Wendelin Böhmer <j.w.bohmer@tudelft.nl> TU Delft due: 07-12-2023 at 10:45

Deep reinforcement learning

A2.1: Implement and test DQN **(7 points)**

For this exercise you will extend a custom learning framwork that is given as a Jupyter Notebook `dqn.ipynb`. Make sure you have understand the code in week 3, so that you can ask any questions that you may have in time.

(a) *[1 point]* Go through the implementation in the given Jupyter Notebook. Run online Q-learning, that is, use the `QLearningExperiment` with the `QLearner` class on the `CartPole-v01` environment for $200k$ steps in the environment. You can restrict the maximal episode-length to 200 steps.

(b) *[1 point]* Implement online Q-learning with an experience replay buffer by extending the given skeleton of the `DQNExperiment` class. Train your implementation again in the `CartPole-v1` environment for $200k$ steps.

(c) *[1 point]* Extend the `QLearning` class with target-networks that use a hard update rule. Train your implementation again in the `CartPole-v1` environment for $200k$ steps.

(d) *[1 point]* Extend your implementation with a soft-update rule for the target network and test it in the environment `CartPole-v1` for $200k$ steps.

(e) *[1 point]* Extend your implementation with double-Q-learning and test it in the `CartPole-v1` environment for $200k$ steps.

(f) *[1 point]* Run your implementation of `DoubleQLearner` on the `MountainCar-v0` environment for $200k$ steps. In all likelihood, your agent will not be able to solve the task (reach the goal), and learning should not pick up at all. Explain why that is.

(g) *[1 point]* Run your implementation on the `LunarLander-v2` environment for at least 2 million environment steps (more is better), with one gradient update step every 50 environmental steps and a maximal episode length of 500. This can take a while, expect 1-2 hours of computation time. Do you get similar results as shown in the lecture?

Solution A sample implementation can be found on Brightspace. The reason why `MountainCar-v0` does not learn is that the dynamics draw random episodes *away* from the goal state. At the default episode length of 200 steps, epsilon-greedy exploration observes therefore almost never a rewarded trajectory and DQN can therefore not learn anything (all sampled trajectories have the same return).

A2.2: RNNs as linear layers (old exam question) **(2 points)**

A given linear recurrent neural network takes a time series of n input vectors $\boldsymbol{x}_t \in \mathbb{R}^d$ and outputs a time-series of n vectors $\boldsymbol{y}_t \in \mathbb{R}^b$. The RNN computes hidden outputs $\boldsymbol{h}_t \in \mathbb{R}^q$ without non-linearities:

$$\boldsymbol{h}_t := \mathbf{W}\boldsymbol{h}_{t-1} + \mathbf{U}\boldsymbol{x}_t \in \mathbb{R}^q\,, \qquad \boldsymbol{y}_t := \mathbf{V}\boldsymbol{h}_t \in \mathbb{R}^b\,, \qquad 1 \leq t \leq n\,, \qquad \boldsymbol{h}_0 := \mathbf{0}\,.$$

(a) *[1 point]* Prove by induction that the above update equations are equivalent to the multivariate function $g : \{\mathbb{R}^d\}_{t=1}^n \times \mathbb{R}^{q\times q} \times \mathbb{R}^{q\times d} \times \mathbb{R}^{b\times q} \to \mathbb{R}^{b\times n}$:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{W}, \mathbf{U}, \mathbf{V})_{k,m} := \sum_{t=1}^m (\mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t)_k = (\boldsymbol{y}_m)_k\,, \qquad 1 \leq k \leq b\,, \quad 1 \leq m \leq n\,.$$

Hint: start your proof with showing by induction that $\boldsymbol{h}_m \overset{(ind)}{=} \sum_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t$.

