Vertex AI: Hyperparameter Tuning

2 hours 30 minutesFree

Overview

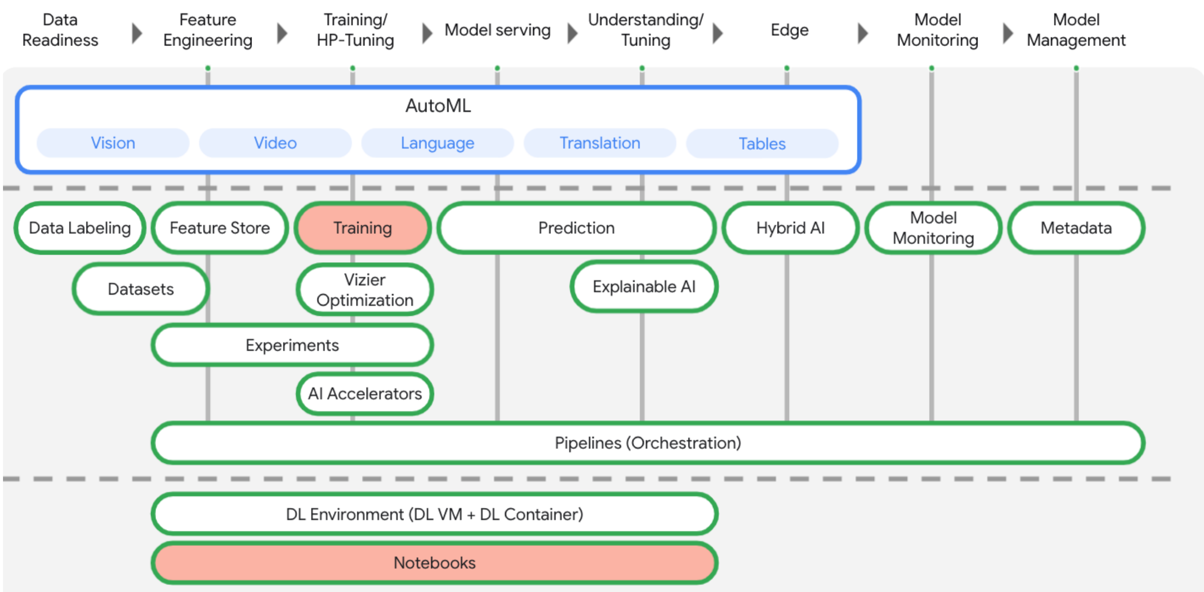
In this lab, you learn how to use [Vertex AI] to run a hyperparameter tuning job for a TensorFlow model. While this lab uses TensorFlow for the model code, you could easily replace it with another framework.

Learning objectives

- **Modify training application code for hyperparameter tuning.**
- **Launch a hyperparameter tuning job from the Vertex AI UI.**

Introduction to Vertex AI

This lab uses the newest AI product offering available on Google Cloud. [Vertex AI](#) integrates the ML offerings across Google Cloud into a seamless development experience. Previously, models trained with AutoML and custom models were accessible via separate services. The new offering combines both into a single API, along with other new products. You can also migrate existing projects to Vertex AI. If you have any feedback, please see the [support page](#).
Vertex AI includes many different products to support end-to-end ML workflows. This lab focuses on the products highlighted below: Training/HP-Tuning and Notebooks.



Setup and requirements

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. **Sign in to Qwiklabs using an incognito window.**

2. Note the lab's access time (for example, `1:15:00`), and make sure you can finish within that time. There is no pause feature. You can restart if needed, but you have to start at the beginning.

3. When ready, click Start lab.

4. Note your lab credentials (Username and Password). You will use them to sign in to the Google Cloud Console.

5. Click Open Google Console.

6. Click Use another account and copy/paste credentials for this lab into the prompts. If you use other credentials, you'll receive errors or incur charges.

7. Accept the terms and skip the recovery resource page.

Note: Do not click End Lab unless you have finished the lab or want to restart it. This clears your work and removes the project.

Enable the Compute Engine API

1. In the Cloud Console, click Navigation menu > API & Services > Library.

2. Search for Compute Engine API, then click Enable if it isn't already enabled. You'll need this to create your notebook instance.

