Deep Q-Learning (DQN)
EP3

MAC0434 e MAC5788 - Planejamento e Aprendizado por Reforço (2023)

Profa. Leliane Nunes de Barros e Viviane Bonadia (monitora)

Sobre o EP3

- Recaptulando, no EP2 foi implementado o algoritmo Q-Learning tabular

Q-learning para controle off-policy

μ e π são melhoradas a cada iteração

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

política μ

política π

Note que:
- a ação *a* usada para atualizar Q não é a mesma usada para gerar a experiencia (como no SARSA)
- uso implícito da política π para a escolha da próxima ação

Usando ML para transformar Q-learning em DQN

- Mesmo com muita memória disponível, **não é possível armazenar tabelas muito grandes**
- Além disso, é muito lento atualizar o valor de cada estado (ou par estado/ação) individualmente
- Inspirados em técnicas modernas de aprendizado de máquina para outras tarefas, podemos **GENERALIZAR** o que é aprendido em um estado, para outros estados similares (não visitados)!
- Ao invés de usar uma tabela grande para representar **V** ou **Q**, usamos uma **função parametrizada** (garantindo que o número de parâmetros seja bem menor que o número de estados (exponencialmente menor, em geral)
- **GENERALIZAÇÃO**: quando atualizamos os parâmetros com base nas observações em um estado, as funções de avaliação também mudam (generalizam) para outros estados similares
- Assim, conforme visto em sala de aula, uma versão mais eficiente do Q-learning usa uma rede neural para aproximar a função Q(s,a), chamamos essa técnica de *Deep Q-learning* (DQN) e também adicionamos algumas regularizações (melhorias para aumentar sua eficiência) :

DQN+Experience-Replay+mini-batch+Target-Network

DQN: Q-Learning w/ Randomized Experience Replay + mini-batch + Target Network

1. Inicialize *experience replay data set* D
2. Inicialize os parâmetros θ da rede de atualização (*update network*)
3. Inicialize os parâmetros θ' da rede de valores meta (*target network*)
4. Selecione ação a de acordo com uma política de exploração/exploitação
5. Adicione a experiência (s, a, r, s) em D (tamanho limitado)
6. Amostra aleatória de um *mini-batch* B experiências $\{(s_k, a_k, r_k, s'_k)\}$ de D
7. Faça atualizações Q-learning em θ usando B.

   Use a *target network* para obter os valores target a partir de θ':

   $$r + \gamma \max_{a'} \hat{Q}_{\theta'}(s', a')$$

   para fazer K atualizações de cada θ_i com todas as experiências do *mini-batch* B
8. $\theta' \leftarrow \theta$ (atualiza os parâmetros da *target network*)
9. Vá para o passo 4

Sobre o EP3

- Neste terceiro EP, veremos a versão de *Deep Learning* do algoritmo Q-Learning, chamada de *Deep Q-Learning* ou simplesmente DQN. O algoritmo DQN também foi visto em sala de aula e, além de aproximar a função Q(s, a) por uma rede neural (*update network*), na prática é feito algumas melhorias (regularizações):
  - *target network*: uso de uma segunda rede neural (*target network*) para representar os valores meta (fornece um sinal de aprendizado mais estável para evitar o estouro dos valores dos parâmetros)
  - *randomized experience replay*: permite o reuso dos dados construindo um *dataset* D de experiências passadas que são amostradas para a atualização dos parâmetros (uso mais eficiente das experiências)
  - *mini-batches*: são criados pequenos lotes de experiências aleatórias (mini-batches) $B \subseteq D$ e a cada iteração atualiza Q(s,a) com todas as experiências em B (quebram correlações entre atualizações reduzindo variância, isto é, garante que os dados seguem o princípio de i.i.d. (*independent and identically distributed random variables*). Este é um uso ainda mais eficiente das experiências.

