Homework 5

Rademacher Complexity, MDPs, Policy Optimization

CMU 10-701: Machine Learning (Spring 2021)

piazza.com/cmu/spring2021/10701
OUT: April 23, 2021

DUE: Wednesday, May 05, 2021 11:59pm
TAs: Arundhati Banerjee, Jeffrey Tsaw, Zhe Chen

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: https://www.cs.cmu.edu/~aarti/Class/10701_Spring21/index.html.
- **Extension Policy:** See the homework extension policy here: https://www.cs.cmu.edu/~aarti/Class/10701_Spring21/index.html.
- **Submitting your work:**
  - All portions of the assignments should be submitted to Gradescope (https://gradescope.com/).
  - **Programming:** We will autograde your Python code in Gradescope. After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). We recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly before any submission. Our autograder requires that you write your code using Python 3.6.9 and Numpy 1.17.0.
  - **Written questions:** You must type the answers in the provided .tex file. Handwritten solutions will not be accepted. Make sure to answer each question in the provided box. DO NOT change the size of the boxes, because it may mess up the autograder. Upon submission, make sure to label each question using the template provided by Gradescope. Please make sure to assign ALL pages corresponding to each question.

1 Rademacher Complexity [30 pts total]

1. **[12 pts]** Assume $\mathcal{G}$, $\mathcal{G}_1$,$\mathcal{G}_2 \subseteq \mathbb{R}^{\mathcal{Z}}$ are classes of real valued functions on $\mathcal{Z}$ and $z \in \mathcal{Z}^n$. Which of the following always hold for the empirical Rademacher complexity w.r.t z defined as $\hat{\mathcal{R}}_z(\mathcal{G}) = \mathbb{E}_\epsilon[\sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^n \epsilon_i g(z_i)]$?

   ■ If constants $b, c \in \mathbb{R}$, then $\hat{\mathcal{R}}(c\mathcal{G} + b) = |c|\hat{\mathcal{R}}(\mathcal{G})$

   □ If constants $b, c \in \mathbb{R}$, then $\hat{\mathcal{R}}(c\mathcal{G} + b) = c\hat{\mathcal{R}}(\mathcal{G})$

   □ If constants $b, c \in \mathbb{R}$, then $\hat{\mathcal{R}}(c\mathcal{G} + b) = c\hat{\mathcal{R}}(\mathcal{G}) + b$

   ■ $\hat{\mathcal{R}}(\mathcal{G}_1 + \mathcal{G}_2) = \hat{\mathcal{R}}(\mathcal{G}_1) + \hat{\mathcal{R}}(\mathcal{G}_2)$

   □ $\hat{\mathcal{R}}(\mathcal{G}_1 + \mathcal{G}_2) \leq \hat{\mathcal{R}}(\mathcal{G}_1) + \hat{\mathcal{R}}(\mathcal{G}_2)$

   ■ if $\mathcal{G}_1 \subseteq \mathcal{G}_2$, then $\hat{\mathcal{R}}(\mathcal{G}_1) \leq \hat{\mathcal{R}}(\mathcal{G}_2)$

   □ if $\mathcal{G}_1 \subseteq \mathcal{G}_2$, then $\hat{\mathcal{R}}(\mathcal{G}_1) \geq \hat{\mathcal{R}}(\mathcal{G}_2)$

2. **[4 pts]** Consider a function $\psi : \mathbb{R} \to \mathbb{R}$ which is L_ψ-Lipschitz. $\mathcal{G}$ is a class of real valued functions and $\mathcal{F} = \{f \mid f = \psi(g), g \in \mathcal{G}\}$. $\hat{\mathcal{R}}(\mathcal{G})$ is the empirical Rademacher complexity of $\mathcal{G}$ and $\hat{\mathcal{R}}(\mathcal{F})$ is the empirical Rademacher complexity of $\mathcal{F}$. How are $\hat{\mathcal{R}}(\mathcal{F})$ and $\hat{\mathcal{R}}(\mathcal{G})$ related? Please write your final answer only in the form of the most simplified relationship involving $\hat{\mathcal{R}}(\mathcal{F})$, $\hat{\mathcal{R}}(\mathcal{G})$ and L_ψ.

