**Input Pipeline**

많은 사람들이 어려워하는 중요한 컨셉을 그 동안은 다루지 못했음

**Queues and Coordinators**

TensorFlow는 Queues를 “important TensorFlow objects for computing tensors asynchronously in a graph.”라고 설명하고 있음

Input pipeline에서 multi threads는 data phase를 읽는 과정에서 병목현상을 줄여줌

training model을 위한 input을 준비할 때, Queue를 사용하면 우리는 다음과 같은 결과를 얻음

* Multi threads가 training example을 준비하고, queue에 push
* Training threads는 queue에 있는 mini batch를 dequeue하는 op training을 실행

TensorFlow Session Object는 multi threads로 디자인 됨 → 쉽게 same session을 사용하고, 병렬적으로 실행가능

그러나, 위에서 설명한 대로 thread를 구동하는 python program을 구현하는 것이 항상 쉬운 것은 아님

모든 thread들은 항상 함께 멈출 수 있어야 하며, 예외들은 반드시 잡히고, 기록되어야 하고, queue가 멈출 때는 반드시 닫혀야함

documentation에서는 threading이 queue를 실행하는데 optional한 것처럼 나와있지만, 실제로 threading이 없으면 프로그램이 gridlock(하나의 op가 다른 op를 기다림 like traffic jam)에 빠지고 crush될 가능성이 많음

다행히, Tensorflow는 두개의 class를 제공함으로써 threading문제를 도와주고 있음

→ **tf.Coordinator** & **tf.QueueRunner**

두 개의 class는 함께 사용되도록 디자인 되었음

Coordinator class는 multi thread가 함께 멈추고, 다른 op가 멈추길 기다리는(gridlock에 빠진) 예외를 기록해주는 데 도움을 줌

QueueRunner class는 같은 queue에 있는 tensor를 queue에 넣기 위해 coorperating하는 많은 thread를 생성

**tf.FIFOQueue**와 **tf.RandomShuffleQueue**는 queue class의 main

FIFOQueue는 dequeue element가 FIFO(first in first out)순서를 갖도록 queue를 생성

RandomShuffleQueue는 dequeue element가 random한 순서를 갖도록 queue를 생성

이 두 queue들은 enqueue, enqueue\_many, dequeue를 지원함

Data를 읽을 때, enqueue할때는 한꺼번에 많은 예제를 넣지만, dequeue는 one by one으로 하는 것이 보통의 실험임 (dequeue\_many는 허용되지 않음)

만약 batch training에서 한꺼번에 많은 element들을 얻고 싶다면 tf.train.batch을 사용하면 되고, 만약 batch가 shuffle되기를 원하면 tf.train.shuffle\_batch를 사용 하면 됨

**tf.PaddingFIFOQueue** : FIFOQueue를 padding으로 tensor의 variable한 batching size를 일괄 처리를 지원

예를 들어, NLP 중 seq2seq 모델을 사용할 때 문장을 batch로 넣기 원하지만 문장마다 길이가 같지 않을 때 사용

PaddingFIFOQueue는 다양한 shape의 component를 포함하고 있고, dequeue\_many도 지원함

**tf.PriorityQueue** : enqueu 대기열 및 dequeue 대기열에서 다른 인수, 즉 우선순위를 갖는 FIFOQueue

왜 PaddingDIDOQueue에서만 dequeue\_many가 허용되고 다른 곳에서는 안되는지 정확한 이유를 모름

저자가 TensorFlow GitHub repo에서 reported된 issue를 읽고 난 후 추측으로는 dequeue\_many는 FIFOQueue와 RandomShuffleQueue를 support하는데 사용 되었지만, 사람들이 실행하는데 수 많은 problem들이 생겨서 TensorFlow에서 허용하지 않게 만든 것임

