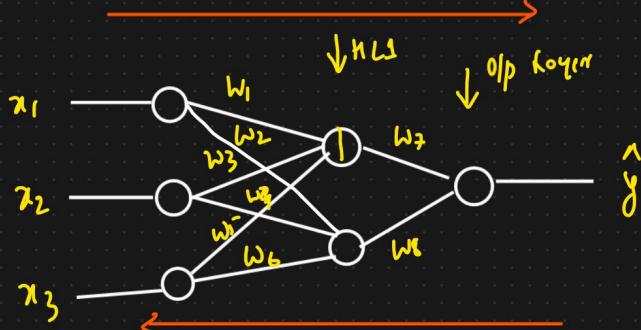


Optimizers → used for reducing cost function  
or reach Global minima.

- ① Gradient Descent
  - ② Stochastic Gradient Descent (SGD)
  - ③ Mini batch SGD
  - ④ SGD With Momentum
  - ⑤ Adagrad and RMS Prop
  - ⑥ Adam Optimizers
- Epoch → completed iteration i.e. | Forward P.  
| Backward P.
- Best of both worlds  
but bit noisy

### Gradient Descent Optimizer



$$\text{loss} = [ \quad ] \downarrow \downarrow \downarrow$$

Optimizers ↑

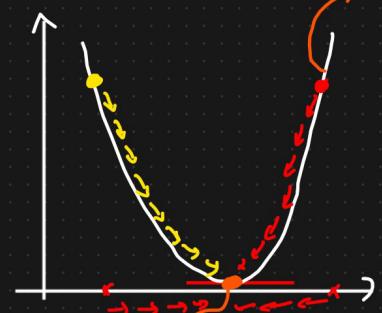
loss or cost

Gradient Descent

loss ↓ ↓

$w_{\text{new}} \approx w_{\text{old}}$

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial h}{\partial w_{\text{old}}}$$



MSE

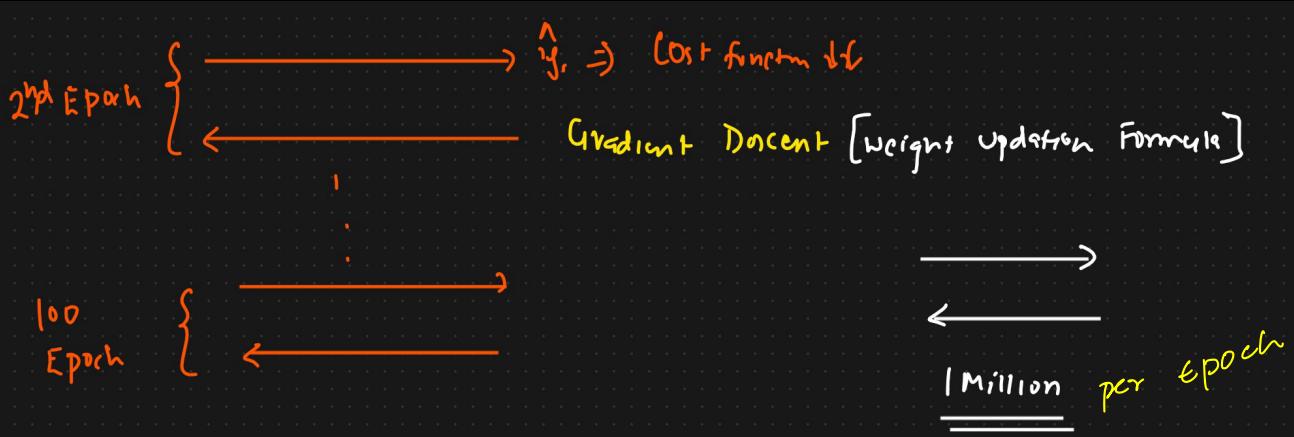
$$\text{Loss fn} = (y - \hat{y})^2 \quad \text{Cost fn} = \frac{1}{h} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

DataSet = 1000 Data Points

Epochs, Iteration  
1 Epoch = 1 Iteration  
1 Epoch →  $\hat{y}_i \Rightarrow$  Cost function w.  
Weights will get updated

