CSCI567 Machine Learning (Fall 2025)

Haipeng Luo

University of Southern California

Nov 14, 2025

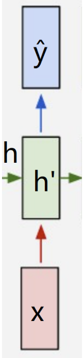
Administration

HW4 is due on Nov 26th.

Outline

Outline

A recurrent layer



from $\widehat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x})$ to $(\widehat{\boldsymbol{y}}, \boldsymbol{h}') = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{h})$

- $\boldsymbol{h}$ is "hidden state" (like HMM), updated via

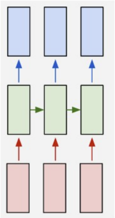
$$\boldsymbol{h}' = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}_h)$$

  where $\boldsymbol{\sigma}$ is an activation function

- $\widehat{\boldsymbol{y}} = \boldsymbol{V}\boldsymbol{h}' + \boldsymbol{b}_y$ is the output

Recurrent layer applied recursively

Given a **sequence** $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, can apply $\boldsymbol{f}$ recursively:

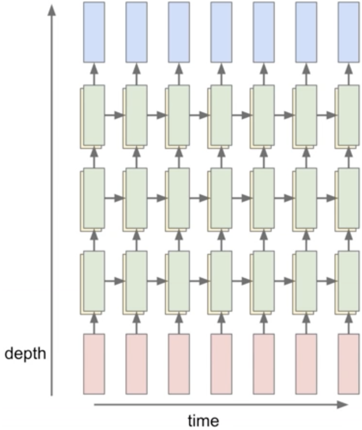


- $\boldsymbol{h}_0 = \mathbf{0}$
- $(\widehat{\boldsymbol{y}}_1, \boldsymbol{h}_1) = \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{h}_0)$
- $(\widehat{\boldsymbol{y}}_2, \boldsymbol{h}_2) = \boldsymbol{f}(\boldsymbol{x}_2, \boldsymbol{h}_1)$
- $\cdots$

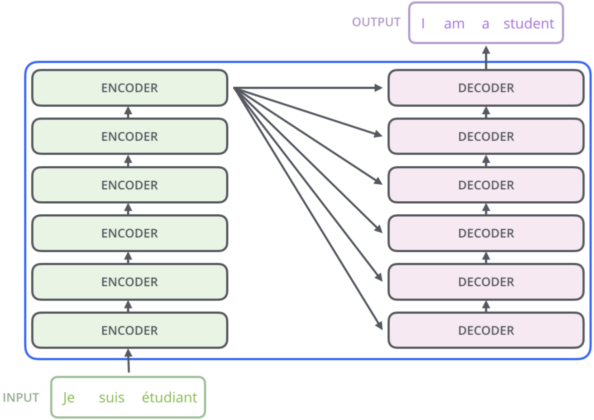
Making it "deep"

Stack multiple recurrent layers:

- hidden states become the inputs of the next layer
- different layers learn different $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}_h$
- last layer learns $\boldsymbol{V}, \boldsymbol{b}_y$ and output $\widehat{\boldsymbol{y}}$



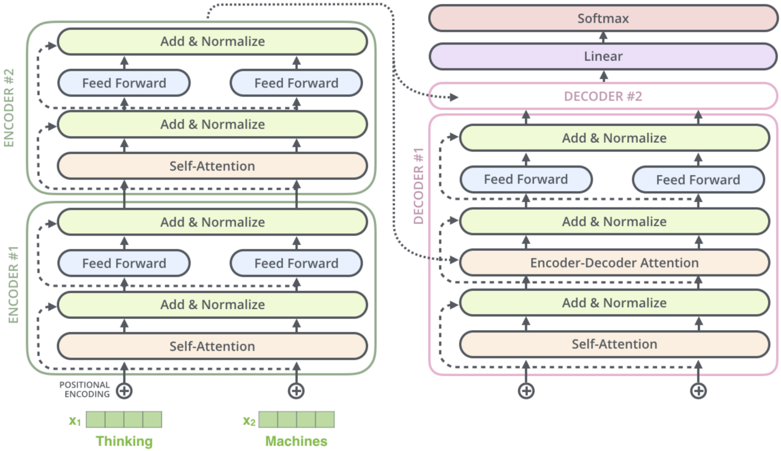
Transformer: overview



Encoder: summarizes the input into a useful representation

Decoder: generates outputs

Transformer: a closer look



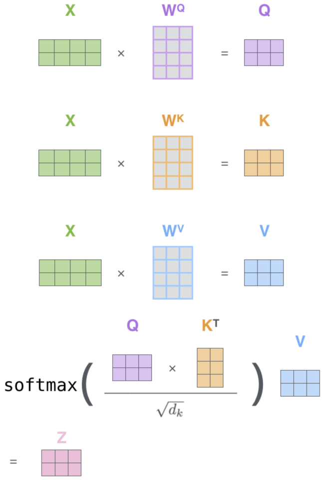
Transformer: a closer look at self-attention

Matrix notation:

- input matrix $\boldsymbol{X} \in \mathbb{R}^{T \times d}$, obtained by stacking $\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_T^\top$
- query matrix $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{T \times d_k}$
- key matrix $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{T \times d_k}$
- value matrix $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{T \times d_v}$
- attention score matrix $\boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{T \times T}$
- output matrix $\boldsymbol{Z} \in \mathbb{R}^{T \times d_v}$ is

$$\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

  where softmax is *applied row-wise*



$O(T^2)$ complexity (ignoring d, d_k, d_v)

Outline

1. Review of last lecture
2. Multi-Armed Bandits
   - Online decision making
   - Motivation and setup
   - Exploration vs. Exploitation
3. Dueling Bandits
4. Learning in Games

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online decision making problems**.

