S&DS 365 / 665

Intermediate Machine Learning

Reinforcement Learning

October 30

Yale

Goings on

- Assignment 3 due Wednesday
- Assignment 4 posted Wednesday
- Quiz 4 next Wednesday
  - Variational inference and VAEs
  - Undirected graphs and glasso
  - Graph neural nets
- Wednesday: Deep Q-Learning

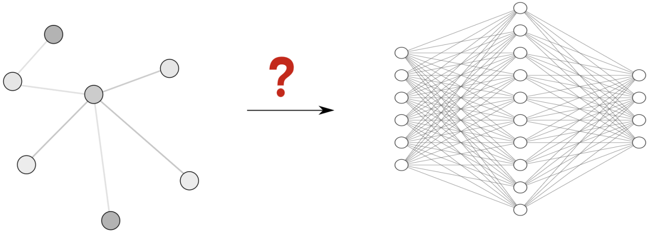
Outline for today

- Graph neural nets (continued)
- RL concepts
- Q-learning

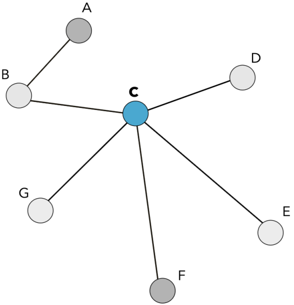
Graph neural networks

Let’s recap and continue graph neural networks from last time

How to use graph structure in neural nets?



Graph Laplacian



Input Graph G

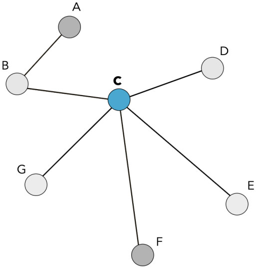
$$
\begin{array}{c}
\\ A \\ B \\ \mathbf{C} \\ D \\ E \\ F \\ G
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & \mathbf{C} & D & E & F & G \end{array} \\
\begin{bmatrix}
1 & -1 & & & & & \\
-1 & 2 & -1 & & & & \\
 & -1 & 5 & -1 & -1 & -1 & -1 \\
 & & -1 & 1 & & & \\
 & & -1 & & 1 & & \\
 & & -1 & & & 1 & \\
 & & -1 & & & & 1
\end{bmatrix}
\end{array}
$$

Laplacian L of G

Filters and Laplacians

- A smoothing kernel relates each position to its neighbors
- An edge filter compares input positions in a particular direction
- The graph Laplacian compares the value at each node to the values at neighboring nodes

Graph Laplacian



Input Graph G

$$
\begin{array}{c|ccccccc|}
 & A & B & \mathbf{C} & D & E & F & G \\
\hline
A & 1 & -1 & & & & & \\
B & -1 & 2 & -1 & & & & \\
\mathbf{C} & & -1 & 5 & -1 & -1 & -1 & -1 \\
D & & & -1 & 1 & & & \\
E & & & -1 & & 1 & & \\
F & & & -1 & & & 1 & \\
G & & & -1 & & & & 1
\end{array}
$$

Laplacian L of G

$$\frac{1}{5}(Lx)_C = x_C - \frac{1}{5}(x_B + x_G + x_F + x_E + x_D)$$
$$= \text{value at } C \text{ minus average of neighbors}$$

Polynomials of the Laplacian

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \cdots w_d L^d$$

If $\text{dist}(u, v) > i$ then the (u, v) entry of L^i is zero

- This is analogous to a CNN filter (kernel)
- The weights w_i play role of filter coefficients
- Degree d of polynomial plays role of the size of the kernel

Equivariance

If we compute a feature map f, we'd like it to not depend on a reordering P of the nodes.

The reordering

$$
\begin{aligned}
1 &\leftarrow 2 \\
2 &\leftarrow 3 \\
3 &\leftarrow 1
\end{aligned}
$$

corresponds to permutation matrix

$$
P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}
$$

Whence equivariance

A transformation $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is *equivariant* if

$$f(Px) = Pf(x)$$

for any permuation matrix P, where $PP^T = I$.

The transformed data and Laplacian are

$$x \longrightarrow Px$$
$$L \longrightarrow PLP^T$$
$$L^i \longrightarrow PL^iP^T$$

Whence equivariance

A transformation $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is *equivariant* if

$$f(Px) = Pf(x)$$

for any permuation matrix P, where $PP^T = I$.

The transformed polynomial kernels are

$$\begin{aligned} p_w(Px) &= \sum_{i=0}^{d} w_i (PL^i P^T) Px \\ &= \sum_{i=0}^{d} w_i PL^i x \\ &= P \sum_{i=0}^{d} w_i L^i x \\ &= P\, p_w(x) \end{aligned}$$

Building layers

Let $h^{(k)}$ be the neurons at layer k.

We start with $h^{(0)} = x$, a value x_j at each node j

The next layer is

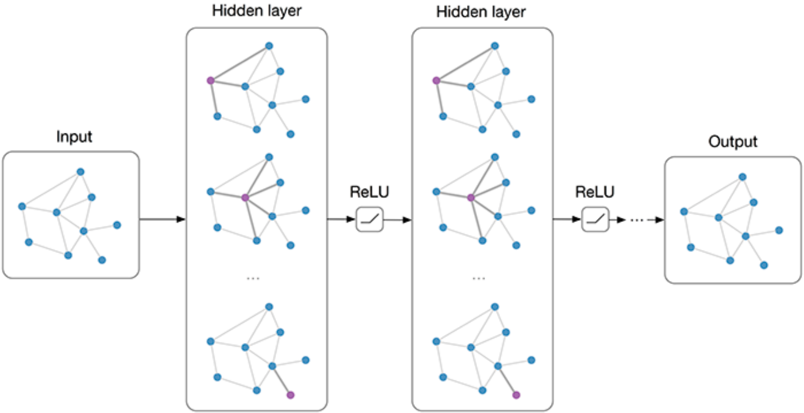
$$h^{(k+1)} = \varphi\left(p_{w^{(k)}}(L)h^{(k)}\right)$$

where φ is an activation function (applied component-wise)

See tutorial for other ways of building layers

https://distill.pub/2021/understanding-gnns/

Building layers



https://distill.pub/2021/understanding-gnns/. Another tutorial: https://tkipf.github.io/graph-convolutional-networks/

Building layers

- In this example, the first layer gives three “feature maps”

$$h_1 = \mathrm{relu}(p_{w_1}(L)x)$$
$$h_2 = \mathrm{relu}(p_{w_2}(L)x)$$
$$h_3 = \mathrm{relu}(p_{w_3}(L)x)$$

- The final output is then computed as

$$y = \mathrm{relu}(\beta^T h)$$

- The hidden states h_i and output y are vectors, with a component for each node in the graph.

Summary: Graph neural nets

- Certain data have natural graphical structure
- GNNs are analogues of CNNs for graphs
- Based on use of graph Laplacian
- Independent of ordering of nodes (equivariant)
- This is just a quick intro to an interesting current ML topic

Next topic: Reinforcement learning

10	Oct 30, Nov 1	Deep reinforcement learning	Q-learning demo DQN demo	Mon: Reinforcement learning Wed: Deep reinforcement learning	Sutton and Barto, Section 6.5	Nov 1: Assn 3 in Assn 4 out
11	Nov 6, 8	Policy gradient methods	Policy gradients demo Actor-critic demo	Mon: Policy gradient methods Wed: Actor-critic methods	Sutton and Barto, Section 13.1-13.3, 13.5	Quiz 4

Reinforcement learning

- An agent interacts with an environment
- The actions the agent takes change the state of the environment
- The agent receives rewards for each action, and seeks to maximize the total cumulative reward

Reinforcement learning is a framework for sequential decision making to achieve a long-term goal.

Reinforcement learning: Motivating examples

- Learning to walk or ride a bike
- A robot vacuum cleaning up the house
- Playing chess, backgammon, Atari games, etc.
- Drug discovery, personalized health, energy management

Article

Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

Alhussein Fawzi[1,2]✉, Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4 × 4

https://www.deepmind.com/blog/discovering-novel-algorithms-with-alphatensor

Reinforcement learning: Formalization

- The environment is in state s at a given time
- The agent takes action a
- The environment transitions to state $s' = \text{next}(s, a)$
- The agent receives reward $r = \text{reward}(s, a)$

Reinforcement learning: Formalization

- The environment is in state s at a given time
- The agent takes action a
- The environment transitions to state $s' = \text{next}(s, a)$
- The agent receives reward $r = \text{reward}(s, a)$

This is said to be a *Markov decision process*. It's "Markov" because the next state only depends on the current state and the action selected. It's a "decision process" because the agent is making choices of actions in a sequential manner.

