Problem 1

In this problem, you will explore strategies for *leveraging demonstrations to learn policies*. This is approach is known under various names such as imitation learning, learning from demonstrations, and behavior cloning [1]. As in the previous homework, you will use Atari games, but rather than learning solely through reinforcement, i.e., only through the experience of your agent interacting with the environment and collecting rewards, you will play the role of the expert and provide demonstrations on how to play (yes, you will have to **play** video games to solve this problem). These demonstrations will then be used to learn a policy that maps states to actions. Having access to demonstrations effectively *turns a policy optimization problem into a supervised learning problem*.

You will implement two approaches to imitation: vanilla supervised learning and data aggregation (DAgger) [4]. In the vanilla supervised learning approach, you simply play the game, collect pairs of the form: state $s \in S$ visited; action $a \in A$ taken by you, the (human) expert, in s. You will use these pairs to train a policy in a supervised fashion (i.e., train a classifier). In the literature, the expert is also sometimes called the oracle or the demonstrator.

The problem with the vanilla supervised learning approach to imitation is that the policy is trained only based on data collected along expert trajectories, and therefore it never learns to recover from its own mistakes as it is not exposed to them during training. DAgger addresses this problem by using the currently learned policy to sample trajectories for the expert to label and uses all data collected so far to train a new policy. Data collection and policy retraining is done repeatedly for some number of iterations. Querying the human expert in an Atari game is effectively asking "which action would you choose if you were in this game state?".

Guidelines on implementation

We have provided starter code to preprocess frames, play a game, and collect training data. You will need to create and train the models in both imitation settings. We made it so that it is easy for the human expert to provide demonstrations — rather than the environment transitioning with the clock regardless of the user having input an action or not, we made it so that the environment only transitions when the user inputs an action. *The environment only transitions on action inputs and it is frozen until the user inputs a action.*

To make sure that the vanilla supervised learning approach and DAgger are comparable, you will collect the same number of training pairs per round for each of the different approaches and models. We recommend you use increments of N training pairs that are comparable to the number of steps needed to complete a game, e.g., for `SpaceInvaders-v0`, N is around 1000 steps. You will place around 5 games for each of the different methods.

Be careful providing the demonstrations: roughly, the higher the quality of the demonstrations, the higher the performance you can expect by learning a policy based on them. This means it is worth to make an effort to play reasonably well. You may choose either Enduro (`Enduro-v0`) or Space Invaders (`SpaceInvaders-v0`) for this problem. For fun, you may also train and run on additional games if you want.

Let $\mathcal{D}_1, \ldots, \mathcal{D}_5$ be the datasets of pairs $(s, \pi^*(s)) \in S \times A$ where $\pi^* : S \to A$ is the

expert policy. Let $\hat{\pi}_{\mathcal{D}_{1:i}}$ represent the policy trained using datasets $\mathcal{D}_1, \ldots, \mathcal{D}_i$. We ask you to, for $i = 1, \ldots, 5$, train $\hat{\pi}_{\mathcal{D}_{1:i}}$, run 100 episodes with it, and report the mean undiscounted cummulative reward $\pm$ standard deviation. This will generate a table with 20 entries, where each of the 4 methods has 5 entries. We also ask you to submit a video of one episode played by policy $\hat{\pi}_{\mathcal{D}_{1:5}}$ for each of the 4 methods.

Each time a new dataset is collected, start retraining from scratch, i.e., from a random initialization rather than warm starting with the previously trained policy. Use mini-batches of size 32 and reshuffle the examples after each epoch. Hold out 20% of the data for validation and compute the validation performance after each epoch. Use the policy that achieved the best validation performance. Measure the validation performance in terms of the log loss of incurred by your policy in the validation expert predictions. Stop the training process if the validation loss fails to improve after some number of epochs (e.g., 10). Briefly describe your exact experimental setup.

For collecting data under policy you will have to sample trajectories under that policy, i.e., at each state you choose an action by sampling from action distribution, rather than taking the argmax. Except perhaps the bonus question, you should be able to solve this problem using only the CPU.