Gradient Descent optimizer [Weight update]

10 Iteration  
 $\frac{100}{10} = 100$   
100 →  
←  
←  
←  
←  
←



### Advantages

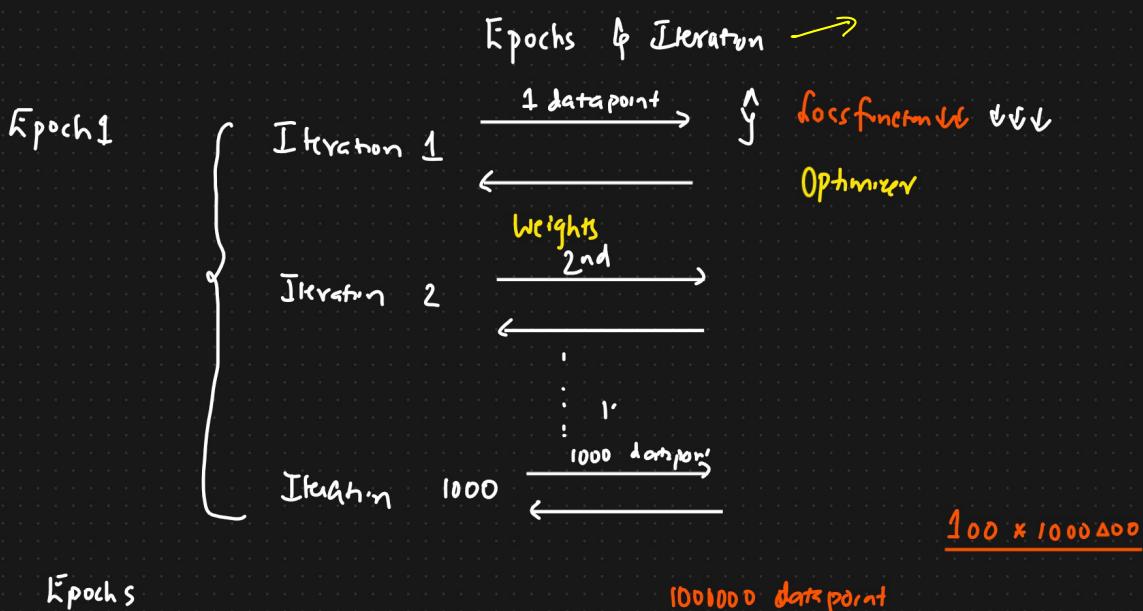
- ① Convergence will happen.
  - Accurate Gradients
  - Good for Small Datasets
- ② Stochastic Gradient Descent (SGD)

### Disadvantage

- ① Huge Resource RAM, GPU

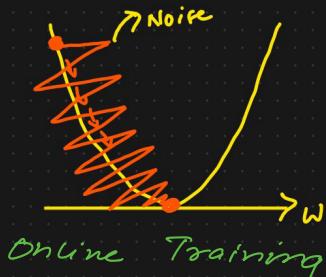
Resource Intensive -

1000 datapoint



### Advantage

- ① Solve Resource Issue



### Disadvantage

- ② Time Complexity  $\uparrow$

- ③ Convergence will also take More time.
- ④ Noise gets Introduced

→ may skip Global minima.

Cost

### ③ Mini Batch SGD

*Batch size*

$$\text{No. of iterations} = \frac{100000}{1000} = 100 \text{ iterations}$$

Epoch, Iteration, Batch-size

Data points = 100000

batch.size = 1000

MSG

$$\text{Cost fn} = \sum_{i=1}^{1000} (y_i - \hat{y}_i)^2$$

Epoch 1

Iteration 1

Optimizer  $\Rightarrow$  Mini Batch SGD

change the weight

1000

↓↓

[8gb]

1

Iteration 2

[16gb]

