Preference-based Learning of Reward Function Features

Sydney M. Katz*, Amir Maleki*, Erdem Bıyık, and Mykel J. Kochenderfer

Abstract— **Preference-based learning of reward functions, where the reward function is learned using comparison data, has been well studied for complex robotic tasks such as autonomous driving. Existing algorithms have focused on learning reward functions that are linear in a set of trajectory features. The features are typically hand-coded, and preference-based learning is used to determine a particular user's relative weighting for each feature. Designing a representative set of features to encode reward is challenging and can result in inaccurate models that fail to model the users' preferences or perform the task properly. In this paper, we present a method to learn both the relative weighting among features as well as additional features that help encode a user's reward function. The additional features are modeled as a neural network that is trained on the data from pairwise comparison queries. We apply our methods to a driving scenario used in previous work and compare the predictive power of our method to that of only hand-coded features. We perform additional analysis to interpret the learned features and examine the optimal trajectories. Our results show that adding an additional learned feature to the reward model enhances both its predictive power and expressiveness, producing unique results for each user.**

I. Introduction

If designed properly, reward functions provide a means for humans to convey desired behavior to a robot. However, creating a reward function that accurately encodes human intent can be difficult in complicated, high-dimensional problem settings. Preference-based learning has been proposed as a way to address this challenge. By querying the human user with a set of demonstrated trajectories and asking which demonstration they prefer, the robot learns the reward function it should optimize. Its motivation is similar to that of inverse reinforcement learning (IRL) in that it allows humans to train a policy without explicitly specifying the values of the reward function parameters [1]. For this reason, preference-based learning has been successfully applied to challenging problems in robotics [2]–[16].

While some preference-based learning algorithms focus on learning a policy directly [3], [4], others focus on learning a reward or utility function [10]–[19], which typically involves a lower-dimensional parameter space to make results easier to interpret. Current state-of-the-art algorithms provide methods to learn reward functions that are linear in a set of features [12]–[17], [20], [21]. Features are hand-coded functions of a robot trajectory that remain constant throughout the learning process. Responses to preference queries are used to learn weights that indicate the relative importance of each feature. For example, in a driving scenario, preference-based learning has been used to determine the relative importance of features such as collision avoidance and keeping speed by obtaining pairwise preferences over driving trajectories [15].

While there have been studies to relax the linearity assumption by modeling reward as a mixture of linear functions conditioned on latent states of the user [22] or as a Gaussian process [11], the features were still hand-designed. Because reward functions are based entirely on hand-coded features, the features must be expressive enough to model human intent and can be difficult to design appropriately [15]–[17]. Feature selection presents three main challenges: selecting representative features, selecting expressive features, and selecting the proper functional form. Features must be representative enough to accurately model the reward function for a particular user. Sadigh *et al.* [15] found that applying preference-based learning to understand driving behavior resulted in similar reward functions among study participants. In order to distinguish between specific users' personal driving preferences, a more expressive set of features may be required [15]. Finally, applying preference-based learning to a variety of tasks requires complex, hand-tuned, nonlinear feature functions, and designing them is challenging [12].

In this work, we address some of these challenges by providing a framework to not only learn the weights of a linear reward function but also to learn additional nonlinear features. We represent features as a neural network that is trained on a user's responses to queries. We apply the learning framework to the driving problem originally presented by Sadigh *et al.* [15] and compare the results using the learned features to those using only hand-coded features. Not only do we evaluate the model's predictive power, but we also visualize and interpret the features learned by the neural network. Our results demonstrate that an additional nonlinear feature improves the predictive power of the framework and enables it to better personalize users' preferences.

II. Approach

Our approach to feature learning requires two phases. The first involves obtaining expert preferences using an active querying method. The second is to use the preferences to learn features. Let $\phi(\mathbf{x}^t)$ be a function that maps the state $\mathbf{x}$ of a robotic system at time t to a set of features. The reward function takes the form

$$r(\mathbf{x}^t) = \mathbf{w}^\top \phi(\mathbf{x}^t) \tag{1}$$

where $\mathbf{w}$ is a vector that specifies the relative weighting of each feature. Defining a trajectory τ to consist of an initial state and a set of k subsequent states, we let $\Phi(\tau) =$

*Denotes equal contribution

Stanford University, Stanford, CA 94305, {smkatz, amir.maleki, ebiyik, mykel}@stanford.edu