Questions

1. [**10pts**] Implement the vanilla supervised learning approach to imitation learning. Use a linear model trained directly on pixel feature representations. Before implementing the model in Tensorflow or Keras, you can use `LogisticRegression` from scikit-learn to train a linear policy. Note that in the supervised case, you can play all games upfront, i.e., you can play the $5N$ game steps at once. This will not be the case for DAgger.

2. [**10pts**] Implement the supervised learning approach to imitation learning. Use the convolutional neural network architecture of [2], i.e., the CNN architecture used in the previous homework. You can use the same training data collected for this problem.

3. [**10pts**] Implement DAgger. Similarly to the previous part of this problem using a linear model, you can use scikit-learn to quickly get a policy, and then write an implementation in Tensorflow or Keras. Note that contrary to the vanilla supervised approach, DAgger requires you to use your currently trained policy to sample trajectories that will be then labeled by you. You will still train the first policy $\hat{\pi}_{\mathcal{D}_1}$ based on a dataset $\mathcal{D}_1$ collected from expert trajectories (you can use the same $\mathcal{D}_1$ collected in the supervised learning questions).

4. [**10pts**] Similar to the previous question, but now using the convolutional architecture of [2].

5. [**5pts**] Plot as images the weight vectors associated with each of the actions in the linear models $\hat{\pi}_{\mathcal{D}_{1:5}}$. Note that each weight vector spans 4 images; reshape the weight vector into these 4 images and plot them side by side. Do these images exhibit interpretable

structure? Are there significant differences between the images of the vanilla supervised approach and DAgger? Discuss the results.

6. **[5pts]** Plot as images the filters of the first convolutional layer in the convolutional architectures $\hat{\pi}_{\mathcal{D}_{1:5}}$. Note that each filter has 4 channels, corresponding to the 4 frames stacked used the representation of the state, so you will have to display them side by side. Do these filters exhibit interpretable structure? Are there significant differences between the filters of the vanilla supervised approach and DAgger? Discuss the results.

7. **[20pts, bonus]** A common strategy in imitation learning is to use expert demonstrations to train a policy and then improve it using reinforcement learning. This usually leads to good policies faster than doing reinforcement learning from scratch with a random policy initialization. You will use one of the policies that you trained using demonstrations to initialize a reinforcement learning algorithm to learn a better policy.

   One way of accomplishing this while reusing code from the previous homework is to use one of the Q-learning implementations. The policy learned from demonstrations can be used as the behavior policy for interacting with the environment (remember that Q-learning is off-policy).

   We can start with Q-networks randomly initialized and collect (s, a, r, s') tuples using $\hat{\pi}_{\mathcal{D}_{1:5}}$ that are used to fill the replay memory and update the Q-networks. After some number of training updates (e.g., 500000), switch to an implementation of the type used in the previous homework, i.e., using the Q-networks for interaction with the environment. Use the same experimental setup of the previous homework when applicable.

   Compare this with doing reinforcement learning from scratch using a random initialization. Evaluate the policy induced by the Q-network every 10000 or 100000 interactions with the environment. To evaluate a policy, run 20 episodes and compute the mean undiscounted accumulated reward. Also generate a table at the end of the training process with the scores for each of the different models, running each model for 100 episodes and computing the undiscounted reward $\pm$ standard deviation. As in the previous homework, create videos at different points of the training process (0/3, 1/3, 2/3, and 3/3 through the training process).

   The previous approach is a bit indirect because it requires us to go from policies to Q-functions. A natural alternative is to initialize a policy gradient method with the policy that we learned through demonstrations. Rather than the approach described in the previous paragraphs, consider implementing an actor critic policy gradient method of your choice. **Do one of the two.** You should report the same information in your report in both cases. Even for the policy gradient case, code from the previous homework may still be useful, namely, the replay memory and the learning schedules.

References

[1] J Andrew Bagnell. An invitation to imitation. Technical report, DTIC Document, 2015.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[3] Stephane Ross. Interactive learning for sequential decisions and predictions. 2013.

[4] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.