RENEG AND BACKSEAT DRIVER: LEARNING FROM DEMONSTRATION WITH CONTINUOUS HUMAN FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) is a powerful framework for solving problems by exploring and learning from mistakes. However, in the context of autonomous vehicle (AV) control, requiring an agent to make mistakes, or even allowing mistakes, can be quite dangerous and costly in the real world. For this reason, AV RL is generally only viable in simulation. Because these simulations have imperfect representations, particularly with respect to graphics, physics, and human interaction, we find motivation for a framework similar to RL, suitable to the real world. To this end, we formulate a learning framework that learns from restricted exploration by having a human demonstrator do the exploration. Existing work on learning from demonstration typically either assumes the collected data is performed by an optimal expert, or requires potentially dangerous exploration to find the optimal policy. We propose an alternative framework that learns continuous control from only safe behavior. One of our key insights is that the problem becomes tractable if the feedback score that rates the demonstration applies to the atomic action, as opposed to the entire sequence of actions. We use human experts to collect driving data as well as to label the driving data through a framework we call "Backseat Driver", giving us state-action pairs matched with scalar values representing the score for the action. We call the more general learning framework ReNeg, since it learns a regression from states to actions given negative as well as positive examples. We empirically validate several models in the ReNeg framework, testing on lane-following with limited data. We find that the best solution in this context outperforms behavioral cloning has strong connections to stochastic policy gradient approaches.

1 PROBLEM FORMULATION

We seek a way to learn a deterministic continuous control policy from demonstration, including both good and bad actions and scores representing their optimality. Our problem statement is somewhere in between behavioral cloning and RL: we focus on a general approach that is capable of mapping continuous sensor input to an arbitrary (differentiable) continuous policy output and capable of using feedback for singular predetermined actions. We refer to this problem setting as the ReNeg framework. We want to use this learning algorithm to teach an autonomous vehicle to follow the lanes on a road. For the demonstration, a human demonstrator drives, yielding state-action pairs the agent can learn from. In order to teach the agent which actions are good or bad, an expert critic, or a "backseat driver," labels the actions as good or bad.

For a viable solution to this problem, we need to define a loss function that induces an effective and robust policy. We also need to choose what driving data to collect to expose the agent to a variety of good and bad states, and in particular show the agent how to get out of bad states where a crash is imminent. What data to collect is non-obvious: we want to explore a good range of good and bad states so the agent learns a reasonable policy for how to act in all kinds of states. However, to make this feasible in the real world, we want to avoid exploring dangerous states. Finally, we need to carefully choose a way to collect human feedback that contains signal from which the agent can learn, but is not too hard to collect. We will discuss the choices we made for these three parts of the problem in the Our Approach section.

2 RELATED WORK

Learning from demonstration has mostly been studied in the supervised learning framework and the Markov Decision Process (MDP) framework. In the former, it is generally known as imitation learning (or behavioral cloning in the AV setting), and in the latter, it is generally known as apprenticeship learning. The supervised learning in this area generally amounts to a least squares regression that maps from an input state to action, with some research addressing the fact that our runtime distribution will differ from our training distribution. (We will discuss this later.) Supervised learning also requires explicit labels for each visited state, which are difficult to collect.

The MDP research on apprenticeship learning has largely been on inverse reinforcement learning (IRL) (Abbeel & Ng, 2004), in which a reward function is estimated given the demonstration of an expert. However, often in this framework, the reward function is restricted to the class of linear combinations of the discrete features of the state, and the framework only allows for positive examples. In addition, there has been work on inverse reinforcement learning from failure (Shiarlis et al., 2016), which allows for a sequence of positive or a sequence of negative examples. There is also distance minimization for reward learning from scored trajectories (Burchfiel et al., 2016), which allows for gradation in the scores, but does not allow for an arbitrary reward function on continuous inputs or labels for atomic actions as opposed to a trajectory or sequence of actions. Moreover, these IRL methods are not a candidate for our problem, since they require an exact MDP solution with a tractable transition function and exploration to find the optimal policy. The issue we have is not that we don't have the reward function, but that even with the more informative feedback, we cannot use exploration to learn the optimal policy.

We want to combine controlled off-policy exploration with expert human feedback to learn a continuous control output. Several areas of research have explored various intersections of these ideas. First, on-policy imitation learning with human expert actions (a la DAgger (Ross et al., 2010), AggreVaTeD (Ross & Bagnell, 2014), Deeply AggreVaTeD (Sun et al., 2017)) require the agent to explore and make mistakes to make use of expert feedback in the form of action labels for the visited states. Second, off-policy RL (a la Off-Policy Actor-Critic(Degris et al., 2012), Q-Learning(Watkins & Dayan, 1992), Retrace(Munos et al., 2016)) learns from actions drawn from a distribution other than the one being learned, so does not require the agent to explore as in on-policy learning. We will discuss the importance sampling method that many of these approaches use, but off-policy RL still assumes the behavior policy used to explore has a non-zero probability of choosing each possible action from each state to ensure exploration. Thus off-policy RL does not allow for safely restricting exploration as we want. Third, Normalized Actor Critic (Gao et al., 2018) works with bad as well as good demonstration. However, Normalized Actor Critic does not allow for restricted exploration either, since it adds entropy to the objective function to encourage exploration. Normalized Actor Critic has also only been done for discrete action control, not continuous as we want to do. Finally, COACH (MacGlashan et al., 2017) is an on-policy RL algorithm that uses human-feedback to label the agent's actions while exploring. COACH is very similar to our algorithm, as we will discuss later. However, COACH was designed for on-policy exploration and uses discrete feedback values of 1 and -1, whereas we use continuous values in $[-1, 1]$. We find in our experiments that this additional granularity in feedback significantly helps training.

