Tutorial: Deep Reinforcement Learning

David Silver, Google DeepMind

Outline

Reinforcement Learning in a nutshell

RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximise future reward

Deep Learning in a nutshell

DL is a general-purpose framework for representation learning

- Given an objective
- Learn representation that is required to achieve objective
- Directly from raw inputs
- Using minimal domain knowledge

Deep Reinforcement Learning: AI = RL + DL

We seek a single agent which can solve any human-level task

- RL defines the objective
- DL gives the mechanism
- RL + DL = general intelligence

Examples of Deep RL @DeepMind

- Play games: Atari, poker, Go, ...
- Explore worlds: 3D worlds, Labyrinth, ...
- Control physical systems: manipulate, walk, swim, ...
- Interact with users: recommend, optimise, personalise, ...

Outline

Introduction to Deep Learning

Introduction to Reinforcement Learning

Value-Based Deep RL

Policy-Based Deep RL

Model-Based Deep RL

Deep Representations

- A deep representation is a composition of many functions

$$\begin{tikzcd}
x \arrow[r] & h_1 \arrow[r] & \dots \arrow[r] & h_n \arrow[r] & y \arrow[r] & l \\
 & \mathbf{w_1} \arrow[u] & \dots & \mathbf{w_n} \arrow[u] & &
\end{tikzcd}$$

- Its gradient can be backpropagated by the chain rule

$$\begin{tikzcd}
\frac{\partial l}{\partial x} & \frac{\partial l}{\partial h_1} \arrow[l, "\frac{\partial h_1}{\partial x}"'] \arrow[d, "\frac{\partial h_1}{\partial w_1}"'] & \dots \arrow[l, "\frac{\partial h_2}{\partial h_1}"'] & \frac{\partial l}{\partial h_n} \arrow[l, "\frac{\partial h_n}{\partial h_{n-1}}"'] \arrow[d, "\frac{\partial h_n}{\partial w_n}"'] & \frac{\partial l}{\partial y} \arrow[l, "\frac{\partial y}{\partial h_n}"'] \\
 & \frac{\partial l}{\partial \mathbf{w_1}} & \dots & \frac{\partial l}{\partial \mathbf{w_n}} &
\end{tikzcd}$$

Deep Neural Network

A deep neural network is typically composed of:

- Linear transformations

$$h_{k+1} = Wh_k$$

- Non-linear activation functions

$$h_{k+2} = f(h_{k+1})$$

- A loss function on the output, e.g.
    - Mean-squared error $l = ||y^* - y||^2$
    - Log likelihood $l = \log \mathbb{P}[y^*]$

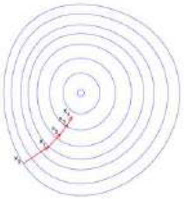
Training Neural Networks by Stochastic Gradient Descent

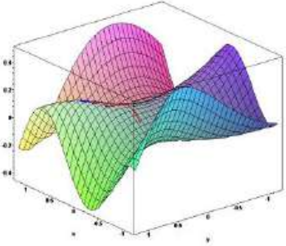
- Sample gradient of expected loss $L(\mathbf{w}) = \mathbb{E}\left[l\right]$

$$\frac{\partial l}{\partial \mathbf{w}} \sim \mathbb{E}\left[\frac{\partial l}{\partial \mathbf{w}}\right] = \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

- Adjust $\mathbf{w}$ down the sampled gradient

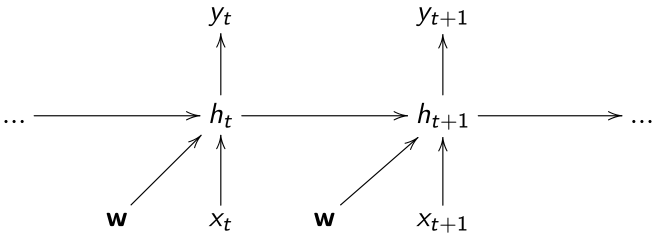
$$\Delta w \propto \frac{\partial l}{\partial \mathbf{w}}$$



