5b: Policy Learning and Deep RL

Policy Learning

Policy learning algorithms do not use a value function but instead operate directly on the policy, chosen from a family of policies $\pi_\theta : \mathcal{S} \mapsto \mathcal{A}$ determined by parameters θ.

Typically, π_θ is a neural network with weights θ which takes a state s as input and produces action a as output, which may be either continuous or discrete. If there is a discrete choice of actions, the network has one output for each possible action and uses Softmax to determine the conditional probability $\pi_\theta(a \mid s)$ of performing action a in state s.

For episodic domains like Backgammon, we do not need a discount factor, and the "fitness" of policy π_θ can be taken as the Value function of the initial states s_0 under this policy, which is the expected (or average) total reward received in each game by an agent using policy π_θ.

$$\text{fitness}(\pi_\theta) \;=\; V^{\pi_\theta}(s_0) \;=\; \mathbf{E}_{\pi_\theta}(r_{\text{total}})$$

Policy Learning algorithms include Policy Gradients, which use gradient descent to modify the parameters θ, and Evolution Strategies, which make random changes to θ and keep only those updates that are seen to increase the reward.

Optional video

Evolution Strategies

We will not go into details of evolutionary strategies in this course, but this video explains the basic idea using a simple example. Further details can be found in (Blair & Sklar, 1999) and (Salimans, 2017).

Optional video

Policy Gradients

Let us first consider episodic games where the reward is received only at the end of the episode. The agent takes a sequence of actions

$$a_1 a_2 \ldots a_t \cdots a_m$$

At the end it receives a reward r_{total}. We don't know which actions contributed the most, so we just reward all of them equally. If r_{total} is high (low), we change the parameters to make the agent more (less) likely to take the same actions in the same situations. In other words, we want to increase (decrease)

$$\log \prod_{t=1}^{m} \pi_\theta \left(a_t \mid s_t\right) = \sum_{t=1}^{m} \log \pi_\theta \left(a_t \mid s_t\right)$$

If $r_{\text{total}} = +1$ for a win and -1 for a loss, we can simply multiply the log probability by r_{total}. Differentials can be calculated using the gradient

$$\nabla_\theta r_{\text{total}} \sum_{t=1}^{m} \log \pi_\theta \left(a_t \mid s_t\right) = r_{\text{total}} \sum_{t=1}^{m} \nabla_\theta \log \pi_\theta \left(a_t \mid s_t\right)$$

The gradient of the log probability can be calculated nicely using Softmax.

If r_{total} takes some other range of values, we can replace it with $(r_{\text{total}} - b)$ where b is a fixed value, called the baseline.

The REINFORCE Algorithm

We then get the following REINFORCE algorithm (Williams, 1992):

$$
\begin{array}{l}
\text{for each trial} \\
\quad \text{run trial and collect states } s_t, \text{ actions } a_t, \text{ and reward } r_{\text{total}} \\
\quad \text{for } t = 1 \text{ to length(trial)} \\
\quad\quad \theta \leftarrow \theta + \eta(r_{\text{total}} - b)\nabla_\theta \log \pi_\theta(a_t \mid s_t) \\
\quad \text{end} \\
\text{end}
\end{array}
$$

This algorithm has successfully been applied, for example, to learn to play the game of Pong from raw image pixels.

Optional video

Policy Gradients with Incremental Rewards

We wish to extend the framework of Policy Gradients to non-episodic domains where rewards are received incrementally throughout the game (e.g. PacMan, Space Invaders). Every policy π_θ determines a distribution $\rho_{\pi_\theta}(s)$ on S

$$\rho_{\pi_\theta}(s) = \sum_{t \geq 0} \gamma^t \text{prob}_{\pi_\theta, t}(s)$$

where prob $\text{prob}_{\pi_\theta, t}(s)$ is the probability that, after starting in state s_0 and performing t actions, the

agent will be in state s. We can then define the fitness of policy π as

$$\text{fitness } (\pi_\theta) = \sum_s \rho_{\pi_\theta}(s) \sum_a Q^{\pi_\theta}(s,a)\pi_\theta(a \mid s)$$

