Feudal Steering: Hierarchical Learning for Steering Angle Prediction

Faith Johnson
Rutgers University
faith.johnson@rutgers.edu

Kristin Dana
Rutgers University
kristin.dana@rutgers.edu

Abstract

We consider the challenge of automated steering angle prediction for self driving cars using egocentric road images. In this work, we explore the use of feudal networks, used in hierarchical reinforcement learning (HRL), to devise a vehicle agent to predict steering angles from first person, dash-cam images of the Udacity driving dataset. Our method, Feudal Steering, is inspired by recent work in HRL consisting of a manager network and a worker network that operate on different temporal scales and have different goals. The manager works at a temporal scale that is relatively coarse compared to the worker and has a higher level, task-oriented goal space. Using feudal learning to divide the task into manager and worker sub-networks provides more accurate and robust prediction. Temporal abstraction in driving allows more complex primitives than the steering angle at a single time instance. Composite actions comprise a subroutine or skill that can be re-used throughout the driving sequence. The associated subroutine id is the manager network's goal, so that the manager seeks to succeed at the high level task (e.g. a sharp right turn, a slight right turn, moving straight in traffic, or moving straight unencumbered by traffic). The steering angle at a particular time instance is the worker network output which is regulated by the manager's high level task. We demonstrate state-of-the art steering angle prediction results on the Udacity dataset.

1. Introduction

Reinforcement Learning (RL) has made major strides over the past decade, from learning to play Atari games [22] to mastering chess and Go [25]. However, RL tends to be unable to generalize policies enough to apply them to new environments and still struggles to solve problems with sparse reward signals. In response to this brittleness, Hierarchical Reinforcement Learning (HRL) is growing in popularity. In HRL, a manager network operates at a lower temporal resolution and produces goal vectors that it passes to the worker network. The worker network uses these goal

Figure 1. Four frames from the Udacity dataset are shown with their corresponding ground truth (blue) and predicted (orange) steering angles using our Feudal Steering network. The orientation of the lines corresponds to the egocentric steering angle. Our model predicts steering angles within 2.67 degrees of the ground truth angle.

vectors to guide its learning of a policy over micro-actions, also called primitive actions, in the environment at a higher temporal resolution than the manager network [6]. The temporal abstraction created through this relationship helps the networks to learn and execute macro-actions or tasks, also called *subroutines*, in the environment while lessening the negative effects of sparse rewards on network training.

Autonomous driving is an application that struggles with this issue of sparse reward signals. However, most HRL work emphasizes video game and other simulated domains instead of autonomous driving applications. At all times, human drivers are paying attention to two levels of their environment. The first level goal is on a finer grain: don't hit obstacles in the immediate vicinity of the vehicle. The second level goal is on a coarser grain: plan actions a few steps ahead to maintain the proper course efficiently. It is even possible to conceive of higher levels of abstraction comprised of path planning and other more complicated driving tasks.

Autonomous vehicles need to have tight constraints on hardware and software in order to be effective in real world applications [18]. Current successful HRL networks are large and take a long time to train [26, 30], making them unsuitable to implement in autonomous vehicles despite the theoretical benefits. Additionally, many HRL methods that do focus on the driving domain require handcrafted subroutines and do not focus on primitive navigation directly, choosing to find policies over macro-actions instead. Handcrafting subroutines limits environment exploration and requires a high level of domain specific knowledge in order to yield good model performance.

We propose a vehicle agent to predict steering angles using feudal networks. Feudal networks are typically applied in hierarchical reinforcement learning. However, in this work, we train these networks with ground-truth data from the Udacity dataset [29], instead of with rewards, allowing us to retain the advantageous hierarchical structure of HRL without using reinforcement learning. We present two methods. The first method predicts steering angles with subroutines (driving tasks) obtained from the t-SNE embedding of the driving data. We also use t-SNE to refine and structure the subroutine embedding space discovered by the manager in order to visualize the driving data subroutines and observe their semantic meaning. The second method allows the manager to discover the existing subroutines in the data instead of handcrafting them. Our results show that feudal networks with learned subroutines provide improved training stability and prediction performance.

2. Related Work

2.1. Temporal Abstraction

In hierarchical reinforcement learning, the manager network operates at a lower temporal resolution than the worker network and communicates with the worker network through a goal vector that encapsulates a temporally extended action (called a subroutine, skill, option, or macro-action). The worker executes atomic actions in the environment based on this goal vector and its own state information. This process of manager/worker communication through temporal abstraction helps to break down a problem into more tractable pieces as outlined by the options framework [27].

To explain the concept of temporal abstraction further, consider the case of an agent attempting to leave a room through a door. When a human plans this action, they don't compose a low level sequence of movements such as *straight, straight, left, straight, right*. In other words, humans do not consciously think of each atomic action required to exit the room. Instead, they think in terms of temporal abstraction: *Find the door. Approach it. Pass through it.* Each of these actions encapsulates multiple atomic actions that need to be executed in a specific order for the agent to complete the higher level task.

