

19-cu Fəsil. TensorFlow Modellərinin Miqyasda Təlimi və İstifadəyə Verilməsi

Bir gözəl modeliniz varsa və o, möhtəşəm proqnozlar verirsə, onunla nə etməlisiniz? Yaxşı, onu istehsalata buraxmalısınız! Bu, sadəcə olaraq modeli bir data paketində işlətmək və ya bəlkə də bu modeli hər gecə işlədən bir skript yazmaq qədər sadə ola bilər. Lakin, bu çox vaxt daha mürəkkəb olur. İnfrastrukturunuzun müxtəlif hissələri bu modeli canlı məlumatlar üzərində istifadə etməli ola bilər, belə bir halda siz yəqin ki, modelinizi veb xidmətə bələd etmək istəyəcəksiniz: bu şəkildə infrastrukturunuzun istənilən hissəsi sadə bir REST API (və ya başqa bir protokol) istifadə edərək hər hansı bir zamanda modeldən sorğu göndərə bilər, 2-ci Fəsildə danışdığımız kimi. Ancaq zaman keçidkə, modelinizi təzə məlumatlarla müntəzəm olaraq yenidən öyrətmək və yenilənmiş versiyani istehsalata göndərmək lazımlı olacaq. Model versiyalarını idarə etməli, bir modeldən digərinə zərif kecid etməli, problem yaranarsa əvvəlki modelə geri dönməli və bəlkə də A/B sınallarını yerinə yetirmək üçün bir neçə müxtəlif modeli paralel olaraq işlətməli olacaqsınız. Məhsulunuz uğur qazanarsa, xidmətiniz saniyədə çox sayıda sorğu (QPS) qəbul etməyə başlaya bilər və bu yükü dəstəkləmək üçün miqyaslanmalıdır. Xidmətinizi miqyaslaşdırmaq üçün əla bir həll, bu fəsildə görəcəyiniz kimi, TF Serving-dən istifadə etməkdir, istər öz hardware infrastrukturunuzda, istərsə də Google Vertex AI kimi bir bulud xidməti vasitəsilə. O, modelinizi effektiv şəkildə xidmət etmək, model keçidlərini zərif idarə etmək və daha çoxunu təmin edəcək. Bulud platformasından istifadə etsəniz, əlavə olaraq güclü monitoring alətləri kimi bir çox əlavə xüsusiyyətlər də əldə edəcəksiniz.

Üstəlik, əgər çox sayıda təlim məlumatlarınız və hesablama baxımından intensiv modelləriniz varsa, təlim vaxtı dözülməz dərəcədə uzun ola bilər. Məhsulunuz dəyişikliklərə tez uyğunlaşmalı olarsa, uzun təlim vaxtı problem ola bilər (məsələn, keçən həftənin xəbərlərini təbliğ edən xəbər tövsiyə sistemi). Bəlkə də daha vacibdir ki, uzun təlim vaxtı yeni fikirləri sınadandan keçirməyinizə mane olacaq. Maşın öyrənməsində (başqa sahələrdə olduğu kimi), əvvəlcədən hansı ideyaların işləyəcəyini bilmək çətindir, buna görə mümkün qədər çoxunu mümkün qədər tez sınadandan keçirməlisiniz. Təlimi sürətləndirmək üçün bir yol hardware sürətləndiricilərindən, məsələn, GPU və ya TPU-lardan istifadə etməkdir. Daha sürətli getmək üçün, hər biri bir neçə hardware sürətləndiricisi ilə təchiz olunmuş bir neçə maşında modeli təlim edə bilərsiniz. TensorFlow-un sadə, lakin güclü paylama strategiyaları API-si bunu asanlaşdırır, bunu görəcəksiniz.

Bu fəsildə biz modellərin necə yerləşdirilməsini müzakirə edəcəyik, əvvəlcə TF Serving istifadə edərək, sonra Vertex AI istifadə edərək. Həmçinin modellərin mobil tətbiqlərə, gömülü cihazlara və veb tətbiqlərə yerləşdirilməsinə qısa bir nəzər salacaqıq. Daha sonra, hesablama sürətini necə artıracağımızı, GPU-lardan istifadə etməyi və paylama strategiyaları API-sindən istifadə edərək modelləri bir neçə cihazda və serverdə təlim etməyi müzakirə edəcəyik. Nəhayət, modelləri miqyasda necə təlim edəcəyimizi və onların hiperparametrlərini necə təkmilləşdirəcəyimizi Vertex AI istifadə edərək öyrənəcəyik. Müzakirə edəcəyimiz çox mövzu var, buna görə gəlin başlayaqq!

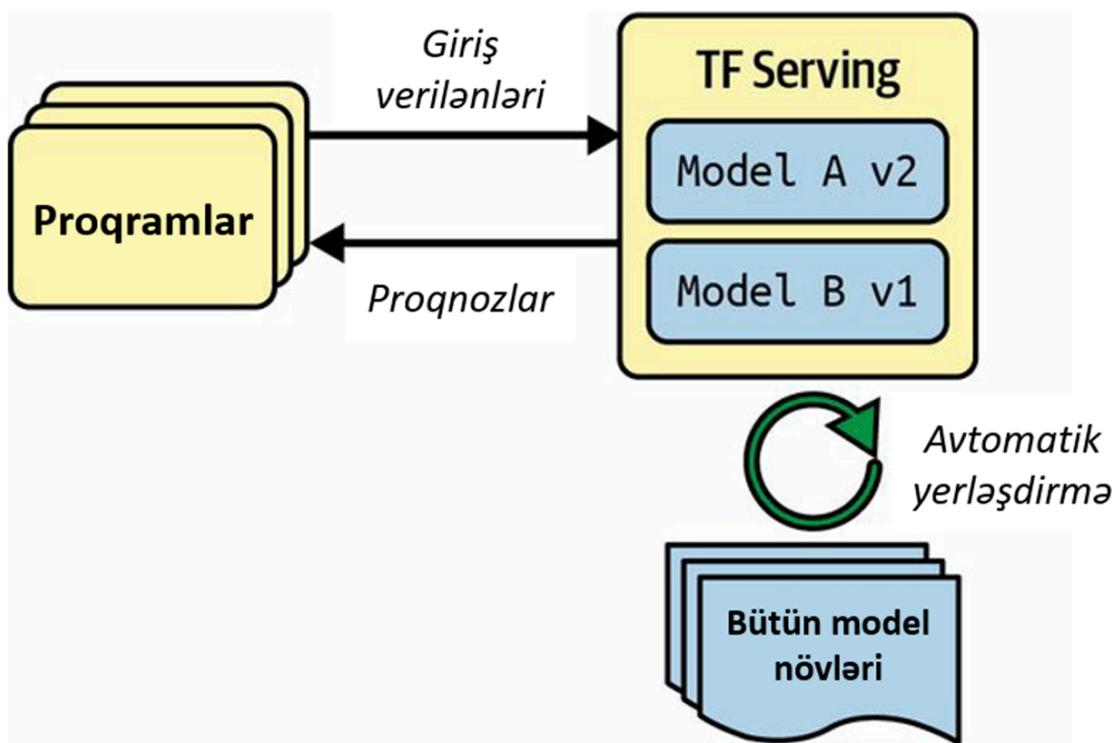
TensorFlow Modelinin İstifadəyə Verilməsi

TensorFlow modelini təlim etdikdən sonra onu istənilən Python kodunda asanlıqla istifadə edə bilərsiniz: əgər bu bir Keras modeli olarsa, sadəcə onun predict() metodunu çağırın! Ancaq infrastrukturunuz böyük, modelinizi yalnız proqnozlar etmək üçün kiçik bir xidmətə bələd etmək

və infrastrukturun qalan hissəsinin bunu (məs. REST və ya gRPC API vasitəsi ilə) sorğulayacağı bir nöqtəyə gəlir. Bu, modelinizi infrastrukturun qalan hissəsindən ayıır, model versiyalarını asanlıqla dəyişdirməyi və ya xidmətinizi lazım olduqda məqyaslaşdırmağı (infrastrukturunuzun qalan hissəsindən müstəqil olaraq), A/B sınallarını həyata keçirməyi və bütün program komponentlərinizin eyni model versiyalarına etibar etməsini təmin edir. Bu, həmçinin test və inkişafı sadələşdirir və daha çoxunu da edir. Öz mikroxdmətinizi istənilən texnologiyadan istifadə edərək yarada bilərsiniz (məsələn, Flask kitabxanasından istifadə edərək), lakin təkəri yenidən icad etməyə nə ehtiyac var ki, sadəcə TF Serving-dən istifadə edə bilərsiniz?

TensorFlow Serving-dən İstifadə Edilməsi

TF Serving çox səmərəli və geniş şəkildə test edilmiş bir model serveridir, C++ dilində yazılmışdır. Bu server yüksək yükləmələri idarə edə bilir, modellərinizin bir neçə versiyasını xidmət göstərə bilir və model anbarını izləyərək ən son versiyaları avtomatik olaraq yerləşdirir və daha çoxunu həyata keçirir (bax Şəkil 19-1).



Şəkil 19-1. TF Serving bir neçə modeli xidmət göstərə bilir və hər modelin ən son versiyasını avtomatik olaraq yerləşdirir.

Beləliklə, Keras istifadə edərək MNIST modelini hazırladığınızı və onu TF Serving-də yerləşdirmək istədiyinizi təsəvvür edək. İlk olaraq bu modeli **SavedModel** formatında ixrac etməlisiniz, bu format **Bölüm 10**-da təqdim olunub.

SavedModel-lərin İxrac Edilməsi

Modelin necə yadda saxlanıldığını artıq bilirsiniz: sadəcə olaraq `model.save()` funksiyasını çağırın. İndi modeli versiyalamaq üçün hər bir model versiyası üçün bir altqovluq yaratmaq lazımdır. Sadədir!

python

Copy code

```
from pathlib import Path

import tensorflow as tf

X_train, X_valid, X_test = [...] # MNIST datasetini yükleyin və
bölün

model = [...] # MNIST modelini qurun və öyrədin (əvvəlcədən şəkil
emalını da idarə edir)

model_name = "my_mnist_model"

model_version = "0001"

model_path = Path(model_name) / model_version

model.save(model_path, save_format="tf")
```

İxrac etdiyiniz son modeldə bütün ön işləmə təbəqələrini daxil etmək adətən yaxşı bir fikirdir, beləliklə, istehsala yerləşdirildikdən sonra təbii formada məlumat qəbul edə bilər. Bu, modeli istifadə edən tətbiq daxilində əvvəlcədən emalın ayrıca idarəsindən qaçınmağa kömək edir. Əvvəlcədən emal addımlarını model daxilində birləşdirmək onları sonradan yeniləməyi daha sadə edir və model ilə tələb olunan əvvəlcədən emal addımları arasında uyğunsuzluq riskini məhdudlaşdırır.

XƏBƏRDARLIQ

Bir SavedModel hesablama qrafikini saxladığı üçün, bu yalnız TensorFlow əməliyyatlarına əsaslanan modellərlə istifadə edilə bilər, `tf.py_function()` əməliyyati istisna olmaqla, bu əməliyyat təsadüfi Python kodunu əhatə edir.

TensorFlow, SavedModel-ləri yoxlamaq üçün kiçik bir `saved_model_cli` komanda xətti interfeysi ilə gəlir. İxrac etdiyimiz modeli yoxlamaq üçün onu istifadə edək:

```
$ saved_model_cli show --dir my_mnist_model/0001
```

Verilən SavedModel aşağıdakı etiket dəstlərinə malikdir:

```
'serve'
```

Bu çıxış nə deməkdir? SavedModel bir və ya bir neçə metaqrafi ehtiva edir. Metagraf hesablama qrafiki və bir neçə funksiya imzası təyinini ehtiva edən bir qrafikdir, onların giriş və çıkış adları, növləri və formalarını da əhatə edir. Hər bir metagraf etiketlər dəsti ilə təyin olunur. Məsələn, tam hesablama qrafiki, o cümlədən öyrədici əməliyyatları ehtiva edən bir metagrafiniz ola bilər: adətən bu, "train" olaraq etiketlənir. Ayrıca, yalnız proqnoz əməliyyatlarını ehtiva edən, GPU-ya xüsusi əməliyyatları da daxil edən bir qrafika malik ola bilərsiniz: bu biri "serve", "gpu" olaraq etiketlənə bilər. Başqa metagraflara da sahib olmaq istəyə bilərsiniz. Bu, TensorFlow-un aşağı səviyyəli SavedModel API-sindən istifadə etməklə həyata keçirilə bilər. Lakin, bir Keras modelini `save()` metodu ilə saxladığınız zaman, yalnız "serve" olaraq etiketlənmiş bir metagraf saxlanılır. Bu "serve" etiket dəstini yoxlayaq:

bash

Copy code

```
$ saved_model_cli show --dir 0001/my_mnist_model --tag_set serve
```

Verilən SavedModel MetaGraphDef aşağıdakı SignatureDef-lərlə açarları ehtiva edir:

```
--saved_model_init_op serving_default
```

Bu metagraf iki imza təyinini ehtiva edir: "`__saved_model_init_op`" adlı bir başlanğıc funksiyası, bu barədə narahat olmayın, və "`serving_default`" adlı standart xidmət funksiyası. Bir Keras modelini saxladığınız zaman, standart xidmət funksiyası modelin `call()` metodudur, bu, proqnozlar verir, artıq bildiyiniz kimi. Bu xidmət funksiyası haqqında daha ətraflı məlumat alaq:

bash

Copy code

```
$ saved_model_cli show --dir 0001/my_mnist_model --tag_set serve \
--signature_def serving_default
```

```
Verilən SavedModel SignatureDef aşağıdakı giriş(ər)i ehtiva edir: inputs['flatten_input']
tensor_info: dtype: DT_UINT8 shape: (-1, 28, 28) name:
serving_default_flatten_input:0
```

```
Verilən SavedModel SignatureDef aşağıdakı çıxış(lar)ı ehtiva edir: outputs['dense_1']
tensor_info: dtype: DT_FLOAT shape: (-1, 10) name: StatefulPartitionedCall:0
```

Metod adı: `tensorflow/serving/predict`

Diqqət yetirin ki, funksiyanın girişi "flatten_input" adlanır və çıxışı "dense_1" adlanır. Bunlar Keras modelinin giriş və çıkış qatlarının adlarına uyğundur. Giriş və çıkış məlumatlarının növü və forması da görünə bilər. Əla görünür!

İndi sənin SaveModel in var, indi isə TF Serving i quraşdırmaq vaxtıdır.

TensorFlow Serving quraşdırılması və başlatılması

TensorFlow Serving (TF Serving) quraşdırmaq üçün bir neçə üsul mövcuddur: sistemin paket menecerindən istifadə etmək, Docker şəkilindən istifadə etmək, mənbədən quraşdırmaq və s. Colab Ubuntu üzərində işlədiyi üçün Ubuntu-nun apt paket menecerindən istifadə edə bilərik:

bash

Copy code

```
url = "https://storage.googleapis.com/tensorflow-serving-apt"

src = "stable tensorflow-model-server
tensorflow-model-server-universal"

!echo 'deb {url} {src}' >
/etc/apt/sources.list.d/tensorflow-serving.list

!curl '{url}/tensorflow-serving.release.pub.gpg' | apt-key add -
!apt update -q && apt-get install -y tensorflow-model-server

%pip install -q -U tensorflow-serving-api
```

Bu kod TensorFlow-nun paket anbarını Ubuntu-nun paket mənbələri siyahısına əlavə edir. Daha sonra TensorFlow-nun ictimai GPG açarını yükləyir və paket menecerinin açar siyahısına əlavə edir ki, TensorFlow-nun paket imzalarını təsdiqləyə bilsin. Sonra apt-dən istifadə edərək `tensorflow-model-server` paketini quraşdırır. Nəhayət, `tensorflow-serving-api` kitabxanasını quraşdırırıq, bu kitabxanaya serverlə ünsiyyət qurmaq üçün ehtiyacımız olacaq.

İndi serveri başlatmaq istəyirik. Komanda əsas model direktori (yəni `my_mnist_model` yolunu, 0001 deyil) üçün tam yolu tələb edəcək, buna görə bu yolu `MODEL_DIR` mühit dəyişəninə saxlayaq:

python

Copy code

```
import os

os.environ["MODEL_DIR"] = str(model_path.parent.absolute())
```

Sonra serveri başlada bilərik:

bash

Copy code

```
%%bash --bg

tensorflow_model_server \
--port=8500 \
--rest_api_port=8501 \
--model_name=my_mnist_model \
--model_base_path="${MODEL_DIR}" >my_server.log 2>&1
```

Jupyter və ya Colab-də `%%bash --bg` magiya komutu hüceyrəni bash skripti kimi icra edir və arxa planda işləyir. `>my_server.log 2>&1` hissəsi standart çıxışı və standart səhvləri `my_server.log` faylına yönəldir. Və bu qədər! TF Serving indi arxa planda işləyir və qeydləri `my_server.log` faylında saxlanılır. O, MNIST modelimizi (versiya 1) yüklədi və indi müvafiq olaraq gRPC və REST sorğularını port 8500 və 8501-də gözləyir.

TF Serving-i Docker konteynerində işlətmək

Əgər öz maşınınızda notebooku işlədirsinizsə və Docker quraşdırılırsa, `docker pull tensorflow/serving` əmrini terminalda işlətdikdən sonra TF Serving şəkilini yükleyə bilərsiniz. TensorFlow komandası bu quraşdırma üsulunu tövsiyə edir, çünki bu sadədir, sisteminizi qarışdırmas və yüksək performans təklif edir. Docker konteynerində serveri başlatmaq üçün terminalda aşağıdakı əmri işlətdik:

bash

Copy code

```
$ docker run -it --rm -v  
"/path/to/my_mnist_model:/models/my_mnist_model" \  
-p 8500:8500 -p 8501:8501 -e MODEL_NAME=my_mnist_model  
tensorflow/serving
```

Buradakı əmr sətri seçən variantların mənaları bunlardır:

- **-it**: Konteyneri interaktiv edir (Ctrl-C ilə dayandırmaq olar) və serverin çıkışını göstərir.
- **--rm**: Konteyneri dayandırğıınızda silir: maşınınızı dayandırılmış konteynerlərlə qarışdırmasın. Lakin, şəkili silmir.
- **-v "/path/to/my_mnist_model:/models/my_mnist_model"**: Host-un **my_mnist_model** direktori konteynerdə **/models/my_mnist_model** yolunda mövcud olacaq. Siz **/path/to/my_mnist_model** ilə bu direktori mütləq yol ilə əvəz etməlisiniz.
- **-p 8500:8500**: Docker mühərrikinin host-un TCP port 8500-ni konteynerin TCP port 8500-ə yönəldir.
- **-p 8501:8501**: Host-un TCP port 8501-i konteynerin TCP port 8501-ə yönəldir.
- **-e MODEL_NAME=my_mnist_model**: Konteynerin **MODEL_NAME** mühit dəyişənini təyin edir ki, TF Serving hansı modeli təqdim edəcəyini bilsin. Əsasən **/models** direktoryasında modelləri axtarır və ən son versiyani avtomatik təqdim edir.
- **tensorflow/serving**: İşlədiləcək şəkil adı.

Server başa çatdıqda, əvvəlcə REST API ilə sonra isə gRPC API ilə sorğulayaq.

TF Serving-i REST API ilə sorğulamaq

Sorğunu yaratmaq üçün, çağrımaq istədiyiniz funksiya imzasının adını və əlbəttə ki, giriş məlumatlarını daxil etməlisiniz. Sorğu JSON formatında olmalıdır, buna görə giriş şəkillərini NumPy massivindən Python siyahısına çevirməliyik:

python

Copy code

```
import json  
  
X_new = X_test[ :3] # təxminən 3 yeni rəqəm şəkli təsnif etməyə  
çalışırıq  
  
request_json = json.dumps({
```

```
    "signature_name": "serving_default",
    "instances": X_new.tolist(),
}
```

JSON formatı 100% mətn əsaslıdır. Sorğu stringi belə görünür:

python

Copy code

```
>>> request_json

'{"signature_name": "serving_default", "instances": [[[0, 0, 0, 0,
... ]]]}'
```

İndi bu sorğunu TF Serving-ə HTTP POST sorğusu ilə göndərəcəyik. Bu, requests kitabxanası ilə edilə bilər (Python-un standart kitabxanasına daxil deyil, lakin Colab-da quraşdırılıb):

python

Copy code

```
import requests

server_url =
"http://localhost:8501/v1/models/my_mnist_model:predict"

response = requests.post(server_url, data=request_json)

response.raise_for_status() # səhv halında istisna qaldırır

response = response.json()
```

Hər şey yaxşı gedərsə, cavab bir "predictions" açarı olan bir lügət olacaq. Bu açarın müvafiq dəyəri isə təsnifatların siyahısıdır. Bu siyahı Python siyahısıdır, buna görə bunu NumPy massivinə çevirmək və saxlanmış float-ları ikinci onluğa yuvarlaqlaşdırmaq olar:

python

Copy code

```
>>> import numpy as np
```

```
>>> y_proba = np.array(response["predictions"])

>>> y_proba.round(2)

array([[0. , 0. , 0. , 0. , 0. , 0. , 1. , 0. , 0. ],
       [0. , 0. , 0.99, 0.01, 0. , 0. , 0. , 0. , 0. ],
       [0. , 0.97, 0.01, 0. , 0. , 0. , 0.01, 0. , 0. ]])
```

Uğurla, biz təsnifatları əldə etdik! Model ilk şəkilin 7 olduğunu təxminən 100% əmin, ikinci şəkilin 2 olduğunu 99% əmin və üçüncü şəkilin 1 olduğunu 97% əmindir. Bu doğrudur.

REST API yaxşı və sadədir, və giriş və çıkış məlumatları çox böyük olmadıqda yaxşı işləyir. Həmçinin, demək olar ki, hər hansı bir müştəri tətbiqi REST sorğuları edə bilər, əlavə asılılıqlara ehtiyac yoxdur, digər protokollar isə hər zaman belə əlçatan olmur. Lakin, JSON əsaslıdır və olduqca ətraflıdır. Məsələn, NumPy massivini Python siyahısına çevirməli olduğ və hər float string kimi təmsil olundu. Bu həm serializasiya/deserializasiya vaxtı, həm də yükün ölçüsü baxımından çox qeyri-effektivdir. Bu yüksək gecikmə və bant genişliyi istifadəsinə səbəb ola bilər. Buna görə gRPC istifadə edək.

TF Serving-i gRPC API ilə sorğulamaq

gRPC API seriyalasdırılmış `PredictRequest` protokolunun açarı gözləyir və seriyalasdırılmış `PredictResponse` protokolunu çıxarır. Bu protobuf-lar əvvəlcə quraşdırduğumuz `tensorflow-serving-api` kitabxanasındadır.

İlk növbədə sorğunu yaradacaqıq:

python

Copy code

```
from tensorflow_serving.apis.predict_pb2 import PredictRequest

request = PredictRequest()

request.model_spec.name = model_name

request.model_spec.signature_name = "serving_default"

input_name = model.input

request.inputs[input_name].CopyFrom(tf.make_tensor_proto(X_new))
```

Bu kod PredictRequest protokol bufferini yaradır və tələb olunan sahələri doldurur, o cümlədən əvvəlcədən müəyyən edilmiş model adı, çağrımaq istədiyimiz funksiyanın imzası və nəhayət, giriş məlumatları, Tensor protokol bufferi şəklində. `tf.make_tensor_proto()` funksiyası verilmiş tensor və ya NumPy array əsasında Tensor protokol bufferi yaradır, bu halda `X_new`.

Sonra, tələbi serverə göndərəcəyik və cavabı alacaqıq. Bunun üçün, Colab-da öncədən quraşdırılmış grpcio kitabxanasına ehtiyacımız var:

python

Copy code

```
import grpc

from tensorflow_serving.apis import prediction_service_pb2_grpc

channel = grpc.insecure_channel('localhost:8500')

predict_service =
prediction_service_pb2_grpc.PredictionServiceStub(channel)

response = predict_service.Predict(request, timeout=10.0)
```

Kod olduqca sadədir: idxal etdikdən sonra, localhost-da TCP port 8500 üzərində gRPC əlaqə kanalını yaradırıq, sonra bu kanalda gRPC xidməti yaradırıq və bunu istifadə edərək tələbi göndəririk, 10 saniyəlik vaxt həddi ilə. Qeyd edin ki, çağrış sinxron olur: cavabı alana qədər bloklanacaq və ya vaxt həddi bitəcək. Bu nümunədə kanal qeyri-əmin (şifrlənmə, autentifikasiya yoxdur), lakin gRPC və TF Serving SSL/TLS üzərindən təhlükəsiz kanalları da dəstəkləyir.

Sonra, PredictResponse protokol bufferini tensora çevirmək üçün:

python

Copy code

```
output_name = model.output_names[0] # == "dense_1"

outputs_proto = response.outputs[output_name]

y_proba = tf.make_ndarray(outputs_proto)
```

Bu kodu işlətsəniz və `y_proba.round(2)` çap etsəniz, əvvəller olduğu kimi dəqiq eynilə qiymətləndirilmiş sinif ehtimallarını alacaqsınız. Və bu qədər: cəmi bir neçə sətir kodla, TensorFlow modelinizi uzaqdan, REST və ya gRPC istifadə edərək əldə edə bilərsiniz.

