CS 446 / ECE 449 — Homework 6

your NetID here

Version 2.0

Instructions.

- Homework is due **Tuesday, April 25, at noon CST**; no late homework accepted.
- Everyone must submit individually on Gradescope under `hw6` and `hw6code`.
- The "written" submission at `hw6` **must be typed**, and submitted in any format Gradescope accepts (to be safe, submit a PDF). You may use LaTeX, Markdown, Google Docs, MS Word, whatever you like; but it must be typed!
- When submitting at `hw6`, Gradescope will ask you to select pages for each problem; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full academic integrity information. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw6code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- Coding problems come with suggested "library routines"; we include these to reduce your time fishing around APIs, but you are free to use other APIs.
- When submitting to `hw6code`, only upload the two python files `hw6.py` and `hw6_utils.py`. Don't upload a zip file or additional files.

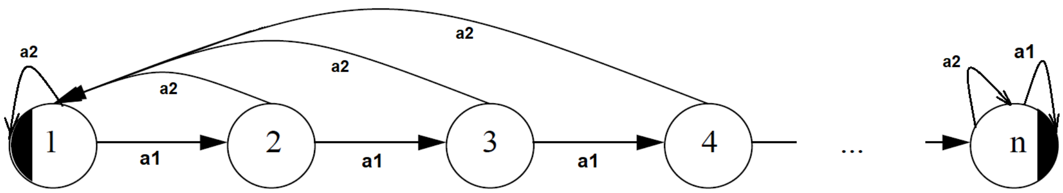
Version history.

2.0. Updated version.

1. Combination Lock.

Consider an MDP with n states $s_1, s_2, \ldots, s_n$, where there are 2 actions a_1, a_2 at each state. At each state s_i, $i < n$ action a_1 takes the agent to the s_{i+1} and for state s_n, action a_1 is a self-loop which keeps the agent in s_n. At any state, action a_2 takes the agent back to s_1, except for the last state, which again has a self-loop. See the figure for an overall picture of the setting. $R(s_i, a_j)$ (reward of action a_j at state s_i) is 0, except for (s_n, a_1), which has a value of 1. The agent takes one action at each step.

With *uniform random policy*, the agent at each state chooses one of the available actions uniformly at random. Now considering this *combination lock* MDP, answer the questions below. You need to show your work to receive full credit for each of the sections.



(a) Compute the expected number of steps for the uniform random policy to go from state s_1 to state s_n.

(b) Compute the formula for $Q(s_i, a_j)$, $\forall i, j$ for the uniform random policy considering a discounted reward setting with a discount factor of γ.

(c) Prove that:

$$\forall i < n : \; Q(s_i, a_1) > Q(s_i, a_2).$$

(d) Now consider a new *greedy policy* using the $Q(s, a)$ function you computed in the previous part. Specifically, $\pi(s_i) = argmax_a Q(s_i, a)$ (i.e., the agent, at each state, chooses the action with the highest value of the Q function computed in part (b)). Compute the new expected number of steps to get from state s_1 to s_k.

Hint: Use part (c); In particular, the estimate you get should have a simple form and be much smaller.

Solution.

(a)

(b)

(c)

(d)

2. Attention.

In this problem, you are given a set of binary vectors that are mapped to another set of binary vectors of the same length. You will construct convolutional and attention-based models to learn and predict this mapping. Training data and the code for training and testing your model are provided for you.

(a) Implement a model that uses convolutional layer with the given kernel size and ouput channels as inputs to generating the embedding for a given sequence. After using ReLU activation function on the embedding, use a Transposed Convolution module to generate the output vector of the same length as the original input and a single channel.

Remark: Consider using nn.Conv1d, nn.ConvTranspose1d, and torch.reshape functions.

(b) Implement an Attention-based model with a single head. The general structure is provided for you. You have to compute the queries, keys, and values using linear layers with no bias and complete the "compute_new_values" function that computes the attention matrix and the new values.

Library routines: nn.Linear, F.softmax, torch.matmul, and transpose()

(c) Use your convolutional model with a kernel size of 10 (length of the binary sequences) and another convolutional model with a kernel size of 3. Train both for 25 epochs and plot the training accuracy curves against each other (in the same plot) for both models. Explain your observations and reason about them.

(d) Train the attention model once with the positional encoding flag set to True (the default value) and once with this flag set to False. Plot the training accuracy curves against each other (in the same plot) for both models. Explain your observations and reason about them.

(e) Plot the attention matrix. Can you explain in words how the sequences are mapped? Include your figure. Also, discuss how using attention and attention values are different from convolutions.

Note: Make sure not to plot the attention matrix after the model is called on the test data in the test function (you can either comment out the call to the test function or change the number of epochs to a value that the procedure does not end with a call to test function).

Solution.

(c)

(d)

(e)