Examples

Amazon/Netflix/MSN **recommendation systems**:

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Reinforcement Learning

Reinforcement learning, a general framework to model such problems:

Reinforcement Learning

Reinforcement learning, a general framework to model such problems:

- focus on a special case called **multi-armed bandits** this week

Reinforcement Learning

Reinforcement learning, a general framework to model such problems:

- focus on a special case called **multi-armed bandits** this week
- discuss general reinforcement learning next week

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm

Of course there are many slot machines in the casino

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm

Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm

Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)
- **if I can play for 10 times, which machines should I play?**

Applications

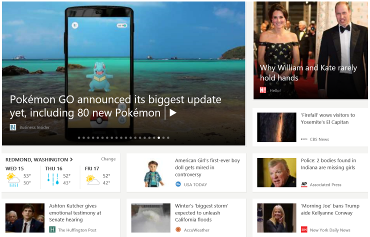
This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm

Applications

This simple model and its variants capture many real-life applications

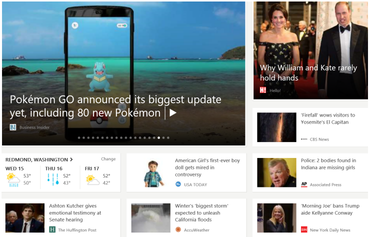
- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)



Applications

This simple model and its variants capture many real-life applications

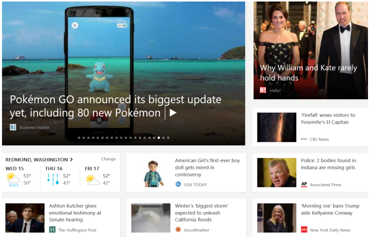
- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm

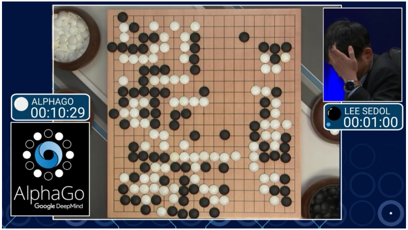


Applications

This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm (AlphaGo indeed has a bandit algorithm as one of the components)





Formal setup

There are K **arms** (actions/choices/options/etc.)

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner:

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

Formal setup

There are K **arms** (actions/choices/options/etc.)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

This kind of limited feedback is now usually referred to as bandit feedback

Objective

What is the goal of this problem?

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful,

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*.

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^T r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a\in[K]} \sum_{t=1}^T r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a\in[K]} \sum_{t=1}^T r_{t,a} - \sum_{t=1}^T r_{t,a_t}$$

This is called the **regret**: *how much I regret for not sticking with the best fixed arm in hindsight?*

Environments

How are the rewards generated by the environments?

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

We first focus on a simple setting:

- rewards of arm a are i.i.d. samples of $\mathrm{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1 - \mu_a$, independent of anything else.

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

We first focus on a simple setting:

- rewards of arm a are i.i.d. samples of $\mathrm{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1 - \mu_a$, independent of anything else.
- each arm has a different mean $(\mu_1, \ldots, \mu_K)$; the problem is essentially about finding the best arm $\mathrm{argmax}_a \, \mu_a$ as quickly as possible

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t:

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t : a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau = a]$$

is the **number of times** we have picked arm a.

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t:

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t : a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau = a]$$

is the **number of times** we have picked arm a.

Concentration: $\hat{\mu}_{t,a}$ should be close to μ_a if $n_{t,a}$ is large

Exploitation only

Greedy

Pick each arm once for the first K rounds.

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the algorithm first picks arm 1 and sees reward 0, then picks arm 2 and sees reward 1 (this happens with decent probability)

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the algorithm first picks arm 1 and sees reward 0, then picks arm 2 and sees reward 1 (this happens with decent probability)
- the algorithm will never pick arm 1 again!

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

We next discuss **three ways** to trade off exploration and exploitation for our simple multi-armed bandit setting.

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\text{argmax}_a \ \hat{\mu}_{T_0,a}$

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\mathrm{argmax}_a \ \hat{\mu}_{T_0,a}$

Parameter T_0 clearly controls the exploration/exploitation trade-off

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled for T_0/K times

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled for T_0/K times
- clearly it won't work if the environment is changing

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \; \hat{\mu}_{t-1,a}$

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \; \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \text{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K+1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \dots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \; \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

Is there a *more adaptive* way to explore?

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \; \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- exploration never stops due to the $\ln t$ term (i.e., no arms will be forever discarded) (see demo)

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathrm{UCB}_{t,a}$ where

$$\mathrm{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathrm{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- exploration never stops due to the $\ln t$ term (i.e., no arms will be forever discarded) (see demo)
- a parameter-free algorithm,

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- exploration never stops due to the $\ln t$ term (i.e., no arms will be forever discarded) (see demo)
- a parameter-free algorithm, and *it enjoys optimal regret!*

Upper confidence bound

picture link

Why is it called upper confidence bound?

Upper confidence bound

picture link

Why is it called upper confidence bound? One can prove that with high probability,

$$\mu_a \leq \mathsf{UCB}_{t,a}$$

so $\mathsf{UCB}_{t,a}$ is indeed an upper bound on the true mean.

