Imitation Learning on Suboptimal Expert Trajectories

Kaiyuan Huang **Lyla Kiratiwudhikul** **Yilin Wang**

1 Introduction

Imitation learning (IL) and behavior cloning (BC) are powerful techniques for scalability in reinforcement learning. However, these methods assume that expert demonstrations are optimal and are generated from the same reward function. In this project, we aim to investigate the effectiveness of imitation learning in different settings where the optimality assumptions are violated. In particular, we aim to explore different kinds of suboptimality in the expert demonstrations: (1) sub-optimal, incorrect actions (2) action biases, and (3) limited rounds of expert demonstrations. We seek to address this research question: which kind of suboptimality leads to the most performance degradation? To do so, we conduct an experiment on `ALE/Breakout` atari game using expert data from a pre-trained DQN from Mnih et al. [2015]. Results show that a model trained using a small, limited size of expert demonstrations fails at the behavioral cloning task where the game terminates after only the first few rounds. However, an expert's bias in taking a certain action less than the others (implemented using a downsampling method) leads to only a slight decrease in performance, compared to a model trained on suboptimal, incorrect actions.

Section 2 describes the methodology of our experiment and introduces different types of expert data suboptimality. We detail our experimental setup including data and model tuning in Section 3. Results are shown and discussed in Section 4. Section 6 concludes. The code for this project is available here.

2 Methodology

2.1 Notations

We consider an undiscounted, infinite-horizon MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu\}$, where $\mathcal{S}$ refers to the state space, $\mathcal{A}$ refers to the action space, $\mathcal{P}$ refers to the transition probabilities, $\mathcal{R}$ refers to the reward, and μ refers to the initial state distribution. We denote a set of expert demonstrations $\mathcal{D}^E = \{(s_{i,j}, a_{i,j})_{j=1}^H\}_{i=1}^N$, where $s_{i,j} \in \mathcal{S}, a_{i,j} \in \mathcal{A}$. The expert data contains N trajectories, each with time horizon H.

2.2 Behavior Cloning

We use behavior cloning as the baseline for analysis. That is, we optimize over a policy class $\Pi = \{\pi : \mathcal{S} \to \Delta(\mathcal{A})\}$ to find the policy that minimizes the empirical risk:

$$\hat{\pi} = \operatorname*{argmin}_{\pi \in \Pi} \sum_{i=1}^{N} \sum_{j=1}^{H} \mathcal{L}(\pi, s_{i,j}, a_{i,j}),$$

where $\mathcal{L}$ is the loss function. In the context of this project, we will only consider discrete action spaces, so the empirical risk minimization procedure can be formulated as a classification problem, and $\mathcal{L}$ is the cross entropy loss.

In this project, we will train a BC agent to play Atari, whose state space is $4 \times 84 \times 84$ images (the first dimension refers to 4 channels representing colors and other attributes of the image). Therefore,

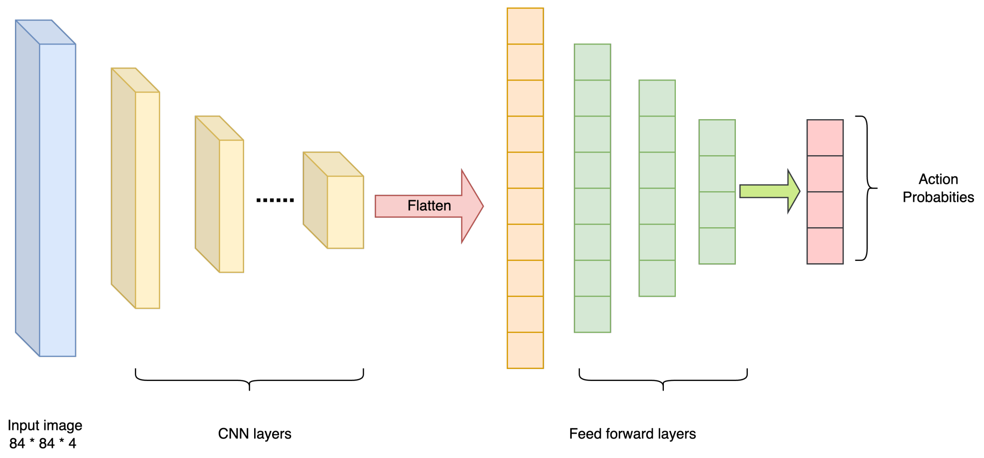

Figure 1: CNN architecture of the BC agent

we will consider the following policy classes: convolutional neural networks (CNN), feed-forward neural networks (FFN), decision trees (DT), and random forests (RF).

