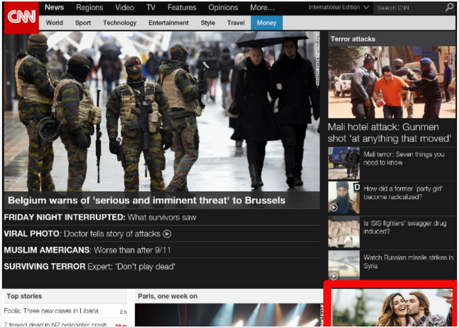
Online Learning

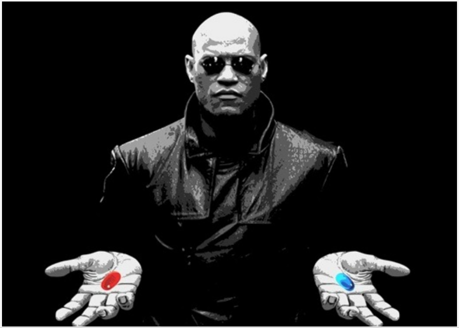
Goals

1. introduce online learning
2. introduce the notion of regret
3. present basic algorithms
4. create building blocks for many imitation learning algorithms

- advertising (which ad to display)



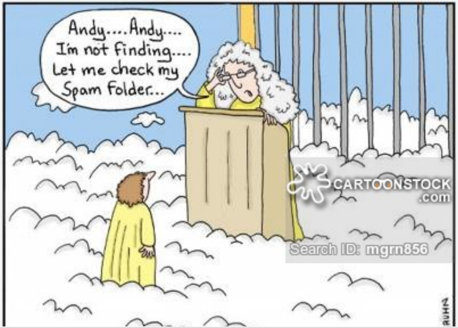
- advertising (which ad to display)
- medical treatment (which drug to prescribe)



Real world online learning tasks

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)
- spam/malware filtering (filter or keep)



Real world online learning tasks

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)
- spam/malware filtering (filter or keep)
- stock market (sell or acquire bonds)

Real world online learning tasks

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)
- spam/malware filtering (filter or keep)
- stock market (sell or acquire bonds)
- network routing (which path to take)

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)
- spam/malware filtering (filter or keep)
- stock market (sell or acquire bonds)
- network routing (which path to take)
- compression (what's the next symbol)

- advertising (which ad to display)
- medical treatment (which drug to prescribe)
- design/functionality rollouts (works or not)
- spam/malware filtering (filter or keep)
- stock market (sell or acquire bonds)
- network routing (which path to take)
- compression (what's the next symbol)
- weather (will it rain tomorrow), etc.

in every task there is a decision to be made under missing information

Batch learning

many i.i.d examples $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$

define some loss $\ell(\mathcal{D})$ (e.g. negative log-likelihood, square error)

learn a model by $\ell(\mathcal{D}) \to \min$

deploy on a test set

Batch learning

many i.i.d examples $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$

define some loss $\ell(\mathcal{D})$ (e.g. negative log-likelihood, square error)

learn a model by $\ell(\mathcal{D}) \to \min$

deploy on a test set

Online learning

one example x_t

predict $\hat{y}_t$

get feedback

suffer some penalty $\ell_t(x_t, \hat{y}_t)$

improve the model

repeat

Batch learning

many i.i.d examples $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$

define some loss $\ell(\mathcal{D})$ (e.g. negative log-likelihood, square error)

learn a model by $\ell(\mathcal{D}) \to \min$

deploy on a test set

Online learning

one example x_t

predict $\hat{y}_t$

get feedback

suffer some penalty $\ell_t(x_t, \hat{y}_t)$

improve the model

repeat

- note: no training/testing set distinction

$$\begin{aligned} \text{input } x_t &\in \mathcal{X} \quad \text{input space} \\ \text{truth } y_t &\in \mathcal{Y} \quad \text{truth space} \\ \text{prediction } \hat{y}_t &\in \mathcal{P} \quad \text{decision space} \end{aligned}$$

$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{P}$	penalty/loss

input $x_t \in \mathcal{X}$ input space

truth $y_t \in \mathcal{Y}$ truth space

prediction $\hat{y}_t \in \mathcal{P}$ decision space

	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{P}$	penalty/loss
online regression	$\mathbb{R}^d$	$\mathbb{R}$	$\mathbb{R}$	$\lvert y_t - \hat{y}_t \rvert$

input $x_t \in \mathcal{X}$ input space

truth $y_t \in \mathcal{Y}$ truth space

prediction $\hat{y}_t \in \mathcal{P}$ decision space

	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{P}$	penalty/loss
online regression	$\mathbb{R}^d$	$\mathbb{R}$	$\mathbb{R}$	$\lvert y_t - \hat{y}_t \rvert$
online classification	$\mathbb{R}^d$	$\{1, \ldots, K\}$	$\{1, \ldots, K\}$	$[\![y_t \neq \hat{y}_t]\!]$

input $x_t \in \mathcal{X}$ input space

truth $y_t \in \mathcal{Y}$ truth space

prediction $\hat{y}_t \in \mathcal{P}$ decision space

	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{P}$	penalty/loss
online regression	$\mathbb{R}^d$	$\mathbb{R}$	$\mathbb{R}$	$\lvert y_t - \hat{y}_t \rvert$
online classification	$\mathbb{R}^d$	$\{1, \dots, K\}$	$\{1, \dots, K\}$	$[\![y_t \neq \hat{y}_t]\!]$
expert advice	$\mathbb{R}^N$	$\mathbb{R}^d$	$\{1, \dots, N\}$	$y_t[\hat{y}_t]$

input $x_t \in \mathcal{X}$ input space

truth $y_t \in \mathcal{Y}$ truth space

prediction $\hat{y}_t \in \mathcal{P}$ decision space

	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{P}$	penalty/loss
online regression	$\mathbb{R}^d$	$\mathbb{R}$	$\mathbb{R}$	$\lvert y_t - \hat{y}_t \rvert$
online classification	$\mathbb{R}^d$	$\{1, \ldots, K\}$	$\{1, \ldots, K\}$	$[\![y_t \neq \hat{y}_t]\!]$
expert advice	$\mathbb{R}^N$	$\mathbb{R}^d$	$\{1, \ldots, N\}$	$y_t[\hat{y}_t]$
structured prediction	K^m	K^m	K^m	$\sum_{i=1}^m [\![y_t^i \neq \hat{y}_t^i]\!]$

Why online learning?

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!
- **2000s-now:**
  - ➡ computers are very powerful, memory is cheap ☺

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!
- **2000s-now:**
  - ➡ computers are very powerful, memory is cheap ☺
  - ➡ still batch algorithms explode memory and time ☹

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!
- **2000s-now:**
  - ➡ computers are very powerful, memory is cheap ☺
  - ➡ still batch algorithms explode memory and time ☹
    - easy access to data made datasets practically infinite
    - discarding data is a bad idea, we want it all!

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!
- **2000s-now:**
  - ➡ computers are very powerful, memory is cheap ☺
  - ➡ still batch algorithms explode memory and time ☹
    - easy access to data made datasets practically infinite
    - discarding data is a bad idea, we want it all!
    - some people say that "data acquisition outpaced the Moore's law"

- **early days 50-70s:** online learning is a requirement
  - ➡ first computers, very low memory, very slow CPUs
  - ➡ perceptron from 1957 is originally an online algorithm!
- **later 70-90s:** batch learning became possible
  - ➡ reasonable CPU power, reasonable memory
  - ➡ great convergence guarantees!
- **2000s-now:**
  - ➡ computers are very powerful, memory is cheap ☺
  - ➡ still batch algorithms explode memory and time ☹
    - easy access to data made datasets practically infinite
    - discarding data is a bad idea, we want it all!
    - some people say that "data acquisition outpaced the Moore's law"

effectively are back into the 50s

not only a question of resources:

- the larger the data, the harder it is
  - ➡ to guarantee stationarity
  - ➡ to ensure that test/train instances come from the same $\mathcal{D}$
  - ➡ to guarantee i.i.d
  - ➡ to ensure labels are stochastic as well

not only a question of resources:

- the larger the data, the harder it is
  - to guarantee stationarity
  - to ensure that test/train instances come from the same $\mathcal{D}$
  - to guarantee i.i.d
  - to ensure labels are stochastic as well
- hence algorithms need to be adaptive

not only a question of resources:

- the larger the data, the harder it is
  - ➡ to guarantee stationarity
  - ➡ to ensure that test/train instances come from the same $\mathcal{D}$
  - ➡ to guarantee i.i.d
  - ➡ to ensure labels are stochastic as well
- hence algorithms need to be adaptive
- frequent re-training is not always an option (because resources, ...)

