Deep Value

CISC 7404 - Decision Making

Steven Morad

University of Macau

Admin

Admin

I noticed the last 1 hour of class everyone looks tired and sad

Three hours is a long time to pay attention, especially at night

Would you prefer:

- To have a long break (30 mins) in the middle?
- No breaks, end lecture after 2 or 2.5 hours
- Keep as-is (approximately 3 hours + 10 min break)

Admin

HW1 bug:

There was a bug in the `update_Q_TD0` starter code, thanks He Zhe!

Before:

```
terminateds = jnp.concatenate([jnp.zeros(states.shape[0],
dtype=bool), jnp.array([1], dtype=bool)])
```

After:

```
terminateds = jnp.concatenate([jnp.zeros(states.shape[0] - 1,
dtype=bool), jnp.array([1], dtype=bool)])
```

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

$$\eta = Q_i(s_0, a_0, \theta_\pi) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q_i(s_1, a, \theta_\pi) \right)$$

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

Predicted value

$$\eta = \boxed{Q_i(s_0, a_0, \theta_\pi)} - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q_i(s_1, a, \theta_\pi) \right)$$

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

Predicted value

$$\eta = Q_i(s_0, a_0, \theta_\pi) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q_i(s_1, a, \theta_\pi) \right)$$

Empirical value

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

Predicted value

$$\eta = Q_i(s_0, a_0, \theta_\pi) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q_i(s_1, a, \theta_\pi)\right)$$

Empirical value

小心! If s_1 is a terminal state, future value is 0 ($\neg d = \text{not terminated}$)

Admin

$$Q_{i+1}(s_0, a_0, \theta_\pi) = Q_i(s_0, a_0, \theta_\pi) - \alpha \cdot \eta$$

Predicted value

$$\eta = Q_i(s_0, a_0, \theta_\pi) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q_i(s_1, a, \theta_\pi) \right)$$

Empirical value

小心! If s_1 is a terminal state, future value is 0 ($\neg d = \text{not terminated}$)

Without the $\neg d$ term, takes longer to train!

Admin

I thought about coding deep Q networks in class today

But I realize if I do this, then you will not learn as much

Learning to debug is the #1 skill to succeed in reinforcement learning

Instead, **you** will implement deep Q learning for your second homework

Homework 2:
- Deep Q learning
- Deep policy gradient

Will release after homework 1

Admin

Next quiz in 2-3 weeks

Focus on

- Returns
- Value functions
- Q learning
- Deep Q learning
- Policy gradient

Review

Deep Learning Review

Deep Learning Review

We model neural networks as parameterized functions

$$f : X \times \Theta \mapsto Y$$

Map an input $\boldsymbol{x} \in X$ and parameters $\boldsymbol{\theta} \in \Theta$ to output space Y

$$f(\boldsymbol{x}, \boldsymbol{\theta})$$

Deep Learning Review

Neural networks consist of **artificial neurons**

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x + 1}$$

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x + 1}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i \cdot x_i\right)$$

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x+1}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i \cdot x_i\right)$$

σ is a nonlinearity like sigmoid

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x + 1}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i \cdot x_i\right)$$

σ is a nonlinearity like sigmoid

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

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x + 1}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i \cdot x_i\right)$$

σ is a nonlinearity like sigmoid

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

Or ReLU

Deep Learning Review

Neural networks consist of **artificial neurons**

$$\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{\theta} \in \mathbb{R}^{d_x+1}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i \cdot x_i\right)$$

σ is a nonlinearity like sigmoid

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

Or ReLU

$$\sigma(x) = \max(0, x)$$

Deep Learning Review

We combine individual neurons into a **layer**

Deep Learning Review

We combine individual neurons into a **layer**

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

Deep Learning Review

We combine individual neurons into a **layer**

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

$$\Theta = \mathbb{R}^{d_x + 1}$$

Deep Learning Review

We combine individual neurons into a **layer**

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

$$\Theta = \mathbb{R}^{d_x + 1}$$

d_y neurons (wide):

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}^{d_y}$$

$$\Theta = \mathbb{R}^{(d_x + 1) \times d_y}$$

Deep Learning Review

For a single neuron

Deep Learning Review

For a single neuron

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sigma\left(\sum_{i=0}^{d_x} \theta_i \overline{x}_i\right)$$

Deep Learning Review

For a single neuron

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix} \right) = \sigma \left(\sum_{i=0}^{d_x} \theta_i \overline{x}_i \right)$$

For a wide network

Deep Learning Review

For a single neuron

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sigma\left(\sum_{i=0}^{d_x} \theta_i \overline{x}_i\right)$$

For a wide network

$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_{0,1} & \theta_{0,2} & \cdots & \theta_{0,d_y} \\ \theta_{1,1} & \theta_{1,2} & \cdots & \theta_{1,d_y} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d_x,1} & \theta_{d_x,2} & \cdots & \theta_{d_x,d_y} \end{bmatrix}\right) = \begin{bmatrix} \sigma\left(\sum_{i=0}^{d_x} \theta_{i,1} \overline{x}_i\right) \\ \sigma\left(\sum_{i=0}^{d_x} \theta_{i,2} \overline{x}_i\right) \\ \vdots \\ \sigma\left(\sum_{i=0}^{d_x} \theta_{i,d_y} \overline{x}_i\right) \end{bmatrix}$$

Deep Learning Review

We can combine layers to create a **deep** neural network

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}})$$

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad ...$$

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{x}})$$

Deep Learning Review

We can combine layers to create a **deep** neural network

A wide network:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

A deep network has many internal functions

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{x}})$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = f_\ell(\dots f_2(f_1(\boldsymbol{x}, \boldsymbol{\varphi}), \boldsymbol{\psi})\dots \boldsymbol{\xi})$$

Deep Learning Review

Written another way

Deep Learning Review

Written another way

$$z_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

Deep Learning Review

Written another way

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{z}}_1)$$

Deep Learning Review

Written another way

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{z}}_1)$$

$$\vdots$$

Deep Learning Review

Written another way

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{z}}_1)$$

$$\vdots$$

$$\boldsymbol{y} = f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1})$$

Deep Learning Review

Written another way

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\boldsymbol{1}}, \psi) = \sigma(\psi^\top \overline{\boldsymbol{z}}_1)$$

$$\vdots$$

$$\boldsymbol{y} = f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1})$$

We call each function a **layer**

Deep Learning Review

Written another way

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}})$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{z}}_1)$$

$$\vdots$$

$$\boldsymbol{y} = f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1})$$

We call each function a **layer**

A deep neural network is made of many layers

Deep Learning Review

We can create different models for different tasks

Deep Learning Review

We can create different models for different tasks

Standard tasks:

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Image tasks:

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Image tasks: Convolutional neural network (CNN)

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Image tasks: Convolutional neural network (CNN)

Temporal tasks:

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Image tasks: Convolutional neural network (CNN)

Temporal tasks: Recurrent neural network (RNN)

Deep Learning Review

We can create different models for different tasks

Standard tasks: Multi-layer perceptron (MLP)

Image tasks: Convolutional neural network (CNN)

Temporal tasks: Recurrent neural network (RNN)

Image, temporal tasks: Transformer

Deep Learning Review

What functions can we represent using deep neural networks?

