Introduction to Data with Tensorflow 2.0

Yogesh Kulkarni

November 5, 2020

Introduction to Data Science

What is Data Science ?

*M*ulti-disciplinary field that brings together concepts from computer science, statistics/machine learning, and data analysis to understand and extract insights from the ever-increasing amounts of data.

Paradigms of Data Research

- Hypothesis-Driven: Given a problem, what kind of data do we need to help solve it?
- Data-Driven: Given some data, what interesting problems can be solved with it?

What is core of Data Science ?

- What can we learn from this data?
- What actions can we take once we find whatever it is we are looking for?

What is Statistics?

- Statistics is the science of learning from data.
- The goal of statistics is to summarize data in a way that allows for easy descriptions or inferences of the data.

Data

- Data is by Measurement
- Measurement: Process of assigning numbers to types
- Each Data has Type and Value

Data

id	Home Owner	Marital Status	Annual Income	Defaulted Barrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Vocabulary

- Column-Type: “attribute”, “feature”, “field”, “dimension”, “variable”
- Row-Value: “instance”, “record”, “observation”

Data

- Data Type
- Data Value
- Distinctions:
  - Same type - different values. Example: height can be measured in feet or meters
  - Different types same values. Example: Attribute values for ID and age

Types of Data

- **Categorical types (Qualitative):** Nominal and Ordinal
  - **Nominal** (numbers do not give sense of order/rank): eye color, zip codes
  - **Ordinal** (numbers give sense of order/rank): rankings, size in small-medium-large
- **Numeric types (Quantitative):** Interval and Ratio
  - **Discrete:** A discrete attribute has a finite or countably infinite set of values. **Binary attributes** are a special case of discrete attributes.
  - **Continuous:** A continuous attribute is one whose values are real numbers.
  - **Interval**: calendar dates
  - **Ratio**: counts, time

NOIR: **N**o **O**il **I**n **R**ivers

Ordinal

- To show relative rankings
- Order matters, but not diff
- $\mathit{Diff}(7,5) \neq \mathit{Diff}(5,3)$
- "First is first, however close the second is!!"
- Examples: class rank, levels of wellness.

Interval

- Diff equal if measured between two equivalent variables
- $Diff(100, 90) == Diff(90, 80)$
- Same amount of heat needed to take from 90 to 100 or 80 to 90
- Examples: test scores and temperature

Ratio

- Weight of 8 grams is twice the weight of 4 grams
- Clear definition of 0.0; none of a variable at 0.0
- Example: height, weight, pulse and BP

Ordered Data

- Temporal: time based, e.g. retail transaction
- Sequential: e.g. DNA sequence (ATGC possible letters)
- Time Series: Series of measurements taken over time. e.g.: financial stock price data
- Spatial Data: e.g. geographical locations

Measures of Similarity and Dissimilarity

- **Proximity:** either similarity or dissimilarity
  - **Similarity:** a numerical measure of likeliness.
  - **Dissimilarity:** a numerical measure of unlikeliness
- **Transformations:** re-parametrize proximity to, say, [0, 1].

Similarity and Dissimilarity between Simple Attributes

- **Nominal:** simple equality test
- **Ordinal:** order should be taken into account.
- **Ratio:** simple absolute difference of their values.
- Distance is the measure of similarity-dissimilarity.
- Distance between two data items can either be 0 or 1 for congruence.

Euclidean distance

$$d(a, b) = \sqrt{\sum_{k=1}^{n} (a_k - b_k)^2}$$

Minkowski distances

$d(a, b) = \left(\sum_{k=1}^{n} |a_k - b_k|^r\right)^{1/r}$

- **r = 1** City block (Manhattan, taxicab, L_1 norm) distance.
- **r = 2** Euclidean distance (L_2 norm).
- **r = ∞** Supreme (L_{max} or L_∞) distance. Here r is not put equal to ∞ but tends to or limit to ∞

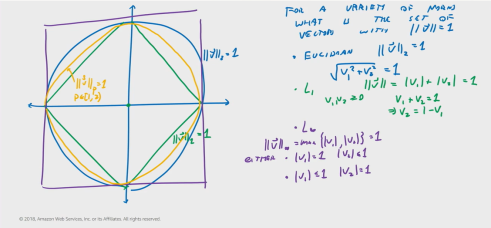
Considering this formula for finding norm (ie length, or tcan be same as length of diff between a and b):

- Putting a very large number, say, 1000, for $v = [0110]$, the norm would be $(0^{1000} + 1^{1000} + 10^{1000})^{1/1000}$.
- Here the last number becomes very big, but its 1000th root gives the same number as answer ie 10.
- So, this method biases towards large numbers.