(b) *[1 point]* Define the *linear function* $f : \mathbb{R}^{\mathcal{J}} \to \mathbb{R}^{\mathcal{I}}$ that is equivalent to $g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{W}, \mathbf{U}, \mathbf{V})$:

$$f(\boldsymbol{z})_i \;:=\; \sum_{j\in\mathcal{J}} \Theta_{i,j}\, z_j\,, \qquad \forall \boldsymbol{z} \in \mathbb{R}^{\mathcal{J}}\,, \quad \forall i \in \mathcal{I}\,,$$

by defining the index sets $\mathcal{J}$ and $\mathcal{I}$, and by constructing the inputs $\boldsymbol{z} \in \mathbb{R}^{\mathcal{J}}$ from $\{\boldsymbol{x}_t\}_{t=1}^n$ and the parameter matrix/tensor $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ from the g's parameters $\mathbf{U}, \mathbf{V}$ and $\mathbf{W}$. Make sure f outputs exactly the same as g!

Solution

(a) We prove by induction $\boldsymbol{h}_m \overset{(ind)}{=} \sum_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t$. Induction beginning $m = 1$:

$$\boldsymbol{h}_1 \overset{(def)}{=} \mathbf{W}\boldsymbol{h}_0 + \mathbf{U}\boldsymbol{x}_1 = \mathbf{U}\boldsymbol{x}_1 = \sum_{t=1}^1 \mathbf{W}^{1-t}\mathbf{U}\boldsymbol{x}_t\,. \qquad \checkmark$$

Induction step for $m+1$:

$$\boldsymbol{h}_{m+1} \overset{(def)}{=} \mathbf{W}\boldsymbol{h}_m + \mathbf{U}\boldsymbol{x}_{m+1} \overset{(ind)}{=} \mathbf{W}\sum_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{x}_{m+1} = \sum_{t=1}^{m+1} \mathbf{W}^{m+1-t}\mathbf{U}\boldsymbol{x}_t\,. \qquad \checkmark$$

Lastly we can use the induction of $\boldsymbol{h}_m$ to show that:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{U}, \mathbf{V}, \mathbf{W})_{k,m} = \Big(\mathbf{V} \underbrace{\sum_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t}_{\boldsymbol{h}_m}\Big)_k \overset{(def)}{=} (\boldsymbol{y}_m)_k\,. \qquad \square$$

Rubrik:

- 1 point for the induction beginning
- 2 points for the induction step:
  - 1 point for application of definition and induction assumption
  - 1 point for construction the correct sum over $m+1$
- 1 point for the final equality of g and $\boldsymbol{y}$

(b) Because we need $\boldsymbol{z} \equiv \{\boldsymbol{x}_t\}_{t=1}^n$, we define:

$$\mathcal{J} := \{(u,v)\,|\,1 \leq u \leq d, 1 \leq v \leq n\} \qquad \text{and} \qquad z_{(u,v)} := (\boldsymbol{x}_v)_u\,.$$

Similarly, the output of f needs to be the same as that of g, so we define

$$\mathcal{I} := \{(k,m)\,|\,1 \leq k \leq b, 1 \leq m \leq n\}\,.$$

Lastly we only need to extend the sum over t until n by zeroing out non-existing summands, i.e. $\sum_{t=1}^m \phi_t = \sum_{t=1}^n \delta(t \leq m)\,\phi_t, \forall m \leq n$:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{U}, \mathbf{V}, \mathbf{W})_{k,m} = \sum_{t=1}^m\sum_{l=1}^d \big(\mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\big)_{k,l}(\boldsymbol{x}_t)_l = \underbrace{\sum_{t=1}^n\sum_{l=1}^d}_{\sum_{(l,t)\in\mathcal{J}}} \underbrace{\big(\delta(t\leq m)\,\mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\big)_{k,l}}_{\Theta_{(k,m),(l,t)}} \underbrace{(\boldsymbol{x}_t)_l}_{z_{(l,t)}}\,.$$

$\square$

Rubrik:

- 1 point for identifying $\mathcal{J}$ correctly
- 1 point for identifying $\mathcal{I}$ correctly
- 1 point for extending the sum to n, e.g. by using a Kronecker-δ function
- 1 point for defining the correct parameter matrix $\mathbf{\Theta}$

