Online Convex Optimization (OCO)

[1]ECE
Technical University of Crete

March 31, 2023

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \dots, w^{t-1}$

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$
- Functions $f^t()$ are **convex** for all t

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$
- Functions $f^t()$ are **convex** for all t (again, we'll define shortly)

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$
- Functions $f^t()$ are **convex** for all t (again, we'll define shortly)
- Our goal is to minimize the regret $Regret^T = \sum_{t=1}^{T} f^t(w^t) - \min_w \sum_{t=1}^{T} f^t(w)$

Introduction to Online Convex Optimization (OCO)

In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$
- Functions $f^t()$ are **convex** for all t (again, we'll define shortly)
- Our goal is to minimize the regret $Regret^T = \sum_{t=1}^{T} f^t(w^t) - \min_w \sum_{t=1}^{T} f^t(w)$ (Note: w is fixed in the "oracle" case.

Introduction to Online Convex Optimization (OCO)

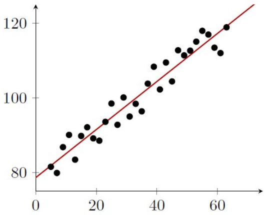
In this setup, the algorithm attempts to minimize/maximize a function $f^t(w^t)$ that changes at each round.

Basic Setup

At time t:

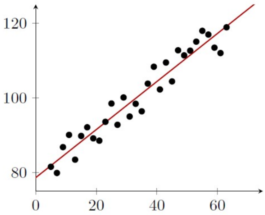
- algorithm must pick a $w^t \in S(\subset \mathcal{R}^k$ (w has dimension k)
- S is a **convex** set (we will see shortly what this means)
- w^t can be a (any) function of past choices $w^0, w^1, \ldots, w^{t-1}$ and past functions $f^0, f^1, \cdots, f^{t-1}$
- $f^t()$ is revealed and algorithm incurs cost $f^t(w^t)$
- Functions $f^t()$ are **convex** for all t (again, we'll define shortly)
- Our goal is to minimize the regret $Regret^T = \sum_{t=1}^{T} f^t(w^t) - \min_w \sum_{t=1}^{T} f^t(w)$ (Note: w is fixed in the "oracle" case. BUT we are allowed to pick the best w **after** we've seen all f^t)

Applications: Online Regression



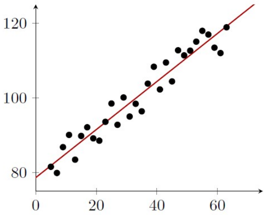
- Samples x^t, y^t are revealed sequentially

Applications: Online Regression



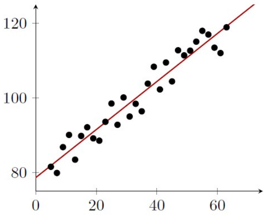
- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)

Applications: Online Regression



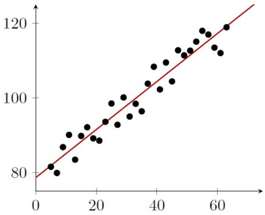
- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$

Applications: Online Regression



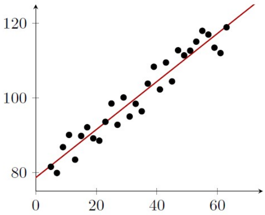
- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$ (e.g. $l(y, \hat{y}) = (y - \hat{y})^2$)

Applications: Online Regression



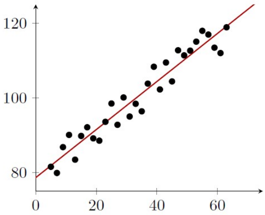
- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$ (e.g. $l(y, \hat{y}) = (y - \hat{y})^2$)
- E.g. for linear (2D) regression, $f^t(w_1^t, w_2^t) = (w_1^t \cdot x^t + w_2^t - y^t)^2$

Applications: Online Regression



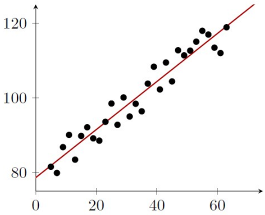
- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$ (e.g. $l(y, \hat{y}) = (y - \hat{y})^2$)
- E.g. for linear (2D) regression, $f^t(w_1^t, w_2^t) = (w_1^t \cdot x^t + w_2^t - y^t)^2$ (in this example $g(x^t) = w_1^t \cdot x^t + w_2^t$ is linear)

Applications: Online Regression



- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$ (e.g. $l(y, \hat{y}) = (y - \hat{y})^2$)
- E.g. for linear (2D) regression, $f^t(w_1^t, w_2^t) = (w_1^t \cdot x^t + w_2^t - y^t)^2$ (in this example $g(x^t) = w_1^t \cdot x^t + w_2^t$ is linear)
  - The convex set $S = \mathcal{R}^k$ is just the real numbers (no constraints).