Visualizing Norms

Vectors having:

- Euclidean distance Norm as 1, when plotted look like circle.
- Manhattan distance as Norm as 1, when plotted makes a inner Square.
- Minkowski distance with $r = \infty$ norm as 1, when plotted looks outer Square.
- Minkowski distance with smaller r norm as 1, when plotted looks somewhere between.



(Ref: Math for Machine Learning - Brent Werness, AWS)

Properties Euclidean distances

- **Positivity**
  - d(a,b) $\geq$ for all a and b,
  - d(a,b) = 0 only if a = b
- **Symmetry**
  d(a,b) = d(b,a) for all a and b
- **Triangle Inequality**
  d(a,c) $\leq$ d(a,b) + d(b,c) for all points a, b, and c

Metrics

Measures that satisfy all three properties are known as metrics.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

TABLE: (a) x and y coordinates, (b) Euclidean distance matrix

Metrics

Measures that satisfy all three properties are known as metrics.

L_1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

L_∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0
p4	5.0	3.0	2.0	0.0

TABLE: (a) L_1 distance matrix, (b) L_∞ distance matrix

Simple Matching Coefficient (SMC)

Have values between 0 and 1.
Let a and b be two objects that consist of n binary attributes i.e., two binary vectors
Possibilities:

- f_{00} = the number of attributes where a is 0 and b is 0
- f_{01} = the number of attributes where a is 0 and b is 1
- f_{10} = the number of attributes where a is 1 and b is 0
- f_{11} = the number of attributes where a is 1 and b is 1

$$SMC = \frac{number\ of\ matching\ attribute\ values}{number\ of\ attributes} = \frac{f_{11}+f_{00}}{f_{01}+f_{10}+f_{11}+f_{00}}$$

Cosine Similarity

$$cos(a,b) = \frac{a.b}{\|a\|\|b\|} = \frac{\sum a_i.b_i}{\sqrt{\sum (a_i)^2}.\sqrt{\sum (b_i)^2}}$$

Introduction to TensorFlow Data

Data Pipeline

- Goal: Having an efficient, scalable as well as generic pipeline
- Multiple sources, multiple formats (image, text, csv, server log file, videos, audio files, etc.)
- Tensorflow tf.data module can be easily customized to consume these data efficiently and send it to the model for further computations.

What ML needs from Data Pipeline

- Machine learning models are data-hungry
- Before data is fed to an ML model it should be:
  - Shuffled
  - Batched
  - Be available before the current epoch is finished

(Ref: Building data pipelines with tf.data - Sayak Paul)

Tensorflow Data

- `tf.data` module
- `tf.data.Dataset` abstraction that represents a sequence of elements, in which each element consists of one or more components, which can be used as a source to consume data.
- E.g. a single element in spam classifier dataset has two components: a text data and its label
- `tf.data.Dataset` behaves as a python iterator, which can be easily accessed using a python *for* loop.

Load Data

(Ref: Building a High-Performance Data Pipeline with Tensorflow 2.x - Mayank Kumar)

Tensorflow Ready Datasets

```
train, test = tf.keras.datasets.fashion_mnist.load_data()

images, labels = train
images = images/255

dataset = tf.data.Dataset.from_tensor_slices((images, labels))
```

From Small CSV

`tf.data.Dataset.from_tensor_slices`

- CSV small enough to fit in memory.
- Read it into Pandas dataframe
- `tf.data.Dataset.from_tensor_slices` to convert dataframe object to `tf.data.Dataset`

```
df = pd.read_csv("sample.csv")
dataset = tf.data.Dataset.from_tensor_slices(dict(df))
```

From Python Generator

`tf.data.Dataset.from_generator`

- Uses python generators for consuming the dataset.
- Can incorporate transformation logic on python side.
- Transformation happens on-the-go during data consumption.

```python
def read_csv(filepath,skip_rows):
  with open(filepath,'r') as csvfile:
    data = csv.reader(csvfile, delimiter=",")
    for index, row in enumerate(data):
      if index < skip_rows:
        continue
      yield row[0],row[1],row[2]

dataset = tf.data.Dataset.from_generator(read_csv, output_types =(tf.string,
                                          tf.int8,tf.float16)
```

