How to create your first AWS DeepRacer model

Welcome to AWS DeepRacer, a 1/18th scale race car which gives you an exciting and fun way to get started with reinforcement learning (RL). AWS DeepRacer provides an online virtual simulator in the AWS console where you can create, train, and tune the RL models needed for autonomous car racing. You can also evaluate the performance of your models in the simulator and then deploy them to AWS DeepRacer for a real-world autonomous experience. In this guide we will introduce you to the simulator and get you started with creating your first reinforcement learning model.

To complete this guide you only need an AWS account and an IAM user, you do not need to own an AWS DeepRacer car. If you have an AWS Account and IAM user set up please skip to the next section, otherwise please continue reading.

Create an AWS account and an IAM user

To use AWS DeepRacer you need an AWS account. The AWS account is free. You only pay for the AWS services that you use.

To sign up for an AWS Account follow these steps

1. Please go to https://portal.aws.amazon.com.
2. Choose Create a Free Account
3. Follow the instructions on the page.
   Part of the sign-up process involves receiving a phone call and entering a PIN using the phone keypad.

If successful, the above sign-up procedure creates a root account as identified by the provided email address. Although you can use this root account to sign in to AWS, you're recommended to create an IAM user with administrative privileges for you to sign in to AWS to manage your account. You can also create IAM users with more restrictive permissions for your team members to call specified AWS services under your account.

To create an IAM User follow these steps

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Users, then choose Add user.
3. For Access type, choose both Programmatic Access to call APIs of permitted AWS services and AWS Management Console Access to launch the AWS Management Console for permitted services.
4. For Console password, choose Autogenerated password or Custom password. If you choose Custom password, type a password.
5. Choose whether to require the user to reset the password at the next sign-in, then choose Next: Permissions.
6. For Set permissions for <user name>, choose Attach existing policies directly, AdministrativeAccess, and Next: Review.
7. Review the settings. To return to the previous page to make changes, choose Previous. To create the user, choose Create user.

Congratulations, now that you have an AWS account and IAM user, you're ready to explore AWS DeepRacer and all other AWS services!

What is reinforcement learning?

Reinforcement learning (RL) is a branch of machine learning interested in the creation of a model that can be used by an agent to choose which actions to take in an environment in order to achieve a specific goal. In the AWS DeepRacer simulator the agent is the virtual car, the environment is a virtual racetrack, the actions are throttle and steering inputs to the car, and the goal is completing the racetrack as quickly as possible and without deviating from the track. As DeepRacer drives around the simulated racetrack it sends pictures of the environment, called states, to the RL model and receives back the best action to take from the current state. Initially the untrained RL model will choose actions at random allowing the car to explore the environment, but over time as it repeatedly observes which actions are better, it will learn and start preferring actions that result in better outcomes. It then starts exploiting this knowledge to repeatedly take the best action in each state.

How does the car know which actions are better, and how does it know what the final goal is? The reward function answers both questions. For each action that the car takes, it will receive a reward based on the outcome of the action. You have to specify a reward function that rewards the car for actions that lead to preferred outcomes, and if the car repeatedly lands in a preferred outcome it should reach your goal. The reward function contains the "logic" to determine if an action was good or bad, and quantify it. Thus creating a good reward function is vital in ensuring the model learns the right behavior, and ultimately reaches the goal. For example, if your goal is for the car to finish a lap around a racetrack, rewarding the car for staying on the track, as opposed to going off the track, is a good starting point.

The RL optimization algorithm trains the model which actions to choose to maximize the cumulative reward that the agent can achieve from any state. The cumulative reward in a state is the sum of all rewards received from the immediate action and each subsequent action, until a termination state (such as finishing the track, or going off track) is reached. The best action in each state is the one that yields the maximum cumulative reward possible, assuming the agent keeps on choosing the best action in each subsequent state.

Training is an iterative process alternating between using the latest model to obtain experience and updating the model using the latest experience. During the experience gathering phase the outcome of each step is stored in the form of a (state, action, next state, reward) tuple. An episode refers to all the steps taken from the car's starting state to the next termination state. After a user-specified number of episodes has run training will start, at which point a number of samples (referred to as batch size) will be pulled from the most recent experience and used to update the model. Once the model is updated the car will start exploring again using the latest version of the model.

Let's now switch gears and create your first AWS DeepRacer RL model.

