CS4910: Deep Learning for Robotics

David Klee
klee.d@northeastern.edu

T/F, 3:25-5:05pm
Behrakis Room 204

https://www.ccs.neu.edu/home/dmklee/cs4910_s22/index.html

https://piazza.com/northeastern/spring2022/cs4910a/home

Reinforcement Learning: Q-Learning

Some slide material taken from CS7180 F18
For more details see: Sutton & Barto 2018

Today’s Agenda

1. Deep Q-Network
2. IK from scratch
3. HW3

Reinforcement Learning (RL) is learning through trial-and-error *without a model of the world*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)}_{\texttt{q\_target}} - \underbrace{Q(S_t, A_t)}_{\texttt{q\_pred}} \Big]$$

$$\texttt{td-error}$$

$$Q(s, a) = f_\theta(s, a)$$

$$\mathcal{L} = \|\texttt{q\_target} - \texttt{q\_pred}\|$$

Extending to Deep Q-learning...

1. Q-function will be implemented as neural network (**Q-network**)

2. We want to use batch gradient descent. However, consecutive transitions from the environment will be tightly correlated and will bias the gradient.

3. We rely on the Q-network for predicting q_target and q_pred, which introduces instability sa the network updates weights

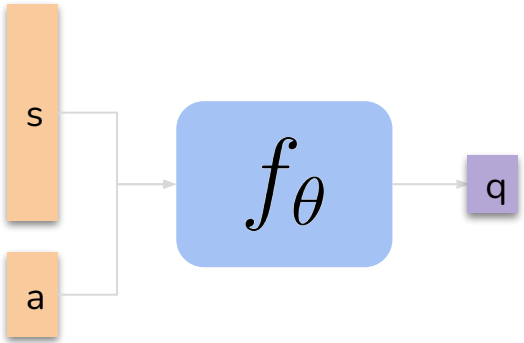
Deep Q-Network (DQN) implementations

State as input

$$\vec{q} = f_\theta(s)$$

State & Action as input

$$q = f_\theta(s, a)$$



Consider how to calculate max Q(s,a)...

Q-function as neural network (Deep Q-Network)

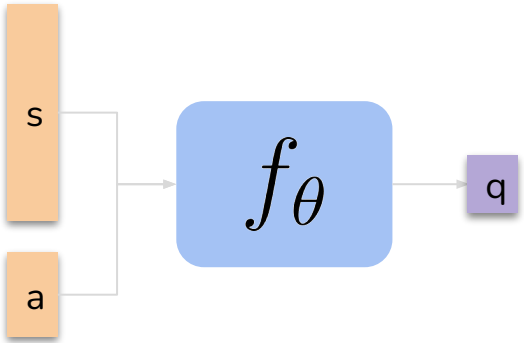
State as input

$$\vec{q} = f_\theta(s)$$

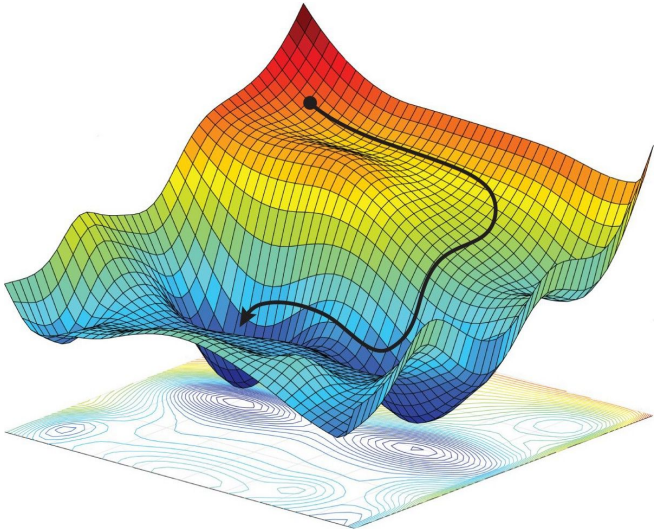
Outputting q-values for all actions associated with a state is the common practice, as it simple to find the maxQ