Yeni Model Versiyasını Yayınlama

İndi yeni bir model versiyası yaradacaqıq və SavedModel-i bu dəfə `my_mnist_model/0002` qovluğununa ixrac edəcəyik:

python

Copy code

```
model = [...] # Yeni MNIST model versiyasını qurun və təlim edin  
model_version = "0002"  
  
model_path = Path(model_name) / model_version  
  
model.save(model_path, save_format="tf")
```

Müntəzəm aralıqlarla (gözləmə müddəti konfiqurasiya edilə bilər) TF Serving model qovluğununu yeni model versiyaları üçün yoxlayır. Əgər taparsa, keçidi avtomatik olaraq idarə edir: standart olaraq, gözləyən tələbləri (əgər varsa) əvvəlki model versiyası ilə cavablandırır, yeni tələbləri isə yeni versiya ilə idarə edir. Hər bir gözləyən tələb cavablandırıldıqda, əvvəlki model versiyası yüklenir. Bunu TF Serving loglarında (`my_server.log`-də) görə bilərsiniz:

css

Copy code

[...]

```
SavedModel oxunur: /models/my_mnist_model/0002
```

```
Meta qraf oxunur { serve }
```

[...]

```
Üğurla yüklenmiş servable versiya {name: my_mnist_model version: 2}
```

```
Quiescing servable versiya {name: my_mnist_model version: 1}
```

```
Done quiescing servable versiya {name: my_mnist_model version: 1}
```

```
Unloading servable versiya {name: my_mnist_model version: 1}
```

TÖVSİYƏ Əgər SavedModel-də assets/extrə qovluğununda bəzi nümunə instansiyalar varsa, TF Serving-in yeni modelin bu instansiyalarda işlədilməsini konfiqurasiya edə bilərsiniz, beləliklə

xidmətə başlamadan əvvəl hər şey düzgün yüklənəcək, ilk tələblərin uzun cavab vaxtlarını qarşısını alacaq.

Bu yanaşma hamar keçid təqdim edir, lakin çox RAM istifadə edə bilər — xüsusilə GPU RAM, ümumiyyətlə ən məhdud olanı. Bu halda, TF Serving-i əvvəlki model versiyası ilə bütün gözləyən tələbləri idarə edəcək və sonra yeni model versiyasını yükləyəcək şəkildə konfiqurasiya edə bilərsiniz. Bu konfiqurasiya eyni anda iki model versiyasının yüklənməsinin qarşısını alacaq, lakin xidmət qısa müddətə əlcətan olmayıcaq.

Göründüyü kimi, TF Serving yeni modellərin yerləşdirilməsini asanlaşdırır. Üstəlik, əgər versiya 2 gözlədiyiniz qədər yaxşı işləməzsə, onda versiya 1-ə geri dönmək my_mnist_model/0002 qovluğununu silmək qədər sadədir.

TÖVSIYƏ TF Serving-in başqa bir əla xüsusiyyəti avtomatik yığma qabiliyyətidir, bunu başlatma zamanı --enable_batching seçimini istifadə edərək aktiv edə bilərsiniz. TF Serving bir qısa müddət ərzində bir neçə tələb aldıqda (gözləmə müddəti konfiqurasiya edilə bilər), onları model istifadə etməzdən əvvəl avtomatik olaraq yığır. Bu, GPU-nun gücündən istifadə edərək əhəmiyyətli bir performans artışı təqdim edir. Model proqnozları qaytarıldıqda, TF Serving hər bir proqnozu doğru müştəriyə yönləndirir. Daha çox throughput əldə etmək üçün yığma gözləmə müddətini artırmaqla müəyyən dərəcədə gecikməni ticarət edə bilərsiniz (baxın --batching_parameters_file seçimi).

Əgər saniyədə çox sayıda sorğu gözləyirsinizsə, TF Serving-i bir neçə serverdə yerləşdirmək və sorğuları yükləmə balansını təmin etmək (Şəkil 19-2-ə baxın) istəyəcəksiniz. Bu, bu serverlərdə çox sayıda TF Serving konteynerini yerləşdirmək və idarə etmək tələb edir. Bunun üçün Kubernetes kimi bir alətdən istifadə edə bilərsiniz, bu çox server arasında konteyner orkestrasiya işlərini sadələşdirən açıq mənbə sistemidir. Bütün avadanlıq infrastrukturunu satın almaq, saxlanmaq və yeniləmək istəmirsinizsə, bulud platformasında virtual maşınlardan istifadə etmək istəyəcəksiniz, məsələn, Amazon AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud, Alibaba Cloud, Oracle Cloud və ya digər bir PaaS təklif edən platforma. Bütün virtual maşınları idarə etmək, konteyner orkestrasiya (Kubernetes küməyi ilə belə), TF Serving konfiqurasiyası, tənzimləmə və izləmə - bunlar tam zamanlı iş ola bilər. Xoşbəxtlikdən, bəzi xidmət təminatçıları bunların hamısını sizin üçün həyata keçirə bilər. Bu fəsildə Vertex AI istifadə edəcəyik: bu gün TPUs olan yeganə platformadır; TensorFlow 2, Scikit-Learn və XGBoost-u dəstəkləyir; və yaxşı bir AI xidmətləri paketini təklif edir. Bu sahədə TensorFlow modellərini xidmət edə bilən bir neçə digər təminatçı da var, məsələn, Amazon AWS SageMaker və Microsoft AI Platform, buna görə də onları da yoxlamağı unutmayın.



Şəkil 19-2. Sorğu Balanslayıcısı ilə TF Serving-in miqyaslanması

Şəkil 19-2. Yük balanslaşdırması ilə TF Serving-in genişləndirilməsi

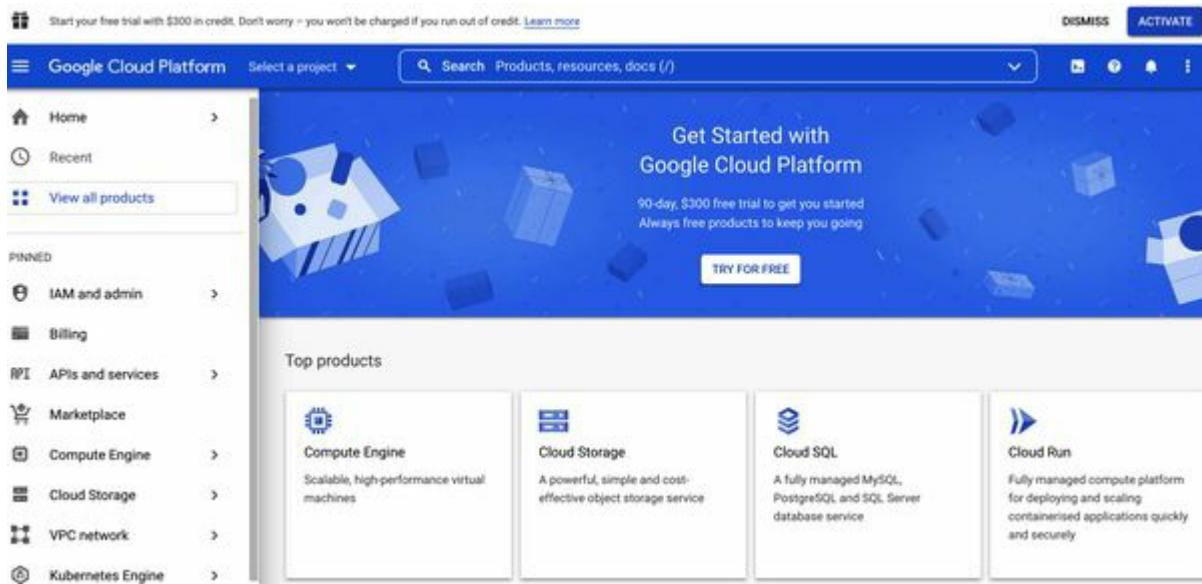
İndi gözəl MNIST modelimizi buludda necə işlədəcəyimizi görək!

Vertex AI-də Proqnoz Xidmətinin Yaradılması

Vertex AI, Google Cloud Platform (GCP) daxilində geniş çəşidli AI ilə əlaqəli alətlər və xidmətlər təqdim edən bir platformadır. Siz məlumat dəstlərini yükləyə, insanlardan etiketləmə alına, tez-tez istifadə olunan xüsusiyyətləri bir xüsusiyyət mağazasında saxlaya və bunları təlim və ya istehsalda istifadə edə, çoxlu GPU və ya TPU serverləri üzərində avtomatik hiperparametr tənzimləməsi və ya model memarlığı axtarışı (AutoML) ilə modelləri təlim edə bilərsiniz. Təlim edilmiş modellərinizi idarə edə, onları böyük miqdarda məlumat üzərində batch proqnozlari vermək üçün istifadə edə, məlumat iş axınlarınız üçün bir neçə iş planlaşdırıra, modellərinizi REST və ya gRPC vasitəsilə geniş miqyasda təqdim edə və məlumatlarınız və modelləriniz ilə host edilmiş Jupyter mühitində (Workbench) təcrübə edə bilərsiniz. Vektorları çox səmərəli müqayisə etməyə imkan verən Matching Engine xidməti də var (yəni, yaxınlıqdakı ən yaxın qonşular). GCP ayrıca kompüter görmə, tərcümə, səsdən mətinə və daha çoxu üçün API-lər daxil olmaqla digər AI xidmətlərini də təqdim edir.

Başlamazdan əvvəl bir neçə qurğulama işi var:

1. Google hesabınıza daxil olun və sonra Google Cloud Platform konsoluna keçin (Şəkil 19-3-ə baxın). Əgər Google hesabınız yoxdursa, birini yaratmalısınız.
2. Əgər GCP-ni ilk dəfə istifadə edirsinzə, şərtləri oxuyub qəbul etməlisiniz. Yeni istifadəçilərə 90 gün ərzində istifadə edə biləcəyiniz \$300 dəyərində GCP krediti təklif olunur (May 2022 tarixindən etibarən). Bu fəsildə istifadə edəcəyiniz xidmətlər üçün bu məbləğin kiçik bir hissəsinə ehtiyacınız olacaq. Pulsuz sınaq üçün qeydiyyatdan keçidkən sonra, hələ də bir ödəmə profili yaratmalı və kredit kartı nömrənizi daxil etməlisiniz: bu, doğrulama məqsədilə istifadə olunur—bəlkə də insanların pulsuz sınaqdan dəfələrlə istifadə etmələrinin qarşısını almaq üçün—amma ilk \$300 üçün hesabınızdan ödəniş edilmir, və bundan sonra yalnız ödənişli hesab seçsəniz sizə haqq ediləcək.



Şəkil 19-3. Google Cloud Platform konsolu

3. Əgər GCP-ni əvvəller istifadə etmisinizsə və pulsuz sınaq müddətiniz bitib, bu fəsildə istifadə edəcəyiniz xidmətlər sizə bir qədər maliyyətə başa gələ bilər. Bu, çox olmamalıdır, xüsusən də xidmətləri artıq lazım olmadıqda söndürməyi unutmadığınız halda. Xidmətləri işə salmadan əvvəl qiymət şərtlərini başa düşdürüyünüzə və razılışdırığınıza əmin olun. Xidmətlərin gözlədiyinizdən daha çox xərcə başa gəlməsi halında məsuliyyəti qəbul etmirəm! Eyni zamanda, ödəniş hesabınızın aktiv olduğuna əmin olun. Bunu yoxlamaq üçün, sol üst küncdəki **≡** naviqasiya menyusunu açın və “Billing” (Ödəniş) seçin, sonra ödəniş metodunu təyin etdiyinizə və ödəniş hesabınızın aktiv olduğuna əmin olun.
4. GCP-dəki hər bir resurs bir layihəyə aiddir. Bu, istifadə edə biləcəyiniz bütün virtual maşınları, saxladığınız faylları və apardığınız təlim işlərini əhatə edir. Hesab yaradarkən, GCP avtomatik olaraq sizin üçün “My First Project” (Mənim İlk Layihəm) adında bir layihə yaradır. İstəsəniz, layihənin görüntüsü adını layihə parametrlərinə gedərək dəyişə bilərsiniz: **≡** naviqasiya menyusunda “IAM and admin → Settings” (IAM və İdarə → Parametrlər) seçin, layihənin görüntüsü adını dəyişin və “SAVE” (Yadda saxla) düyməsini basın. Layihənin həmçinin unikal bir ID və nömrəsi var. Layihə yaratdığınız zaman layihə ID-sini seçə bilərsiniz, lakin sonradan dəyişdirmək mümkün deyil. Layihə nömrəsi avtomatik olaraq yaradılır və dəyişdirilə bilmir. Yeni layihə yaratmaq istəyirsinzsə, səhifənin üst hissəsindəki layihə adını klikləyin, sonra “NEW PROJECT” (YENİ LAYİHƏ) seçin və layihənin adını daxil edin. Layihə ID-sini təyin etmək üçün “EDIT” (DÜZENLƏ) seçimini də klikləyə bilərsiniz. Bu yeni layihə üçün ödənişlərin aktiv olduğunu əmin olun ki, xidmət haqqı (varsə, pulsuz kreditlərinizdən) hesablanabiləsin.

XƏBƏRDARLIQ Xidmətlərin yalnız bir neçə saat lazım olduğunu bildiyiniz zaman onları söndürmək üçün həmişə xəbərdarlıq təyin edin, eks halda günlər və ya aylar ərzində işlək vəziyyətdə qalıb, potensial olaraq əhəmiyyətli xərclərə səbəb ola bilər.

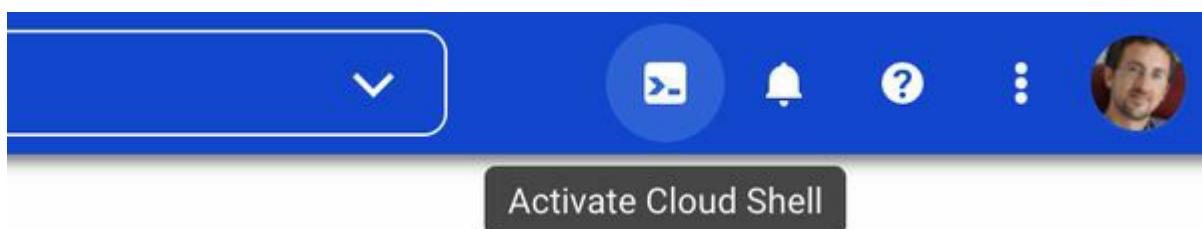
5. İndi sizdə GCP hesabı və layihə var və ödəniş aktivləşdirilmişdir, siz lazım olan API-ləri aktivləşdirməlisiniz. **≡** naviqasiya menyusunda “APIs and services” (API-lər və xidmətlər) seçin və “Cloud Storage API”nın aktiv olduğuna əmin olun. Lazım gələrsə, “+ ENABLE

APIS AND SERVICES” (API-ləri və Xidmətləri AKTİV ET) seçin, “Cloud Storage”ı tapın və aktivləşdirin. Həmçinin “Vertex AI API”ni aktivləşdirin.

Hər şeyi GCP konsolu vasitəsilə etməyə davam edə bilərsiniz, amma Python istifadə etməyi tövsiyə edirəm: beləliklə, GCP ilə istədiyiniz hər şeyi avtomatlaşdırın skriptlər yaza bilərsiniz və bu, tez-tez menyular və formalar vasitəsilə klikləməkdən daha rahatdır, xüsusilə də adi işlər üçün.

GOOGLE CLOUD CLI VƏ SHELL Google Cloud-un komanda xətti interfeysi (CLI) “gcloud” əmri ilə GCP-də demək olar ki, hər şeyi idarə etməyə imkan verir və “gsutil” Google Cloud Storage ilə qarşılıqlı əlaqə qurmağa kömək edir. Bu CLI Colab-də əvvəlcədən quraşdırılmışdır: sadəcə olaraq “google.auth.authenticate_user()” istifadə edərək autentifikasiya etməlisiniz və başlaya bilərsiniz. Məsələn, !gcloud config list konfiqurasiyanı göstərəcək.

GCP həmçinin Google Cloud Shell adlı əvvəlcədən konfiqurasiya olunmuş bir shell mühiti təqdim edir ki, onu birbaşa veb brauzerinizdə istifadə edə bilərsiniz; bu, sizin üçün artıq əvvəlcədən quraşdırılmış və konfiqurasiya edilmiş Google Cloud SDK ilə pulsuz bir Linux VM (Debian) üzərində işləyir, beləliklə, autentifikasiya etməyə ehtiyac yoxdur. Cloud Shell GCP-də hər yerdə mövcuddur: səhifənin sağ üst küncündəki Activate Cloud Shell (Cloud Shell-i Aktivləşdir) ikonunu klikləyin (Şəkil 19-4-ə baxın).



Şəkil 19-4. Google Cloud Shell-i Aktivləşdirmək

GCP xidmətlərindən istifadə etməyə başlamazdan əvvəl edəcəyiniz ilk şey autentifikasiya etməkdir. Colab-dan istifadə edərkən ən sadə həll yolu aşağıdakı kodu icra etməkdir:

```
python
Copy code
from google.colab import auth
auth.authenticate_user()
```

Autentifikasiya prosesi OAuth 2.0 əsaslanır: bir pop-up pəncərəsi sizdən Colab notbukuna Google məlumatlarınızın əldə olunmasına icazə vermək istədiyinizi təsdiqləməyinizi istəyəcək. Əgər razı olsanız, GCP üçün istifadə etdiyiniz eyni Google hesabını seçməlisiniz. Sonra sizdən Colab-a Google Drive və GCP-də olan bütün məlumatlarınıza tam giriş icazəsi verməyinizi təsdiqləməyiniz xahiş ediləcək. Əgər icazə versəniz, yalnız cari notbuka və yalnız Colab runtime-ı bitənə qədər giriş olacaq. Təbii ki, yalnız notbukdakı koda güvəndiyiniz halda bunu qəbul etməlisiniz.

XƏBƏRDARLIQ Əgər siz <https://github.com/ageron/handson-ml3> rəsmi notbukları ilə işləmirsinizsə, çox diqqətli olmalısınız: əgər notbuk müəllifi xain olsa, onlar kodda məlumatlarınızı istədikləri şeyi edə bilərlər.

GCP-də AUTENTİFİKASIYA VƏ İCAZƏ Ümumiyyətlə, OAuth 2.0 autentifikasiyasından istifadə yəniz tətbiqin istifadəçinin şəxsi məlumatlarına və ya başqa bir tətbiqdən resurslara onun adından daxil olmalı olduğu hallarda tövsiyə olunur. Məsələn, bəzi tətbiqlər istifadəçiyə məlumatları Google Drive-a saxlamağa imkan verir, lakin bunun üçün tətbiq əvvəlcə istifadəcidən Google-a autentifikasiya etməsini və Google Drive-a giriş icazəsi verməsini xahiş etməlidir. Ümumiyyətlə, tətbiq yəniz ehtiyacı olan səviyyədə giriş icazəsi istəyəcək; bu, məhdudiyyətsiz giriş olmayıcaq: məsələn, tətbiq yəniz Google Drive-a giriş icazəsi istəyəcək, Gmail-ə və ya digər Google xidmətlərinə deyil. Bundan əlavə, icazə adətən müəyyən bir müddətdən sonra başa çatır və hər zaman geri alınması mümkündür.

Əgər tətbiqin GCP xidmətinə istifadəçi adından deyil, öz adından daxil olması lazımdırsa, o zaman ümumiyyətlə xidmət hesabından istifadə etməlidir. Məsələn, əgər siz bir veb-sayt yaradırsınızsa və o veb-sayt bir Vertex AI endpoint-ə proqnoz sorğuları göndərməlidirsə, onda veb-sayt xidmətə öz adından daxil olacaq. Burada istifadəçinin Google hesabında daxil olmağa ehtiyac olan hər hansı məlumat və ya resurs yoxdur. Əslində, veb-saytin bir çox istifadəçisi heç Google hesabına malik olmayıcaq. Bu sənari üçün əvvəlcə xidmət hesabı yaratmalısınız. GCP konsolunun  naviqasiya menyusunda "IAM və administrasiya → Xidmət hesabları" seçin (və ya axtarış qutusundan istifadə edin), sonra + XİDMƏT HESABI YARAT klikləyin, formanın birinci səhifəsini doldurun (xidmət hesabının adı, ID, təsviri) və YARAT VƏ DAVAM ET klikləyin. Sonra bu hesaba bəzi giriş hüquqları verməlisiniz. "Vertex AI istifadəçi" rolunu seçin: bu, xidmət hesabına proqnozlar verməyə və digər Vertex AI xidmətlərindən istifadə etməyə imkan verəcək, lakin başqa heç nə etməyəcək. DAVAM ET klikləyin. İndi bəzi istifadəçilərə xidmət hesabına giriş hüquqları vermək imkanı var: bu, GCP istifadəçi hesabınız bir təşkilatın bir hissəsi olduqda və siz bu təşkilatdakı digər istifadəçilərə bu xidmət hesabını əsasında tətbiqləri yerləşdirməyi və ya xidmət hesabını idarə etməyi səlahiyyətləndirmək istədiyiniz zaman faydalıdır. Sonra BİTDİ klikləyin.

Xidmət hesabı yaratıldıqdan sonra tətbiqiniz bu xidmət hesabı ilə autentifikasiya olunmalıdır. Bunu etmək üçün bir neçə yol var. Əgər tətbiqiniz GCP-də yerləşdirilibsə, məsələn, əgər siz Google Compute Engine-də yerləşdirilmiş bir veb-sayt kodlayırsınızsa, o zaman ən sadə və təhlükəsiz həll yolu xidmət hesabını veb-saytinizi yerləşdirən GCP resursuna, məsələn, bir VM instansiyasına və ya Google App Engine xidmətinə bağlamaqdır. Bu, GCP resursunu yaratmaq zamanı "Kimlik və API girişi" bölməsində xidmət hesabını seçməklə edilə bilər. Bəzi resurslar, məsələn, VM instansiyaları, xidmət hesabını VM instansiyası yaratıldıqdan sonra da bağlamağa imkan verir: siz onu dayandırmalı və parametrlərini redaktə etməlisiniz. Hər halda, xidmət hesabı VM instansiyasına və ya kodunu işlədən hər hansı digər GCP resursuna bağlandıqdan sonra, GCP-nin müştəri kitabxanaları (qısa müddətdə müzakirə olunacaq) heç bir əlavə addım tələb etmədən avtomatik olaraq seçilmiş xidmət hesabı kimi autentifikasiya edəcək.

Əgər tətbiqiniz Kubernetes istifadə edərək yerləşdirilibsə, o zaman hər bir Kubernetes xidmət hesabına uyğun olan xidmət hesabını xəritələmək üçün Google-un İş Yükü Kimliyi xidmətindən istifadə etməlisiniz. Əgər tətbiqiniz GCP-də yerləşdirilməyib, məsələn, əgər sadəcə Jupyter notbukunu öz kompüterinizdə işlədirsinizsə, o zaman ya İş Yükü Kimliyi Federasiyası xidmətindən istifadə edə bilərsiniz (bu, ən təhlükəsiz, lakin ən çətin seçimdir), ya da sadəcə xidmət hesabınız üçün giriş açarı yarada bilərsiniz, onu JSON faylında saxlayın və müştəri tətbiqinizin ona daxil ola bilməsi

üçün GOOGLE_APPLICATION_CREDENTIALS mühit dəyişənini ona yönəldin. Giriş açarlarını idarə etmək üçün yeni yaratdığınız xidmət hesabını klikləyin və sonra AÇARLAR nişanını açın. Açıfaylınlı gizli saxladığınızdan əmin olun: bu, xidmət hesabı üçün bir parol kimidir.

Tətbiqinizin GCP xidmətlərinə daxil ola bilməsi üçün autentifikasiya və icazənin qurulması haqqında daha ətraflı məlumat üçün sənədləri yoxlayın.

İndi isə SavedModels-ləri saxlamaq üçün bir Google Cloud Storage (GCS) bucket (məlumatlarınız üçün konteyner) yaradaq. Bunun üçün Colab-da əvvəlcədən quraşdırılmış googlecloud-storage kitabxanasından istifadə edəcəyik. Əvvəlcə Client obyektini yaradacaqıq, bu GCS ilə interfeys kimi xidmət edəcək, sonra ondan istifadə edərək bucket yaradacaqıq:

python

Copy code

```
from google.cloud import storage

project_id = "my_project" # bunu layihə ID-nizə dəyişin

bucket_name = "my_bucket" # bunu unikal bucket adına dəyişin

location = "us-central1"

storage_client = storage.Client(project=project_id)

bucket = storage_client.create_bucket(bucket_name,
location=location)
```

MƏSLƏHƏT Əgər mövcud bucket-i yenidən istifadə etmək istəyirsizsə, son sətiri belə əvəz edin:

python

Copy code

```
bucket = storage_client.bucket(bucket_name)
```

Regionu bucket-in yerləşdiyi yerə uyğun təyin etdiyinizdən əmin olun. GCS bucket-lər üçün bütün dünyada vahid adlandırma sistemindən istifadə edir, buna görə də “machine-learning” kimi sadə adlar çox güman ki, mövcud olmayıacaq. Bucket adı DNS adlandırma qaydalarına uyğun olmalıdır, çünkü o, DNS qeydlərində istifadə oluna bilər. Üstəlik, bucket adları ictimaidır, ona görə də adın özündə heç bir şəxsi məlumat yerləşdirməyin. Adın unikal olmasını təmin etmək üçün domen adınızı, şirkət adınızı və ya layihə ID-nizi prefiks kimi istifadə etmək və ya sadəcə adın bir hissəsi kimi təsadüfi rəqəm istifadə etmək yaygındır.