A2.3: CNN are GNN are MHA **(2 points)**

Many neural architectures are equivalent to each other. In this question we will look at the example of 2-dimensional CNN layer (see Slides 10-12 of Lecture 4). These CNN take images $\mathbf{X} \in \mathbb{R}^{3\times H\times W}$ as input, and output a feature map $\mathbf{X}' \in \mathbb{R}^{F\times(H-ky+1)\times(W-kx+1))}$ with F features. The only parameters of the layer are the kernel $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{F\times 3\times hy\times hx}$. The CNN computes the output feature map:

$$X'_{l,a,b} \quad := \quad \mathrm{CNN}(\mathbf{X}, \boldsymbol{\mathcal{K}})_{l,a,b} \quad := \quad \sum_{i=1}^{H_K}\sum_{j=1}^{W_K}\sum_{p=1}^{3} \mathcal{K}_{l,p,i,j}\, X_{p,a+i-1,b+j-1}$$

(a) *[1 point]* Assume a relational GCN (see Slide 20 of Lecture 4) with $\mathcal{V} := H \times W$ nodes, one for each pixel, i.e., $V_{\binom{a}{b},p} := X_{p,a,b}$. Design such an R-GCN that is *exactly* equivalent to the above CNN, that is, which outputs the same feature map $\mathbf{X}'$ for each input image $\mathbf{X}$ as the CNN.

Hint: it is sufficient to define the R-GNN's parameter matrices $\mathbf{W}^k$ and $\mathbf{B}^k$ and the input $\mathbf{V}$ using the dimensionality of image $\mathbf{X}$ and the CNN's kernel.

(b) *[1 point]* Let's assume a simpler GCN (no relational GCN needed), where the $(H-ky+1)\times(W-kx+1)$ nodes are annotated with the image patches (of dimension $3 \times ky \times kx$) that correspond to each kernel application, that is, $V_{\binom{a}{b},\left(\begin{smallmatrix}p\\i\\j\end{smallmatrix}\right)} := X_{p,a+i-1,b+j-1}$. Design such a GCN that is exactly equivalent to the above CNN. How does the weight-matrix $\mathbf{W}$ look like? If you would learn the weight matrix with a single-head self-attention layer (MHA with $K=1$, see Slide 21 of Lecture 4), can you derive a parameter matrix $\mathbf{A}^1$ that would be able to repesent the the above CNN *arbitrary well* for one fixed input image, where all node annotations (rows in $\mathbf{V}$) are linearly independent?

Solution

(a) The R-GNN is defined like this:

$$\text{R-GCN}(\mathbf{V}, \{\mathbf{W}^k, \mathbf{B}^k\}_{k=1}^K)_{nl} \quad := \quad V'_{nl} \quad := \quad \sum_{k=1}^{K}\sum_{m\in\mathcal{V}}\sum_{q=1}^{Q} W^k_{nm}\, V_{mq}\, B^k_{ql}$$

For each position $\binom{a}{b}$ in the CNN's output feature map, the CNN computes for each pixel $\binom{a+i}{b+j}$ within the kernel's reach the linear function $\sum_{q=1}^3 \mathcal{K}_{l,q,i,j} X_{q,a+i,b+j}$. Each pixel $\binom{a+i}{b+j}$ get's multiplied by it's own weights in the kernel, and we therefore need $K = kx\,ky$ heads (or topologies or relation-types) in the R-GNN. We can define $B^{(i,j)}_{ql} := \mathcal{K}_{l,q,i,j}$. Lastly, we need to define the graph topology in matrix W^k. Here we define a topology where each head $k = \binom{i}{j}$ has exactly one edge for each pixel $\binom{a}{b}$ in the input: from $\binom{a+i-1}{b+j-1}$ to $\binom{a}{b}$. This yields a weight matrix $W^{(i,j)}_{\binom{a}{b},\binom{c}{d}} = \delta_{(c=a+i-1)}\,\delta_{(d=b+h-1)}$, where $\delta_{(\text{boolean})}$ are Kronecker delta functions.

