Inference PyTorch Bert Model for High Performance in ONNX Runtime

In this tutorial, you'll be introduced to how to load a Bert model from PyTorch, convert it to ONNX, and inference it for high performance using ONNX Runtime with transformer optimization. In the following sections, we are going to use the Bert model trained with Stanford Question Answering Dataset (SQuAD) dataset as an example. Bert SQuAD model is used in question answering scenarios, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.

0. Prerequisites

First you need to check if the following packages exist and install them if needed.

In [1]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install wget               # used to download data files
# !{sys.executable} -m pip install --user torch==1.3.1 torchvision==0.4.2+cpu -f https://download.pytorch.org/wh
# !{sys.executable} -m pip install transformers       # used to load pytorch bert model
```

```
Requirement already satisfied: wget in /home/dn/anaconda3/envs/nlp_onnx/lib/python3.7/site-packages (3.2)
```

1. Load Pretrained Bert model

We begin by downloading the data files and store them in the specified location.

In [2]:
```
import os

# Create a directory to store predict file
output_dir = "./pytorch_output"
cache_dir = "./pytorch_squad"
predict_file = os.path.join(cache_dir, "dev-v1.1.json")
# create cache dir
if not os.path.exists(cache_dir):
    os.makedirs(cache_dir)

# Download the file
predict_file_url = "https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v1.1.json"
if not os.path.exists(predict_file):
    import wget
    print("Start downloading predict file.")
    wget.download(predict_file_url, predict_file)
    print("Predict file downloaded.")
```

Specify some model config variables.

In [3]:
```python
# Define some variables
model_type = "bert"
model_name_or_path = "bert-base-cased"
max_seq_length = 384
doc_stride = 128
max_query_length = 64
per_gpu_eval_batch_size = 48
eval_batch_size = 48
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
! nvidia-smi
device
```

```
Tue Mar 10 01:34:18 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.64       Driver Version: 440.64       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   25C    P5    23W / 250W |     10MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   33C    P5    13W / 250W |     10MiB / 11178MiB |      2%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Out[3]:
```
device(type='cuda')
```

Start to load model from pretrained. This step could take a few minutes.

In [4]:
```
# The following code is adapted from HuggingFace transformers
# https://github.com/huggingface/transformers/blob/master/examples/run_squad.py#L290

from transformers import (WEIGHTS_NAME, BertConfig, BertForQuestionAnswering, BertTokenizer)
from torch.utils.data import (DataLoader, SequentialSampler)

# Load pretrained model and tokenizer
config_class, model_class, tokenizer_class = (BertConfig, BertForQuestionAnswering, BertTokenizer)
config = config_class.from_pretrained(model_name_or_path, cache_dir=cache_dir)
tokenizer = tokenizer_class.from_pretrained(model_name_or_path, do_lower_case=True, cache_dir=cache_dir)
model = model_class.from_pretrained(model_name_or_path,
                                    from_tf=False,
                                    config=config,
                                    cache_dir=cache_dir)
# load_and_cache_examples
from transformers.data.processors.squad import SquadV2Processor

processor = SquadV2Processor()
examples = processor.get_dev_examples(None, filename=predict_file)

from transformers import squad_convert_examples_to_features
features, dataset = squad_convert_examples_to_features(
            examples=examples,
            tokenizer=tokenizer,
            max_seq_length=max_seq_length,
            doc_stride=doc_stride,
            max_query_length=max_query_length,
            is_training=False,
            return_dataset='pt'
        )

cached_features_file = os.path.join(cache_dir, 'cached_{}_{}_{}'.format(
        'dev',
        list(filter(None, model_name_or_path.split('/'))).pop(),
        str(384))
    )

torch.save({"features": features, "dataset": dataset}, cached_features_file)
print("Saved features into cached file ", cached_features_file)

# create output dir
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# n_gpu = torch.cuda.device_count()
# eval_batch_size = 8 * max(1, n_gpu)

eval_sampler = SequentialSampler(dataset)
```

```
eval_dataloader = DataLoader(dataset, sampler=eval_sampler, batch_size=eval_batch_size)

# multi-gpu evaluate
# if n_gpu > 1 and not isinstance(model, torch.nn.DataParallel):
#     model = torch.nn.DataParallel(model)
# device = torch.device("cpu")
model.to(device)
```

```
Saved features into cached file  ./pytorch_squad/cached_dev_bert-base-cased_384
```

2. Export the loaded model

Once the model is loaded, we can export the loaded PyTorch model to ONNX.

