Crowd-sourcing Advice from Multiple Experts in Reinforcement Learning

Sunny Kant
Summer Intern
StARLinG Lab
University of Texas at Dallas
Supervisor: Prof. Sriraam Natarajan

Contents

1 Problem Motivation **1**

2 Problem Statement **1**

3 Proposed Model **1**

4 Algorithms **2**

5 Domain & Experiment Setup **5**

6 Plots and Discussion **6**

7 Final Result **7**

8 Future Work **8**

9 Links: **8**

10 References **8**

1 Problem Motivation

The final goal of any reinforcement learning or planning algorithm is to explore the environment and learn the required objective within a limited number of trails. However, with complex and dynamic environments, simple reinforcement learning fails to inherently learn all the desired objectives. Crowdsourcing can be very useful in such cases. In general, it refers to a large group of people collectively solving a problem or completing a task. Crowdsourcing has been widely used across medical disciplines. Using Crowdsourcing in order to incorporate advices from experts can make the model learn the parameters quickly.Moreover, these advice rules can help the agent learn behaviour which may otherwise be impossible to learn due to the partially observable states.

Prior work to incorporate advice has been in the form of state and action preferences, Kunapali et al.[1], preferential labels in supervised learning, Odom and Natarajan [2], or by action preferences over a set of states, Maclin et al.[3].

In this work we consider the presence of multiple experts. Let us consider an example, suppose a self driving car considers advice by multiple drivers, some drivers may stop just before a "STOP" sign, others may gradually decelerate from a long way before, both of which may be beneficial to the algorithm at different instances. Or, consider a scenario in heavy traffic, different experts may have a different way of navigating through the traffic quickly. In single advice case, we always consider that we have an optimal expert. This may not always be true. Such cases require advice from multiple experts instead of a single one.

Hence, we need a method which can incorporate advice from the multiple experts and figure out the best advice. The algorithm should also be able to combine advice rules from the experts and the policy from the base algorithm to generate the best policy.

2 Problem Statement

The aim is to develop a model to incorporate advice from multiple experts in reinforcement learning and to predict the best policy by Approx. Q-learning faster than models with no-advice. The algorithm will learn key parameters required to determine action in each state, modeled as MDP for the problem of Pacman game.

3 Proposed Model

The advice from experts are taken at crucial states while Q-learning and in the case of pacman game, crucial states are states when ghosts are nearby to agent. The rest of the time algorithm learns by self Q-learning through epsilon-greedy or Boltzmann exploration with probability p and with 1-p probability following the learned policy.

We consider policy shaping as the approach to incorporate the advice taken from experts. The advice rules are first converted into policy distributions or policy factors, which can then be combined with the policy distribution from the reinforcement learning algorithm. This approach is similar to the bayesian approach followed by Griffith(2013)[4]. The proposed model is described below.

It is assumed that we have k expert preferences which are used to generate k expert policy distributions π_{ak}. The advice is taken in the form of conditional statements which are only valid for certain states. Hence advice is only incorporated in the algorithm only on the states where it is applicable.

These multiple advice policies are weighted and then combined together to give the complete expert policy π_a . This expert policy is then combined with the policy given by the reinforcement learning algorithm, in this case we consider a Q-Learning algorithm, π_q to give the final policy π_f. The action taken at every step is according to this policy π_f. The weights of the advice comprises a neural network consisting one hidden layer and one softmax layer and it can be updated by back propagation algorithm.

In the following algorithm,
π_q denotes the policy from the Q-learning / RL algorithm
π_a denotes the advice policy
π_f denotes the final combined policy

