Queues, Threads, and Reading Data

The Input Pipeline

MNIST 이미지처럼 메모리에 올릴 수 있는 작은 데이터를 다룰 때는 모든 데이터를 메모리에 올리고 tensorflow 그래프에 데이터를 밀어 넣는 방식이 합리적이다. 하지만 데이터가 좀 더 클 때는 이러한 방법이 거추장스러워진다. 이러한 경우를 처리하는 자연스러운 패러다임은 데이터를 디스크에 보관하고 필요에 따라 데이터 덩어리를 로드하는 것이다.

전형적인 데이터 파이프라인은 서로 다른 종류의 입력 파일을 읽거나 입력의 형태나 구조를 바꾸거나 다른 형태의 전처리를 정규화하거나 입력을 셔플링하는 단계를 포함하는 경우가 많다.

이 과정의 많은 부분은 분리하여 모듈로 쪼갤 수 있다. 전처리의 예를 들면 전처리는 학습에 관여하지 않으므로 입력을 그대로 한꺼번에 전처리해서 학습으로 넘길 수 있다. 데이터를 배치로 학습시키기 때문에 원칙적으로는 디스크에서 데이터를 읽고, 전처리한 후 학습의 연산 그래프에 밀어 넣는 일괄 처리로 데이터를 다룰 수 있다.

하지만 이 방법은 효율적이지 않다. 전처리는 학습과 무관한데 전처리할 각 일괄 작업을 기다리느라 심각한 I/O 지연이 발생하고 각 학습 단계에서 로딩하여 처리될 데이터의 미니배치를 하염없이 기다리게 된다. 확장성 있는 접근법 중 하나는 데이터를 미리 가져와서 데이터의 로딩/처리와 학습에 별도의 스레드를 사용하는 것이다.

전처리를 염두에 두지 않더라도 지금까지 사용해온 표준 피딩 메커니즘을 사용하는 것은 그 자체로 낭비라는 사실에 주목할 필요가 있다. feed\_dict는 파이썬 런타임에서 tensorflow 런타임으로 데이터를 단일 스레드로 복사하는데, 이로 인해 지연이 발생하고 속도가 느려진다.

tensorflow에는 이 작업을 더 쉽고 더 빠르게 만들어주는, 입력 파이프라인 프로세스를 간소화할 수 있는 일련의 도구가 있다. 주요 구성 요소는 tensorflow의 표준 파일 형식과 이 형식을 인코딩/디코딩할 수 있는 유틸리티, 데이터 큐와 멀티스레드이다.

TFRecord

데이터셋의 형식은 매우 다양하며 때로는 혼재되어 있을 수도 있다. 원래의 형식이 무엇이든 간에 입력 파일들을 하나의 통합된 형식으로 변환하는 것이 편리하고 유용하다. tensorflow에서 기본 데이터 형식은 TFRecord이다. TFRecord 파일은 이진 파일로서 직렬화된 입력 데이터가 담겨 있다. 직렬화는 프로토콜 버퍼(Protocol Buffers)에 기반을 두고 있는데, 간단히 말해 프로토콜 버퍼는 데이터의 구조를 설명하는 스키마를 사용해 데이터를 저장용으로 변환하는 역할을 한다. 프로토콜 버퍼에서 저장하는 데이터는 XML과 비슷하게 사용하는 플랫폼이나 개발 언어와는 무관하다.

tensorflow 환경에서는 원래의 데이터 파일을 사용하는 것에 비해 TFRecord를 사용하는 것이 여러 가지 면에서 유리하다. 이 통합 형식을 사용하면 입력 인스턴스의 모든 관련 속성을 유지하면서도 많은 디렉터리나 하위 디렉터리가 필요하지 않도록 입력 데이터를 깔끔하게 정리할 수 있다. TFRecord 파일은 처리 속도도 매우 빠르다. 모든 데이터는 메모리의 하나의 블록에 저장되므로, 입력 파일이 개별로 저장된 입력 파일에 비해 메모리에서 데이터를 읽는 데 필요한 시간이 단축된다.

Writing with TFRecordWriter

먼저 입력 파일을 TFRecord 형식으로 작성하여 이를 사용할 수 있게 만들자. 이 예제에서는 MNIST 이미지를 이 형식으로 바꿀 것이다. 같은 방법으로 다른 유형의 데이터도 다룰 수 있다.

우선 tensorflow.contrib.learn의 유틸리티 함수를 사용하여 MNIST 데이터를 save\_dir에 내려받는다.(tfrecords\_read\_write.py)

내려받은 데이터에는 학습, 테스트, 유효성 검증 이미지가 각각 분리되어 들어 있다. 이를 스플릿이라고 부르고, 각 스플릿에 대해 데이터를 적당한 형식으로 만들어 TFRecordWriter()를 사용해 디스크에 저장한다.

먼저 TFRecordWriter 객체를 인스턴스화할 때 각 데이터 스플릿에 해당하는 경로를 지정했다. 그다음, 각 이미지에 대해서 넘파이 배열의 값을 바이트 스트링으로 변환한다. 다음에는 이미지를 프로토콜 버퍼 형식으로 변환한다. tf.train.Example이 예제 데이터를 저장하는 자료구조이다. Example 객체는 Features 객체를 포함하는데, Features 객체는 속성 이름에서 Feature로의 맵을 포함한다. Feature는 하나의 Int64List나 BytesList 또는 FloatList를 포함할 수 있다. 예를 들어 다음은 이미지의 레이블을 인코딩한다.

tf.train.Feature(int64\_list=tf.train.Int64List(value=[int(data\_set.labels[index])]))

그리고 다음은 실제 원본 이미지를 인코딩한다.

tf.train.Feature(bytes\_list=tf.train.BytesList(value=[image]))

저장된 데이터가 어떻게 생겼는지를 살펴보자. tf.python\_io.tf\_record\_iterator를 사용하면 된다. TFRecord 파일에서 레코드를 읽을 수 있는 이터레이터이다.

serialized\_img는 바이트 스트링이다. TFRecord에 이미지를 저장할 때 사용했던 구조체를 복구하려면 바이트 스트링을 파싱하면 된다. 그러면 앞에서 저장한 모든 속성에 접근할 수 있다.

이미지도 바이트 스트링으로 저장되었으므로 이를 넘파이 배열로 되돌려 (28, 28, 1)의 형태를 가진 tensor로 다시 바꾼다.