Meta-Learning DAgger

Bryan Chan *, **Zihang Fu** *, **Zeqi Li** *
Department of Computer Science
University of Toronto
Canada
chanb,fuzihang,lizeqi@cs.toronto.edu

Abstract: In recent years, Imitation Learning has achieved many impressive results in robotics and game AI. behavioral cloning (BC) is a fundamental process to enable and realize many of these successes. However, BC often leads to poor performance in practice as the accumulation of compound error in test time. As a resolution to this issue, DAgger was proposed to drastically alleviate this problem. DAgger iteratively corrects the policy by modifying the policy weights with an aggregated dataset generated by the expert. Therefore, we hypothesize the existence of a correlation between BC policy and after DAgger (DA) policy. We attempt to use a deep neural network (DNN) to find this correlation and apply it to achieve reduction of querying the expert and Meta-Learning across similar tasks. We demonstrate that our proposed mapping network achieves a convincing mapping performance in terms of success rate and average cumulative reward in the CarRacing environment and prove to be a more effective initializer to enable Few-Shot Learning in FetchPickAndPlace environment.

Keywords: CORL, Robots, Learning, Imitation, Meta

1 Introduction

In the field of Imitation Learning, BC is widely used to teach an agent to learn from expert demonstration and it achieves phenomenal success in various applications [1, 2, 3]. However, BC is known to suffer from covariate shift problem as the accumulation of cascading error [4]. DAgger was then proposed to solve this problem via an interactive supervision, by aggregating training data from expert in the test run [5]. Querying expert is often very time-consuming and scenario-specific. To reduce and eliminate the need of querying experts during evaluation, in addition to generalize across various similar tasks, we propose to combine Meta-Learning with DAgger (MetaL DAgger).

Prior methods such as MMD-IL [6], SHIV [7], SafeDAgger [8], and Dropout DAgger [9] reduce the amount of expert queries during evaluation by using distances and uncertainties of the model. On the other hand, Finn et al. [10] proposes to apply Meta-Learning to Imitation Learning and learn the parameters based on the expert's actions and the agent predictions.

Our approach, however, tries to learn a mapping between BC policy and DA policy to unveil their correlation. This mapping network can be interpreted in two aspects: a mapping from a BC policy to DA policy to fix the covariate shift problem where we refer it to MetaL DAgger Mapper (MDM), a good initialization for BC training to realize a faster and more robust learning via Few-Shot Learning where we refer it to MetaL DAgger Initializer (MDI).

We conduct extensive evaluation for our models on two environments CarRacing and FetchPickAndPlace, and the models exhibit both characteristics as we mentioned previously. In the CarRacing environment, the best performing model achieves a 89% success rate after applying MDM on the failed BC policy. All mapping mechanisms we evaluated remarkably improve the cumulative reward from BC policy on most unseen tasks. In the FetchPickAndPlace environment, the model treated as an task-dependent initializer achieves a 53% and 74% success rate for BC applying MDI and one

*The three authors contributed equally. Order was determined based on alphabetic ordering of last names.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

DAgger iteration applying MDI respectively. This result outperforms the DA training with either random initialization or BC initialization by a significant margin.

2 Related Work

There is a substantial growth in demand for artificial agents to mimic the behaviour of human experts. One way to achieve this is through learning from demonstrations provided by experts. To address covariate shift in BC, many methods have been proposed for more robust imitation, such as DART [11] and DAgger [5].

However, the above methods require excessive amount of expert demonstrations in order for the agent to learn properly. New methods are proposed to improve sample efficiency by querying expert with high uncertainty or distance [6, 7, 8, 9]. An alternative to address this problem is by Few-Shot Learning. Few-Shot Learning has been studied extensively for supervised learning tasks [12, 13]. It is a paradigm that learns using small number of training examples. In the Imitation Learning community, Few-Shot Imitation Learning often involves Meta-Learning. However, most contributions focus on the Reinforcement Learning settings [14, 15, 16].

The term Meta-Learning first appeared in the field of educational psychology [17]. In Machine Learning, Meta-Learning is a sub-field that focuses on the metadata of different learning tasks. A sub-stream of Meta-Learning aims to avoid over-fitting to a single task and generalize to similar tasks. A popular approach is to train the meta-learner on a distribution of tasks with the hope of generalizing to the entire task distribution [18]. Recently, researchers have been focusing on using DNN to improve the performance of meta-learners [19, 13].

The idea of Meta-Imitation Learning has received very limited exploration due to its novelty and interdisciplinary nature. One such exploration of the idea is proposed by Finn et al. [10]. In their approach, they combine MAML [12] and learning from demonstration [20] to achieve One-Shot Imitation Learning. In this work, we propose to combine Meta-Learning with DAgger, which could possibly generalize to similar but different tasks and reduce the need of querying experts.

3 Methods

3.1 Problem Formulation

Our problem formulation is based on the hypothesis that a correlation between BC policy and after DA policy exists. To validate it and potentially identify this correlation, we define this problem as a supervised learning task by a input/output pair (π_{BC}, π_{DA}), in which π_{BC} is the weight of BC policy and π_{DA} is the weight of after DA policy. A universal function approximator such as neural network f_θ is then learned to map π_{BC} to $\hat{\pi}_{DA}$. This mapping process is expected to achieve $\hat{\pi}_{DA} \approx \pi_{DA}$, which corresponds to the correlation we proposed.