In the AV context, the most notable work on learning from demonstration has either been on behavioral cloning (Bojarski et al., 2016) (Pan et al., 2017) or using IRL to solve sub-tasks such as driving preferences that act on top of a safely functioning trajectory planner (Kuderer et al., 2015). To the best of our knowledge, no research so far has focused on using any kind of human feedback in the context of AV control with learning from demonstration. We believe that this is a major oversight: many AV research groups are investing huge amounts of time into collecting driving data; if they used our model, they could improve performance simply by having an expert labeler sit in the car with the driver for no additional real time.

3 OUR APPROACH

The task we evaluated our models on is is lane following in a Unity car simulator. The correctness of an example will be labeled using feedback from a human. Our goal is to produce a policy network (here on abbreviated as PNet) to map states to actions. In our case, the states are RGB images

from the driver's point of view, and the PNet will have to output a steering angle that keeps the car as close to the center of the road as possible. We will discuss in detail finding a useful loss function, what driving data we chose to demonstrate to the agent, and how to collect human feedback that encourages correct behavior in the context of lane following. And finally, we will discuss the architecture implementation.

First, a note: Our problem setting is "off-policy", meaning that the policy we are learning is not the one being used to collect data. Thus, the states we encounter are drawn not from the distribution induced by our agent's policy, but in this case, a policy chosen by the driver. This typically leads to an issue: once the trained agent is acting on its own after training and encounters states it has not seen, it does not know how to act, and strays further from the intended behavior. Thus the assumption that our training data is independent and identically distributed (i.i.d.) from the agent's encountered distribution goes out the window. Various off-policy algorithms have different strategies to mitigate this. We choose to mitigate this issue by having the expert collect data in "bad" states as well as good ones. ReNeg allows us to intentionally expose the distribution to states a suboptimal driver would encounter. Thus, when the agent is eventually acting on its own, if it reaches dangerous states, it will have experience on which to draw.

In the off-policy policy gradient RL framework, this issue is typically circumvented by changing the objective function from an expectation of the learned policy's value function over the learned policy state-visitation distribution to an expectation of the learned policy's value function over the behavior (exploratory) state-visitation distribution (Degris et al., 2012). In the RL framework, this could be dealt with by an approximation off-policy stochastic policy gradient that scales the stochastic policy gradient by an importance sampling ratio (Silver et al., 2014) (Degris et al., 2012). If we make a few assumptions, we may be able to use this gradient, but there are a few reasons why we chose not to. First, importance sampling consistently reduced performance for learning from demonstration in the NAC paper (Gao et al., 2018). Second, we would have to assume that when we go to use our learned policy, we will use the predicted value as a mean for a stochastic runtime distribution; but we want to learn a deterministic policy to avoid potentially dangerous actions. Third, we would have to assume the expert human driver's policy was stochastic and know the probability with which they took their action. Fourth, and more practically, the importance sampling would scale our gradient update based on the current distance between θ and $\hat{\theta}$. This may be helpful if it also took into account whether our examples are "good" or "bad", for example by scaling down the effect of far away negative examples or scaling up the effect of far away positive examples, but since it does not, it can have the opposite effect.

There is also a solution to this problem in the supervised learning framework: Stephane Ross el al. Ross et al. (2010) present a solution known as DAgger, that allows the agent to explore and then uses the expert to label the new dataset, then training on all of the combined data. Such an approach has even been improved upon both to address RL by incorporating experts that can label Q values with the AggreVaTeD algorithm (Ross & Bagnell, 2014), and to and to address deep neural networks by finding a policy gradient with the Deeply AggreVaTeD algorithm (Sun et al., 2017). We note that these approaches will not work for us because they require the learning agent to freely make decisions, which we cannot allow. However, we believe that these methods could be of use after the initial training for fine-tuning, once the policy is acceptably safe to control a car with supervision. In the AV industry, it is common to have a safety driver in the car; we could easily have the car drive on-policy, and then have the safety driver take over as needed, storing the correct actions for training. Such supplemental tuning is beyond the scope of this paper, but we point the reader to (Pan et al., 2017) for an AV application of DAgger.

