Business Analytics
Convex Optimization in Machine Learning

Prof. Bichler

Decision Sciences & Systems

Department of Informatics

Technische Universität München

Course Content

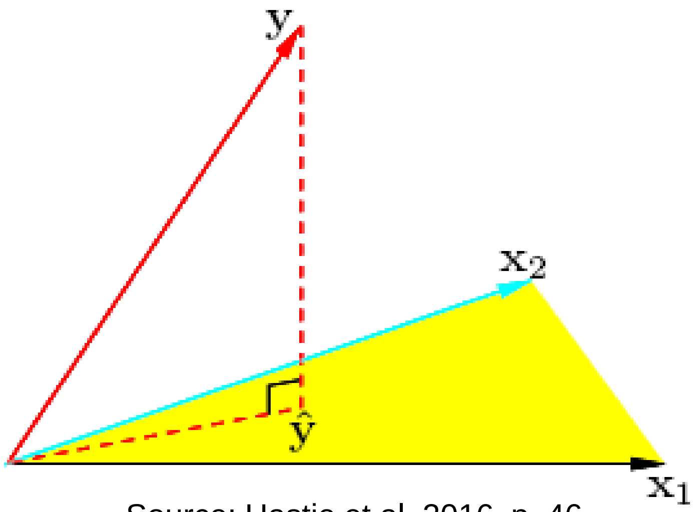
- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- Ensemble Methods and Clustering
- High-Dimensional Problems
- Association Rules and Recommenders
- Neural Networks
- **Convex Optimization in Machine Learning**

Recap Linear Regression:
Quadratic Optimization in Least Squares Estimation

- Least square estimates in $\mathbb{R}^n$
- We minimize the squared distance between observed and estimated values: RSS(β)=$||y - \mathbf{X}\beta||^2$, s.t. residual vector y – ŷ is orthogonal to this subspace $\mathbf{X}$.
- We found an analytical solution:$\hat{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$

Definition (Projection):
The set $X \subset \mathbb{R}^n$ is non-empty, closed and convex. For a fixed $y \in \mathbb{R}^n$ we search a point $\hat{y} \in X$, with the smallest distance to y (wrt. the Euclidean norm), i.e. we solve the minimization problem

$$P_X(y) = \min_{\hat{y}\in X} ||y - \hat{y}||^2$$



Source: Hastie et al. 2016, p. 46

Recap Logistic Regression:
Maximizing the LL Function in a Logistic Regression

$$\beta = \text{argmax}_\beta LL(\beta) = \text{argmax}_\beta \left[\sum_{i=1} y_i \ln \sigma(\mathbf{X}\beta) + (1 - y_i) \ln(1 - \sigma(\mathbf{X}\beta)) \right]$$

We used the chain rule and gradient descent as numerical optimization technique:

$$LL(\beta) = y \ln p + (1 - y) \ln(1 - p)$$

$$\frac{\partial LL(\beta)}{\partial p} = \frac{y}{p} - \frac{1 - y}{1 - p}$$

$$p = \sigma(z), z = \mathbf{X}\beta$$

$$\frac{\partial p}{\partial z} = \sigma(z)[1 - \sigma(z)]$$

$$z = \mathbf{X}\beta, \frac{\partial z}{\partial \beta_j} = x_j$$

$$\frac{\partial LL(\beta)}{\partial \beta_j} = \frac{\partial LL(\beta)}{\partial p} * \frac{\partial p}{\partial z} * \frac{\partial z}{\partial \beta_j} =$$

$$\left[\frac{y}{p} - \frac{1 - y}{1 - p}\right] \sigma(z)[1 - \sigma(z)]x_j =$$

since $p = \sigma(z)$

$$\left[\frac{y}{p} - \frac{1 - y}{1 - p}\right] p[1 - p]x_j =$$

$$[y(1 - p) - p(1 - y)]x_j =$$

$$[y - p]x_j =$$

$$[y - \sigma(\mathbf{X}\beta)]x_j$$ => Gradient

Recap: Neural Networks
Gradient Descent in Backpropagation

Minimizing the **empirical risk function $\boldsymbol{R(\theta)}$**, which is modeling ***expected loss*** (as we don’t know the true distribution of data). This means, the empirical risk $\boldsymbol{R(\theta)}$ is the average loss over the training data.

$$R(\theta) = \frac{1}{N}\sum_n L(y_n, f(x_n)) = \frac{1}{2N}\sum_n \left(y_n - g(\theta^T x_n)\right)^2$$

derivative of $f(z)^2 \Rightarrow 2f(z)f'(z)$ (chain rule)

$$\nabla_\theta R = \frac{1}{2N}\sum_n 2(y_n - g(\theta^T x_n))(-1)g'(\theta^T x_n)x_n = 0$$

$$g(z) = (1 + exp(-z))^{-1}$$

Unfortunately, there is no “closed-form” solution. We used gradient descent to backpropagate the error of training examples .

Optimization in Machine Learning

Many machine learning problems can be cast as optimization problems:

- The OLS estimator for the linear regression solves a convex optimization problem.
- The MLE estimator of a logistic regression solves a convex optimization problem.
- Lasso and ridge regression are convex optimization problems.
- Neural networks are non-convex, but convex optimization methods are used.
- Support Vector Machines use quadratic optimization.
- k-means clustering can be formulated as nonlinear mixed-integer programming problem.
- etc.

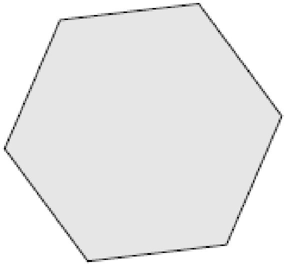
=> **Convex optimization** plays a crucial role, because it is often easier to "convexify" a problem (make it convex optimization friendly), rather than to use non-convex optimization.

In our last class, we

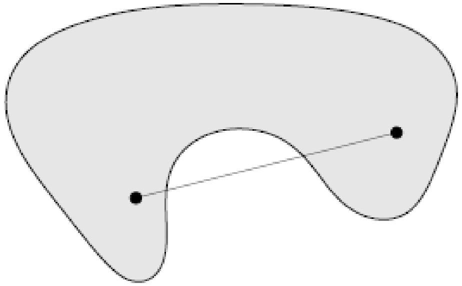
- discuss **convex optimization** to deepen your understanding of machine learning.
- introduce **extensions of gradient** descent for **non-smooth** functions.
- explore **online convex optimization**, which provides a foundation for much contemporary research in machine learning.

