Racing F-Zero with Imitation Learning

Theory and Reinforcement Learning

Stephanie Wang, Michael Brennan
{steph17, mrbren}@stanford.edu

December 16, 2017

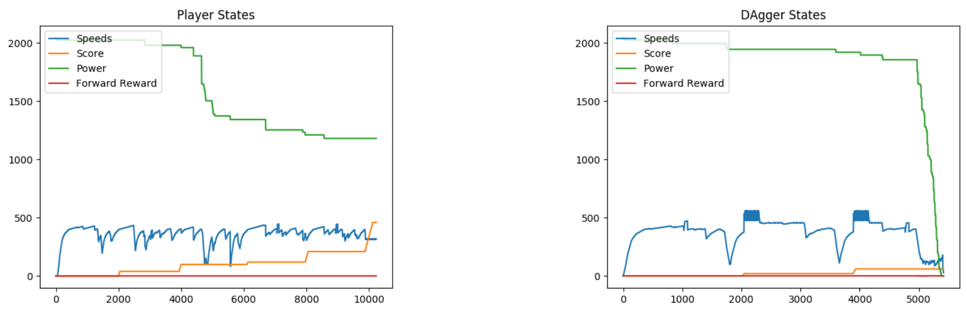
1 Introduction

With this project we use the Retro Learning Environment for the SNES game *F-Zero* in order to reproduce work in the Imitation Learning space. We were inspired by some of the recent successes in Reinforcement Learning and Imitation Learning, and hoped to achieve a deeper understanding of what kind of impact both model architecture, human, and hand-refined policies could have on the eventual learned policy. By playing the game ourselves, and by using supervised learning with respect to the action choices we made for each state, we developed a model of our behavior for which to set as a baseline for generating new data-sets using the DAgger algorithm Ross et al. [2010]. With our work we created a learning algorithm capable of producing successful policies by generalizing from the expert action choices for the various pixel states of the game.

For our approach we started with a dataset of $29,375$ samples of human-level playing of *F-Zero*, which we then trained a model on to generate our initial policy π. With this established we went about applying a hand-tuned reward function to see if we could iterate on π and optimize the performance by retraining π after the aggregation step of the DAgger algorithm to obtain a new policy π_i. Our new policy is formed by iteratively augmenting our dataset with a hybrid of greedy action choices picked at random and based on the confidence of the predicted action choice from our model. Using our new policy we then fine-tuned further the data we pass into the DAgger algorithm with a selection bias on what data would be recorded in the aggregation step to achieve our final policy π^*.

2 Related Work

The field of imitation learning stretches back much farther than some of the more current research in things like deep reinforcement learning, so a survey of the field is beyond the scope of this paper; however, we drew many influences from some of the more prominent works in this area. One such work, Pomerleau [1989], extended extremely well to our use case, as such we drew inspiration from this paper for our first attempt at creating a model of our playing for *F-Zero*. We had differing inputs and outputs for our use case given our inputs have just been the pixels of the game's screen and not simulated images of a real road, and the output had more control options than the 1-axis steering of Pomerleau's work on ALVINN, but our architecture for the model and our approach to solving the problem started out almost identically.



(a) Human

(b) DAgger

Figure 1: Human Performance vs DAgger (5 iterations)

After our first attempt at solving this problem we looked at some of the more recent literature on the subject of imitation learning, such as Ross et al. [2010] and the survey by Argall et al. [2009] for a better idea of what approach we could try next. Both document a similar approach to mapping state / action pairs to an actual policy based on demonstrations, with the drawbacks and pitfalls documented well in each. Learning from Demonstration was a source of inspiration for the simple use of Dataset Aggregation as shown in the Argall et al. paper as well, which we chose to move forward with.

While using the DAgger algorithm we found the architecture we chose unable to generalize well, with it always predicting the one dominant action and unable to predict various alternative actions. From our reading on Duan et al. [2017] we decided to experiment further with making architectural changes for our policy approximation.