3.1 Loss Function

If θ is the angle in the demonstrated example, $\hat{\theta}$ is the angle predicted by the PNet, and f is the feedback, then the loss function we choose should have the following 3 properties:

1. Minimizing the loss should minimize the distance between θ and $\hat{\theta}$ for positive examples. That is, for positive examples, the loss should increase with the distance between θ and $\hat{\theta}$.
2. Minimizing the loss should maximize the distance between θ and $\hat{\theta}$ for negative examples. That is, for negative examples, the loss should decrease with the distance between θ and $\hat{\theta}$.

3. The rate at which the loss is minimized should be determined by the magnitude of the feedback. That is, the magnitude of the loss should increase with the magnitude of f.

These three properties together ensure that the network avoids the worst negative examples as much as possible, while seeking the best examples. Given an input state s, the first loss function that comes to mind is what we term "scalar loss":

$$Loss_{scalar} = f_s * (\theta_s - \hat{\theta}(s))^2$$

This loss function is notable for several reasons. First, it is a generalization of the behavioral cloning loss function:

$$Loss_{clone} = (\theta_s - \hat{\theta}(s))^2$$

We experimented with two hyperparameters that, when set appropriately, will recover this behavioral cloning loss. The first such parameter is the ability to threshold feedback values. If we threshold, we simply replace every f with $sign(f)$. Thresholding eliminates gradations in positive and negative data and in fact makes the scalar loss effectively identical to the loss used in COACH. Additionally, we introduced the parameter α, which scales down all our negative examples' feedback: $f' = max(f, \alpha f)$. This trades off between behavioral cloning and avoiding negative examples. We apply α after we threshold, so if we threshold with and set α to 0.0, we recover behavioral cloning.

The second reason our scalar loss is notable is that it closely resembles a loss that induces a stochastic policy gradient in standard continuous control reinforcement learning. In a standard RL policy network such as REINFORCE, the loss function would be $Loss = R * -log(Pr(\theta))$, which encourages the probability of the action taken by an amount proportional to the return, R (Williams, 1992). In continuous control, it would be standard to instead predict a mean $\hat{\theta}$ for a normal distribution and then sample your action θ from that normal distribution. If you replace $log(Pr(\theta))$ with the probability density function for a normal distribution, the loss you wind up with looks a lot like our scalar loss: (See the appendix for the derivation.)

$$Loss = R * (\theta - \hat{\theta})^2$$

In addition, this gradient is very similar to that of COACH. The primary differences are: 1) Our feedback is continuous. 2) We have an α parameter for adjusting the importance of the negative samples, which may be necessary for making sure our off-policy loss function is not continually "pushed away" from these negative examples. If set to 1.0, it is the same as COACH. 3) we do not use eligibility traces, since we assume explicit feedback is always given. (Differences 2 and 3 will be elaborated upon shortly.) COACH correctly points out that if we view f as Q^π or A^π, the action-value function and the advantage function respectively, then this converges to a local optimum. Although our feedback does not depend on the current learned policy, π, we could also view our problem as a contextual bandit, since the feedback for every action falls in the same range (although if we take enough bad actions in a row, we will prematurely terminate our episode), or we could look to Deeply AggreVaTeD to recover the same gradient using Q^* (Sun et al., 2017).

Although this similarity provides inspiration, it is in fact not justified in this context by reinforcement learning. Since we are "off-policy", the neural network cannot not influence the probability of seeing an example again, and this leads can lead to problems. In RL, the network could try a bad action, and then would move away from it and not revisit it. Whereas, if we have a bad example in our training set for a given state, on every epoch of training, our neural net will encounter this example and take a step away from it, thus pushing our network as far away from it as possible. In fact, even if we have a positive example for that very state, if we have more negative examples than positive examples, if we are not careful, we may wind up ignoring our positive examples completely in an effort to get away from our negative examples. This case highlights the trouble inherent in using negative examples: It is hard to know how and when to take into account the negative examples and by how much.

Now, let us consider negative examples in the ReNeg framework. For example, as shown in Figure 1, if we perform a regression on positive and negative examples with more negative examples than positive in one state, we may wind up in a case where our loss is minimized by a prediction of positive or negative infinity, and thus our regression is "overwhelmed" by the negative examples. It is nice that in the scalar loss function, negative examples make the loss function grow no faster with the distance between θ and $\hat{\theta}$ than do the positive examples, but we can do better than that. It would be ideal if the rate of growth of the loss slowed for negative examples as the distance increased. In an effort to avoid this, we introduce a fourth desired property:

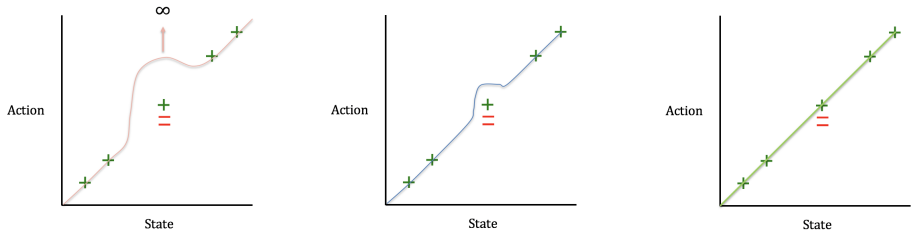

Figure 1: Potential outcomes of regression with negative examples

4. The magnitude of the loss should grow at least linearly with the distance between θ and $\hat{\theta}$ for positive examples. The magnitude of the loss should grow less than linearly with the distance between θ and $\hat{\theta}$ for negative examples.

