Computational Intelligence

Samaneh Hosseini

Isfahan University of Technology

Outline

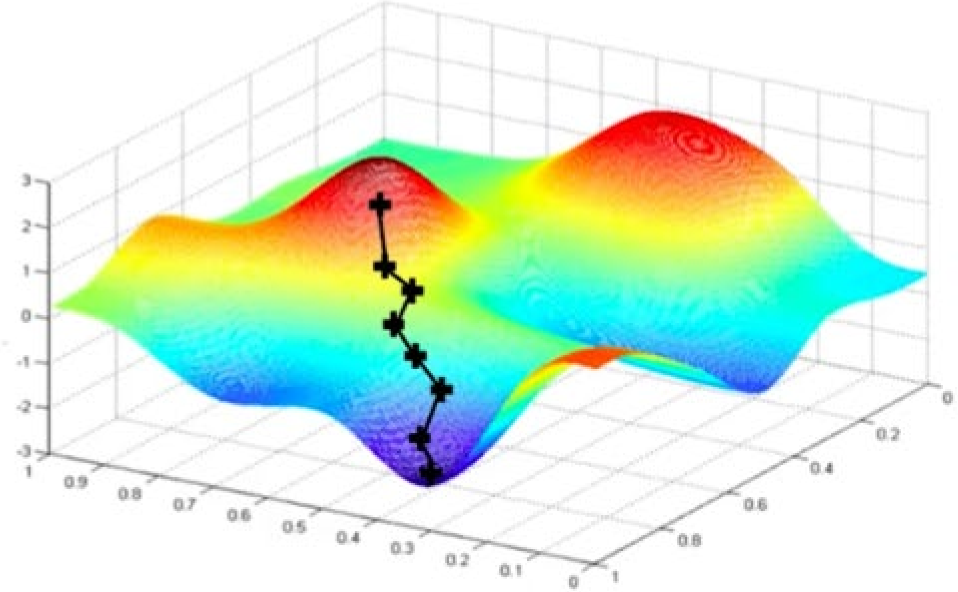
- Neural Networks in Practice:
  - Mini-batches
  - Mini-batch Gradient Descent
  - Understanding Mini-batch Gradient Descent
  - Exponentially Weighted Averages