3 Dataset and Features

The dataset we start with is one we generated by playing the game ourselves. The Retro Learning Environment (RLE) is an environment for algorithms to play games on the Super Nintendo Entertainment System (SNES) Bhonker et al. [2016], and using the RLE we were able to display and control the game, while also saving both the screen state as well as the corresponding controller inputs. The RLE currently supports over 70 games, but for this paper we choose to focus our efforts on *F-Zero*, a 2D back-view racing game in which the player must control a vehicle to race against opponents. To complete the game the player must finish 5 laps, and the one with the highest score (equivalent to completing laps in a good position) wins.

From our 3 game plays we generate $29,375$ RGBA images labeled with the controller input $\in [1, 12]$. We feed this data into the model described below by first preprocessing the input so that it's scaled from 0-1 instead of the original 0-255 RGBA values, and by transforming the labels through a one-hot encoding so that they can be directly compared against the output of the softmax layer of the model. We then use the model to generate new datasets by playing the game using this policy approximation, which we then augment with expert data from our greedy search algorithm. Once this new dataset is generated we then repeat this process. Throughout the creation of our datasets, the shape of the input, and the actions available remain consistent. The actions, range from steering left/right, driving, braking, and boosting, and each can be done in combination, the span of these are encoded $\in [1, 12]$, while our state space remains a consistent. Visualizations of the state that we use for the greedy search algorithm to augment the dataset can be seen over the span of a game for both the raw policy after running DAgger for 5 iterations, as well as the human policy in 1a and 1b.

4 Methods

4.1 Convolutional Neural Network

We train a convolutional neural network (CNN) to map raw pixel inputs of the game screen directly to game actions. We adopt the network architecture implemented by Bojarski et al. [2016] in DARPA Autonomous Vehicle 2 with a few adjustments. We train the weights of our network to minimize the categorical cross entropy loss defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{1}$$

Where N is the size of the training set, M is the number of classes, y_{ij} is a binary indicator of example i being classified correctly and p_{ij} is the model probability of assigning label j to example i.

Our final network architecture is shown in Figure 2. The network consists of 9 total layers, in which the first is a batch normalization layer, followed by 5 convolutional layers with ReLU activations and ending with 3 dense layers. We are classifying images in one of 12 gamepad commands so we apply the softmax function to output the most probable class.

[1]This figure is generated by adapting the code from https://github.com/gwding/draw_convnet

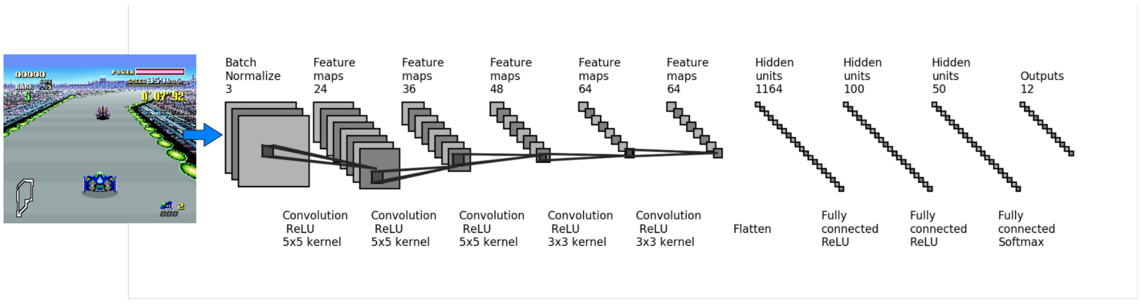


Figure 2: Final convolutional neural network[1]

4.2 Dataset Aggregation

The problem with simply using supervised learning to predict expert behavior given an observation is that the model may enter new states that have not been visited by the expert and thus have no policy in place. Thus there is a distribution mismatch between the training dataset and testing dataset, and future observations may lead to compounding errors from which supervised learning methods will not recover. For example, because we don't train our CNN on data in which we turn around from reverse positions, and because the image inputs going reverse appear similar to going forward, the CNN may continue to drive in reverse or have low confidence in choosing an appropriate action.