Enable the Container Registry API

1. In the Cloud Console, click Navigation menu > API & Services > Library.

2. Search for Google Container Registry API and select Enable if it isn't already. You'll use this to create a container for your custom training job.

Enable the Notebooks API

1. In the Google Cloud Console, in the Navigation menu, click APIs & Services > Library.

2. Search for Notebooks API and press ENTER.

3. Click on the Notebooks API result.

4. If the API is not enabled, click Enable.

Enable the Vertex AI API

- In the Google Cloud Console, on the Navigation menu, click Vertex AI > Dashboard, and click Enable Vertex AI API.

Task 1. Launch Vertex AI Notebooks instance

1. In the Google Cloud Console, on the Navigation Menu, click Vertex AI > Workbench. Select User-Managed Notebooks.

2. On the Notebook instances page, click New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs.

3. In the New notebook instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click Create. The new VM will take 2-3 minutes to start.

4. Click Open JupyterLab. A JupyterLab window will open in a new tab.

5. You will see "Build recommended" pop up, click Build. If you see the build failed, ignore it.

Note: The model you'll be training and tuning in this lab is an image classification model trained on the horses or humans dataset from TensorFlow Datasets.

Task 2. Containerize training application code

You'll submit this hyperparameter tuning job to Vertex by putting your training application code in a Docker container and pushing this container to Google Container Registry. Using this approach, you can tune hyperparameters for a model built with any framework.

1. **In JupyterLab, from the Launcher menu, open a Terminal window in your notebook instance.**
2. **Create a new directory called** `horses_or_humans` **and cd into it:**

```
mkdir horses_or_humans
cd horses_or_humans
```

Copied!
content_copy

Create a Dockerfile

The first step in containerizing your code is to create a Dockerfile. In the Dockerfile you'll include all the commands needed to run the image. It'll install all the necessary libraries, including the [CloudML Hypertune library](), and set up the entry point for the training code.

1. **From your Terminal, create an empty Dockerfile:**

```
touch Dockerfile
```

Copied!
content_copy

2. **Open the Dockerfile from the left menu and copy the following into it:**

```
FROM gcr.io/deeplearning-platform-release/tf2-gpu.2-5
WORKDIR /
# Installs hypertune library
RUN pip install cloudml-hypertune
# Copies the trainer code to the docker image.
COPY trainer /trainer
# Sets up the entry point to invoke the trainer.
ENTRYPOINT ["python", "-m", "trainer.task"]
```

Copied!
content_copy

3. **Save the file by pressing CTRL+S.**

This Dockerfile uses the [Deep Learning Container TensorFlow Enterprise 2.5 GPU Docker image](). The Deep Learning Containers on Google Cloud come with many common ML and data science frameworks pre-installed.

After downloading that image, this Dockerfile sets up the entrypoint for the training code. You haven't created these files yet – in the next step, you'll add the code for training and tuning the model.

Add model training code

1. **From your Terminal, run the following to create a directory for the training code and a Python file where you'll add the code:**

```
mkdir trainer
touch trainer/task.py
```

Copied!

content_copy

You should now have the following in your `horses_or_humans/` **directory:**

```
+ Dockerfile
+ trainer/
    + task.py
```