$\sum_{t=0}^{k} \phi(\mathbf{x}^t)$. The reward over a particular trajectory is then

$$R(\tau) = \mathbf{w}^\top \Phi(\tau) \tag{2}$$

In order to preserve the interpretability of the final model, we select a mixed feature model in which the feature function contains both hand-coded and neural network features. Thus,

$$r(\mathbf{x}^t) = [\mathbf{w}_{hc}, \mathbf{w}_{nn}]^\top [\phi_{hc}(\mathbf{x}^t), \phi_{nn}(\mathbf{x}^t)] \tag{3}$$

where $\phi_{hc}(\mathbf{x}^t)$ represents the hand-coded feature function with corresponding weight vector $\mathbf{w}_{hc}$, and $\phi_{nn}(\mathbf{x}^t)$ represents the neural network feature function with corresponding weight vector $\mathbf{w}_{nn}$.

The methods outlined in our approach discuss techniques to learn $\mathbf{w}_{hc}$, $\mathbf{w}_{nn}$, and the function $\phi_{nn}(\mathbf{x}^t)$. We note that requiring hand-coded features still presents a challenge in feature design. However, learning additional features using a neural network can capture important trajectory qualities that the feature designer may have missed.

A. Preference Model

Preference-based learning is an iterative process between querying the user for their preference and updating our model. We keep a distribution $p(\mathbf{w})$ over possible values of $\mathbf{w}$ and perform Bayesian updates to it as we obtain preferences. Let the nth pairwise comparison query contain the trajectories $\tau_a^{(n)}$ and $\tau_b^{(n)}$. We define I_n as the response to the nth query, where

$$I_n = \begin{cases} +1, & \tau_a^{(n)} \succ \tau_b^{(n)} \\ -1, & \tau_a^{(n)} \prec \tau_b^{(n)} \end{cases} \tag{4}$$

with $\tau_a \succ \tau_b$ representing the user's preference of τ_a to τ_b. The Bayesian update can be written as follows:

$$p(\mathbf{w} \mid I_n) \propto p(I_n \mid \mathbf{w}) p(\mathbf{w}) \tag{5}$$

where $p(\mathbf{w})$ encodes the current distribution over $\mathbf{w}$ that takes into account responses $I_{1:n-1}$. Before any preferences have been obtained, we assume a uniform prior over the search space of possible values.

In order to perform this update, we must specify a likelihood model for $p(I_n \mid \mathbf{w})$ keeping in mind that we expect occasional errors from the user. As in Sadigh *et al.* [15], we use a sigmoid likelihood function:

$$p(I_n \mid \mathbf{w}) = \frac{1}{1 + \exp[-I_n(R(\tau_a^{(n)}) - R(\tau_b^{(n)}))]} \tag{6}$$

Samples from the posterior $p(\mathbf{w} \mid I_n)$ can be generated using Markov Chain Monte Carlo (MCMC) methods. In particular, we use the adaptive Metropolis algorithm to efficiently generate samples at each iteration [23].

B. Active Querying

Active querying methods are desirable because they decrease the number of required queries by asking questions based on current model uncertainty inferred from prior user preferences. Methods for active querying have been proposed in the literature [12], [14]–[17]. These methods typically rely on solving an optimization problem over a continuous action space or control input [12], [15], [16]. One objective is to select the set of control inputs that maximizes the volume removed from the current distribution over model parameters [15], [16]. In later work, better performance was achieved using an objective that seeks to maximize the information gained from each query [12].

Although these objectives generate queries that are maximally informative, they do not provide a direct incentive to generate query trajectories that are realistic. Figure 1 shows an example of two possible queries that can be shown to a user for a driving scenario. The user is presented with two options and is asked to select their preferred trajectory for the blue vehicle.

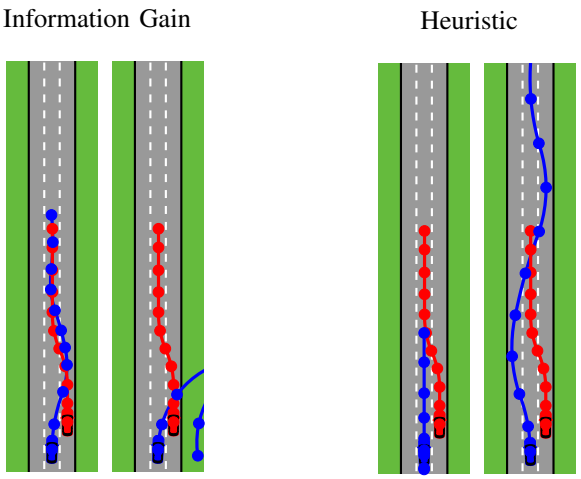

Fig. 1. Example queries with information gain objective (left) and heuristic method (right). Marks are placed at equal time intervals on all trajectories. The car in the right panel of the information gain query leaves the road and backs up to the starting latitude.

