Assignment 3: Graphical Models /Recurrent Neural Networks/ Reinforcement Learning

Due April 3 at 11:59pm

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use Google to find solutions for assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment.

Submitting Your Assignment

The assignment must be submitted online at `https://coursys.cs.sfu.ca`. You must submit two files:

1. An assignment report in **PDF format**, named `report.pdf`. This report should contain the solutions to questions 1-3.
2. Your code for question 3, named `cartpole_stabilize.py`.

---

1 Graphical Models (22 marks)

Consider the problem of determining whether a local high school student will attend SFU or not. Define a boolean random variable A (true if the person will attend SFU), discrete random variables L (maximum of parents' education level: can take values o for non-university or u for university) and G (current provincial government: l for Liberal Party, d for NDP), and continuous valued random variables E (current provincial economy size) and T (SFU tuition level).

1. **4 marks**. Draw a simple Bayesian network for this domain.
2. **2 marks**. Write the factored representation for the joint distribution $p(A, L, G, E, T)$ that is described by your Bayesian network.
3. **8 marks**. Supply all necessary conditional distributions. Provide the type of distribution that should be used and give rough guidance / example values for parameters (do this by hand, educated guesses).
4. **8 marks**. Suppose we had a training set and wanted to **learn** the parameters of the distributions using maximum likelihood. Denote each of the N examples with its values for each random variable by $\boldsymbol{x}_n = (a_n, l_n, g_n, e_n, t_n)$. The training set is $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.

   Which elements of the training data are needed to learn the parameters for $p(A|pa_A)$? Why?

   (Note that pa_A denotes parents of A.)

   Start by writing down the likelihood and argue from there.

2 Gated Recurrent Unit (10 marks)

A Gated Recurrent Unit (GRU) is another type of recurrent neural network unit with the ability to remember and forget components of the state vector (see Cho et al. EMNLP 2014 `https://arxiv.org/abs/1406.1078`).

Read Sec. 2.3 of the linked paper for the description of the GRU. Note that the GRU's state consists of a vector of $\mathbf{h}$ values. There are two gates, r_j and z_j, which control the update of h_j, the j^{th} component of the GRU state.

- What values of r_j and z_j would cause the new state for h_j to be similar to its old state? Give a short, qualitative answer.
- If r_j and z_j are both close to 0, how would the state for h_j be updated? Give a short, qualitative answer.

3 Reinforcement Learning (17 marks)

This question guides you through implementing the policy gradient algorithm with average reward baseline.

Preparation:

- Install gym and TensorFlow for Python. Documentation can be found at `https://gym.openai.com/` and `https://www.tensorflow.org/install`.
- Replace `cartpole.py` in gym with the version provided. The included file `cartpole_stabilize.py` contains the skeleton code for training a cartpole to achieve its goal of keeping its position centred and pole upright.

The cartpole environment consists of a rotatable pole mounted on top of a cart. The states of the system are the position and velocity (x, v) of the cart, and the angular position and velocity (θ, ω) of the pole. The two possible actions are to push the cart left or right with a constant force.

Our goal in this problem is to keep the cart's position near zero and the pole near upright for as long as possible. To encourage this, in the custom environment defined in the provided `cartpole.py` file, the cartpole system receives a reward of 1 for every time step in which its state satisfies $(|x| \leq 0.5 \quad \text{and} \quad |\theta| \leq \frac{4\pi}{180})$. Training episodes terminate when the system state violates $(|x| \leq 1.5 \quad \text{or} \quad |\theta| \leq \frac{12\pi}{180})$.

1. In the `__init__` method of the agent class, define a policy network that takes as input the state, has two fully hidden layers of the desired number of neurons with ReLU activation, and outputs the probability distribution of applying the two possible actions.
2. In the `__init__` method of the agent class, compute the probability of applying the actions in the input data.
3. In the `__init__` method of the agent class, define the loss function such that its gradient is $(\nabla_\theta J(\theta))$.
4. Complete the compute_advantage function, which should compute a list of advantage values $(A_t = \Sigma_{t' \geq t} \gamma^{t'-t} r(s_t, a_t) - b)$ for every time step across a batch of episodes, where $(b = \mathbb{E}_{\tau \sim p(\tau;\theta)} \Sigma_{t \geq 0} \gamma^t r(s_t, a_t))$ is the average reward across the batch of episodes. Note that the batch size is specified by the `update_frequency` variable.
5. Complete the main part of the script (fill in the unmodified `cartpole_stabilize.py` at lines 73-78, 104-107, 122-124).
6. Produce several plots showing the state of cart-pole system at different snapshots in time for a well-performing episode.
7. Produce a plot showing sum of discounted reward in each episode vs. episode number.

4 Attention Models (Optional)

As an alternative to recurrent neural network structures, attention models can be used to analyze an input sequence directly to compute a sequence of output state representations.

If you are interested in learning more, consider reading Vaswani et al. NIPS 2017 `https://arxiv.org/abs/1706.03762`.