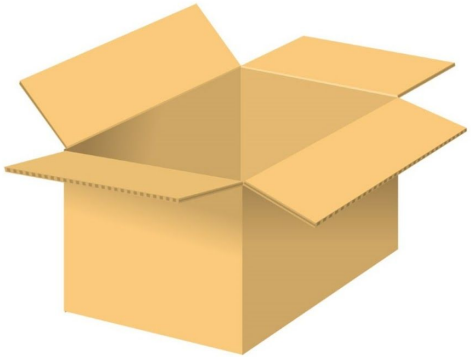
TensorFlow

Курс “Практическое применение по TensorFlow”
Шигапова Фирюза Зинатуллаевна
1-й семестр, 2019 г.

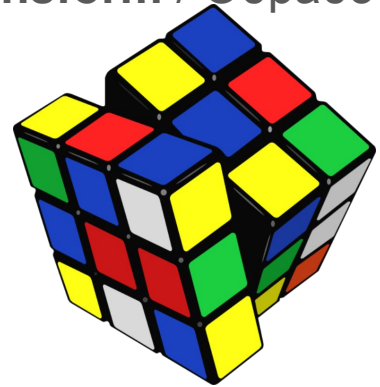
Data Input Pipeline

ETL



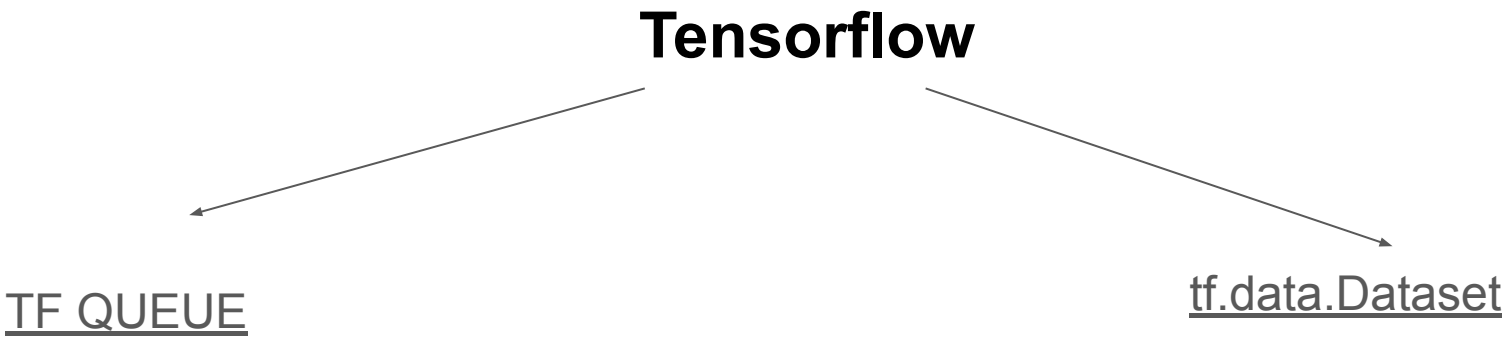
Extract / Извлечение

Transform / Обработка

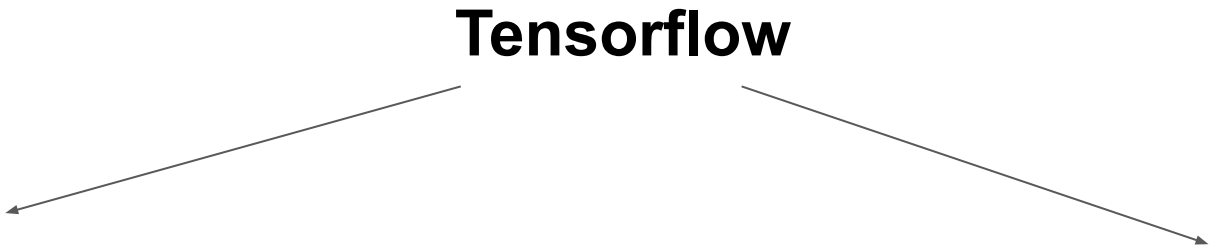


Load / Загрузка

Data Input Pipeline



Data Input Pipeline



TF QUEUE

- FIFOQueue
- PriorityQueue
- PaddingFIFOQueue
- RandomShuffleQueue

tf.data.Dataset

- map
- batch
- prefetch
- shuffle
- cache
- repeat
- ...

