_2 S_{db^k} + (1 - \beta_2)(db^k)^2$$

Now both in RMSprop and Momentum, the initial averaged-out values are biased,

- so to kickstart the algorithm Biasness correction is done such that:

$$V_{dw^k}^{Corrected} = \frac{V_{dw^k}}{1-\beta_1^{\ t}}$$
$$V_{db^k}^{Corrected} = \frac{V_{db^k}}{1-\beta_1^{\ t}}$$

$$S_{dw^k}^{Corrected} = \frac{S_{dw^k}}{1-\beta_2^{\ t}}$$
$$S_{db^k}^{Corrected} = \frac{S_{db^k}}{1-\beta_2^{\ t}}$$

Therefore Weight updation becomes:

$$w^k = w^k - \alpha \times \frac{V_{dw^k}^{Corrected}}{\sqrt{S_{dw^k}^{Corrected} + \epsilon}}$$
$$b^k = b^k - \alpha \times \frac{V_{db^k}^{Corrected}}{\sqrt{S_{db^k}^{Corrected} + \epsilon}}$$