Anexo: código de aprendizaje por refuerzo en planeación de movimiento para coche autónomo

Mariana R. Hernández Rocha 150845

December 17, 2020

La identación en este documento ha sido modificada para una mejor visualización.

1 Función de recompensa

```
# Mariana Hernandez
# 150845
import math

def producto_punto(a,b):
        return (a[0]*b[0])+(a[1]*b[1])

def get_norma(a):
        return math.sqrt((a[0]**2)+(a[1]**2))

def normaliza(a):
    if (a[0]==0) and (a[1]==0):
        return [1,0] # para evitar divisiones problematicas
    else:
        norma = get_norma(a)
        return [a[0]/norma,a[1]/norma]

def reward_function(params):

    # recibir parámetros de entrada
    x = params['x'] #actual
    y = params['y'] #actual
    heading = params['heading']
    waypoints = params['waypoints']
    speed = params['speed']
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
```

```
track_width = params['track_width']
steering = abs(params['steering_angle'])
closest_points = params['closest_waypoints']

# encontrar los vecinos de adelante y atras sobre la pista
    next_point = waypoints[closest_points[1]]
    prev_point = waypoints[closest_points[0]]

    # obtener vectores entre puntos vecinos
    next_vector = [next_point[0]-x, next_point[1]-y]
    prev_vector = [x-prev_point[0], y-prev_point[1]]

    # normalizar vectores
    next_vector = normaliza(next_vector)
    prev_vector = normaliza(prev_vector)

    # calcular angulo entre puntos
    producto = producto_punto(next_vector,prev_vector)
    norma_next = get_norma(next_vector)
    norma_prev = get_norma(prev_vector)
    theta = math.acos(producto/(abs(norma_next)*abs(norma_prev)))
    theta = theta*(180/math.pi)

# la recompensa empieza con un valor muy baja
reward = 1e-3

# marcas que delimitan posiciones a lo ancho de la pista
marca1 = 0.1 * track_width
marca2 = 0.25 * track_width
marca3 = 0.5 * track_width

# la primer condicion es que el coche se mantenga
# dentro de la pista
if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:

    # la segunda condicion es que el coche se mantenga
    # cerca de la linea central
    if distance_from_center <= marca1:
        reward = 1
    elif distance_from_center <= marca2:
        reward = 0.5
    elif distance_from_center <= marca3:
        reward = 0.1
    else:
        reward = 1e-3
    # si hay un ángulo entre los puntos
    # previo, actual y futuro, tal vez exista una curva
    if theta<=20 :

               # tolerancia de steering
               # depende del espacio de accion
               steering_tol = 20

            # si gira mucho el volante se penaliza
            # esto para evitar movimientos zig zag
            if (steering > steering_tol):
                reward *=0.8

    elif (theta > 20) and (speed>0.3):

            # viene curva y va rapido
            reward *= 0.4

# evitar que conduzca en reversa
else:
    reward = params['progress']

return float(reward)
```

2 Código para lanzar proyecto

```
# Mariana Hernandez 150845
# codigo para inicializar block de notas AWS
# inicializacion del entrenamiento

#########################
### Prerequisitos AWS
#########################
# #
# # Run these commands if you want to modify the simapp
# #
# # Clean the build directory if present
# !python3 sim_app_bundler.py --clean

# # Download Robomaker simApp from the deepracer public s3 bucket
# simulation_application_bundle_location =
# "s3://deepracer-managed-resources-us-east-1/deepracer-simapp-notebook.tar.gz"
# !aws s3 cp {simulation_application_bundle_location} ./

# # Untar the simapp bundle
# !python3 sim_app_bundler.py --untar ./deepracer-simapp.tar.gz

