

HW2 – MNIST Digit Recognition Training and Deployment- A Comparison

In this assignment, I will be comparing the workflow of building, training and deploying a model for digit recognition using MNIST notebook code for IBM Watson Cloud, AWS SageMaker and Google Vertex AI. The three platforms have different MNIST notebooks for digit recognition, please find the git repos used for getting the code as follows:

- IBM Watson Studio – Please find attached code in `ibm_mnist.ipynb`. Also see reference list for repo referred to.[1]
- GCP Vertex AI – Please see reference list for repo referred to.[2]
- AWS SageMaker – Please find attached code in `aws_mnist.ipynb`.

The model training and deployment for digit recognition using the MNIST dataset has a similar workflow on the three platforms, and I have divided it into the following major modules- setup, creating buckets, loading data into buckets, training the model, deploying the model and cleanup.

I will now present the implementation details of the model on the three platforms, keeping these modules in mind. At the end of this section, I also present a comparison of model deployment on the 3 platforms.

1. ENVIRONMENT SETUP

1.1. IBM

The IBM Cloud CLI helps in generating the API Key, with which notebooks can connect to other IBM services using the `APIClient` from the `ibm_watson_machine_learning` package. I faced a few issues while trying to get the API Key using the IBM CLI as the setup in Windows was not very straightforward. Once I figured that out, I did not face any hassle in other steps.

```
) api_key = 'Entered my API Key here'
  location = 'us-south'

) wml_credentials = {
  "apikey": api_key,
  "url": 'https://' + location + '.ml.cloud.ibm.com'
}
```

Figure 1: Setting up connection with API Key for IBM Cloud

1.2. GCP

I had to create a GCP account first. Google Cloud Platform's Cloud Engine is used to perform digit recognition using the MNIST dataset. I created a new project on GCP, and had to link it to the resources I needed on GCP. This included enabling the Cloud Machine Learning API from the API Manager in the Google Cloud Console. All commands are executed through command line on the Google Cloud Shell and not on Jupyter Notebook (unlike IBM and Amazon). As part of the setup, I also cloned the git repository using the link provided.

1.3. AWS

Amazon Sagemaker on the Amazon cloud provides a platform to prepare, build, train and deploy ML models. First I created a SageMaker domain with an EC2 instance. To grant SageMaker access to other AWS services, I imported and specified an execution role.

```
# Define IAM role
import boto3
import re
from sagemaker import get_execution_role

role = get_execution_role()
```

Figure 2: Importing get_execution in AWS SageMaker

2. CREATING BUCKETS

2.1. IBM

I needed some space where I can do my work, and using Cloud Object Storage (storage service) and Watson Machine Learning instance (ML service), I created a space in the deployment space.

```
client.spaces.list(limit=10)
```

ID	NAME	CREATED
e7f2b8bb-f82f-4195-a918-41bdb1167034	cmldep	2022-02-23T01:52:46.108Z

2.2. GCP

For storage, GCP provides Cloud Storage Buckets. I created the buckets via command line as below:

```
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ PROJECT_ID=$(gcloud config list project --format "value(core.project)")
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ BUCKET="gs://${PROJECT_ID}-ml"
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ gsutil mkdir -c regional -l us-central1 $BUCKET
Creating gs://cml-niharika-ml/...
```

Figure 3: Commands to create buckets on GCP

The interface is convenient and user-friendly as well, and we can modify the configuration for the bucket. GCP allows us to set the geographic location where we want to store the data, choose the storage class, provide access control and set up protection for the buckets.

