Playground for Caffe2 Models

Playground is created to allow modelers to reuse the components of their models. It is based on data parallel model of Caffe2. Playground provide a framework that takes care of regular trainer iteration procedures and abstracting out APIs that allows user to apply customized model components of their own. User can swap / exchange /reuse these components without rewriting the whole training script. Once the components are in place, user can use parameterized launch command to drive their experiments. This will be convenient for creating large amount of experiments with different components defined for each of them. It may be used as a tool to explore different model architect / algorithms like optimizer / momentum / learning rate / batch normalization parameters.

Playground project highlight:

1. parameter driven: no need to create py script for each experiment, just swap components using parameters. Very customizable, add your own component and add your opts in the command as you want.
2. All models follows a typical way of train/testing epoch iteration. Many aspects can be customized, example: run epoch by loss instead of predetermined iteration
3. customizable components, trained metrics, also specified with parameters
4. gpu or cpu training supported
5. parallel training on multiple host supported
6. checkpoint, pre-trained model helps with recover interrupted / failed experiment

Example Usage

Playground comes with a resnet example, located in resnetdemo folder. To see how playground works, do the following:

1. make sure your caffe2 build successful with openCV and lmdb dependencies supported.
2. make sure you have training/testing datasets ready in folders that can be accessible to trainer / distributed trainers
3. specify a folder that you would like to store your checkpoint model files.
4. use this command to launch a training, verify epochs are running with metrics reported in log and model file store in your checkpoint folder

$ python caffe2/contrib/playground/AnyExpOnTerm.py --parameters-json '{ "opts":{

```
"input":{
    "input_name_py":"gfs_IN1k",
    "train_input_path":"/path/to/your/training/data_lmdb/",
    "test_input_path":"/path/to/your/testing/data_lmdb/",
    "scale_jitter_type": 1, "color_jitter": true,      "color_lighting": true,
    "namespace": "aml",  "table": "imagenet_data",  "column_handle":
"everstore_handle",
    "column_label": "label", "column_id": "image_id",  "label_type": 0,
    "train_partition": {"ds": "2017-07-31", "config": "imagenet1k", "is_train": "1"},
    "test_partition": {"ds": "2017-07-31", "config": "imagenet1k", "is_train": "0"},
    "num_classes":1000, "loadsize" : 256, "imsize": 224, "decode_threads": 8,
"datasets":[]},

"model":{
    "model_name_py":"IN1k_resnet",
```

```
    "forward_pass_py":"caffe2_resnet50_default_forward",
    "parameter_update_py":"explicit_resnet_param_update",
    "optimizer_py":"",
    "rendezvous_py":"rendezvous_filestore",
    "additional_override_py":""},

"model_param":{
    "pretrained_model":"", "reset_epoch":true, "memonger" : true, "cuda_nccl": true,
    "combine_spatial_bn":true, "max_concurrent_distributed_ops" : 16,
    "base_learning_rate":0.05, "bn_epsilon":0.00001, "bn_momentum":0.9,
"custom_bn_init": true,
    "bn_init_gamma":1e-323, "weight_decay":1e-4, "weight_decay_bn":1e-323,
"engine":"CUDNN"},

"epoch_iter":{
    "num_train_sample_per_epoch":10240,
    "num_test_sample": 5000,
    "num_epochs":10,
    "num_epochs_per_flow_schedule":5,
    "num_train_iteration_per_test": 10,
    "batch_per_device":32,
    "num_test_iter":2},

"distributed":{
    "num_shards":1,
    "num_gpus":2,
    "first_gpu_id":0,
    "num_cpus":4,
    "first_cpu_id":0},

"output":{
    "gen_output_py":"output_generator",
    "gen_checkpoint_path_py":"gen_checkpoint_path",
    "checkpoint_folder":"/home/your_user_name/model_checkpoint/",
    "metrics":[
        {"name":"train_loss",
         "meter_py":"ComputeLoss",
         "meter_kargs":{"blob_name":"loss"},
         "is_train":true},
        {"name":"test_loss",
         "meter_py":"ComputeLoss",
         "meter_kargs":{"blob_name":"loss"},
         "is_train":false},
        {"name":"train_accuracy_top1",
         "meter_py":"ComputeTopKAccuracy",
         "meter_kargs":{"blob_name":["softmax", "label"], "topk":1},
         "is_train":true},
        {"name":"train_accuracy_top5",
         "meter_py":"ComputeTopKAccuracy",
         "meter_kargs":{"blob_name":["softmax", "label"], "topk":5},
         "is_train":true},
        {"name":"test_accuracy_top1",
```

```
         "meter_py":"ComputeTopKAccuracy",
         "meter_kargs":{"blob_name":["softmax", "label"], "topk":1},
         "is_train":false},
        {"name":"test_accuracy_top5",
         "meter_py":"ComputeTopKAccuracy",
         "meter_kargs":{"blob_name":["softmax", "label"], "topk":5},
         "is_train":false}],
    "plots":[
        {"x":"", "x_title":"", "ys":["train_loss", "test_loss"],
         "y_title":"train and test loss"},
        {"x":"epochs", "x_title":"epochs",
         "ys":["train_accuracy_top1","test_accuracy_top1",
               "train_accuracy_top5","test_accuracy_top5"],
         "y_title":"Accuracy: Train top1, Test top1, Train top5, Test top5"}]}}
```