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

It can approximate **any** continuous function $g(x)$ to precision η

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

It can approximate **any** continuous function $g(x)$ to precision η

$$| \, g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \, | < \eta$$

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

It can approximate **any** continuous function $g(x)$ to precision η

$$| \ g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \ | < \eta$$

Making the network deeper or wider decreases η

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

It can approximate **any** continuous function $g(x)$ to precision η

$$| \; g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \; | < \eta$$

Making the network deeper or wider decreases η

<u>Very powerful finding! The basis of deep learning.</u>

Deep Learning Review

What functions can we represent using deep neural networks?

A deep neural network is a **universal function approximator**

It can approximate **any** continuous function $g(x)$ to precision η

$$| \; g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \; | < \eta$$

Making the network deeper or wider decreases η

<u>Very powerful finding! The basis of deep learning.</u>

Although such $\boldsymbol{\theta}$ exists, it can be hard to find

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

In particular, we optimize a **loss function**

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

In particular, we optimize a **loss function**

$$\mathcal{L} : X \times Y \times \Theta \mapsto \mathbb{R}$$

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

In particular, we optimize a **loss function**

$$\mathcal{L} : X \times Y \times \Theta \mapsto \mathbb{R}$$

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

In particular, we optimize a **loss function**

$$\mathcal{L} : X \times Y \times \Theta \mapsto \mathbb{R}$$

$$\arg \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

Loss function measures the error between $f(\boldsymbol{x}, \boldsymbol{\theta})$ and desired $g(\boldsymbol{x}) = \boldsymbol{y}$

Deep Learning Review

Finding $\boldsymbol{\theta}$ is an optimization problem

In particular, we optimize a **loss function**

$$\mathcal{L} : X \times Y \times \Theta \mapsto \mathbb{R}$$

$$\arg \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

Loss function measures the error between $f(\boldsymbol{x}, \boldsymbol{\theta})$ and desired $g(\boldsymbol{x}) = \boldsymbol{y}$

In this class, we will build loss functions from two error functions

Deep Learning Review

Square error: The squared distance over a dataset of size n

Deep Learning Review

Square error: The squared distance over a dataset of size n

$$\sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - g(\boldsymbol{x})_j \right)^2 = \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Deep Learning Review

Square error: The squared distance over a dataset of size n

$$\sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - g(\boldsymbol{x})_j \right)^2 = \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Cross entropy error: The probabilistic error over a dataset of size n

Deep Learning Review

Square error: The squared distance over a dataset of size n

$$\sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - g(\boldsymbol{x})_j \right)^2 = \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Cross entropy error: The probabilistic error over a dataset of size n

$$-\sum_{i=1}^{n} \sum_{j=1}^{d_y} P\left(g\left(\boldsymbol{x}_{[i]}\right)_j \mid \boldsymbol{x}_{[i]} \right) \log f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j = -\sum_{i=1}^{n} \sum_{j=1}^{d_y} P\left(y_{[i],j} \mid \boldsymbol{x}_{[i]} \right) \log f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j$$

Deep Learning Review

Square error:

$$\sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Deep Learning Review

Square error:

$$\sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Cross entropy error:

$$- \sum_{i=1}^{n} \sum_{j=1}^{d_y} P\left(y_{[i],j} \mid \boldsymbol{x}_{[i]}\right) \log f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j$$

Question: Which one will we use for Q learning?

Answer: Predict a scalar (expected return), so square error (regression)

Deep Learning Review

We can use these errors in a loss function

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Deep Learning Review

We can use these errors in a loss function

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = - \sum_{i=1}^{n} \sum_{j=1}^{d_y} P\left(y_{[i],j} \mid \boldsymbol{x}_{[i]}\right) \log f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j$$

Deep Learning Review

When we train, we search for neural network parameters θ

Deep Learning Review

When we train, we search for neural network parameters θ

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

Deep Learning Review

When we train, we search for neural network parameters θ

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{j=1}^{d_y} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j - y_{[i],j} \right)^2$$

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} - \sum_{i=1}^{n} \sum_{j=1}^{d_y} P\left(y_{[i],j} \mid \boldsymbol{x}_{[i]}\right) \log f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)_j$$

Deep Learning Review

Question: Which search method do we use?

Deep Learning Review

Question: Which search method do we use?

Answer: Gradient-based methods (gradient descent, Adam, etc)

Deep Learning Review

Question: Which search method do we use?

Answer: Gradient-based methods (gradient descent, Adam, etc)

Deep Learning Review

function GRADIENT DESCENT($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α)

  ▷ Randomly initialize parameters

  $\boldsymbol{\theta} \leftarrow \text{Glorot}()$

  for $i \in 1 ... t$ **do**

    ▷ Compute the gradient of the loss

    $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

    ▷ Update the parameters using the negative gradient

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{J}$

  return $\boldsymbol{\theta}$

Deep Learning Review

function GRADIENT DESCENT($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α)

  ▷ Randomly initialize parameters

  $\boldsymbol{\theta} \leftarrow \text{Glorot}()$

  for $i \in 1 \ldots t$ **do**

    ▷ Compute the gradient of the loss

    $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

    ▷ Update the parameters using the negative gradient

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{J}$

  return $\boldsymbol{\theta}$

Gradient descent computes $\nabla \mathcal{L}$ over all $\boldsymbol{X}$

Deep Learning Review

We can put it all together in jax and equinox

```
from jax import random
from equinox import nn

seed = random.key(0)
key, *net_keys= random.split(seed, 4)
net = nn.Sequential([
  nn.Linear(d_x, d_h, key=net_keys[0]),
  nn.Lambda(jax.nn.leaky_relu),
  nn.Linear(d_h, d_h, key=net_keys[1]),
  nn.Lambda(jax.nn.leaky_relu),
  nn.Linear(d_h, d_y, key=net_keys[2]),
])
```

Deep Learning Review

We can extract parameters using `eqx.partition`

```python
import equinox as eqx
# Get all arrays (parameters) in the network
theta, model = eqx.partition(net, eqx.is_array)
# Add one to every parameter
theta = jax.tree.map(theta, lambda x: x + 1)
# Put the new parameters back into network
net = eqx.combine(theta, model)
```

Deep Learning Review

We can define loss functions

```python
import equinox as eqx

def L_square(net, x, y):
  # vmap applies network to batch of data
  prediction = eqx.filter_vmap(net)(x)
  return ((prediction - y) ** 2).mean()

def L_cross_entropy(net, x, y):
  # Net outputs probabilities
  # And y is one-hot, e.g. [0, 0, 1, 0]
  prediction = eqx.filter_vmap(model)(x)
  return -(y * jnp.log(prediction)).sum(-1).mean()
```

Deep Learning Review

```
import optax
import equinox as eqx
opt = optimizer.adam(learning_rate=3e-4)
# Adam needs to track momentum and variance
opt_state = opt.init(eqx.filter(net, eqx.is_array))
# Gradient of loss function is a function
grad_L = eqx.filter_grad(L_square)
# Evaluate grad_L at x, y, theta to find J
J = grad_L(net, x, y)
# Compute parameter update using J (adam)
updates, opt_state = opt.update(
    grads, opt_state, params=eqx.filter(net, eqx.is_array)
)
net = eqx.apply_updates(net, updates) # Update params
```

Deep Learning Review

```python
def train_one_batch(net, dataset):
  x, y = batch
  grads = eqx.filter_grad(L_square)(net, x, y)
  updates, opt_state = opt.update(
    grads, opt_state, params=eqx.filter(net, eqx.is_array)
  )
  net = eqx.apply_updates(net, updates) # Update params
  return net, opt_state

for epoch in range(num_epochs):
  for batch in dataset:
    # Can use eqx.filter_jit(f) for speedup
    net, opt_state = train_one_batch(net, batch, opt_state)
```

Deep Learning Review

Dirty secret of deep learning:

Deep Learning Review

Dirty secret of deep learning: We do not understand deep learning

Biological inspiration, theoretical bounds and mathematical guarantees

Deep Learning Review

Dirty secret of deep learning: We do not understand deep learning

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

Deep Learning Review

Dirty secret of deep learning: We do not understand deep learning

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

No accepted theory for why deep neural networks are so effective

Deep Learning Review

Dirty secret of deep learning: We do not understand deep learning

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

No accepted theory for why deep neural networks are so effective

In modern deep learning, we progress using trial and error

Deep Learning Review

Dirty secret of deep learning: We do not understand deep learning

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

No accepted theory for why deep neural networks are so effective

In modern deep learning, we progress using trial and error

Today we experiment, and maybe tomorrow we discover the theory

Deep Q Learning

Deep Q Learning

After the homework and last lecture, you are experts in Q learning

Deep Q Learning

After the homework and last lecture, you are experts in Q learning

You quickly trained a Q function to solve a task

Deep Q Learning

After the homework and last lecture, you are experts in Q learning

You quickly trained a Q function to solve a task

Why introduce deep learning to Q learning?

Deep Q Learning

After the homework and last lecture, you are experts in Q learning

You quickly trained a Q function to solve a task

Why introduce deep learning to Q learning?

Consider an example problem

Deep Q Learning

Example: Learn a policy to pick up trash and put it in the bin

Deep Q Learning

Example: Learn a policy to pick up trash and put it in the bin

Deep Q Learning

Question: What is S?

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_{+}^{4096}$

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_{+}^{4096}$
- Arm position $[0, 1]^3$

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_{+}^{4096}$
- Arm position $[0, 1]^3$
- Map position $[0, 1]^2$

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_+^{4096}$
- Arm position $[0, 1]^3$
- Map position $[0, 1]^2$
- Trash position $[0, 1]^{2k}$

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_+^{4096}$
- Arm position $[0, 1]^3$
- Map position $[0, 1]^2$
- Trash position $[0, 1]^{2k}$

In the assignment, we store Q value for each state in a matrix

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_{+}^{4096}$
- Arm position $[0, 1]^3$
- Map position $[0, 1]^2$
- Trash position $[0, 1]^{2k}$

In the assignment, we store Q value for each state in a matrix

Question: What is the size of the matrix?

Deep Q Learning

Question: What is S?

- Camera image $\mathbb{R}^{256 \times 256 \times 3}$
- Lidar scan $\mathbb{R}_+^{4096}$
- Arm position $[0, 1]^3$
- Map position $[0, 1]^2$
- Trash position $[0, 1]^{2k}$

In the assignment, we store Q value for each state in a matrix

Question: What is the size of the matrix?

$$S \times A$$

Deep Q Learning

Let us consider a simplification, only have the map position

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Question: What is the size of our Q matrix?

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Question: What is the size of our Q matrix?

Answer: There are infinitely many numbers between 0 and 1

$$0.01, 0.001, 0...1$$

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Question: What is the size of our Q matrix?

Answer: There are infinitely many numbers between 0 and 1

$$0.01, 0.001, 0...1$$

The state space is infinite, our Q value matrix is infinitely big

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Question: What is the size of our Q matrix?

Answer: There are infinitely many numbers between 0 and 1

$$0.01, 0.001, 0...1$$

The state space is infinite, our Q value matrix is infinitely big

Question: What can we do (besides neural networks)?

Deep Q Learning

Let us consider a simplification, only have the map position

$$S \in [0, 1]^2$$

Question: What is the size of our Q matrix?

Answer: There are infinitely many numbers between 0 and 1

$$0.01, 0.001, 0...1$$

The state space is infinite, our Q value matrix is infinitely big

Question: What can we do (besides neural networks)?

Answer: Discretize the space

Deep Q Learning

$$S \in [0, 1]^2$$

Deep Q Learning

$$S \in [0, 1]^2$$

Discretize to 128×128 grid squares

Deep Q Learning

$$S \in [0, 1]^2$$

Discretize to 128×128 grid squares

$$S \in \{1, ..., 128\}^2$$

Deep Q Learning

$$S \in [0, 1]^2$$

Discretize to 128×128 grid squares

$$S \in \{1, ..., 128\}^2$$

Question: What is the size of the Q matrix?

Deep Q Learning

$$S \in [0, 1]^2$$

Discretize to 128×128 grid squares

$$S \in \{1, ..., 128\}^2$$

Question: What is the size of the Q matrix?

$$16384 \times A$$

Deep Q Learning

$$S \in [0, 1]^2$$

Discretize to 128×128 grid squares

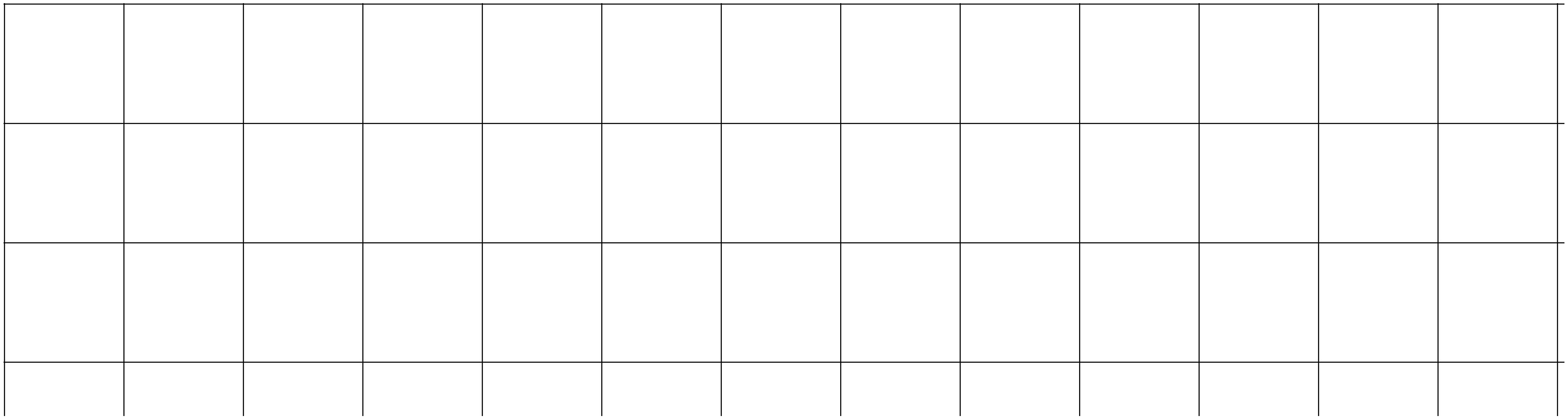
$$S \in \{1, ..., 128\}^2$$

Question: What is the size of the Q matrix?

