CS 237B: Principles of Robot Autonomy II
Problem Set 3

Name: Pei-Chen Wu
SUID: pcwu1023

Problem 1: Getting Started

(i) Weight initialization is an important consideration in the design of a neural network model. Explain what problems or advantages may arise in a deep neural network if we initialize all weights and biases with 0.

Initializing all the weights with zeros leads the neurons to learn the same features during training. The cost will be the same over the training. In fact, any constant initialization scheme will perform very poorly. For example, assume we initialize all the biases to 0 and the weights with a constant. Thus, all the hidden units will have identical influence on the cost, which will lead to identical gradients and all the neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

(ii) The data you are going to use in this homework is complex and requires modeling nonlinearities. Often in such cases, it becomes very important how the weights are initialized for training time and finding better optima. Read about "Xavier initialization" [1] and explain what it is trying to do and why it is helpful. In your codes for the subsequent problems, use Xavier initialization for the weights you are going to train. You may manually implement it or you may use tf.keras.initializers.GlorotUniform.

The aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. Xavier Initialization initializes the weights such that the variance of the activations are the same across every layer. This constant variance helps prevent the gradient from exploding or vanishing. Using Xavier initialization would either initialize the weights as normal distribution around $-\frac{1}{\sqrt{N}}$ and $\frac{1}{\sqrt{N}}$. The variance term of the latter distribution is the harmonic mean of $\frac{1}{N}$. Xavier initialization works well with tanh activations, therefore, I will use tanh activation functions for the neural network models for the following homework questions. If you are using ReLU, for example, a common initialization is He initialization.

(iii) In the starter code we provided for this homework, we incorporated a different optimizer called Adam [3]. Read about Adam optimizer, and explain how it might be helpful to accelerate learning or how it might bring other advantages. Also discuss the potential drawbacks you think Adam might have.

Adam optimizer computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It combines RMSProp and Momentum update together to update weights and bias in the gradient descent. It also has bias correction. The algorithm has been proven to be very effective for many cases, computationally efficient and has little memory requirement. Adam is a versatile algorithm that scales to large-scale high-dimensional machine learning problems.

(iv) Let's say you are training a neural network using SGD in two different ways: (1) You train the network with some constant learning rate for 500 epochs. For each epoch, you record the batches you used. (2) Using the same initialization and same batches for every epoch, you train the network with the

same learning rate for 250 epochs, and then stop the execution. You then restart the optimization and train for another 250 epochs starting from the resultant network of the previous 250 epochs, again by following the batches of (1). So you exactly followed (1) in (2), but stopped and restarted the optimization once. Would you obtain the same network from (1) and (2)? What if you repeat the same experiment using Adam optimizer? Explain.

Stopping and resuming model training process doesn't effect accuracy and model performance and time for training by itself.
Technically speaking we use model split training for two reasons:
1) your training session is eliminated and training procedure is stopped (due to a power outage, GPU session timing out, etc.).
2) modify your learning rate to improve your model accuracy.

Using Adam optimizer sometimes it might be a bit different (this didn't happen to my training. I had to restart over again, so it will be slightly different), however, most of the time I got the same results when I trained the model used (1) and (2) respectively.

Problem 2

(i) The most straightforward way to perform behavior cloning is to assume there exists an underlying deterministic policy, but the data set contains some noise in the actions. Then, you can simply try to learn a function h(o,g) that outputs the estimated expert action. Modeling this function as a neural network, let's write it as h_θ(o,g) where θ denotes the network parameters (weights and biases). Assuming a loss function L between two actions, write an optimization problem to learn the expert policy by optimizing θ.

Optimization Problem:

$$argmin_\theta E_{(s,a^*)\backsim P^*} L(a^*, \pi_\theta(o,g)) \tag{1}$$

$P^* = P(s|\pi^*)$

π^* is an expert policy such that $\pi^*(a|o,g)$ is the probability of taking action $a \in A$ when the agent observes $o \in O$ and the goal is $g \in G$. The probability distribution over the set of observations given the state is $\Omega(o|s)$.

(ii) CODE

(iii) Describe the full training procedure you used for each goal.

Goal and Training Procedure:
Left: 1200 epochs, 1e-4 learning rate
Right: 1000 epochs + restore 200 epochs. 1e-4 learning rate.
Straight: 1200 epochs. 1e-4 learning rate.

(iv) Report the success rate you achieved.

Successful Rate:
Left: 0.99
Right: 0.8
Straight: 0.75

(v) What loss function did you use? Write it mathematically. If you perfectly overfit the data, what would be the minimum loss you see? Is it bounded below?

I use L2 norm for both steering (0th dimension) and throttle (1st dimension) and combine these two L2 norm linearly.

$$L_{steering} = \sqrt{\sum_{i=0}^{N} (y_i[0] - y_i^{est}[0])^2} \tag{2}$$

$$L_{throttle} = \sqrt{\sum_{i=0}^{N} (y_i[1] - y_i^{est}[1])^2} \tag{3}$$

$$L_{total} = L_{steering} * 0.99 + L_{throttle} * 0.01 \tag{4}$$

If we perfectly overfit the data, the minimum loss we will see will be 0. It is bounded at 0, since the L2 norm loss needs to be larger or equal to 0.

(vi) You will instead learn the entries of the mean vector and the covariance matrix. However, the covariance matrix needs to be a positive semi-definite (PSD) matrix! How can you ensure this?

