Reinforcement Learning

Deep Reinforcement Learning I

Stefano Albrecht, Arrasy Rahman, Filippos Christianos, Lukas Schäfer

3 March 2020

THE UNIVERSITY of EDINBURGH
informatics

Lecture Outline

- Motivation
- Deep Learning
- Deep Reinforcement Learning
    - Experience replay
    - Target networks
    - Deep Q-Networks
    - Extensions of DQN and best practices

Motivation

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

- $\mathbf{x}(s) = (x_1(s), \cdots, x_d(s))^T$ is a *feature vector* of state s

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

- $\mathbf{x}(s) = (x_1(s), \cdots, x_d(s))^T$ is a *feature vector* of state s
- Gradient descent using

$$\nabla_w \hat{v}(s, \mathbf{w}) = \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \cdots \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_d} \right)^T$$

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

- $\mathbf{x}(s) = (x_1(s), \cdots, x_d(s))^T$ is a *feature vector* of state s
- Gradient descent using

$$\begin{aligned} \nabla_w \hat{v}(s, \mathbf{w}) &= \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \cdots \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_d} \right)^T \\ &= \left(\frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_1}, \cdots \frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_d} \right)^T \end{aligned}$$

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

- $\mathbf{x}(s) = (x_1(s), \cdots, x_d(s))^T$ is a *feature vector* of state s
- Gradient descent using

$$
\begin{aligned}
\nabla_w \hat{v}(s, \mathbf{w}) &= \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \cdots \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_d} \right)^T \\
&= \left(\frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_1}, \cdots \frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_d} \right)^T \\
&= \mathbf{x}(s)
\end{aligned}
$$

Recap: Linear Value Function Approximation

Linear Value Function Approximation: $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i \, x_i(s)$

- $\mathbf{x}(s) = (x_1(s), \cdots, x_d(s))^T$ is a *feature vector* of state s
- Gradient descent using

$$
\begin{aligned}
\nabla_w \hat{v}(s, \mathbf{w}) &= \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \cdots \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_d} \right)^T \\
&= \left(\frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_1}, \cdots \frac{\partial \mathbf{w}^T \mathbf{x}(s)}{\partial w_d} \right)^T \\
&= \mathbf{x}(s)
\end{aligned}
$$

- Gradient update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \mathbf{x}(S_t)$

Problems with Linear Approximator

Linear approximation assumes value function is linear in state features

Problems with Linear Approximator

Linear approximation assumes value function is linear in state features

- Difficult to find state representation that upholds this assumption

Problems with Linear Approximator

Linear approximation assumes value function is linear in state features

- Difficult to find state representation that upholds this assumption
- Difficult to find good state representation for high dimensional data *(e.g. images, time series)*

Problems with Linear Approximator

Linear approximation assumes value function is linear in state features

- Difficult to find state representation that upholds this assumption
- Difficult to find good state representation for high dimensional data *(e.g. images, time series)*
- Simple model might not generalise well

Problems with Linear Approximator

Linear approximation assumes value function is linear in state features

- Difficult to find state representation that upholds this assumption
- Difficult to find good state representation for high dimensional data *(e.g. images, time series)*
- Simple model might not generalise well

We need an alternative model for generalisation!

Deep Learning

Requirements for new model:

Requirements for new model:

- Model parameters can be tuned to minimise TD-error
- Model should be able to represent non-linear functions
- Model should enable better generalisation across different states

Requirements for new model:

- Model parameters can be tuned to minimise TD-error
- Model should be able to represent non-linear functions
- Model should enable better generalisation across different states

$\Rightarrow$ Neural network fits all these requirements

Neural Networks - Inspiration from the Brain

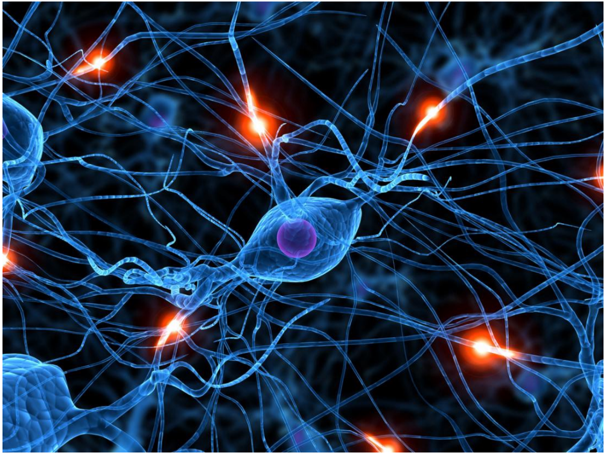
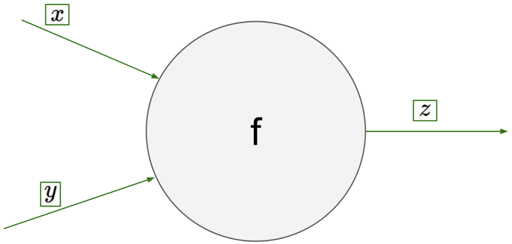