Create your first AWS DeepRacer RL model

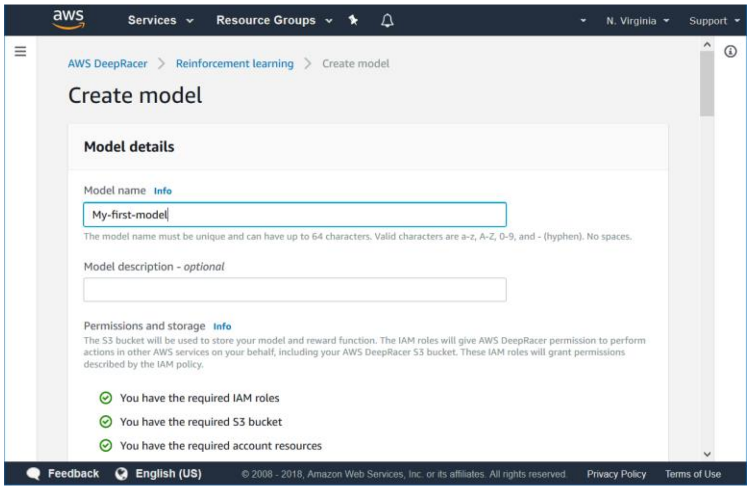
You can apply for whitelist access to AWS DeepRacer at https://aws.amazon.com/deepracer. If you have been whitelisted please log into the AWS Console, choose US East (N Virginia) as your Region, and search for DeepRacer or go to this address https://aws.amazon.com/deepracer and Choose **Create Model**.

Note that if you want to read up on any terms used in the console please make use of the Info buttons when available. The Info buttons will slide an information section onto the right-hand-side of the page and won't interrupt your current flow. Also feel free to consult the AWS DeepRacer Developer Documentation https://docs.aws.amazon.com/deepracer/latest/developer/.

Model details

1. Provide a model name
2. Provide a model description
3. Choose **Create resources.** This will set up the IAM roles to grant AWS DeepRacer permission to call other AWS services on your behalf. AWS Amazon SageMaker for reinforcement learning model training, AWS RoboMaker to provide the racing simulator, Amazon Kinesis Video Streams for video streaming of virtual simulation footage, Amazon S3 for model storage, and Amazon CloudWatch for log capture. If successful you should see a message similar to the one shown at the bottom of the following Model Details image.

Model Details



Create model

Model details

Model name Info

My-first-model

The model name must be unique and can have up to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces.

Model description - *optional*

Permissions and storage Info

The S3 bucket will be used to store your model and reward function. The IAM roles will give AWS DeepRacer permission to perform actions in other AWS services on your behalf, including your AWS DeepRacer S3 bucket. These IAM roles will grant permissions described by the IAM policy.

✓ You have the required IAM roles

✓ You have the required S3 bucket

✓ You have the required account resources

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Environment simulation

This section allows you to select the track on which you want to train your model.

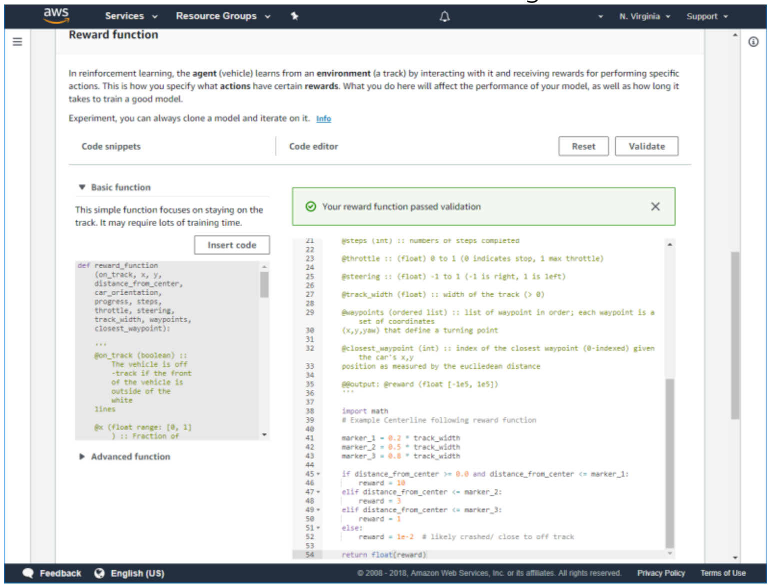
1. Please choose **re:Invent 2018**

Reward function

The reward function section allows you to program your own reward function using Python syntax and the variables exposed from the simulator. The variables provide information on the agent such as its location and orientation within the simulation environment. Using these variables you can create your own logic to determine if an outcome was good or bad, and how much to reward the outcome. For now we will use the basic reward function. The basic reward function rewards your car for staying on the track, and will provide a higher reward if your car stays closer to the middle of the track.

