Optimization in Deep Learning

10/10/22.

- GD
- SGD
- Minibatch SGD
- SGD with momentum
- Adagrad (Adaptive stochastic Gradient)
- RMS prop
- Adadelta
- Adam.

GD

- Takes whole data in 1 iteration

epoch = epoch-1
iter-1

iter = iter-1 sample

Considers ~~each~~ all samples as single unit.

SGD

- Say we have 100 samples, then in (each sample individualy)

epoch-1

iter 1 1st sample $\underset{\text{backward}}{\overset{\text{forward}}{\rightleftarrows}}$ prop / backward prop

iter 2 2nd sample $\rightleftarrows$

⋮

iter 100 100th sample $\rightleftarrows$

Minibatch SGradient descent

total = 100 samples

batch size = 10

no. of ~~samples for each~~ batches = $\frac{100}{10}$ = 10

iter = 10

epoch1 10 iters

1st iter 1 to 10

2nd iter 11 to 20

3rd iter 21 to 30

⋮

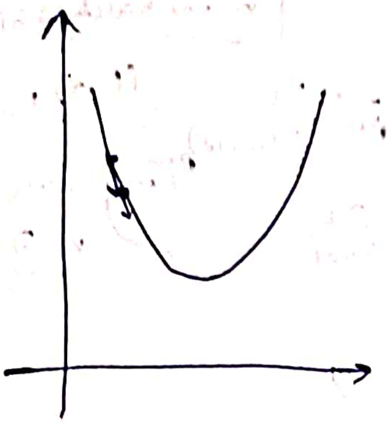
10th iter 91 to 100

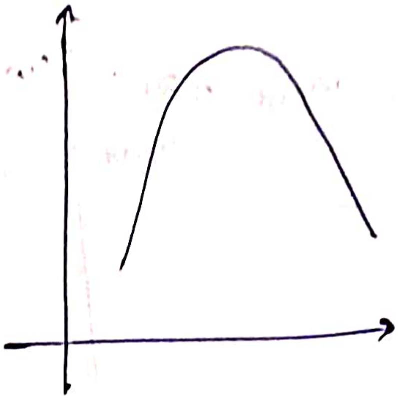
---

disadv:-

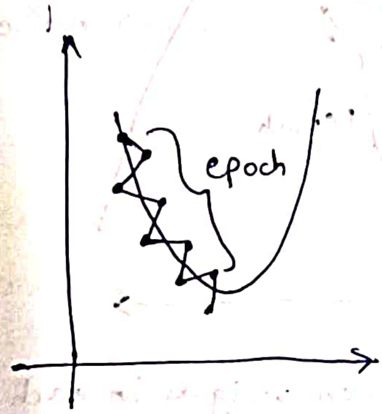
GD:
- more RAM
- computationally expensive

SGD:
- It will take less RAM
- But still computationally expensive.

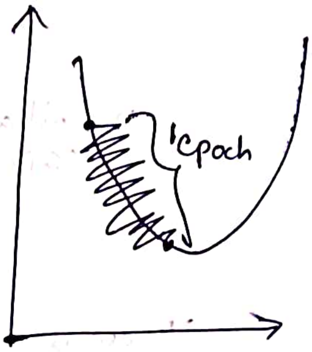




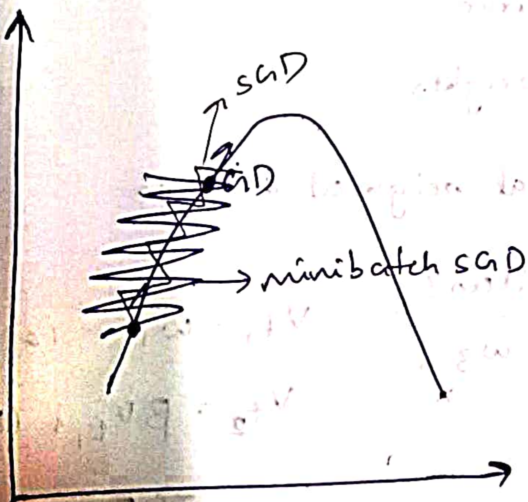
(For GD we get only one value)



For SGD, for 1 epoch, it will go in zig-zag fashion



more zigzag in minibatch SGD.



minibatch SGD (takes extreme left/right)
doesnot give a smooth curve like

(in all 3)

→ Disadv for all three :- It doesnot give optimum path to reach the goal.

