

# Stochastic Gradient Descent

Julia Kempe & David S. Rosenberg

NYU CDS

February 5, 2019

## Review: Statistical Learning Theory Framework

---

# Our Setup from Statistical Learning Theory

## The Spaces

- $\mathcal{X}$ : input space
- $\mathcal{Y}$ : outcome space
- $\mathcal{A}$ : action space

## Prediction Function (or “decision function”)

A **prediction function** (or **decision function**) gets input  $x \in \mathcal{X}$  and produces an action  $a \in \mathcal{A}$  :

$$\begin{aligned} f: \mathcal{X} &\rightarrow \mathcal{A} \\ x &\mapsto f(x) \end{aligned}$$

## Loss Function

A **loss function** evaluates an action in the context of the outcome  $y$ .

$$\begin{aligned} \ell: \mathcal{A} \times \mathcal{Y} &\rightarrow \mathbf{R} \\ (a, y) &\mapsto \ell(a, y) \end{aligned}$$

# Risk and the Bayes Prediction Function

## Definition

The **risk** of a prediction function  $f : \mathcal{X} \rightarrow \mathcal{A}$  is

$$R(f) = \mathbb{E} \ell(f(x), y).$$

In words, it's the **expected loss** of  $f$  on a new example  $(x, y)$  drawn randomly from  $P_{\mathcal{X} \times \mathcal{Y}}$ .

## Definition

A **Bayes prediction function**  $f^* : \mathcal{X} \rightarrow \mathcal{A}$  is a function that achieves the *minimal risk* among all possible functions:

$$f^* \in \arg \min_f R(f),$$

where the minimum is taken over all functions from  $\mathcal{X}$  to  $\mathcal{A}$ .

- The risk of a Bayes prediction function is called the **Bayes risk**.

# The Empirical Risk

Let  $\mathcal{D}_n = ((x_1, y_1), \dots, (x_n, y_n))$  be drawn i.i.d. from  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ .

## Definition

The **empirical risk** of  $f : \mathcal{X} \rightarrow \mathcal{A}$  with respect to  $\mathcal{D}_n$  is

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- But we saw that the **unconstrained** empirical risk minimizer overfits.
  - i.e. if we minimize  $\hat{R}_n(f)$  over **all functions**, we overfit.

# Constrained Empirical Risk Minimization

## Definition

A **hypothesis space**  $\mathcal{F}$  is a set of functions mapping  $\mathcal{X} \rightarrow \mathcal{A}$ .

- It is the collection of prediction functions we are choosing from.
- **Empirical risk minimizer** (ERM) in  $\mathcal{F}$  is

$$\hat{f}_n \in \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- From now on “ERM” always means “constrained ERM”.
- So we should always specify the hypothesis space when we're doing ERM.

# Example: Linear Least Squares Regression

## Setup

- Input space  $\mathcal{X} = \mathbf{R}^d$
  - Output space  $\mathcal{Y} = \mathbf{R}$
  - Action space  $\mathcal{Y} = \mathbf{R}$
  - Loss:  $\ell(\hat{y}, y) = (y - \hat{y})^2$
  - **Hypothesis space:**  $\mathcal{F} = \{f : \mathbf{R}^d \rightarrow \mathbf{R} \mid f(x) = w^T x, w \in \mathbf{R}^d\}$
- 
- Given data set  $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ ,
    - Let's find the ERM  $\hat{f} \in \mathcal{F}$ .

## Example: Linear Least Squares Regression

### Objective Function: Empirical Risk

The function we want to minimize is the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^n (w^T x_i - y_i)^2,$$

where  $w \in \mathbf{R}^d$  parameterizes the hypothesis space  $\mathcal{F}$ .

- Now let's think more generally...



## Gradient Descent for Empirical Risk - Scaling Issues

# Gradient Descent for Empirical Risk and Averages

- Suppose we have a hypothesis space of functions  $\mathcal{F} = \{f_w : \mathcal{X} \rightarrow \mathcal{A} \mid w \in \mathbf{R}^d\}$ 
  - Parameterized by  $w \in \mathbf{R}^d$ .
- ERM is to find  $w$  minimizing

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(f_w(x_i), y_i)$$

- Suppose  $\ell(f_w(x_i), y_i)$  is differentiable as a function of  $w$ .
- Then we can do gradient descent on  $\hat{R}_n(w)$ ...

## Gradient Descent: How does it scale with $n$ ?

- At every iteration, we compute the gradient at current  $w$ :

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^n \nabla_w \ell(f_w(x_i), y_i)$$

- We have to touch all  $n$  training points to take a single step.  $[O(n)]$
- Will this scale to “big data”?
- Can we make progress without looking at all the data?

# Stochastic Gradient Descent

# “Noisy” Gradient Descent

- We know gradient descent works.
- But the gradient may be slow to compute.
- What if we just use an estimate of the gradient?
- Turns out that can work fine.
- **Intuition:**
  - Gradient descent is an iterative procedure anyway.
  - At every step, we have a chance to recover from previous missteps.

# Minibatch Gradient

- The **full gradient** is

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^n \nabla_w \ell(f_w(x_i), y_i)$$

- It's an average over the **full batch** of data  $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .
- Let's take a random subsample of size  $N$  (called a **minibatch**):

$$(x_{m_1}, y_{m_1}), \dots, (x_{m_N}, y_{m_N})$$

- The **minibatch gradient** is

$$\nabla \hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^N \nabla_w \ell(f_w(x_{m_i}), y_{m_i})$$

- What can we say about the minibatch gradient? It's random. What's its expectation?

# Minibatch Gradient

- What's the expected value of the **minibatch gradient**?

$$\begin{aligned}\mathbb{E} \left[ \nabla \hat{R}_N(w) \right] &= \frac{1}{N} \sum_{i=1}^N \mathbb{E} [\nabla_w \ell(f_w(x_{m_i}), y_{m_i})] \\ &= \mathbb{E} [\nabla_w \ell(f_w(x_{m_1}), y_{m_1})] \\ &= \sum_{i=1}^n \mathbb{P}(m_1 = i) \nabla_w \ell(f_w(x_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla_w \ell(f_w(x_i), y_i) \\ &= \nabla \hat{R}_n(w)\end{aligned}$$

- Technical note:* We only assumed that each point in the minibatch is equally likely to be any of the  $n$  points in the batch – no independence needed. So still true if we're sampling without replacement. Still true if we sample one point randomly and reuse it  $N$  times.

# Minibatch Gradient Properties

- Minibatch gradient is an **unbiased estimator** for the [full] batch gradient:

$$\mathbb{E} \left[ \nabla \hat{R}_N(w) \right] = \nabla \hat{R}_n(w)$$

- The bigger the minibatch, the better the estimate.



# Minibatch Gradient – In Practice

- Tradeoffs of minibatch size:
  - Bigger  $N \implies$  Better estimate of gradient, but slower (more data to touch)
  - Smaller  $N \implies$  Worse estimate of gradient, but can be quite fast
- Even  $N = 1$  works, it's traditionally called **stochastic gradient descent** (SGD).
- These days, people use SGD to refer to minibatch SGD as well.
- If someone says “SGD”, you ask – “What’s your [mini]batch size?”, to avoid ambiguity.

# Terminology Review (Rough)

- **Gradient descent** or “full-batch” gradient descent
  - Use full data set of size  $n$  to determine step direction
- **Minibatch gradient descent**
  - Use a random subset of size  $N$  to determine step direction
  - Yoshua Bengio says<sup>1</sup>:
    - $N$  is typically between 1 and few hundred
    - $N = 32$  is a good default value
    - With  $N \geq 10$  we get computational speedup (per datum touched)
- **Stochastic gradient descent**
  - Minibatch with  $m = 1$ .
  - Use a single randomly chosen point to determine step direction.

But these days terminology isn't used so consistently, so always clarify the [mini]batch size.

---

<sup>1</sup>See Yoshua Bengio's “Practical recommendations for gradient-based training of deep architectures”  
<http://arxiv.org/abs/1206.5533>.

# Minibatch Gradient Descent

## Minibatch Gradient Descent (minibatch size $N$ )

- initialize  $w = 0$
- repeat
  - randomly choose  $N$  points  $\{(x_i, y_i)\}_{i=1}^N \subset \mathcal{D}_n$
  - $w \leftarrow w - \eta \left[ \frac{1}{N} \sum_{i=1}^N \nabla_w \ell(f_w(x_i), y_i) \right]$

# Stochastic Gradient Descent (SGD)

## Stochastic Gradient Descent

- initialize  $w = 0$
- repeat
  - randomly choose training point  $(x_i, y_i) \in \mathcal{D}_n$
  - $w \leftarrow w - \eta \underbrace{\nabla_w \ell(f_w(x_i), y_i)}_{\text{Grad(Loss on i'th example)}}$

## Step Size: In practice

- For SGD, fixed step size can work well in practice.
- *Typical approach*: Fixed step size reduced by constant factor whenever validation performance stops improving.
- But no theorem for this giving performance guarantees (to my knowledge).

# Robbins-Monro conditions

- For convergence guarantee, use decreasing step sizes (dampens noise in step direction).
- Let  $\eta_t$  be the step size at the  $t$ 'th step.

## Robbins-Monro Conditions

Many classical convergence results depend on the following two conditions:

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty \quad \sum_{t=1}^{\infty} \eta_t = \infty$$

- As fast as  $\eta_t = O\left(\frac{1}{t}\right)$  would satisfy this... but should be faster than  $O\left(\frac{1}{\sqrt{t}}\right)$ .
- A useful reference for practical techniques: Leon Bottou's "Tricks":  
<http://research.microsoft.com/pubs/192769/tricks-2012.pdf>

## Practical Comparison of GD vs SGD

---

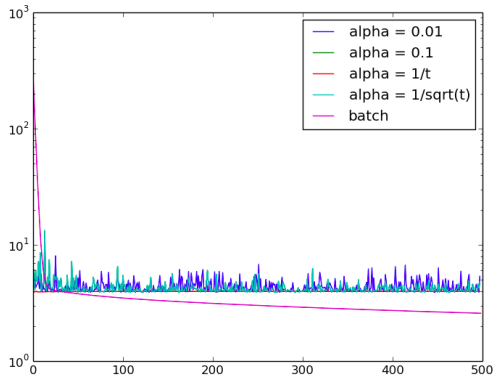
# Practical Comparison of GD vs SGD

- For huge data, GD isn't practical.
- In a theoretical sense, GD is much faster than SGD... (i.e. better convergence rates)
  - but most of that benefit happens once you're already pretty close to the solution
  - much faster to add an extra decimal place of accuracy on the minimum



# Does SGD Catch Up to GD?

- Ridge regression objective function value for GD and SGD with various stepsizes



- Why doesn't SGD catch up to batch GD? It does, just takes a **very** long time.
- Is it worth the wait? As we discuss in next module, probably not...