REINFORCEMENT LEARNING (DD3359)

O-03 END-TO-END LEARNING

RIKA ANTONOVA  ALI GHADIRZADEH
ANTONOVA@KTH.SE  ALGH@KTH.SE
Formulate the problem as an MDP (or POMDP)

- State space captures information about environment
  - e.g. positions and velocities of the objects in the scene

- Action space captures what our agent can do
  - e.g. position/acceleration/torque commands to each joint

- Select appropriate representation and parameters
  - state/action space continuous vs. discrete
  - horizon length and discount factor
  - fully or partially observed state (MDP vs POMDP)
RL: What We Know So Far

- Apply an appropriate RL algorithm to solve the problem
- RL has been used for a variety of research problems in Robotics
- To get an overview of what approach might be appropriate for your problem – start by looking through the relevant surveys

E.g.:

- “A Survey on Policy Search for Robotics”. Marc Peter Deisenroth, Gerhard Neumann, Jan Peters 2013
- “Learning control in robotics”. Stefan Schaal, Christopher G. Atkeson, 2010
- ... and many more sources for specific subtasks/problems
RL: Deeper Challenges

- When state/action space is large or continuous function approximation is employed
- Most recently, deep neural networks were successfully used to approximate value and policy functions
- But getting NNs to train well for an RL problem is not trivial

  more difficult than supervised and unsupervised/structure learning!
“End-to-end” Training

- Notable work on training NNs for RL was done by DeepMind in the context of games
  - DQN: first visible demonstration of learning from pixels from “scratch” (no prior domain knowledge) using a generic algorithm (NN structure is not task-specific)

  Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller.

- Approaches from this line of work are useful to know about when working with NN-based RL in general
Recall: Q-Learning

- Bellman Optimality Equation:

\[
q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \right| S_t = s, A_t = a] = \sum_{s', r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right].
\]

- Q-Learning - off-policy TD learning:

Repeat (for each episode):
  Repeat (for each step of episode):
    Choose \( A \) from \( S \) using policy derived from \( Q \) (e.g., \( \epsilon \)-greedy)
    Take action \( A \), observe \( R, S' \)
    \( Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_a Q(S', a) - Q(S, A) \right] \)
    \( S \leftarrow S' \)
Deep Q-Learning?

- We want deep neural network as function approximator for Q
- Can we simply use TD error as a “loss” to train our NN in a standard “supervised learning” way?

Problems?
Deep Q-Learning?

Problems:

1. \((s,a,r,s')\) tuples are not iid (independent identically distributed)
   - but standard supervised learning approaches would need iid

2. distribution of samples can change when policy changes
   - but supervised learning usually makes stationarity assumption

3. large reward values (e.g. from longer episodes)
   might cause instabilities when training NNs
DQN: Human-level control through deep RL

- Use experience replay
  1. break correlations in the data by shuffling \((s,a,r,s')\) tuples
  2. learn from all past policies that explored the space

3. Reduce oscillations/instabilities
   - freeze weights of NN \((\theta_{i-1})\) while updating current weights \((\theta_i)\) on a batch of training data
   - clip rewards or normalize them adaptively

DQN: Human-level control through deep RL

- Bellman Optimality Equation:

\[ q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \]

- Same as:

\[ Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right] \]


Slide by: Rika
DQN: Human-level control through deep RL

- Construct loss function based on Bellman Optimality Equation

\[ y_i = \mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a] \]

\[ L_i (\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \]

"behavior" distribution: states and actions encountered by the agent when learning

target for training iteration \(i\)

NN weights from previous training iteration

NN weights for current iteration


Slide by: Rika
Differentiate the “squared loss” $L_i(\theta_i) = \mathbb{E}(y_i - Q(s, a; \theta_i))^2$
with respect to NN weights $\theta_i$

holding NN weights from previous iteration fixed when differentiating

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Do gradient descent to find optimal NN weights

DEMO from
“Human-level control through deep reinforcement learning”.
Mnih et al, Nature 2015
Recall from the lecture on continuous action spaces:

- DDPG is a model-free off-policy RL method
  - learns a deterministic policy (actor), and can use any stochastic policy during training for exploration
  - maintains a separate NN for learning Q function (critic)

- Why learn deterministic policies?
  - could be easier to learn than stochastic and desirable when executing on robots

“Continuous Control with Deep Reinforcement Learning”. Lillicrap et al, ICLR 2016
Replay Buffer

