In Woo Park

ICS 635

15 April 2022

OpenAI CarRacing with Behavioral Cloning

1. Introduce the Problem

OpenAI CarRacing with behavioral cloning is a machine learning program built with python in google colab where we have to train a car to drive on a track for as long as possible. We are given a dataset which will be used for training our car which consists of a list of states and actions. Once we preprocess the data, we will train the model and return a video showing the car attempting it's drive along a randomly generated track and measure performance by returning a score (reward).

2. Introduce the dataset

The dataset is provided by the Professor where it has 11,132 example (state, action) pairs you can use for training. These were sampled from simulations of a highly-skilled AI agent. The first cell downloads the data and installs many of the dependencies needed to run the simulations and generate videos in Google Colab. You should be able to train your agent and view videos of your agent within Colab.

3. Introduce the model

In the context of reinforcement learning, our training strategy is known as behavioral cloning. We will be using the Keras to build our neural network model, or more specifically, to build our convolutional neural network as it's commonly used to analyze visual imagery.

4. **Specify features and pre-processing.**

The data is provided by Professor Peter Sadowski's google drive and will be called ```carracing_behavior.gzip```. It is a dataset of 11,132 example pairs of states and actions we will use for training. The features are the state and actions. The states are a (96,96,3) color image which shows the position of the car along with the current speed, the steering position, and braking status. The actions are (-1, 1) for steering, (0,1) for acceleration, and (0,1) for brakes. This has been expanded and simplified by converting the choices to 7 discrete actions (i.e., 0 Do nothing, (1) Left, (2) Left+Break, (3) Right, (4) Right+Break, (5) Accelerate, (6) Break.). For preprocessing the data, I shuffled the data randomly before I split the data into training and validation sets.

5. **Specify data splits and how they are used.**

The data was split into a 80/20 ratio where 80% of the dataset was used for training (11,132*80% = 8905.6) and 20% of the dataset was used for validation (11132 x 20% = 2226.4). Other split ratios I attempted were 90/10 but I ended up with good training accuracy but low validation accuracy, and 50/50 split which had less than 0.50 for both training and validation accuracy. I ended up choosing the 80/20 split because it seemed to provide the best accuracy when none of the other hyperparameters were touched during my simulation attempts which helped with overfitting issues.

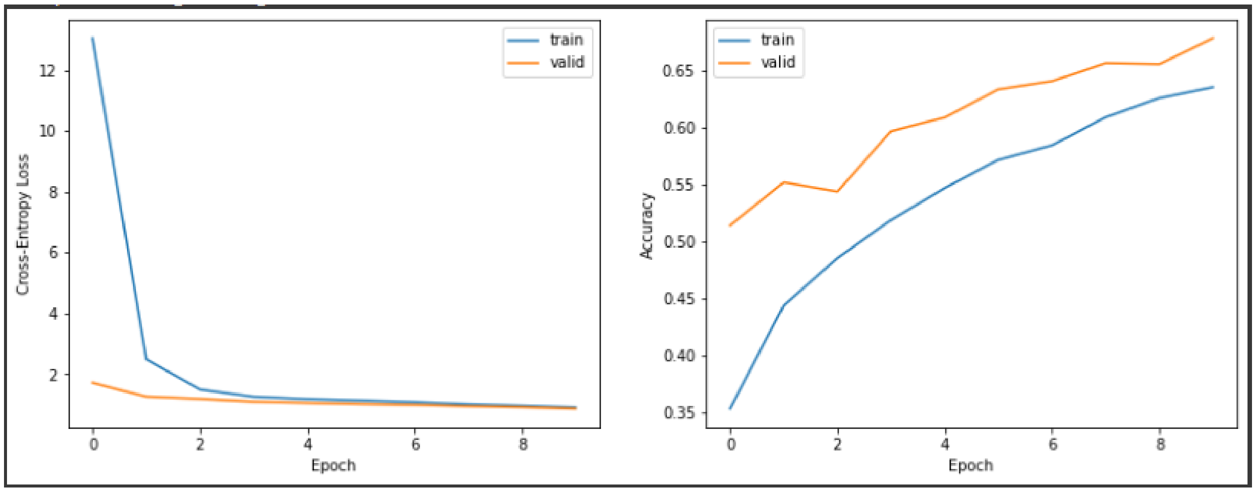
6. **Specify hyperparameter search space/how hyperparameters were optimized.**