Convex Sets and Functions

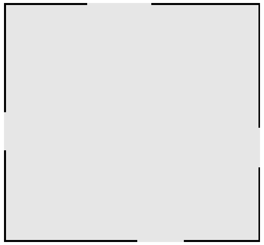
Convex sets: $x, y \in \ C, 0 \leq \alpha \leq 1 \Rightarrow \alpha x + (1-\alpha)y \in C$



convex

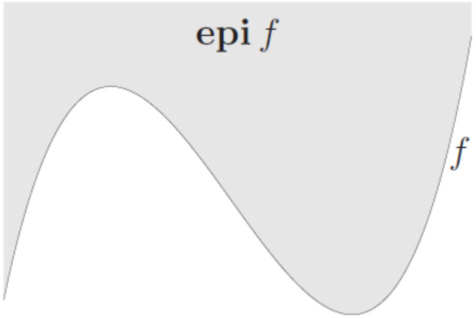


not convex



not convex

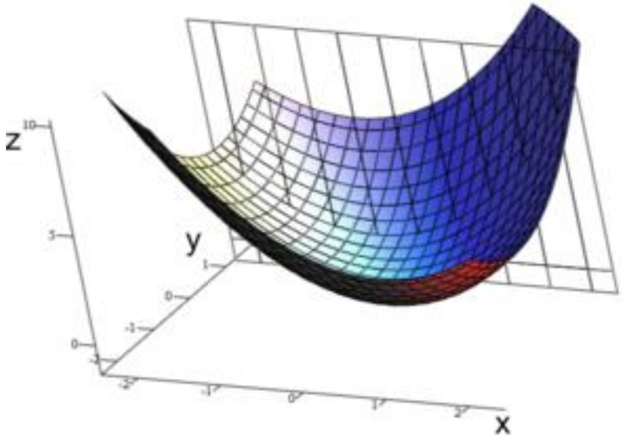
A function is convex, if the epigraph is a convex set (not the case below):



Convex Functions

Definition (Convex function):
If $X \subset \mathbb{R}^n$ is a non-empty, convex set, then the function $f : X \rightarrow \mathbb{R}$ is strictly convex on X, if for all $x, y \in X$ and $x \neq y$ Jensen‘s inequality holds:

$$f\big((1-\alpha)x + \alpha\, y\big) < (1-\alpha)f(x) + \alpha f(y)$$



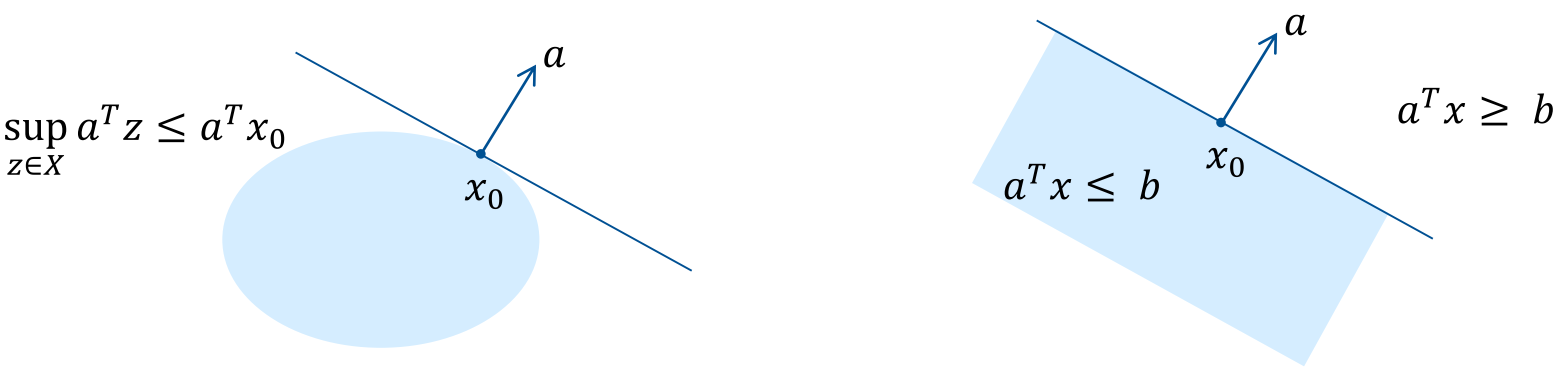
Examples:

- $f(x) = x^2$ is strictly convex, but $f(x) = |x|$ is convex, not strictly convex.
- $\exp x\,, -\log x\,, x \log x\,, x^{\alpha},$
- linear and affine functions are convex and concave
- norms ||·||

Supporting Hyperplane

Given a hyperplane $H = \{x \in \mathbb{R}^n | a^T x = b\}$, we say that the hyperplane H passes through a vector x_0 when $x_0 \in H$ which is equivalent to $a^T x_0 = b$.

The hyperplane H contains a set X in one of its halfspaces when either $a^T x \leq b$ for all $x \in X$, or $a^T x \geq b$ for all $x \in X$

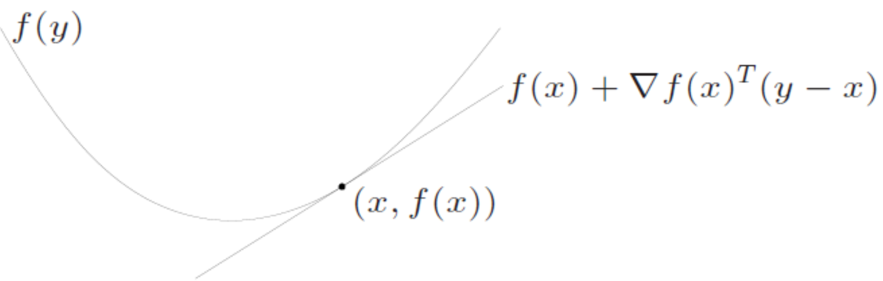


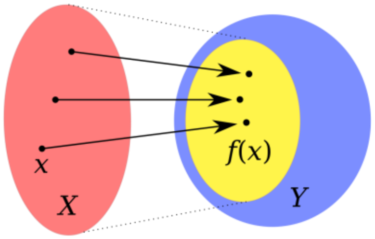
Supporting hyperplane theorem: Let $X \subseteq \mathbb{R}^n$ be a nonempty convex set and x_0 be at the boundary of this set. Then, there exists a hyperplane passing through x_0 and containing the set X in one of its halfspaces. This means, there is a vector $a \in \mathbb{R}^n, a \neq 0$, such that $\sup_{z \in X} a^T z \leq a^T x_0$. This is a *supporting hyperplane*.

Differentiable Convex Functions

A differentiable function f is convex, iff dom f is convex and for all feasible y.

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \ \forall x, y \in dom\ f$$





$f : X \rightarrow Y$

The red oval is dom f

This means, in one dimension, the graph lies above its tangents.