Online learning

one example x_t

predict $\hat{y}_t$

get feedback

suffer some penalty $\ell_t(x_t, \hat{y}_t)$

improve the model

repeat

- small memory footprint
- faster updates
- faster adaptation
- better test performance (in a certain sense)

- environment
  - i.i.d assumption is convenient
  - often cannot be guaranteed or is obviously violated
  - sometimes we assume nothing about distribution: ‘adversarial case’

- environment
  - i.i.d assumption is convenient
  - often cannot be guaranteed or is obviously violated
  - sometimes we assume nothing about distribution: ‘adversarial case’
- feedback
  - full information is best
  - but correct labels are expensive and slow to get
  - often partial feedback is all you have: ‘bandit case’

- environment
  - ➡ i.i.d assumption is convenient
  - ➡ often cannot be guaranteed or is obviously violated
  - ➡ sometimes we assume nothing about distribution: ‘adversarial case’
- feedback
  - ➡ full information is best
  - ➡ but correct labels are expensive and slow to get
  - ➡ often partial feedback is all you have: ‘bandit case’

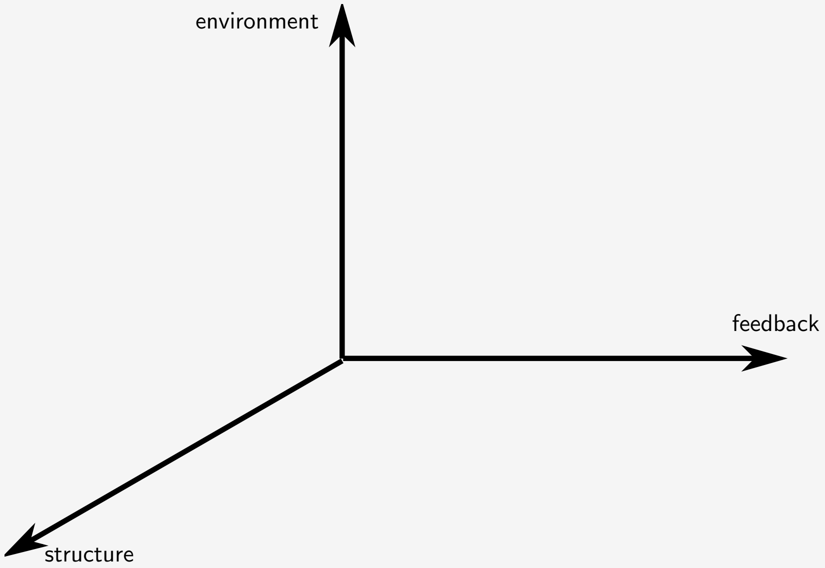
One-armed bandits

➡ you have to find a machine that gives you most money

➡ you only know your current reward from the chosen machine

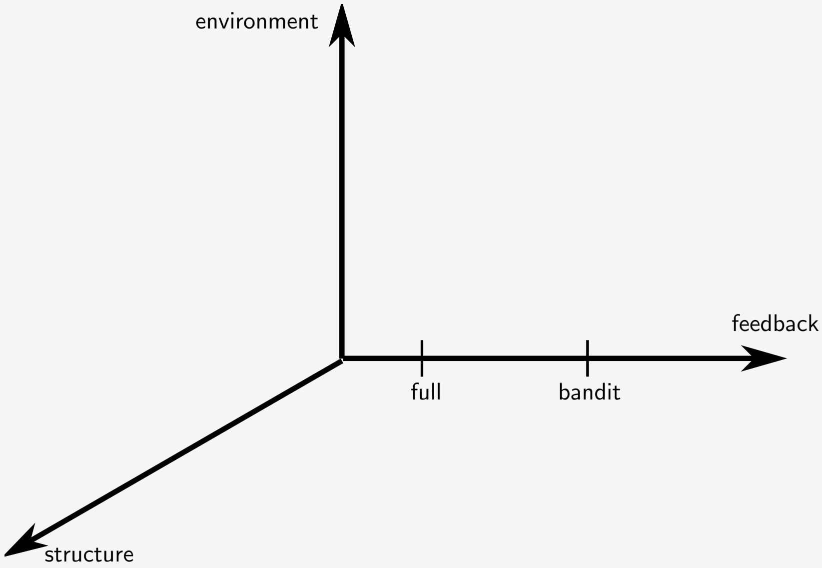
- environment
  - i.i.d assumption is convenient
  - often cannot be guaranteed or is obviously violated
  - sometimes we assume nothing about distribution: ‘adversarial case’
- feedback
  - full information is best
  - but correct labels are expensive and slow to get
  - often partial feedback is all you have: ‘bandit case’
- structure
  - no state (important but rare case)
  - usually there is some state or context
  - structured spaces (actions change the environment)

The space of online learning algorithms



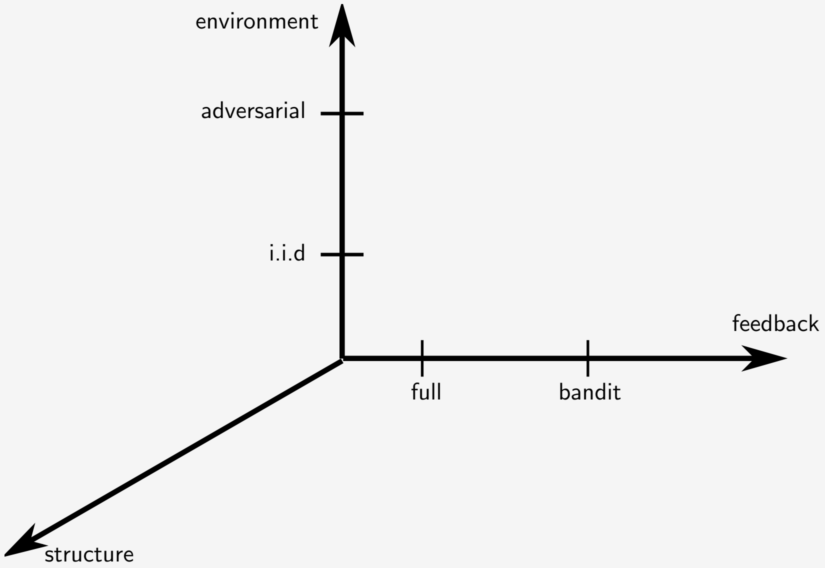
[Seldin'15]