We use imitation learning, in which an agent aims to mimic human behavior in a given task, to tackle the problem of sequential decision making in a gaming environment. Specifically we make use of Dataset Aggregation (DAgger), an iterative algorithm that uses expert demonstrations to learn a policy that approximates the expert's decision making Ross et al. [2010]. DAgger collects a dataset at each iteration under the current policy, and trains the next policy on the aggregate collection of all datasets.

```
Train CNN on dataset $D$ as initial policy $\hat{\pi}_1$;
for each iteration of DAgger do
    begin new game;
    while game has not ended do
        run the CNN to obtain new states $s$;
        if if rand$(0,1) < \epsilon$ then
            get $D_i = \{(s, \pi^*(s))\}$ given by automated search algorithm;
            $D \leftarrow D \cup D_i$ aggregate datasets;
        end
    end
    Train CNN $\hat{\pi}_{i+1}$ on $D$;
end
```

Algorithm 1: Dataset Aggregation

We use the CNN trained on our game-plays to play *F-Zero*, randomly stopping the game to run an automated search algorithm which finds the best action to take for the next 50 frames. The automated search algorithm is our "expert player". We then augment the dataset with the newly generated labeled data and repeat the process, retraining the CNN initialized with previous weights on the augmented dataset.

4.3 Automated Search Algorithm

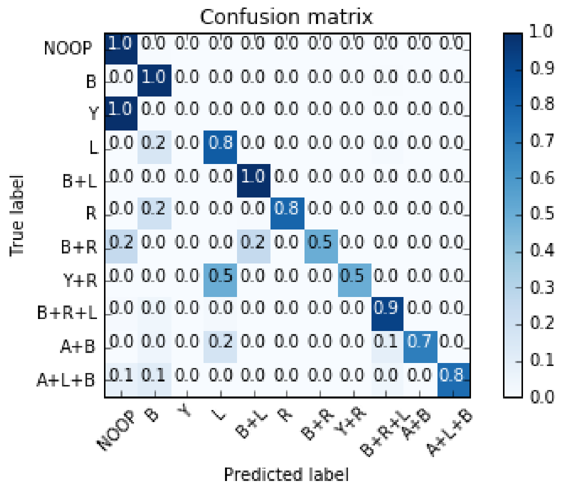
We implemented a simple search algorithm for *F-Zero* to automate the production of expert action choices while running DAgger. The search algorithm will, for each of three actions `RIGHT, NOOP, LEFT`, save the current game state and follow this action for 50 frames before evaluating future reward obtained from following that action. Then we return to the saved game state, taking, for the next 50 frames, the action that yields highest reward. We define reward as:

$$\texttt{reward = (isForward * power) + (isForward * speed * 100) + (score * 10)} \tag{2}$$

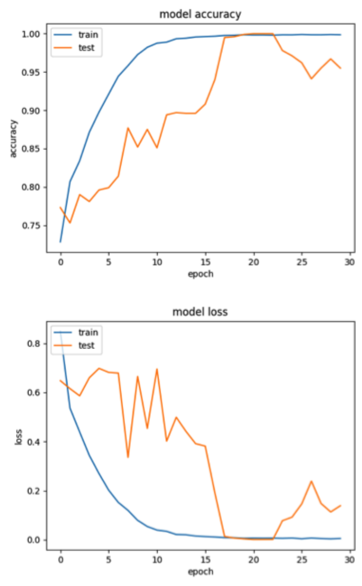
Where `isForward` $\in \{-1, 1\}$ indicates whether the vehicle faces reverse or forward, `power` $\in [0, 8000]$ indicates the power level remaining. Power decreases upon collision with another vehicle or when driving over green strips by track boundaries. `speed` $\in [0, 500]$ is the speed of the vehicle. These values are extracted from the *F-Zero* RAM.

