CS 237B: Principles of Robot Autonomy II  
Problem Set 03

Name: Ricardo Paschoeto  
SUID: rp304154

**Problem 1: Getting Started.**

(i) When we initialized all weights with “0” the derivative with respect to loss function will be the same for every weight in the layers W, all weights will have the same value in the iterations. The hidden units are symmetric and continues, the model works as a linear model and neurons learn the same features during training process, in some cases this will be too simply to achieve good results when used in complex systems. The biases have no effect when initialized with 0 and will not create any problems.

(ii) Xavier Initialization prevent the gradients of network’s activation from vanishing or exploding based on the following theory:

* The activation’s mean should be zero ->
* The activation’s variance should stay the same for every layer ->
* Use activation function which is linear (approx.) when works with small inputs.

Where,

Therefore, for Xavier Initialization, for every layer we have the weights from a normal distribution with , the biases are initialized with zeros:

Or

Then, the conclusion, the Xavier Initialization avoid choosing the weights initialization too-high (exploding gradients) or too-small (vanish gradients) setting .

(iii) Different from classical SGD that applies the same learning rate to all parameters update, Adam algorithm provide a heuristic approach without requiring tunning learning rate manually, it’s important mainly when we have a sparse data, and we want to update the parameters in different extent instead. Adam algorithm allows the learning rate to adapt over time and achieves faster convergence, these factors helps in the learning process.

Some other advantages of Adam optimizer:

* Implementation is simple, therefore, could be implemented with fewer lines of code, e.g., Python.
* Computationally efficient
* Little memory requirement
* The magnitude of parameter updates are invariant to diagonal rescaling of gradient:

The iteration step in the dimension of the parameter (in the iteration) is given by,

When scaling the partial derivative of the parameter, we can multiply the value by *c>0*,

,

,

Thus, we get:

,

Considering very small,

,

Therefore, scaling the partial derivative of the parameter by factor *c* doesn’t affect

* Appropriate for problems with large data and/or parameters
* Does not require a stationary objective.
* Appropriate for problems with very noise and/or sparse gradients
* Hyperparameters have intuitive interpretation and require little tuning.

Some Disadvantages:

* Adam does not converge to an optimal solution in some areas
* Adam can suffer a weight decay problem

***Source:* *https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d***

(iv) SGD uses the constant learning rate to update NN parameters, in (1) and (2) we have the same initialization, batches and learning rate, in (2) we slice the epochs (250 + 250), although despite this difference we just split the train process from (1) because we restart from resultant neural network with the same initialization, batches and learning rates and we obtain the same network from (1). Although if we use Adam optimizer the NN will be different because Adam uses a variable learning rate to update de parameters.

**Problem 2: Behavior Cloning.**

(i) Optimization Problem:

, *Xavier Initialization.*

*.*

*Where optimization process (update weights) is given by Adam algorithm:*

(iii) Full training process:

* First, we initialize the weights using Xavier Initialization in the neural network layers (weights and bias).
* For each epoch - to “left” and “right” goals (g) the number was 1000 for “straight” using --restore command 3 times:
  + For each batch the forward pass (*call()* function) using the pair (o, g) is done by call function and the prediction action is evaluated in the output ().
  + The loss function is evaluated (L2 loss) from predict action and action from samples (). For “left” and “right” goals were used the same penalties factors to “steering”, “throttle” and “straight” - 1.0
  + The gradient and Optimizer were applied, Adam optimizer is used with learning rate of 0.0003 (“left” and “right” goals) and with learning rates 0.0003 and 0.0001 for the “straight” goal. The parameters () are updated and learning rate () is adapted over time by:

The process is repeated for each epoch and inner batches.

(iv) Success Report:

* “left”: Success Rate = 0.84
* “straight”: Success Rate = 0.99
* “right”: Success Rate = 0.97

Texto

Descrição gerada automaticamente com confiança baixa

Figure - Left Success Rate.