The space of online learning algorithms



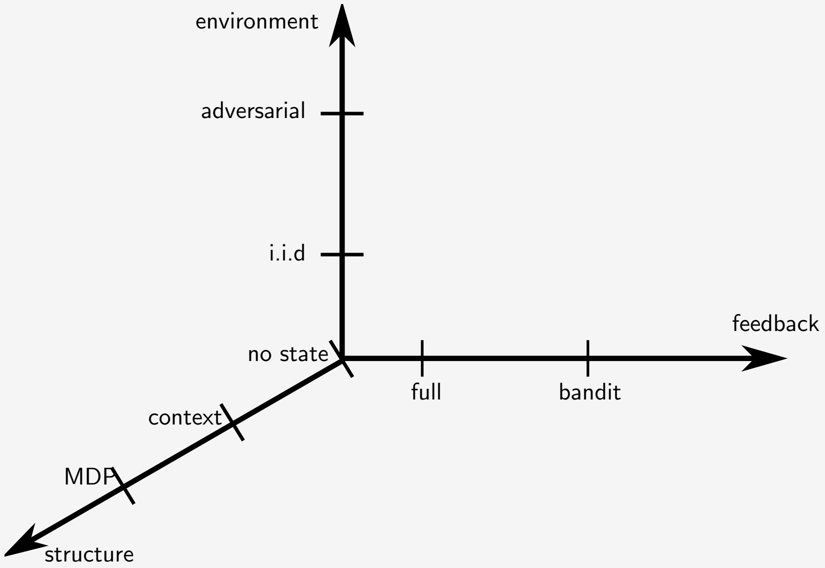
[Seldin'15]

The space of online learning algorithms



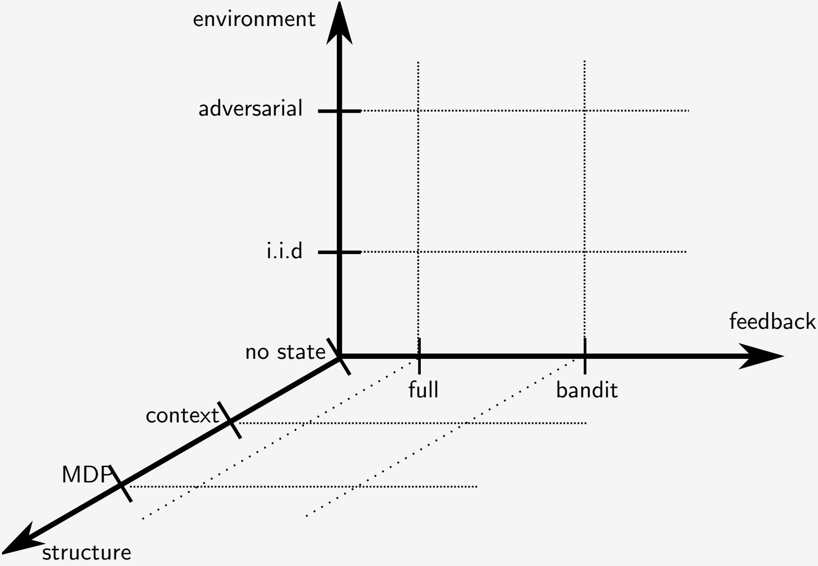
[Seldin'15]

The space of online learning algorithms



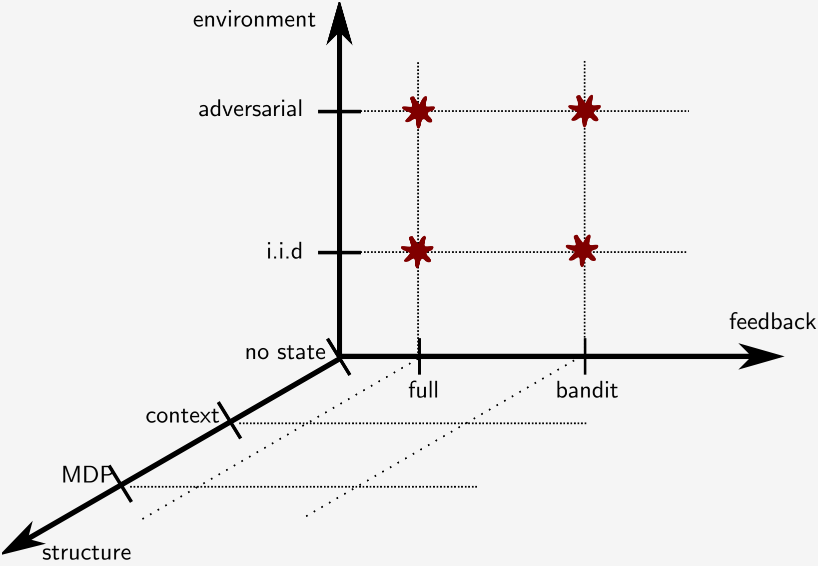
[Seldin'15]

The space of online learning algorithms

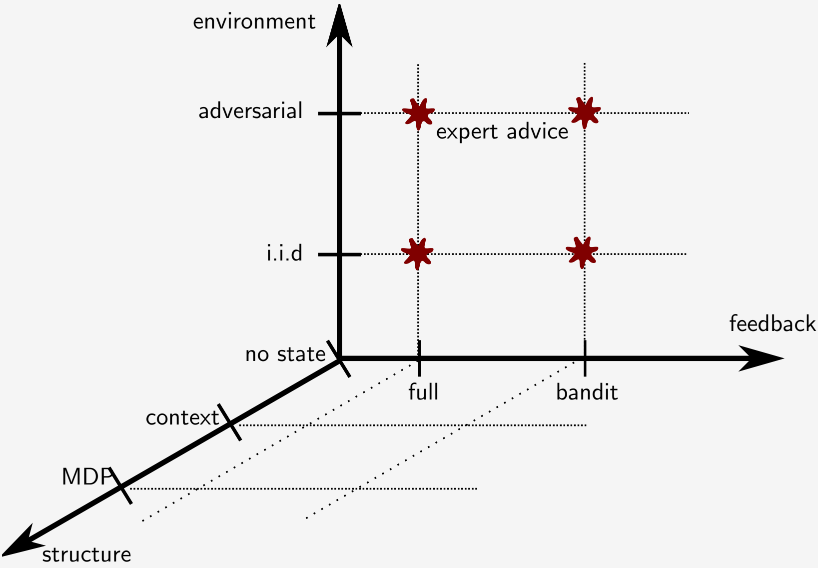


[Seldin'15]

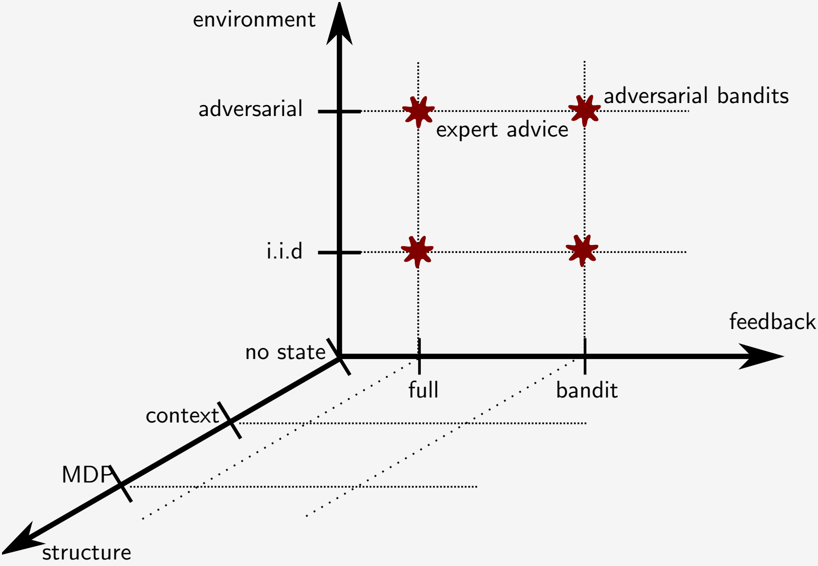
The space of online learning algorithms



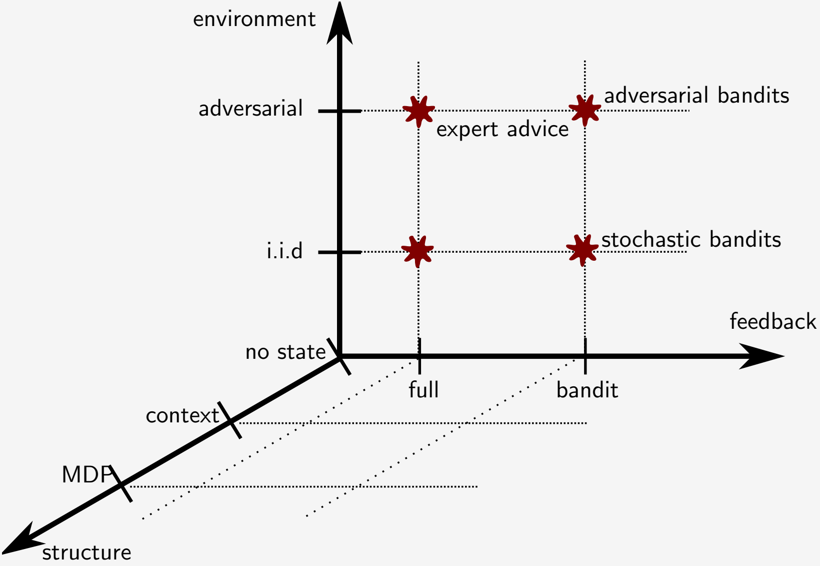
[Seldin'15]