The way of ensuring this is during training, we learn the entries of the mean vector and the covariance matrix, in order to make the covariance matrix to be positive semi-definite (PSD) all the time we simply multiply the matrix we got from the output of neural network with its' transpose and add a diagonal matrix of small values, which can make the covariance matrix to be PSD. Then, we can sample the action from the probability distribution.

(vii) We are going to use mean log-likelihood. Since we will be maximizing this quantity, we can think of the negative mean log-likelihood as the loss function. Then, if you perfectly overfit the data, what would be the minimum loss you see? Is it bounded below?

The smallest value of negative log will be $-inf$. Therefore, during training, we will see negative loss and it's not bounded below.

(viii) CODE

(ix) Describe the full training procedure you used to train the mixture density network for each goal.

Goal and Training Procedure:
Left: 1200 epochs at 1e-4 learning rate.
Right: 1200 epochs and restore for another 1200 epochs at 1e-4 learning rate.

Straight:
100 epochs at 1e-3 learning rate.
restore 1980 epochs at 1e-4 learning rate.

(x) Report the success rate you achieved.

Left: 0.9
Right: 0.8
Straight: 0.77

Problem 3: Conditional Imitation Learning

(i) Using the same code you wrote in the previous question to train a policy for the intersection scenario collectively for all goals and check the result. What are the some problems associated with this policy? Do you have control over with which direction the car is going?

I trained the model for 1500 epochs at 1e-4 learning rate for all goals.
Sometimes, I feel I don't have control over the car, but sometimes I do feel that I have control over the car. Therefore, I ran the test without visualization. The success rate is about 0.6, however, I actually feel that I don't have actual control most of time, when I feel I have control over the car it might just simply because the goal of the car is the same as the direction I want to go.
The problem of this is we don't want every time it encounters an intersection, the car just takes the same direction.

(ii) CODE

(iii) Test your trained CoIL policy for 10 episodes and report the success rate. Repeat this last step for every goal.

Success Rate:
Left: 0.92
Right: 0.88
Straight: 0.79

(iv) Describe the full training procedure you used.

Goal and Training Procedure:
For all the left, right, straight, I have done the following procedure.
100 epochs at 1e-3 learning rate
restore 100 epochs at 1e-4 learning rate.
restore 100 epochs at 1e-5 learning rate
restore 100 epochs at 1e-6 learning rate.

Problem 4: Intent Inference and Shared Autonomy

(i) Assuming a uniform prior over the goals (for $P(g|o)$), how can you compute $P(g|o,a)$?

$P(g|a,o) = P(g,a,o)/P(a,o) = P(g,a,o) * P(g,o)/(P(a,o) * P(g,o)) = P(a|o,g) * P(g,o)/P(a,o) = P(a|o,g) * P(g,o)/(\sum_g P(a,o,g)) = P(a|o,g)/(\sum_g P(a|o,g))$

The following equations have been used to get the answer.
$P(a,o) = \sum_g P(a,o,g)$
$1/P(a|o,g) = P(o,g)/P(a,o,g)$
$P(a|o,g) = P(a,o,g)/P(o,g)$

Therefore, $P(g|a,o) = P(a|o,g)/(\sum_g P(a|o,g))$

We can get $P(a|o,g)$ from the mixture density network. The result is actually the normalized value of $P(a|o,g)$.

(ii) CODE

(iii) Running the above command, drive the car in each direction for 5 times. Out of the 15 episodes you drove, how many of them ended up predicting the correct intent? What might be some reasons why it fails? Discuss.

Most of the episodes end up predicting the correct intent, I could see it's either very high probability that it shows the correct intent or it's blinking to the correct intent however, when the car finishes turning, it shows straight. I think this actually makes sense because no matter which direction the car turns to, after turning, it's just like going straight, therefore, some cases just keep showing straight after turnning.

Here is the detail description for some of the cases it failed:
When I drove the car, the car moved to where I want it to move and predict the correct intent for 1 second, but it soon goes back to the other prediction, for example, when I keep driving left, it would show left for a second, then back to straight, when the car actually turns left, it will keep showing straight.

(iv) Lane Change Scenario Training

I got pretty good success rate from the lanechange scenario testing.
Left: 1000 epochs at 1e-4 learning rate. Success Rate: 1.
Right: 1000 epochs at 1e-4 learning rate. Success Rate: 0.69

(v) We further want to enforce some constraints on a_R. Specifically, we don't want the steering and throttle provided by a_R to be too large. Hence, you will simply apply a threshold to the computed a_R. From a human-robot interaction perspective, what is the reason we are enforcing this constraint on a_R?

The reason is definitely the safety issue. We can see that the default maximum steering is 0.05 and the max throttle is only 0. This means that we don't actually want the robot to move the car forward without human's permission. We only want the robot to help us drive the direction of the car, however, we still don't want the robot to overtake the whole car without human's control.

(vi) CODE

(vii) When you control the vehicle with Shared autonomy, do you feel the help by the robot? Is it really helpful or does it make things worse? Elaborate on how it is helpful or why it might be harming your performance.

I feel the robot is not helping to drive the car. It always tries to switch to the right lane for some reasons no matter where it starts. It might because my training for the right is worse than left (left has 100% success rate, right has only 0.69).

However, I can still drive the robot easily without any issues, I think it's because we only set maximum steering to be 0.05, which prevents the robot to takeover the car from human driver.