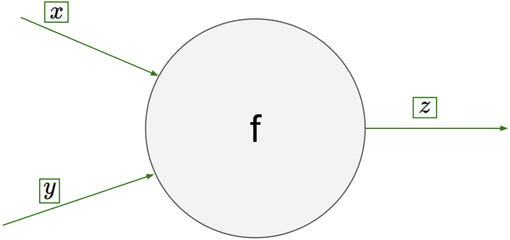
Figure 1: From Artificial Neural Networks — Mapping the Human Brain `https://medium.com/predict/artificial-neural-networks-mapping-the-human-brain-2e0bd4a93160`

Neural Network Units

Basic building block of neural network



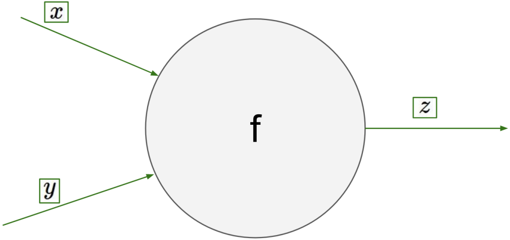
Basic building block of neural network



Computes non-linear function of the inputs in two steps:

Neural Network Units

Basic building block of neural network



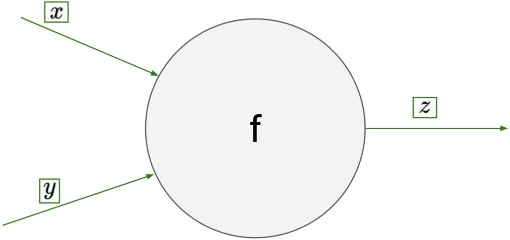
Computes non-linear function of the inputs in two steps:

1. Compute linear transformation of the inputs parameterised by θ

$$o = \theta_1 x + \theta_2 y$$

Neural Network Units

Basic building block of neural network



Computes non-linear function of the inputs in two steps:

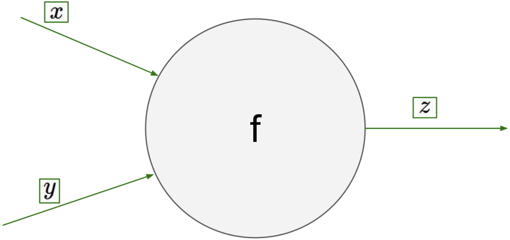
1. Compute linear transformation of the inputs parameterised by θ

$$o = \theta_1 x + \theta_2 y$$

2. Pass output from linear transform as input to non-linear activation function f

$$z = f(o)$$

Basic building block of neural network



Linear approximation is special case: $f(o) = o$

Computes non-linear function of the inputs in two steps:

1. Compute linear transformation of the inputs parameterised by θ

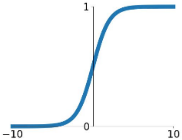
$$o = \theta_1 x + \theta_2 y$$

2. Pass output from linear transform as input to non-linear activation function f

$$z = f(o)$$

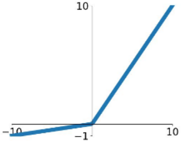
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



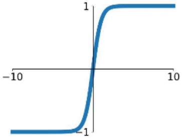
Leaky ReLU

$$\max(0.1x, x)$$



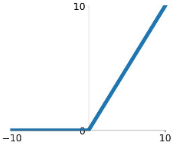
tanh

$$\tanh(x)$$



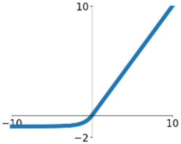
ReLU

$$\max(0, x)$$



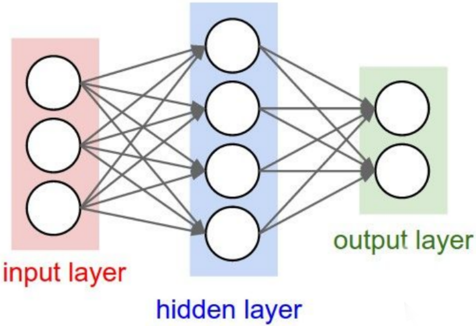
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