Objetivo do EP3: geral

Dada a implementação do algoritmo DQN+Exp-Replay+mini-batch+Target-Network (disponível em um **notebook no E-Disciplinas**), e a especificação de todas as bibliotecas usadas pela nossa implementação, vocês deverão:

- Configurar o ambiente para executar o notebook para o problema do Cart Pole;
- Treinar o algoritmo com diferentes hiperparâmetros conforme especificaremos nos próximos slides;
- Enviar o notebook com os resultados da execução do algoritmo e a pasta **ckpt** contendo os dados das redes aprendidas (*checkpoints*);
- Gerar e enviar um relatório **EP3-Rel** como arquivo pdf (submissão separada) contendo:
    a. os gráficos gerados no notebook (Tensorboard) com valores default para as variáveis do programa. Para cada um desses gráficos, explique o que eles representam com relação ao agente de RL do Cart-Pole.
    b. os gráficos gerados no notebook variando os valores de alguns hiper-parâmetros do programa. Para cada variação, fazer uma análise sobre o impacto das mudanças no resultado do aprendizado quando comparados ao aprendizado com os valores default.

Objetivo do EP3: gráficos

- Hiper-parâmetros que você deve variar, um por vez:
  1. **total_timesteps:** quantidade de passos de tempo acumulados durante todos os episódios usados no treinamento (Note que a contagem dos passos de tempo (time-steps) é feita sequencialmente, isto é, não é zerada a cada novo episódio,) (default=20000)
     MODIFIQUE O VALOR DESTE HIPERPARÂMETRO PARA 30000.
  2. **ajuste a política epsilon-greedy:**
     - start_val ⇒ valor inicial de epsilon (default = 1)
     - end_val ⇒ valor final de epsilon (default = 0.01)
     - start_step ⇒ passo de tempo (contagem sequencial) que inicia o decaimento de epsilon (default = 1000)
     - end_step ⇒ passo de tempo (contagem sequencial) que finaliza o decaimento de epsilon (default = 10000)
     - MODIFIQUE APENAS O VALOR DO HIPERPARÂMETRO end_step, de 10000 para 5000
  3. **replay_buffer_batch_size:** tamanho do minibach que será amostrado (default = 64)
     MODIFIQUE O VALOR DESTE HIPERPARÂMETRO PARA 96.
- O aprendizado deve ser feito com pelo menos 3 *trials* de aprendizagem (isto é, 3 execuções completas de aprendizado para então avaliar a média das execuções, conforme é gerado pelos gráficos pré-definidos)

Objetivo do EP3: relatório (EP3-Rel)

- Explique cada um dos gráficos gerados pelo Tensorboard, e faça uma comparação dos gráficos com os valores dos hiperparâmetros *default* e os gráficos com valores modificados, conforme especificado no slide anterior.
- Você pode se inspirar no artigo: *Playing Atari with Deep Reinforcement Learning* da DeepMind (David Silver et.al., 2013) .
- Varie um dos parâmetros indicados por vez. Opcionalmente, gere um aprendizado usando combinações de mudanças dos hiperparâmetros indicados, ou ainda, modifique outros parâmetros que você achar que podem impactar no aprendizado do Cart Pole. Caso sua proposta de combinação de novos valores traga bons resultados, será dada um ponto extra.
- Responda as seguintes perguntas:
  a. De forma geral, os resultados obtidos neste EP3 usando DQN foram melhores ou piores em comparação com os resultados obtidos com o Q-learning tabular (EP2)?
  b. Qual dos hiperparâmetros mais impactou no aprendizado do agente RL do Cart Pole?

