清華大學
Tsinghua University

Deep Reinforcement Learning （**Ultra!**）

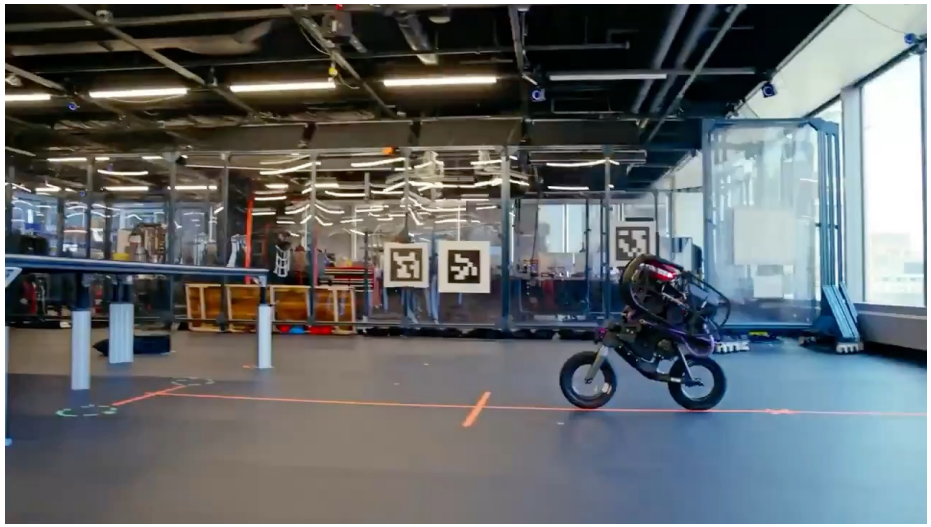
Lecture 2: Value Estimation from MDP

Huazhe Xu

Tsinghua University

HW1 will be released this week!
Recitation Tues 3:20-4:05, everyweek

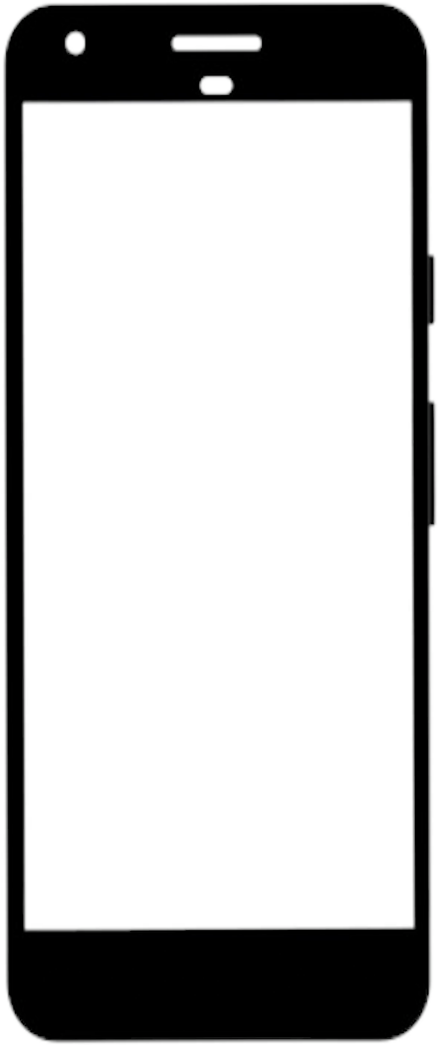
AI This Week



Ultra Mobile Vehicle from RAI

Digi-Q

Digi-Q https://digirl-agent.github.io/agent.github.io/

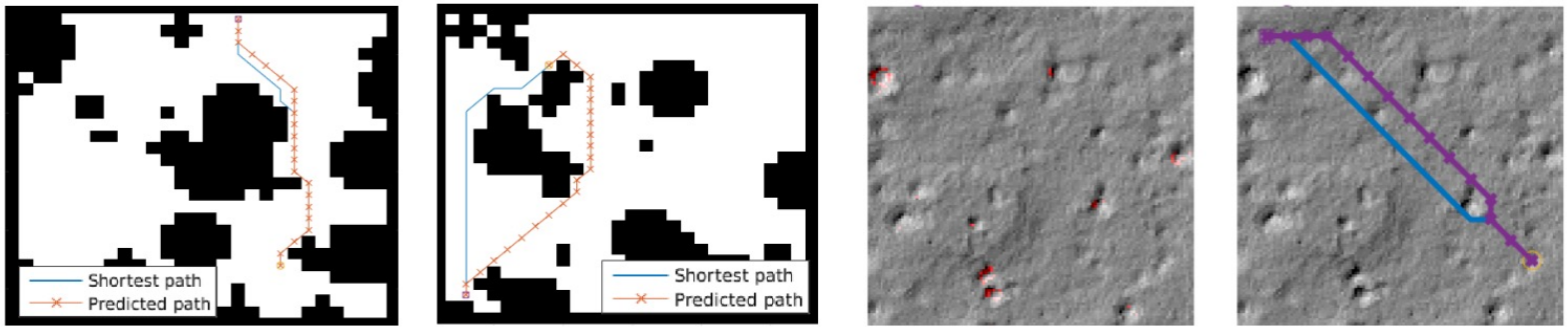


Bellman Alert!

You all must learn this!

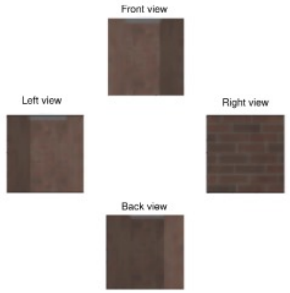
Wait, wait, wait!

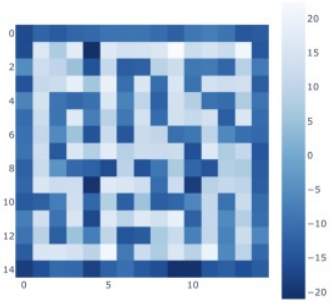
- Conventional wisdom: usually the content in Lecture 1 is useless, ancient stuff (e.g., value iteration), later you will learn cool stuff (RL).
- But it does not apply in this class…
- Value Iteration Networks, NeurIPS 2016 Best Paper, A. Tamar, Y. Wu, G. Thomas, S. Levine, P. Abbeel
  - Intuition: convolution operator is very similar to value iteration… can we



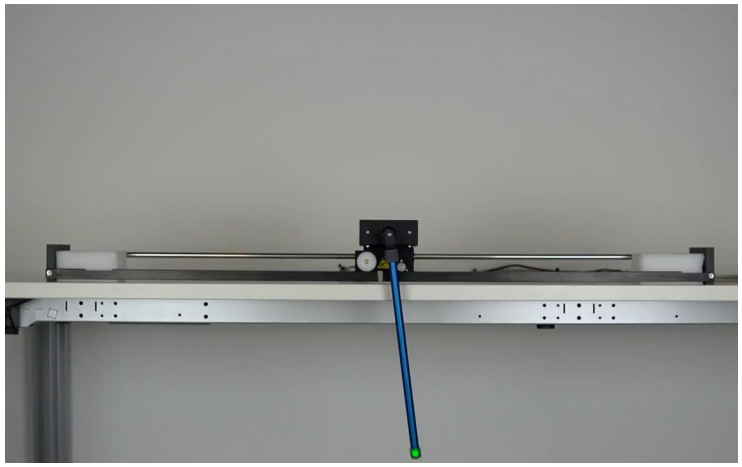
Value iteration is still something actively studied!

