Mini-batch Gradient Descent

Batch vs Mini-batch Gradient Descent

Vectorization is slow for very large dataset..

$$X = [\underbrace{x^{(1)}, \cdots, x^{(1000)}}_{X^{\{1\}}} \mid \underbrace{x^{(1001)}, \cdots, x^{(2001)}}_{X^{\{2\}}} \mid \cdots \mid \underbrace{x^{(49001)}, \cdots x^{(50000)}}_{X^{\{50\}}}]$$

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

One Epoch

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

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$\vdots$$

$$A^{[e]} = g^{[e]}(Z^{[e]})$$

Compute cost $J = \frac{1}{1000} \sum_{i=1}^{e} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ for $X^{\{t\}}, Y^{\{t\}}$ mini batches

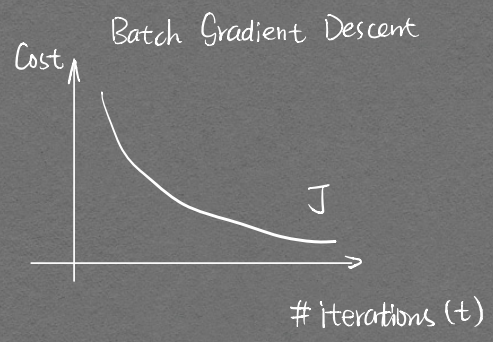
Backprop to compute gradients wrt $J^{\{t\}}$ (using $X^{\{t\}}, Y^{\{t\}}$)

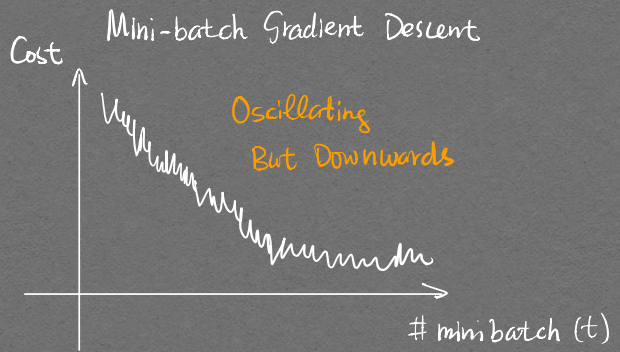
$$W^{[e]} := W^{[e]} - \alpha dW^{[e]}$$

$$b^{[e]} := b^{[e]} - \alpha db^{[e]}$$

}

Understanding Mini-batch Gradient Descent





Choose Minibatch Size

Single Training Example

1

All Training Example

m

Stochastic Gradient Descent

Lose speeding up from vectorization

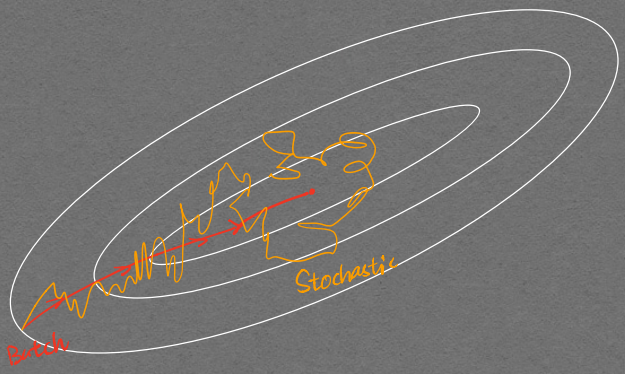
Batch Size

Vectorization Speed Up

Make progress without waiting too long

Batch Gradient Descent

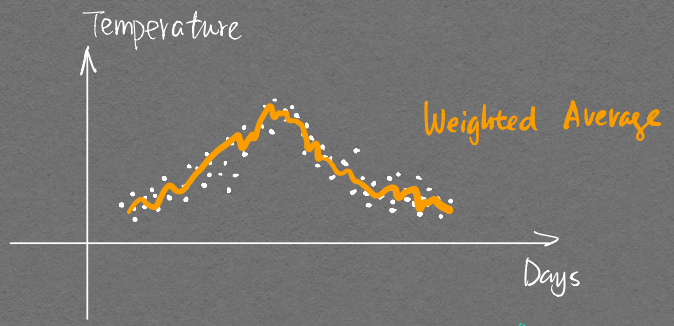
Too long Per Iteration



if $m \leq 2000$ → Batch Gradient

else 64, 128, 256, 512, 1024

Exponentially Weighted Averages



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

$\approx$ Average over $\frac{1}{1-\beta}$ days' temperature

$\beta = 0.9$ 10 day average

$\beta = .98$ 50 day average

$\beta = .5$ 2 day average

Understanding Exponentially Weighted Averages

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

$$= 0.9 \times (0.9 V_{98} + 0.1 \theta_{99}) + 0.1 \theta_{100}$$

$$= \cdots$$

$$= 0.1\theta_{100} + 0.1 \times 0.9\, \theta_{99} + 0.1 \times 0.9^2 \times \theta_{98} + \cdots + 0.1 \times 0.9^{99} \times \theta_1$$