Figure 1 shows the architecture of the CNN BC agent. The CNN takes in a 84×84 image with 4 channels and passes it to a number of CNN layers. It then flattens the output from the last CNN layer and feeds it through three fully connected feed-forward layers (with ReLU activation function) with intermediate sizes of (1) the length of the flattened array, (2) 128, and (3) 84, consecutively. The last feed-forward layer then produces a length-4 array, which represents the predicted probability logits of each action. For the CNN layers, the first layer outputs 64 channels, the second layer outputs 128 channels, and the third layer outputs 258 channels. Max-pooling is applied after each convolutional layer. Each convolutional layer has kernel size = 3 and stride = 1, and each max=pooling layer has kernel size = 2 and stride = 2. We conduct experiments with multiple CNN agents, each with 1, 2, and 3 CNN layers. For more details, please refer to our code at `model.py` line 39-65 (class `cnn_agent`)

2.3 Types of Suboptimality

In this project, we explore different types of suboptimalities. We generate sub-optimal trajectories from the optimal expert data in three ways:

1. **Alter expert actions**: For randomly chosen expert state-action pair (s_i, a_i), we replace a_i with a “wrong" action randomly selected from $\mathcal{A} \setminus \{a_i\}$. This perturbation on expert data takes up a given portion of the original data (i.e. 20%, 50%, 80% of data).
2. **Downsampling certain actions**: For a given action, we downsample the expert data using that action to train a model. Given a ratio of perturbation p and action a_i, we randomly select $p\%$ of the data and remove all samples with action a in that portion of data. This creates a class imbalance problem for our supervised learning algorithms.
3. **Limit expert data size**: We will investigate the performance degradation with limited expert demonstrations, under different dataset sizes.

2.4 Evaluation Metrics

We evaluate the learned policies using two measures. First, we want to assess whether the behavior cloning algorithm succeeds in imitating the expert behavior. We will report the learned policy’s accuracy and F1 score on the test set of expert demonstrations. Secondly, we generate 10 rollouts using the learned policy and record the average reward. Though not presented here, we provide API in our code base that provides a visualization of the learned policies through OpenAI Gym’s interface.

3 Experimental Setup

3.1 Data

In this study, we focus on Atari games, which are widely used benchmarks in Reinforcement Learning (RL). More specifically, we use the `ALE/Breakout` game. To generate expert trajectories for the Atari game, we leverage a pre-trained deep-Q network (DQN) as provided by Mnih et al. [2015], which gives a training dataset of size $N = 30000$ and $H = 32$, and a test set of size $N = 5005$ and $H = 32$. The state space $\mathcal{S} = \mathbb{R}^{4\times 84\times 84}$ represents 84×84 images, and the action space $\mathcal{A} = \{0, 1, 2, 3\}$, where 0, 1, 2, and 3 respectively represents No action, Fire, Left, and Right, respectively. A more detailed description of this game can be found here

3.2 Optimization and Hyperparameter Tuning

For CNN model training, we use an AdamW optimizer [Loshchilov and Hutter, 2019] with a learning rate of 0.01 and weight decay of 1e-8. We train the model for 3 epochs, as we observe that the accuracy converges after 1-2 epochs. We then evaluate the model on the test set and report evaluation metrics.

For decision trees and random forests, we use `Scikit-Learn`'s implementation. We train the model on the training set and evaluate it on the test set. For all models, we experiment with different specifications (e.g., the number of convolutional layers in CNN, maximum depth in decision trees, and the number of estimators in random forests).

For suboptimality types (1) and (2) in Section 2.3, we use the 2-layer CNN trained on 960K samples for all experiments. We chose this particular model as the CNN model trained on 960K data is the only kind of model that achieves the desired level of average reward, as discussed in section 4.1. Thus, introducing suboptimality on this model could allow us to more clearly observe the performance degradation. For suboptimality type (1), we perform an ablation study by changing the percentage of the samples to perturb, and we conduct experiments with 20%, 50%, and 80% of the samples perturbed. For suboptimality type (2), we perturb 50% of the samples, but we perform four sets of experiments, which respectively downsample each of the four actions at a time.

4 Results

4.1 Behavior Cloning

Table 1 and figure 4 show F1 and accuracy scores on the test set using different supervised learning algorithms as described above. Due to computational constraints concerning RAM, we randomly chose 100,000 data from the whole dataset for decision trees and random forests. For CNN, we experiment with both dataset sizes. We can train CNN with all data as neural network training could be batched, while tree training could not.