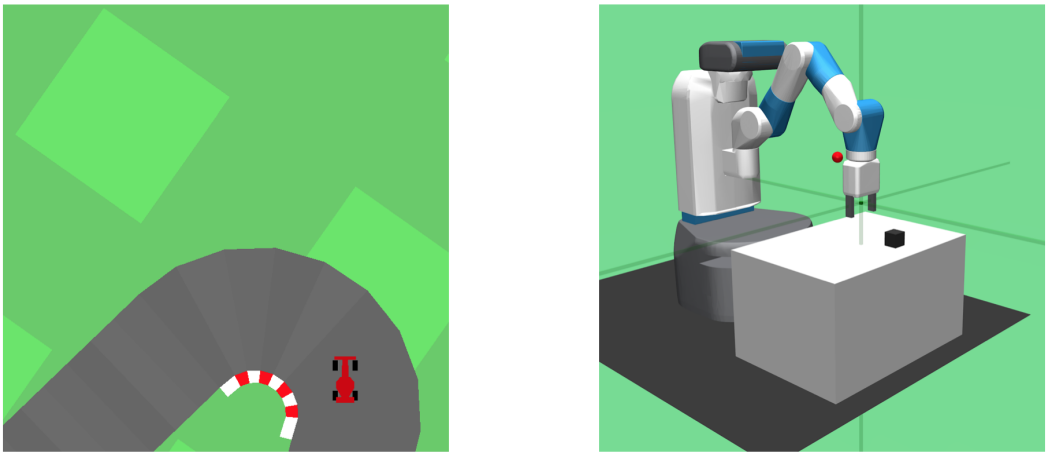

Figure 1: Evaluations are done in CarRacing (left) and MuJoCo FetchPickAndPlace (right) environments.

To test our hypothesis, we pick the environment Box2D CarRacing and MuJoCo Robotics FetchPickAndPlace because they are widely experimented in the field of Imitation Learning and the experts required to perform DAgger are easily accessible. CarRacing[2] is a control task to drive a car on a track based on the pixel input. The reward is the distance traveled by the car within the generated track. FetchPickAndPlace[3] is a continuous control task to navigate a robotic arm to pick up an object and then place it to certain target location in a 3D space. It provides a sparse reward indicating success/fail at the end of the episode.

3.2 Challenges

As a supervised learning problem, acquiring label, in our case π_{DA}, is essential. However, DAgger in general is a time-consuming process because it needs to query the expert during evaluations. With increasing of DAgger iterations, more training data is aggregated so the training time will inevitably increase.

The other challenge is the curse of dimensionality encountered for directly mapping the entire policy. The minimal policies we used for both experiments contain around 150K parameters among all layers. For a single one-layer fully-connected neural network, the parameter size will explode (e.g. 150K $\times$ 150K $\approx$ 22.5B parameters). Although we can bottleneck the network in the earlier layers, it is detrimental to the expressive power of the network, which is critical to finding the hypothetical correlation. Also, training such a large network to map the entire policy will necessarily require a significant amount of training data. Due to the computation limitation and constraint of the project time-frame, massive data acquisition is unrealistic to achieve.

A specific challenge to the task of FetchPickAndPlace is the intrinsic stochasticity in the environment. As the environment is in a continuous action space, the covariate shift problem is more prone to appear because the robotic arm is very sensitive to small perturbation of the command and the unpredictable interaction with the target object. These factors cause DAgger to fail in most scenarios within reasonable training time. Therefore, we need to enhance DAgger to be applicable to this environment.

3.3 Approaches

To address the parameter size challenge from section 3.2, we attempt three approaches: direct mapping with restricted layers, single-autoencoder mapping (SAM) and twin-autoencoder mapping (TAM). All approaches are MDMs and training would require a notion of "ground truth", which is described in section 3.4.1. The "ground truth" is an approximation of the optimal weights of the policy such that the agent can complete the specified task in most of the scenarios. We use the "ground truth" to bound the amount of trainable weights within the policy so that learning is more manageable. In the following subsections, we use weights in place of policy weights.

3.3.1 Directed Mapping with Restricted Layers

We reduce the number of trainable weights by fixing all weights except the last few consecutive layers with the "ground truth" weights. Formally, the weights π can be represented as n layers of weights, $\pi_{(1:n)}$ (See Figure 2). Similarly, we represent the "ground truth" weights with $\pi^*_{(1:n)}$. We can choose the number of layers to fix, $1 \leq k \leq n$, such that $\pi_{(1:k)} = \pi^*_{(1:k)}$. During backpropagation, only $\pi_{(k+1:n)}$ will receive gradient flow (i.e. only $\pi_{(k+1:n)}$ is trainable). Since only $\pi_{(k+1:n)}$ are trainable, we only need to learn a direct mapping (DM) between $\pi_{BC(k+1:n)}$ and $\hat{\pi}_{DA(k+1:n)}$. In our experiments, we fix all layers but the last layer (i.e. the n'th layer is the only trainable layer). It's similar to a fine-tuning process.

3.3.2 Autoencoder (AE)

The idea is to further reduce the dimensionality of inputs and outputs by learning a latent representation and a latent space mapping. The latent space mapping can be trained faster under the more compact latent representation. However, since the underlying distributions of π_{BC} and π_{DA} may

[2]CarRacing Environment: https://gym.openai.com/envs/CarRacing-v0/

[3]FetchPickAndPlace Environment: https://gym.openai.com/envs/FetchPickAndPlace-v0/

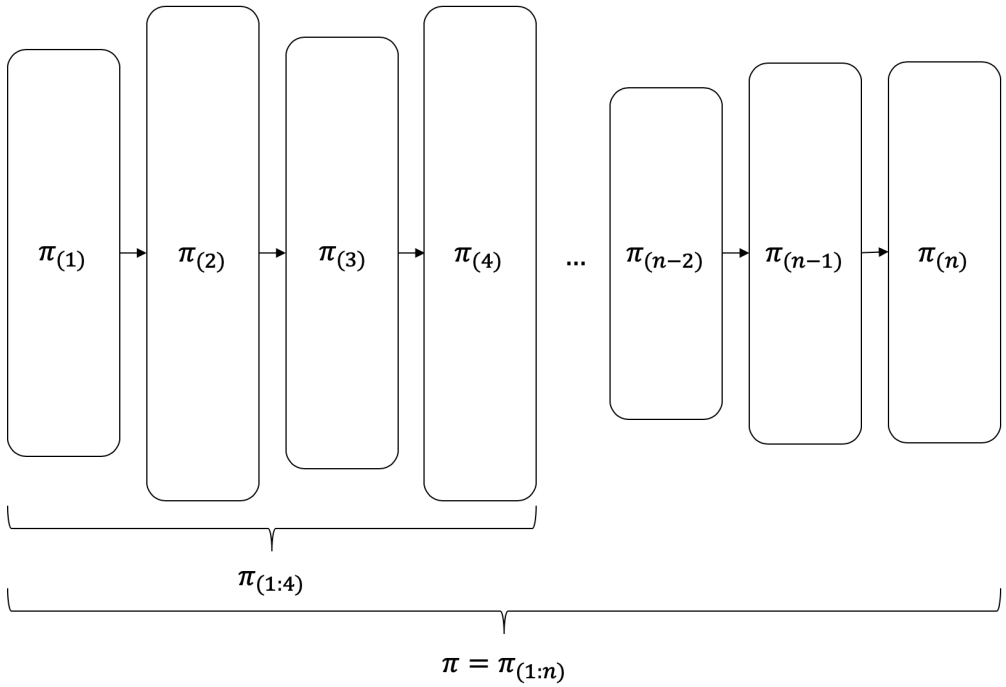

Figure 2: Suppose there are n layers in the neural network. We represent each layer of the weights π above, where $\pi_{(i)}$ is the weights of layer i. We specify subsets of contiguous layers from layer j to layer k with $\pi_{(j:k)}$, where $j \leq k$.