The query on the left of fig. 1 is generated using the information gain objective [12]. The user is asked to select between a scenario in which the blue vehicle repeatedly crashes into the other vehicle and a scenario in which the blue vehicle drives off the road and begins to drive backward. While these trajectories are informative with respect to the collision avoidance and staying within the lanes on the road, neither trajectory is realistic. Because we want to learn features that allow us to express specific user intentions, it is especially important to show users realistic trajectories during the active learning process. For this reason, we adopt an approach that shows users trajectories that have been optimized for reward functions defined by different vectors $\mathbf{w}$ [17]. By maximizing a reward function for each trajectory, we ensure that the trajectories will have realistic features. A query generated using this approach is shown in the right half of fig. 1.

To choose these $\mathbf{w}$ vectors from the samples from $p(\mathbf{w})$, we formulate a multiobjective optimization problem similar to prior work [3], [17]. Let M be the number of MCMC samples generated after each Bayesian update, and let $\mathbf{w}_i$ be the ith sample in this set. The optimization problem can be

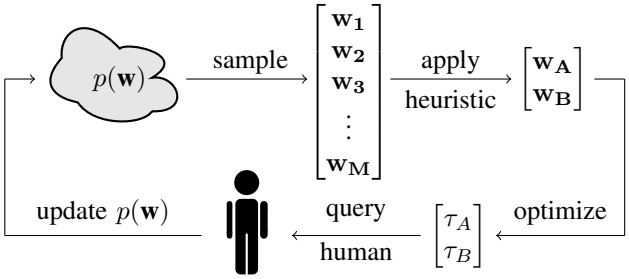

Fig. 2. Overview of the iterative learning process.

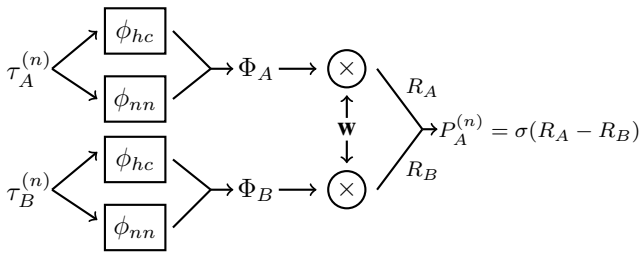

Fig. 3. Estimating the probability of selecting trajectory A for a particular query based on the current feature functions and weights. The σ in the last step represents a sigmoid function.

written as

$$\underset{i,j \text{ s.t. } i \neq j}{\text{maximize}} \; p(\mathbf{w}_i)p(\mathbf{w}_j) + \mu \|\mathbf{w}_i - \mathbf{w}_j\|_2 \tag{7}$$

where $\mu \geq 0$ controls the balance between the objectives. The first term incentivizes selecting weights that are likely based on the current estimate of $p(\mathbf{w})$, and the second term ensures we select samples that are different enough to produce a distinguishable query. To calculate the first term, we use unnormalized posteriors obtained by eq. (5).

Figure 2 outlines the overall active querying process. First, samples are drawn from $p(\mathbf{w})$ using MCMC. Next, we apply our multiobjective optimization heuristic to select two of these samples to use for the next query. Query trajectories are generated by maximizing the reward function of each sample and subsequently shown to the user. The optimization is nonconvex and is solved using the quasi-Newton method L-BFGS [18]. We solve the optimization problem 10 times starting from random initial points and show the query with the best objective value. After we obtain the user's preference, we update the posterior $p(\mathbf{w})$ and the process resumes with sampling.

C. Feature Learning

We seek to learn neural network features to augment our feature function $\Phi(\tau)$ so that there exists a weight vector $\mathbf{w}$ such that the linear function $\mathbf{w}^\top \Phi(\tau)$ is a good predictor of reward. More specifically, we want to predict higher reward for the trajectory that the user selected in each query. Note that in the mixed feature setting, $\phi(\mathbf{x}^t)$ is influenced not only by $\phi_{nn}(\mathbf{x}^t)$ but also by $\phi_{hc}(\mathbf{x}^t)$, and we must take this into account in our training.