Priority Queue

Конструктор:

```
__init__(
    capacity,
    types,
    shapes=None,
    names=None,
    shared_name=None,
    name='priority_queue'
)
```

Priority Queue

```
priority = tf.placeholder(shape=(None), dtype=tf.int64)
image_path = tf.placeholder(shape=(None), dtype=tf.string)
image_label = tf.placeholder(shape=(None), dtype=tf.int64)

queue = tf.PriorityQueue(capacity=10, types=[tf.string, tf.int64], shapes=[(), ()])

enqueue_op = queue.enqueue_many([priority, image_path, image_label])
dequeue_op = queue.dequeue()

session = tf.Session()

session.run(enqueue_op, feed_dict={priority: [2, 1, 4, 3],
                                   image_path: ['item2', 'item1', 'item4', 'item3'],
                                   image_label: [0, 1, 1, 0]
                                   })

output = session.run(dequeue_op)
# [1, 'item1', 1]
```

Padding FIFO Queue

Конструктор:

```
__init__(
    capacity,
    dtypes,
    shapes,
    names=None,
    shared_name=None,
    name='padding_fifo_queue'
)
```

Padding FIFO Queue

```
sequence = tf.placeholder(shape=(None), dtype=tf.int64)
sequence_label = tf.placeholder(shape=(), dtype=tf.int64)
queue = tf.PaddingFIFOQueue(capacity=20, dtypes=[tf.int64, tf.int64], shapes=[(None,), ()])
enqueue = queue.enqueue([sequence, sequence_label])
dequeue = queue.dequeue_many(n=2)

session = tf.Session()

session.run(enqueue, feed_dict={sequence: [1, 2, 3],
                                sequence_label: 0})
session.run(enqueue, feed_dict={sequence: [1, 2],
                                sequence_label: 1})
output = session.run(dequeue)

print(output)
# [array([[1, 2, 3], [1, 2, 0]]),
# array([0, 1])]
```

Random Shuffle Queue

Конструктор:

```
__init__(
    capacity,
    min_after_dequeue,
    dtypes,
    shapes=None,
    names=None,
    seed=None,
    shared_name=None,
    name='random_shuffle_queue'
)
```

Random Shuffle Queue

```
image_path = tf.placeholder(shape=(None), dtype=tf.string)
image_label = tf.placeholder(shape=(None), dtype=tf.int64)

queue = tf.RandomShuffleQueue(capacity=10, dtypes=[tf.string, tf.int64],
                              shapes=[(), ()], min_after_dequeue=0)
enqueue_op = queue.enqueue_many([image_path, image_label])
dequeue_op = queue.dequeue()

session = tf.Session()

session.run(enqueue_op, feed_dict={image_path: ['item2', 'item1', 'item4', 'item3'],
                                   image_label: [0, 1, 1, 0]
                                   })

output = session.run(dequeue_op)
# random item will be chosen

print(output)
```

FIFO Queue

Конструктор:

```
__init__(
    capacity,
    dtypes,
    shapes=None,
    names=None,
    shared_name=None,
    name='fifo_queue'
)
```

FIFO Queue

```
image_path = tf.placeholder(shape=(None), dtype=tf.string)
image_label = tf.placeholder(shape=(None), dtype=tf.int64)

queue = tf.FIFOQueue(capacity=10, dtypes=[tf.string, tf.int64],
                     shapes=[(), ()])
enqueue_op = queue.enqueue_many([image_path, image_label])
dequeue_op = queue.dequeue()

session = tf.Session()

session.run(enqueue_op, feed_dict={image_path: ['item2', 'item1', 'item4', 'item3'],
                                   image_label: [0, 1, 1, 0]
                                  })

output = session.run(dequeue_op)
# ['item2', 0]
```

tf.train.Coordinator &. tf.train.QueueRunner

tf.train.Coordinator:

- controls a set of threads
- handles exceptions (*stop_on_exception* Context manager)

tf.train.QueueRunner:

- create multi threading

tf.train.Coordinator &. tf.train.QueueRunner

```
# QueueRunner for running enqueue op in parallel using NROF_THREADS
queue_runner = tf.train.QueueRunner(queue, [enqueue_op] * NROF_THREADS)
tf.train.add_queue_runner(queue_runner)

# Coordinator for launching QueueRunner
coordinator = tf.train.Coordinator()

enqueue_threads = queue_runner.create_threads(session, coord=coordinator, start=True)
```

Example. No batch

```
NROF_THREADS = 4
NROF_ITERATIONS = 4

image_path = ['item2', 'item1', 'item4', 'item3']
image_label = [0, 1, 1, 0]

queue = tf.FIFOQueue(capacity=10, dtypes=[tf.string, tf.int32], shapes=[(), ()])
enqueue_op = queue.enqueue_many([image_path, image_label])
dequeue_op = queue.dequeue()

session = tf.Session()

# QueueRunner for running enqueue op in parallel using NROF_THREADS
queue_runner = tf.train.QueueRunner(queue, [enqueue_op] * NROF_THREADS)
tf.train.add_queue_runner(queue_runner)

# Coordinator for launching QueueRunner
coordinator = tf.train.Coordinator()

enqueue_threads = queue_runner.create_threads(session, coord=coordinator, start=True)

for step in range(NROF_ITERATIONS):
    if coordinator.should_stop():
        break
    image_path_batch, label_batch = session.run(dequeue_op)

coordinator.request_stop()
coordinator.join(enqueue_threads)
```

Example. Batch

```
queue = tf.FIFOQueue(capacity=10, dtypes=[tf.string, tf.int64], shapes=[(), ()])

enqueue_op = queue.enqueue_many([image_path, image_label])

paths_images_and_labels = []
for _ in range(NROF_THREADS):
    filename, label = queue.dequeue()
    paths_images_and_labels.append([filename, label])

# Fill Queue to create a batch of examples
path_batch, label_batch = tf.train.batch_join(paths_images_and_labels,
                                              batch_size=BATCH_SIZE,
                                              enqueue_many=False)

session = tf.Session()

# Coordinator for launching QueueRunner
coordinator = tf.train.Coordinator()
# Run all queues that were launched by threads
enqueue_threads = tf.train.start_queue_runners(coord=coordinator, sess=session)
```

Example. Batch

```
NROF_THREADS = 4
NROF_ITERATIONS = 2
BATCH_SIZE = 2

image_path = tf.placeholder(shape=(None), dtype=tf.string)
image_label = tf.placeholder(shape=(None), dtype=tf.int64)

queue = tf.FIFOQueue(capacity=10, dtypes=[tf.string, tf.int64], shapes=[(), ()])

enqueue_op = queue.enqueue_many([image_path, image_label])

paths_images_and_labels = []
for _ in range(NROF_THREADS):
    filename, label = queue.dequeue()
    paths_images_and_labels.append([filename, label])

# Fill Queue to create a batch of examples
path_batch, label_batch = tf.train.batch_join(paths_images_and_labels,
                                              batch_size=BATCH_SIZE,
                                              enqueue_many=False)

session = tf.Session()

# Coordinator for launching QueueRunner
coordinator = tf.train.Coordinator()
# Run all queues that were launched by threads
enqueue_threads = tf.train.start_queue_runners(coord=coordinator, sess=session)

session.run(enqueue_op, feed_dict={image_path: ['item2', 'item1', 'item4', 'item3'],
                                   image_label: [0, 1, 1, 0]
                                   })

for step in range(NROF_ITERATIONS):
    if coordinator.should_stop():
        break
    image_path_output, label_output = session.run([path_batch, label_batch])
    print(image_path_output, label_output)

coordinator.request_stop()
coordinator.join(enqueue_threads)
```

TF FIFO Queue for image processing

1. 5-10 images with labels.
2. Use TF **FIFO** Queue for preprocessing images through **batch** via QueueRunner & Coordinator.
3. Apply any transformation functions to images

   (rotation, crop, resize etc. -- **tf.image**).

4. Save processed images.