COMP 545: Advanced topics in optimization
From simple to complex ML systems

Lecture 4

Overview

- In the last lecture, we:
  - Talked how we can **accelerate** performance in optimization:
    Both in theory (**Newton's method**) and in practice (e.g., **SGD**)
  - Three practical cases: **acceleration, stochasticity, and variance-reduction**
  - Experiments showed advantage of the methods

Overview

- In the last lecture, we:
  - Talked how we can **accelerate** performance in optimization:
    Both in theory (**Newton's method**) and in practice (e.g., **SGD**)
  - Three practical cases: **acceleration, stochasticity, and variance-reduction**
  - Experiments showed advantage of the methods
- Until now, we introduced several hyper-parameters. In this lecture, we will:
  - Discuss what are these **hyper-parameters**
  - Discuss how we can set them correctly or at least heuristically
    in order to **maximize performance**

Gradient descent

- Reminder: how these step size selections are made
- For example, when the objective has **Lipschitz continuous gradients**:

$$f(x_{t+1}) \leq f(x_t) - \eta \left(1 - \frac{\eta L}{2}\right) \cdot \|f(x_t)\|_2^2$$

Gradient descent

- Reminder: how these step size selections are made
- For example, when the objective has **Lipschitz continuous gradients**:

$$f(x_{t+1}) \leq f(x_t) - \eta \underbrace{\left(1 - \frac{\eta L}{2}\right)}_{\text{Maximize this term}} \cdot \|f(x_t)\|_2^2$$

Gradient descent

- Reminder: how these step size selections are made
- For example, when the objective has **Lipschitz continuous gradients**:

$$f(x_{t+1}) \leq f(x_t) - \underbrace{\eta\left(1 - \frac{\eta L}{2}\right)}_{\text{Maximize this term}} \cdot \|f(x_t)\|_2^2$$

- Define $g(\eta) = -\eta\left(1 - \frac{\eta L}{2}\right)$; take derivative and set to zero.

Gradient descent

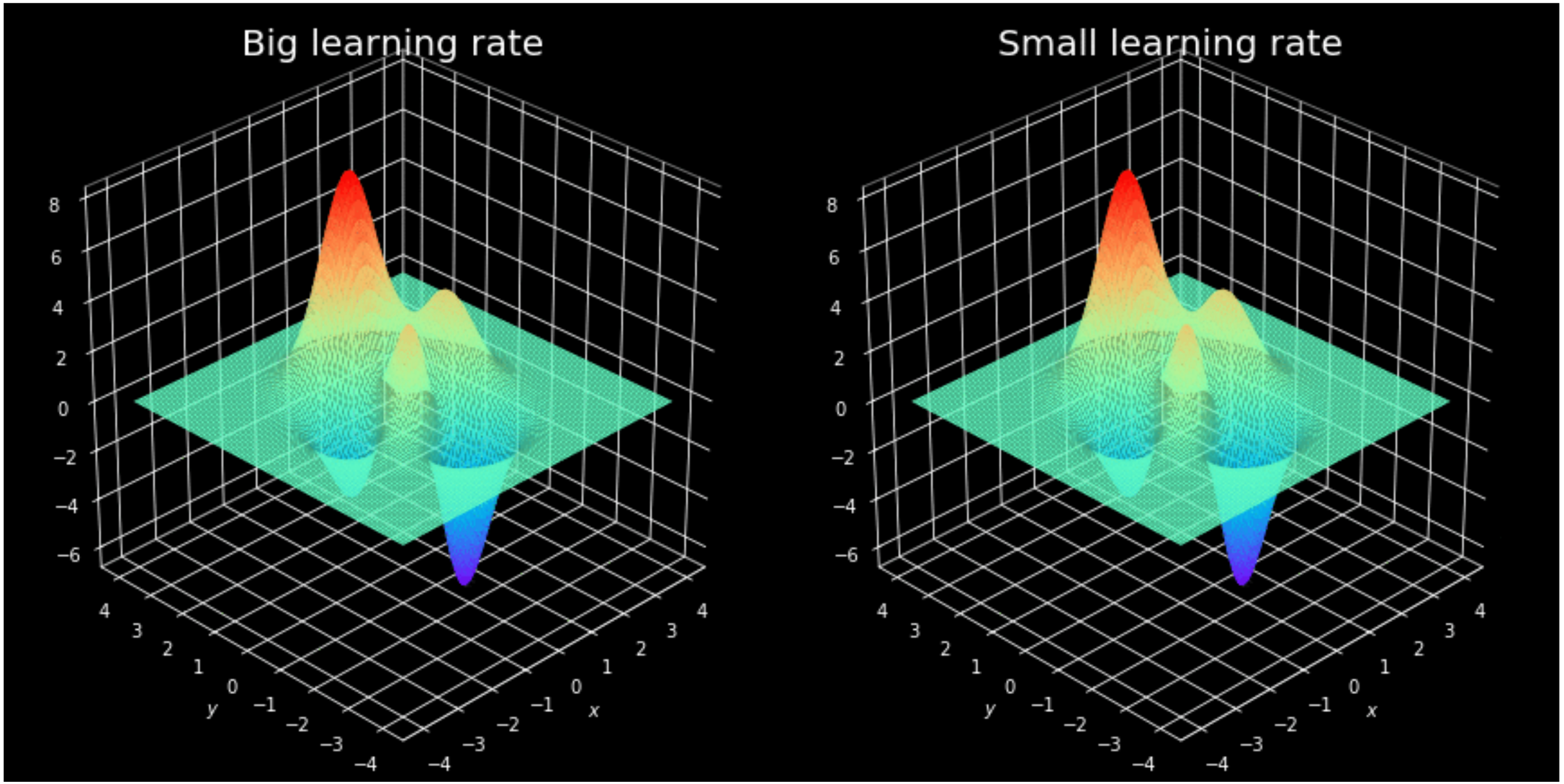
- Reminder: how these step size selections are made
- For example, when the objective has **Lipschitz continuous gradients**:

$$f(x_{t+1}) \leq f(x_t) - \underbrace{\eta\left(1 - \frac{\eta L}{2}\right)}_{\text{Maximize this term}} \cdot \|f(x_t)\|_2^2$$

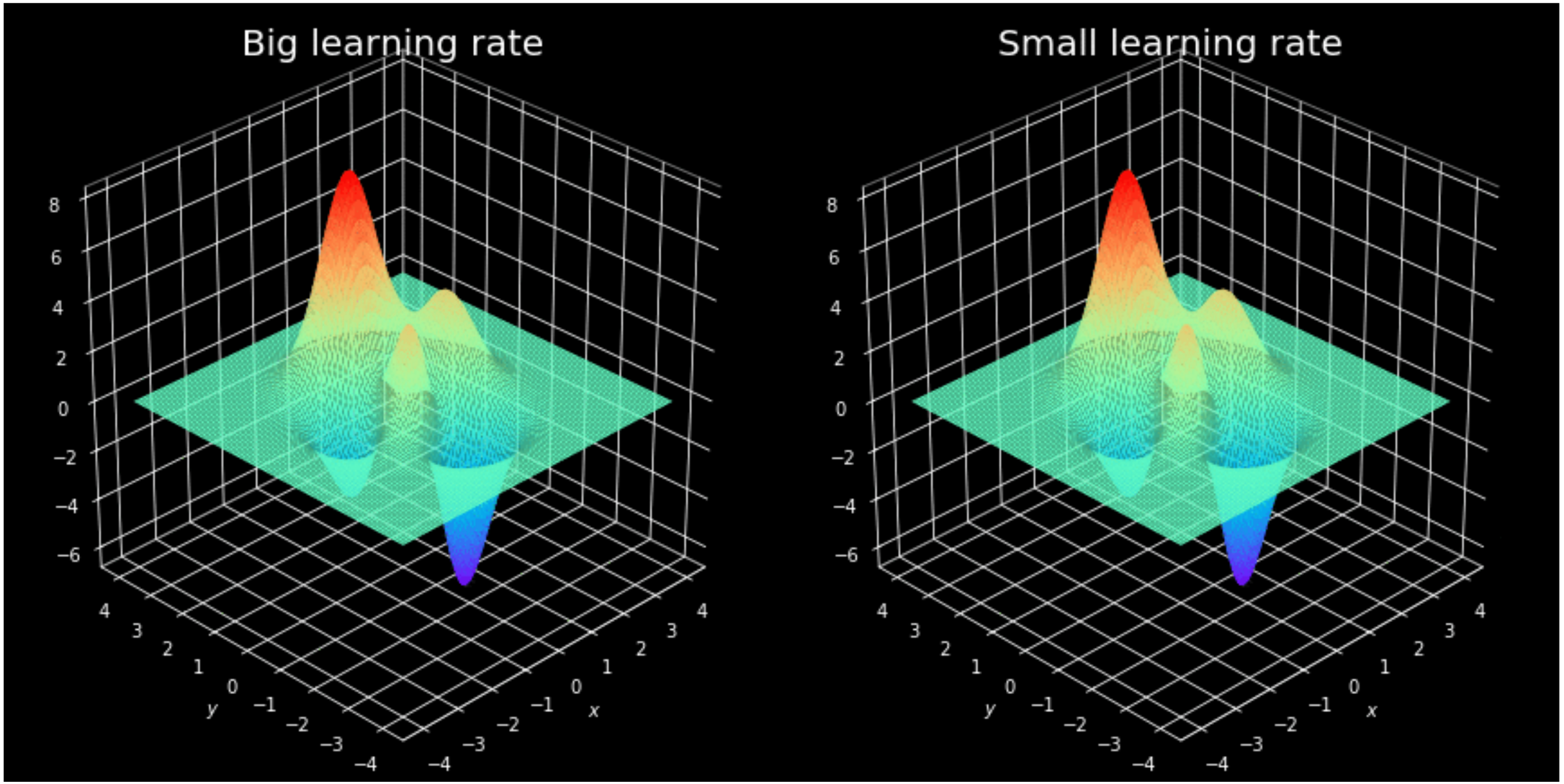
- Define $g(\eta) = -\eta\left(1 - \frac{\eta L}{2}\right)$; take derivative and set to zero.
- Using our theory, we obtain:

$$\eta = \frac{1}{L}$$

Gradient descent



Gradient descent



Gradient descent

- How well does $\eta = \frac{1}{L}$ perform?

Demo

Gradient descent

- How well does $\eta = \frac{1}{L}$ perform?

Demo

(After all, theory is not that useless..)

Gradient descent

- When the objective has Lipschitz continuous gradients **and** is strongly convex:

$$\|x_{t+1} - x^\star\|_2^2 \leq \left(1 - \frac{2\eta\mu L}{\mu + L}\right) \|x_t - x^\star\|_2^2 + \eta\left(\eta - \frac{2}{\mu + L}\right) \|\nabla f(x_t)\|_2^2$$

Gradient descent

- When the objective has Lipschitz continuous gradients **and** is strongly convex:

$$\|x_{t+1} - x^\star\|_2^2 \leq \left(1 - \frac{2\eta\mu L}{\mu + L}\right) \|x_t - x^\star\|_2^2 + \underbrace{\eta\left(\eta - \frac{2}{\mu + L}\right)}_{\text{Make negative}} \|\nabla f(x_t)\|_2^2$$

Gradient descent

- When the objective has Lipschitz continuous gradients **and** is strongly convex:

$$\|x_{t+1} - x^\star\|_2^2 \leq \left(1 - \frac{2\eta\mu L}{\mu+L}\right) \|x_t - x^\star\|_2^2 + \underbrace{\eta \left(\eta - \frac{2}{\mu+L}\right)}_{\text{Make negative}} \|\nabla f(x_t)\|_2^2$$

- We then obtain:

$$\eta < \frac{2}{\mu + L}$$

Demo

Projected gradient descent

- We briefly discussed the case of simply-constrained problems
  E.g.,

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & \|x\|_1 \leq \lambda \end{aligned} \qquad \text{or} \qquad \min_x \quad f(x) + \gamma \|x\|_1$$

Projected gradient descent

- We briefly discussed the case of simply-constrained problems
  E.g.,

$$\min_x \quad f(x) \qquad \text{or} \qquad \min_x \quad f(x) + \gamma\|x\|_1$$
$$\text{s.t.} \quad \|x\|_1 \le \lambda$$

Regularization

Projected gradient descent

- We briefly discussed the case of simply-constrained problems
  E.g.,

$$\min_x \quad f(x) \qquad \text{or} \qquad \min_x \quad f(x) + \gamma\|x\|_1$$
$$\text{s.t.} \quad \|x\|_1 \le \lambda$$