Applications: Online Regression



- Samples x^t, y^t are revealed sequentially (x^t : vector of "features", y^t : true label)
- Objective: Given x^t we need to find a function $\hat{y}^t = g(x^t)$ to minimize a loss function $l(y^t, \hat{y}^t)$ (e.g. $l(y, \hat{y}) = (y - \hat{y})^2$)
- E.g. for linear (2D) regression, $f^t(w_1^t, w_2^t) = (w_1^t \cdot x^t + w_2^t - y^t)^2$ (in this example $g(x^t) = w_1^t \cdot x^t + w_2^t$ is linear)
  - The convex set $S = \mathcal{R}^k$ is just the real numbers (no constraints).
- Observe that the "oracle/optimal" performance here is just the performance of **offline regression** (what we'd pick if we had all samples available immediately).

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
    - Each expert incurs a loss l_i^t
    - Multiplicative Weights (or Hedge) gives probabilities of picking each expert:

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^k w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t
- The popularity x_i^t of video i that day affects the average *cache hit ratio* that day

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t
- The popularity x_i^t of video i that day affects the average *cache hit ratio* that day $r_i^t = \sum_i x_i^t \cdot w_i^t$

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t
- The popularity x_i^t of video i that day affects the average *cache hit ratio* that day $r_i^t = \sum_i x_i^t \cdot w_i^t$
- Video i has size s_i Bytes and cache fits a total of C Bytes

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t
- The popularity x_i^t of video i that day affects the average *cache hit ratio* that day $r_i^t = \sum_i x_i^t \cdot w_i^t$
- Video i has size s_i Bytes and cache fits a total of C Bytes $\Rightarrow \sum_i w_i^t \cdot s_i \leq C$.

Other Applications

Experts

- Function to be minimized: $f^t(w^t) = \sum_{i=1}^{k} w_i^t \cdot l_i^t$
- Algorithm chooses w_i^t, at every round t
- This is just the previous experts setup!
  - Each expert incurs a loss l_i^t
  - Multiplicative Weights (or Hedge) gives probabilities of picking each expert: $\Rightarrow w_i^t = p_i^t$ (from experts lecture)
- The convex set now is $S = \{w_i^t \in [0,1], \sum_i w_i = 1\}$ (aka. "simplex")

Other Applications: Online Caching Problem (see 1st lecture for more)

- On day/round t: cache video i with probability w_i^t
- The popularity x_i^t of video i that day affects the average *cache hit ratio* that day $r_i^t = \sum_i x_i^t \cdot w_i^t$
- Video i has size s_i Bytes and cache fits a total of C Bytes $\Rightarrow \sum_i w_i^t \cdot s_i \leq C$.
- The above defines our (convex) set S (together with $w_i \in [0,1]$)

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0, 1] : tx + (1 - t)y \in S$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables:

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables:
  $\{x_1, x_2 \geq 0\}$,

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0, 1] : tx + (1 - t)y \in S$
- Examples with two variables:
  $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\},$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables:
  $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0, 1] : tx + (1 - t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)
- Examples with two variables:

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

<u>**Convex Sets:**</u> What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables:
  $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

<u>**Convex Function:**</u> What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)
- Examples with two variables:
  $f(x) = 2x_1 + 3x_2,$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)
- Examples with two variables: $f(x) = 2x_1 + 3x_2, f(x) = x_1^2 + 3x_2^2 + xy + 5,$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}} \; f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0, 1] : tx + (1 - t)y \in S$
- Examples with two variables:
  $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)
- Examples with two variables:
  $f(x) = 2x_1 + 3x_2, f(x) = x_1^2 + 3x_2^2 + xy + 5, f(x) = 3e^{x_1^2 + x_2^2} + e^{-2x_1}$

First things first: Solving (non-online) Convex Optimization Problems

Reminder: Convex Optimization Setup

- $\underset{x \in S}{\text{minimize}}\ f(x)$
- x: control variables, $f(x)$: our (convex) objective, S: a set of (convex) constraints for x

Convex Sets: What is it?