← Bucket details

cml-niharika-ml

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Regional	Subject to object ACLs	None

OBJECTS **CONFIGURATION** PERMISSIONS PROTECTION LIFECYCLE

Overview

Created	February 23, 2022 at 5:08:07 PM GMT-5
Updated	February 23, 2022 at 5:08:07 PM GMT-5
Location type	Region
Location	us-central1 (Iowa)
Replication	—
Default storage class	Regional
Requester Pays	OFF
Labels	None
Cloud Console URL	https://console.cloud.google.com/storage/browser/cml-niharika-ml
gsutil URI	gs://cml-niharika-ml

Permissions

Access control	Fine-grained
Public access prevention	Not enabled by org policy or bucket setting
Public access status	Subject to object ACLs

Protection

Object versioning	Off
Retention policy	None
Encryption type	Google-managed key

Object lifecycle

Lifecycle rules	None
-----------------	------

Figure 4: UI of configuration details of the bucket created

2.3. AWS

For storage, AWS provides the Amazon Simple Storage Service or S3. The S3 bucket only has a logical hierarchy and stores objects using their key names to imply a folder structure.

```
bucket = sagemaker.Session().default_bucket()
prefix = "sagemaker/DEMO-linear-mnist"
```

Figure 5: Setup for bucket creation on SageMaker

I downloaded the training data from S3 bucket using the following commands:

```

%%time
import pickle, gzip, numpy, urllib.request, json

fobj = boto3.client('s3').get_object(
    Bucket='sagemaker-sample-files',
    Key='datasets/image/MNIST/mnist.pkl.gz'
)['Body'].read()

with open('mnist.pkl.gz', 'wb') as f:
    f.write(fobj)

# Load the dataset
with gzip.open("mnist.pkl.gz", "rb") as f:
    train_set, valid_set, test_set = pickle.load(f, encoding="latin1")

CPU times: user 761 ms, sys: 351 ms, total: 1.11 s
Wall time: 1.83 s

```

Figure 6: Downloading the training data into S3 bucket on SageMaker

The train data set contained 2 values, the actual training images and their associated classification label.

3. LOADING DATA INTO BUCKETS

3.1. *IBM*

I created a COS service to write to the Object Cloud Storage. I created 2 COS buckets to be used for storing training and test data. To get the training data, I created a set of links for them, and uploaded files from these links to the Cloud Object Storage. I established connections to the COS buckets.





Name	Type	Tags	Modified by	Modified on	
Input COS connection	Connection		 Niharika Sinha	Feb 22, 2022	
Output COS connection	Connection		 Niharika Sinha	Feb 22, 2022	

Figure 7: Seeing the available buckets through UI on IBM Watson Console

I got the sample model definition content file from git.

3.2. *GCP*

With the buckets ready, I uploaded the MNIST dataset into the training bucket. The dataset is stored in TFRecords format.

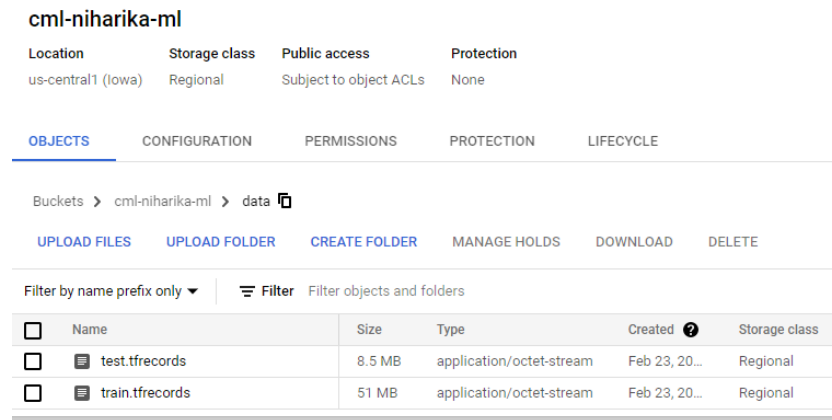


Figure 8: Seeing the train and test data buceks through UI on GCP

3.3. AWS

The XGBoost algorithm used for training the model requires the data to be in csv format. After transforming the data from numpy.array format to csv format, I uploaded the data into a S3 bucket for this session.

```
key = '{}/{}examples'.format(prefix, data_partition_name)
url = 's3://{}{}'.format(bucket, key)

boto3.Session().resource('s3').Bucket(bucket).Object(key).upload_file('data.csv')

print('Done writing to {}'.format(url))
```

Figure 9: Uploading data into S3 bucket on SageMaker

4. TRAIN THE MODEL

4.1. IBM

I prepared the training metadata and started training the model. Using the training id, a check can be kept on the training status. The status of training can also be checked by list() command.

```
client.training.list(limit=5)
```

ID (training)	STATE	CREATED
31ef55b7-24ef-44c1-9998-1c37140e9f61	running	2022-02-23T02:00:59.677Z

Figure 10: Checking the training status on IBM Watson

To persist the trained model, I downloaded the trained model from the COS, unpacked and compressed it to tar.gz format. I published this model, and could check it using the list_models() function.