Note: In the case of episodic games, we can take $\gamma = 1$, in which case $Q^{\pi_\theta}(s,a)$ is simply the expected reward at the end of the game. However, the above equation holds in the non-episodic case as well. The gradient of $\rho_{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$ are extremely hard to determine, so we ignore them and instead compute the gradient only for the last term $\pi_\theta(a \mid s)$

$$\nabla_\theta \text{ fitness } (\pi_\theta) = \sum_s \rho_{\pi_\theta}(s) \sum_a Q^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a \mid s)$$

Optional video

The Log Trick

$$\begin{aligned}\sum_a Q^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a \mid s) &= \sum_a Q^{\pi_\theta}(s,a)\pi_\theta(a \mid s)\frac{\nabla_\theta\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \\ &= \sum_a Q^{\pi_\theta}(s,a)\pi_\theta(a \mid s)\nabla_\theta \log \pi_\theta(a \mid s)\end{aligned}$$

So

$$\begin{aligned}\nabla_\theta \text{ fitness } (\pi_\theta) &= \sum_s \rho_{\pi_\theta}(s) \sum_a Q^{\pi_\theta}(s,a)\pi_\theta(a \mid s)\nabla\theta \log \pi_\theta(a \mid s) \\ &= \mathbf{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]\end{aligned}$$

The reason for the last equality is this: $\rho_{\pi_\theta}(s)$ is the number of times (discounted by γ^t) that we expect to visit state s when using policy π_θ. Whenever state s is visited, action a will be chosen with

probability $\pi_\theta(a \mid s)$

Optional video

Actor-Critic

Recall:

$$\nabla_\theta \text{ fitness } (\pi_\theta) = \mathbf{E}\pi_\theta \left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]$$

For non-episodic games, we cannot easily find a good estimate for $Q^{\pi_\theta}(s,a)$. One approach is to consider a family of Q-functions Q_w determined by parameters w (different from θ) and learn w so that Q_w approximates Q^{π_θ}, at the same time that the policy π_θ itself is also being learned.

This is known as an Actor-Critic approach because the policy determines the action, while the Q-Function estimates how good the current policy is, and thereby plays the role of a critic.

Actor Critic Algorithm

$$
\begin{aligned}
&\textbf{for each trial} \\
&\quad \text{sample } a_0 \text{ from } \pi_\theta\left(a \mid s_0\right) \\
&\quad \textbf{for each timestep } t \textbf{ do} \\
&\quad\quad \text{sample reward } r_t \text{ from } \mathcal{R}\left(r \mid s_t, a_t\right) \\
&\quad\quad \text{sample next state } s_{t+1} \text{ from } \delta\left(s \mid s_t, a_t\right) \\
&\quad\quad \text{sample action } a_{t+1} \text{ from } \pi_\theta\left(a \mid s_{t+1}\right) \\
&\quad\quad \frac{dE}{dQ} = -\left[r_t + \gamma Q_w\left(s_{t+1}, a_{t+1}\right) - Q_w\left(s_t, a_t\right)\right] \\
&\quad\quad \theta \leftarrow \theta + \eta_\theta Q_w\left(s_t, a_t\right) \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) \\
&\quad\quad w \leftarrow w - \eta_w \frac{dE}{dQ} \nabla_w Q_w\left(s_t, a_t\right) \\
&\quad \textbf{end} \\
&\textbf{end}
\end{aligned}
$$

Optional video

References

Blair, A.D., & E. Sklar, 1999. Exploring evolutionary learning in a simulated hockey environment, *Congress on Evolutionary Computation* (CEC), 197-203.