5 Discussion

5.1 Experiments



(a) Normalized confusion matrix of CNN predictions



(b) CNN accuracy and loss

Figure 3: CNN learning on human game-play dataset

We implemented the CNN in Keras using Tensorflow backend and a NVIDIA Tesla P100 GPU. Inputs to the network are the raw $224 \times 256 \times 3$ screen pixels saved from the game and corresponding action label $\in [1, 12]$. To train our initial policy CNN, the data set of our 3 game-plays are split 80% for training, 20% for test. We train the CNN for 30 epochs with batch size of 128, using Adadelta optimizer for speed. Figure 3b show the model accuracy and loss on training and test sets per epoch. The CNN achieves 95% accuracy on the test set. Even though high accuracy does not indicate that our CNN will drive well on the track, we report it to benchmark how well our model learns the given dataset.

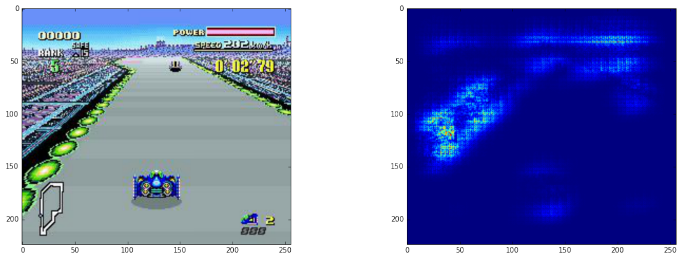
Figure 3a shows the confusion matrix of the CNN's predictions on the test set, where the class label has been converted to the associated gamepad action for interpretability. Command `Y` corresponding to brake is often misclassified as `NOOP`, and in 50% of the cases we label `Y+R` corresponding to brake and turn right, it is labeled `L` which is turn left. These two mis-classifications are interpretable – the latter may be situations in which the player is turning right but suddenly needs to turn left so the brake button is briefly applied.

5.2 Benchmark Comparisons

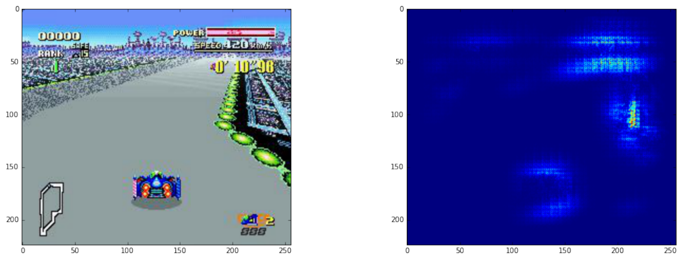
Animal	Avg Game Score (10 plays)	Avg Laps (10 plays)
Human	3400	5.0
Random	0	0.0
CNN only	390	1.2
DAgger	700	3.0

We run DAgger for **16** iterations, each iteration we anneal the probability that the automated search algorithm interjects to predict the next action to take. We save the dataset generated by the search algorithm each time the car completes a lap. This is to avoid saving unsuccessful actions that don't lead to our goal of completing the race. We find that DAgger is able to complete all five laps to beat the race on a good run, where as just the CNN trained on our game-play is unable to finish all five laps in 10 trials. Table 5.2 shows that on average DAgger obtains higher score and completes more laps than when predicting with the CNN trained without augmented data. Note that the table shows the in-game scores which include multipliers for completing the later laps. Our implementation of DAgger does not match the scores of the perfect human player since there are states where the automated search algorithm itself makes imperfect predictions. For example, areas by the track boundaries have bumpy protrusions that slow the vehicle down. To optimize for speed, the search algorithm then navigates onto the green power-sapping strip which does not have slowing properties, but does decrease power. We consider making our search algorithm more robust in future work.