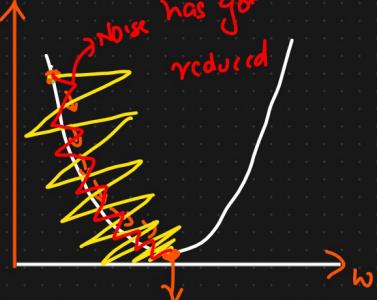
5000

Iteration 3

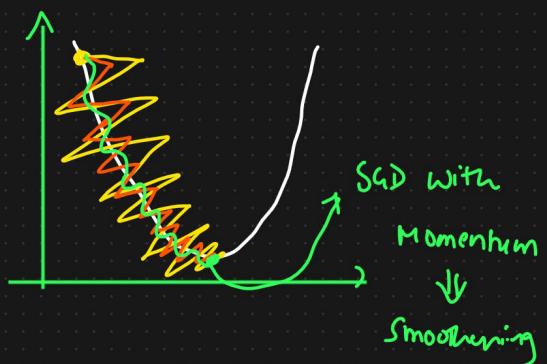
⋮

Iteration 100

Cost function



Batch Size



#### Advantages

- ① Convergence speed will increase
- ② Noise will be less when compared to SGD
- ③ Efficient Resource Usage (RAM)

#### Disadvantage

- ① Noise still exists

in every epoch of 1000 iterations  
1000 DP ob

# CONCLUSION

Optimizers = Reduce cost function

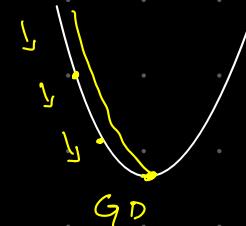
working



## 1. Batch Gradient Descent (GD)

You taste all 100 cakes at once, take notes, average out all your feedback, and then make one big adjustment to the recipe.

- One big decision after thorough review
- Takes time (you must finish all cakes)
- Very precise, but slow
- ✗ 1 update per round (epoch)
- ✗ Uses all 100 cakes each time



## 2. Stochastic Gradient Descent (SGD)

You're in a hurry — you taste one cake, make a quick adjustment. Then you taste the next cake, make another tweak.

- Fast but very chaotic
- Some tweaks may help, others might hurt
- But over time, you may stumble upon the perfect recipe
- ✗ 100 updates per round (epoch)
- ✗ Uses 1 cake at a time, very noisy

### Summary (Cake Edition):

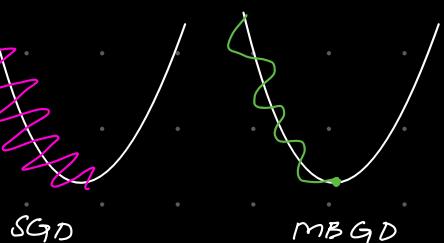
Method	How You Taste	How You Adjust	Feeling
GD	Taste all 100	One big tweak	Slow but sure
SGD	Taste 1 cake	Constant tweaking	Fast but shaky
Mini-Batch	Taste 10 at a time	Small steady tweaks	Best of both worlds



## 3. Mini-Batch Gradient Descent (batch size = 10)

You group the 100 cakes into 10 plates (batches) of 10 cakes each. You taste a plate, make a small adjustment, and repeat.

- Balanced speed
- Not too noisy, not too slow
- Gets better with practice
- ✗ 10 updates per round (epoch)
- ✗ Each update uses 10 cakes



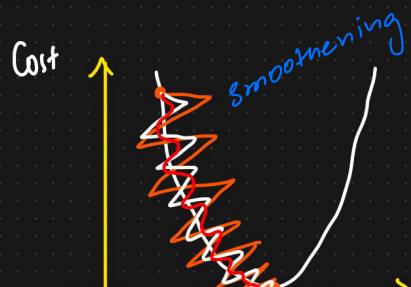
EXPLANATION ;