1) Network Structure: Our learned feature function $\phi_{nn}(\mathbf{x}^t)$ is represented by a neural network in which the input is a function of the state of the system. For a simple system, this function can be the identity mapping (i.e. the input to the network is the state of the system). The final layer of the network contains one neuron per feature and represents the learned features. Because the hand-coded features are normalized to have a magnitude less than or equal to 1 when averaged over the trajectory, we select hyperbolic tangent as the activation function for the output layer. This activation ensures that features all have similar magnitude.

2) Loss Function: The feature learning problem can be thought of as a classification problem in that we would like to find features that allow us to make binary predictions on queries regarding whether or not a user prefers trajectory A to trajectory B. Framing the problem in this way allows us to use a cross entropy loss function that takes in the probability of the user selecting τ_A given the current reward function model:

$$\text{loss} = \frac{1}{N} \sum_{n=1}^{N} \left[z_n \log(P_A^{(n)}) - (1 - z_n) \log(1 - P_A^{(n)}) \right] \tag{8}$$

where $z_n = I_n$ if $I_n = 1$, $z_n = 0$ if $I_n = -1$, $P_A^{(n)}$ is the probability of the user selecting $\tau_A^{(n)}$ over $\tau_B^{(n)}$, and N is the number of preference queries in the training set. Equation (6) can be used to calculate $P_A^{(n)}$, and fig. 3 provides a visual representation of this process. We allow our model to train both the neural network parameters and the linear reward weights $\mathbf{w}$.

III. Experimental Setup

A. Problem Domain

We tested our approach on the driving simulation[1] that has been used in previous preference-based learning works [12], [14]–[16]. The right panels in fig. 1 show an example of a preference query that would be shown to a user. The red car, which represents the human-driven vehicle, starts in the rightmost lane and switches to the middle lane. The human-driven vehicle follows the same trajectory in every scenario. The user is asked to select the trajectory they would prefer the blue car, or robot-driven vehicle, to follow. The state is defined as $[x_r, y_r, \theta_r, v_r, x_h, y_h, \theta_h, v_h]$ where x is the vehicle's latitudinal position on the road, y is the vehicle's longitudinal position, θ is the vehicle's heading ($\theta = 90^\circ$ corresponds to driving straight along the road), and v is the vehicle's speed. A subscript r denotes a state corresponding to the robot car, and a subscript h denotes a state associated with the human car.

1) Hand-coded Features: The hand-coded features are simlar to those used in Bıyık *et al.* [16] In order to narrow our search space of vectors $\mathbf{w}$, we design our features such that higher values are preferred and restrict the elements of $\mathbf{w}$ to be nonnegative. The features are summarized in table I.

[1]Source is at github.com/sisl/FeatureLearningPrefs.

TABLE I
HAND-CODED FEATURES

Description	Expression
staying in lane	$\exp\left[\min((x_r-0.17)^2, x_r^2, (x_r+0.17)^2)\right]$
keeping speed	$-(v_r-1)^2$
heading	$\sin(\theta_r)$
collision avoidance	$-\exp\left[-(7(x_r-x_h)^2+3(y_r-y_h)^2)\right]$

In this particular scenario, lane centers are located at $x = -0.17$, $x = 0$, and $x = 0.17$, so the first feature represents the minimum distance to a lane center. The collision avoidance feature was designed for a vehicle with an aspect ratio of 7/3. These features are expressive enough to produce a reasonable optimal trajectory given a weight vector $\mathbf{w}$; however, we hypothesize that this set of features alone is not expressive enough to fully describe a driver's reward function and differentiate driving styles [12], [15].

2) Neural Network Structure and Inputs: For baseline testing, the input to our neural network is a function of the robot and human cars' states. Specifically, we input x_r, θ_r, and v_r from the robot state. However, while x_r provides an indication of whether or not the car is staying in its lane, y_r does not provide relevant information on its own. Instead, we to provide the network with the distance between the robot car and human car defined as

$$d = \sqrt{(x_r - x_h)^2 + (y_r - y_h)^2} \tag{9}$$

Thus, our neural network input is the vector $[x_r, d, \theta_r, v_r]$. After initial testing, we experimented with adding an extra input to the network as discussed in section IV. In this experiment, we learn one extra feature to augment the hand-coded features, so the final layer of the network consists of a single neuron. The network is a fully connected feedforward network with one hidden layer containing 100 neurons with Rectified Linear Unit (ReLU) activation functions. We tested the effect of using more than one extra feature. However, we found the learned features to be redundant without a significant performance improvement. A single neural network feature was complex enough to capture many aspects of driving (see section IV).