GD

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

SGD

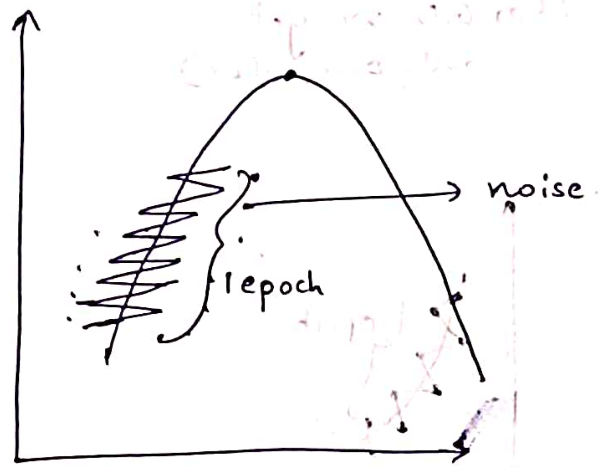
$$(w_{new})_{iter} = (w_{old})_{prev\ iter} - \eta \frac{\partial L}{\partial (w_{old})_{prev\ iter}}$$

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

minibatch SGD

$$((w)_{batch})_t = (w_{batch})_{t-1} - \eta \frac{\partial L}{(\partial w_{batch})_{t-1}}$$

minibatch SGD



There is no relation between weights in each batch
So, it is giving steep values

SGD with momentum

→ This gives a smooth curve

→ It considers the new weights

→ It calculates exponential weighted average

item1	item2	item3
w_1	w_2	w_3

$V_{t_1} = w_1$ (a1)

$V_{t_2} = \beta V_{t_1} + (1-\beta) w_2$ (a2)

If $\beta = 0.9$,
90% importance to previous iteration and 10% to current iteration.

equivalent to physics acceleration.

$V_{t_3} = \beta V_{t_2} + (1-\beta) w_3$ (a3)

$= \beta(\beta V_{t_1} + (1-\beta) w_2) + (1-\beta) w_3$

For SGD, $w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$

For SGD with momentum, $w_t = w_{t-1} - \eta \, V_{dw_t}$ → This term is derivative of V_{t_i}

$$V_{dw_1} = \frac{\partial V}{\partial w_t}(w_1)$$

$$V_{dw_2} = \beta V_{\partial w_1} + (1-\beta)\frac{\partial L}{(\partial w_2)}$$

For iteration 1, $w_1 = w_0 - \eta V_{dw_1}$

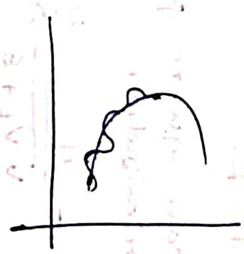
For iteration2, $w_2 = w_1 - \eta V_{dw_2}$

$$V_{dw_3} = \beta V_{dw_2} + (1-\beta)\frac{\partial L}{\partial w_3}$$

→ SGD is independent of previous weights

→ SGD with momentum gives importance to previous weights (90%) and remaining 10% to current weight.

SGD with momentum gives smooth curve



11/10/22

- Adagrad
- RMS prop
- Adadelta
- Adam.

Adagrad

- Adaptive gradient descent

→ η is dynamic

At each iteration, if we change learning rate, it will give a better result.

→ So, works better than SGD with momentum

$$w_t = w_{t-1} - \eta_t \left(\frac{\partial L}{\partial w_{t-1}}\right)$$

$$\eta_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}} \quad \varepsilon?$$

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_i}\right)^2 \longrightarrow$$ cumulative sum of current plus previous values

e.g.:- if $t=3$,

$$\alpha_t = \left(\frac{\partial L}{\partial w_1}\right)^2 + \left(\frac{\partial L}{\partial w_2}\right)^2 + \left(\frac{\partial L}{\partial w_3}\right)^2$$

RMS prop

→ Root mean square propagation optimization.