# # Now modify the simapp from build directory and run this command.

# # Most of the simapp files can be found here (Robomaker changes)
# # bundle/opt/install/sagemaker_rl_agent/lib/python3.5/site-packages/
# # bundle/opt/install/deepracer_simulation_environment/share/
# deepracer_simulation_environment/
# # bundle/opt/install/deepracer_simulation_environment/lib/
# deepracer_simulation_environment/

# # Copying the notebook src/markov changes to the simapp
# (For sagemaker container)
# !rsync -av ./src/markov/ ./build/simapp/bundle/opt/install/sagemaker_rl_agent/
# lib/python3.5/site-packages/markov

# !python3 sim_app_bundler.py --tar

###############################
### Imports
###############################

import boto3
import sagemaker
import sys
import os
import re
import numpy as np
import subprocess
sys.path.append("common")
from misc import get_execution_role, wait_for_s3_object
from docker_utils import build_and_push_docker_image
from sagemaker.rl import RLEstimator, RLToolkit, RLFramework
from time import gmtime, strftime
import time
# from IPython.display import Markdown
# from markdown_helper import *

###################################
# Inicializar parametros basicso
###################################

# Select the instance type
instance_type = "ml.c4.2xlarge"
#instance_type = "ml.p2.xlarge"
#instance_type = "ml.c5.4xlarge"
```

```
# Starting SageMaker session
sage_session = sagemaker.session.Session()

# Create unique job name.
job_name_prefix = 'deepracer-notebook'

# Duration of job in seconds (1 hours)
job_duration_in_seconds = 3600

# AWS Region
aws_region = sage_session.boto_region_name
if aws_region not in ["us-west-2", "us-east-1", "eu-west-1"]:
    raise Exception("This notebook uses RoboMaker which is available
    only in US East (N. Virginia)," "US West (Oregon) and EU (Ireland).
    Please switch to one of these regions.")

################################
# Setup S3 bucket
################################

# S3 bucket
s3_bucket = sage_session.default_bucket()

# SDK appends the job name and output folder
s3_output_path = 's3://{}/'.format(s3_bucket)

#Ensure that the S3 prefix contains the keyword 'sagemaker'
s3_prefix = job_name_prefix + "-sagemaker-" + strftime("%y%m%d-%H%M%S", gmtime())

# Get the AWS account id of this account
sts = boto3.client("sts")
account_id = sts.get_caller_identity()['Account']

print("Using s3 bucket {}".format(s3_bucket))
print("Model checkpoints and other metadata will be stored at:
\ns3://{}/{}".format(s3_bucket, s3_prefix))

#################################3
# Create an IAM role
#################################

try:
    sagemaker_role = sagemaker.get_execution_role()
except:
    sagemaker_role = get_execution_role('sagemaker')
```

```
print("Using Sagemaker IAM role arn: \n{}".format(sagemaker_role))

##################################
# Permission setup for invoking AWS RoboMaker from this notebook
##################################

display(Markdown(generate_help_for_robomaker_trust_relationship(sagemaker_role)))

###########################
# Permission setup for Sagemaker to S3 bucke
###########################

display(Markdown(generate_s3_write_permission_for_sagemaker_role(sagemaker_role)))

#######################
# Permission setup for Sagemaker to create KinesisVideoStreams
########################

display(Markdown(generate_kinesis_create_permission_for_sagemaker_role(sagemaker_role)))

##########################
# Build and push docker image
##########################

%%time
from copy_to_sagemaker_container import get_sagemaker_docker,
copy_to_sagemaker_container, get_custom_image_name
cpu_or_gpu = 'gpu' if instance_type.startswith('ml.p') else 'cpu'
repository_short_name = "sagemaker-docker-%s" % cpu_or_gpu
custom_image_name = get_custom_image_name(repository_short_name)
try:
    print("Copying files from your notebook to existing sagemaker container")
    sagemaker_docker_id = get_sagemaker_docker(repository_short_name)
    copy_to_sagemaker_container(sagemaker_docker_id, repository_short_name)
except Exception as e:
    print("Creating sagemaker container")
    docker_build_args = {
        'CPU_OR_GPU': cpu_or_gpu,
        'AWS_REGION': boto3.Session().region_name,
    }
    custom_image_name = build_and_push_docker_image(repository_short_name,
    build_args=docker_build_args)
    print("Using ECR image %s" % custom_image_name)
```

```
#########################33
# Configure VPC
#########################3

ec2 = boto3.client('ec2')

