Cost optimization and online learning: a high level overview

Yoav Freund

January 5, 2010

Outline

Goals of the course

- The hedging / regret minimization framework.

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:
  1. Computer Resource management.

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:
  1. Computer Resource management.
  2. Estimation, Tracking and control.

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:
  1. Computer Resource management.
  2. Estimation, Tracking and control.
  3. Financial portfolio management.

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:
  1. Computer Resource management.
  2. Estimation, Tracking and control.
  3. Financial portfolio management.
  4. Many many more...

Goals of the course

- The hedging / regret minimization framework.
- A few hedging algorithms, with proofs.
- A more theoretical treatment available in books, many papers, and previous course (available online).
- Applications:
  1. Computer Resource management.
  2. Estimation, Tracking and control.
  3. Financial portfolio management.
  4. Many many more...
- This class: An introduction through a resource management example.

Resource sharing in computation

- Traditional resources:

Resource sharing in computation

- Traditional resources:
  1. Run time.

Resource sharing in computation

- Traditional resources:
  1. Run time.
  2. Memory.

Resource sharing in computation

- Traditional resources:
  1. Run time.
  2. Memory.
- Emerging resources:

Resource sharing in computation

- Traditional resources:
  1. Run time.
  2. Memory.
- Emerging resources:
  1. Energy consumption.

Resource sharing in computation

- Traditional resources:
  1. Run time.
  2. Memory.
- Emerging resources:
  1. Energy consumption.
  2. Cooling.

Resource sharing in computation

- Traditional resources:
  1. Run time.
  2. Memory.
- Emerging resources:
  1. Energy consumption.
  2. Cooling.
  3. Communication bandwidth.

The changing landscape

- Moore's law

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward
  1. Multi-core.

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward
  1. Multi-core.
  2. Datacenters instead of super-computers.

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward
  1. Multi-core.
  2. Datacenters instead of super-computers.
- Computation as a commodity

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward
  1. Multi-core.
  2. Datacenters instead of super-computers.
- Computation as a commodity
  1. From the Lab, to the Office, to the home.

The changing landscape

- Moore's law
  1. Doubling of clock speed: stopped around 2000.
  2. Doubling of number of transistors: continuing.
- Parallelism is the only way forward
  1. Multi-core.
  2. Datacenters instead of super-computers.
- Computation as a commodity
  1. From the Lab, to the Office, to the home.
  2. Movement from one-of(protein folding) to scalable applications (google maps).

Resource allocation / the standard approach

- Several tasks need to use a common resource.

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.
- Problems with priorities:

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.
- Problems with priorities:
  - Who assigns priorities? (Unix “nice”)

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.
- Problems with priorities:
  - Who assigns priorities? (Unix “nice”)
  - Low priority queues can get starved. (“Your call will be answered in the order in which it was recieved.”)

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.
- Problems with priorities:
  - Who assigns priorities? (Unix “nice”)
  - Low priority queues can get starved. (“Your call will be answered in the order in which it was recieved.”)
  - No deadline guarantees (real-time operating systems).

Resource allocation / the standard approach

- Several tasks need to use a common resource.
- The scheduler arbitrates use of resource.
- Conflicts are resolved based on priorities
  - High priority tasks complete before low priority tasks.
  - Tasks with the same priority wait in a queue.
  - Example: Fatal error > user input > computation > Housekeeping.
- Problems with priorities:
  - Who assigns priorities? (Unix “nice”)
  - Low priority queues can get starved. (“Your call will be answered in the order in which it was recieved.”)
  - No deadline guarantees (real-time operating systems).
  - Very hard to scale up to large distributed systems.

An economic model of resource allocation

- Assign price to each resource.

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.
- Scheduler task is to maximize
  profit = total revenues - total costs.

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.
- Scheduler task is to maximize
  profit = total revenues - total costs.
- Example: scheduling search tasks on a 4-core machine.

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.
- Scheduler task is to maximize
  profit = total revenues - total costs.
- Example: scheduling search tasks on a 4-core machine.
  - Revenue: search completed < 0.5 sec – ¢0.1

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.
- Scheduler task is to maximize
  profit = total revenues - total costs.
- Example: scheduling search tasks on a 4-core machine.
  - Revenue: search completed < 0.5 sec – ¢0.1
  - Total cost: ¢10 per hour.

An economic model of resource allocation

- Assign price to each resource.
- Assign revenue to the completion of each task.
- Scheduler task is to maximize
  profit = total revenues - total costs.
- Example: scheduling search tasks on a 4-core machine.
  - Revenue: search completed < 0.5 sec – ¢0.1
  - Total cost: ¢10 per hour.
  - Scheduling heuristic: Schedule tasks in order recieved, occupying all available cores.

Contension resolution through prices

- Profit analysis

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.
- **bursty load:** we get 1000 queries in 10 seconds and nothing the rest of the hour.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.
- **bursty load:** we get 1000 queries in 10 seconds and nothing the rest of the hour.
  - Potential revenue: $1.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.
- **bursty load:** we get 1000 queries in 10 seconds and nothing the rest of the hour.
  - Potential revenue: $1.
  - Revenue with limited resources: /textcent20

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.
- **bursty load:** we get 1000 queries in 10 seconds and nothing the rest of the hour.
  - Potential revenue: $1.
  - Revenue with limited resources: /textcent20
  - Better to use $5 \times 4 = 20$ cores for 10 seconds and nothing for the rest of the hour.

Contension resolution through prices

- Profit analysis
  - Assume average search taked 0.1 sec on a single core.
  - Max throughput: 40 queries per second.
  - Max Revenue: ¢2 per second. Break even after 5 seconds / hour.
- **bursty load:** we get 1000 queries in 10 seconds and nothing the rest of the hour.
  - Potential revenue: $1.
  - Revenue with limited resources: /textcent20
  - Better to use $5 \times 4 = 20$ cores for 10 seconds and nothing for the rest of the hour.
- Can we benefit from computation on the cloud?

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.
- Reserving computers.

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.
- Reserving computers.
  - Paying $227.5 per year reduces the price to ¢4 per hour.

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.
- Reserving computers.
  - Paying $227.5 per year reduces the price to ¢4 per hour.
  - Break even point is 28%

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.
- Reserving computers.
  - Paying $227.5 per year reduces the price to ¢4 per hour.
  - Break even point is 28%
  - If load in coming year > 28% - reservation is worthwhile.

Amazon's Elastic cloud (EC2)

- Small instance, Windows OS, North California
  - cost ¢13 per hour
  - $93.6 per month if used 100% of time.
- Reserving computers.
  - Paying $227.5 per year reduces the price to ¢4 per hour.
  - Break even point is 28%
  - If load in coming year > 28% - reservation is worthwhile.
- Spot pricing - a market system.

The spot market in EC2

Spot Instances

Spot Instances enable you to bid for unused Amazon EC2 capacity. Instances are charged the Spot Price, which is set by Amazon EC2 and fluctuates periodically depending on the supply of and demand for Spot Instance capacity. To use Spot Instances, you place a Spot Instance request, specifying the instance type, the Region desired, the number of Spot Instances you want to run, and the maximum price you are willing to pay per instance hour. To determine how that maximum price compares to past Spot Prices, the Spot Price history is available via the Amazon EC2 API and the AWS Management Console. If your maximum price bid exceeds the current Spot Price, your request is fulfilled and your instances will run until either you choose to terminate them or the Spot Price increases above your maximum price (whichever is sooner).

Simpler problem: putting screen to sleep

- Many variants: energy management in data-center, WiFi link, hard-disk spindown.

Simpler problem: putting screen to sleep

- Many variants: energy management in data-center, WiFi link, hard-disk spindown.
- Standard heuristic: timeout.

