CS 234: Assignment #3

Due date: February 26, 2023 at 6:00 PM (18:00) PST

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. **However, each student must finish the problem set and programming assignment individually, and must turn in their assignment.** We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work is done by yourself. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code.

Please review any additional instructions posted on the assignment page at http://web.stanford.edu/class/cs234/assignments.html. When you are ready to submit, please follow the instructions on the course website.

1 Policy Gradient Methods (54 pts coding + 26 pts writeup)

The goal of this problem is to experiment with policy gradient and its variants, including variance reduction and off-policy methods. Your goals will be to set up policy gradient for both continuous and discrete environments, using a neural network baseline for variance reduction, and implement the off-policy Proximal Policy Optimization algorithm. The programming has detailed instructions for each implementation task, including a README for instructions to setup an environment, but an overview of key steps in the algorithm is provided here.

1.1 REINFORCE

Recall the policy gradient theorem,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]$$

REINFORCE is a Monte Carlo policy gradient algorithm, so we will be using the sampled returns G_t as unbiased estimates of $Q^{\pi_\theta}(s,a)$. The REINFORCE estimator can be expressed as the gradient of the following objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log\big(\pi_\theta(a_t^i|s_t^i)\big) G_t^i$$

where D is the set of all trajectories collected by policy π_θ, and $\tau^i = (s_0^i, a_0^i, r_0^i, s_1^i, \ldots, s_{T_i}^i, a_{T_i}^i, r_{T_i}^i)$ is trajectory i.

1.2 Baseline

One difficulty of training with the REINFORCE algorithm is that the Monte Carlo sampled return(s) G_t can have high variance. To reduce variance, we subtract a baseline $b_\phi(s)$ from the estimated returns when computing the policy gradient. A good baseline is the state value function, $V^{\pi_\theta}(s)$, which requires a training

update to ϕ to minimize the following mean-squared error loss:

$$L_{\text{MSE}}(\phi) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} (b_\phi(s_t^i) - G_t^i)^2$$

1.3 Advantage Normalization

After subtracting the baseline, we get the following new objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log\big(\pi_\theta(a_t^i|s_t^i)\big) \hat{A}_t^i$$

where

$$\hat{A}_t^i = G_t^i - b_\phi(s_t^i)$$

A second variance reduction technique is to normalize the computed advantages, $\hat{A}_t^i$, so that they have mean 0 and standard deviation 1. From a theoretical perspective, we can consider centering the advantages to be simply adjusting the advantages by a constant baseline, which does not change the policy gradient. Likewise, rescaling the advantages effectively changes the learning rate by a factor of $1/\sigma$, where σ is the standard deviation of the empirical advantages.

1.4 Proximal Policy Optimization

One might notice that the REINFORCE algorithm above (with or without a baseline function) is an on-policy algorithm; that is, we collect some number of trajectories under the current policy network parameters, use that data to perform a single batched policy gradient update, and then proceed to discard that data and repeat the same steps using the newly updated policy parameters. This is in stark contrast to an algorithm like DQN which stores all experiences collected over several past episodes. One might imagine that it could be useful to have a policy gradient algorithm "squeeze" a little more information out of each batch of trajectories sampled from the environment. Unfortunately, while the Q-learning update immediately allows for this, our derived REINFORCE estimator does not in its standard form.

Ideally, an off-policy policy gradient algorithm will allow us to do multiple parameter updates on the same batch of trajectory data. To get a suitable objective function that allows for this, we need to correct for the mismatch between the policy under which the data was collected and the policy being optimized with that data. Proximal Policy Optimization (PPO) restricts the magnitude of each update to the policy (i.e., through gradient descent) by ensuring the ratio of the current and former policies on the current batch is not too different. In doing so, PPO tries to prevent updates that are "too large" due to the off-policy data, which may lead to performance degradation. This technique is related to the idea of importance sampling which we will examine in detail later in the course. Consider the following ratio $r_t(\theta)$, which measures the probability ratio between a current policy π_θ (the "actor") and an old policy π_θ^{old}:

$$r_\theta(s_t^i, a_t^i) = \frac{\pi_\theta(a_t^i \mid s_t^i)}{\pi_{\theta_{\text{old}}}(a_t^i \mid s_t^i)}$$