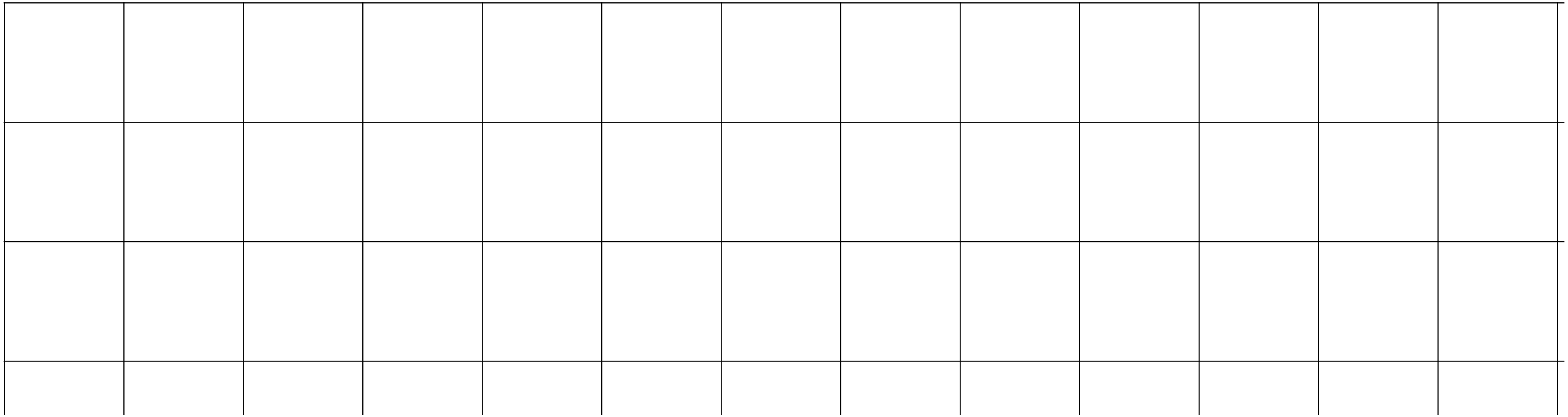
$$16384 \times A$$

Very large but not infinite

Deep Q Learning

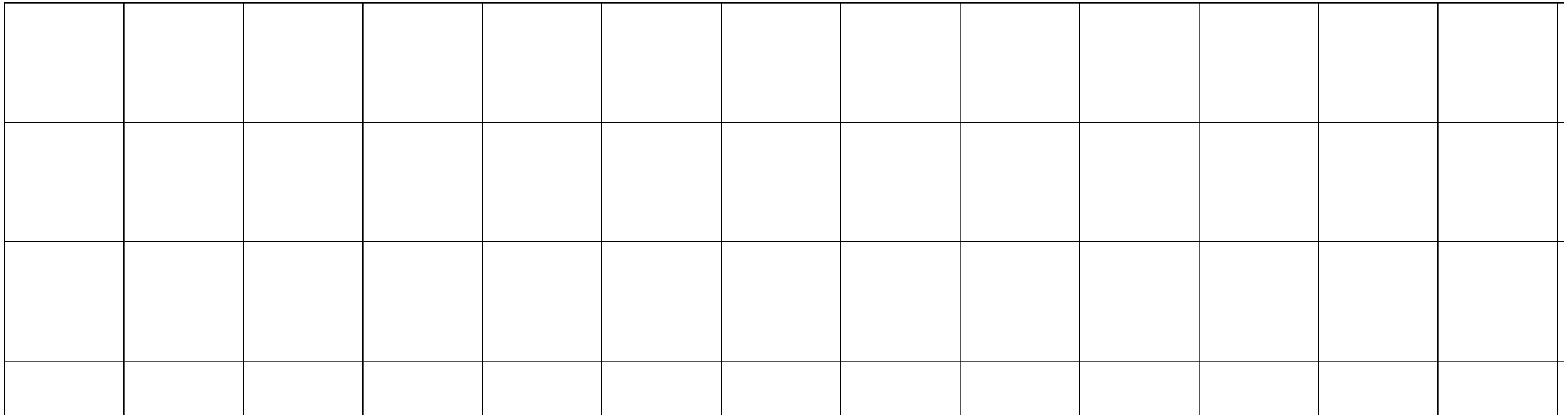


Deep Q Learning



We must update Q for each s, a separately

Deep Q Learning



We must update Q for each s, a separately

With TD updates, updating one cell means we must update all cells

Deep Q Learning

With TD updates, updating one cell means we must update all cells

Deep Q Learning

With TD updates, updating one cell means we must update all cells

Convergence takes at least this many samples

$$\frac{|S| \; |A|}{(1 - \gamma)^5 \cdot \varepsilon^2}$$

Deep Q Learning

With TD updates, updating one cell means we must update all cells

Convergence takes at least this many samples

$$\frac{|S| \; |A|}{(1 - \gamma)^5 \cdot \varepsilon^2}$$

Assume $\gamma = 0.99, \varepsilon = 0.1$

Deep Q Learning

With TD updates, updating one cell means we must update all cells

Convergence takes at least this many samples

$$\frac{|S| \ |A|}{(1 - \gamma)^5 \cdot \varepsilon^2}$$

Assume $\gamma = 0.99, \varepsilon = 0.1$

$$\frac{16348 \mid A \mid}{(0.01)^5 \cdot 0.1^2} \approx 1.6 \times 10^{16}$$

Deep Q Learning

With TD updates, updating one cell means we must update all cells

Convergence takes at least this many samples

$$\frac{|S| \ |A|}{(1 - \gamma)^5 \cdot \varepsilon^2}$$

Assume $\gamma = 0.99, \varepsilon = 0.1$

$$\frac{16348 \mid A \mid}{(0.01)^5 \cdot 0.1^2} \approx 1.6 \times 10^{16}$$

This is the **minimum** number of samples to learn Q

Deep Q Learning

Simple Q learning works very well when the state space is small

Deep Q Learning

Simple Q learning works very well when the state space is small

We need a solution for infinite/continuous/large state spaces

Deep Q Learning

Simple Q learning works very well when the state space is small

We need a solution for infinite/continuous/large state spaces

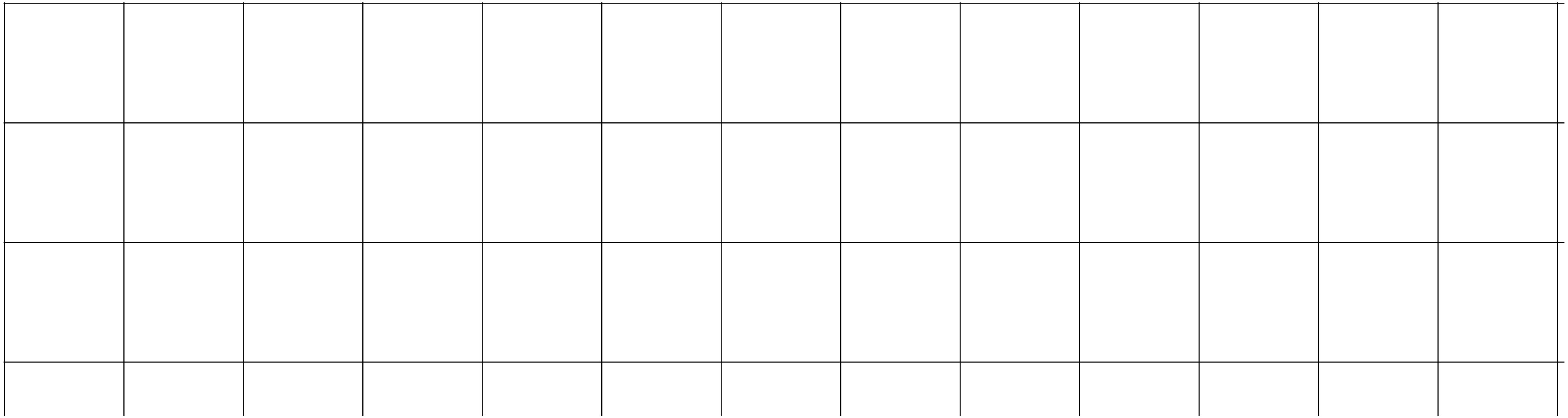
We need a Q function that generalizes to new states

Deep Q Learning

Simple Q learning works very well when the state space is small

We need a solution for infinite/continuous/large state spaces

We need a Q function that generalizes to new states



Deep Q Learning

We know that deep learning generalizes well

Deep Q Learning

We know that deep learning generalizes well

Can approximate any function with a deep neural network

Deep Q Learning

We know that deep learning generalizes well

Can approximate any function with a deep neural network

Represent the Q function using a deep neural network

Deep Q Learning

We know that deep learning generalizes well

Can approximate any function with a deep neural network

Represent the Q function using a deep neural network

Just because parameters exist, does not mean we can find them

Deep Q Learning

We know that deep learning generalizes well

Can approximate any function with a deep neural network

Represent the Q function using a deep neural network

Just because parameters exist, does not mean we can find them

There is **no guarantee** we will find parameters for the Q function

Deep Q Learning

We know that deep learning generalizes well

Can approximate any function with a deep neural network

Represent the Q function using a deep neural network

Just because parameters exist, does not mean we can find them

There is **no guarantee** we will find parameters for the Q function

But we will try!

Deep Q Learning

Deep Q Learning

First, define the Q function as a deep neural network

Deep Q Learning

First, define the Q function as a deep neural network

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Deep Q Learning

First, define the Q function as a deep neural network

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

$$Q(s, a, \theta_\pi, \theta_Q)$$

Deep Q Learning

First, define the Q function as a deep neural network

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

$$Q(s, a, \theta_\pi, \theta_Q)$$

The Q function estimates the policy-conditioned discounted return

Deep Q Learning

$$Q\left(\ s_0, a_0, \theta_\pi, \theta_Q\right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Deep Q Learning

$$Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

Deep Q Learning

$$Q\left(s_0, a_0, \theta_\pi, \theta_Q \right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

Deep Q Learning