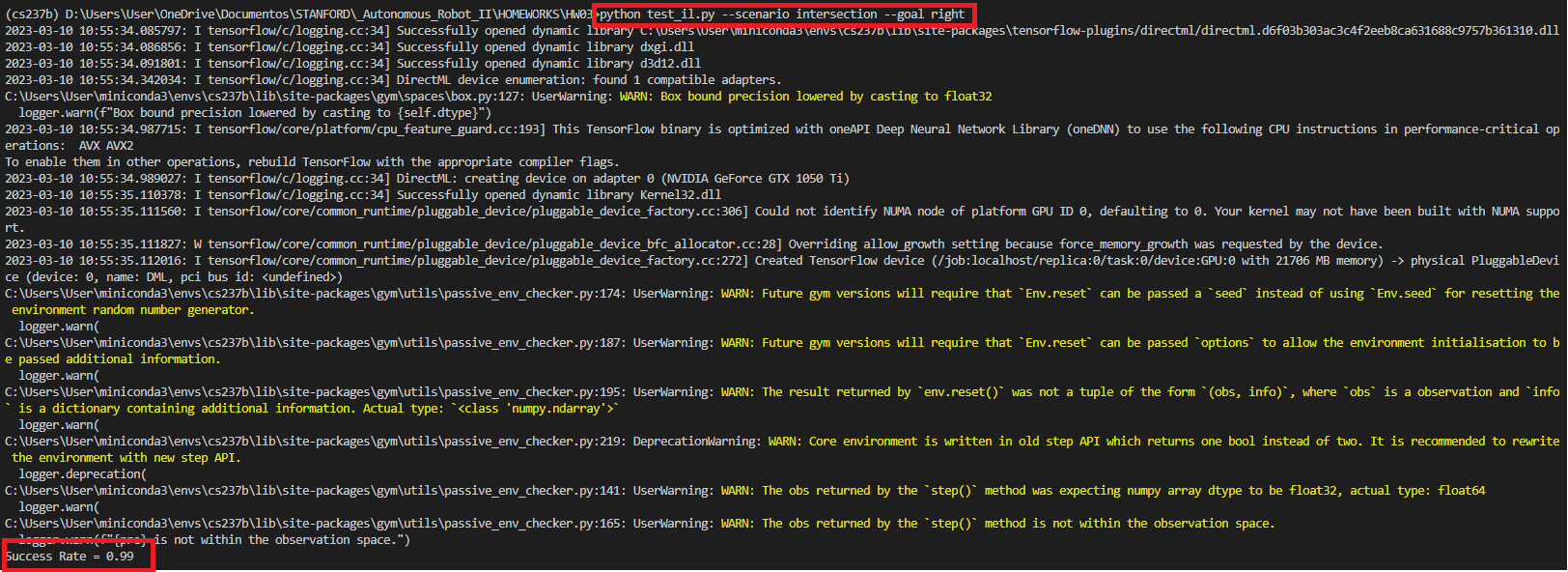


Figure - Right Success Rate.