5.3 Visualization of Internal CNN state



(a) Straight road



(b) Curved road

Figure 4: Saliency maps

Figures 4a and 4b show the saliency maps of the DAgger trained CNN given two inputs, computed by taking the gradient of output category with respect to the input (Equation 3) Kotikalapudi [2017]. The former is of a straight area of track and the latter is of a curved area. As expected, input regions that are most salient are the imminent track boundaries which, when hit, slow the vehicle and sap power. Additionally note the "halo" surrounding the vehicle which may be attention directed to vehicles approaching from behind and incoming course obstacles. Less interpretable is the saliency of the power bar. One explanation is that our automated search agent computes power levels to determine the next best action, and thus when data is aggregated incorporating the search agent's action, the CNN learns falling power levels indicate a certain action to take.

$$\frac{\partial output}{\partial input} \tag{3}$$

6 Conclusion

We've successfully implemented DAgger to train a Convolutional Neural Network to race F-Zero and complete all laps, making turns while avoiding power-sapping strips, slowing pads and navigating launch pads that can send one veering off course and instantly end the game. While our implementation does not match that of an expert human player due to our naively greedy search algorithm serving as a proxy for the expert player, we show over 2× improvement using DAgger over using standard supervised learning to make action predictions in a sequential decision making task.

In the future we will try using Deep Q Networks (DQN) to race *F-Zero*. Since much of the framework is already in place and we have a defined reward function we have already shown the usefulness of. We believe this would consist of a simple extension of our current implementation. Additionally, we hope to run our model trained with DAgger on unseen tracks in *F-Zero* to test how well the model generalizes to similar yet slightly altered levels. Finally, we consider improving our search algorithm to act optimally under non-greedy action selection, or even having ourselves serve as the expert policy when overseeing the generation of new data on each DAgger iteration.

Contributions

Stephanie: Developed CNN in Keras (`run_dagger.py`), preprocessed images and labels and ran experiments using GPU tuning hyperparameters such as num epochs and batch size. Generated CNN visualization 2. Generated final Figure 3b on the test set. Wrote `confusion_matrix.ipynb` to create confusion matrix shown in 3a and provided report analysis. Developed `data_generator.py` to fit the model on a generator of our large image dataset (cannot fit image arrays memory). Contributions to `game.py` and modularized code. Iterated on reward function through experimentation, extracted `POWER` metric from *F-Zero* RAM. Implemented DAgger in `run_dagger.py`, saving expert action only after each lap is complete, model able to finish all 5 laps, ran experiment results to obtain 5.2 and provided report analysis. Generated internal visualizations of CNN in `cnn_filter_vis.ipynb`, generated Figures 4a and 4b to investigate CNN saliency and provided report analysis. Wrote and printed final project poster.

Michael: Developed the mechanism for playing the game by hand and automatically via greedy search, allowed for storing and saving out of game state from the RLE, as well as surfaced the actions and created the initial reward function, as seen in `game.py`. Contributions to `run_dagger.py` in implementing the inner main loop for file handling and serialization / de-serialization of game state data for feeding into the model, as well as reward and action data in aggregate to be passed into the model training function. Refactored `game.py` in order to enable the model policy to control the game and correctly store the new data generated. Extracted all other states from the game necessary to implement the 'expert' playing from RAM and wrote the model serialization. Contributed automatic player / data acquisition sections of the poster, and contributed to the report analysis by way of figures 1a and 1b.

References

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2008.10.024. URL http://www.sciencedirect.com/science/article/pii/S0921889008001772.

Nadav Bhonker, Shai Rozenberg, and Itay Hubara. Playing SNES in the retro learning environment. *CoRR*, abs/1611.02205, 2016. URL http://arxiv.org/abs/1611.02205.

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

Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *CoRR*, abs/1703.07326, 2017. URL http://arxiv.org/abs/1703.07326.

Raghavendra Kotikalapudi. keras-vis. https://github.com/raghakot/keras-vis, 2017.

Dean Pomerleau. Alvinn, an autonomous land vehicle in a neural network. 1989. URL http://repository.cmu.edu/cgi/viewcontent.cgi?article=2874&context=compsci.

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