Slide 14 Questions

1. Understand the differences between Google Dataflow and Dataproc

- **Google Dataflow:** place to run Apache Beam based jobs, without the need to address the technicalities of running jobs on a cluster (such as work balancing and scaling), since this is automatically done for you. With Google Dataflow you focus on the logical computation rather than how the runner works. Additionally, Dataflow provides base templates that simplify common tasks. Beams has the downside of only supporting *Python 2.7*.
- **Dataproc:** provides you with a Hadoop Cluster and access to Apache Hadoop / Spark ecosystem. Indicated when the manual provisioning of clusters is necessary (as seen before, GD does it automatically) OR when there are dependencies to tools belonging to the aforementioned ecosystem.

2. Understand how a pipeline is created

To construct a pipeline, the program performs the following general steps:

- Creates a Pipeline object.

```
with beam.Pipeline(options=beam_options) as p
```

- Uses a Read transform to create a PCollections for our pipeline data from an external source.

```
beam.io.Read(pubchem.ParseSDF(data_files_pattern))
```

- Applies transforms to each PCollection. Transforms can change, filter, group, analyze, or otherwise process the elements in a PCollection. Each transform creates a new output PCollection, to which we can apply additional transforms until processing is complete.

```
  | 'Feature extraction' >> feature_extraction`
  | 'Predict' >> beam.ParDo(Predict(model_dir, 'ID'))
  | 'Format as JSON' >> beam.Map(json.dumps)
```

- Write the final output, transformed PCollections.

```
| 'Write predictions' >> sink)
```

3. Understand the learning task

The Molecules code sample extracts files that contain molecular data and counts the number of carbon, hydrogen, oxygen, and nitrogen atoms are in each molecule. Then, the code normalizes the counts to values between 0 and 1, and feeds the values into a TensorFlow Deep Neural Network estimator. The Estimator uses the training dataset to train the model, and then uses the evaluation dataset to verify that the model accurately predicts molecular energy given some of the molecule's properties.

4. Understand the contents of each script in the pipeline

- `data-extractor.py`: Downloads the SDF files.

- `preprocess.py`: Parses the SDF files to then count how many Carbon, Hydrogen, Oxygen and Nitrogen atoms a molecule has and then normalizes that values (from 0 to 1). Uses `tf.Transform` to find the minimum and maximum number of counts (full pass over the dataset). The process up to this phase can be nominated **Feature Extraction**. Splits the dataset into training (80%) and test (20%).
- `trainer/task.py`: Loads the data that was processed in the preprocessing phase and then uses the training dataset to train the model, and then uses the evaluation dataset to verify that the model accurately predicts molecular energy given some of the molecule's properties.
- `predict.py`: Provide the model with inputs and it will make predictions. The pipeline can act as either a **batch pipeline** or a **streaming pipeline**. Batch pipeline is indicated when there are a large amount of predictions and the user can wait for all of them to finish. Streaming pipeline is indicated when the User is sending sporadic predictions and wants to get the results as soon as possible. The batch and streaming pipeline differ on the source and sink interactions.
- `publisher.py`: Parses SDF files from a directory and publishes them to the inputs topic.
- `subscriber.py`: Listens for prediction results and logs them.

5. Understand how to use transformations and estimators in Tensorflow

TensorFlow Transformations are great for preprocessing input data for TensorFlow, including creating features that require a full pass over the training dataset.

In the molecules example a Transform is used to make a full pass over the dataset and find the maximum and minimum count of molecules, using then the computed values to normalize the input data.

Tensroflow Estimators base themselves on the Estimator class, which wraps a model which is specified by a model function, which, given inputs and a number of other parameters, returns the operations necessary to perform training, evaluation, or predictions.

Estimators encapsulate the following actions:

- training
- evaluation
- prediction
- export for serving

6. Run the pipeline as is locally (run-local) and in the cloud (run-cloud) (are there any differences in performance?)

For smaller datasets, running them locally is much faster than running them on the cloud (makes sense as the communicaton with the cloud induces overhead). However, to larger datasets, the local machine becomes slower than the cloud, since the cloud resources surpass the local machine resources.