State & Action as input

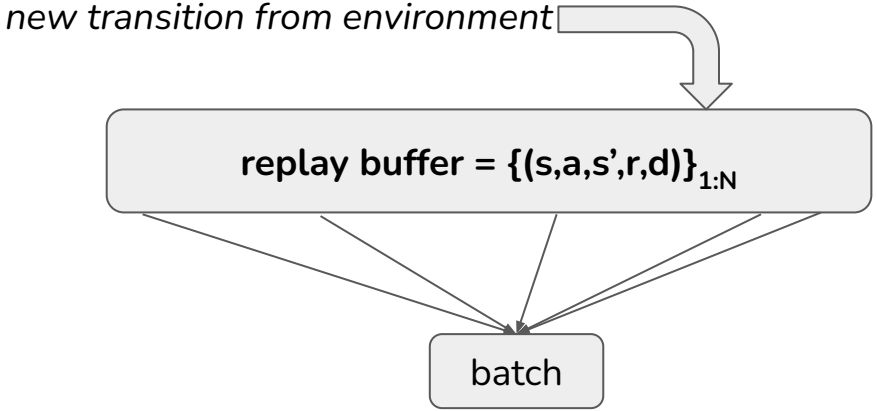
$$q = f_\theta(s, a)$$



Storing transitions in **replay buffer** reduces the correlation between samples in a batch

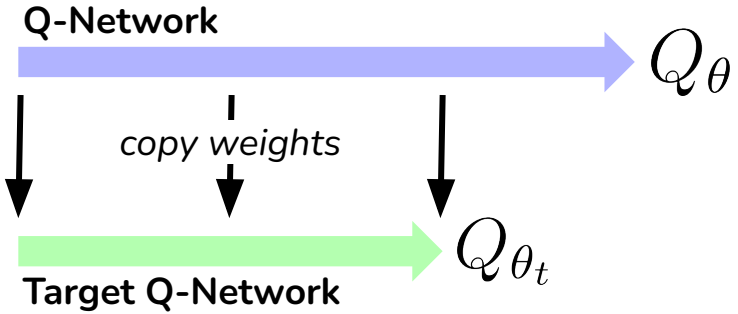


The gradient of a single batch is a sample estimate of the true gradient. If the batch is correlated then the estimated gradient will be biased



The replay buffer acts like a 'moving average' of the agent's experience. There should be some turnover, since learning is faster if distribution is closer to that of optimal agent

Target network (frozen copy of Q-network) is used to calculate more stable `q_target` values



$$\texttt{q\_pred} = Q_\theta(s, a)$$

$$\texttt{q\_target} = r + \gamma \max_a Q_{\theta_t}(s', a)$$

$$\texttt{loss} = \left\| \texttt{q\_target.detach()} - \texttt{q\_pred} \right\|^2$$

The frequency with which you copy weights from the Q-network to the target Q-network is a hyperparameter (often performed every 1000 optimization steps)

Reacher Environment as MDP

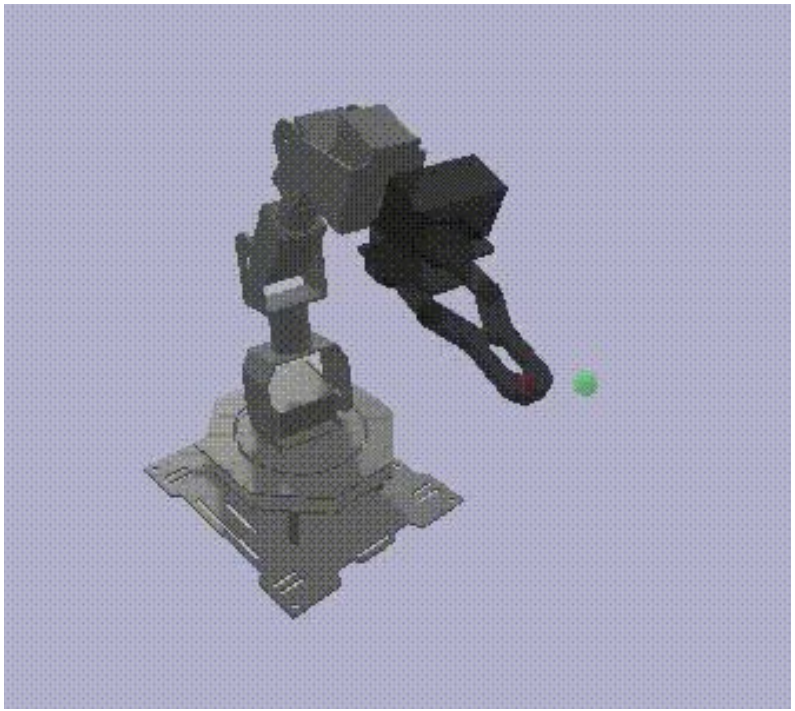
GOAL: move joints to achieve desired end-effector position as fast as possible

STATE:

ACTION:

REWARD:

GAMMA:



**these next slides refer to example script: `examples/dqn_reacher.py`*

Reacher Environment ass MDP

GOAL: move joints to achieve desired end-effector position as fast as possible

STATE: joint positions for [base,shoulder,elbow,wrist];
ignore wristRotation and gripper since we only want
end-effector position

ACTION: move single joint $\pm\Delta\theta$ (total 2*4=8 actions)

REWARD: sparse (+1 if at goal); dense (~1/dist2goal)

GAMMA: 0.98 (should be <1 for urgency)

Reacher Environment as gym.Env

__init__: create pybullet, add robot, create observation space & action space, set goal end-effector position

reset: set joint state of robot randomly within joint limits

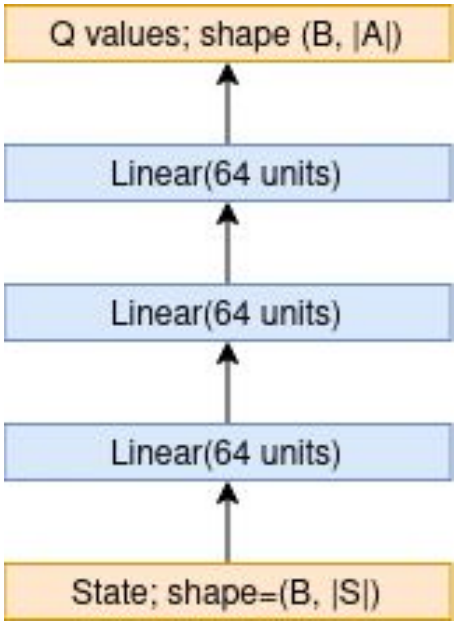
step: adjust single joint position accordingly, prevent robot from moving outside joint limits

get_obs: return current joint states

get_reward: calculate end-effector position, calculate reward based on distance between end-effector position and goal position

is_done: True if t_steps > episode_length or end-effector at goal position

Implementing Deep Q-Network as MLP



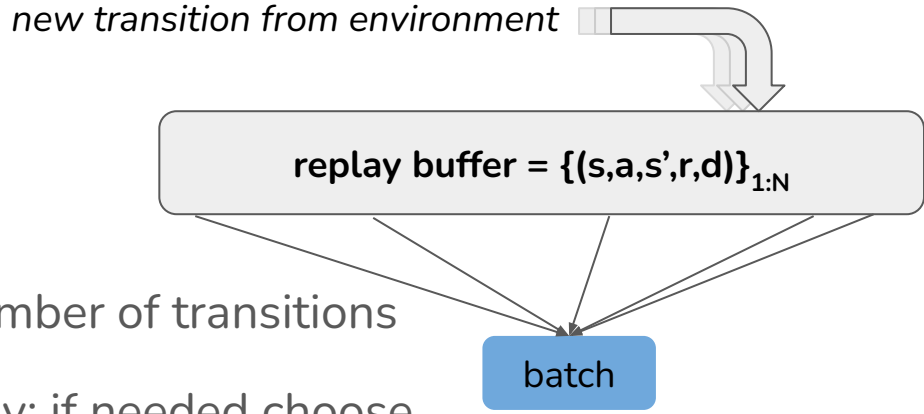
* ReLU between each layer

In addition to **forward**, it is often useful to have a method called **predict** which returns the action that maximizes the q-function. This is used during training to select actions so the gradient does not need to be computed

```
@torch.no_grad()
def predict(self, s: Tensor) -> Tensor:
    q_vals = self.forward(s)
    return torch.max(q_vals, dim=1)[1]
```

As an exercise, instantiate the layers in **QNetwork.__init__** and implement **QNetwork.forward**

Creating Replay Buffer



__init__: creates data arrays to store a set number of transitions

add_transition: stores <s,a,r,s',d> in data array; if needed choose oldest transition to replace (circular indexing)]

sample: samples a random batch of transitions from data arrays; it is common to return numpy arrays (but torch.Tensors would work)

