# Optimization Algorithms

# Mini-Batch Gradient Descent

$$\underset{(n_x, m)}{X} = [x^{(1)}\ x^{(2)}\ x^{(3)} \ldots \ldots x^{(m)}]$$

$$\underset{(1, m)}{Y} = [y^{(1)}\ y^{(2)}\ y^{(3)} \ldots \ldots y^{(m)}]$$

Where X is the vectorized input and Y is their corresponding outputs. But what will happen if we have a large training set, say m = 5,000,000?
If we process through all of these training examples in every training iteration, the gradient descent update will take a lot of time.
Instead, we can use a mini batch of the training examples and update the weights based on them.

# Mini-Batch Gradient

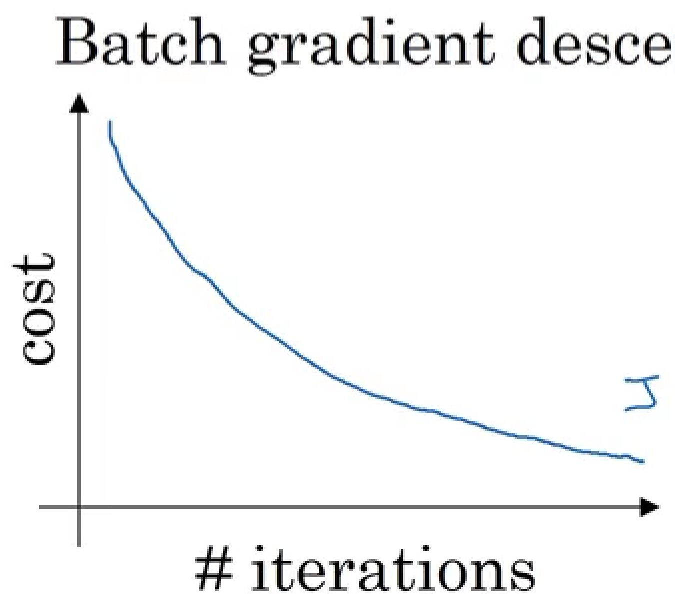
- Suppose we make a mini-batch containing 1000 examples each. This means we have 5000 batches and the training set will look like this:

$$\underset{(n_x,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5,000\}}}]$$

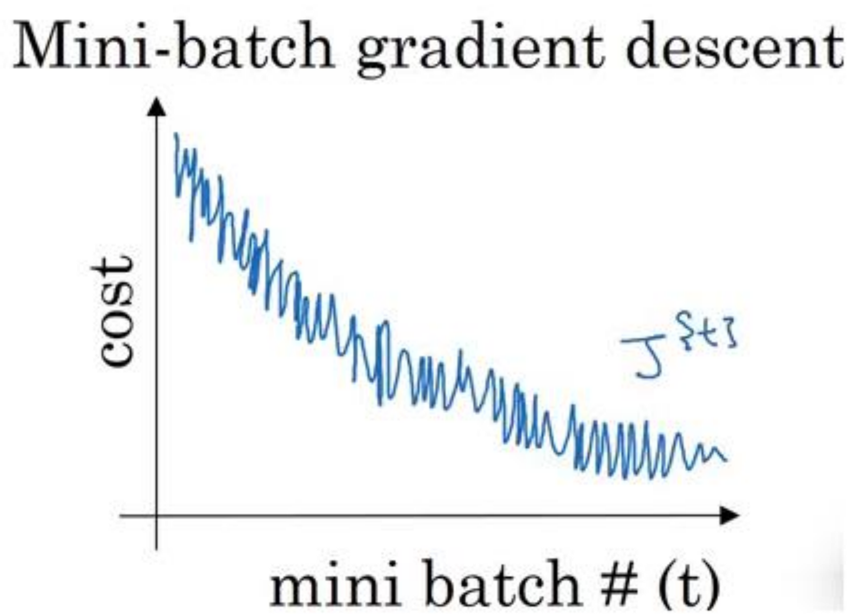
$$\underset{(1,m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5,000\}}}]$$

# Understanding Mini-Batch Gradient Descent

- In batch gradient descent, our cost function should decrease on every single iteration:



- In the case of mini-batch gradient descent, we only use a specified set of training examples. As a result, the cost function can decrease for some iterations:



# How can we choose a mini-batch size?

- **<u>If the mini-batch size = m:</u>**
  It is a batch gradient descent where all the training examples are used in each iteration. It takes too much time per iteration.

- **<u>If the mini-batch size = 1:</u>**
  It is called stochastic gradient descent, where each training example is its own mini-batch. Since in every iteration we are taking just a single example, it can become extremely noisy and takes much more time to reach the global minima.

