Chapter 13: Loading and Preprocessing Data with TensorFlow

- Introduction
  - Create dataset object
  - TFRecord: supports varying types, binary format, protocol buffers
  - Preprocessing handled too
- The Data API
  - Chaining Transformations
    - repeat(), batch()
    - Dataset methods do not modify datasets, create new ones
    - Use map() to apply transformation to each item, apply() to dataset as whole
      - Function must be convertible to TF Function as in Chap 12
    - Use filter() and take()
  - Shuffling the Data
    - Want data to be iid, use shuffle() → pulls out data from buffer randomly
      - Specify buffer size, fills up buffer from first items from source dataset, then when asks for item pulls it out randomly from dataset and replaces with fresh one from source until iterated through source
    - Can shuffle source data too
    - Can randomly split into separate files and source, but data in same file will always be close, so pick multiple files randomly, and read simultaneously
    - Function list_files() returns a dataset that shuffles file paths
    - Use interleave() to read from different files and interleave lines
      - Creates dataset that pulls number of file paths from a file path dataset
      - Calls lambda function one created to create new dataset
      - Data sets: file path, interleave, and number of data sets from lambda function
      - Iterates over interleave dataset to read a line from each of lambda function datasets continuing until out of items
      - That will go to next file paths
      - Best practice files of similar length
  - Preprocessing the Data
    - Function tf.io.decode_csv() takes line to parse and array containing default value for each column in CSV
  - Putting Everything Together
    - Load files from multiple CSV files, preprocess, shuffle, repeat, bacth
  - Prefetching
    - Creating a dataset that is one batch ahead
    - Ensures loading and preprocessing are multithreaded
    - Memory bandwidth: number of gigs can get into or out of RAM important
    - Small enough dataset: use cache() after loading and preprocessing, but before shuffling, repeating, batching and prefetching

            - Each instance read and preprocessed only once, but data shuffled differently at each epoch and next batch prepared in advance
        - Other dataset methods to look at:
            - Methods: concatenate(), zip(), window(), reduce(), shard(), flat_map(), padded_batch()
            - Class methods: from_generator() and from_tensors()
    - Using the Dataset with tf.keras
        - Possible to create a function with decorator that performs whole training loop with loading data
- The TFRecord Format
    - Intro: TF preferred format for storing large amounts of data and reading it efficiently
        - Create with tf.io.TFRecordWriter()
        - Then use tf.data.TFRecordDataset() to read
    - Compressed TFRecord Files
        - Create compresed files by setting compression_type=”GZIP”
        - The specify compression_type when reading
    - A Brief Introduction to Protocol Buffers
    - TensorFlow Protobufs
    - Loading and Parsing Examples
    - Handling Lists of Lists Using the SequenceExample Protobuf
- Preprocessing the Input Features
    - Intro
        - Preparing data for NN requires converting all features into numbers, normalizing, and more
        - If data is categorical or text, needs to be converted to numbers
        - Can do ahead, on the fly when loading data with Data API, or can include preprocessing layer in model
    - Encoding Categorical features Using One-Hot Vectors
        - Out-of-vocabulary (oov) buckets: for large or changing data sets to handle unknown categories and to prevent collisions
        - Rule of thumb:
            - number of categories < 10, use one-hot encoding
            - number of categories > 50, use embedding
            - 10 < number of categories < 50, experiment
    - Encoding Categorical features Using Embeddings
        - Embedding is a trainable dense vector that represents a category
        - Represented initially by random vector, number of dimensions is hyperparameter
        - Values will change as model is trained
        - Word embeddings
            - Vectors to represent words
            - Similar words, meanings will have similar proximity and axes in embedding space

            - King to Man, Queen to Woman, Doctor to Man, Nurse to Woman, bias!
        - Keras.layers.Embedding() initializes random embedding matrix
        - One-hot encoding followed by a Dense layer (no activation, no biases) equivalent to an Embedding layer, but Embedding uses fewer computations
            - Wasteful to use more embedding dimensions that number of units in layer that follows Embedding layer
    - Keras Preprocesing Layers
        - In development
- TF Transform
    - Handling preprocessing before training can speed up process
    - If data can fit in RAM use cache() method
    - If too large use Apache Beam or Spark
    - This process will struggle if deploying in app, will still need preprocessing
    - Time-consuming and leads to bugs: training/serving skew
    - Can take training model and add extra preprocessing layers to take of preproc on fly or defined preproc ops once using TF Transform
    - Define preproc using TF Transform funcs for scaling, bucketizing, et al.
- The TensorFlow Datasets (TFDS) Project
    - Easy download of common datasets
    - Call tfds.load(). Shuffles data shard it downloads only for training set. Might not be sufficient, so shuffle training data too