differ, we train two separate autoencoders: one for each set of weights. This approach is called twin-autoencoder mapping (TAM).

Formally, after fixing the layers of the policy, we construct two autoencoders $\sigma_{BC}(\pi_{BC}) = D_{BC}(E_{BC}(\pi_{BC}))$ and $\sigma_{DA} = D_{DA}(E_{DA}(\pi_{DA}))$, where D corresponds to the decoder and E corresponds to the encoder. We can train both σ_{BC}, σ_{DA} with training datasets $\{\pi_{BC}^{(i)}\}_{i=1}^{N}$, $\{\pi_{DA}^{(i)}\}_{i=1}^{N}$, respectively. To learn the latent mapping h, we formulate a regression problem and use $E_{BC}(\pi_{BC})$, $E_{DA}(\pi_{DA})$ as input/output pairs. During inference, the complete mapping process, f, is simply $f(\pi_{BC}) = D_{DA}(h(E_{BC}(\pi_{BC}))) = \hat{\pi}_{DA} \approx \pi_{DA}$ (See Figure 3).

Alternatively, we construct a single-autoencoder σ_{ALL} that finds latent representation for both π_{BC} and π_{DA}. In this case, we modify the complete process f such that E_{BC}, D_{DA} are from σ_{ALL} instead of σ_{BC}, σ_{DA}, respectively. σ_{ALL} is trained with dataset $\{\pi_{BC}^{(i)}\}_{i=1}^{N} \cup \{\pi_{DA}^{(i)}\}_{i=1}^{N}$. This alternative is used in single-autoencoder mapping (SAM).

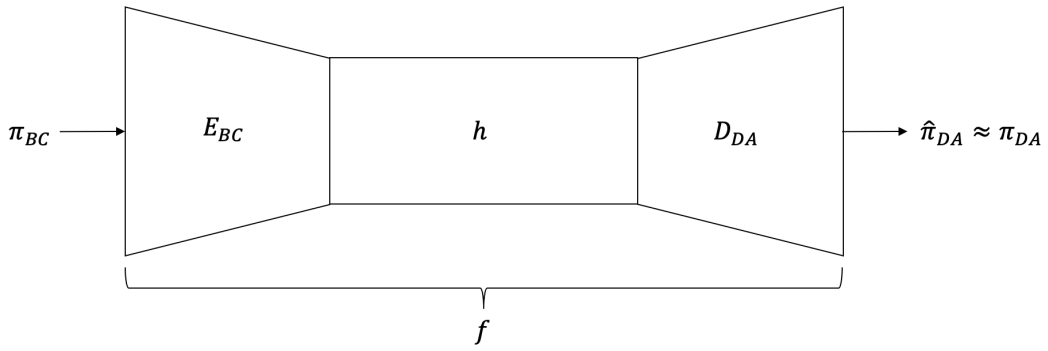

Figure 3: This is a high level representation of twin (single) autoencoder f. $E_{BC}(E_{ALL})$ is the encoder of $\sigma_{BC}(\sigma_{ALL})$, h is the latent mapping that learns θ_{BC} to $\hat{\theta}_{DA}$, and $D_{DA}(D_{ALL})$ is the decoder of $\sigma_{DA}(\sigma_{ALL})$. The input of f is π_{BC} and the output is $\hat{\pi}_{DA}$, an approximation of π_{DA}.

3.4 Data Collection

3.4.1 Generating the Ground Truth Model

As specified in section 3.3, only the last layers are trainable and all other layers are fixed with the "ground truth" weights π^*. We use DAgger with multiple seeds for both environments to generate π^*. Firstly, a number of different seeds are generated. For each seed, an instance of the environment is generated using this seed and a demonstration is produced by the expert. Here, a demonstration consists a series of state image and expert's action pairs. All demonstrations are then aggregated into one training dataset to train a single behavioral cloning policy. Afterwards, we perform multiple DAgger iterations – for each iteration, the policy is run on all generated seeds again, aggregating the new state image and action pairs generated by the expert to the training set. The policy is then trained with the new aggregated dataset. For both environments, five DAgger iterations are executed to generate π^*.

3.4.2 Generating the Training Data

For policy π, we follow section 3.3.1 and replace and fix $\pi_{(1:k)}$ with $\pi^*_{(1:k)}$ and reinitialize $\pi_{(k+1:n)}$. Then, for each seed, we train π with BC and DAgger, and collect π_{BC} and π_{DA}, a training input/output pair. As mentioned in section 3.3.2, we also use this training dataset for training the autoencoders.

For the CarRacing environment, the expert is a PID controller, and three DAgger iterations are performed for each seed. During training, π outputs the steering directions and the expert provides the gas and brake signal. After all input/output pairs are collected for all seeds, a filtering process is performed such that we only keep pairs in our training and validation set which fail the task with BC policy and complete the task with DA policy.

For the FetchPickAndPlace environment, the expert is a neural network that has access to the vectorized state representation of the environment. Similar to CarRacing environment, π outputs the x, y, and z displacement of the robotic arm and the expert controls the gripping action. Also, an early-stopping mechanism is implemented such that the task is killed and determined failure if the accumulated cosine distance between actions from π and the expert exceeds a certain threshold. This guarantees that the training set will not be overwhelmed with highly improbable state images. Without this early-stopping mechanism, learning will become much more complicated and slower.