2.2. Hierarchical Reinforcement Learning

One difficulty with reinforcement learning is delayed rewards and sparse credit assignment. This problem is especially prevalent with RL in autonomous vehicles, as an agent may only receive a reward when it completes a larger sub-task. Hierarchical reinforcement learning is used to increase model performance through temporal abstraction and intrinsic rewards [15], but has limited implementations in the autonomous driving domain as prior work opts for simulated environments. Feudal networks [30] learns to play the Atari game, Montezuma's Revenge. Their hierarchical network has a manager that learns a latent space for its goals, which take on a directional meaning and allow the manager to be updated regardless of the worker's actions in the environment. However, this method requires a lot of data and time to train, which is not necessarily available or possible in the driving domain.

Because of the complexity of the driving domain, there is a trend of manually defining subroutines for HRL networks [4, 7, 17]. Our method diverges from this practice by allowing the manager network to learn its own subroutines. There are other frameworks [26, 28, 1] that also attempt to learn subroutines implicitly from the data. Kumar et al. [16] propose a method to learn subroutines through imitation learning and propose using HRL to refine them. Another approach explores the nature of the subroutines themselves by focusing on learning the states of the subgoals instead of learning the policy between these states [24]. This hierarchical approach is taken a step further by [9, 23] which use the states in the latent space of the lower layer as the action space for the layer above it.

2.3. Steering Angle Prediction

Most of the work in steering angle prediction uses some form of alternative representation of the driving scene beyond RGB images, from attention maps [14, 11] to segmentation and optical flow [12, 21, 13]. While these representations contain valuable information, we aim for a method that predicts steering angles using only raw visual input, as humans do. Additionally, in the case of segmentation and optical flow, these alternative scene representations add latency to the prediction pipeline which is undesirable for real world applications. CNN-based methods such as [19] use features directly from the RGB image input and use multiple fully connected layers to predict steering angle, speed, and acceleration, thereby allowing them to create a fully functional, end to end, autonomous vehicle model.

In order to create an autonomous driving system that is robust to real world driving scenarios, it is desirable that real world data is used to train and test the networks as
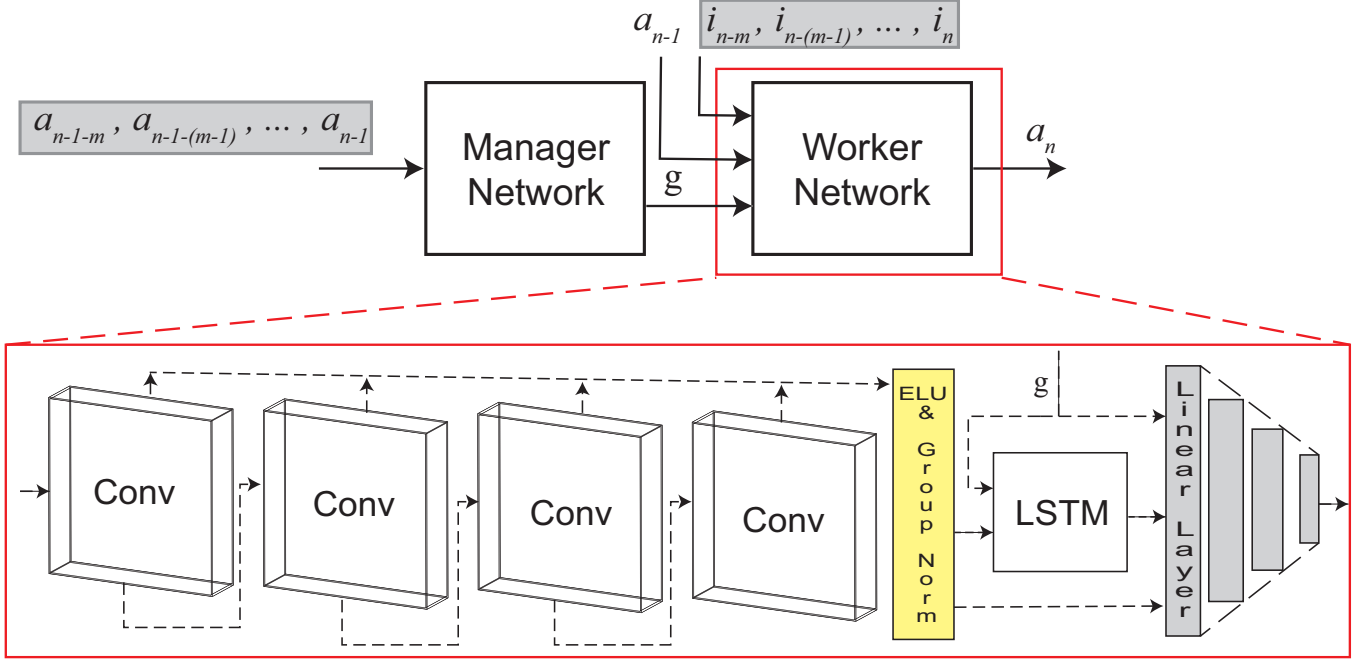
Figure 2. Feudal Steering Network. The overall network is comprised of a manager network and a worker network. The worker network (expanded in the red box) acts as the steering angle prediction network. The input to the manager network is a sequence of the previous m predicted steering angles, $[a_{n-1-m}, a_{n-1-(m-1)}, ..., a_{n-1}]$. The input to the worker network is a sequence of m frames, $[i_{n-m}, i_{n-(m-1)}, ..., i_n]$, a goal, g, obtained from the manager network, and the previous steering angle, a_{n-1}. The yellow box represents the ELU (exponential linear unit) and group normalization step in the pipeline.