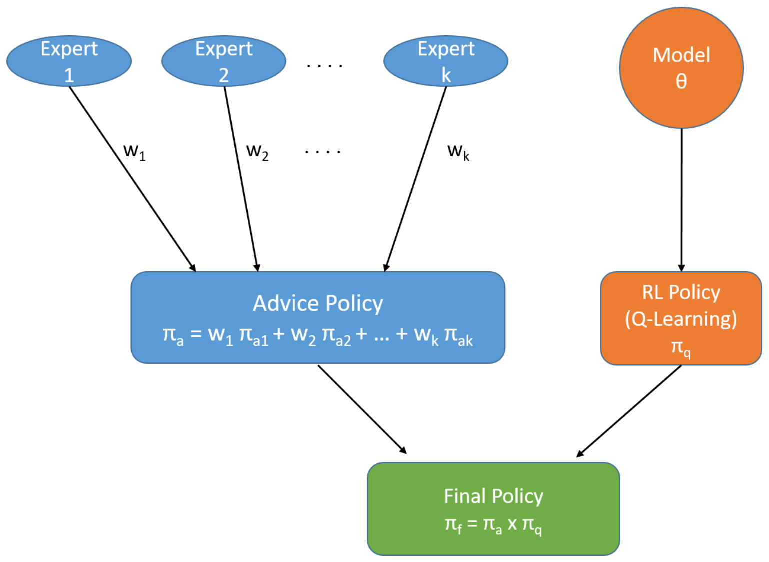


Figure 1: Policy Shaping by Incorporating Advice

θ is the model/weights of the RL algorithm
w_k denotes the weight defined to the policy of an expert k
α is the learning rate of the RL algorithm
γ is the discount factor

4 Algorithms

Algorithm 1 :Approximate Q-learning without advice

```
Initialize $weight\_vector$
for each iteration $i$ in number of training games do
    while (Not win or loss) do
        $s \leftarrow$ current state
        $A \leftarrow$ set of possible actions for state $s$
        if $X \sim Uniform(0,1) < \epsilon$ then
            action = choose any of a(actions) randomly
        else
            for each action $a \in A$ do
                $feature\_vector \leftarrow$ get feature(state,action)
                use $weight\_vector$ to evaluate Q(s,a)
                action = Choose a having higher Q-value
            end for
        end if
        perform action and get reward
        {Now we update the model $weight\_vector$ }
        $\nabla = \alpha * ((R(s,a,s') + \gamma \max_{a'} Q_{\pi_f}(s',a')) - Q(s,a))$
        for ith feature $infeature\_vector$ do
            $Weight\_vector^i = Weight\_vector^i + \nabla *$Value of that feature in (s,a)
        end for
    end while
end for
```

Algorithm 2 :Approximate Q-learning with single advice

```
Initialize $w\_vector$
$lw\_vector \leftarrow$ load weight learned vector(as advice)
for each iteration $i$ in number of training games do
    while (Not win or loss) do
        $s \leftarrow$ current state
        $A \leftarrow$ set of possible actions for state $s$
        $Advice\_cond \leftarrow$ number of ghosts nearby
        if $Advice\_cond == 0$ then
            if $X \sim Uniform(0,1) < \epsilon$ then
                action = choose any of a(actions) randomly
            else
                self learning by following learned policy as in Algorithms1
            end if
        else
            for each action $a \in A$ do
                $feature\_vector \leftarrow$ get feature(state,action)
                use $lw\_vector$ to evaluate Q(s,a)
                action = Choose a having higher Q-value
            end for
        end if
        perform action and get reward
        {Now we update the model $weight\_vector$ }
        $\nabla = \alpha * ((R(s,a,s') + \gamma \max_{a'} Q_{\pi_f}(s',a')) - Q(s,a))$
        for ith feature $infeature\_vector$ do
            $Weight\_vector^i = Weight\_vector^i + \nabla *$Value of that feature in (s,a)
        end for
    end while
end for
```

Algorithm 3 :Approximate Q-learning with multiple advice

```
Initialize $w\_vector$ (learning to predict Q-value)
Initialize $expert\_vector$ (learning weight associated with each expert)
$lw\_vector[i] \leftarrow$ load weight learned vector (advice from ith expert)
for each iteration $i$ in number of training games do
    while (Not win or loss) do
        $s \leftarrow$ current state
        $A \leftarrow$ set of possible actions for state $s$
        $Advice\_cond \leftarrow$ number of ghosts nearby
        if $Advice\_cond == 0$ then
            if $X \sim Uniform(0,1) < \epsilon$ then
                action = choose any of a(actions) randomly
            else
                self learning by following learned policy as in Algorithms1
            end if
        else
            for each expert i from experts do
                use $lw\_vector[i](feature)$ to evaluate Q(s,a)[i]
                for each action $a \in A$ do
                    $feature\_vector \leftarrow$ get feature(state,action)
                    $Q(s,a)[i] = lw\_vector * feature\_vector$
                end for
                $Q(s,a)[i] \leftarrow$ softmax(Q(s,a)[i]
                $bestaction[i] \leftarrow$ action having higher Q-value for each expert i
            end for
            $action =$ bestaction having higher Q-value for actions in a
            (update expert_vector)
            $expert\_vector[i] = expert\_vector[i] - \alpha * (Q - Q[i])$
        end if
        perform action and get reward
        {Now we update the model $weight\_vector$ }
        $\nabla = \alpha * ((R(s,a,s') + \gamma \max_{a'} Q_{\pi_f}(s',a')) - Q(s,a))$
        for ith feature $infeature\_vector$ do
            $Weight\_vector^i = Weight\_vector^i + \nabla *$Value of that feature in (s,a)
        end for
    end while
end for
```