$(1-\varepsilon)^{1/\varepsilon} = \frac{1}{e}$

↳ how many days are we averaging over approximately

$v_0 = 0$

Repeat {

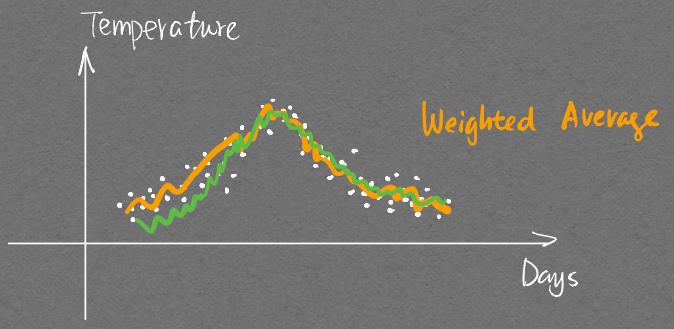
Get next θ_t

$v_0 := \beta v_0 + (1-\beta)\theta_t$

}

Simple & Efficient

Bias Correction in Exponentially Weighted Average

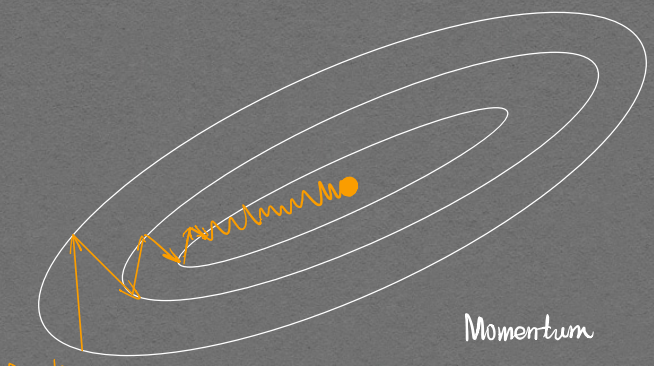


Use $\frac{v_t}{1-\beta^t}$

instead of v_t

especially for initial base

Gradient Descent with Momentum



Batch Gradient Descent

Oscillating takes time

We want

learning slow on vertical direction

fast on horizontal direction

Momentum

On Iteration t:

Compute dW, db wrt current mini-batch

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

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

might be omitted

$W \leftarrow W - \alpha v_{dw}$

$b \leftarrow b - \alpha v_{db}$

$\beta = .9$

(Average over last ≈ 10 iterations)

RMSprop Group Mean Squared

On Iteration t:

Compute dW, db wrt current mini-batch

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dW^{\underline{2}}$$ Smaller Value

element-wise

$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^{\underline{2}}$$ Larger Value

$$W \leftarrow W - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$$ Quicker

10^{-8} to avoid blow up.

$$b \leftarrow b - \alpha \frac{db}{\sqrt{S_{db}}}$$ Slower

Adam Optimization Algorithm

$V_{dw}=0 \quad S_{dw}=0 \quad V_{db}=0 \quad S_{db}=0$

On Iteration t:

Compute dW, db wrt current mini-batch

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

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

$$V_{dw}^{Correct} = V_{dw} / (1-\beta_1^t)$$
$$V_{db}^{Correct} = V_{db} / (1-\beta_1^t)$$
$$S_{dw}^{Correct} = S_{dw} / (1-\beta_2^t)$$
$$S_{db}^{Correct} = S_{db} / (1-\beta_2^t)$$ Bias Correction

$$W \leftarrow W - \alpha \frac{V_{dw}^{Correct}}{\sqrt{S_{dw}^{Correct}} + \varepsilon}$$

$$b \leftarrow b - \alpha \frac{V_{db}^{Correct}}{\sqrt{S_{db}^{Correct}} + \varepsilon}$$ Update

Hyparameter Choice:

α: needs to be tune

β_1: 0.9 First Moment

β_2: 0.999 Second Moment

ε: 10^{-8}

Learning Rate Decay Take smaller update steps when approaching the minimum

$$\alpha = \frac{1}{1 + \text{Decay Rate} \times \text{Epoch Number}} \alpha_0$$

1 Epoch : 1 Pass over training data (One iteration of
forward prop
cost calculate
backward prop
for loop)

Exponential Decay : $\alpha = 0.95^{\text{Epoch Number}} \alpha_0$

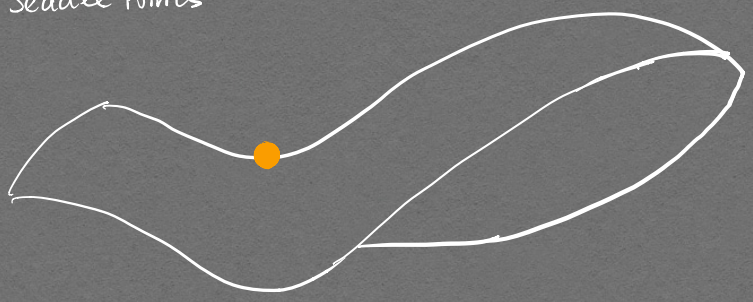
$$\alpha = \frac{k}{\sqrt{\text{Epoch Number}}} \alpha_0$$ (k for some constant Tunable)

Discrete Decay

Manual Decay

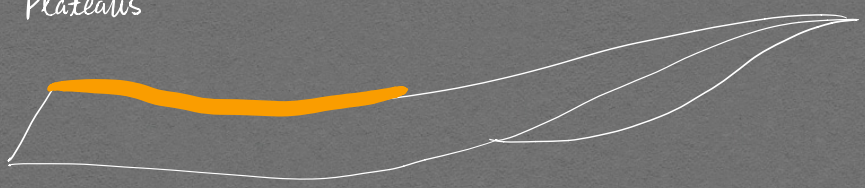
Local Optima

Seddle Points



- Unlike to get stuck in local optima in high-D space.

Plateaus



- Plateaus can make learning slow.