Neural Networks in Practice: Mini-batches

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



زمان اجرای هر raund

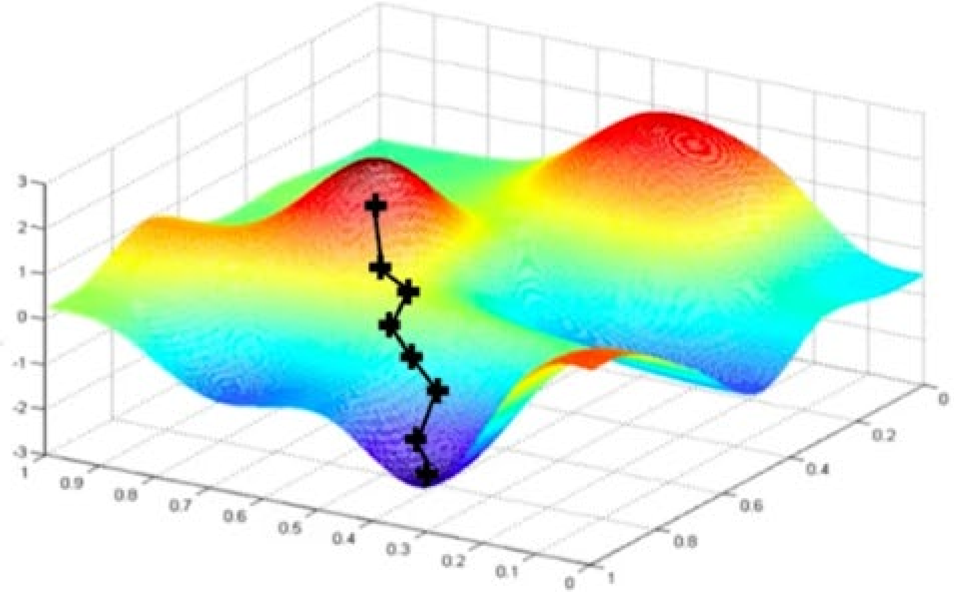
تعداد تکرار حلقه

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

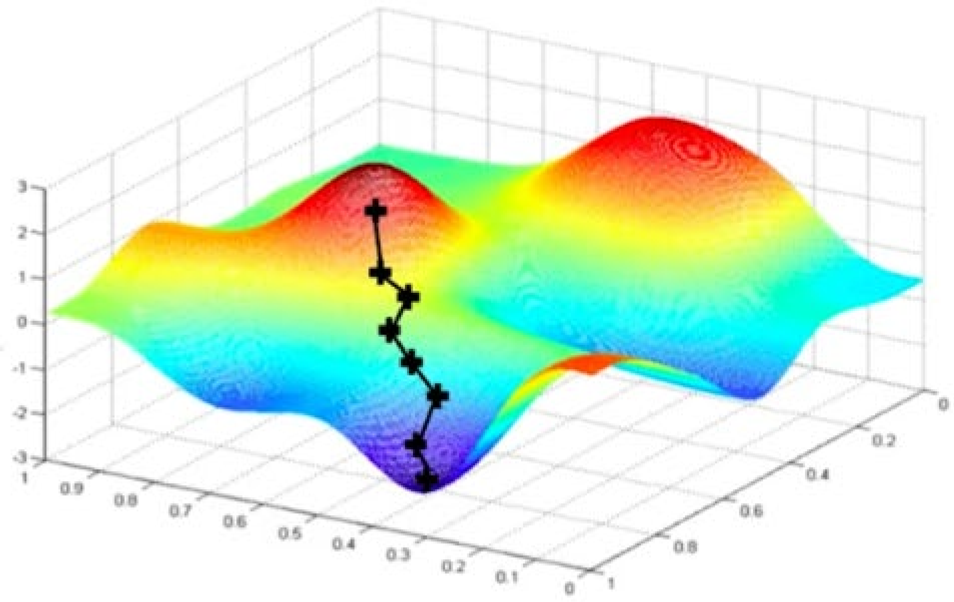
Can be very **computationally intensive** to compute!



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

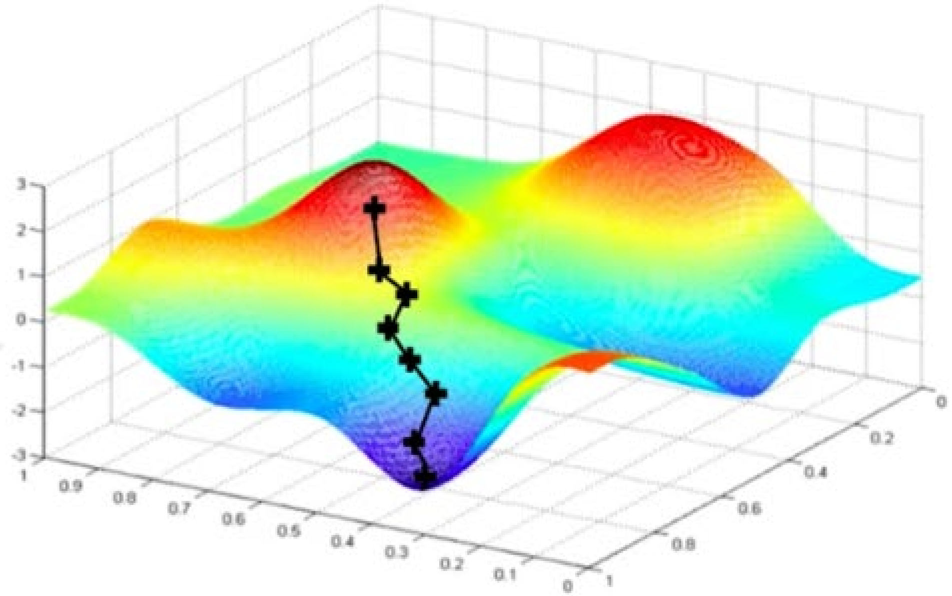


Can be very **computationally intensive** to compute!