$$\text{We have therefore} \quad B^{(i,j)}_{ql} := \mathcal{K}_{l,q,i,j}\,, \quad \text{and} \quad W^{(i,j)}_{\binom{a}{b},\binom{c}{d}} = \delta_{(c=a+i-1)}\,\delta_{(d=b+h-1)}\,.$$

Substituting these matrices $\mathbf{V}$, $\mathbf{B}^k$ and $\mathbf{W}^k$ in the definition of R-GCN above yields exactly the computation of the CNN. As the output feature map of the CNN is smaller than the input image, some nodes in the R-CGN will have a meaningless output. One can either ignore those, or remove all incoming edges, which will return a zero (or otherwise unspecified) annotation.

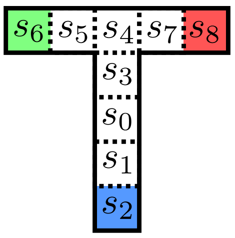
(b) Due to the definition of $\mathbf{V}$, we can simply set $B_{\left(\begin{smallmatrix}p\\i\\j\end{smallmatrix}\right),l} := \mathcal{K}_{l,p,i,j}$, which is exactly what the CNN computes. This means the weight-matrix $\mathbf{W} := \mathbf{I}$ is an identity matrix. This corresponds to a GCN where every node is only connected to itself.

Since MHA computes the weight matrix $\mathbf{W}^1 := \mathbf{S}^k \oslash \mathbf{S}^1\mathbf{1}\mathbf{1}^\top, \mathbf{S}^1 := \exp(\mathbf{V}\mathbf{A}^{1\top}\mathbf{A}^1\mathbf{V}^\top)$, we therefore need to find parameters for $\mathbf{A}$ such that $\mathbf{W} \to \mathbf{I}$. We know the rows of $\mathbf{V}^{n\times d}$, with $n = (H - hy + 1)(W - kx + 1)$ and $d = 3\,kx\,ky$, are linearly independent, which implies $d \geq n$ (otherwise some rows would have to be linearly dependent). To make $\mathbf{W}^1$ arbitrary close to an identity matrix for some specific given $\mathbf{V}$, we can use a *singular value decomposition* (an *eigenvalue decomposition* would work as well, but is more complicated) of $\mathbf{V} =: \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}'^\top$, where $\mathbf{U}^\top\mathbf{U} = \mathbf{I} = \mathbf{U}'^\top\mathbf{U}'$, and $\boldsymbol{\Sigma}$ is a diagonal matrix of the singular values. Due to linear independent rows we also know that $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$. For any $m > 0$ we can now set

$$\mathbf{A}^1 := \sqrt{m}\,\boldsymbol{\Sigma}^{-1}\mathbf{U}'^\top \quad\Rightarrow\quad \mathbf{S}^1 = \exp(\overbrace{\mathbf{U}\boldsymbol{\Sigma}\mathbf{U}'^\top}^{\mathbf{V}}\overbrace{\mathbf{U}'\boldsymbol{\Sigma}^{-1}\sqrt{m}}^{\mathbf{A}^{1\top}}\overbrace{\sqrt{m}\boldsymbol{\Sigma}^{-1}\mathbf{U}'^\top}^{\mathbf{A}^1}\overbrace{\mathbf{U}'\boldsymbol{\Sigma}\mathbf{U}^\top}^{\mathbf{V}^\top}) = \exp(m\mathbf{I})$$

As $\exp(0) = 1$ and $\exp(m) = e^m$, increasing m will increase the diagonal of $\mathbf{S}^1$ arbitrarily large, and the normalization will therefore make $\mathbf{W}^1$ arbitrarily close to an identity matrix.