5 Domain & Experiment Setup

The domain selected for this problem is the Pacman Project by UC Berkeley. The experiments were conducted on the 'Medium Classic' layout as described below. As number of states are much larger than medium Grid, so Approx. Q-Learning is used as the base reinforcement learning algorithm.

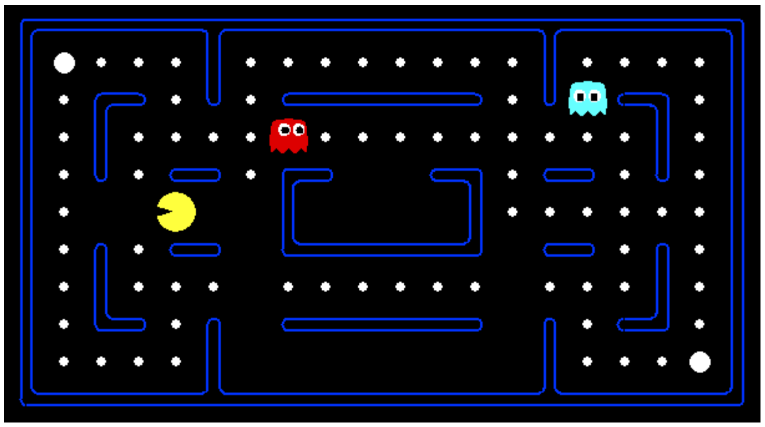


Figure 2: Pacman Medium Classic Layout

The main advantage of Pacman that makes it attractive for reinforcement learning methods is that both the state space and the action space are discrete and defining the game as an MDP is a straightforward process:

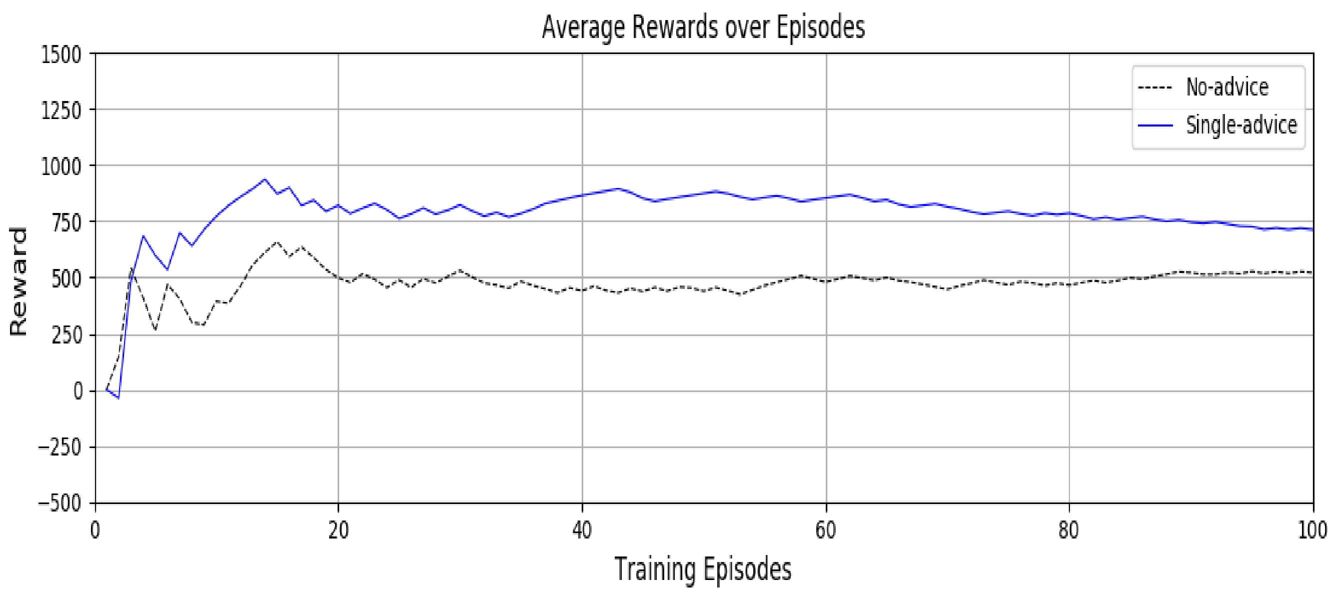
- A state $s \in S$ consists of the map layout and the current game score as well as the positions of Pacman, all ghosts and all remaining pac-dots.
- There are a maximum of five actions $a \in A$ available in each state: moving one step in one of the four cardinal directions as long as the path is not obstructed by a wall or standing still.
- The transition model $T(s, a, s')$ depends on the behavior of the ghosts. While there is no noise when applying an action, i.e. if the action a is going eastwards Pacman's position in the next state s' will always be one step further to the east than in s, the ghosts behave in a stochastic way.
- The reward function $R(s, a, s')$ is defined as the difference in game score that occurs when performing the action a in s that leads to s'. If Pacman eats a pac-dot he gains 10 points, if he wins the game by eating the final pac-dot on the map he gains 500 points and if he loses by colliding with a ghost he is punished by receiving -500 points. Additionally, in each time step the score is reduced by one to encourage fast play. In the implementation that was used eating a power pellet is not rewarded with any points but eating a ghost afterwards leads to 200 extra points.