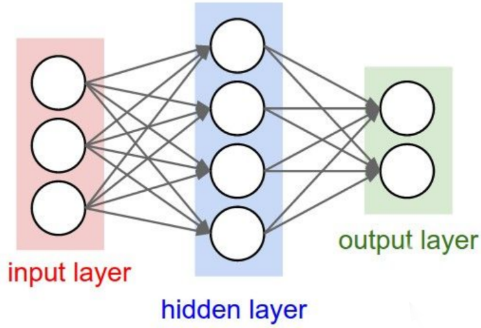


http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture04.pdf

Multi-Layer Perceptron

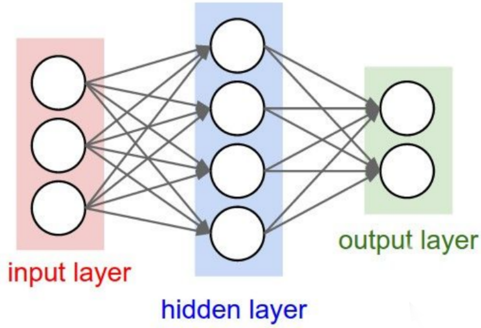


Multi-Layer Perceptron



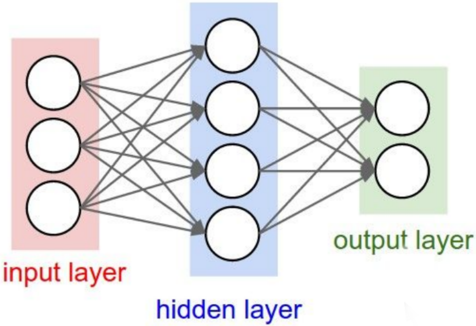
- Arrange multiple units into layers
- Outputs from one layer used as input to next layer

Multi-Layer Perceptron



- Arrange multiple units into layers
- Outputs from one layer used as input to next layer
- Layers can use different activation functions

Multi-Layer Perceptron



- Arrange multiple units into layers
- Outputs from one layer used as input to next layer
- Layers can use different activation functions
- Formulate a loss function of the output
- Adjust network parameters θ to minimise the loss

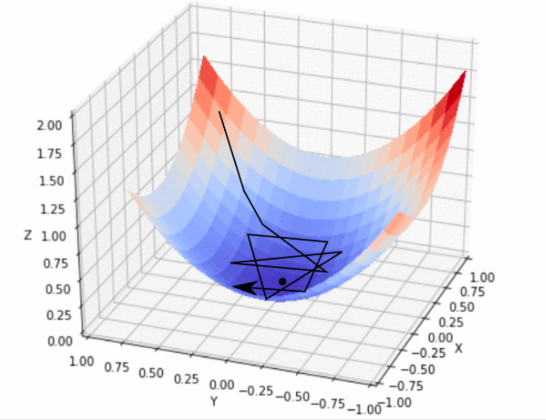
Recap: Stochastic Gradient Descent

- Numerical optimisation method using the gradients of the loss L

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L$$

- Numerical optimisation method using the gradients of the loss L

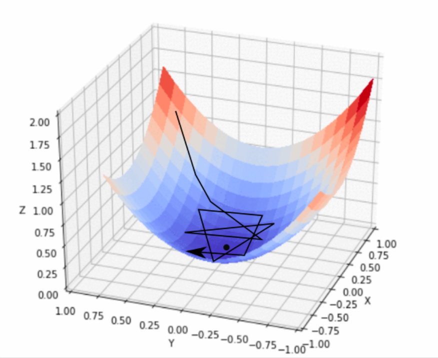
$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L$$



Recap: Stochastic Gradient Descent

- Numerical optimisation method using the gradients of the loss L

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L$$

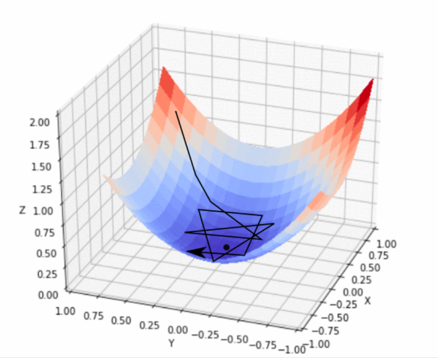


- $\nabla_{\theta_t} L$ is direction of maximum increase of loss function

Recap: Stochastic Gradient Descent

- Numerical optimisation method using the gradients of the loss L

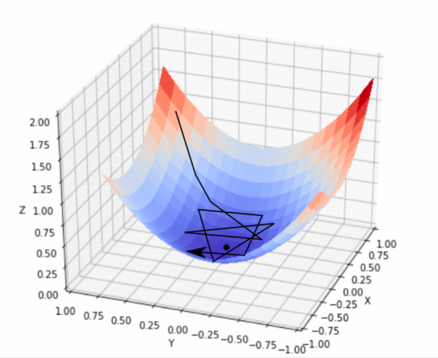
$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L$$