We observe that our CNN model trained on 960k data points significantly outperforms other methods. This is expected, as the CNN is trained on more data and is specialized in processing images, which is how the state space is represented. Overall, we see that the models trained on less data achieve relatively high accuracy (around 80%) but low F1 and very low reward.

Additionally, decision trees perform on par with random forests in terms of F1, and random forests perform slightly better in terms of average reward and accuracy. It is surprising that with the same amount of training data, decision trees and random forests outperform CNN by around 10% F1, given trees are not designed to process images.

Model	# Data	# layers	# tree	max depth	F1	Accuracy	Reward
CNN	960k	1	/	/	67.98	92.77	50.4
CNN	960k	2	/	/	66.08	91.62	51.3
CNN	960k	3	/	/	66.82	91.98	51.0
CNN	100k	1	/	/	23.31	80.61	0.7
CNN	100k	2	/	/	23.11	81.13	1.2
CNN	100k	3	/	/	22.34	80.78	0.6
Decision Tree	100k	/	/	40	37.43	78.62	0.9
Decision Tree	100k	/	/	50	37.50	80.39	1.9
Decision Tree	100k	/	/	60	36.06	78.54	1.0
Decision Tree	10k	/	/	40	31.69	76.19	1.8
Decision Tree	10k	/	/	50	32.57	71.60	0.7
Decision Tree	10k	/	/	60	34.08	75.70	0.6
Random Forest	100k	/	100	50	35.26	83.52	1.4
Random Forest	100k	/	100	60	35.97	82.79	1.0
Random Forest	100k	/	200	50	35.83	82.93	2.2
Random Forest	100k	/	200	60	33.18	83.06	1.4
Random Forest	10k	/	100	60	26.26	81.17	2.2
Random Forest	10k	/	100	60	28.71	79.55	1.8
Random Forest	10k	/	200	50	27.45	80.35	0.6
Random Forest	10k	/	200	60	23.98	80.10	1.4

Table 1: Behavior cloning results using different supervised learning algorithms. # layers refers to the number of convolutional layers in the CNN. Max depth refers to the maximum depth of the decision tree. N estimators refers to the number of trees in the random forest.

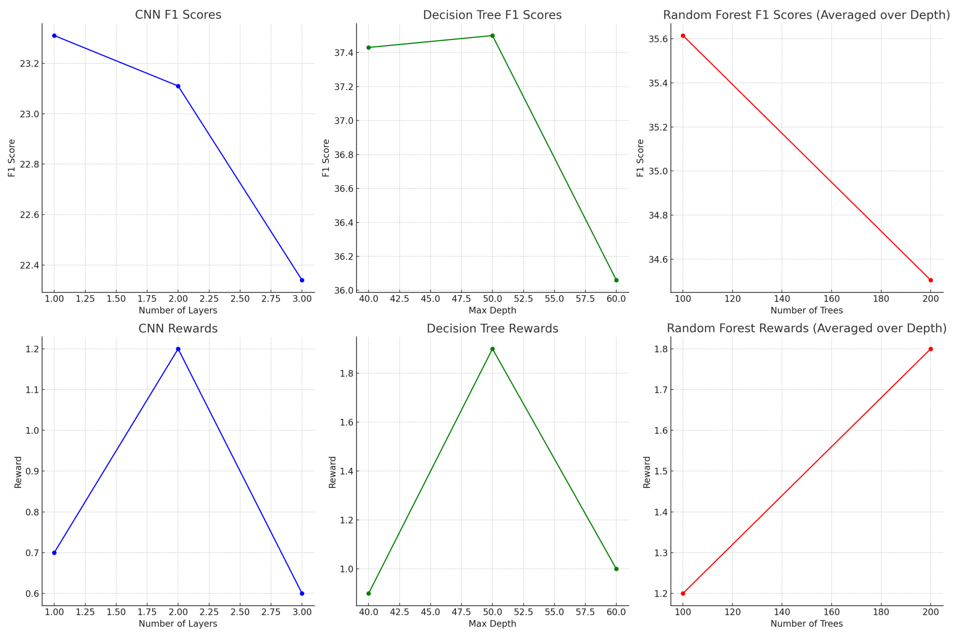

Figure 2: F1 scores on test set(the first row) and and Reward using CNN, Decision Tree and Random Forest across the numbers of layers and maximum depths, respectively

4.2 Suboptimality: Limited Expert Data

From table 1, we notice that although the models trained on less data could achieve around 80% accuracy, they achieve much lower reward compared to the 960K-data-CNN. Empirically, we observe that these BC agents “die" in the game after one or two iterations. We hypothesize that this is due to the imbalance in action distributions in the training data. Most data points have action “0", which represents “no action". The BC agents trained on this smaller set of data may predict “0" more often than they should, and thus it achieves low reward despite high accuracy. This is reflected in the low F1 scores as well, which more comprehensively measure the agent’s ability to predict the expert actions as it takes false positive and false negative errors into account for each action class.