2. **Next, open the task.py file you just created and copy the code below:**

```
import tensorflow as tf
import tensorflow_datasets as tfds
import argparse
import hypertune
NUM_EPOCHS = 10
def get_args():
  '''Parses args. Must include all hyperparameters you want to tune.'''
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--learning_rate',
      required=True,
      type=float,
      help='learning rate')
  parser.add_argument(
      '--momentum',
      required=True,
      type=float,
      help='SGD momentum value')
  parser.add_argument(
      '--num_neurons',
      required=True,
      type=int,
      help='number of units in last hidden layer')
  args = parser.parse_args()
  return args
def preprocess_data(image, label):
  '''Resizes and scales images.'''
  image = tf.image.resize(image, (150,150))
  return tf.cast(image, tf.float32) / 255., label
def create_dataset():
```

```
  '''Loads Horses Or Humans dataset and preprocesses data.'''
  data, info = tfds.load(name='horses_or_humans', as_supervised=True,
with_info=True)
  # Create train dataset
  train_data = data['train'].map(preprocess_data)
  train_data  = train_data.shuffle(1000)
  train_data  = train_data.batch(64)
  # Create validation dataset
  validation_data = data['test'].map(preprocess_data)
  validation_data  = validation_data.batch(64)
  return train_data, validation_data
def create_model(num_neurons, learning_rate, momentum):
  '''Defines and complies model.'''
  inputs = tf.keras.Input(shape=(150, 150, 3))
  x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu')(inputs)
  x = tf.keras.layers.MaxPooling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(x)
  x = tf.keras.layers.MaxPooling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu')(x)
  x = tf.keras.layers.MaxPooling2D((2, 2))(x)
  x = tf.keras.layers.Flatten()(x)
  x = tf.keras.layers.Dense(num_neurons, activation='relu')(x)
  outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
  model = tf.keras.Model(inputs, outputs)
  model.compile(
      loss='binary_crossentropy',
      optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate,
momentum=momentum),
      metrics=['accuracy'])
  return model
def main():
  args = get_args()
  train_data, validation_data = create_dataset()
  model = create_model(args.num_neurons, args.learning_rate,
args.momentum)
  history = model.fit(train_data, epochs=NUM_EPOCHS,
validation_data=validation_data)
  # DEFINE METRIC
  hp_metric = history.history['val_accuracy'][-1]
  hpt = hypertune.HyperTune()
  hpt.report_hyperparameter_tuning_metric(
      hyperparameter_metric_tag='accuracy',
      metric_value=hp_metric,
      global_step=NUM_EPOCHS)
if __name__ == "__main__":
    main()
```

Copied!

content_copy

3. **Save the file by pressing CTRL+S.**

Before you build the container, let's take a deeper look at the code. There are a few components that are specific to using the hyperparameter tuning service.

- **The script imports the** `hypertune` **library. Note that the Dockerfile from Step 1 included instructions to pip install this library.**

- The function `get_args()` defines a command-line argument for each hyperparameter you want to tune. In this example, the hyperparameters that will be tuned are the learning rate, the momentum value in the optimizer, and the number of neurons in the last hidden layer of the model, but feel free to experiment with others. The value passed in those arguments is then used to set the corresponding hyperparameter in the code.

- At the end of the `main()` function, the `hypertune` library is used to define the metric you want to optimize. In TensorFlow, the keras `model.fit` method returns a `History` object. The `History.history` attribute is a record of training loss values and metrics values at successive epochs. If you pass validation data to `model.fit` the `History.history` attribute will include validation loss and metrics values as well. For example, if you trained a model for three epochs with validation data and provided `accuracy` as a metric, the `History.history` attribute would look similar to the following dictionary.

```
{
 "accuracy": [
   0.7795261740684509,
   0.9471358060836792,
   0.9870933294296265
 ],
 "loss": [
   0.6340447664260864,
   0.16712145507335663,
   0.04546636343002319
 ],
 "val_accuracy": [
   0.3795261740684509,
   0.4471358060836792,
   0.4870933294296265
 ],
 "val_loss": [
   2.044623374938965,
   4.100203514099121,
   3.0728273391723633
 ]
```

If you want the hyperparameter tuning service to discover the values that maximize the model's validation accuracy, you define the metric as the last entry (or `NUM_EPOCS - 1`) of the `val_accuracy` list. Then, pass this metric to an instance of `HyperTune`. You can pick whatever string you like for the `hyperparameter_metric_tag`, but you'll need to use the string again later when you kick off the hyperparameter tuning job.