Əgər istəsəniz, regionu dəyişə bilərsiniz, lakin GPU-ları dəstəkləyən birini seçdiyinizə əmin olun. Bundan əlavə, bəzi regionların daha çox CO₂ istehsal etdiyini, bəzi regionların bütün xidmətləri dəstəkləmədiyini və tək regionlu bucket-dən istifadə etməyin performansı yaxşılaşdırduğunu nəzərə alaraq, regionlar arasında qiymətlərin çox dəyişdiyini nəzərə ala bilərsiniz. Əgər əmin deyilsinizsə, "us-central1"-də qalmaq yaxşı ola bilər.

İndi isə my_mnist_model kataloqunu yeni bucket-ə yükləyək. GCS-dəki fayllar bloblar (və ya obyektlər) adlanır və onların hamısı sadəcə olaraq bucket-ə yerləşdirilir, heç bir kataloq strukturu olmadan. Blob adları istənilən Unicode sətri ola bilər, hətta irəli slash-lər (/) də daxil ola bilər. GCP konsolu və digər alətlər bu slash-lərdən kataloqların mövcud olması illüziyasını yaratmaq üçün istifadə edirlər. Beləliklə, my_mnist_model kataloqunu yükləyərkən, yalnız fayllarla maraqlanırıq, kataloqlarla yox:

python

Copy code

```
def upload_directory(bucket, dirpath):  
    dirpath = Path(dirpath)  
  
    for filepath in dirpath.glob("*/*"):  
  
        if filepath.is_file():  
  
            blob =  
            bucket.blob(filepath.relative_to(dirpath.parent).as_posix())  
  
            blob.upload_from_filename(filepath)  
  
upload_directory(bucket, "my_mnist_model")
```

Bu funksiya indi yaxşı işləyir, lakin yüklənəcək çoxlu fayl olsaydı, çox yavaş olardı. Onu çox mövzulama ilə (notbukda həyata keçirilməsini görmək üçün) çox asanlıqla inanılmaz dərəcədə sürətləndirmək olar. Alternativ olaraq, əgər Google Cloud CLI varsa, onda bunun əvəzinə aşağıdakı əmrdən istifadə edə bilərsiniz:

bash

Copy code

```
!gsutil -m cp -r my_mnist_model gs://{{bucket_name}}/
```

İndi isə Vertex AI-yə MNIST modelimiz haqqında məlumat verək. Vertex AI ilə ünsiyyət qurmaq üçün google-cloud-aiplatform kitabxanasından istifadə edə bilərik (hələ də Vertex AI əvəzinə köhnə

AI Platform adından istifadə edir). Colab-da əvvəlcədən quraşdırılmayıb, ona görə də onu quraşdırmaçıq. Bundan sonra kitabxanani idxal edib onu işə sala bilərik—bu, layihə ID-si və yer üçün bəzi standart dəyərləri göstərmək üçün—sonra yeni Vertex AI modeli yarada bilərik: bir ekran adı, modelimiz üçün GCS yolunu (bu halda 0001 versiyası) və modelimizi işlətmək üçün Vertex AI-dən istifadə etmək istədiyimiz Docker konteynerinin URL-sini göstəririk. Əgər həmin URL-ə keçsəniz və bir səviyyə yuxarı naviqasiya etsəniz, istifadə edə biləcəyiniz digər konteynerləri tapa bilərsiniz. Bu biri GPU ilə TensorFlow 2.8 dəstəkləyir:

python

Copy code

```
from google.cloud import aiplatform

server_image =
"gcr.io/cloud-aiplatform/prediction/tf2-gpu.2-8:latest"

aiplatform.init(project=project_id, location=location)

mnist_model = aiplatform.Model.upload(
    display_name="mnist",
    artifact_uri=f"gs://{bucket_name}/my_mnist_model/0001",
    serving_container_image_uri=server_image,
)
```

İndi bu modeli yerləşdirək ki, proqnozlar etmək üçün gRPC və ya REST API vasitəsilə ona sorğular göndərə bilək. Bunun üçün əvvəlcə bir endpoint yaratmaçıq. Bu, müştəri tətbiqlərinin xidmətə daxil olmaq istədikləri zaman qoşulduqları şeydir. Sonra modelimizi bu endpoint-ə yerləşdirməliyik:

python

Copy code

```
endpoint = aiplatform.Endpoint.create(display_name="mnist-endpoint")

endpoint.deploy(
    mnist_model,
    min_replica_count=1,
    max_replica_count=5,
```

```
    machine_type="n1-standard-4",  
    accelerator_type="NVIDIA_TESLA_K80",  
    accelerator_count=1  
)
```

Bu kodun işlənməsi bir neçə dəqiqə çəkə bilər, çünkü Vertex AI virtual maşın qurmalıdır. Bu nümunədə biz n1-standard-4 tipli olduqca əsas maşından istifadə edirik (başqa növlər üçün <https://homl.info/machinetypes> saytına baxın). Biz həmçinin NVIDIA_TESLA_K80 tipli əsas GPU-dan istifadə edirik (başqa növlər üçün <https://homl.info/accelerators> saytına baxın). Əgər "us-central1"-dən başqa bir region seçmisinizsə, onda maşın tipini və ya sürətləndirici tipini həmin regionda dəstəklənən dəyərlərlə dəyişməli ola bilərsiniz (məsələn, bütün regionlar Nvidia Tesla K80 GPU-larına malik deyil).

QEYD Google Cloud Platform müxtəlif GPU kvotalarını tətbiq edir, həm dünya üzrə, həm də region üzrə: siz Google-dan əvvəlcədən icazə almadan minlərlə GPU nodu yarada bilməzsınız. Kvotalarınızı yoxlamaq üçün GCP konsolunda “IAM və administrasiya → Kvotalar”ı açın. Əgər bəzi kvotalar çox azdırsa (məsələn, müəyyən bir regionda daha çox GPU lazımdırsa), onların artırılmasını istəyə bilərsiniz; bu, adətən təxminən 48 saat çəkir.

Vertex AI ilkin olaraq minimum sayıda hesablama nodlarını (bu halda cəmi birini) işə salacaq və saniyədə sorğuların sayı çoxaldıqda, daha çox nod yaradacaq (bu halda maksimum təyin etdiyiniz sayıda, bu halda beşə qədər) və sorğuları onların arasında yüksəkliklərə cək. QPS dərəcəsi bir müddət aşağı düşsə, Vertex AI əlavə hesablama nodlarını avtomatik olaraq dayandıracaq. Buna görə də qiymət birbaşa yükə, həmçinin seçdiyiniz maşın və sürətləndirici növləri və GCS-də saxladığınız məlumatların miqdarı ilə bağlıdır. Bu qiymət modeli bəzən istifadə edən istifadəçilər və mühüm istifadəyə artımlar ilə xidmətlər üçün əladır. O həmçinin startup-lar üçün idealdır: startup həqiqətən işə başlayana qədər qiymət aşağı qalır.

Təbrik edirik, siz buluda ilk modelinizi yerləşdirdiniz! İndi işə bu proqnoz xidmətinə sorğu göndərək:

python

Copy code

```
response = endpoint.predict(instances=X_new.tolist())
```

Əvvəlcə təsvir etmək istədiyimiz şəkilləri, əvvəlki kimi TF Serving REST API vasitəsilə sorğular göndərərkən etdiyimiz kimi, bir Python siyahısına çevirməliyik. Cavab obyekti, float-ların Python siyahıları kimi təsvir edilən proqnozları ehtiva edir. Gəlin onları iki onluq yerə yuvarlaqlaşdırıb NumPy massivinə çevirək:

python

Copy code

```
import numpy as np  
  
np.round(response.predictions, 2)
```

python

Copy code

```
array([[0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 0. , 0. ],  
       [0. , 0. , 0.99, 0.01, 0. , 0. , 0. , 0. , 0. , 0. ],  
       [0. , 0.97, 0.01, 0. , 0. , 0. , 0. , 0.01, 0. , 0. ]])
```

Bəli! Biz əvvəlki ilə eyni proqnozları alırıq. İndi buludda təhlükəsiz şəkildə istənilən yerdən sorğu göndərə biləcəyimiz və sorğuların sayına görə avtomatik olaraq miqyaslama bilən gözəl bir proqnoz xidmətimiz var. Endpoint-i istifadə etdikdən sonra, boş yerə ödəniş etməmək üçün onu silməyi unutmayın:

python

Copy code

```
endpoint.undeploy_all() # endpoint-dən bütün modelləri yerləşdirmə  
endpoint.delete()
```

İndi isə Vertex AI-də işə salmaq və potensial olaraq çox böyük bir məlumat kütləsində proqnozlar vermək üçün bir işin necə idarə olunduğunu nəzərdən keçirək.

Vertex AI üzərində Batch Prediction İşlərinin İcrası

Əgər çox sayıda proqnozlar etmək lazımdırsa, təkrar-təkrar proqnoz xidmətinə zəng etmək əvəzinə, Vertex AI-dən bizim üçün bir proqnoz işini icra etməsini istəyə bilərik. Bunun üçün endpoint tələb olunmur, yalnız model lazımdır. Məsələn, gəlin MNIST modelimizdən istifadə edərək test dəstindəki ilk 100 təsvir üzərində bir proqnoz işi icra edək. Bunun üçün əvvəlcə batch (toplú) hazırlamalı və onu GCS-ə yükləməliyik. Bunu etmənin bir yolu, hər sətirdə bir instansiyani JSON dəyəri olaraq formatlanmış şəkildə ehtiva edən bir fayl yaratmaqdır—bu formata JSON Lines deyilir—və sonra bu faylı Vertex AI-yə ötürməkdir. Beləliklə, gəlin yeni bir kataloqda bir JSON Lines faylı yaradaq və sonra bu kataloqu GCS-ə yükləyək:

python

Copy code

```
batch_path = Path("my_mnist_batch")

batch_path.mkdir(exist_ok=True)

with open(batch_path / "my_mnist_batch.jsonl", "w") as jsonl_file:

    for image in X_test[:100].tolist():

        jsonl_file.write(json.dumps(image))

        jsonl_file.write("\n")

upload_directory(bucket, batch_path)
```

İndi proqnoz işini başlatmağa hazırlıq: işin adını, istifadə ediləcək məşinlərin və sürətləndiricilərin növünü və sayını, yeni yaratdığımız JSON Lines faylı üçün GCS yolunu və Vertex AI-nin modelin proqnozlarını saxlayacağı GCS kataloqun yolunu təyin edəcəyik:

python

Copy code

```
batch_prediction_job = mnist_model.batch_predict(

    job_display_name="my_batch_prediction_job",

    machine_type="n1-standard-4",

    starting_replica_count=1,

    max_replica_count=5,

    accelerator_type="NVIDIA_TESLA_K80",

    accelerator_count=1,


    gcs_source=[f"gs://{bucket_name}/{batch_path.name}/my_mnist_batch.jsonl"],

    gcs_destination_prefix=f"gs://{bucket_name}/my_mnist_predictions/",
```

```
    sync=True # tamamlanmamı gözləmək istəmirsinizsə, False olaraq  
    təyin edin  
)
```

MƏSLƏHƏT

Böyük batch-lər üçün girişləri birdən çox JSON Lines faylına bölə bilərsiniz və onların hamısını `gcs_source` arqumenti vasitəsilə qeyd edə bilərsiniz.

Bu əmrin icrası bir neçə dəqiqliq çəkəcək, əsasən Vertex AI-də hesablama nodlarını (kompüter düyünlərini) işə salmaq üçün. Bu əmr dən sonra proqnozlar adları `prediction.results-00001-of-00002` kimi olan fayllarda əlçatan olacaq. Bu fayllar standart olaraq JSON Lines formatından istifadə edir və hər bir dəyər, bir instansiya və onun müvafiq proqnozunu (yəni, 10 ehtimalı) ehtiva edən bir lüğət şəklindədir. İnstansiyalar girişlərə uyğun olaraq eyni qaydada siyahıya alınır. İş həmçinin `prediction-errors*` fayllarını da çıxarıır, bu fayllar bir şeyin səhv getməsi halında səhvləri ayıklamaq üçün faydalı ola bilər. Gəlin bütün bu çıxış fayllarını istifadə edərək `batch_prediction_job.iter_outputs()` metodundan istifadə edərək bütün proqnozlardan keçək və onları `y_probas` massivində saxlayaq:

python

Copy code

```
y_probas = []  
  
for blob in batch_prediction_job.iter_outputs():  
  
    if "prediction.results" in blob.name:  
  
        for line in blob.download_as_text().splitlines():  
  
            y_proba = json.loads(line)["prediction"]  
  
            y_probas.append(y_proba)
```

İndi isə bu proqnozların nə qədər yaxşı olduğunu görək:

python

Copy code

```
>>> y_pred = np.argmax(y_probas, axis=1)  
  
>>> accuracy = np.sum(y_pred == y_test[:100]) / 100
```

0.98

Gözəl, 98% dəqiqlik!

JSON Lines formatı standartdır, lakin böyük instansiyalar, məsələn, təsvirlər üçün çox yer tutar. Xoşbəxtlikdən, `batch_predict()` metodu `instances_format` arqumentini qəbul edir, bu arqument vasitəsilə istəsəniz başqa bir format seçə bilərsiniz. Bu, standart olaraq "jsonl" olaraq təyin edilib, lakin onu "csv", "tf-record", "tf-record-gzip", "bigquery" və ya "file-list" olaraq dəyişdirə bilərsiniz. Əgər onu "file-list" olaraq təyin etsəniz, o zaman `gcs_source` arqumenti bir giriş faylı yol üçün sətir başına bir fayl yolu ehtiva edən mətn faylinə işarə etməlidir; məsələn, PNG təsvir fayllarına işarə edər. Vertex AI bu faylları binary (ikili) olaraq oxuyacaq, onları Base64 istifadə edərək kodlayacaq və alınan bayt sətirlərini modelə ötürəcək. Bu, modelinizdə Base64 sətirlərini təhlil etmək üçün `tf.io.decode_base64()` istifadə edərək bir preprocessing (ön işləmləmə) təbəqəsi əlavə etməyinizi tələb edir. Əgər fayllar təsvirlərdirsə, nəticəni `tf.io.decode_image()` və ya `tf.io.decode_png()` kimi bir funksiya istifadə edərək təhlil etməlisiniz.

Modeldən istifadəni bitirdiyiniz zaman onu silmək istəsəniz, `mnist_model.delete()` əmri ilə silə bilərsiniz. GCS bucket-də yaratdığınız kataloqları və ya opşional olaraq boş olduqda bucket-i özünü və batch prediction işini də silə bilərsiniz:

python

Copy code

```
for prefix in ["my_mnist_model/", "my_mnist_batch/",
"my_mnist_predictions"]:

    blobs = bucket.list_blobs(prefix=prefix)

    for blob in blobs:

        blob.delete()

bucket.delete() # əgər bucket boşdursa

batch_prediction_job.delete()
```

İndi Vertex AI-ə modelin necə yerləşdiriləcəyini, bir proqnoz xidmətinin necə yaradılacağını və batch prediction işlərinin necə icra ediləcəyini bilərsiniz. Lakin modelinizi mobil tətbiqə yerləşdirmək istəsəniz necə? Yaxud da quraşdırılmış bir cihazda, məsələn, istilik idarəetmə sistemi, fitnes izləyicisi və ya özünü idarə edən avtomobildə?

Modelin Mobil və ya Quraşdırılmış Cihaza Yerləşdirilməsi

Maşın öyrənmə modelləri çoxlu GPU-lu böyük mərkəzləşdirilmiş serverlərdə işləməklə məhdudlaşdırır: onlar məlumat mənbəyinə daha yaxın, məsələn, istifadəçinin mobil cihazında və ya quraşdırılmış cihazda da işləyə bilərlər (sahədə buna edge computing - kənar hesablama adı verilib). Hesablama proseslərini mərkəzləşdirilmiş strukturlardan çıxarmağın və onları cihazlara doğru hərəkət etdirməyin bir çox faydası var: bu, cihazın internetə qoşulmadığı hallarda belə ağıllı olmasına imkan verir, məlumatları uzaq serverə göndərmədən gecikməni azaldır, serverlər üzərindəki yükü azaldır və məxfiliyi artırır, çünkü istifadəçinin məlumatları cihazda qala bilər.

Bununla belə, modelləri bu tip cihazlara yerləşdirməyin mənfi tərəfləri də var. Cihazın hesablama resursları çox GPU-lu güclü serverlə müqayisədə adətən çox kiçikdir. Büyük model cihazda yerləşməyə bilər, çoxlu RAM və CPU istifadə edə bilər və yüklənməsi çox vaxt ala bilər. Nəticədə tətbiq qeyri-reaktiv ola bilər, cihaz qızış və batareyası tez tükənə bilər. Bütün bunların qarşısını almaq üçün yüngül və səmərəli bir model yaratmalısınız, lakin bu zaman onun dəqiqliyini çox da itirməməlisiniz. TFLite kitabxanası modellərinizi bu tip cihazlara yerləşdirməyiniz üçün üç əsas məqsədi olan bir neçə alət təqdim edir:

- Modelin ölçüsünü azaltmaq:** Yükləmə vaxtını qısaltmaq və RAM istifadəsini azaltmaq üçün.
- Hər bir proqnoz üçün tələb olunan hesablamaların sayını azaltmaq:** Gecikməni, batareyanın istifadəsini və istiliyi azaltmaq üçün.
- Modeli cihazın xüsusi məhdudiyyətlərinə uyğunlaşdırmaq.**

Modelin ölçüsünü azaltmaq üçün TFLite model çeviricisi SavedModel-i alaraq FlatBuffers əsasında çox daha yüngül bir formata sıxışdırır. Bu, əvvəlcə Google tərəfindən oyun üçün yaradılmış səmərəli, cross-platform serializasiya kitabxanasıdır (bir az protocol buffers kimidir). Bu, FlatBuffers-i RAM-ə yüklemək üçün heç bir ön işə ehtiyac qalmadan birbaşa yükləməyə imkan verir: bu da yükləmə vaxtını və yaddaş tutumunu azaldır. Model mobil və ya quraşdırılmış cihazda yükləndikdən sonra, TFLite interpreteri onu icra edərək proqnozlar verəcək. Burada SavedModel-i FlatBuffer-ə çevirmək və onu .tflite faylına necə saxlamaq olar:

python

Copy code

```
converter =
tf.lite.TFLiteConverter.from_saved_model(str(model_path))

tflite_model = converter.convert()

with open("my_converted_savedmodel.tflite", "wb") as f:
    f.write(tflite_model)
```

MƏSLƏHƏT

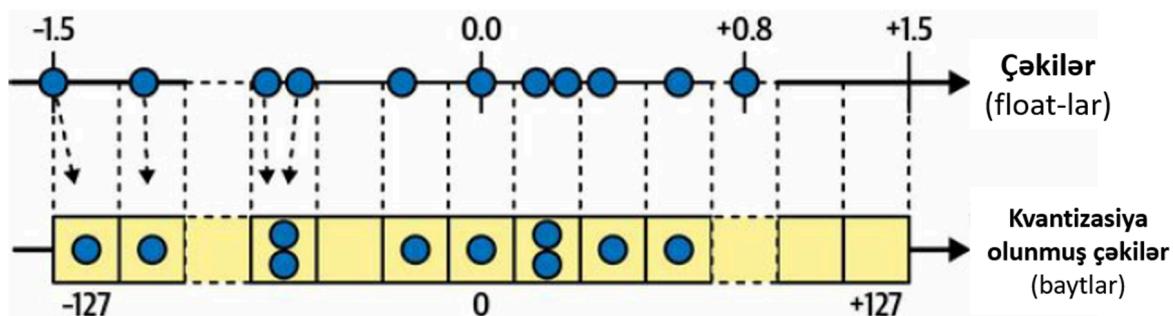
Keras modelini də birbaşa FlatBuffer-ə

`tf.lite.TFLiteConverter.from_keras_model(model)` vasitəsilə saxlaya bilərsiniz.

Çevirici həmçinin modeli optimallaşdırır, həm onu kiçitmək, həm də gecikməni azaltmaq üçün. Proqnozlar vermək üçün lazım olmayan bütün əməliyyatları (məsələn, təlim əməliyyatları kimi) təmizləyir və mümkün olduqda hesablama əməliyyatlarını optimallaşdırır; məsələn, $3 \times a + 4 \times a + 5 \times a$ əməliyyati $12 \times a$ olaraq çevrilir. Həmçinin, mümkün olduqda əməliyyatları birləşdirməyə çalışır. Məsələn, mümkünürsə, batch normalization qatları əvvəlki qatın toplama və vurma əməliyyatlarına birləşdirilir. TFLite-in bir modeli nə qədər optimallaşdırıra biləcəyini yaxşı anlamaq üçün əvvəlcədən hazırlanmış TFLite modellərindən birini, məsələn, Inception_V1_quant modelini yükleyin, arxivə açın, sonra Netron qrafik vizualizasiya alətini açın və .pb faylını yükleyin ki, orijinal modeli görə biləsiniz. Bu böyük, mürəkkəb bir qrafikdir, elə deyilmə? Sonra optimallaşdırılmış .tflite modelini açın və onun gözəlliyinə heyran qalın!

Modelin ölçüsünü azaltmağın başqa bir yolu—sadəcə olaraq daha kiçik sinir şebəkəsi arxitekturalarından istifadə etməklə yanaşı—kiçik bit-widths-dən (bit eni) istifadə etməkdir: məsələn, adı 32-bit float-lar yerinə yarı-float (16-bit) istifadə etsəniz, modelin ölçüsü 2 dəfə kiçiləcək, amma bu, kiçik (adətən az) dəqiqlik itkisi ilə nəticələnə bilər. Bundan əlavə, təlim daha sürətli olacaq və təxminən yarı qədər GPU RAM istifadə edəcəksiniz.

TFLite-in çevircisi bundan da irəli gedə bilər, modelin çəkilərini sabit-nöqtəli 8-bit tam ədədlərə çevirməklə! Bu, 32-bit float-lar ilə müqayisədə dördqat ölçü azalması ilə nəticələnir. Ən sadə yanaşma post-təlim kvantizasiyası adlanır: təlimdən sonra çəkiləri sadəcə olaraq kvantlaşdırır, kifayət qədər sadə, lakin effektiv simmetrik kvantizasiya texnikasından istifadə edir. Maksimum mütləq çəki dəyərini tapır, m , sonra floating-point (süzülən nöqtəli) aralığı $-m-dən +m-ə$ sabit nöqtəli (tam ədədlər) aralığı $-127-dən +127-yə$ qədər xəritələyir. Məsələn, eğer çəkilər $-1.5-dən +0.8-ə$ qədərdirsə, onda $-127, 0 və +127$ baytları uyğun olaraq $-1.5, 0.0 və +1.5$ float-larına uyğun gələcək (Şəkil 19-5-ə baxın). Simmetrik kvantizasiya istifadə edərkən 0.0 həmişə $0-a$ uyğun gəlir. Həmçinin, bu nümunədə $+68-dən +127-yə$ qədər olan bayt dəyərlərinin istifadə olunmayacağını unutmayın, çünki onlar $+0.8-dən$ böyük olan float-lara uyğun gəlir.



Şəkil 19-5. 32-bit float-dan 8-bit tam ədədlərə, simmetrik kvantizasiya istifadə etməklə

Bu post-təlim kvantizasiyasını yerinə yetirmək üçün, sadəcə **DEFAULT** optimizasiyasını çeviricinin optimizasiya siyahısına əlavə edin və **convert()** metodunu çağırmadan əvvəl:

python

Copy code

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

Bu texnika modelin ölçüsünü əhəmiyyətli dərəcədə azaldır, yükləmə müddətini sürətləndirir və yaddaş sahəsini daha az istifadə edir. İcra zamanı kvantlaşdırılmış çəkilər yenidən float-a çevrilir və istifadə olunur. Bu bərpa edilmiş float-lar orijinal float-lar ilə tamamilə eyni olmasa da, çox yaxın olur, buna görə də dəqiqlik itkisi adətən qəbul edilə bilər. Modelin sürətini ciddi şəkildə azaldacaq float dəyərlərini daim yenidən hesablamadan qəçməq üçün, TFLite onları keşləyir: təəssüf ki, bu texnika RAM istifadəsini azaltmır və modeli də sürətləndirmir. Əsasən tətbiqin ölçüsünü azaltmaq üçün istifadə olunur.

Gecikməni və enerji istehlakını azaltmaq üçün ən effektiv yol aktivasiyaları da kvantlaşdırmaqdır, beləliklə bütün hesablamalar tam ədədlərlə aparıla bilər, heç bir floating-point əməliyyatına ehtiyac olmadan. Eyni bit-enini istifadə edərkən belə (məsələn, 32-bit float-lar əvəzinə 32-bit tam ədədlər), tam ədəd hesablamaları daha az CPU dövrü istifadə edir, daha az enerji sərf edir və daha az istilik yaradır. Və əgər siz bit-enini də azaltsanız (məsələn, 8-bit tam ədədlərə qədər), böyük sürət artımı əldə edə bilərsiniz. Üstəlik, bəzi sinir şəbəkəsi sürətləndirici cihazlar—məsələn, Google-un Edge TPU-su—yalnız tam ədədləri işləyə bilər, buna görə də həm çəkilərin, həm də aktivasiyaların tam kvantizasiyası məcburidir. Bu, təlimdən sonra edilə bilər; aktivasiyaların maksimum mütləq dəyərini tapmaq üçün bir kalibrasiya addımı tələb olunur, buna görə də TFLite-a təlim məlumatlarının təmsil edici bir nümunəsini təqdim etməlisiniz (bunun çox böyük olmasına ehtiyac yoxdur) və o, məlumatları modeldən keçirərək kvantizasiya üçün lazım olan aktivasiya statistikasını ölçəcək. Bu addım adətən sürətlidir.

