서버=AWS EC2 Instance Seoul

디렉토리=bitnami@ip-10-0-1-184:~/aws\_polly/polly4bona$

메인 스크립트=run을 고쳐서 만듬.

명령어:

$ cp run run[id]

$ nano run[id]

1. 안에서 파일명을 id로 설정

2. Text값에 아래의 text를 복사

$ ./run[id]

**성공적으로 TTS가 만들어진 경우**

$ ./run13

aws polly synthesize-speech --output-format mp3 --voice-id Joey --text 'Slide 30 for 2 min. [추가text 내용] Thank you and Q&A for 9 mins.' rl4bona-13.mp3

/home/bitnami/.local/lib/python2.7/site-packages/urllib3/util/ssl\_.py:354: SNIMissingWarning: An HTTPS request has been made, but the SNI (Server Name Indication) extension to TLS is not available on this platform. This may cause the server to present an incorrect TLS certificate, which can cause validation failures. You can upgrade to a newer version of Python to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings

SNIMissingWarning

{

"ContentType": "audio/mpeg",

"RequestCharacters": "500"

}

**실패한 경우**

$ ./run7

aws polly synthesize-speech --output-format mp3 --voice-id Joey --text 'Slide 18 for 2 minutes. [추가 text 내용]' rl4bona-3.mp3

/home/bitnami/.local/lib/python2.7/site-packages/urllib3/util/ssl\_.py:354: SNIMissingWarning: An HTTPS request has been made, but the SNI (Server Name Indication) extension to TLS is not available on this platform. This may cause the server to present an incorrect TLS certificate, which can cause validation failures. You can upgrade to a newer version of Python to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings

SNIMissingWarning

An error occurred (TextLengthExceededException) when calling the SynthesizeSpeech operation: Maximum text length has been exceeded