- At each training step:
  - sample a “minibatch” uniformly from the buffer
  - use batch normalization (normalize each dimension to get unit mean and variance)
  - update the critic and the actor

“Continuous Control with Deep Reinforcement Learning”. Lillicrap et al, ICLR 2016
“Soft” Target Networks

- use a copy of the actor and critic networks for target values when computing loss
- weights of these target networks are updated by slowly tracking the learned networks

\[
\begin{align*}
\theta_Q' & \leftarrow \tau \theta_Q + (1 - \tau) \theta_Q' \\
\theta_\mu' & \leftarrow \tau \theta_\mu + (1 - \tau) \theta_\mu'
\end{align*}
\]

pick a rate ($\tau \ll 1$)

“Continuous Control with Deep Reinforcement Learning”. Lillicrap et al, ICLR 2016
DDPG: Deep Deterministic Policy Gradient

Learn critic weights $\theta^Q$ by minimizing the loss:

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^\mu')|\theta^Q')$$

"target" networks with weights slowly tracking actor and critic NN weights

“Continuous Control with Deep Reinforcement Learning”. Lillicrap et al, ICLR 2016
DDPG: Deep Deterministic Policy Gradient

Learn actor weights $\theta^\mu$ using deterministic policy gradient theorem

$$\nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

- states $s_i$ from a minibatch of size $N$
  (collected when running actor with weights $\theta^\mu$ during training episode)

- this is a deterministic version of the stochastic policy gradient theorem that we studied in one of the previous lectures

“Continuous Control with Deep Reinforcement Learning”. Lillicrap et al, ICLR 2016

Slide by: Rika
“End-to-end” RL Challenges

- Approaches like DQN and DDPG learn “from scratch”
  - upside: deep NNs will automatically learn to extract features useful for the task
    - e.g. can learn directly from pixels / images of the scene!
  - downside: might not be sample-efficient
    - it might take millions of samples to learn something useful
    - this could be prohibitively slow for learning on real hardware in real time

- So, the next part of the lecture is on data-efficient algorithms designed to learn on real robots

Slide by: Rika
End-to-end deep learning

Recap

\[ \theta = \arg\min_{\theta} \mathbb{E}_{\pi_{\theta}} \left[ \sum_{t=1}^{T} c(s_t, u_t) \right] \]
RL – Policy Search

- Generate trajectories given the current policy
- Evaluate sampled trajectories
- Update policy to make good samples more likely

Inefficient for large policies
Randomly initialized policies are less likely to generate good trajectories to learn from
Guided Policy Search

Ingredients

Policy Search RL  Complex Dynamics  Complex Policies  difficult
## Guided Policy Search

### Ingredients

<table>
<thead>
<tr>
<th>Method</th>
<th>Complex Dynamics</th>
<th>Complex Policies</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Search RL</td>
<td></td>
<td></td>
<td>difficult</td>
</tr>
<tr>
<td>Supervised learning</td>
<td></td>
<td>Complex Policies</td>
<td>manageable</td>
</tr>
<tr>
<td>Optimal Control</td>
<td>Complex Dynamics</td>
<td></td>
<td>manageable</td>
</tr>
</tbody>
</table>
Guided Policy Search

Ingredients

Policy Search RL
- Complex Dynamics
- Complex Policies
  - difficult

Supervised learning
- Complex Policies
  - manageable

Optimal Control
- Complex Dynamics
  - manageable

Optimal Control + Supervised Learning → Policy

Slide by: Ali
GPS

Trajectory Optimization

• Find a trajectory based on optimal control
• Solve the regression problem to match the policy to the observed trajectory
• Find a trajectory based on optimal control
• Solve the regression problem to match the policy to the observed trajectory

This naïve approach would fail once the policy deviates from the demonstrated trajectory
GPS

Trajectory Optimization

- Find a trajectory based on optimal control
- Solve the regression problem to match the policy to the observed trajectory

This naïve approach would fail once the policy deviates from the demonstrated trajectory

Solution

- Find the widest trajectory distribution
- Sample from this distribution
- Solve the regression problem to lean the policy

\[ p_t(u_t|x_t) = N(K_t x_t + k_t, C_t) \]
Produced action trajectories may not be well suited to train a neural network policy. Adapt teacher to produce samples well-suited for policy training.