environment
adversarial
expert advice
i.i.d
feedback
no state
full
bandit
context
MDP
structure

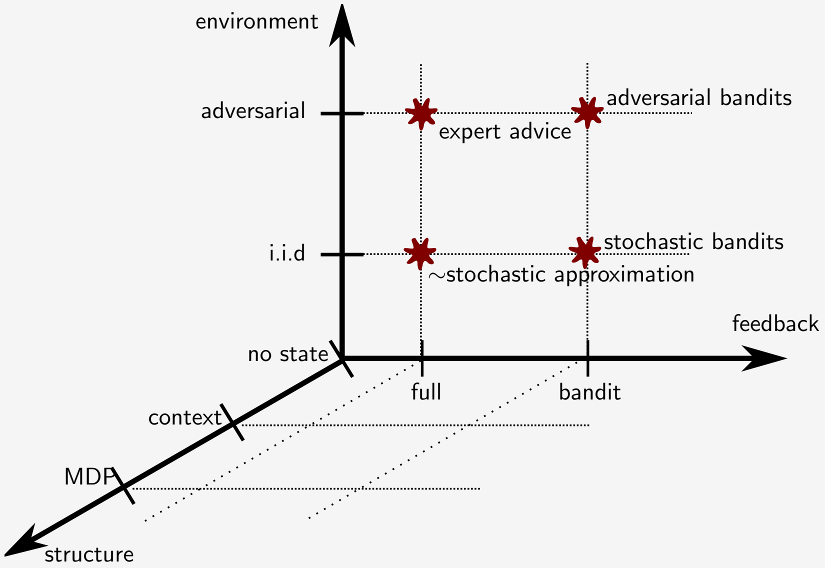
[Seldin'15]


environment
adversarial
i.i.d
adversarial bandits
expert advice
feedback
no state
full
bandit
context
MDP
structure

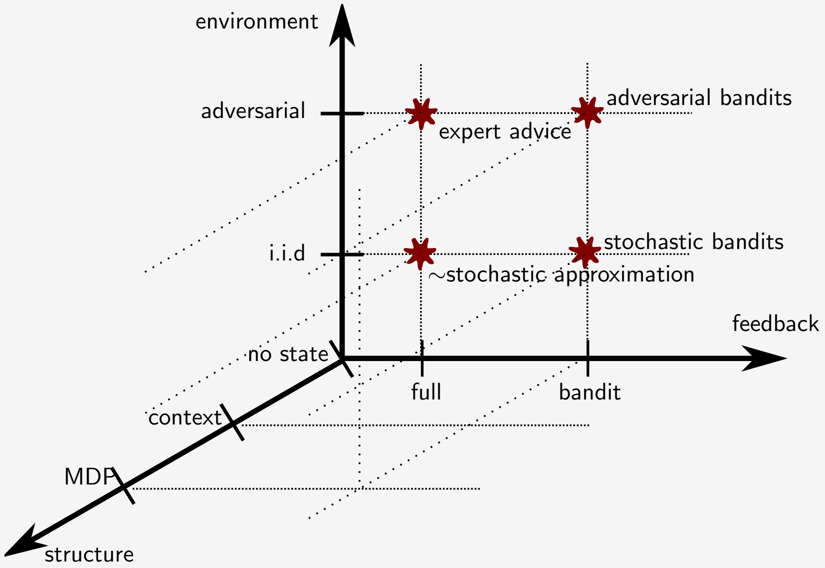
[Seldin'15]


environment
adversarial
i.i.d
adversarial bandits
expert advice
stochastic bandits
feedback
no state
full
bandit
context
MDP
structure

[Seldin'15]


environment
adversarial
adversarial bandits
expert advice
i.i.d
stochastic bandits
∼stochastic approximation
feedback
no state
full
bandit
context
MDP
structure

[Seldin'15]



IL is at the frontal plane (., ., MDP)

[Seldin'15]

Adversarial Environment with Full Information

Adversarial Environment with Full Information

(adversarial just means there are no statistical assumptions)

Online learning protocol

1: **for** $t = 0, \ldots$ **do**
2: observe x_t (if available)
3: predict $\hat{y}_t$
4: suffer loss $\ell_t(\hat{y}_t)$
5: update

- ℓ_t are arbitrary (e.g., does not mean these are uniformly distributed)
- could be random or non-random, depend on previous history
- we want algorithms that work in any case

Online learning protocol

for $t = 0, \ldots$ **do**
  observe x_t (if available)
  predict $\hat{y}_t$
  suffer loss $\ell_t(\hat{y}_t)$
  update

- ℓ_t are arbitrary (e.g., does not mean these are uniformly distributed)
- could be random or non-random, depend on previous history
- we want algorithms that work in any case

What about the goal?

Online learning protocol

1: **for** $t = 0, \ldots$ **do**
2: observe x_t (if available)
3: predict $\hat{y}_t$
4: suffer loss $\ell_t(\hat{y}_t)$
5: update

- ℓ_t are arbitrary (e.g., does not mean these are uniformly distributed)
- could be random or non-random, depend on previous history
- we want algorithms that work in any case

What about the goal?

- no training set, so cannot minimize loss over training set

Online learning protocol

1: **for** $t = 0, \dots$ **do**
2: observe x_t (if available)
3: predict $\hat{y}_t$
4: suffer loss $\ell_t(\hat{y}_t)$
5: update

- ℓ_t are arbitrary (e.g., does not mean these are uniformly distributed)
- could be random or non-random, depend on previous history
- we want algorithms that work in any case

What about the goal?

- no training set, so cannot minimize loss over training set
- even if we could, does not always make sense as ℓ_t can be anything

Online learning protocol

1: **for** $t = 0, \ldots$ **do**
2: observe x_t (if available)
3: predict $\hat{y}_t$
4: suffer loss $\ell_t(\hat{y}_t)$
5: update

- ℓ_t are arbitrary (e.g., does not mean these are uniformly distributed)
- could be random or non-random, depend on previous history
- we want algorithms that work in any case

What about the goal?

- no training set, so cannot minimize loss over training set
- even if we could, does not always make sense as ℓ_t can be anything
- ➡ **measure of success has to be calculated w.r.t. to the whole interaction, not just some end objective**

What do we want to achieve?

What do we want to achieve?