- *Scaling up and Stabilizing Differentiable Planning with Implicit Differentiation, L. Zhao, H. Xu, L. Wong, ICLR'23*





- Value Iteration in Continuous Actions, States and Time, M. Lutter, S. Mannor, J. Peters, D. Fox, A. Garg, ICML'21



In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(\lambda)
4. From Estimation to Policy

What's our first goal?

Estimate the value function of an unknown MDP.

Estimate from Experiences…

Model-based: GMM/R^2

But maybe the model is to hard to learn…

Model-free, learn from experience: Let it drop for a thousand times, and we learn to dodge the apple.

In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(λ)
4. From Estimation to Policy

Monte-Carlo Policy Evaluation (Direct Evaluation)

- Goal: learn V^{π} from episodes of experience under policy π

$$s_1, a_1, r_1, \ldots, s_k \sim \pi$$

- Return is discounted reward-to-go:

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_T$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s]$$

- MC policy evaluation uses empirical mean of return instead of expectation.

Monte-Carlo Policy Evaluation (Direct Evaluation)

- To evaluate state s every time-step t that state s is visited in an episode
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + Gt$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Is MC policy evaluation good or bad?

- Good, why?
  - Only from experience
  - Simple
  - No bootstrapping

- Bad, why?
  - All episodes must run until termination
  - Require a lot of samples
  - Does not exploit bellman equation (Markov property)

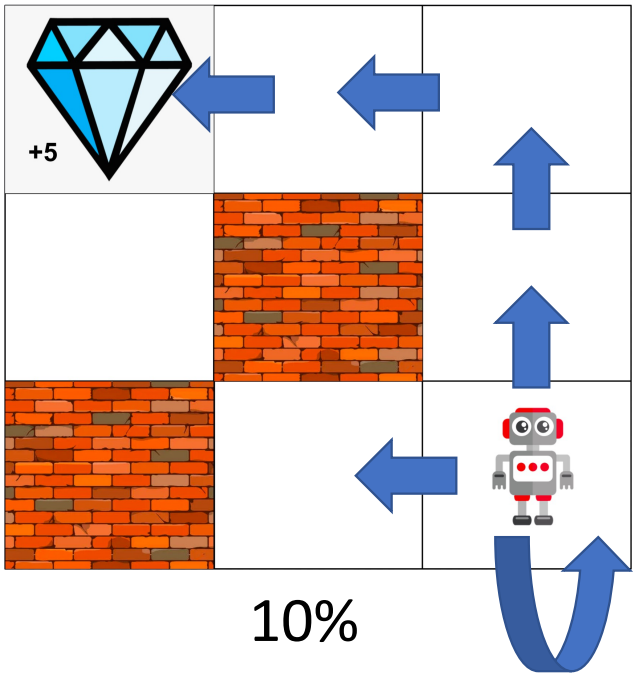
You've already got the MC estimation.

- But a new sample comes
- Incremental mean:

$$
\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}
$$

$$
\begin{aligned}
N(S_t) &\leftarrow N(S_t) + 1 \\
V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))
\end{aligned}
$$

MC will ultimately find the correct value.



80% -> appears in 9 samples

10% ->appears in 1 sample

10% -> appears in 2 samples

In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(λ)
4. From Estimation to Policy

Temporal Difference Learning

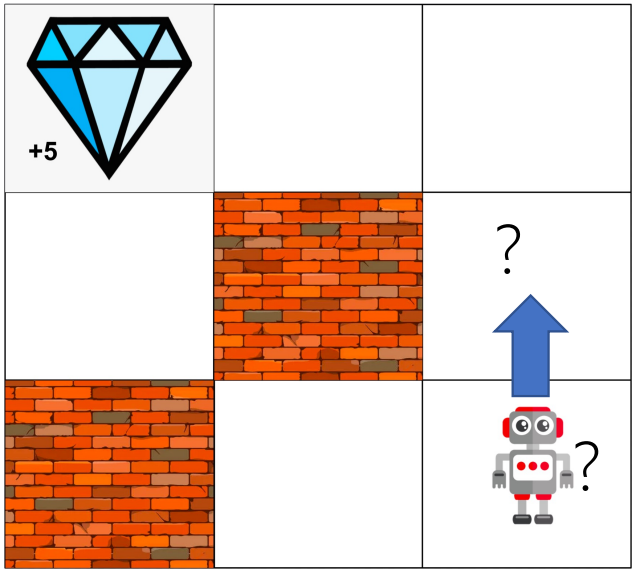
- learn V^{π} online from experience under policy π
- Local updates of value on a per-action basis
- Simplest algorithm: TD(0)

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- Drag the value toward what bellman equation requires.
- $R_{t+1} + \gamma V(S_{t+1})$ is called TD target.
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called TD error.
- α is the learning rate.

MC vs. TD

- A guess from a guess



Which is better?

- TD:
  - Learn from every step
  - Learn from incomplete sequence
- MC:
  - Does not require bootstrap
  - Seemingly more “correct”?

Bias vs. Variance

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is **unbiased** estimate of $\mathrm{V}^{\pi}(s_t)$
- True TD target $R_{t+1} + \gamma V^{\pi}(S_{t+1})$ is **unbiased** estimate of $\mathrm{V}^{\pi}(s_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is **biased** estimate of $\mathrm{V}^{\pi}(s_t)$
- How about variance?
  - TD has lower variance than MC.
  - MC depends on many random things. TD target depends on one random action, transition, reward.

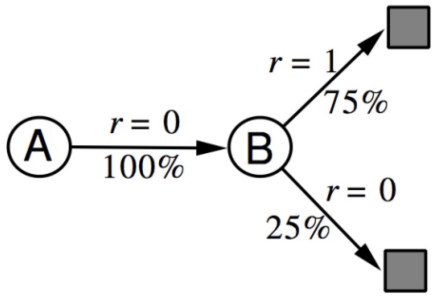
What does “seemingly more correct” mean?

- MC has high variance, zero bias
  - Good convergence properties
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to V^{π}(s) (but not always with function approximation)
  - More sensitive to initial value

If you have infinite data and compute, you are happy with both methods! But…

But what if you only have a batch?

- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0



MC believers
V(A)=?

TD believers
V(A)=?

Credit: David Silver

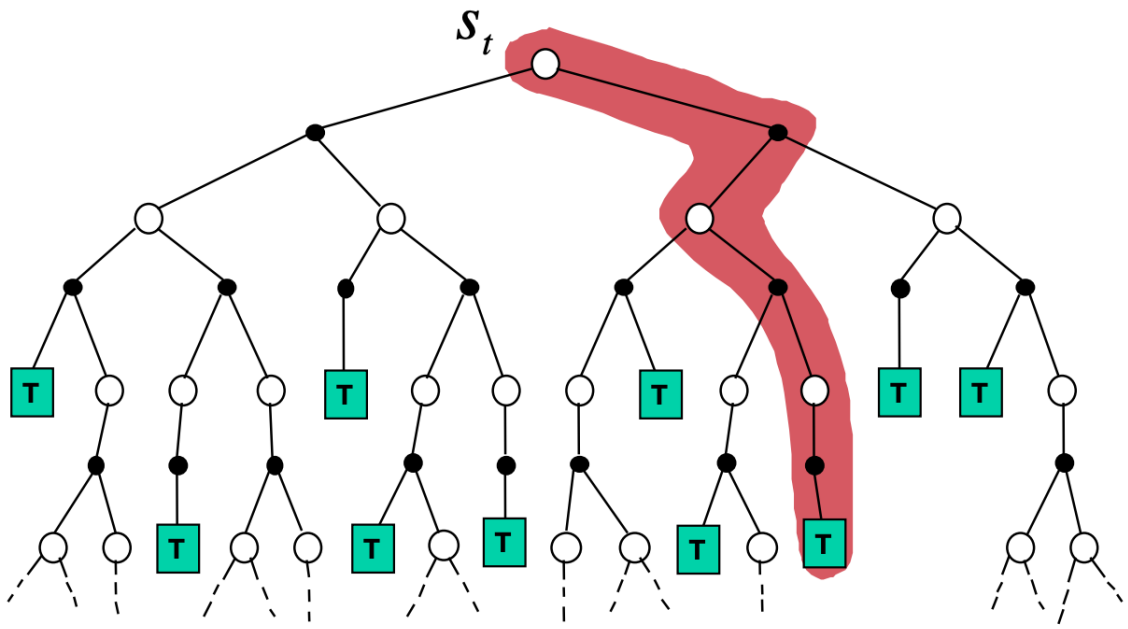
- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - V\left(s_t^k\right) \right)^2$$