in [2], that deploys their implementation in a vehicle. The most comparable steering angle prediction methods to Feudal Steering are [5, 31], which use a sequence of RGB images to predict steering angles using recurrent units. However, our approach demonstrates the effectiveness of feudal learning for steering angle prediction by estimating subroutines (macro-action states) across the driving data.

3. Methods

3.1. Steering Angle Prediction Network

Our approach to predicting steering angles is inspired by [8] from the Udacity steering angle challenge. During training, this network inputs images to a CNN to extract the relevant features, then passes these features through two, jointly trained recurrent units. The first recurrent cell uses the feature vector combined with the ground truth steering angles from the previous batch as input. The second recurrent cell uses the feature vector combined with the predicted steering angles from the previous batch as input. The weighted sum of the loss from both cells is used to update the network. During testing, only the recurrent cell with trained with the previous predicted angles is used.

We take a more simplified approach to Feudal Steering, as shown in Figure 2. Our network uses a 3D convolutional layer with a ReLU activation function followed by

a dropout layer. The output of this convolution is saved to use later on in the network. This process is repeated four times before the output is fed through a series of fully connected layers with ReLU activation functions. At this point, the output and the intermediary representations from each of the convolutions are added together, passed through an ELU (exponential linear unit) layer, and normalized. Then, the previous predicted steering angle and the output of the ELU layer are passed through an LSTM. Finally, the output of the LSTM is passed through a fully connected layer with the output from the ELU layer to produce the steering angle.

Compared to the Udacity network [8], we also use a set of 3D convolutional layers with ReLU, dropout regularization, and skip connections to glean relevant features from the images. However, we only train one LSTM with the concatenated feature vectors and previously predicted steering angle as input. Using the previous ground truth steering angle is feasible in the problem domain with the addition of extra sensors to the vehicle. However, our goal is to create a self-contained network that predicts steering angles based solely on image input, so we choose to use the previous predicted angle as input instead. Additionally, for a fully trained model, the difference between the previous ground truth and predicted angles will be negligible, so our performance at test time will not be greatly effected.
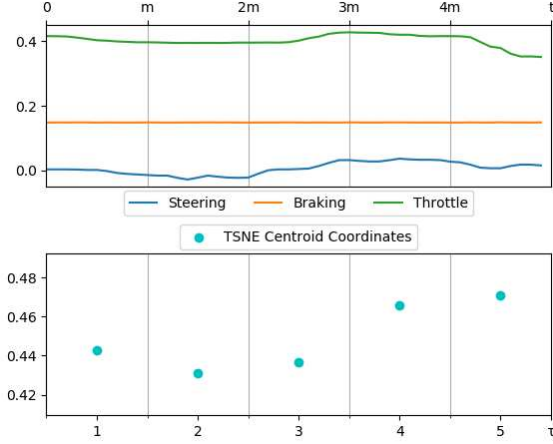
Figure 3. Steering, braking, and throttle data are concatenated every m time steps to make a vector of length 3m. Each vector is projected to 2D t-SNE coordinates that act like a manager for the steering angle prediction and operate at a lower temporal scale. In our experiments m=10; t and τ are the temporal axes for the driving data and t-SNE coordinates respectively.
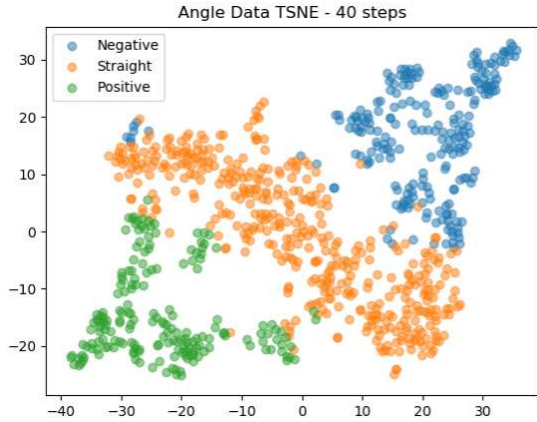