Objetivo do EP3: agente RL para o CartPole-V1

- CartPole-V1 é um domínio de planejamento probabilístico da biblioteca Gymnasium
- **Objetivo**: manter um pêndulo invertido (*pole*), preso em um carrinho (*cart*), equilibrado em relação à sua base móvel. Veja o video: https://www.youtube.com/watch?v=rvKnKtFgE7U
- O simulador oferece uma interface para a execução de ações e devolução das observações: posição do carrinho, velocidade, ângulo do pêndulo e velocidade angular do pêndulo
- Função recompensa: 1 se a ação realizada pontuou de forma a alcançar o objetivo (pêndulo na vertical) e 0 (zero) caso contrário.
- Caso o pêndulo caia o ambiente sinaliza que a simulação terminou indicando assim fim de episódio.
- O desafio do CartPole-1 é considerado concluído se o agente conseguir manter o pêndulo equilibrado por 500 passos de interação
    - https://gymnasium.farama.org/environments/classic_control/cart_pole/

Gráficos gerados pelo programa

Exemplos de gráficos gerados pelo programa são mostrados a seguir e são classificados em 3 categorias:

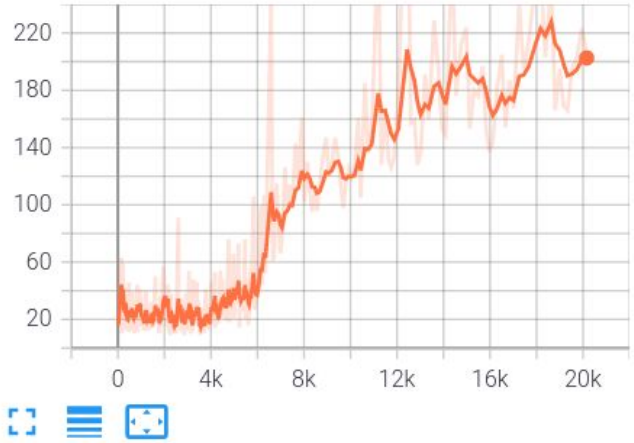
- collect
- evaluate
- train

Note que todos os gráficos descrevem os passos de tempo (time-steps) no eixo x e dados de *collect*, *evaluate* ou *train* no eixo y.

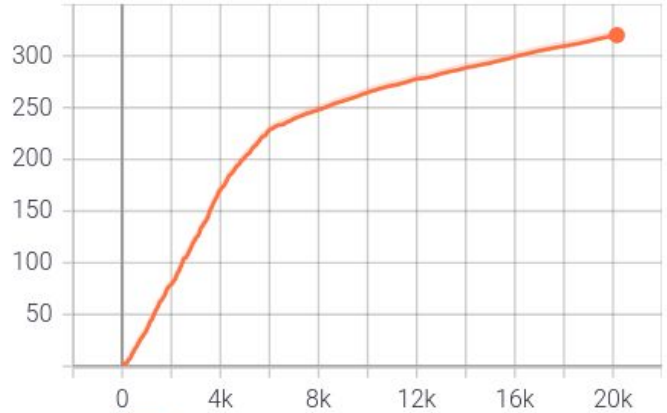
Você deve gerar os seus próprios gráficos nessas 3 categorias e explicar o que todos esses gráficos representam.

Collect

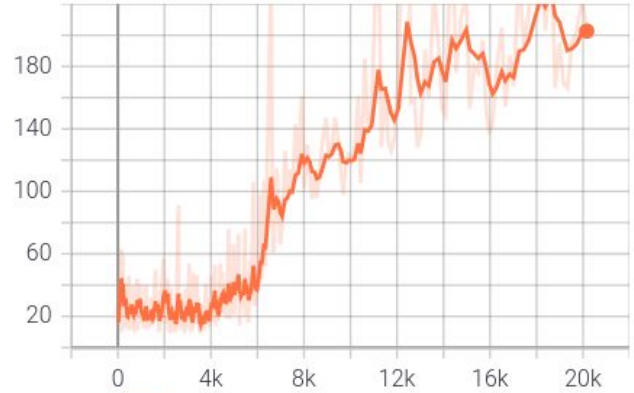
collect/episode_length
tag: collect/episode_length