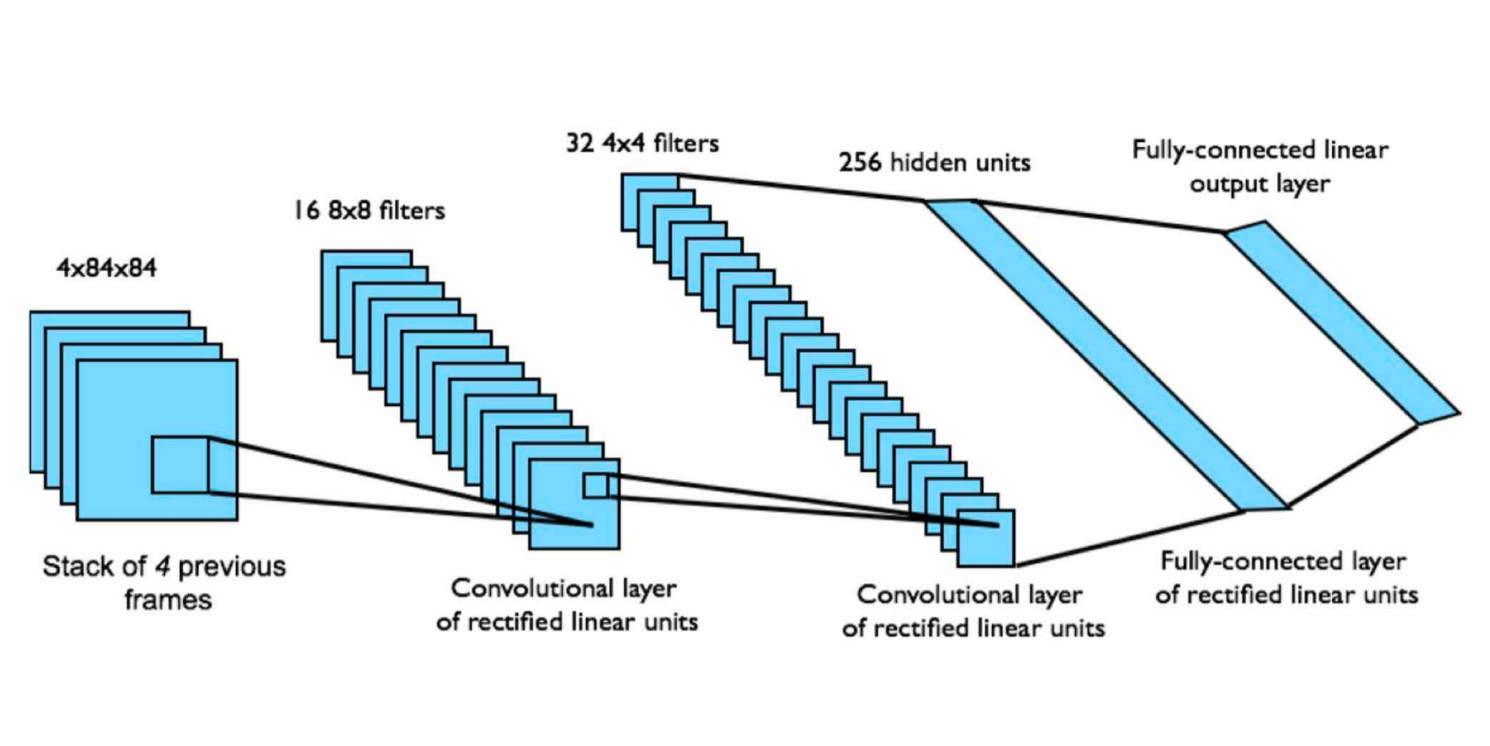
Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I., 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864*.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4), 229-256.