Kvantizasiyanın əsas problemi, bir qədər dəqiqlik itkisidir: bu, çəkilərə və aktivasiyalara səs-küy əlavə etmək kimidir. Əgər dəqiqlik itkisi çox ciddi olarsa, onda kvantizasiyadan xəbərdar təlimdən istifadə etməyiniz lazım ola bilər. Bu, modelə təlim zamanı kvantizasiya səs-küyünü nəzərə almayı öyrənmək üçün saxta kvantizasiya əməliyyatları əlavə etmək deməkdir; son çəkilər daha çox kvantizasiyaya qarşı davamlı olacaq. Üstəlik, kalibrasiya addımı təlim zamanı avtomatik olaraq həyata keçirilə bilər, bu da bütün prosesi sadələşdirir.

Mən TFLite-in əsas anlayışlarını izah etdim, lakin mobil və ya quraşdırılmış tətbiqi kodlaşdırmaq üçün tam bir kitab lazımdır. Xoşbəxtlikdən, belə kitablar mövcuddur: əgər TensorFlow tətbiqlərini mobil və quraşdırılmış cihazlar üçün necə qurmaq barədə daha çox öyrənmək istəyirsinizsə, Pete Warden (TFLite komandasının keçmiş rəhbəri) və Daniel Situnayake tərəfindən yazılmış **TinyML: Maşın Öyrənməsi ilə TensorFlow Arduino və Ultra-Aşağı Güclü Mikro-Kontrolerlər** kitabını və ya Laurence Moroney tərəfindən yazılmış **On-Device Development** üçün AI və **Maşın Öyrənməsi** kitabını nəzərdən keçirin.

İndi isə əgər modelinizi bir vəbsaytda, istifadəçinin brauzerində birbaşa işlətmək istəsəniz nə olacaq?

Veb Səhifəsində Modelin İşlədilməsi

Maşın öyrənməsi modelinizi server tərəfində deyil, istifadəçinin brauzerində işlətmək bir çox ssenarilərdə faydalı ola bilər, məsələn:

- Veb tətbiqiniz tez-tez istifadəçinin əlaqəsinin aralıq və ya yavaş olduğu vəziyyətlərdə istifadə edilirsə (məsələn, gəzintiyə çıxanlar üçün bir sayt), beləliklə modeli birbaşa müştəri tərəfində işlətmək vəbsaytin etibarlı olmasını təmin etməyin yeganə yoludur.
- Modelin cavablarının mümkün qədər sürətli olmasını tələb edən bir vəziyyətdədirsinizsə (məsələn, bir onlayn oyun üçün). Serverə sorğu göndərmədən proqnozlar etmək gecikməni azaldacaq və vəbsaytin daha reaksiya qabiliyyətini artıracaq.
- Veb xidmətiniz bəzi şəxsi istifadəçi məlumatlarına əsaslanan proqnozlar verirsə və bu şəxsi məlumatların istifadəçinin cihazını tərk etmədən məxfi qalmasını istəyirsinizsə.

Bütün bu ssenarilər üçün TensorFlow.js (TFJS) JavaScript kitabxanasından istifadə edə bilərsiniz. Bu kitabxana bir TFLite modelini yükleyib birbaşa istifadəçinin brauzerində proqnozlar verə bilər. Məsələn, aşağıdakı JavaScript modulu TFJS kitabxanasını idxal edir, əvvəldən təlim edilmiş MobileNet modelini yükleyir və bu modeldən bir şəkili təsnif etmək üçün istifadə edir və proqnozları jurnal olaraq saxlayır. Kodu <https://homl.info/tfjscode> ünvanında Glitch.com-da sinaqdan keçirə bilərsiniz; bu sayt veb tətbiqlərini brauzerinizdə pulsuz qurmağa imkan verir; səhifənin sağ alt küçündəki **PREVIEW** düyməsinə klikləyin ki, kodun necə işlədiyini görün:

javascript

Copy code

```
import "https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest";

import
"https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0";

const image = document.getElementById("image");

mobilenet.load().then(model => {

  model.classify(image).then(predictions => {

    for (var i = 0; i < predictions.length; i++) {

      let className = predictions[i].className;

      let proba = (predictions[i].probability * 100).toFixed(1);

      console.log(className + " : " + proba + "%");

    }

  });

});
```

Hətta bu vefsayı progresiv veb tətbiqinə (PWA) çevirmək də mümkündür: bu, bir sıra meyarlara cavab verən və hər hansı bir brauzerdə görünə bilən bir vefsayıdır və hətta mobil cihazda müstəqil tətbiq kimi quraşdırıla bilər. Məsələn, <https://homl.info/tfjswp> ünvanını bir mobil cihazda açmağa cəhd edin: müasir brauzerlərin əksəriyyəti sizdən TFJS Demo-nu ana ekrana əlavə etmək istədiyinizi soruşacaq. Qəbul edirsınızsa, tətbiqlər siyahınızda yeni bir simvol görəcəksiniz. Bu simvola klikləmək, TFJS Demo vefsayıtnı öz pəncərəsində, adı mobil tətbiq kimi yükləyəcək. Bir PWA hətta oflayn işləmək üçün konfiqurasiya edilə bilər, bu, bir JavaScript moduludur və brauzerdə öz ayrı sapında işləyir və şəbəkə sorğularını intercept edərək PWA-nın daha sürətli işləməsini və ya hətta tamamilə oflayn işləməsini təmin edir. O, həmçinin push mesajlarını çatdırmaq, arxa planda vəzifələri icra etmək və daha çoxunu yerinə yetirmək üçün istifadə edilə bilər. PWA-lar veb və mobil cihazlar üçün tək bir kod bazasını idarə etməyi asanlaşdırır. Onlar həmçinin bütün istifadəçilərin tətbiqinizin eyni versiyasını işlətdiklərinə əmin olmayı asanlaşdırır. Bu TFJS Demo-nun PWA kodunu Glitch.com-da <https://homl.info/wpacode> ünvanında sinaqdan keçirə bilərsiniz.

TIP: Brauzerinizdə işləyən maşın öyrənmə modellərinin daha çox demosuna baxmaq üçün <https://tensorflow.org/js/demos> ünvanını yoxlayın.

TFJS həmçinin modelin birbaşa veb brauzerinizdə öyrədilməsini dəstəkləyir! Və əslində olduqca sürətlidir. Kompüterinizdə GPU kartı varsa, TFJS ümumiyyətlə onu istifadə edə bilər, hətta bu Nvidia kartı olmasa belə. Həqiqətən də, TFJS WebGL-dən istifadə edəcək, nə vaxt ki, bu mümkünür, və müasir veb brauzerlər ümumiyyətlə geniş GPU kart diapazonunu dəstəklədiyi üçün, TFJS əslində adı TensorFlow-dan daha çox GPU kartını dəstəkləyir (hansı ki, yalnız Nvidia kartlarını dəstəkləyir).

Bir istifadəçinin veb brauzerində model öyrətmək, bu istifadəçinin məlumatlarının məxfi qalmasını təmin etmək üçün xüsusilə faydalı ola bilər. Model mərkəzləşdirilmiş şəkildə öyrədilə bilər və sonra həmin istifadəçinin məlumatlarına əsaslanaraq, brauzerdə yerli olaraq incə tənzimlənə bilər. Əgər bu mövzuya marağınız varsa, federated learning haqqında məlumat əldə edin.

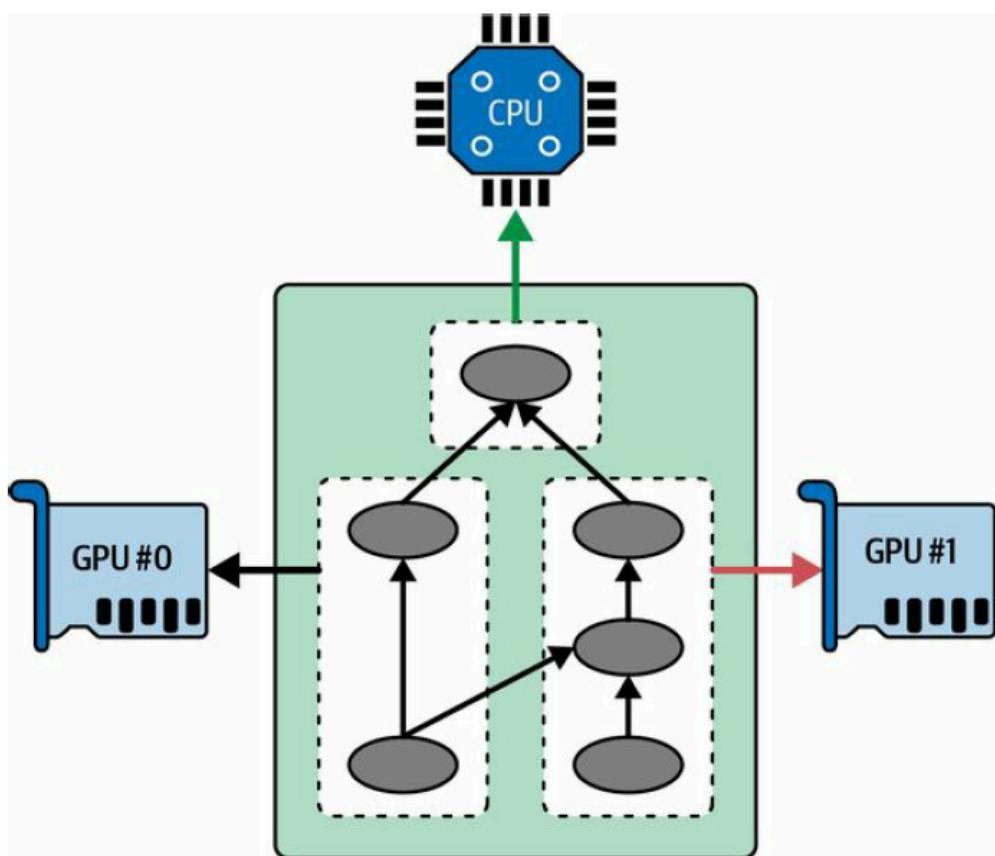
Yenidən qeyd etməliyəm ki, bu mövzuya layiqincə diqqət ayırmaq üçün bir kitab lazım olardı. Əgər TensorFlow.js haqqında daha çox öyrənmək istəyirsinizsə, O'Reilly-nin Practical Deep Learning for Cloud, Mobile, and Edge kitabı, Anirudh Koul və s. tərəfindən yazılmış və ya Learning TensorFlow.js kitabını, Gant Laborde tərəfindən yazılmış kitabları nəzərdən keçirin.

İndi ki, TensorFlow modellərini TF Serving-a, ya da Vertex AI ilə buluda, ya da mobil və quraşdırılmış cihazlara TFLite ilə və ya veb brauzerinə TFJS ilə yerləşdirməyi öyrəndiniz, gəlin GPU-lardan istifadə edərək hesablamaları necə sürətləndirməyi müzakirə edək.

Hesablamaları Sürətləndirmək üçün GPU-ların İstifadəsi

11-ci fəsildə təlimi əhəmiyyətli dərəcədə sürətləndirə bilən bir neçə texnikanı nəzərdən keçirdik: daha yaxşı çəki başlanğııcı, inkişaf etmiş optimizatorlar və s. Lakin bütün bu texnikalarla belə, tek CPU ilə bir anda böyük bir sinir şəbəkəsini öyrətmək tapşırıq bağlı olaraq saatlar, günlər və hətta həftələr çəkə bilər. GPU-lar sayəsində bu təlim müddətini dəqiqlərə və ya saatlara qədər azaltmaq olar. Bu, təkcə böyük vaxt qənaət etmək demək deyil, həm də müxtəlif modellərlə daha asan və tez-tez təcrübə aparmağa və modellərinizi təzə məlumatlarla tez-tez yenidən öyrətməyə imkan verir.

Əvvəlki fəsillərdə Google Colab-da GPU ilə işləyən iş mühitlərindən istifadə etdik. Bunun üçün yalnız "Change runtime type" seçimini Runtime menyusundan seçmək və GPU sürətləndirici növünü seçmək kifayətdir; TensorFlow avtomatik olaraq GPU-nu aşkar edir və hesablamaları sürətləndirmək üçün istifadə edir, və kod heç bir GPU olmadan olduğu kimi qalır. Sonra, bu fəsildə modellərinizi bir neçə GPU ilə işləyən hesablamalı qoşaqlara Vertex AI ilə yerləşdirməyi gördünüz: bu, Vertex AI modelini yaratarkən uyğun GPU ilə işləyən Docker imicini seçmək və endpoint.deploy() çağırışını edərkən arzu olunan GPU növünü seçməkdən ibarətdir. Bəs öz GPU-nuzu almaq istəyirsinizsə? Və əgər hesablamaları bir maşında CPU və bir neçə GPU cihazı arasında paylamaq istəyirsinizsə (bax, Şəkil 19-6)? Bu, indi müzakirə edəcəyimiz mövzudur, sonra isə bu fəsildə hesablamaları bir neçə server arasında necə paylamağı müzakirə edəcəyik.



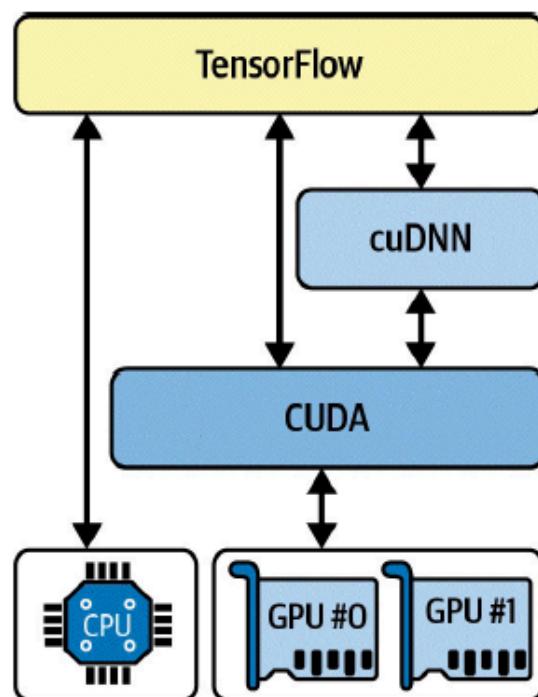
Şəkil 19-6. TensorFlow qrafının bir neçə cihazda paralel olaraq icra edilməsi

Öz GPU-nuzu əldə etmək

Əgər GPU-dan uzun müddət ərzində və intensiv şəkildə istifadə edəcəyinizi bilirsinizsə, öz GPU-nuzu almaq maliyyə baxımından məntiqli ola bilər. Eyni zamanda, modellərinizi yerli olaraq öyrətmək istəyə bilərsiniz, çünki məlumatlarınızı buluda yüklemək istəmirsiniz. Yaxud, sadəcə oyun oynamamaq üçün bir GPU kartı almaq istəyirsiniz və onu dərin öyrənmə üçün də istifadə etmək istəyirsiniz.

Əgər GPU kartı almaq qərarına gəlsəniz, doğru seçimi etmək üçün vaxt ayırın. Tapşırıqlarınız üçün nə qədər RAM-a ehtiyacınız olacağınızı nəzərə almalısınız (məsələn, tipik olaraq şəkil emalı və ya NLP üçün ən azı 10 GB), bant genişliyi (yəni məlumatların GPU-ya daxil edilməsi və çıxarılması nə qədər sürətli ola bilər), nüvələrin sayı, soyutma sistemi və s. Tim Dettmers bu mövzuda sizə kömək edəcək mükəmməl bir blog yazısı yazıb: onu diqqətlə oxumağı tövsiyə edirəm. Bu mətn yazılar kən, TensorFlow yalnız CUDA Hesablaşdırma İmkanı 3.5+ olan Nvidia kartlarını (əlbəttə ki, Google-un TPU-larını da) dəstəkləyir, lakin gələcəkdə dəstəklədiyi cihazların sayını artırıa bilər, buna görə də TensorFlow-un sənədlərini yoxlayaraq bu gün hansı cihazların dəstəkləndiyini öyrənməyiniz yaxşı olardı.

Əgər Nvidia GPU kartını seçsəniz, uyğun Nvidia sürücülərini və bir neçə Nvidia kitabxanasını quraşdırma olacaqsınız. Bunlara Compute Unified Device Architecture (CUDA) Kitabxanası daxildir ki, bu da tərtibatçılar CUDA-dan istifadə edən GPU-ları hər cür hesablamalar üçün istifadə etməyə imkan verir (yalnız qrafik sürətləndirilməsi üçün deyil), və CUDA Dərin Sinir Şəbəkəsi Kitabxanası (cuDNN), aktivləşdirmə qatları, normallaşdırma, irəli və geri çevrilmiş konvolusiyalar və pulinq (həmçinin 14-cü fəsilə baxın) kimi ümumi DNN hesablamaları üçün GPU ilə sürətləndirilən kitabxanadır. cuDNN, Nvidia-nın Dərin Öyrənmə SDK-sının bir hissəsidir. Onu yükləmək üçün Nvidia tərtibatçı hesabı yaratmalı olacağınızı qeyd edin. TensorFlow, GPU kartlarını idarə etmək və hesablamaları sürətləndirmək üçün CUDA və cuDNN-dən istifadə edir (bax, Şəkil 19-7).



Şəkil 19-7. TensorFlow CUDA və cuDNN-dən istifadə edərək GPU-ları idarə etdir və DNN-ləri gücləndirir

GPU kartlarını və bütün tələb olunan sürücüləri və kitabxanaları quraşdırıldıqdan sonra hər şeyin düzgün quraşdırıldığı yoxlamaq üçün [nvidia-smi](#) əmri ilə yoxlama apara bilərsiniz. Bu əmr mövcud GPU kartlarını, həmçinin hər bir kartda işləyən prosesləri siyahıya alır. Bu nümunədə,

təxminən 15 GB RAM ilə Nvidia Tesla T4 GPU kartı var və hazırda heç bir proses işləmədiyi göstərilir:

bash

Copy code

```
$ nvidia-smi

Sun Apr 10 04:52:10 2022

+-----+
| NVIDIA-SMI 460.32.03 Driver Version: 460.32.03 CUDA Version: 11.2
| |
|-----+-----+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
| ======+=====+=====
| ======+=====+=====+
| 0 Tesla T4 Off | 00000000:00:04.0 Off | 0 |
| N/A 34C P8 9W / 70W | 3MiB / 15109MiB | 0% Default |
| | | N/A |
+-----+-----+
+-----+-----+
| Processes: |
| GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
```

```
| ====== |  
==== |  
  
| No running processes found |  
  
+-----+  
-----+
```

TensorFlow-un həqiqətən GPU-nuzu gördüyünü yoxlamaq üçün aşağıdakı əmrləri çalışdırın və nəticənin boş olmadığından əmin olun:

python

Copy code

```
>>> physical_gpus = tf.config.list_physical_devices("GPU")  
  
>>> physical_gpus  
  
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

GPU RAM-ını idarə etmək

Defolt olaraq, TensorFlow, ilk dəfə hesablama apararkən bütün mövcud GPU-larda RAM-ın demək olar ki, hamısını avtomatik olaraq tutur. Bu, GPU RAM-ının fragmentləşməsini məhdudlaşdırmaq üçün edilir. Bu o deməkdir ki, ikinci TensorFlow programını (və ya GPU tələb edən hər hansı bir program) başladırsanız, tez bir zamanda RAM tüketəcək. Bu, düşündüyüiniz qədər tez-tez baş vermir, çünki bir maşında adətən bir TensorFlow programı işləyir: adətən bir təlim skripti, TF Serving node-u və ya Jupyter dəftəri. Hər hansı bir səbəbdən bir neçə programı paralel işlətməlisinizsə (məsələn, eyni maşında iki fərqli modeli eyni vaxtda öyrətmək üçün), o zaman GPU RAM-ını bu proseslər arasında daha bərabər bölüşdurməlisiniz.

Maşınızdə bir neçə GPU kartınız varsa, sadə həll olaraq hər birini yalnız bir prosesə təyin edə bilərsiniz. Bunu etmək üçün **CUDA_VISIBLE_DEVICES** mühit dəyişənini təyin edə bilərsiniz ki, hər bir proses yalnız müvafiq GPU kart(lar)ını görsün. Həmçinin, **CUDA_DEVICE_ORDER** mühit dəyişənini **PCI_BUS_ID** olaraq təyin edin ki, hər bir ID həmişə eyni GPU kartını göstərsin. Məsələn, əgər dörd GPU kartınız varsa, iki programı başladıb onlardan hər birinə iki GPU təyin edə bilərsiniz:

bash

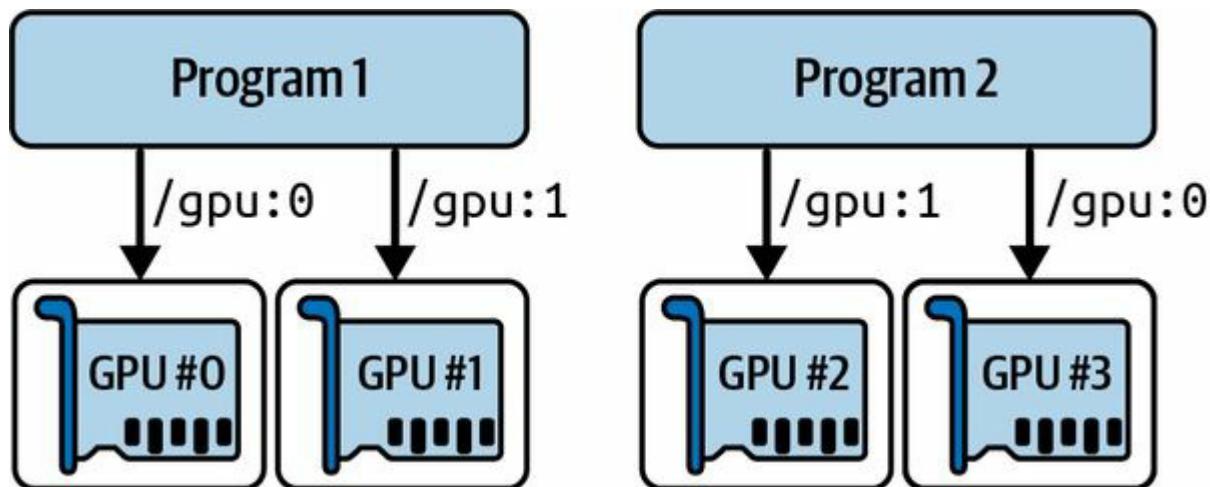
Copy code

```
$ CUDA_DEVICE_ORDER=PCI_BUS_ID CUDA_VISIBLE_DEVICES=0,1 python3  
program_1.py
```

```
# və başqa terminalda:
```

```
$ CUDA_DEVICE_ORDER=PCI_BUS_ID CUDA_VISIBLE_DEVICES=3,2 python3  
program_2.py
```

Beləliklə, Program 1 yalnız 0 və 1 sayılı GPU kartlarını görəcək, TensorFlow-da müvafiq olaraq "/gpu:0" və "/gpu:1" adlandırılacaq və Program 2 yalnız 2 və 3 sayılı GPU kartlarını görəcək, TensorFlow-da müvafiq olaraq "/gpu:1" və "/gpu:0" adlandırılacaq (sıralamaya diqqət yetirin). Hər şey düzgün işləyəcək (şəkil 19-8-ə baxın). Əlbəttə ki, bu mühit dəyişənlərini Python-da da təyin edə bilərsiniz, `os.environ["CUDA_DEVICE_ORDER"]` və `os.environ["CUDA_VISIBLE_DEVICES"]` dəyişənlərini TensorFlow-dan istifadə etmədən əvvəl təyin etmək şərtilə.



Şəkil 19-8. Hər program iki GPU əldə edir

Başqa bir seçim TensorFlow-a yalnız müəyyən bir miqdarda GPU RAM-ı götürməyi söyləməkdir. Bu, TensorFlow-u idxlə etdikdən dərhal sonra edilməlidir. Məsələn, TensorFlow-a hər bir GPU-da yalnız 2 GiB RAM götürməsini təmin etmək üçün hər bir fiziki GPU cihazı üçün məntiqi GPU cihazı (bəzən virtual GPU cihazı adlanır) yaratmalı və onun yaddaş limitini 2 GiB (yəni 2,048 MiB) olaraq təyin etməlisiniz:

```
python
```

```
Copy code
```

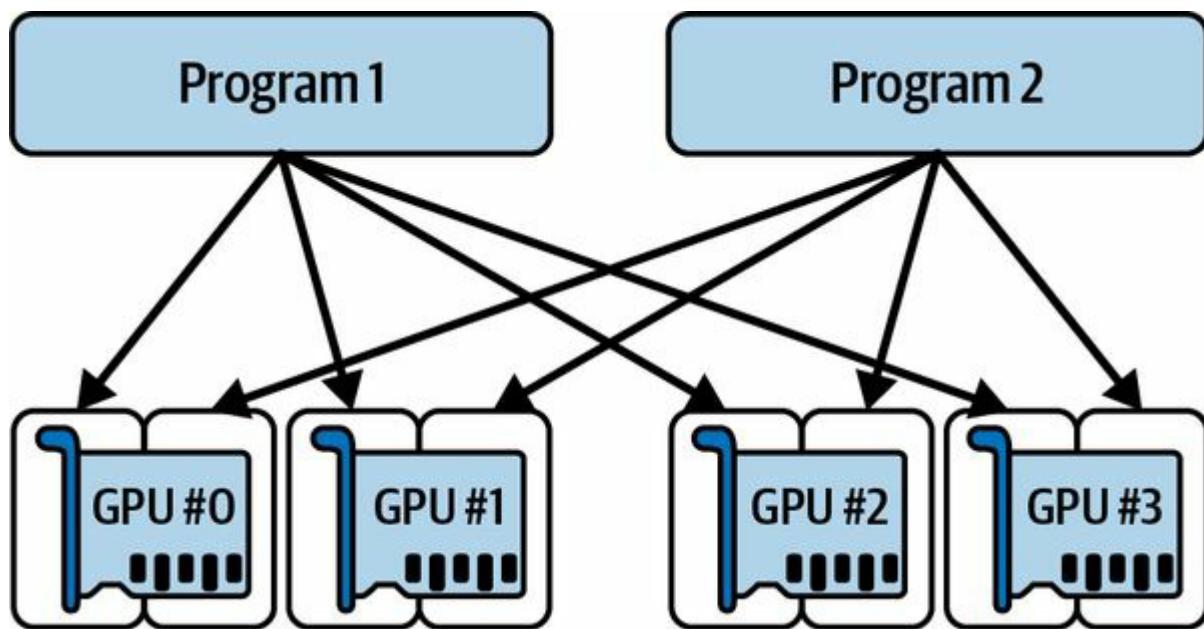
```
for gpu in physical_gpus:
```

```

tf.config.set_logical_device_configuration(
    gpu,
    [tf.config.LogicalDeviceConfiguration(memory_limit=2048)]
)

```

Tutaq ki, hər biri ən azı 4 GiB RAM-a malik olan dörd GPU-nuz var: bu halda, bu tip iki program paralel işləyə bilər və hər biri bütün dörd GPU kartından istifadə edə bilər (şəkil 19-9-a baxın). Hər iki program işləyərkən [nvidia-smi](#) əmrinin işlətsəniz, hər bir prosesin hər kartda 2 GiB RAM tutduğunu görməlisiniz.



Şəkil 19-9. Hər program bütün dörd GPU-dan istifadə edir, amma hər GPU-da yalnız 2 GiB RAM ilə

Başqa bir seçim isə TensorFlow-a yalnız ehtiyacı olduqda yaddaşı götürmək üçün göstəriş verməkdir. Bu, TensorFlow-u idxlə etdikdən dərhal sonra edilməlidir:

python

Copy code

```

for gpu in physical_gpus:

    tf.config.experimental.set_memory_growth(gpu, True)

```

Başqa bir üsul isə `TF_FORCE_GPU_ALLOW_GROWTH` mühit dəyişənini `true` olaraq təyin etməkdir. Bu seçimlə, TensorFlow yaddaşı götürdükdən sonra heç vaxt sərbəst buraxmayacaq (yaddaş frgmentasiyasını qarşısını almaq üçün), program bitənə qədər. Bu seçimlə deterministik davranışını etmək daha çətin ola bilər (məsələn, bir program digər programın yaddaş istifadəsi çoxaldığı üçün çökəməyə bilər), buna görə istehsalda əvvəlki seçimlərdən birini istifadə etmək daha məqsədə uyğundur. Lakin, bu seçim çox faydalı ola bilər: məsələn, bir maşında bir neçə Jupyter notebook işlədirsinizsə, onlardan bir neçəsi TensorFlow istifadə edirə.

`TF_FORCE_GPU_ALLOW_GROWTH` mühit dəyişəni Colab mühitlərində `true` olaraq təyin edilmişdir.

Son olaraq, bəzən bir GPU-nu iki və ya daha çox məntiqi cihazlara bölmək istəyirsiniz. Məsələn, bu, yalnız bir fiziki GPU-ya sahib olduğunuz halda—məsələn, Colab mühitində—amma çox GPU-lu bir alqoritmi sinaqdan keçirmək istəyirsinizsə faydalıdır. Aşağıdakı kod GPU #0-i iki məntiqi cihaz halına gətirir, hər biri 2 GiB RAM ilə (yenə də, bu TensorFlow-u idxlə etdikdən dərhal sonra edilməlidir):

python

Copy code

```
tf.config.set_logical_device_configuration(  
    physical_gpus[0],  
    [tf.config.LogicalDeviceConfiguration(memory_limit=2048),  
     tf.config.LogicalDeviceConfiguration(memory_limit=2048)])
```

Bu iki məntiqi cihaz "/gpu:0" və "/gpu:1" adlanır və onları iki normal GPU kimi istifadə edə bilərsiniz. Bütün məntiqi cihazları aşağıdakı kimi siyahıya ala bilərsiniz:

python

Copy code

```
>>> logical_gpus = tf.config.list_logical_devices("GPU")  
>>> logical_gpus  
[LogicalDevice(name='/device:GPU:0', device_type='GPU'),  
 LogicalDevice(name='/device:GPU:1', device_type='GPU')]
```

İndi TensorFlow-un dəyişənləri və əməliyyatları hansı cihazlarda yerləşdiriyini necə müəyyən etdiyinə baxaq.

Əməliyyatları və Dəyişənləri Cihazlarda Yerləşdirmək

Keras və tf.data adətən əməliyyatları və dəyişənləri uyğun yerlərə yerləşdirməkdə yaxşı iş görür, amma daha çox nəzarət etmək istəyirsinizsə, əməliyyatları və dəyişənləri əl ilə hər bir cihazda yerləşdirə bilərsiniz:

Ən ümumi halda, məlumatların işlənməsi əməliyyatlarını CPU-da, neyron şəbəkəsi əməliyyatlarını isə GPU-larda yerləşdirmək istəyirsiniz.

GPU-ların ümumiyyətlə məhdud əlaqə bant genişliyinə malik olduğunu nəzərə alsaq, GPU-lara lazımsız məlumat köçürmələrindən qaçınmaq vacibdir. Bir maşına əlavə CPU RAM əlavə etmək sadə və nisbətən ucuzdur, buna görə də adətən kifayət qədər RAM olur, lakin GPU RAM GPU-ya integrasiya olunub: bu bahalı və beləliklə məhdud bir qaynaqdır, buna görə də bir dəyişən növbəti təlim addımlarında lazım deyilsə, onu CPU-da yerləşdirmək daha məqsədə uyğundur (məsələn, məlumat dəstləri adətən CPU-da yerləşir).

Varsayılan olaraq, bütün dəyişənlər və əməliyyatlar birinci GPU-da ("/gpu:0" adlandırılan) yerləşdiriləcəkdir, GPU nüvəsi olmayan dəyişənlər və əməliyyatlar isə CPU-da (həmişə "/cpu:0" adlandırılan) yerləşdiriləcəkdir. Bir tensorun və ya dəyişənin cihaz atributu, onun hansı cihazda yerləşdirildiyini göstərir:

python

Copy code

```
>>> a = tf.Variable([1., 2., 3.]) # float32 dəyişəni GPU-da  
yerləşir  
  
>>> a.device  
  
'/job:localhost/replica:0/task:0/device:GPU:0'  
  
>>> b = tf.Variable([1, 2, 3]) # int32 dəyişəni CPU-da yerləşir  
  
>>> b.device  
  
'/job:localhost/replica:0/task:0/device:CPU:0'
```

Bu ön əlavə `/job:localhost/replica:0/task:0`-i indiki mərhələdə təhlükəsiz şəkildə nəzərə almaya bilərsiniz; bu bölmədə işlər, təkrarlayıcılar və tapşırıqlar haqqında daha sonra danışacaq. Göründüyü kimi, ilk dəyişən GPU #0-da yerləşdirilmişdir ki, bu da default cihazdır. Lakin, ikinci dəyişən CPU-da yerləşdirilmişdir: bu, integer dəyişənlər üçün və ya integer tensörlər ilə əməliyyatlar üçün GPU nüvələrinin olmaması səbəbindən TensorFlow CPU-ya geri dönmüşdür.

Əgər bir əməliyyatı default cihazdan fərqli bir cihazda yerləşdirmək istəyirsizsə, `tf.device()` kontekstini istifadə edin:

python

Copy code

```
>>> with tf.device("/cpu:0"):  
...     c = tf.Variable([1., 2., 3.])  
  