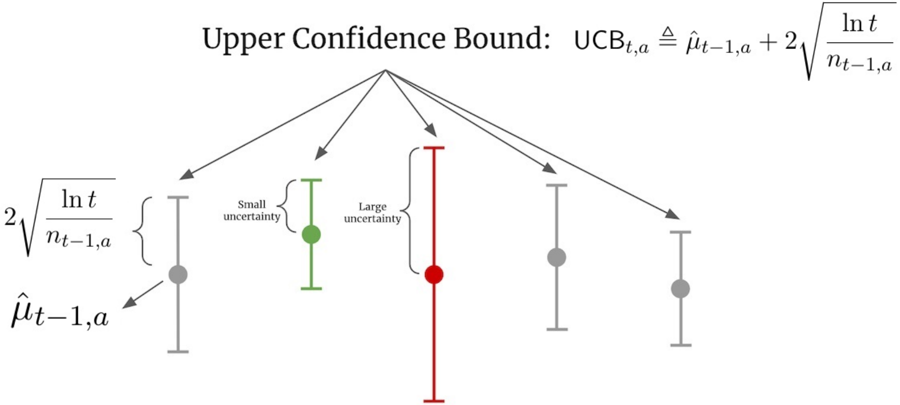
Upper confidence bound

picture link

Why is it called upper confidence bound? One can prove that with high probability,

$$\mu_a \leq \mathsf{UCB}_{t,a}$$

so $\mathsf{UCB}_{t,a}$ is indeed an upper bound on the true mean.



Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)

Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

This principle is useful for many other bandit problems.

Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

This principle is useful for many other bandit problems.

Q: what happens if we apply pessimism instead?

Lower Confidence Bound (LCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{LCB}_{t,a}$ where

$$\mathsf{LCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} - 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

Exploration via Optimism

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

This principle is useful for many other bandit problems.

Q: what happens if we apply pessimism instead?

Lower Confidence Bound (LCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathrm{LCB}_{t,a}$ where

$$\mathrm{LCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} - 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

Does not work: pessimism discourages exploration!

Outline

Limitation of standard multi-armed bandits

Direct numerical reward feedback is rare or hard to obtain.

Limitation of standard multi-armed bandits

Direct numerical reward feedback is rare or hard to obtain.

On the other hand, preference feedback is ubiquitous and easier to collect

Limitation of standard multi-armed bandits

Direct numerical reward feedback is rare or hard to obtain.

On the other hand, preference feedback is ubiquitous and easier to collect

- especially when only a pair of options is given: *do you prefer A or B?*

Limitation of standard multi-armed bandits

Direct numerical reward feedback is rare or hard to obtain.

On the other hand, preference feedback is ubiquitous and easier to collect

- especially when only a pair of options is given: *do you prefer A or B?*

- a key idea in training LLM, RLHF (reinforcement learning with human feedback) is exactly based on preference feedback for a pair of answers

Limitation of standard multi-armed bandits

Direct numerical reward feedback is rare or hard to obtain.

On the other hand, preference feedback is ubiquitous and easier to collect

- especially when only a pair of options is given: *do you prefer A or B?*

- a key idea in training LLM, RLHF (reinforcement learning with human feedback) is exactly based on preference feedback for a pair of answers

We focus on the special case of multi-armed bandits, i.e., **dueling bandits**.

Formal setup of dueling bandits

There are K **arms** (same as before)

Formal setup of dueling bandits

There are K **arms** (same as before)

The problem proceeds in rounds between the environment and a learner:

Formal setup of dueling bandits

There are K **arms** (same as before)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the learner picks a pair of arms $a_t, b_t \in [K]$

Formal setup of dueling bandits

There are K **arms** (same as before)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the learner picks a pair of arms $a_t, b_t \in [K]$
- the learner observes the binary preference feedback $\mathbb{I}[a_t \succ b_t]$: a_t is preferred over b_t or not

Formal setup of dueling bandits

There are K **arms** (same as before)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the learner picks a pair of arms $a_t, b_t \in [K]$
- the learner observes the binary preference feedback $\mathbb{I}[a_t \succ b_t]$: a_t is preferred over b_t or not

Importantly, learner does not observe

- any direct reward feedback

Formal setup of dueling bandits

There are K **arms** (same as before)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the learner picks a pair of arms $a_t, b_t \in [K]$
- the learner observes the binary preference feedback $\mathbb{I}[a_t \succ b_t]$: a_t is preferred over b_t or not

Importantly, learner does not observe

- any direct reward feedback
- preference for any other not selected pairs

How is the preference feedback generated?

Consider a simple setup:

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$
- $\mathbb{I}[a \succ b]$ is a sample of $\mathrm{Ber}(P_{a,b})$, that is, 1 with prob. $P_{a,b}$ and 0 with prob. $1 - P_{a,b}$.

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$
- $\mathbb{I}[a \succ b]$ is a sample of $\mathrm{Ber}(P_{a,b})$, that is, 1 with prob. $P_{a,b}$ and 0 with prob. $1 - P_{a,b}$.

Natural assumptions:

- $P_{a,b} + P_{b,a} = 1$ for any pair (a, b),

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$
- $\mathbb{I}[a \succ b]$ is a sample of Ber$(P_{a,b})$, that is, 1 with prob. $P_{a,b}$ and 0 with prob. $1 - P_{a,b}$.

