Breakthrough optimizations for transformer inference on GPU and CPU

Emma Ning | Senior product manager, Microsoft
Using cutting-edge NLP techniques like transformers to better understand user queries, webpages, and other documents.
Transformer for natural language processing

- Transformers - breakthrough in natural language understanding
- BERT - Bidirectional Encoder Representations from Transformers
  - Architecture (L: layers (transformer block), H: hidden size, A: self-attention head)
    - BERT base: L=12, H=768, A=12, Parameters=110M
    - BERT large: L=24, H=1024, A=16, Parameters=340M
  - Running a 12- or 24-layer BERT for every query real-time is prohibitively expensive
Problem - Reimplement the model was time-consuming

BERT optimizations in Bing

Finetuned BERT-Base, a 12-layer model

- Significantly improves Precision and Coverage

Leveraged knowledge distillation to create a 3-layered BERT

- No any significant loss in accuracy
- Reduced inference cost significantly
- Still benchmarked at 77ms serving latency on CPU

Re-implemented the model using TensorRT C++ APIs

- Take full advantage of NVIDIA GPU architecture
- 800x throughput improvement on GPU
Transformer inference acceleration with ONNX Runtime
BERT Optimization Opportunity

• Model
  • Too many elementary operators
  • Multi transformer cells

• Kernels in ONNX Runtime
  • Not fully utilize hardware characteristic
    • CPU cores
    • Tensor-Core
BERT optimization in ONNX Runtime

- Graph optimization
- Hardware-based kernel optimization
ONNX Runtime – Graph Optimization

GraphTransformer
- An interface created for finding patterns (with specific nodes) and applying rewriting rules against a sub-graph.
- An interface created for applying graph transformation with full graph editing capability.

Graph Optimization Level
**Basic:** General transformers not specific to any specific execution provider (e.g. drop out elimination)
**Extended:** Execution provider specific transformers
**Layout Optimizations:** change the data layout for applicable nodes (NCHW layout to NHWC layout)
BERT Encoder Block

- Positional embeddings
- Multi-headed self-attention
- Feed-forward layers
- Layer norm and residuals
Embedding and Positional Encoding fusion
Multi-headed self-attention Fusion
Gelu Fusion

\[ \text{GELU}(x) := x \mathbb{P}(X \leq x) = x \Phi(x) = 0.5x \left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \]
Skip Layer Normalization Fusion
BERT Graph Optimizations

**Basic Level**
- Constant Folding
- Reshape Fusion

**Extended Level**
- GELU Fusion
- Layer Normalization Fusion
- BERT Embedding Layer Fusion
- Attention Fusion
- Skip Layer Normalization Fusion
- Bias GELU Fusion
- Fast GELU Fusion
After graph optimization
BERT optimization in ONNX Runtime

- Graph optimization
  - Hardware-based kernel optimization
    - Optimized CPU and CUDA kernels in ONNX Runtime
      - Take full advantage of the GPU architecture
      - In self-attention layer CPU implementation
        - Increase the parallelization and fully leverage available CPU cores
        - Leverage GEMM to further reduce the computation cost
Bing’s 3-layer BERT with 128 sequence length

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Inference on</th>
<th>Throughput (Query per second)</th>
<th>Latency (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original 3-layer BERT</td>
<td>1</td>
<td>Azure Standard F16s_v2 (CPU)</td>
<td>6</td>
</tr>
<tr>
<td>ONNX Model</td>
<td>1</td>
<td>Azure Standard F16s_v2 (CPU) with ONNX Runtime</td>
<td>111</td>
</tr>
<tr>
<td>GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original 3-layer BERT</td>
<td>4</td>
<td>Azure NV6 GPU VM</td>
<td>200</td>
</tr>
<tr>
<td>ONNX Model</td>
<td>4</td>
<td>Azure NV6 GPU VM with ONNX Runtime</td>
<td>500</td>
</tr>
<tr>
<td>ONNX Model</td>
<td>64</td>
<td>Azure NC6S_v3 GPU VM with ONNX Runtime + System Optimization (Tensor Core with mixed precision, Same Accuracy)</td>
<td>10667</td>
</tr>
</tbody>
</table>

On NVIDIA V100 GPUs we saw ~10,000 queries per second throughput

Development time for new BERT scenarios was cut from multiple days to a few hours
ONNX Runtime powered BERT inference in office

Keypoints model

- 3-layer BERT
- The P50 latency reduced by 3x over the original traditional ML based solution
- The development cost was significantly reduced

“At a glance” in In OneDrive and Sharepoint

Key points In Word
ONNX Runtime to power BERT inference

- Bing Ranking: 3-layer BERT
- Key Point in Office: 3-layer BERT
- Bing Ads: 3-layer BERT
- Hugging Face: 12-layer BERT
- Text Analytics in Azure AI: 12-layer BERT
- Bing Feeds: 12-layer BERT
- Speech & Language in Azure Cognitive service: 2-layer BERT
- Questions suggestions in Bing: 24-layer BERT
- ...
Model operationalization with ONNX