B. Training Details

While the linear reward weight for the neural network feature $\mathbf{w}_{nn}$ and neural network parameters were randomly initialized, the linear reward weights for the hand-coded features $\mathbf{w}_{hc}$ were initialized to the estimates obtained during the active query process. For each training epoch, the gradient of the loss function is computed with respect to the neural network parameters, and the neural network parameters are updated using the NADAM algorithm with a learning rate of 0.001 and β parameters of (0.9, 0.999) [24].

As the neural network parameters are updated and its output feature value changes relative to the hand-coded feature values, the linear reward weights may no longer be optimal. For this reason, we update the linear reward during training as well; however, because the linear reward weights have already been learned from the active querying process, we found that we could improve learning by only updating them every 20 epochs via gradient descent. We divide the data into training, validation, and test sets and repeat the training process over 40 trials, resetting the linear reward weights between trials. We select the 5 neural networks out of the 40 trials that perform best at prediction on the validation set for our evaluations on the test set.

C. Dependent Measures

We assess the *predictive power* of the models by evaluating the ratio of user comparisons (in the test set) that are correctly predicted by the learned reward function. We also qualitatively evaluate the model's ability to generate safe, customized optimal trajectories that reflect the different driving preferences of the users.

D. Hypotheses

We test the following hypotheses with our experiments. Using the learned neural network features in addition to the hand-coded features (**H1**) *improves the predictive power of the reward model* and (**H2**) *enables better customized optimal driving trajectories*.

E. User Study Procedure

To test the hypotheses, we conducted a user study with 15 participants, all of whom have a valid driver's license. Each participant was shown 100 actively generated queries using the method described in section II-B and the hand-coded features. Because the queries are actively generated based on user responses, each participant was shown a different set of trajectories during this part of the study. For the next part of the study, we created a standardized set of 75 queries as a test set. The queries in the test set were generated by sampling random pairs of values for $\mathbf{w}_{hc}$ and solving for the locally optimal trajectories.

For neural network training, we use the first 70 trajectory pairs presented to the user as the training data with the preferences obtained from the user as the training labels. We use the remaining 30 as the validation set. The 75 standardized trajectory pairs from the second part of the study and corresponding user responses serve as the test data and labels. We compared our method, which involves both the hand-coded and the learned features, with the method in prior works that uses only hand-coded features.

IV. RESULTS

We analyzed the neural network features from the user study for both their predictive power and interpretability.

A. Predictive Power

We analyzed the prediction accuracy of both the hand-coded and mixed feature sets for each user on the standardized test set. Figure 4 shows the results for each user. For all users except user 3, the mixed feature set shows an improvement over the hand-coded only accuracy. Improvements range from 2% to 21%. This result strongly supports **H1**. Users with small improvements tended to have high

test accuracy using the hand-coded only features. A high test accuracy with the hand-coded features indicates that the hand-coded features alone provide an adequate encoding of the user's reward function, and further learning may not be necessary. For instance, user 3 had the highest hand-coded test accuracy of all users in the study, which may explain the inability to learn a feature that improves the predictive power of their reward function. On the other hand, the mixed feature set showed significant increases in performance for users whose preferences could not be fully described by the hand-coded features.

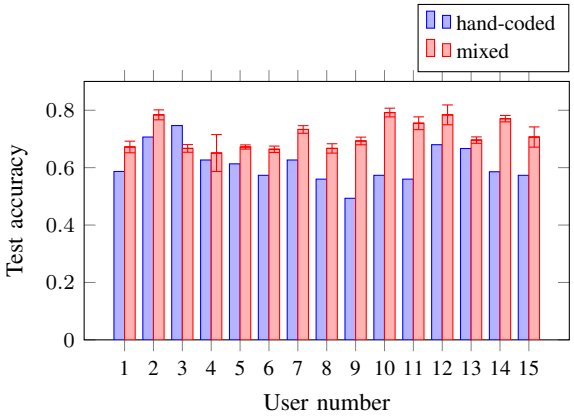

Fig. 4. Test accuracy of hand-coded and mixed feature sets for each user. While obtaining the hand-coded test accuracy (blue bars) is a deterministic process, the red bar heights represent the mean of the test accuracy of the neural networks that had the top 5 validation accuracy values. Error bars show standard deviation among the 5 neural network trials.

B. Feature Interpretation

Because the input to the neural network is four dimensional, we can only visualize the feature values for slices of the input space. Figure 5 shows the change in the neural network feature value for changes in heading for user 1 and compares it with the hand-coded feature for maintaining heading. The vehicle is located in the center lane at a distance $d = 0.5$ from the other vehicle and is traveling at 80% of maximum speed.