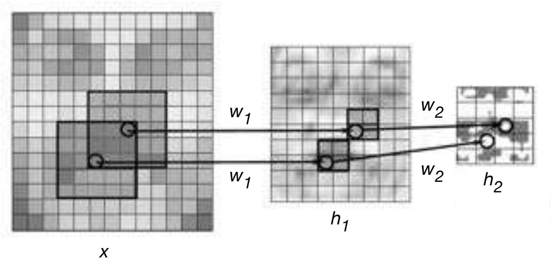


Weight Sharing

Recurrent neural network shares weights between time-steps



Convolutional neural network shares weights between local regions



Outline

Introduction to Deep Learning

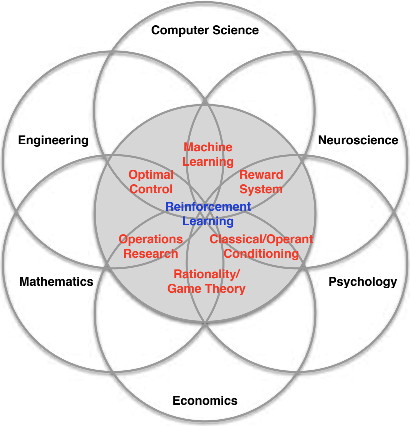
Introduction to Reinforcement Learning

Value-Based Deep RL

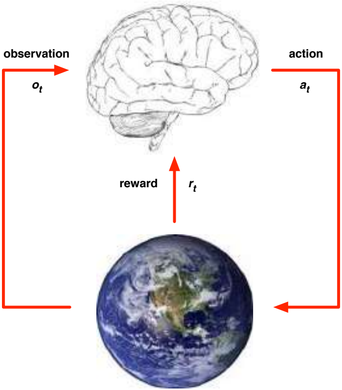
Policy-Based Deep RL

Model-Based Deep RL

Many Faces of Reinforcement Learning



Agent and Environment



- At each step t the agent:
    - Executes action a_t
    - Receives observation o_t
    - Receives scalar reward r_t
- The environment:
    - Receives action a_t
    - Emits observation o_{t+1}
    - Emits scalar reward r_{t+1}

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t)$$

- In a fully observed environment

$$s_t = f(o_t)$$

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action:
    - Deterministic policy: $a = \pi(s)$
    - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s, a\right]$$

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) = Q^{\pi^*}(s, a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s, a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s, a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximises over all decisions. Informally:

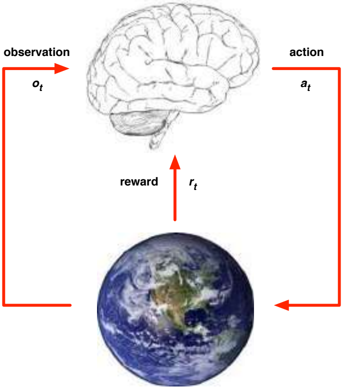
$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

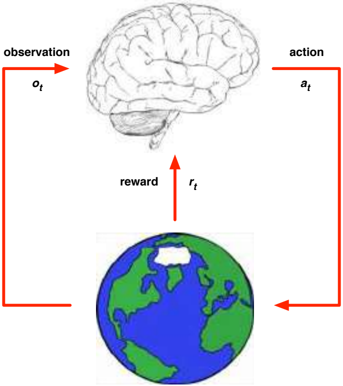
$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Value Function Demo

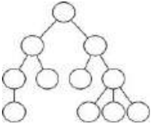
Model



Model



- Model is learnt from experience
- Acts as proxy for environment
- Planner interacts with model
- e.g. using lookahead search



Approaches To Reinforcement Learning

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

Deep Reinforcement Learning

- Use deep neural networks to represent
    - Value function
    - Policy
    - Model
- Optimise loss function by stochastic gradient descent

Outline

Introduction to Deep Learning

Introduction to Reinforcement Learning

Value-Based Deep RL

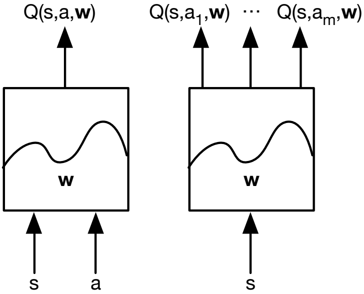
Policy-Based Deep RL

Model-Based Deep RL

Q-Networks

Represent value function by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Q-Learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a'} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

Q-Learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a'} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Converges to Q^* using table lookup representation

Q-Learning

- Optimal Q-values should obey Bellman equation

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

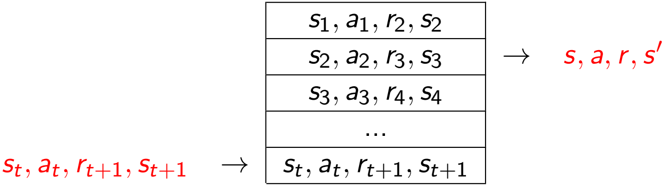
- Treat right-hand side $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Deep Q-Networks (DQN): Experience Replay

