OPTIMISATION ALGORITHMS

Video 1: Mini batch gradient descent

* If m = ~~500~~ 50 million then training such a huge dataset is difficult and applying gradient descent makes the algorithm slower

* Solution : Splitting m to mini batches of size 1000, and applying this to X & Y

* $$\underset{(n_x,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)} \dots}_{X^{\{2\}}\ (n_x,1000)}]$$

$$\underset{(1,m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \dots]$$

minibatch t : $X^{\{t\}}, Y^{\{t\}}$

↑

we run gradient descent on these data sets. Thus, we can calculate it for 5 million examples at same time using vectorisation

Algorithm:

epoch - 1 pass thru set

repeat {

for $t = 1, \ldots, 5000$ {

forward prop on $X^{\{t\}}$

$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X^{\{t\}} + b^{[1]} \\ A^{[1]} &= g^{[1]} Z^{[1]} \\ &\vdots \\ A^{[l]} &= g^{[l]} Z^{[l]} \end{aligned}\right\} \text{vectorised implementation}$$

compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum \|w^{[l]}\|_F^2$

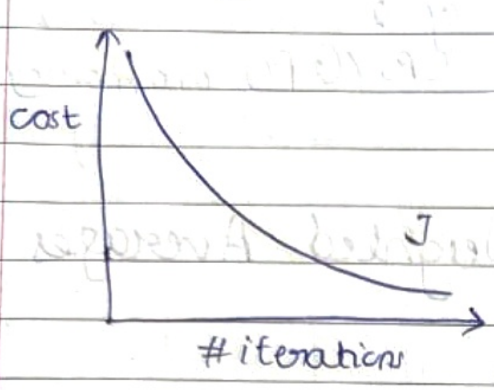
backprop to compute gradient w.r.t $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$

$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$
$b^{[l]} = b^{[l]} - \alpha db^{[l]}$

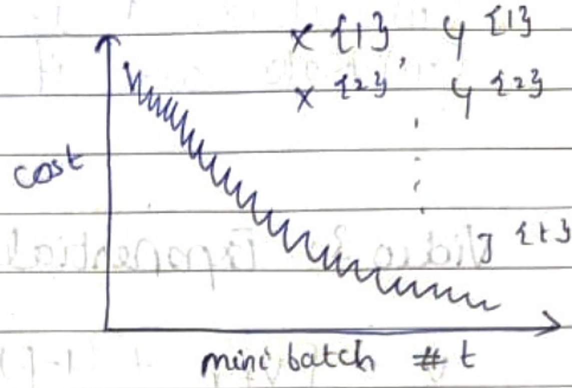
}
}

Video 2: Training with mini batch gradient descent

Batch gradient descent



Mini batch gradient descent



Choosing your mini batch size:

mini batch size: (m) → batch gradient descent ← too long

(1) → stochastic gradient descent

between (1) & (m) → mini batch gradient descent

→ too noisy
lose speedup
won't reach min. cost

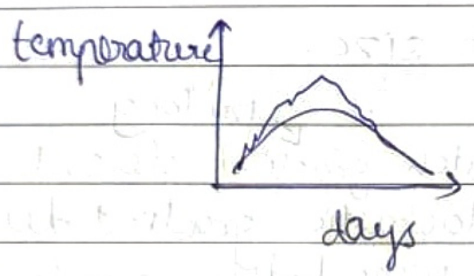
mini batch gradient descent:

- faster learning:
  - → vectorisation advantage
  - → make progress without waiting to ~~train~~ process the entire training set
- doesn't exactly converge (oscillates in small region, but you can reduce α)

- guidelines for choosing mini-batch size:
  (i) if trainingset size is less than 2000 use batch gradient descent.
  (ii) It has to be power of 2
  (iii) Make sure it fits in CPU/GPU memory

Video 3: Exponentially Weighted Averages

$V = \beta V_{t-1} + (1-\beta)\theta_t$



Exponentially weighted averages are useful for optimising gradient descent algorithm. It gives different θ based on β. This reduces oscillations in gradient descent and makes smooth path towards minima

