How to improve your model with an advanced reward function

In this guide we will discuss how you can create better performing models by using advanced reward functions. The RL optimization algorithms rely on the reward function to help determine the best action to take in each state. An advanced reward function helps your model to better differentiate between good and bad actions, as it can better assess the outcome of actions. If you have not yet created a model with a basic reward function, consider following the steps in the How to Create Your First Model, otherwise please continue reading.

To create a model with an advanced reward function please log into the DeepRacer console, navigate to Reinforcement learning, choose **Create model**, and complete each of the sections.

Model details

1. Provide a model name
2. Provide a model description
3. If you have not yet performed the **Create resources** step, please do so now.

Environment simulation

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

Reward function

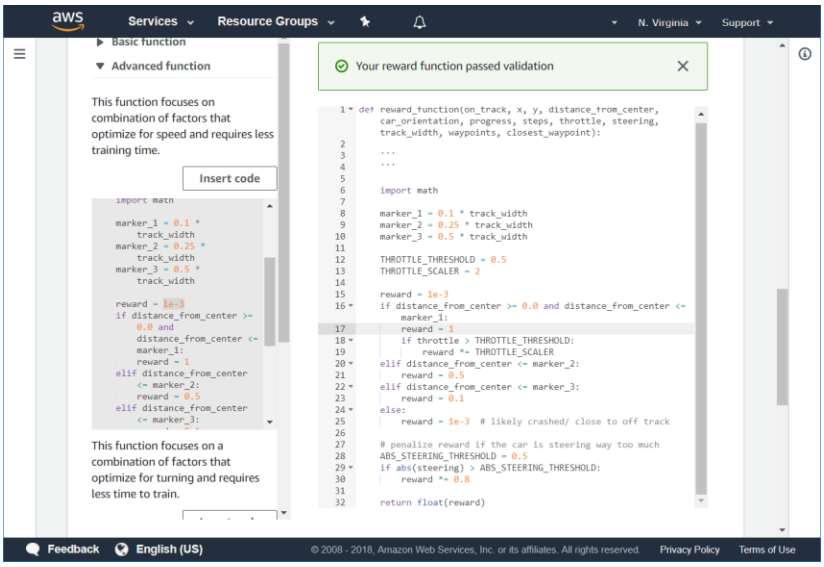
In this section we will make use of an advanced reward function. By default the code editor displays a basic reward function written in Python3. The basic function contains a list of variables which are observed from the simulator after each action. You can use these variables in your reward function logic, to reward the car based on the outcome of its actions. The RL optimization algorithm will try to maximize the cumulative reward achievable from each state by choosing the appropriate actions, so your reward function directly impacts the behavior of your model.

The variables you can use are:

Variable Name	Type	Description
on_track	Boolean	If the front of the vehicle is in-between the white track lines, the vehicle is on-track.
x	Float, range [0,1]	Fraction indicating location of car along the x-axis. 0 indicates minimum, and 1 indicates maximum x value in the coordinate system.
y	Float, range [0,1]	Fraction indicating location of car along the y-axis. 0 indicates minimum, and 1 indicates maximum y value in the coordinate system.
distance_from_center	Float, [0, track_width/2]	Displacement from the center line of the track as defined by way points.
car_orientation	Float, range [-3.14,3.14]	Yaw of the car with respect to the car's x-axis in radians i.e. (-180°,180°)
progress	Float, range [0,1]	% of track complete
steps	Integer	Number of steps completed
throttle	Float, range [0,1]	0 indicates stop, 1 max throttle
steering	Float, range [-1,1]	-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	Integer	Index of the closest waypoint (0-indexed) given the car's x, y position as measured by the Euclidean distance.
reward	Float, range [-1e5,1e5]	The reward you calculate. Please keep it in the range, and try avoid getting zero rewards.

1. Expand the **Advanced** Function, and choose **Insert Code** for the first example
   This provides an example of how you can penalize the car if it is steering too much, based on some steering threshold that we decide. The car takes roughly 10 images per second, and each image is a state that is used to determine new steering and throttle inputs. By penalizing excessive steering we want to incentivize smooth driving, and avoid the car from potentially learning to turn maximum left and right on alternating states. Assume now we also want to reward the car for driving fast when it is close to the middle of the track. We can simply scale our reward function if the throttle is above some threshold we specify.

Advanced reward function



To get a good reward function you have to experiment with many possibilities and build an intuition for the outcomes you want to incentivize and how to reward the model to reach these outcomes. Visually inspecting driving behavior, using the simulator, the AWS DeepRacer, or both, will also help you build a better intuition on the behaviors to reward and the behaviors to penalize.

The rest of the steps will be used to finish model creation and start training.

Algorithm settings

1. Leave all as default

Stop Conditions

1. Specify the maximum training time as 120 minutes. Don't worry you can always stop it before 120 minutes pass. 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 training environment 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. This will open up the model detail page. Here you will be able to track the model training and visually inspect how your car trains over time. 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 reward function.