Stochastic Gradient Descent

Algorithm

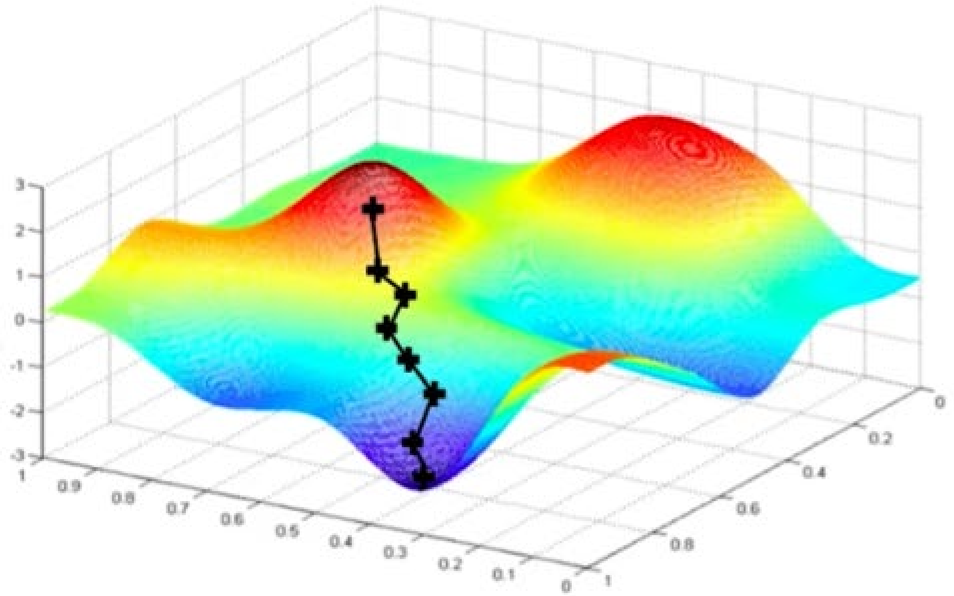
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\boxed{\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



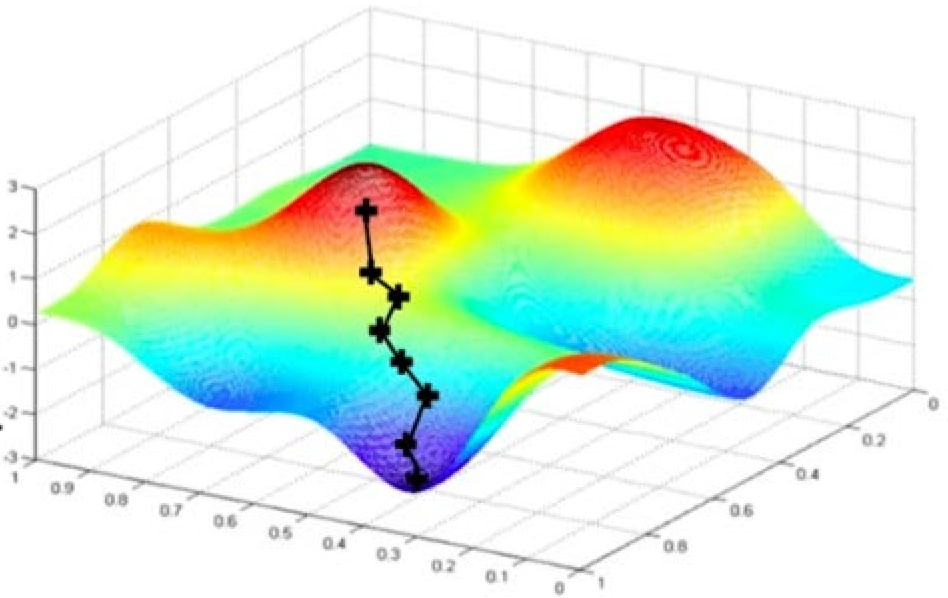
Easy to compute but
very noisy (stochastic)!