To remove correlations, build data-set from agent's own experience



s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

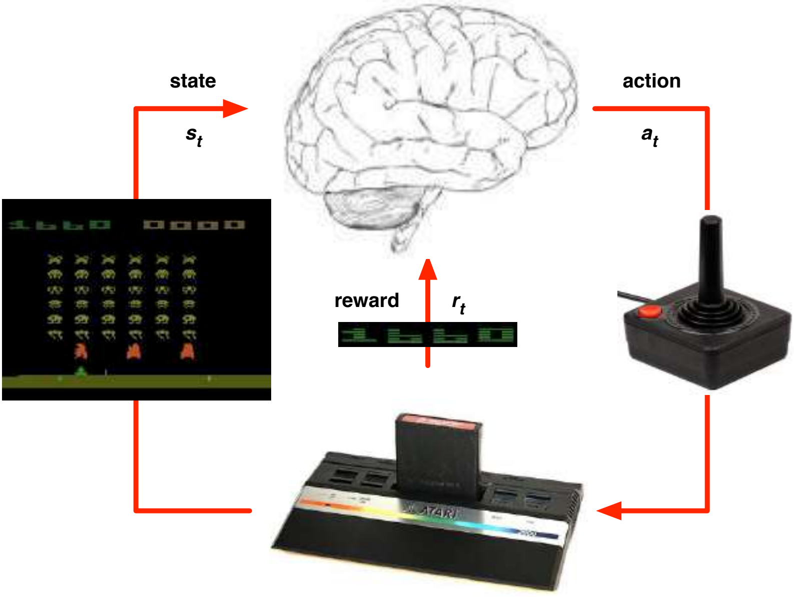
$\rightarrow s, a, r, s'$

Sample experiences from data-set and apply update

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

Deep Reinforcement Learning in Atari



state

s_t

action

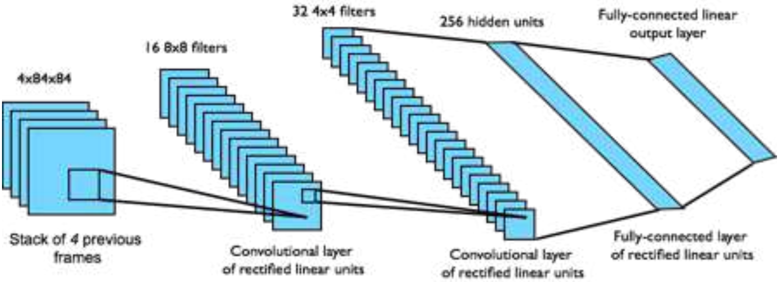
a_t

reward

r_t

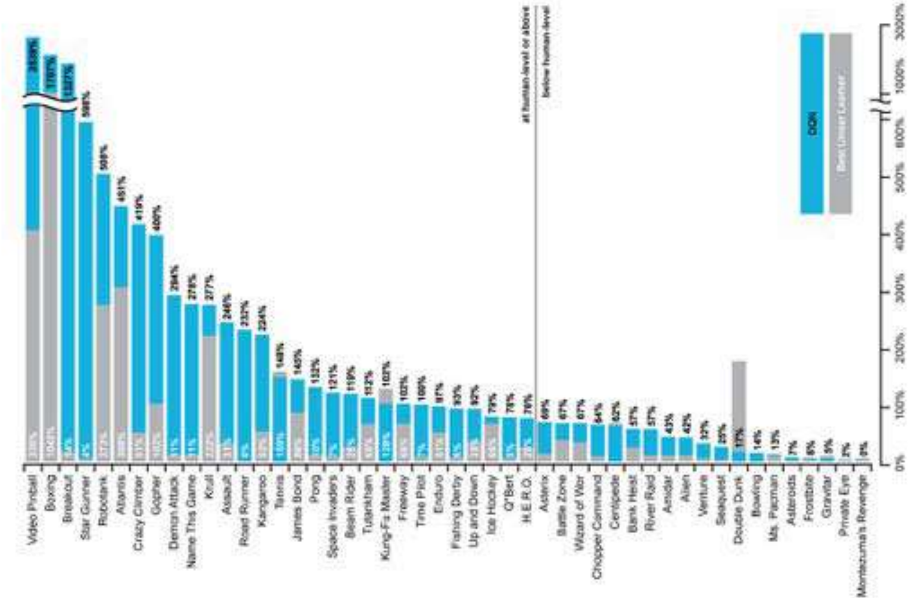
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