   $\hat{\mathcal{R}}(\mathcal{F}) \leq L_\psi \hat{\mathcal{R}}(\mathcal{G})$

3. Suppose we are trying to bound the empirical Rademacher complexity of a 2-layer neural network. We have a d-dimensional input x to the network, with a bound on $\|x\|_\infty$. The first layer has weight matrix $\mathbf{w}$ resulting in the pre-activations $\mathbf{w}x$.

   (a) **[6 pts]** Assume $\mathcal{F}$ is the class of linear predictors $\mathcal{F} = \{\langle w, x\rangle \mid ||w||_1 \leq W, x \in \mathbb{R}^d, ||x||_\infty \leq X\}$ over the set $\{x_1, x_2, \ldots, x_m\}$. The Rademacher complexity of $\mathcal{F}$ is upper bounded by $\hat{\mathcal{R}}(\mathcal{F}) \leq C$ where C is a tight upper bound. Select all correct options.

   □ $C \propto \sqrt{d}$

   □ $C \propto \sqrt{\log d}$

   □ $C \propto \sqrt{m}$

   ■ $C \propto \frac{1}{\sqrt{m}}$

   ■ $C \propto XW$

   □ $C \propto \sqrt{XW}$

   □ $C \propto \frac{X}{W}$

(b) **[2 pts]** Let us represent by $f'(x)$ the output of the first layer, the linear computation of the pre-activations followed by an activation $\phi : \mathbb{R} \to \mathbb{R}$ that is L_ϕ-Lipschitz. Assuming $\mathcal{F}' = \{f'\}$, and the L_1 norm of each row of $\mathbf{w}$ is bounded by W, what is the tightest upper bound on $\mathcal{R}(\mathcal{F}')$? You do not need to show the derivation, only the final upper bound expression.

$L_\phi X W \sqrt{\frac{2 \log 2d}{m}}$

(c) **[6 pts]** The second layer is a linear function followed by an activation function $\psi : \mathbb{R} \to \mathbb{R}$ that is L_ψ-Lipschitz. It can be defined by the class of functions $\mathcal{G} = \{\psi(\sum_j w'_j f'_j(x)), ||w'_j||_1 \leq b, f'_j \in \mathcal{F}'\}$. What is the tightest upper bound on $\mathcal{R}(\mathcal{G})$ in terms of $\mathcal{R}(\mathcal{F}')$ (you should not substitute in the expression for $\mathcal{R}(\mathcal{F}')$)? You do not need to show the derivation, only the final upper bound expression is needed.

$2 L_\psi \mathcal{R}(\mathcal{F}') b$

2 Markov Decision Process [20 points, 2 per box]

Consider an MDP as follows:

$$S = \{S_1, S_2, S_3, S_4, S_5\}$$

$$A = \{a_1, a_2\}$$

The state transition probabilities are as follows where the entry at location (i,j) denotes the probability of transition from S_i to S_j.

	S_1	S_2	S_3	S_4	S_5
S_1	0	1	0	0	0
S_2	0	0	1	0	0
S_3	0	0	0.5	0.5	0
S_4	0	0	0	0	1
S_5	0	0	0	0	1

Table 1: $T(S, a_1, S')$

	S_1	S_2	S_3	S_4	S_5
S_1	0	0	1	0	0
S_2	0	0	0	0	1
S_3	0	1	0	0	0
S_4	0.7	0	0.3	0	0
S_5	0	0	0	0	1

Table 2: $T(S, a_2, S')$

The reward function R indicates getting a reward $R(s)$ on reaching state s. Please note that this is an infinite horizon problem with no terminal state.

$$R(S_1) = 0$$

$$R(S_2) = 1$$

$$R(S_3) = -1$$

$$R(S_4) = 1$$

$$R(S_5) = -1$$

Consider a policy π such that $\pi(a_1) = 0.6$ and $\pi(a_2) = 0.4$ for all states.