collect/episodes
tag: collect/episodes

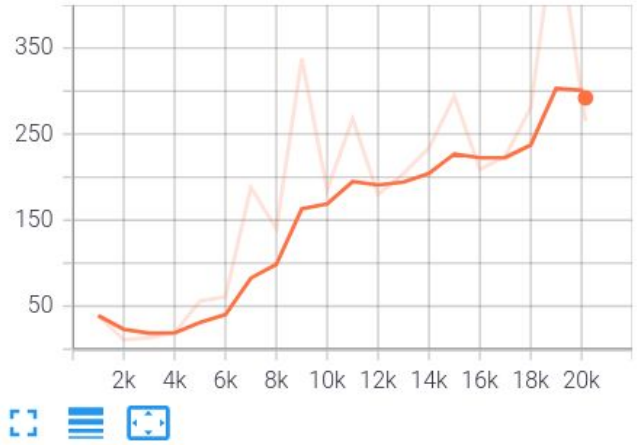


collect/episode_return
tag: collect/episode_return

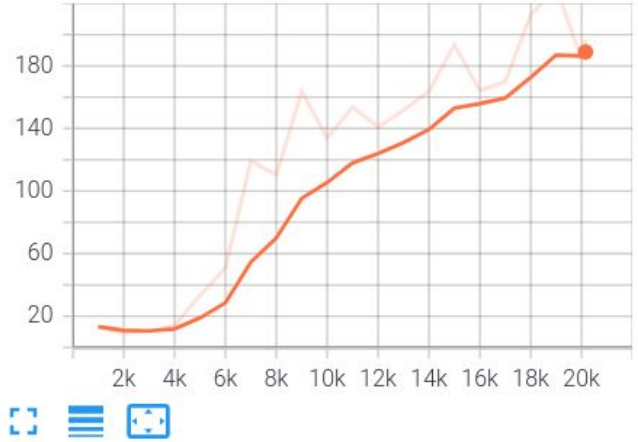


Evaluate

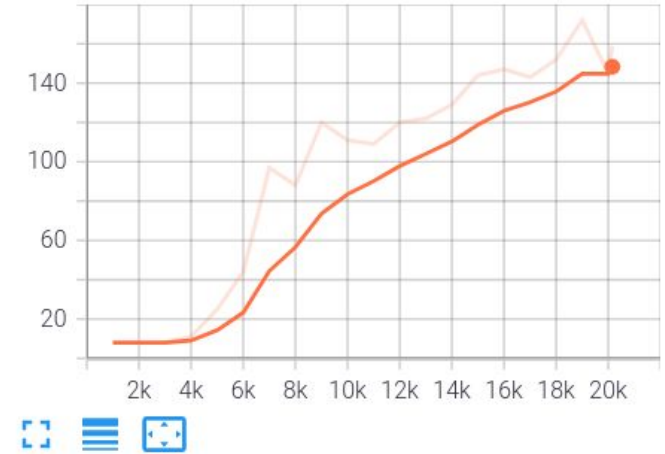
evaluation/episode_return_max
tag: evaluation/episode_return_max