#
# Check if the user has Deepracer-VPC and use that if its present.
# This will have all permission.
# This VPC will be created when you have used the Deepracer console and
# created one model atleast
# If this is not present. Use the default VPC connnection
#
deepracer_security_groups = [group["GroupId"] for group in
ec2.describe_security_groups()
['SecurityGroups']\
if group['GroupName'].startswith("aws-deepracer-")]

# deepracer_security_groups = False
if(deepracer_security_groups):
    print("Using the DeepRacer VPC stacks. This will be created if you run
    one training job from console.")
    deepracer_vpc = [vpc['VpcId'] for vpc in ec2.describe_vpcs()['Vpcs'] \
                     if "Tags" in vpc for val in vpc['Tags'] \
                     if val['Value'] == 'deepracer-vpc'][0]
    deepracer_subnets = [subnet["SubnetId"] for subnet in
    ec2.describe_subnets()["Subnets"] \
                         if subnet["VpcId"] == deepracer_vpc]
else:
    print("Using the default VPC stacks")
    deepracer_vpc = [vpc['VpcId'] for vpc in ec2.describe_vpcs()['Vpcs'] if
    vpc["IsDefault"] == True][0]

    deepracer_security_groups = [group["GroupId"] for
    group in ec2.describe_security_groups()['SecurityGroups'] \

    if 'VpcId' in group and group["GroupName"] == "default" and
    group["VpcId"] == deepracer_vpc]

    deepracer_subnets = [subnet["SubnetId"] for
    subnet in ec2.describe_subnets()["Subnets"] \
    if subnet["VpcId"] == deepracer_vpc and subnet['DefaultForAz']==True]

print("Using VPC:", deepracer_vpc)
print("Using security group:", deepracer_security_groups)
print("Using subnets:", deepracer_subnets)
```

```
#######################
# Create Route Table
#######################

#TODO: Explain to customer what CREATE_ROUTE_TABLE is doing
CREATE_ROUTE_TABLE = True

def create_vpc_endpoint_table():
    print("Creating ")
    try:
        route_tables = [route_table["RouteTableId"] for route_table in
        ec2.describe_route_tables()['RouteTables']\
                        if route_table['VpcId'] == deepracer_vpc]
    except Exception as e:
        if "UnauthorizedOperation" in str(e):
            display(Markdown(generate_help_for_s3_endpoint_permissions(sagemaker_role)))
        else:
            display(Markdown(create_s3_endpoint_manually(aws_region, deepracer_vpc)))
        raise e

    print("Trying to attach S3 endpoints to the following route tables:", route_tables)

    if not route_tables:
        raise Exception(("No route tables were found.
        Please follow the VPC S3 endpoint creation "
                         "guide by clicking the above link."))
    try:
        ec2.create_vpc_endpoint(DryRun=False,
                                VpcEndpointType="Gateway",
                                VpcId=deepracer_vpc,
                                ServiceName="com.amazonaws.{}.s3".format(aws_region),
                                RouteTableIds=route_tables)
        print("S3 endpoint created successfully!")
    except Exception as e:
        if "RouteAlreadyExists" in str(e):
            print("S3 endpoint already exists.")
