Knowledge Transfer

Neural Network Distributed Training and Optimization Library (NNLO)

The 11th Conference of the Balkan Physical Union (BPU11 Congress), 28.08-01.09.2022, Belgrade, Serbia

Irena Veljanović[1], Maurizio Pierini[1], Jean-Roch Vlimant[2], Mahmoud Shadi Atari[1]

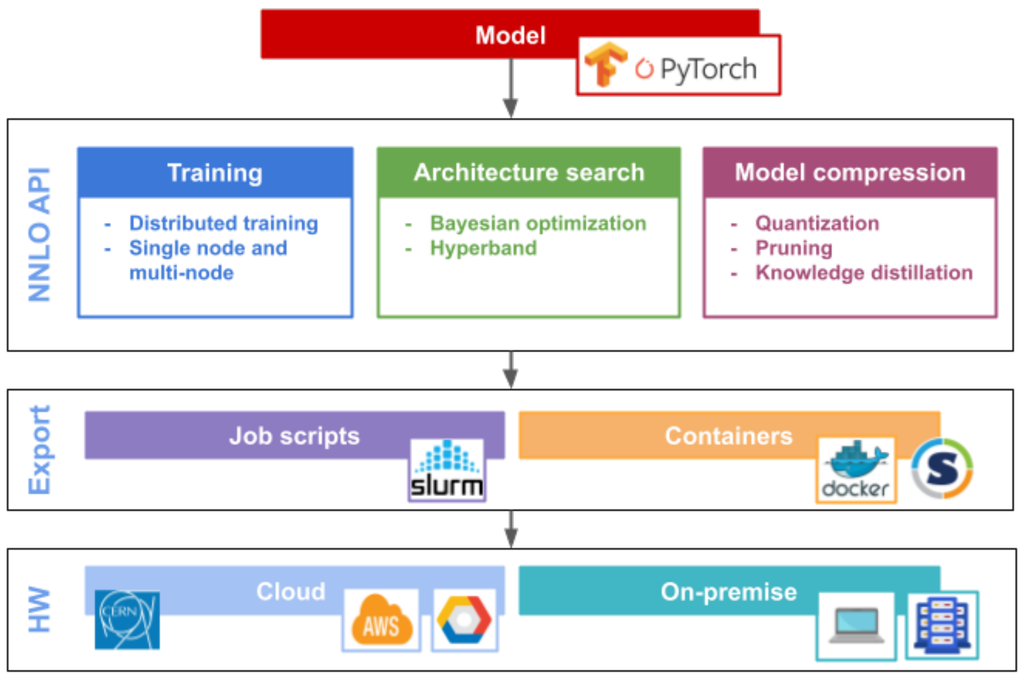
[1]European Organization for Nuclear Research (CERN), [2]California Institute of Technology (Caltech)

Motivation

With deep learning becoming very popular among Large Hadron Collider (LHC) experiments, it is expected that speeding up the neural network training and optimization will soon be an issue. To this purpose, we are developing a dedicated tool at the Compact Muon Solenoid (CMS) collaboration, namely, the Neural Network Learning and Optimization library (NNLO). NNLO aims to support both widely known deep learning frameworks Tensorflow and PyTorch. It should help engineers and scientists to easier scale neural network training and hyperparameter optimization. Compared to manual distributed training, NNLO facilitates the transition from a single to multiple GPUs without losing performance.

Architecture of NNLO

Shown below is the high-level software architecture of the NNLO. Some of the features are already developed and tested, some of them are in the development phase. The idea is to support distributed training using one or more nodes with one or more GPUs each, using one of the two frameworks TensorFlow or PyTorch. As well as an hyperparameter optimization and model compression.



Training API for NNLO

The training API is primarily built on 3 main classes, namely, the distribution strategy class which defines all the details of the distribution such as number of nodes, and number of GPUs. Then comes our driver class which carries all the specifications we need to start training, including the distribution strategy. Finally, the runner class which is completely responsible for running the training as specified by the driver and the distribution strategy.

