Deep reinforcement learning

All assignments (questions marked with an A that yield points) must be submitted on Brightspace before the corresponding tutorial begins (see due date). Please submit all answers in one PDF file. Scans of handwritten solutions (e.g. for math answers) are permitted, but must be readable and have a file-size below 5MB. Do not submit the voluntary exercises (marked with an E), which will not earn you any points, but can help you practice the math and prepare for the exam.

You will also be asked to implement and test something in python/pytorch. We recommend that you use a Jupyter Notebook, convert your final version (including result plots) to PDF ("Download as" → "PDF via LaTeX") and attach the PDF at the end of your submission PDF. You are welcome to use other editors, but please make sure you submit the code (or the crucial code segments) and the results in an easily readable format together with your theory answers as one PDF file.

A sample solution will be published after the tutorial. Please always demonstrate how you arrived at your solution. You will receive the points when you convince us that you have seriously attempted to answer the question, even if your answer is wrong. You qualify for the exam when you have earned 75% of the total achievable points (sum over all 4 exercise sheets, not for individual sheets).

Good luck!

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 5 million environment steps (more is better). This can take a while, expect 1-3 hours of computation time. Do you get similar results as shown in the lecture?

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!

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:

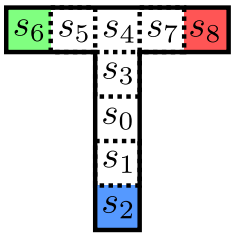
$$X'_{l,a,b} \quad := \quad \text{CNN}(\mathbf{X}, \boldsymbol{\mathcal{K}})_{l,a,b} \quad := \quad \sum_{i=1}^{H_K}\sum_{j=1}^{W_K}\sum_{p=1}^{3} \mathcal{K}_{l,p,i,j}\, X_{p,a+i-1,b+j-1}$$

(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, $V_{\binom{a}{b},\binom{p}{i}{j}} := X_{p,a+i-1,b+j-1}$. 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?

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.

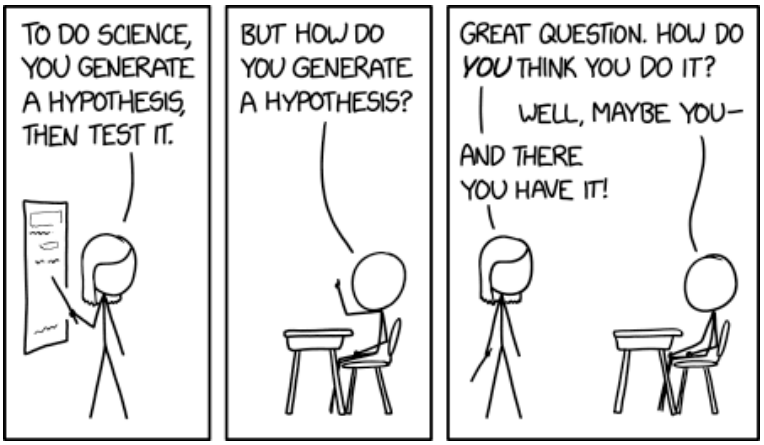
A2.5: Policy Gradient Theorem **(3 points)**

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

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



https://xkcd.com/2569

Total 17 points.