}'

5. now you can switch to different components that supplied in resnetdemo folder like so:

   "forward_pass_py":"caffe2_resnet50_default_forward", --> "explicit_resnet_forward" (which is a resnet model that allow you specify layers with "model_param"."num_layer")

   and/or

   "parameter_update_py":"caffe2_resnet50_default_param_update", --> "explicit_resnet_param_update"

   playground should be able to launch training epochs and give you results

General Usage Guideline

1. mandatory non empty opts: input_name_py, datasets, model_name_py, forward_pass_py, (parameter_update_py or optimizer_py), rendezvous_py, memonger, all epoch_iter opts, all distributed opts, gen_output_py
2. mandatory nullable opts: pretrained_model, max_concurrent_distributed_ops, combine_spatial_bn
3. other module dependent opts can be changed or removed: the rest of the opts.
4. specify any additional opts depends on your modules' need, directly add them into the command line opts dictionary and no need to change any py code. You should create your module to make sure they knows how to handle these new opts. You access your own opts in such a manner: self.opt['your_own_arg'] ['your_own_sub_arg']
5. checkpoint is performed at the end of each epoch by default and generated model file can be find in log. Each checkpoint can be used as pre-trained model to start new experiment. Make sure new experiment is compatible with pre-trained model if you specified it. For example, gpu experiment and cpu experiment can not share checkpoint, because the blob names are different. Any experiments with different blob names can not share checkpoint.
6. The metric and plots are reported when experiment finish running. Intermediate results are reported in the log of the CreateTrainerAndRunManyEpochs operator as iteration goes on.
7. if num_gpus is specified, the trainer will try to use gpu. if num_gpus = 0, the trainer will use cpu to train. For gpu training, batch_per_device are typically 32, for cpu training, batch_per_device is normally set to 2 with num_cpus higher like 8 or 16 depends on your machines' configuration.

8. if train on single host, let "num_shards" = 1, if multiple hosts, specify your "num_shards" and start parallelized training from each shards similarly to the resnet50_trainer.py example in caffe2/python/examples/ folder.

Develop Your Own Components

1. Create a folder for your own experiment under caffe2/contrib/playground/ and go to this folder.

2. Create a base model file, for example IN1kResnet.py. In this script you need to implement init function and in it, instantiate your train/test model and give them to self.train_model and self.test_model. In this base model class, you can also chose to override other functions you'd like to customize, for example if you want to iterate according to accuracy instead of fixed number of loops, override list_of_epochs(), and list_of_epoch_iters()

3. Create component py scripts implementing the generators arguments of data_parallel_model.Parallelize(). Total four of them: input_builder_fun, forward_pass_builder_fun, one of param_update_builder_fun or optimizer_builder_fun, and rendezvous. This is where you can switch between different components. Examples: for the demo IN1k_resnet experiments, I created two different forward function: explicit_resnet_forward.py and caffe2_resnet50_default_forward.py. Both implemented the API "gen_param_update_builder_fun", which is abstract method in the framework class AnyExp.py

4. Next import the module components you created into module_map.py. This import is needed to include these packages during building. Give imported module a module name, normally if module is just a simple file contains some functions, just use the py script file name. If the module contains class and the class is needed for module input, name it with the class name, examples are the meter classes like compute_loss. When launching your experiment, in opts for the term “xxx_py” fill in the name you chose in module_map.py. Playground will find your module and load it.

5. Create as many modules as you need. Then when you perform your experiment, specify the module you want in opts correspondingly and you can run your experiment with ease.

6. In the demo, the opts item “gen_output_py” uses output_generator.py , which provides a minimum way to generating final experimental result, stored in the form of a dict. It will allow user to do whatever visualization with these data after the training is finished.

7. Customize your experimental result. A meter interface is provided to implement your own metrics calculators. Example compute_loss.py and compute_topk_accuracy.py. For training metrics, results are calculated right away in each iteration. For testing metrics, results are accumulated for the whole loop and finally calculated after test iteration finishes. Once your have your meter class defined, you can start defining what metrics to report in your opts['output']['metrics'] list. The name you give to your metrics can later be used when you define your plots. The Playground will always record throughput metrics secs_per_train and samples_per_sec.

8. an additional_override_py option is provided for the modules to allow user override any existing methods defined in the main framework AnyExp.py. This make it easy to shut down part of the model to focus on remaining modules for experimenting or debugging. An example is given as override_no_test_model_no_checkpoint.py, which turns off checkpointing and does neither prepare nor run test model.