- Def.: S is convex iff, $\forall x, y \in S, t \in [0,1] : tx + (1-t)y \in S$
- Examples with two variables: $\{x_1, x_2 \geq 0\}, \{x_1 + x_2 \leq 1\}, \{x_1^2 + x_2^2 \leq 3\}$

Convex Function: What is it?

- Def.: f is convex in S iff, $\forall x, y \in S : f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$ (note: all are vectors)
- Examples with two variables: $f(x) = 2x_1 + 3x_2, f(x) = x_1^2 + 3x_2^2 + xy + 5, f(x) = 3e^{x_1^2 + x_2^2} + e^{-2x_1}$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully!

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)}-3), 2(x_2^{(t-1)}-2))$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)} - 3), 2(x_2^{(t-1)} - 2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10-3), 2(10-2))$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)}-3), 2(x_2^{(t-1)}-2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10-3), 2(10-2)) = (3, 6)$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)} - 3), 2(x_2^{(t-1)} - 2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10-3), 2(10-2)) = (3, 6)$
- $(x_1^{(2)}, x_2^{(2)}) = (3, 4)$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1, x_2}{\text{minimize}}\ 2(x_1 - 3)^2 + (x_2 - 2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)} - 3), 2(x_2^{(t-1)} - 2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10 - 3), 2(10 - 2)) = (3, 6)$
- $(x_1^{(2)}, x_2^{(2)}) = (3, 4)$
- $(x_1^{(3)}, x_2^{(3)}) = (3, 3)$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)} - 3), 2(x_2^{(t-1)} - 2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10-3), 2(10-2)) = (3, 6)$
- $(x_1^{(2)}, x_2^{(2)}) = (3, 4)$
- $(x_1^{(3)}, x_2^{(3)}) = (3, 3) \quad \rightarrow (x_1^{(4)}, x_2^{(4)}) = (3, 2.5) \rightarrow \dots$

Convex Optimization: Gradient Descent

- Consider the toy problem: $\underset{x_1,x_2}{\text{minimize}}\ 2(x_1-3)^2+(x_2-2)^2$

Gradient Descent Algorithm

- Start with (any) initial point $x^{(0)}$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat: $x^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$ until convergence
- α_t is the **learning rate** and must be chosen carefully! (again in RL)

For the above example:

- for our example (let $\alpha_t = 1$), the update is:
  $(x_1^{(t)}, x_2^{(t)}) = (x_1^{(t-1)}, x_2^{(t-1)}) - \alpha_t(4(x_1^{(t-1)}-3), 2(x_2^{(t-1)}-2))$
- $(x_1^{(1)}, x_2^{(1)}) = (10, 10) - 0.25(4(10-3), 2(10-2)) = (3, 6)$
- $(x_1^{(2)}, x_2^{(2)}) = (3, 4)$
- $(x_1^{(3)}, x_2^{(3)}) = (3, 3) \quad \rightarrow (x_1^{(4)}, x_2^{(4)}) = (3, 2.5) \rightarrow \ldots$
- What do you observe?

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast?

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$
- But what if x is constrained in S?

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$
- But what if x is constrained in S?

Projected Gradient Descent

- Start with (any) initial point $x^(0)$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$
- But what if x is constrained in S?

Projected Gradient Descent

- Start with (any) initial point $x^(0)$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat until convergence
  - $y^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$
- But what if x is constrained in S?

Projected Gradient Descent

- Start with (any) initial point $x^(0)$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat until convergence
  - $y^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$
  - $x^{(t)} = \underset{x \in S}{argmin} \|x - y^{(t)}\|_2$

Convex Optimization (cnt'd)

Theorem

If $f(x)$ convex and differentiable, and L is its maximum eigenvalue, gradient descent with $\alpha_t = 1/L$ **always converges to the global minimum** of f.

- How fast? Depends on other properties of $f(x)$
- But what if x is constrained in S?