Natural assumptions:

- $P_{a,b} + P_{b,a} = 1$ for any pair (a,b), which also implies $P_{a,a} = 0.5$

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$
- $\mathbb{I}[a \succ b]$ is a sample of Ber$(P_{a,b})$, that is, 1 with prob. $P_{a,b}$ and 0 with prob. $1 - P_{a,b}$.

Natural assumptions:

- $P_{a,b} + P_{b,a} = 1$ for any pair (a, b), which also implies $P_{a,a} = 0.5$
- $\exists$ "optimal action" $a^\star$ such that $P_{a^\star,b} \geq 0.5$ for any $b \in [K]$, called **Condorcet winner** (this assumption can be relaxed)

How is the preference feedback generated?

Consider a simple setup:

- $\exists$ a fixed but unknown **preference feedback matrix** $P \in [0,1]^{K \times K}$
- $\mathbb{I}[a \succ b]$ is a sample of Ber$(P_{a,b})$, that is, 1 with prob. $P_{a,b}$ and 0 with prob. $1 - P_{a,b}$.

Natural assumptions:

- $P_{a,b} + P_{b,a} = 1$ for any pair (a,b), which also implies $P_{a,a} = 0.5$
- $\exists$ "optimal action" $a^\star$ such that $P_{a^\star,b} \geq 0.5$ for any $b \in [K]$, called **Condorcet winner** (this assumption can be relaxed)

Intuitively, the goal is to find $a^\star$ as quickly as possible.

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t
- intuitively, we can treat $\mathbb{I}[a_t \succ b_t]$ as *reward* feedback (i.e., the goal is to beat the opponent as often as possible)

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t
- intuitively, we can treat $\mathbb{I}[a_t \succ b_t]$ as *reward* feedback (i.e., the goal is to beat the opponent as often as possible)
- this is exactly a standard multi-armed bandit problem!

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t
- intuitively, we can treat $\mathbb{I}[a_t \succ b_t]$ as *reward* feedback (i.e., the goal is to beat the opponent as often as possible)
- this is exactly a standard multi-armed bandit problem!
- now how should the "opponent" select b_t?

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t
- intuitively, we can treat $\mathbb{I}[a_t \succ b_t]$ as *reward* feedback (i.e., the goal is to beat the opponent as often as possible)
- this is exactly a standard multi-armed bandit problem!
- now how should the "opponent" select b_t? same, apply a multi-armed bandit algorithm, but with $\mathbb{I}[a_t \succ b_t]$ as *loss* feedback!

How to solve dueling bandits?

There are different approaches; we focus on the idea of **sparring/self-play**

- suppose we only get to select a_t, and an opponent/adversary selects b_t
- intuitively, we can treat $\mathbb{I}[a_t \succ b_t]$ as *reward* feedback (i.e., the goal is to beat the opponent as often as possible)
- this is exactly a standard multi-armed bandit problem!
- now how should the "opponent" select b_t? same, apply a multi-armed bandit algorithm, but with $\mathbb{I}[a_t \succ b_t]$ as *loss* feedback!
- usually apply two copies of the **same** algorithm to select a_t and b_t; hence the name **self-play**.

A closer look at self-play

"This is exactly a standard multi-armed bandit problem", *or is it?*

A closer look at self-play

"This is exactly a standard multi-armed bandit problem", *or is it?*

- the reward for picking arm a, $\mathbb{I}[a \succ b_t]$, no longer follows a fixed distribution due to the changing b_t

A closer look at self-play

"This is exactly a standard multi-armed bandit problem", *or is it?*

- the reward for picking arm a, $\mathbb{I}[a \succ b_t]$, no longer follows a fixed distribution due to the changing b_t $\Rightarrow$ *can't apply UCB anymore!*

A closer look at self-play

"This is exactly a standard multi-armed bandit problem", *or is it?*

- the reward for picking arm a, $\mathbb{I}[a \succ b_t]$, no longer follows a fixed distribution due to the changing $b_t \Rightarrow$ *can't apply UCB anymore!*
- in fact, b_t can be completely arbitrary

A closer look at self-play

"This is exactly a standard multi-armed bandit problem", *or is it?*

- the reward for picking arm a, $\mathbb{I}[a \succ b_t]$, no longer follows a fixed distribution due to the changing $b_t \Rightarrow$ *can't apply UCB anymore!*
- in fact, b_t can be completely arbitrary $\Rightarrow$ need an **"adversarial" multi-armed bandit algorithm**

Exponential Weights for **Exp**loration and **Exp**loitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Exponential Weights for **Exp**loration and **Exp**loitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exponential Weights for **Exp**loration and **Exp**loitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

Exponential Weights for **Exp**loration and **Exp**loitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$

Exponential Weights for **Exp**loration and **Exp**loitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$

Exponential Weights for Exploration and Exploitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$
- observe loss feedback $\mathbb{I}[a_t \succ b_t]$ (a_t selected by opponent)

Exponential Weights for Exploration and Exploitation

The very first adversarial multi-armed bandit algorithm: **Exp3**

Applying Exp3 to select b_t (will come back to a_t later):

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$
- observe loss feedback $\mathbb{I}[a_t \succ b_t]$ (a_t selected by opponent)
- construct loss estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Well-defined using only the information $\mathbb{I}[a_t \succ b_t]$.

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Well-defined using only the information $\mathbb{I}[a_t \succ b_t]$.

Why scale by inverse importance weight? Make the estimator *unbiased*:

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Well-defined using only the information $\mathbb{I}[a_t \succ b_t]$.