To do so, we introduce the clipped PPO loss function, shown below, where $\text{clip}(x, a, b)$ outputs x if $a \leq x \leq b$, a if $x < a$, and b if $x > b$:

$$J_{\text{clip}}(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \min(r_\theta(s_t^i, a_t^i)\hat{A}_t^i, \text{clip}(r_\theta(s_t^i, a_t^i), 1-\epsilon, 1+\epsilon)\hat{A}_t^i)$$

where $\hat{A}_t^i = G_t^i - V_\phi(s_t^i)$. Note that in this context, we will refer to $V_\phi(s_t^i)$ as a "critic"; we will train this exactly like the baseline network described above.

To train the policy, we collect data in the environment using π_θ^{old} and apply gradient ascent on $J_{\text{clip}}(\theta)$ for each update. After every K updates to parameters $[\theta, \phi]$, we update the old policy π_θ^{old} to equal π_θ.

1.5 Coding Questions (50 pts)

The functions that you need to implement in `network_utils.py`, `policy.py`, `policy_gradient.py`, and `baseline_network.py` are enumerated here. Detailed instructions for each function can be found in the comments in each of these files.
Note: The "batch size" for all the arguments is $\sum T_i$ since we already flattened out all the episode observations, actions, and rewards for you.
In `network_utils.py`, you need to implement:

- `build_mlp`

In `policy.py`, you need to implement:

- `BasePolicy.act`
- `CategoricalPolicy.action_distribution`
- `GaussianPolicy.__init__`
- `GaussianPolicy.std`
- `GaussianPolicy.action_distribution`

In `policy_gradient.py`, you need to implement:

- `PolicyGradient.init_policy`
- `PolicyGradient.get_returns`
- `PolicyGradient.normalize_advantage`
- `PolicyGradient.update_policy`

In `baseline_network.py`, you need to implement:

- `BaselineNetwork.__init__`
- `BaselineNetwork.forward`
- `BaselineNetwork.calculate_advantage`
- `BaselineNetwork.update_baseline`

In `ppo.py`, you need to implement:

- `PPO.update_policy`

1.6 Debugging

To help debug and verify that your implementation is correct, we provide a set of sanity checks below that pass with a correct implementation. Note that these are not exhaustive (i.e., they do not verify that your implementation is correct) and that you may notice oscillation of the average reward across training.
Across most seeds:

- Policy gradient (without baseline) on Pendulum should achieve around an average reward of 100 by iteration 10.

- Policy gradient (with baseline) on Pendulum should achieve around an average reward of 700 by iteration 20.
- PPO on Pendulum should achieve an average reward of 200 by iteration 20.
- All methods should reach an average reward of 200 on Cartpole, 1000 on Pendulum, and 200 on Cheetah at some point.

1.7 Writeup Questions (26 pts)

(a) (3 pts) To compute the REINFORCE estimator, you will need to calculate the values $\{G_t\}_{t=1}^{T}$ (we drop the trajectory index i for simplicity), where

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Naively, computing all these values takes $O(T^2)$ time. Describe how to compute them in $O(T)$ time.
Solution:

Algorithm 1 REINFORCE get returns

Require: Trajectories $\tau = (s_0, a_0, r_0, \cdots, r_{T-1})$
  Generate a trajectories of rewards $\mathcal{R} = (r_0, \cdots, r_{T-1})$ from τ
  Initialize $G_T = 0$ and an empty list $L = []$
  for $t \leftarrow T-1$ to 0 **do**
    $G_t \leftarrow r_t + \gamma G_{t+1}$
    $L.append(G_t)$
  end for
  $L \leftarrow L.reverse()$
  return L