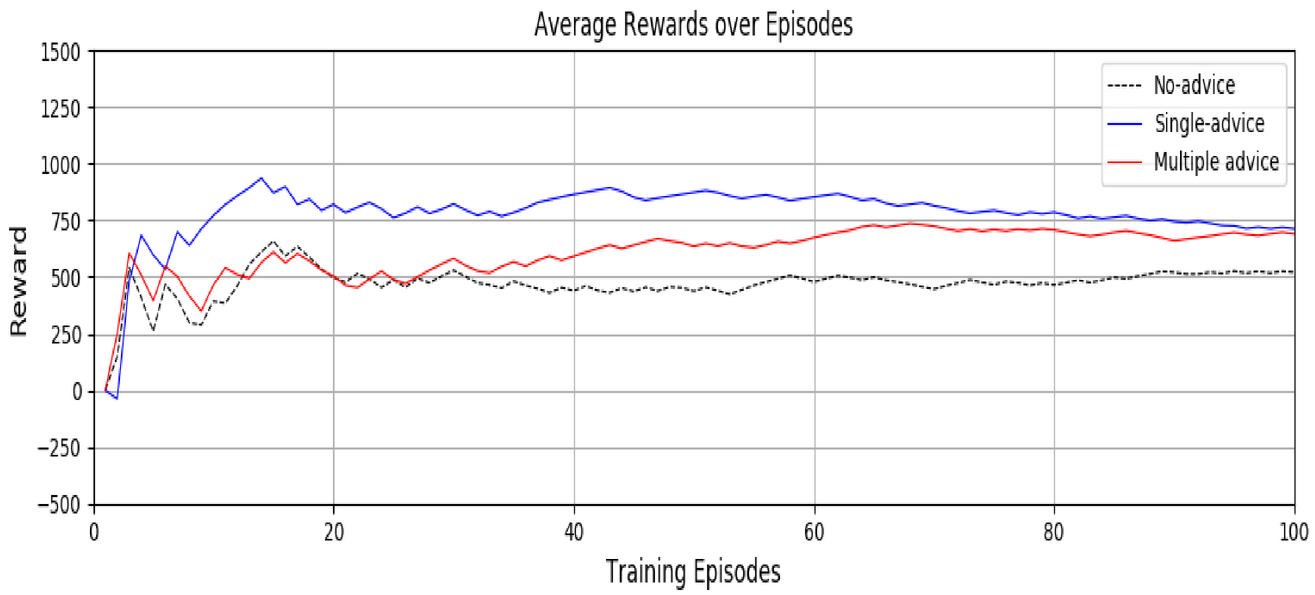
The algorithm was run for 200 episodes with an exploration probability of 0.10 to take random action. All the advice is given to the algorithm before the training starts. The advice considered for this experiment is to take proper action when ghost is nearby. Results are compared for fopllowing three cases -

