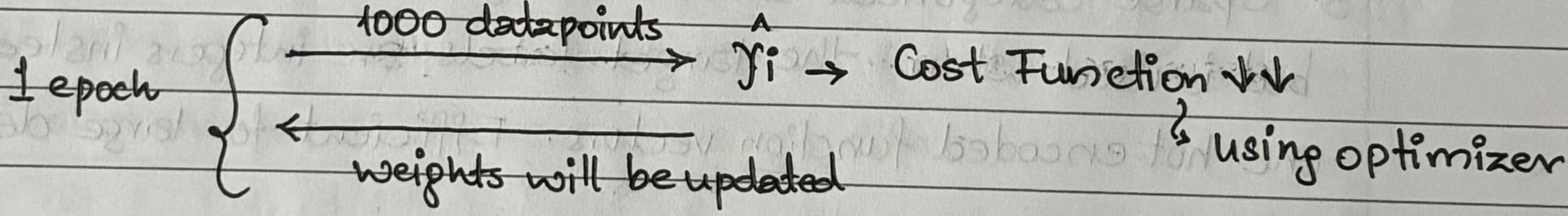
⑦ *Optimizers*

An optimizer improves the model by adjusting it's parameters (weights and biases) to minimize the loss function value.
The optimizers role is to find the best combination of weights and biases that leads to the most accurate predictions.

→ Epochs: One complete pass of the entire training dataset through the learning algorithm



[1 epoch = 1 iteration]

1. Gradient Descent: It works by iteratively adjusting the model parameters in the direction that minimises the loss functi

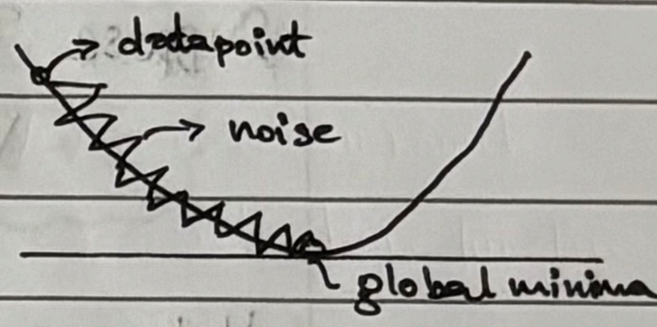
⇒ Advantages: Convergance will happen

⇒ Disadvantages: High Resource (RAM, GPU) needed

__/__/____

2. Stochastic Gradient Descent : SGD updates the model parameters after each training example, meking it more efficient for large datasets compared to traditional Gradint Descent which uses the entire dataset for each update.

Epoch 1 Iteration 1 —1 datapoint→ $\hat{y}_i$ → Cost function ↓
↳ n iterations : : ←weight updation
Iteration n



⇒ Advantages: It solves resource issue

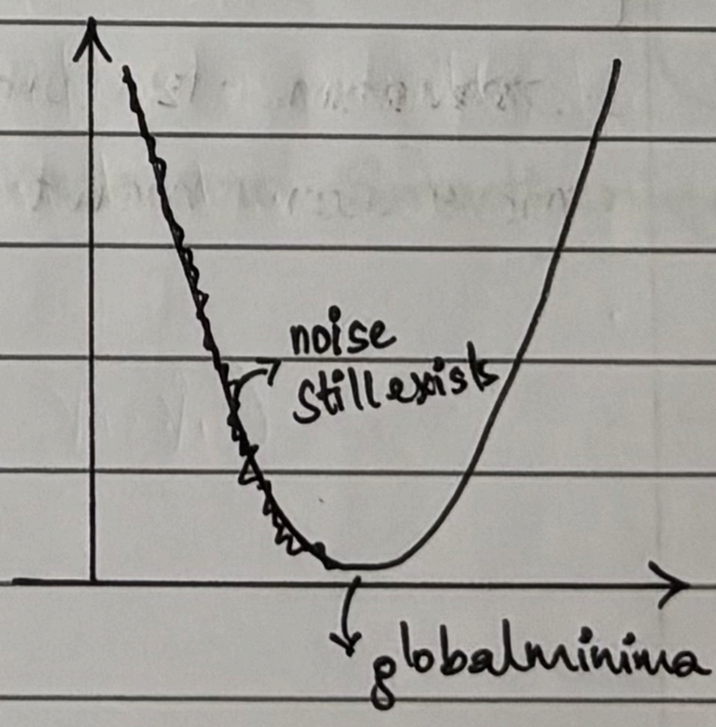
⇒ Disadvantages: 1. Time Complexity increases, convergance will also take time
2. Noise gets introduced

3. Mini Batch Gradient Descent : consicts of predetermined number of training examples. smaller than the full dataset. This approach combines the advantages and features of the previous variants.