(b) (3 pts) Consider the cases in which the PPO update results in the gradient equalling zero. Express these cases mathematically and explain why PPO behaves in this manner.
Solution:
The PPO objective function is for one sample tupe $(s, a, r, A, \log \pi_{\text{old}}(a|s))$:

$$\mathcal{L} = \min(r_\theta(s,a)A, \text{clip}(r_\theta(s,a), 1-\epsilon, 1+\epsilon)A)$$

The cases in which the gradient $\nabla_\theta \mathcal{L}$ is equalling zero is when:

$$\mathcal{L}(\theta) = \begin{cases} (1-\epsilon)A & \text{if } r_\theta < 1-\epsilon \text{ and } A < 0 \\ (1+\epsilon)A & \text{if } r_\theta > 1+\epsilon \text{ and } A > 0 \end{cases}$$

We first consider the case $\mathcal{L}(\theta) = (1-\epsilon)A$. When $r_\theta < 1-\epsilon$, this mean the action a_t becomes more unlikely under the current policy $\pi_\theta(a_t|s_t)$ than the old policy $\pi_{\text{old}}(a_t|s_t)$ and this action also result in negativer advatnage, i.e $A_t < 0$. As a consequence, a corresponding training example will not encourage the gradient to point into a direction making the probability of selecting action a_t being associated with a negative advantage.
Consider the case $\mathcal{L}(\theta) = (1+\epsilon)A$. In this case, the probabiltiy of choosing an action a_t associated with positive advantage A_t in state s_t has become considerably larger already under the current policy than it used to be under the old state of the policy. Thus, since the overall objective value is clipped, only the zero-gradient will result from this example.

(c) (3 pts) Notice that the method which samples actions from the policy also returns the log-probability with which the sampled action was taken. Why does REINFORCE not need this information while PPO does? Suppose this log-probability information had not been collected during the rollout. How would that affect the implementation (that is, change the code you would write) of the PPO update?
Solution:

- Policy gradient with REINFORCE algorithm is an on-policy methods, which is use the current policy to explore and update its parameter. We need to calculate the log probability directly in order to backpropagate through the objective function J_θ.
- While PPO with the main objective function is particularly designed to allow for multiple epochs of weight updates on the same set of training data. The log-probability information collected during the rollout is needed to calculate denominator π_{old} of the ratio r_θ. If we not collected the log-probability inforamtion during the rollout, we have to maintain an old policy π_{old} weights to get the old log probability.

If the log-probability information had not been collected during the rollout. The implementation of the PPO update:

```
def update_policy(observations, actions, advantages, old_policy):
    optimizer.zero_grad()
    probs = policy.action_distribution(observations)
    log_probs = probs.log_prob(actions)
    with torch.no_grad():
        old_probs = old_policy.action_distribution(observations)
        old_logprobs = old_probs.log_prob(actions)
    ratio = torch.exp(logprobs-old_logprobs)
    clipped_ratio = torch.clamp(ratio, 1-self.eps_clip, 1+self.eps_clip)
    loss = -torch.mean(torch.min(ratio*advantages, clipped_ratio*advantages))
    loss.backward()
    optimizer.step()
```

(d) (12 pts) The general form for running your policy gradient implementation is as follows:

```
python main.py --env-name ENV --seed SEED --METHOD
```

`ENV` should be `cartpole`, `pendulum`, or `cheetah`, `METHOD` should be either `baseline`, `no-baseline`, or `ppo`, and `SEED` should be a positive integer.

For the `cartpole` and `pendulum` environments, we will consider 3 seeds (`seed = 1, 2, 3`). For `cheetah`, we will only require one seed (`seed = 1`) since it's more computationally expensive, but we strongly encourage you to run multiple seeds if you are able to. Run each of the algorithms we implemented (PPO, PG with baseline, PG without baseline) across each seed and environment. In total, you should end up with at least 21 runs.