A twice differentiable function f is convex, iff dom f is convex and the Hessian matrix of second partial derivatives is positive semidefinite (eigenvalues > 0).

$$H(x) = \left(\frac{\partial f^2(x)}{\partial x_i \partial x_j} \right)_{\mathrm{i,j}}$$

Operations that Perserve Convexity

- Non-negative multiples: αf is convex, if f is convex and $\alpha \geq 0$

- Sum: $f + g$ is convex, if f, g are convex

- Composition with affine functions: $f(Ax + b)$ is convex, if f is convex
  - $f(x) = \|Ax + b\|$
  - $f(x) = -\sum_{i=1} \log(b_i - a_i^T x)$

- $f(x) = \max\{f_1(x), \cdots, f_m(x)\}$ is convex, if the components are convex.

- $f(x) = h\big(g(x)\big)$ is convex, if g is convex, and h is convex and non-decreasing.

Unconstrained Convex Optimization

Definition (Convex optimization):

Given a convex set $X \subset \mathbb{R}^n$ and a convex function $f: X \to \mathbb{R}$. Compute the minimum of the function f on the feasible set X: $\min_{x \in X} f(x)$

Solutions to a convex optimization problem are global solutions.

Special cases

- Is $X = \mathbb{R}^n$ we have a unrestricted optimization problem
- Linear optimization is a special case of convex optimization
- Projections

Definition (Projection):

If the set $X \subset \mathbb{R}^n$ is non-empty, closed and convex and if $y \notin X$, then $\min_{y \in X} ||y - \hat{y}||$ is a convex optimization problem. The solution $\hat{y}$ is a projection from y onto X.

$$P_X(y) = \min_{\hat{y} \in X} ||y - \hat{y}||^2$$

Optimality Conditions for Constrained Convex Optimization Problems

$$\begin{array}{lll} \min & f(x) & \\ \text{s.t.} & g_i(x) \leq 0 & i = 1, \ldots, m \\ & h_j(x) = 0 & j = 1, \ldots, p \\ & x \in \mathbb{R}^n & \end{array}$$

where $f(x)$ and $g_i(x)$ are convex functions and the equality is affine.

- $h(x) = mx + n$ for all $m, n \in \mathbb{R}$ is an *affine linear* function.

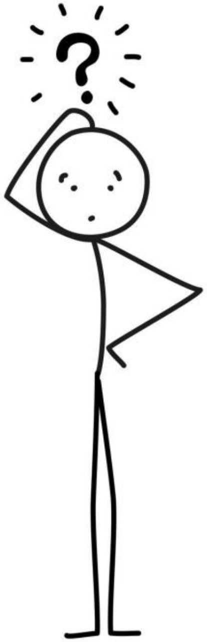
Lagrange funktion: $L = f(x) + \lambda^T h(x) + \mu^T g(x)$

KKT conditions as necessary first-order conditions for optimality:

$\nabla_x f(x) + \lambda^T \nabla_x h(x) + \mu^T \nabla_x g(x) = 0$	The gradient of the Lagrange function should be 0 in OPT.
$h(x) = 0$	Feasibility
$g(x) \leq 0$	Feasibility
$\mu^T g(x) = 0$	Complementarity
$\mu \geq 0$	The Lagrange multipliers of the inequalities ≥ 0

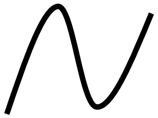
Complementarity conditions: also written as $0 \leq \mu \perp g(x) \leq 0$

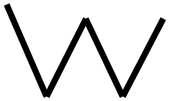
Why is OLS estimation a convex optimization problem?

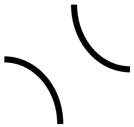


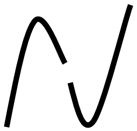
Continuity, Differentiability, Smoothness

- Differentiable => continuous
- Not continuous => not differentiable
- Continuously differentiable functions are sometimes said to be of *class* C^1.
- If derivatives $f^{(n)}$ exist for all positive integers n, the function is smooth or equivalently, of *class* C^∞.
- For example, $f(x) = e^{2x}$ is C^∞, because its nth derivative $f^{(n)}(x) = 2^n e^{2x}$ and is continuous.


continuous,
differentiable


continuous,
non-differentiable


not continuous,
not differentiable


not continuous,
not differentiable

Caution: Differentiability and continuity depend on the **domain** of the function.

Lipschitz Continuity

A Lipschitz continuous function is a continuous function, which is limited in how fast it can change.

Definition (Ball):
For $x \in \mathbb{R}^n, r > 0$ and an arbitrary norm $\|\cdot\|$ in $\mathbb{R}^n$, an *open* ball around x with radius r is defined as $B(x, r) = \{y \in \mathbb{R}^n | \ \|x - y\| < r\}$, a *closed* ball as $B(x, r) = \{y \in \mathbb{R}^n | \ \|x - y\| \leq r\}$.

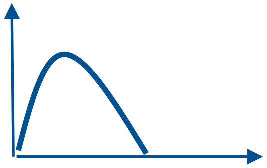
Definition (Lipschitz continuity):
If $X \subset \mathbb{R}^n$ is a non-empty, convex set and $f: \mathbb{R}^n \to \mathbb{R}$, then f is Lipschitz continuous on X, if for all $\mathrm{z} \in X$ there is a $\delta = \delta(z) > 0$ and a $L = L(z) \geq 0$, such that

$$|f(x) - f(y)| \leq L \cdot \|x - y\| \ \ \forall x, y \in B(z, \delta)$$

i.e., f is Lipschitz continuous on $B(z, \delta)$

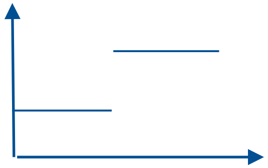
Examples

Continuous differentiable => **Lipschitz continuous** => continuous



continuous

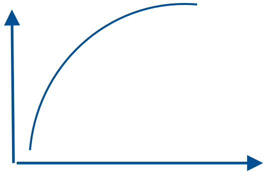
Not continuous => not Lipschitz continuous



not Lipschitz continuous

Continuous => not necessarily Lipschitz continuous

- The function $\sqrt{x}$ becomes infinitely steep as x approaches 0 since its derivative becomes infinite
  - $\left|\sqrt{x} - \sqrt{y}\right| \leq L|x - y| \; \forall x, y \in [0,1]$
  - Suppose $x = 0$ and $y = 1/4L^2$
  - $\left|\sqrt{x} - \sqrt{y}\right| = \frac{1}{2L} > \frac{1}{4L} = L|x - y|$