### Gradient Descent Variants Explained

Type	How it Works	Speed	Stability	Used When	Pros	Cons
Batch Gradient Descent	Uses the entire dataset to compute gradient before updating weights.	✗ Slow (per update)	✓ Very stable	Small datasets	+ Accurate gradients + Converges smoothly	- Very slow on large data - High memory usage
Stochastic GD (SGD)	Uses 1 data point at a time to update weights.	✓ Fast (per update)	✗ Very noisy	Large datasets, online learning	+ Fast updates + Good for huge/streaming data + Can escape local minima	- Noisy convergence - May overshoot or bounce around
Mini-Batch SGD	Uses a small subset of data (e.g., 32, 64) for each update.	✗ Balanced	✓ Smooth + efficient	Most common in deep learning	+ Fast and stable + Efficient on GPUs + Generalization-friendly	- Needs tuning of batch size - Still a bit noisy

we need Momentum biased SGD to smooth noise (exponential weighted average)

#### (4) SGD With Momentum



Weight Updation formula

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

$$b_{\text{new}} = b_{\text{old}} - \eta \left[ \frac{\partial h}{\partial b_{\text{old}}} \right]$$

$$w_t = w_{t-1} - \eta \left[ \frac{\partial h}{\partial w_{t-1}} \right]$$

Exponential Weight Average {Smoothing}  $\Rightarrow$  ARIMA, SARIMAX

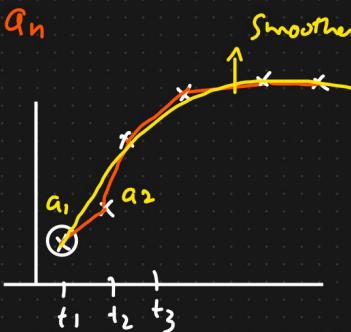
Time =  $t_1 \ t_2 \ t_3 \ t_4 \ \dots \ t_n$  Time Series

Values =  $a_1 \ a_2 \ a_3 \ a_4 \ \dots \ a_n$

$$V_{t_1} = a_1$$

$$\beta = 0.95$$

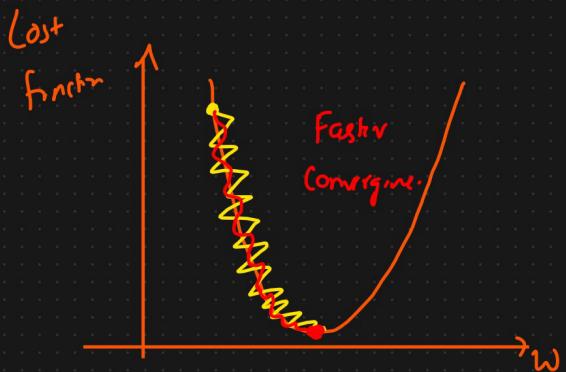
$$V_{t_2} = \boxed{\beta} * V_{t_1} + (1-\beta) * a_2$$



$$\underline{\beta = 0.95} = \underline{0.95 * a_1 + (0.05) a_2}$$

$$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$$

$$= 0.95 \left[ 0.95 * a_1 + (0.05) a_2 \right] + (0.05) * a_3$$



#### Advantage

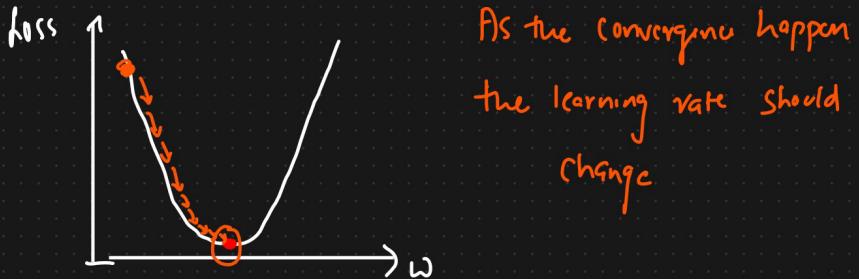
- ① Reduces the noise
- ② Quick Convergence