        elif "UnauthorizedOperation" in str(e):
            display(Markdown(generate_help_for_s3_endpoint_permissions(role)))
            raise e
        else:
            display(Markdown(create_s3_endpoint_manually(aws_region, deepracer_vpc)))
            raise e

if CREATE_ROUTE_TABLE:
    create_vpc_endpoint_table()
```

```
# Mariana Hernandez 150845
# configuracion del entrenamiento
# editar el archivo default.py con la funcion de recompensa deseada

#####################
# Configure the preset for RL algorithm
##################

# Uncomment the pygmentize code lines to see the code

# Environmental File
#!pygmentize src/markov/environments/deepracer_racetrack_env.py

# Reward function
# modificar el archivo default.py con la funcion deseada
!pygmentize src/markov/rewards/default.py

# Action space
#!pygmentize src/markov/actions/model_metadata_10_state.json

# Preset File
#!pygmentize src/markov/presets/default.py
#!pygmentize src/markov/presets/preset_attention_layer.py

# Mariana Hernandez 150845
# entrenamiento del modelo

##########################
# Copy custom files to S3 bucket so that sagemaker & robomaker can pick it up
##########################

s3_location = "s3://%s/%s" % (s3_bucket, s3_prefix)
print(s3_location)

# Clean up the previously uploaded files
!aws s3 rm --recursive {s3_location}

# Make any changes to the environment and preset files below and upload these files
!aws s3 cp src/markov/environments/deepracer_racetrack_env.py
{s3_location}/environments/deepracer_racetrack_env.py

!aws s3 cp src/markov/rewards/default.py {s3_location}/rewards/reward_function.py

!aws s3 cp src/markov/actions/model_metadata_10_state.json
```

```
{s3_location}/model_metadata.json

!aws s3 cp src/markov/presets/default.py
{s3_location}/presets/preset.py
#!aws s3 cp src/markov/presets/preset_attention_layer.py
{s3_location}/presets/preset.py

###########################
# Train the RL model using the Python SDK Script mode
##########################

metric_definitions = [
    # Training> Name=main_level/agent, Worker=0, Episode=19, Total
    reward=-102.88, Steps=19019, Training iteration=1
    {'Name': 'reward-training',
     'Regex': '^Training>.*Total reward=(.*?),'},

    # Policy training> Surrogate loss=-0.32664725184440613, KL
    divergence=7.255815035023261e-06, Entropy=2.83156156539917,
    training epoch=0, learning_rate=0.00025
    {'Name': 'ppo-surrogate-loss',
     'Regex': '^Policy training>.*Surrogate loss=(.*?),'},
     {'Name': 'ppo-entropy',
     'Regex': '^Policy training>.*Entropy=(.*?),'},

    # Testing> Name=main_level/agent, Worker=0, Episode=19, Total
    reward=1359.12, Steps=20015, Training iteration=2
    {'Name': 'reward-testing',
     'Regex': '^Testing>.*Total reward=(.*?),'},
]

estimator = RLEstimator(entry_point="training_worker.py",
                        source_dir='src',
                        image_name=custom_image_name,
                        dependencies=["common/"],
                        role=sagemaker_role,
                        train_instance_type=instance_type,
                        train_instance_count=1,
                        output_path=s3_output_path,
                        base_job_name=job_name_prefix,
                        metric_definitions=metric_definitions,
                        train_max_run=job_duration_in_seconds,
                        hyperparameters={
                            "s3_bucket": s3_bucket,
                            "s3_prefix": s3_prefix,
                            "aws_region": aws_region,
                            "preset_s3_key": "%s/presets/preset.py"%
                            s3_prefix,
                            "model_metadata_s3_key":

                            "%s/model_metadata.json" % s3_prefix,
                            "environment_s3_key":
                        "%s/environments/deepracer_racetrack_env.py"
                        % s3_prefix,
                        },
                        subnets=deepracer_subnets,
                        security_group_ids=deepracer_security_groups,
                    )

estimator.fit(wait=False)
job_name = estimator.latest_training_job.job_name
print("Training job: %s" % job_name)

##################
# Create the Kinesis video stream
##################

kvs_stream_name = "dr-kvs-{}".format(job_name)