Continuous, but not
Lipschitz continuous

Optimizing Differentiable Convex Functions

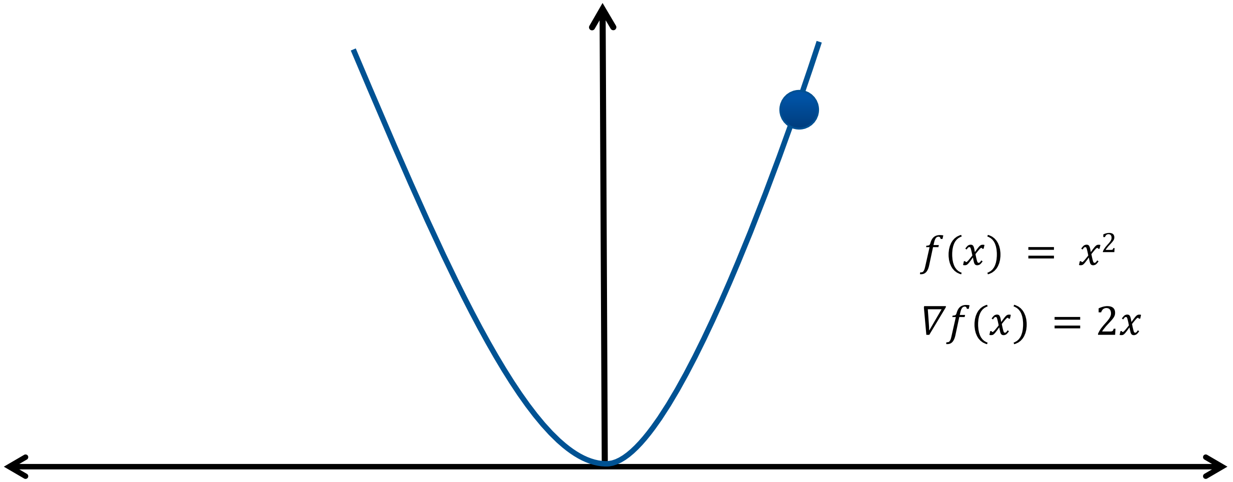
$$min_x\, f(x)$$

So far, we used gradient descent

Start at x_0 and move along the direction of the negative gradient

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

Adapt the step size via line search



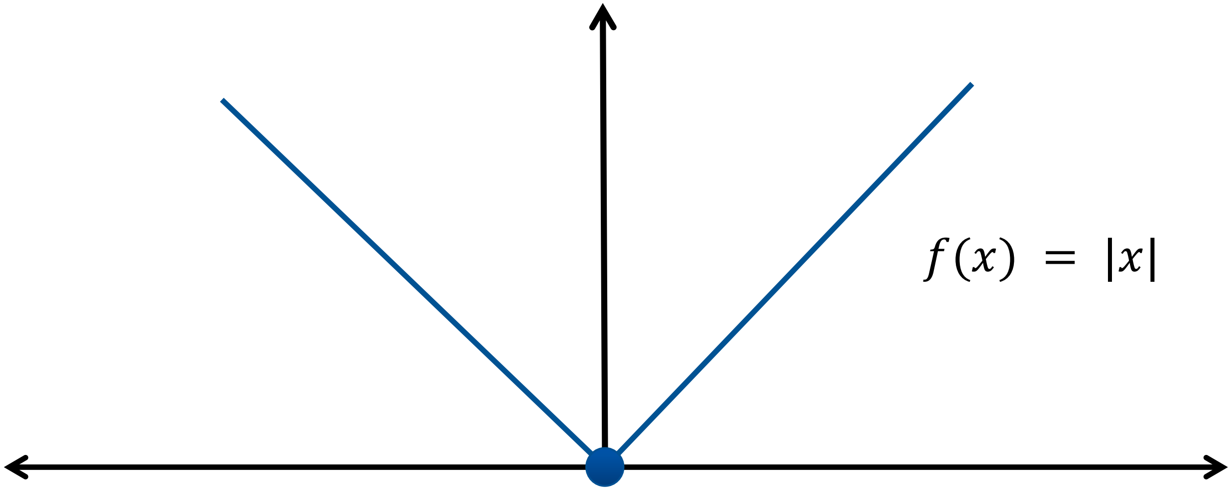
$f(x) = x^2$

$\nabla f(x) = 2x$

Non-Differentiable Functions

For non-differentiable functions, there might be not gradient at a point x_0 or the gradient might not be unique.

- Discrete convex functions are non-smooth functions.
- $f(x) = |x|$ is a non-smooth function, which is non-differentiable at $x = 0$.



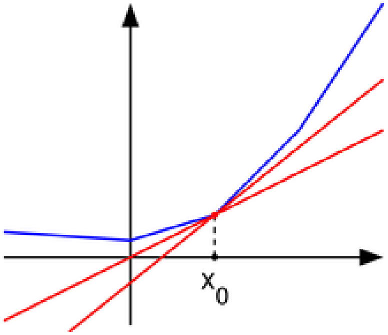
Subdifferential

Conv $\mathbb{R}^n$ describes the set of convex functions on $\mathbb{R}^n$ with values in $\overline{\mathbb{R}} := \mathbb{R} \cup \{+\infty\}$.
Recap: for differentiable continuous functions the gradient is $f(x) \geq f(x_0) + \nabla f(x_0)^T(x - x_0)$.

Definition (Subgradient):
For $f \in$ Conv $\mathbb{R}^n$ and $x_0 \in \; dom\, f$ we call a vector $s \in \mathbb{R}^n$ the *subgradient* of f in x_0, if

$$f(x) \geq f(x_0) + s^T(x - x_0) \;\forall x \in \mathbb{R}^n$$

The *subdifferential* of f in x, described with $\partial f(x_0)$, is the set of all subgradients of f in x_0.
A subgradient is a support hyperplane to the epigraph of f in point $(x_0, f(x_0))$:



Example: subdifferential of $f(x) = |x|$ at $x_0 = 0$

$$|x| = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$$

$f(x) \geq f(x_0) + s(x - x_0), [a, b]$

$f(x) - f(x_0) \geq c(x - x_0)$

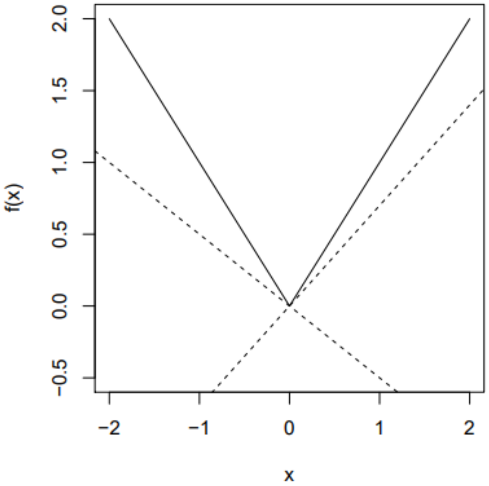
$x_0 = 0$

$$a = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0} = \lim_{x \to x_0^-} \frac{|x| - |0|}{x - 0} = \lim_{x \to x_0^-} \frac{-x}{x} = -1$$

$$b = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0} = \lim_{x \to x_0^+} \frac{x}{x} = 1$$

