13. Loading and Preprocessing Data with TensorFlow

What we’ll cover:

- The Data API
- The TFRecord Format
- Preprocessing the Input Features
- TF Transform
- The TensorFlow Datasets (TFDS) Project

The Data API

Create dataset object

Tell it where to get the data (text, binary, SQL, etc.)

…and how to transform it

Processing - write your own, or use standard layers in Keras:

Normalization

Encoding of non-numerical features

The Data API

Dataset

- Usually reads gradually from disk, but can operate entirely in RAM

  ```
  tf.data.Dataset.from_tensor_slices(<any data tensor>)
  ```

- Slices of <tensor> along the 1st dimension

The Data API - Chaining Transformations

```
X =  tf.range(10) # any data tensor
dataset = tf.data.Dataset.from_tensor_slices(X)
dataset = dataset.repeat(3).batch(7)
for item in dataset:
. . . <do something>
```

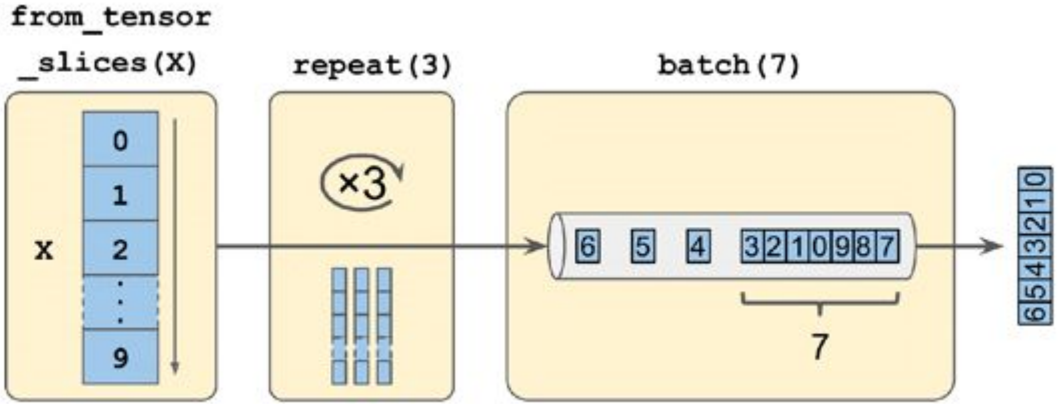
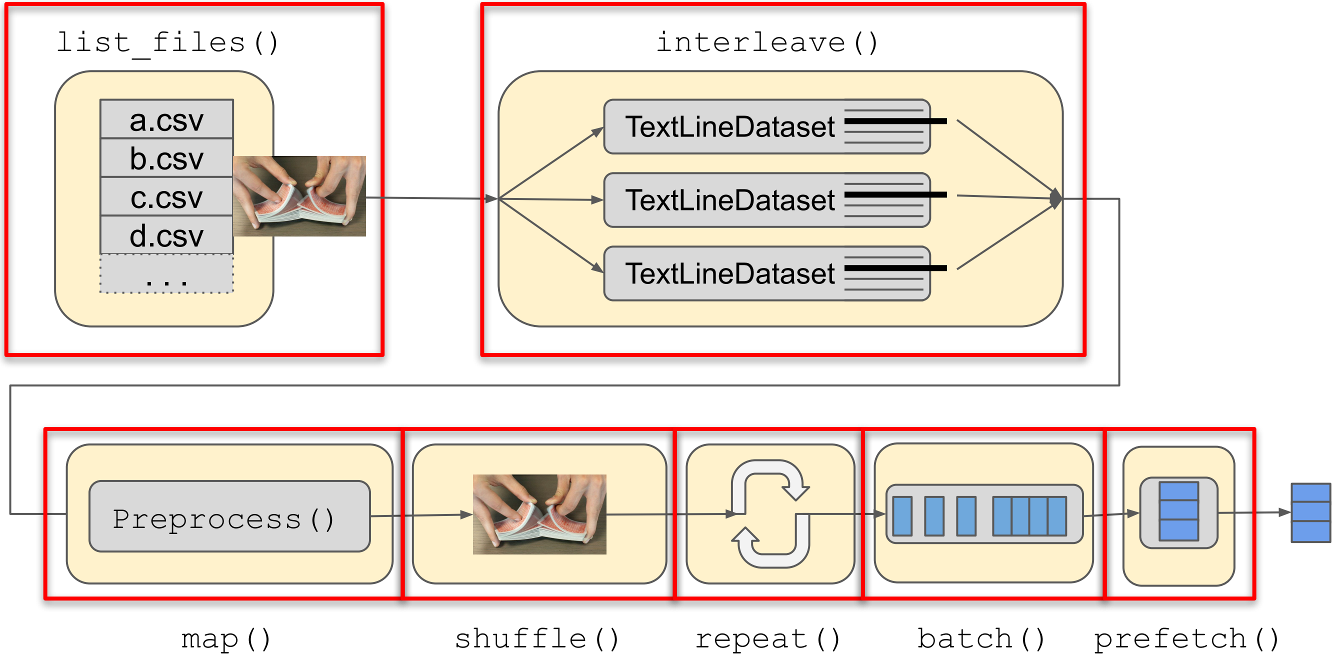