...  
>>> c.device  
'/job:localhost/replica:0/task:0/device:CPU:0'
```

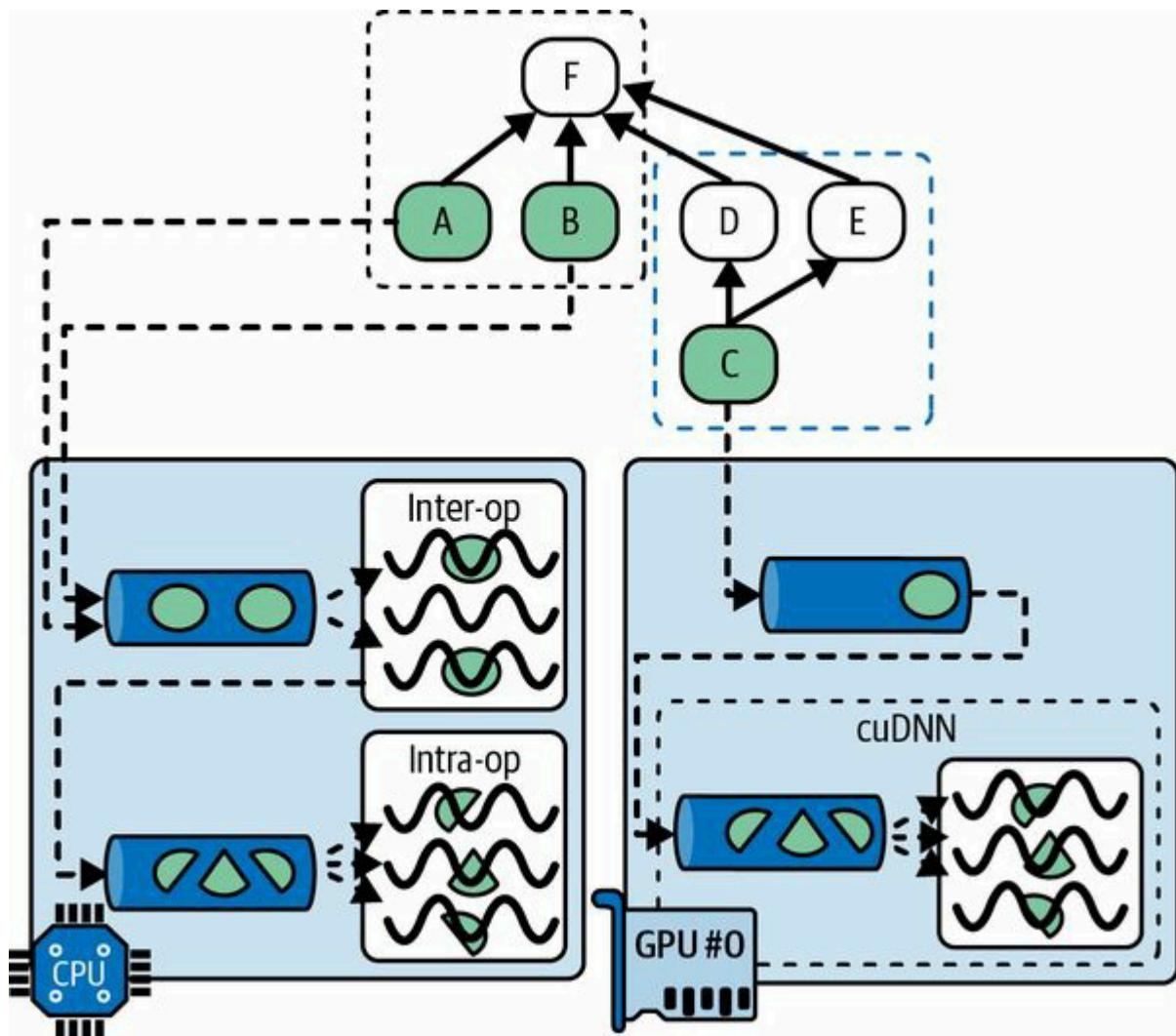
Qeyd: CPU həmişə tək bir cihaz olaraq ("/cpu:0") müalicə olunur, hətta maşınınız bir neçə CPU nüvəsinə malik olsa belə. CPU-da yerləşdirilən hər bir əməliyyat çox nüvəli kernelə malikdir, bir neçə nüvədə paralel olaraq icra oluna bilər.

Əgər qəsdən mövcud olmayan bir cihazda və ya heç bir kernelin olmadığı bir cihazda əməliyyat və ya dəyişən yerləşdirməyə çalışsanız, TensorFlow susqun şəkildə default olaraq seçdiyi cihazda fəaliyyət göstərəcəkdir. Bu, müxtəlif GPU sayına sahib olan fərqli maşınlarda eyni kodu işlədə bilmək üçün faydalıdır. Lakin, əgər istəsəniz, `tf.config.set_soft_device_placement(False)` istifadə edərək istisna almağı seçə bilərsiniz.

İndi TensorFlow-un əməliyyatları necə paralel şəkildə icra etdiyinə baxaq.

Bir neçə Cihaz Üzərində Paralel İcra

Əvvəlki fəsildə gördüyüümüz kimi, TF funksiyalarını istifadə etmənin faydalarından biri paralelliktir. Bunun üzərinə daha yaxından baxaq. TensorFlow bir TF funksiyasını icra edərkən, əvvəlcə onun qrafını analiz edir ki, hansı əməliyyatların qiymətləndirilməsi lazımlı olduğunu tapıb, hər birinin neçə asılılığı olduğunu sayıb. TensorFlow sonra asılılığı sıfır olan hər əməliyyatı (yəni, hər mənbə əməliyyatı) bu əməliyyatın cihazının qiymətləndirmə növbəsinə əlavə edir (Şəkil 19-10-a baxın). Bir əməliyyat qiymətləndirildikdən sonra, ona asılı olan hər bir əməliyyatın asılılıq saygıçı azaldılır. Bir əməliyyatın asılılıq saygıçı sıfır endikdə, bu əməliyyatın cihazının qiymətləndirmə növbəsinə əlavə edilir. Və bütün çıxışlar hesablamalı bitdikdən sonra, onlar qaytarılır.



Şəkil 19-10. TensorFlow qrafının paralelləşdirilmiş icrası

CPU-nun qiymətləndirmə növbəsindəki əməliyyatlar bir iplik hovuzuna (inter-op thread pool) göndərilir. Əgər CPU bir neçə nüvəyə malikdirsə, bu əməliyyatlar paralel olaraq qiymətləndiriləcəkdir. Bəzi əməliyyatların çoxlu iplikli CPU nüvələri var: bu nüvələr öz tapşırıqlarını bir neçə alt əməliyyata bölür, bu alt əməliyyatlar başqa bir qiymətləndirmə növbəsinə yerləşdirilir və digər bir iplik hovuzuna (intra-op thread pool) göndərilir (bütün çoxlu iplikli CPU nüvələri tərəfindən paylaşıılır). Qısaca, bir neçə əməliyyat və alt əməliyyat paralel olaraq müxtəlif CPU nüvələrində qiymətləndirilə bilər.

GPU üçün vəziyyət biraz daha sadədir. GPU-nun qiymətləndirmə növbəsindəki əməliyyatlar sıralı şəkildə qiymətləndirilir. Lakin, əksər əməliyyatların çoxlu iplikli GPU nüvələri var, adətən TensorFlow-un asılı olduğu CUDA və cuDNN kimi kitabxanalar tərəfindən həyata keçirilir. Bu tətbiqlər öz iplik hovuzlarına malikdirlər və adətən mümkün qədər çox GPU ipiliyindən istifadə edirlər (bu səbəbdən GPU-larda inter-op thread pool-a ehtiyac yoxdur: hər əməliyyat artıq əksər GPU ipliklərini doldurur).

Məsələn, Şəkil 19-10-da, A, B və C əməliyyatları mənba əməliyyatlarıdır, beləliklə onlar dərhal qiymətləndirilə bilər. A və B əməliyyatları CPU-da yerləşdirilir, buna görə də CPU-nun

qiymətləndirmə növbəsinə göndərilir, sonra inter-op thread pool-a yönəldilir və dərhal paralel olaraq qiymətləndirilir. A əməliyyatı çoxlu iplikli nüvəyə malikdir; onun hesablamaları üç hissəyə bölünür, bu hissələr paralel olaraq intra-op thread pool tərəfindən icra edilir. C əməliyyatı GPU #0-nın qiymətləndirmə növbəsinə gedir və bu nümunədə onun GPU nüvəsi cuDNN-dən istifadə edir, hansı ki, öz intra-op thread pool-u idarə edir və əməliyyatı paralel olaraq çoxlu GPU iplikləri üzərində icra edir. Tutaq ki, C ilk olaraq tamamlanır. D və E əməliyyatlarının asılılıq saygacı azalır və 0-ə çatır, beləliklə hər iki əməliyyat GPU #0-nın qiymətləndirmə növbəsinə göndərilir və sıralı şəkildə icra edilir. C yalnız bir dəfə qiymətləndirilir, baxmayaraq ki, həm D, həm də E ona asılıdır. Tutaq ki, B növbəti olaraq tamamlanır. Bu zaman F-nin asılılıq saygacı 4-dən 3-ə azalır və 0 olmadığı üçün hələ icra edilmir. A, D və E tamamlandıqdan sonra, F-nin asılılıq saygacı 0-a çatır və CPU-nun qiymətləndirmə növbəsinə göndərilir və qiymətləndirilir. Nəhayət, TensorFlow tələb olunan çıxışları qaytarır.

TensorFlow-un yerinə yetirdiyi əlavə bir sehr isə TF funksiyası bir vəziyyətli resursu, məsələn, dəyişəni dəyişdirəndə həyata keçirilir: əməliyyatların icra sırasını koddakı sıra ilə uyğunlaşdırır, hətta bəyannamələr arasında açıq asılılıq olmasa belə. Məsələn, əgər TF funksiyanızda `v.assign_add(1)` və sonra `v.assign(v * 2)` varsa, TensorFlow bu əməliyyatların həmin sıraya uyğun olaraq icra olunmasını təmin edəcəkdir.

İpucu: Inter-op iplik hovuzundakı ipliklərin sayını

`tf.config.threading.set_inter_op_parallelism_threads()` ilə tənzimləyə bilərsiniz. Intra-op ipliklərin sayını təyin etmək üçün isə `tf.config.threading.set_intra_op_parallelism_threads()` istifadə edin. Bu, TensorFlow-un bütün CPU nüvələrindən istifadə etməməsini və ya tək iplikli olmasını istəyirsinizsə faydalı ola bilər.

Bununla, istənilən əməliyyatı istənilən cihazda icra etmək və GPU-larınızın gücündən faydalanaq üçün lazım olan hər şeyə sahibsiniz! Aşağıdakılardan bəzilərini edə bilərsiniz:

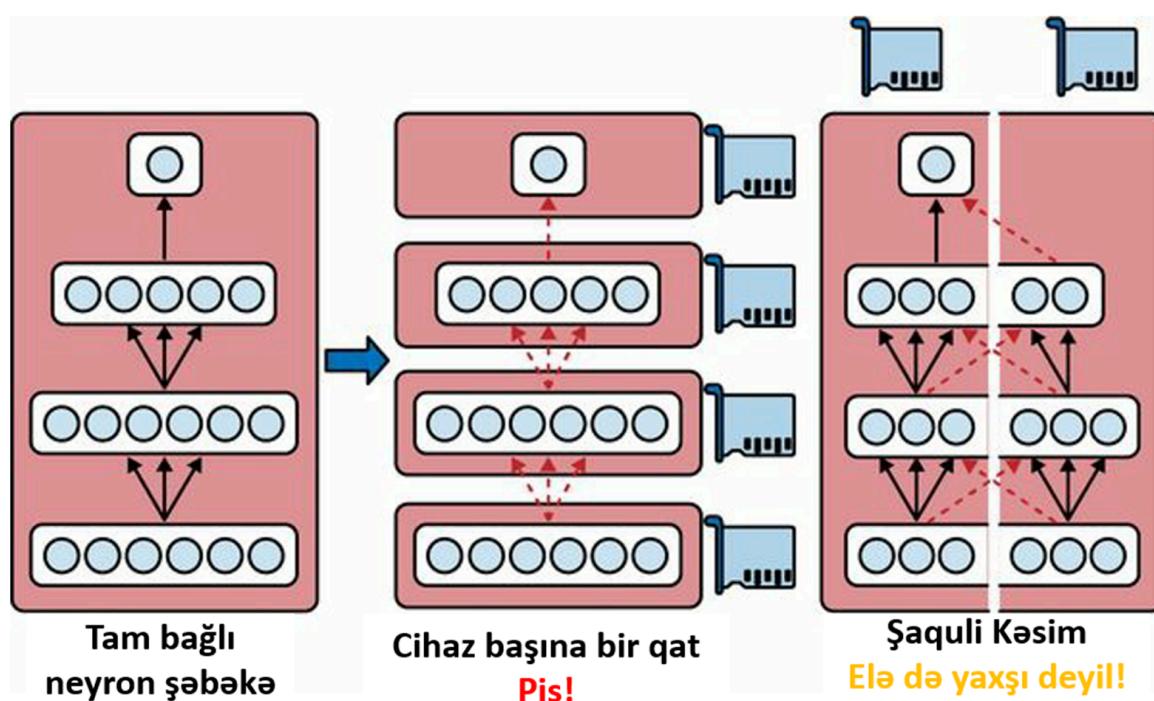
- Bir neçə modeli paralel olaraq, hər birini öz GPU-sunda təlim edə bilərsiniz: hər model üçün bir təlim skripti yazın və onları paralel olaraq çalışdırın, `CUDA_DEVICE_ORDER` və `CUDA_VISIBLE_DEVICES` təyin edərək hər skriptin yalnız bir GPU cihazını görməsini təmin edin. Bu, hiperparametr tənzimlənməsi üçün mükəmməldir, çünkü paralel olaraq müxtəlif hiperparametrlərlə bir neçə model təlim edə bilərsiniz. Əgər bir maşında iki GPU varsa və bir modeli bir GPU-da təlim etmək bir saat çəkirsə, onda iki modeli paralel olaraq, hər biri öz xüsusi GPU-sunda təlim etmək, yalnız bir saat çəkəcəkdir. Sadə!
- Bir modeli bir GPU-da təlim edə və bütün məlumat emalını CPU-da paralel olaraq həyata keçirə bilərsiniz, məlumat dəstinin `prefetch()` metodundan istifadə edərək növbəti bir neçə dəsti əvvəlcədən hazırlayaraq GPU-nun ehtiyacı olduğu zaman hazır olmasına təmin edə bilərsiniz (bax Şəxs 13).
- Modeliniz iki şəkil qəbul edirsə və bunları iki CNN istifadə edərək işləyib çıxışlarını birləşdirirsə, hər bir CNN-i fərqli GPU-da yerləşdirərək daha sürətli işləyəcəkdir.
- Effektiv bir ansambl yarada bilərsiniz: sadəcə olaraq, hər bir təlim edilmiş modeli fərqli GPU-da yerləşdirərək, bütün proqnozları daha sürətli əldə edə bilərsiniz ki, ansamblın son proqnozunu tez bir zamanda əldə edə biləsiniz.

Bir neçə Cihazda Model Təlimi

Bir modeli bir neçə cihazda təlim etməyin iki əsas yanaşması var: model paralelliyi, burada model cihazlar arasında bölünür və məlumat paralelliyi, burada model hər cihazda təkrarlanır və hər təkrarlanma fərqli məlumat alt dəsti ilə təlim olunur. Bu iki seçimi nəzərdən keçirək.

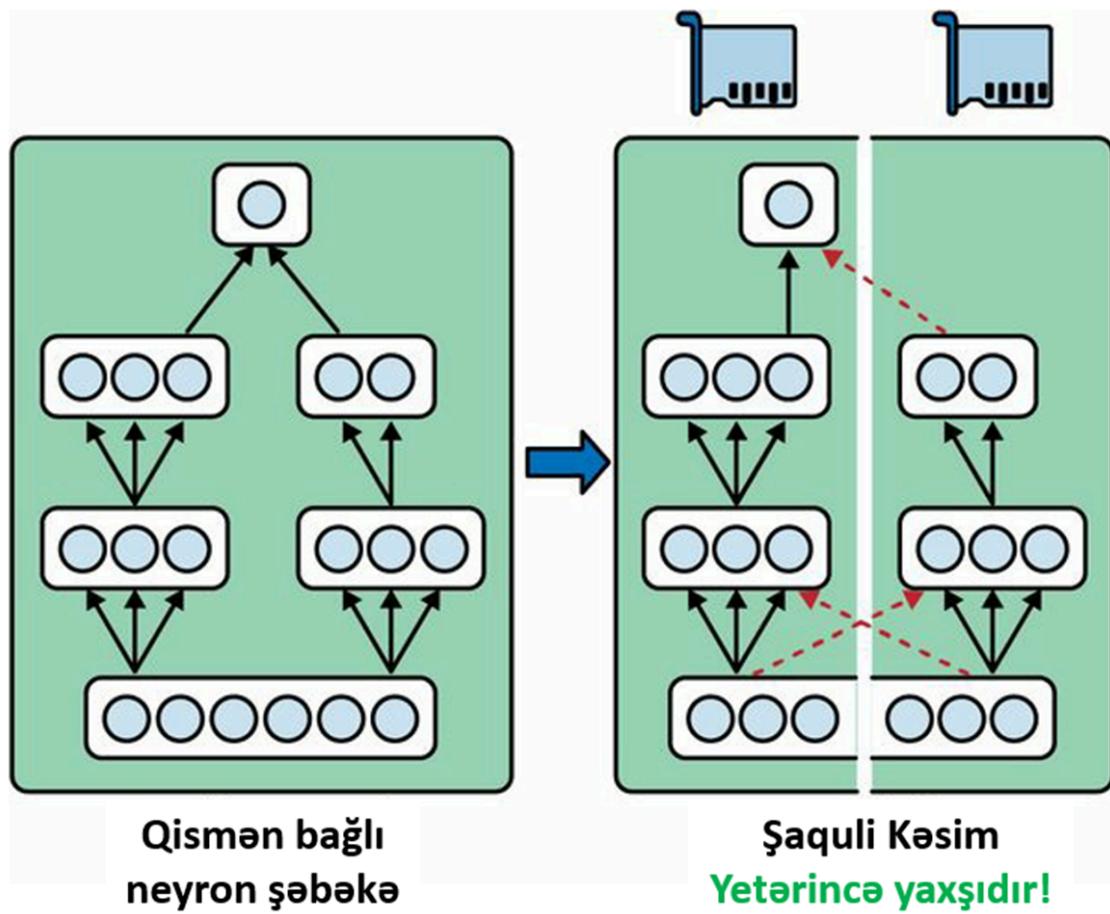
Model Paralelliyi

İndiyə qədər hər neyron şəbəkəsini bir cihazda təlim etdi. Bəs, bir neyron şəbəkəsini bir neçə cihazda təlim etmək istəsək necə olur? Bu, modeli ayrı hissələrə bölmək və hər hissəni fərqli cihazda işə salmaq tələb edir. Təəssüf ki, bu cür model paralelliyi olduqca çətindir və effektivliyi neyron şəbəkəsinin memarlığından asılıdır. Tam bağlı şəbəkələr üçün, bu yanaşmadan adətən çox şey qazanılmır (bax Şəkil 19-11). Intuitiv olaraq, modelin bölünməsinin asan bir yolu kimi hər təbəqəni fərqli cihazda yerləşdirmək görünə bilər, amma bu, hər təbəqənin əvvəlki təbəqənin çıxışını gözləməsi səbəbindən işləməz. Bəlkə, modeli şaquli olaraq kəsə bilərsiniz—for example, hər təbəqənin sol yarısı bir cihazda, sağ hissəsi digər cihazda? Bu, bir qədər yaxşıdır, çünki hər təbəqənin iki yarısı parallel işləyə bilər, amma problem odur ki, növbəti təbəqənin hər iki yarısı da hər iki yarının çıxışına ehtiyac duyur, buna görə də çox cihazlı əlaqə (xəttli oxlarla təmsil olunan) olacaqdır. Bu, parallel hesablamanın faydasını tamamilə azalda bilər, çünki çox cihazlı əlaqə yavaşdır (və cihazlar fərqli maşınlarda yerləşdikdə daha da yavaş olur).



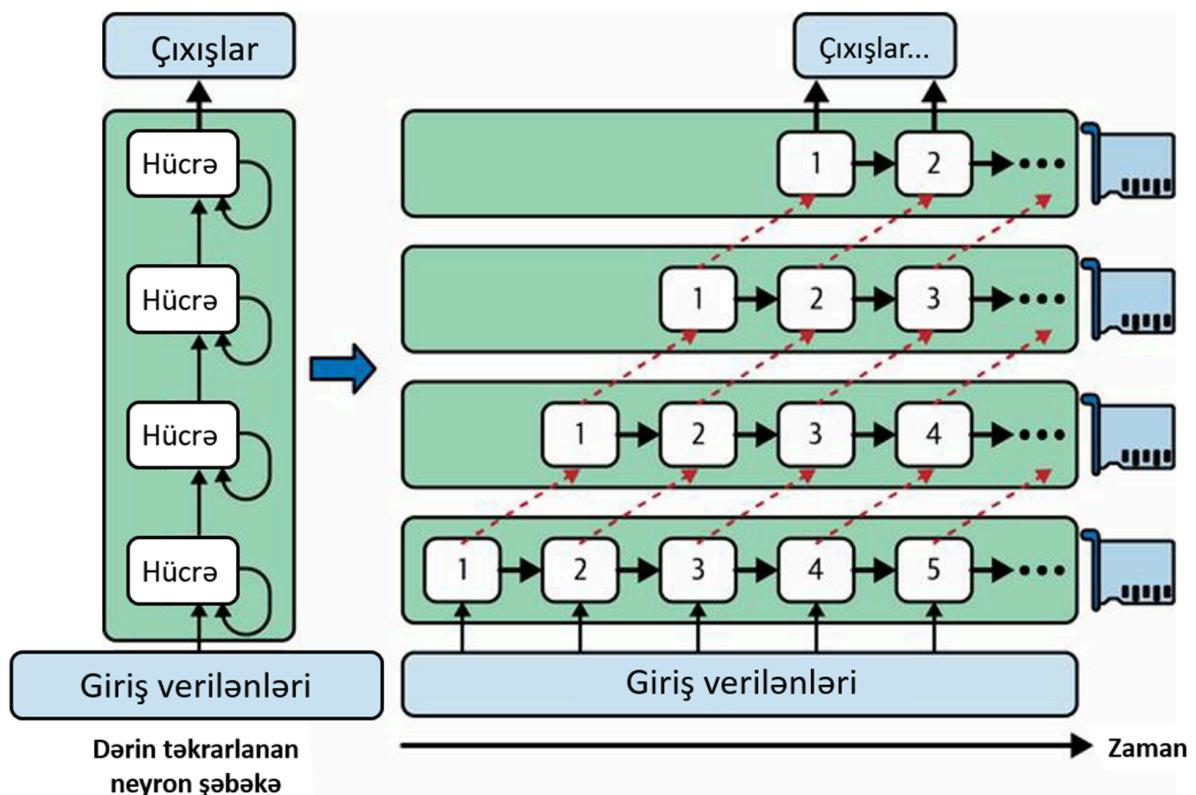
Şəkil 19-11. Tam bağlı neyron şəbəkəsinin bölünməsi

Bəzi neyron şəbəkə memarılıarı, məsələn, konvolyusional neyron şəbəkələri (bax Şəxs 14), aşağı təbəqələrə yalnız qismən bağlı olan təbəqələrə malikdir. Bu səbəbdən, bu cür memarılıarı cihazlar arasında səmərəli şəkildə bölmək daha asandır (Şəkil 19-12).



Şəkil 19-12. Qismən bağlı neyron şəbəkəsinin bölünməsi

Dərin təkrarlanan neyron şəbəkələri (bax Şəxs 15) bir qədər daha səmərəli şəkildə çoxlu GPU-lar arasında bölünə bilər. Şəbəkəni üfüqi olaraq bölərək hər təbəqəni fərqli bir cihazda yerləşdirsiniz və şəbəkəni işlənməsi üçün bir giriş ardıcılılığı ilə təmin etsəniz, ilk vaxt addımdında yalnız bir cihaz aktiv olacaq (ardıcılığın ilk dəyəri üzərində işləyəcək), ikinci addımda iki cihaz aktiv olacaq (ikinci təbəqə ilk dəyər üçün ilk təbəqənin çıxışını idarə edəcək, birinci təbəqə isə ikinci dəyəri idarə edəcək) və siqnal çıkış təbəqəsinə çatdıqda bütün cihazlar eyni vaxtda aktiv olacaq (Şəkil 19-13). Hələ də çoxlu cihazlar arasında ünsiyyət var, amma hər bir hücrə kifayət qədər mürəkkəb ola bilər, buna görə də nəzəri olaraq çoxlu hücrələri paralel işlətmək ünsiyyət cərimini üstələyə bilər. Lakin praktikada, adı bir LSTM təbəqələr yığını tək GPU-da daha sürətli işləyir.



Şəkil 19-13. Dərin təkrarlanan neyron şəbəkəsinin bölünməsi

Qısacısı, model paralelliyi bəzi növ neyron şəbəkələrinin işləmə və ya təlimini sürətləndirə bilər, amma hamısına aid deyil və xüsusi diqqət və tənzimləmə tələb edir, məsələn, ən çox ünsiyyət tələb edən cihazların eyni anda çalışmasını təmin etmək. İndi çox daha sadə və ümumiyyətlə daha səmərəli bir seçimə baxacaqıq: məlumat paralelliyi.

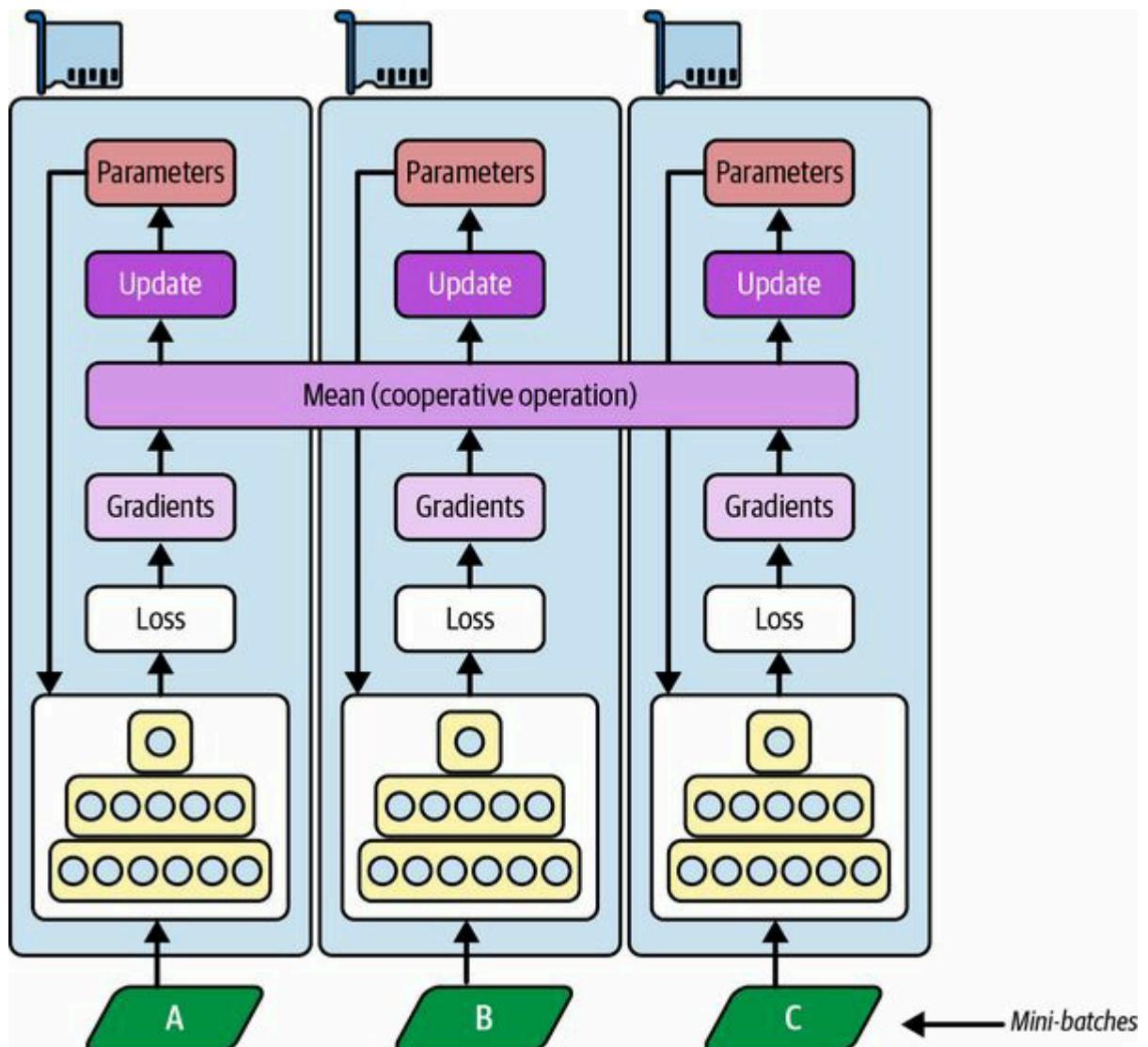
Məlumat Paralelliyi

Neyron şəbəkəsinin təlimini paralelləşdirmək üçün digər bir üsul, onu hər cihazda çoxaltmaq və hər bir təlim addımını bütün çoxaltmalar üzərində eyni vaxtda icra etməkdir, hər biri üçün fərqli mini-partiya istifadə edərək. Hər bir çoxaltmanın hesablaşığı gradientlər sonra ortalanır və nəticə model parametrlərini yeniləmək üçün istifadə olunur. Bu, məlumat paralelliyi, bəzən isə bir program, çoxlu məlumat (SPMD) adlanır.

Bu ideyanın bir çox variantı var, buna görə də ən vaciblərinə nəzər salaq.

Güzəştli strategiya ilə məlumat paralelliyi

Ən sadə yanaşmalardan biri, bütün model parametrlərini bütün GPU-lar arasında tam şəkildə eks etdirmək və hər GPU-da eyni parametrləri yeniləməkdir. Bu şəkildə, bütün çoxaltmalar hər zaman mükəmməl eyni qalır. Bu, güzəştli strategiya adlanır və bir anda istifadə edildikdə olduqca səmərəli olur (Şəkil 19-14).

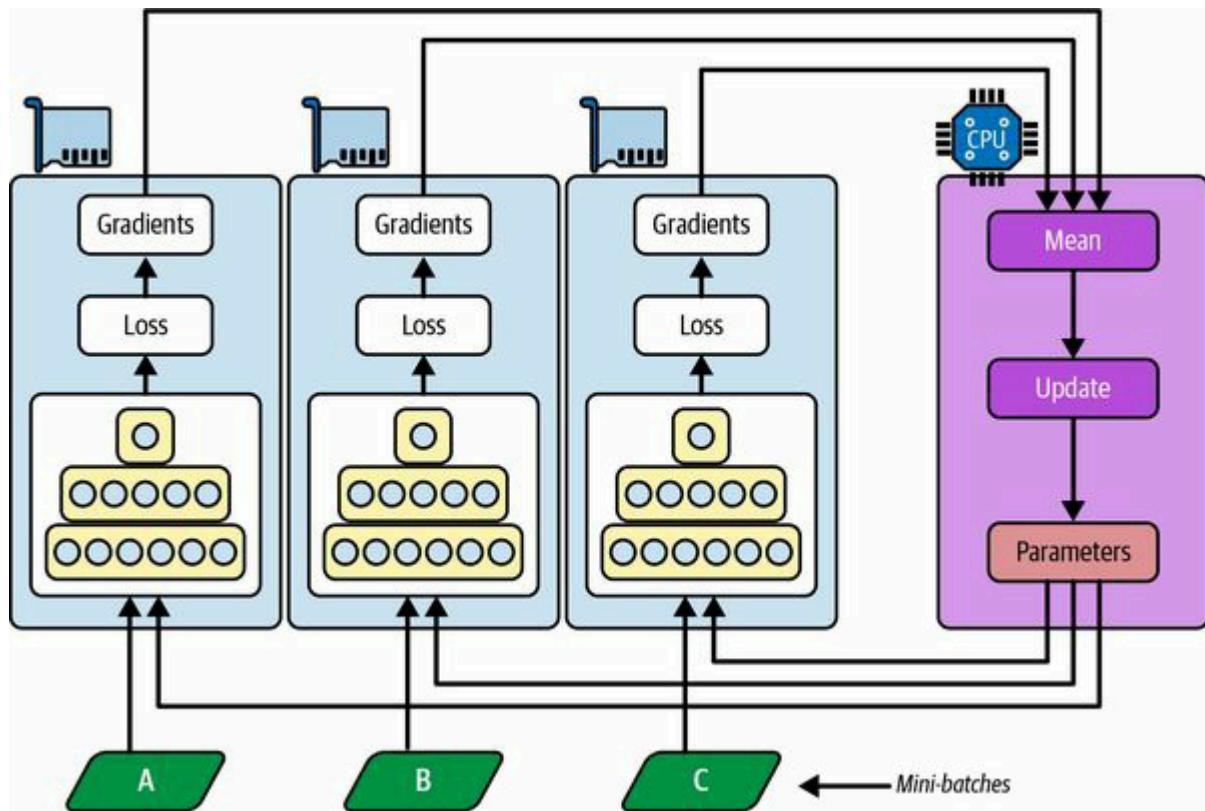


Şəkil 19-14. Güzəştli strategiya ilə məlumat paralelliyi

Bu yanaşma istifadə edərkən çətin tərəfi, bütün GPU-lardan gələn gradientlərin ortalamasını effektiv şəkildə hesablamaq və nəticəni bütün GPU-lara paylamaqdır. Bu, AllReduce algoritmi istifadə edərək həyata keçirilə bilər, bu algoritmalar çoxlu düyünlərin səmərəli şəkildə bir reduce əməliyyatı (məsələn, ortalama, cəmləmə və maksimum hesablamaları) yerinə yetirmək üçün əməkdaşlıq etməsini təmin edir, eyni zamanda bütün düyünlərin eyni son nəticəni əldə etməsini təmin edir. Şükürlər olsun ki, belə algoritmaların hazır implementasiyaları mövcuddur, bunları görəcəksiniz.

Mərkəzləşdirilmiş parametrlərlə məlumat paralelliyi

Digər bir yanaşma model parametrlərini hesablama aparan GPU cihazlarının xaricində saxlamaqdır (işçi adlandırılan); məsələn, CPU-da (Şəkil 19-15). Paylanmış bir quruluşda, bütün parametrləri yalnız parametrləri saxlayıb yeniləyən bir və ya daha çox CPU-yə əsaslanan serverlərdə yerləşdirə bilərsiniz, bunlar parametr serverləri adlanır.



Şəkil 19-15. Mərkəzləşdirilmiş parametrlərlə məlumat paralelliyi

Güzəştli strategiya bütün GPU-larda sinxron çəki yeniləmələri tətbiq edərkən, bu mərkəzləşdirilmiş yanaşma həm sinxron, həm də asinxron yeniləmələrə imkan tanır. Gəlin hər iki seçimin müsbət və mənfi tərəflərinə baxaq.

Sinxron yeniləmələr

Sinxron yeniləmələrdə, toplayıcı bütün gradientlər mövcud olana qədər gözləyir, sonra ortalama gradientləri hesablayır və optimizerə ötürür ki, bu da model parametrlərini yeniləyəcək. Bir nüsxə gradientlərini hesablaması bitirdikdə, növbəti mini-batch-ə keçməkdən əvvəl parametrlərin yenilənməsini gözləməlidir. Mənfi tərəfi, bəzi cihazların digərlərindən daha yavaş olmasına buna görə də sürətli cihazlar hər addımda yavaş olanları gözləməli olacaq, bu da bütün prosesi ən yavaş cihazın sürəti qədər yavaş edir. Bundan əlavə, parametrlər hər cihazda demək olar ki, eyni anda (gradientlər tətbiq edildikdən dərhal sonra) kopyalanacaq, bu da parametr serverlərinin bant genişliyini doydura bilər.

İpucu

Hər addımda gözləmə müddətini azaltmaq üçün, ən yavaş bir neçə nüsxənin (adətən ~10%) gradientlərini görməməzlikdən gələ bilərsiniz. Məsələn, 20 nüsxə işlədə bilərsiniz, amma hər addımda yalnız ən sürətli 18 nüsxənin gradientlərini toplayaraq, son 2-nin gradientlərini görməzlikdən gələ bilərsiniz. Parametrlər yeniləndikdən dərhal sonra ilk 18 nüsxə yenidən işə başlayacaq, ən yavaş 2 nüsxəni gözləməyə ehtiyac olmayacaq. Bu quruluş ümumiyyətlə 18 nüsxə və 2 ehtiyat nüsxə kimi təsvir edilir.

Asinxron yeniləmələr

Asinxron yeniləmələrdə, hər dəfə bir nüsxə gradientləri hesablaması bitirdikdə, bu gradientlər dərhal model parametrlərini yeniləmək üçün istifadə olunur. Burada toplama yoxdur (Şəkil 19-15-də "orta" addımı aradan qaldırır) və sinxronizasiya da yoxdur. Nüsxələr digər nüsxələrdən müstəqil olaraq işləyir. Digər nüsxələri gözləmədiyi üçün bu yanaşma dəqiqlikədə daha çox təlim addımı yerinə yetirir. Bundan əlavə, parametrlərin hər addımda hər cihazda kopyalanması hələ də baş verir, lakin bu, hər nüsxə üçün fərqli vaxtlarda baş verir, buna görə də bant genişliyinin doydurulma riski azalır.

Asinxron yeniləmələr ilə məlumat paralelliyi sadəliyi, sinxronizasiya gecikmələrinin olmaması və bant genişliyindən daha yaxşı istifadə etməsi səbəbindən cəlbedici bir seçimdir. Lakin, praktikada olduqca yaxşı işləsə də, belə işlədiyi demək olar ki, təəccüblüdür! Həqiqətən də, bir nüsxə bəzi parametr dəyərlərinə əsaslanaraq gradientləri hesablaşmayı bitirdikdə, bu parametrlər digər nüsxələr tərəfindən dəfələrlə (ortalama $N - 1$ dəfə, əgər N nüsxə varsa) yenilənmiş olacaq və hesablanmış gradientlərin hələ də doğru istiqamətə işaret etməsi zəmanət olunmur (Şəkil 19-16). Gradientlər ciddi şəkildə köhnəldikdə, köhnəlmış gradientlər adlanır: bunlar konvergensiyani yavaşlada bilər, səs-küy və titrəmə effektləri (öyrənmə əyrisi müvəqqəti osilasiya göstərə bilər) əlavə edə bilər və ya hətta təlim algoritmini yayındırı bilər.

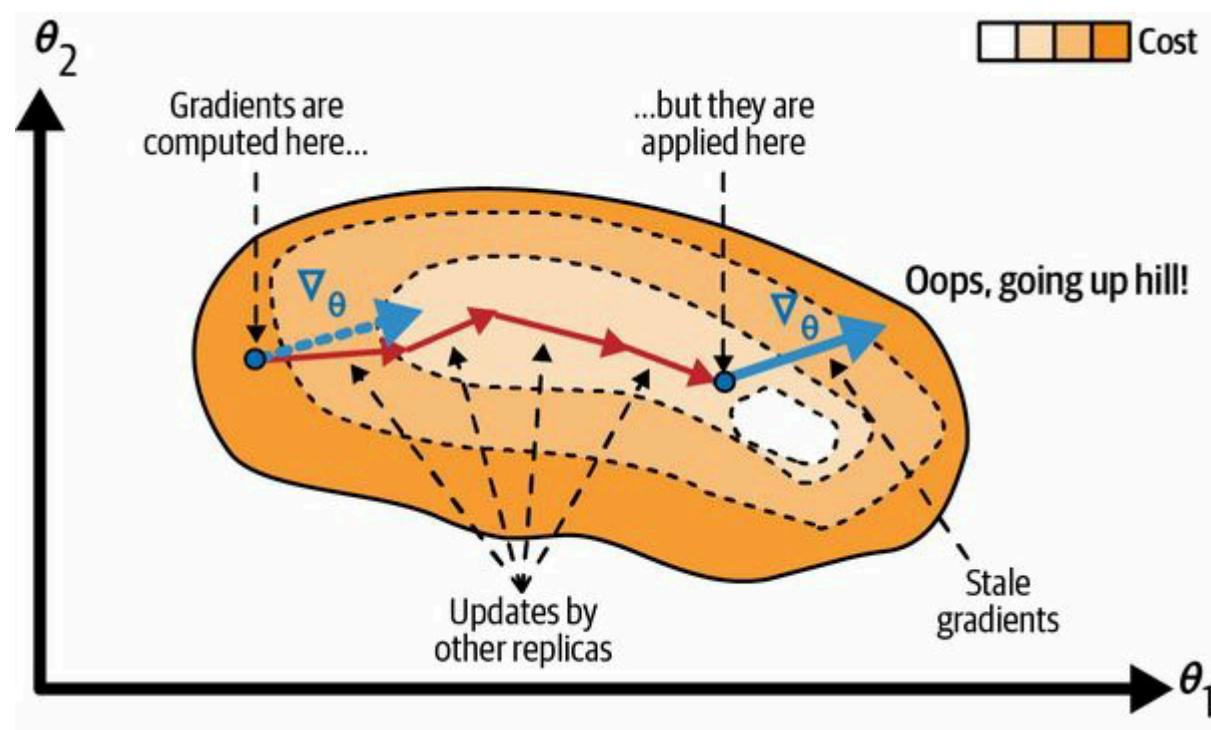
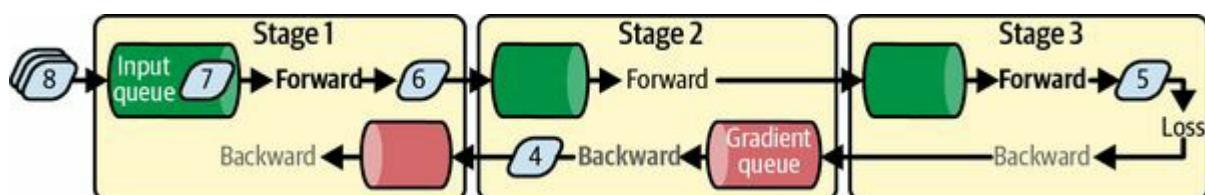


Figure 19-16. Stale gradients when using asynchronous updates

Stal gradientlərin təsirini azaltmağın bir neçə yolu var: Öyrənmə sürətini azaltmaq. Stal gradientləri atmaq və ya onları kiçitmək. Mini-batch ölçüsünü tənzimləmək. İlk bir neçə epoch-u sadəcə bir replikada başlamaq (bu, warmup mərhələsi adlanır). Stal gradientlər adətən təlimin başlangıcında daha zərərli olur, çünki bu zaman gradientlər adətən böyük olur və parametrlər hələ də xərclər funksiyasının çökəkliyinə yerləşməyiylər, beləliklə müxtəlif replikalar parametrləri fərqli istiqamətlərə itləyə bilirlər. 2016-cı ildə Google Brain komandası tərəfindən nəşr olunan bir məqalədə müxtəlif yanaşmaların müqayisəsi aparılmış və bir neçə ehtiyat replika ilə sinxron

yeniləmələrdən istifadə etməyin asinxron yeniləmələrdən daha səmərəli olduğu, yalnız daha sürətli yaxınlaşmaqla yanaşı, həm də daha yaxşı model yaratdığı müəyyən olunmuşdur. Lakin, bu hələ də aktiv tədqiqat sahəsidir, buna görə asinxron yeniləmələri hələ ki istisna etməməlisiniz. Bant genişliyinin doyması Sinxron və ya asinxron yeniləmələrdən istifadə etməyinizdən asılı olmayaraq, mərkəzləşdirilmiş parametrlərlə verilənlər paralelliyi hələ də hər təlim mərhələsinin əvvəlində parametrlərin parametrlər serverlərindən hər replikağa ötürülməsini və hər təlim mərhələsinin sonunda gradientlərin əks istiqamətdə ötürülməsini tələb edir. Eynilə, əks etdirilən strategiyadan istifadə edərkən, hər bir GPU tərəfindən istehsal edilən gradientlər digər GPU-larla paylaşılmalıdır. Təəssüf ki, əlavə bir GPU əlavə etməyin performansın heç bir şəkildə yaxşılaşdırılmayacağı bir nöqtə gəlir, çünki məlumatların GPU RAM-ına və GPU RAM-ından köçürülməsinə sərf olunan vaxt (və paylanması bir quruluşda şəbəkə boyunca) hesablama yükünü bölməklə əldə edilən sürətləndirməni üstələyəcək. O nöqtədə, daha çox GPU əlavə etmək sadəcə bant genişliyinin doymasını pisləşdirəcək və təlimi həqiqətən yavaşlaşdıracaq. Doyma böyük six modellər üçün daha ciddi olur, çünki onlarda ötürüləcək çoxlu parametrlər və gradientlər olur. Kiçik modellər üçün isə daha az ciddi olur (lakin paralelləşdirmə qazancı məhduddur) və böyük seyrək modellər üçün isə, burada gradientlər adətən əsasən sıfır olur və buna görə də effektiv şəkildə ötürülmə bilir. Google Brain layihəsinin təşəbbüskarı və rəhbəri Jeff Dean six modellər üçün 50 GPU arasında hesablama işlərini paylayarkən tipik sürətləndirmələrin $25\text{--}40\times$ olduğunu və 500 GPU arasında öyrədilmiş daha seyrək modellər üçün isə $300\times$ sürətləndirmə olduğunu bildirdi. Gördüyünüz kimi, seyrək modellər həqiqətən daha yaxşı miqyaslanır. Budur bir neçə konkret nümunə: Neural machine translation: 8 GPU-da $6\times$ sürətləndirmə Inception/ImageNet: 50 GPU-da $32\times$ sürətləndirmə RankBrain: 500 GPU-da $300\times$ sürətləndirmə Bant genişliyinin doyması problemini yüngülləşdirmək üçün çoxlu tədqiqatlar aparılır, məqsəd mövcud GPU-ların sayı ilə təlimin xətti şəkildə miqyaslanması imkan verməkdir. Məsələn, Carnegie Mellon Universiteti, Stanford Universiteti və Microsoft Research-dən bir qrup tədqiqatçının 2018-ci ildə nəşr etdirdiyi bir məqalədə PipeDream adlanan bir sistem təklif edildi ki, bu da şəbəkə kommunikasiya yükünü $90\%-dən$ çox azaltdı və bir çox maşında böyük modellərin təliminə imkan verdi. Onlar bunu model paralelliyi və verilənlər paralelliyini birləşdirən yeni texnika olan boru kəməri paralelliyi adlı üsuldan istifadə edərək nail oldular: model ardıcıl hissələrə, mərhələlərə bölünür və hər biri fərqli maşında öyrədir. Bu, bütün maşınların çox az boş vaxtla paralel işlədiyi asinxron bir boru kəməri ilə nəticələnir. Təlim zamanı hər bir mərhələ irəli yayılmanın bir turunu və bir geri yayılmanın bir turunu növbə ilə keçirir (Şəkil 19-17-ə baxın): o, giriş növbəsindən bir mini-batch çəkir, onu işləyir və nəticələri növbəti mərhələnin giriş növbəsinə göndərir, sonra gradient növbəsindən bir mini-batch gradient çəkir, bu gradientləri geri yayır və öz model parametrlərini yeniləyir və geri yayılmış gradientləri əvvəlki mərhələnin gradient növbəsinə göndərir. Sonra bu prosesi təkrar-təkrar təkrar edir. Hər bir mərhələ, digər mərhələlərdən asılı olmayaraq, müntəzəm məlumat paralelliyindən (məsələn, əks etdirilən strategiyadan istifadə etməklə) istifadə edə bilər.



Şəkil 19-17. PipeDream-in boru kəməri paralelliyi

Lakin burada təqdim edildiyi şəkildə, PipeDream o qədər də yaxşı işləməyəcək. Bunun səbəbini başa düşmək üçün Şəkil 19-17-də mini-batch #5-i nəzərə alın: bu mini-batch irəliləyən kecid zamanı mərhələ 1-dən keçidkədə, mini-batch #4-dən gələn gradientlər həmin mərhələdən geri yayılmamışdı,

amma #5-in gradientləri mərhələ 1-ə qayıtdıqda, #4-ün gradientləri model parametrlərini yeniləmək üçün istifadə olunacaq, buna görə də #5-in gradientləri bir az köhnəlmış olacaq. Gördüyüümüz kimi, bu təlim sürətini və dəqiqliyini azalda bilər və hətta onun sapmasına səbəb ola bilər: mərhələlərin sayı artıraq, bu problem daha da pisləşir. Məqalənin müəllifləri bu problemi yüngülləşdirmək üçün metodlar təklif etdilər: məsələn, hər mərhələ irəliləyən yayılma zamanı çəkiləri saxlayır və geri yayılma zamanı onları bərpa edir, beləliklə həm irəliləyən kecid, həm də geri kecid üçün eyni çəkilər istifadə olunur. Bu, ağırlıq saxlama adlanır. Bunun sayesində, PipeDream təsirli miqyaslanma qabiliyyəti göstərir, sadə məlumat paralelliyyindən çox daha irəlidir.

Bu sahədəki son irəliləyiş, 2022-ci ildə Google tədqiqatçıları tərəfindən nəşr olunan bir məqalədə təqdim edildi: onlar avtomatlaşdırılmış model paralelliyi, asinxron qrup planlaşdırması və digər texnikalardan istifadə edən Pathways adlı bir sistem inkişaf etdirdilər ki, bu da minlərlə TPU üzərində hardware istifadəni 100%-ə yaxınlaşdırır! Planlaşdırma, hər bir tapşırığın nə zaman və harada icra olunacağına təşkil etməyi nəzərdə tutur, qrup planlaşdırması isə əlaqəli tapşırıqları eyni vaxtda parallel və bir-birinə yaxın şəkildə icra etməyi, beləliklə tapşırıqların digər tapşırıqların nəticələrini gözləmək vaxtnı azaltmağı nəzərdə tutur. Bölüm 16-da gördüyüümüz kimi, bu sistem 6,000-dən çox TPU üzərində böyük bir dil modelini təlim etmək üçün istifadə edildi və hardware istifadəni 100%-ə yaxınlaşdırıldı: bu, həqiqətən təsir edici bir mühəndislik nailiyyətidir.

Məqalənin yazılıdığı vaxtda Pathways hələ ictimaiyyətə təqdim edilməyib, amma yaxın gələcəkdə Pathways və ya oxşar bir sistemlə Vertex AI üzərində böyük modelləri öyrətmək mümkün olacaq. Bu arada, doyması problemini azaltmaq üçün, çox sayda zəif GPU əvəzinə bir neçə güclü GPU istifadə etmək istəyə bilərsiniz və bir modeli bir neçə serverdə öyrətmək lazımlı olduqda, GPU-ları bir neçə və çox yaxşı əlaqələndirilmiş serverdə qruplaşdırılmalısınız. Həmçinin, float dəqiqliyini 32 bitdən (`tf.float32`) 16 bitə (`tf.bfloat16`) keçirməyi sinaya bilərsiniz. Bu, ötürüləcək məlumat miqdarını yarıya endirəcək, tez-tez yaxınlaşma sürətinə və modelin performansına çox təsir etmədən. Nəhayət, əgər mərkəzləşdirilmiş parametrlərdən istifadə edirsizsə, parametrləri bir neçə parametr serveri arasında bölgə bilərsiniz: əlavə parametr serverləri əlavə etmək hər bir serverdə şəbəkə yükünü azaldacaq və bant genişliyinin doyması riskini məhdudlaşdıracaq.

İndi bütün nəzəriyyəni keçdik, gəlin faktiki olaraq bir modeli bir neçə GPU arasında öyrədək!

Təqdimat Strategiyaları API istifadə edərək Miqyasda Təlim

Xoşbəxtlikdən, TensorFlow çox gözəl bir API ilə gəlir ki, bu da modelinizi bir neçə cihaz və maşın arasında bölüşdürümə kompleksliyini idarə edir: bölüşdürümə strategiyaları API-si. Data paralelliyi ilə mirrör strategiyasını istifadə edərək bir Keras modelini mövcud bütün GPU-lar arasında (hazırda tək bir maşında) öyrətmək üçün sadəcə bir MirroredStrategy obyekti yaradın, onun `scope()` metodunu çağıraraq bir bölüşdürümə konteksti əldə edin və modelinizi yaratma və tərtib etməni bu kontekst daxilində həyata keçirin. Sonra modelin `fit()` metodunu adı şəkildə çağırın:

python

Copy code

