Name : Mohamed Ahamed Mohamed Abo El-Enen

Subject: Proteina (Machine Learning Vacancy)

Questions

1. **How many GPUs will you need to train your model for a batch size of 32 ?**
   By using this formula
   Max batch size= available GPU memory bytes / 4 / (size of tensors + trainable parameters)

   And the 4/(size of tensors + trainable parameters) is constant for model
   If we assume that the model only fit with 1 batch size as maximum and we have n infinite GPUs

   So we need 32 GPUs or more

2. **What is the specific PyTorch or TensorFlow module you will use to conduct such a type of training? You may write a maximum of 280 characters about how you will use it?**

   Use Mixed precision

   Is the use of both 16-bit and 32-bit floating-point types in a model during training to make it run faster and use less memory. By keeping certain parts of the model in the 32-bit types for numeric stability.

   Using the float32 dtype, which takes 32 bits of memory. However, there are two lower-precision dtypes, float16 and bfloat16, each which take 16 bits of memory instead. Modern accelerators can run operations faster in the 16-bit dtypes, as they have specialized hardware to run 16-bit computations and 16-bit dtypes can be read from memory faster.

   Example

```
from tensorflow.keras import mixed_precision
policy = mixed_precision.Policy('mixed_float16')
mixed_precision.set_global_policy(policy)
```

3. **You connect the required GPUs, and try to train your model on them, but you face the exact OOM error again when you go beyond a batch size of 1. When you check the code, you realize the code has access only to one GPU. Given hardware is connected OK, what do you check next ?**

   Use distribute.MirroredStrategy

   We need to make sure the code actual use the new multi GPUs Parallel or as a One set

So writing

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

    BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_sync

    with strategy.scope():
      Define and compile model
```

Here we make sure we use distribute.MirroredStrategy supports synchronous distributed training on multiple GPUs on one machine, to perform in-graph replication with *synchronous training on many GPUs on one machine*. The strategy essentially copies all of the model's variables to each processor. Then, it uses all-reduce to combine the gradients from all processors, and applies the combined value to all copies of the model.

4. **Why does the exact same code produce different results every time she runs it ?**
    a. Randomness in Data Collection
       Trained with different data, machine learning algorithms will construct different models.
    b. Randomness in Observation Order
       The order that the observations are exposed to the model affects internal decisions.
    c. Randomness in the Algorithm
       An algorithm may be initialized to a random state. Such as the initial weights in an artificial neural network.
    d. Randomness in Data augmentation
       Some Augmentation Techniques on image like rotation shearing use range of angles that can create random shapes of same image
    e. Randomness in Sampling
       Up or down resampling with Data like adding white noise to signal or using Smote.
    f. Randomness in Resampling
       Like splitting the data into a random training and test set or use k-fold cross validation that makes k random splits of the data.
    g. Randomness in External Tools
       Like sensing form outsider sensor can cause some sort of noise and randomness.

Code and Run Notes

1. To Run the code just run the main.py or mnistgan.ipython file both have same code and then select which model you want to run
2. Each model create a its own folder for result the loss.csv and images in plot folder for each step prediction result

3. The config.py is the configuration for models like changing the neurons in each hidden layer, learning rate, number of epochs….etc.
4. The models build using tensorflow.v1.compat and disabled all support from v2 but in requirement it say tensorflow version 2.3
5. Linear models kept same as given example without adding any more layers than 2 and 16 neurons on each
6. The CNN models changing a little bit as given example too
7. Models inherit from a main one call GanModel that the parent for train, predict
8. Shortage of my nvidia gpu as (an AMD user) trained the model on small number of epochs not like in examples 10K just 100 with batch_size 256.
9. There are 2 ipython, one as using keras platform just like prototype for my work and some code help me answering the previous questions, second used to build the current model
10. The result not high so much because no nonlinear activation function in model 1,2 and require more hidden, neurons and more epochs, model 3, 4 work fine on same number of epochs.
11. Some tuning can like learning rate and decay + checkpoint + early stopping and previous questions solutions could help here for reduce memory and improve result but stuck as much as to the example template.