Agent class

__init__: stores hyperparameters, instantiates ReplayBuffer, Q-Network, target Q-Network and optimizer to train Q-network

train: runs given number of environment steps, adding transitions to buffer, and optimizes Q-network to minimize TD-error on batches from buffer (make sure the target network is updated accordingly)

optimize: samples batch from buffer, calculates td-error, and back propagates td-error loss to network

select_action: performs epsilon-greedy action selection

policy: calculates action as the argmax of the q-network for a given state, input is numpy array provided by environment

Hyper Parameters

Deep Learning

Architecture (linear vs conv, etc)

Model capacity (hidden units or channels)

Loss function (mse, bce, huber)

Input (one-hot encodings, normalized imgs, etc.)

Output (activation function?)

Data augmentation

Learning rate

Deep Reinforcement Learning

Replay Buffer Size

Target Network Update Frequency

Discount factor (gamma)

Epsilon Schedule

MDP formulation (state and action space, reward scheme)

Typical Hyperparameter Values for DQN

Learning rate (1e-3 to 1e-4): decreasing may improve stability

Target update freq (~1000 opt steps): decreasing may improve stability; you should see the td-loss flatten out between updates

Epsilon schedule (usually linear from 1 to 0 for 90% of training); learning is faster with less epsilon, but insufficient exploration may end at local optima; remember that the reward curve is impacted by exploration

Gamma (0.98 is common); for tasks that take fewer time steps you may want to decrease it; if you are using

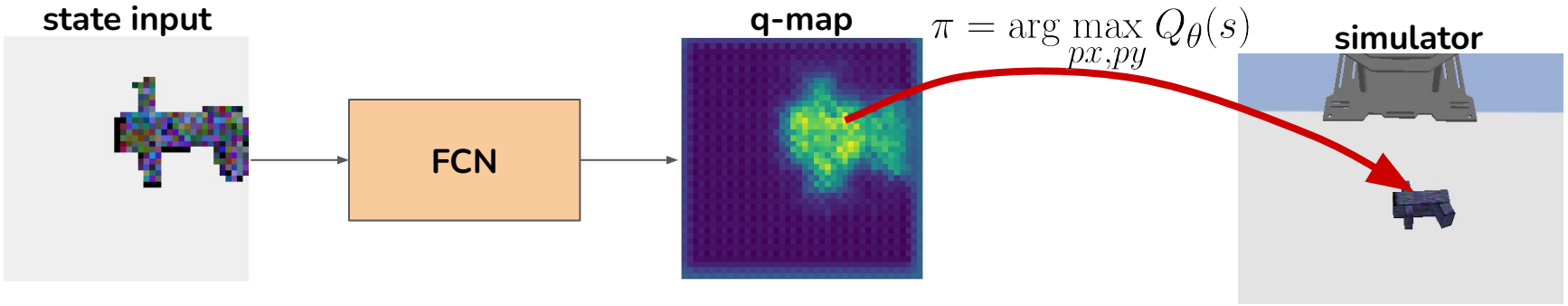
Buffer size: aim for a few turn-overs during training (if you train for 100k steps, then 50k buffer size will turn-over twice). If buffer is too small, it can cause catastrophic forgetting; if it is too large it will slow down training since most samples will be from bad policy