Why scale by inverse importance weight? Make the estimator *unbiased*:

$$\mathbb{E}\left[\boldsymbol{\ell}_{t,b}\right] = (1 - q_{t,b}) \times 0 + q_{t,b} \times \frac{\mathbb{I}[a_t \succ b]}{q_{t,b}} = \mathbb{I}[a_t \succ b]$$

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Well-defined using only the information $\mathbb{I}[a_t \succ b_t]$.

Why scale by inverse importance weight? Make the estimator *unbiased*:

$$\mathbb{E}\left[\boldsymbol{\ell}_{t,b}\right] = (1 - q_{t,b}) \times 0 + q_{t,b} \times \frac{\mathbb{I}[a_t \succ b]}{q_{t,b}} = \mathbb{I}[a_t \succ b]$$

Estimate the loss of all K actions by only seeing the loss of one action
(think how SGD estimates the gradient using only one example)

Importance-weighted loss estimators

Exp3 for dueling bandits (selecting b_t)

For $t = 1, \ldots, T$,

- $\cdots$
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Well-defined using only the information $\mathbb{I}[a_t \succ b_t]$.

Why scale by inverse importance weight? Make the estimator *unbiased*:

$$\mathbb{E}\left[\boldsymbol{\ell}_{t,b}\right] = (1 - q_{t,b}) \times 0 + q_{t,b} \times \frac{\mathbb{I}[a_t \succ b]}{q_{t,b}} = \mathbb{I}[a_t \succ b]$$

Estimate the loss of all K actions by only seeing the loss of one action
(think how SGD estimates the gradient using only one example)

Only selected action could have positive loss $\Rightarrow$ **exploration encouraged!**

Full algorithm

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$
- observe loss feedback $\mathbb{I}[a_t \succ b_t]$ (a_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t=b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

Full algorithm

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$
- observe loss feedback $\mathbb{I}[a_t \succ b_t]$ (a_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t=b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

- η controls exploration-exploitation trade-off

Full algorithm

Exp3 for dueling bandits (selecting b_t)

Input: a learning rate parameter $\eta > 0$

For $t = 1, \ldots, T$,

- compute arm distribution $\boldsymbol{q}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- sample b_t from $\boldsymbol{q}_t$
- observe loss feedback $\mathbb{I}[a_t \succ b_t]$ (a_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each b: $\boldsymbol{\ell}_{t,b} = \frac{\mathbb{I}[b_t = b]\mathbb{I}[a_t \succ b]}{q_{t,b}}$

- η controls exploration-exploitation trade-off
- with the right choice of η, Exp3 ensures regret $\lesssim \sqrt{KT}$ (**optimal**) even against an adversarial exponent

Losses versus rewards

Importantly, exploration is encouraged when using *losses but not rewards*

Losses versus rewards

Importantly, exploration is encouraged when using *losses but not rewards*

To understand this, go back to the question: *how should* a_t *be selected?*

Losses versus rewards

Importantly, exploration is encouraged when using *losses but not rewards*

To understand this, go back to the question: *how should a_t be selected?*

Self-play means applying Exp3 to select a_t as well,

Losses versus rewards

Importantly, exploration is encouraged when using *losses but not rewards*

To understand this, go back to the question: *how should a_t be selected?*

Self-play means applying Exp3 to select a_t as well, so how about:

Exp3 for dueling bandits (selecting a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{r}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$

Losses versus rewards (cont.)

Exp3 for dueling bandits (WRONG way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{r}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t = a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$

Losses versus rewards (cont.)

Exp3 for dueling bandits (**WRONG** way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{r}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$

- only selected action could have positive reward $\Rightarrow$ **exploration discouraged!**

Losses versus rewards (cont.)

Exp3 for dueling bandits (**WRONG** way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{r}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t = a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$

- only selected action could have positive reward $\Rightarrow$ **exploration discouraged!**
- after a_t wins one round, the algorithm increases its weight and decreases all other arms' weights aggressively

Losses versus rewards (cont.)

Exp3 for dueling bandits (**CORRECT** way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$

Losses versus rewards (cont.)

Exp3 for dueling bandits (**CORRECT** way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$

- from softmax $\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$ to softmax $\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$

Losses versus rewards (cont.)

Exp3 for dueling bandits (**CORRECT** way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$

- from softmax $\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$ to softmax $\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- from $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$ to $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$

Losses versus rewards (cont.)

Exp3 for dueling bandits (CORRECT way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$

- from softmax $\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$ to softmax $\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- from $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \succ b_t]}{p_{t,a}}$ to $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a \prec b_t]}{p_{t,a}}$
- when a_t wins, from "increase p_{t,a_t} and decrease everything else" to "no updates"

Losses versus rewards (cont.)

Exp3 for dueling bandits (CORRECT way to select a_t)

For $t = 1, \ldots, T$,

- sample a_t from arm distribution $\boldsymbol{p}_t = \text{softmax}\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- observe reward feedback $\mathbb{I}[a_t \succ b_t]$ (b_t selected by opponent)
- construct estimator $\boldsymbol{\ell}_t \in \mathbb{R}_+^K$ where for each a: $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a\prec b_t]}{p_{t,a}}$

- from softmax $\left(\eta \sum_{\tau=1}^{t-1} \boldsymbol{r}_\tau\right)$ to softmax $\left(-\eta \sum_{\tau=1}^{t-1} \boldsymbol{\ell}_\tau\right)$
- from $\boldsymbol{r}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a\succ b_t]}{p_{t,a}}$ to $\boldsymbol{\ell}_{t,a} = \frac{\mathbb{I}[a_t=a]\mathbb{I}[a\prec b_t]}{p_{t,a}}$
- when a_t wins, from "increase p_{t,a_t} and decrease everything else" to "no updates"
- when a_t loses, from "no updates" to "decrease p_{t,a_t} and increase everything else"

What does low regret mean here?

Using Exp3 to select a_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] \precsim \sqrt{KT}$$

What does low regret mean here?

Using Exp3 to select a_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] \lesssim \sqrt{KT}$$

Similarly, using Exp3 to select b_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ a^\star] \lesssim \sqrt{KT}$$

What does low regret mean here?

Using Exp3 to select a_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] \lesssim \sqrt{KT}$$

Similarly, using Exp3 to select b_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ a^\star] \lesssim \sqrt{KT}$$

Summing up gives $\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ a^\star] \lesssim \sqrt{KT}$

What does low regret mean here?

Using Exp3 to select a_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] \lesssim \sqrt{KT}$$

Similarly, using Exp3 to select b_t guarantees

$$\sum_{t=1}^{T} \mathbb{I}[a_t \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ a^\star] \lesssim \sqrt{KT}$$

Summing up gives $\sum_{t=1}^{T} \mathbb{I}[a^\star \succ b_t] - \sum_{t=1}^{T} \mathbb{I}[a_t \succ a^\star] \lesssim \sqrt{KT}$

Taking expectation and using $P_{a,b} + P_{b,a} = 1$ shows

$$\sum_{t=1}^{T} (P_{a^\star, a_t} + P_{a^\star, b_t} - 1) \lesssim \sqrt{KT},$$

so can only pick suboptimal actions a negligible fraction ($\sqrt{K/T}$) of times

Outline

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)
- can involve two or more players

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)
- can involve two or more players
- each player can have a different set of actions (arms)

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)
- can involve two or more players
- each player can have a different set of actions (arms)
- each player can have a different reward/loss function

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)
- can involve two or more players
- each player can have a different set of actions (arms)
- each player can have a different reward/loss function
- the game can be **competitive, cooperative, or a mix of both**

Self-play in general games

Dueling bandits can be seen as a **game** between two players

- each player makes a move each time and observes reward/loss

The idea of **self-play** naturally extends to other games:

- "games" here does not only narrowly refer to card/video games, but broadly any interaction between agents (from commuting to work, to arms race between countries)
- can involve two or more players
- each player can have a different set of actions (arms)
- each player can have a different reward/loss function
- the game can be **competitive, cooperative, or a mix of both**
- self-play: use the same learning algorithm for all players

Zero-sum games

Focus on a special case: **two-player zero-sum games**

Zero-sum games

Focus on a special case: **two-player zero-sum games**

- can be described by a **game matrix** $M \in [0,1]^{K \times J}$ (generalizing P)

Zero-sum games

Focus on a special case: **two-player zero-sum games**

- can be described by a **game matrix** $M \in [0,1]^{K \times J}$ (generalizing P)
- row-player has K actions, column-player has J actions

Zero-sum games

Focus on a special case: **two-player zero-sum games**

- can be described by a **game matrix** $M \in [0,1]^{K\times J}$ (generalizing P)
- row-player has K actions, column-player has J actions
- if row-player selects $a \in [K]$ and column-player selects $b \in [J]$, then row/column-player gets reward/loss $M_{a,b}$ (hence **zero-sum**)

Zero-sum games

Focus on a special case: **two-player zero-sum games**

- can be described by a **game matrix** $M \in [0,1]^{K\times J}$ (generalizing P)
- row-player has K actions, column-player has J actions
- if row-player selects $a \in [K]$ and column-player selects $b \in [J]$, then row/column-player gets reward/loss $M_{a,b}$ (hence **zero-sum**)
- canonical examples: **rock-paper-scissors**

$$\begin{array}{c} \\ \text{Rock} \\ \text{Paper} \\ \text{Scissors} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rock} & \text{Paper} & \text{Scissors} \end{array} \\ \begin{pmatrix} 0.5 & 0 & 1 \\ 1 & 0.5 & 0 \\ 0 & 1 & 0.5 \end{pmatrix} \end{array}$$

More interesting examples

Figure: chess

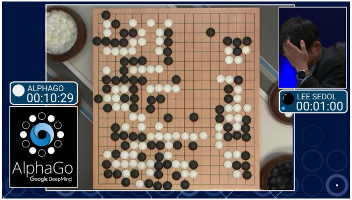


Figure: Go

Figure: poker

Figure: StarCraft

More interesting examples

Figure: chess

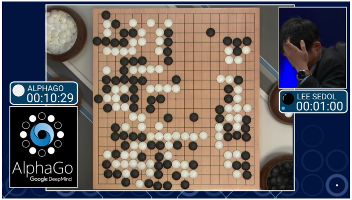


Figure: Go

Figure: poker

Figure: StarCraft

Extremely large action space; still, all can be *"solved" by self-play*

Solving Go via self-play

Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis

Nature **529**, 484–489 (2016) | Cite this article

Abstract

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks.