$

관련 Command History

420 ssh ubuntu@59.10.6.134

421 ls

422 cd aws\_polly/

423 ls

424 cp run\_polly run\_polly4rl

425 nano run\_polly4rl

426 mkdir polly4bona

427 cd polly4bona/

428 ls

429 nano run

430 ls

431 nano run

432 ./run

433 chmod +x run

434 ./run

435 nano run

436 ./run

437 ls

438 mv run run1

439 ls

440 cp run1 run2

441 nano run2

442 ls

443 cp run2 run

444 ls

445 nano run2

446 ./run2

447 nano run

448 cp run run2

449 nano run2

450 ./run2

451 ls

452 cp run run3

453 nano run3

454 ./run3

455 ls

456 cp run run3

457 nano run3

458 ./run3

459 ls

460 nano run1

461 mv rl4bona.mp3 rl4bona-1.mp3

462 ls

463 cp run run4

464 nano run4

465 ./run4

466 ls

467 nano run4

468 ./run4

469 ls

470 nano run

471 cp run run5

472 nano run5

473 ./run5

474 cp run run6

475 nano run6

476 ./run6

477 ls

478 cp run run7

479 nano run7

480 ./run7

481 ls

482 cp run run8

483 nano run8

484 ./run8

485 ls

486 cp run run9

487 nano run9

488 ./run9

489 ls

490 cp run run10

491 nano run10

492 ./run10

493 ls

494 nano run run11

495 cp run run11

496 ls

497 nano run11

498 ./run11

499 cp run run12

500 nano run12

501 ./run12

502 ls

503 cp run run13

504 nano run13

505 ./run13

506 ls

507 pwd

508 history

**작성한 Text**

Slide 3. Before we get started, let’s do a quick recap on AWS DeepRacer. Slide 4 for 1 minute. Developers told us they love the hands-on learning approach used in DeepLens. In 2018 Amazon SageMaker added support for Reinforcement Learning. We asked the question “How can we put Reinforcement Learning in the hands of all developers literally?” And so the idea of AWS DeepRacer was born. Slide 5 for 2 minutes. Before you tell customers they can order/pre-order the car, check if available in your region! Otherwise don’t call it out. At reinvent 2018 we launched AWS DeepRacer. AWS DeepRacer is a 1/18th scale robotic car which gives you an exciting and fun way to get started with reinforcement learning by applying it to autonomous racing. You can order/pre-order your AWS DeepRacer from Amazon today. AWS DeepRacer has a virtual racing simulator that allows you to train, evaluate, and iterate on your RL models in a racing environment, quickly and easily. And if you get really good, and want to showcase your machine learning skills in a competitive environment, there is the AWS DeepRacer league. You can compete in a global championship - racing your DeepRacer model- for a chance to win several prizes and advance to the AWS DeepRacer Championship Cup. Throughout 2019 there will be in person events at selected Aws Summits and a series of virtual competitions hosted in the virtual simulator, giving all developers the ability to compete.

Slide 7 for 1 minute. Machine Learning has three main categories. Supervised learning. Build a model to predict a value or a classify data. Models are trained using large amounts of curated training data, consisting of label and data pairs, to learn. Models learn to accomplish a well defined single task, and don’t scale easily to other tasks or sub-tasks without new data. An example is linear regression. Unsupervised learning. Build a model to classify data. Models are trained to identify similarities in large amounts of data to aid classification. Training data does not have explicit labels. An example is clustering. Reinforcement learning. Build a model to make autonomous decisions in an environment. Models are trained in an environment, by learning from interaction with the environment which decisions/actions led to good outcomes, and which led to bad outcomes. You need to supply the environment and very importantly the reward function to help it determine if an action was good or bad. The example is policy optimization. Slide 8 for 1 minute. Reinforcement learning is built on and idea that is used quite often, perhaps daily, by humans. When was the last time you used a reward to incentivize the right behavior. Think about the method used to train a pet – treats are used to incentivize desired behavior. Training can be for simple actions like sit or stay, to more complicated series of behaviors, like you sometimes see in dog agility courses – crawling through tunnels, jumping over hurdles, and weaving through poles. Slide 9 for 2 minutes. Like using rewards with pets, we will use a reward function to incentivize the desired driving behavior in our agent. What is a reward function? Simply a function helps your agent determine if the action it just took was good or bad and how good or bad. Let’s look at this race grid to see a practical example of a reward function. The blue square is our agent’s starting position, green represents its goal. It can move to every other square, except purple, one square at a time. Each square is called a state, and our agent can choose from the following actions in each state: move forward, left, or right, always facing in the direction of the goal and only visit each state once. This is an example of a “sparse reward function”, it simply says that when the agent reaches the goal it will get a reward of 2. This is like trying to teach your dog to sit, then lie down, then bark, and then shake and only then will you reward them. It will probably take a long time to learn this behavior, if at all. Our agent will also take a long time to learn the right behavior as there are many paths to get to the goal, and it won’t know which one is better.

Slide 10 for 4 minutes. We introduce a reward function that provides a reward for each state. In our example we want to incentivize the agent to follow the centerline because we think that will work best, and so we give higher rewards for actions that result in the agent ending up in the centerline. Why will providing a higher reward in the center incentivize center line driving behavior? Pause. RL algorithms seek to train models that will choose that set of actions, a path so to speak, that maximizes the cumulative expected rewards from any state. Simply put from any state choose the action that will give the highest sum of future rewards. We call this the value function. We also introduce the concept of a discount, to penalize the car for each step taken. Discounting limits how far into the future we look. To build up this value function our agent will explore all state and actions and build a table reflecting the value of being in each state, shown on the right. This value is the maximum cumulative reward achievable from each state when it selects the actions in subsequent states leading to states with highest value. Learning doesn’t just happen in the first go, it takes some iteration because it first needs to explore and see where it can get the highest rewards, before it can exploit that knowledge. How can we improve this? Slide 11 for **2 minutes.** We are interested in the creation of a model that can be used by AWS DeepRacer, our agent, to choose which actions to take from any state in our race track environment in order to reach our goal of completing the track. When we have our trained model the only input into the model is the state, the image from the camera, and the output is the action, the speed and direction to our engine and steering. The reward function is only used during training to help optimize the RL model to choose the actions that lead to the maximum cumulative reward. Once the model is trained there is no way to give feedback on how good or bad an action was, not in the simulator and not in the real world.