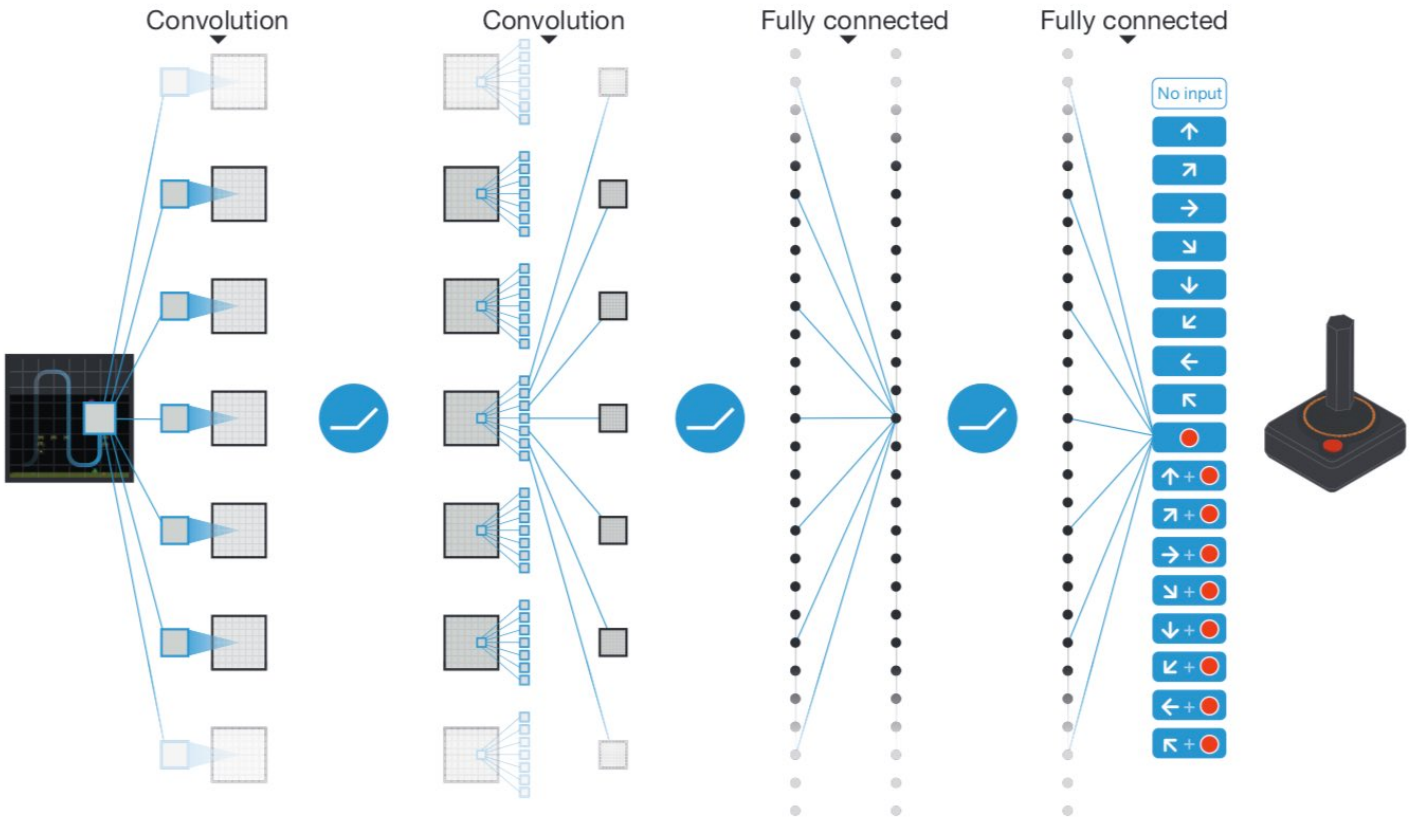
Deep Reinforcement Learning

Deep Q-Learning for Atari Games

Mnih (2015) demonstrated how Q-learning could be combined with deep CNNs to learn to play Atari games from pixels.



The input state s is a stack of raw pixels from four consecutive frames. The images are converted from 8-bit RGB images at resolution 210×160 pixels to 84×84 greyscale images. The 18 outputs are the Q-values $Q(s, a)$ for 18 different combinations of joystick/button positions. The reward is the change in score during the timestep.



Recall the Q-Learning update rule:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \eta \left[r_t + \gamma \max_b Q\left(s_{t+1}, b\right) - Q\left(s_t, a_t\right)\right]$$

If a lookup table is used to store the Q values for every state and action separately, the algorithm is guaranteed to eventually converge to an optimal policy. But, if the number of states is exponentially large, we must instead use a neural network Q_w and adjust the weights w according to the Q-Learning update rule, which is equivalent to minimizing:

$$\left[r_t + \gamma \max_b Q_w\left(s_{t+1}, b\right) - Q_w\left(s_t, a_t\right)\right]^2$$

The gradient is applied only to $Q_w\left(s_t, a_t\right)$, **not** to $Q_w(s_{t+1}, b)$.

Experience Replay

Training of deep neural networks for classification tasks generally requires that each mini-batch should include a variety of different inputs and target outputs. For Atari games, many similar states may occur in succession, often with the same action being selected. We can remove this **temporal correlation** between samples by storing experiences in a Replay Buffer.

In this scenario, one thread repeatedly plays the game, selecting its actions using an ε-greedy strategy based on the current Q-values, and builds a database of experiences $\left(s_t, a_t, r_t, s_{t+1}\right)$. Another thread samples asynchronously from this database and applies the Q-learning rule to minimise

$$\left[r_t + \gamma \max_b Q_w\left(s_{t+1}, b\right) - Q_w\left(s_t, a_t\right)\right]^2$$

This removes temporal correlations by sampling from a variety of game situations in random order, and also makes it easier to parallelise the algorithm on multiple GPUs.