- $\nabla_{\theta_t} L$ is direction of maximum increase of loss function
  - Follow the direction that minimises the function $(-\nabla_{\theta_t} L)$

- Numerical optimisation method using the gradients of the loss L

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L$$



- $\nabla_{\theta_t} L$ is direction of maximum increase of loss function
    - Follow the direction that minimises the function $(-\nabla_{\theta_t} L)$
    - Converges to local optimum under standard α-reduction

Backpropagation

We need gradients $\nabla_{\theta_t} f(x; \theta_t)$ to compute $\nabla_{\theta_t} L$

$\Rightarrow$ How to compute $\nabla_{\theta_t} f(x; \theta_t)$ when f is represented as neural network?

Backpropagation

We need gradients $\nabla_{\theta_t} f(x; \theta_t)$ to compute $\nabla_{\theta_t} L$

$\Rightarrow$ How to compute $\nabla_{\theta_t} f(x; \theta_t)$ when f is represented as neural network?

Backpropagation

Backpropagation

We need gradients $\nabla_{\theta_t} f(x; \theta_t)$ to compute $\nabla_{\theta_t} L$

$\Rightarrow$ How to compute $\nabla_{\theta_t} f(x; \theta_t)$ when f is represented as neural network?

Backpropagation

- Efficiently calculates the gradient in a single backward pass of the neural network

We need gradients $\nabla_{\theta_t} f(x; \theta_t)$ to compute $\nabla_{\theta_t} L$

$\Rightarrow$ How to compute $\nabla_{\theta_t} f(x; \theta_t)$ when f is represented as neural network?

Backpropagation

- Efficiently calculates the gradient in a single backward pass of the neural network
- Automated differentiation packages have commands to quickly backpropagate

  $\Rightarrow$ In Pytorch, use **backward()**

Backpropagation

We need gradients $\nabla_{\theta_t} f(x; \theta_t)$ to compute $\nabla_{\theta_t} L$

$\Rightarrow$ How to compute $\nabla_{\theta_t} f(x; \theta_t)$ when f is represented as neural network?

Backpropagation

- Efficiently calculates the gradient in a single backward pass of the neural network
- Automated differentiation packages have commands to quickly backpropagate

  $\Rightarrow$ In Pytorch, use **backward()**

We won't discuss details of backpropagation algorithm here;
see *Deep Learning* book by Goodfellow et al. or MLPR notes for more details

Deep Reinforcement Learning

Naive Deep Reinforcement Learning

- Tabular RL is unable to scale to large state or action spaces

Naive Deep Reinforcement Learning

- Tabular RL is unable to scale to large state or action spaces
- Discretisation is required for continuous state spaces

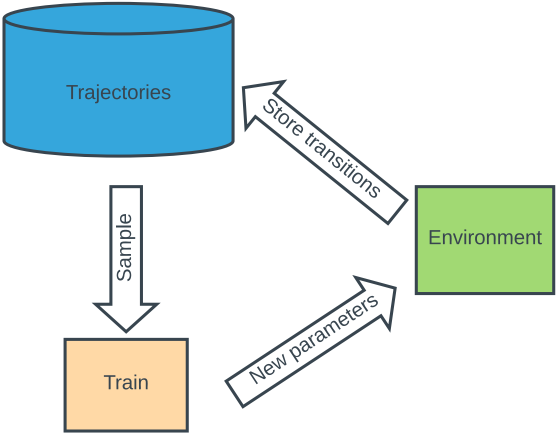
Naive Deep Reinforcement Learning

- Tabular RL is unable to scale to large state or action spaces
- Discretisation is required for continuous state spaces
- Naive replacement of linear model with neural network is problematic:
  - High correlation between consecutive experiences
  - Moving target values in TD methods