1. Expand the **Basic** Function
2. Choose **Insert Code**, to insert the sample basic code
3. Choose **Validate** to confirm that the code is error free

Reward Function Configuration



Reward function

In reinforcement learning, the **agent** (vehicle) learns from an **environment** (a track) by interacting with it and receiving rewards for performing specific actions. This is how you specify what **actions** have certain **rewards**. What you do here will affect the performance of your model, as well as how long it takes to train a good model.

Experiment, you can always clone a model and iterate on it. Info

Code snippets | Code editor | Reset | Validate

▼ Basic function

This simple function focuses on staying on the track. It may require lots of training time.

Insert code

```
def reward_function
    (on_track, x, y,
    distance_from_center,
    car_orientation,
    progress, steps,
    throttle, steering,
    track_width, waypoints,
    closest_waypoint):

    '''
    @on_track (boolean) ::
        The vehicle is off
        -track if the front
        of the vehicle is
        outside of the
        white
    lines

    @x (float range: [0, 1]
        ) :: Fraction of
```

▶ Advanced function

Your reward function passed validation

```
    @steps (int) :: numbers of steps completed

    @throttle :: (float) 0 to 1 (0 indicates stop, 1 max throttle)

    @steering :: (float) -1 to 1 (-1 is right, 1 is left)

    @track_width (float) :: width of the track (> 0)

    @waypoints (ordered list) :: list of waypoint in order; each waypoint is a
        set of coordinates
    (x,y,yaw) that define a turning point

    @closest_waypoint (int) :: index of the closest waypoint (0-indexed) given
        the car's x,y
    position as measured by the eucliedean distance

    @@output: @reward (float [-1e5, 1e5])
    '''

    import math
    # Example Centerline following reward function

    marker_1 = 0.2 * track_width
    marker_2 = 0.5 * track_width
    marker_3 = 0.8 * track_width

    if distance_from_center >= 0.0 and distance_from_center <= marker_1:
        reward = 10
    elif distance_from_center <= marker_2:
        reward = 3
    elif distance_from_center <= marker_3:
        reward = 1
    else:
        reward = 1e-2  # likely crashed/ close to off track

    return float(reward)
```

Algorithm settings

The algorithm setting control the hyperparameters used by the RL Optimization Algorithm. The default values are good starting points for each parameter and for now we will not change them. The following table outlines how each of the parameters impact how our model is trained. In our case the model is a neural network.

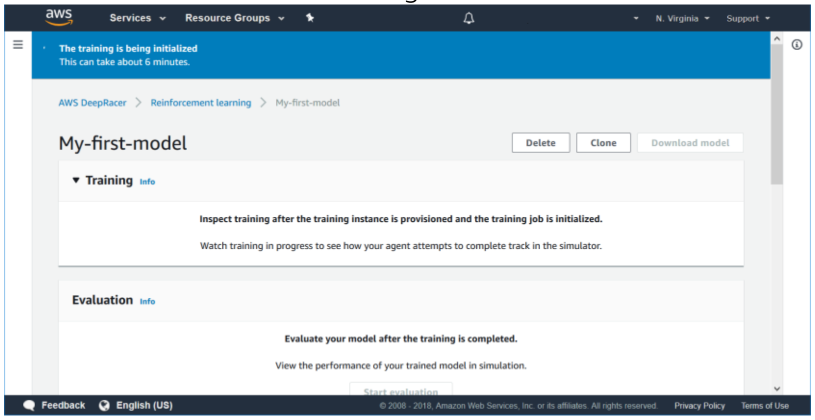
Hyperparameter	Impact on training
Batch size	Batch size determines the number of steps, randomly sampled from the most recent experience, we will use to train our network. Thus it is the amount of experience you use to update your network.
Number of Epochs	This is the number of times that our optimization algorithm will run through the batch and update the weights of our network, each time that training is performed.
Learning rate	The learning rate determines the size of the updates to network weights.
Exploration	This is the method used to provide the exploration/exploitation trade-off. Thus to what extent will your model explore vs exploit.
Entropy	Entropy controls the amount of exploration, by adding noise to the probability distribution of actions. Smaller entropy implies less exploration.
Discount factor	This determines how far forward the model should look in time to value its actions. In our basic environment 0.9 looks about 20 to 30 steps ahead, and it can take 100s of steps to make a turn.
Loss type	Loss functions used during our weight update process.
Number of episodes between each training	This determines how much experience we want to obtain, in the form of episodes, before we run training. After each training step, we will use the newly trained model to obtain more episodes and iteratively continue the process.