Optional video

Prioritised Replay

Instead of sampling experiences uniformly, it may be more efficient to store and retrieve them in a priority queue with priority based on the DQN error (Schaul, 2015).

$$\left|r_t + \gamma \max_b Qw(s_{t+1}, b) - Q_w(s_t, a_t)\right|$$

This ensures the system will concentrate more effort on situations where the Q value was "surprising" (in the sense of being far away from what was predicted).

Double Q-Learning

If the same weights w are used to select actions and to evaluate actions (as well as states) the network may learn a suboptimal strategy due to a kind of "confirmation bias". One way to avoid this is to maintain two sets of weights w and $\bar{w}$, with one used for action selection and the other for evaluation (then swap their roles).

In the context of Deep Q-learning, a simpler approach is to use the current "online" version of w for selection, and an older "target" version $\bar{w}$ for evaluation (Van Hasselt, 2016). We therefore minimize

$$\left[r_t + \gamma Q_{\bar{w}}\left(s_{t+1}, \arg\max_b Q_w\left(s_{t+1}, b\right)\right) - Q_w\left(s_t, a_t\right)\right]^2$$

A new version of $\bar{w}$ is periodically calculated from the distributed values of w, and this $\bar{w}$ is broadcast to all processors.

Advantage Function

The Q Function $Q^{\pi}(s, a)$ can be written as a sum of the value function $V^{\pi}(s)$ plus an **advantage function**

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

$A^{\pi}(s, a)$ represents the advantage (or disadvantage) of taking action a in state s, compared to taking the action preferred by the current policy π. We can learn approximations for these two components separately:

$$Q(s, a) = V_u(s) + A_w(s, a)$$

Note that actions can be selected just using $A_w(s, a)$, because

$$\arg\max_b Q(s_{t+1}, b) = \arg\max_b A_w(s_{t+1}, b)$$

Optional video

Advantage Actor-Critic

Recall that in the REINFORCE algorithm, a baseline b could be subtracted from r_{total} for the purpose of variance reduction.

$$\theta \leftarrow \theta + \eta \left(r_{\text{total}} - b\right) \nabla_{\theta} \log \pi_{\theta}\left(a_t \mid s_t\right)$$

In the actor-critic framework, r_{total} is replaced by $Q\left(s_t, a_t\right)$

$$\theta \leftarrow \theta + \eta_\theta Q\left(s_t, a_t\right) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

We can also subtract a baseline from $Q\left(s_t, a_t\right)$. This baseline must be independent of the action a_t, but it could be dependent on the state s_t. A good choice of baseline is the value function $V_u(s)$, in which case the Q function is replaced by the advantage function

$$A_w(s, a) = Q(s, a) - V_u(s)$$

Asynchronous Advantage Actor-Critic

The Asynchronous Advantage Actor-Critic or A^{3}C Algorithm combines a policy network π_θ, a value function network V_u and an (estimated) Q-function.

- use polity network π_θ to choose actions
- learn a parameterised value function $V_u(s)$ by TD-learning
- estimate Q-value by n-step sample

$$Q\left(s_t, a_t\right) = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_u\left(s_{t+n}\right)$$

- update policy π_θ by

$$\theta \leftarrow \theta + \eta_\theta \left[Q\left(s_t, a_t\right) - V_u\left(s_t\right)\right] \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)$$

- update value function V_u by minimising

$$\left[Q\left(s_t, a_t\right) - V_u\left(s_t\right)\right]^2$$

Optional video

References

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., et al. 2015. Human-level control through deep reinforcement learning, *Nature* 518(7540) 529-533.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D., 2015. Prioritized experience replay, *arXiv:1511.05952*.

Van Hasselt, H., Guez, A., & Silver, D., 2016. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 30, No. 1).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T., Silver, D., & Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning* (pp. 1928-1937).

Examples of Deep Reinforcement Learning

Dr Alan Blair discusses a variety of examples of Deep Reinforcement Learning in practice.