4.3 Suboptimality: Randomly-perturbed Expert Actions

Model	# Data	Perturb%	F1	Accuracy	Reward
CNN	960k	1%	58.00	90.77	8.8
CNN	960k	5%	51.55	88.66	7.6
CNN	960k	10%	38.28	80.09	3.2
CNN	960k	20%	35.79	72.64	4.1
CNN	960k	50%	15.12	43.38	1.0

Table 2: Behavior cloning results using different percentages of random perturbation. Perturb% refers to the percentage of expert demonstrations that have randomly shuffled actions.

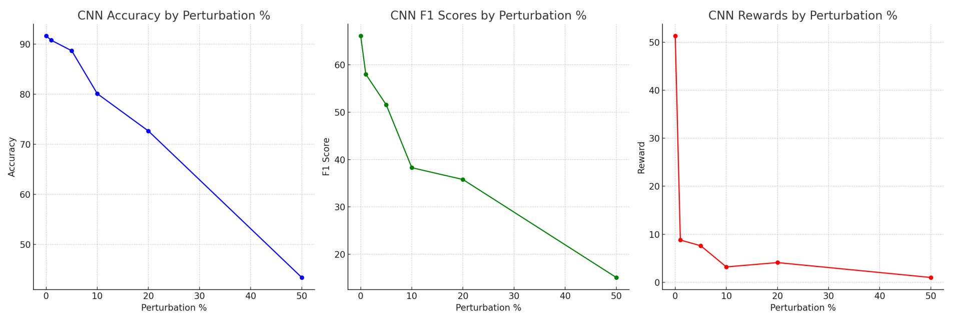

Figure 3: Accuracy, F1 scores and Reward using CNN across the perturbation rate, respectively

Table 2 shows the performance of CNN models trained on 960K data with random perturbations on the action variable on different percentages of training data. The result is consistent with our hypothesis, that with more perturbations (and, hence, less optimal), the BC agents perform worse in terms of every measure: F1, accuracy, and average reward. Most noticeably, even with only 1% of data randomly perturbed, the average reward of the BC agents dropped significantly from 51.3 to 8.8 and its F1 dropped from 66.08 to 58.00, while its accuracy merely decreased by two points.

We make two hypotheses from this observation. First, without the ability to do rollouts and compute average rewards, the F1 metric is a better way to approximate the BC agent’s performance than accuracy. Secondly, we hypothesize that there exists a threshold in the agents’ F1/accuracy. When the agent’s F1/accuracy is smaller than the threshold, the reward increases very slowly when the accuracy/F1 increases. However, once the agent’s F1/accuracy is greater than that threshold, the reward can increase rapidly along with F1/accuracy and eventually stabilize.

4.4 Suboptimality: Systematically Downsampled Expert Action

Model	# Data	Action	Perturb%	F1	Accuracy	Reward
CNN	960k	0	10%	63.52	91.63	7.8
CNN	960k	0	20%	59.45	91.26	11.2
CNN	960k	0	50%	63.46	91.41	7.2
CNN	960k	1	50%	53.74	91.57	6.1
CNN	960k	2	50%	61.46	92.53	3.7
CNN	960k	3	50%	67.17	88.01	19.7

Table 3: Behavior cloning results using different percentages of systematic downsampling perturbation. Perturb% refers to the percentage of expert demonstrations that are down-sampled. Action refers to the action that are downsampled