DQN Results in Atari



DQN Atari Demo

DQN paper

www.nature.com/articles/nature14236

DQN source code:

sites.google.com/a/deepmind.com/dqn/

Improvements since Nature DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$l = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

Improvements since Nature DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

Improvements since Nature DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Improvements since Nature DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
    - Current Q-network $\mathbf{w}$ is used to select actions
    - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
    - Store experience in priority queue according to DQN error

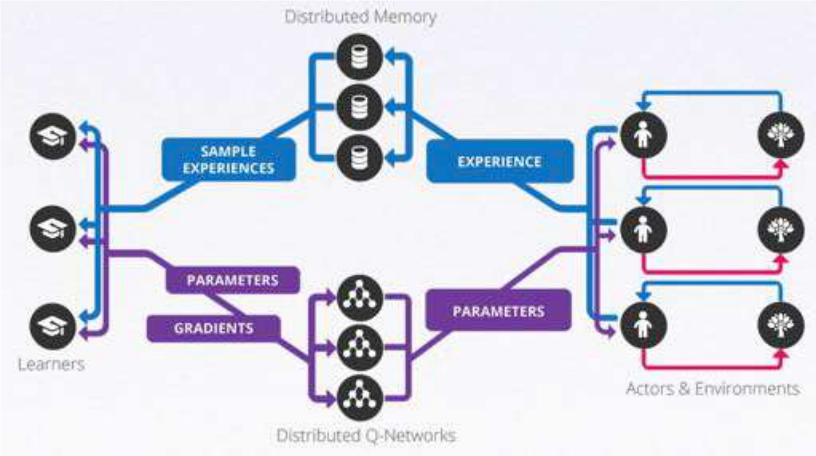
$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
    - Action-independent value function $V(s, v)$
    - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Combined algorithm: 3x mean Atari score vs Nature DQN

Gorila (General Reinforcement Learning Architecture)



- 10x faster than Nature DQN on 38 out of 49 Atari games
- Applied to recommender systems within Google

Asynchronous Reinforcement Learning

- Exploits multithreading of standard CPU
- Execute many instances of agent in parallel
- Network parameters shared between threads
- Parallelism decorrelates data
    - Viable alternative to experience replay
- Similar speedup to Gorila - on a single machine!

Outline

Introduction to Deep Learning

Introduction to Reinforcement Learning

Value-Based Deep RL

Policy-Based Deep RL

Model-Based Deep RL

Deep Policy Networks

- Represent policy by deep network with weights $\mathbf{u}$

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Define objective function as total discounted reward

$$L(\mathbf{u}) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \mid \pi(\cdot, \mathbf{u})\right]$$

- Optimise objective end-to-end by SGD
- i.e. Adjust policy parameters $\mathbf{u}$ to achieve more reward

Policy Gradients

How to make high-value actions more likely:

- The gradient of a stochastic policy $\pi(a|s, \mathbf{u})$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)\right]$$

Policy Gradients

How to make high-value actions more likely:

- The gradient of a stochastic policy $\pi(a|s, \mathbf{u})$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)\right]$$

- The gradient of a deterministic policy $a = \pi(s)$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}\left[\frac{\partial Q^{\pi}(s, a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}}\right]$$

- if a is continuous and Q is differentiable

Actor-Critic Algorithm

- Estimate value function $Q(s, a, \mathbf{w}) \approx Q^{\pi}(s, a)$
- Update policy parameters $\mathbf{u}$ by stochastic gradient ascent

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q(s, a, \mathbf{w})$$

or

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

Asynchronous Advantage Actor-Critic (A3C)

- Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + ...|s\right]$$

- Q-value estimated by an *n*-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

Asynchronous Advantage Actor-Critic (A3C)

- Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + ...|s\right]$$

- Q-value estimated by an *n*-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

- Actor is updated towards target

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t|s_t, \mathbf{u})}{\partial \mathbf{u}} (q_t - V(s_t, \mathbf{v}))$$

- Critic is updated to minimise MSE w.r.t. target

$$l_v = (q_t - V(s_t, \mathbf{v}))^2$$

Asynchronous Advantage Actor-Critic (A3C)

- Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + ...|s\right]$$

- Q-value estimated by an *n*-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