evaluation/episode_return_mean
tag: evaluation/episode_return_mean

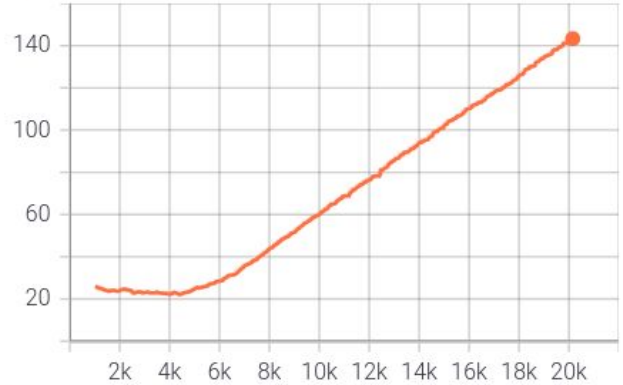


evaluation/episode_return_min
tag: evaluation/episode_return_min

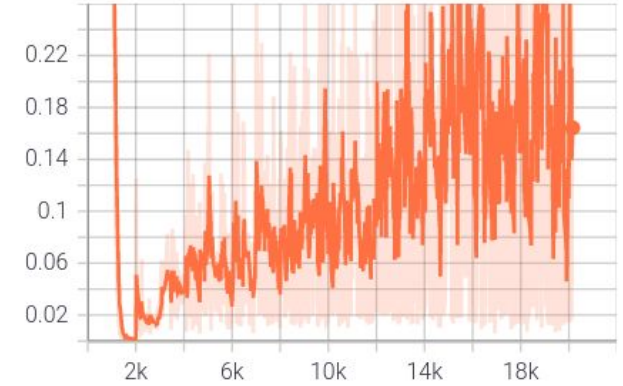


Train

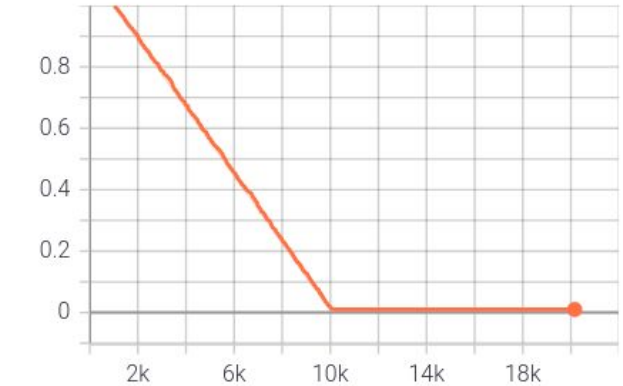
train/episode_return_mean
tag: train/episode_return_mean



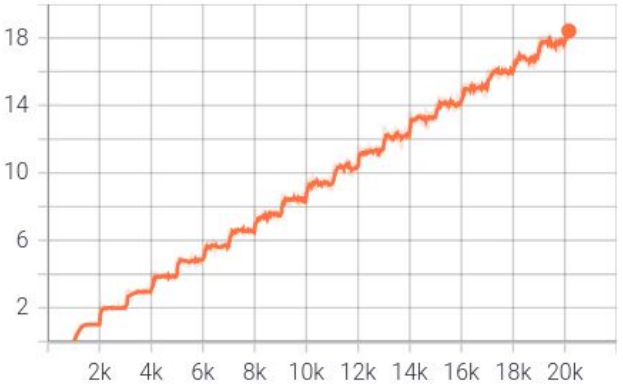
train/loss
tag: train/loss



train/epsilon
tag: train/epsilon



train/q_values_mean
tag: train/q_values_mean

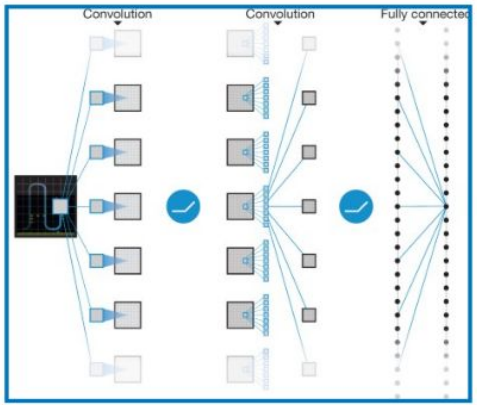


O algoritmo DQN implementado

DQN = *Q-learning* + Redes Neurais

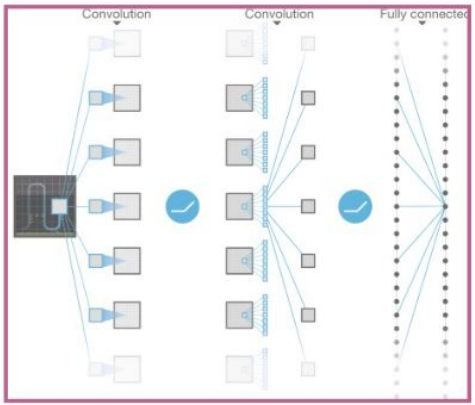
$$\min_{\phi \in \mathbb{R}^d} \mathbb{E}_{(s,a,r,s')} \left[Q_\phi(s,a) - \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\phi}}(s',a') \right) \right]^2$$