This enforces that positive examples that are far away matter at least proportionally as much as close positive examples, and negative examples that are far away should matter proportionally less than negative examples close by. This additional property led us to our second "exponential" loss:

$$Loss_{exp} = |\theta(s) - \hat{\theta}(s)|^{2f}$$

Using this loss, negative examples will have infinite loss at distance 0, and then drop off exponentially with distance. We hope that this will create regressions more akin to the second image in Figure 1. In this image, adding more negative points will still nudge the regression away more and more, but one positive point not too close by should be enough to prevent it from diverging to positive or negative infinity. It should be noted that the loss in a particular state still could only have negative examples, especially in a continuous state-space like ours where states are unlikely to be revisited. However, the reduction in loss caused by diverging to infinity would be so small that it should not happen simply due to continuity with nearby states enforced by the structure of the network. In addition, one concern with this loss could be that for positive fractional differences, and negative non-fractional differences, the desired property 3) of loss functions no longer holds. That is, our positive loss will not grow with f if the difference being exponentiated is a fraction. And for negative exponents, the loss will only grow if the difference is a fraction that shrinks as it is raised to increasing powers of f. However, we hope that for negative examples, distances that are more than 1 unit away will not occur often (since 1 unit is half the distance range). We discuss a potential future solution in the appendix to patch this loss function.

Our final loss function should produce regressions more like the final image in Figure 1: We propose directly modelling the feedback with another neural network (which we call the FNet) for use as a loss function. If this FNet is correctly able to learn to copy how we label data with feedback, it could be used as a loss function for regression. Thus, in order to maximize feedback, our loss function would be as follows:

$$Loss_{FNet} = -FNet(s, \hat{\theta})$$

After learning this FNet, we can either use it as a loss function to train a policy network or, every time we want to run inference, we can run a computationally expensive gradient descent optimization to pick the best action. Because the latter does not depend on the training distribution (so we do not have the issue of the runtime and training distributions being different), and it is more efficient, we choose an even easier version of the latter: we pick the best action out of a list of discrete options according to the FNet's predictions. (These approaches would be analogous to a deep deterministic policy-gradient algorithm and a batch-RL continuous DQN in the RL framework (Lillicrap et al., 2015) (Mnih et al., 2013).) One feature of the FNet is that adding more negative points will not "push" our regression further away from this point, but rather just make our FNet more confident of the negative feedback there. This may not be the desired effect for all applications. Moreover, the FNet cannot operate on purely positive points with no gradation. That is, behavioral cloning cannot be recovered from it. There may be ways to regularize the model by maximizing entropy, such as in NAC (Gao et al., 2018) and SAC (Haarnoja et al., 2018), to push down the values of actions our networks have not seen. However, for our purposes, we prefer having a model that can

recover behavioral cloning for the sake of generalizability, so we did not pursue this after seeing the unpromising results for our FNet.

3.2 DRIVING DATA

We recorded 20 minutes of optimal driving and labeled all of this data with a feedback of 1.0. Choosing the suboptimal data and how to label it was a bit more tricky. One reason that we are not using reinforcement learning is that letting the car explore actions is dangerous. In this vein, we wanted to collect data only on "safe" driving. However, the neural network needs data that will teach it about bad actions well as good actions that recover the car from bad states. In order to explore these types of states and actions, we collected two types of bad driving: "swerving" and "lane changing". The first image in Figure 2 is swerving. In swerving, the car was driving in a "sine wave" pattern on either side of the road. We collected 10 minutes of this data on the right side of the road and 10 minutes on the left. The second image is lane changing. For this, we drove to the right side of the road, straightened out, stayed there, and then returned to the middle. We repeated this for 10 minutes, and then collected 10 minutes on the left-hand side as well.

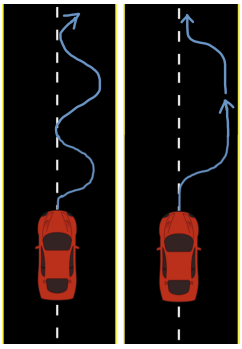

Figure 2: Types of driving: swerving (left), lane change (right)

3.3 "BACKSEAT DRIVER" FEEDBACK

To label the data, we decided to use expert feedback, not expert actions, because it is very hard for an expert labeler to estimate the exact angle at which to turn the wheel to make a turn when you can't see how your actions affect the car. We came up with a feedback framework called Backseat Driver that is easy to collect in the AV context (and others) and constitutes a strong learning signal. Our feedback includes much more information than just a reward (as is used in RL): we take our label to directly measure how "good" an action is relative to other possible actions. We use this approach instead of labeling rewards for actions both because we found it an easy way to label data with feedback, and because it contains more signal. How exactly to label the data with feedback, however, is non-obvious. At first, we considered labeling using a slider from -1 to 1. However, using a slider can be non-intuitive in many cases and there would be discontinuities in feedback you would want to give. For example, if the demonstrator is driving straight off the road and then starts to turn left back onto the road, there would be a large discontinuity in the very negative and then slightly positive feedback.