3.5 Model Architectures

As both environments use images as input states, convolutional neural networks are used for extracting perception features for both environments. The convolutional layers are then followed by a few fully-connected layers. For the mapping networks, a fully-connected neural network with regularization layers such as BatchNorm and Dropout is used. Please see appendix 7 for more details.

4 Evaluations & Results

We design the experiments to answer the following questions:

1. Can the mapping achieve good performance and how effective is it?
2. If the mapping fails, can we use it as a good initializer for the policy to enable Few-Shot Learning?

For the first question, we evaluate the mapping for the policy's last layer on the environment CarRacing and FetchPickAndPlace. We compare the performance of the model on three approaches mentioned in 3.3: direct mapping with restricted layers (DM), twin-autoencoder mapping (TAM) and single-autoencoder mapping (SAM). The effectiveness is measured in terms of success rate and overall improvement compared before mapping and after mapping on unseen seeds.

For the second question, we treat the mapping as a initialization (MDI) to the policy if one-time mapping of the behavioral cloning policy fails to complete the task. We demonstrate that our approach can significantly speedup the learning process by running only one pass of behaviour cloning. In

this case, our process is similar to Few-Shot Learning with a minor difference: traditional one-shot learning would only train the agent once with the BC. However, since our MDI is trained on π_{BC}, we need to first train the agent with BC and apply MDI, then train the agent again with BC.

4.1 CarRacing

The original CarRacing environment from Box2D is a continuous control task to pilot a car in a 2D generated track with boundary and grass on the side. However, in our project, it is modified to a discrete control task by quantizing the steering angle of the car to 20 classes. The implementation of training the driving policy via behavioral cloning and DAgger is majorly migrated from the Imitation Learning course at University of Toronto[4] of the course. To better illustrate the effectiveness of our mapping mechanism, we tweaked the capacity of the driving policy and the training process to make the behavioral cloning policy π_{BC} more prone to failure and the DAgger process can eventually fix the deviation issue.

As we suggested in section 3, we compare the three variants of the mapping mechanism in three aspects: timestep, reward and success rate. Both TAM and SAM are mappings within latent representations. The complete process for both SAM and TAM also involves encoding the π_{BC} and decoding the latent representation $h(E(\pi_{BC}))$. There is a small difference between SAM and TAM while encoding and decoding phase. The input of the TAM is encoded by E_{BC} and the output of it is decoded by D_{DA}. Compared to TAM, both the encoding and decoding for the input and output respectively are done by the same autoencoder.

As we can see from the reward and timestep comparisons (See Figure 4), all three variants improve the performance in a significant amount in both metrics since most points lie above the identity line ($y = x$). As opposed to our hypothesis, DM and TAM have a similar performance and SAM outperforms both variants (See Tables 2 and 1). Looking specifically at Table 1, SAM with an adjusted success rate 89% outperforms the other two variants by a large margin. One possible explanation is that the process of encoding to a latent space extracts the principle components of the policy and this ends up to be an extremely helpful step to learn more robust weight mapping later on. As the size of our training dataset is very small, DM cannot mitigate the effect from non-principle components, which is an arguably more complicated task. The extra performance boost from SAM to TAM might benefit from larger training dataset (two times larger than TAM), which is essential to both better encode the policy and decode the latent representation.

For those failed tasks after mapping from evaluation dataset, we also evaluate the performance of our model regarding it as a task-dependent initialization. Performance is measured after one pass of behaviour cloning applying mapped policy as the initialization. As we can see from the Tables 4, 3, and 5, our models used as the initializer can help the model quickly adapt to the working policy for a few scenarios. However, this is not very convincing compared to the one-time mapping. We suspect it is due to the overfitting situation happened to our trained mapping networks and autoencoders. The mapped policy might be stuck in some bad local minimum, which makes it much harder to unlearn from this bad policy. If we have more data to train the mapping network and apply more effective regularization, this issue should be alleviated.

	DM	SAM	TAM
Raw	53.33%	86.67%	53.33%
Adjusted	57.14%	89.29%	57.14%

Table 1: The success (completing the task) rates over 20 unseen seeds under CarRacing environment. The raw success rate includes seeds (in total of 3 seeds) where behavioral cloning works. The modified success rate excludes those seeds.

[4]Modified CarRacing Environment: https://github.com/florianshkurti/csc2621w19

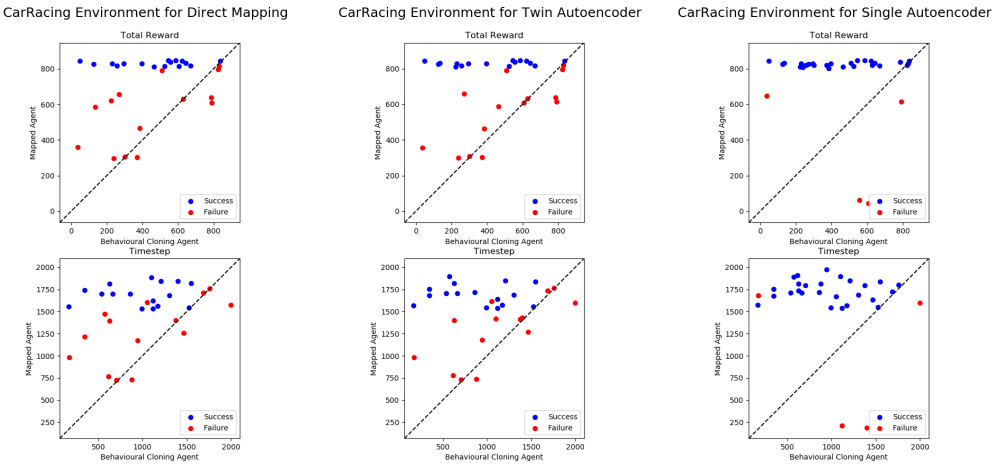

Figure 4: The performance of DM (left), TAM (middle), and SAM (right) are compared under OpenAI Gym CarRacing environment across multiple unseen seeds. The top row represents the total reward by the end of the trajectory and the bottom row represents the total timesteps taken for the same trajectory. If the datapoint is above the diagonal line, the performance of the method is better than behavioral cloning.