Stochastic Gradient Descent

mini-batch

Algorithm

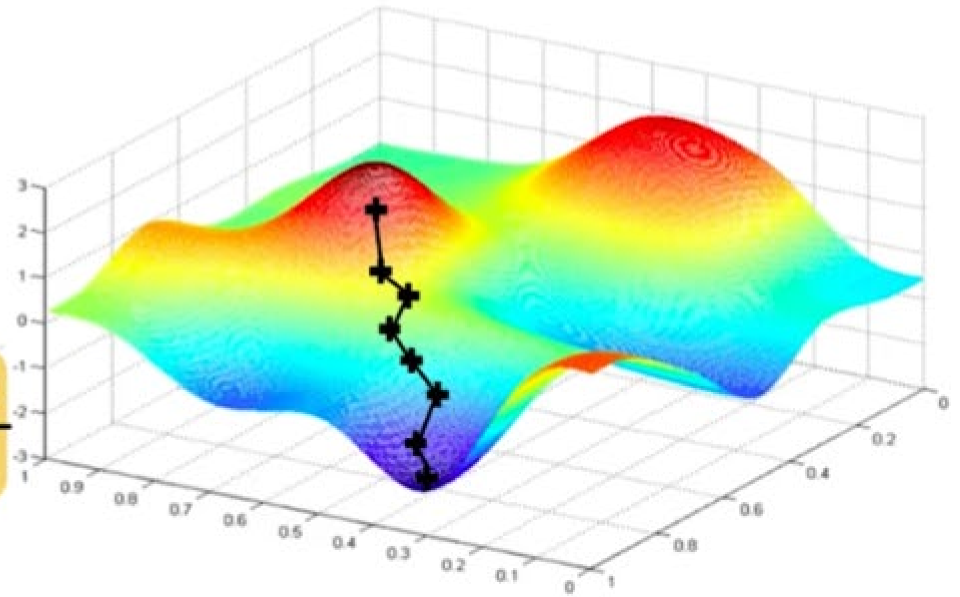
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

Batch vs. mini-batch gradient descent

X, Y

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

- Vectorization allows you to efficiently compute on m examples

$$\underset{(n_x, m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \ldots x^{(1000)}}_{X^{\{1\}}\ (n_x \times 1000)} \mid \underbrace{x^{(1001)} \ldots x^{(2000)}}_{X^{\{2\}}\ (n_x, 1000)} \mid x^{(2001)} \ldots \mid \underbrace{\ldots x^{(m)}}_{X^{\{5000\}}\ (n_x, 1000)}]$$

$$\underset{(1, m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \ldots y^{(1000)}}_{Y^{\{1\}}\ (1 \times 1000)} \mid \underbrace{y^{(1001)} \ldots y^{(2000)}}_{Y^{\{2\}}\ (1 \times 1000)} \mid \ldots \mid \underbrace{\ldots y^{(m)}}_{Y^{\{5000\}}\ (1 \times 1000)}]$$

$m = 5{,}000{,}000$

5000 mini batches of 1000 each

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

$x^{(i)}$

$z^{[L]}$

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

Mini-batch gradient descent

1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$

1000

repeat {

for $t = 1 \ldots 5000$ {

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

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

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

$\vdots$

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

vectorized implementation (1000 example)