Characteristics

- RL is inherently sequential
- In between supervised and unsupervised learning
- Agent can't act too greedily; needs to be strategic

The aim of RL is to learn to make optimal decisions from experience

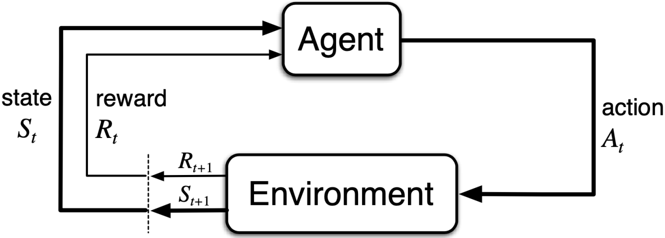
Principles of RL

Some key RL concepts and principles:

Policy, reward signal, value function, model, Bellman equation

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Principles of RL



Rewards and state transitions are probabilistic, in general

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Principles of RL

Policy: A mapping from states to actions. An algorithm/rule to make decisions at each time step, designed to maximize the long term reward.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Principles of RL

Reward signal: The sequence of rewards received at each time step. An abstraction of “pleasure” (positive reward) and “pain” (negative reward) in animal behavior.

“Reinforcement Learning: An Introduction” (Second Edition), Richard S. Sutton and Andrew G. Barto

Principles of RL

Value function: A mapping from states to total reward. The total reward the agent can expect to accumulate in the future, starting from that state.

Rewards are short term. Values are expectations of future rewards.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Principles of RL

Model: Used for planning to mimic the behavior of the environment, to predict rewards and next states.

A *model-free* approach directly estimates a value function, without modeling the environment.

Analogous to distinction between generative and discriminative classification models

---

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Taxi problem

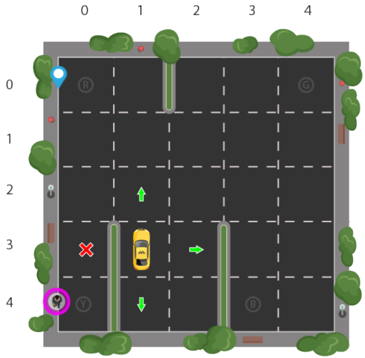
We'll introduce the important Q-learning algorithm with the toy "Taxi problem"

The code uses OpenAI gym

Our presentation follows the tutorial
https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

Taxi problem

A taxicab drives around the environment, picking up and delivering a passenger at four locations



Taxi problem

A taxicab drives around the environment, picking up and delivering a passenger at four locations

”Ascii art” rendition:

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Taxi problem: Description

- Four designated locations: R(ed), G(reen), Y(ellow), and B(lue)
- Taxi starts off at random square and passenger is at random location
- Taxi drives to passenger's location, picks up the passenger, drives to passenger's destination, drops off passenger
- Once the passenger is dropped off, the episode ends.

Taxi problem: State space

- 25 taxi positions
- 5 possible locations of passenger: At waiting location or in taxi
- 4 possible destination locations
- Total number of states: $25 \times 5 \times 4 = 500$

Taxi problem: State space

Passenger location coded as integers:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)
- 4: in taxi

Taxi problem: State space

Destinations coded as:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)

Taxi problem: State space

Agent actions coded as:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Taxi problem: State space

Rewards:

- Default reward per step: -1
- Reward for delivering passenger: +20
- Illegal “pickup” or “drop-off”: -10

Taxi problem: State space

State space represented as a tuple:

state = (taxi row, taxi column, passenger location, destination)

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action *a* in state *s*
- A measure of the cumulative rewards obtained by the algorithm when it takes action *a* in state *s*

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s
- Quality should not be assessed purely based on the reward the action has in the current time step

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s
- Quality should not be assessed purely based on the reward the action has in the current time step
- Need to take into account the future rewards

Q-learning update

Update:

$$Q(s,a) \longleftarrow$$
$$Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

Q-learning update

Update:

$$Q(s,a) \longleftarrow$$
$$Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

- When action a is taken in state s, reward reward(s,a) is given
- Then, the algorithm moves to a new state next(s,a)

Q-learning update

Update:

$$Q(s,a) \longleftarrow$$
$$Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

- For example, if the taxi is location $(2,2)$ and takes the "West" action ($a = 3$), then there is a reward of -1, and the taxi moves to the new location $(2,1)$
- If cab is empty, it remains empty, and if it contains the passenger, the passenger remains.