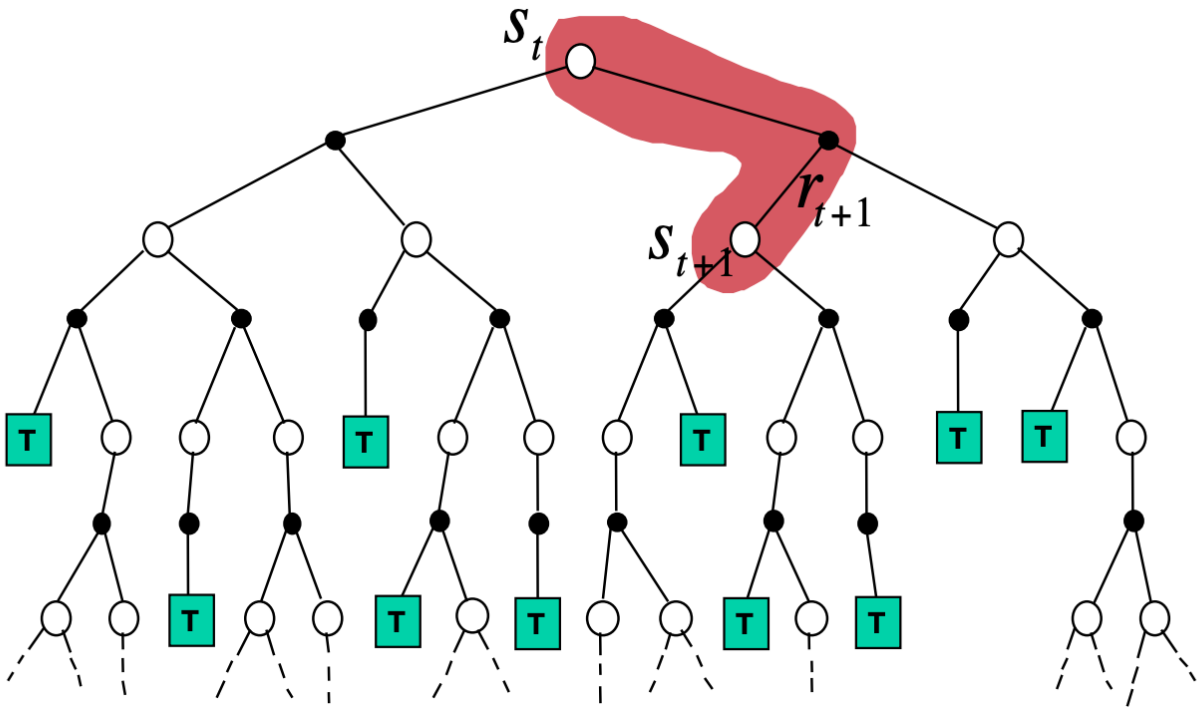
  - In the AB example, V(A) = 0
- TD converges to solution of max likelihood Markov model
  - Solution to the MDP that best fits the data
  - In the AB example, V(A) = 0.75