Slide 12 for **5 minutes.** As in our grid race we saw we have to first explore the environment to know which actions from which states are the best actions. In the simulator AWS DeepRacer will drive around the track and take 15 pictures per second. Each time it takes a picture we refer to as a step. Each picture represents the state it is in, and using an RL model it will inference which action to take. This will lead to a new state (the outcome of the action) and using the reward function it will then calculate the reward based on the outcome. The process will continue in each state, until we get to a terminal state where the car goes off track or finishes the lap. The car will then reset and repeat the process. Each step is a (state, action, new state, reward) tuple and all steps from the reset point till the terminal state is called an episode. You should think of these episodes as experience, or training data for our model. The initial model used to determine which action to take doesn’t know anything. And so our RL algorithm will initially select actions at random. This is so we can explore the environment. After a set number of episodes, we will train our model and improve it using the experience we built up. The improved model will then be used to collect more experience, after which we will train again and get an even better model. We determine how long we will select random actions over the actions determine by our model. If we start exploiting our new found experience too soon, we may end up missing a better path. Similarly if we keep on randomly choosing actions to explore more, we may end up taking a very long time to train a model. If you can explore all state, action combinations you will end up with a value function, like in our grid race. We can then introduce a policy function. A policy function is a mapping from a state to an action. A simple policy once we know the value function is to say always choose the action that will end up in the state with the highest value. So this is easy when you can explore every possible state/action combination – just build a lookup table like in our grid race track example. BUT for many environments, like in AWS DeepRacer, you can’t explore every state/action combination and so you can’t fully determine the value function or a policy function. Two methods we use are Value Approximation and Policy Optimization.

Slide 13 for **4 minutes.** Let’s look at an example of policy optimization – vanilla policy gradient. This is a method where we parameterized our policy function, the parameters are simply the weights in a neural network, and the neural network represents our policy function. All this policy function does is take an image as input and outputs an action. Thus mapping state to action. We then optimize that policy to get the best action from each state. Our goal is to get the maximum cumulative reward. We train our model we update the weights by trying to maximize the cumulative future reward, and in doing so we give higher probability to the action that leads to the higher cumulative future reward. DeepRacer uses a form of Policy Optimization, called Proximal Policy Optimization. Slide 14 for **4 minutes.** On this slide we provide a high level overview of the whole training process when using PPO. On the left we have our simulator using the latest policy (model) to get new experience. The experience is fed into an experience replay buffer which feeds our Proximal Policy Optimization algorithm. While PPO is a policy optimization method that uses an actor critic method to learn. **Actor Updates Policy using Clipped updates.** We clip updates top prevent the policy from being updated too much, a common issue with policy optimization methods. Typically we keep the ratio of new and old policy in [0.8, 1.2]. **Critic tells actor h**ow good the action taken was and how the actor should adjust its network. Once the policy is updated the new policy is sent to get more experience.

Slide 15 for **2 minutes.** RL agent gets training data by itself in simulated environment. RL algorithm. Behavioral Cloning – expert driver controls car and logs images from car and steering input. Supervised learning. We will take about how to bridge the simulation to real domain transfer challenge later. Slide 16 for 0.5 minutes. The previous section should have taken 29 minutes. You have 91 minutes left or 1hr and 31 minutes. For the remainder of this talk we will be focusing on the virtual simulator and what you should expect when building your reinforcement learning models. **Slide 17 for 3 min.** Just a quick warm up lap before the race starts. AWS DeepRacer uses Amazon SageMaker to train the RL models, AWS RoboMaker to provide the simulation environment, Amazon S3 to store models, Amazon CloudWatch to store logs, and Amazon Kinesis Video Stream to display the video in the console. The service is built on top of other AWS services. When you start training a model in AWS DeepRacer the following happens. AWS DeepRacer starts an Amazon SageMaker container, and an AWS RoboMaker container in your service account and links the two. It then passes the right parameters to start the training. The experience (state, action, new state, reward) tuples are generated in AWS RoboMaker and after a specified amount of experience is obtained it is sent back to Amazon SageMaker to train the model. The new model is then sent back to AWS RoboMaker to get more experience, and so the process continues. The outputs (models, video, and metrics) are stored in other AWS services in your account. By logging into each of these services you will see the data created from your DeepRacer training jobs.