$$Q\left(s_0, a_0, \theta_\pi, \theta_Q \right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y}$$

Deep Q Learning

$$\overbrace{Q}^{f}\left(s_0, a_0, \theta_\pi, \theta_Q \right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y}$$

Deep Q Learning

$$\overbrace{Q}^{f}\left(\underbrace{s_0, a_0}_{\boldsymbol{x}}, \theta_\pi, \theta_Q\right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y}$$

Deep Q Learning

$$\overset{f}{Q}\left(\underset{\boldsymbol{x}}{\underbrace{s_0, a_0}}, \theta_\pi, \underset{\boldsymbol{\theta}}{\underbrace{\theta_Q}}\right) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y}$$

Deep Q Learning

$$\overbrace{Q}^{f}\left(\underbrace{s_0, a_0}_{\boldsymbol{x}}, \theta_\pi, \underbrace{\theta_Q}_{\boldsymbol{\theta}}\right) = \underbrace{\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]}_{\boldsymbol{y}}$$

Make this an optimization objective, so we can train a network

But this is different than what we usually see

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y}$$

Deep Q Learning

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Deep Q Learning

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Move everything to left

Deep Q Learning

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Move everything to left

$$Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] = 0$$

Deep Q Learning

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Move everything to left

$$Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] = 0$$

Use a distance measure so we can minimize, choose square error

Deep Q Learning

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$$

Move everything to left

$$Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] = 0$$

Use a distance measure so we can minimize, choose square error

$$(Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi])^2 = 0$$

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

$$\arg\min_{\theta_Q} \left[\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

$$\arg\min_{\theta_Q} \left[\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Question: Missing anything?

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

$$\arg\min_{\theta_Q} \left[\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Question: Missing anything? Hint: Other loss functions have sums?

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

$$\arg\min_{\theta_Q} \left[\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Question: Missing anything? Hint: Other loss functions have sums?

Minimize over all states and actions

Deep Q Learning

$$\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2 = 0$$

Minimize over neural network parameters

$$\arg\min_{\theta_Q} \left[\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Question: Missing anything? Hint: Other loss functions have sums?

Minimize over all states and actions

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]\right)^2\right]$$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Need to replace $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Need to replace $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$

Question: What are the two methods to compute $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$?

Deep Q Learning

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Need to replace $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$

Question: What are the two methods to compute $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0, a_0; \theta_\pi]$?

Answer: Monte Carlo and Temporal Difference

Deep Q Learning

Monte Carlo Objective:

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Temporal Difference Objective:

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Still have expectations, which we do not know

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Still have expectations, which we do not know

Question: What can we do?

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Still have expectations, which we do not know

Question: What can we do?

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Deep Q Learning

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: Which is harder to optimize?

Deep Q Learning

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: Which is harder to optimize?

Answer: Temporal difference

Deep Q Learning

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: Which is harder to optimize?