4.2. GCP

With the setup and training buckets in place, I created a training job and submitted it to the Cloud Machine Learning. This is a pretty straightforward step and the training job I submitted was:

```

niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ JOB_ID="$(USER)_$(date +%Y%m%d_%H%M%S)"
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ gcloud ai-platform jobs submit training ${JOB_ID} \
--package-path trainer \
--module-name trainer.task_p3 \
--staging-bucket ${BUCKET} \
--job-dir ${BUCKET}/${JOB_ID} \
--runtime-version 1.14 \
--python-version 3.5 \
--region us-central1 \
--config config/config.yaml \
-- \
--data_dir ${BUCKET}/data \
--output_dir ${BUCKET}/${JOB_ID} \
--train_steps 10000

```

Figure 11: The training job submitted to Cloud Machine Learning

There is no specification for the job id, but it has to be unique. We can modify the resources allocated for the job. The accuracy against the test data is shown at the end of training. Due to an issue related to linking billing account to the project, the training of my model was stuck in the queue. I was able to get rid of the problem when I changed the Billing account linked to the project. While the model was training, I was able to monitor its real time progress, and was also able to fetch real time logs being generated. It took about 10 minutes to train the model.

4.3. AWS

The model is trained using the XGBoost algorithm. I used an instance of the sagemaker.estimator to build the model my_model. I passed the container, role, output path and the sagemaker session as parameters.

```

# :: create an instance of the SageMaker Estimator

container = sagemaker.image_uris.retrieve('xgboost', boto3.Session().region_name, 'latest')

my_model = sagemaker.estimator.Estimator(
    container,
    role,
    instance_count = 1,
    instance_type = 'ml.m5.xlarge',
    volume_size = 5,
    output_path = output_location,
    sagemaker_session = sagemaker.Session()
)

```

Figure 12: Preparing the training model on SageMaker

Once defined, I finetuned the model by setting a few of the hyperparameters. Few of them have been renamed in the latest SageMaker versions, due to which I had to modify the code given in the git repo.

```

# Set the hyperparameters of the defined model

my_model.set_hyperparameters(
    max_depth = 5,
    eta = 0.2,
    gamma = 4,
    min_child_weight = 6,
    silent = 0,
    objective = 'multi:softmax',
    num_class = 10,
    num_round = 10
)

```

Figure 13: Setting the hyperparameters for the training model on SageMaker

Batch training was used for this job. Once all the parameters have been set, I gave the command to begin model training. It took about 7 minutes to complete the training.

```
[*]: # Start Model Training
my_model.fit(
    inputs = data_channels,
    logs = True
)

2022-02-25 05:32:15 Starting - Starting the training job...
2022-02-25 05:32:40 Starting - Preparing the instances for trainingProfilerReport-1645767135: InProgress
..
```

Figure 14: Model training on SageMaker

5. DEPLOY THE MODEL

5.1. IBM

I deployed the published model as a web service and checked its details using dump. To score the deployed model, I prepared a scoring dictionary and sent it to the deployed model.

```
for i, image in enumerate([x_test[0], x_test[1]]):
    plt.subplot(2, 2, i + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
```



Figure 15: Testing the model with scoring dictionary on IBM

The model gave correct predictions for the images in the scoring dictionary.