In [5]:
```
# Eval!
print("***** Running evaluation {} *****")
print("  Num examples = ", len(dataset))
print("  Batch size = ", eval_batch_size)

output_model_path = './pytorch_squad/bert-base-cased-squad.onnx'
inputs = {}
outputs= {}
# Get the first batch of data to run the model and export it to ONNX
batch = dataset[0]

# Set model to inference mode, which is required before exporting the model because some operators behave differe
# inference and training mode.
model.eval()
batch = tuple(t.to(device) for t in batch)
inputs = {
    'input_ids':      batch[0].reshape(1, max_seq_length),              # using batch size = 1 here, adjust as ne
    'attention_mask': batch[1].reshape(1, max_seq_length),
    'token_type_ids': batch[2].reshape(1, max_seq_length)
}

with torch.no_grad():
    symbolic_names = {0: 'batch_size', 1: 'max_seq_len'}
    torch.onnx.export(model,                                            # model being run
                      (inputs['input_ids'],                             # model input (or a tuple for multiple i
                       inputs['attention_mask'],
                       inputs['token_type_ids']),
                      output_model_path,                                # where to save the model (can be a file
                      opset_version=11,                                 # the ONNX version to export the model t
                      do_constant_folding=True,                         # whether to execute constant folding fo
                      input_names=['input_ids',                         # the model's input names
                                   'input_mask',
                                   'segment_ids'],
                      output_names=['start', 'end'],                    # the model's output names
                      dynamic_axes={'input_ids': symbolic_names,        # variable length axes
                                    'input_mask' : symbolic_names,
                                    'segment_ids' : symbolic_names,
                                    'start' : symbolic_names,
                                    'end' : symbolic_names})
    print("Model exported at ", output_model_path)
```

```
***** Running evaluation {} *****
  Num examples =  10970
  Batch size =  48
Model exported at  ./pytorch_squad/bert-base-cased-squad.onnx
```

3. Inference the Exported Model with ONNX Runtime

Install ONNX Runtime

Install ONNX Runtime if you haven't done so already.

Install `onnxruntime` to use CPU features, or `onnxruntime-gpu` to use GPU.

In [6]:
```
if torch.cuda.is_available():       # Install onnxruntime-gpu if cuda is available
    ONNXRUNTIME = 'onnxruntime-gpu'
else:
    ONNXRUNTIME = 'onnxruntime'     # Install ONNX Runtime

import sys
!{sys.executable} -m pip install -U $ONNXRUNTIME
```

```
Requirement already up-to-date: onnxruntime-gpu in /home/dn/anaconda3/envs/nlp_onnx/lib/python3.7/site-packages
(1.1.2)
```

Now we are ready to inference the model with ONNX Runtime

In [7]:
```
import onnxruntime as rt
import time

sess_options = rt.SessionOptions()

# Set graph optimization level to ORT_ENABLE_EXTENDED to enable bert optimization.
sess_options.graph_optimization_level = rt.GraphOptimizationLevel.ORT_ENABLE_EXTENDED

# To enable model serialization and store the optimized graph to desired location.
sess_options.optimized_model_filepath = os.path.join(output_dir, "optimized_model.onnx")
session = rt.InferenceSession(output_model_path, sess_options)

# evaluate the model
start = time.time()
res = session.run(None, {
          'input_ids': inputs['input_ids'].cpu().numpy(),
          'input_mask': inputs['attention_mask'].cpu().numpy(),
          'segment_ids': inputs['token_type_ids'].cpu().numpy()
        })
end = time.time()
print("ONNX Runtime inference time: ", round(end - start, 4), '\n')
! nvidia-smi
rt.get_device()
```

```
ONNX Runtime inference time:  0.0176

Tue Mar 10 01:35:40 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.64       Driver Version: 440.64       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   26C    P2    70W / 250W |   2535MiB / 11178MiB |     50%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   33C    P8     7W / 250W |     10MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     20234      C   ...e/dn/anaconda3/envs/nlp_onnx/bin/python  2525MiB |
+-----------------------------------------------------------------------------+
```

Out[7]: 'GPU'

Get perf numbers from the original PyTorch model.

In [8]:
```
start = time.time()
outputs = model(**inputs)
end = time.time()
print("PyTorch Inference time = ", round(end - start, 4))

print("***** Verifying correctness *****")
import numpy as np
for i in range(2):
    print('PyTorch and ORT matching numbers:', np.allclose(res[i], outputs[i].cpu().detach().numpy(), rtol=1e-04
print()
! nvidia-smi
```

```
PyTorch Inference time =  0.0266
***** Verifying correctness *****
PyTorch and ORT matching numbers: True
PyTorch and ORT matching numbers: True

Tue Mar 10 01:35:40 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.64       Driver Version: 440.64       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   27C    P2    70W / 250W |   2535MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   33C    P8     8W / 250W |     10MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     20234      C   ...e/dn/anaconda3/envs/nlp_onnx/bin/python  2525MiB |
+-----------------------------------------------------------------------------+
```

In [ ]: