May 31, 2021

The best way to get a high-performance machine learning system is to take a low-bias learning algorithm and train it on a lot of data.

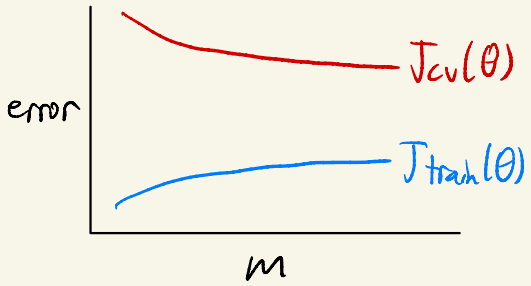
Learning w/ Large Datasets:

e.g. $m = 100$ million

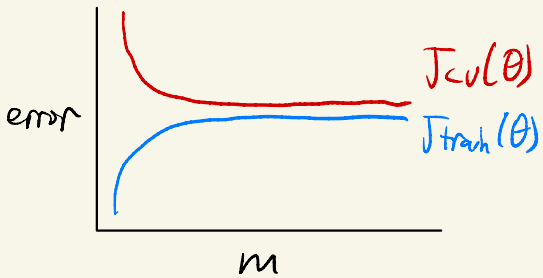
Gradient Descent: sums over 100 mil each update; inefficient

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

What if we use $m = 1000$ instead?



⛤ Low bias, needs $m > 1000$



⛤ High bias, $m > 1000$ will not help

Batch Gradient Descent

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

(for every $j = 0 \dots n$)

}

Stochastic Gradient Descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

↳ How well is the model doing on a single training example?

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset
2. Repeat { → m can be anything
   for $i = 1 \dots m$ {
   $\theta_j := \theta_j - \alpha\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$
   (for $j = 0 \dots n$)
   }
}

⊘ Nested loop!

$$\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$$

↳ Updates θ_j for $(x^{(1)}, y^{(1)})$, then $(x^{(2)}, y^{(2)})$ ... instead of updating θ_j for all the examples at once.

Mini-Batch Gradient Descent

Instead of using all or just one example in each iteration, use b examples (where $b = 2\text{-}100$).

b = mini-batch size ⇒ i.e. Update θ_j once every b examples

e.g. $b = 10$

$(x^{(i)}, y^{(i)}) \dots (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

$$i := i + 10$$

e.g. $b = 10, m = 1000$

Repeat {

  for $i = 1, 11, 21, 31 \dots 991$ {

    $$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

    (for every $j = 0 \dots n$)

  }

}

Checking for Convergence:

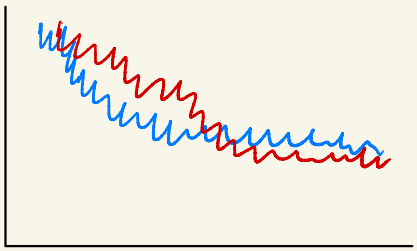
Batch Gradient Descent:

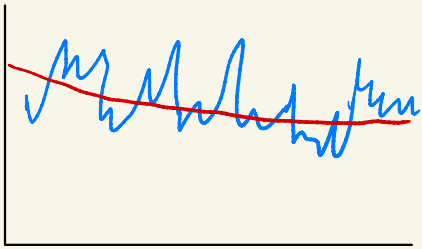
→ Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.

Stochastic Gradient Descent:

→ During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ.

→ Every 1000 iterations (e.g.), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed.





↳ Decrease # iterations averaged over