In order to circumvent these issues, and to make the labeling process more intuitive, we decided to collect feedback using the steering wheel. We found that it is easier for people to focus on the steering angle, since that is how we are used to controlling cars. Our first thought was to just turn the steering wheel to the correct angle. However, this is very difficult to estimate, especially on turns, when you cannot see the actual effects your steering is having on the car. (Note that if we did this, our algorithm would turn into behavioral cloning.) Instead, we decided to label the differential. That is, we turned the wheel to the left if the car should steer more left. This signal shows where the error is (i.e. "You should be turning more" or "You're turning the wrong way"). Note that the label does not need to be the exact correct angle; it just needs to show in which direction the current action is erring, and proportionally how much. We call this method of human feedback collection "Backseat Driver." In order to process the angle labels into a feedback value in $[-1, 1]$, we used the equation below:

FEEDBACK(c, θ)

1 **if** sign(c) == sign(θ) or $|c| \leq \epsilon$
2   **return** $1 - |c|$
3 **else**
4   **return** $-|c|$

Note: We first normalize all of our corrections by dividing by the greatest collected correction c, so all of our c values fall in $[-1, 1]$. In line 1 above, if we are turning the steering wheel in the same

direction as the car (with some ϵ of error), then the feedback should be positive. (We set epsilon to $\frac{5}{\theta_{max}}$ so that it allows up to 5 degrees of tolerance.) Since c represents a delta in steering, a greater delta should result in a less positive signal. Therefore, the feedback should be proportional to $-|c|$. If c is in a different direction than we were steering (line 3), then the feedback should be negative, so we just return $-|c|$ as the feedback. Thus the greater the delta, the more negative the feedback will be. If c is in the same direction, on the other hand, we chose to scale these feedbacks up so that the feedback is positive, but less positive for a greater differential. This makes sense since if, for example, the car is steering left and we tell it to steer more left, this is not as bad as if the car is steering the wrong way. Thus, slow actions back to the center of the road will be rewarded less than quick actions back to the center of the road.

3.4 ARCHITECTURE

We chose to use only an hour of data because we wanted to see how far we could get with limited data. While our feedback is relatively easy to collect compared to other options, it still takes up human hours, so we would like to limit its necessity. We sampled states at a rate of approximately two frames per second, since states that are close in time tend to look very similar. We augmented our data by flipping it left-right, inverting the angle label, and leaving the feedback the same. After this augmentation, we had 17,918 training images and 3,162 validation images (a 85:15 split).

We chose to learn our policy with an end-to-end optimization of a neural network to approximate a continuous control function. Such networks are capable of tackling a wide array of general problems and end-to-end learning has the potential to better optimize all aspects of the pipeline for the task at hand. Given the option to use separate (differentiable) modules for sub-tasks such as computer vision, or to connect these modules in a way that is differentiable for the specific task, the latter will always perform better, since the end-to-end model always has the ability to simply not update the sub-module if it will not help training loss. We used transfer learning to help to bootstrap learning with limited data for both the PNet and Fnet. We decided to start with a pretrained Inception v3 since it has relatively few parameters, but has been trained to have strong performance on ImageNet, giving us a head start on any related computer vision task. We kept some early layers of Inception and added several of our own, followed by a tanh activation for the output. We tried splitting the network at various layers and found that one about halfway through the network (called mixed_2) worked best.

4 EXPERIMENTS

We tested our trained models by running them eight times in the simulator and recording the time until the car crashed or all four tires left the road. In the case that the car drove off the road and came back (which happened very rarely), we discarded that run and started over. We first tested our PNet with the scalar loss, our PNet with the exponential loss, and our FNet. We plotted their mean times over the eight runs with standard deviation. See Figure 3 below. (All of these were on the default settings, except for the exponential loss model, for which we set α to 0.1 since otherwise we could not get it to converge.) Note: Based on the predicted angles, the FNet seemed to primarily predict feedback based on state, not angle; this makes sense given that the feedback in "bad" states is generally "bad", except for the split second when the "good" action takes place.