Transcipt

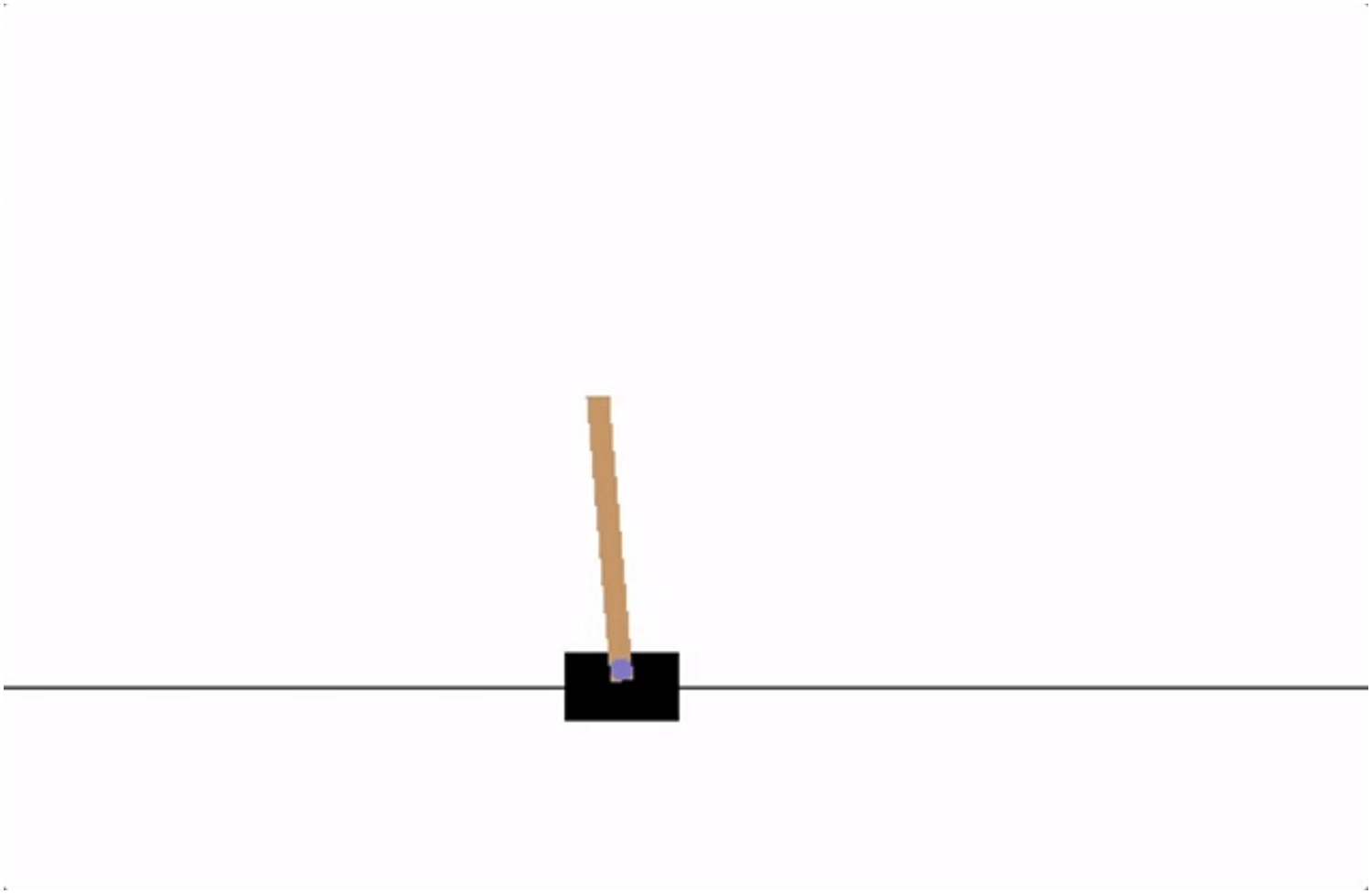
Mr Alex Long discusses frameworks for understanding Deep Reinforcement Learning with demonstrations.

Coding: Deep Q-Learning

In this exercise, you will practice how to use PyTorch to train a Deep Q-learning (DQN) agent on the CartPole-v0 task from OpenAI Gym.

Please **download** the notebook and implement/run it locally.

Installation instruction: https://gym.openai.com/docs/



This exercise is modified from Adam Paszke's reinforcement learning (DQN) tutorial with necessary simplification.