Problem: Correlation of Consecutive Experiences

Problem: Correlation of Consecutive Experiences

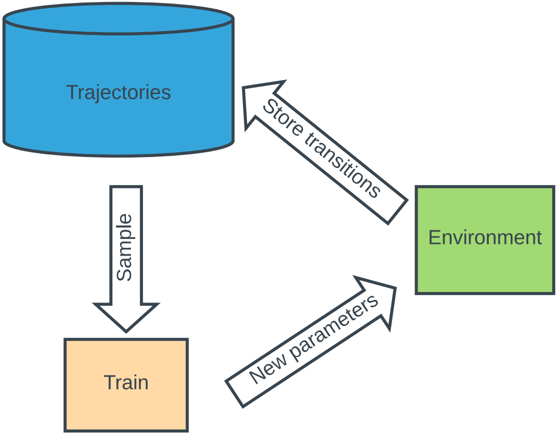
Solution Idea: Experience Replay



Solution Idea: Experience Replay

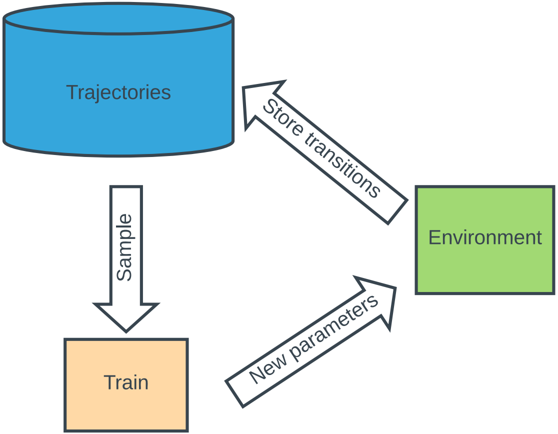
Replay buffer:

- Store most recent experience tuples (s, a, r, s') in FIFO buffer D



Replay buffer:

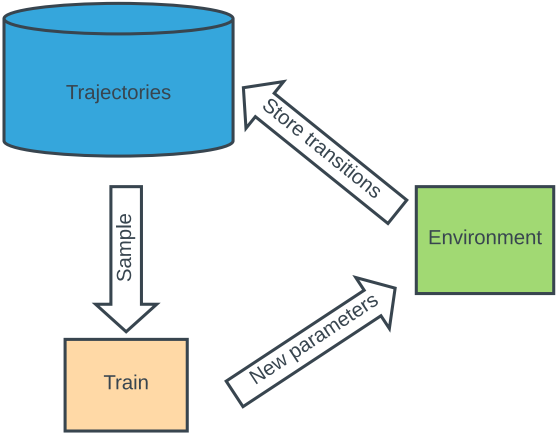
- Store most recent experience tuples (s, a, r, s') in FIFO buffer D
- Create training batches by uniformly sampling from buffer
- Random sampling “breaks” correlation between experiences



Solution Idea: Experience Replay

Replay buffer:

- Store most recent experience tuples (s, a, r, s') in FIFO buffer D
- Create training batches by uniformly sampling from buffer
- Random sampling "breaks" correlation between experiences
- Loss defined over batch:



$$L(\theta_t) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_t) - Q(s, a; \theta_t) \right)^2 \right]$$

Problem: Moving Targets

Target values computed through value function

$$r + \gamma \max_a Q(s', a; \theta)$$

Problem: Moving Targets

Target values computed through value function

$$r + \gamma \max_a Q(s', a; \theta)$$

$\Rightarrow$ Target values *change* each time value function is modified

Problem: Moving Targets

Target values computed through value function

$$r + \gamma \max_a Q(s', a; \theta)$$

$\Rightarrow$ Target values *change* each time value function is modified

- Non-stationarity makes learning optimal θ more difficult
- Require a way to make target values change less frequently

Solution Idea: Target Networks

Use two sets of network parameters:

- θ for *value network* $Q(s, a; \theta)$

Solution Idea: Target Networks

Use two sets of network parameters:

- θ for *value network* $Q(s, a; \theta)$
- θ^- for *target network* $Q(s, a; \theta^-)$

Solution Idea: Target Networks

Use two sets of network parameters:

- θ for *value network* $Q(s, a; \theta)$
- θ^- for *target network* $Q(s, a; \theta^-)$

Use target network to compute update targets: $r + \gamma \max_{a'} Q(s', a'; \theta^-)$

Solution Idea: Target Networks

Use two sets of network parameters:

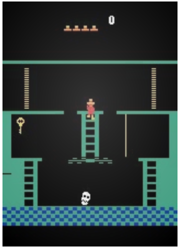
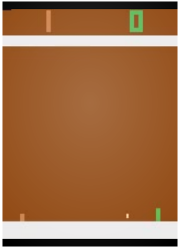
- θ for *value network* $Q(s, a; \theta)$
- θ^- for *target network* $Q(s, a; \theta^-)$