Given that the scalar loss performed best (and was training correctly), we spent more time tuning the hyperparameters for this model. This can be seen in Figure 4. We note several interesting things from figure 4 exploring the scalar loss. First, the pink group is identical to the solid blue group except that all the α values are 0, meaning that negative-feedback examples are zeroed out and effectively ignored in training. The blue bars are much higher than their pink counterparts, indicating that the negative data is useful. Second, we can see that thresholding the feedback to -1 and 1 (the blue crosshatch pattern) increased the scalar performance about 30 seconds (compared to 6 seconds with the default hyperparameters). At first this could be taken to indicate that having gradations in positive and negative data could be harmful to training. However, we see that the same performance is achieved (and surpassed) by increasing the learning rate instead of thresholding, by looking to the three blue bars to the right. The reason for this is likely that we tuned the learning rate to work well on thresholded data, and so, when we don't threshold or clone our data, the scalar on the loss drops significantly, forcing the network to take smaller steps, and effectively decreasing the

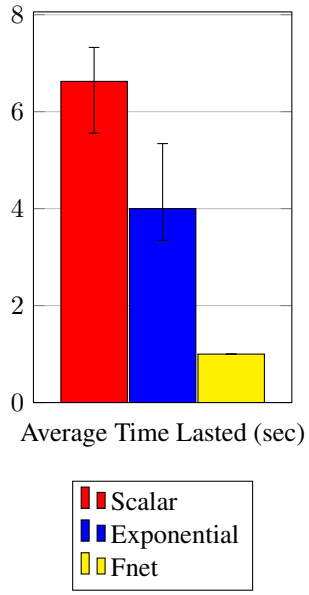

Figure 3: Initial comparison of loss functions.

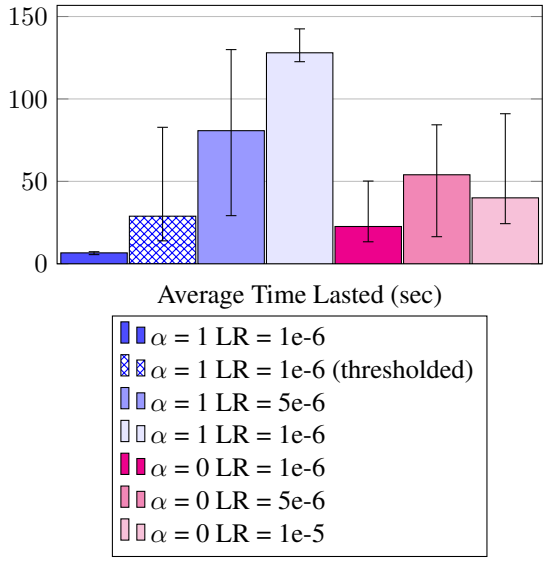

Figure 4: Scalar loss function performance organized by hyperparameters

learning rate. Increasing the learning rate instead of thresholding yields much better performance, indicating that gradations in the data (with a high learning rate) do help training.

After selecting the best learning rate for each model, we then trained new versions of each network 2 more times. (The best learning rate for both turned out to be 1e-5; see the appendix for more details on tuning.) Each time, we let it drive for 8 trials and calculated the performance as the mean time before crashing over these 8 trials. We then calculated the mean performance for each over the 3 training sessions. Figure 5 shows the results.

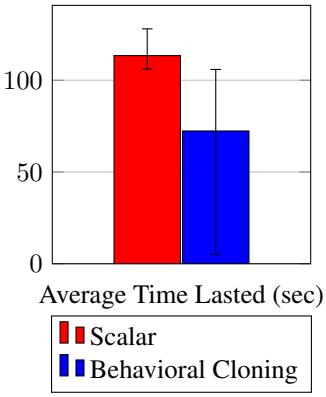

Figure 5: Over 3 training runs, our scalar loss model performed over 1.5 times as well as the behavioral cloning benchmark, with significantly less variance.

5 CONCLUSION

We hypothesized that for the task of learning lane following for autonomous vehicles from demonstration, adding in negative examples would improve model performance. Our scalar loss model performed over 1.5 times as well as the behavioral cloning baseline, showing our hypothesis to be true. The specific method of regression with negative examples we used allows for learning deterministic continuous control problems from demonstration from any range of good and bad behavior. Moreover, the loss function that empirically worked the best in this domain does not require an additional neural network to model it, and it induces a stochastic policy gradient that could be used for fine-tuning with RL. We also introduced a novel way of collecting continuous human feedback for autonomous vehicles intuitively and efficiently, called Backseat Driver. We thus believe our work could be extremely useful in the autonomous control industry: with no additional real world time, we can increase performance over supervised learning by simply having a backseat driver.

REFERENCES

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press, 2004.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

Benjamin Burchfiel, Carlo Tomasi, and Ronald Parr. Distance minimization for reward learning from scored trajectories. In *AAAI*, 2016.

Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. *CoRR*, abs/1205.4839, 2012. URL http://arxiv.org/abs/1205.4839.

Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *CoRR*, abs/1802.05313, 2018. URL http://arxiv.org/abs/1802.05313.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. Learning driving styles for autonomous vehicles from demonstration. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2641–2646, 2015.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

James MacGlashan, Mark K. Ho, Robert Tyler Loftin, Bei Peng, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive learning from policy-dependent human feedback. *CoRR*, abs/1701.06049, 2017. URL http://arxiv.org/abs/1701.06049.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. *CoRR*, abs/1606.02647, 2016. URL http://arxiv.org/abs/1606.02647.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *CoRR*, abs/1709.07174, 2017. URL http://arxiv.org/abs/1709.07174.