Figure 4. Total plot of the t-SNE coordinates for the Udacity data. The colors correspond to the average sign of the angles in each length 3m vector used to generate the points. The horizontal and vertical axes correspond to the two t-SNE dimensions.
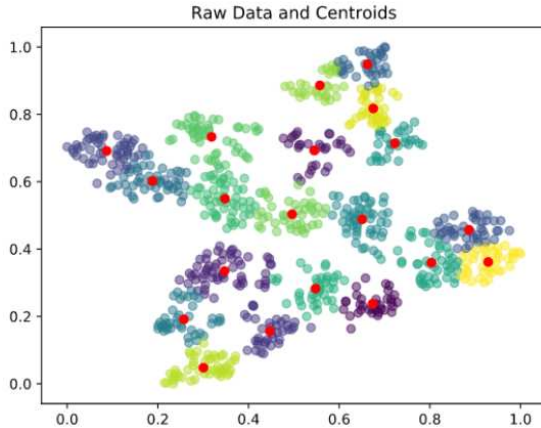

Figure 5. K-Means clustering (k=20) of the TSNE coordinates of the Udacity data with the centroids pictured in red. Not only do distinct clusters form in the data, but each cluster corresponds to a unique action of the vehicle.

3.2. Subroutine ID

For a hierarchical framework, we aim to classify the steering angles into their temporally abstracted subroutines, also called options or macro-actions, associated with highway driving such as "follow the sharp right bend", "bumper-to-bumper traffic", "bear left slightly". This could be done by hand, but it would be a lengthy process, and the created subroutines would most likely be too simplistic to describe the wide variety of driving scenarios a vehicle may encounter. For driving, the high level tasks are numerous and it is preferable to compute or learn *subroutine ids* rather than manually label semantic tasks. We demonstrate that our automatically extracted subroutine ids have observable semantic meaning in terms of driving tasks (see Figure 6).

3.3. t-SNE Embedding as Subroutine ID

We explore using t-SNE [20] as an embedding space for our driving data and as the subroutine ids themselves. To do this, we arranged the steering angle, braking, and throttle pressure data into vectors of length m. Then, the vectors from each category that correspond to the same time steps are concatenated together to make vectors of length $3m$. During training, the collection of these vectors is passed through the unsupervised t-SNE algorithm to create a coordinate space for the driving data. For our networks, we use $m = 10$, however this is a hyperparameter that can be tuned.

Each vector of length $3m$ is given one x and y coordinate pair as illustrated in Figure 3. The greater collection of all of the generated points is shown in Figure 4. The coloring of the points in this figure is hard coded. The points corresponding to vectors with primarily negative steering angles are in blue. The points corresponding to vectors with positive steering angles are in green. The orange points correspond to vectors with steering angles that are relatively close to zero.

Once we have the t-SNE embedding of the data, we use K-Means clustering on the coordinates and take the centroids of the clusters as our new subroutine ids, as shown in Figure 5. We vary k from ten to twenty to determine if different numbers of clusters improve prediction performance. Then, we train our manager network to predict subroutines similar to the t-SNE centroids given a sequence of images as input. In order to ensure that no data pertaining to the predicted steering angle is used as input to this network, we use the t-SNE centroid corresponding to the previous m steering, braking, and throttle data as input to the network.