Slide 18 for **2 minutes.** In the AWS DeepRacer console, 1. Create a model, 2. Configure training. From our reinforcement learning overview we saw that the reward function is a pretty important item when it comes to reinforcement learning. And so we will pay close attention to the reward function. Our reinforcement learning algorithm has certain hyperparameters (such as after how many episodes of experience we update the model) that we need to specify, and we also need to provide the actions from which it should choose. 3. Train the model. 4. Evaluate the model, how good is your model? If you are happy go race in the League (virtual or in person) or deploy to the car. 5. Tweak mode. Slide 19 for **2 minutes.** Lets look at the reward function again. During training we take an action in each step and update the position of the car. The reward function will be used to determine how good or bad the outcome of the action is. How would the reward function know if the outcome is good or bad? **Pause.** You supply the logic for it to determine how good or bad, and quantify it. And you get variables containing measurements from the simulator after each action that you can use to build reward function logic. You do this in python 3 syntax. For example, the simulator tells you how close your car is from the center of the track and how wide the track is so you can give a higher reward for being closer to the center of the track. Let’s look at track components and simulator measurements that can be used for our reward function.

Slide 20 for **2 minutes.** This is the first person view of AWS DeepRacer, as it drives down the track. The main components of the track are the track wall, the field, which is also considered off-track, the track boundaries are the two lines around the track surface. The track surface, or on-track (this includes the two boundary lines). And the center of the track. These components are important because we can use them to help determine whether an action resulted in a good or bad outcome. There are a few other components we should also look at. Slide 21 for **3 minutes.** And that is the coordinate system and track waypoints. This is a 3D environment so it has an X, Y, and Z axis. For simplicity we are only showing the X, and Y axis. Your car has an (X,Y) position associated with it. We provide “waypoints” spread around the track in the center of the track as a series of (X,Y) points (the superimposed pink line in center of the track). The waypoints help us to programmatically determine a how much of the track DeepRacer has completed, where the center line of the track is, in (x,y) coordinates, the distance that DeepRacer is from the center of the track, and the direction of the flow of the track. We can go one step further and determine the direction, with respect to the x-axis, from one waypoint to the next. We leave this to you. We also have waypoints for the outer boundary, and inner boundary. All of these track components and track waypoints are important because after every action in the state – action – reward – new state loop, they are variables that we can use to build logic to determine whether an outcome was good or bad. Also important to call out here, and reinforce it, mind the pun, is that this feedback loop only exists in the virtual world, and as such training can only happen in the virtual world. There is no feedback loop on the physical car to help it determine if an action was good or bad.

Slide 22 for **4 mins. What are hyper parameters?** These are parameters that feed into the reinforcement learning algorithm. They control various aspects of the algorithm, such as how quickly it should learn, how often we should train the network. What is the network again, the network is the “model” that when we give it an image or state as input, will return the action that should maximize the cumulative rewards from all future states. Weights are the “coefficients” in our neural network that help determine where we “look” in each picture to select the action that will maximize our expected cumulative rewards. Some PPO Parameters are. **Learning rate.** Controls how big the updates are to your network weights. If your learning rate is big the model will train fast but may not converge**. Batch.** When we update the network we don’t typically use all the experience at once. We carve up experience into batch sizes, and use each batch in turn to update the weights. Thus the network is updated one batch at a time. **Epochs.** Also we don’t always update our weights only once during each training run. 1 epoch means we update network only once, that is run through all batches once. 2 epochs means we run through all batches twice, so will train through all batches, and then retrain through all batches. **Discount factor.** This specifies how much future rewards contribute to the expected reward. The larger the discount factor the farther out the rewards that the model will consider, and slow down training. With a discount of 0.9 the vehicle includes rewards from an order of 10 future steps. With a discount of 0.999 the vehicle considers rewards from an order of 1000 future steps. We recommend 0.99, 0.999, 0.9999. **Espisodes between training.** Specifies how much experience to obtain before training the model.

