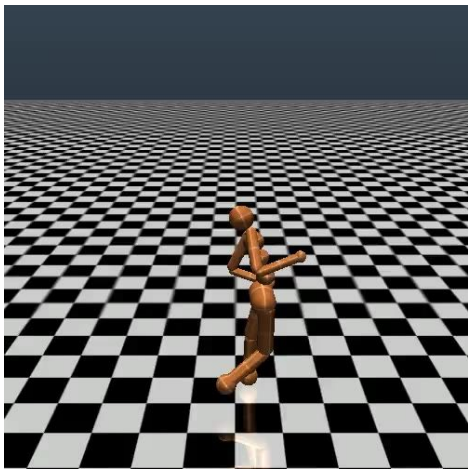
Distillation vs Multi-Task Learning

Sergey Levine (svlevine@eecs.berkeley.edu)

Policy distillation is a technique where policies from multiple tasks (e.g. multiple Atari games) are combined into a single policy via supervised learning. Some recent work (actor-mimic, policy distillation) has shown that distillation works better in some cases than training a single policy simultaneously on multiple tasks. An interesting question is why, and when. This project would investigate in which situations multi-task learning beats distillation, and vice-versa, and attempt to formulate either a set of conditions to determine whether distillation or multi-task learning should be used, or improve one or the other technique to be better.

Learning to Walk in a Day

Sergey Levine (svlevine@eecs.berkeley.edu)



A recent paper called “learning to play in a day” proposes an interesting modification to Q-learning to accelerate learning for Atari games with discrete state spaces. An interesting extension to this method would be to figure out how to apply it to continuous control tasks (such as the OpenAI gym tasks), for actor-critic (DDPG) or Q-learning (NAF) formulations.

OpenAI gym: https://gym.openai.com/

Learning to play in a day: https://openreview.net/pdf?id=rJ8Je4clg

DDPG: https://arxiv.org/pdf/1509.02971v5.pdf

NAF: https://arxiv.org/pdf/1603.00748.pdf

Inverse RL for Language Generation

Chelsea Finn (cbfinn@eecs.berkeley.edu)

Generating language is key to many tasks in AI such as translation, speech recognition, and dialog. While reinforcement learning has been used for generative models of language, the natural language community lacks a good objective for quantifying the quality of a piece of text. Inverse RL has a natural application to language generation tasks, as it can learn an objective for good language. The recently proposed guided cost learning algorithm scales maximum entropy IRL to neural network cost functions and can be shown to be equivalent to certain form of generative adversarial training, where the generator/policy is trained with reinforcement learning. While generative adversarial networks (GANs) have had limited success when applied

to natural language, the proposed guided cost learning formulation has the potential to be much more stable by using the density of the generator, which is easy to obtain for language tasks.

Empirical Comparison of Models for Model Learning
Chelsea Finn (cbfinn@eecs.berkeley.edu)
In the past, Gaussian processes (GPs) and other nonparametric models have been popular choices for representing dynamics models in model-based RL. However, such models are generally not scalable to large amounts of data and high-dimensional observations. Furthermore, there has been limited success with using neural networks for learning dynamics models for model-based RL. This project would involve an empirical investigation of various types of models (e.g. GPs, nearest neighbor, neural networks, Bayesian neural networks) and their effectiveness at predicting the future. Recently, there have been several neural network architectures that draw similarities to nonparametric models and would be worth including in such an investigation.

Botstrapping
John Schulman (joschu@openai.com)
How can we devise a learning algorithm that allows a pupil to surpass its master? Find a video game with pre-programmed bots that give decent performance. Devise an algorithm that combines imitation learning with reinforcement learning to exceed the performance of the bots. One baseline approach is to train on a bot's actions using DAGGER, and then fine-tune with reinforcement learning. Is there a good way to learn from multiple different bots?

Investigate Regularization in Reinforcement Learning
John Schulman
In supervised learning, regularization is extremely important for preventing overfitting. In deep learning, the most successful methods are dropout, batch normalization, and L2 regularization. However, regularization has not been shown to provide a benefit when used with reinforcement learning algorithms such as policy gradients and Q-learning. Experimentally investigate the effect of different regularization methods on your chosen RL algorithm. (I recommend using policy gradients, because they are generally better understood than Q-learning.) Try to provide some explanations for your results.