Build the container

1. **From your Terminal, run the following to define an env variable for your project, making sure to replace** `your-cloud-project` **with your project ID:**

Note: You can get your project ID by running `gcloud config list --format 'value(core.project)'` **in your terminal.**

```
PROJECT_ID='your-cloud-project'
```

Copied!
content_copy

2. **Define a variable with the URI of your container image in Google Container Registry:**

```
IMAGE_URI="gcr.io/$PROJECT_ID/horse-human:hypertune"
```

Copied!
content_copy

3. **Then, build the container by running the following from the root of your horses_or_humans directory:**

```
docker build ./ -t $IMAGE_URI
```

Copied!
content_copy

4. **Lastly, push it to Google Container Registry:**

```
docker push $IMAGE_URI
```

Copied!
content_copy
With the container pushed to Container Registry, you're now ready to kick off a custom model hyperparameter tuning job.

Task 3. Run a hyperparameter tuning job on Vertex AI

This lab uses custom training via a custom container on Google Container Registry, but you can also run a hyperparameter tuning job with the Pre-built containers.

- **In cloud console, navigate to the Models section in the Vertex AI.**

Configure training job

1. Click Create to enter the parameters for your hyperparameter tuning job:

    - Under Dataset, select No managed dataset.
    - Select Custom training (advanced) as your training method and click Continue.
    - Enter `horses-humans-hyptertune` (or whatever you'd like to call your model) for Model name.
    - Click Continue.
2. In the Training container step, select Custom container:

    - In the Custom container settings, for Container image, enter the value of your IMAGE_URI variable from the previous section. It should be: `gcr.io/<your-cloud-project>/horse-human:hypertune`, with your own project name. Leave the rest of the fields blank and click Continue.

Configure hyperparameter tuning job

- Select Enable hyperparameter tuning.

Configure hyperparameters

Next, you'll need to add the hyperparameters that you set as command line arguments in the training application code. When adding a hyperparameter, you'll first need to provide the name. This should match the argument name that you passed to `argparse`.

1. Enter `learning_rate` for Parameter name.

2. Select Double as Type.

3. Enter `0.01` for Min, and `1` for Max.

4. Select Log as Scaling.

5. Click DONE.

6. After adding the `learning_rate` hyperparameter, add parameters for `momentum` and `num_neurons`.

- **For momentum:**
    - **Click ADD NEW PARAMETER.**
    - **Enter** `momentum` **for Parameter name.**
    - **Select Double as Type.**
    - **Enter** `0` **for Min, and** `1` **for Max.**
    - **Select Linear as Scaling.**
    - **Click DONE.**
- **For num_neurons:**
    - **Click ADD NEW PARAMETER.**
    - **Enter** `num_neurons` **for Parameter name.**
    - **Select Discrete as Type.**
    - **Enter** `64,128,512` **for Values.**
    - **Select No scaling as Scaling.**
    - **Click DONE.**

Configure Metric

After adding the hyperparameters, you'll next provide the metric you want to optimize as well as the goal. This should be the same as the hyperparameter_metric_tag you set in your training application.

1. **Enter accuracy for Metric to optimize.**
2. **Select Maximize as Goal.**

The Vertex AI Hyperparameter tuning service will run multiple trials of your training application with the values configured in the previous steps. You'll need to put an upper bound on the number of trials the service will run.

More trials generally leads to better results, but there will be a point of diminishing returns after which additional trials have little or no effect on the metric you're trying to optimize. It is a best practice to start with a smaller number of trials and get a sense of how impactful your chosen hyperparameters are before scaling up to a large number of trials.

You'll also need to set an upper bound on the number of parallel trials. Increasing the number of parallel trials will reduce the amount of time the hyperparameter tuning job takes to run; however, it can reduce the effectiveness of the job over all. This is because the default tuning strategy uses results of previous trials to inform the assignment of values in subsequent trials. If you run too many trials in parallel, there will be trials that start without the benefit of the result from the trials still running.

3. **For demonstration purposes, you can set the Maximum number of trials to be** `15` **and the maximum number of parallel trials to be** `3`**. You can**

experiment with different numbers, but this can result in a longer tuning time and higher cost.

4. Select Default as the Algorithm, which will use Google Vizier to perform Bayesian optimization for hyperparameter tuning. You can learn more about this algorithm from the blog [Hyperparameter tuning in Cloud Machine Learning Engine using Bayesian Optimization](#).
5. Click Continue.