To illustrate, refer back to Figure 3. If we are predicting an angle from the range $t \in [2m, 3m]$, then the t-SNE centroid used for the subroutine id input to the angle prediction network will be the centroid at $\tau = 2$, which was made with the steering, braking, and throttle data from $t \in [m, 2m]$. In this way, the angle we are attempting to predict will not be
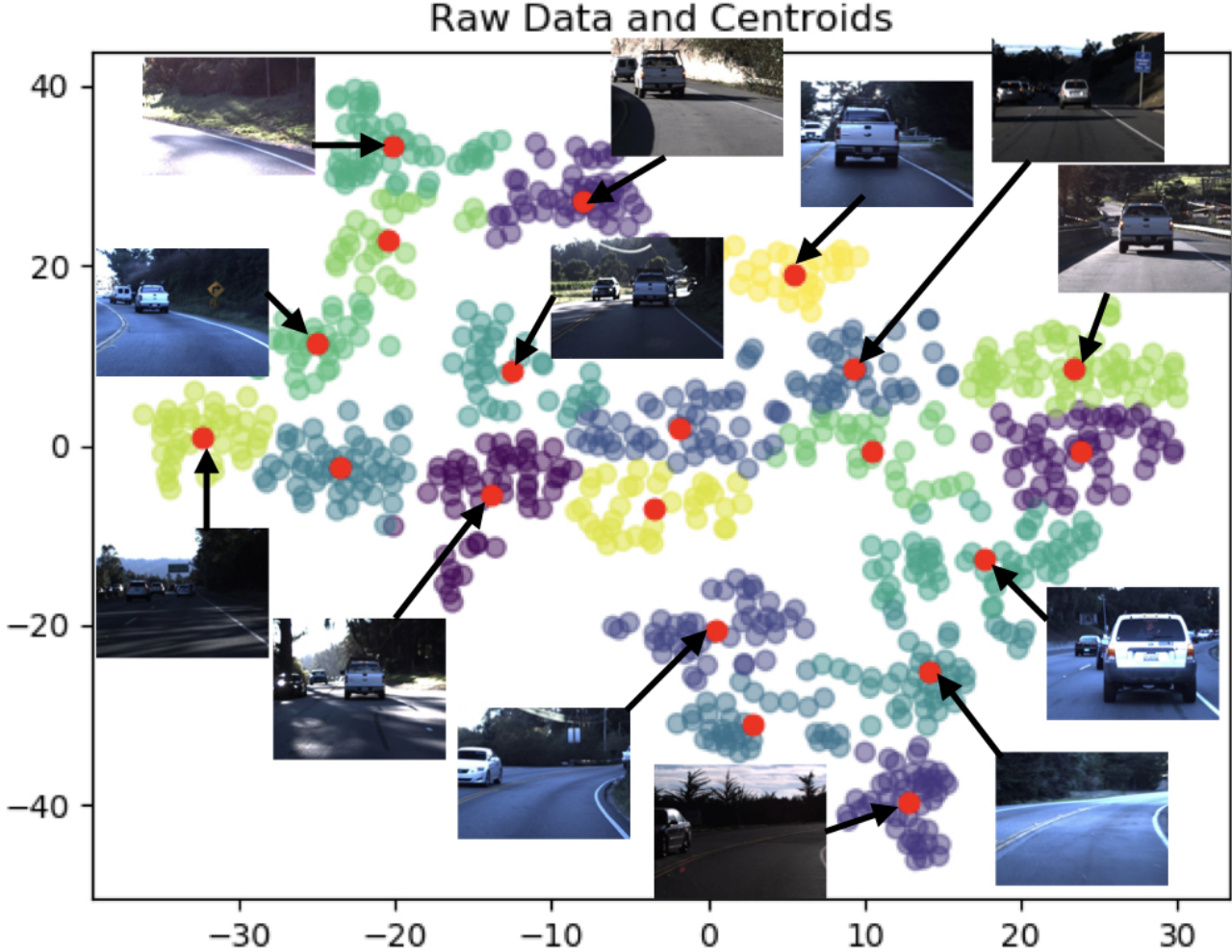
Figure 6. Example training images are shown with their corresponding t-SNE centroids. Notice the bottom right of the figure contains sharp right turns. Moving upwards and to the left, the right turn gets less sharp until the vehicle begins to go straight. Eventually this straight behavior starts to become a left turn until the vehicle is making sharp left turns in the upper left hand corner.

used to compute the t-SNE centroid that is input to the network as the subroutine id. This shift also incorporates an extra level of temporal abstraction into our network.

Figure 6 shows example training images that correspond to some of the t-SNE centroids. Notice that the bottom right of the figure contains sharp right turns. Moving diagonally upwards, the right turns get less sharp until the vehicle begins to go straight. Then, this straight motion gradually begins to become a left turn until, by the top left of the figure, the vehicle is making sharp left turns. Figure 7 shows that the points contained in each cluster exhibit the same, or comparable, behavior. The left column of images are a subset of the t-SNE centroid frames from Figure 6. Each row contains frames from points adjacent to the associated centroid that are contained within the same cluster. The behavior in each row is consistent, showing that the points in each cluster behave similarly.

3.4. t-SNE Prediction Network

Since our results (Section 4) show that t-SNE coordinates prove useful as a subroutine ID, we also explore prediction of t-SNE coordinates directly from images, as a t-SNE network following a concept introduced in prior work [32]. The t-SNE prediction network is jointly trained with and our steering angle prediction network. For this t-SNE manager network, we fine tune the FBResNet152 model [10, 3]. We train the steering angle prediction network to take in the predicted centroids as the subroutine id, as well as a sequence of images, in order to predict the next steering angle.

3.5. Subroutine ID Prediction Network

While t-SNE provides convenient visualization of the subroutine id semantic meaning, we take inspiration from [16] to allow the manager learn the subroutines over the

Figure 7. The left column of images are a subset of centroid frames from Figure 6. The images to the right of each centroid frame come from different, adjacent points in the corresponding cluster for each centroid. Notice that the points in each cluster display similar behavior as their respective centroids.

driving data. This work trains multiple networks on completely unlabeled data in order to label frames based on an agent's actions during an initial exploration of an environment. The subroutines across these labeled frames are then learned and represented as discrete random variables. However, the Udacity dataset [29] already provides low-level action labels between consecutive frames in the form of steering angles. So we only need to create a network to learn subroutines across these actions.