See presentation at [https://www.youtube.com/watch?v=EtMyH--vnU](https://www.youtube.com/watch?v=EtMyH--vnU)
GPS

Constraints

Solution

- Alternatively
  - Optimize the NN policy to match produced action trajectories
  - Optimize trajectories with an extra constraint to avoid samples very different from the policy

\[ p_i(u_t | x_t) \]

Train NN policy parameters with observed trajectories

\[ \pi_\theta(u_t | x_t) \]

Optimize local policies to minimize the loss function

Full state → Local policies

Neural Network Policy → Observation

Slide by: Ali
Guided Policy Search

\[
\min_{\theta} \mathbb{E}_{\pi_\theta}[c(\tau)] \\
s.t. \quad D_t(\pi_\theta, p) = 0 \quad \forall t
\]

Dual gradient decent

\[
\mathcal{L}(\theta, p, \lambda) = \mathbb{E}_p[c(\tau)] + \sum_{t=1}^{T} \lambda_t D_t(\pi_\theta, p)
\]
Guided Policy Search

\[
\min_{\theta} \mathbb{E}_{\pi_{\theta}}[c(\tau)]
\]

s.t. \quad D_t(\pi_{\theta}, p) = 0 \quad \forall t

\[\mathcal{L}(\theta, p, \lambda) = \mathbb{E}_p[c(\tau)] + \sum_{t=1}^{T} \lambda_t D_t(\pi_{\theta}, p)\]

Optimize \(\mathcal{L}(\theta, p, \lambda)\) w.r.t. \(p\)

Optimize \(\mathcal{L}(\theta, p, \lambda)\) w.r.t. \(\theta\)

Optimize \(\lambda\)

\[\lambda_t \leftarrow \lambda_t + \eta D_t(\pi_{\theta}, p)\]
**Time-varying linear-Gaussian controllers**

\[
p_i(u_t|x_t) = \mathcal{N}(K_tx_t + k_t, C_t)
\]
GPS

Local policy optimization

*Time-varying linear-Gaussian controllers*

\[ p_i(u_t | x_t) = \mathcal{N}(K_t x_t + k_t, C_t) \]

- Sample from each local policy and apply it to the real robot
GPS Local policy optimization

Time-varying linear-Gaussian controllers

\[ p_i(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, C_t) \]

- Sample from each local policy and apply it to the real robot
- Fit local linear-Gaussian dynamics for each local policy

\[ p_i(x_{t+1}|x_t, u_t) = \mathcal{N}(f_{xt} x_t + f_{ut} u_t + f_{ct}, F_t) \]
GPS

Local policy optimization

Time-varying linear-Gaussian controllers

\[ p_i(u_t | x_t) = \mathcal{N}(K_t x_t + k_t, C_t) \]

- Sample from each local policy and apply it to the real robot
- Fit local linear-Gaussian dynamics for each local policy

\[ p_i(x_{t+1} | x_t, u_t) = \mathcal{N}(f_{xt} x_t + f_{ut} u_t + f_{ct}, F_t) \]

- Update local policies using the fitted dynamics via modified LQR algorithm

\[
\min_{K_t,k_t,C_t} \sum_{t=1}^{T} \mathbb{E}_{p_i(x_t,u_t)} [c(x_t,u_t)] \\
\text{s.t. } D_{KL}(p_i(\tau), \bar{p}_i(\tau)) < \epsilon
\]
GPS

Local policy optimization

\[
\min_{K_t,k_t,C_t} \sum_{t=1}^{T} \mathbb{E}_{p_t(x_t,u_t)}[c(x_t,u_t)] \\
\text{s.t.} \quad D_{KL}(p_t(\tau), \tilde{p}_t(\tau)) < \epsilon
\]
\[
\min_{K_t, k_t, C_t} \sum_{t=1}^{T} \mathbb{E}_{p_i(x_t, u_t)}[c(x_t, u_t)]
\]

s.t. \quad D_{KL}(p_i(\tau), \tilde{p}_i(\tau)) < \epsilon

\[
D_{KL}(p_i(\tau)||\tilde{p}_i(\tau)) = \sum_{t=1}^{T} D_{KL}(p_i(u_t|x_t)||\tilde{p}_i(u_t|x_t)) = \sum_{t=1}^{T} -E_{p_i(x_t, u_t)}[\log \tilde{p}_i(u_t|x_t)] - H(p_i(u_t|x_t))
\]
\[
p(\tau) = p(x_1) \prod_{t=1}^{T} p(x_{t+1}|x_t, u_t)p(u_t|x_t), \quad q(\tau) = p(x_1) \prod_{t=1}^{T} p(x_{t+1}|x_t, u_t)q(u_t|x_t).
\]