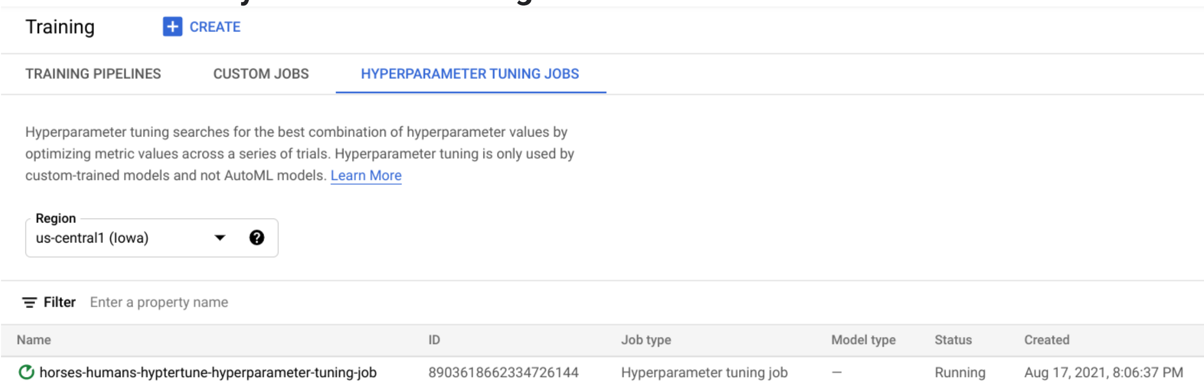
Configure compute

In Compute and pricing, leave the selected region as-is and configure Worker pool 0 as follows:

1. For Machine type, select Standard > n1-standard-4.
2. For Accelerator type, select NVIDIA_TESLA_T4.
3. For Acceleraor count, select 1 .
4. For Disk type, select SSD .
5. For Disk size (GB), enter `100` .

Note: Using the GPU is completely optional. The hyperparameter tuning job will just take a little longer to complete if you do not use the GPU.

6. Click START TRAINING to kick off the hyperparameter tuning job. In the Training section of your console under the HYPERPARAMETER TUNING JOBS tab you'll see something like this:



Note: The hyperparameter tuning job will take about 30-40 minutes to complete. When it's finished, you'll be able to click on the job name and see the results of the tuning trials.

Hyperparameter tuning trials

Filter Enter property name or value

Trial ID	accuracy ↓	Training step	learning_rate	momentum	num_neurons	Log
5	0.863	10	3.0998846321388662e-2	7.542808033097481e-1	512	View logs
8	0.863	10	1.5500957671284583e-2	9.048156494355999e-1	512	View logs
10	0.844	10	2.797941182764173e-2	1.9129967704805848e-1	128	View logs
12	0.828	10	2.1148856220143585e-2	-	128	View logs
15	0.816	10	1.582151532529689e-2	7.855723676903749e-1	128	View logs
13	0.727	10	3.629441074615115e-2	-	512	View logs
1	0.5	10	1.0000000000000002e-1	5e-1	128	View logs
2	0.5	10	1.4261924857754274e-1	7.125085901213717e-1	128	View logs
3	0.5	10	1.3541276772773478e-1	2.718250139251958e-1	512	View logs
4	0.5	10	4.3413513896765965e-1	9.54218711640798e-1	128	View logs

Rows per page: 10 1 – 10 of 15

Task 4. Cleanup

1. If you'd like to continue using the notebook you created in this lab, it is recommended that you turn it off when not in use. From the Notebooks UI in your Cloud Console, select the notebook and then select Stop.

   If you'd like to delete the notebook entirely, simply click the Delete button in the top right.

2. To delete the Storage Bucket, go to Navigation menu > Cloud Storage, select your bucket, and click Delete.

Congratulations!

You've learned how to use Vertex AI to:

Launch a hyperparameter tuning job for training code provided in a custom container. You used a TensorFlow model in this example, but you can train a model built with any framework using custom or built-in containers. To learn more about different parts of Vertex, check out the Vertex AI documentation.

End your lab

When you have completed your lab, click End Lab. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click Submit.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the Support tab.