!aws --region {aws_region} kinesisvideo create-stream --stream-name
{kvs_stream_name} --media-type video/h264 --data-retention-in-hours 24
print ("Created kinesis video stream {}".format(kvs_stream_name))

######################
# Start the Robomaker job
#######################

robomaker = boto3.client("robomaker")

####################
# Create Simulation Application
###################

robomaker_s3_key = 'robomaker/simulation_ws.tar.gz'
robomaker_source = {'s3Bucket': s3_bucket,
                    's3Key': robomaker_s3_key,
                    'architecture': "X86_64"}
simulation_software_suite={'name': 'Gazebo',
                           'version': '7'}
robot_software_suite={'name': 'ROS',
                      'version': 'Kinetic'}
rendering_engine={'name': 'OGRE',
                  'version': '1.x'}

if not os.path.exists('./build/output.tar.gz'):
    print("Using the latest simapp from public s3 bucket")
    # Download Robomaker simApp for the deepracer public s3 bucket
    simulation_application_bundle_location = "s3://deepracer-managed-resources-us-east-1/
    deepracer-simapp-notebook.tar.gz"
    !aws s3 cp {simulation_application_bundle_location} ./

    # Remove if the Robomaker sim-app is present in s3 bucket
    !aws s3 rm s3://{s3_bucket}/{robomaker_s3_key}

    # Uploading the Robomaker SimApp to your S3 bucket
    !aws s3 cp ./deepracer-simapp-notebook.tar.gz s3://{s3_bucket}/{robomaker_s3_key}

    # Cleanup the locally downloaded version of SimApp
    !rm deepracer-simapp-notebook.tar.gz
else:
    print("Using the simapp from build directory")
    !aws s3 cp ./build/output.tar.gz s3://{s3_bucket}/{robomaker_s3_key}

    app_name = "deepracer-notebook-application" + strftime("%y%m%d-%H%M%S", gmtime())

print(app_name)
try:
    response = robomaker.create_simulation_application(name=app_name,
                                                       sources=[robomaker_source],
                                                       simulationSoftwareSuite=
                                                       simulation_software_suite,
                                                       robotSoftwareSuite=
                                                       robot_software_suite,
                                                       renderingEngine=
                                                       rendering_engine)
    simulation_app_arn = response["arn"]
    print("Created a new simulation app with ARN:", simulation_app_arn)
except Exception as e:
    if "AccessDeniedException" in str(e):
        display(Markdown(generate_help_for_robomaker_all_permissions(role)))
        raise e
    else:
        raise e

#######################
# Launch the Simulation job on RoboMake
########################
```

```
num_simulation_workers = 1

envriron_vars = {
    "WORLD_NAME": "reinvent_base",
    "KINESIS_VIDEO_STREAM_NAME": kvs_stream_name,
    "SAGEMAKER_SHARED_S3_BUCKET": s3_bucket,
    "SAGEMAKER_SHARED_S3_PREFIX": s3_prefix,
    "TRAINING_JOB_ARN": job_name,
    "APP_REGION": aws_region,
    "METRIC_NAME": "TrainingRewardScore",
    "METRIC_NAMESPACE": "AWSDeepRacer",
    "REWARD_FILE_S3_KEY": "%s/rewards/reward_function.py" % s3_prefix,
    "MODEL_METADATA_FILE_S3_KEY": "%s/model_metadata.json" % s3_prefix,
    "METRICS_S3_BUCKET": s3_bucket,
    "METRICS_S3_OBJECT_KEY": s3_bucket + "/training_metrics.json",
    "TARGET_REWARD_SCORE": "None",
    "NUMBER_OF_EPISODES": "0",
    "ROBOMAKER_SIMULATION_JOB_ACCOUNT_ID": account_id
}

simulation_application = {"application":simulation_app_arn,
                          "launchConfig": {"packageName":
                          "deepracer_simulation_environment",
                                           "launchFile":
                                           "distributed_training.launch",
                                           "environmentVariables": envriron_vars}
                         }


vpcConfig = {"subnets": deepracer_subnets,
             "securityGroups": deepracer_security_groups,
             "assignPublicIp": True}

responses = []
for job_no in range(num_simulation_workers):
    client_request_token = strftime("%Y-%m-%d-%H-%M-%S", gmtime())
    response =  robomaker.create_simulation_job(iamRole=sagemaker_role,
                                            clientRequestToken=client_request_token,