Regularization

- To connect with modern applications: **weight decay in neural networks**

Projected gradient descent

- We briefly discussed the case of simply-constrained problems
  E.g.,

$$\min_x \quad f(x) \qquad \text{or} \qquad \min_x \quad f(x) + \gamma\|x\|_1$$
$$\text{s.t.} \quad \|x\|_1 \le \lambda$$

Regularization

- To connect with modern applications: **weight decay in neural networks**

- How to set these parameters: **More art, rather than science.**
  Often, if we have some prior knowledge about the model, we use it

Projected gradient descent

- We briefly discussed the case of simply-constrained problems
  E.g.,

$$\min_x \quad f(x) \qquad \text{or} \qquad \min_x \quad f(x) + \gamma\|x\|_1$$
$$\text{s.t.} \quad \|x\|_1 \le \lambda$$

Regularization

- To connect with modern applications: **weight decay in neural networks**

- How to set these parameters: **More art, rather than science.**
  Often, if we have some prior knowledge about the model, we use it

- While non-convex objects are harder to optimize, they provide more intuition in tuning: sparsity k or norm-ball radius λ?

There is statistical theory for this selection

Initialization in gradient descent

- Usually, when we work in the convex domain, we (more or less) do not care about the initialization.

Initialization in gradient descent

- Usually, when we work in the convex domain, we (more or less) do not care about the initialization.

- Common practice is to set $x_0 = 0_{p \times 1}$

Demo

Initialization in gradient descent

- Usually, when we work in the convex domain, we (more or less) do not care about the initialization.

- Common practice is to set $x_0 = 0_{p\times 1}$

Demo

- Setting initial value to any other value could potentially decrease or increase the number of iterations

Initialization in gradient descent

- Usually, when we work in the convex domain, we (more or less) do not care about the initialization.

- Common practice is to set $x_0 = 0_{p\times 1}$

Demo

- Setting initial value to any other value could potentially decrease or increase the number of iterations

- We will see later in the course that initialization plays a key role in more involved scenaria

Newton's method

- The only hyper-parameter for convex methods is the step size

$$x_{t+1} = x_t - \eta \left(\nabla^2 f(x_t) \right)^{-1} \nabla f(x_t)$$

Newton's method

- The only hyper-parameter for convex methods is the step size

$$x_{t+1} = x_t - \eta \left(\nabla^2 f(x_t) \right)^{-1} \nabla f(x_t)$$

- For simple settings, even $\eta = 1$ works just fine

Newton's method

- The only hyper-parameter for convex methods is the step size

$$x_{t+1} = x_t - \eta \left(\nabla^2 f(x_t)\right)^{-1} \nabla f(x_t)$$

- For simple settings, even $\eta = 1$ works just fine
- If $\eta = 1$ fails, one can use Damped version:
  1. Start with $\eta < 1$; if algorithm does not converge, reduce η
  2. Otherwise, increase η until $\eta = 1$

Newton's method

- The only hyper-parameter for convex methods is the step size

$$x_{t+1} = x_t - \eta \left(\nabla^2 f(x_t)\right)^{-1} \nabla f(x_t)$$

- For simple settings, even $\eta = 1$ works just fine

- If $\eta = 1$ fails, one can use Damped version:

  1. Start with $\eta < 1$; if algorithm does not converge, reduce η
  2. Otherwise, increase η until $\eta = 1$

(Similar strategies for L-BFGS)

(Disclaimer: this is a very rough description. There are many rules that one can use to determine whether to increase/decrease the step size (backtracking line search, Armijo rule, using Newton's decrement, etc). There is also theory that characterizes when we leave the linear rate and we enter the quadratic convergence rate)

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD

(More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

Hyper-parameters

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD

(More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

Hyper-parameters

- For **step size**, there are various schedules:

(More often than not, step size is called "learning rate")

1. Constant step size

$$\eta_t \equiv \eta = c \quad (\text{usually } c = 0.1)$$

Demo

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD

(More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

Hyper-parameters

- For **step size**, there are various schedules:

(More often than not, step size is called "learning rate")

2. Inverse time decay

$$\eta_t = \frac{\eta}{1 + ct}$$

Step size (as if it was constant step size)

Constant c

Demo

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD

(More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

Hyper-parameters

- For **step size**, there are various schedules:

(More often than not, step size is called "learning rate")

3. Step decay

$$\eta_t = \eta \cdot q^{(1+e)/k}$$

Step size (as if it was constant step size)

E.g., q = 0.5, e is the epoch number, k = 10

Demo

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- Intuition:
  - We want **step size be large at the beginning** of the execution

(We are far away from the solution, thus we want to be aggressive)

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- Intuition:
  - We want **step size be large at the beginning** of the execution

    (We are far away from the solution, thus we want to be aggressive)
  - We want **step size be small near the (local) minimum**

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- Intuition:
  - We want **step size be large at the beginning** of the execution

    (We are far away from the solution, thus we want to be aggressive)
  - We want **step size be small near the (local) minimum**

    (We want to explore better around that area)

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- For **batch size**, there are only fixed-size schedules:
  - For example, batch sizes of 32, 64, 128 a quite common in practice

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- For **batch size**, there are only fixed-size schedules:
  - For example, batch sizes of 32, 64, 128 a quite common in practice
- Theoretically (convex), **mini-batch SGD decreases error** by a factor $|\mathcal{I}_t|$ compared to standard SGD

Stochastic gradient descent

- So far, the stochastic method we discussed is SGD (More to come!)

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- For **batch size**, there are only fixed-size schedules:
  - For example, batch sizes of 32, 64, 128 a quite common in practice
- Theoretically (convex), **mini-batch SGD decreases error** by a factor $|\mathcal{I}_t|$ compared to standard SGD
- For larger batch sizes, **mini-batch SGD allows more parallelism**

(To be discussed in future lectures)

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training

The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function.

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training

The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function. These minimizers are characterized by a significant number of large positive eigenvalues in $\nabla^2 f(x)$, and tend to generalize less well. In

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training

The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function. These minimizers are characterized by a significant number of large positive eigenvalues in $\nabla^2 f(x)$, and tend to generalize less well. In contrast, small-batch methods converge to *flat minimizers* characterized by having numerous small eigenvalues of $\nabla^2 f(x)$.

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

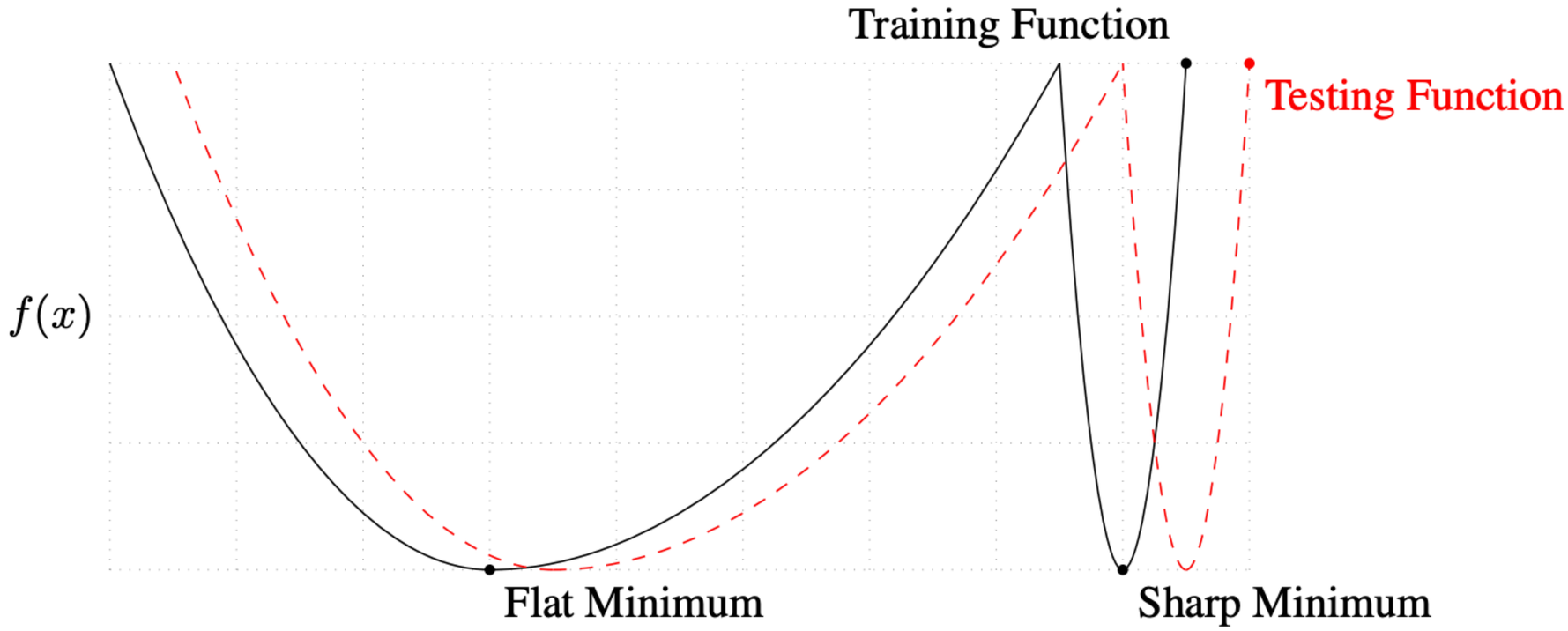
- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training