Assume initial Q-values for all state-action pairs to be set to 0. Take discount factor $\gamma = 0.9$.

For the following questions, there is no need to write code; you may work them out on paper or use a calculator or code as desired. Wherever needed, you should round the final answer to 4 decimal places.

We will compute the value functions over the states of our MDP using parallel (aka synchronous) Bellman backups.

After the first iteration of parallel Bellman backup updates, what are the value functions $V_1(S_i)$ of states S_i:

1. $V_1(S_1)$ 0.2
2. $V_1(S_2)$ -1
3. $V_1(S_3)$ 0.4
4. $V_1(S_4)$ -0.72
5. $V_1(S_5)$ -1

After the second iteration of parallel Bellman backup updates, what are the value functions $V_2(S_i)$ of states S_i:

6. $V_2(S_1)$ -0.196
7. $V_2(S_2)$ -1.144
8. $V_2(S_3)$ -0.0464
9. $V_2(S_4)$ -1.1664
10. $V_2(S_5)$ -1.9

3 REINFORCE [50 pts total]

3.1 Implementation [36 pts]

In this section, you will implement episodic REINFORCE[1], a policy-gradient learning algorithm, and use it to train a policy to solve the `LunarLander-v2` problem.

Note: training a good policy can take some time (at least 2 hours), so please start early. Most of the compute time will be spent on simulation and your network doesn't have to be very big (we'll recommend one below), so you don't need a GPU for this.

You will use PyTorch and OpenAI Gym to help you train your policy. Please write your code in `reinforce.py`; Apart from the `compute_expected_cost` function (you may add parameters with default values) and all class names, which are autograded, feel free to change other method signatures as you wish.

To set up the training environment, you'll need the following packages:

- PyTorch. Any reasonably up-to-date version should work.
- Box2D: Required for the Lunar lander environment. `pip install Box2D`
- gym: Open AI gym simulator. `pip install gym`

3.2 REINFORCE algorithm

Policy gradient methods directly optimize the policy $\pi(A \mid S, \theta)$, which is parameterized by θ. The REINFORCE algorithm proceeds as follows. We generate an episode by following policy π, sampling our action from its probability distribution output at each step. After each episode ends, for each time step t during that episode, we accumulate the the REINFORCE update. This update is proportional to the product of the return G_t experienced from time step t until the end of the episode and the gradient of $\log \pi(A_t \mid S_t, \theta)$. Finally we update the policy parameters θ using the computed direction. See Algorithm 1 for details — but note that you will be implementing a slightly modified version to reduce variance (see below).

procedure REINFORCE
  Start with policy model π_θ
  repeat: iteration i
    Generate an episode $S_0, A_0, r_0, \ldots, S_{T-1}, A_{T-1}, r_{T-1}$ *following* $\pi_\theta(\cdot)$
    for t *from* $T-1$ *to* 0:
      $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$
    $\hat{J}_i(\theta) = -\frac{1}{T} \sum_{t=0}^{T-1} G_t \ln \pi_\theta(A_t|S_t)$
    $\theta \leftarrow \theta -$ *learning rate* $\times \nabla \hat{J}_i(\theta)$
end procedure

Algorithm 1: REINFORCE

[1] Ronald J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, 1992, p229-256

3.3 Starter Guide

For the policy model $\pi(A \mid S, \theta)$, we recommend starting with a model that has:

- three hidden layers with 16 units each, each followed by ReLU activations
- a final output of 4 units (the number of actions)
- a softmax layer (so the output is a proper distribution)

Initialize bias for each layer to zero. We recommend using Kaiming initialization for weights. HINT: Read the PyTorch documentation. The `model.apply` method should help you.

You can use `print(model)` to inspect the model architecture.