MC Visualized

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



TD Visualized



Value Iteration Visualized

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$

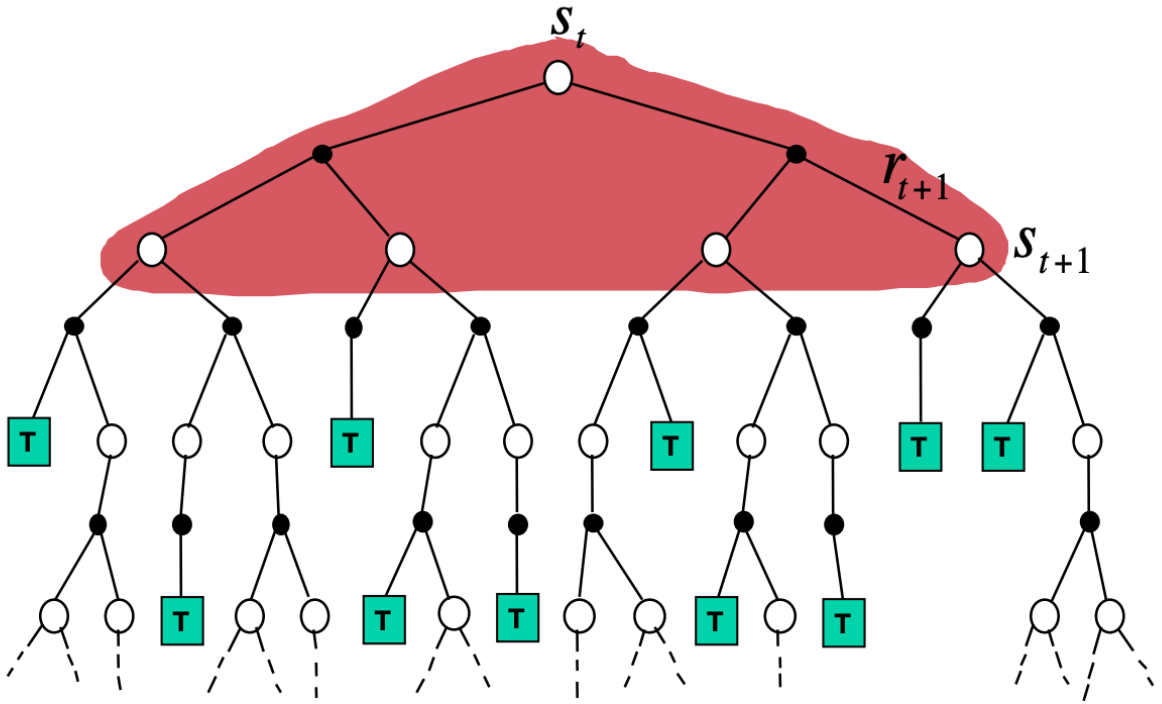
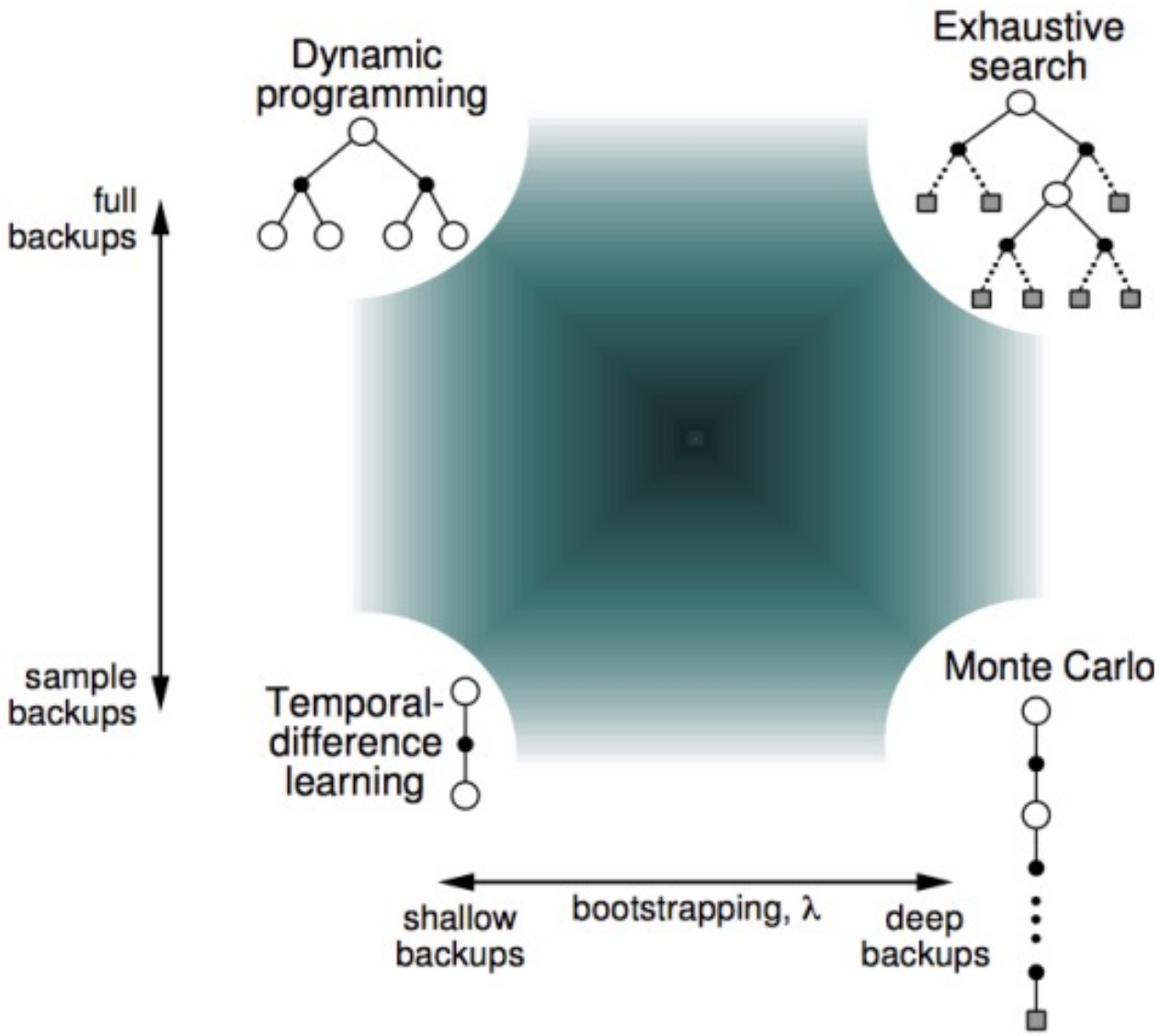
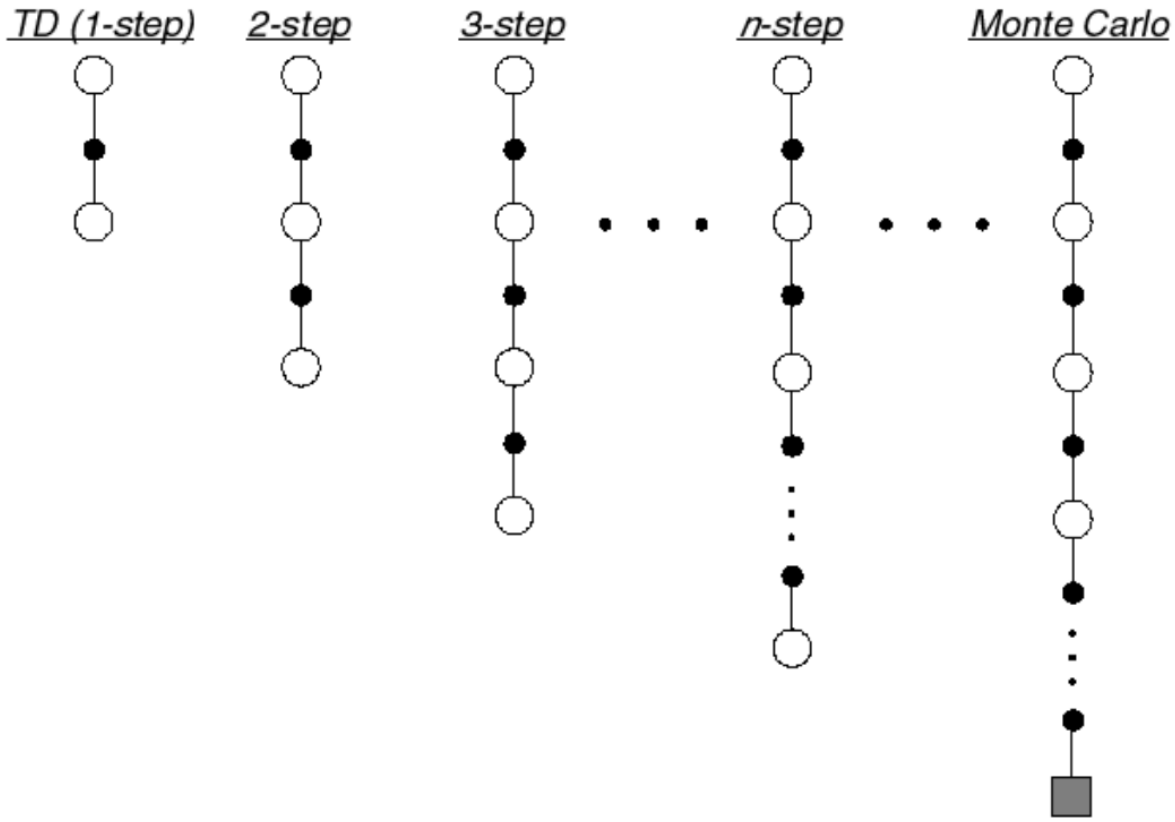


Photo Credit: David Silver RL Lectures


Dynamic programming
Exhaustive search
full backups
sample backups
Temporal-difference learning
Monte Carlo
shallow backups
bootstrapping, λ
deep backups

n-Step Return



n-Step Return

$$
\begin{aligned}
n = 1 \quad (TD) \qquad & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 \qquad & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots \qquad & \vdots \\
n = \infty \quad (MC) \quad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{aligned}
$$

- n-Step Return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-Step TD:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(G_t^{(n)} - V(S_t)\Big)$$

In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(λ)
4. From Estimation to Policy

It is so hard to decide when to use which!

- If the initialization is too bad … too much bias!
  - I might want to do MC
- If value is already somewhat good
  - I'd prefer TD because it has less variance
- Can we combine the best of both worlds?