> The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function. These minimizers are characterized by a significant number of large positive eigenvalues in $\nabla^2 f(x)$, and tend to generalize less well. In contrast, small-batch methods converge to *flat minimizers* characterized by having numerous small eigenvalues of $\nabla^2 f(x)$. We have observed that the loss function landscape of deep neural networks is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers.

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

- Main drawback of SGD: sequential
- Also: mini-batches do not leave much space for parallelization
- A natural way to increase parallelization: large-batch training



Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)

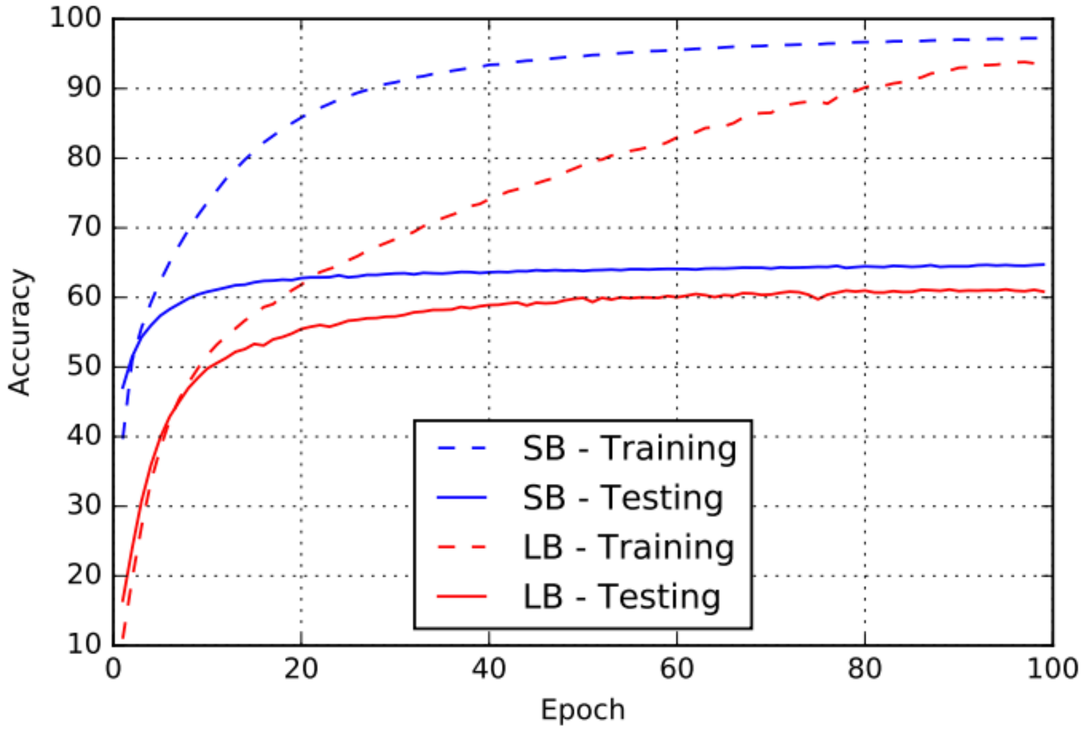
Name	Network Type	Architecture	Data set
F_1	Fully Connected	Section B.1	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	Section B.2	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	Section B.3	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	Section B.4	CIFAR-10
C_3	(Shallow) Convolutional	Section B.3	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	Section B.4	CIFAR-100