Answer: Temporal difference

Deep Q Learning

Rewrite as loss functions to help with implementation

Deep Q Learning

Rewrite as loss functions to help with implementation

The Monte Carlo loss uses an episode of states, actions, and rewards

Deep Q Learning

Rewrite as loss functions to help with implementation

The Monte Carlo loss uses an episode of states, actions, and rewards

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q} \left[\sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

Deep Q Learning

Rewrite as loss functions to help with implementation

The Monte Carlo loss uses an episode of states, actions, and rewards

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q} \left[\sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

Can train over batch or dataset containing many episodes

Deep Q Learning

Rewrite as loss functions to help with implementation

The Monte Carlo loss uses an episode of states, actions, and rewards

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q} \left[\sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

Can train over batch or dataset containing many episodes

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q} \left[\sum_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

Deep Q Learning

Now write TD loss function

Deep Q Learning

Now write TD loss function

$$\arg \min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}_Q) = \arg \min_{\theta_Q}$$

$$\sum_{\substack{s_i, a_i, r_i, s_{i+1} \\ \in \boldsymbol{x}}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \left(r_t + \neg d \gamma \arg \max_{a \in A} Q(s_i, a, \theta_\pi, \theta_Q) \right) \right)^2$$

Deep Q Learning

Now write TD loss function

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q}$$

$$\sum_{\substack{s_i, a_i, r_i, s_{i+1} \\ \in \boldsymbol{x}}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \left(r_t + \neg d \gamma \arg\max_{a \in A} Q(s_i, a, \theta_\pi, \theta_Q) \right) \right)^2$$

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q}$$

$$\sum_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{\substack{s_i, a_i, r_i, s_{i+1} \\ \in \boldsymbol{x}}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \left(r_t + \neg d \gamma \arg\max_{a \in A} Q(s_i, a, \theta_\pi, \theta_Q) \right) \right)^2$$

Deep Q Learning

To summarize, our two loss functions:

Deep Q Learning

To summarize, our two loss functions:

$$\underset{\boldsymbol{\theta}_Q}{\arg\min} \, \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}_Q) = \underset{\theta_Q}{\arg\min} \left[\sum_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

Deep Q Learning

To summarize, our two loss functions:

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q} \left[\sum_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{s_i, a_i, r_i \in \boldsymbol{x}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \sum_{t=i}^{\infty} \gamma^t r_t \right)^2 \right]$$

$$\arg\min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}_Q) = \arg\min_{\theta_Q}$$

$$\sum_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{\substack{s_i, a_i, r_i, s_{i+1} \\ \in \boldsymbol{x}}} \left(Q(s_i, a_i, \theta_\pi, \theta_Q) - \left(r_t + \neg d \gamma \arg\max_{a \in A} Q(s_i, a, \theta_\pi, \theta_Q) \right) \right)^2$$

Deep Q Learning

Can optimize both loss functions using gradient descent

Deep Q Learning

Can optimize both loss functions using gradient descent

RL optimization is more difficult than supervised learning

Deep Q Learning

Can optimize both loss functions using gradient descent

RL optimization is more difficult than supervised learning

Supervised Learning:

- Static inputs
- Static labels
- Limited dataset
  - Human can clean
  - Bad to overfit

Reinforcement Learning:

- Inputs change as θ_π changes
  - Visit new/different states
- Labels change as θ_π changes
  - $\mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid \theta_\pi]$
- Infinite dataset
  - Can always collect from env
  - Bad θ_π means bad dataset
  - Overfitting no problem

Experience Replay

Experience Replay

Optimization is difficult in RL

Experience Replay

Optimization is difficult in RL

Most RL papers train for 10M-10B environment steps

Experience Replay

Optimization is difficult in RL

Most RL papers train for 10M-10B environment steps

It takes a long time to train a deep Q function

Experience Replay

Optimization is difficult in RL

Most RL papers train for 10M-10B environment steps

It takes a long time to train a deep Q function

Let us see if we can improve training speed

Experience Replay

```
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  # Step between 1 and infinity times to get one episode
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  # Compute gradient over episode
  J = grad(L)(theta_Q, episode)
  theta_Q = update(theta_Q, grad)
```

Experience Replay

```
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  # Step between 1 and infinity times to get one episode
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  # Compute gradient over episode
  J = grad(L)(theta_Q, episode)
  theta_Q = update(theta_Q, grad)
```

Question: Which part is slowest?

Experience Replay

```
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  # Step between 1 and infinity times to get one episode
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  # Compute gradient over episode
  J = grad(L)(theta_Q, episode)
  theta_Q = update(theta_Q, grad)
```

Question: Which part is slowest? **Answer:** Collecting episodes

Experience Replay

```
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  # Step between 1 and infinity times to get one episode
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  # Compute gradient over episode
  J = grad(L)(theta_Q, episode)
  theta_Q = update(theta_Q, grad)
```

Experience Replay

```python
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  # Step between 1 and infinity times to get one episode
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  # Compute gradient over episode
  J = grad(L)(theta_Q, episode)
  theta_Q = update(theta_Q, grad)
```

Collect episode, train, throw away episode, start again

Experience Replay

What if we reuse episodes?

Experience Replay

What if we reuse episodes?

```
episodes = []
for epoch in range(num_epochs):
  terminated = False
  s = env.reset()
  episode = []
  while not terminated:
    a = policy(s, theta_Q)
    next_s, r, d = env.step(action)
    episode.append([s, a, r, d, next_s])
  episodes.append(episode)
  J = grad(L)(theta_Q, episodes) # Train over ALL episodes
  theta_Q = update(theta_Q, grad)
```

Experience Replay

When we reuse episodes, we call it **experience replay**

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer**
(list)

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$B_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$B_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Create a dataset from the buffer

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$\boldsymbol{B}_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Create a dataset from the buffer

$$\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{s}_{31} & \boldsymbol{a}_{31} & \boldsymbol{r}_{31} & \boldsymbol{d}_{31} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_4 & \boldsymbol{a}_4 & \boldsymbol{r}_4 & \boldsymbol{d}_4 \end{bmatrix}$$

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$\boldsymbol{B}_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Create a dataset from the buffer

$$\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{s}_{31} & \boldsymbol{a}_{31} & \boldsymbol{r}_{31} & \boldsymbol{d}_{31} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_4 & \boldsymbol{a}_4 & \boldsymbol{r}_4 & \boldsymbol{d}_4 \end{bmatrix}$$

Train on the dataset

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$\boldsymbol{B}_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Create a dataset from the buffer

$$\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{s}_{31} & \boldsymbol{a}_{31} & \boldsymbol{r}_{31} & \boldsymbol{d}_{31} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_4 & \boldsymbol{a}_4 & \boldsymbol{r}_4 & \boldsymbol{d}_4 \end{bmatrix}$$

Train on the dataset

$$\arg \min_{\boldsymbol{\theta}_Q} \mathcal{L}(\boldsymbol{X}_t, \boldsymbol{\theta}_Q)$$

Experience Replay

When we reuse episodes, we call it **experience replay**

Store episodes in a **replay buffer** (list)

$$\boldsymbol{B}_t = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{a}_1 & \boldsymbol{r}_1 & \boldsymbol{d}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_t & \boldsymbol{a}_t & \boldsymbol{r}_t & \boldsymbol{d}_t \end{bmatrix}$$

Create a dataset from the buffer

$$\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{s}_{31} & \boldsymbol{a}_{31} & \boldsymbol{r}_{31} & \boldsymbol{d}_{31} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{s}_4 & \boldsymbol{a}_4 & \boldsymbol{r}_4 & \boldsymbol{d}_4 \end{bmatrix}$$

Train on the dataset

$$\underset{\boldsymbol{\theta}_Q}{\arg\min}\, \mathcal{L}(\boldsymbol{X}_t, \boldsymbol{\theta}_Q)$$

Humans do experience replay when they dream!