	BC	DM	SAM	TAM
Average Total Reward	458.09	705.56	761.45	705.58
Average Timestep	996.37	1495.23	1628.70	1505.10

Table 2: The average total reward and average timesteps over the 20 unseen seeds using BC, DM, SAM, and TAM.

	BC		SAM		One-Shot SAM		
Seed	Reward	Timesteps	Reward	Timesteps	Reward	Timesteps	Success
1033	557.34	1117	60.68	210	837.30*	1626	Yes
1056	790.96*	1999	614.40	1596	617.10	1569	No
1327	36.03	176	646.53*	1683	640.87	1650	No
5275	606.03*	1398	43.09	187	604.37	1382	No

Table 3: The performance of one-shot learning using SAM under CarRacing environment. All seeds above failed to complete the task with SAM.

	BC		DM		One-Shot DM		
Seed	Reward	Timesteps	Reward	Timesteps	Reward	Timesteps	Success
1053	509.15	1053	790.83*	1618	511.89	1062	No
1068	301.61	703	306.04*	727	303.63	717	No
1153	222.85	577	620.34	1473	812.30*	1876	Yes
1327	36.03	176	358.62	980	646.04*	1658	No
5003	385.18	946	467.44*	1176	465.94	1160	No
5120	628.50	1375	629.49	1399	788.02*	1712	No
5125	270.43	628	657.09	1393	659.00*	1374	No
5159	371.26	881	301.60	733	819.40*	1805	Yes
5252	785.94*	1468	638.93	1259	308.03	650	No
5278	239.58	616	298.20	767	811.90*	1880	Yes
5381	135.29	349	584.82*	1216	569.41	1189	No

Table 4: The performance of one-shot learning using DM under CarRacing environment. All seeds above failed to complete the task with DM.

	BC		TAM		One-Shot TAM		
Seed	Reward	Timesteps	Reward	Timesteps	Reward	Timesteps	Success
1053	509.15	1053	790.83*	1618	508.85	1056	No
1068	301.61	703	308.75*	734	301.61	703	No
1327	36.03	176	355.13	985	794.03*	1999	Yes
5003	385.18	946	463.84	1181	807.20*	1927	Yes
5120	628.50	1375	631.38*	1414	630.89	1385	No
5125	270.43	628	659.80*	1401	658.89	1375	No
5149	466.05	1099	589.29*	1420	464.95	1110	No
5159	371.26*	881	301.40	735	294.06	707	No
5252	785.94*	1468	637.73	1271	635.69	1254	No
5275	606.03	1398	609.04*	1433	602.01	1373	No
5278	239.58	616	300.11	780	812.30*	1876	Yes

Table 5: The performance of one-shot learning using TAM under CarRacing environment. All seeds above failed to complete the task with TAM.

4.2 FetchPickAndPlace

FetchPickAndPlace is a continuous state and action space robotic simulation environment in MuJoCo. The goal of this task is to control a robotic gripper to move a square object to a specific location. The original state space of this task is a vector representation of the gripper position, object position and goal position. To make this task harder, we use images generated by the simulator, and transformed these images to 64×64 PIL images as state input. The action space is a four-dimension vector, where the first three dimensions denote the **x**, **y**, and **z** displacements of the gripper, respectively, and the fourth dimension denote how wide the gripper is open. In our case, the agent learns how to correctly control the spatial displacement (i.e. the first three-dimension of the action, and the fourth dimension of the action is always controlled by the expert). Furthermore, the goal position is confined to always have the same height as the object, and the maximum number of steps per task is set to 50.

As mentioned in section 3.4, we first generate a π^* with 60 randomly generated seeds. We then re-initialize the weights in the last layer in the policy network $\pi_{(1:n-1)}$ and fix all other weights with $\pi^*_{(1:n-1)}$, and re-train the weights in the last layer $\pi_{(n)}$ for each seed using DAgger. Eventually, 25 seeds are successfully solved by using DAgger, and thus 25 pairs of π_{BC} and π_{DA} are generated.

In this environment, we treat the mapping network as MDI. During evaluation, for each unseen seed, we first generate the expert demonstration and run BC. Then, we feed $\pi_{BC(n)}$ to the mapping network to generate the initialization weights $\pi_{init(n)}$. The current trainable weight of the agent $\pi_{BC(n)}$ is replaced with $\pi_{init(n)}$ and the agent is trained again on the demonstration data. The reason we re-train the agent on demonstration data again is that we observe through experiments that the mapping process "removes" some task-specific characteristics of the original π_{BC}, thus the agent needs to "see" the demonstration again. The resulting weights, π_{one_shot}, is tested on the current seed. In case it fails, we perform one DAgger iteration to generate $\pi_{one\_shot+DA}$ and test again. We record the percentage success rate of using both π_{one_shot} and $\pi_{one\_shot+DA}$.

We compare the percentage success rate of using DAgger with MDI (DAgger-MDI) to vanilla DAgger with random initialization. To illustrate the superiority of DAgger-MDI, we also compare it to DAgger with BC initialization (DAgger-BC). Table 6 illustrates in detail the process of the above mentioned three methods.

Step	DAgger-MDI	DAgger-BC	DAgger
1	Generate demonstration data		
2	Train BC weights π_{bc}		
3	Map π_{bc} to π_{mapped}	None	
4	Save π_{mapped}	Save π_{bc}	None
5	Train on Demo w/ init using saved π		None
6	Test on current seed (one-shot test)		
7	If succeeded, terminate		
8	Else, run DAgger w/ init from step 4		Else, run DAgger w/ Xavier init
9	Test on current seed (two-shot test)		

Table 6: The evaluation process of the DAgger with MetaL DAgger initialization, DAgger with behavioral cloning initialization, and DAgger.

Method	One-shot	Two-shot
DAgger-MDI	53%	74%
DAgger-BC	17%	34%
DAgger	27%	44%

Table 7: The performance of one-shot and two-shot learning using DAgger-MDI v.s. DAgger-BC v.s. DAgger under FetchPickAndPlace environment. All seeds were unseen in the training process of MetaL DAgger as well as the generation process of the "ground-truth" policy. Here, the "One-shot" column records the percentage of testing seeds that are solved at step 6 in table 6; the "two-shot" column records the percentage of testing seeds that are solved up until step 8 (including the seeds that are solved in step 6) in table 6.