```
strategy = tf.distribute.MirroredStrategy()  
  
with strategy.scope():
```

```
model = tf.keras.Sequential([...]) # Keras modelini adı şəkildə  
yaradın  
  
model.compile([...]) # Modeli adı şəkildə tərtib edin  
  
batch_size = 100 # Preferably divisible by the number of  
replicas  
  
model.fit(X_train, y_train, epochs=10,  
  
           validation_data=(X_valid, y_valid),  
batch_size=batch_size)
```

Arxa planda, Keras bölüşdurmə ilə əlaqədardır, beləliklə bu MirroredStrategy kontekstində bütün dəyişənləri və əməliyyatları mövcud GPU cihazları arasında təkrarlamalı olduğunu bilir. Modelin çəkilərinə baxsanız, onların növü **MirroredVariable** olacaq:

python

Copy code

```
>>> type(model.weights[0])  
  
tensorflow.python.distribute.values.MirroredVariable
```

Qeyd edək ki, **fit()** metodu hər təlim batch-ını bütün replikalar arasında avtomatik olaraq böləcək, buna görə də batch ölçüsünün replikaların (yəni mövcud GPU-ların) sayı ilə bölünməsini təmin etmək daha yaxşıdır ki, bütün replikalar eyni ölçüdə batch-lar alsın. Və bu qədər! Təlim adətən tək bir cihazdan istifadə edərkən çox daha sürətli olacaq və kod dəyişikliyi həqiqətən minimaldır.

Modelinizi öyrətdikdən sonra, onu effektiv şəkildə proqnozlaşdırmaq üçün istifadə edə bilərsiniz: **predict()** metodunu çağırın və o, batch-ı bütün replikalar arasında avtomatik olaraq böləcək, proqnozları paralel olaraq həyata keçirəcək. Yenə də, batch ölçüsü replikaların sayı ilə bölünməlidir. Modelin **save()** metodunu çağırığınızda, model adı model kimi saxlanacaq, çoxlu replikalarla mirrored model kimi deyil. Beləliklə, onu yüklədikdə, model adı şəkildə, tək bir cihazda işləyəcək: standart olaraq GPU #0-da və ya GPU yoxdursa CPU-da. Modeli yüklemək və bütün mövcud cihazlarda işlətmək istəyirsinizsə, bir bölüşdurmə kontekstində **tf.keras.models.load_model()** çağırımlısınız:

python

Copy code

```
with strategy.scope():
```

```
model = tf.keras.models.load_model("my_mirrored_model")
```

Əgər mövcud GPU cihazlarının bir alt qrupunu istifadə etmək istəyirsinizsə, siyahını MirroredStrategy-nin konstrukturuna keçirə bilərsiniz:

python

Copy code

```
strategy = tf.distribute.MirroredStrategy(devices=[ "/gpu:0",
"/gpu:1" ])
```

MirroredStrategy sinfi standart olaraq AllReduce orta əməliyyatı üçün NVIDIA Collective Communications Library (NCCL) istifadə edir, lakin bunu `cross_device_ops` arqumentini `tf.distribute.HierarchicalCopyAllReduce` sinfinin nümunəsi və ya `tf.distribute.ReductionToOneDevice` sinfinin nümunəsi ilə dəyişə bilərsiniz. Standart NCCL seçimi `tf.distribute.NcclAllReduce` sinfi əsasında qurulub ki, bu da adətən daha sürətlidir, lakin bu GPU-ların sayı və növlərindən asılıdır, buna görə də alternativləri sınamağı istəyirsiniz.

Mərkəzləşdirilmiş parametrlərlə data paralelliyindən istifadə etməyi sınamaq istəyirsinizsə, MirroredStrategy-ni CentralStorageStrategy ilə əvəz edin:

python

Copy code

```
strategy = tf.distribute.experimental.CentralStorageStrategy()
```

Opsiyonel olaraq, `compute_devices` arqumentini istifadə edərək işləyici kimi istifadə etmək istədiyiniz cihazların siyahısını təyin edə bilərsiniz — standart olaraq bütün mövcud GPU-ları istifadə edəcək — və `parameter_device` arqumentini istifadə edərək parametrləri saxlamaq istədiyiniz cihazı təyin edə bilərsiniz. Standart olaraq, CPU istifadə ediləcək, yalnız bir GPU varsa GPU istifadə ediləcək.

İndi TensorFlow klasterində bir modeli necə öyrətmək olacağını görə bilək!

TensorFlow Klasterində Modelin Təlimi

TensorFlow klasteri paralel şəkildə çalışan TensorFlow proseslərindən ibarət bir qrupdur, adətən müxtəlif maşınlarda olur və bəzi işləri tamamlayır—məsələn, neyron şəbəkə modelinin təlimi və ya icrası. Klasterdəki hər bir TF prosesi tapşırıq, və ya TF serveri adlanır. Onun IP ünvanı, portu və növü

(həmçinin rolü və ya işi) var. Növ ya "worker" (işçi), "chief" (başçı), "ps" (parametr serveri) və ya "evaluator" (qiymətləndirici) ola bilər:

- **Worker**: Hesablamalar həyata keçirir, adətən bir və ya daha çox GPU olan maşında.
- **Chief**: Hesablamalar da həyata keçirir (işçi kimi fəaliyyət göstərir), lakin əlavə işlər də görür, məsələn, TensorBoard loglarını yazmaq və ya checkpoint-ləri saxlamaq. Klasterdə tək bir başçı olur. Əgər başçı açıq şəkildə təyin edilməyibsa, adətən ilk işçi başçı olur.
- **Parameter Server**: Yalnız dəyişən dəyərlərini saxlayır və adətən CPU yalnız maşındadır. Bu tip tapşırıq yalnız ParameterServerStrategy ilə istifadə olunur.
- **Evaluator**: Qiymətləndirmə işini yerinə yetirir. Bu tip adətən çox istifadə edilmir və istifadə edildikdə adətən tək qiymətləndirici olur.

TensorFlow klasterini başlatmaq üçün ilk əvvəl onun spesifikasiyasını müəyyən etməlisiniz. Bu, hər bir tapşırığın IP ünvanını, TCP portunu və növünü təyin etmək deməkdir. Məsələn, aşağıdakı klaster spesifikasiyası üç tapşırıqdan ibarət bir klasteri (iki işçi və bir parametr serveri) təyin edir. Klaster spesifikasiyası hər iş üçün bir açar olan bir lüğətdir və dəyərlər tapşırıq ünvanlarının (IP

) siyahılarıdır:

python

Copy code

```
cluster_spec = {

    "worker": [
        "machine-a.example.com:2222", # /job:worker/task:0
        "machine-b.example.com:2222"   # /job:worker/task:1
    ],
    "ps": [ "machine-a.example.com:2221" ] # /job:ps/task:0
}
```

Ümumiyyətlə, hər maşında bir tapşırıq olacaq, amma bu nümunədə göstərildiyi kimi, eyni maşında bir neçə tapşırıq konfiqurasiya edə bilərsiniz, əgər istəyirsizsə. Bu halda, əgər eyni GPU-ları paylaşırlar, RAM-ın uyğun şəkildə bölündüyündən əmin olun, əvvəlki müzakirələrdə qeyd olunduğu kimi.

XƏBƏRDARLIQ

Standart olaraq, klasterdəki hər bir tapşırıq digər bütün tapşırıqlarla ünsiyyət qura bilər, buna görə də bu maşınlar arasında bu portlarda bütün ünsiyyətləri icazə verən firewall tənzimləmələrini həyata keçirdiyinizə əmin olun (hər maşında eyni portu istifadə etməklə daha sadə olur).

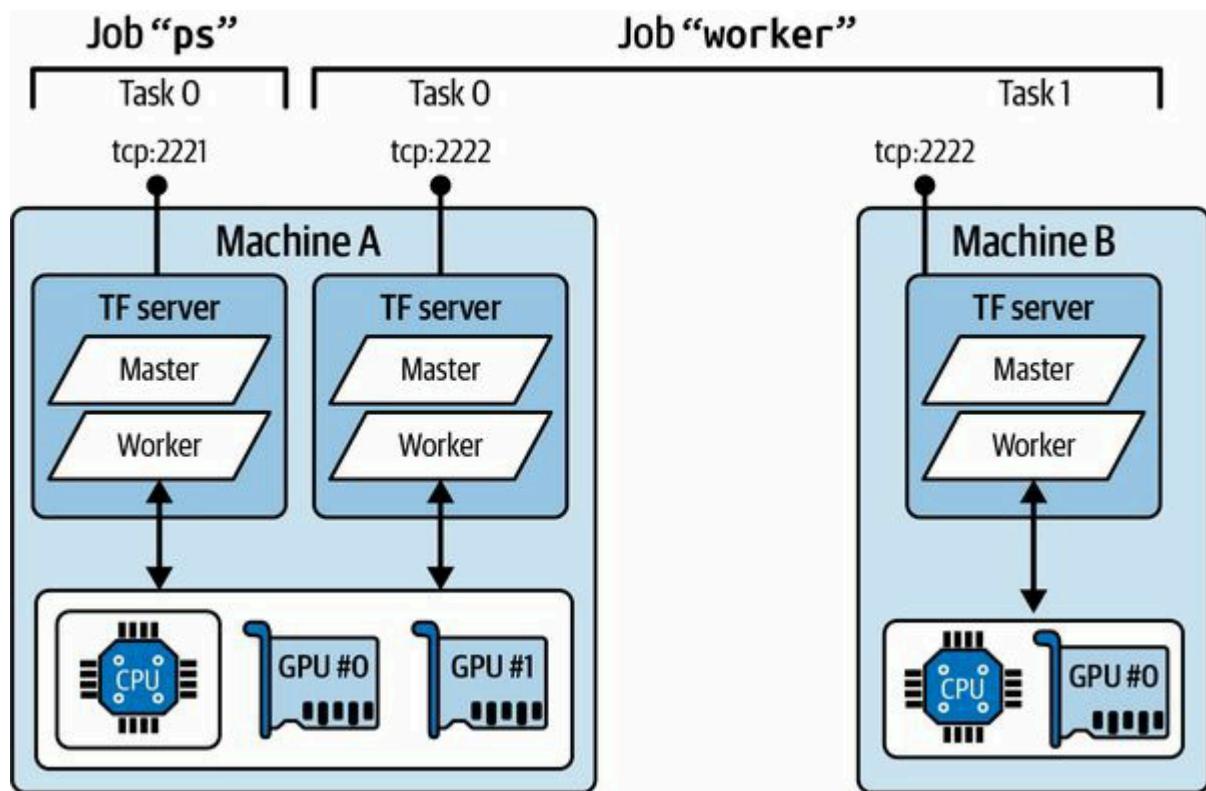


Figure 19-18. An example TensorFlow cluster

Bir tapşırıga başladığınızda, ona klaster spesifikasyasını vermelі və onun növü və indeksini də bildirməlisiniz (məsələn, worker #0). Hər şeyi bir anda (həm klaster spesifikasiyasını, həm də cari tapşırığın növü və indeksini) təyin etmənin ən sadə yolu, TensorFlow-u işə salmadan əvvəl **TF_CONFIG** mühit dəyişənini təyin etməkdir. Bu, "cluster" açarı altında klaster spesifikasiyasını və "task" açarı altında cari tapşırığın növü və indeksini ehtiva edən JSON ilə kodlaşdırılmış bir lüğət olmalıdır. Məsələn, aşağıdakı **TF_CONFIG** mühit dəyişəni, az əvvəl təyin etdiyimiz klasterdən istifadə edir və işə salınacaq tapşırığın worker #0 olduğunu göstərir:

```
python
```

```
os.environ["TF_CONFIG"] = json.dumps({
    "cluster": cluster_spec,
    "task": {"type": "worker", "index": 0}
})
```

İpucu

Ümumiyyətlə, `TF_CONFIG` mühit dəyişənini Python-dan kənarda təyin etmək istəyərsiniz, belə ki, kod cari tapşırığın növü və indeksini daxil etməyə ehtiyac duymayacaq (bu, kodu bütün workers-lər arasında istifadə etməyi mümkün edir).

İndi Klasterdə Modeli Təlim Edək!

Biz `MultiWorkerMirroredStrategy` ilə başlayacaqıq. Əvvəlcə, hər bir tapşırıq üçün `TF_CONFIG` mühit dəyişənini düzgün şəkildə təyin etməlisiniz. Parametr serveri olmamalıdır (klaster spesifikasiyasında "ps" açarını çıxarıın) və ümumiyyətlə hər bir maşında bir worker olmalıdır. Hər bir tapşırıq üçün fərqli bir tapşırıq indeksini təyin etdiyinizə əmin olun. Nəhayət, aşağıdakı skripti hər bir worker-da işə salın:

```
import tempfile  
  
import tensorflow as tf  
  
  
strategy = tf.distribute.MultiWorkerMirroredStrategy() # başında!  
resolver = tf.distribute.cluster_resolver.TFConfigClusterResolver()  
  
print(f"Başlayan tapşırıq {resolver.task_type} #{resolver.task_id}")  
  
  
[...] # MNIST datasetini yükləyin və bölüşdürün  
  
  
with strategy.scope():  
  
    model = tf.keras.Sequential([...]) # Keras modelini yaradın  
  
    model.compile([...]) # Modeli kompilyasiya edin  
  
    model.fit(X_train, y_train, validation_data=(X_valid, y_valid),  
epoch=10)
```

```
if resolver.task_id == 0: # başçı modeli düzgün yerə saxlayır
    model.save("my_mnist_multiworker_model", save_format="tf")
else:
    tmpdir = tempfile.mkdtemp() # digər workers müvəqqəti bir yerə
    # saxlayır
    model.save(tmpdir, save_format="tf")
    tf.io.gfile.rmtree(tmpdir) # və bu müvəqqəti direktori silə
    # bilərik!
```

Bu, əvvəlki istifadə etdiyiniz kodun demək olar ki, eynisidir, yalnız bu dəfə `MultiWorkerMirroredStrategy` istifadə edirsınız. İlk workers bu skripti başladığınızda, AllReduce mərhələsində bloklanacaq, amma təlim son worker başladığı zaman başlayacaq və siz onların hər birinin eyni sürətlə irəlilədiyini görəcəksiniz, çünki hər addımda sinxronizasiya olunurlar.

Xəbərdarlıq

`MultiWorkerMirroredStrategy` istifadə edərkən, bütün workers-lerin eyni işi etdiyinə əmin olmaq vacibdir, məsələn, model checkpoint-lərini saxlamaq və ya TensorBoard log-larını yazmaq, hərçənd ki, yalnız başçı tərəfindən yazılımı saxlayacaqsınız. Bunun səbəbi odur ki, bu əməliyyatlar AllReduce əməliyyatlarını tələb edə bilər, buna görə də bütün workers-lər sinxron olmalıdır.

Bu yayım strategiyası üçün iki AllReduce tətbiqi var: şəbəkə kommunikasiya üçün gRPC əsasında olan bir ring AllReduce alqoritması və NCCL-in tətbiqi. Hansı alqoritmin istifadə ediləcəyi, workers-lərin sayı, GPU-ların sayı və növləri, həmçinin şəbəkəyə bağlıdır. TensorFlow default olaraq bəzi heuristiklərdən istifadə edərək sizin üçün doğru alqoritmi seçəcək, lakin siz aşağıdakı şəkildə NCCL (və ya RING) tətbiqini məcbur edə bilərsiniz:

```
strategy = tf.distribute.MultiWorkerMirroredStrategy()
```

```
communication_options=tf.distribute.experimental.CommunicationOptions(  
    implementation=tf.distribute.experimental.CollectiveCommunication.NCCL  
)  
)
```

Asinxron məlumat paralelliyini parametr serverləri ilə həyata keçirmək istəyirsinzsə, strategiyani **ParameterServerStrategy**-ə dəyişdirin, bir və ya daha çox parametr serveri əlavə edin və **TF_CONFIG**-i hər tapşırıq üçün uyğun şəkildə konfiqurasiya edin. Qeyd edək ki, workers asinxron şəkildə işləsələr də, hər bir worker-dəki replikalar sinxron şəkildə işləyəcək.

Son olaraq, Google Cloud-da TPU-lara çıxışınız varsa—məsələn, Colab-dan istifadə edirsinzə və accelerator növünü TPU olaraq təyin edirsinzə—onda **TPUStrategy** yarada bilərsiniz:

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver()  
  
tf.tpu.experimental.initialize_tpu_system(resolver)  
  
strategy = tf.distribute.experimental.TPUStrategy(resolver)
```

Bu, TensorFlow-u idxal etdikdən dərhal sonra icra edilməlidir. Sonra bu strategiyani normal şəkildə istifadə edə bilərsiniz.