[Q-net online] [Q-net target]

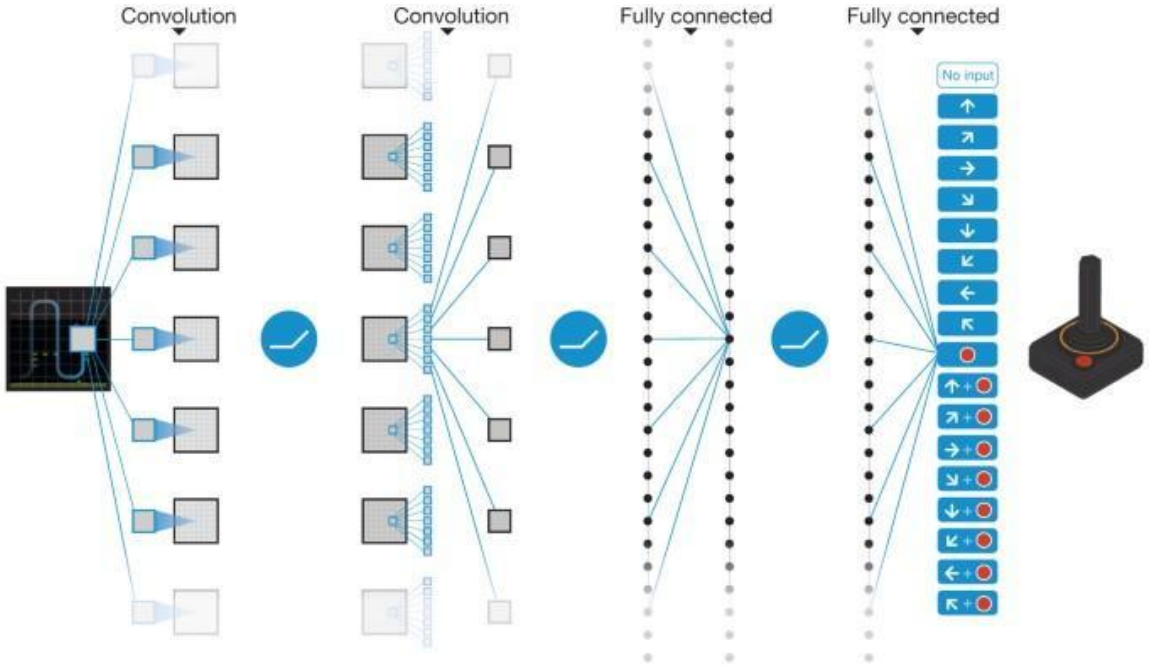


Os parâmetros da rede target são mantidos fixos durante o aprendizado...

... mas são "sincronizados" com a rede online periodicamente!