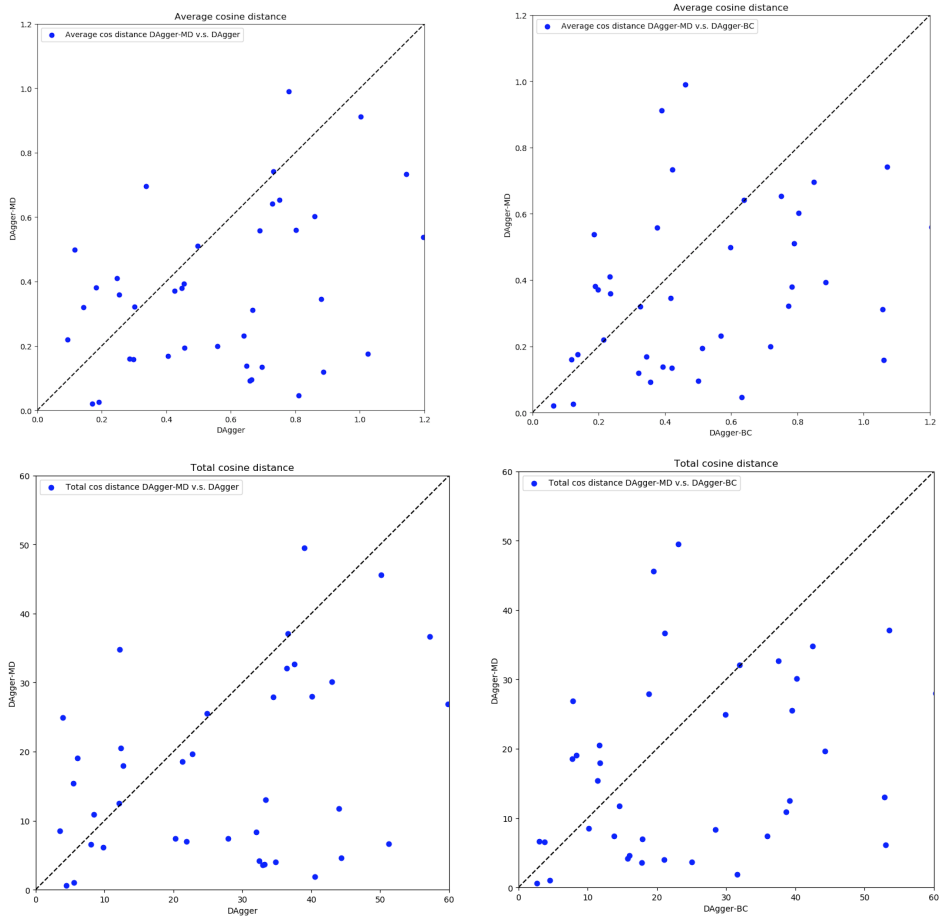

Figure 5: The cosine distance between actions performed by the learner agents and those labelled by the expert. Each dot represents a specific task. The figure shows the comparison among DAgger-MDI, DAgger-BC and DAgger in terms of total cosine distance accumulated in each task as well as the average cosine distance per action. For every plot, the majority of dots are under the diagonal line, meaning that for most tasks DAgger-MDI produces policies with smaller cosine distance from the expert.

5 Limitations

For the current experiments, the biggest constraint we set is to only map for the last layer of the network. This is good for proof-of-concept and scale down both the model and training data required. However, it is desired to achieve the direct mapping or good initialization for the entire policy.

There are two approaches we can take to relax this constraint in the future. The first approach is to use autoencoder for each layer to learn a compressed latent representation. As the success of our experiment, we are convinced that the latent representation with significantly lower dimension does not significantly degrade the mapping performance. This way the entire policy can be largely compressed and a capacity-sufficient model can be then built and learned within latent space.

The second approach is to construct a mapping network for each layer. This can be realized in either an autoregressive model [21] or a bi-directional sequence model [22]. For layer i's mapping network, the mapping network would take $\{\pi_{BC}^{(i-1)}, \pi_{BC}^{(i)}, \pi_{BC}^{(i+1)}\}$, where $\pi_{BC}^{(i)}$ is current layer i's weight. The output would be $\hat{\pi}_{DA}^{(i)}$. The reason why we need to pass in the two adjacent layers' weight is to provide the context, which is necessary to determine the neurons' permutation state. Analogous to autoregressive model, task of layer mapping or transformation can be framed as a sequence generation task. Each layer is regarded as a single timestep in the input sequence, the model will output corresponding $\hat{\pi}_{DA}^{(i)}$ as the sequence goes. Since we need both forward and backward context (previous layer and next layer), the bi-directional RNN model is a natural match for the purpose.

Another aspect of the limitation is the lack of extensive hyperparameters' tuning and model regularization. We find our model is a bit overfitting to the training dataset as the training loss is considerably lower than the validation loss (See Figure 8). We believe a better set of hyperparameters exists and more effective regularization techniques can be applied.

6 Conclusion

In this work, we explore Meta-Learning in the context of Imitation Learning by uncovering the potential correlation between a BC policy π_{BC} and its corresponding DA policy π_{DA} using DNN. We then evaluate our models on CarRacing and FetchPickAndPlace environments.

In CarRacing, the best performing mapping network SAM is able to directly map a non-functional BC policy to a fully-functional policy with a 89% adjusted success rate. In FetchPickAndPlace, due to the lack of training data and the complexity of this task, the mapping network is insufficient to make a non-functional BC policy work in a single-pass. However, the network serves as an MDI is capable of drastically speedup the learning process and induce a more robust policy compared to the other two commonly used alternative initialization schemes in DAgger training.

7 Appendix

7.1 Model Architectures

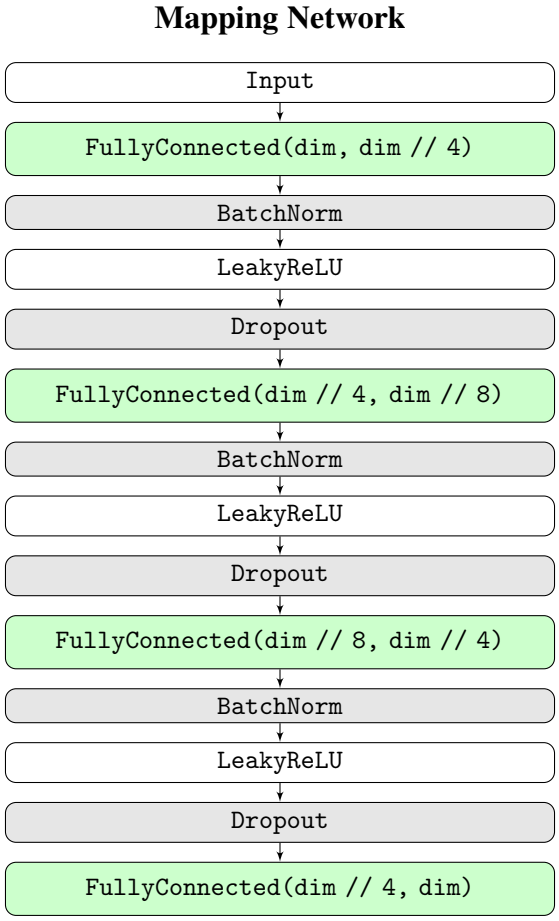

Figure 6: Architecture of mapping network. It is composed of fully connected layers `FullyConnected(input_size, output_size)` with LeakyReLU non-linearity. `dim` denotes the number of weights. Some grey blocks (`BatchNorm` and `Dropout`) are tuned/removed for specific models.

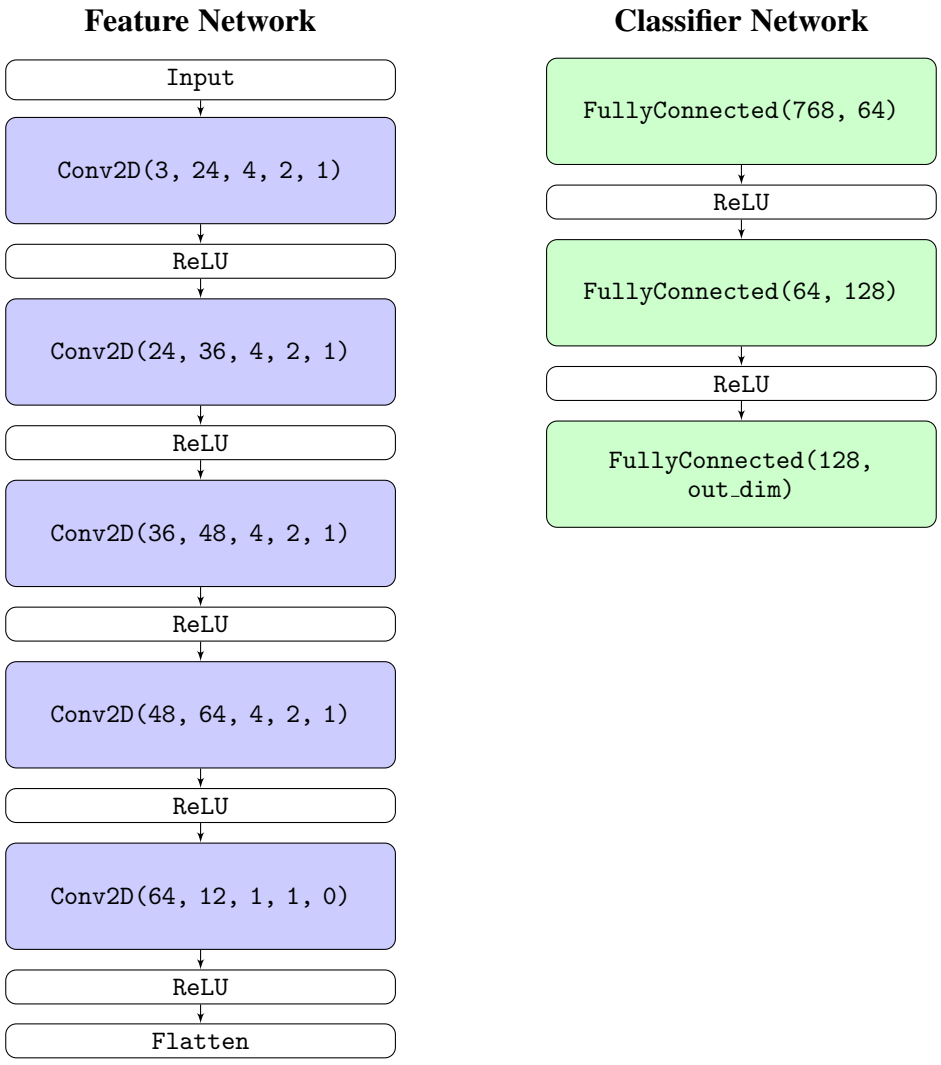

Figure 7: The architecture for ag et policynis shared across both environments. It consists a feature extractor followed by a classifier. Feature extractor uses 2D convolution layers (`Conv2D(input_channel_size, output_channel_size, kernel_width, stride, padding)` with ReLU non-linearity, followed by a flatten layer at the end. Classifier uses fully connected layers `FullyConnected(input_size, output_size)` interleaving with ReLU non-linearity. `out_dim` denotes the output dimensionality for the specific environment.

7.2 Learning Curves

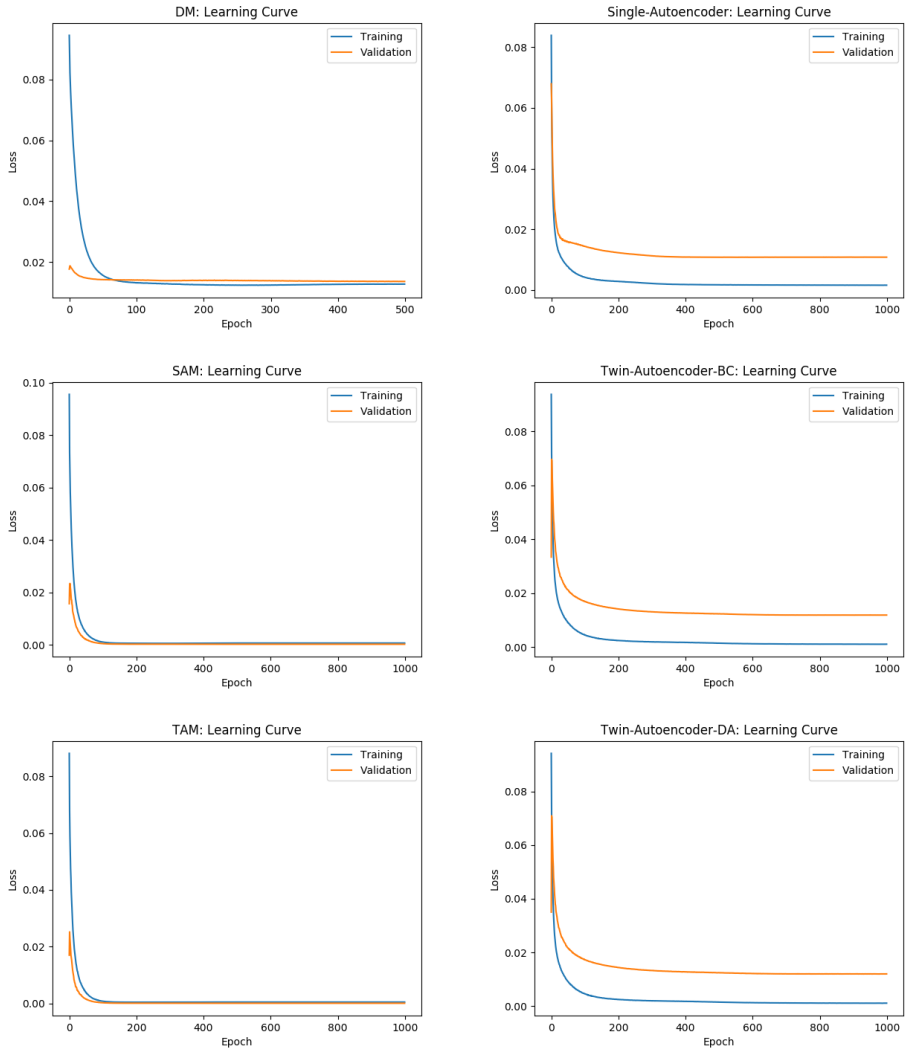

Figure 8: The learning curves of all models for CarRacing environment. The left column (DM (top), SAM (middle), and TAM (bottom)) are the training curves for all mapping networks. The right column (σ_{ALL} (top), σ_{BC} (middle), and σ_{DA} (bottom)) are the autoencoders trained for SAM and TAM approaches.

7.3 Hyperparameters

	DM	SAM	TAM	σ_{ALL}	σ_{BC}	σ_{DA}
Learning Rate	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-5}	10^{-5}	10^{-5}
Batch Size	10	10	10	10	10	10
Training Epochs	500	1000	1000	1000	1000	1000
Training set size	300	300	300	600	300	75
Validation set size	75	75	75	150	300	75

Table 8: Hyperparameters we used for each network in CarRacing environment. All networks use Adam optimizer (default unless specified) [23]. The weights are initialized with Xavier Initialization [24], and the biases are initialized with zeroes.

	DAgger-MDI
Learning Rate	10^{-4}
Weight Decay	10^{-4}
Batch Size	10
Training Epochs	2000
Training set size	18
Validation set size	5

Table 9: Hyperparameters we used for the mapping network for FetchPickAndPlace environment. Adam optimizer (default unless specified) [23] is used. The weights are initialized with Xavier Initialization [24], and the biases are initialized with zeroes.

7.4 Members' Contribution

7.4.1 Bryan Chan

1. Write scripts to automate data preprocessing.
2. Construct autoenocoder for CarRacing environment.
3. Tune hyperparameters for models.
4. Collect data for CarRacing environment.
5. Generate tables and figures for the report.

7.4.2 Zeqi Li

1. Write scripts to automate data collection for CarRacing.
2. Construct all mapping networks and autoencoders for CarRacing environment involving DM, SAM and TAM.
3. Tune hyperparameters and regularize for all mapping networks in CarRacing environment.
4. Construct expert policy for FetchPickAndPlace.

7.4.3 Zihang Fu

1. Construct expert policy for FetchPickAndPlace.
2. Construct and tune mapping network for FetchPickAndPlace environment.
3. Write scripts to train and evaluate models in FetchPickAndPlace environment.
4. Collect data in FetchPickAndPlace environment.
5. Train and evaluate mapping network using one-shot.

References

[1] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann, 1989. URL http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf.

[2] A. Giusti, J. Guzzi, D. C. Cirean, F. He, J. P. Rodrguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, July 2016. ISSN 2377-3766. doi:10.1109/LRA.2015.2509024.

[3] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

[4] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/ross10a.html.

[5] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[6] B. Kim and J. Pineau. Maximum mean discrepancy imitation learning. In *Robotics: Science and systems*, 2013.

[7] M. Laskey, S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Shiv: Reducing supervisor burden in using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *ICRA*, pages 462–469, 2016.

[8] J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

[9] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer. Dropoutdagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1709.06166*, 2017.

[10] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[11] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

[12] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

[13] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017. URL http://arxiv.org/abs/1707.03141.

[14] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. *CoRR*, abs/1703.07326, 2017. URL http://arxiv.org/abs/1703.07326.

[15] S. James, M. Bloesch, and A. J. Davison. Task-embedded control networks for few-shot imitation learning. *CoRR*, abs/1810.03237, 2018. URL http://arxiv.org/abs/1810.03237.

[16] T. L. Paine, S. G. Colmenarejo, Z. Wang, S. E. Reed, Y. Aytar, T. Pfaff, M. W. Hoffman, G. Barth-Maron, S. Cabi, D. Budden, and N. de Freitas. One-shot high-fidelity imitation: Training large-scale deep nets with RL. *CoRR*, abs/1810.05017, 2018. URL http://arxiv.org/abs/1810.05017.

[17] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

[18] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016. URL http://arxiv.org/abs/1606.04474.

[19] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL http://arxiv.org/abs/1611.02779.

[20] S. Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[21] H. Akaike. Fitting autoregressive models for prediction. *Annals of the institute of Statistical Mathematics*, 21(1):243–247, 1969.

[22] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.