λ-Return & TD(λ)

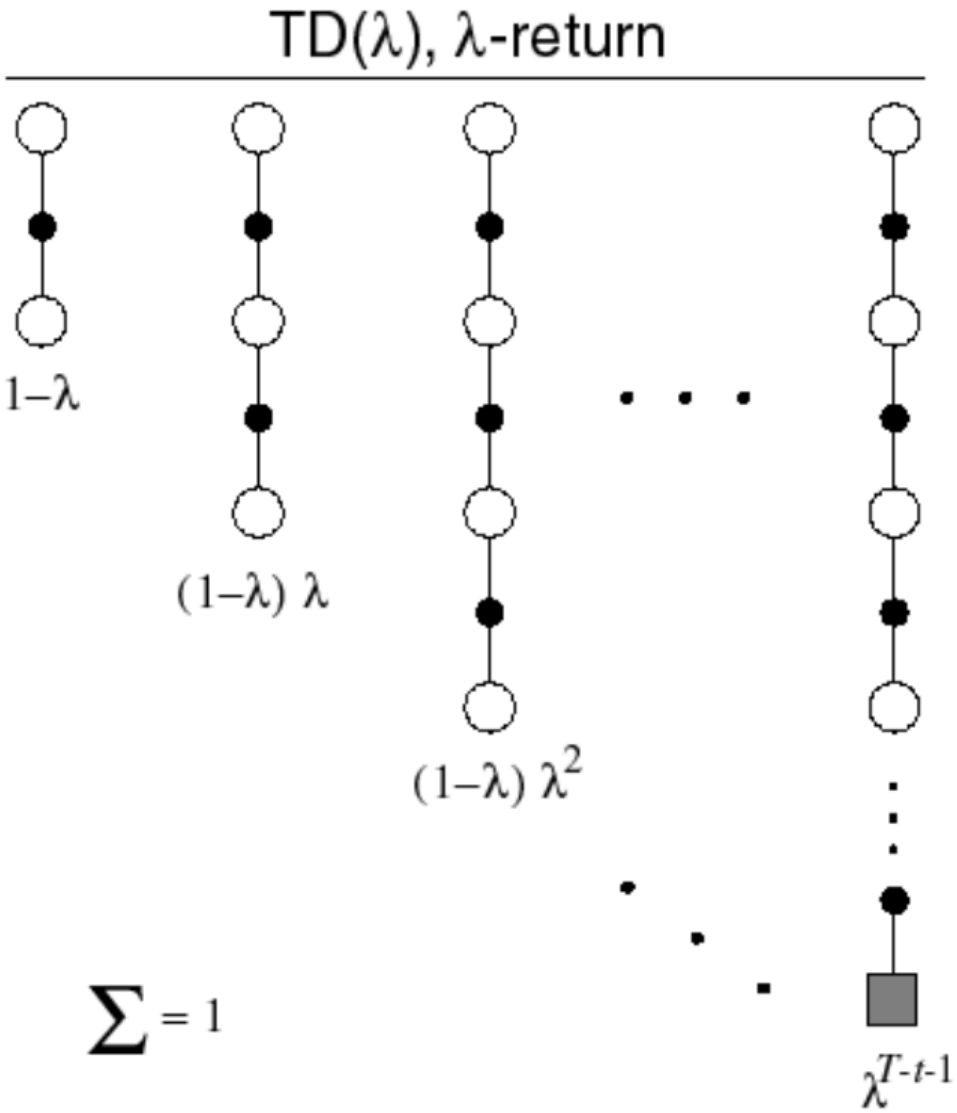
The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$
Using weight $(1-\lambda)\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

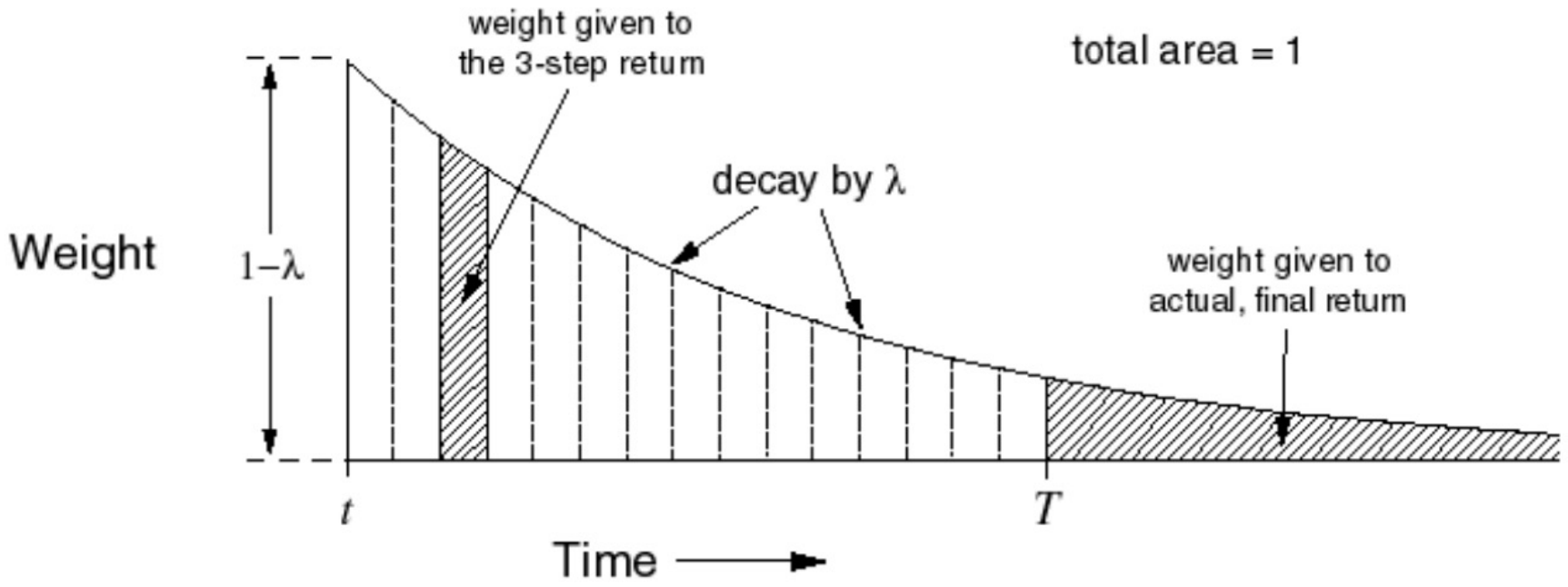
Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t^\lambda - V(S_t)\big)$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1}R_{t+n} + \gamma^n V(S_{t+n})$$