- Actor is updated towards target

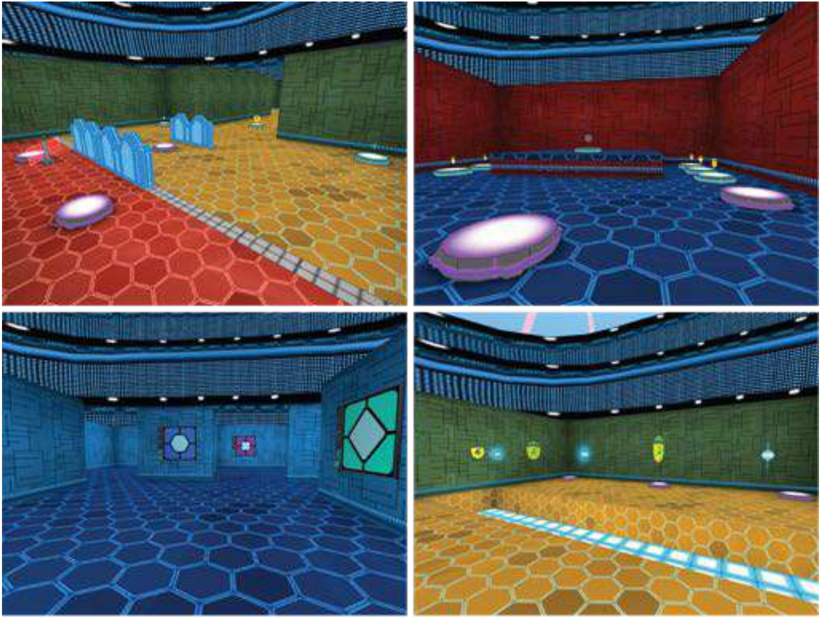
$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t|s_t, \mathbf{u})}{\partial \mathbf{u}}(q_t - V(s_t, \mathbf{v}))$$

- Critic is updated to minimise MSE w.r.t. target

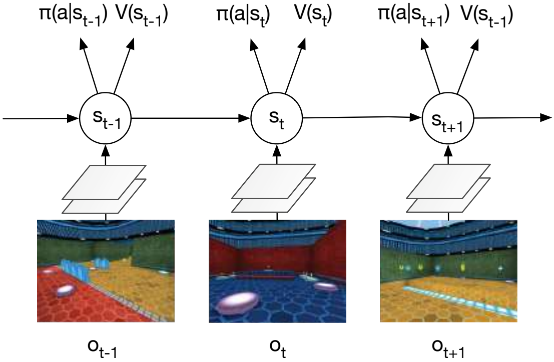
$$l_v = (q_t - V(s_t, \mathbf{v}))^2$$

- 4x mean Atari score vs Nature DQN

Deep Reinforcement Learning in Labyrinth



A3C in Labyrinth



- End-to-end learning of softmax policy $\pi(a|s_t)$ from pixels
- Observations o_t are raw pixels from current frame
- State $s_t = f(o_1, ..., o_t)$ is a recurrent neural network (LSTM)
- Outputs both value $V(s)$ and softmax over actions $\pi(a|s)$
- Task is to collect apples ($+1$ reward) and escape ($+10$ reward)

A3C Labyrinth Demo

Demo:

www.youtube.com/watch?v=nMR5mjCFZCw&feature=youtu.be

Labyrinth source code (coming soon):

sites.google.com/a/deepmind.com/labyrinth/

Deep Reinforcement Learning with Continuous Actions

How can we deal with high-dimensional continuous action spaces?

- Can't easily compute $\max_a Q(s, a)$
    - Actor-critic algorithms learn without max
- Q-values are differentiable w.r.t a
    - Deterministic policy gradients exploit knowledge of $\frac{\partial Q}{\partial a}$

Deep DPG

DPG is the continuous analogue of DQN

- Experience replay: build data-set from agent's experience
- Critic estimates value of current policy by DQN

$$l_w = \left(r + \gamma Q(s', \pi(s', u^-), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

  To deal with non-stationarity, targets $\mathbf{u}^-, \mathbf{w}^-$ are held fixed