Projected Gradient Descent

- Start with (any) initial point $x^(0)$, e.g. $(x_1^{(0)}, x_2^{(0)}) = (10, 10)$
- repeat until convergence
  - $y^{(t)} = x^{(t-1)} - \alpha_t \nabla f(x^{(t-1)})$
  - $x^{(t)} = \underset{x \in S}{argmin} \|x - y^{(t)}\|_2$ (project back in S)

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough!

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$
- Let $G(\lambda) = min_x L(x, \lambda)$

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$
- Let $G(\lambda) = min_x L(x, \lambda)$
- If $f(x), g(x)$ are convex: $\underset{\lambda \geq 0}{max}\, G(\lambda) = \underset{x:g(x) \leq 0}{min} f(x)$

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$
- Let $G(\lambda) = min_x L(x, \lambda)$
- If $f(x), g(x)$ are convex: $\underset{\lambda \geq 0}{max}\, G(\lambda) = \underset{x:g(x) \leq 0}{min} f(x)$

We can choose to solve the dual problem $\underset{\lambda \geq 0}{max}\, G(\lambda)$ instead. Why?

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$
- Let $G(\lambda) = min_x L(x, \lambda)$
- If $f(x), g(x)$ are convex: $\underset{\lambda \geq 0}{max}\, G(\lambda) = \underset{x: g(x) \leq 0}{min}\, f(x)$

We can choose to solve the dual problem $\underset{\lambda \geq 0}{max}\, G(\lambda)$ instead. Why?

- Dual might be (much) easier to optimize than original ("Primal")

Lagrange Multipliers and Dual Optimization

- Consider the problem: $\underset{x}{\text{minimize}}\ f(x)$, subject to $g(x) \leq 0$
- Projection to $S = \{g(x) \leq 0\}$ can be very tough! (remember: projection is a quadratic minimization problem)

Lagrange Dual

- Lagrangian $L(x, \lambda) = f(x) + \lambda g(x)$
- Note: $L(x, \lambda) \leq f(x)$ for all feasible $x \in S$
- Let $G(\lambda) = min_x L(x, \lambda)$
- If $f(x), g(x)$ are convex: $\underset{\lambda \geq 0}{max}\, G(\lambda) = \underset{x:g(x) \leq 0}{min} f(x)$

We can choose to solve the dual problem $\underset{\lambda \geq 0}{max}\, G(\lambda)$ instead. Why?

- Dual might be (much) easier to optimize than original ("Primal")
- Projection is *definitely* easier!

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset:

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset:
$\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) =$

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above → but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset:
  $\min_{w} \sum_{i=1}^{N} f_i(x_i|w) = \sum_{i=1}^{N}(h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model ($h(x_i|w)$) and the true value y_i,

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset: $\min_{w} \sum_{i=1}^{N} f_i(x_i|w) = \sum_{i=1}^{N}(h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model ($h(x_i|w)$) and the true value y_i, **for all N samples**

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset: $\min_w \sum_{i=1}^N f_i(x_i|w) = \sum_{i=1}^N (h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model ($h(x_i|w)$) and the true value y_i, **for all** N **samples**
- $\nabla_w f(x)$: taking a gradient of our objective means

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) = \sum_{i=1}^{N}(h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model ($h(x_i|w)$) and the true value y_i, **for all N samples**
- $\nabla_w f(x)$: taking a gradient of our objective means $\sum_{i=1}^{N} \nabla_w (h(x_i|w) - y_i)^2$:

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset: $\min_w \sum_{i=1}^N f_i(x_i|w) = \sum_{i=1}^N (h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model $(h(x_i|w))$ and the true value y_i, **for all N samples**
- $\nabla_w f(x)$: taking a gradient of our objective means $\sum_{i=1}^N \nabla_w (h(x_i|w) - y_i)^2$: taking a gradient of the error for every sample

Convex Optimization for Machine Learning (ML)

- Most traditional ML problems (regression, SVM, etc.) are convex optimization problems
- Even modern ML problems (e.g. Deep Neural Network training) can be treated with convex methods like the above $\rightarrow$ but convergence is to **local optimal** (at best)
- BUT, the "learning" in machine learning introduces some extra complications.