It is clear that the neural network feature for user 1 is influenced by vehicle heading. The feature value has a sharp peak around $90°$, which corresponds to driving straight down the road. Because reward weights are positive, a higher neural network feature value indicates a greater positive contribution to the total reward. Therefore, the neural network feature will contribute to a higher reward for trajectories with vehicle headings near $90°$. While the hand-coded heading feature penalizes facing backward (heading of 270 degrees) more than it penalizes facing left and right (heading of 0 or 180 degrees), the neural network feature does not show the same trend. Instead, it has a sharp peak around $90°$ but then quickly decreases to the minimum value.

Figure 6 shows the variation in the neural network feature value for user 1 for various locations of the human-driven vehicle on the road. Speed is held constant at the maximum speed, and heading is held constant at $90°$. This visualization

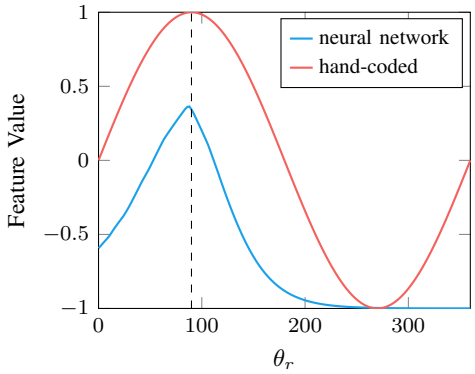

Fig. 5. Variation in neural network feature value and hand-coded heading feature value with heading for user 1. The blue line shows the output of the neural network when $x_r = 0.0$, $d = 0.5$, and $v_r = 0.8$ and θ_r is varied from 0 to 360 degrees. The red line represents $\sin\theta_r$, the hand-coded feature for heading. The dashed line shows a heading of $90°$, corresponding to driving straight down the road.

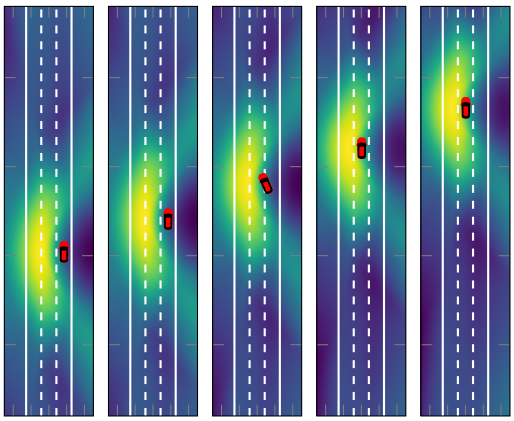

Fig. 6. Heat map of neural network feature value for various locations of the human-driven vehicle (represented by the red car) for user 1. The speed is held constant at $v_r = 1$, and heading is held constant at $\theta_r = 90°$. Brighter colors indicate higher feature values.

demonstrates that the neural network feature has a form of collision avoidance built into it. The feature value decreases in close proximity to the other vehicle on the road. Additionally, the value drops off outside the lanes of the road. The value is highest in the region of the road just to the left of the other vehicle. This trend can be interpreted as a preference to maintain speed and keep up with traffic.

The heat map in fig. 6 is symmetric across the front and back of the red vehicle. This symmetry is due to the fact that we are using the distance between the two cars as input to the neural network; the neural network does not have access to the $y-$position of the vehicle. The network therefore has no notion of whether or not the human-driven vehicle on the road is in front of or behind the robot-driven vehicle. To confirm this, we trained a new neural network with an additional input defined by $(y_r - y_h)/(v_r - v_h)$. Positive values mean the longitudinal distance between the two vehicles is increasing, while negative values indicate it is decreasing. Figure 7 shows that when we add this fifth input to the network, we are able to break the symmetry.

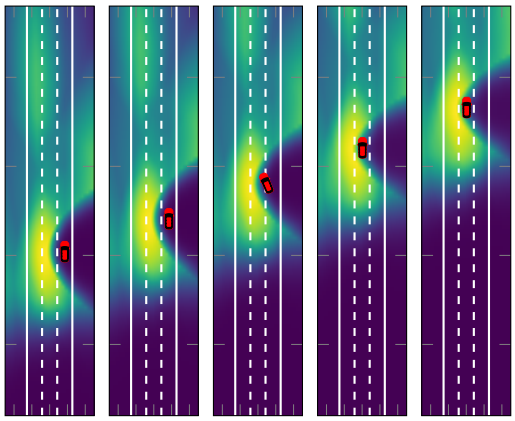

Fig. 7. Heat map of neural network feature value with augmented input space for various locations of the other vehicle (represented by the red car) for user 1. The speed is held constant at $v_r = 1$, and heading is held constant at $\theta_r = 90°$. Brighter colors indicate higher feature values.

