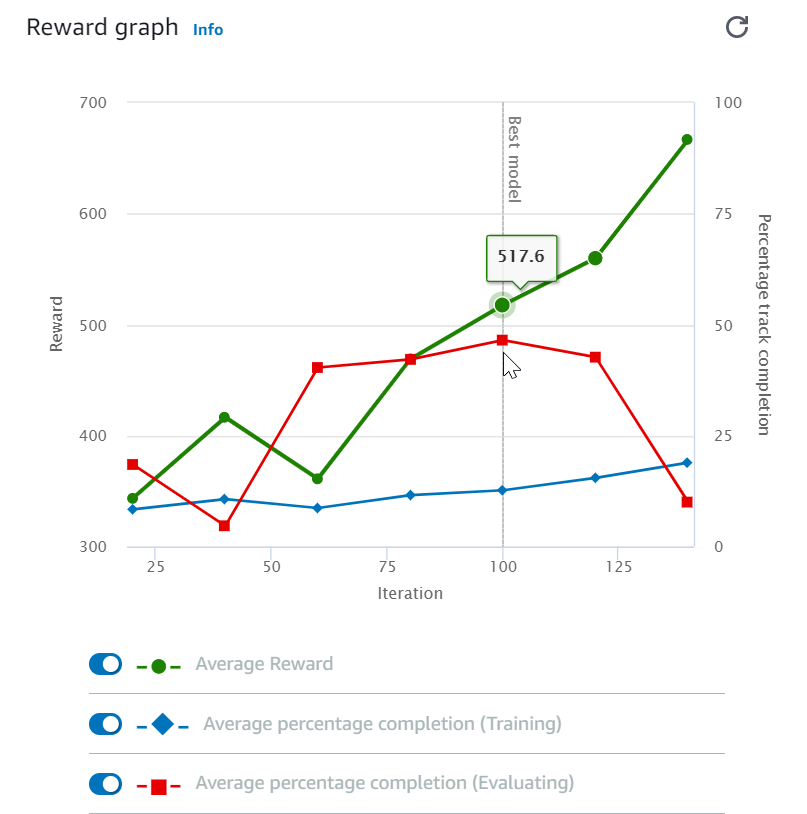
**Exercise Solution**

To get a sense of how well your training is going, watch the reward graph. Here is a list of its parts and what they do:

* **Average reward**
  + This graph represents the average reward the agent earns during a training iteration. The average is calculated by averaging the reward earned across all episodes in the training iteration. An episode begins at the starting line and ends when the agent completes one loop around the track or at the place the vehicle left the track or collided with an object. Toggle the switch to hide this data.
* **Average percentage completion (training)**
  + The training graph represents the average percentage of the track completed by the agent in all training episodes in the current training. It shows the performance of the vehicle while experience is being gathered.
* **Average percentage completion (evaluation)**
  + While the model is being updated, the performance of the existing model is evaluated. The evaluation graph line is the average percentage of the track completed by the agent in all episodes run during the evaluation period.
* **Best model line**
  + This line allows you to see which of your model iterations had the highest average progress during the evaluation. The checkpoint for this iteration will be stored. A checkpoint is a snapshot of a model that is captured after each training (policy-updating) iteration.
* **Reward primary y-axis** 
  + This shows the reward earned during a training iteration. To read the exact value of a reward, hover your mouse over the data point on the graph.
* **Percentage track completion secondary y-axis** 
  + This shows you the percentage of the track the agent completed during a training iteration.
* **Iteration x-axis**
  + This shows the number of iterations completed during your training job.