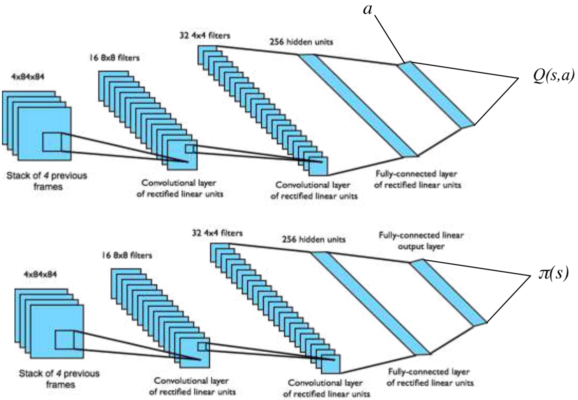
- Actor updates policy in direction that improves Q

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

- In other words critic provides loss function for actor

DPG in Simulated Physics

- Physics domains are simulated in MuJoCo
- End-to-end learning of control policy from raw pixels s
- Input state s is stack of raw pixels from last 4 frames
- Two separate convnets are used for Q and π
- Policy π is adjusted in direction that most improves Q

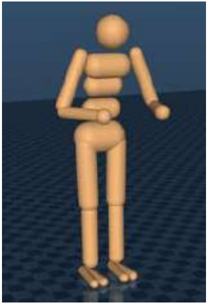


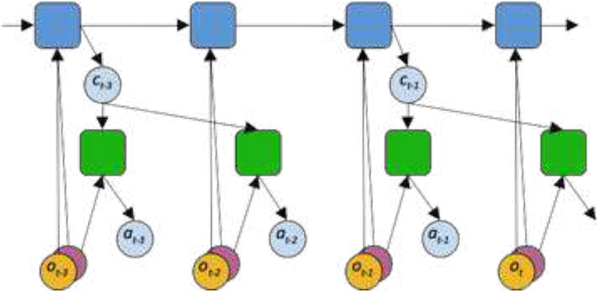
DPG in Simulated Physics Demo

- Demo: DPG from pixels

A3C in Simulated Physics Demo

- Asynchronous RL is viable alternative to experience replay
- Train a hierarchical, recurrent locomotion controller
- Retrain controller on more challenging tasks





Fictitious Self-Play (FSP)

Can deep RL find Nash equilibria in multi-agent games?

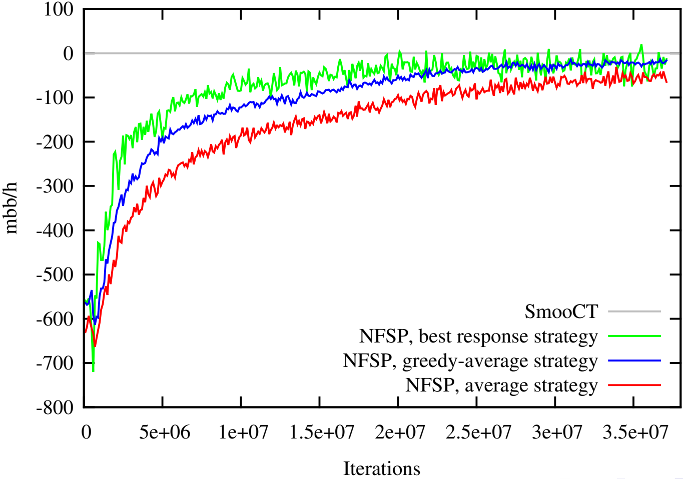
- Q-network learns "best response" to opponent policies
  - By applying DQN with experience replay
  - c.f. fictitious play
- Policy network $\pi(a|s, \mathbf{u})$ learns an average of best responses

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}}$$

- Actions *a* sample mix of policy network and best response

Neural FSP in Texas Hold'em Poker

- Heads-up limit Texas Hold'em
- NFSP with raw inputs only (no prior knowledge of Poker)
- vs SmooCT (3x medal winner 2015, handcrafted knowlege)



Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning
- Value-Based Deep RL
- Policy-Based Deep RL
- Model-Based Deep RL

Learning Models of the Environment

- Demo: generative model of Atari
- Challenging to plan due to compounding errors
  - Errors in the transition model compound over the trajectory
  - Planning trajectories differ from executed trajectories
  - At end of long, unusual trajectory, rewards are totally wrong

Deep Reinforcement Learning in Go

What if we have a perfect model? e.g. game rules are known

AlphaGo paper:
www.nature.com/articles/nature16961

AlphaGo resources:
deepmind.com/alphago/

Conclusion

- General, stable and scalable RL is now possible
- Using deep networks to represent value, policy, model
- Successful in Atari, Labyrinth, Physics, Poker, Go
- Using a variety of deep RL paradigms