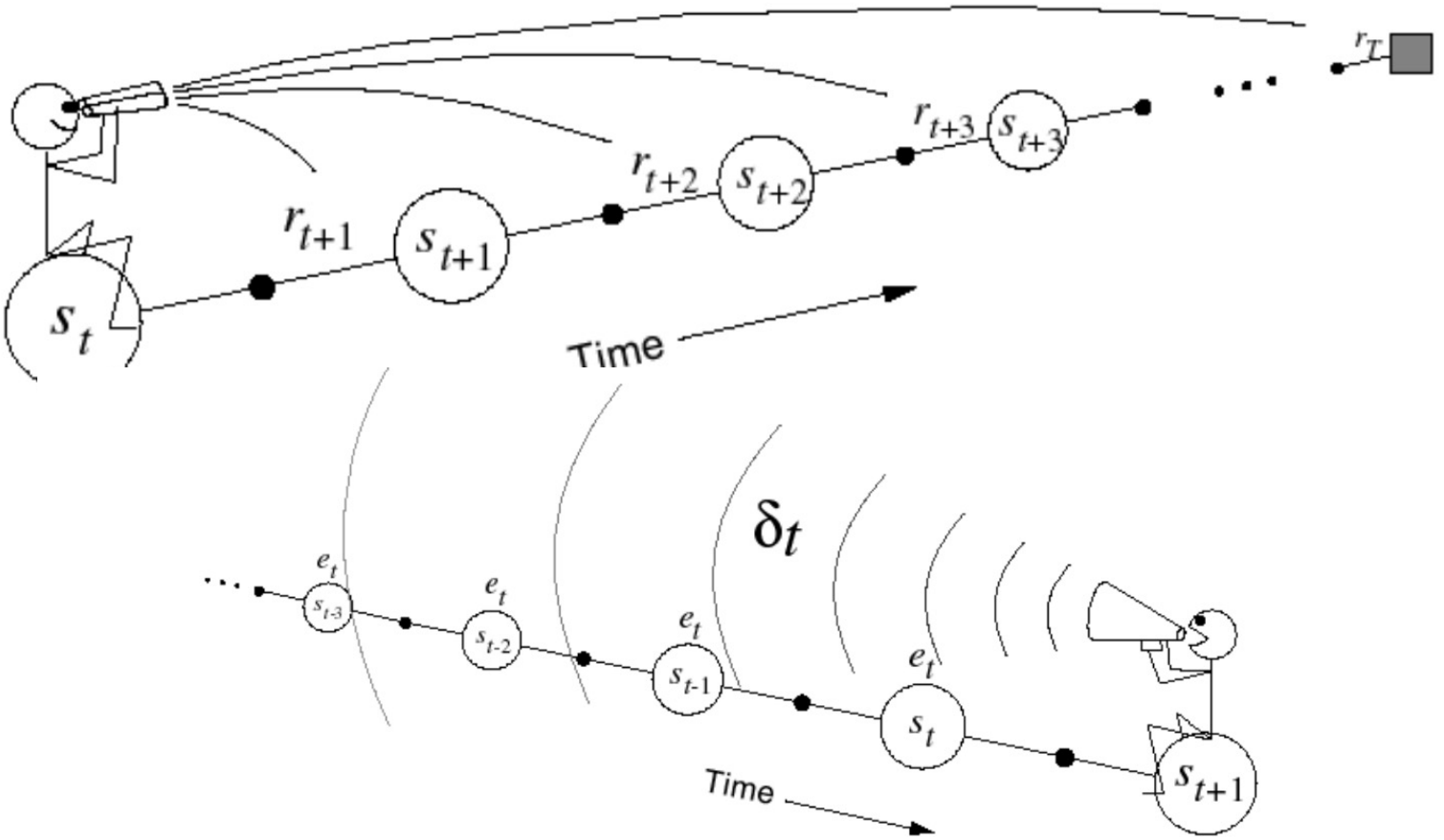
Credit: Richard Sutton


weight given to
the 3-step return
total area = 1
decay by λ
Weight
1−λ
weight given to
actual, final return
t
T
Time

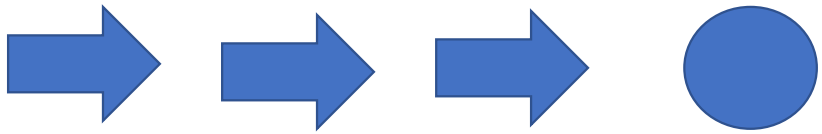
Wait, there is something wrong with TD(λ)

- We prefer TD because we want to update with incomplete sequences.
- But you drag us back to offline, episodic updates --- you need to wait for the whole trajectory/episode to finish.
- How would you solve this?

Let’s look backward! A nifty idea ☺



Eligibility Traces



- What makes Mario meet the princess?
- Frequency heuristics
- Recency heuristics

Eligibility traces combine both heuristics

$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

Backward View TD(λ)

- Every past state has an influence on the current TD error —— Eligibility Trace!
- Then we propagate the TD error now to all the past states!

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

Backward View TD(λ)

- When λ = 0, only current state is updated

$$E_t(s) = \mathbf{1}(S_t = s)$$
$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- This is exactly equivalent to TD(0) update

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

- When $\lambda = 1$, credit is deferred until end of episode
- Over the course of an episode, total update for TD(1) is the same as total update for MC

They actually don’t look the same… I don’t understand.

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \big(G_t^\lambda - V(S_t)\big) \mathbf{1}(S_t = s)$$

They actually don't look the same… I don't understand.

- In offline update, forward == backward
- Consider this case: s is visited at time k

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$
$$= \begin{cases} 0 & \text{if } t < k \\ (\gamma\lambda)^{t-k} & \text{if } t \geq k \end{cases}$$

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \alpha \sum_{t=k}^{T} (\gamma\lambda)^{t-k}\delta_t = \alpha\big(G_k^\lambda - V(S_k)\big)$$

For general λ, TD errors also telescope to λ-error, $G_t^\lambda - V(S_t)$

$$
\begin{aligned}
G_t^\lambda - V(S_t) &= -V(S_t) + (1-\lambda)\lambda^0(R_{t+1} + \gamma V(S_{t+1})) \\
&\quad + (1-\lambda)\lambda^1\big(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})\big) \\
&\quad + (1-\lambda)\lambda^2\big(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})\big) \\
&\quad + \dots \\
&= -V(S_t) + (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})) \\
&\quad + (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})) \\
&\quad + (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) \\
&\quad + \dots \\
&= (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \\
&\quad + (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1})) \\
&\quad + (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2})) \\
&\quad + \dots \\
&= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots
\end{aligned}
$$

After one whole episode, forward == backward

- But in the middle, we don’t know..

In online update, forward != backward

- They are slightly different
- But exact online forward == backward
- True Online TD(λ), ICML'14, H. Seijen, R. Sutton, they use a different Eligibility Trace ☺
- If you are interested, read it.

The idea of TD(λ) is useful!

- High-Dimensional Continuous Control Using Generalized Advantage Estimation, J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, ICLR'16
- Proximal Policy Optimization Algorithms, J. Schulman et al, never published.. But it is the most influential DRL paper!

$$
\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots\right) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \dots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \lambda^3 + \dots) \\
&\qquad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \lambda^4 + \dots) + \dots) \\
&= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \dots\right) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \qquad (16)
\end{aligned}
$$

In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(\lambda)
4. From Estimation to Policy: Q-Learning & SARSA

So far…

- We have MC, TD, TD(λ) to estimate values.
- If we can find a good policy to follow high value states, then the job is done.
- You can now play games, karaoke, watch bilibili because everything is solved!

But can we apply policy improvement?

$$\arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

But can you?

- Who might help you to get the policy?
  - Q, why?

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') V(s')$$
$$= R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Q is given. Is this complete?

- Active Reinforcement Learning
- Zijing Yuan: +1 for 100 times; Yu Yuan, +0 for 1 time.
- Q(ZJY) > Q(YY), is it wise to always choose ZJY?

Exploration vs. Exploitation

- Exploitation: To try to get reward. We exploit our current knowledge to get a payoff.

- Exploration: Get more information about the world. How do we know if there is not a pot of gold around the corner?

- The tradeoff is important in RL and life.
  - My naïve approach: if you know enough about the world, you exploit. You know nothing, you explore.

ϵ-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a \mid s) = \mathbf{1}\left(a = \operatorname*{argmax}_{a' \in \mathcal{A}} Q_k(s, a')\right)$$

$$\text{For example, } \epsilon\text{-greedy is GLIE if } \epsilon \text{ reduces to zero at } \epsilon_k = \frac{1}{k}$$

But in practice, no one does this!