In summary, we obtain subroutine ids using three methods: 1) Set the subroutine id to the ground truth t-SNE cluster centroids where t-SNE is computed on steering, throttle, and braking data m time steps prior to the prediction time n. 2) Set the subroutine id to the t-SNE network output following the general concept introduced in [32] by predicting t-SNE coordinates from images. 3) Learn subroutine ids jointly with steering angle prediction with a subroutine id network. The best results are obtained by the third method.

4. Experiments and Results

4.1. Dataset and Augmentation

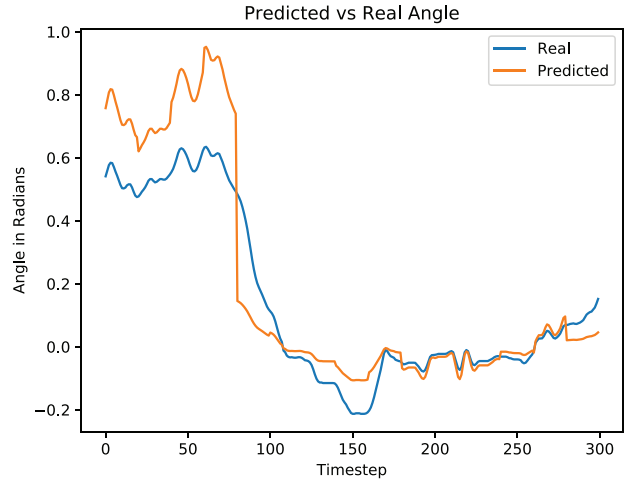We test our feudal networks in the domain of autonomous vehicles using the Udacity driving dataset[29],

Figure 8. Results of steering angle prediction when the ground truth t-SNE coordinates of the input data are used as the subroutine ids in the steering angle prediction network of Feudal Steering. Notice that, for these results, we use a network that does not take the previous angle as input.

which provides steering angles, first-person dash cam images, braking, and throttle pressure data. We use frames from the CH2_002 partition of the dataset and use a 75%/25% train/test split. We augment our training data to increase its size and influence model training by implementing a horizontal flip, which effectively doubles the size of the dataset. For this change, we negate the angles associated with the flipped images. Additionally, all images are scaled and normalized so that their pixel values lie in the range $[-1, 1]$.

4.2. t-SNE as Subroutine ID

First, we use t-SNE as the embedding space for our subroutine ids by embedding the data into 2D space, using K-means clustering to create centroids, and using the coordinate pairs of those clusters as the subroutine ids. However, before we attempt to predict the t-SNE coordinates from the image data, we determine if the t-SNE coordinates will function as subroutine ids. We use the ground truth value of the t-SNE centroids as the subroutine id in our angle prediction network, along with an image sequence of length ten, to determine whether or not it would be worthwhile to attempt to predict the centroids.

The results of this experiment are in Figure 8. The blue lines are the real steering angle, and the orange lines are the predicted angle. While the results in this figure show that the predicted angles diverge slightly from the ground truth angles, these predictions are more relevant to real world applications because they are computed using only visual input. Additionally, the quality of these predictions is high
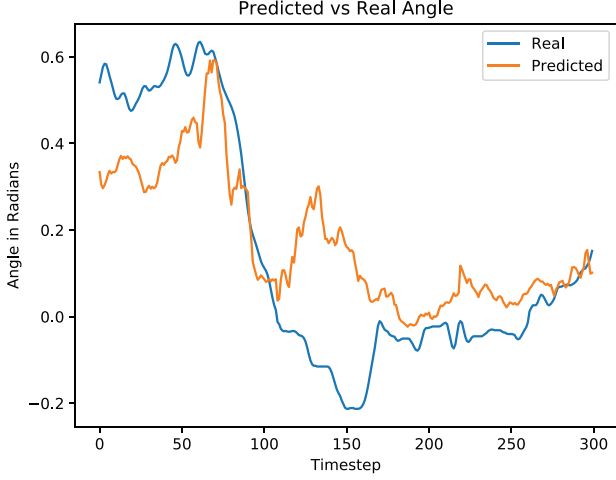
Figure 9. The steering angle prediction results using the predicted t-SNE network as the manager are pictured above in orange. The blue represents the ground truth angles. While these results as worse than our subroutine id netowrk, they were achieved without using the previous steering angle as input to the network.

Number of Centroids	10	15	20
RMSE	0.2093	0.2240	0.2251

Table 1. The manager network predicts subroutine ids to be close to the t-SNE centroids of the training data embedding space. We test 10, 15, and 20 centroids in our network and find that 10 centroids has the lowest RMSE.

enough to motivate us to use additional methods of predicting the subroutine id's with the manager network.