Datapoints = 100,000 Batch Size = 1000
No of iterations - Datapoints / BatchSize = 100,000 / 1000 = 100 iteration

(1 Epoch = No of Itervations) → Noise will be reduced

⇒ Disadvantages: Noise still exists



__/__/____

4. Stochastic Gradient Descent with Momentum:

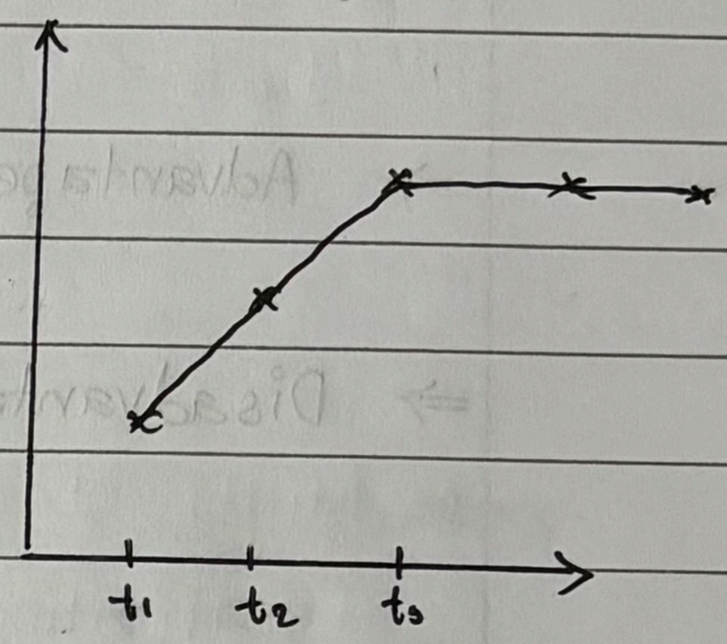
Momentum helps accelerate convergence by smoothing out the noisy gradients of SGD, thus reducing fluctuations and improving speed of convergence.

→ Exponential Weight Average {Smoothening}

Suppose Time = t_1 t_2 t_3 t_4 ... t_n

Values = a_1 a_2 a_3 a_4 ... a_n

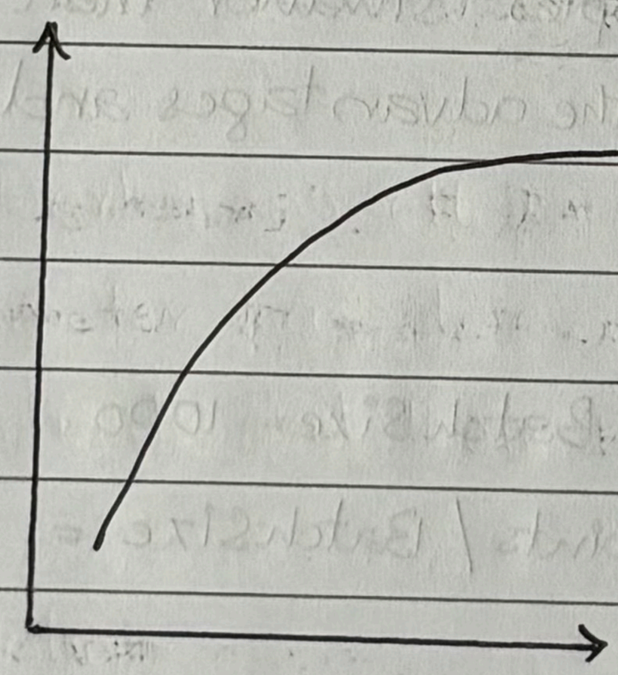
$Value_{t_1} = a_1$, $Value_{t_2} = a_2$ ...



If we introduce a term β, we can smoothen this.

$\therefore \boxed{V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_{t_2}}$ $\quad \beta = [0-1]$

If we want to smoothen w.r.t the previous value give $\beta \uparrow$ value.



As we can see the curve got smoothened.

$$\boxed{w_t = w_{t-1} - \eta \frac{\partial Loss}{\partial w_{t-1}}}$$

__/__/____

5. AdaGrad Optimizer: AdaGrad adapts the learning rate for each parameter based on the historical gradient information. The learning rate decreases over time, making AdaGrad effective for sparse features.

→ As the convergence happens the learning rate should change.

$$w_t = w_{t-1} - \eta' \frac{\partial loss}{\partial w_{t-1}} \quad , \quad \eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

η' = dynamic learning rate , ϵ = small value

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

As $\alpha_t \uparrow\uparrow \rightarrow \eta' \downarrow\downarrow$

→ Disadvantage: η', there is a possibility to become a very small value ≈ 0

6. AdaDelta and RMS PROP : Improves upon AdaGrad by introducing a decay factor to prevent the learning rate from decreasing too rapidly.

Here $\eta' = \frac{\eta}{\sqrt{Sdw_t + \epsilon}}$, Sdw_t → Exponential Weight Average

$$Sdw_t = \beta * Sdw_{t-1} + (1-\beta) \left(\frac{\partial loss}{\partial w_{t-1}} \right)^2$$

It handles both dynamic learning rate and Smoothening EWA.

__/__/____

7. Adam Optimizer : Adaptive Moment Estimation optimizer combines the advantages of Momentum and RMSProp techniques to adjust learning rates during training. It works well for large datasets and complex models because it uses memory efficiently and adapts the learning rate for each parameter automatically.

$$[w_t = w_{t-1} - \eta' V_{dw}], \quad [b_t = b_{t-1} - \eta' V_{db}]$$

↳ weight updation ↳ bias updation

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

↳ dynamic learning ↳ smoothening

$$\begin{cases} V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta)\dfrac{\partial loss}{\partial w_{t-1}} \\ V_{db_t} = \beta * V_{db_{t-1}} + (1-\beta)\dfrac{\partial loss}{\partial b_{t-1}} \end{cases} \Rightarrow \text{Momentum (smoothening weight updates)}$$

→ Why Adam works so well?

1. Each parameter has its own dynamic learning rate based on past gradients and their magnitudes.

2. Bias correction

3. Adam typically requires fewer hyperparameter tuning adjustments compared to other optimization algos making it more convenient choice for most problems.