Hyperparameter	**Selected**	**Reason**
Train-Test split	80/20 ~ test/validation	Best fit after testing other splits.
Learning Rate	**lr = 0.0001** (other: 0.1, 0.001)	Trial and error showed that 0.0001 was the best choice.
Optimization Algorithm	Adam optimizer	Good choice for quick results.
Activation Function	ReLU (3), Softmax (1)	Based on Deep Learning Quick Start, they seemed to be the best choices.
# of hidden layers	8 hidden layers: .Conv2D **(relu)** .MaxPooling2D .Conv2D **(relu)** .MaxPooling2D .Dropout .Dense **(relu)** .Dense **(softmax)**	The discord server was recommending using more convolutional and max pooling layers.
Loss Function	Categorical Cross Entropy	Categorical cross entropy is the appropriate loss function for the softmax output.
Dropout Rate	0.2. **0.4**, 0.6	Documentation said anything above 0.2 is a bit extreme with the dropout rate but I found that 0.4 provided better results.
Epochs	4, **10**, 50, 100	I chose a low epoch of 10 because when I tried anything above 50 I was getting overfitting results. 10 was also a good number for fast training and tuning other parameters. We also want to stop as soon as the accuracy starts to decrease to not overfit.
Pool Size	(2, 2)	I asked around to decide the best pool size.
Batch Size	64, 128, **256**	After trial and error, 256 returned the best score of 921 during one of my trials.

7. Evaluate model on clean test set.

If the car is stuck or starts to drive the wrong way, the reward will be a negative value, lowering the score. At every action where we successfully drive through the track, we increment 1 point to the score. The scoring/reward that is returned after an agent is simulated is correlated with how far a car gets through the track. I believe 1 timestep = +1 /-1 reward increment.

In the section where we simulate our agent, I believe we are randomly generating a new track to test our car on so I thought it would be a clean test set but according to the discord channel, "You can just report the validation set performance, even though it's not totally clean." Therefore, after I run the initial 10 epochs during training, the cell will print a graph showing the cross entropy loss and the accuracy of training the car.



This figure shows the results of the training. Observing this, I believe my hyperparameter optimization is actually underfitting. Leaving every hyperparameter as is, I simulated the agent with the current training multiple times to get a scope of how the car does on the new track. The scope of the returned scores was anywhere between [-14: 921] where -14 was when my car spawned and did nothing but idle or 921 (my highest score) when my car went surprisingly super far considering my parameter tuning was mostly trial and error. For runs with negative values, I made sure to retry another simulation just in case something was buggy on my end. However, the average of my last 5 simulation attempts returned (876 + 921 + 863 + 890 + 731)/5 = 856.2. I would say an average score of 856 is pretty good considering that the validation and test accuracy doesn't reach past 0.7.

However, this doesn't always happen. I probably ran the training like 20 different times until it returned one where it matches the figure. When you run the training again, there are instances where it can majorly overfit and the simulation can return poor scores [100 : 300]. The first five simulations in google colab are based on the same training and just multiple simulations. The last simulation cell was not based on the same training. After the initial 5 runs with the same training, I trained it once more and ran the 6th simulation just to check on the behavior of the code. The simulation returned a score of 887. This doesn't guarantee that the simulations are reproducible. If I were to do this assignment again, I would suggest saving the training weights or setting a seed to make sure it is reproducible.

8. **Explain any differences in the train/test datasets.**

From what I observed by manually printing the training dataset to see what happens behind the scenes, on top of the preprocessing step where we shuffle the data, there seems to be more instances where the frames are more biased towards left turns than right turns. This is probably what's messing up the training part a little bit because when we train the car, there's probably more weight for left turns than right. This is maybe why increasing the dropout rate to 0.4 (which was considered extreme) was beneficial in the end because I was lowering the weight of training on left turns. The test dataset should just be randomized because the track is generated randomly when we run the agent simulation cell.

9. Other observations

When I call the simulate agent cell, we set the number of timesteps to be 2000 but when the simulation is complete, it returns that the episode finishes after 1000 timesteps. I don’t understand if this is just a visual bug and we’re running the full number of timesteps or if we’re only running half the amount of time steps. I tried increasing the number of timesteps but it still seems to return 1000. I could not find a working solution but it wasn’t much of a problem because I was still returning “pretty-high” scores (average: 856). Perhaps it’s because it completes within 1000 timesteps so increasing it doesn’t change anything.