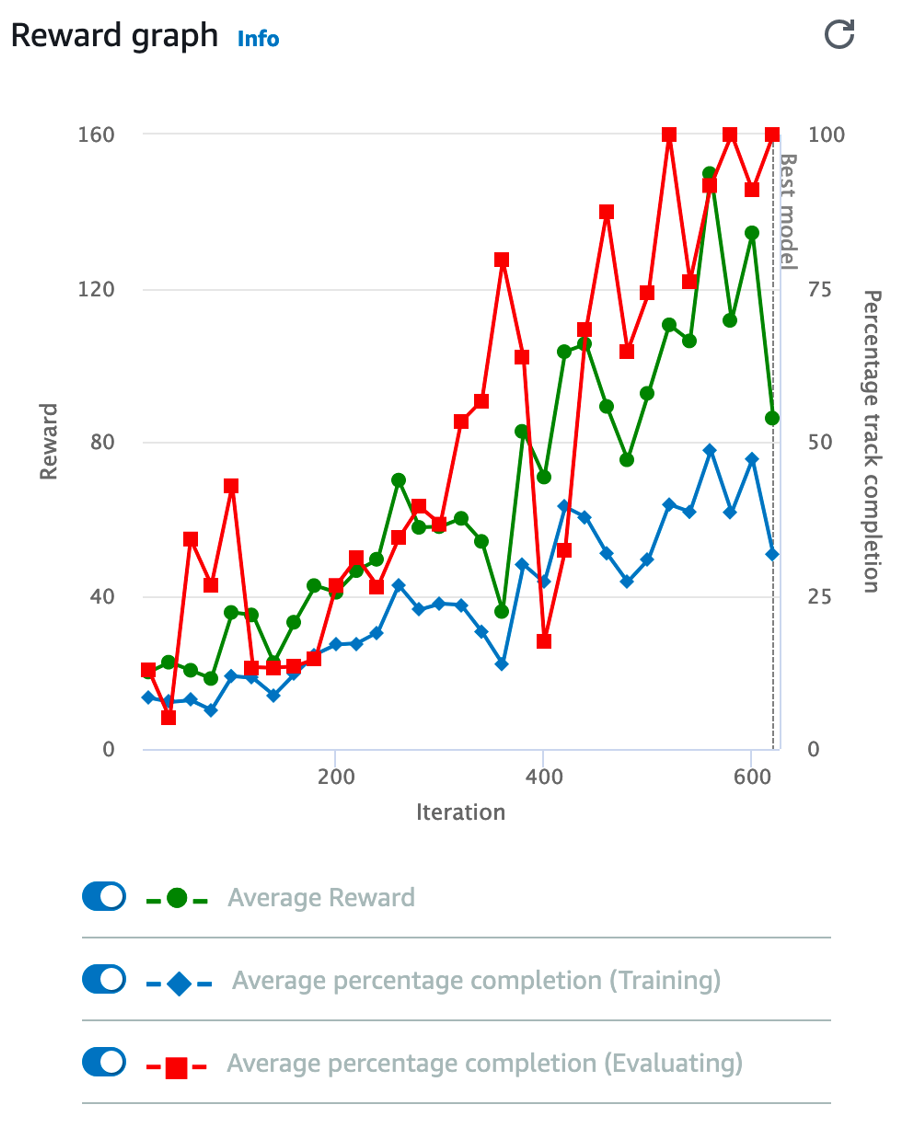
List of reward graph parts and what they do

**Reward Graph Interpretation**

The following four examples give you a sense of how to interpret the success of your model based on the reward graph. Learning to read these graphs is as much of an art as it is a science and takes time, but reviewing the following four examples will give you a start.

**Needs more training**

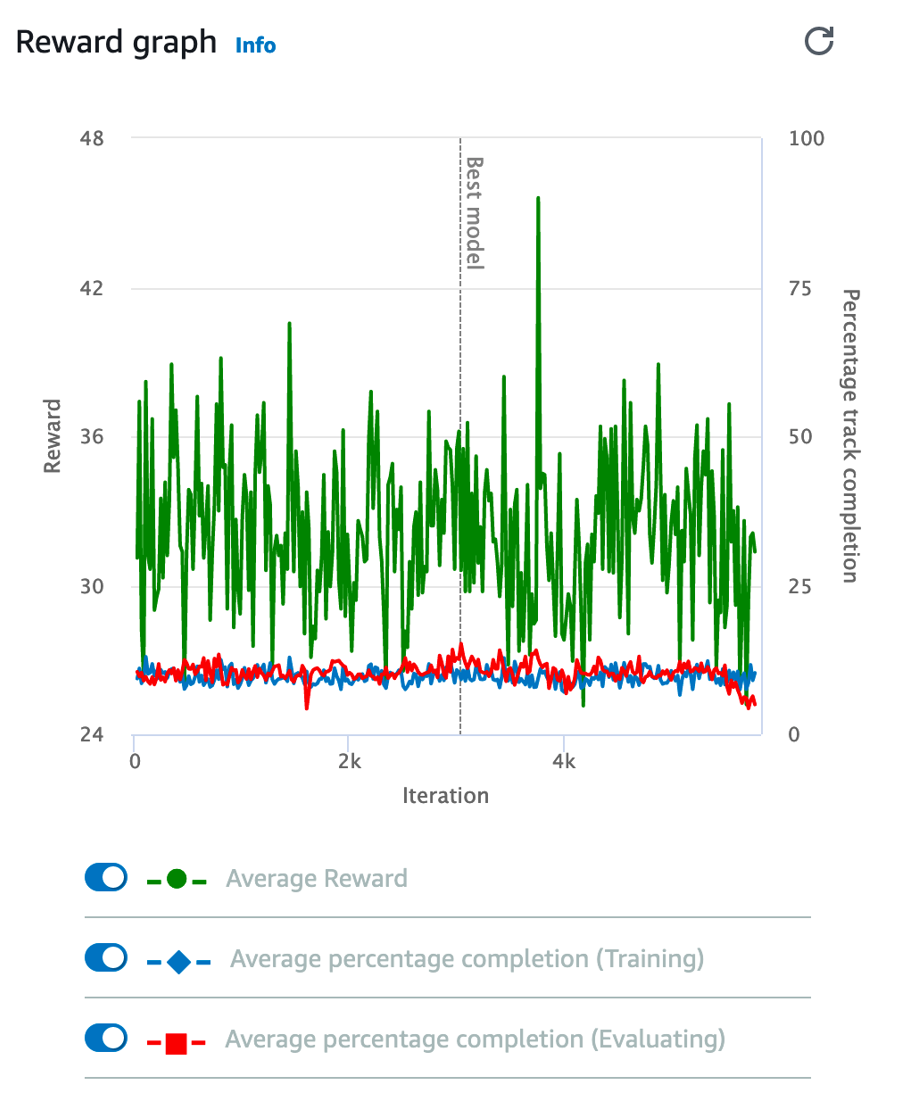
In the following example, we see there have only been 600 iterations, and the graphs are still going up. We see the evaluation completion percentage has just reached 100%, which is a good sign but isn’t fully consistent yet, and the training completion graph still has a ways to go. This reward function and model are showing promise, but need more training time.