Plot the results using:

```
python plot.py --env-name ENV --seeds SEEDS
```

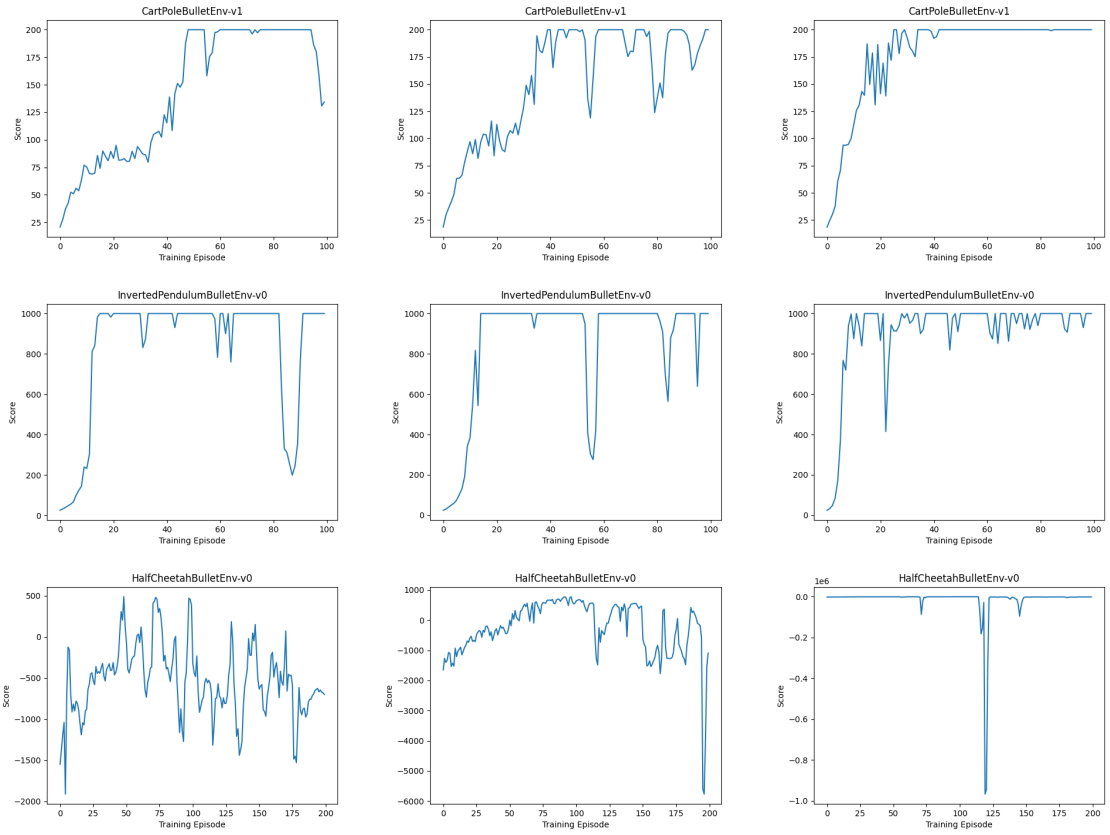
where `SEEDS` should be a comma-separated list of seeds which you want to plot (e.g. `--seeds 1,2,3`). **Please include the plots (one for each environment) in your writeup, and comment on the performance of each method.**

We have the following expectations about performance to receive full credit:

- cartpole: Should reach the max reward of 200 (although it may not stay there)

- pendulum: Should reach the max reward of 1000 (although it may not stay there)
- cheetah: Should reach at least 200 (could be as large as 900)

Solution:



- **CartPole**: Policy gradient methods, with or without baseline have almost identical performance, PPO methods seems to converge faster and stable than Policy gradient methods
- **Pendulum**: the same with CartPole environments
- **Cheetah**: The Cheetah environments is more complex, Policy gradient methods with baseline perform better than without baseline with noisier performance curve, PPO methods seems to failed to converge on this task.

2 Best Arm Identification in Multi-armed Bandit (35pts)

In many experimental settings we are interested in quickly identifying the "best" of a set of potential interventions, such as finding the best of a set of experimental drugs at treating cancer, or the website design that maximizes user subscriptions. Here we may be interested in efficient pure exploration, seeking to quickly identify the best arm for future use.