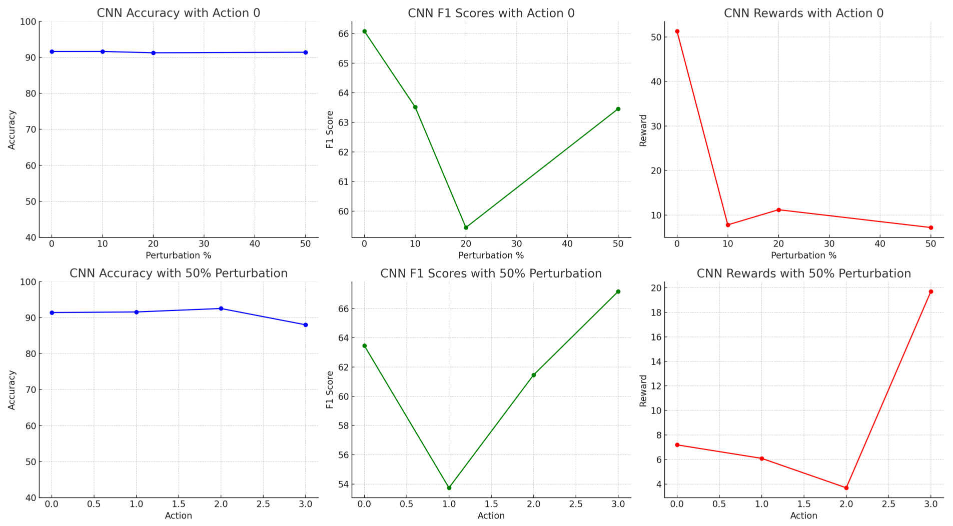

Figure 4: Accuracy, F1 scores and Reward using CNN across the downsample rate and downsampled action, respectively.

Table 3 shows the performance of CNN models trained on 960K data but with systematic downsampling perturbations of certain actions. We observe that downsampling different actions leads to different levels of performance degradation. It appears that downsampling action 2 (Right) leads to the most performance degradation, compared to the other three actions. A possible explanation is that the agent's initial state sits on the left-most position of the map, and downsampling action 2 (Right) traps the agent in its initial position. This also explains the observation that downsampling action 3 (Left) does not lead to a significant performance decrease.

Further, compared to random perturbations, which introduce "incorrect" samples, the BC agents trained under the downsampled dataset appear to perform better. For example, under a perturbation rate of 50%, the BC agent trained under a randomly shuffled dataset archives an average reward of 1.0, while the BC agent trained with the downsampled dataset achieves average rewards ranging from 3.7 to 19.7. This suggests that the BC algorithms are more robust to the class imbalance problem introduced by the downsampling, compared to the "incorrect" samples introduced by random shuffles. Additionally, note that with downsampling, the F1/accuracy measures become a less reliable predictor of average reward. We hypothesize this is because the downsampling procedure might manually decrease the false negative errors for the downsampled action, which increases the F1/accuracy of that particular action and raises the average F1/accuracy.

5 Visualization

Figure 5 shows the interface of the `ALE/Breakout` game. We provide an example of our BC agent (a 2-layer CNN model trained using 960K data) playing this game. We see that the BC agent succeeds in mimicking human behavior in this game. Please see the video demonstration in our code submission, named `demo.mp4`. The video is also available on our Github page here.

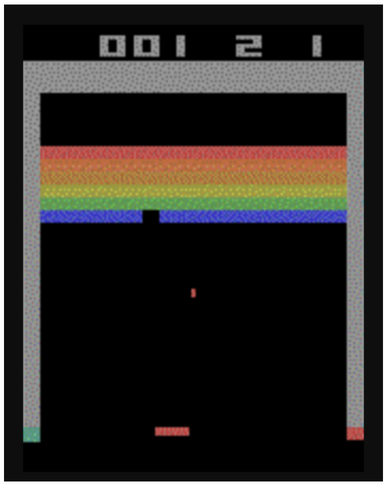

Figure 5: Visualization of game playing by a BC agent trained using a 2-layer CNN model using 960K data points

6 Conclusion

In this project, we explore the effectiveness of behavior cloning on sub-optimal expert data. To do so, we conduct an experiment on an Atari game, `ALE/Breakout` using expert data from a pre-trained DQN from Mnih et al. [2015]. For the baseline, we develop behavior cloning models using CNN and tree-based algorithms on a full dataset of size 960K data points. These baseline models achieve an accuracy of $79.9 - 92.1$, an F1 score of $22.2 - 68.20$ approximately on the test set. We observe that CNN models outperform tree-based models and generate higher average rewards of around 50-51 on multiple rollouts.

We investigate three types of suboptimality in this project: (1) limited training expert data, (2) incorrect, suboptimal expert actions, and (3) expert's biases in taking a certain action. The last type of suboptimality is implemented by training a model on downsampled expert data on a given action. The results show that limited training expert data leads to the worst performance while expert's biases in taking a certain action appear to affect the performance by the least significant amount. In addition, we note that class imbalance in the distribution of actions is present in certain suboptimality experiments, which results in poorer performance of our models in behavioral cloning tasks.

References

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL `http://dx.doi.org/10.1038/nature14236`.