                                            maxJobDurationInSeconds=
                                            job_duration_in_seconds,
                                            failureBehavior="Continue",
                                            simulationApplications=
                                            [simulation_application],
                                            vpcConfig=vpcConfig
                                            )
    responses.append(response)
```

```
print("Created the following jobs:")
job_arns = [response["arn"] for response in responses]
for response in responses:
    print("Job ARN", response["arn"])

######################
# Visualizing the simulations in RoboMaker
#######################

display(Markdown(generate_robomaker_links(job_arns, aws_region)))

##################
# Creating temporary folder top plot metrics
##################

tmp_dir = "/tmp/{}".format(job_name)
os.system("mkdir {}".format(tmp_dir))
print("Create local folder {}".format(tmp_dir))

####################
# Plot metrics for training job
####################

%matplotlib inline
import pandas as pd
import json

training_metrics_file = "training_metrics.json"
training_metrics_path = "{}/{}".format(s3_bucket, training_metrics_file)
wait_for_s3_object(s3_bucket, training_metrics_path, tmp_dir)

json_file = "{}/{}".format(tmp_dir, training_metrics_file)
with open(json_file) as fp:
    data = json.load(fp)

df = pd.DataFrame(data['metrics'])
x_axis = 'episode'
y_axis = 'reward_score'

plt = df.plot(x=x_axis,y=y_axis, figsize=(12,5), legend=True, style='b-')
plt.set_ylabel(y_axis);
plt.set_xlabel(x_axis);

######################
# Evaluation - ReInvent Track
```

```
#######################

sys.path.append("./src")

num_simulation_workers = 1

envriron_vars = {
    "WORLD_NAME": "reinvent_base",
    "KINESIS_VIDEO_STREAM_NAME": "SilverstoneStream",
    "MODEL_S3_BUCKET": s3_bucket,
    "MODEL_S3_PREFIX": s3_prefix,
    "APP_REGION": aws_region,
    "MODEL_METADATA_FILE_S3_KEY": "%s/model_metadata.json" % s3_prefix,
    "METRICS_S3_BUCKET": s3_bucket,
    "METRICS_S3_OBJECT_KEY": s3_bucket + "/evaluation_metrics.json",
    "NUMBER_OF_TRIALS": "5",
    "ROBOMAKER_SIMULATION_JOB_ACCOUNT_ID": account_id
}

simulation_application = {
    "application":simulation_app_arn,
    "launchConfig": {
         "packageName": "deepracer_simulation_environment",
         "launchFile": "evaluation.launch",
         "environmentVariables": envriron_vars
    }
}

vpcConfig = {"subnets": deepracer_subnets,
             "securityGroups": deepracer_security_groups,
             "assignPublicIp": True}

responses = []
for job_no in range(num_simulation_workers):
    response =  robomaker.create_simulation_job(clientRequestToken=
    strftime("%Y-%m-%d-%H-%M-%S", gmtime()),
                                                outputLocation={
                                                  "s3Bucket": s3_bucket,
                                                  "s3Prefix": s3_prefix
                                                },
                                                maxJobDurationInSeconds=
                                                job_duration_in_seconds,
                                                iamRole=sagemaker_role,
                                                failureBehavior="Continue",
                                                simulationApplications=
                                                [simulation_application],
                                                vpcConfig=vpcConfig)
    responses.append(response)

# print("Created the following jobs:")
for response in responses:
    print("Job ARN", response["arn"])

#######################
# Creating temporary folder top plot metrics
#######################

evaluation_metrics_file = "evaluation_metrics.json"
evaluation_metrics_path = "{}/{}".format(s3_bucket, evaluation_metrics_file)
wait_for_s3_object(s3_bucket, evaluation_metrics_path, tmp_dir)

json_file = "{}/{}".format(tmp_dir, evaluation_metrics_file)
with open(json_file) as fp:
    data = json.load(fp)

df = pd.DataFrame(data['metrics'])
# Converting milliseconds to seconds
df['elapsed_time'] = df['elapsed_time_in_milliseconds']/1000
df = df[['trial', 'completion_percentage', 'elapsed_time']]

display(df)
```