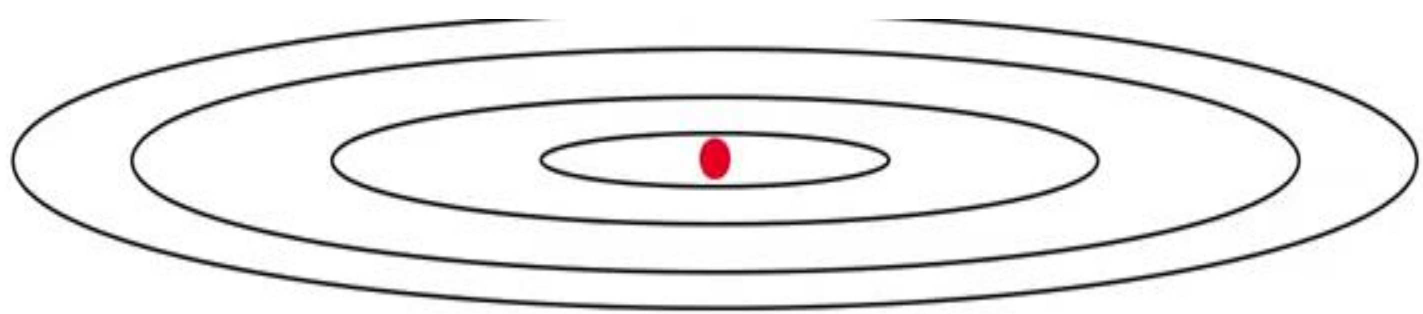
- **<u>If the mini-batch size is between 1 to m:</u>**
  It is mini-batch gradient descent. The size of the mini-batch should not be too large or too small.

# Guidelines:

- If the training set is small, we can choose a mini-batch size of $m<2000$
- For a larger training set, typical mini-batch sizes are: 64, 128, 256, 512
- Make sure that the mini-batch size fits your CPU/GPU memory

# Gradient Descent with Momentum

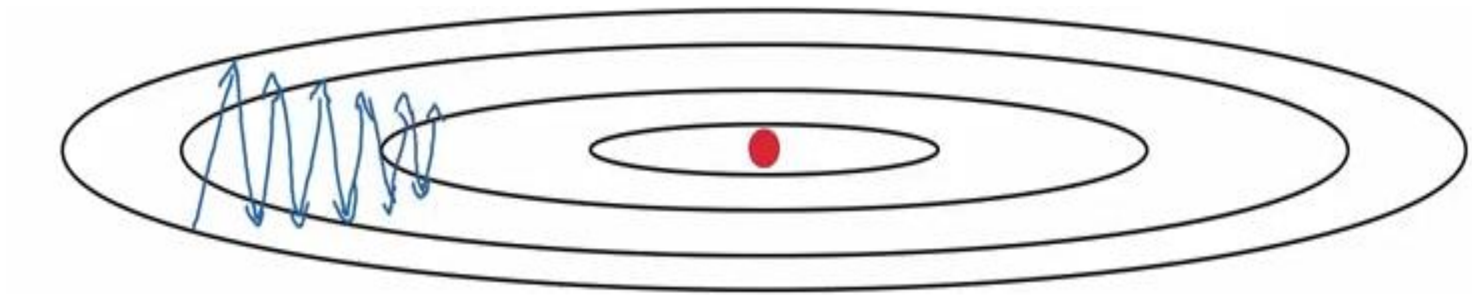
- The underlying idea of gradient descent with momentum is to calculate the exponential weighted average of gradients and use them to update weights. Suppose we have a cost function whose contours look like this:

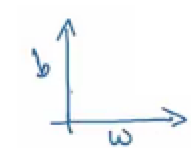


- One more way could be to use a larger learning rate. But that could result in large upgrade steps, and we might not reach global minima. Additionally, too small a learning rate makes the gradient descent slower.
- We want a slower learning in the vertical direction and a faster learning in the horizontal direction which will help us to reach the global minima much faster.

# RMSprop

- Consider the example of a simple gradient descent:






Look at the contour shown above and the parameters graph. We want to slow down the learning in b direction, i.e., the vertical direction, and speed up the learning in w direction, i.e., the horizontal direction

# Adam

- **Adam optimization algorithm**
- Adam is essentially a combination of momentum and RMSprop.

- There are a range of hyperparameters used in Adam and some of the common ones are:

- **Learning rate** $\alpha$ : needs to be tuned
- **Momentum term** $\beta_1$: common choice is 0.9
- **RMSprop term** $\beta_2$: common choice is 0.999
- $\varepsilon$ : $10^{-8}$
- Adam helps to train a neural network model much more quickly than the techniques we have seen earlier.