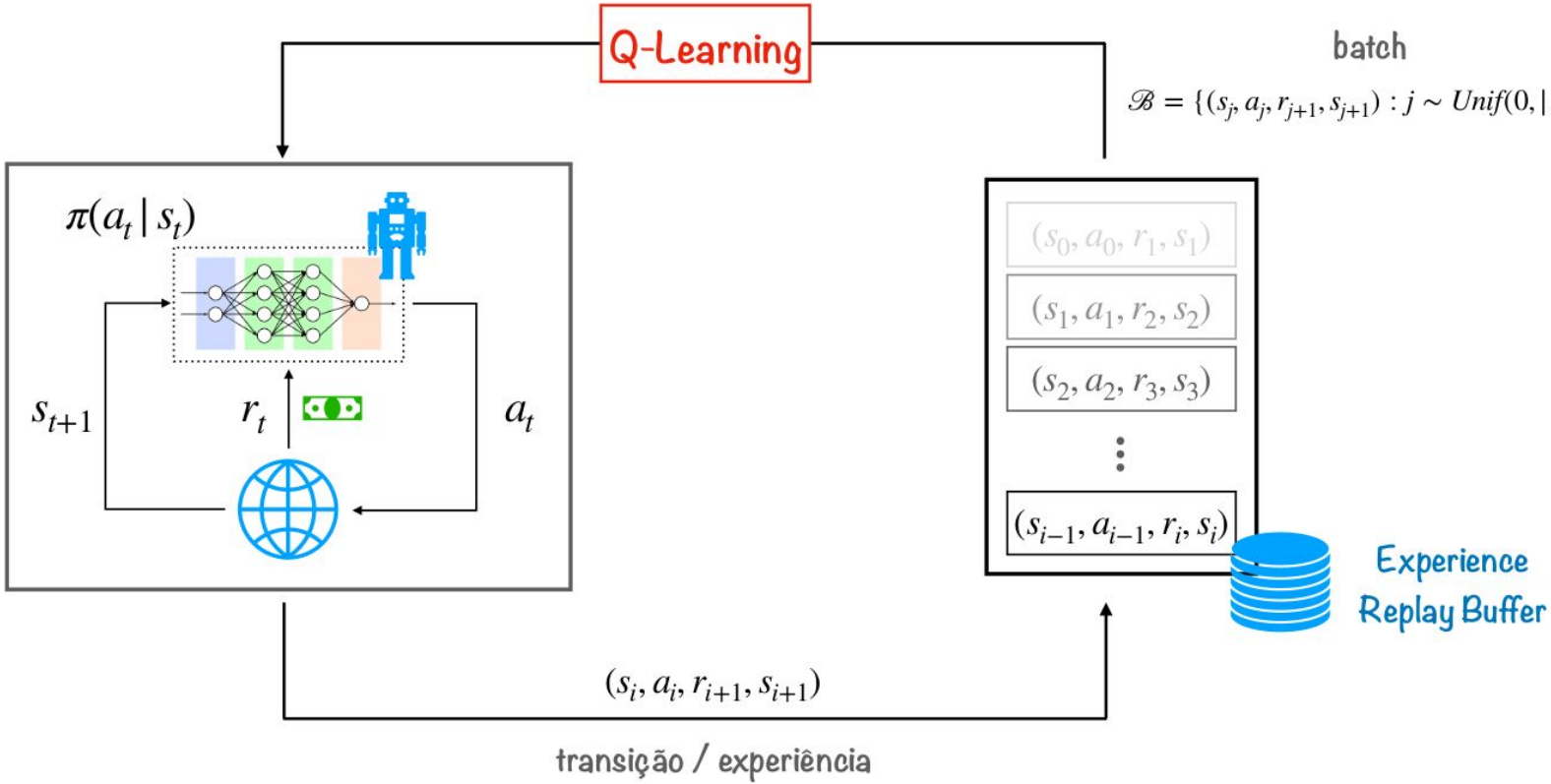


Q-net online e Q-net target: Redes Neurais Convolucionais



https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

DQN = *Q-learning* + Redes Neurais + *Replay Buffer*



Algorithm 1: Deep Q-learning + Replay Buffer + Target Network

Inicialize memória $\mathcal{D} \leftarrow \{\}$
Inicialize aleatoriamente as redes neurais Q_ϕ e $Q_{\bar{\phi}}$
$\bar{\phi} \leftarrow \phi$
$timestep \leftarrow 0$
for $episode = 1, \ldots, M$ **do**
    Inicialize estado inicial $s_0 \sim \rho$
    for $t = 0, \ldots, H-1$ **do**
        Escolha ação $a_t \sim \text{Unif}(1, |\mathcal{A}|)$ com probabilidade ϵ
        caso contrário escolha ação gulosa $a_t = \arg\max_{a \in \mathcal{A}} Q_\phi(s, a)$
        Execute a_t e observe recompensa r_{t+1} e próximo estado s_{t+1}
        $timestep \leftarrow timestep + 1$
        Adicione transição $(s_t, a_t, r_{t+1}, s_{t+1})$ em $\mathcal{D}$
        Amostre uniformemente um *minibatch* $B = \{(s_j, a_j, r_{j+1}, s_{j+1})\}$
        if $s_{t+1} \in \mathcal{S}_{\text{TERMINAL}}$ **then** $y_j \leftarrow r_j$
        else $y_j \leftarrow r_j + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\phi}}(s_{j+1}, a')$
        Defina a função objetivo $\mathcal{L}_\phi = \frac{1}{|B|} \sum_j (Q_\phi(s_j, a_j) - y_j)^2$
        Atualize os parâmetros via SGD $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_\phi$
        if $timestep \bmod N = 0$ **then** $\bar{\phi} \leftarrow \phi$
        if $s_{t+1} \in \mathcal{S}_{\text{TERMINAL}}$ **then**
            break
        end
    end
end

Componentes do algoritmo implementado

1. **Redes neurais (networks)**: inicialmente construiremos usando a biblioteca `dm-sonnet` a rede neural para a função $\mathcal{Q}(s,a)$; Q-net target (target-network), Q-net online (update-network)
2. **Função objetivo (loss)**: uma vez definida a classe da função $\mathcal{Q}(s,a)$, implementaremos a função objetivo utilizada no problema de "regressão" que o Q-Learning tenta resolver (é comum usarmos a função de loss "mean square error";