여러분들은 매개 변수를 사용하여 독립적으로 대기열을 만들 수 있음

* min\_after\_dequeue (사용자가 dequeue한 후 한 queue안에 있는 element의 최소 수)
* 제한된 용량 (한 queue에 들어갈 수 있는 elements의 최대 수)
* queueu안에 있는 element의 shape (만약 shape이 None이면 element는 어떤 shape이라도 가능)

실제로는 queue자체를 단독으로 사용하는 경우는 거의 없지만, string\_input\_producer를 항상 사용하므로 이 section을 간략히 살펴 볼 것임 우리는 string\_input\_producer를 자세히 살펴 볼 것임

<코드>

tf.Coordinator를 TensorFlow Queue와 사용할 필요는 없지만, 여러분이 만든 어떤 threads이던지 간에 threads를 관리하는 데에는 사용할 수 있음

예를 들면, 여러분은 어떤 미친 일(crazy job)을 하기위해 thread를 만들 때, python threading package를 사용하고, 그 thread를 관리하기 위해 tf.Coordinator도 사용할 수 있음

Target과 args의 구문은 고전적인 threadpool과 비슷threading의 detail을 더 보려면, CS110을 들으면 됨

<TensorFlow doc에 있는 예제-코드>

**Data Reader**

Lecture5에서 우리는 data reader에 관해서 배움 그러나 data reader는 사용하기에 약간의 까다롭고, 모호한 문서는 실제로 도움이 되지 않음

우리는 여러분의 TensorFlow를 위한 data를 읽는 3가지 다른 방법이 있다는 것을 배웠음

첫번째, 상수를 통하는 것 (그래프를 심각하게 부풀릴 것임)

두번째는 feed dict를 통하는 것인데, 저장소에서 클라이언트에게 데이터를 load한 다음에 클라이언트에서 작업자에게 데이터를 load한다는 단점이 있고, 이는 클라이언트와 작업자가 서로 다른 machine에 있을 때 속도가 느려짐

일반적인 practice는 date reader를 저장소에서 작업자에게 직접 load해주기 위해 사용함

이론상으로는, 이는 많은 양의 데이터를 load할 수 있는 것이 여러분의 저장소가 아닌 디바이스에서만 제한된다는 것임

여러 일반적인 데이터 타입에 대해 몇 개의 내장된 reader가 있음

제일 다재다능한 것은 TextLineReader인데, 이것은 newlines로 구분된 파일은 읽고 각 호출과 line을 반환

또한 고정된 length의 파일을 읽는 reader와, 문서 전체를 읽는 reader, 그리고 TFRecord(밑에서 얘기) 타입의 파일을 읽는 reader가 있음

Data reader를 사용하기 위해서 우리는 첫째로 여러분이 tf.train.string\_input\_producer를 통해 읽기를 원하는 모든 파일의 이름을 hold할 수 있는 queue를 만들어야 함

저자의 친구는 reader를 파이썬 generator와 비슷한 방식으로, 호출을 할 때 마다 다른 value를 return 시켜주는 ops라고 생각하기를 권장함

그래서 reader.read()를 호출할 때 마다 그것은 너에게 key, value 쌍을 반환할 것이고, 그것은 파일과 record(만약 너가 몇몇의 이상한 record를 가지고 있다면, 그것을 디버깅 하기에 유용한)와 scalar string 값을 identify해 주는 키임

각 예시들을 보면 read()를 호출하면 다음과 같은 결과가 반환됨(코드)

“144, 0.01, 4.41 …” 값은 heart.csv 파일의 두번째 행임(header행 제외)

tf.train.string\_input\_producer는 FIFOQueue를 hood 아래에 생성하기 때문에, queue를 실행시키려면 우리는 tf.Coordinator와 tf.QueueRunner가 필요함(코드)

반환된 value는 그냥 string tensor임

만약 너가 원하는 전부가 너의 모델에 feed할 string이 라면 괜찮음

그러나 거의 모든 시간은, 너는 string data가 feature들의 vector표현으로 바뀌는 것을 원해 왔음

예시로 우리의 heart.csv 파일은 10 columns이고, 처음 9column들은 9개의 feature들과 연관성 있고, 마지막column은 label과 연관 있음

