Mini-bach Gradient Descent

Felix Cheysson, Tristan Mary-Huard

Batch Gradient Descent (= "classical" GD)

Consider the learning task where one aims at finding

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} f(x_i, \theta),$$

θ: unknown parameters $(x_1, ..., x_n)$: training set.

Batch GD Update $\hat{\theta}$ sequentially as follows:

for $t = 1, ..., T$ **do**
$Grad(t) = \sum_{i=1}^{n} \nabla f(x_i, \theta)$
$\hat{\theta}^{t+1} = \hat{\theta}^{t} - \lambda^{t+1} Grad(t)$
end for
return $\hat{\theta}^{T}$

Pros and cons

- tends to converge very well to local optima
- very few hyper-parameters to tune
- computing the cost and gradient for the entire training set can be very slow and sometimes intractable

Stochastic Gradient Descent

Principle

Rather than using the full training set at each iteration, consider only 1 observation:

for $t = 1, ..., T$ **do**
  for $i = 1, ..., n$ **do**
    $Grad(t, i) = \nabla f(x_i, \theta)$
    $\hat{\theta}^{t,i} = \hat{\theta}^{t,i-1} - \lambda^t Grad(t, i)$
  end for
end for
return $\hat{\theta}^{T,n}$

A pass through the dataset (i.e. an iteration t) is called an **epoch**.

Pros and cons

- computing $\nabla f(x_i, \theta)$ is cheap $\Rightarrow$ no space / memory problem but vectorization shorcuts are lost
- considering a single observation adds noise to the optimization process $\Rightarrow$ escape local optima but optimization is erratic

Mini batch Gradient Descent

Principle

Use a small subset of observations at each iteration.

for $t = 1, ..., T$ **do**
  Split the data into batches of size m: $B^1, ..., B^J$
  for $j = 1, ..., J$ **do**
    $Grad(t, j) = \nabla \sum_{i \in B^j}^{m} f(x_i, \theta)$
    $\hat{\theta}^{t,j} = \hat{\theta}^{t,j-1} - \lambda^t Grad(t, j)$
  end for
end for
return $\hat{\theta}^{T,J}$

Pros and cons

- Providing m is small, computing $Grad(t, j)$ is cheap $\Rightarrow$ no space / memory problem and benefit from vectorization !
- Using m observations ($m > 1$) adds noise to the optimization process $\Rightarrow$ escape local optima and much less erratic than SGD.

The best of the 2 worlds ?

Using m examples in a minibatch requires $O(m)$ computations and use $O(m)$ memory, and reduce the amount of uncertainty in the gradient by a factor of $O(\sqrt{m})$.

$\Rightarrow$ Increasing the mini-batch size results in diminishing marginal rewards.

See:
http://www.deeplearningbook.org/contents/optimization.html