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

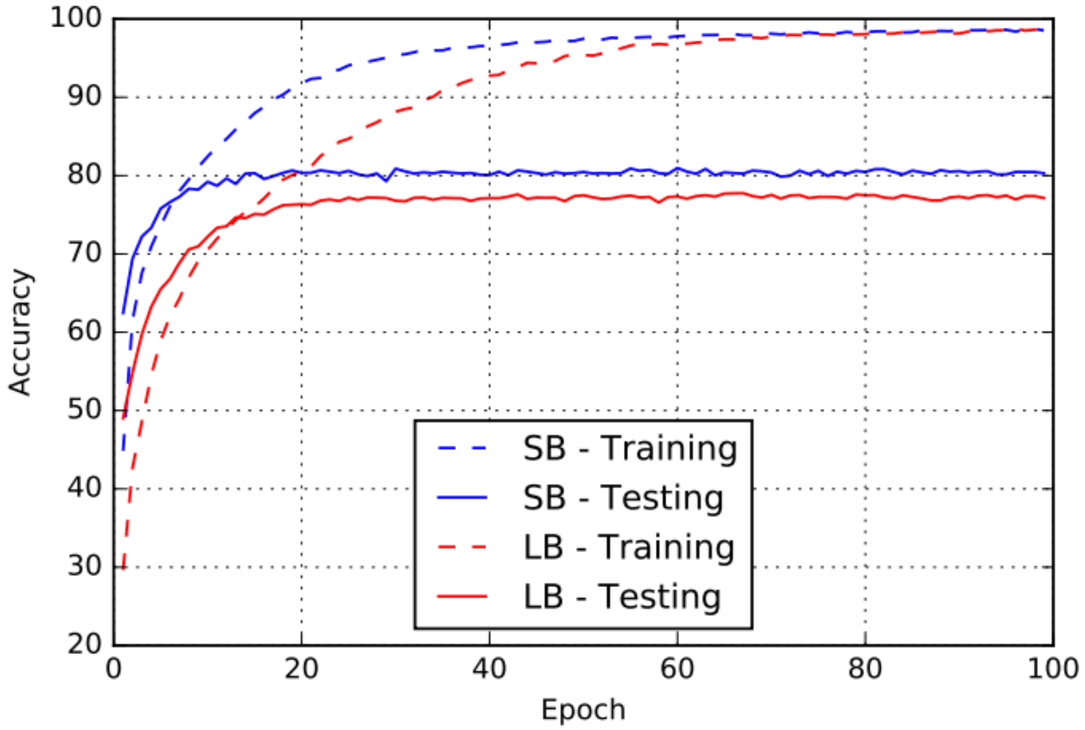
	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

Large-batch training

(On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima - Extra slides)