- **No Advice :** Learning works only according to the baseline Approximate Q-Learning algorithm.
- **Single Expert :** A single advice is given to the learning algorithm. The advice is given as learnt policy from No_advice, can be incorporated real expert advices as matrix of probability of states and actions.
- **Multiple Experts :** Multiple advises are considered from different experts as learned weights for now.

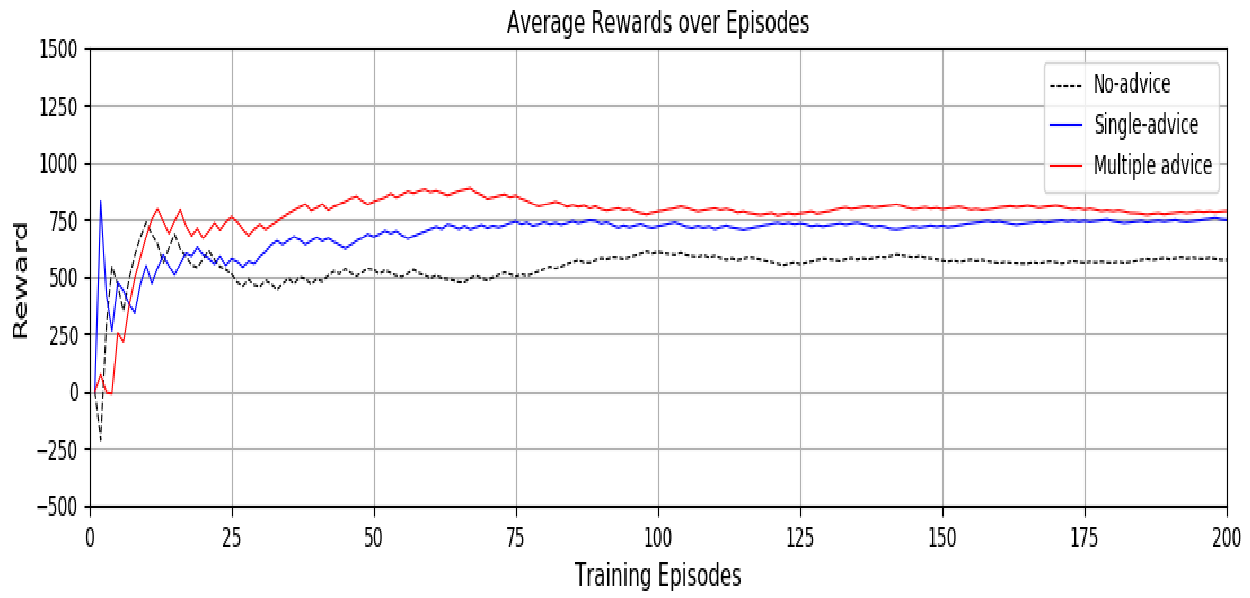
6 Plots and Discussion

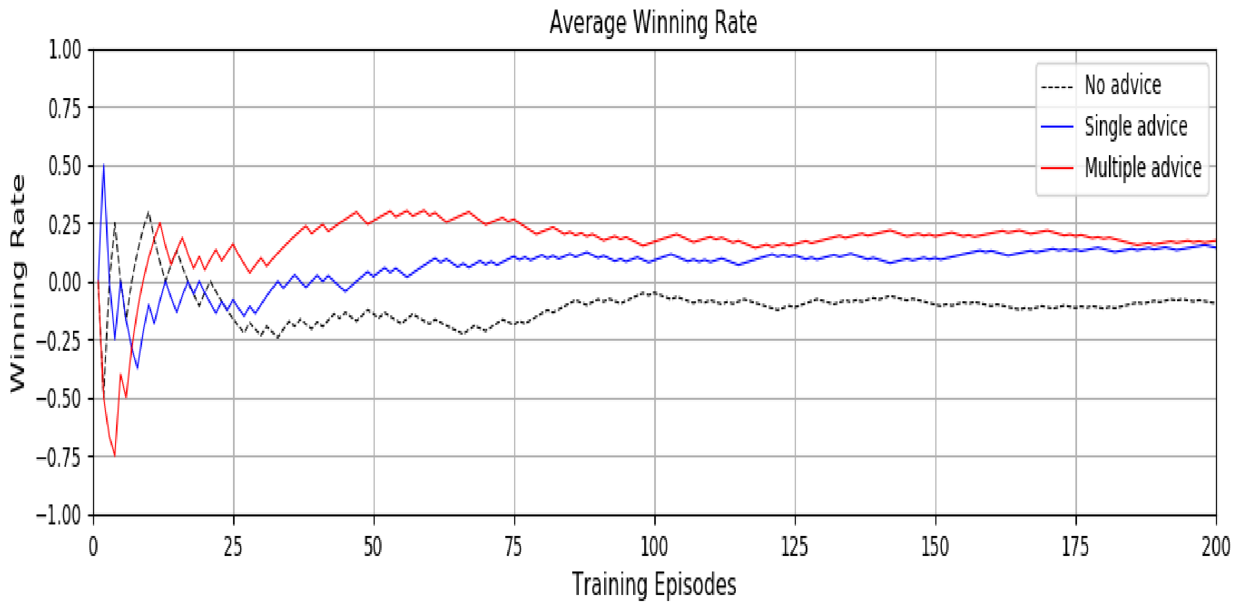
Initially taking 100 episodes of training games. Following plots are obtained.





Error in Multiple Adice Model: Advices from different experts in form of learned parameters are used directly causing different ranges of Q-values for each experts. Learned Weights(Advices) should be first normalized to give same range of Q-values. Or, We should learn parameters in such a way that parameters of different experts are in the same smaller range. I have apllied softmax to normalize advices taken from multiple experts before feeding the advice into actual learning algorithm and got the following plots.





7 Final Result

After appling Softmax function to normalize the advices given from different experts, we get the above plots. Algorithm without advices also converge to the optimal policy but it takes a lot of training episodes to converge, which may not be possible in complex situations. The ideal advice would be something that the algorithm is unable to learn on its own, or would take a long time to do so. In such a case the advice driven algorithms would converge to the optimal reward values significantly faster. Finally, I have used the approximate Q-learning to address the problem in larger grid where simple Q-learning takes around 40,000 episodes to train (as shown by my colleague Rohit), Approx. Q-learning takes around 80 episodes to learn the best policy. As the algorithm without advice is doing so well, so taking advices from experts and incorporating in this algorithm would have shown not much difference but the above plots shows significant improvement in training.