3. **Atualização (update)**: atualização dos parâmetros de um otimizador baseado em gradientes responsável por minimizar uma função objetivo previamente **definida**; e
4. **Política ϵ-greedy**: por fim definiremos a política estocástica para exploração.

Ambiente de programação: Python 3.7

- O código utilizado na aula prática foi desenvolvido em 2021. É importante observar que as versões das bibliotecas utilizadas não são necessariamente as versões mais recentes, e o uso de bibliotecas com versões distintas das especificadas pode resultar em conflitos no código.
- A versão do Python utilizada é a **3.7**, e as versões das bibliotecas estão especificadas no arquivo "**requirements.txt**".
- Para garantir que não haja conflitos com o código disponibilizado, sugerimos a instalação dos pacotes e bibliotecas utilizados em algum ambiente virtual de sua escolha. Uma possibilidade é criar um ambiente virtual utilizando o `virtualenv` conforme instruções a seguir:

Preparação do ambiente (1)

1. Instale o python **3.7**
2. Em seguida, execute os comandos a seguir para a criar seu ambiente virtual:

```
# cria ambiente virtual
$ virtualenv -p python3.7 env-ep3

# acessa a pasta do ambiente virtual criado
$ cd env-ep3

# ativa o ambiente virtual
$ source bin/activate

# instala o jupyter no ambiente virtual
$ pip install jupyter jupyterlab
```

Preparação do ambiente (2)

3. Após instalar o jupyter, coloque no diretório que representa seu ambiente virtual (neste exemplo `env-ep3`) o diretório "**utils**" e os arquivos "**requirements.txt**" e "**MAC0434_MAC5788_EP3.ipynb**"

4. Em seguida, execute o comando abaixo para instalar no seu ambiente virtual as bibliotecas necessárias

```
# instala as bibliotecas necessárias no ambiente virtual
$ pip install -r requirements.txt
```

Preparação do ambiente (3)

5. Após instalar as bibliotecas, execute o comando abaixo dentro do seu ambiente virtual e abra o notebook **MAC0434_MAC5788_EP3.ipynb** que contém a implementação do DQN.

```
$ jupyter notebook
```

Entrega no E-disciplinas

- o EP3 pode ser realizado em dupla (opcional)
- data de entrega: 21/12/20123