5.2. GCP

Once trained, I deployed the model from the shell. First I created an AI Platform model MNIST1 in the us-central1 region.

```
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ MODEL_NAME=MNIST1
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ ORIGIN=$(gsutil ls ${BUCKET}/${JOB_ID}/export/exporter | tail -1)
niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ gcloud ai-platform models create ${MODEL_NAME} --regions us-central1
Using endpoint [https://ml.googleapis.com/]
Created ai platform model [projects/cml-niharika/models/MNIST1].
```

Figure 16: Creating AI Platform Model on GCP

I had to include the runtime version in the deployment command, and issued the following:

```

niharikagcp1@cloudshell:~/cloudml-dist-mnist-example (cml-niharika)$ gcloud ai-platform versions create \
--origin ${ORIGIN} \
--model ${MODEL_NAME} \
--runtime-version=2.7 \
${VERSION_NAME}
Please specify a region:
(For the global endpoint the region needs to be specified as 'global'.)
[1] global
[2] asia-east1
[3] asia-northeast1
[4] asia-southeast1
[5] australia-southeast1
[6] europe-west1
[7] europe-west2
[8] europe-west3
[9] europe-west4
[10] northamerica-northeast1
[11] us-central1
[12] us-east1
[13] us-east4
[14] us-west1
[15] cancel
Please enter your numeric choice: 11

To make this the default region, run `gcloud config set ai_platform/region us-central1`.

Using endpoint [https://us-central1-ml.googleapis.com/]
Creating version (this might take a few minutes).....done.

```

Figure 17: Model Deployment command on GCP

It took around 10 minutes for the deployed model to become ready. To test the deployed model, I submitted 10 images through JSON files, and submitted a request to the model for online prediction. The model returned the following output, which includes the most probable digit of the input image and the class probabilities corresponding to the 10 digits [0-9].

```

Using endpoint [https://us-central1-ml.googleapis.com/]
CLASSES: 7
PROBABILITIES: [1.25091991e-18, 3.720129e-16, 1.62488329e-14, 7.40394609e-17, 1.14457847e-15, 5.66890366e-17, 4.08000496e-21, 1.0, 1.98698378e-17, 8.01407378e-12]

CLASSES: 2
PROBABILITIES: [4.95663e-13, 2.98575461e-16, 1.0, 2.01673952e-20, 8.949856e-18, 7.60847533e-22, 3.38610352e-13, 5.8271558e-21, 4.57656285e-19, 6.33783045e-19]

CLASSES: 1
PROBABILITIES: [2.25211928e-12, 0.999999642, 4.56585568e-11, 1.18570586e-13, 2.02035579e-07, 4.7063551e-11, 8.09110912e-09, 1.13026766e-09, 7.48825357e-08, 3.88145661e-11]

CLASSES: 0
PROBABILITIES: [1.0, 2.04899352e-15, 7.79714563e-13, 2.01829168e-15, 5.64758413e-15, 6.74801461e-15, 1.69785588e-10, 1.05304342e-13, 2.33848e-14, 5.10357902e-11]

CLASSES: 4
PROBABILITIES: [2.6572137e-18, 1.27212861e-16, 1.90012022e-17, 1.17422357e-21, 1.0, 1.06019874e-15, 1.4437071e-15, 2.4472622e-15, 2.49893219e-14, 9.87971441e-11]

CLASSES: 1
PROBABILITIES: [1.65790572e-13, 1.0, 5.30183861e-13, 4.74296734e-16, 7.17388787e-11, 3.50946316e-14, 1.22591e-12, 2.19042673e-10, 6.36109637e-11, 1.7060215e-12]

CLASSES: 4
PROBABILITIES: [2.06647308e-19, 1.77109592e-14, 2.77531113e-16, 2.71798175e-22, 1.0, 1.5401966e-14, 2.38334954e-16, 7.17174803e-16, 9.43524e-11, 1.18008745e-11]

CLASSES: 9
PROBABILITIES: [2.96143725e-14, 1.07294574e-15, 3.4359811e-15, 1.65968843e-15, 4.1627e-09, 9.19628499e-15, 1.00245711e-20, 4.04695808e-13, 1.47851544e-12, 1.0]

CLASSES: 5
PROBABILITIES: [2.01431916e-13, 2.1634616e-13, 2.12122258e-18, 2.30808041e-17, 2.04565932e-15, 0.999949455, 1.59064234e-06, 3.06813168e-17, 4.90386847e-05, 5.54365885e-12]

CLASSES: 9
PROBABILITIES: [4.35137224e-19, 7.86604757e-18, 3.58598331e-19, 9.7698633e-19, 2.80097083e-08, 4.85148179e-16, 1.94366519e-19, 7.77874917e-11, 3.03539529e-12, 1.0]

```

Figure 18: Classification made by model deployed on GCP. All classifications made were correct.