Another GLIE Policy Example: Boltzman Exploration

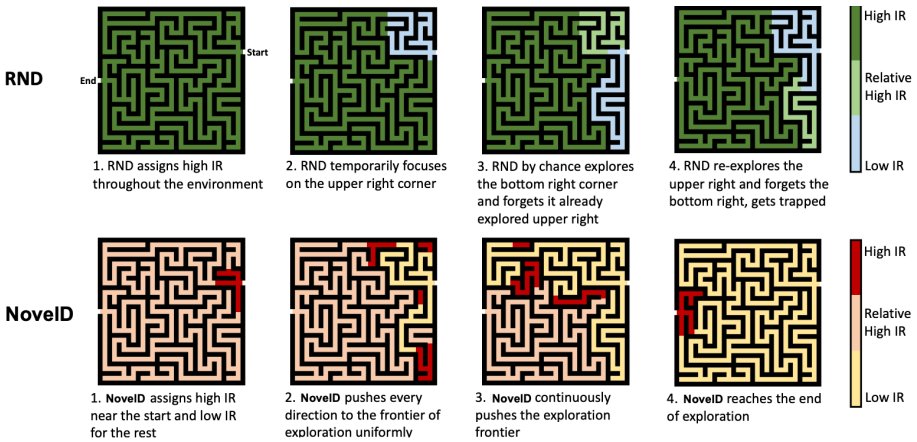
$$\Pr(a \mid s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in A} \exp(Q(s,a')/T)}$$

Exploration is an active research area!

- Before I show you cool stuff, how do you do exploration in your life?
- Curiosity
  - Curiosity-driven Exploration by Self-supervised Prediction, Pathak et al., ICML'17
  - Exploration by Random Network Distillation, Burda et al, ICLR'19

Exploration is an active research area…

- NovelD: A Simple yet Effective Exploration Criterion, Zhang et al., NeurIPS'21



Of course, there are more:

Explore when you get stuck

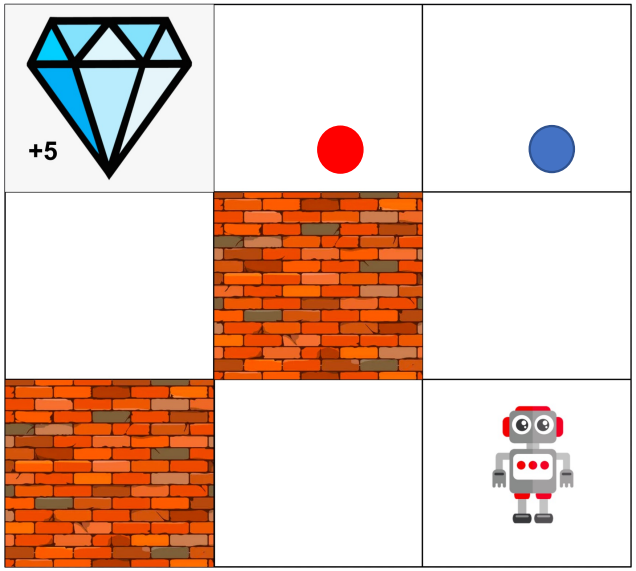
Explore if you make an impact

Explore if you influence others

Let recap here…(If you understand this slide, you are on the right track!)

- We've learned value iteration style stuff…
- But sometimes, we don't have transition functions and reward functions.
- We decide to use experience to estimate value: MC, TD
- But it seems it still requires transition function if we need the policy: maybe we should use TD to get Q value
- Even you have Q, you are still passive. You can never know the part you don't know. You need to explore.
- ϵ –greedy for policy to explore
- Ok, up to here, you almost invent Q learning together with me ☺

Recursive Definition of Value: Bellman Equation (again)!



- $V^*(s) = \max_a Q^*(s,a)$
  - V*(blue dot) = max _a {Q*(bd, N), Q*(bd, S), Q*(bd, W), Q*(bd, E)}
- $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
  - Q*(bd, W) = (0.8+0.1)(reward_if_go_W + γ V*(rd)) +

    0.1(reward_if_go_E + γ V*(bd))
- $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

Q-learning

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') V(s')$$
$$= R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

- We can perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do:

(note that we directly observe reward R(s))

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{R(s) + \gamma \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

Q-learning (formally)

- Start with initial Q-function (e.g. all zeros)
- Take action from an exploration/exploitation policy giving new state s' (should converge to greedy policy, i.e. GLIE)
- Perform TD update

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a)\right)$$

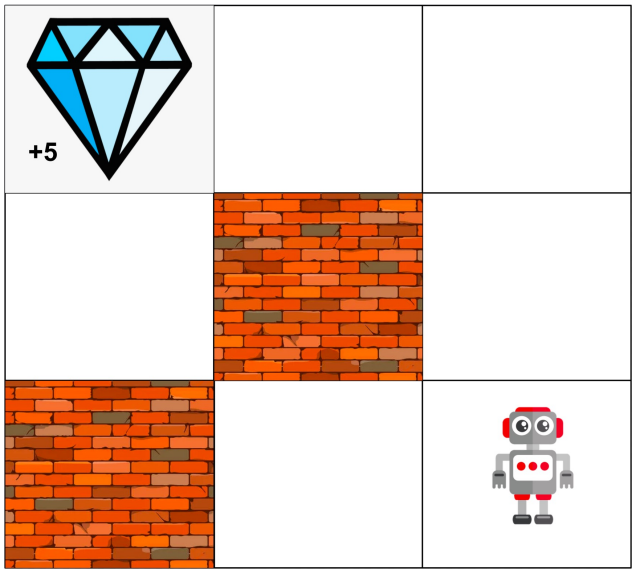
- Goto 2

Note: If an RL algorithm does not depend on the actual next action(s), it is off-policy. The greedy action is neither the current policy nor the policy that collected the sample. If it follows another behavior policy, an importance sampling term might be required.

Q-Learning with sparse reward

Trajectory Replay: store each trajectory and do several iterations of Q-updates on each one

Reverse updates: store trajectory and do Q-updates in reverse order.



State-Action-Reward-State-Action (SARSA)

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n[r(s_n, a_n) + \lambda Q_n(s_{n+1}, a_{n+1}) - Q_n(s_n, a_n)]$$

on-policy: The Q-value update depends on the actual next action

In Lec2

1. Model-free Estimation: Monte-Carlo Learning
2. Model-free Estimation: Temporal Difference Learning
3. Discussion: TD(\lambda)
4. From Estimation to Policy: Q-Learning & SARSA

Next, we will scale up…

- Value Approximation for RL so that we can deal with real problems

Deep Reinforcement Learning

Recitation: PyTorch & Demos

Huazhe Xu

Tsinghua University

Credits to Kaizhe Hu & Ray Zheng

Contents

- What is PyTorch and Why?
- Setup and Installation
- Tensors
- Datasets
- Models
- Optimizers
- Checkpoints
- All Together

What is PyTorch?

PyTorch

- ML & DL open-source framework
- Originated from Facebook (Meta)
- Now governed by PyTorch Foundation
- Competitors: Tensorflow (Google), MXNet (Amazon), …
- Why?
  - Easy to code (readability)
  - Version compatibility
  - Widely used in academia