Deep Kernel Based Reinforcement Learning
John Schulman
Kernel Based Reinforcement Learning (KBRL) is an algorithm with nice theoretical properties, and it admits an efficient approximation called Kernel-Based Stochastic Factorization (KBSF). A similar idea was recently proposed, under the name Model-Free Episodic Control. Modify KBRL or KBSF to use learned kernel functions. For example, $K(x,y) = f(x) \cdot f(y)$ or $K(x) = \exp(-||f(x) - f(y)||^2)$, where f is represented by a neural network. See [1,2] and citations therein for an example of using learned kernels represented by neural networks.

Policy Gradients with Optimistic Value Functions

John Schulman
Policy gradient methods use value functions for variance reduction (e.g., see A3C or GAE). To obtain unbiased gradient estimates, the value function is chosen to approximate V^{\pi}, the value function of the current policy. There is reason to believe that we would obtain faster learning on many problems by instead using a value function that approximates V^*, the optimal value function. You can fit V^* by using Q-learning (to fit Q^*) or simply by fitting V to satisfy the inequality *V(s) <= empirical return after state s* rather than the equality *V(s) = empirical return after state*.

Reinforcement Learning Using Difference-of-Value Functions
John Schulman
One problem with many reinforcement learning methods is that they rely on accurately approximating a value function V(s) or Q(s,a), however, these functions have a large dynamic range, making them hard to approximate. Furthermore, we only actually care about the different in value between nearby states or actions--not the absolute magnitude. Bertsekas proposed to learn a value function D(s1, s2) that estimates the difference in returns between states. Implement this idea using modern deep RL ideas.

Solving Integer Programs Using Deep Reinforcement Learning
John Schulman
Certain optimization problems, such as integer linear programs and integer quadratic programs (ILP, IQP) are NP-complete; however, in practice, there are heuristics that allow for a large-fraction of real-world problems to be solved quickly. These problems are well-studied (including large benchmark datasets), and there is commercial software such as Gurobi devoted to solving them. Your task is to use deep RL to make some of the decisions that are otherwise made in a heuristic way in ILP/IQP solvers, and try to compete with these solvers on benchmark datasets. Here's a related paper by Bello et al., which solves traveling salesman problem end-to-end with an RNN. For this project, you may want to take an approach that preserves the basic structure of ILP/IQP solvers (unlike Bello et al.) but uses neural networks to replace the heuristics.

PGRD+MCTS with Learned Models
John Schulman
Policy Gradient for Reward Design is the idea of implementing a search algorithm, but learning the dynamics model and/or reward using a model-free method. Guo et al. recently proposed an elegant version of this idea, which learns a reward function to be used in Monte Carlo Tree Search. Implement this idea on continuous control problems using a learned dynamics model.

More Optimal Auxiliary Objectives
John Schulman
Several recent papers (1, 2, 3, 4) use auxiliary objectives to speed up reinforcement learning: the idea is to add a prediction objective that encourages the neural network to learn good features of the observation stream, and these features speed up the reinforcement learning.

(This is also closely related to the idea of semi-supervised learning.) In these recent papers, the choice of auxiliary objectives and their relative weighting is completely ad-hoc. For this project, develop a strategy for optimizing over the auxiliary objective, automating the process that would otherwise be carried out by the researcher. One approach is to add an outer loop, where hyperparameter optimization / evolution tunes the auxiliary objective. Another approach is to use gradients to tune the auxiliary objective using gradient-based hyperparameter optimization.

Q(λ) Combined With Double Q-Learning
John Schulman
This paper describes a way to do Q-learning with a TD(λ) backup that nevertheless converges to Q* (in contrast to other TD(λ) methods that converge to Q^{π}.) It performs poorly due to overestimation of Q-values (see here for experiments); however, that problem is partly solved by using double Q-learning [1, 2].