You can choose which optimizer and hyperparameters to use, so long as they work for learning on `LunarLander-v2`. We recommend using Adam as the optimizer. It will automatically adjust the learning rate based on the statistics of the gradients it's observing. You can think of it like a fancier SGD with momentum. PyTorch provides a version of Adam https://pytorch.org/docs/stable/optim.htm. You don't have to tune the other parameters apart from the learning rate.

Note: Be sure to check out the "Additional Guidelines on Implementation" section for more tips.

Toy environment: First test your implementation on a simple toy environment `BanditEnv` defined in `reinforce.py`. You will not be graded for this; testing on this environment serves as a sanity check for your implementation since the optimal policy will be learnt extremely fast. As it is a bandit set up, each 'episode' (pull of an arm/action) is of length one and the state is fixed. At convergence, your policy would get a reward of around 0 (will be close to 0 but not equal) consistently for 100 consecutive pulls/actions/episodes, which can be expected within 10k episodes. Because the bandit arms have no effect on each other, setting $\gamma = 0$ would work well in this setting. Confirm that your implementation works as expected on the toy environment before proceeding further. Note: Do not rescale the return by its variance here due to having only 1 step per episode.

Training environment: the `LunarLander-v2` agent (https://gym.openai.com/envs/LunarLander-v2/) produces **8** observations at each step: 1. horizontal coordinate; 2. vertical coordinate; 3. horizontal speed; 4. vertical speed; 5. angle; 6. angular speed; 7. 1 if first leg has contact, else 0; 8. 1 if second leg has contact, else 0. The agent can take one of **4** actions at each step: 1. do nothing; 2. fire left orientation engine; 3. fire main engine; 4. fire right orientation engine. The goal of the lander is to land in a upright position on the landing pad, located at coordinate (0, 0).

Train your implementation on the environment until convergence. An episode is considered solved if your implementation can attain a reward of at least 200. Be sure to keep training your policy for at least 700 more episodes after it reaches 200 reward so that you are sure it consistently achieves 200 reward.

Sometimes the agent learns to 'hover' in this environment, leading to long episodes causing

a bottleneck in training. If the reward has stagnated below 160 for more than 500 episodes, consider reinitializing the network and restart training.

Implementing a simple baseline: As we mentioned in class, the vanilla REINFORCE algorithm may have high variance of the gradient estimates. This leads to instability (with a larger learning rate) or slow progress (with a smaller learning rate). Also, the discounted return G_t can be widely different in scale for different policies. This scale difference makes it difficult to select a good learning rate.

To mitigate these problems, we'll subtract from G_t a baseline B, that is a running mean discounted return, and rescale it by its standard deviation, which is effectively an adaptive learning rate, before calculating $\hat{J}$:

procedure SIMPLEBASELINE
  $B = 0$
  in iteration i**:**
    $\mu = \frac{1}{n}\sum_t G_t$
    $\sigma = \sqrt{\frac{1}{n}\sum_t (G_t - \mu)^2}$
    $G_t \leftarrow (G_t - B)/\sigma$
    $B \leftarrow pB + (1-p)\mu$
end procedure

Algorithm 2: Simple Baseline

You can choose your own p or use a different way of calculating running average. We recommend starting with $p = 0.99$. The sums are over all of the time steps of the current trajectory.

3.4 Submission and Autograding

You will submit the following 2 files:

- Completed `reinforce.py`
- `mypolicy.pth` containing the `state_dict` of your final policy defined in `reinforce.py`

You can either submit a .zip file (make sure the 2 files are not in any sub-directories) or just drag-and-drop the 2 files onto Gradescope.

The autograder has 2 tests:

1. [**20 pts**] `compute_expected_cost` function. It will be tested on the correctness of the expected cost as well as backpropagated gradients.
    - Note: This test assumes that you implemented standard deviation rescaling as described above. Baseline parameter will be 0.
2. [**16 pts**] Effectiveness of your trained policy.
    - If your policy solved LunarLander 80 out of 100 times or more, you get full score.