$s \in [-1,1]$

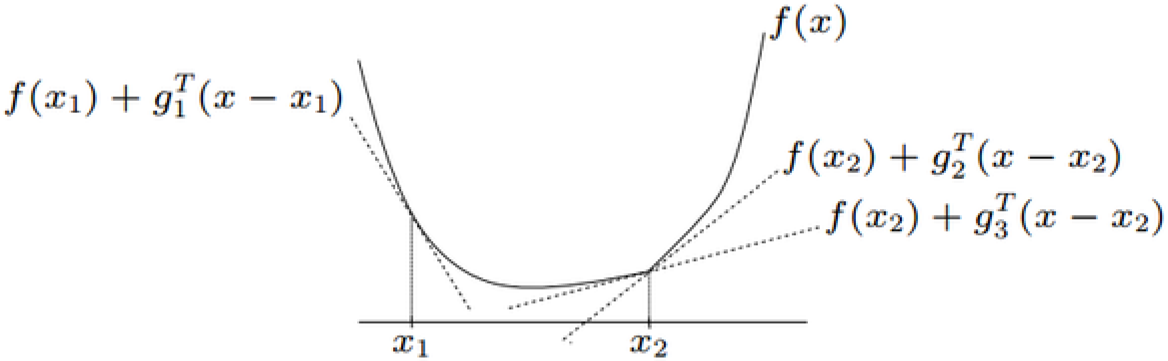
$$\partial_x |x| = \begin{cases} -1, x < 0 \\ [-1,1], x = 0 \\ 1, x > 0 \end{cases}$$



$f(x) = |x|$

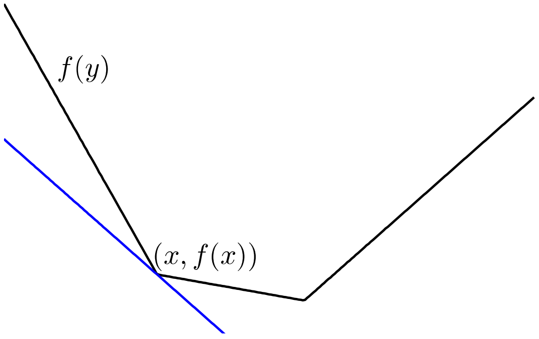
The Subgradient Method

- The subgradient method is a generalization of gradient descent for non differentiable (non-smooth) functions f developed by Naum Z Shor (Soviet Union) in the 1960s and 1970s.
- The subgradient method is not called subgradient descent, because the objective function can also increase.
- If f is differentiable, then its only subgradient at x is the gradient vector $\nabla f(x)$ itself, and **subgradient** methods use the same search direction as the method of **steepest descent (aka. gradient descent)**.
- The step sizes are fixed a priori.


$f(x)$
$f(x_1) + g_1^T(x - x_1)$
$f(x_2) + g_2^T(x - x_2)$
$f(x_2) + g_3^T(x - x_2)$
x_1
x_2

Quelle: https://optimization.mccormick.northwestern.edu/

The Subgradient Method



```
Input: Concave, continuous (non-)differentiable funktion $f\colon \mathbb{R}^n \to \mathbb{R}$,
       feasible start point $x^{(1)} \in \mathbb{R}^n$, parameter $\varepsilon > 0$
$k = 1$, $f_{best}^{(k)} = M$
While($\lim_{k\to\infty} f_{best}^{(k)} - f\left(x^{(k)}\right) < \varepsilon$){
    • Choose a subgradient $s^{(k)} \in \partial f\left(x^{(k)}\right)$
    • $d^{(k)} = -s^{(k)}/\left\|s^{(k)}\right\|$        // gives a unit vector
    • Select $\alpha_k > 0$ and set $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
    • $f_{best}^{(k)} = \min\{f_{best}^{(k)}, f\left(x^{(k)}\right)\}$
    • $k++$
}
```

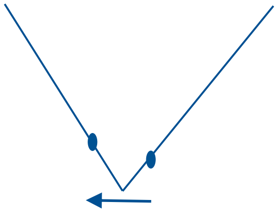
- If $0 \in \partial f\left(x^{(k)}\right)$, the function f is optimized. This criterion is rarely satisfied. Instead one uses $\lim_{k\to\infty} f_{best}^{(k)} - f\left(x^{(k)}\right) < \varepsilon$
- In case of differentiable functions f, this is equivalent to gradient descent and $s^{(k)} = \nabla f\left(x^{(k)}\right)$, the gradient of function f.

Remark

The subgradient method is no descent method, because the objective function value can increase.

$$x^{(k+1)} = x^{(k)} + t_k s^{(k)}$$

This means $-s^{(k)}$ leads to $f\left(x^{(k+1)}\right) > f\left(x^{(k)}\right)$.



For this reason, the algorithm stores the currently smallest value

$$f\left(x_{best}^{(k)}\right) = \min\{f_{best}^{k-1}, f\left(x^{(k)}\right)\}$$

Step Size

- Constant step size:
  - $\alpha_k = \alpha$ for all k
- Square summable, but not summable
  - $\alpha_k \geq 0, \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \sum_{k=1}^{\infty} \alpha_k = \infty$
  - Example: $\alpha_k = \frac{a}{b+k} \sim 1/k,$ with $a > 0$ and $b \geq 0$.
  - Thus α_k converges to 0, but not too fast.
- Decreasing but non summable step size:
  - $\alpha_k \geq 0, \lim_{k\to\infty} \alpha_k = 0, \sum_{k=1}^{\infty} \alpha_k = \infty$ (z. B. $\alpha_k = a/\sqrt{k}$ with $a > 0$)
  - $\lim_{k\to\infty} \alpha_k = 0$ is required, such that the method converges for non-smooth functions.
- etc.

In gradient descent the step size depends on the point evaluated and the search direction.
In contrast to gradient descent, the step size is defined a priori.

The Subgradient Method with Constraints

The subgradient method can be extended to convex optimization problems with constraints:

$$\min f(x)\,, \text{s.t.}\, x \in C$$

where C is a convex set, i.e. $C \subseteq \mathbb{R}^n$ is convex if for $x, y \in C$ and $\lambda \in [0,1]$: $\lambda x + (1-\lambda)y \in C$.

As usual $x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$. Now $g^{(k)}$ is a subgradient of the objective function or of a constraint at $x^{(k)}$.

$$g^{(k)} = \begin{cases} \partial f_0(x) \ \text{ if } f_i(x) \leq 0 \ \forall i = 1 \cdots m \\ \partial f_j(x) \text{ for } j \text{such that } f_j(x) \geq 0 \end{cases}$$

In other words, if the current point satisfies a constraint, we use a subradient of the objective function (as if the problem had no constraints). If this is not the case, we use a violated constraint and a subgradient of this constraint.