ML Concern 1: Full Gradient is Expensive

- Assume we fit a regression model to a large dataset: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) = \sum_{i=1}^{N} (h(x_i|w) - y_i)^2$
- i.e. we minimize the prediction error of our model ($h(x_i|w)$) and the true value y_i, **for all N samples**
- $\nabla_w f(x)$: taking a gradient of our objective means $\sum_{i=1}^{N} \nabla_w (h(x_i|w) - y_i)^2$: taking a gradient of the error for every sample $\rightarrow$ possibly millions of gradients to calculate (per step)!

Full Gradient Expensive $\rightarrow$ Stochastic Gradient Descent

Assume a problem like this: $\min_{w} \sum_{i=1}^{N} f_i(x_i|w)$

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\min_{w} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\min_{w} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
  - pick index i_t uniformly in $\{1, N\}$

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\min_{w} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
    - pick index i_t uniformly in $\{1, N\}$
    - $w^{(t)} = w^{(t-1)} - \alpha_t \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
  - pick index i_t uniformly in $\{1, N\}$
  - $w^{(t)} = w^{(t-1)} - \alpha_t \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Important Remarks:

- Requires diminishing learning rate α_t to converge (e.g. $\alpha_t \sim 1/t$)

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
  - pick index i_t uniformly in $\{1, N\}$
  - $w^{(t)} = w^{(t-1)} - \alpha_t \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Important Remarks:

- Requires diminishing learning rate α_t to converge (e.g. $\alpha_t \sim 1/t$)
- **Mini-batch**: is simply picking randomly B samples (indices) out of N, instead of just 1

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
  - pick index i_t uniformly in $\{1, N\}$
  - $w^{(t)} = w^{(t-1)} - \alpha_t \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Important Remarks:

- Requires diminishing learning rate α_t to converge (e.g. $\alpha_t \sim 1/t$)
- **Mini-batch**: is simply picking randomly B samples (indices) out of N, instead of just 1
- $w^{(t)} = w^{(t-1)} - \alpha_t \sum_{i_t \in B} \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Full Gradient Expensive → Stochastic Gradient Descent

Assume a problem like this: $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

Observe that we denote our control variable as w here (not x).

Stochastic Gradient Descent (SGD)

- Start with (any) initial point $w^{(0)}$
- Repeat until convergence:
  - pick index i_t uniformly in $\{1, N\}$
  - $w^{(t)} = w^{(t-1)} - \alpha_t \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$

Important Remarks:

- Requires diminishing learning rate α_t to converge (e.g. $\alpha_t \sim 1/t$)
- **Mini-batch**: is simply picking randomly B samples (indices) out of N, instead of just 1
- $w^{(t)} = w^{(t-1)} - \alpha_t \sum_{i_t \in B} \nabla_w f_{i_t}(x_{i_t}|w^{(t-1)})$
- We will see SGD and mini-batch A LOT in Reinforcement Learning

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML:

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$.

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w)$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is Regularization

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$
- $R(w)$: is a **regularizer**

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$
- $R(w)$: is a **regularizer**
  - L_2 regularizer: $R(w) = \|w\|_2^2 = \sum_{j=1}^{k} w_j^2$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \to$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$
- $R(w)$: is a **regularizer**
    - L_2 regularizer: $R(w) = \|w\|_2^2 = \sum_{j=1}^{k} w_j^2$
    - L_1 regularizer: $R(w) = \|w\|_1 = \sum_{j=1}^{k} |w_j|$

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ \ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$
- $R(w)$: is a **regularizer**
  - L_2 regularizer: $R(w) = \|w\|_2^2 = \sum_{j=1}^{k} w_j^2$
  - L_1 regularizer: $R(w) = \|w\|_1 = \sum_{j=1}^{k} |w_j|$
  - We'll see others too

ML Concern 2: Overfitting

- Traditional (convex) optimization: find global minimum of function $f(w)$
- Optimization in ML: minimizine the error over all **training data**
- Hence, we are finding w_{opt} by minimizing $f_{train}(w)$. BUT, we are evaluating w_{opt} using a **different function** $f_{test}(w_{opt})$.
- Finding $\underset{w}{min}\ f_{train}(w) \rightarrow$ (can lead to) overfitting