A2.4: Partial observability **(3 points)**



The figure to the right shows a T-maze with 9 states, where the actions space is $\mathcal{A} := \{left, right, up, down\}$. The transitions are deterministic, if no state exist in the corresponding direction, the agent remains in the same state. A hidden variable $c \in \{red, green\}$ determines whether s_6 is rewarded with +1 and s_8 is punished with -1 (for $c = green$) or vice versa (for $c = red$). Those are also the only terminal states, and all other transitions yield no reward. The start state is random and equally chosen from $\{s_0, s_1, s_3\}$. The hidden variable c is determined randomly with $p(c = green) = \frac{1}{3}$ at the beginning of an episode and is only observable in state s_2. Additionally, the agent always observes the state $s_0, \ldots, s_8$ it is in.

(a) *[1 point]* Derive the values $V^*(s_0)$ for the optimal policies *with* and *without memory* and a discount factor $\gamma \in (0, 1)$. For which range of γ should the agent prefer the memory-less policy?

(b) *[1 point]* Let's assume the agent *only* observes c in s_2 (all other observations are blank). Give an example of the agent's *minimal memory representation*, that is, the smallest set of past observations and actions the agent must remember to solve the task optimally, and justify your answer. Is this minimal representation the same throughout the an episode, or can the agent forget some past observations or actions at some point?

(c) *[1 point]* With the reduced observability from the above question (b), how does the optimal policy change when the transitions are *stochastic*, i.e., executing an action has only a 50% chance to move into the desired direction and a 50% chance to remain in the current state? Consider both the case of policies with and without memory. You do not need to prove your answer, but please justify it.

Solution

(a) The optimal policy with memory has to go down to s_2 (2 steps) and then up and either left or right to s_6 or s_8 (6 steps), which yields a return of $V^*(s_0) = \gamma^7$. The memory-less policy always goes to s_8 (as it is more often rewarding) in 4 steps, yielding $V'^*(s_0) = \gamma^3(\frac{2}{3} - \frac{1}{3}) = \frac{\gamma^3}{3}$.

Small discount factors make the memory-less policy more attractive. We can find the switch point by setting

$$\gamma^7 = \frac{\gamma^3}{3} \quad \Leftrightarrow \quad \gamma^4 = \tfrac{1}{3} \quad \Leftrightarrow \quad \gamma = 3^{-\frac{1}{4}} \approx 0.76\,.$$

For $\gamma < 3^{-\frac{1}{4}}$ the agent should therefore prefer the memory-less policy, as it has a higher value!

(b) The agent starts at a random state in $\{s_0, s_1, s_3\}$.

(i) From each starting state it only needs to go down, so the agent does not need to remember anything before it observes c in s_2.

(ii) Afterwards, it needs to remember (or count) the next 4 up actions to deduce it is in state s_4. And c, of course.

(iii) Now it chooses either $left$ or $right$, depending on the observed c. This decision will be repeated until termination, and at this point the agent can forget everything except the last action (that it needs to repeat).

Note that an LSTM would learn to remember only such a minimal subset, *not* the true underlying state.

(c) The memory-endowed policy would act the same until c is observed in s_2. However, afterwards it is no longer sufficient to count 4 up actions, as the number of actions until the agent reaches s_4 is now random (but ≥ 4). Instead, the optimal policy will choose the first 4 actions after observing c to be up, followed by an infinite number of random decisions of either up, or the correct $left$ or $right$ action (according to c). This way the agent chooses either the optimal action or an action that does not change the state.

The memory-less policy works the same way, but always randomly chooses up or $right$ (which has the higher chance of reward). Computing the optimal probability to choose up and $right$ (with or without history) is quite complex, and not needed for this question.

A2.5: Policy Gradient Theorem **(3 points)**

Let $R_t = \sum_{k=t}^{n-1} \gamma^{k-t} r(s_k, a_k)$ denote the discounted return after time-step t of a trajectory sampled from a Markov chain of length n, defined by MDP $\langle \mathcal{S}, \mathcal{A}, \rho, P, r\rangle$ and policy π_θ, parameterized by θ, and let $\mathbb{E}_{\pi_\theta}[\cdot]$ denote the expectation over that Markov chain.