From CSV

`tf.data.experimental.make_csv_dataset`

- Batch-wise loading
- Shuffling possible

```
path = tf.keras.utils.get_file(filepath)
dataset = tf.data.Dataset.make_csv_dataset(path,batch_size=32,
                                            label_name = target_column,
                                            select_columns=feature-columns,
                                            shuffle=True)
```

From BIG Data

`tf.data.TFRecordDataset`

- Used to consume any amount of data in a most efficient way.
- To build a streaming input pipeline that can stream over the content of one or more TFRecord files.
- Store any dataset like CSVs, Images, Texts, etc., into a set of TFRecord files, using `TFRecordWriter`
- For training, need to get original format back using `tf.train.Example` in case of scalar features and `tf.serialize_tensor` in case of non-scalar features

```python
# Creating a serializer to serialize our dataset
# as per datatype a particular column has
def encoder(row0, row1, row2):
    return tf.train.Example(features=tf.train.Features(feature={
        "name": _bytes_feature(row0),
        "age": _int64_feature(row1),
        "score": _float_feature(row2)
    })).SerializeToString()
```

From BIG Data

tf.data.TFRecordDataset

```python
# Creating csv genrator to iterate over csv data and generate
# serilizable data
def read_csv(file_path="./sample.csv", skip_rows=1):
    with open('./sample.csv', 'r') as csvfile:
        data = csv.reader(csvfile, delimiter=',')
        for index, row in enumerate(data):
            if index < skip_rows:
                continue
            yield encoder(row0=row[0].encode("utf-8"),
                          row1=int(row[1]),
                          row2=float(row[2]))
```

```python
# Creating a TFRecord writer
def tf_record_writer(in_file="./sample.csv", out_file="./sample.tfreco
    with tf.io.TFRecordWriter(out_file) as writer:
        for record in read_csv(file_path=in_file):
            writer.write(record)

# Writing our CSV into TFRecords format
tf_record_writer()
```

From BIG Data

`tf.data.TFRecordDataset`

```python
# Creating a decoder to decode the TFRecord data at consumption phase
def decoder(record):
    return tf.io.parse_single_example(
      record,
      {"name": tf.io.FixedLenFeature([], dtype=tf.string),
       "age": tf.io.FixedLenFeature([], dtype=tf.int64),
       "score":tf.io.FixedLenFeature([], dtype=tf.float32)
      }
    )
```

```python
# Consuming our TFRecord file as a tf.data.Dataset object
dataset = tf.data.TFRecordDataset(["./sample.tfrecords"]).map(decoder)

```

Text Data

Loading text data

- Many datasets are distributed as one or more text files.
- The tf.data.TextLineDataset provides an easy way to extract lines from one or more text files.
- Given one or more filenames, a TextLineDataset will produce one string-valued element per line of those files.

```
directory_url =
     'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
file_names = ['cowper.txt', 'derby.txt', 'butler.txt']

file_paths = [
    tf.keras.utils.get_file(file_name, directory_url + file_name)
    for file_name in file_names
]
```

Loading text data

- By default, a TextLineDataset yields every line of each file, which may not be desirable, for example, if the file starts with a header line, or contains comments.
- These lines can be removed using the Dataset.skip() or Dataset.filter() transformations.
- Here, you skip the first line, then filter to find only survivors.

```
dataset = tf.data.TextLineDataset(file_paths)

titanic_file = tf.keras.utils.get_file("train.csv",
     "https://storage.googleapis.com/tf-datasets/titanic/train.csv")
titanic_lines = tf.data.TextLineDataset(titanic_file)
```

After Dataset

- Use `.prefetch()` to send next batch data to GPUs or TPUs while current batch is getting completed
- Use `.interleave()` with parallelism for transformation of the dataset, if transformation is needed at the time of consumption phase.
- Use `.cache()` to cache some computations like transformation which may be overlapping during each epoch.

After Dataset

```
train_dataset = train_dataset.\
    shuffle(buffer_size=1000).\
    repeat().\
    batch(256).\
    prefetch(buffer_size=1000)
```

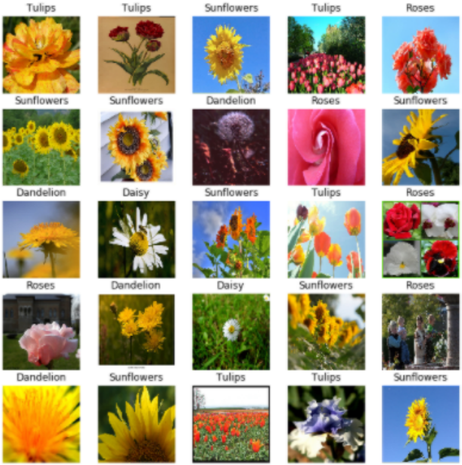
```
# Define and compile a model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(train_dataset,
    steps_per_epoch=len(X_train)//256,
    epochs=5,
    validation_data=test_dataset.batch(256)
```