Use target network to compute update targets: $r + \gamma \max_{a'} Q(s', a'; \theta^-)$

Change target network more slowly than value network:

- *hard update:* set $\theta^- \leftarrow \theta$ every C time steps
- or *soft update:* at each time step, move parameters slightly closer to the value network: $\theta^- \leftarrow (1 - \tau)\theta^- + \tau\theta$

Deep Q-Networks [Mnih *et al.*, 2015]

- Use replay buffer and target networks

  $\Rightarrow$ First successful application of deep neural networks to reinforcement learning
- Play Atari games beyond human level

Deep Q-Networks [Mnih *et al.*, 2015]

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Replay buffer $\rightarrow$ Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Hard update $\rightarrow$ Every C steps reset $\hat{Q} = Q$

Markov Property:

$$\Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \ldots, S_0, A_0\} = \Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t\}$$

Given below state of Breakout, does Markov property hold?

State Pre-Processing

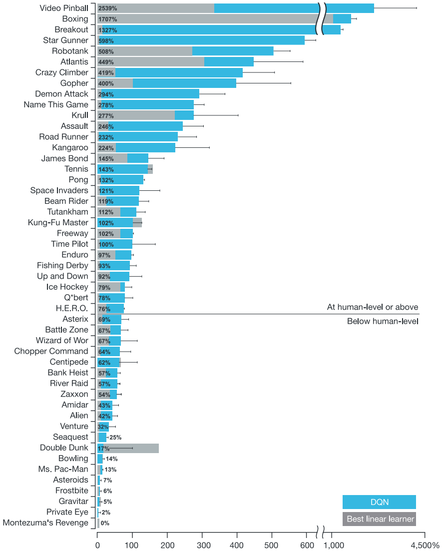
Markov Property:

$$\Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, ..., S_0, A_0\} = \Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t\}$$

Given below state of Breakout, does Markov property hold?

No $\rightarrow$ Use as state the last 4 observations (frames) in order to model the velocity of the ball

DQN Results [Mnih *et al.*, 2015]

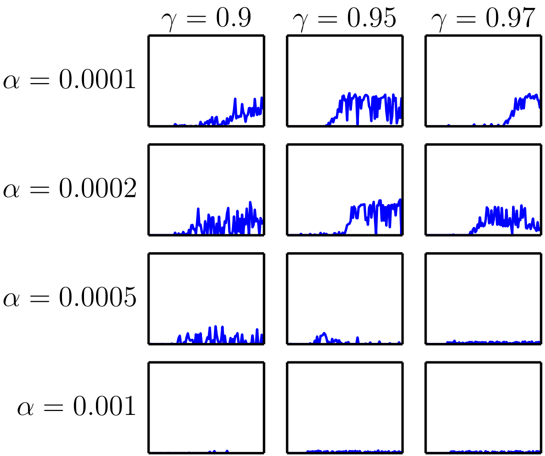


- Exceeded human level performance in most of the Atari games
- Fails in games with very sparse rewards, like Montezuma's revenge

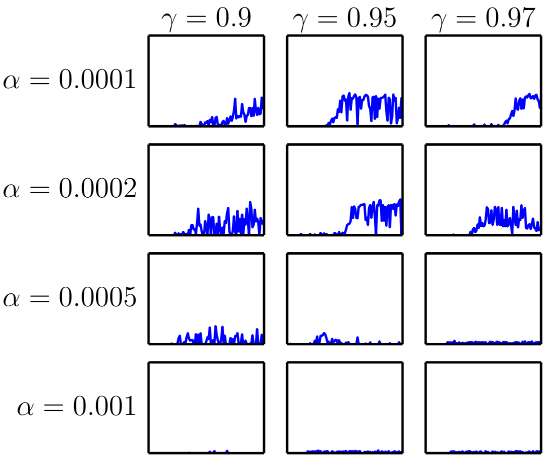
Limitations of DQN

- No convergence guarantees in theory
- Sensitive to hyperparameters
- Only for discrete action space

DQN Hyperparameter Sensitivity in Enduro [Sprague, 2015]

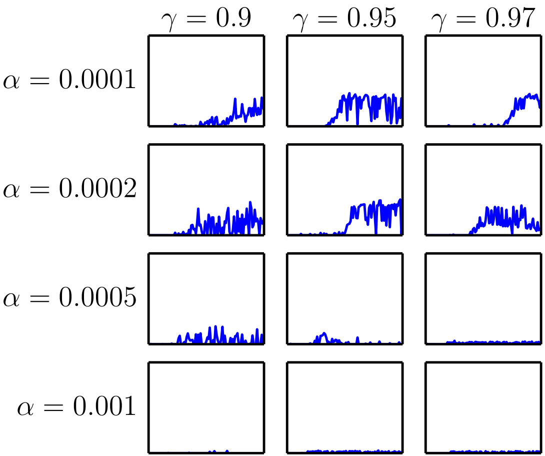


DQN Hyperparameter Sensitivity in Enduro [Sprague, 2015]