Solution is **Regularization**

- $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w) + \eta R(w)$ instead of $\underset{w}{min} \sum_{i=1}^{N} f_i(x_i|w)$
- $R(w)$: is a **regularizer**
  - L_2 regularizer: $R(w) = \|w\|_2^2 = \sum_{j=1}^{k} w_j^2$
  - L_1 regularizer: $R(w) = \|w\|_1 = \sum_{j=1}^{k} |w_j|$
  - We'll see others too
- Forces model weights to be small (or zero - in L_1) $\Rightarrow$ improves overfitting (η is a hyperparameter)

Other Optimization Tricks

- Adagrad, Adam, RMSProp ("Optimizers" in PyTorch - e.g. "torch.optim.Adam()")

Other Optimization Tricks

- Adagrad, Adam, RMSProp (“Optimizers” in PyTorch - e.g. “torch.optim.Adam()”)
  - Improving (stochastic) gradient descent (sometimes heuristic)

Other Optimization Tricks

- Adagrad, Adam, RMSProp (“Optimizers” in PyTorch - e.g. “torch.optim.Adam()”)
  - Improving (stochastic) gradient descent (sometimes heuristic)
  - They (mainly) optimize the learning rates α_t seperately for each weight/variable w_i

Other Optimization Tricks

- Adagrad, Adam, RMSProp ("Optimizers" in PyTorch - e.g. "torch.optim.Adam()")
  - Improving (stochastic) gradient descent (sometimes heuristic)
  - They (mainly) optimize the learning rates α_t seperately for each weight/variable w_i
- Acceleration/Momentum:
  - Also attempt to speed up gradient descent (offer guarantees)

Other Optimization Tricks

- Adagrad, Adam, RMSProp ("Optimizers" in PyTorch - e.g. "torch.optim.Adam()")
  - Improving (stochastic) gradient descent (sometimes heuristic)
  - They (mainly) optimize the learning rates α_t seperately for each weight/variable w_i
- Acceleration/Momentum:
  - Also attempt to speed up gradient descent (offer guarantees)
  - They introduce some memory into the calculation of gradients

Other Optimization Tricks

- Adagrad, Adam, RMSProp (“Optimizers” in PyTorch - e.g. “torch.optim.Adam()”)
  - Improving (stochastic) gradient descent (sometimes heuristic)
  - They (mainly) optimize the learning rates α_t seperately for each weight/variable w_i
- Acceleration/Momentum:
  - Also attempt to speed up gradient descent (offer guarantees)
  - They introduce some memory into the calculation of gradients
  - Adam implements momentum

Back to Online (Convex Optimization)

- Before: minimize **once** (static/offline) $\rightarrow \min_{w} f(w)$

Back to Online (Convex Optimization)

- Before: minimize **once** (static/offline) $\rightarrow \min_{w} f(w)$
- Now: minimize **at very round t** (online/dynamic): $\min_{w^t} f^t(w^t)$

Back to Online (Convex Optimization)

- Before: minimize **once** (static/offline) $\rightarrow \min\limits_{w} f(w)$
- Now: minimize **at very round t** (online/dynamic): $\min\limits_{w^t} f^t(w^t)$
- How to choose w^t??

Back to Online (Convex Optimization)

- Before: minimize **once** (static/offline) $\rightarrow \min_{w} f(w)$
- Now: minimize **at very round t** (online/dynamic): $\min_{w^t} f^t(w^t)$
- How to choose w^t??
- Seems reasonable to try some gradient descent type of algorithm (we'll get back to this soon)

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \; \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_w \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min\limits_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min\limits_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1 \Rightarrow$ loss is then $f^{2k+1}(1) = 1$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1 \Rightarrow$ loss is then $f^{2k+1}(1) = 1$
- Even steps: sum minimizes for $w = -1$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min\limits_{w} \; \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1 \Rightarrow$ loss is then $f^{2k+1}(1) = 1$
- Even steps: sum minimizes for $w = -1 \Rightarrow$ loss is then $f^{2k}(1) = 1$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1 \Rightarrow$ loss is then $f^{2k+1}(1) = 1$
- Even steps: sum minimizes for $w = -1 \Rightarrow$ loss is then $f^{2k}(1) = 1$
- Oracle can make its total loss 0 up to any round t, by picking $w = 0$

OCO: Follow the Leader

Algo 1: Follow The Leader (FTL)

- $w^t = \min\limits_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w)$
- Reasonable: would have 0 regret, if $f^t(w)$ was included in the sum