Video 4: Understanding Exponentially Weighted Averages

$V(t) = \beta V_{t-1} + (1-\beta)\theta_t$

average is represented over $\frac{1}{1-\beta}$ entries.

best beta is between 0.9 & 0.98

$V_0 = 0$

Video 7: RMS Prop

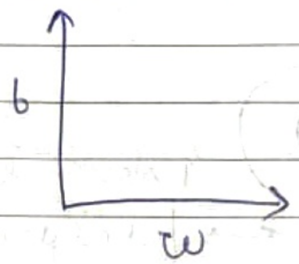
Modified algorithm:

On iteration t:

compute dw, db on current minibatch

element wise

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2 \leftarrow \text{small}$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2 \leftarrow \text{large}$$



ε is added so that $S_{dw} \neq 0$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw} + \varepsilon}} \qquad b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$

$\varepsilon = 10^{-8}$

Video 8: Adam optimisation

* RMSprop + momentum = adam optimisation
* ADAM - adaptive moment estimation

* Pseudocode / Algorithm

on iteration t, compute dw, db using current mini batch

$$\left.\begin{aligned} V_{dw} &= \beta_1 V_{dw} + (1-\beta_1) dw \\ V_{db} &= \beta_1 V_{db} + (1-\beta_1) db \end{aligned}\right\} \text{momentum}$$

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

$$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)} \qquad V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_2^t)} \qquad S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$$

$$w = w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

Hyperparameters:

α - to be tuned
$\beta_1 = 0.9$ (dw)
$\beta_2 = 0.99$ (dw^2)
$\varepsilon = 10^{-8}$

Video 9: Learning Rate Decay

While implementing mini batch gradient descent your steps will be noisy and it won't converge at minimum

PAGE NO.
DATE: / /

But as you reduce ~~alpha~~ α with time, at the beginning learning rate would be fast, but then as α keeps decreasing, small α will help oscillate in tighter region around minimum.

$$\alpha = \frac{1}{1 + \text{decay rate} * \text{epochnumber}} \alpha_0$$

decay rate ← hyperparameter

Other methods:

$$\alpha = 0.95^{\text{epoch number}} \times \alpha_0$$

$$\alpha = \frac{k}{\sqrt{\text{epochnumber}}} \times \alpha_0$$

changes to learning rate can be made discretely - decrease after some no. of epochs. Otherwise manually.

Video 10: The problem of local optima

The normal local optima is not likely to appear in a deep NN

Plateau - region of slow learning

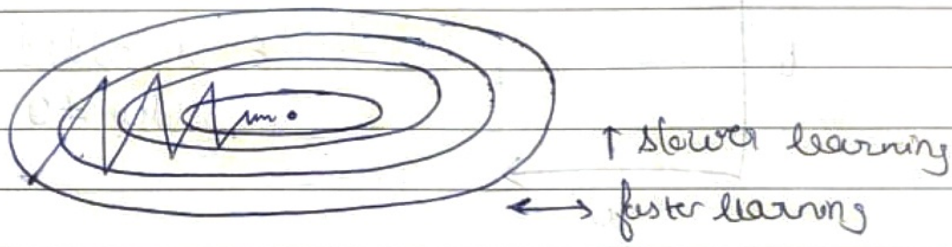
PAGE NO.
DATE: / /

Video 5: Bias correction in exponentially weighted avg.

Bias correction makes exponentially weighted averages smooth and more accurate.

$\cancel{\frac{V_t}{1-\beta^t} = V_t}$ $\cancel{V_t = \frac{V_t}{1-\beta^t}}$ $V_t = \frac{V_t}{1-\beta^t}$

Video 6: Gradient Descent With Momentum



* Oscillations slow down gradient descent and prevent you from using a larger learning rate.

* Larger learning rate could cause oscillations to shoot up.

* Implementing gradient descent + momentum

Momentum:
On iteration t:
compute dw, db, on current mini batch

$V_{dw} = \beta V_{dw} + (1-\beta)dw$ $\quad V_{\theta t} = \beta V_{\theta t} + (1-\beta)\theta_t$

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

Hyperparameters

$w = w - \alpha dw$ $\quad \alpha, \beta$

$b = b - \alpha V_{db}$