Distribution Strategy

Properties	Methods
nr_gpus nr_nodes rank master_ip master_port	get_nr_gpus get_nr_nodes get_rank get_master_ip get_master_port

Training Driver

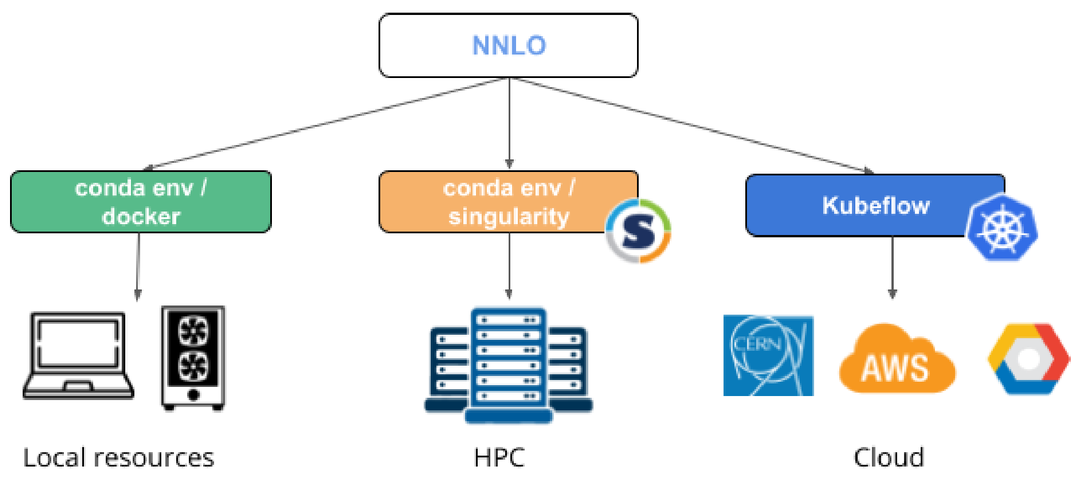
Properties	Methods
project_name model_builder train_data val_data batch_size loss optimizer metrics callbacks dist_strategy learning_rate	get_project_name get_mode get_train_data get_val_data get_batch_size get_loss get_optimizer get_metrics get_callbacks get_dist_strategy get_learning_rate set_data

Training Runner

Properties	Methods
driver epochs save_path	run_single run_distributed

Usability across different platforms

One of our main pillars to define the success of our library is scalability and compatibility. Consequently, as part of developing NNLO, we made sure to support different working environments for practitioners, currently, NNLO can be used on local machines, and on machines accessible through SSH. Moreover, we are developing exporters which will help users to easier switch from local resources to cloud or HPC sites.



NNLO makes training code more compact

How the code looks like without using NNLO:

```
batch_size = 32
epochs = 3

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

y_train = y_train / 127.5 - 1.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(batch_size)

y_test = y_test / 127.5 - 1.0
y_test = tf.keras.utils.to_categorical(y_test, 10)
validation_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)

strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    model = MobileNetV2(input_shape=(32, 32, 3), alpha=1.0, weights=None, classes=10)

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=None)

    model.fit(train_dataset,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=validation_dataset,
              callbacks=None)
```

How the code gets simplified when using NNLO:

```
def model_builder():
    return MobileNetV2(input_shape=(32, 32, 3), alpha=1.0, weights=None, classes=10)

data_generator = Cifar10InMemoryDataSetGenerator(32)

driver = TensorflowTrainingDriver('nnlo_mobile_net',
                                  model_builder,
                                  data_generator,
                                  loss='categorical_crossentropy',
                                  optimizer='adam',
                                  dist_trategy=SingleNodeStrategy())

runner = TensorflowTrainingRunner(driver)
runner.run()
```

Future work

Two primary goals we want to achieve in the future are to extend the NNLO library to support hyperparameter optimization and model compression. The idea is to integrate some of the well-known hyperparameter optimization frameworks, in order to use them seamlessly alongside NNLO library. Model compression can be achieved through different methods, and the future work on the library will be focused on some of them: quantization, pruning, knowledge distillation.

References

1. TensorFlow, https://www.tensorflow.org/
2. PyTorch, https://pytorch.org/
3. Slurm workload manager, https://slurm.schedmd.com/