Simpler problem: putting screen to sleep

- Many variants: energy management in data-center, WiFi link, hard-disk spindown.
- Standard heuristic: timeout.
- Failures: screen turns off while watching a movie, giving a talk on the computer.

Simpler problem: putting screen to sleep

- Many variants: energy management in data-center, WiFi link, hard-disk spindown.
- Standard heuristic: timeout.
- Failures: screen turns off while watching a movie, giving a talk on the computer.
- Underlying problem: predicting when screen will be turned back on (user keystroke).

Relevant information for prediction

- time since last keystroke.

Relevant information for prediction

- time since last keystroke.
- number of keystrokes in last minute, 10 minutes, hour ...

Relevant information for prediction

- time since last keystroke.
- number of keystrokes in last minute, 10 minutes, hour ...
- active application (powerpoint, DVD player, skype ...)

Relevant information for prediction

- time since last keystroke.
- number of keystrokes in last minute, 10 minutes, hour ...
- active application (powerpoint, DVD player, skype ...)
- detection of user through microphone, video camera ...

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.
- Timeout Expert: Keyboard idle for 5 minutes.

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.
- Timeout Expert: Keyboard idle for 5 minutes.
- Relative Timeout Expert: Kbd idle $\geq$
  ($C\times$ average time between keystrokes in last hour)

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.
- Timeout Expert: Keyboard idle for 5 minutes.
- Relative Timeout Expert: Kbd idle $\geq$
  ($C\times$ average time between keystrokes in last hour)
- Night Expert: Kbd idle 30 minutes after 6pm.

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.
- Timeout Expert: Keyboard idle for 5 minutes.
- Relative Timeout Expert: Kbd idle $\geq$ ($C\times$ average time between keystrokes in last hour)
- Night Expert: Kbd idle 30 minutes after 6pm.
- App Expert: Keep on 2 hours if DVD is playing.

Heuristics / Experts

- Expert = a rule for deciding when to turn off screen.
- Timeout Expert: Keyboard idle for 5 minutes.
- Relative Timeout Expert: Kbd idle $\geq$ ($C\times$ average time between keystrokes in last hour)
- Night Expert: Kbd idle 30 minutes after 6pm.
- App Expert: Keep on 2 hours if DVD is playing.
- Sensor Expert: Don't turn off if detecting user watching screen.

Hedging

- **Hedge:** A **master algorithm** that combines the experts.

Hedging

- **Hedge:** A **master algorithm** that combines the experts.
- Each Expert is assigned a **weight**.

Hedging

- **Hedge:** A **master algorithm** that combines the experts.
- Each Expert is assigned a **weight**.
  - Large weight - expert is performing well - use this expert more.

Hedging

- **Hedge:** A **master algorithm** that combines the experts.
- Each Expert is assigned a **weight**.
  - Large weight - expert is performing well - use this expert more.
  - Small weight - expert performing poorly - use this expert less.