Q-learning update

Update:

$$Q(s,a) \longleftarrow$$
$$Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

- Cumulative future reward of this action is $\max_{a'} Q(\text{next}(s,a), a')$
- Future rewards discounted by factor $\gamma < 1$
- Trades off short-term against long-term rewards
- A gradient *ascent* algorithm, with step size α

Let's go to the notebook!

Bellman equation

The optimal value function is the largest expected discounted long term reward starting from that state.

---

https://en.wikipedia.org/wiki/Richard_E._Bellman

Bellman equation

Curse of dimensionality [edit]

Main article: Curse of dimensionality

The *curse of dimensionality* is an expression coined by Bellman to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space. One implication of the curse of dimensionality is that some methods for numerical solution of the Bellman equation require vastly more computer time when there are more state variables in the value function. For example, 100 evenly spaced sample points suffice to sample a unit interval with no more than 0.01 distance between points; an equivalent sampling of a 10-dimensional unit hypercube with a lattice with a spacing of 0.01 between adjacent points would require 10^{20} sample points: thus, in some sense, the 10-dimensional hypercube can be said to be a factor of 10^{18} "larger" than the unit interval. (Adapted from an example by R. E. Bellman, see below.) [15]

https://en.wikipedia.org/wiki/Richard_E._Bellman

Bellman equation: Deterministic case

The optimality condition for the value function v_* is

$$v_*(s) = \max_a \Big\{ \text{reward}(s, a) + \gamma v_*(\text{next}(s, a)) \Big\}$$

Bellman equation: Deterministic case

The optimality condition for the Q-function is

$$Q_*(s, a) = \text{reward}(s, a) + \gamma \max_{a'} Q_*(\text{next}(s, a), a')$$

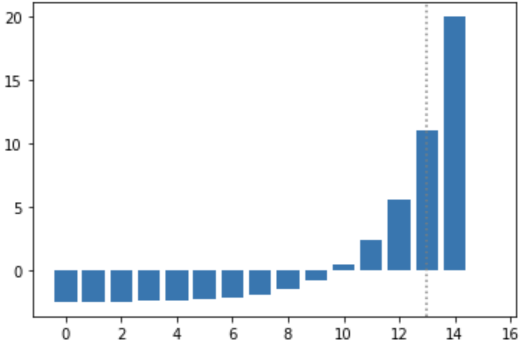
and then $v_*(s) = \max_{a'} Q_*(s, a')$

Q-learning update

Note how this makes sense in terms of the update rule:

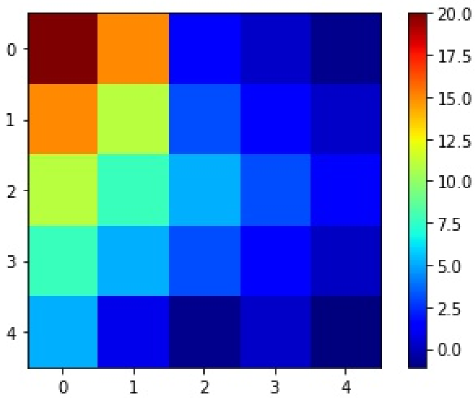
$$Q(s,a) \longleftarrow$$
$$Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
```



Question

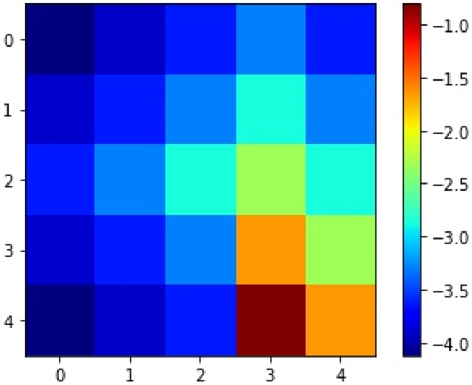
For a fixed passenger location and destination, the value function $v(\mathtt{row}, \mathtt{col})$ assigns a value to each of the $25 = 5 \times 5$ grid points.



Is the passenger waiting, or in the taxi?

Question

For a fixed passenger location and destination, the value function $v(\texttt{row}, \texttt{col})$ assigns a value to each of the $25 = 5 \times 5$ grid points.



Is the passenger waiting, or in the taxi?

Summary

- Reinforcement learning is a framework for sequential decision making to achieve a long-term goal
- The agent receives rewards for each action, and seeks to maximize the total cumulative reward
- The value of a state is the total reward the agent can expect to accumulate in the future, starting from that state
- Q-learning is an iterative algorithm that estimates the value of each state-action pair
- The Bellman equations are optimality conditions that characterize the value function