Stop conditions

1. Specify the **max. time** as 150 minutes. Don't worry you can always stop it before 150 minutes elapse. Furthermore, if you are not satisfied with the model you can clone it from the Reinforcement Learning menu and restarting training with new parameters.

You are now ready to start training your model, please choose **Start training**. Over the next 6 minutes the AWS DeepRacer service will orchestrate various AWS Services to create the virtual simulator in which your RL model will be trained. The blue bar at the top of the screen should indicate whether the instances are still being provisioned or whether training has started. Once training starts, select your model from the list under Reinforcement learning. This will open up the model detail page.

Model training initialization



The training is being initialized
This can take about 6 minutes.

AWS DeepRacer > Reinforcement learning > My-first-model

My-first-model

Training Info

Inspect training after the training instance is provisioned and the training job is initialized.

Watch training in progress to see how your agent attempts to complete track in the simulator.

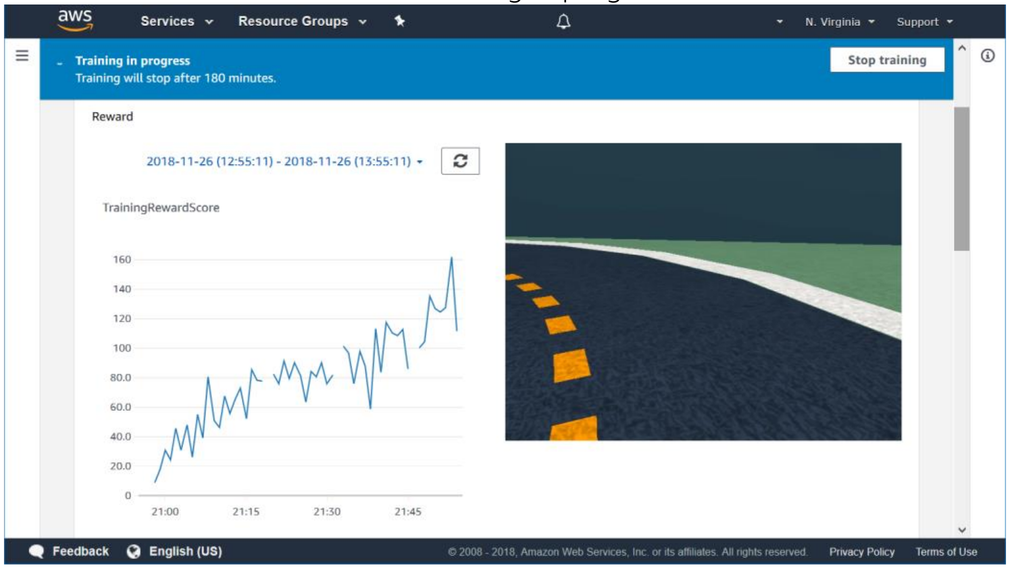
Evaluation Info

Evaluate your model after the training is completed.

View the performance of your trained model in simulation.

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Model training in progress



Training in progress
Training will stop after 180 minutes.

2018-11-26 (12:55:11) - 2018-11-26 (13:55:11)

TrainingRewardScore

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Model training

Once your model starts training you should see a TrainingRewardScore graph showing the cumulative reward your agent receives for each episode. An episode is the number of steps from the car's starting point to the next termination state. Termination states are specified as going off-track, or when the car stays on track and reaches 1,000 steps. As your model trains you should see the reward line increase, however expect it to fluctuate initially. Determining when your model has converged is an art. We would advise you to observe training and consider stopping as soon as you see the car in the simulator gets close to completing two laps. The risk is that you over-train the model and while this will produce good results in the simulator, the model may not perform well when downloaded onto AWS DeepRacer for a real-world race. In the event that your car is not completing the virtual track and your model stops improving, as judged by the reward graph or the video, consider stopping the training and proceed to tweak your model.

Congratulations, you have now created your first AWS DeepRacer model and can proceed to evaluate it, or deploy it to your AWS DeepRacer.