Training DQN: Debugging Techniques

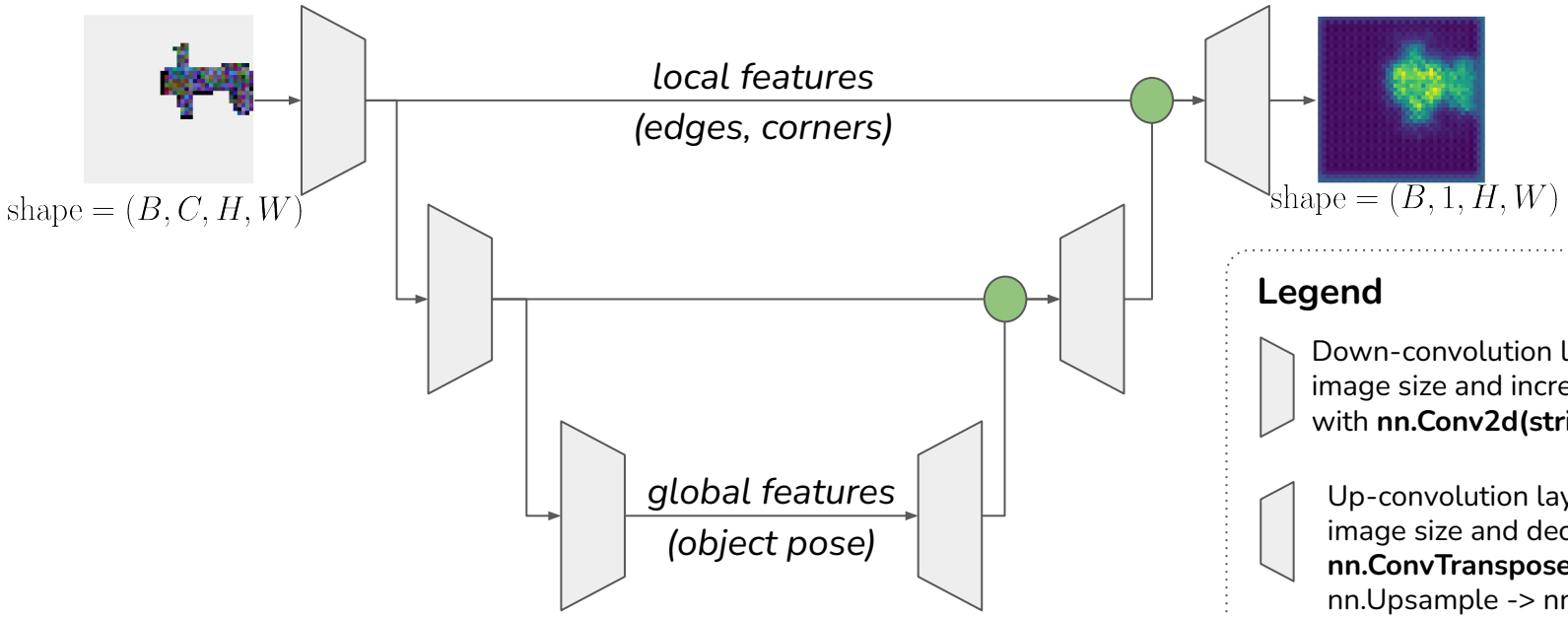
- Make sure to plot rewards and loss during training; if rewards suddenly drop off or td-loss rises rapidly, then the learning is unstable
- In addition, it may be insightful to log the number of steps per episode or the success rate (especially for complex reward function)
- It is almost always worth writing a ‘render’ function so that you can watch the policy’s actions and understand the failure modes
- In some cases, it is also useful to log or plot the q-values, which can indicate how confident the model is
- There can be a lot of variance between different seeds, so it is best to run multiple trials before making a conclusion

HW3: Top-Down Grasping with Pixel-wise Action Space

In pixel-wise action space, each pixel (px,py) corresponds to performing a grasp at the corresponding position in the real world. Thus, we predict a **q-map**, that represents the q-value of grasping at each pixel. A fully-convolutional network (FCN) is very effective at predicting q-maps.



A common FCN is called a U-Net, which is designed to process information at multiple receptive fields



Showcase your work at RISE.

It's time again for RISE -- the Research, Innovation, Scholarship, Entrepreneurship Virtual Expo!

RISE is the showcase for research and creative projects being undertaken by everyone at Northeastern: learners from every year of study, every major, every campus!

Whether it's nanotechnology, social science, design, engineering, historical analysis, game development, drug discovery, original creative work -- it's eligible for RISE. If you created it or discovered it during the past year, you can share it at RISE.

- **Abstracts due March 7.** The first step in showcasing your work at RISE is submitting an abstract or brief description of the work you hope to present by March 7. If your project is still underway, don't worry and don't wait! Share the abstract with the information you have now and your best ideas about what you hope to learn and discover.
- **Approved presentations due April 1.** If that abstract gets approved, we'll ask you to create a short video presentation about your project, due April 1.
- **Live question and answer on RISE Virtual Expo Day, April 14.** On the day of RISE, April 14, you'll then talk with judges, peers, faculty, staff, and friends and family about your project during one of two (you pick which one) live video judging rounds, from 10:00 AM - 12:00 PM ET or 2:00 PM - 4:00 PM ET.
- **RISE Awards announced April 15.** Cash prizes awarded for top projects!

Questions? Contact RISE@Northeastern.edu.

Survey to provide feedback

https://forms.gle/3XjKf1U4hvT3cSup8