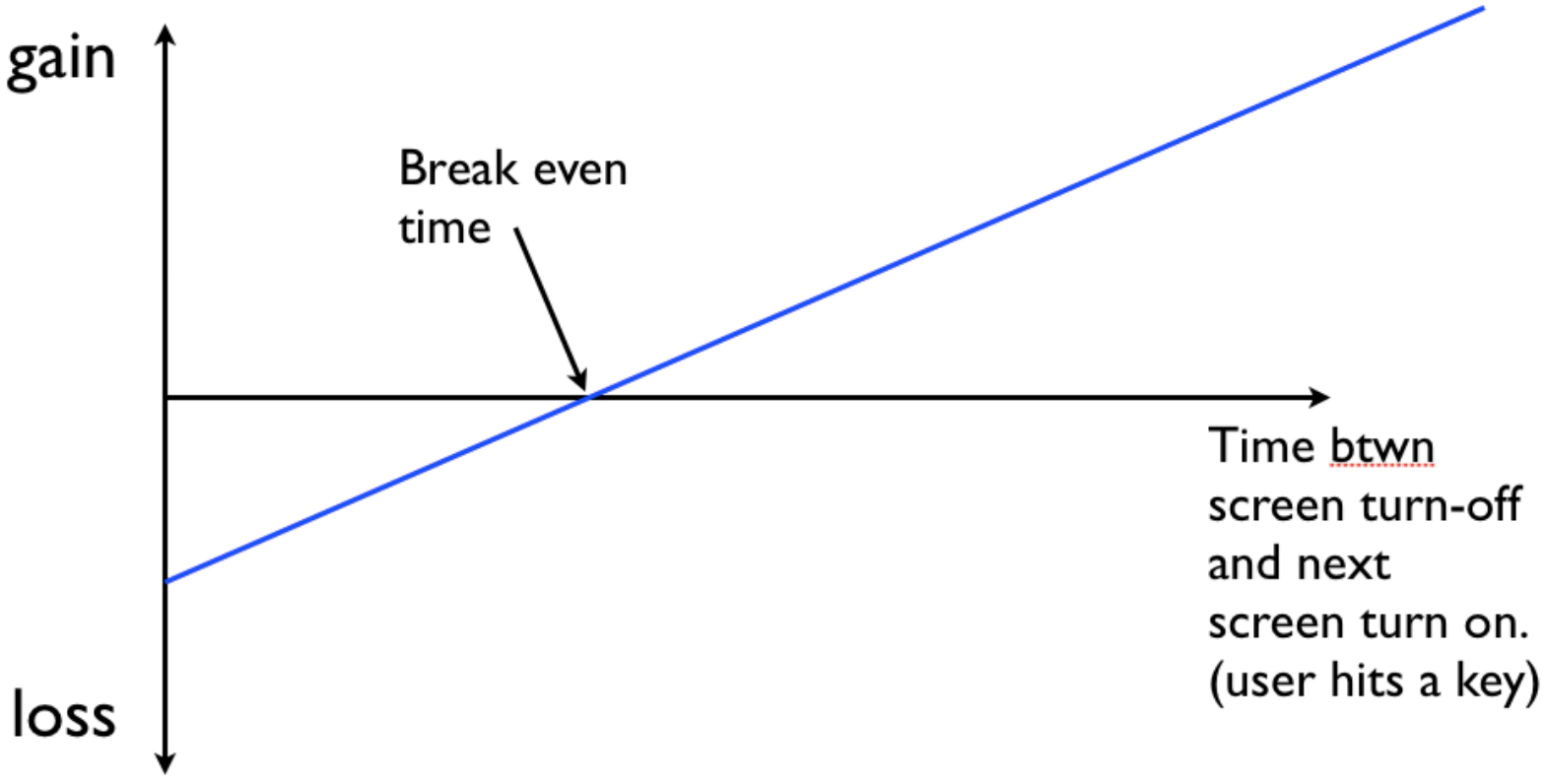
Hedging

- **Hedge:** A **master algorithm** that combines the experts.
- Each Expert is assigned a **weight**.
  - Large weight - expert is performing well - use this expert more.
  - Small weight - expert performing poorly - use this expert less.
- **Goal of Hedge:** Perform almost as well as the best expert in hind-sight.

Hedging

- **Hedge:** A **master algorithm** that combines the experts.
- Each Expert is assigned a **weight**.
  - Large weight - expert is performing well - use this expert more.
  - Small weight - expert performing poorly - use this expert less.
- **Goal of Hedge:** Perform almost as well as the best expert in hind-sight.
- **Performance measure:** cumulative gain/loss.

Loss/gain for single screen turn-off



Hedge: the rough idea

- Hedge updates the weights after each iteration - after the screen is turned back on.

