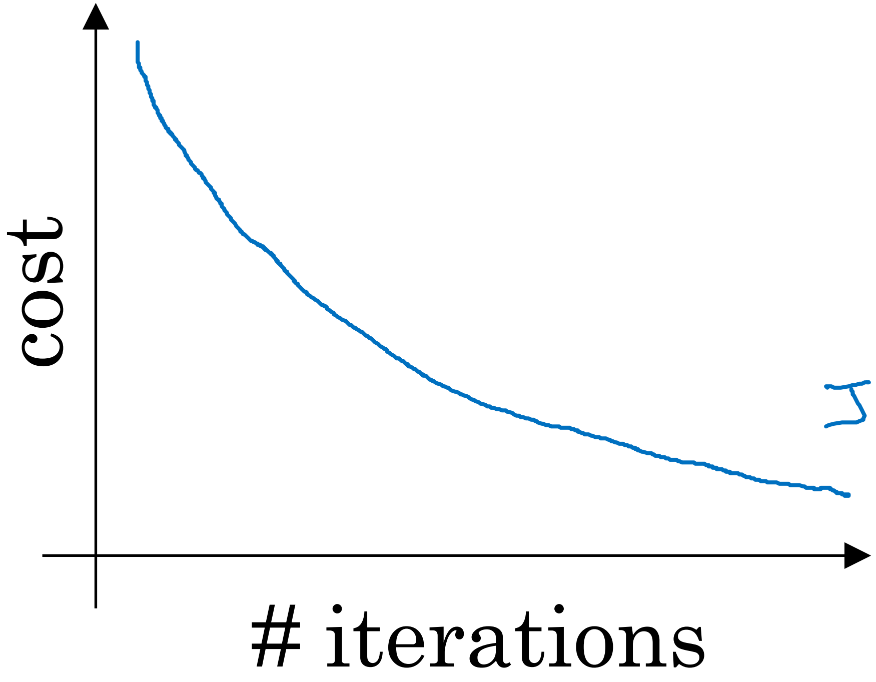
deeplearning.ai

Optimization Algorithms

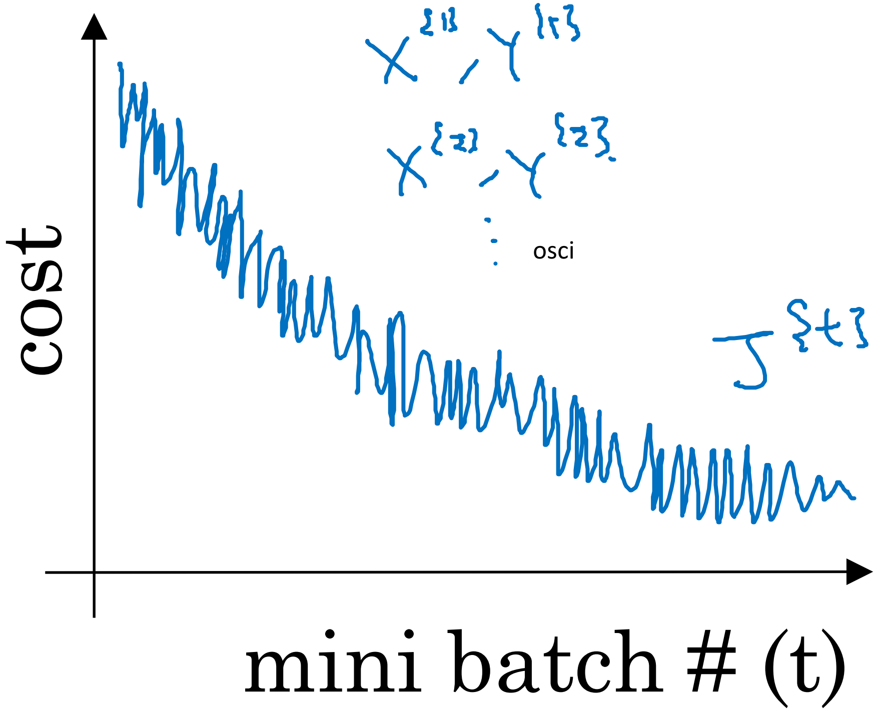
Understanding mini-batch gradient descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent



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

It's ok if it doesn't go down on every iteration. But it should trend downwards

Andrew Ng

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own

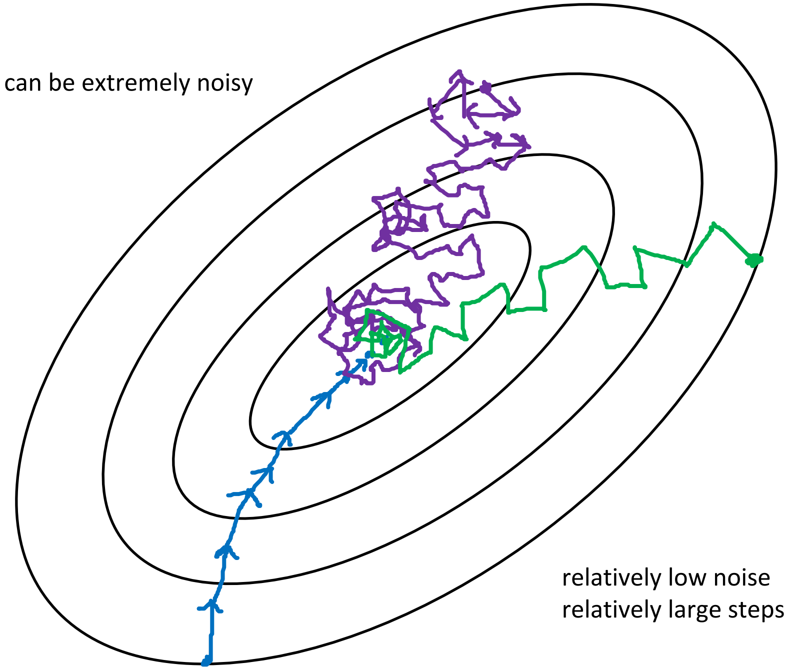
stochastic gradient descent

$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)}) \ldots (x^{(2)}, y^{(2)})$ mini-batch.

In practice: Somewh in-between 1 and m

As stochastic gradient descent won't ever converge, it'll always just kind of oscillate and wander around the region of the minimum. But it won't ever just head to the minimum and stay there

can be extremely noisy



relatively low noise
relatively large steps

Stochastic gradient descent

noise can ameliorated or can be reduced by just using a smaller learning rate.

Lose speedup from vectorization

In-between (mini-batch size not too big/small)

Fastest learning.

- Vectorization. (~1000)
- Make progress without processing entire training set.

And then it doesn't always exactly converge or oscillate in a very small region. if that's the issue you can always reduce the learning rate slowly.

Batch gradient descent (mini-batch size = m)

Too long per iteration

Andrew Ng

Choosing your mini-batch size

If small trainig set: Use batch gradient descent.
$(m \leq 2000)$

Typical mini-batch sizes:

Because of the way computer memory is laid out and accessed, sometimes your code runs faster if you mini-batch size is a power of 2.

$\rightarrow \underbrace{\underset{2^6}{64}, \underset{2^7}{128}, \underset{2^8}{256}, \underset{2^9}{512}}_{\text{a little bit common}} \qquad \frac{1024}{2^{10}}$

Make sure mini-batch fit in CPU/GPU memory.
$X^{\{t\}}, Y^{\{t\}}$