Train models with various frameworks or services

Convert into ONNX with ONNX Converters

HW accelerated inference with ONNX Runtime

Frameworks
- PyTorch
- Caffe2
- Keras
- Cognitive Toolkit
- MathWorks
- PaddlePaddle
- XGBoost
- MXNet
- Chainer

Services
- Azure Custom Vision Service
- Auto Machine Learning Service

Azure
- Azure Machine Learning service
- Ubuntu VM
- Windows Server 2019 VM

Devices
- Edge Cloud & Appliances
- Edge & IoT Devices
1. Load Pretrained BERT model

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

```python
import os
cache_dir = "./squad"
if not os.path.exists(cache_dir):
    os.makedirs(cache_dir)
predict_file_url = "https://rajpurkar.github.io/SQuAD-v1.1/dataset/train-v1.1.json.tar.gz"
predict_file = os.path.join(cache_dir, "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 configuration variables and constants.

```python
# For fine tuned large model, the model name is
# e.g. bert-base for base-Pretrained model
model_name_or_path = "bert-base-cased"
max_seq_length = 128
doc_stride = 128
max_query_length = 64
# Enable overwrite to export onnx model and save output
enable_overwrite = True
# Total samples to inference. It shall be large enough.
total_samples = 100
```

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

```python
The following code is adapted from HuggingFace's transformers:
https://github.com/huggingface/transformers/blob/main/src/transformers/
```

```python
import torch, transformers
from transformers import (BartConfig, BartForQuestionAnswering)

# Load pretrained model and tokenizer
config_class, model_class, tokenizer_class = (BartConfig, model_class, tokenizer_class)
config = config_class.from_pretrained(model_name_or_path)
tokenizer = tokenizer_class.from_pretrained(model_name_or_path)
model = model_class.from_pretrained(model_name_or_path, from_tf=True)

# Load some examples
from transformers.data.processors.squad import SquadV2Processor
processor = SquadV2Processor()
examples = processor.get_dev_examples("es_10k")
from transformers import squad_convert_examples_to_features
features = squad_convert_examples_to_features(examples=examples, total_samples=total_samples, tokenizer=tokenizer,
max_seq_length=max_seq_length,
doc_stride=doc_stride,
max_query_length=max_query_length,
is_training=False,
return_dataset=False)
```

2. Export the loaded model

Once the model is loaded, we can export:

```python
output_dir = "./onnx"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
model_path = os.path.join(output_dir, export_model_file_path)
```

```python
import onnx
```

```python
# example
```

```python
# Get the first example data to export
inputs = {
    "input_ids": data[0],
    "attention_mask": data[1],
    "token_type_ids": data[2],
}
```

```python
import torch
device = torch.device(
"cpu")
```

```python
# Set model to inference mode, differently from training mode.
model.eval()
```

```python
# Enable overwrite on or not
if enable_overwrite or not os.path.exists(output_dir):
    model.to(device)
```

```python
import onnx
```

```python
# Use onnxruntime to infer and export:
```

4. Inference ONNX Model with ONNX Runtime

OpenMP environment variables are very important for CPU inference of BERT model. It has large performance impact on BERT model so you might need set i carefully according to Performance Tuning Guide! In this part of the notebook.

Getting environment variables shall be done before importing onnxruntime. Otherwise, they might not take effect.

```python
import onnxruntime
```

```python
# ATTENTION: these environment variables must be set before importing onnxmlruntime.
if use_openmp:
    os.environ["OMP_NUM_THREADS"] = int(os.pct1.cpu_count(logical=True))
else:
    os.environ["OMP_NUM_THREADS"] = 1
```

```python
model = onnxruntime.InferenceSession(export_model_path, sess_options, providers=["OpenMPExecutionProvider")
```

```python
# Specify providers when you use onnxruntime-gpu for GPU inference
# onnxruntime.InferenceSession(export_model_path, sess_options, providers=["CUDAExecutionProvider")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Model exported at \nModel exported at ./onnx/bart-base")
```

```python
print("Inference time:")
```

```python
latency = []
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Model exported at ./onnx/bart-base")
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```

```python
print("Inference time:")
```

```python
for i in range(total_samples):
data = dataset[i]
```

```python
# Use onnxruntime as input
```
ONNX Runtime Adoption

- **Up to 18x** performance gains seen by Microsoft services
- **10+** platforms integrated with ONNX Runtime
- **Millions** of devices running ONNX Runtime
- **Billions** of requests handled in prod

Logos of various companies and technologies such as SQL Server, ORACLE, Visual Studio Code, PowerApps, CERN, AXIOMA, Office 365, Power BI, MarkLogic, Windows, Bing, QAS, Skype, Azure Kinect DK, and Deep Learning Inference Service (DLIS).
Thanks

ONNX  ONNX Runtime  BERT acceleration in ONNX Runtime