The Subgradient Method with Constraints

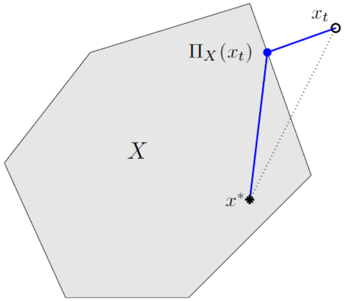
For constrained problems, the subgradient method $x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$ might chose a point $x^{(k+1)}$ outside the feasible region with $x^{(k+1)} \notin C$. This point needs to be projected onto the feasible region:

Projected Subgradient Method

Select $x_1 \in X$ randomly

$x^{(k+1)} = \Pi_X(x^{(k)} - \alpha_k g^{(k)})$

where Π_X is a Euclidean projection onto X: $\Pi_X(x) = \arg\min_{\mathrm{x}' \in X} \|x' - x\|^2$

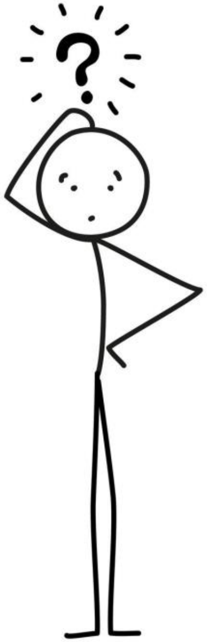


Extensions of the projected subgradient method:

- Online Mirror Descent (OMA) (Shalev-Shwartz, 2007)
- Dual Averaging (DA) (Nesterov, 2009)

These methods play a role in online machine learning and online optimization, where the input arrives step by step.

What are differences between gradient descent and the subgradient method?



Online (Machine) Learning

In contrast to supervised learning the data in **online learning** is **no longer i.i.d.**, but the sequence of observations is **ordered**

$$O_t = (x_1, y_1), (x_2, y_2), \cdots, (x_{t-1}, y_{t-1})$$

The objective is to predict the next y_t given the next x_t as well as all observations O_t so far:
$\hat{y}_t = h(x_t, \theta_t)$
The **hypothesis changes over time** and you want to adapt online.

Online convex optimization is used to minimize the aggregate loss incurred:

1. At every stage $t = 1,2, \cdots, T$ the optimizer selects action x_t from a closed convex subset $X \subseteq \mathbb{R}^n$
2. Once an action has been selected, the optimizer incurs a loss $L_t(x_t)$ based on an (a priori) unknown loss function $L_t: X \mapsto \mathbb{R}$.
3. Based on the incurred loss and/or any other feedback received, the optimizer updates their action and the process repeats.

Online Learning in Neural Networks

Training Neural Networks

- The standard framework is empirical risk minimization where draws are i.i.d.
- Training data consists of i.i.d. unordered samples from D

$$(x, y) \sim D$$

- Minimize empirical risk:

$$R(\theta) = \mathbb{E}_{x,y\sim D} L(y_n, f(x_n, \theta))$$

- Gradient descent

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta R \Big|_{\theta_t} = \theta_t - \alpha \mathbb{E}_{x,y\sim D} \nabla_\theta L(y_n, f(x_n, \theta))$$

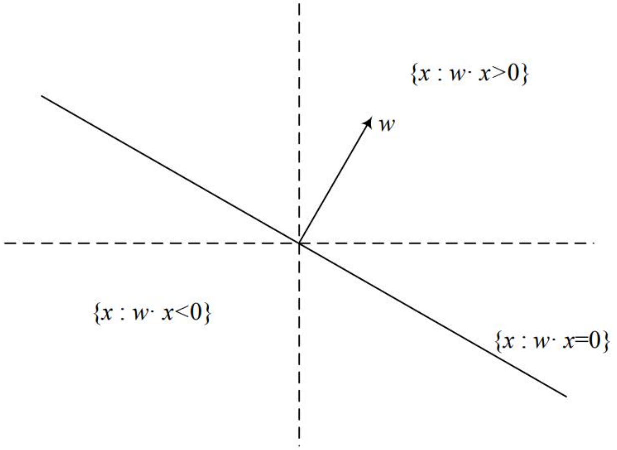
- However, we can also view the learning process as online learning
- The data is **no longer i.i.d.**, but the sequence of observations is **ordered**

$$O_t = (x_1, y_1), (x_2, y_2), \cdots, (x_{t-1}, y_{t-1})$$

- The objective is to predict the next y_t given the next x_t as well as all observations O_t so far: $\hat{y}_t = h(x_t, \theta_t)$

Online Optimization: Another Example

- Online binary predictions (e.g. spam classification):
  - Let $h_w: \mathbb{R}^n \mapsto \{-1,1\}$ where $h_w(x) = \begin{cases} +1, if\ w \cdot x > 0 \\ -1, if\ w \cdot x < 0 \end{cases}$
  - The data may fall into one of two halfspaces.
  - Player chooses $w_t \in W = \{w \in \mathbb{R}^n : \|w\|_2 \leq 1\}$, a unit ball in $\mathbb{R}^n$.
  - **Adversary** chooses (x_t, y_t)
  - Player incurs a loss $L_t\big(w_t; (x_t, y_t)\big) = \max\{0,1 - y_t w \cdot x_t\}$ (aka. Hinge loss)
  - Player receives feedback (x_t, y_t)



Regret Minimization

Instead of average loss, we minimize **regret** in online optimization as an achievable goal:

$$Reg(T) = \max_{x \in X} \sum_t [L_t(x_t) - L_t(x)] = \sum_t L_t(x_t) - \min_{x \in X} \sum_t L_t(x)$$

where x is a (hypothetical) ex post optimal fixed parameter vector, an offline algorithm. x_t is chosen by our online algorithm.

This is the difference between the aggregate loss incurred by the agent after T stages and that of the best action $x^* \in \underset{x \in X}{\operatorname{argmin}} \sum_t L_t(x)$in hindsight.

The main goal in online optimization is to design online policies that achieve no regret, i.e., for any sequence of loss functions $L_t, t = 1,2, \cdots, T,$ i.e., the regret grows sublinearly in t.

Feedback Models

The optimizers might have different types and amounts of information available.
In the oracle model, the optimizer gains acces to each loss function via a black-box feedback mechanism. An oracle $Or_L(x)$ for a function L can include:

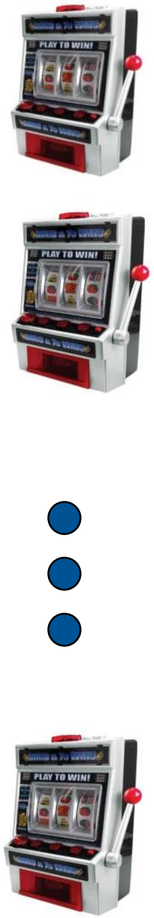
- Full information: $Or_L(x) = L.$ The oracle returns the entire function L.