(a) *[1 point]* Prove that $\nabla_\theta \mathbb{E}[f(s,a)\,|_{a\sim\pi_\theta(a|s)}] = \mathbb{E}[f(s,a)\,\nabla_\theta \ln \pi_\theta(a|s)|_{a\sim\pi_\theta(a|s)}], \forall s \in \mathcal{S},\ \forall f : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$. Note that f does not (and is not allowed to) depend on θ here.

(b) *[1 point]* Prove that $\nabla_\theta \mathbb{E}_{\pi_\theta}\big[R_t|^{s_t}_{a_t}\big] = \gamma \mathbb{E}_{\pi_\theta}\big[R_{t+1} \nabla_\theta \ln \pi_\theta(a_{t+1}|s_{t+1})|^{s_t}_{a_t}\big] + \gamma \mathbb{E}_{\pi_\theta}\big[\nabla_\theta \mathbb{E}_{\pi_\theta}\big[R_{t+1}|^{s_{t+1}}_{a_{t+1}}\big]\,|^{s_t}_{a_t}\big]$.

(c) *[1 point]* Prove the Policy Gradient Theorem $\nabla_\theta J[\pi_\theta] = \nabla_\theta \mathbb{E}_{\pi_\theta}[R_0] = \mathbb{E}_{\pi_\theta}\Big[\sum\limits_{t=0}^{n-1} \gamma^t R_t\, \nabla_\theta \ln \pi_\theta(a_t|s_t)\Big]$.

Solution

(a) First, note that $\nabla_\theta \ln \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$ and therefore $\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s) \nabla_\theta \ln \pi_\theta(a|s)$.

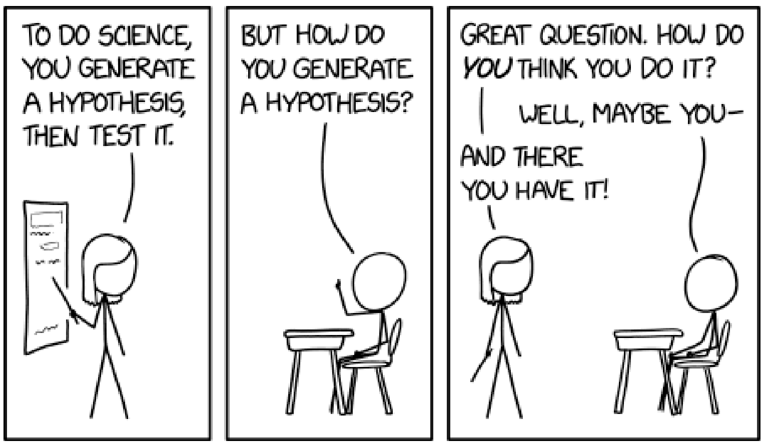
$$\begin{aligned}
\nabla_\theta \mathbb{E}[f(s,a)\,|_{a\sim\pi_\theta(a|s)}] &= \int f(s,a)\,\nabla_\theta \pi_\theta(a|s)\,da \\
&= \int f(s,a)\,\pi_\theta(a|s)\,\nabla_\theta \ln \pi_\theta(a|s)\,da \\
&= \mathbb{E}[f(s,a)\,\nabla_\theta \ln \pi_\theta(a|s)\,|\,{}_{a\sim\pi_\theta(a|s)}]
\end{aligned}$$

(b)