- in principle we want to minimize our total loss

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - ?$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure
  - ➡ e.g., w.r.t. to some fixed (but unknown) strategy $h = h_t$

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - \sum_{t=1}^{T} \ell_t(h_t)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure
  - ➡ e.g., w.r.t. to some fixed (but unknown) strategy $h = h_t$
  - ➡ or w.r.t. to the <u>best</u> strategy from a set $\mathcal{H}$

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure
  - ➡ e.g., w.r.t. to some fixed (but unknown) strategy $h = h_t$
  - ➡ or w.r.t. to the best strategy from a set $\mathcal{H}$
  - ➡ note: the larger is $\mathcal{H}$ the harder is the task

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure
  - ➡ e.g., w.r.t. to some fixed (but unknown) strategy $h = h_t$
  - ➡ or w.r.t. to the best strategy from a set $\mathcal{H}$
  - ➡ note: the larger is $\mathcal{H}$ the harder is the task
- we measure a 'cost of ignorance' or 'regret for not following that strategy'

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h)$$

What do we want to achieve?

- in principle we want to minimize our total loss
- still not ideal, because ℓ_t can scale arbitrary
- so we need a **relative** measure
  - ➡ e.g., w.r.t. to some fixed (but unknown) strategy $h = h_t$
  - ➡ or w.r.t. to the best strategy from a set $\mathcal{H}$
  - ➡ note: the larger is $\mathcal{H}$ the harder is the task
- we measure a 'cost of ignorance' or 'regret for not following that strategy'

$$R_T = \sum_{t=1}^{T} \ell_t(\hat{y}_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h)$$

Our ultimate goal:

- average regret $R_T / T \to 0$
- as fast as possible
- as the learning goes on, our loss is less and less different from the alternative one ('we have no regret')
- such algorithms are even called like that, 'no-regret algorithms'

Different definitions of regret

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions
- w/o any restrictive assumptions online learning is too hard (or impossible)

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions
- w/o any restrictive assumptions online learning is too hard (or impossible)
- need to restrict the power of adversary and vary R_T accordingly

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions
- w/o any restrictive assumptions online learning is too hard (or impossible)
- need to restrict the power of adversary and vary R_T accordingly
- different regret definitions than reflect our knowledge about the environment:

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions
- w/o any restrictive assumptions online learning is too hard (or impossible)
- need to restrict the power of adversary and vary R_T accordingly
- different regret definitions than reflect our knowledge about the environment:
  1. if we believe that true data is generated by some fixed function h^*, $y_t = h^*(x_t)$, it's reasonable to minimize R_T w.r.t. to that function

$$R_T(h^*) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(h^*)$$

Different definitions of regret

- regret is a tool to analyze a problem,
  to test it under different assumptions
- w/o any restrictive assumptions online learning is too hard (or impossible)
- need to restrict the power of adversary and vary R_T accordingly
- different regret definitions than reflect our knowledge about the environment:
  1. if we believe that true data is generated by some fixed function h^*, $y_t = h^*(x_t)$, it's reasonable to minimize R_T w.r.t. to that function

$$R_T(h^*) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(h^*)$$

  2. if not, the adversary must not at least change his mind at will, i.e. has to commit to some y_t before seeing $\hat{y}_t$; then it makes sense to optimize R_T w.r.t. to the best function from some set $\mathcal{H}$:

$$R_T(\mathcal{H}) = \sum_{t=1}^{T} \ell_t(w_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h)$$

What happens if we don't have the commitment requirement?

What happens if we don't have the commitment requirement?

Example: online classification

```
for t = 0, ... do
    observe x_t
    predict ŷ_t ∈ {0, 1}
    receive true y_t
    suffer loss ℓ_t(ŷ_t) = |y_t − ŷ_t|
    update w_{t+1}
```

$$R_T = \sum_{t=1}^{T} \ell(\hat{y}_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h(x_t))$$

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
    observe x_t
    predict $\hat{y}_t \in \{0, 1\}$
    receive true y_t
    suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
    update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
  observe x_t
  predict $\hat{y}_t \in \{0, 1\}$
  receive true y_t
  suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
  update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- take simplest $\mathcal{H} = \{h_0, h_1\}$, where $h_a \equiv a$ (constant, 0 or 1, function)

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
  observe x_t
  predict $\hat{y}_t \in \{0, 1\}$
  receive true y_t
  suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
  update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- take simplest $\mathcal{H} = \{h_0, h_1\}$, where $h_a \equiv a$ (constant, 0 or 1, function)
- **Exercise:** can you make the learner always lose? [Shalev-Shwartz'12]

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \dots$ **do**
  observe x_t
  predict $\hat{y}_t \in \{0, 1\}$
  receive true y_t
  suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
  update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- take simplest $\mathcal{H} = \{h_0, h_1\}$, where $h_a \equiv a$ (constant, 0 or 1, function)
- **Exercise:** can you make the learner always lose? [Shalev-Shwartz'12]
- wait until $\hat{y}_t$ and set $y_t = 1 - \hat{y}_t$

Adversary restriction

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
  observe x_t
  predict $\hat{y}_t \in \{0, 1\}$
  receive true y_t
  suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
  update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- wait until $\hat{y}_t$ and set $y_t = 1 - \hat{y}_t$

Adversary restriction

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
    observe x_t
    predict $\hat{y}_t \in \{0, 1\}$
    receive true y_t
    suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
    update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- wait until $\hat{y}_t$ and set $y_t = 1 - \hat{y}_t$
- for any $h \in \{h_0(\cdot), h_1(\cdot)\}$, $\min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)| \leq T/2$

Adversary restriction

What happens if we don't have the commitment requirement?

Example: online classification

for $t = 0, \ldots$ **do**
  observe x_t
  predict $\hat{y}_t \in \{0, 1\}$
  receive true y_t
  suffer loss $\ell_t(\hat{y}_t) = |y_t - \hat{y}_t|$
  update w_{t+1}

$$R_T = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)|$$

- wait until $\hat{y}_t$ and set $y_t = 1 - \hat{y}_t$
- for any $h \in \{h_0(\cdot), h_1(\cdot)\}$, $\min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)| \leq T/2$
- $R_T(\mathcal{H}) = \sum_{t=1}^{T} |y_t - \hat{y}_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |y_t - h(x_t)| \geq T - T/2 = T/2$

Realizability assumption: $\exists h^* \in \mathcal{H}$ s.t. $\forall t \; y_t = h^*(x_t)$. Also $|\mathcal{H}| < \infty$

Realizability assumption: $\exists h^* \in \mathcal{H}$ s.t. $\forall t\ y_t = h^*(x_t)$. Also $|\mathcal{H}| < \infty$

Consistent

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose **any** $h \in V_t$
5: predict $\hat{y}_t = r$
6: receive true $y_t = h^*(x_t)$
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Realizability assumption: $\exists h^* \in \mathcal{H}$ s.t. $\forall t\; y_t = h^*(x_t)$. Also $|\mathcal{H}| < \infty$

Consistent

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose **any** $h \in V_t$
5: predict $\hat{y}_t = r$
6: receive true $y_t = h^*(x_t)$
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Analysis:

- $\forall t$ at least one h is removed if there was an error (and none if not)
- $1 \leq |V_t| \leq |\mathcal{H}| - \#$errors
- $R_T = \#$errors $- 0 = \#$errors $\leq |\mathcal{H}| - 1$

Realizability assumption: $\exists h^* \in \mathcal{H}$ s.t. $\forall t\; y_t = h^*(x_t)$. Also $|\mathcal{H}| < \infty$

Consistent

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose **any** $h \in V_t$
5: predict $\hat{y}_t = r$
6: receive true $y_t = h^*(x_t)$
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Analysis:

- $\forall t$ at least one h is removed if there was an error (and none if not)
- $1 \leq |V_t| \leq |\mathcal{H}| - \#$errors
- $R_T = \#$errors $- 0 = \#$errors $\leq |\mathcal{H}| - 1$
- can we do better? hint: purge hypotheses faster

Realizability assumption: $\exists h^* \in \mathcal{H}$ s.t. $\forall t \; y_t = h^*(x_t)$. Also $|\mathcal{H}| < \infty$

Halving

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose **by majority vote** $r = \arg\max_{r \in \{0,1\}} |h \in V_t : h(x_t) = r|$
5: predict $\hat{y}_t = r$
6: receive true $y_t = h^*(x_t)$
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Analysis:

- $\forall t$ at least one half of V_t is removed if there was an error
- $1 \leq |V_t| \leq |\mathcal{H}|/2^{\#\text{errors}}$
- $R_T(h^*) = \#\text{errors} \leq \log_2 |\mathcal{H}|$

Finiteness of $\mathcal{H}$ is crucial

Example

- real line $\mathcal{X} = (0,1)$, thresholds $\mathcal{H} = \left\{h_\theta : (0,1) \to \{0,1\}\right\}$
- $h_\theta(x) = sign(\theta - x)$
- $\exists$ a sequence of x_t, y_t generated by some θ on which the Halving will have $R_T = T$

Finiteness of $\mathcal{H}$ is crucial

Example

- real line $\mathcal{X} = (0,1)$, thresholds $\mathcal{H} = \big\{h_\theta : (0,1) \to \{0,1\}\big\}$
- $h_\theta(x) = sign(\theta - x)$
- $\exists$ a sequence of x_t, y_t generated by some θ on which the Halving will have $R_T = T$

Exercise: construct such a sequence [Shalev-Shwartz'12]

Finiteness of $\mathcal{H}$ is crucial

Example

- real line $\mathcal{X} = (0, 1)$, thresholds $\mathcal{H} = \left\{h_\theta : (0, 1) \to \{0, 1\}\right\}$
- $h_\theta(x) = sign(\theta - x)$
- $\exists$ a sequence of x_t, y_t generated by some θ on which the Halving will have $R_T = T$

Solution:

- maintain L_t (left) and R_t (right)
- $L_0 = 0$, $R_0 = 1$
- pick a random $x_t \in (L_t, R_t)$
- receive $\hat{y}_t$
- report $y_t = 1 - \hat{y}_t$
- $R_{t+1} = x_t y_t + R_t \hat{y}_t$
- $L_{t+1} = x_t \hat{y}_t + L_t y_t$
- $\forall t\ R_t - L_t > 0$

Randomization

- realizability assumption may be too harsh for our application
- instead add an element of surprise to our predictions:
  - ➡ remember to require the adversary to commit to y_t before seeing $\hat{y}_t$
  - ➡ will change lines 4 and 5 in the Consistent

Consistent

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose $h = \arg\max_{r \in \{0,1\}} |h \in V_t : h(x_t) = r|$
5: predict $\hat{y}_t(w_t) = h(x_t)$
6: receive true y_t
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Randomization

- realizability assumption may be too harsh for our application
- instead add an element of surprise to our predictions:
  - ➡ remember to require the adversary to commit to y_t before seeing $\hat{y}_t$
  - ➡ will change lines 4 and 5 in the Consistent

Randomized

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose probability p_t
5: predict $\hat{y}_t(w_t) = 1$ with prob. p_t
6: receive true y_t
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Randomization

- realizability assumption may be too harsh for our application
- instead add an element of surprise to our predictions:
  - ➡ remember to require the adversary to commit to y_t before seeing $\hat{y}_t$
  - ➡ will change lines 4 and 5 in the Consistent

Randomized

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose probability p_t
5: predict $\hat{y}_t(w_t) = 1$ with prob. p_t
6: receive true y_t
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

$$R_T = \sum_{t=1}^{T} \mathbb{E}_{p_t}[\![\hat{y}_t \neq y_t]\!] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} [\![h(x_t) \neq y_t]\!]$$

← note regret changed again

- realizability assumption may be too harsh for our application
- instead add an element of surprise to our predictions:
  - ➡ remember to require the adversary to commit to y_t before seeing $\hat{y}_t$
  - ➡ will change lines 4 and 5 in the Consistent

Randomized

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose probability p_t
5: predict $\hat{y}_t(w_t) = 1$ with prob. p_t
6: receive true y_t
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

$$R_T = \sum_{t=1}^{T} \mathbb{E}_{p_t} [\![\hat{y}_t \neq y_t]\!] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} [\![h(x_t) \neq y_t]\!] \qquad \leftarrow \text{note regret changed again}$$

$$= \sum_{t=1}^{T} |p_t - y_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |h(x_t) - y_t|$$

Randomization

- realizability assumption may be too harsh for our application
- instead add an element of surprise to our predictions:
  - ➡ remember to require the adversary to commit to y_t before seeing $\hat{y}_t$
  - ➡ will change lines 4 and 5 in the Consistent

Randomized

1: Initialize $V_0 = \mathcal{H}$
2: **for** $t = 0, \ldots$ **do**
3: observe x_t
4: choose probability p_t
5: predict $\hat{y}_t(w_t) = 1$ with prob. p_t
6: receive true y_t
7: update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

$$R_T = \sum_{t=1}^{T} \mathbb{E}_{p_t} [\![\hat{y}_t \neq y_t]\!] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} [\![h(x_t) \neq y_t]\!] \qquad \leftarrow \text{note regret changed again}$$

$$= \sum_{t=1}^{T} |p_t - y_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |h(x_t) - y_t| \leq \sqrt{0.5 T \ln |\mathcal{H}|}$$

- we had full information (i.e., we received the true y_t)
- different adversary restrictions help to get regret bounds

- we had full information (i.e., we received the true y_t)
- different adversary restrictions help to get regret bounds
  - ➡ realizability + finiteness
    - Consistent $R_T \leq |\mathcal{H}| - 1$
    - Halving $R_T \leq \log_2 |\mathcal{H}|$

- we had full information (i.e., we received the true y_t)
- different adversary restrictions help to get regret bounds
  - ➡ realizability + finiteness
    - Consistent $R_T \leq |\mathcal{H}| - 1$
    - Halving $R_T \leq \log_2 |\mathcal{H}|$
  - ➡ randomization
    - Randomized $R_T \leq \sqrt{0.5T \ln |\mathcal{H}|}$

Learning with Experts' Advice

- imagine horse-races
- you know nothing about horses ☹
- luckily you have knowledgeable friends willing to give you advice ☺
- you need to apportion a fixed sum of money between them

- imagine horse-races
- you know nothing about horses ☹
- luckily you have knowledgeable friends willing to give you advice ☺
- you need to apportion a fixed sum of money between them

➡ goal: minimize losses / maximize profit

I actually make a lot more money as a bookmaker than I ever did as a race horse...

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^{N} p_t[i] = 1$ your distribution of money

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^N p_t[i] = 1$ your distribution of money
- loss $\sum_{i=1}^N p_t[i]\ell_t[i] = \langle p_t, \ell_t \rangle$

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^N p_t[i] = 1$ your distribution of money
- loss $\sum_{i=1}^N p_t[i]\ell_t[i] = \langle p_t, \ell_t \rangle$
- goal

$$R_T = \sum_{t=1}^{T} \langle p_t, \ell_t \rangle - \underbrace{\min_{i=1,\dots,N} \sum_{t=1}^{T} \ell_t[i]}_{\text{loss of the best friend}} \quad \to \min$$

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^N p_t[i] = 1$ your distribution of money
- loss $\sum_{i=1}^N p_t[i]\ell_t[i] = \langle p_t, \ell_t \rangle$
- goal

$$R_T = \sum_{t=1}^{T} \langle p_t, \ell_t \rangle - \underbrace{\min_{i=1,\ldots,N} \sum_{t=1}^{T} \ell_t[i]}_{\text{loss of the best friend}} \quad \to \min$$

- note: you don't know how good your friends are

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^N p_t[i] = 1$ your distribution of money
- loss $\sum_{i=1}^N p_t[i]\ell_t[i] = \langle p_t, \ell_t \rangle$
- goal

$$R_T = \sum_{t=1}^T \langle p_t, \ell_t \rangle - \underbrace{\min_{i=1,\dots,N} \sum_{t=1}^T \ell_t[i]}_{\text{loss of the best friend}} \quad \to \min$$