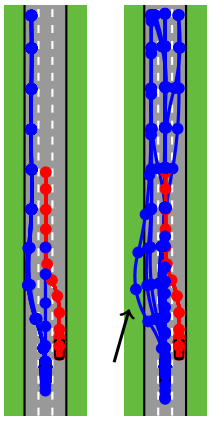

Fig. 9. Optimized trajectories for all users when reward function is encoded with the hand-coded feature set (left) and with the mixed feature set (right). Marks indicate equal time intervals. The arrow points to user 3 for whom the training was unsuccessful.

While the user still prefers to be in the region of the road to the left of the other vehicle, it is now evident that the user would prefer being in front of the other vehicle to being behind it. The collision avoidance and staying on the road characteristics are preserved with the augmented input space. Adding the extra input did not have the same effect for all users, and future work should investigate the effect of changing the input space of the neural network.

While similarities exist among the learned features, the features are unique to each particular user. Figure 8 shows the variation in the learned feature for each user with heading and speed. The neural network feature favors driving straight

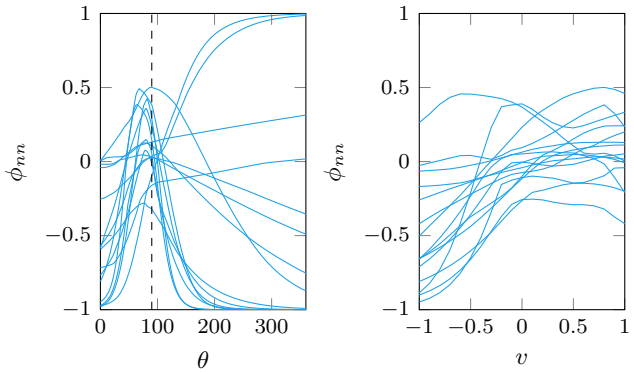

Fig. 8. Variation in neural network feature value and hand-coded heading feature value with heading for all users. The vertical dashed line shows a heading of 90°, which corresponds to driving straight down the road. The remaining neural network inputs are held constant at $x_r = 0.0$, $d = 0.5$, $\theta_r = 90°$ (right), and $v_r = 0.8$ (left).

down the road for most users, but it is less pronounced for some users and represents a trend that favors turning right, in which case the weight for the hand-coded heading feature is high enough to null out this trend. Similarly, most users have a negative feature value when driving backward, but the trend is more pronounced for some users.

C. Customized Optimal Trajectories

Using the hand-coded driving features in previous work resulted in users converging to similar reward functions. While these reward functions resulted in safe driving, their optimal trajectories were almost indistinguishable from each other [15]. Figure 9 shows the optimal trajectories for the users in our study selected using only the hand-coded feature set and using the mixed feature set. Optimal trajectories were generated by selecting the trajectory that had the highest reward out of 10,000 trajectories.

Consistent with the results of Sadigh *et al.* [15], the optimal trajectories using the hand-coded feature set are almost identical among all users, except for user 13 (our further inspection shows that this user has chosen a large number of backward trajectories, hence their optimal trajectory obtained by the hand-coded features starts by moving backward). In contrast, there is some variation among the optimal trajectories using the mixed feature set. Furthermore, with the exception of user 3 (in which the mixed feature set did not improve predictive accuracy over the hand-coded features), all users have safe trajectories that stay on the road and avoid collisions. These observations qualitatively support **H2**.

V. Conclusion

In this work, we developed a method for learning extra features for a linear reward function based on user responses to preference queries. We applied our methods to a driving scenario used in previous work and found improvements in predictive power. Upon interpreting the neural network features of the users, we found that the neural network is able to represent a complex feature that is fine-tuned to user preferences. While different features were learned for different users, most users converged on features with some notion of speed, heading, and collision avoidance.

By providing a method for feature learning, we have taken a step towards overcoming the challenges of feature design. By learning an extra feature directly from users' preferences, we ensure that our features are representative enough to adequately model user reward. This benefit is reflected in the improvement in prediction accuracies over hand-coded only features on the test set. Moreover, the neural network

feature is able to create a more expressive model that better distinguishes between the preferences of different users as evidenced by fig. 9. Finally, our method relieves some of the burden of feature design by allowing the hand-coded features to be augmented with an additional learned feature.