Transfer Learning - OpenAI Gym Retro Contest
Sid Reddy (sgr@berkeley.edu)
Many existing deep RL benchmarks (e.g., Arcade Learning Environment and MuJoCo) tend to encourage training and testing agents in the same environment, which can lead to overfitting. Gym Retro provides a suite of classic video games that makes it easy to test an RL algorithm's ability to generalize to unseen environments. The winners of the initial transfer learning contest were only able to reach less than half of the theoretical maximum score, and relied on fine-tuned implementations of existing algorithms like PPO and Rainbow. Can you improve upon their performance, e.g., through meta-learning?

Adaptive Discretization of Continuous States
Sid Reddy (sgr@berkeley.edu)
One of the motivations behind using function approximation to deal with continuous states in RL is that naive attempts to discretize a continuous state space will result in a discrete state space whose size grows exponentially with the dimensionality of the continuous observations. Can you come up with an algorithm that partitions a continuous state space into n discrete states such that a policy trained via dynamic programming performs well? Can you extend your approach to MDPs with continuous actions?

Dynamics Curriculum Learning, or Training Wheels for Deep RL Agents
Sid Reddy (sgr@berkeley.edu)
Existing curriculum learning algorithms for RL provide an agent with a sequence of tasks characterized by different initial state distributions or goals. Another approach could be to present the agent with a sequence of environments characterized by different state transition dynamics, ranging from "easy" environments in which the dynamics are simplified or tuned to make it easier to reach the goal, to "hard" environments that are closer to the real world.

Inverse RL from Non-Expert Demonstrations
Sid Reddy (sgr@berkeley.edu)

One of the central assumptions in most existing inverse RL or Bayesian inverse planning frameworks is that the demonstrator is optimal, but this isn't always the case, especially when the demonstrator is a human and the task is challenging for them (e.g., teleoperating a non-humanoid robot with many degrees of freedom). While models of suboptimal behavior exist, they typically assume that suboptimal actions are the result of some type of random noise (e.g., Boltzmann-rationality in maximum entropy IRL) or a known cognitive bias, like temporal inconsistency, false beliefs, or internal model misspecification. Can you come up with an algorithm that learns a reward function from suboptimal demonstrations, without making assumptions about human irrationality? One approach could be to combine IRL with learning from preferences.

Application: Automated Theorem Proving
Sid Reddy (sgr@berkeley.edu)
I don't know much about this area, but it seems cool. For example, it would be awesome if you could improve the auto tactic in Coq using an AlphaZero-style search algorithm. See https://arxiv.org/pdf/1701.06972.pdf for ideas on how to get started.

Application: Learning to Teach Humans
Sid Reddy (sgr@berkeley.edu)
Guiding a student through a sequence of lessons and helping them retain knowledge is one of the central challenges in education. Online learning platforms like Khan Academy and Duolingo tackle this problem in part by using interaction data to estimate student proficiency and recommend content. While the literature proposes a variety of algorithms for modeling student learning, there is relatively little work on principled methods for sequentially choosing items for the student to review in order to maximize learning. See https://people.eecs.berkeley.edu/~reddy/files/DRL_Tutor_NIPS17_MT_Workshop.pdf and https://web.stanford.edu/~cpiech/bio/papers/Uncovering%20Patterns%20in%20Student%20Work.pdf for ideas on how to get started.

Which model should I deploy?
Gregory Kahn (gkahn@berkeley.edu)
Learning models (e.g., dynamics models) is a promising approach for reinforcement learning. One of the main advantages of model-based methods is that the model can be used to achieve a variety of tasks at test time. However, it is not clear which model you train will end up performing best on the test tasks; for example, evaluating the model's prediction error on a holdout set (as is done in supervised learning) is not always a good indicator of performance due to the sequential decision making of reinforcement learning. You can run each model on the test time tasks to determine how good the model is, but this approach is expensive (e.g., if your agent is a real-world robot). Given a set of trained models (and associated controllers) and test time tasks, can you predict which model will perform the best on the test task without attempting the test time task (or using very few samples)?