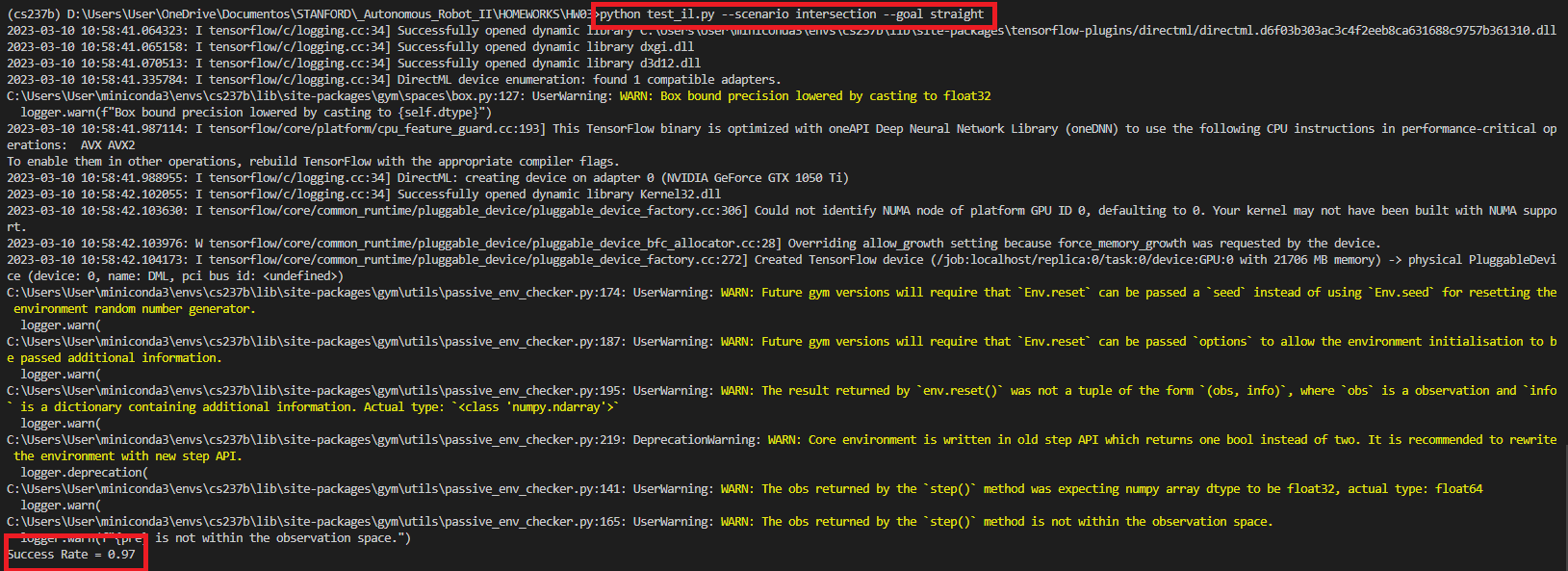


Figure - Straight Success Rate.

(v) The L2 loss function was used with different penalties factors for “steering” and “throttle”.

If the data perfectly overfit the minimum loss = 0.0. it’s bounded below because isn’t possible the L2 norm error be less than zero.

(vi) Some of the properties of covariance matrix:

* A covariance matrix is always symmetric, implying that the transpose of a covariance matrix is always equal to the original matrix.
* A covariance matrix is always positive and semi-definite.
* The eigenvalues of a covariance matrix are always real and non-negative (from second property).

One of the ways to ensure that covariance matrix needs to be a PSD could be done applying Cholesky decomposition in PSD covariance matrix and get the lower triangular matrix where the . If the Cholesky decomposition is not possible, therefore the matrix is not a PSD matrix.

(vii) The problem is structured bellow, for bivariate normal distribution:

That can be written,

Then the negative Mean Log-Likelihood,

Where is the determinant of covariance matrix.

If we perfectly overfit the data in training process, we have the statement, , then the minimum loss is,

The determinant of covariance matrix is always positive but could be less than one and the natural logarithm become a negative scalar, therefore the result is not bounded below, we have the case where: , when .

(ix) Full training process:

* First, we initialize the weights using Xavier Initialization in the neural network layers (weights and bias), using 3 layers NN.
* The number of epochs for “left” and “right” was changed in the interval of 100 to 500 and learning rate between 0.003 and 0.0001, the command *--restore* was used approximately five times. For “straight” the number of epochs was 500 and learning rate of 0.003:
  + For each batch proceed the forward pass was (*call()* function) and the parameters of distribution was evaluated
  + The loss function is evaluated (negative log-loss) from forward pass results

The process is repeated for each epoch and inner batches.

(x)

* “left”: Success Rate = 0.98
* “right”: Success Rate = 0.85
* “straight”: 0.83

Texto

Descrição gerada automaticamente

Figure - Left Success Rate.

Uma imagem contendo Texto

Descrição gerada automaticamente

Figure - Right Success Rate.

Uma imagem contendo Texto

Descrição gerada automaticamente

Figure - Straight Success Rate.

**Problem 3: Conditional Imitation Learning.**

(i) Training the neural network to all goals we can verify that the car actions are undefined/random. The car in the test results go out of the road when approaches to the intersection, the input data is not sufficient to determine the car action (left, right or straight). Another point is that the car is out of user’s control, applying this strategy in an autonomous car system is unsafety.

(iii)

* “left”: Success Rate = 0.95
* “right”: Success Rate = 0.96
* “straight”: Success Rate = 0.75

Texto

Descrição gerada automaticamente

Figure - Left Success Rate.

Texto

Descrição gerada automaticamente com confiança média

Figure - 4 - Right Success Rate.

Texto

Descrição gerada automaticamente

Figure 9 - Straight Success Rate.

(iv) Full training process:

* First, we initialize the weights using Xavier Initialization (weights and bias) for the input neural network (3 layers) and 3 branches neural networks regarding to each goal (3 layers).
* For each epoch = 1000 and learning rate = 1e-4:
  + For each batch the forward pass (*call ()* function) using the pair (o, u) and the output of input NN is the input for the Branches NN. The weighted sum of Branches generates the y (action) estimated.
  + The loss function is evaluated (L2 loss) from predict action and action from samples. Different weights for steering and throttle was used, therefore the loss function was split in two branches, one for steering and another for throttle, the full loss is the summation.
  + The gradient and Optimizer were applied, Adam optimizer is used. The parameters () are updated and learning rate () is adapted over time by:

The process is repeated for each epoch and inner batches.

**Problem 4: Intent Inference & Shared Autonomy.**

(i)