- note: you don't know how good your friends are
- note: horses/friends can conspire against you

- stateless case
  (you have friend's identity, but not horses' breakfast menu or expert history)
- N friends
- loss vector $\ell_t \in [0,1]^N$ e.g., $\ell_t[i] = 0.3$ if ith friend lost 30 cents
- prediction $p_t \in [0,1]^N$, $\sum_{i=1}^N p_t[i] = 1$ your distribution of money
- loss $\sum_{i=1}^N p_t[i]\ell_t[i] = \langle p_t, \ell_t \rangle$
- goal

$$R_T = \sum_{t=1}^{T} \langle p_t, \ell_t \rangle - \underbrace{\min_{i=1,\ldots,N} \sum_{t=1}^{T} \ell_t[i]}_{\text{loss of the best friend}} \rightarrow \min$$

- note: you don't know how good your friends are
- note: horses/friends can conspire against you
- but in the limit you can do as good as the best friend in hindsight!
- (in terms of average loss per race)

- if one of the friends is perfect can get $\leq \log_2 N$ mistakes with Halving

- if one of the friends is perfect can get $\leq \log_2 N$ mistakes with Halving
- but making a mistake does not necessarily mean we should disqualify a friend

- if one of the friends is perfect can get $\leq \log_2 N$ mistakes with Halving
- but making a mistake does not necessarily mean we should disqualify a friend

Hedge

1: init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
2: **for** $t = 1, \ldots$ **do**
3: compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
4: receive loss ℓ_t
5: update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$ $\leftarrow$ "soft disqualification"

- if one of the friends is perfect can get $\leq \log_2 N$ mistakes with Halving
- but making a mistake does not necessarily mean we should disqualify a friend

Hedge

1: init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
2: **for** $t = 1, \ldots$ **do**
3: compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
4: receive loss ℓ_t
5: update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$ $\leftarrow$ "soft disqualification"

Theorem

For any $\ell^1, \ldots, \ell^T$ and any $i \in \{1, \ldots, N\}$

$$R_T = \sum_{t=1}^{T} \langle p_t, \ell_t \rangle - \min_j \sum_{t=1}^{T} \ell_t[j] \leq \sqrt{2T \ln N} + \ln N$$

Hedge

1: init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
2: **for** $t = 1, \dots$ **do**
3: compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
4: receive loss ℓ_t
5: update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$

Exercise: [Marchetti-Spaccamela'11]

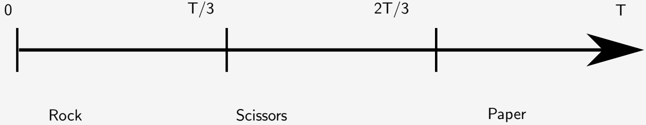
- 3 experts: 1st playing always **Rock**, 2nd – **Scissors**, and 3rd – **Paper**

Hedge

init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
for $t = 1, \ldots$ **do**
  compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
  receive loss ℓ_t
  update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$

Exercise: [Marchetti-Spaccamela'11]

- 3 experts: 1st playing always **Rock**, 2nd – **Scissors**, and 3rd – **Paper**
- your opponent plays first Rock $T/3$ times, then Scissors $T/3$ times and then Paper $T/3$ times



Hedge

init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
for $t = 1, \ldots$ **do**
    compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
    receive loss ℓ_t
    update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$

Exercise: [Marchetti-Spaccamela'11]

- 3 experts: 1st playing always **Rock**, 2nd – **Scissors**, and 3rd – **Paper**
- your opponent plays first Rock $T/3$ times, then Scissors $T/3$ times and then Paper $T/3$ times

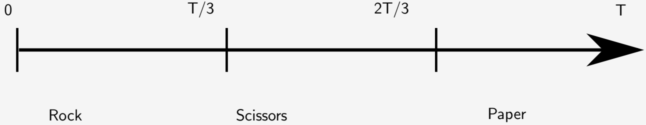
- loss: -1 if won, +1 if lost, 0 if tie

Hedge

init vector $w_1 \in \mathbb{R}_+^N$ s.t. $w_1[i] = 1/N$, learning rate $\mu > 0$
for $t = 1, \ldots$ **do**
  compute $p_t = \frac{w_t}{\sum_{i=1}^N w_t[i]}$
  receive loss ℓ_t
  update $w_{t+1}[i] = w_t[i] e^{-\mu \ell_t[i]}$

Exercise: [Marchetti-Spaccamela'11]

- 3 experts: 1st playing always **Rock**, 2nd – **Scissors**, and 3rd – **Paper**
- your opponent plays first Rock $T/3$ times, then Scissors $T/3$ times and then Paper $T/3$ times



- loss: -1 if won, +1 if lost, 0 if tie
- describe roughly 1) the most probable strategies played by Hedge, 2) when they switch and 3) the final distribution

- Hedge inspired Boosting – a powerful concept of combining weak algorithms into a strong one

- Hedge inspired Boosting – a powerful concept of combining weak algorithms into a strong one
- idea:
    - ➡ treat your training examples as experts
    - ➡ changing weights focuses attention on difficult examples

- Hedge inspired Boosting – a powerful concept of combining weak algorithms into a strong one
- idea:
  - ➡ treat your training examples as experts
  - ➡ changing weights focuses attention on difficult examples
- ★ Gödel Prize 2003

Infinite hypotheses space

- we'll introduce online convex optimization
- map some the problems we talked to the new language

Online convex optimization

1: Input: a convex set $S \subset \mathbb{R}^d$
2: **for** $t = 0, \ldots$ **do**
3: predict $w_t \in S$
4: receive a convex loss function $\ell_t : S \to \mathbb{R}$
5: suffer loss $\ell_t(w_t)$

- measurements (features) $\mathcal{X} = \mathbb{R}^d$
- truths and decision $\mathcal{Y} = \mathcal{D} = \mathbb{R}$
- common loss functions:
  - ➡ $\ell_t(p_t, y_t) = (p_t - y_t)^2$
  - ➡ $\ell_t(p_t, y_t) = |p_t - y_t|$
- simple hypothesis class $\mathcal{H} = \{x \mapsto \sum_{i=1}^{d} w[i]x[i] : w \in \mathbb{R}^d\}$ (linear predictors)

note: both loss functions ℓ_t are convex

- measurements $\mathcal{X} = \mathbb{R}^d$, where x_i is the advice of the ith expert
- truths $\mathcal{Y} = [0, 1]^d$
- decisions $p_t \in \mathcal{D} = \{1, \ldots, d\}$
- loss function: $\ell(p, y) = y_t[p_t]$
- hypothesis class $\mathcal{H} = \{h_1, \ldots, h_d\}$, where $h_i(x) = i, \forall x$ (constant predictors)

note: since $\mathcal{D}$ is discrete, the losses ℓ_t are **not** convex

- measurements $\mathcal{X}$
- binary truths and decisions $\mathcal{Y} = \mathcal{D} = \{0, 1\}$
- loss function: $\ell_t(p_t, y_t) = |p_t - y_t|$
- finite hypothesis class $\mathcal{H} = \{h_1, \ldots, h_k\}$

note: since $\mathcal{D}$ is discrete, the losses ℓ_t are again **not** convex

- how can we map non-convex to convex tasks?
  - randomization
  - surrogate losses

Expert Advice

- measurements $\mathcal{X} = \mathbb{R}^d$, where x_i is the advice of the ith expert
- truths $\mathcal{Y} = [0, 1]^d$
- decisions $p_t \in \mathcal{D} = \{1, \dots, d\}$
- loss function: $\ell(p, y) = y_t[p_t]$

Mapping

- let the learner maintain a vector $w_t \in \mathbb{R}^d$, s.t. $\sum_{i=1}^d w_{t,i} = 1$
- the learner randomly picks the expert according to the distribution w_t
- the adversary cannot base his ℓ_t on the sample from w_t
- the loss suffered is now $\mathbb{E}[y_t[p_t]] = w_t^\top y_t$ (linear function)