This research provides multiple avenues for future work. In this work, queries shown to the users were based entirely on the hand-coded features, and the feature learning was performed offline after the data was collected. Future studies will explore an online approach, in which feature learning is interleaved with active querying of the user. In terms of interpretability, further analysis of the learned features and optimal trajectories could help to better understand and distinguish driving styles among users. While this study focused on learning features for individual users, future work will focus on collating the data to learn a universal feature that improves the prediction accuracy for all users. Finally, we note that the neural network learned a complex feature from the data that combined a number of aspects of driving such as maintaining heading and avoiding other vehicles. In this work, we used a mixed set of features to preserve some interpretability in the final reward function; however, we note based on the results presented here that a feature function of only neural network features may perform well and should be a subject of future work. Even though such an approach might be too data-hungry to avoid overfitting, it could be further extended beyond feature learning to the learning of nonlinear reward functions.

Acknowledgments

The authors would like to acknowledge Dorsa Sadigh for her helpful input throughout the progression of this work. A. Maleki acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning.," in *International Conference on Machine Learning (ICML)*, 2000.

[2] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz, "A survey of preference-based reinforcement learning methods," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4945–4990, 2017.

[3] A. Wilson, A. Fern, and P. Tadepalli, "A Bayesian approach for policy learning from trajectory preference queries," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[4] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, 2017, pp. 4299–4307.

[5] R. Akrour, M. Schoenauer, and M. Sebag, "Preference-based policy learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2011, pp. 12–27.

[6] M. Cakmak, S. S. Srinivasa, M. K. Lee, J. Forlizzi, and S. Kiesler, "Human preferences for robot-human hand-over configurations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 1986–1993.

[7] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in Atari," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 8011–8023.

[8] D. S. Brown and S. Niekum, "Deep Bayesian reward learning from preferences," in *Workshop on Safety and Robustness in Decision Making, Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[9] N. Wilde, D. Kulić, and S. L. Smith, "Bayesian active learning for collaborative task specification using equivalence regions," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1691–1698, 2019.

[10] M. Tucker, E. Novoseller, C. Kann, Y. Sui, Y. Yue, J. W. Burdick, and A. D. Ames, "Preference-based learning for exoskeleton gait optimization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2351–2357.

[11] E. Biyik, N. Huynh, M. J. Kochenderfer, and D. Sadigh, "Active preference-based gaussian process regression for reward learning," in *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[12] E. Bıyık, M. Palan, N. C. Landolfi, D. P. Losey, and D. Sadigh, "Asking easy questions: A user-friendly approach to active reward learning," in *Conference on Robot Learning (CoRL)*, 2019.

[13] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh, "Learning reward functions by integrating human demonstrations and preferences," in *Robotics: Science and Systems (RSS)*, 2019.

[14] N. Wilde, D. Kulic, and S. L. Smith, "Active preference learning using maximum regret," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[15] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, "Active preference-based learning of reward functions," in *Robotics: Science and Systems (RSS)*, 2017.

[16] E. Bıyık and D. Sadigh, "Batch active preference-based learning of reward functions," in *Conference on Robot Learning (CoRL)*, 2018.

[17] S. M. Katz, A.-C. L. Bihan, and M. J. Kochenderfer, "Learning an urban air mobility encounter model from expert preferences," in *Digital Avionics Systems Conference (DASC)*, 2019.

[18] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

[19] J. R. Lepird, M. P. Owen, and M. J. Kochenderfer, "Bayesian preference elicitation for multiobjective engineering design optimization," *Journal of Aerospace Information Systems*, vol. 12, no. 10, pp. 634–645, 2015.

[20] R. Shah, D. Krasheninnikov, J. Alexander, P. Abbeel, and A. Dragan, "Preferences implicit in the state of the world," in *International Conference on Learning Representations*, 2019.

[21] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning robot objectives from physical human interaction," in *Conference on Robot Learning*, 2017, pp. 217–226.

[22] C. Basu, E. Bıyık, Z. He, M. Singhal, and D. Sadigh, "Active learning of reward dynamics from hierarchical queries," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[23] H. Haario, E. Saksman, and J. Tamminen, "An adaptive Metropolis algorithm," *Bernoulli*, vol. 7, no. 2, pp. 223–242, 2001.

[24] T. Dozat, "Incorporating Nesterov momentum into ADAM," in *International Conference on Learning Representations (ICLR) Workshop*, 2016.