Solving poker games via self-play

Description of Pluribus

The core of Pluribus's strategy was computed through self-play, in which the AI plays against copies of itself, without any data of human or prior AI play used as input. The AI starts from scratch by playing randomly and gradually improves as it determines which actions, and which probability distribution over those actions, lead to better outcomes against earlier versions of its strategy. Forms of self-play have previously been used to generate powerful AIs in two-player zero-sum games such as backgammon (18), Go (9, 19), Dota 2 (20), StarCraft 2 (21), and two-player poker (4–6), though the precise algorithms that were used have varied widely. Although it is easy to construct toy games with more than two players in which commonly used self-play algorithms fail to converge to a meaningful solution (22), in practice self-play has nevertheless been shown to do reasonably well in some games with more than two players (23).

Nash Equilibra

What does "*solving a game*" mean?

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE),

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE), that is, a pair of distributions $p^\star$ (over $[K]$) and $q^\star$ (over $[J]$) such that

$$\max_p p^\top M q^\star = (p^\star)^\top M q^\star = \min_q (p^\star)^\top M q$$

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE), that is, a pair of distributions $p^\star$ (over $[K]$) and $q^\star$ (over $[J]$) such that

$$\max_p p^\top M q^\star = (p^\star)^\top M q^\star = \min_q (p^\star)^\top M q$$

- $p^\top M q$ is the expected reward/loss for the row/column-player if they sample their actions according to p and q respectively

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE), that is, a pair of distributions $p^\star$ (over $[K]$) and $q^\star$ (over $[J]$) such that

$$\max_p p^\top M q^\star = (p^\star)^\top M q^\star = \min_q (p^\star)^\top M q$$

- $p^\top M q$ is the expected reward/loss for the row/column-player if they sample their actions according to p and q respectively
- $(p^\star, q^\star)$ is an NE means $p^\star$ and $q^\star$ are **best-responding** to each other

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE), that is, a pair of distributions $p^\star$ (over $[K]$) and $q^\star$ (over $[J]$) such that

$$\max_p p^\top M q^\star = (p^\star)^\top M q^\star = \min_q (p^\star)^\top M q$$

- $p^\top M q$ is the expected reward/loss for the row/column-player if they sample their actions according to p and q respectively
- $(p^\star, q^\star)$ is an NE means $p^\star$ and $q^\star$ are **best-responding** to each other
- so <u>no player has incentive to deviate from an NE</u>

Nash Equilibra

What does "*solving a game*" mean?

For zero-sum games, this could mean finding a **Nash Equilibrium** (NE), that is, a pair of distributions $p^\star$ (over $[K]$) and $q^\star$ (over $[J]$) such that

$$\max_p p^\top M q^\star = (p^\star)^\top M q^\star = \min_q (p^\star)^\top M q$$

- $p^\top M q$ is the expected reward/loss for the row/column-player if they sample their actions according to p and q respectively
- $(p^\star, q^\star)$ is an NE means $p^\star$ and $q^\star$ are **best-responding** to each other
- so no player has incentive to deviate from an NE
- they are also the best strategies assuming the worst opponent:

$$p^\star = \operatorname*{argmax}_p \min_q p^\top M q \quad \text{and} \quad q^\star = \operatorname*{argmin}_q \max_p p^\top M q$$

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*?

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*? **Can approximately find one via self-play and regret minimization!**

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*? **Can approximately find one via self-play and regret minimization!**

Self-play for zero-sum games

Input: multi-armed bandit algorithms $\mathcal{A}$ and $\mathcal{B}$

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*? **Can approximately find one via self-play and regret minimization!**

Self-play for zero-sum games

Input: multi-armed bandit algorithms $\mathcal{A}$ and $\mathcal{B}$
For $t = 1, \ldots, T$,

- get arm distributions $\boldsymbol{p}_t$ and $\boldsymbol{q}_t$ from $\mathcal{A}$ and $\mathcal{B}$ respectively

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*? **Can approximately find one via self-play and regret minimization!**

Self-play for zero-sum games

Input: multi-armed bandit algorithms $\mathcal{A}$ and $\mathcal{B}$
For $t = 1, \ldots, T$,

- get arm distributions $\boldsymbol{p}_t$ and $\boldsymbol{q}_t$ from $\mathcal{A}$ and $\mathcal{B}$ respectively
- sample a_t from $\boldsymbol{p}_t$ and b_t from $\boldsymbol{q}_t$

How to find Nash Equilibra?

Easy for small games (e.g., a pair of uniform distributions is the NE for rock-paper-scissors)

But how about game *as large as poker*? **Can approximately find one via self-play and regret minimization!**

Self-play for zero-sum games

Input: multi-armed bandit algorithms $\mathcal{A}$ and $\mathcal{B}$
For $t = 1, \ldots, T$,

- get arm distributions $\boldsymbol{p}_t$ and $\boldsymbol{q}_t$ from $\mathcal{A}$ and $\mathcal{B}$ respectively
- sample a_t from $\boldsymbol{p}_t$ and b_t from $\boldsymbol{q}_t$
- observe M_{a_t,b_t} (plus noise), feed it as reward to $\mathcal{A}$ and as loss to $\mathcal{B}$

Low regret means convergence to NE

Suppose $\mathcal{A}$ and $\mathcal{B}$ have at most $\sqrt{T}$ regret:

$$\max_{a\in[K]} \sum_{t=1}^{T} M_{a,b_t} - \sum_{t=1}^{T} M_{a_t,b_t} \precsim \sqrt{T}$$

$$\sum_{t=1}^{T} M_{a_t,b_t} - \min_{b\in[J]} \sum_{t=1}^{T} M_{a_t,b} \precsim \sqrt{T}$$