Counterexample

- Let $S = [-1, 1]$
- Let $f^1(w) = \frac{w}{2}$, $f^{2k}(w) = -w$, $f^{2k+1}(w) = w$, $\forall k \in \mathcal{N}$
- $(t > 3)$ At odd steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = -\frac{w}{2}$
- $(t > 3)$ At even steps: $\sum_{t'=1}^{t-1} f^{t'}(w) = \frac{w}{2}$
- Odd steps: sum minimizes for $w = 1 \Rightarrow$ loss is then $f^{2k+1}(1) = 1$
- Even steps: sum minimizes for $w = -1 \Rightarrow$ loss is then $f^{2k}(1) = 1$
- Oracle can make its total loss 0 up to any round t, by picking $w = 0$
- Regret of FTL is unfortunately T

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min_{w} \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable “swings” too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min\limits_{w} \;\; \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min\limits_{w} \;\; \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta} w^2$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min\limits_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = \eta/2$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable “swings” too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min\limits_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = \eta/2$
- Even t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = \frac{w}{2} + \frac{1}{2\eta} w^2$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min\limits_{w} \;\; \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = \eta/2$
- Even t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = \frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = -\eta/2$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min_{w} \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta}\sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta}\sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta}w^2 \rightarrow$ min at $w^t = \eta/2$
- Even t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = \frac{w}{2} + \frac{1}{2\eta}w^2 \rightarrow$ min at $w^t = -\eta/2$
- If η is small enough, $w^t \rightarrow 0, \forall t$

Follow The Regularized Leader

- The main problem with FTL in the above example is that the control variable "swings" too violently between extreme values
- What if we introduced *regularization* to control changes of w from round to round?

Algo 2: Follow The Regularized Leader (FTRL)

- $w^t = \min_w \ \sum_{t'=1}^{t-1} f^{t'}(w) + R(w)$
- Euclidean Regularizer: $R(W) = \frac{1}{2\eta} \sum_{i=1}^{k} w_i^2$
- Entropy Regularizer: $R(W) = \frac{1}{\eta} \sum_{i=1}^{k} w_i \log w_i$ (common for w probabilities)

- Previous (counter)example:
- Odd t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = -\frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = \eta/2$
- Even t: $\sum_{t'=1}^{t-1} f^{t'}(w) + R(w) = \frac{w}{2} + \frac{1}{2\eta} w^2 \rightarrow$ min at $w^t = -\eta/2$
- If η is small enough, $w^t \rightarrow 0, \forall t$ (which was optimal)

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S \left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1})\right]$

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S \left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1}) \right]$
- (where $\Pi_S[x]$ denotes projection of vector x onto convex set S)

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S \left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1}) \right]$
- (where $\Pi_S[x]$ denotes projection of vector x onto convex set S)
- Turns out that this is **FTRL with Euclidean Regularizer**.

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S\left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1})\right]$
- (where $\Pi_S[x]$ denotes projection of vector x onto convex set S)
- Turns out that this is **FTRL with Euclidean Regularizer**.

- Q: What about FTRL with Entropy Regularizer?

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S\left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1})\right]$
- (where $\Pi_S[x]$ denotes projection of vector x onto convex set S)
- Turns out that this is **FTRL with Euclidean Regularizer**.

- Q: What about FTRL with Entropy Regularizer?
- A: This is a very popular/fast algorithm called **Online Mirror Descent**

Online Gradient Descent (as a subcase of FTRL)

Algo 3: Online Gradient Descent [Zinkevich et al., 2003]

- Start with (any) initial point w^0 and function f^0
- At round t: $w^t = w^{t-1} - \alpha_t f^{t-1}(w^{t-1})$

Remarks:

- (perhaps) Surprisingly, this simple algorithm has $O(\sqrt{T})$ regret
- Generalizes to constrained problems:
  $w^t = \Pi_S \left[w^{t-1} - \alpha_t f^{t-1}(w^{t-1})\right]$
- (where $\Pi_S[x]$ denotes projection of vector x onto convex set S)
- Turns out that this is **FTRL with Euclidean Regularizer**.

- Q: What about FTRL with Entropy Regularizer?
- A: This is a very popular/fast algorithm called **Online Mirror Descent**
- FTRL achieves order optimal $O(\sqrt{T})$ regret.