X, Y

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

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

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

}

}

1 epoch

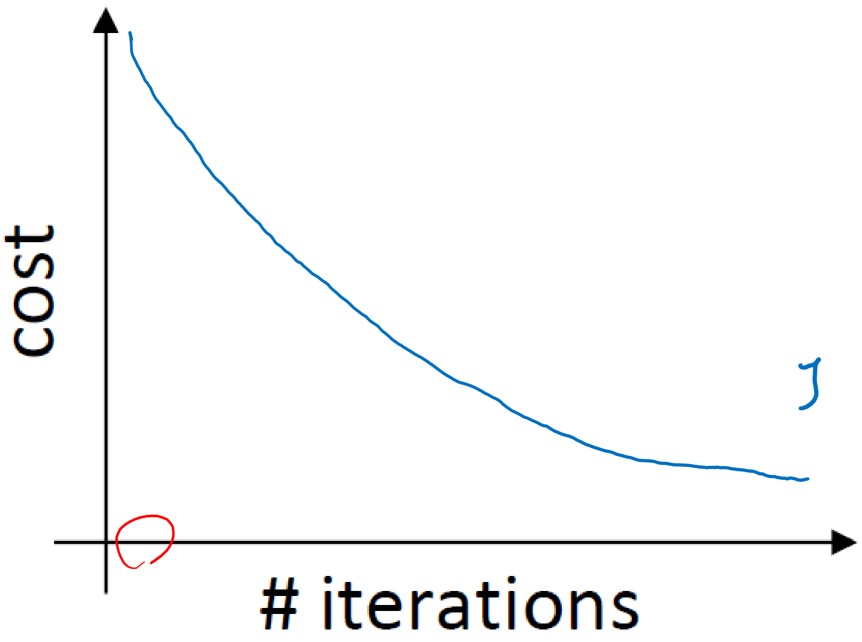
↓

1 pass through all training set

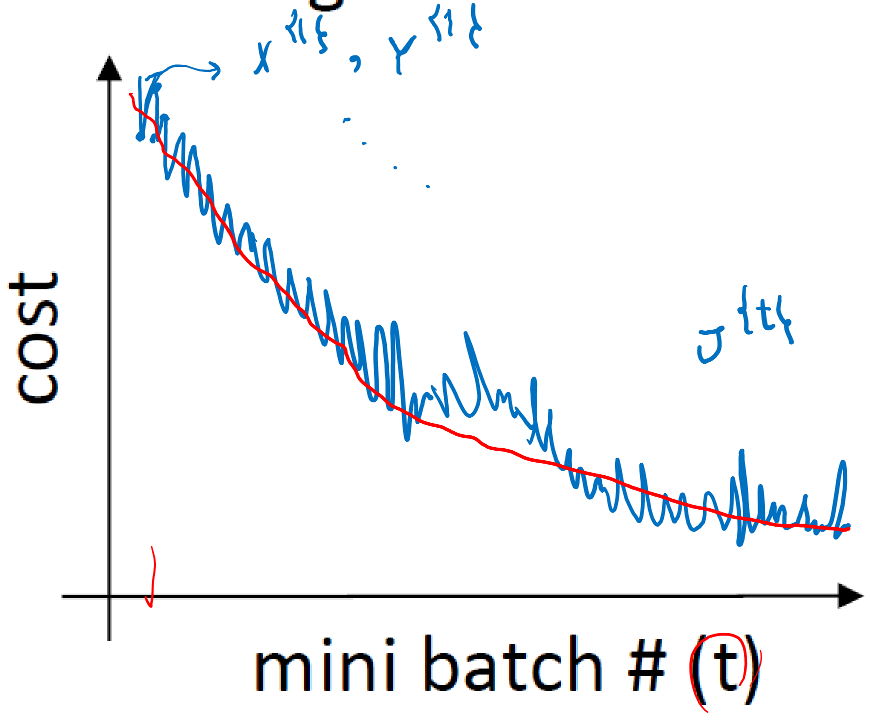
Optimization Algorithms: Understanding Mini-batch Gradient Descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent



plot $J^{\{t\}}$ computed using $X^{\{t\}}, Y^{\{t\}}$

Choosing your mini-batch size

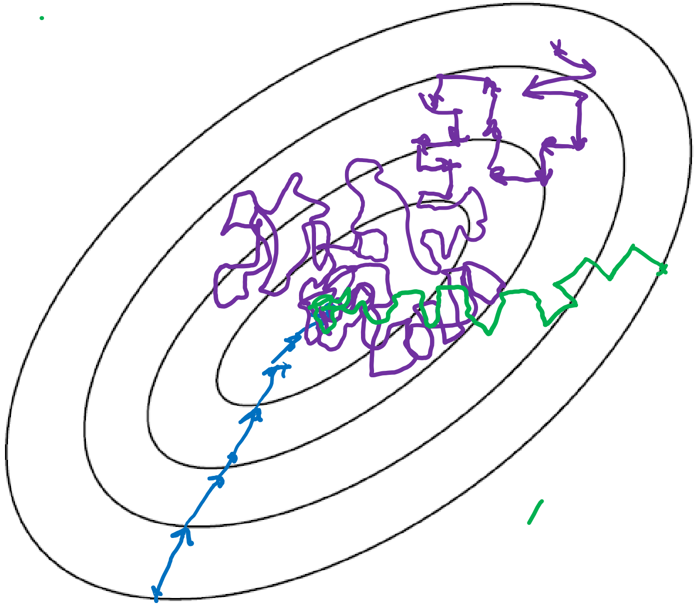
$(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

→ if mini_batch size = m : Batch graidient descent

→ if mini_batch size = 1 : stochastic gradient descent

$(X^{\{1\}}, Y^{\{1\}}) = (X^{(1)}, Y^{(1)})$

In practice : somewhere between 1 and m



stochastic graidient descent → loose speedup from vectorization

In_between (mini_batch size not too big / small) → Fastest learning

- vectorization on per iteration (1000)
- Make progress without using entire train data

batch graidient descent (minibatch size = m) → too long

Choosing your mini-batch size

if small traing set : use batch graidant descent
($m \lesssim 2000$)

Typical mini_ batch size :