Setup and Installation

https://pytorch.org/get-started/locally/

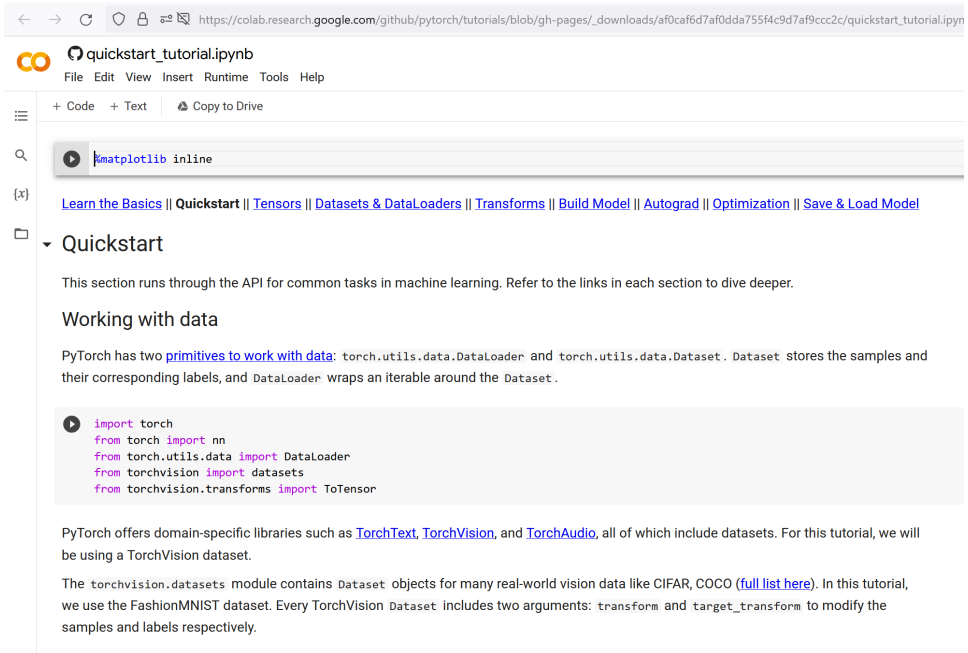
START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	Stable (1.13.1)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 11.6	CUDA 11.7	ROCm 5.2	CPU
Run this Command:	`conda install pytorch torchvision torchaudio pytorch-cuda=11.6 -c pytorch -c nvidia`			

Setup and Installation

- **Linux** recommended
- Virtual Machine
- Windows Subsystem for Linux (WSL)
- Colab might be helpful (https://colab.research.google.com)



Where this tutorial came from

https://pytorch.org/tutorials/beginner/basics/intro.html

Introduction to PyTorch [-]

Learn the Basics

Quickstart

Tensors

Datasets & DataLoaders

Transforms

Build the Neural Network

Automatic Differentiation with `torch.autograd`

Optimizing Model Parameters

Save and Load the Model

Tensors

- High-dimensional matrices
- Accelerated operations
- E.g., an image (3 * 100 * 100)

- Shape
- Data type (`dtype`)

- Can convert to & from `numpy`

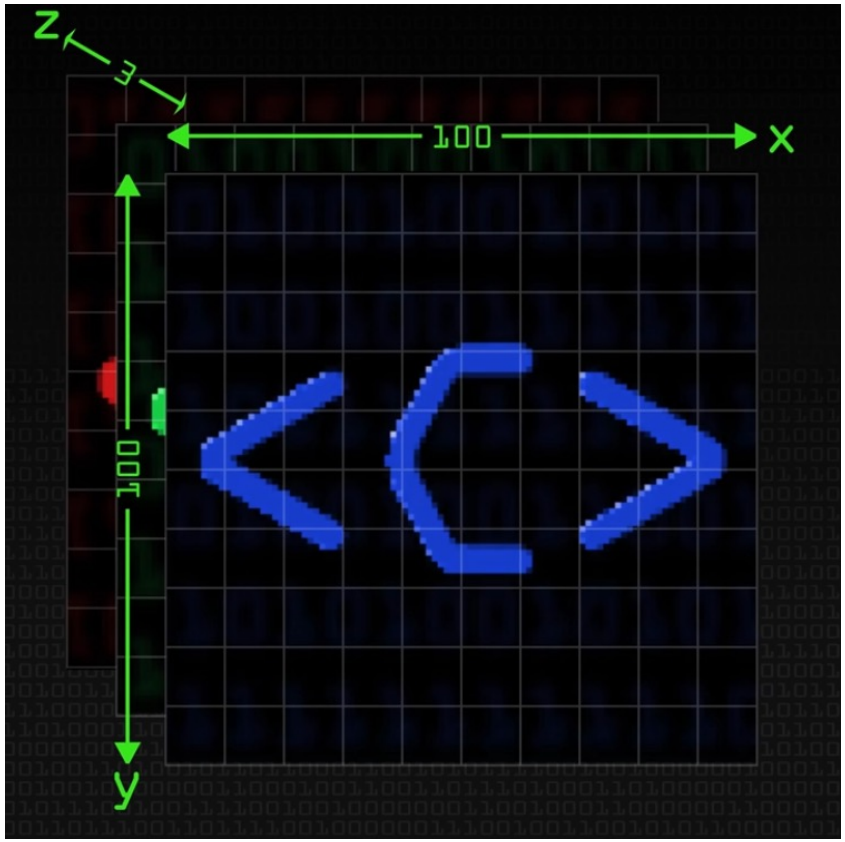


Image source: https://www.youtube.com/watch?v=DfK83xEtJ_k

Tensors are fast!

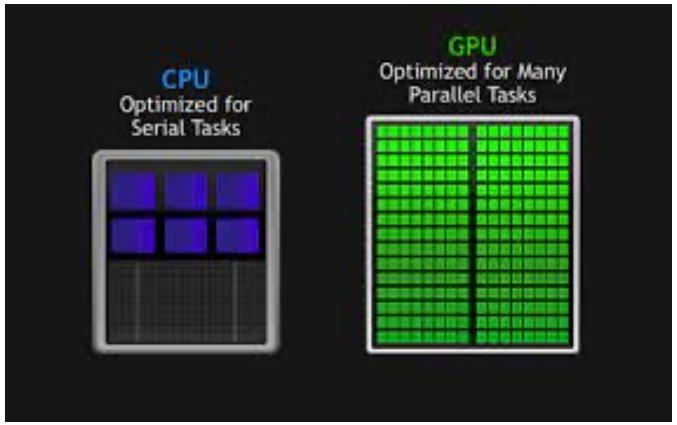


Image source: https://www.analyticsvidhya.com/blog/2021/10/leveraging-pytorch-to-speed-up-deep-learning-with-gpus/

Time in CPU

```
>>> import time
>>> x=torch.rand(10000,10000)
>>> y=torch.rand(10000,10000)

>>> t = time.time()
>>> z=x@y
>>> t = time.time()-t
>>> print(t)

6.999474763870239
```

Time in GPU

```
>>> yc=y.cuda(0)
>>> t = time.time()
>>> z=xc@yc
>>> t = time.time()-t
>>> print(t)

0.4787747859954834
```

It is nearly **15 times faster** than Numpy for simple matrix multiplication!

Image source: https://towardsdatascience.com/what-is-pytorch-a84e4559f0e3

Datasets and Dataloaders

- Can be built in or custom
- `Dataset` is a class to inherit
  - `__init__`, `__len__`, `__getitem__`, …
- A `Dataloader` loads data from a `Datase`
  - Sample method, batch size, …

```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
```

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_tran
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Models

- Built in models
  - `resnet18, transformer, …`
- Inherits `nn.Module`
  - `forward`
- Library of model layers from `torch.nn`

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

Optimization

- Optimizer
  - `optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)`
  - Algorithms: `Adam, RMSprop, SGD`, …

- Loss function
  - Built-in: MSE loss, Cross Entropy Loss, …
  - Can write custom loss

Checkpoints

- Save model or model weights
- Save best training results
- Load for evaluation/testing

```
model = models.vgg16(pretrained=True)
torch.save(model.state_dict(), 'model_weights.pth')
```

To load model weights, you need to create an instance of the same method.

```
[ ] model = models.vgg16() # we do not specify pretrained=Tr
    model.load_state_dict(torch.load('model_weights.pth'))
    model.eval()
```

```
[ ] torch.save(model, 'model.pth')
```

We can then load the model like this:

```
[ ] model = torch.load('model.pth')
```

All Together

https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/af0caf6d7af0dda755f4c9d7af9ccc2c/quickstart_tutorial.ipynb