(Ref: Building data pipelines with tf.data - Sayak Paul)

Image Data

Loading Images



(Ref: Building data pipelines with tf.data - Sayak Paul)

Download Data

Assume Data is in following directory tree:

```
import pathlib
dataset_url =
     "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos
data_dir = tf.keras.utils.get_file(origin=dataset_url,
                                   fname='flower_photos',
                                   untar=True)
data_dir = pathlib.Path(data_dir)

flowers_photos/
  daisy/
  dandelion/
  roses/
  sunflowers/
  tulips/
```

Load Data

```python
batch_size = 32
img_height = 180
img_width = 180

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Visualize Data

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

(Ref: https://www.tensorflow.org/tutorials/load_data/images)

Image Preprocessing

```
list_ds = tf.data.Dataset.list_files(str(flowers_root/'*/*'))

# Reads an image from a file, decodes it into a dense tensor, and resizes it
# to a fixed shape.
def parse_image(filename):
  parts = tf.strings.split(filename, os.sep)
  label = parts[-2]

  image = tf.io.read_file(filename)
  image = tf.image.decode_jpeg(image)
  image = tf.image.convert_image_dtype(image, tf.float32)
  image = tf.image.resize(image, [128, 128])
  return image, label

file_path = next(iter(list_ds))
image, label = parse_image(file_path)

def show(image, label):
  plt.figure()
  plt.imshow(image)
  plt.title(label.numpy().decode('utf-8'))
  plt.axis('off')

show(image, label)
```

Standardize Data

```
from tensorflow.keras import layers
normalization_layer =
     tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
```

There are two ways to use this layer. You can apply it to the dataset by calling map:

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

Standardize Data

Or, you can include the layer inside your model definition to simplify deployment.

```
num_classes = 5
model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Image Preprocessing

If you want to apply a random rotation, the tf.image module only has tf.image.rot90, which is not very useful for image augmentation.

```
def tf_random_rotate_image(image, label):
  im_shape = image.shape
  [image,] = tf.py_function(random_rotate_image, [image], [tf.float32])
  image.set_shape(im_shape)
  return image, label

rot_ds = images_ds.map(tf_random_rotate_image)

for image, label in rot_ds.take(2):
  show(image, label)
```

ImageDataGenerator

Initialize ImageDataGenerator with the augmentations.

```
train_aug = ImageDataGenerator(
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

(Ref: Building data pipelines with tf.data - Sayak Paul)

tf.data

Initialize TF Dataset from directory of files (see "train_aug" from ImageDataGenerator)

```
train_set = tf.data.Dataset.from_generator(
    lambda: train_aug.flow_from_directory(flowers,
            class_mode="categorical",
            target_size=(224, 224),
            color_mode="rgb",
            shuffle=True),
    output_types=(tf.float32, tf.float32),
    output_shapes=([None,224,224,3],[None,5])
)
```

(Ref: Building data pipelines with tf.data - Sayak Paul)

Time Series Data

Time series windowing

```
range_ds = tf.data.Dataset.range(100000)
batches = range_ds.batch(10, drop_remainder=True)
for batch in batches.take(5):
  print(batch.numpy())

[0 1 2 3 4 5 6 7 8 9]
[10 11 12 13 14 15 16 17 18 19]
[20 21 22 23 24 25 26 27 28 29]
[30 31 32 33 34 35 36 37 38 39]
[40 41 42 43 44 45 46 47 48 49]
```

Time series windowing

To make dense predictions one step into the future, you might shift the features and labels by one step relative to each other:

```
def dense_1_step(batch):
  # Shift features and labels one step relative to each other.
  return batch[:-1], batch[1:]

predict_dense_1_step = batches.map(dense_1_step)

for features, label in predict_dense_1_step.take(3):
  print(features.numpy(), " => ", label.numpy())

[0 1 2 3 4 5 6 7 8]  =>  [1 2 3 4 5 6 7 8 9]
[10 11 12 13 14 15 16 17 18]  =>  [11 12 13 14 15 16 17 18 19]
[20 21 22 23 24 25 26 27 28]  =>  [21 22 23 24 25 26 27 28 29]
```

Time series windowing

To predict a whole window instead of a fixed offset you can split the batches into two parts:

```
batches = range_ds.batch(15, drop_remainder=True)

def label_next_5_steps(batch):
  return (batch[:-5],   # Take the first 5 steps
          batch[-5:])   # take the remainder

predict_5_steps = batches.map(label_next_5_steps)

for features, label in predict_5_steps.take(3):
  print(features.numpy(), " => ", label.numpy())

[0 1 2 3 4 5 6 7 8 9]  =>  [10 11 12 13 14]
[15 16 17 18 19 20 21 22 23 24]  =>  [25 26 27 28 29]
[30 31 32 33 34 35 36 37 38 39]  =>  [40 41 42 43 44]
```

Time series windowing

To allow some overlap between the features of one batch and the labels of another, use Dataset.zip:

```
feature_length = 10
label_length = 5

features = range_ds.batch(feature_length, drop_remainder=True)
labels = range_ds.batch(feature_length).skip(1).map(lambda labels: labels[:-5])

predict_5_steps = tf.data.Dataset.zip((features, labels))

for features, label in predict_5_steps.take(3):
  print(features.numpy(), " => ", label.numpy())

[0 1 2 3 4 5 6 7 8 9]  =>  [10 11 12 13 14]
[10 11 12 13 14 15 16 17 18 19]  =>  [20 21 22 23 24]
[20 21 22 23 24 25 26 27 28 29]  =>  [30 31 32 33 34]
```

Time series windowing

Putting this together you might write this function:

```
def make_window_dataset(ds, window_size=5, shift=1, stride=1):
  windows = ds.window(window_size, shift=shift, stride=stride)
  def sub_to_batch(sub): # function defined within function!!
    return sub.batch(window_size, drop_remainder=True)
  windows = windows.flat_map(sub_to_batch)
  return windows

ds = make_window_dataset(range_ds, window_size=10, shift = 5, stride=3)
for example in ds.take(10):
  print(example.numpy())

[ 0  3  6  9 12 15 18 21 24 27]
[ 5  8 11 14 17 20 23 26 29 32]
[10 13 16 19 22 25 28 31 34 37]
[15 18 21 24 27 30 33 36 39 42]
[20 23 26 29 32 35 38 41 44 47]
[25 28 31 34 37 40 43 46 49 52]
[30 33 36 39 42 45 48 51 54 57]
[35 38 41 44 47 50 53 56 59 62]
[40 43 46 49 52 55 58 61 64 67]
[45 48 51 54 57 60 63 66 69 72]
```

Time series windowing

Then it's easy to extract labels, as before:

```
dense_labels_ds = ds.map(dense_1_step)

for inputs,labels in dense_labels_ds.take(3):
  print(inputs.numpy(), "=>", labels.numpy())

[ 0  3  6  9 12 15 18 21 24] => [ 3  6  9 12 15 18 21 24 27]
[ 5  8 11 14 17 20 23 26 29] => [ 8 11 14 17 20 23 26 29 32]
[10 13 16 19 22 25 28 31 34] => [13 16 19 22 25 28 31 34 37]
```

Model Training

```
model = get_me_a_good_model()
model.fit(train_set,
          steps_per_epoch=train_data//32,
          epochs=5)
```

(Ref: Building data pipelines with tf.data - Sayak Paul)

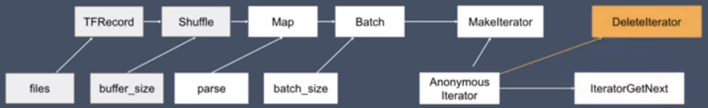
Conclusions

Advantages of tf data

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up the model training.

Summary

```
dataset = tf.data.TFRecordDataset(files)

dataset = dataset.shuffle(buffer_size=X)

dataset = dataset.map(lambda record: parse(record))

dataset = dataset.batch(batch_size=Y)


for element in dataset:

  ...

#  iterator goes out of scope
```



(Ref: Inside TensorFlow: tf.data - TF Input Pipeline)

References

Many publicly available resources have been refereed for making this presentation. Some of the notable ones are:

- Introduction to Data Science - Daisy Zhe Wang
- Introduction to Data Science - Kamal Al Nasr, Matthew Hayes and Jean-Claude Pedjeu
- Introduction to Data Science - Thomas M. Carsey
- Big Data [sorry] & Data Science: What does a data scientist do? - Carlos Somohano

Thanks ... yogeshkulkarni@yahoo.com