Low regret means convergence to NE

Suppose $\mathcal{A}$ and $\mathcal{B}$ have at most $\sqrt{T}$ regret:

$$\max_{a \in [K]} \sum_{t=1}^{T} M_{a,b_t} - \sum_{t=1}^{T} M_{a_t,b_t} \precsim \sqrt{T}$$

$$\sum_{t=1}^{T} M_{a_t,b_t} - \min_{b \in [J]} \sum_{t=1}^{T} M_{a_t,b} \precsim \sqrt{T}$$

Summing up gives

$$\max_{a \in [K]} \sum_{t=1}^{T} M_{a,b_t} - \min_{b \in [J]} \sum_{t=1}^{T} M_{a_t,b} \precsim \sqrt{T}$$

Low regret means convergence to NE

Suppose $\mathcal{A}$ and $\mathcal{B}$ have at most $\sqrt{T}$ regret:

$$\max_{a \in [K]} \sum_{t=1}^{T} M_{a,b_t} - \sum_{t=1}^{T} M_{a_t,b_t} \lesssim \sqrt{T}$$

$$\sum_{t=1}^{T} M_{a_t,b_t} - \min_{b \in [J]} \sum_{t=1}^{T} M_{a_t,b} \lesssim \sqrt{T}$$

Summing up gives

$$\max_{a \in [K]} \sum_{t=1}^{T} M_{a,b_t} - \min_{b \in [J]} \sum_{t=1}^{T} M_{a_t,b} \lesssim \sqrt{T}$$

and thus

$$\max_{p} p^\top M \bar{q} - \min_{q} \bar{p}^\top M q \lesssim \frac{1}{\sqrt{T}}$$

for $\bar{p}$ and $\bar{q}$ being the **empirical distribution** of $a_{1:T}$ and $b_{1:T}$

Low regret means convergence to NE (cont.)

$$\max_p p^\top M \bar{q} - \min_q \bar{p}^\top M q \precsim \frac{1}{\sqrt{T}}$$

means

$$\bar{p}^\top M \bar{q} \leq \max_p p^\top M \bar{q} \precsim \min_q \bar{p}^\top M q + \frac{1}{\sqrt{T}}$$

Low regret means convergence to NE (cont.)

$$\max_p p^\top M\bar{q} - \min_q \bar{p}^\top Mq \lesssim \frac{1}{\sqrt{T}}$$

means

$$\bar{p}^\top M\bar{q} \le \max_p p^\top M\bar{q} \lesssim \min_q \bar{p}^\top Mq + \frac{1}{\sqrt{T}}$$

and

$$\bar{p}^\top M\bar{q} \ge \min_q \bar{p}^\top Mq \gtrsim \max_p p^\top M\bar{q} - \frac{1}{\sqrt{T}}$$

Low regret means convergence to NE (cont.)

$$\max_p p^\top M\bar{q} - \min_q \bar{p}^\top Mq \precsim \frac{1}{\sqrt{T}}$$

means

$$\bar{p}^\top M\bar{q} \leq \max_p p^\top M\bar{q} \precsim \min_q \bar{p}^\top Mq + \frac{1}{\sqrt{T}}$$

and

$$\bar{p}^\top M\bar{q} \geq \min_q \bar{p}^\top Mq \succsim \max_p p^\top M\bar{q} - \frac{1}{\sqrt{T}}$$

Therefore,

$$\max_p p^\top M\bar{q} - \frac{1}{\sqrt{T}} \precsim \bar{p}^\top M\bar{q} \precsim \min_q \bar{p}^\top Mq + \frac{1}{\sqrt{T}}$$

Low regret means convergence to NE (cont.)

$$\max_p p^\top M \bar{q} - \min_q \bar{p}^\top M q \precsim \frac{1}{\sqrt{T}}$$

means

$$\bar{p}^\top M \bar{q} \le \max_p p^\top M \bar{q} \precsim \min_q \bar{p}^\top M q + \frac{1}{\sqrt{T}}$$

and

$$\bar{p}^\top M \bar{q} \ge \min_q \bar{p}^\top M q \succsim \max_p p^\top M \bar{q} - \frac{1}{\sqrt{T}}$$

Therefore,

$$\max_p p^\top M \bar{q} - \frac{1}{\sqrt{T}} \precsim \bar{p}^\top M \bar{q} \precsim \min_q \bar{p}^\top M q + \frac{1}{\sqrt{T}}$$

meaning that $(\bar{p}, \bar{q})$ **converges to NE as** $T \to \infty$

Summary

Bandit problems, a special case of reinforcement learning

Summary

Bandit problems, a special case of reinforcement learning

- exploration and exploitation trade-off under limited feedback

Summary

Bandit problems, a special case of reinforcement learning

- exploration and exploitation trade-off under limited feedback
- **UCB** for direct reward feedback (adaptive exploration via optimism)

Summary

Bandit problems, a special case of reinforcement learning

- exploration and exploitation trade-off under limited feedback
- **UCB** for direct reward feedback (adaptive exploration via optimism)
- **self-play** with **Exp3** for preference feedback (dueling bandits)

Summary

Bandit problems, a special case of reinforcement learning

- exploration and exploitation trade-off under limited feedback
- **UCB** for direct reward feedback (adaptive exploration via optimism)
- **self-play** with **Exp3** for preference feedback (dueling bandits)
- self-play for solving games (low regret $\Rightarrow$ convergence to NE)