- If your policy solved LunarLander 20 out of 100 times or less, you get 0.
- Otherwise, your score is linearly interpolated based on the number of successful solutions.
- Note: the autograder's simulations are seeded such that the results are more or less deterministic.

3.5 Written Questions

Answer all of the following questions using the training environment, i.e., `LunarLander-v2`, unless otherwise stated.

1. **[2 pts]** Consider a shallow policy with a 8×4 linear layer followed by a softmax layer. Assume all inputs to the policy are positive. In a horizon-1 episode, the four different actions' probabilities and rewards are shown below. Which of these actions, if we happen to choose it, will result in the largest (positive) gradient updates to the policy's weight matrix when trained using REINFORCE?

   ○ $p = 0.12, \text{reward} = 20$

   ○ $p = 0.56, \text{reward} = -50$

   ● $p = 0.24, \text{reward} = -100$

   ○ $p = 0.08, \text{reward} = 70$

2. **[2 pts]** The baseline strategy given above reduces the variance of our gradient estimates. Try training a policy **without** using baseline for at least 1000 episodes and briefly report your observations. What other strategies might we use to reduce variance? Please give one or two strategies, with no more than a sentence of explanation for each strategy.

   I can see that the rewards vary a lot among different episodes without using the baseline.
   We can use advantage actor critic method to reduce variance.
   Explanation:
   We obtain advantage function by estimating state value function and action value function, replacing some terms in original algorithm with their expectations, and therefore reduce variance.

3. **[4 pts]** Consider a hallway problem: $N = 25$ states in a line, with actions that move the agent left or right. The agent starts at the leftmost state. The cost at each step is zero, except for -1 when we reach the rightmost state, at which point the simulation ends. Trying to move left past the leftmost state does nothing. The horizon is 200, with no discounting ($\gamma = 1$). The policy class is just a constant probability θ of going right at each state (the same at every state; note that $1 - \theta$ is therefore the probability of going left). The initial policy is $\theta = 0.5$. Imagine running REINFORCE on each of the following environment variations:

   ○ The environment as described above.

- ● The same environment, but the left action moves two states to the left (while the right action continues to move one state to the right).
- ○ The original environment (moving one state at a time in either direction), but with $N = 30$ states instead.

In which of these environments will REINFORCE take longest to find a near-optimal value of θ? Why? (You can assume that we pick good hyperparameters separately for each environment.)

Since the cost is 0 except for the ending state, we only make gradient update if we can reach the rightmost state.
In the second environment, it takes the most steps in expectation for our policy to reach the rightmost state.
Therefore, REINFORCE takes the longest time to find near-optimal value of θ in the second environment.

4. **[2 pts]** Describe your implementation, including the optimizer and any hyperparameters you used (learning rate, γ, layer sizes, activations and anything else that's relevant). Your description should be detailed enough that someone could reproduce your results.

I used Adam optimizer with with learning rate 0.001.
My model consists of 4 linear layers. The first linear layer has input size of 8 and output size of 16. The second and the third layer both have input size of 16 and output size of 16. Each of these 3 layers is followed by a ReLU layer. The fourth linear layer has input size of 16, and output size of 4. Then it is followed by a softmax layer.
My γ is 0.99 and I trained for 10000 episodes.

5. **[4 pts]** Partway through a training run, Zhe notices that his policy almost always outputs the following probabilities at every step: $[10^{-5}, 10^{-3}, 0.9989, 10^{-23}]$ after 500 episodes. Can further training (with reward normalization as described earlier) make this policy capable of solving the environment? Explain why in one sentence. If your answer is "no", offer him one helpful suggestion.

No. Further training cannot make this policy solve the environment.
The policy does not explore enough, and I think we can reinitialize the network and/or adjust the learning rate.

3.6 Additional Guides on Implementation

(optional) Training progress logging: Training will take a long time. You can make use of TensorBoard to keep track of the training progress. https://www.tensorflow.org/tensorboard/get_started. Printing out running reward over the past 20 to 50 episodes periodically would also work.