- Bandit feedback: $Or_L(x) = L(x)$

- Gradient feedback: $Or_L(x) = \nabla L(x)$

- Noisy feedback:
  - $Or_L(x) = L(x) + \varepsilon(x; \omega)$ for some additive noise variable ε.
  - $Or_L(x) = \nabla L(x) + \varepsilon(x; \omega)$ for some observational noise variable ε.
    Aka. stochastic first-order oracle.

Bandit Feedback: The Multi-Armed Bandit



Each machine provides a random reward from a probability distribution specific to that machine. The gambler wants to maximize the sum of rewards earned through a sequence of lever pulls.

Iteratively, for time $t = 1, 2, \ldots$:

1. Nature chooses payoffs or losses on the machines L_t
2. Player picks a machine i_t
3. Loss is revealed $L_t(i_t)$

Process is repeated T times!
(with different, unknown, arbitrary losses every round)

Goal: Minimize the regret: $\sum_t L_t(i_t) - \min_j \sum_t L_t(j)$

Loss(ALG) – Loss(lowest-cost fixed machine)

Balance exploration (of new machines) with exploitation!

Simple greedy strategies:

- Exploration phase followed by a pure exploitation phase.
- The best lever is selected for a proportion $1 - \varepsilon$ of the trials, and a random lever for a proportion ε.

Follow-the-Leader

- **Follow the Leader** (**FTL**) play the action that is optimal in hindsight up to stage t
- Minimize the sum of all losses that you encountered for all data points $s < t$ so far (greedy):

$$x_{t+1} = \mathrm{argmin}_{x \in X} \sum_{s<t} L_s(x)$$

- For example, use all seen examples as a batch ML problem and solve for the best weight vector.
- The policy requires a **full information oracle** and the ability to compute the arg min in the FTL update rule.
- Used in fictitious play in game theory (an early method to compute equilibria for certain games).

FTL Might Perform Poorly

- The player chooses $x_t \in X = [-1,1]$ while $g_t \in [-1,1]$ is chosen by the adversary.
- $L_t(x) = g_t x$
- The player arbitrarily picks $x_1 = 0,$ and the adversary wants to cause the player to incur losses. The adversary knows that the FTL strategy is deterministic.
- The be-the-leader (BTL) strategy cheats by looking into the next example.

t	**FTL** x_t	g_t	$\sum_{s=1\ldots t-1} g_s$	**FTL** $L_t(x)$	**BTL** x'_t=x_{t+1}	**BTL** $L_t(x)$
1	0	0.5	0.5	0	-1	-0.5
2	-1	-1	-0.5	1	1	-1
3	1	1	0.5	1	-1	-1
4	-1	-1	-0.5	1	1	-1

- Regret of FTL: $(T-1) - (-0.5) \approx T$ with 0 being the best fixed strategy.
- Regret of BTL: $\approx -T$

Follow the Regularized Leader

Follow the Regularized Leader (**FTRL**):

$$x_{t+1} = \mathrm{argmin}_{x \in X} \sum_{s<t} L_s(x) + \frac{1}{\gamma} h(x)$$

- $h(x)$ is a penalty function (regularization term) and γ is a tunable parameter.
- FTRL is closely related to smooth fictitious play in game theory.

Special Cases of FTRL:

- **Hedge** (similar to Multiplicative Weights Update) uses negative entropy

$$x_{t+1} = \mathrm{argmin}_{x \in X} \sum_{s<t} L_s(x) + x \log x$$

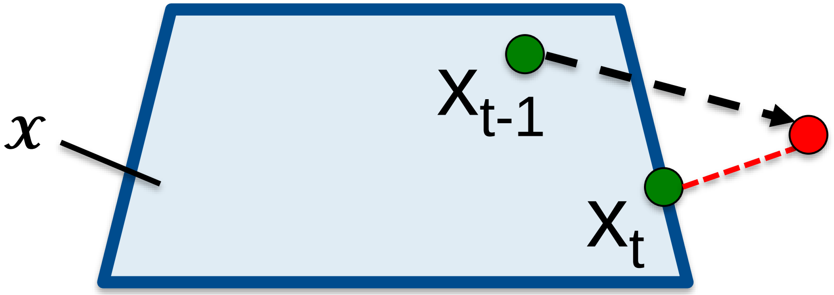
- **Online Gradient Descent** determines the gradient for every new data point and is a version fo FTRL with L_2 (Euclidean) regularization:

$$x_{t+1} = \mathrm{argmin}_{x \in X} \sum_{s<t} L_s(x) + \frac{1}{2\delta} \|x\|^2$$

Online (Projected) Gradient Descent

```
Input: convex set $X, T, x_i \in X$, step sizes $\alpha_t > 0$
for $t = 1$ to $T$ do
        Play $x_t$
        Observe noisy gradient $v_t = -[\nabla L_t(x_t) + \varepsilon_t]$    // Oracle feedback
        Update $x_{t+1} = \Pi_X[x_t + \alpha_t v_t]$              // Projection onto the feasible set of actions
```

Again the projection is the Euclidean projector $\Pi(x) = argmin_{x' \in X} \left|\left|x' - x\right|\right|^2$



- Regret of OGD with stochastic first-order feedback is $Reg(T) = O\left(\sqrt{T}\right)$, with strongly convex loss functions even $Reg(T) = O(\log T)$
- **Online Mirror Descent** replaces the projection step by a non-Euclidean norm to get better regret bounds.

Online LP/IP

$$\max \sum_{t=1}^{s} r_t x_t \quad \xleftarrow{\text{Accept?}} \quad \begin{pmatrix} r_{s+1} \\ a_{1,s+1} \\ \dots \\ a_{m,s+1} \end{pmatrix} x_{s+1}, \quad \begin{pmatrix} r_{s+2} \\ a_{1,s+2} \\ \dots \\ a_{m,s+2} \end{pmatrix} x_{s+2}, \dots$$