8 Future Work

Real advices can be taken from experts as Transition Probability matrix over states and function. Single advice based algorithm can be more fruitful when boltzmannn exploration is applied instead of epsilon greedy. Multiple advices can be incorporated in a more effective way by using EM algorithm and more robust Neural Nets. The future work would be to build a better framework for incorporating more general advice from different experts. Also, multiple advice rules can be combined by other methods such as Noisy Max to compare with the current algorithm.

9 Links:

The code written for this project can be found here:

1. `https://github.com/sunnykant/Reinforcement-Learning-with-Advice/tree/master/no-advice`
2. `https://github.com/sunnykant/Reinforcement-Learning-with-Advice/tree/master/single-advice`
3. `https://github.com/sunnykant/Reinforcement-Learning-with-Advice/tree/master/multi_advice`

The rewards, wining rate and other data obtained while training can be seen from:

- `https://github.com/sunnykant/Reinforcement-Learning-with-Advice.git`

10 References

[1]Kunapuli, G., Odom, P., Shavlik, J., & Natarajan, S., Guiding Autonomous Agents to Better Behaviors through Human Advice, *IEEE International Conference on Data Mining (ICDM)*, 2013.

[2]Odom, P., Natarajan, S., Actively Interacting with Experts: A Probabilistic Logic Approach,*European Conference on Machine Learning and Principles of Knowledge Discovery in Databases*, 2016.

[3]R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving Advice about Preferred Actions to Reinforcement Learners Via Knowledge-Based Kernel Regression.*Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.

[4] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in Neural Information Processing Systems*, 2013.