Some hyperparameter and implementation tips and tricks:

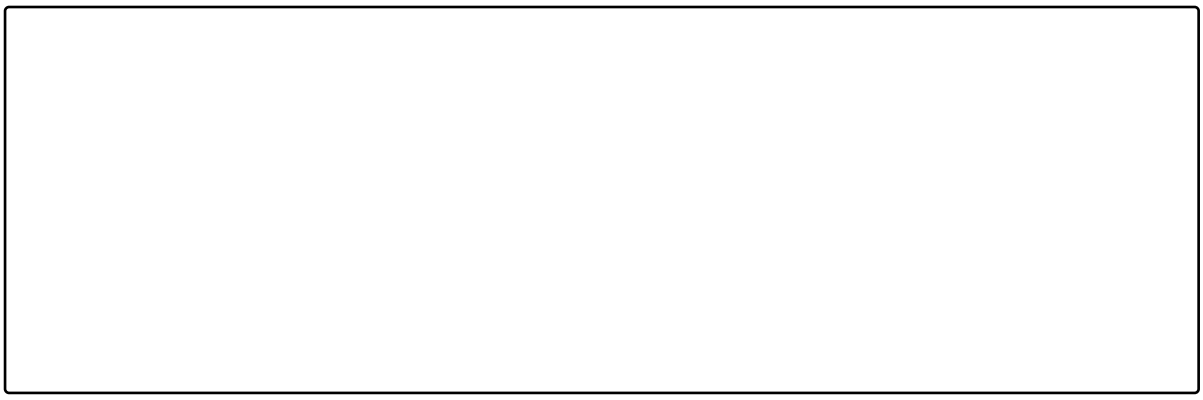
- For efficiency, you should try to vectorize your code as much as possible and use **as few loops as you can** in your code. For example, in lines 5 and 6 of Algorithm 1 (REINFORCE) you should not use two nested loops. How can you formulate a single loop to calculate the cumulative discounted rewards? Hint: Think backwards!
- Like the previously mentioned reward normalization, if needed, batch normalization between layers can improve stability and convergence rate of both REINFORCE and ordinary SGD. PyTorch has a built-in batch normalization layer `https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html`.
- `nn.Sequential` is strongly recommended for defining the policy model here as it's very easy to add/remove layers and you do not need a complex network.
- PyTorch's backpropagation, i.e., the `loss.backward()` call, works only for an 1-element tensor (which is the loss value).
- Make sure your policy gradient computation is correct before full-on training. Use the toy environment and simpler environments in the gym like `CartPole-v1`, which can be trained consistently in about 600 episodes, to verify your implementation first.
- Feel free to experiment with different policy architectures. Increasing the number of hidden units in earlier layers may improve performance.
- We recommend using a discount factor of $\gamma = 0.99$, unless specified otherwise in a subquestion.
- Try out different learning rates. A good place to start is in the range [`1e-5`, `1e-3`].
- Policy gradient algorithms can be fairly noisy. You may have to run your code for several thousand or more training episodes to see a consistent improvement for REINFORCE.
- Instead of training one episode at a time, you can try generating a fixed number of steps in the environment, possibly encompassing several episodes, and training on such a batch instead.

Collaboration Questions Please answer the following:

1. Did you receive any help whatsoever from anyone in solving this assignment?
   Yes / **No**.
   - If you answered 'yes', give full details: _____
   - (e.g. "Jane Doe explained to me what is asked in Question 3.4")