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta}\big[R_t\big|^{s_t}_{a_t}\big] &= \nabla_\theta\Big(r(s_t,a_t) + \gamma \iint P(s_{t+1}|s_t,a_t)\,\pi_\theta(a_{t+1}|s_{t+1})\,\mathbb{E}_{\pi_\theta}\big[R_{t+1}\big|^{s_{t+1}}_{a_{t+1}}\big]\,ds_{t+1}\,da_{t+1}\Big) \\
&= \gamma \iint P(s_{t+1}|s_t,a_t)\Big(\mathbb{E}_{\pi_\theta}\big[R_{t+1}\big|^{s_{t+1}}_{a_{t+1}}\big]\,\nabla_\theta\pi_\theta(a_{t+1}|s_{t+1}) \\
&\qquad\qquad + \pi_\theta(a_{t+1}|s_{t+1})\nabla_\theta\mathbb{E}_{\pi_\theta}\big[R_{t+1}\big|^{s_{t+1}}_{a_{t+1}}\big]\Big)\,ds_{t+1}\,da_{t+1} \\
&\overset{\text{(a)}}{=} \gamma\mathbb{E}_{\pi_\theta}\big[R_{t+1}\nabla_\theta \ln\pi_\theta(a_{t+1}|s_{t+1})\big|^{s_t}_{a_t}\big] + \gamma\mathbb{E}_{\pi_\theta}\big[\nabla_\theta\mathbb{E}_{\pi_\theta}\big[R_{t+1}\big|^{s_{t+1}}_{a_{t+1}}\big]\big|^{s_t}_{a_t}\big]
\end{aligned}$$

(c)

$$\begin{aligned}
\nabla_\theta J[\pi_\theta] &= \nabla_\theta \iint \rho(s_0)\,\pi_\theta(a_0|s_0)\,\mathbb{E}_{\pi_\theta}\big[R_0\big|^{s_0}_{a_0}\big]\,ds_0\,da_0 \\
&\overset{\text{(a)}}{=} \iint \rho(s_0)\,\pi_\theta(a_0|s_0)\Big(\mathbb{E}_{\pi_\theta}\big[R_0\big|^{s_0}_{a_0}\big]\,\nabla_\theta\ln\pi_\theta(a_0|s_0) + \nabla_\theta\mathbb{E}_{\pi_\theta}\big[R_0\big|^{s_0}_{a_0}\big]\Big)ds_0\,da_0 \\
&\overset{\text{(b)}}{=} \iint \rho(s_0)\,\pi_\theta(a_0|s_0)\Big(\mathbb{E}_{\pi_\theta}\big[R_0\big|^{s_0}_{a_0}\big]\,\nabla_\theta\ln\pi_\theta(a_0|s_0) + \gamma\mathbb{E}_{\pi_\theta}\big[R_1\,\nabla_\theta\ln\pi_\theta(a_1|s_1)\big|^{s_0}_{a_0}\big] \\
&\qquad\qquad + \gamma\mathbb{E}_{\pi_\theta}\big[\nabla_\theta\mathbb{E}_{\pi_\theta}\big[R_1\big|^{s_1}_{a_1}\big]\big|^{s_0}_{a_0}\big]\Big)ds_0\,da_0 \\
&\overset{\text{(b)}}{=} \iint \rho(s_0)\,\pi_\theta(a_0|s_0)\Big(\mathbb{E}_{\pi_\theta}\big[R_0\big|^{s_0}_{a_0}\big]\,\nabla_\theta\ln\pi_\theta(a_0|s_0) + \gamma\mathbb{E}_{\pi_\theta}\big[R_1\,\nabla_\theta\ln\pi_\theta(a_1|s_1)\big|^{s_0}_{a_0}\big] \\
&\qquad\qquad + \gamma^2\mathbb{E}_{\pi_\theta}\big[R_2\,\nabla_\theta\ln\pi_\theta(a_2|s_2)\big|^{s_0}_{a_0}\big] + \gamma^2\mathbb{E}_{\pi_\theta}\big[\nabla_\theta\mathbb{E}_{\pi_\theta}\big[R_2\big|^{s_2}_{a_2}\big]\big|^{s_0}_{a_0}\big]\Big)ds_0\,da_0 \\
&\;\vdots \\
&= \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{n-1}\gamma^t\,R_t\,\nabla_\theta\ln\pi_\theta(a_t|s_t)\Big]
\end{aligned}$$


TO DO SCIENCE, YOU GENERATE A HYPOTHESIS, THEN TEST IT.
BUT HOW DO YOU GENERATE A HYPOTHESIS?
GREAT QUESTION. HOW DO *YOU* THINK YOU DO IT?
WELL, MAYBE YOU–
AND THERE YOU HAVE IT!

`https://xkcd.com/2569`

Total 17 points.