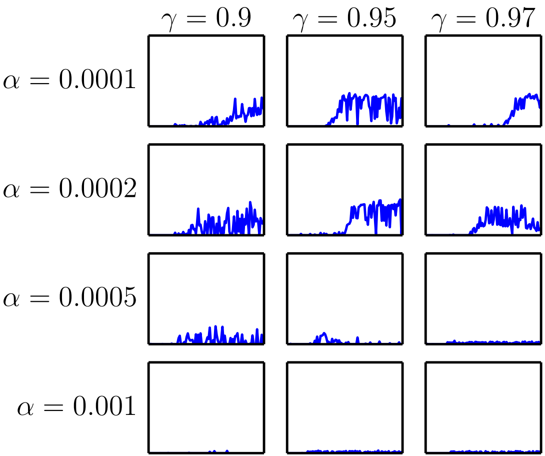
- The graphs display rewards of an agent trained using various hyperparameters in the Enduro game

DQN Hyperparameter Sensitivity in Enduro [Sprague, 2015]



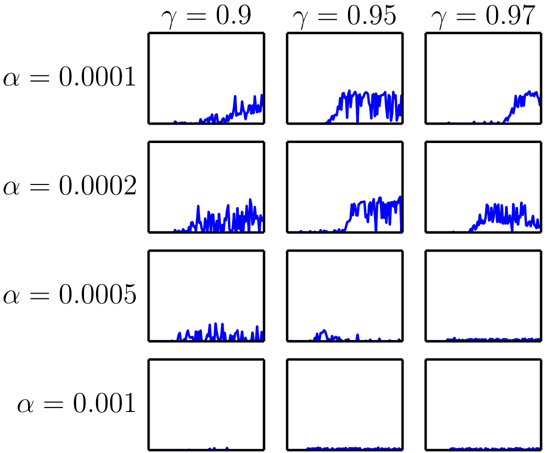
- The graphs display rewards of an agent trained using various hyperparameters in the Enduro game
- Small learning rates might cause agents to learn slowly

DQN Hyperparameter Sensitivity in Enduro [Sprague, 2015]



- The graphs display rewards of an agent trained using various hyperparameters in the Enduro game
- Small learning rates might cause agents to learn slowly
- Large learning rates might cause value networks to diverge

DQN Hyperparameter Sensitivity in Enduro [Sprague, 2015]



- The graphs display rewards of an agent trained using various hyperparameters in the Enduro game
- Small learning rates might cause agents to learn slowly
- Large learning rates might cause value networks to diverge
- Performance from different runs with the same parameters can vary widely

Problem: Value Network Overestimation [Van Hasselt *et al.*, 2016]

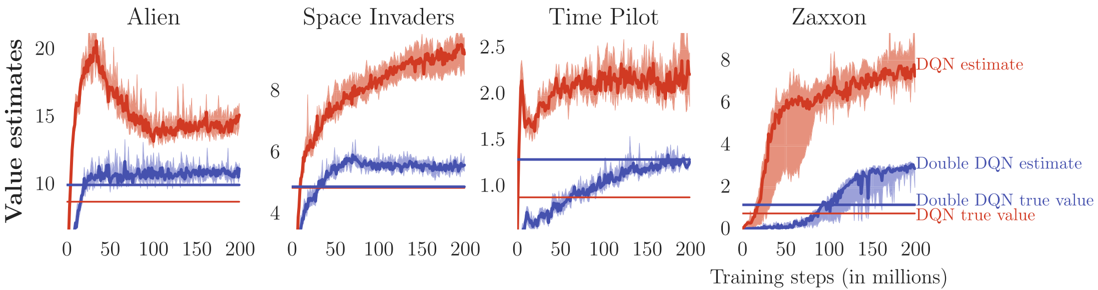
- Q-network tends to overestimate the true value of the agent
- Update target: $r + \gamma \max_a Q(s', a; \theta)$

Problem: Value Network Overestimation [Van Hasselt *et al.*, 2016]

- Q-network tends to overestimate the true value of the agent
- Update target: $r + \gamma \max_a Q(s', a; \theta)$
- Considering two random variables: $\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$
- $\max_a Q(s', a; \theta)$ overestimates the next value