Needs more training

**No improvement**

In the next example, we can see that the percentage of track completions haven’t gone above around 15 percent and it's been training for quite some time—probably around 6000 iterations or so. This is not a good sign! Consider throwing this model and reward function away and trying a different strategy.



No improvement

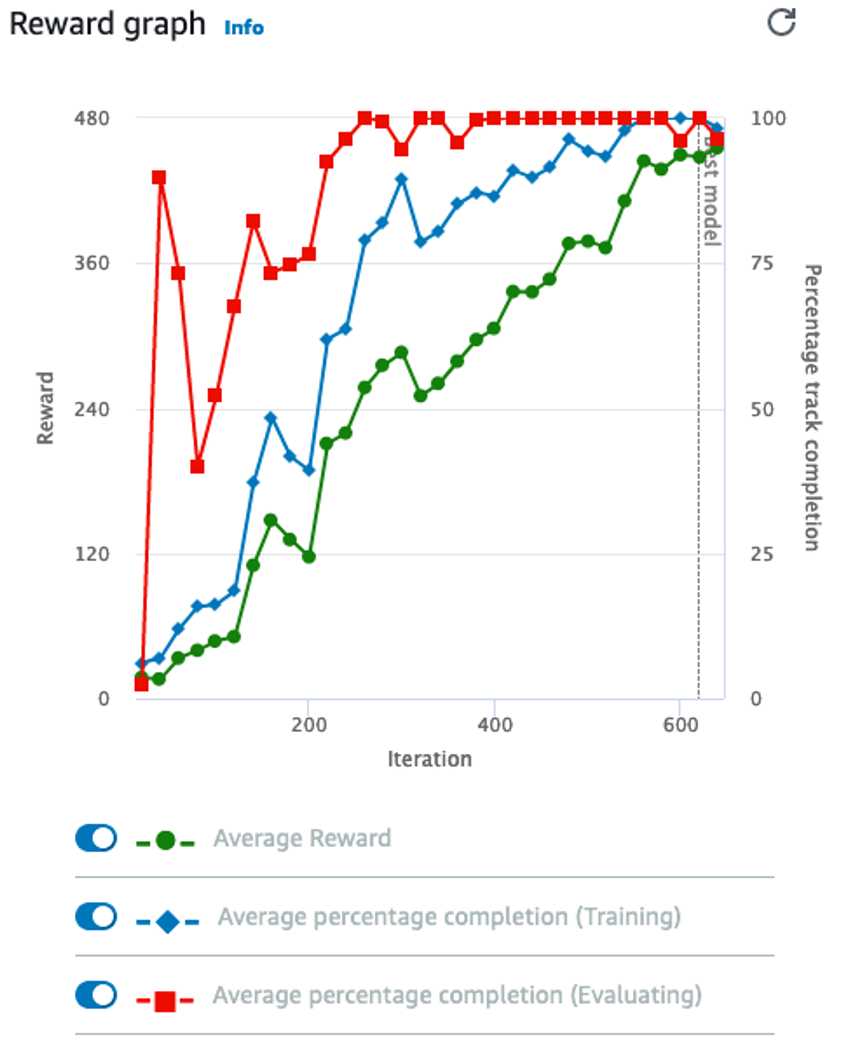
**A well-trained model**

In the following example graph, we see the evaluation percentage completion reached 100% a while ago, and the training percentage reached 100% roughly 100 or so iterations ago. At this point, the model is well trained. Training it further might lead to the model becoming overfit to this track.