Experience Replay

On-policy algorithms must throw away episodes after training

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

- Previous policy

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

- Previous policy
- Previous training run

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

- Previous policy
- Previous training run
- Human policy

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

- Previous policy
- Previous training run
- Human policy

Question: Which is Q learning?

Experience Replay

On-policy algorithms must throw away episodes after training

Must collect data using the current policy, cannot use experience replay

Off-policy algorithms can reuse old episodes and use experience replay

In fact, for off policy algorithms, data can come from anywhere

- Previous policy
- Previous training run
- Human policy

Question: Which is Q learning?

Let us find out!

Experience Replay

Start with the Monte Carlo return

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy?

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** On-policy. Why?

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** On-policy. Why?

Our return is conditioned on the policy

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** On-policy. Why?

Our return is conditioned on the policy

If the policy changes, the return $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ is not valid!

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** On-policy. Why?

Our return is conditioned on the policy

If the policy changes, the return $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ is not valid!

Old episode gives us $\mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_{\text{old}}]$

Experience Replay

Start with the Monte Carlo return

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi] \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** On-policy. Why?

Our return is conditioned on the policy

If the policy changes, the return $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ is not valid!

Old episode gives us $\mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_{\text{old}}]$

We need $\mathbb{E}[\mathcal{R}(s_{t+1}) \mid s_0, a_0; \theta_\pi]$

Experience Replay

What about TD return?

Experience Replay

What about TD return?

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Experience Replay

What about TD return?

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: On-policy or off-policy?

Experience Replay

What about TD return?

$$\underset{\theta_Q}{\arg\min} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** Off-policy. Why?

Experience Replay

What about TD return?

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** Off-policy. Why?

Experience Replay

What about TD return?

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** Off-policy. Why?

Q function depends on θ_π, but reward does not!

Do we know $\arg\max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)$?

Experience Replay

What about TD return?

$$\arg\min_{\theta_Q} \left[\sum_{s_0 \in S} \sum_{a_0 \in A} \right.$$

$$\left. \left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2 \right]$$

Question: On-policy or off-policy? **Answer:** Off-policy. Why?

Q function depends on θ_π, but reward does not!

Do we know $\arg\max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)$? Yes! Just plug in s_1

Experience Replay

To summarize:

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Cannot reuse data, takes a long time to train

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Cannot reuse data, takes a long time to train

Temporal Difference Q learning is special!

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Cannot reuse data, takes a long time to train

Temporal Difference Q learning is special!

It is off-policy, can reuse data and train faster

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Cannot reuse data, takes a long time to train

Temporal Difference Q learning is special!

It is off-policy, can reuse data and train faster

TD is not always better than MC

Experience Replay

To summarize:

Monte Carlo Q learning is on-policy

Cannot reuse data, takes a long time to train

Temporal Difference Q learning is special!

It is off-policy, can reuse data and train faster

TD is not always better than MC

MC needs more training data, but TD has harder optimization

Target Networks

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Question: Can you see why?

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Question: Can you see why? Hint: What if $s_0 \approx s_1$?

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Question: Can you see why? Hint: What if $s_0 \approx s_1$?

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = r_0 + \max_{a \in A} Q(s_0, a_0, \theta_\pi, \theta_Q)$$

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Question: Can you see why? Hint: What if $s_0 \approx s_1$?

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = r_0 + \max_{a \in A} Q(s_0, a_0, \theta_\pi, \theta_Q)$$

Question: If $r_0 = 1$, what happens?

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) \right) \right)^2$$

Question: Can you see why? Hint: What if $s_0 \approx s_1$?

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = r_0 + \max_{a \in A} Q(s_0, a_0, \theta_\pi, \theta_Q)$$

Question: If $r_0 = 1$, what happens?

$$Q_{i+1} = 1 + Q_i$$

Target Networks

If you train a deep Q network using TD, you will find

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = \infty$$

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Question: Can you see why? Hint: What if $s_0 \approx s_1$?

$$Q(s_0, a_0, \theta_\pi, \theta_Q) = r_0 + \max_{a \in A} Q(s_0, a_0, \theta_\pi, \theta_Q)$$

Question: If $r_0 = 1$, what happens?

$$Q_{i+1} = 1 + Q_i \qquad \lim_{i \to \infty} ?$$

Target Networks

It is difficult to train deep neural networks recursively

Target Networks

It is difficult to train deep neural networks recursively

The label depends on the function we train!

Target Networks

It is difficult to train deep neural networks recursively

The label depends on the function we train!

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

Target Networks

It is difficult to train deep neural networks recursively

The label depends on the function we train!

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

We use **target networks** to break this dependence

Target Networks

It is difficult to train deep neural networks recursively

The label depends on the function we train!

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)\right)\right)^2$$

We use **target networks** to break this dependence

$$\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - \left(\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \neg d \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_T)\right)\right)^2$$

Target Networks

Usually, the target parameters are older parameters

```
theta_Q = ... # Initialize parameters
theta_T = theta_Q.copy()

for epoch in range(num_epochs):
  grad = grad(L)(theta_Q, theta_T, X)
  theta_Q = optimizer.update(theta_Q, grad)
  if epoch % 200 == 0:
    # Update target parameters
    theta_T = theta_Q.copy()
```

Deep Q Networks

Deep Q Networks

Deep reinforcement learning was first discovered in the 1980s

[1] Human-level control through deep reinforcement learning. *Nature.* 2014.

Deep Q Networks

Deep reinforcement learning was first discovered in the 1980s

However, it did not work very well and could only solve simple tasks

[1] Human-level control through deep reinforcement learning. *Nature.* 2014.

Deep Q Networks

Deep reinforcement learning was first discovered in the 1980s

However, it did not work very well and could only solve simple tasks

We discovered deep learning, experience replay, and target networks

Deep Q Networks (DQN) combined them to beat humans on Atari[1]

[1] Human-level control through deep reinforcement learning. *Nature.* 2014.

Deep Q Networks

Deep reinforcement learning was first discovered in the 1980s

However, it did not work very well and could only solve simple tasks

We discovered deep learning, experience replay, and target networks

Deep Q Networks (DQN) combined them to beat humans on Atari[1]

After this paper, people realized that RL can work for hard tasks

[1] Human-level control through deep reinforcement learning. *Nature.* 2014.

Deep Q Networks

Deep reinforcement learning was first discovered in the 1980s

However, it did not work very well and could only solve simple tasks

We discovered deep learning, experience replay, and target networks

Deep Q Networks (DQN) combined them to beat humans on Atari[1]

After this paper, people realized that RL can work for hard tasks

You have all the tools you need to implement DQN, except for one

[1] Human-level control through deep reinforcement learning. *Nature.* 2014.