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{V_t + \varepsilon}} \frac{\partial L}{\partial w_{t-1}}$$

$$V_t = \beta V_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_t}\right)^2$$

→ improvement to adagrad.

→ Instead of taking cumulative sum, it takes exponential weighted average.

Adadelta

→ Adaptive delta optimization

→ η term is completely removed and delta term is added.

$$w_t = w_{t-1} - \frac{\sqrt{D_{t-1} + \varepsilon}}{\sqrt{V_t + \varepsilon}} \times (\Delta w_t)^2$$

$$\Delta w_t = w_t - w_{t-1}$$

$$D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta w_t)^2$$

$$V_t = \beta_2 V_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

Adam

→ adaptive moment estimation

→ momentum + RMS prop (combines both)

$$w_t = w_{t-1} - \frac{\eta * \overbrace{V_{dw_t}}^{\text{momentum}}}{\sqrt{\underbrace{S_{dw_t}}_{\text{RMS prop}} + \epsilon}}$$

$$V_{dw_t} = \beta_1 V_{dw_{t-1}} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

↑ (momentum)

$$S_{dw_t} = \beta_2 S_{dw_{t-1}} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

↳ RMS prop

This technique is further improved by bias correction

$$\cancel{w_t = w_{t-1} - \frac{\eta * V_{dw_t} / (1-\beta_1)^t}{\sqrt{S}}}$$

$$w_t = w_{t-1} - \frac{\dfrac{\eta * V_{dw_t}}{(1-\beta_1)^t}}{\sqrt{\dfrac{S_{dw_t}}{(1-\beta_2)^t} + \epsilon}}$$

$$w_t = w_{t-1} - \frac{\eta * V_{dw_t}}{\sqrt{S_{dw_t} + \epsilon}}$$

?

$$V_{dw_t} = \frac{\beta_1}{(1-\beta_1)} V_{dw_t} + \frac{\partial L}{\partial w_t}$$

$$S_{dw_t} = \frac{\beta_2}{(1-\beta_2)} S_{dw_t} + \left(\frac{\partial L}{\partial w_t}\right)^2$$

Disadvantage

1) It may suffer from vanishing gradient problem

$$\eta_t' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

Vanishing gradient ⇒ $\alpha_t \approx \alpha_{t-1}$

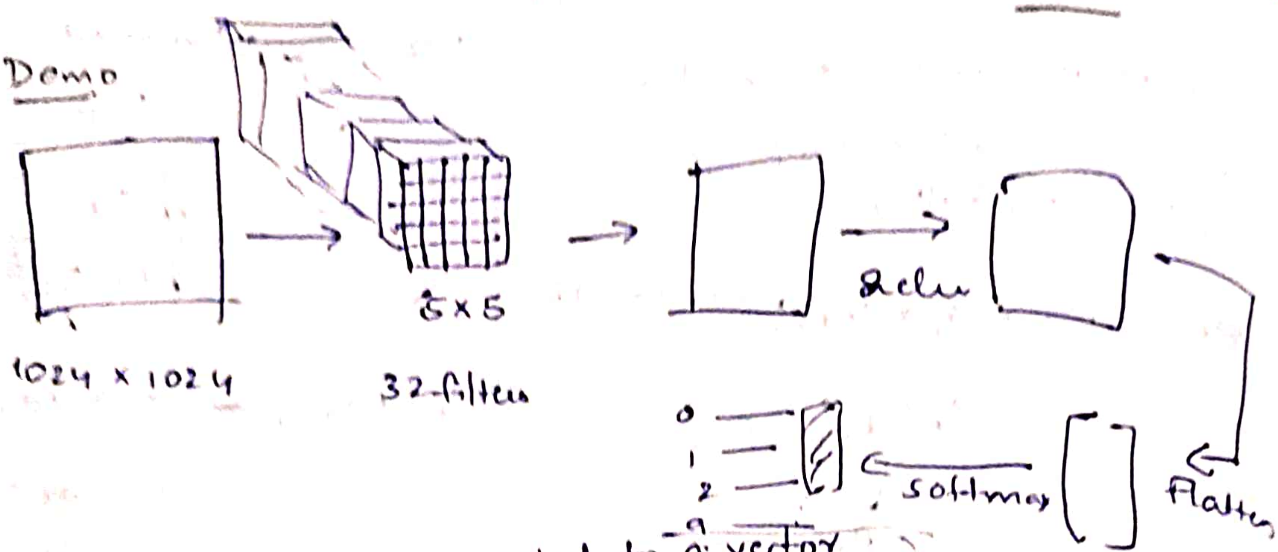
⇒ $\eta_t = \eta_{t-1}$

advantage

1) It ~~may~~ provides better convergence than SGD with momentum.