$$s.t. \sum_{t=1}^{s} a_{i,t} x_t \le b_i, \qquad i = 1, \dots, m$$

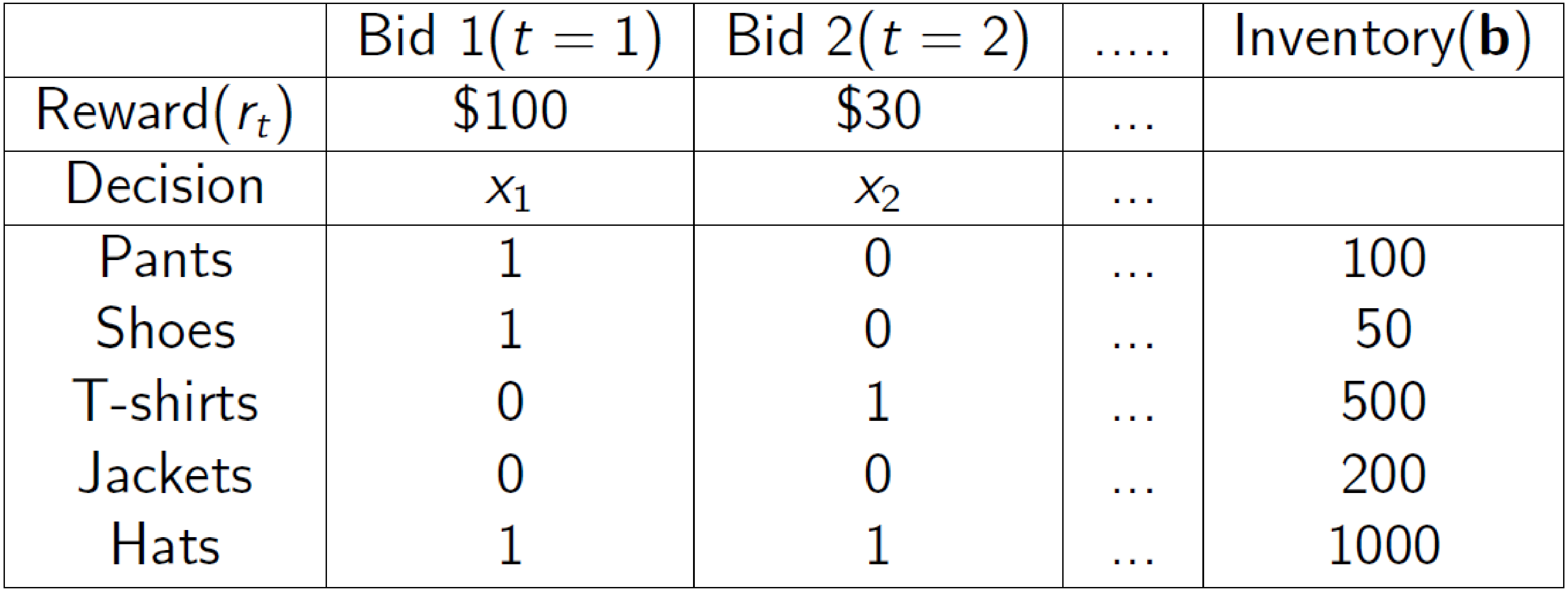
$$x_t \in \{0,1\}, \qquad t = 1, \dots, s$$

x_t: binary allocation variable
r_t: willingness to pay
s: number of requests so far
$a_{i,t}$: amount of resource i requested by request t
b_i: capacity limit of resource i
m: number of resources

Online linear programming (OLP) problem:

- also an online convex optimization problem, but it requires different techniques.
- the constraint matrix and objective function coefficient is revealed column by column i.i.d from an unknown distribution P .

Example: An Online Auction



	Bid 1$(t = 1)$	Bid 2$(t = 2)$	.....	Inventory$(\mathbf{b})$
Reward(r_t)	\$100	\$30	...	
Decision	x_1	x_2	...	
Pants	1	0	...	100
Shoes	1	0	...	50
T-shirts	0	1	...	500
Jackets	0	0	...	200
Hats	1	1	...	1000

Acceptance via Dual Prices

Compute **threshold / shadow prices $\boldsymbol{p_i}$** using the dual while assuming the **number of requests $\boldsymbol{n}$ is known a priori.**

Primal (relaxation):

$$\max \sum_{t=1}^{s} r_t x_t$$

$$s.t. \sum_{t=1}^{s} a_{i,t} x_t \le \frac{\boldsymbol{s}}{\boldsymbol{n}} b_i, \qquad i = 1, \dots, m$$

$$0 \le x_t \le 1 \qquad t = 1, \dots, s$$

Dual:

$$\min \sum_{i=1}^{m} \frac{\boldsymbol{s}}{\boldsymbol{n}} b_i p_i + \sum_{t=1}^{s} y_t$$

$$s.t. \sum_{i=1}^{m} a_{i,t} p_i + y_t \ge r_t \qquad t = 1, \dots, s$$

$$p_i, y_t \ge 0 \qquad i = 1, \dots, m; t = 1, \dots, s$$

Interpretation: one more unit of resource i would increase the obj. function value by p_i.

Acceptance criteria:

- Prices: $x_{s+1}(p) = \begin{cases} 0 \; if \; r_{s+1} \le p^T a_{s+1} \\ 1 \; if \; r_{s+1} > p^T a_{s+1} \end{cases}$
- Availability: $a_{i,s+1} x_{s+1} \le b_i - \sum_{t=1}^{s} a_{i,t} x_t \;\; \forall i$

S Agrawal, Z Wang, Y Ye (2014): A dynamic near-optimal algorithm for online linear programming.

Gradient Descent ... Again

Primal (relaxation):

$$\max \sum_{t=1}^{n} r_t x_t$$

$$s.t. \sum_{t=1}^{n} a_{i,t} x_t \le b_i, \qquad i = 1, \dots, m$$

$$0 \le x_t \le 1 \qquad t = 1, \dots, n$$

Dual:

$$\min \sum_{i=1}^{m} b_i p_i + \sum_{t=1}^{n} y_t$$

$$s.t. \sum_{i=1}^{m} a_{i,t} p_i + y_t \ge r_t \qquad t = 1, \dots, s$$

$$p_i, y_t \ge 0 \qquad i = 1, \dots, m; t = 1, \dots, s$$

Optimality conditions: $x_t^* = \begin{cases} 0\ if\ r_t \le p^{*T} a_t \\ 1\ if\ r_t > p^{*T} a_t \end{cases}$

The opt. dual price vector p^* in an online algorithm is learned and dynamically updated over time.

Equivalent Dual:

$$\min \sum_{i=1}^{m} b_i p_i + \sum_{t=1}^{n} \left(r_j - \sum_{i=1}^{m} a_{i,j} p_i \right)^+$$

$$s.t.\ p_i \ge 0 \qquad i = 1, \dots, m$$

$(\cdot)^+$ is the positive-part function (ReLU).

X Li, C Sun, Y Ye (2020): Simple and Fast Algorithm for Binary Integer and Online Linear Programming

Machine Learning Paradigms

- **Supervised learning** is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

- **Unsupervised learning** is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.

- **Online learning** is a method of machine learning in which data becomes available in a sequential order and is used to update the best predictor for future data at each step. It is used in situations where it is necessary for the algorithm *to dynamically adapt to new patterns in the data*, or when the data itself is generated as a function of time.

- **Reinforcement learning** is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.