İpucu

Əgər bir tədqiqatçısısanız, TPUs-dan pulsuz istifadə edə bilərsiniz; daha ətraflı məlumat üçün <https://tensorflow.org/tfrc> saytına baxın.

İndi modelləri bir neçə GPU və bir neçə server üzərində təlim edə bilərsiniz: özünüzə bir təbrik edin! Lakin çox böyük bir modeli təlim etmək istəyirsinzsə, çoxlu GPU-lara və çoxlu serverlərə ehtiyacınız olacaq, bu da ya çox hardware almanız, ya da çoxlu cloud virtual maşınlarını idarə

etməyi tələb edir. Cox hallarda, bu infrastruktur təmin etmək və idarə etmək üçün cloud xidmətindən istifadə etmək daha az problemlidir və daha ucuzdur. İndi bunu Vertex AI istifadə edərək necə edəcəyimizi görək.

Vertex AI-də Böyük Təlim İşlərini İcra Etmək

Vertex AI sizə öz təlim kodunuzla xüsusi təlim işləri yaratmağa imkan verir. Əslində, demək olar ki, öz TF klasterinizdə istifadə edəcəyiniz eyni təlim kodunu istifadə edə bilərsiniz. Əsas dəyişiklik edəcəyiniz şey, başçı tərəfindən modelin, yoxlama nöqtələrinin və TensorBoard log-larının harada saxlanmasıdır. Modeli yerli bir direktoriyaya saxlamaq əvəzinə, başçı onu Vertex AI tərəfindən təmin edilən `AIP_MODEL_DIR` mühit dəyişəni vasitəsilə GCS-ə (Google Cloud Storage) saxlamalıdır. Modelin yoxlama nöqtələri və TensorBoard log-ları üçün müvafiq olaraq `AIP_CHECKPOINT_DIR` və `AIP_TENSORBOARD_LOG_DIR` mühit dəyişənlərində olan yolları istifadə etməlisiniz. Əlbəttə, təlim məlumatlarının virtual maşınlardan əldə oluna biləcəyinə əmin olmalısınız, məsələn GCS-də, başqa bir GCP xidməti olan BigQuery-də və ya birbaşa vebdən. Son olaraq, Vertex AI "chief" tapşırıq növünü açıq şəkildə təyin edir, ona görə də başçını `resolved.task_type == "chief"` istifadə edərək müəyyən etməlisiniz, `resolved.task_id == 0` əvəzinə:

```
python
```

```
import os

[...] # digər importlar, MultiWorkerMirroredStrategy yaradın və
resolver

if resolver.task_type == "chief":

    model_dir = os.getenv("AIP_MODEL_DIR") # Vertex AI tərəfindən
    təmin edilən yollar

    tensorboard_log_dir = os.getenv("AIP_TENSORBOARD_LOG_DIR")

    checkpoint_dir = os.getenv("AIP_CHECKPOINT_DIR")

else:

    tmp_dir = Path(tempfile.mkdtemp()) # digər workers müvəqqəti
    direktoriyalar istifadə edir

    model_dir = tmp_dir / "model"
```

```
tensorboard_log_dir = tmp_dir / "logs"
checkpoint_dir = tmp_dir / "ckpt"

callbacks = [tf.keras.callbacks.TensorBoard(tensorboard_log_dir),
            tf.keras.callbacks.ModelCheckpoint(checkpoint_dir)]

[...] # strategiya sahəsi daxilində yaradın və kompilyasiya edin,
əvvəlki kimi

model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
           epochs=10,
           callbacks=callbacks)

model.save(model_dir, save_format="tf")
```

İpucu

Əgər təlim məlumatlarını GCS-ə yerləşdirsiniz, ona daxil olmaq üçün `tf.data.TextLineDataset` və ya `tf.data.TFRecordDataset` yarada bilərsiniz: sadəcə olaraq GCS yollarını fayl adları kimi istifadə edin (məsələn, `gs://my_bucket/data/001.csv`). Bu datasetlər fayllara daxil olmaq üçün `tf.io.gfile` paketinə əsaslanır: həm yerli faylları, həm də GCS fayllarını dəstəkləyir.

İndi Vertex AI-də Xüsusi Təlim İşini Necə Yaratmaq Olar?

Bu skriptə əsaslanan xüsusi təlim işini Vertex AI-də yarada bilərsiniz. İşin adını, təlim skriptinin yolunu, təlim üçün istifadə ediləcək Docker görüntüsünü, təlimdən sonra proqnozlar üçün istifadə ediləcək görüntünü, lazım ola biləcək əlavə Python kitabxanalarını və son olaraq Vertex AI-nın təlim skriptini saxlamaq üçün istifadə etməli olduğu staging direktoriyasını təyin etməlisiniz. Standart olaraq, bu həm də təlim skriptinin təlim edilmiş modeli, TensorBoard loglarını və modelin yoxlama nöqtələrini (əgər varsa) saxlayacağı yerdir. Gəlin işi yaradaq:

```
custom_training_job = aiplatform.CustomTrainingJob(
    display_name="my_custom_training_job",
    script_path="my_vertex_ai_training_task.py",
```

```
container_uri="gcr.io/cloud-aiplatform/training/tf-gpu.2-4:latest",
    model_serving_container_image_uri=server_image,
    requirements=[ "gcsfs==2022.3.0" ], # lazımlı deyil, bu sadəcə bir
nümunədir

    staging_bucket=f"gs://{bucket_name}/staging"

)
```

Və indi bunu iki worker üzərində, hər biri iki GPU ilə işə salaq:

python

Copia codice

```
mnist_model2 = custom_training_job.run(
    machine_type="n1-standard-4",
    replica_count=2,
    accelerator_type="NVIDIA_TESLA_K80",
    accelerator_count=2,
)
```

Bu qədər: Vertex AI sizin tələblərinizə uyğun hesablama nodelarını təmin edəcək (kvotalar daxilində) və təlim skriptinizi onların arasında icra edəcək. İş başa çatdıqdan sonra `run()` metodu təlim edilmiş bir model qaytaracaq ki, bu modeli əvvəllər yaratdığınız model kimi istifadə edə bilərsiniz: onu bir endpoint-ə yerləşdirə bilərsiniz və ya batch proqnozları üçün istifadə edə bilərsiniz. Təlim zamanı hər hansı bir problem yaranarsa, GCP konsolunda logları görə bilərsiniz:  naviqasiya menyusunda Vertex AI → Təlim-i seçin, təlim işinizi klikləyin və **VIEW LOGS**-a klikləyin. Alternativ olaraq, **CUSTOM JOBS** tabını seçib işin ID-sini (məsələn, 1234) kopyalaya bilərsiniz, sonra  naviqasiya menyusundan **Logging**-i seçib `resource.labels.job_id=1234` axtarışını aparın.

İpucu

Təlim prosesini vizuallaşdırmaq üçün sadəcə TensorBoard-u başladın və onun `--logdir`-ini logların GCS yoluna işarə edin. O, application default credentials-dan istifadə edəcək, hansını ki, siz `gcloud auth application-default login` istifadə edərək təyin edə bilərsiniz. Əgər istəyirsinizsə, Vertex AI ayrıca hosted TensorBoard serverləri təklif edir.

Bir neçə hiperparametr dəyərini yoxlamaq istəyirsinizsə, bir seçim çoxsaylı işləri icra etməkdir. `run()` metodunu çağırarkən `args` parametrimizi təyin edərək hiperparametr dəyərlərini skriptinizə komanda xətti arqumentləri kimi ötürə bilərsiniz, ya da onları `environment_variables` parametri ilə mühit dəyişənləri kimi ötürə bilərsiniz. Lakin buludda böyük bir hiperparametr tənzimləmə işini icra etmək istəyirsinizsə, çox daha yaxşı bir seçim Vertex AI-nın hiperparametr tənzimləmə xidmətindən istifadə etməkdir. Gəlin bunun necə olduğunu görək.

Vertex AI-də Hiperparametr Tənzimlənməsi

Vertex AI-nın hiperparametr tənzimləmə xidməti sürətli şəkildə optimal hiperparametr kombinasiyalarını tapmağa qadir olan Bayes optimallaşdırma alqoritminə əsaslanır. Ondan istifadə etmək üçün əvvəlcə komanda xətti arqumentləri kimi hiperparametr dəyərlərini qəbul edən bir təlim skripti yaratmalısınız. Məsələn, skriptiniz `argparse` standart kitabxanasından belə istifadə edə bilər:

```
import argparse

parser = argparse.ArgumentParser()

parser.add_argument("--n_hidden", type=int, default=2)

parser.add_argument("--n_neurons", type=int, default=256)

parser.add_argument("--learning_rate", type=float, default=1e-2)

parser.add_argument("--optimizer", default="adam")

args = parser.parse_args()
```

Hiperparametr tənzimləmə xidməti skriptinizi bir neçə dəfə çağıracaq, hər dəfə fərqli hiperparametr dəyərləri ilə: hər bir icra "trial" adlanır və "trial"ların cəmi isə "study" adlanır. Təlim skriptiniz verilmiş hiperparametr dəyərlərindən istifadə edərək modeli qurmalı və kompilyasiya etməlidir. Əgər hər bir "trial" çoxlu GPU olan maşında icra olunursa, mirrored distribution strategiyasından istifadə edə bilərsiniz. Daha sonra skript dataset-i yükləyib modeli təlim edə bilər. Məsələn:

```
import tensorflow as tf

def build_model(args):

    with tf.distribute.MirroredStrategy().scope():

        model = tf.keras.Sequential()

        model.add(tf.keras.layers.Flatten(input_shape=[28, 28],
                                         dtype=tf.uint8))

        for _ in range(args.n_hidden):

            model.add(tf.keras.layers.Dense(args.n_neurons,
                                         activation="relu"))

        model.add(tf.keras.layers.Dense(10, activation="softmax"))

        opt = tf.keras.optimizers.get(args.optimizer)

        opt.learning_rate = args.learning_rate

        model.compile(loss="sparse_categorical_crossentropy",
                      optimizer=opt,

                      metrics=[ "accuracy"])

    return model
```

```
[...] # dataset-i yükleyin

model = build_model(args)

history = model.fit([...])
```

İpucu

Yoxlama nöqtələrini, TensorBoard loglarını və son modeli harada saxlamağı müəyyən etmək üçün əvvəl qeyd etdiyimiz [AIP_*](#) mühit dəyişənlərindən istifadə edə bilərsiniz.

Son olaraq, skript modelin performansını Vertex AI-nın hiperparametr tənzimləmə xidmətinə geri bildirməlidir ki, hansı hiperparametrlərin növbəti dəfə yoxlanılacağını qərarlaşdırırsın. Bunun üçün siz Vertex AI təlim VM-lərində avtomatik quraşdırılmış [hypertune](#) kitabxanasından istifadə etməlisiniz:

```
import hypertune

hypertune = hypertune.HyperTune()

hypertune.report_hyperparameter_tuning_metric(
    hyperparameter_metric_tag="accuracy", # bildirilən metrikin adı
    metric_value=max(history.history["val_accuracy"]), # metrik
    dəyəri

    global_step=model.optimizer.iterations.numpy(),
)

)
```

İndi ki təlim skriptiniz hazırkı, onun hansı növ maşında icra olunacağını təyin etməlisiniz. Bunun üçün Vertex AI hər bir "trial" üçün şablon kimi istifadə edəcək xüsusi bir iş müəyyən etməlisiniz:

```
trial_job = aiplatform.CustomJob.from_local_script(
    display_name="my_search_trial_job",
    script_path="my_vertex_ai_trial.py", # təlim skriptinizin yolu

    container_uri="gcr.io/cloud-aiplatform/training/tf-gpu.2-4:latest",
    staging_bucket=f"gs://{bucket_name}/staging",
    accelerator_type="NVIDIA_TESLA_K80",
```

```
    accelerator_count=2, # bu nümunədə, hər bir "trial"da 2 GPU  
olacaq  
)  
  
Nəhayət, hiperparametr tənzimləmə işini yaratmağa və icra etməyə hazırlısanız:
```

```
python
```

```
Copia codice
```

```
from google.cloud.aiplatform import hyperparameter_tuning as hpt  
  
  
hp_job = aiplatform.HyperparameterTuningJob(  
    display_name="my_hp_search_job",  
    custom_job=trial_job,  
    metric_spec={"accuracy": "maximize"},  
    parameter_spec={  
        "learning_rate": hpt.DoubleParameterSpec(min=1e-3, max=10,  
scale="log"),  
        "n_neurons": hpt.IntegerParameterSpec(min=1, max=300,  
scale="linear"),  
        "n_hidden": hpt.IntegerParameterSpec(min=1, max=10,  
scale="linear"),  
        "optimizer": hpt.CategoricalParameterSpec(["sgd", "adam"]),  
    },  
    max_trial_count=100,  
    parallel_trial_count=20,  
)  
hp_job.run()
```

Burada biz Vertex AI-ya "accuracy" adlı metriyi maksimumlaşdırmağı bildiririk: bu ad təlim skripti tərəfindən bildirilən metrik adı ilə uyğun olmalıdır. Biz həmçinin axtarış məkanını təyin edirik, burada learning rate üçün log miqyasından, digər hiperparametrlər üçün isə xətti (yəni, bərabər) miqyasdan istifadə edirik. Hiperparametr adları təlim skriptinin komanda xətti arqumentləri ilə uyğun olmalıdır. Sonra, maksimum "trial" sayını 100, paralel işləyən maksimum "trial" sayını isə 20 olaraq təyin edirik. Əgər paralel "trial"ların sayını 60-a qədər artırısaq, ümumi axtarış vaxtı 3 dəfə azalacaq. Lakin ilk 60 "trial" paralel olaraq başlayacaq, beləliklə, onlar digər "trial"ların geri bildirimindən faydalananmayacaqlar. Buna görə də maksimum "trial" sayını kompensasiya etmək üçün artırımalısınız—məsələn, 140-a qədər.

Bu bir qədər vaxt alacaq. İş tamamlandıqdan sonra `hp_job.trials` istifadə edərək "trial" nəticələrini götürə bilərsiniz. Hər bir "trial" nəticəsi hiperparametr dəyərlərini və nəticə metriklərini ehtiva edən bir protobuf obyekti kimi təmsil olunur. Gəlin ən yaxşı "trial"ı tapaq:

```
def get_final_metric(trial, metric_id):  
  
    for metric in trial.final_measurement.metrics:  
  
        if metric.metric_id == metric_id:  
  
            return metric.value  
  
  
trials = hp_job.trials  
  
trial_accuracies = [get_final_metric(trial, "accuracy") for trial in trials]  
  
best_trial = trials[np.argmax(trial_accuracies)]
```

İndi bu "trial"ın doğruluq dəyərinə və hiperparametr dəyərlərinə baxaq:

python

```
>>> max(trial_accuracies)
```

```
0.977400004863739
```

```
>>> best_trial.id  
'98'  
  
>>> best_trial.parameters  
[parameter_id: "learning_rate" value { number_value: 0.001 },  
 parameter_id: "n_hidden" value { number_value: 8.0 },  
 parameter_id: "n_neurons" value { number_value: 216.0 },  
 parameter_id: "optimizer" value { string_value: "adam" }  
]  

```

Bu qədər! İndi bu "trial"ın SavedModel modelini əldə edə bilərsiniz, lazımlı gələrsə, onu bir az daha təlim etdirə bilərsiniz və istehsalata yerləşdirə bilərsiniz.

İpucu

Vertex AI həmçinin AutoML xidməti təklif edir ki, bu da düzgün model arxitekturasını tapmaq və onu sizin üçün təlim etmək işini tamamilə öz üzərinə götürür. Siz sadəcə dataset-i xüsusi formata uyğun olaraq Vertex AI-a yükləməlisiniz (formata dataset-in növündən asılı olaraq dəyişir: şəkillər, mətn, cədvəl, video və s.), sonra AutoML təlim işini yaradıb dataset-ə işaret edərək sərf etmək istədiyiniz maksimum hesablama saatlarını təyin etməlisiniz. Bir nümunə üçün notebook-a baxın.

Vertex AI-də Keras Tuner İstifadə Edərək Hiperparametr Tənzimlənməsi

Vertex AI-nın hiperparametr tənzimləmə xidməti əvvəzinə, Keras Tuner-dən (10-cu fəsildə təqdim edilmişdir) istifadə edə bilərsiniz və onu Vertex AI VM-lərində işə sala bilərsiniz. Keras Tuner hiperparametr axtarışını çoxsaylı maşınlar arasında bölüşdürülməklə miqyaslandırmağın sadə bir yolunu təmin edir: sadəcə olaraq hər bir maşında üç mühit dəyişənini təyin etməyiniz kifayətdir, sonra hər bir maşında adı Keras Tuner kodunuzu işlədin. Eyni skriptdən bütün maşınlarda istifadə edə bilərsiniz. Maşınlardan biri başçı (yəni, oracle) kimi çıxış edir, digərləri isə işçi kimi çıxış edir. Hər bir işçi başçıdan hansı hiperparametr dəyərlərini yoxlaması soruşur, sonra işçi bu hiperparametr dəyərlərindən istifadə edərək modeli təlim etdirir və nəhayət, o, modelin performansını başçıya geri bildirir, bu isə işçiyə hansı hiperparametr dəyərlərini növbəti dəfə yoxlamalı olduğunu qərarlaşdırır bilər.

Hər bir maşında təyin etməli olduğunuz üç mühit dəyişəni bunlardır:

- **KERASTUNER_TUNER_ID**: Başçı maşında bu "chief"ə bərabərdir, hər bir işçi maşında isə "worker0", "worker1" kimi unikal identifikatora bərabərdir.
- **KERASTUNER_ORACLE_IP**: Bu, başçı maşınının IP ünvanı və ya hostname-dır. Ümumiyyətlə, başçı özü hər IP ünvanında qulaq asmaq üçün "0.0.0.0" istifadə etməlidir.
- **KERASTUNER_ORACLE_PORT**: Bu, başçının qulaq asacağı TCP portudur.

Keras Tuner-i hər hansı bir maşın dəsti arasında bölüşdürü bilərsiniz. Əgər onu Vertex AI maşınlarında işə salmaq istəyirsinizsə, onda adı bir təlim işi yarada bilərsiniz və sadəcə olaraq təlim skriptini mühit dəyişənlərini düzgün təyin etmək üçün dəyişdirə bilərsiniz, sonra isə Keras Tuner-dən istifadə edə bilərsiniz. Nümunə üçün notebook-a baxın.

İndi öz infrastrukturunuzda və ya buludda müxtəlif yayılma strategiyalarından istifadə edərək state-of-the-art neyron şəbəkə arxitekturaları yaratmaq və onları geniş miqyasda təlim etdirmək, sonra isə onları istənilən yerdə yerləşdirmək üçün lazım olan bütün alətlərə və biliklərə sahibsiniz. Başqa sözlə, artıq sizdə super güclər var: onlardan yaxşı istifadə edin!

Tapşırıqlar

1. **SavedModel** nə ehtiva edir? Onun məzmununu necə yoxlayırsınız?
2. **TF Serving**-dən nə vaxt istifadə etməlisiniz? Onun əsas xüsusiyyətləri nələrdir? Onu yerləşdirmək üçün hansı alətlərdən istifadə edə bilərsiniz?
3. Bir neçə **TF Serving** instansiyası arasında modeli necə yerləşdirirsınız?
4. **TF Serving** tərəfindən verilən modeli sorğulamaq üçün **gRPC API**-dan nə vaxt istifadə etməlisiniz?
5. **TFLite** modelin ölçüsünü mobil və ya yerləşdirilmiş cihazda işləməsi üçün kiçitmək üçün hansı müxtəlif yolları təmin edir?
6. **Kvantizasiya-şüurlu təlim** nədir və ona niyə ehtiyacınız ola bilər?
7. **Model paralelliyi və məlumat paralelliyi** nədir? Niyə sonuncu ümumiyyətlə tövsiyə olunur?
8. Bir neçə server arasında model təlim edərkən hansı yayılma strategiyalarından istifadə edə bilərsiniz? Hansını istifadə edəcəyinizi necə seçirsiniz?
9. İstənilən bir modeli təlim etdirin və onu **TF Serving** və ya **Google Vertex AI**-ya yerləşdirin. Onu **REST API** və ya **gRPC API** istifadə edərək sorğulamaq üçün müştəri kodunu yazın. Modeli yeniləyin və yeni versiyani yerləşdirin. Müştəri kodunuz indi yeni versiyani sorğulayacaq. İlk versiyaya geri qayıdın.
10. **MirroredStrategy** istifadə edərək eyni maşında bir neçə GPU arasında hər hansı bir modeli təlim etdirin (əgər GPU-lara çıxışınız yoxdursa, **Google Colab**-da bir GPU runtime ilə iki məntiqi GPU yaradaraq bunu edə bilərsiniz). Modeli yenidən **CentralStorageStrategy** istifadə edərək təlim etdirin və təlim vaxtını müqayisə edin.
11. İstənilən bir modeli **Vertex AI**-da təkmilləşdirin, bunun üçün ya **Keras Tuner** ya da **Vertex AI**-nın hiperparametr tənzimləmə xidmətindən istifadə edin.

Bu tapşırıqların həlləri bu fəsilin notebook-unda mövcuddur: <https://homl.info/colab3>.

Random Forest - Təsadüfi Meşə

Support Vector Machine (SVM) - Dəstək Vektor Maşını

Decision Tree - Qərar Ağacı

K-Nearest Neighbors (KNN) - K-Ən Yaxın Qonşular

Linear Regression - Xətti Regressiya

Logistic Regression - Logistik Regressiya

Neural Networks - Neyron Şəbəkələr

Gradient Boosting - Gradient Artırma

AdaBoost - AdaBoost (AdaBoost metodu)

Naive Bayes - Sadə Bayes

K-Means Clustering - K-Ortalar Klasterləşməsi

Principal Component Analysis (PCA) - Əsas Komponentlərin Təhlili

XGBoost - XGBoost (XGBoost alqoritmi)

Artificial Neural Networks (ANN) - Süni Neyron Şəbəkələri

Convolutional Neural Networks (CNN) - Konvolyusiya Neyron Şəbəkələri

Recurrent Neural Networks (RNN) - Təkrar Neyron Şəbəkələri

Long Short-Term Memory (LSTM) - Uzun Qısamüddətli Yaddaş

Autoencoders - Avtomatik Kodlayıcılar

Generative Adversarial Networks (GAN) - Generativ Adversarial Şəbəkələr

Bayesian Networks - Bayes Şəbəkələri

Machine Learning Terminologies:

Supervised Learning - Nəzarətli Öyrənmə

Unsupervised Learning - Nəzarətsiz Öyrənmə

Reinforcement Learning - Gücləndirici Öyrənmə

Overfitting - Həddən artıq uyğunlaşma

Underfitting - Uyğunsuzluq

Cross-Validation - Çarpaz Doğrulama

Feature Selection - Xüsusiyyət Seçimi

Feature Engineering - Xüsusiyyət Mühəndisliyi

Hyperparameter Tuning - Hiperparametr Tənzimlənməsi

Dimensionality Reduction - Ölçü Azaldılması

Confusion Matrix - Qarışılıq Matrisi

ROC Curve - ROC Qövsü

AUC (Area Under Curve) - Əyri Altındaki Sahə

Precision - Dəqiqlik

Recall - Çağırış

F1 Score - F1 Nəticəsi

Clustering - Klasterləşmə

Classification - Təsnifat

Regression - Regressiya

Ensemble Methods - Ensambl Metodları