Stéphane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979, 2014. URL http://arxiv.org/abs/1406.5979.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL http://arxiv.org/abs/1011.0686.

Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pp. 1060–1068, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-4239-1. URL http://dl.acm.org/citation.cfm?id=2936924.2937079.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–387–I–395. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3044850.

Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *CoRR*, abs/1703.01030, 2017. URL http://arxiv.org/abs/1703.01030.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pp. 279–292, 1992.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

6 APPENDIX

6.1 HYPERPARAMETERS

We added fully connected layers of sizes 100, 300, and 20, in order. Our batch size was 100 and we trained for 5 epochs. Unless otherwise specified for a given model in the experiments, we used an α value of 1.0, we did not threshold, and we used a learning rate of 1e-6. As in the Inception model we were using, our input was bilinearly sampled to match the resolution 299x299. Likewise, we subtracted off an assumed mean of 256.0/2.0 and divided by an assumed standard deviation of 256.0/2.0.

6.2 TRAINING METRICS

During training, we kept track of two validation metrics: the loss for the model being trained, and the average absolute error on just the positive data multiplied by 50. The first we refer to as "loss" and the second we refer to as "cloning error" (since it is the 50 times the square root of the cloning error) or just "error". The reason we multiplied by 50 is that this is how Unity converts the -1 to 1 number to a steering angle, so the error is the average angle our model is off by on the positive data. (This is true with the maximum angle set to 50.)

During training, these two metrics generally behaved very similarly, however, in the models for which we increased the learning rate, these eventually start to diverge. In this case, the error on the positive data started to increase, but the loss was still decreasing. For this reason, we tried varying the learning rate on several models, to see if the loss was more important than the "cloning" error. It is clear that the behavioral cloning models (thresholded with $\alpha = 0.0$) should in general do better on the "cloning" error, since they are very closely related. Whereas for non-thresholded data, it was trained with examples weighted differently. And for the negative data, it was trained to also get "away" from negative examples. We hope that even though the cloning error may increase, this means that it is because the model is choosing something better than (yet further away from) the positive examples. We still use the cloning error, however, because it is a useful intuitive metric for training and comparison.

6.3 LEARNING RATE TUNING

We tried several learning rates for both behavioral cloning and ReNeg. We compared the performance, shown in figures 6 and 7, and found that 1e-5 worked best for both.

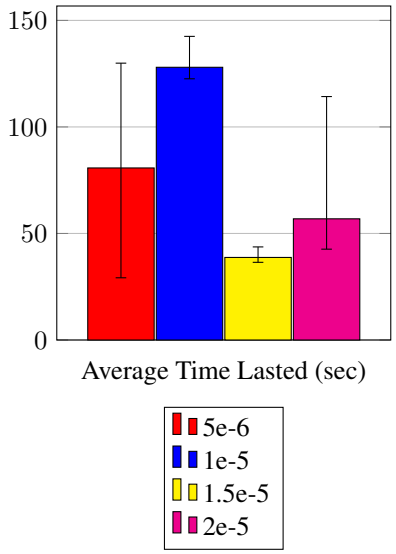

Figure 6: The scalar loss performed best with a learning rate of 1e-5.

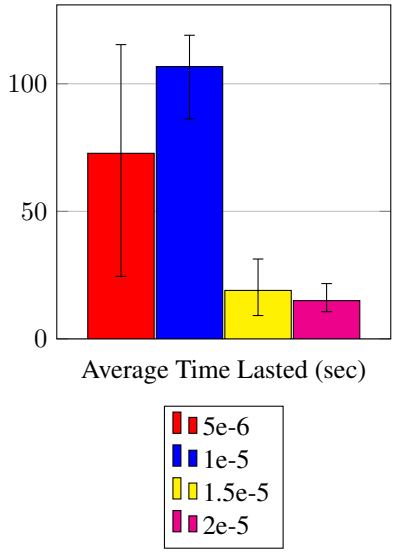

Figure 7: The behavioral cloning loss performed best with a learning rate of 1e-5.

6.4 Policy Gradient Derivation

Here is the derivation from the stochastic policy gradient to the loss that induces it, which is very similar to our scalar loss:

$$\nabla Loss = \nabla(R * -log(Pr(\theta)))$$

$$\nabla Loss = \nabla(R * -log(\frac{e^{-\frac{(\theta-\hat{\theta})^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}))$$

$$\nabla Loss = \nabla(R * -(log(e^{-\frac{(\theta-\hat{\theta})^2}{2\sigma^2}}) - log(\sqrt{2\pi\sigma^2})))$$

$$\nabla Loss = \nabla(R * -(log(e^{-\frac{(\theta-\hat{\theta})^2}{2\sigma^2}}))$$

$$\nabla Loss = \nabla(R * \frac{(\theta-\hat{\theta})^2}{2\sigma^2})$$

$$\nabla Loss \propto \nabla(R * (\theta-\hat{\theta})^2)$$

$$Loss \propto R * (\theta-\hat{\theta})^2$$

6.5 Future Research

Future research involving human feedback should focus on 2 things: the loss function and enforcing continuity.