In this problem, we bound how many samples may be needed to find the best or near-optimal intervention. We frame this as a multi-armed bandit with rewards bounded in $[0, 1]$. Recall a bandit problem can be considered as a finite-horizon MDP with just one state ($|\mathcal{S}| = 1$) and horizon 1: each episode consists of taking a single action and observing a reward. In the bandit setting – unlike in standard RL – the action (or "arm") taken does not affect the distribution of future states. We assume a simple multi-armed bandit,

meaning that $1 < |\mathcal{A}| < \infty$. Since there is only one state, a policy is simply a distribution over actions. There are exactly $|\mathcal{A}|$ different deterministic policies. Your goal is to design a simple algorithm to identify a near-optimal arm with high probability.

We recall Hoeffding's inequality: if $X_1, \ldots, X_n$ are i.i.d. random variables satisfying $0 \leq X_i \leq 1$ with probability 1 for all i, $\overline{X} = \mathbb{E}[X_1] = \cdots = \mathbb{E}[X_n]$ is the expected value of the random variables, and $\widehat{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is the sample mean, then for any $\delta > 0$ we have

$$\Pr\left(|\widehat{X} - \overline{X}| > \sqrt{\frac{\log(2/\delta)}{2n}}\right) < \delta. \tag{1}$$

Assuming that the rewards are bounded in $[0, 1]$, we propose this simple strategy: pull each arm n_e times, and return the action with the highest average payout $\widehat{r}_a$. The purpose of this exercise is to study the number of samples required to output an arm that is at least ϵ-optimal with high probability. Intuitively, as n_e increases the empirical average of the payout $\widehat{r}_a$ converges to its expected value $\overline{r}_a$ for every action a, and so choosing the arm with the highest empirical payout $\widehat{r}_a$ corresponds to approximately choosing the arm with the highest expected payout $\overline{r}_a$.

(a) (15 pts) We start by bounding the probability of the "bad event" in which the empirical mean of some arm differs significantly from its expected return. Starting from Hoeffding's inequality with n_e samples allocated to every action, show that:

$$\Pr\left(\exists a \in \mathcal{A} \quad \text{s.t.} \quad |\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) < |\mathcal{A}|\delta. \tag{2}$$

Note that, depending on your derivation, you may come up with a tighter upper bound than $|\mathcal{A}|\delta$. This is also acceptable (as long as you argue that your bound is tighter), but showing the inequality above is sufficient.

Solution:

$$\begin{aligned}
\Pr\left(\exists a \in \mathcal{A} \quad \text{s.t.} \quad |\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) &= \Pr\left(\bigcup_{a\in\mathcal{A}} |\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) \\
&\leq \sum_{a\in\mathcal{A}} \Pr\left(|\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) \\
&< |\mathcal{A}|\delta.
\end{aligned}$$

Where the first inequality are derived from Boole's inequality.

(b) (20 pts) After pulling each arm (action) n_e times our algorithm returns the arm with the highest empirical mean:

$$a^\dagger = \arg\max_a \widehat{r}_a \tag{3}$$

Notice that $a^\dagger$ is a random variable. Let $a^\star = \arg\max_a \overline{r}_a$ be the true optimal arm. Suppose that we want our algorithm to return at least an ϵ-optimal arm with probability at least $1 - \delta'$, as follows:

$$\Pr\left(\overline{r}_{a^\dagger} \geq \overline{r}_{a^\star} - \epsilon\right) \geq 1 - \delta'. \tag{4}$$

How accurately do we need to estimate each arm in order to pick an arm that is ϵ-optimal? Then derive how many total samples we need total (across all arms) to return an ϵ-optimal arm with prob

at least 1- δ' (that satisfies Equation 4). Express your result as a function of the number of actions, the required precision ϵ and the failure probability δ'.