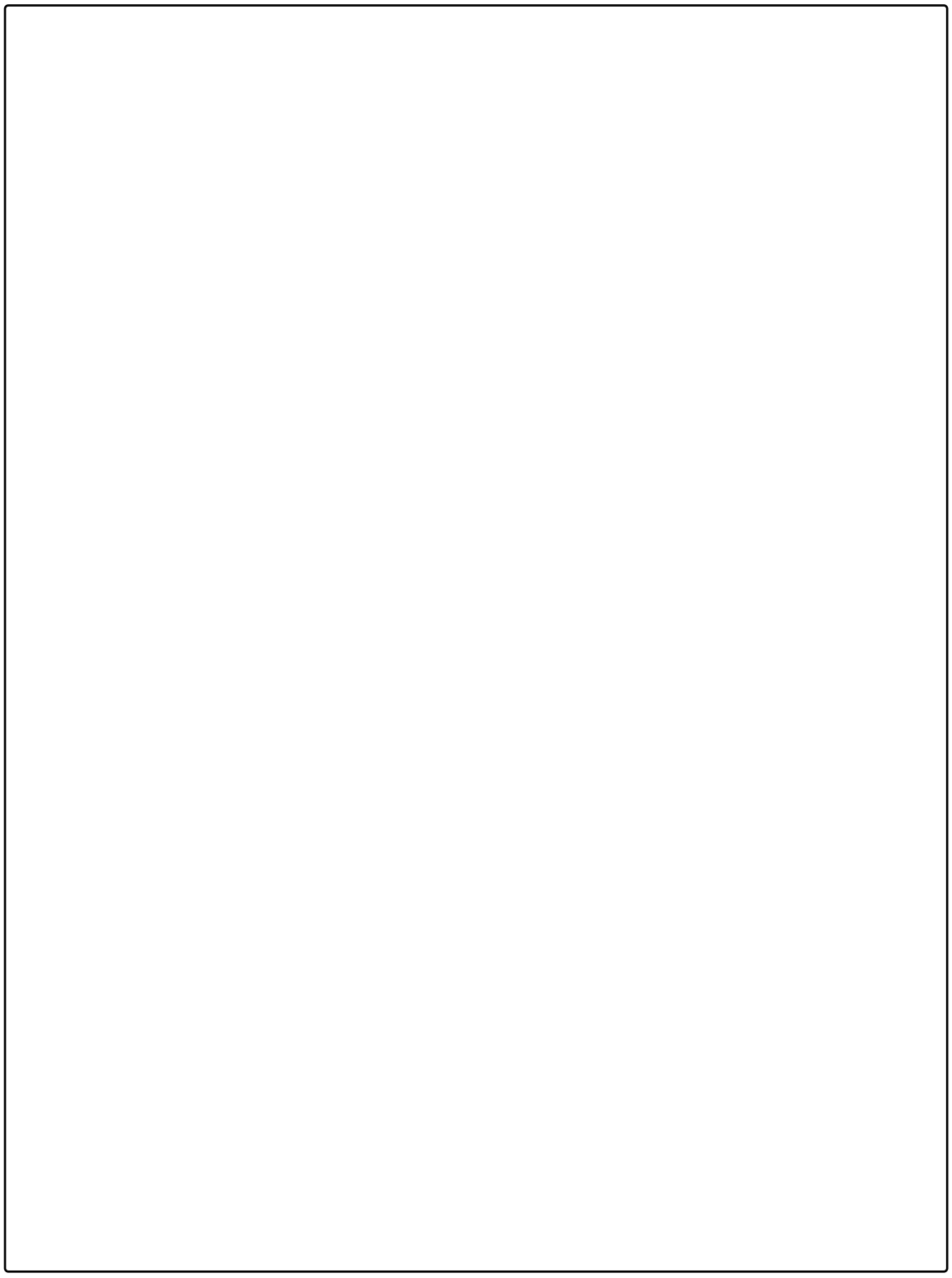
2. Did you give any help whatsoever to anyone in solving this assignment?
   Yes / **No**.
   - If you answered 'yes', give full details: _____
   - (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. Did you find or come across code that implements any part of this assignment ?
   Yes / **No**. (See below policy on "found code")
   - If you answered 'yes', give full details: _____
   - (book & page, URL & location within the page, etc.).