Problem: Value Network Overestimation [Van Hasselt *et al.*, 2016]

- Q-network tends to overestimate the true value of the agent
- Update target: $r + \gamma \max_a Q(s', a; \theta)$
- Considering two random variables: $\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$
- $\max_a Q(s', a; \theta)$ overestimates the next value



Solution Idea: Deep Double Q-Network

- Calculate the action that maximises the value network at next state
  $a_{next} \leftarrow \arg\max_a Q(s', a; \theta)$
- Use this action as input to the target network along with the next state representation

$$y = r + \gamma Q(s', a_{next}; \theta^-)$$

Solution Idea: Deep Double Q-Network

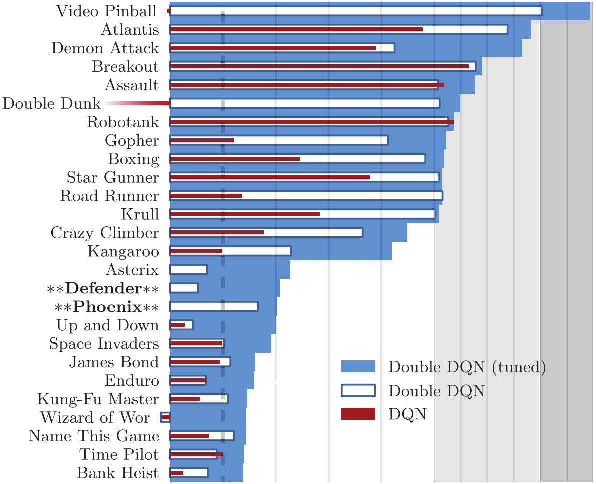
- Calculate the action that maximises the value network at next state
  $a_{next} \leftarrow \arg\max_a Q(s', a; \theta)$
- Use this action as input to the target network along with the next state representation

$$y = r + \gamma Q(s', a_{next}; \theta^-)$$

Deep Double Q-Network:

$$y = r + \gamma Q(s', \arg\max_a Q(s', a; \theta); \theta^-)$$

Deep Double Q-Network Results [Van Hasselt *et al.*, 2016]



- Outperformed DQN
- More accurate prediction of values compared to DQN
- Still failed in more difficult games, like Montezuma's revenge

Prioritised Experience Replay

- Does uniformly sampling experience replay provide the best performance?

Prioritised Experience Replay

- Does uniformly sampling experience replay provide the best performance?
- Not all samples have equal importance

Prioritised Experience Replay

- Does uniformly sampling experience replay provide the best performance?
- **Not all samples have equal importance**
- Like prioritised sweeping in DP, choosing samples based on the TD-error can improve performance

Prioritised Experience Replay

- Does uniformly sampling experience replay provide the best performance?
- **Not all samples have equal importance**
- Like prioritised sweeping in DP, choosing samples based on the TD-error can improve performance

  $\Rightarrow$ Proportionally prioritise experiences with higher TD-error and correct updates based on importance sampling ratio

Prioritised Experience Replay

- Does uniformly sampling experience replay provide the best performance?
- **Not all samples have equal importance**
- Like prioritised sweeping in DP, choosing samples based on the TD-error can improve performance

  $\Rightarrow$ Proportionally prioritise experiences with higher TD-error and correct updates based on importance sampling ratio

- But prioritisation requires additional computation for each insertion, update and removal

  $\Rightarrow$ Difficult to implement efficiently

Best Practices for Implementing Deep Q-Networks

- Carefully consider the storage required for your experience
- Explore aggressively in the beginning
- Decrease ϵ and α as learning progresses
- Periodically store your neural network parameters
- Periodically store contents of your experience replay
- Ensure that your gradients do not explode or vanish

Reading (Optional)

More on neural networks and backprop:

- Section 9.7 in RL book
- Book *Deep Learning* by Ian Goodfellow, Yoshua Bengio, Aaron Courville

  Free online: `https://www.deeplearningbook.org`
- MLPR course notes on
  - Neural networks introduction
  - Fitting and initializing neural networks
  - Backpropagation of Derivatives

Reading (Optional)

Papers:

- Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. ”Human-level control through deep reinforcement learning.” Nature 518, no. 7540 (2015): 529
- Van Hasselt, Hado, Arthur Guez, and David Silver. ”Deep Reinforcement Learning with Double Q-Learning.” In AAAI, vol. 2, p. 5. 2016
- Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. ”Prioritized experience replay.” arXiv preprint arXiv:1511.05952 (2015)