⑤ Adagrad = Adaptive Gradient Descent

$\eta = \text{fixed} \Rightarrow \text{Dynamic learning}$

$$w_t = w_{t-1} - \eta \left[ \frac{\partial h}{\partial w_{t-1}} \right]$$

Learning Rate = 0.001



$0.00001 \approx 0$

$$w_t = w_{t-1} - \eta' \left[ \frac{\partial h}{\partial w_{t-1}} \right]$$

$$\eta' = \frac{\eta}{\sqrt{d_t + \epsilon}}$$

$d_t \uparrow \uparrow$  Epsilon  $\approx$  small value

$t=1 \quad t=2 \quad t=3 \quad w_t \approx w_{t-1}$

$$d_t = \sum_{i=1}^t \left( \frac{\partial h}{\partial w_i} \right)^2$$

$\eta = 0.01 \quad \eta = 0.005 \quad \eta = 0.003$

### Disadvantage

- ①  $\eta' \rightarrow$  Possibility to become a very small value  $\approx 0$

## ⑥ Adadelta And RMSprop

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

$\stackrel{=}{t}$

Exponential Weight Average

$$\beta = 0.95 \quad S_{dw_t} = 0$$

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

[Dynamic LR + Smoothening (EWA)]

$$w_t = w_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

## ⑦ Adam Optimizer

SGD with Momentum + RMSprop [Dynamic LR + Smoothening]

$$w_t = w_{t-1} - \eta' V_{dw} \quad \text{Weight Update}$$

$$b_t = b_{t-1} - \eta' V_{db} \quad \text{Bias Update}$$

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

EWA

$$S_{dw_t} = 0$$

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

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

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

$\Rightarrow$  Momentum  
 $\hookrightarrow$  Smoothening

# Final Conclusion

## ✓ Gradient Descent Variants & Optimizers — Full Comparison

Type	How it Works	Speed	Stability	Used When	Pros	Cons
Batch Gradient Descent	Uses the <b>entire dataset</b> to compute gradient before each weight update.	✗ Slow (per update)	✓ Very stable	Small datasets	+ Accurate gradients + Smooth convergence	– Very slow on large data – High memory usage
Stochastic GD (SGD)	Uses <b>1 data point</b> at a time to compute and apply updates.	✓ Fast (per update)	✗ Very noisy	Large datasets, online learning	+ Very fast updates + Good for real-time/streaming + Can escape local minima	– Noisy updates – May bounce or overshoot
Mini-Batch SGD	Uses a <b>small batch</b> (e.g., 32–128 samples) per update.	⚖ Balanced	✓ Smooth + efficient	Deep learning, GPU training	+ Combines speed & stability + Efficient on hardware + Improves generalization	– Needs batch size tuning – Some noise still persists
SGD with Momentum	Like Mini-Batch SGD, but adds <b>momentum</b> : a fraction of past updates to current updates — helps <b>smooth curves</b> and accelerate convergence.	✓ Faster than SGD	✓ More stable	Deep learning, non-convex optimization	+ Speeds up convergence + Reduces oscillation + Helps escape flat regions (plateaus)	– Needs momentum tuning – Risk of overshooting
AdaGrad	Adapts learning rate <b>per parameter</b> based on past squared gradients — good for sparse data.	⚡ Fast initially	⚖ Decent	Sparse features, NLP tasks	+ No LR tuning needed + Great for sparse data like text (NLP)	– Learning rate decays too fast – Stops learning early
Adam Optimizer	Combines <b>momentum + adaptive learning rate</b> per parameter using moving averages of gradients.	✓ Fast & efficient	✓ Very stable	Deep learning (default choice)	+ Fast convergence + Works well out-of-the-box + Handles sparse & noisy gradients	– May overfit – Sensitive to hyperparameters – May not generalize best

## 🧠 TL;DR:

Situation	Best Choice
Small, simple dataset	<b>Batch GD</b>
Real-time or massive streaming data	<i>Online Train</i> <b>SGD</b>
General deep learning tasks	<b>Mini-Batch SGD / Adam</b>
Complex loss surfaces, faster training	<i>Less Noise</i> <b>SGD with Momentum</b>
NLP or sparse data	<b>AdaGrad / Adam</b>

Imp. Adam optimizer = Best optimizer (moving avg of gradients).  
 Combine Adaptive Learning Rate + momentum