그렇게 하기 위해서 우리는 TensorFlow CSV decoder를 사용해야함(코드)

위 코드 줄은 우리가 스스로 생성해야 하는 tensor record default의 값으로 parse하는 코드임

Record는 두가지 목적으로 제공됨

첫째, 코드는 각 열에서 예상 할 데이터 타입을 디코더에 알려줌

두번째, 만약 column의 공간이 비어 있을 경우, 우리가 지정해 준 데이터 타입의 기본 값으로 그 공간을 채움

이 특정 데이터셋의 record\_default에는 10개의 element가 있어야 함

string타입인 다섯 번째 element를 제외하고, 모든 element가 int이거나 float임

좀 더 쉽게 하려면, 우리는 모든 feature가 int나 float타입이라고 가정해야 함(우리는 label이 int인 것을 좋아하기 때문에 10번째 column을 int로 지정해 줌)(코드)

너의 데이터들을 feed in하기 전에 여러분이 필요한 모든 종류의 전처리를 할 수 있음

예를 들어, 우리는 8개는 float, 1개는 string, 1개는 int 타입인 10개의 element의 list를 우리의 content로 갖고 있음

우리는 string을 float로 바꿔야 하고(Absent는 0, Present는 1로), 그런 다음 처음 9개의 feature를 모델에 feed할 수 있는 tensor로 바꿔야 함(코드)

코드를 보면, reader는 우리의 csv 파일을 한 줄 읽을 때 마다 그 줄을 feature tensor와 label로 변환해 줌

그러나 우리는 종종 우리의 모델의 단일 sample이 feed되는 것을 원치 않지만 일괄처리 하려고 함

만약 너의 batch를 shuffle하고 싶다면 여러분은 tf.train.batch나 tf.train.shuffle\_batch를 사용하여 그렇게 할 수 있음(코드)

니가 feed\_dict parameter를 통해 그것들을 feed해줄 필요가 없는 것을 제외하고, 우리는 input\_placeholder와 label\_placeholder를 우리의 이전 모델에서 사용했던 방법처럼 단순하게 data\_batch나 label\_batch를 사용할 수 있음

전체 코드는 GitHub에 있음~

**TFRecord**

Binary 파일이 귀찮다고 생각하기 때문에 그들을 다소 멀리하는 사람을 많이 만났을 지라도, binary 파일은 매우 유용함

만약 니가 그 사람들 중 한명이라면 나는 이 강의가 니가 binary파일에 대해 느끼는 비합리적인 두려움을 극복하는 데 도움이 되길 바람

그들은 disk cache를 사용이 나아지도록 함

그들은 더 빨리 작동함

그들은 다른 타입의 데이터도 저장할 수 있음(따라서 너는 한 장소에 image와 label을 동시에 넣을 수 있음)

수 많은 머신 러닝 프레임워크처럼 TensorFlow도 TFRecord라고 불리는 자신만의 binary 데이터 포맷을 갖고 있음

TFRecord는 직렬화 된 tf.train.Example Product Object임

그들은 코드의 몇 개의 line을 생성함

밑의 코드는 image를 TFRecord로 변환하는 코드임

(코드)

첫째로 우리는 image를 읽고, 그것을 byte string으로 변환할 필요가 있음(코드)

그 다음, 너는 byte string를 tf.python\_io.TFRecordWriter와 tf.train.Features를 이용해서 TFRecord 파일에 써야함

나중에 binary file format에서 image를 재구성 할 수 있도록 shape에 대한 정보가 필요함(코드)

TFRecord 파일을 읽기 위해서 너는 TFRecordReader와 tf.decode\_raw를 사용하면 됨(코드)

반환되는 tensor 객체는 label, shape, image임

그들의 값은 얻으려면 너는 tf.Session()으로 평가하면 됨