Figure 13-1. Chaining dataset transformations

The Data API - other functions

```
# must be convertible to a TF function
# apply to each item
dataset = dataset.map(lambda x: x * 2)
# apply to entire dataset
dataset = dataset.apply(tf.data.experimental.unbatch())
```



```
def csv_reader_dataset (filepaths, repeat=1, n_readers=5, n_read_threads=None,
shuffle_buffer_size=10000, n_parse_threads=5, batch_size=32):
      dataset = tf.data.Dataset.list_files(filepaths)
      dataset = dataset.interleave(lambda filepath: tf.data.TextLineDataset(filepath).skip( 1), \
                 cycle_length=n_readers, num_parallel_calls=n_read_threads)
      dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
      dataset = dataset.shuffle(shuffle_buffer_size).repeat(repeat)

      return dataset.batch(batch_size).prefetch( 1)
```

Preprocessing the Input Features

```
X_mean, X_std = [ ... ] # mean and scale of each feature in the training set

n_inputs = 8

def preprocess(line):
      defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
      fields = tf.io.decode_csv(line, record_defaults=defs)
      x = tf.stack(fields[:-1])
      y = tf.stack(fields[-1:])
      return (x - X_mean) / X_std, y
```

Preprocessing the Input Features

```
class Standardization(keras.layers.Layer):
     def adapt(self, data_sample):
          self.means_ = np.mean(data_sample, axis=0, keepdims=True)
          self.stds_ = np.std(data_sample, axis=0, keepdims=True)
     def call(self, inputs):
          return(inputs - self.means_) / (self.stds_ + keras.backend.epsilon())

std_layer = Standardization()
std_layer.adapt(data_sample)


model = keras.Sequential()
model.add(std_layer) [ ... ] # create the rest of the model
model.compile([ ... ])
model.fit([ ... ])


# Or
keras.layers.Normalization([...])

# see https://keras.io/api/layers/normalization_layers/
```

The TFRecord Format

```
with tf.io.TFRecordWriter("my_data.tfrecord") as f:
      f.write(b"This is the first record")
      f.write(b"And this is the second record")
```

Protobufs

```
syntax = "proto3";
message BytesList { repeated bytes value = 1; }
message FloatList { repeated float value = 1 [packed = true]; }
message Int64List { repeated int64 value = 1 [packed = true]; }
message Feature {
      oneof kind {
            BytesList bytes_list = 1;
            FloatList float_list = 2;
            Int64List int64_list = 3;
      }
};
message Features { map<string, Feature>feature = 1; };
message Example { Features features = 1; };
```

The TFRecord Format - Example Protobuf

```
from tensorflow.train import BytesList, FloatList, Int64List
from tensorflow.train import Feature, Features, Example

person_example = Example(
      features = Features(feature = {
            "name": Feature(bytes_list=BytesList(value=[b"Alice"])),
            "id": Feature(int64_list=Int64List(value=[123])),
            "emails": Feature(bytes_list=BytesList(value=[b"a@b.com", b" c@d.com"]))
      }))

# Writing
with tf.io.TFRecordWriter("my_contacts.tfrecord") as f:
      f.write(person_example.SerializeToString())
```

The TFRecord Format - reading

```
# Reading
feature_description = {
      "name": tf.io.FixedLenFeature([], tf.string, default_value=""),
      "id": tf.io.FixedLenFeature([], tf.int64, default_value=0),
      "emails":tf.io.VarLenFeature(tf.string),
}

for serialized_example in tf.data.TFRecordDataset(["my_contacts.tfrecord"]):
      parsed_example = tf.io.parse_single_example(serialized_example, feature_description)
```

The TFRecord Format - SequenceExample Protobuf

```
message FeatureList { repeated Feature feature = 1; };
message FeatureLists { map<string, FeatureList>feature_list = 1; };
message SequenceExample {
      Features context = 1;
      FeatureLists feature_lists = 2;
};


# Reading
parsed_context , parsed_feature_lists = tf.io.parse_single_sequence_example( \
      serialized_sequence_example, context_feature_descriptions, sequence_feature_descriptions)
parsed_content = tf.RaggedTensor.from_sparse(parsed_feature_lists["content"])
```

TF Transform (tf.Transform)

```
import tensorflow_transform as tft

def preprocess(inputs): # inputs = a batch of input features
      median_age = inputs["housing_median_age"]
      ocean_proximity = inputs["ocean_proximity"]
      standardized_age = tft.scale_to_z_score(median_age)
      ocean_proximity_id = tft.compute_and_apply_vocabulary(ocean_proximity)
      return {
            "standardized_median_age": standardized_age,
            "ocean_proximity_id": ocean_proximity_id
      }
```

The TensorFlow Datasets (TFDS) Project

```
import tensorflow_datasets as tfds

dataset = tfds.load(name = "mnist", batch_size=32, as_supervised=True)
mnist_train = dataset["train"].prefetch(1)
model = keras.models.Sequential([ ... ])
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd")
model.fit(mnist_train, epochs=5)
```

Questions?

THANK YOU!