deeplearning.ai

Optimization Algorithms

Mini-batch gradient descent

(ways to converge quickly!)

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$X = [x^{(1)}, x^{(2)} \dots x^{(m)}]$ $(n_x \times m)$

$Y = [y^{(1)}, y^{(2)} \dots y^{(m)}]$ $(1 \times m)$

If m = 50 Million, then what? $\Rightarrow$ slow

Soln
↳ split training set into mini batches of 1000 each
$\Rightarrow x^{(1)}, x^{(2)} \dots x^{(1000)} = X^{\{1\}}$
↳ curly brace = mini batch

$\Rightarrow x^{(1001)}, x^{(1002)} \dots x^{(2000)} = X^{\{2\}}$

$\Rightarrow$ 50,000 mini batches of 1000 samples

$\Rightarrow$ Minibatch t : $X^{\{t\}}, Y^{\{t\}}$

$X^{(i)} \rightarrow i^{th}$ sample

$Z^{[l]} \rightarrow$ output at layer l

$X^{\{t\}}, Y^{\{t\}} \rightarrow$ minibatch 't' of the sample data
↳ dim $X^{\{t\}} = (n_x \times 1000)$ ← minibatch size

dim $Y^{\{t\}} \rightarrow (1 \times 1000)$

Andrew Ng

Mini-batch gradient descent

for $t = 1$ to 50,000:

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

// Fwd Prop on $X^{\{t\}}$

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

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

$$\vdots$$

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

} should use vectorized implementation

// Compute cost

$$J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1000} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{i=1}^{l} \| W^{[l]} \|_2$$

↳ considering only $y^{\{t\}}$

// Backprop to find gradient wrt $J^{\{t\}}$ (using $X^{\{t\}}, Y^{\{t\}}$)

$$W^{[l]} = W^{[l]} - \alpha \cdot dW^{[l]}$$

$$b^{[l]} = b^{[l]} - \alpha \cdot db^{[l]}$$

- with gradient descent (batch)
  - 1 epoch (1 pass through training set)
  - allows you to update w, b only 1 time
- with mini batch gradient descent
  - 1 epoch (pass through entire training set)
  - allows you to update w, b 50,000 times

Andrew Ng