(a) Network F_2



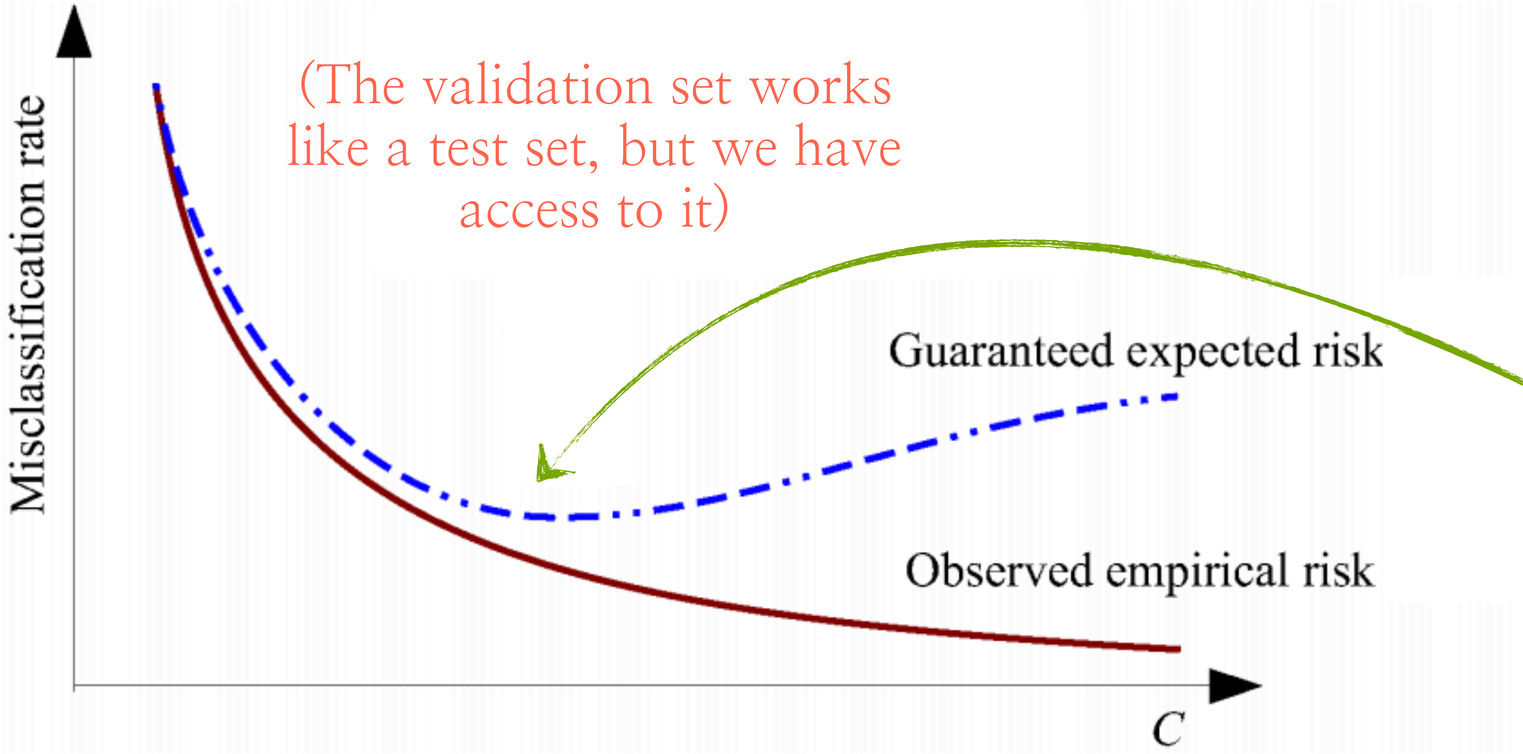
(b) Network C_1

Early stopping

(Requires a bit of detour)

- In machine learning, our aim is to make an algorithm perform well on training data (we study this already), but **also on unseen input.**

(This is the distinction between training error and test error)



- While training error keeps decreasing, our performance on unseen data deteriorates

- This motivates the use of a validation set

Early stopping

- In its simplest form, early stopping just requires **maintaining an (additional) copy of the model**: at every time tick, we compare the current model with the best model so far

Early stopping

- In its simplest form, early stopping just requires **maintaining an (additional) copy of the model**: at every time tick, we compare the current model with the best model so far

- Early stopping does not damage or alter the learning dynamics (compare with explicit regularization techniques that change the objective)

Early stopping

- In its simplest form, early stopping just requires **maintaining an (additional) copy of the model**: at every time tick, we compare the current model with the best model so far

- Early stopping does not damage or alter the learning dynamics (compare with explicit regularization techniques that change the objective)

- There is a connection of early stopping with that of ℓ_2-norm reg.

- However, it does not need hyper-parameter tuning

Early stopping

- Since early stopping requires part of the training data, we can use the following meta-algorithm:

Algorithm A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.

Early stopping

- Since early stopping requires part of the training data, we can use the following meta-algorithm:

Algorithm A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.

Early stopping

- Since early stopping requires part of the training data, we can use the following meta-algorithm:

Algorithm A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.

Early stopping

- Since early stopping requires part of the training data, we can use the following meta-algorithm:

Algorithm A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.

Early stopping

- Since early stopping requires part of the training data, we can use the following meta-algorithm:

Algorithm A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Conclusion

- This short lecture was about **hyper-parameter tuning**
- However, we only scratched the surface of hyper-parameter tuning; more to come in future lectures
- Key observation (even in these simplified scenaria) is that tuning can be even more important than which algorithm to use

Next lecture

- Sparse model selections and non-convex hard thresholding