4.3. Predicted t-SNE as Subroutine ID

Next, we jointly train a t-SNE prediction and steering angle prediction networks. The input to both is an image sequence of length ten. The t-SNE prediction network outputs the coordinates to the corresponding t-SNE centroid of the image input. To train this network, we minimize the MSE loss between the output and the ground truth t-SNE coordinates. The steering angle prediction network takes in this predicted centroid and produces the corresponding steering angle. We also minimize the MSE loss between the predicted and real angles. We conducted this experiment using 10, 15, and 20 t-SNE centroids and found that 10 centroids produced the best results, as shown in Table 1. Figure 9 shows the prediction results. The blue line represents the ground truth angles, and the orange line is the predicted angles.

4.4. Subroutine ID Network

To create this subroutine id network, we mimic the structure of the steering angle network. However, the input to the subroutine id network is a one dimensional sequence

of steering angles, so the network uses 1D convolutions instead of 3D. Additionally, we only use three sets of convolutions for this network instead of four.

We jointly train the subroutine id and steering angle prediction networks. The subroutine id network takes a sequence of historical steering angles as input and outputs a goal vector representing the subroutine id for those angles. The steering angle network takes in the subroutine id, a sequence of images, and the previous predicted angle and outputs the next steering angle in the sequence.

During training and testing, the sequence of angles fed into the subroutine id network consists of $[a_{n-1-m}, a_{n-1-(m-1)}, ..., a_{n-1}]$, in order to ensure that we only use the sequence of angles preceding the angle we aim to predict. The subroutine id is a single number that is able to take on any value in $\mathbb{R}$. The sequence of images input to the steering angle network range from $[i_{n-m}, i_{n-(m-1)}, ..., i_n]$, and the previous angle used as input is a_{n-1}. We choose $m = 10$ for our experiments, but this is a hyperparameter that can be fine tuned.

We use a learning rate of 1×10^{-4} with an Adam optimizer. The other hyperparameters for the optimizer are unchanged from their pytorch defaults of $\beta = (.9, .999)$. We train our model under multiple loss functions and compare the performance. These loss functions are MSE,

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=0}^{N} (\alpha_i - a_i)^2$$

RMSE,

$$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{N} \sum_{i=0}^{N} (\alpha_i - a_i)^2}$$

and MAE

$$\mathcal{L}_{MAE} = \frac{1}{N} \sum_{i=0}^{N} |\alpha_i - a_i|$$

where N is the number of predictions, α is the ground truth angle, and a is the predicted angle. We find that we achieve the best results using MSE loss, but we report our MAE loss for comparison purposes in the results section as well.

Our final experiment is predicting steering angles and subroutines based on visual input using this subroutine id network. We create an image sequence of ten frames that we feed into our feudal network along with the previous steering angle to predict the next steering angle. Figure 10 shows the prediction results. The top graph shows the steering angle predictions. The corresponding subset of real steering angles from the Udacity [29] dataset are in blue, and the predicted steering angles are in orange. The bottom graph in Figure 10 shows the predicted subroutine ids. We can see from these predictions that the learned subroutine ids follow the general pattern of the steering angles, but vary in
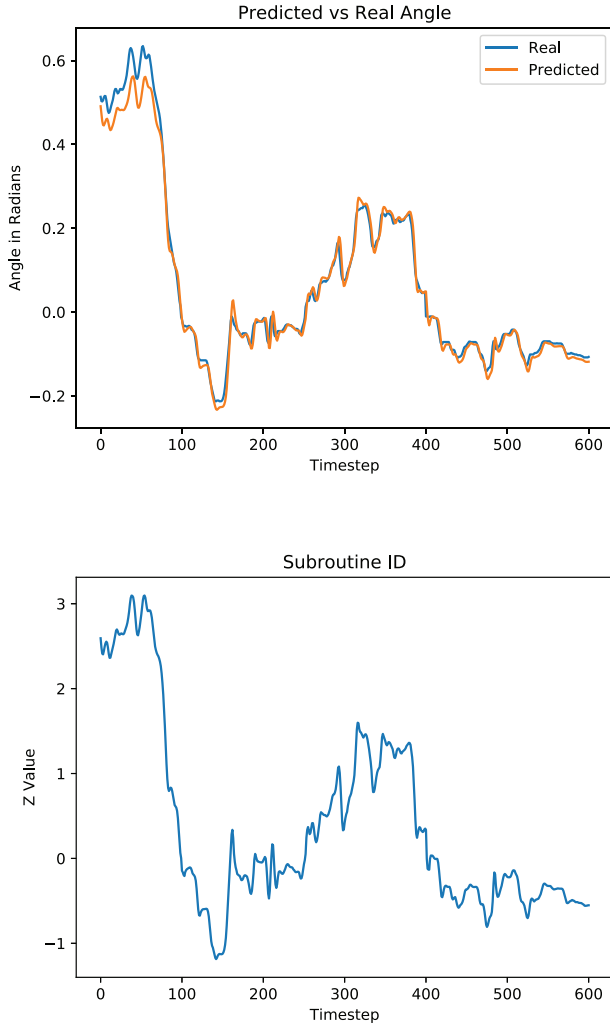
Figure 10. Angle (top) and subroutine id (bottom) prediction results on the Udacity dataset obtained using our Feudal Steering network are shown above. The real angle is pictured in blue and the predicted angle is in orange. The subroutine ids are plotted alone. Notice that the subroutine id's behavior mimics the real angle behavior, but differs in scale.

scale, showing that the subroutine id is a stepping stone to the final steering angle prediction.

We compare this method with several state of the art (SOTA) implementations in Table 2. We show that our RMSE and MAE are lower than [14, 21, 5, 8]. While we did not achieve better loss values than [12], we achieved comparable MSE and MAE values using a much smaller, simpler network. This is beneficial in the autonomous driving domain where memory and latency are limited for efficient, real world applications.

	RMSE	MAE
Interpretable Attention [14]	-	0.07191
Event Based Camera[21]	0.07156	-
Deep Steering [5]	0.0609	-
Udacity Challenge[8]	0.0483	-
Feudal Steering (Ours)	0.04659	0.01902
Learning by Mimicking [12]	0.04110	0.02834

Table 2. The RMSE and MAE of Feudal Steering is compared with other steering angle prediction methods. We outperform all but one of the SOTA methods. We produce comparable RMSE and MAE to [12] despite using a much smaller model.

4.5. Non-Hierarchical Steering Angle Prediction

We attempted to use the steering angle prediction network without a manager network to compare hierarchical and non-hierarchical networks. However, the non-hierarchical network (worker network only) failed to predict any reasonably accurate steering angles.

5. Discussion

In this work, we show that the feudal networks from hierarchical reinforcement learning are more effective than reinforcement learning at the task of steering angle prediction. This effect is due to temporal abstraction. Breaking down the problem into more tractable pieces narrows the focus of the worker agent and allows the optimal policy to be found more quickly. Additionally, temporal abstraction also helps alleviate the problems of long term credit assignment and sparse reward signals. The lower temporal resolution of the manager shortens the period of time between rewards overall.

We also explore a t-SNE embedding space as the goal space for the manager in our steering angle predictions. We use the centroid corresponding to steering angle, braking, and throttle data from the previous m time steps as the subroutine id in our angle prediction network and were able to predict future steering angles without the direct use of the steering angle from the previous time step. However, this network had worse performance than our subroutine id network because of the limitations on the subroutine representation. When we allow the manager network the freedom to be able to define its subroutines for itself, performance increases and surpasses the current SOTA.

Acknowledgements

References

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 3

[3] Remi Cadene. Pretrained models for pytorch. *URL https://github. com/Cadene/pretrained-models. pytorch*, 2018. 5

[4] Jianyu Chen, Zining Wang, and Masayoshi Tomizuka. Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1239–1244. IEEE, 2018. 2

[5] Lu Chi and Yadong Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *arXiv preprint arXiv:1708.03798*, 2017. 3, 8

[6] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993. 1

[7] Jingliang Duan, Shengbo Li, Yang Guan, Qi Sun, and Cheng Bo. Hierarchical reinforcement learning for self-driving decision-making without reliance on labeled driving data. *IET Intelligent Transport Systems*, 2020. 2

[8] Ilya Edrenkin. Komanda team solution, udacity challenge 1st place winner. https://github.com/udacity/self-driving-car/blob/master/steering-models/community-models/komanda/solution-komanda.ipynb, 2016. 3, 8

[9] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[11] Sen He, Dmitry Kangin, Yang Mi, and Nicolas Pugeault. Aggregated sparse attention for steering angle prediction. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2398–2403. IEEE, 2018. 2

[12] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning to steer by mimicking features from heterogeneous auxiliary networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8433–8440, 2019. 2, 8

[13] Qadeer Khan, Torsten Schön, and Patrick Wenzel. Latent space reinforcement learning for steering angle prediction. *arXiv preprint arXiv:1902.03765*, 2019. 2

[14] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, pages 2942–2950, 2017. 2, 8

[15] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016. 2

[16] Ashish Kumar, Saurabh Gupta, and Jitendra Malik. Learning navigation subroutines by watching videos. *arXiv preprint arXiv:1905.12612*, 2019. 2, 5

[17] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 584–599, 2018. 2

[18] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–766, 2018. 2

[19] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019. 2

[20] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 4

[21] Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018. 2, 8

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1

[23] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018. 2

[24] Jacob Rafati and David C Noelle. Learning representations in model-free hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 10009–10010, 2019. 2

[25] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 1

[26] Yuhang Song, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, and Mai Xu. Diversity-driven extensible hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4992–4999, 2019. 2

[27] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. 2

[28] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[29] Udacity. Udacity self-driving car driving data 10/3/2016 (dataset-2-2.bag.tar.gz). 2, 6, 7

[30] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017. 2

[31] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017. 3

[32] Jia Xue, Hang Zhang, and Kristin Dana. Deep texture manifold for ground terrain recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2018. 5, 6