$\rightarrow$ 64, 128, 256, 512

$\underbrace{2^6 \quad 2^7 \quad 2^8 \quad 2^9}$

Make sure mini_batch fit in CPU/GPU memory

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

Optimization Algorithms: Exponentially Weighted Average

Temperature in London

$\theta_1 = 40°F$

$\theta_2 = 49°F$

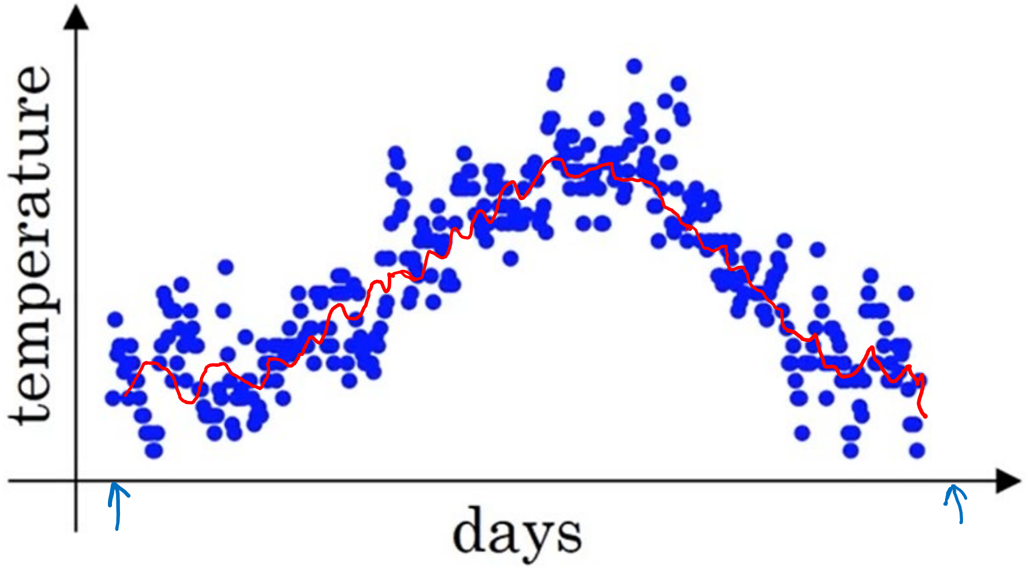
$\theta_3 = 45°F$

$\vdots$

$\theta_{180} = 60°F$

$\theta_{181} = 56°F$

$\vdots$



$v_0 = 0$

$v_1 = 0.9 v_0 + 0.1\ \theta_1$

$v_2 = 0.9 v_1 + 0.1\ \theta_2$

$v_3 = 0.9 v_2 + 0.1\ \theta_3$

$\vdots$

$$\boxed{v_t = 0.9 v_{t-1} + 0.1\ \theta_t}$$

Exponentially weighted averages

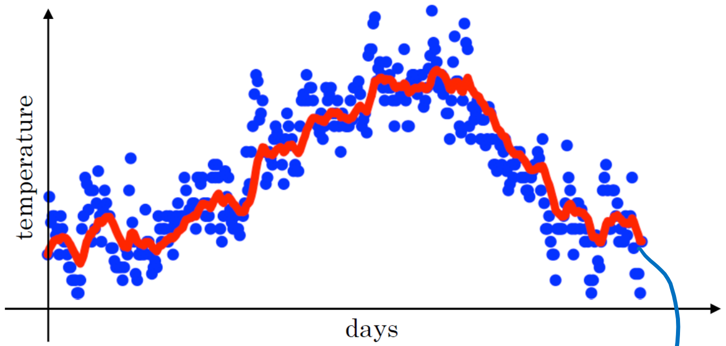
$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$\beta = 0.9$: ≈ 10 datapoint temprature

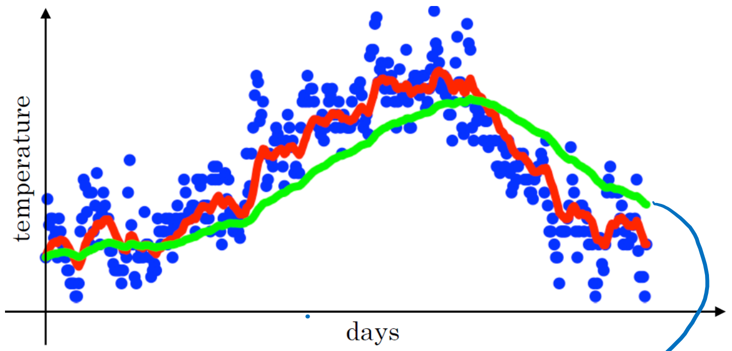
$\beta = 0.98$: ≈ 50 "

$\beta = 0.5$: ≈ 2 "

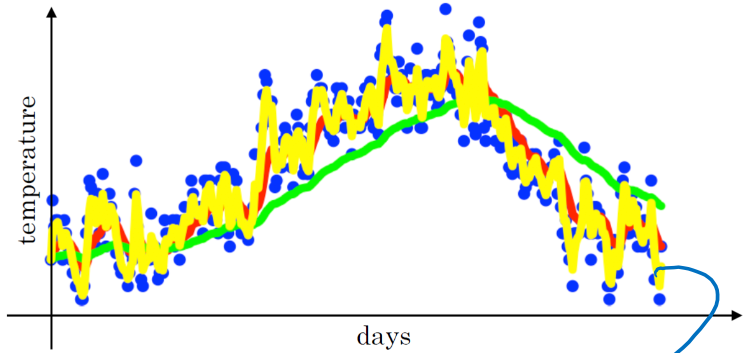
v_t average over $\approx \frac{1}{1-\beta}$ datapoint



$\beta = 0.9$



$\beta = 0.98$



$\beta = 0.5$

Core Foundation Review

- Neural Networks in Practice:
  - Mini-batches
  - Mini-batch Gradient Descent
  - Understanding Mini-batch Gradient Descent
  - Exponentially Weighted Averages