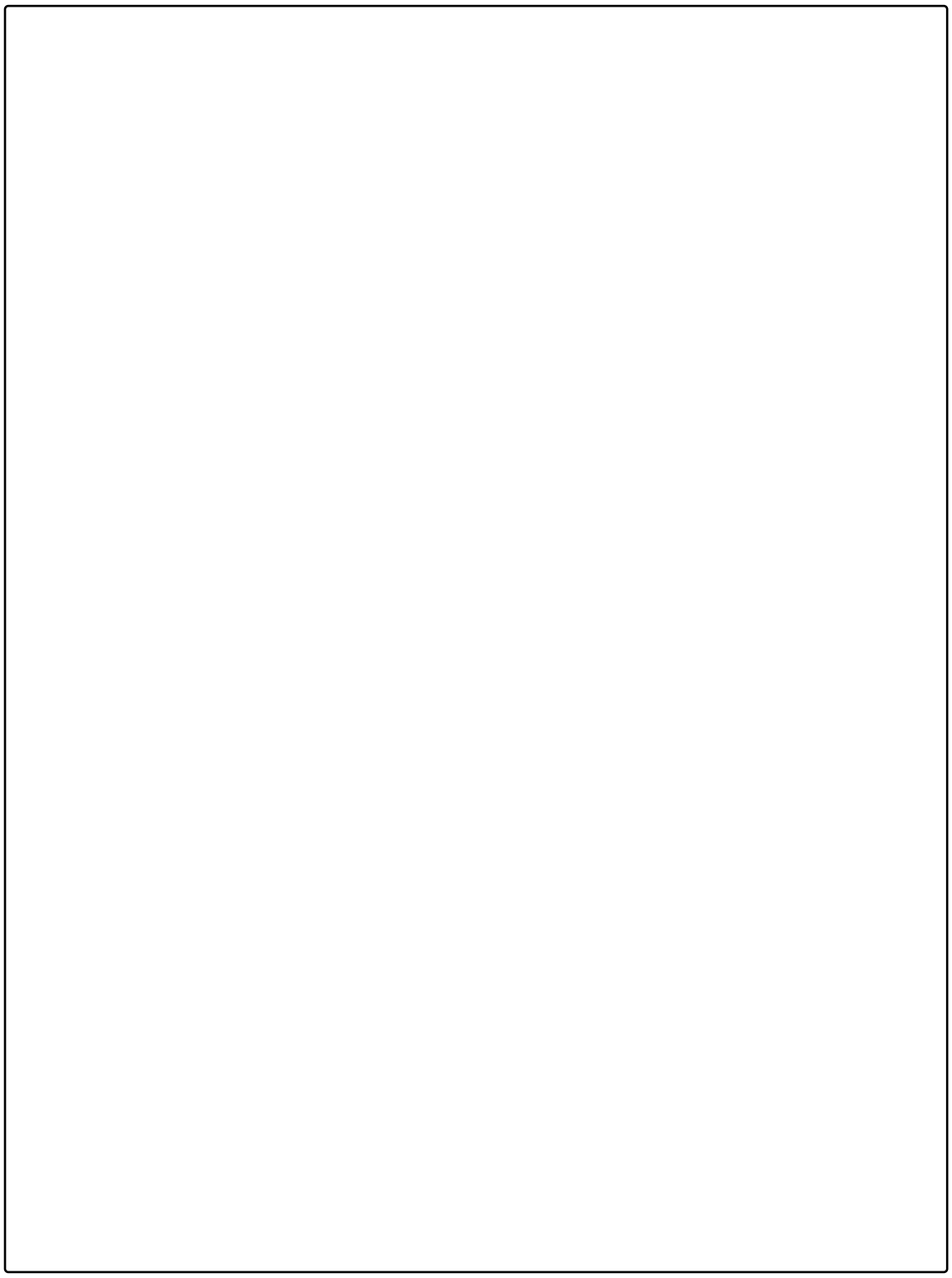
Overflow boxes

Section 1



Section 2

Section 3



Section 4