Slide 23 for **4 mins.** In the virtual simulator, and in the real world, the state (image) of the track is sent to a reinforcement learning model and this returns the action to take. In AWS DeepRacer the actions are signals sent to the car to control how fast or slow it goes and what angle the steering should point in. The action is selected from the action space in the model. This action space is defined before we start training the model. Once trained you can’t change the action space. Think about the possible actions you take when driving a car, or if you don’t drive when you cycle a bike. You can accelerate to go faster, or brake to slow down, or just maintain your speed, and steer from left to right to change direction, or keep the same direction. We create a list of actions that we think the vehicle should perform during driving. It won’t be capable of doing anything outside of this list of actions. The more actions we provide, obviously the longer it will take to train a model but potentially get a better range of actions. Actions are multiplicative, so 2 speeds and 3 driving angles will give an action space of 6. **We also put a throttle limiter on the actual car. This is to protect the car. If your car was train in the simulator at a high speed, first test with caution as you don’t want it to race camera first into a wall. Slide 24. 50 MINUTES. Lab will cover building their first model. Once virtual leaderboards are open then can submit to virtual leaderboards. Slide 25. You have 21 minutes remaining, 9 minutes for Q&A.**

Slide 26 for **2 minutes. Lift up a car and show them.** CAR 18th scale 4WD with monster truck chassis. CPU Intel Atom™ Processor. MEMORY 4GB RAM. STORAGE 32GB (expandable). WI-FI 802.11ac. CAMERA 4 MP camera with MJPEG. SOFTWARE Ubuntu OS 16.04.3 LTS, Intel® OpenVINO™ toolkit, ROS Kinetic. DRIVE BATTERY 1100mAh lithium polymer. COMPUTE BATTERY 13600mAh USB-C PD. PORTS 4x USB-A, 1x USB-C, 1x Micro-USB, 1x HDMI. SENSORS Integrated accelerometer and gyroscope. Slide 27 for **2 minutes.** We use ROS v1.0 on top of Ubuntu to power the device. Every time you load a model for autonomous driving in the car, it is optimized by Intel’s OpenVINO model optimizer. During autonomous driving, the pictures flow from the camera through the media engine, into the Intel OpenVINO inference engine, from where the inference results are converted into driving actions (this is the speed and direction), which flows into the control node that converts these to Pulse Width Modulation (PWN) signals that are sent to the engine and steering servo in the car.

Slide 28 for **4 minutes.** A very important point to call out is the simulation to real domain transfer.

We trained our model in a simulated environment. It learned features from the simulated images. We will now attempt to use it in the real world with real world images. The real world doesn’t always look the same as the simulated world. This challenge is the sim-to-real domain transfer challenge. How can we possibly bridge this gap. Pause. 1. Control the environment. Make the simulator and real world seem as close to each other as possible. This is what you will see we have done. Our real-world tracks mimic the simulation environment. 2. Randomize the environment, this allows model to generalize. How can you do this? You change the textures/colors used in the simulation during the training process. This will make the model “insensitive” to the color of the line, and sensitive to a line. Another way would be to randomize physics, or camera field of view. 3. Modularity and abstraction. Swap out parts of the model architecture, for example put a pre-trained model in place that can detect whether the upcoming road is straight, curves left, or curves right. Your model will then train on top of this additional info provided in the image. Slide 29. **13 minutes remaining.** So that is a brief overview of RL, the console and reward functions. Let’s wrap-up with a quick discussion of how you can get involved.

**Slide 30 for 2 min.** Finally, the AWS DeepRacer League will officially kick off later in 2019 with an online experience for everyone, and physical events at select locations co-hosted with AWS Summits. Stay tuned! Developers train their models via the console for the fastest lap time, and can submit lap times to online leaderboards, or compete in-person at AWS Summits. Winners of each stage progress to the DRL Cup at reinvent 2019, to win the Champions Cup. Slide 31. Thank you and Q&A for 9 mins.