Udacity AWS Machine Learning Engineers Nanodegree (ND189)

Project 4 (Course 5) Submission - Writeup

- **Step 1: Training and deployment on Sagemaker**
  - Check the screenshots
    Check the training and deployment notebook
  - **Set up VPC**

    Since my current AWS role doesn't have the permissions to create an `Internet Gateway`, `NAT Gateway`, or assign an `Elastic IP` address, I set up a SageMaker notebook instance in the default VPC's public subnet in one of the Availability Zones, using the launch-wizard-1 Security Group, which allows all inbound and outbound traffic. I enabled direct internet access for the instance so the notebook kernel can update Python libraries.
  - **Set up S3 bucket**

    The dog images uploaded for Project 3 from an S3 bucket in another account was reused. To do this, I created an `S3 Gateway` endpoint within the VPC and updated the S3 bucket policy to enable **cross-account access**.
  - **Training**

    The `train_and_deploy-solution.ipynb` and `hpo.py` files were uploaded to the notebook instance and ran the notebook. The HPO job, which had 2 runs on an `ml.g4dn.xlarge` instance, took about 40 minutes. Afterward, using the 'best hyperparameters,' I ran a multi-instance training job on two `ml.g4dn.xlarge` instances.

    I chose the `ml.g4dn.xlarge` instance because it's one of the smaller GPU options, and it worked very well for the image classification task in Project 3 training.
  - **Deployment**

    The `inference2.py` file was uploaded to the notebook instance, deployed an inference endpoint, and tested it. Since the focus of this project is on ML operations, inference accuracy isn't the primary concern.
  - **Clean up resources**

    Afterwards, the model, endpoint, and notebook instance were deleted.

- **Step 2: EC2 Training**
  - Check the operation details and screenshots
    Check the demo training code
  - Since the demo training code doesn't appear to use a GPU, we launched a `t2.xlarge` CPU EC2 instance for the training. Obviously, SageMaker is a fully managed service that saves the hassle of installing GPU drivers, CUDA, Python dependencies, and more. However, managing resources ourselves could potentially reduce costs.

Setting	Value
Image	`Amazon Linux 2023 AMI`
Instance Type	`t2.xlarge` (w/o GPU)
VPC	Default VPC (same as Step 1)
Security Group	`launch-wizard-1` (all inbound/outbound traffic allowed)
Role Name	`udacity-p4-ec2` (permissions: `AmazonElasticMapReduceforEC2Role`, SageMaker execution role, and S3 full access)
Dependencies	`torch`, `torchvision`, `Pillow` (including `Numpy`), `tqdm`

- **Step 3: Lambda function setup**
- **Step 4: Security and testing**
- **Step 5: Concurrency and auto-scaling**
  - Check the operation details and screenshots
    Check the deployment notebook and Lambda function code
  - I deployed the model as an endpoint called `p4-dog-image-classification`. It takes the endpoint name and an image URL as input, and outputs a prediction in the form of a label number (the argmax result).
    - input payload:

```
{
  "endpoint_name": "p4-dog-breed-classification",
  "request_dict": "{\"url\": \"https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314
}
```

- output example
  `[56]`
- argmax:

```
response = json.loads(response['Body'].read().decode())
## argmax of 2D list, equivlent to np.argmax(response, 1)
body = [max(range(len(row)), key=row.__getitem__) for row in response]
```

- Then, I configured the concurrency for both the endpoint and the Lambda function. The configuration balances performance and cost by setting the target value for `SageMakerVariantInvocationsPerInstance` to 100, assuming each instance can comfortably handle that many requests. This ensures the system doesn't scale too early or too late. The 10-second `scale-in` and `scale-out` cooldowns allow the system to quickly adapt to changes in traffic without overprovisioning or underprovisioning resources, making it responsive to both traffic spikes and drops. For Lambda concurrency, setting it between 50-100 ensures the function can handle bursts of requests without overwhelming the SageMaker endpoint, while still distributing the load efficiently across instances. This configuration offers a balanced, responsive approach to managing both traffic fluctuations and resource usage.

Setting	Value	Explanation
Target Metric	`SageMakerVariantInvocationsPerInstance`	Average of 100 invocations per instance, adjust as needed based on model capacity
Scale-in Cooldown	10 seconds	Adjust quickly to drops in traffic, but avoid rapid fluctuations
Scale-out Cooldown	10 seconds	React quickly to increased traffic, but avoid over-scaling
Lambda Concurrency	50-100 (depending on load)	Enough concurrency to keep up with the traffic, adjust based on traffic patterns

- Clean up resources