(Note that fine-tuning the policy once it is acceptably safe is also an interesting problem, and supervised approaches involving a safety driver taking over control, and retraining on this data a la DAgger, should probably be explored.)

6.5.1 Loss Function

Immediate next steps should likely focus on alterations to the loss function; there do exist ways we can achieve all desired properties 1) to 4). There are issues with both our scalar loss function and our exponential loss function. Our exponential loss function was created to satisfy property 4): the magnitude of the loss for negative examples should drop off exponentially with respect to the distance. This is a problem since, in our scalar loss, the negative examples actually have an exponentially increasing affect as the distance increases. Although the exponential loss accomplishes this solution, there is a dilemma: our exponential solution violates the desired the desired property

3) as mentioned earlier. That is, we want the magnitude of the loss to increase with the magnitude of the feedback, f. However, for positive examples, this only is the case when the absolute difference between θ and $\hat{\theta}$ is greater than 1, and for negative examples, this is only the case when the absolute difference is less than 1. Although this may not be a large concern for the negative examples, since the difference is at most 2, this could be an issue for our positive examples. This issue with the exponential loss function can actually be solved in two different ways. First, we can think of an elegant solution that will modify our scalar loss to have this exponential decay property and satisfy all desiderata. Second, we can think of a way to modify our scalar loss to have neither an exponential increase with distance nor an exponential increase. And third, we can think of an "ugly" patch for our exponential function, if we really must have the loss be exponential in f.

1. We can accomplish this exponential decay by modifying our scalar function in a very easy way: Move the sign of f into the exponent:

$$Loss_{scalar} = |f| * (\theta(s) - \hat{\theta}(s))^{2*sign(f)}$$

Using this loss function we have all three properties satisfied. That is, positive examples encourage moving towards them, negative examples encourage moving away from them, and the amount of this movement increases with the magnitude of f. Moreover, we also have the properly that, in negative examples, loss drops of exponentially with the distance from the negative example (because we are dividing by it).

2. If we want our scalar loss function to have neither an exponential decay nor an exponential increase with the distance from the negative points, we can simply use the following loss:

$$Loss_{scalar} = f * |\theta(s) - \hat{\theta}(s)|$$

This has the not-so-nice property that, in the positive example, it allows outliers much more easily than the traditional squared loss. However, it has the very nice property that, given a single state input, as long as you have more positive examples than negative examples, your loss will always be minimized in that state by a value between your positive examples. This is because, as soon as you get to your greatest or least positive example, every step away from your positive examples will cost you 1 loss, for each positive example you have, and you will only lose 1 loss for each negative example you have. (Note, if you are not thresholding, then this translates to more total $|f|$ for positive examples than negative examples.)

3. If we really want our feedback to be in the exponent, we can accomplish this with a more complex solution. We can split our examples into positive and negative examples and have a loss for each. The positive and negative loss functions could respectively be:

$$Loss_{exp,positive} = min(|\theta(s) - \hat{\theta}(s)|/tol, 1)^{2f}$$

$$Loss_{exp,negative} = (|\theta(s) - \hat{\theta}(s)|/2)^{2f}$$

For the positive examples, the magnitude of the loss increases with f in the case that the number being raised to the $2f$ is greater than 1. If $|\theta(s) - \hat{\theta}(s)|$ is greater than tol, we have both ensured this is the case and we are back to our original loss function, but scaled by a constant factor of tol. But what happens if the difference is less than tol? In that case, for any estimate $\hat{\theta}$ within tol of the intended $theta$, our parameters will have a derivative of 0 (no effect), due to the min. (If we want this to be in degrees, we can just set adjust this to tol/=50). (This will cause the neural network to forget about examples that it does well on until they become sufficiently problematic again. This may be helpful in that the network can focus on the area it needs to, or it could be harmful in that is prevents convergence.) For the negative examples, the magnitude of the loss will only increase with f if the number being raised to the $2f$ is less than 1, because we are dividing my it. To ensure this is the case, we can just divide by our range, which is 2.0. To combine these two functions, we can just pick the best convex combination of them.

6.5.2 CONTINUITY

Because, in both our scalar and exponential loss, our loss function at a given state with just a negative example is minimized by moving away from the negative example, our regression in that state will

tend toward positive or negative infinity. Certainly having a cost on negative examples that drops of exponentially will help, but it may not be enough. Moreover, we may not want to rely on the structure of neural networks to discourage this discontinuity. Therefore, research could be done on adding a regularization term to the loss that penalizes discontinuity. That is, we would add some small loss based on how dissimilar the answers for nearby states are. Of course, this implies a distance metric over states, but using consecutive frames may suffice.