7. Vary the max-data-files parameter with values 10, 100, 1000

- `./run_local --max-data-files 10`

```
{"id": 100001, "predictions": [52.15007781982422]}
{"id": 100003, "predictions": [85.49195098876953]}
{"id": 100004, "predictions": [69.68114471435547]}
{"id": 100005, "predictions": [55.056121826171875]}
{"id": 100006, "predictions": [62.0178108215332]}
{"id": 100007, "predictions": [76.23307800292969]}
{"id": 100008, "predictions": [27.15519905090332]}
{"id": 100010, "predictions": [35.23750305175781]}
{"id": 100015, "predictions": [35.748111724853516]}
{"id": 100016, "predictions": [33.944026947021484]}
```


- `./run-local --max-data-files 100`


```
{"id": 375001, "predictions": [64.0418701171875]}
{"id": 375002, "predictions": [58.79254913330078]}
{"id": 375003, "predictions": [69.9582748413086]}
{"id": 375004, "predictions": [71.39855194091797]}
{"id": 375005, "predictions": [45.28623580932617]}
{"id": 375006, "predictions": [45.28623580932617]}
{"id": 375007, "predictions": [45.28623580932617]}
{"id": 375008, "predictions": [49.62617874145508]}
{"id": 375009, "predictions": [49.62617874145508]}
{"id": 375010, "predictions": [53.8374137878418]}
```


- `./run-local --max-data-files 1000`

8. Modify this program to include the actual ENERGY of each molecule in the predictions file

Changes on prediction.py

```
yield {
 'id': inputs[self.id_key],
 'predictions': results[self.meta_predictions][0].tolist(),
 'actualEnergy': inputs['Energy'] # New Entry
 }
```

Output:


```
{"id": 25002, "actualEnergy": 33.6141, "predictions": [24.72126007080078]}
{"id": 25003, "actualEnergy": 11.5619, "predictions": [8.44851016998291]}
{"id": 25004, "actualEnergy": 28.495, "predictions": [24.67938995361328]}
{"id": 25005, "actualEnergy": 19.6646, "predictions": [14.54765510559082]}
{"id": 25006, "actualEnergy": 10.6814, "predictions": [9.66574478149414]}
{"id": 25008, "actualEnergy": 20.2605, "predictions":
[14.377355575561523]}
{"id": 25009, "actualEnergy": 16.3172, "predictions": [16.26948356628418]}
```

```
{"id": 25010, "actualEnergy": 14.0353, "predictions":
[27.049715042114258]}
{"id": 25011, "actualEnergy": 18.0687, "predictions": [8.839911460876465]}
{"id": 25012, "actualEnergy": 74.887, "predictions": [55.38291549682617]}
```

9. Modify this program to allow for cross-validation

First, lets introduce the optional argument variable `num_splits`. This variable is naturally related to the `eval_percent` variable.

```
def run(
    ...
    num_splits=5,
    eval_percent=20.0,
    ...):
```

Then, we must alter the section of the code responsible for the Split of the dataset into a training set and an evaluation set. Previous dataset divison:

```
train_dataset, eval_dataset = (
    dataset
    | 'Split dataset' >> beam.Partition(
        lambda elem, _: int(random.uniform(0, 100) < eval_percent), 2))
```

Dataset division that allows cross-validation:

```
splits = (
    dataset
    | 'Split dataset for cross validation' >> beam.Partition(
        lambda elem, _: int(random.uniform(0, num_splits)), num_splits))
```

Next we must run the algorithm with different possibilities of training dataset and evaluation dataset resulting from the splits:

```
for split in splits:
    train_dataset = #Union of all the PCollections of splits except split
    eval_dataset = split

    ...
```

Helpful links:

- https://cloud.google.com/dataflow/docs/samples/molecules-walkthrough
- https://www.tensorflow.org/tfx/transform/tutorials/TFT_simple_example
- https://www.tensorflow.org/tutorials/estimators/linear
- https://www.tensorflow.org/guide/estimators
- https://beam.apache.org/documentation/pipelines/create-your-pipeline/