**Avoid overfitting**

**Overfitting** or **overtraining** is a really important concept in machine learning. With AWS DeepRacer, this can become an issue when a model is trained on a specific track for too long. A good model should be able to make decisions based on the features of the road, such as the sidelines and centerlines, and be able to drive on just about any track.

An **overtrained** model, on the other hand, learns to navigate using landmarks specific to an individual track. For example, the agent turns a certain direction when it sees uniquely shaped grass in the background or a specific angle the corner of the wall makes. The resulting model will run beautifully on that specific track, but perform badly on a different virtual track, or even on the same track in a physical environment due to slight variations in angles, textures, and lighting.

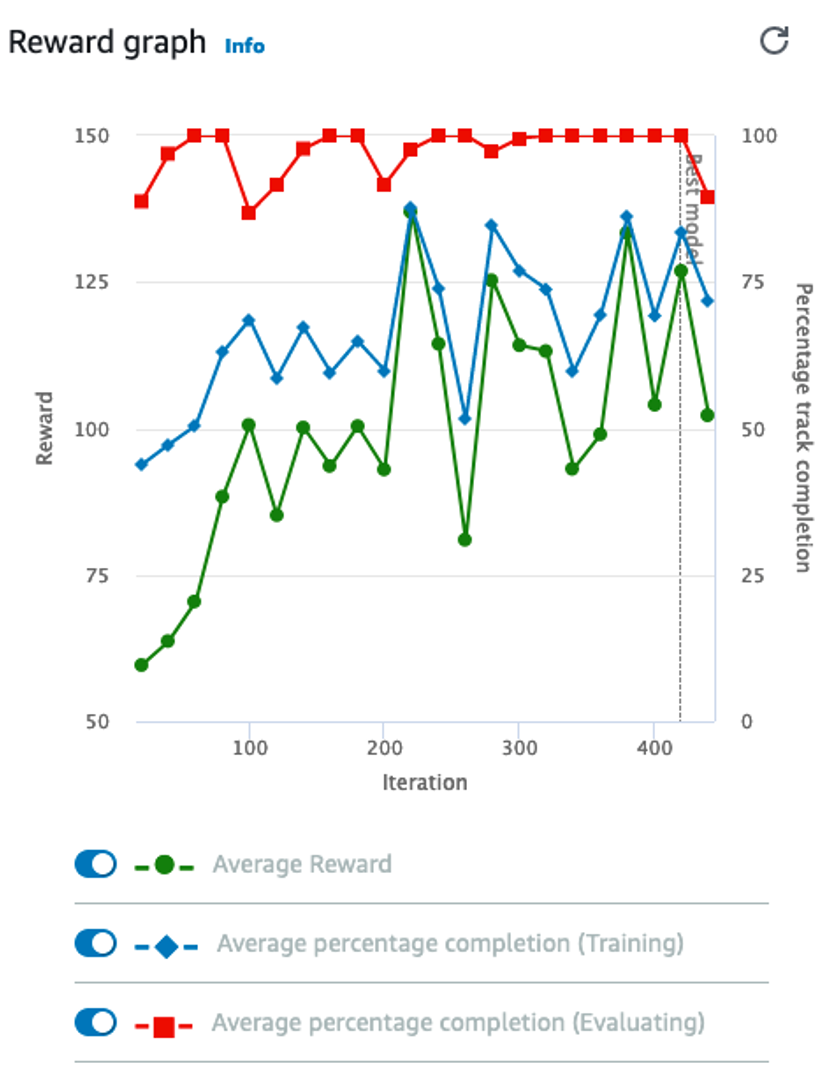


Well-trained - Avoid overfitting

**Adjust hyperparameters**

The AWS DeepRacer console's default **hyperparameters** are quite effective, but occasionally you may consider adjusting the training hyperparameters. The **hyperparameters** are variables that essentially act as settings for the training algorithm that control the performance of your agent during training. We learned, for example, thatthe **learning rate** controls how many new experiences are counted in learning at each step.

In this reward graph example, the training completion graph and the reward graph are swinging high and low. This might suggest an inability to converge, which may be helped by adjusting the learning rate. Imagine if the current weight for a given node is .03, and the optimal weight should be .035, but your learning rate was set to .01. The next training iteration would then swing past optimal to .04, and the following iteration would swing under it to .03 again. If you suspect this, you can reduce the learning rate to .001. A lower learning rate makes learning take longer but can help increase the quality of your model.



Adjust hyperparameters

**Good Job and Good Luck!**

Remember: training experience helps both model and reinforcement learning practitioners become a better team. Enter your model in the monthly [AWS DeepRacer League](https://aws.amazon.com/deepracer/league/?utm_source=Udacity&utm_medium=Webpage&utm_campaign=Udacity%20AWS%20ML%20Foundations%20Course) races for chances to win prizes and glory while improving your machine learning development skills!

;