7) Optimization Algorithms

1) Mini Batch Gradient Descent:

$$X_{(n_x, m)} = [\underbrace{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}_{(n_x,1000)}} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}_{(n_x,1000)}} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5000\}}_{(n_x,1000)}}]$$

$$Y_{(1, m)} = [\underbrace{y^{(1)}, y^{(2)} \dots y^{(1000)}}_{Y^{\{1\}}_{(1,1000)}} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}_{(1,1000)}} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5000\}}_{(1,1000)}}]$$

Training set = 5.000.000 ise

ör, 5000 tane mini batch oluştur. Her biri 1000 gözlem içerir.

* Mini batch size = 1000

$\{\ \}$ ⇝ k. batch

Repeat {

for t = 1, 2, ..., 5000

Forward prob on $X^{\{t\}}$ } vectorized (1000 example) imple. from $X^{\{t\}}, Y^{\{t\}}$

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Backprob to compute gradients

}

Epoch, is a word that means a single pass through the training set.

! With batch gradient descent, a single pass through the training set allows you to take only one gradient descent step.

! With mini-batch gradient descent, a single pass through the training set, allows you to take 5.000 gradient descent steps.

Gözlem sayımız çok fazla ise mini-batch gradient descent kullanalım. Aksi durumda batch gradient descent kullanacağız.

!! Örneğin; elimizde 2000 training example olsun. We can divide dataset of 2000 examples into batches of 500 then, it will take 4 iterations to complete one epoch.

↳ Kendi örneğimizle;

5.000.000 training example

1000 batch size

5.000 iterations

for completing 1 Epoch.

Ek Bilgi:

★ Exponentially Weighted Averages;

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

V_{t-1}: üssel ortalama dün (Dünkü değer); θ_t: Bugün

Son fiyatların (veya neyi ölçüyorsak) önemini artırarak ortalama hesaplar. Basit ortalamada tümünün önemi aynı iken burada sonda olanlara daha çok ağırlık verir.

$V_t \rightarrow$ Exponentially average over $\frac{1}{1-\beta}$ day's temperature.

↳ Ör: $\beta = 0.9$ ise : ≈ (yaklaşık) last 10 days.

β çok (0.98).. büyük olursa; Önceki değere çok fazla önem vermiş olur.

β çok (0.5).. küçük olursa; bugünkü değere çok fazla önem vermiş olur.

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

$$V_{100} = 0.9 V_{99} + 0.1 \theta_{100}$$

$$V_{99} = 0.9 V_{98} + 0.1 \theta_{99}$$

$$V_{98} = 0.9 V_{97} + 0.1 \theta_{98}$$

$$V_{100} = 0.1\theta_{100} + 0.9(V_{99}) \rightarrow (0.1\theta_{99} + 0.9(V_{98})) \rightarrow 0.1\theta_{98} + 0.9 V_{97} \ldots$$

$$V_{100} = 0.1\theta_{100} + 0.1\times0.9\times\theta_{99} + 0.1\times(0.9)^2\times\theta_{98} + 0.1\times(0.9)^3\times\theta_{97} + \ldots$$

$$+ 0.1\times(0.9)^4\theta_{96} + \ldots$$

Bias correction in exponentially weighted average:

$V_0 = 0$

$V_1 = 0.98 V_0 + 0.02\theta_1$

$V_2 = 0.98 V_1 + 0.02\theta_2$

$= 0.0196\theta_1 + 0.02\theta_2$

Bu şekilde yaptığımız durumda ilk günlerdeki değerler beklenenden her zaman az olacaktır.

↳ Bias Correction burada devreye girer;

V_t yerine; ↳ $\frac{V_t}{1-\beta^t}$ → (current) örneğin: $t=2 : 1-\beta^t = 1-(0.98)^2 = 0.0396$

$$\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

Yani, Instead of taking V_t, take V_t divided by $1-\beta^t$. (t: current data)

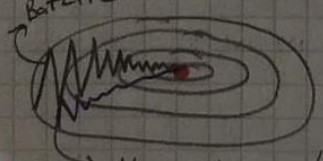
2) Momentum (or Gradient Descent with momentum)

Basic idea; compute an Exponentially Weighted Average of your gradients, and then use that gradient to update your weights.

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

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

$$W = W - \alpha V_{dw} \quad ; \quad b = b - \alpha V_{db}$$



→ Momentum (less oscillation (Dalgalanma), straight line, faster)

(↕ slower learning, ↔ faster learning } Bizim istediğimiz.)

initialize ; $V_{dw}=0$, $V_{db}=0$

on iteration t:

Compute dW, db on the current mini-batch.

$V_{dw} = \beta V_{dw} + (1-\beta)\, dW$

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

$W = W - \alpha V_{dw}$, $b = b - \alpha V_{db}$

Hyperparameters : α, β

Or $\beta = 0.9$ ⇒ average of last ≈ 10 gradients (yaklaşık)

Bu β iyi çalışır genelde.

3) RMSprop (Root mean square prop)

On iteration t:

Compute dW, db on current Mini-batch.

$S_{dw} = \beta S_{dw} + (1-\beta) dW^2$ (Element-wise)

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

$W = W - \alpha \frac{dW}{\sqrt{S_{dw}}}$, $b = b - \alpha \frac{db}{\sqrt{S_{db}}}$

* Larger "Learning Rate" kullanabiliriz. ; Faster.

4) Adam Optimization Algorithm

Taking momentum and RMSprop , then put them together.

Initialize : $V_{dw}=0$, $S_{dw}=0$, $V_{db}=0$, $S_{db}=0$

On iteration t:

Compute dw, db using current mini-batch.

Momentum → $V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ "β_1"

RMSprop → $S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ "β_2"

(Bias correction)

$$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t), \quad S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$$

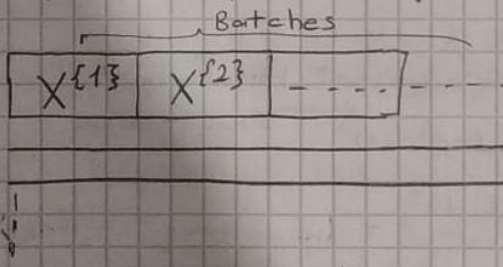
$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}, \quad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

(ε: zero olursa diye)

Learning Rate Decay:

* Help speed up your learning algorithm,
* Slowly reduce your learning rate over time.

1 epoch $\Rightarrow$ 1 pass through all training data



$$\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch\_num}} \alpha_0$$ (α_0 → initial α)

! Learning rate'in her iterasyonda azalmasını isteriz, ki minima of cost function bulunabilsin.

Not: Our goal is to reach a global minimum. But at times when our cost function has an irregular curve (mostly in deep networks), maybe we reach a local minimum, which is not as good as global minimum. One way to overcame this is by using the concept of momentum.