TensorRT 설치

TensorRT 환경을 설정하는 과정은 여러 단계를 포함하며, 주요 요구 사항과 시스템 사양에 따라 조금씩 달라질 수 있습니다. 아래에 일반적인 TensorRT 환경 설정 방법을 단계별로 정리했습니다.

---

1. 사전 요구 사항

운영 체제

- TensorRT는 주로 **Ubuntu** 및 **Windows**에서 사용 가능합니다. Ubuntu 18.04 또는 20.04를 권장합니다.

GPU 드라이버

- NVIDIA GPU가 필요하며, 최신 GPU 드라이버를 설치해야 합니다.
- GPU 드라이버 버전 확인:

```
nvidia-smi
```

- NVIDIA 드라이버 다운로드

CUDA 및 cuDNN

- TensorRT는 CUDA와 cuDNN이 필요합니다. CUDA와 cuDNN을 먼저 설치해야 합니다.
- CUDA Toolkit 다운로드
- cuDNN 다운로드

---

2. TensorRT 설치

방법 1: NVIDIA Developer Zone에서 다운로드

1. TensorRT를 NVIDIA Developer Zone에서 다운로드합니다.
   - CUDA와 호환되는 TensorRT 버전을 선택하세요.
   - `.deb` 또는 `.tar` 파일 형식으로 다운로드할 수 있습니다.
2. **Ubuntu**에서 `.deb` 파일 설치:

```
sudo dpkg -i nv-tensorrt-repo-<version>-cuda-<cuda-version>.deb
sudo apt-key add /var/nv-tensorrt-repo-<version>/7fa2af80.pub
sudo apt-get update
sudo apt-get install tensorrt
```

   추가적으로 Python 바인딩 설치:

```
sudo apt-get install python3-libnvinfer-dev
```

3. **Windows**에서 설치:
   - 설치 파일을 실행하여 지시에 따라 설치합니다.
   - 설치 후 TensorRT 라이브러리 경로를 `PATH` 환경 변수에 추가하세요.

방법 2: Python 패키지로 설치

TensorRT의 일부 Python API는 PyPI를 통해 제공됩니다.

```
pip install nvidia-pyindex
pip install nvidia-tensorrt
```

---

3. 환경 변수 설정

TensorRT 라이브러리와 실행 파일 경로를 환경 변수에 추가해야 합니다.

Ubuntu

`~/.bashrc` 파일에 다음을 추가:

```
export PATH=/usr/local/TensorRT/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/TensorRT/lib:$LD_LIBRARY_PATH
```

변경 사항 적용:

```
source ~/.bashrc
```

Windows

- TensorRT 설치 경로를 `PATH`에 추가:

```
C:\Program Files\NVIDIA GPU Computing Toolkit\TensorRT-<version>\lib
```

---

4. TensorRT Python API 테스트

TensorRT Python API가 제대로 동작하는지 확인하려면 아래 코드를 실행해 보세요:

```
import tensorrt as trt
```

```
print(f"TensorRT Version: {trt.__version__}")
```

5. TensorRT 모델 변환 및 실행

ONNX 모델을 TensorRT 엔진으로 변환

1. TensorRT의 `trtexec`를 사용하여 ONNX 모델을 변환:

   ```
   trtexec --onnx=model.onnx --saveEngine=model.trt --fp16
   ```

   주요 옵션:
   - `--onnx`: 입력 ONNX 모델 파일 경로.
   - `--saveEngine`: 저장할 TensorRT 엔진 파일 이름.
   - `--fp16`: FP16 최적화 활성화 (지원하는 GPU에서만 가능).
2. Python API를 사용하여 변환:
   TensorRT Python API를 활용하여 ONNX 모델을 TensorRT 엔진으로 변환할 수 있습니다:

   ```
   import tensorrt as trt

   TRT_LOGGER = trt.Logger(trt.Logger.WARNING)
   builder = trt.Builder(TRT_LOGGER)
   network = builder.create_network(
       1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH)
   )
   parser = trt.OnnxParser(network, TRT_LOGGER)

   with open("model.onnx", "rb") as model:
       if not parser.parse(model.read()):
           for error in range(parser.num_errors):
               print(parser.get_error(error))

   config = builder.create_builder_config()
   config.set_flag(trt.BuilderFlag.FP16)
   config.max_workspace_size = 1 << 30  # 1GB

   engine = builder.build_engine(network, config)
   with open("model.trt", "wb") as f:
       f.write(engine.serialize())
   ```

6. TensorRT 엔진 실행

TensorRT 엔진을 실행하려면 Python API를 사용합니다:

```
import tensorrt as trt
import pycuda.driver as cuda
import pycuda.autoinit

# TensorRT Logger
TRT_LOGGER = trt.Logger(trt.Logger.WARNING)

# Engine 로드
with open("model.trt", "rb") as f, trt.Runtime(TRT_LOGGER) as runtime:
    engine = runtime.deserialize_cuda_engine(f.read())

# Context 생성
context = engine.create_execution_context()

# 입력/출력 버퍼 할당 및 실행
# (입력 데이터와 출력 데이터 관리 코드 추가)
```

7. 디버깅 및 최적화

- `trtexec` **프로파일링**: 모델 최적화 및 디버깅에 유용합니다.

  ```
  trtexec --onnx=model.onnx --fp16 --dumpProfile --verbose
  ```

- **메모리 최적화**: TensorRT는 최대 메모리 크기를 설정해야 합니다. 적절한 크기로 조정하세요:

  ```
  config.max_workspace_size = 1 << 30  # 1GB
  ```

TensorRT 설정은 처음에는 복잡해 보이지만, 설정이 완료되면 매우 빠른 추론 속도를 얻을 수 있습니다. 문제나 의문 사항이 있으면 언제든지 질문하세요! 😊

› **편집자** 김량수 (01.21 15:15)

댓글