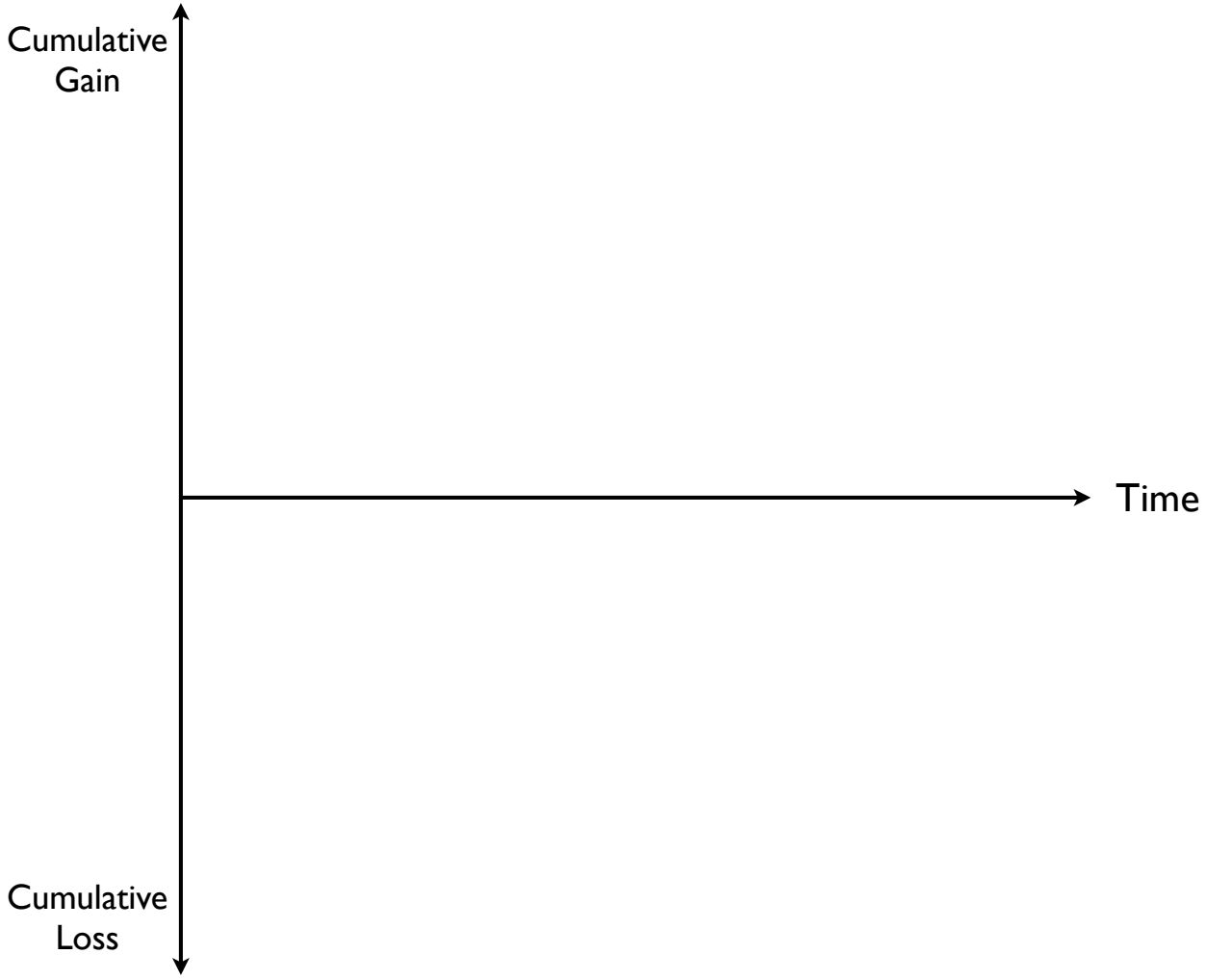
Hedge: the rough idea

- Hedge updates the weights after each iteration - after the screen is turned back on.
- Hedge decides what action to take based on the weighted recommendations of the experts.

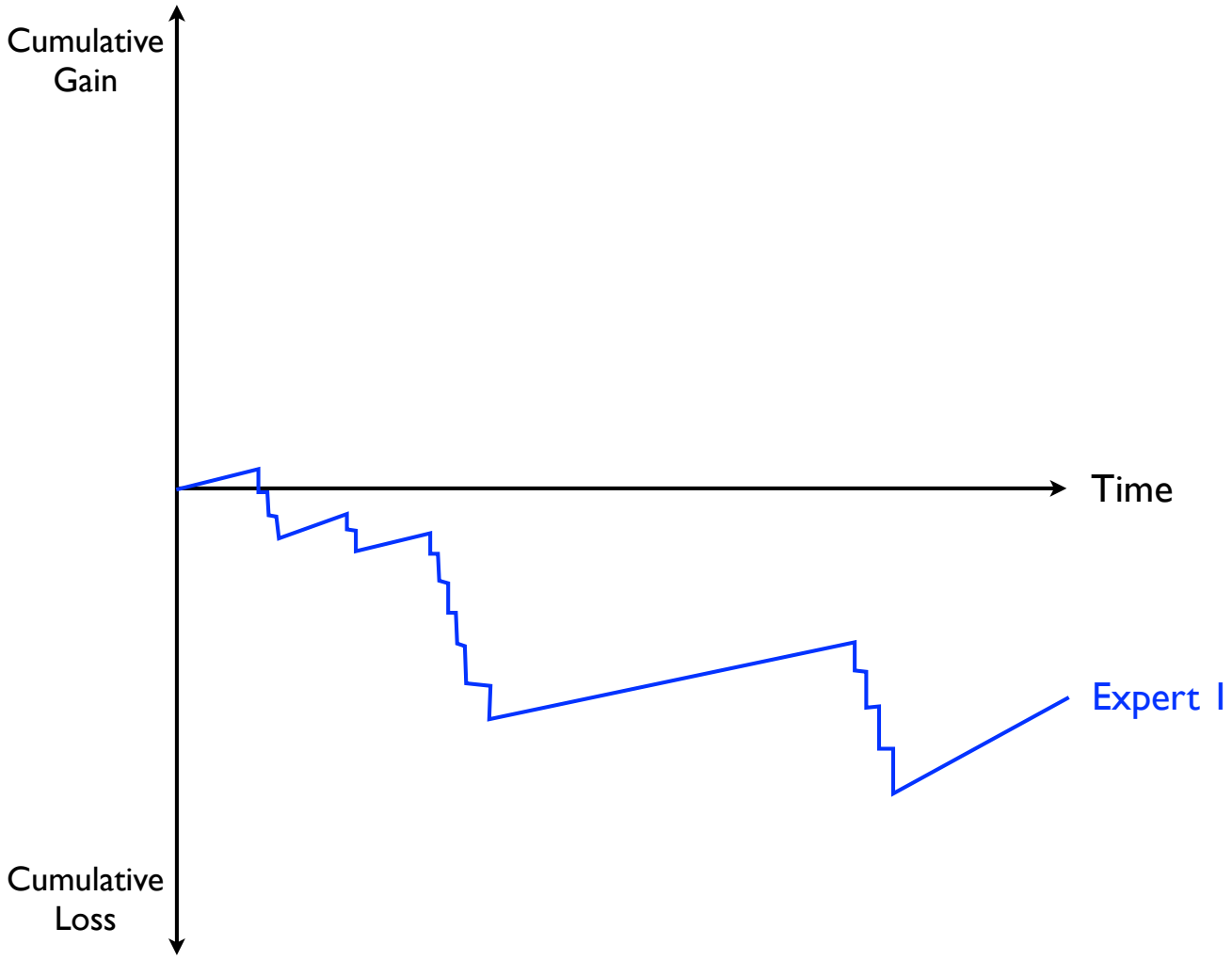
Hedge: the rough idea

- Hedge updates the weights after each iteration - after the screen is turned back on.
- Hedge decides what action to take based on the weighted recommendations of the experts.
- Hedge goal is to track the best performing expert but avoid flukes - good perfomance that is a result of random fluctuations.

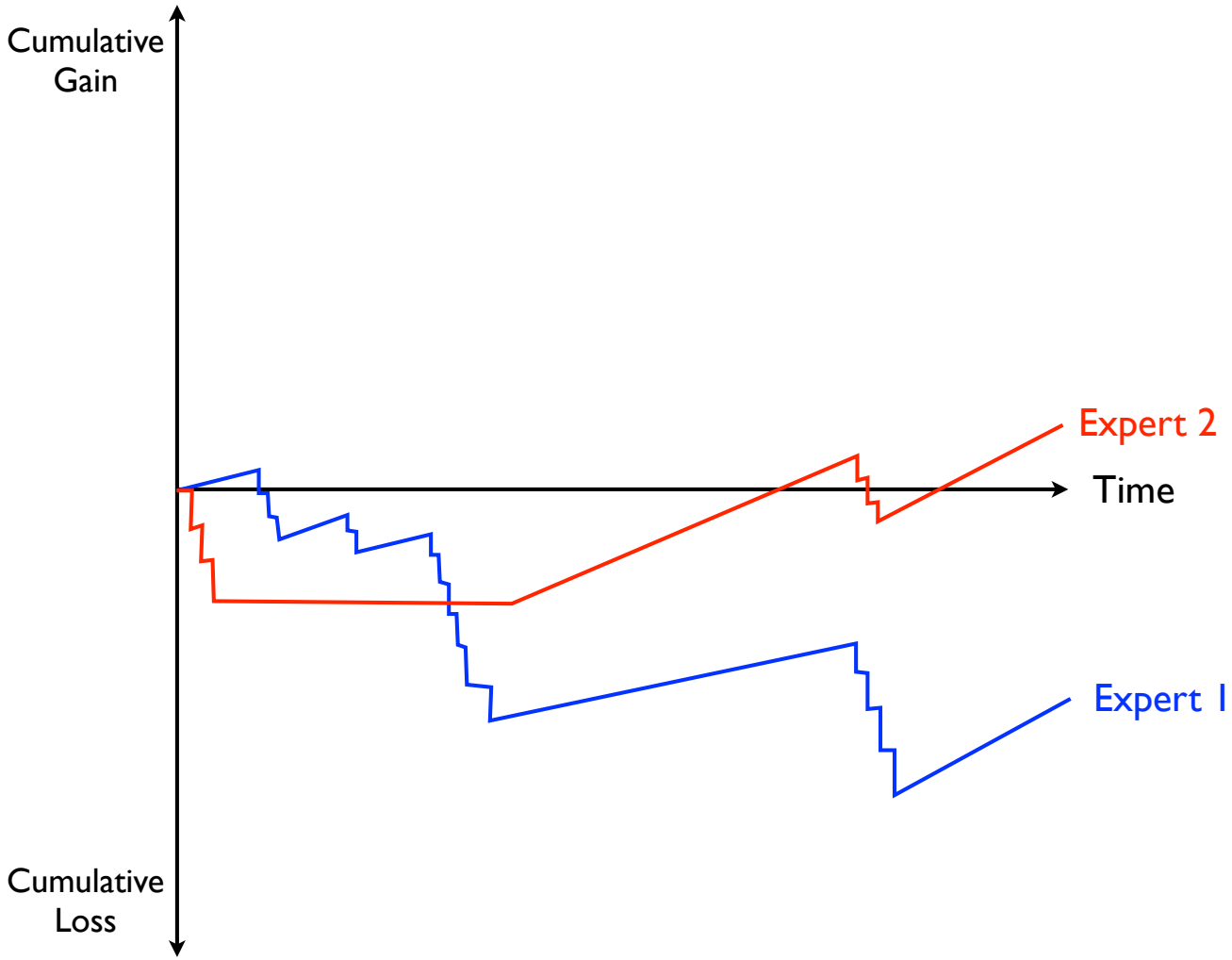
Loss/gain for single screen turn-off



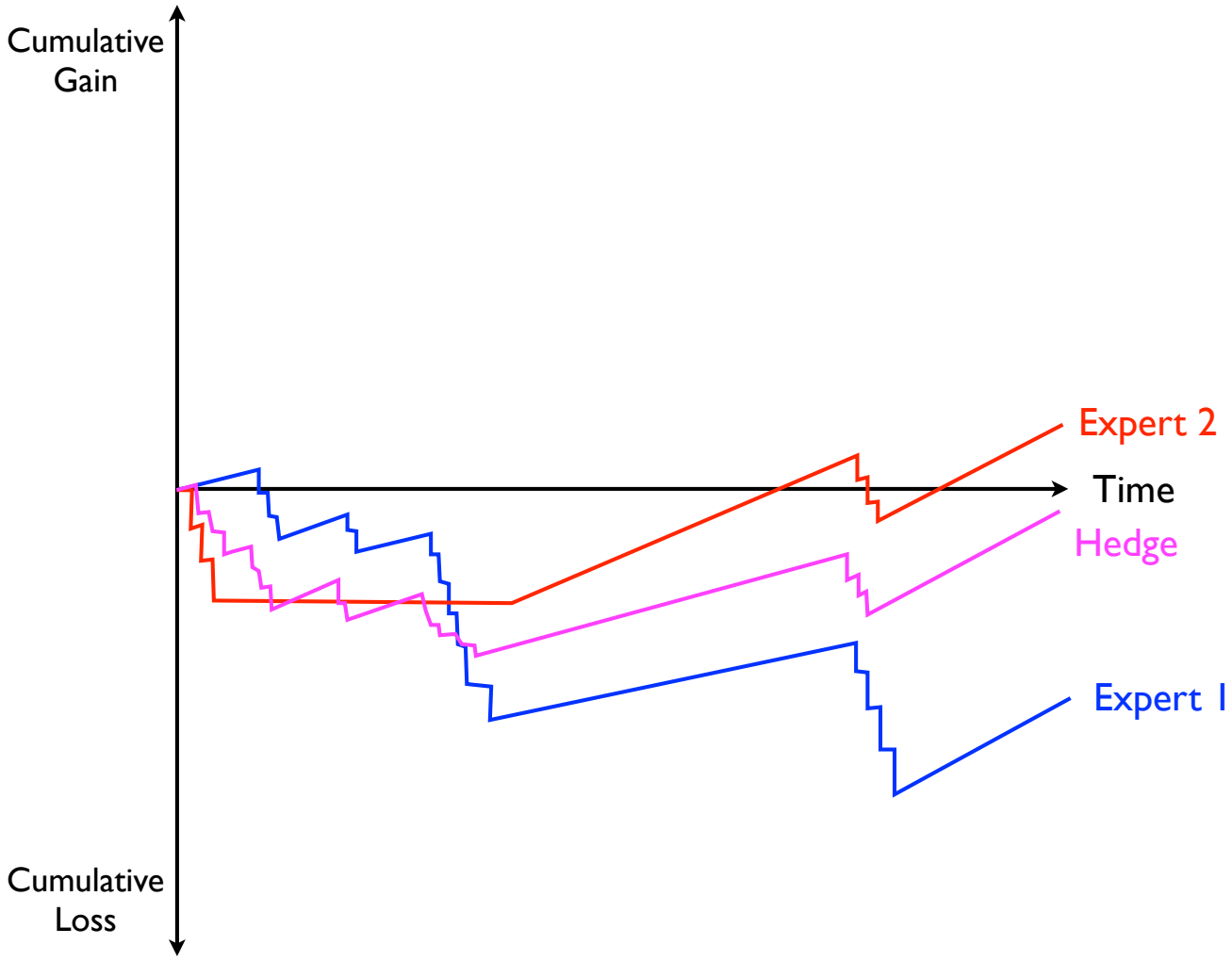
Loss/gain for single screen turn-off



Loss/gain for single screen turn-off



Loss/gain for single screen turn-off



Loss/gain for single screen turn-off