Operationalizing an AWS ML project

Notebook instance

I created a sagemaker instance through the sagemaker notebook instance, I used ml.t3 medium instance type. In my opinion and experience so far, I tried to find a balance between instance type and time it takes to run my codes successfully. I tried using ml.t2 medium initially but ran into problems. The train and deploy solution notebook was later run on a ml.t3 medium instance type without issues. I also used Conda 3 kernel along with it. The cost of ml.t3 medium instance type is comparatively low, yet producing an effective result.

Ec2

Justification of choosing EC2 instance type

After so many trials I created an EC2 environment with m5.4xlarge. Lower instance type was difficult to work with. Since spot requests are cheap to come by, training a job on EC2 with a spot instance and instance type of m5.4xlarge is not too expensive and produces quick results. In my opinion, this is a good balance of cost and time

Comparison with notebook instance

Firstly EC2 requires the use of special commands to operate than sagemaker, this would need extra learning to perform training jobs on EC2. Although they have some similarities like the net() function, but tube different modules. For example, in the Jupyter notebook, we have used SageMaker modules such as Pytorch and HyperparameterTuner whereas none of these was used in EC2. To use pytorch we had to choose a deep learning AMI and activate pytorch on Ec2, unlike simply importing Pytorch in sagemaker jupyterlab

Lambda

Lambda is a very important AWS service that allows external users and applications to interact with the ML models. The lambda function makes use of a test event which is a key(s) and its values. In the case of this project, the lambda function is used to invoke an endpoint created with the jupyterlab notebook instance on dog images. We only needed a key which is a link to a dog image to make predictions (about the dog's class membership) on it, when tested, with the lambda function.

The list of numbers after execution

```
-6.1253204345703125, -5.282441139221191, -0.9243762493133545,
-1.9530177116394043, -4.047039985656738, -4.941751956939697, -1.9494880437850952,
-1.2029922008514404, -6.105813503265381, -1.3924306631088257, 1.2567530870437622,
-3.566382884979248, -3.2391152381896973, 0.2098890244960785, -2.7633554935455322,
-0.28615903854370117, -6.427050590515137, -2.0453379154205322,
-4.983096122741699, 0.8402009606361389, -4.534512996673584, -3.822802782058716,
-5.550165176391602, -4.573188304901123, -4.573231220245361, -8.49843692779541,
-3.425757884979248, -3.26025390625, -4.172610759735107, 0.7006067037582397,
-4.559579849243164, -3.931234836578369, -4.651791095733643, -1.2930560111999512,
-8.20923900604248, -4.679949760437012, -1.409886121749878, -1.1068689823150635,
-1.4276889562606812, -3.8438711166381836, -2.8326382637023926,
-1.8787815570831299, -1.347700595855713, -3.2801012992858887, -2.774076461791992,
-4.732923984527588, -1.639111042022705, -0.9374607801437378, -3.384518623352051,
-3.4544219970703125, -3.7706093788146973, -4.947218418121338, -5.793325424194336,
-4.8078694343566895, -4.388969898223877, -2.3717451095581055,
-3.9877898693084717, -3.96883487701416, -2.6317474842071533, -1.7577662467956543,
-5.398150444030762, -4.748569488525391, -4.1647748947143555, -4.357102394104004,
-5.088834285736084, -5.923055171966553, 1.1046220064163208, -7.392702579498291,
-3.487374782562256, -0.791500449180603, -1.2999095916748047, -2.3230576515197754,
-3.816739320755005, -3.749112367630005, -4.085755348205566, -2.2828474044799805,
-5.054614067077637, -1.7172632217407227, -6.263346195220947, -3.8642261028289795,
-1.8610841035842896, -4.494499683380127, -0.24543574452400208,
-1.9269506931304932, -4.886770248413086, -2.737664222717285,
-0.04345915839076042, -7.2739129066467285, -3.8046183586120605,
-1.2644985914230347, -4.884925842285156, -5.52992582321167, -4.470826148986816,
-6.048333644866943, -6.2546796798706055, -2.5418858528137207,
-1.3115867376327515, -4.05902099609375, -2.528355360031128, -5.386745929718018,
-5.9777069091796875, -0.3759233355522156, -3.6190145015716553,
-3.4574265480041504, -5.216198444366455, -5.574950695037842, -1.6088004112243652,
-0.454285204410553, -3.2335169315338135, -2.0570709705352783,
-0.9610785841941833, -0.3370148837566376, -5.670528411865234, -3.516512632369995,
-5.6414103507995605, -1.0753138065338135, -6.652886390686035, -0.534931480884552,
-3.749701738357544, -0.46211862564086914, -2.598323106765747, -3.192883253097534,
-5.096752643585205, -3.4870738983154297, -7.107240200042725, -1.7112265825271606,
-4.667588233947754, -1.1081674098968506, -3.54787278175354, -5.100436210632324,
-4.7613420486450195, -1.9971816539764404, -3.1278982162475586
```

Testing and security

AWS sagemaker is very secure and well structured, with the diverse policies one can attach to a given IAM role, jobs can successfully be completed in a secured environment without misusing resources and being vulnerable as a result of insecurity. There can be some vulnerabilities while using AWS sagemaker. For instance, full access policies may be too permissive for some lambda jobs, it is safer to only apply the specific policy required for a job.

Concurrency and auto-scaling

In my lambda function, I made 3 reserved concurrencies and 2 provisioned concurrency available to my Lamba through the configuration tab by publishing a version. My aim is to reduce the latency of a job to its lowest and the throughput is expected to be high but exactly known, so it was estimated (hypothetically speaking). A provisioned concurrency and a reserved currency were created to manage this situation. In the same scenario, I created and configured auto-scaling through my endpoint runtime settings. I specified a minimum and maximum instance count as 1 and 3 respectively, with 30 scale in and out cool down each and 10 target values.

This means that the maximum instance my endpoint can autoscale to is 3 instances, the time in which another instance would be deployed or deleted is 30 seconds, this is after 10 invocations are made to the endpoints. In all, a good engineering practice should strike a balance between the cost and delivery of its machine learning product. These decisions were made as a result of a hypothetical situation where my model has a high throughput, this is a reasonable way to manage the traffic and cost as well