\[
D_{KL}(p(\tau)||q(\tau)) = E_{p(\tau)} \left[ \log p(\tau) - \log q(\tau) \right] = E_{p(\tau)} \left[ \sum_{t=1}^{T} \log p(u_t|x_t) - \log q(u_t|x_t) \right] = \sum_{t=1}^{T} E_{p(x_t,u_t)} \left[ \log p(u_t|x_t) - \log q(u_t|x_t) \right] = \sum_{t=1}^{T} -E_{p(x_t,u_t)} \left[ \log q(u_t|x_t) \right] - E_{p(x_t)} \left[ H(p(u_t|x_t)) \right] = \sum_{t=1}^{T} -E_{p(x_t,u_t)} \left[ \log q(u_t|x_t) \right] - H(p(u_t|x_t))
\]
Guided Policy Search

- GPS unnecessarily complicated?
Guided Policy Search

- GPS unnecessarily complicated?

Generate samples from local policies
Guided Policy Search

- GPS unnecessarily complicated?

\[ p_i(x_{t+1}|x_t, u_t) \]

- Generate samples from local policies
- Fit local dynamics
Guided Policy Search

- GPS unnecessarily complicated?

- Generate samples from local policies
- Fit local dynamics
- Optimize global policy parameters

\[ \sum_{i,t} \lambda_{i,t} D_{KL}(p_i(x_t) \| \pi_{\theta}(u_t|x_t)) \]
Guided Policy Search

- GPS unnecessarily complicated?

\[
\min_{K_t, \theta_t, C_t} \sum_{t=1}^{T} \mathbb{E}_{p_i(x_t, u_t)} [c(x_t, u_t)] \quad \text{s.t.} \quad D_{KL}(p_i(\tau), \bar{p}_i(\tau)) < \epsilon
\]
Guided Policy Search

- GPS unnecessarily complicated?

Generate samples from local policies

Fit local dynamics

Optimize global policy parameters

Update local policy

Increment dual variable

\[ \alpha D_{KL}(p_i(x_t)\pi_\theta(u_t|x_t)\|p_i(x_t, u_t)) \]
Mirror Decent Guided Policy Search
Generate
samples from the
clocal or global
policies
Mirror Decent Guided Policy Search

- Generate samples from the local or global policies
- Fit local dynamics

\[ p_i(x_{t+1}|x_t, u_t) \]
Mirror Decent Guided Policy Search

- Generate samples from the local or global policies
- Fit local dynamics
- Linearize global policy using samples

\[ \pi_{\theta_i}(u_t|x_t) \]
Mirror Decent Guided Policy Search

- Generate samples from the local or global policies
- Fit local dynamics
- Linearize global policy using samples
- Update local policy

\[ p_i \leftarrow \arg\min_{p_i} E_{p_i}(\tau) \left[ \sum_{t=1}^{T} \ell(x_t, u_t) \right] \]

\[ D_{KL}(p_i(\tau) \| \bar{p}_{\theta_i}(\tau)) \leq \epsilon \]
Mirror Decent Guided Policy Search

Generate samples from the local or global policies

Fit local dynamics

Linearize global policy using samples

Update local policy

Update global policy

\[ \pi_\theta \leftarrow \arg\min_\theta \sum_{t,i,j} D_{KL}(\pi_\theta(u_t|x_{t,i,j}) || p_i(u_t|x_{t,i,j})) \]
Mirror Decent Guided Policy Search

- Generate samples from the local or global policies
- Fit local dynamics
- Linearize global policy using samples
- Update local policy
- Update global policy

- MDGPS less complicated
- Better convergence properties

Slide by: Ali
Path Integral Guided Policy Search

- LQR policies require smooth and differentiable loss function
- Path integral RL with MDGPS (model-free)

Algorithm 1 MDGPS with PI² and Global Policy Sampling

1: for iteration $k \in \{1, \ldots, K\}$ do
2: Generate samples $\mathcal{D} = \{\tau_i\}$ by running noisy $\pi_\theta$ on each randomly sampled instance
3: Perform one step of optimization with PI² independently on each instance:
   $\min_{\pi} E_p[l(\tau)] \text{ s.t. } D_{KL}(p(u_t|x_t)\|\pi_\theta(u_t|x_t)) \leq \epsilon$
4: Train global policy with optimized controls using supervised learning:
   $\pi_\theta \leftarrow \arg\min_{\pi} \sum_{i,t} D_{KL}(\pi_\theta(u_t|x_{i,t})\|p(u_t|x_{i,t}))$
5: end for