5.3. AWS

Deploying the model was a single line command. It took about 7-10 minutes to finish deploying. After successful deployment, to test the model, I downloaded the test images from another S3 bucket and tested the model against 10 images. The model correctly classified all the 10 images.


```
# Infer 10 test images
from sagemaker.predictor import json_serializer

with open('test_data', 'r') as f:
    for j in range(0, 10):
        single_test = f.readline()
        result = xgb_predictor.predict(single_test)

        print('inferred: {} --- should be {}'.format(result, test_set[1][j]))

inferred: b'7.0' --- should be 7
inferred: b'2.0' --- should be 2
inferred: b'1.0' --- should be 1
inferred: b'0.0' --- should be 0
inferred: b'4.0' --- should be 4
inferred: b'1.0' --- should be 1
inferred: b'4.0' --- should be 4
inferred: b'9.0' --- should be 9
inferred: b'5.0' --- should be 5
inferred: b'9.0' --- should be 9
```

Figure 19: Classification made by the deployed model on SageMaker. All classifications made for the test data were correct.

6. CLEANUP

6.1. **IBM**

Cleanup on IBM Watson was done using the console through UI. The cleanup was pretty simple. I cleared all the objects I had created- COS objects, Watson Studio and ML services and the deployment space. The deployment space provided a good and clean interface to monitor and manage created resources.

6.2. **GCP**

The cleanup on GCP involves deletion of the buckets. The Google Cloud Storage incurs charges. After the buckets, I deleted the Cloud Engine instance and shut down the project.

6.3. **AWS**

First, I deleted the instance associated with the model I had deployed. After this, I deleted the SageMaker domain I had created.

COMPARISON

Feature	IBM	GCP	AWS
Connection to services	Authenticate the Watson Machine Learning service using API Key that is generated using IBM CLI.	Linked the new project to services needed on GCP. Enabled the Cloud ML API from the API Manager in the Console.	Import and specify connection roles.
Code	Model built, trained and deployed using PyTorch. Allows importing projects through URL. Coded using Jupyter Notebook.	TensorFlow based training model. Coded through the Google Cloud Shell. Allows linking Google Collab.	Coded using Jupyter Notebook.

<i>Implementation</i>	Implemented on Watson Machine Learning Service.	Implemented on the Google AI Platform.	Implemented on the SageMaker domain created.
<i>Workspace</i>	Cloud Object Storage used for creating deployment space. No access control storage.	Cloud Storage Buckets. Can modify bucket configuration and access control.	Amazon Simple Storage Service or S3. The bucket must be in the same geographic region as the notebook instance.
<i>Health Check</i>	Allows monitoring, but cumbersome to use.	Allows real time monitoring of status and check real time logs.	Allows real time monitoring of model deployment.
<i>Personal Experience</i>	I found the implementation not as convenient as the other 2, as it required a separate environment setup. I also faced issues in getting the API Key for connecting my notebook to the other IBM services.	I liked working on GCP more than IBM, but did not find it as convenient as AWS. I faced blocker related to Billing information, due to which model training did not progress. Once this got rectified, the rest of the process was very convenient.	I found the implementation on SageMaker to be the most convenient. Once the SageMaker domain was created, I executed the code using Jupyter notebook.

REFERENCES

- [1] [Online] https://github.com/IBM/watson-machine-learning-samples/blob/master/cloud/notebooks/python_sdk/experiments/deep_learning/Use%20PyTorch%20to%20recognize%20hand-written%20digits.ipynb
- [2] [Online] <https://github.com/GoogleCloudPlatform/cloudml-dist-mnist-example>