Solution:

From (a), we have $\forall a \in \mathcal{A}$ subject to:

$$\Pr\left(|\widehat{r}_{a^*} - \overline{r}_{a^*}| < \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) \geq 1 - |\mathcal{A}|\delta.$$

$$\Pr\left(\widehat{r_{a^*}} > \overline{r_{a^*}} - \sqrt{\frac{\log 2/\delta}{2n_e}}\right) > 1 - |\mathcal{A}|\delta$$

$$\Rightarrow Pr\left(\widehat{r_{a^\dagger}} > \overline{r_{a^*}} - \sqrt{\frac{\log 2/\delta}{2n_e}}\right) > 1 - |\mathcal{A}|\delta$$

$$\Rightarrow \Pr\left(\overline{r_{a^\dagger}} > \overline{r_{a^*}} - 2\sqrt{\frac{\log 2/\delta}{2n_e}}\right) > 1 - |\mathcal{A}|\delta$$

From the equation above, we have $\epsilon = 2\sqrt{\frac{\log 2/\delta}{2n_e}}$ with $\delta = |\mathcal{A}|\delta'$. To return an ϵ-optimal arm with prob at least $1 - \delta'$, we need at least a total of samples:

$$n_e = \frac{2\log 2|\mathcal{A}|/\delta'}{\epsilon^2}$$

3 Ethical Concerns with Policy Gradients (5 pts)

In this assignment, we focus on policy gradients, an extremely popular and useful model-free technique for RL. However, policy gradients collect data from the environment with a potentially suboptimal policy during the learning process. While this is acceptable in simulators like MuJoCo or Atari, such exploration in real world settings such as healthcare and education presents challenges.

Consider a case study of a Stanford CS course considering introducing a RL-based chat bot for office hours. For each assignment, some students will be given 100% human CA office hours; others 100% chatbot; others a mix of both. The reward signal is the student grades on each assignment. Since the AI chatbot will learn through experience, at any given point in the quarter, the help given by the chatbot might be better or worse than the help given by a randomly selected human CA.

If each time students are randomly assigned to each condition, some students will be assigned more chatbot hours and others fewer. In addition, some students will be assigned more chatbot hours at the beginning of the term (when the chatbot has had fewer interactions and may have lower effectiveness) and fewer at the end, and vice versa. All students will be graded according to the same standards, regardless of which type of help they have received.

Researchers who experiment on human subjects are morally responsible for ensuring their well being and protecting them from being harmed by the study. A foundational document in research ethics, the Belmont Report, identifies three core principles of responsible research:

(a) **Respect for persons:** individuals are capable of making choices about their own lives on the basis of their personal goals. Research participants should be informed about the study they are considering undergoing, asked for their consent, and not coerced into giving it. Individuals who are less capable of giving informed consent, such as young children, should be protected in other ways.

(b) **Beneficence:** the principle of beneficence describes an obligation to ensure the well-being of subjects. It has been summarized as "do not harm" or "maximize possible benefits and minimize possible harms."

(c) **Justice:** the principle of justice requires treating all people equally and distributing benefits and harms to them equitably.

(a) (4 pts) In 4-6 sentences, describe **two** experimental design or research choices that researchers planning the above experiment ought to make in order to respect these principles. Justify the importance of these choices using one of the three ethical principles above and indicating which principle you have chosen. For example, "Researchers ought to ensure that students advised by the chatbot are able to revise their assignments after submission with the benefit of human advice if needed. If they did not take this precaution, the principle of justice would be violated because the risk of harm from poor advice from the AI chatbot would be distributed unevenly."

At universities, research experiments that involve human subjects are subject by federal law to Institutional Review Board (IRB) approval. The purpose of IRB is to protect human subjects of research: to "assure, both in advance and by periodic review, that appropriate steps are taken to protect the rights and welfare of humans participating as subjects in the research" (reference). The IRB process was established in response to abuses of human subjects in the name of medical research performed during WWII (reference). The IRB is primarily intended to address the responsibilities of the researcher towards the subjects. Familiarize yourself with Stanford's IRB Research Compliance process at this link.

(b) (1 pt) If you were conducting the above experiment, what process would you need to follow at Stanford (who would you email/ where would you upload a research protocol) to get clearance?