Now the problem fits into online convex optimization with $\ell_t = w_t^\top y_t$

Online Classification with finite hypothesis class

- measurements $x_t \in \mathcal{X}$
- binary truths and decisions $\mathcal{Y} = \mathcal{D} = \{0, 1\}$
- loss function: $\ell_t(p_t, y_t) = |p_t - y_t|$
- hypothesis class $\mathcal{H} = \{h_1, \dots, h_k\}$

- let $v_t = (h_1(x_t), \dots, h_k(x_t)) \in \{0, 1\}^k$
- let the learner maintain a vector $w_t \in \mathbb{R}^k$, s.t. $\sum_{i=1}^k w_{t,i} = 1$
- prediction is done via

$$p_t = \begin{cases} 1, & \text{if } w_t^\top v_t \geq 1/2 \\ 0, & \text{if } w_t^\top v_t < 1/2 \end{cases}$$

Online Classification with finite hypothesis class

- measurements $x_t \in \mathcal{X}$
- binary truths and decisions $\mathcal{Y} = \mathcal{D} = \{0, 1\}$
- loss function: $\ell_t(p_t, y_t) = |p_t - y_t|$
- hypothesis class $\mathcal{H} = \{h_1, \ldots, h_k\}$

- let $v_t = (h_1(x_t), \ldots, h_k(x_t)) \in \{0, 1\}^k$
- let the learner maintain a vector $w_t \in \mathbb{R}^k$, s.t. $\sum_{i=1}^k w_{t,i} = 1$
- prediction is done via

$$p_t = \begin{cases} 1, & \text{if } w_t^\top v_t \geq 1/2 \\ 0, & \text{if } w_t^\top v_t < 1/2 \end{cases}$$

- loss $\ell_t(w) = 2\left|w^\top v_t - y_t\right| \mathbb{I}[p_t \neq y_t]$

Online Classification with finite hypothesis class

- measurements $x_t \in \mathcal{X}$
- binary truths and decisions $\mathcal{Y} = \mathcal{D} = \{0, 1\}$
- loss function: $\ell_t(p_t, y_t) = |p_t - y_t|$
- hypothesis class $\mathcal{H} = \{h_1, \ldots, h_k\}$

- let $v_t = (h_1(x_t), \ldots, h_k(x_t)) \in \{0, 1\}^k$
- let the learner maintain a vector $w_t \in \mathbb{R}^k$, s.t. $\sum_{i=1}^k w_{t,i} = 1$
- prediction is done via

$$p_t = \begin{cases} 1, & \text{if } w_t^\top v_t \geq 1/2 \\ 0, & \text{if } w_t^\top v_t < 1/2 \end{cases}$$

- loss $\ell_t(w) = 2 \left| w^\top v_t - y_t \right| \mathbb{I}[p_t \neq y_t]$
- ℓ_t is convex
- $\ell_t \geq |p_t - y_t|$

Follow-The-Leader

$$\forall t, w_t = \arg\min_{w \in S} \sum_{i=1}^{t-1} \ell_i(w)$$

Follow-The-Regularized-Leader

$$\forall t, w_t = \arg\min_{w \in S} \sum_{i=1}^{t-1} \ell_i(w) + R(w)$$

Roughly the same in spirit as the Consistent algorithm.

- $\ell_t(w) = w^\top z_t$
- $R(w) = \frac{1}{2\nu} ||w||_2^2$

Exercise: find FTRL's w_{t+1} in a closed form

- $\ell_t(w) = w^\top z_t$
- $R(w) = \frac{1}{2\nu} ||w||_2^2$

Exercise: find FTRL's w_{t+1} in a closed form

$$w_{t+1} = -\nu \sum_{i=1}^{t} z_i = w_t - \nu z_t$$

- $\ell_t(w) = w^\top z_t$
- $R(w) = \frac{1}{2\nu} ||w||_2^2$

Exercise: find FTRL's w_{t+1} in a closed form

$$w_{t+1} = -\nu \sum_{i=1}^{t} z_i = w_t - \nu z_t$$

$$w_{t+1} = w_t - \nu z_t$$

Linear loss is a special case that links FTRL to SGD.

Lemma

$$\sum_{t=1}^{T} \ell_t(w_t) - \ell_t(u) \leq R(u) - R(w_1) + \sum_{t=1}^{T} \ell_t(w_t) - \ell_t(w_{t+1})$$

Proof idea:

- set $f_0 = R$
- proof by induction

Theorem

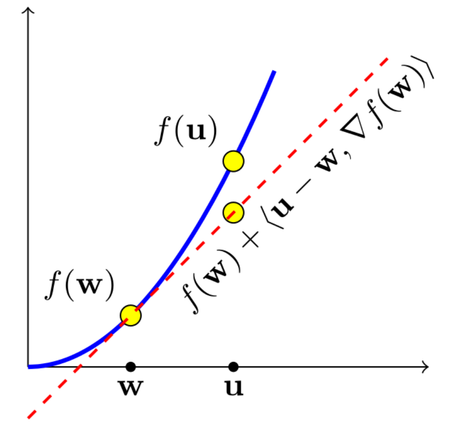
Consider FTRL, linear losses $\ell_t(w) = w^\top z_t$, and Euclidean regularization $R(w) = \frac{1}{2\nu} ||w||_2^2$ and $w, u \in S = \mathbb{R}^d$, then

$$R_T(u) \leq \frac{1}{2\nu} ||u||_2^2 + \nu \sum_{t=1}^{T} ||z_t||_2^2 .$$

Proof:

$$\begin{aligned} R_T(u) &\overset{\text{lemma}}{\leq} R(u) - R(w_1) + \sum_{t=1}^{T} \ell_t(w_t) - \ell_t(w_{t+1}) \\ &\leq \frac{1}{2\nu} ||u||_2^2 + \sum_{t=1}^{T} (w_t - w_{t+1})^\top z_t \\ &\leq \frac{1}{2\nu} ||u||_2^2 + \nu \sum_{t=1}^{T} ||z_t||_2^2 \end{aligned}$$

Convex functions: $\forall u \in S, f(u) \geq f(w) + (u-w)^\top z$, $z \in \partial f$



For convex ℓ_t it follows that

$$\sum_{t=1}^{T} \ell_t(w_t) - \ell_t(u) \leq \sum_{t=1}^{T} w_t^\top z_t - u^\top z_t.$$

- regret on convex functions is upper bounded by regret on tangent linear functions
- if we use sub-gradients as linear approximations of convex functions, we get the regret bound:

$$R_T(u) \leq \frac{1}{2\nu} ||u||_2^2 + \nu \sum_{t=1}^{T} ||\nabla \ell_t||_2^2 .$$

Turns out SGD is an instance of FTRL!

- if $||\nabla \ell_t||_2^2 \leq TL^2$ and $||u||_2^2 \leq B$, minimizing wrt. ν

$$R_T(u) \leq BL\sqrt{2T}.$$

In general:

Theorem [Shalev-Shwartz'12]

For strongly convex R (not only quadratic), the regret w.r.t $u \in S$

$$\sum_{t=1}^{T} \ell_t(w_t) - \min_{u \in S} \sum_{t=1}^{T} \ell_t(u) = O(\sqrt{T})$$

Exact bound depends on

- the actual form of R
- the class of ℓ_t (linear, quadratic, etc.)
- other assumptions on S and ℓ_t

The average regret $R_T/T \to 0$.

- many IL algorithms call online learning as a subroutine
- all of deep learning is based on sub-gradient methods
- analysis and performance depends on the chosen algorithm/regularization
- understanding these foundations allows being more informed when
  - ➡ trying to improve IL approaches
  - ➡ deciding on the regularization, loss functions etc.

[Shalev-Shwartz'12] "Online Learning and Online Convex Optimization"