14/10

Demo



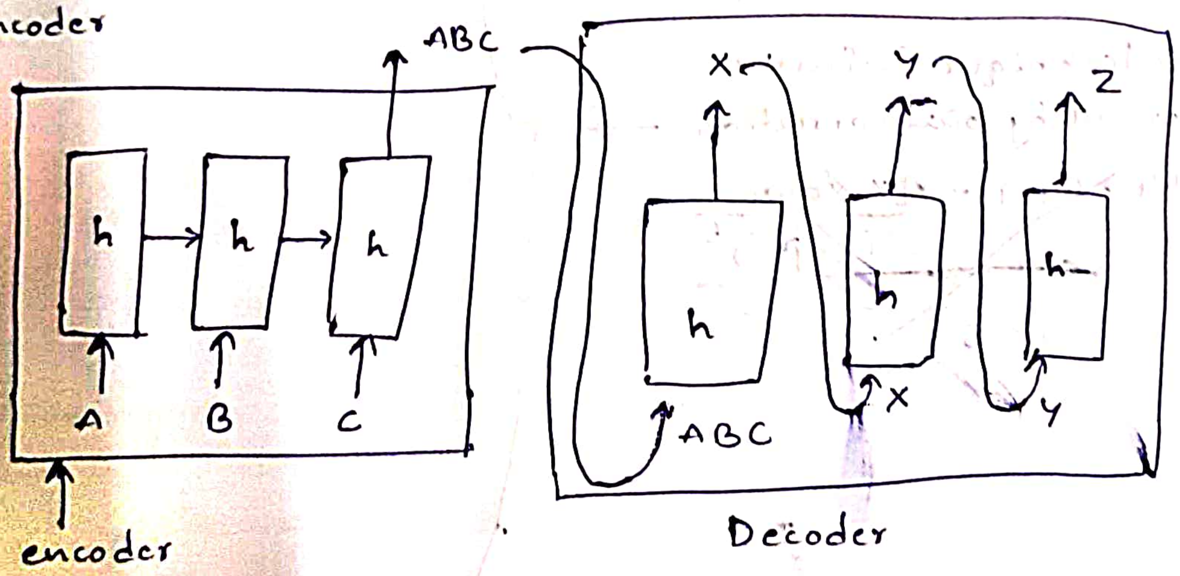
-flatten:- matrix converted to a vector

Densenet:- Each and every neuron is connected with every other (fully connected)

conv 2D - grayscale image

18/10

Encoder-Decoder RNN



ABC → XYZ (translation)

-ABC → WXY (It might not translate accurately)

Incorrect translations → noise

(RNN-1
LSTM-3X

Encoder Decoder LSTM (is also there).

||UNIT-3||

Probabilistic models of the input $P_{model}(x)$

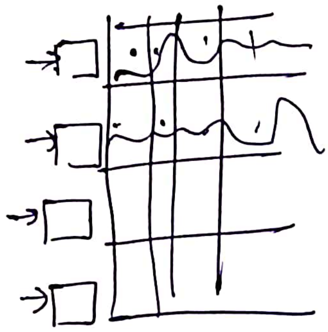
For example, to rate mess food,

- → ambience
- → arrangement } service
- → cleanliness
- → quantity of food
- → quality } food
- → variety in menu

(Reduction in meaningful less features no. of)

In ML, we do this using dimensionality reduction

— Get 4 different signals



convert into discrete values, extract individual signals.

From the observed variable, getting non-observed variable — latent variables.

$$P_{model}(x) = E_h P_{model}(x|h)$$

Latent variables: applns in economics, ...

Linear factor models

→ Data generation in LFMs:

1. Sample the explanatory