Deep Q Networks

Normally, the Q function takes action as input

Deep Q Networks

Normally, the Q function takes action as input

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Deep Q Networks

Normally, the Q function takes action as input

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Then, we run Q for all actions

Deep Q Networks

Normally, the Q function takes action as input

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Then, we run Q for all actions

$$a = \arg \max_i \begin{bmatrix} Q(s, a = 1, \theta_\pi, \theta_Q) \\ Q(s, a = 2, \theta_\pi, \theta_Q) \\ \vdots \\ Q(s, a = i, \theta_\pi, \theta_Q) \end{bmatrix}$$

Deep Q Networks

Normally, the Q function takes action as input

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Then, we run Q for all actions

$$a = \arg \max_i \begin{bmatrix} Q(s, a = 1, \theta_\pi, \theta_Q) \\ Q(s, a = 2, \theta_\pi, \theta_Q) \\ \vdots \\ Q(s, a = i, \theta_\pi, \theta_Q) \end{bmatrix}$$

For each action, we must execute Q network $|A|$ times. Not efficient!

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

In DQN, the authors estimate all Q at once

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

In DQN, the authors estimate all Q at once

$$Q : S \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}^{|A|}$$

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

In DQN, the authors estimate all Q at once

$$Q : S \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}^{|A|}$$

The neural network outputs $|A|$ values – one for each action

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

In DQN, the authors estimate all Q at once

$$Q : S \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}^{|A|}$$

The neural network outputs $|A|$ values – one for each action

$$a = \arg\max_i Q(s, \theta_\pi, \theta_Q)_i$$

Deep Q Networks

$$Q : S \times A \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}$$

In DQN, the authors estimate all Q at once

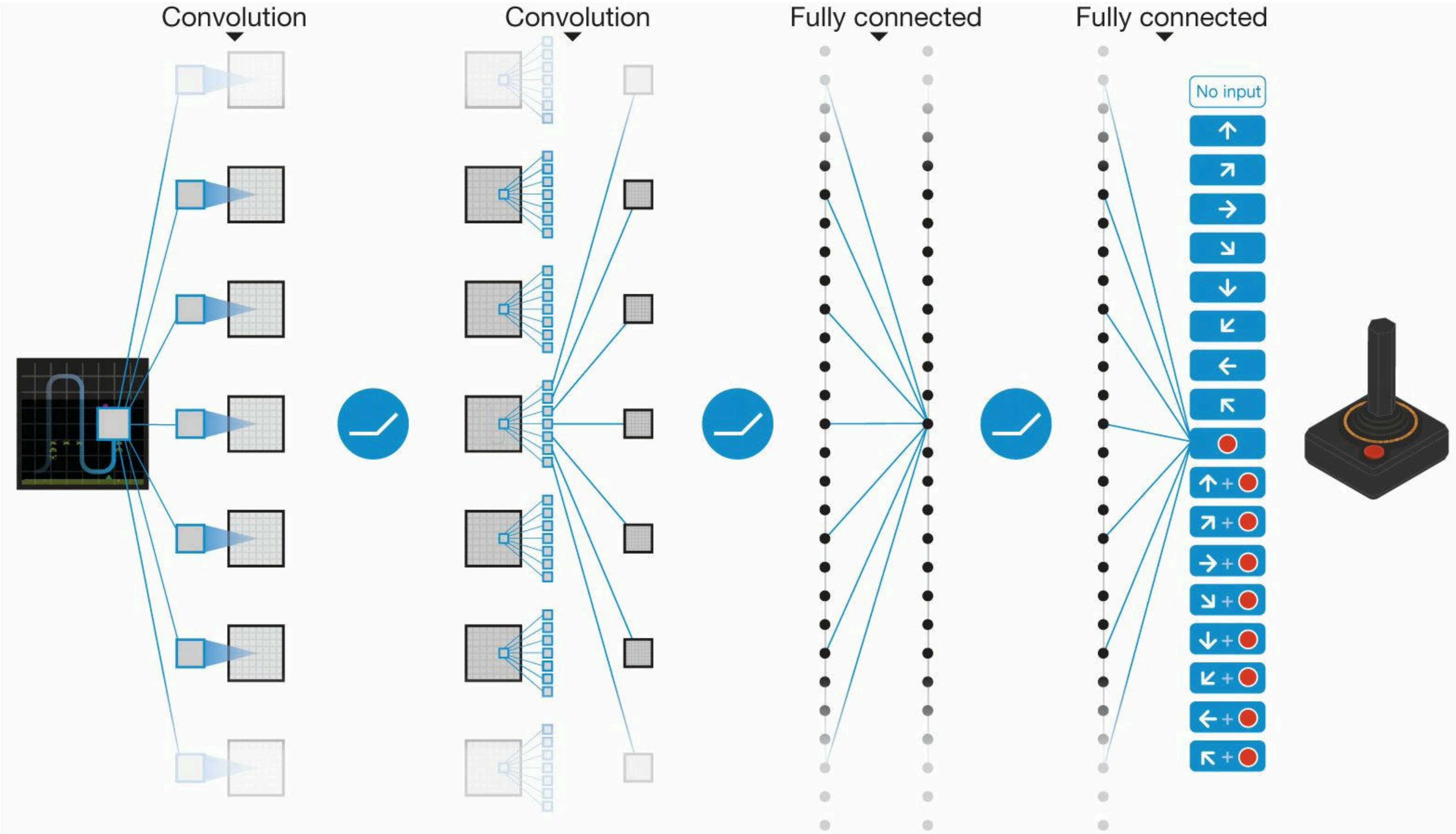
$$Q : S \times \Theta_\pi \times \Theta_Q \mapsto \mathbb{R}^{|A|}$$

The neural network outputs $|A|$ values – one for each action

$$a = \arg \max_i Q(s, \theta_\pi, \theta_Q)_i$$

This is $|A|$ times faster!

Deep Q Networks



Deep Q Networks

```python
Q = nn.Sequential([...])
theta_T = partition(Q, is_array)[0]
replay_buffer = deque(maxsize=50_000)
for epoch in range(num_epochs):
  while not terminated:
    a = random_action if epoch < k else epsilon_greedy(Q)
    s, r, d, next_s = env.step(a)
    replay_buffer.insert((s, a, r, d, next_s))
    X = random.sample(replay_buffer, batch_size)
    theta_Q, model = eqx.partition(Q, is_array)
    theta_Q = td_update(theta_Q, theta_T, Q, X)
    theta_T = copy(theta_Q) if epoch % j == 0 else theta_T
    Q = eqx.combine(theta_Q, model)
```

Deep Q Networks

Finally, let us look at some successes of deep Q learning

Deep Q Networks

Finally, let us look at some successes of deep Q learning

https://huggingface.co/learn/deep-rl-course/en/unit3/hands-on

Deep Q Networks

Finally, let us look at some successes of deep Q learning

https://huggingface.co/learn/deep-rl-course/en/unit3/hands-on

Mario Kart: https://www.youtube.com/watch?v=lnnHmVNO07Q

Deep Q Networks

Finally, let us look at some successes of deep Q learning

https://huggingface.co/learn/deep-rl-course/en/unit3/hands-on

Mario Kart: https://www.youtube.com/watch?v=lnnHmVNO07Q

Super Smash Bros: https://www.youtube.com/watch?v=7rDfIcdszxQ

Pokemon https://youtu.be/DcYLT37ImBY?si=AeR2WkQg4X-tWa5v