



Build the Model



Advanced ML with TensorFlow on GCP

End-to-End Lab on Structured Data ML

Production ML Systems

Image Classification Models

Sequence Models

Recommendation Systems

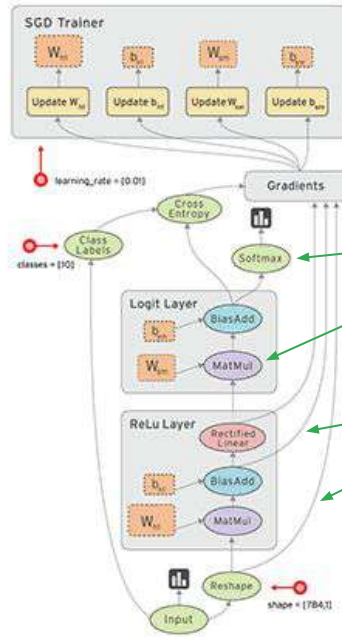


Steps involved in doing ML on GCP

- 1 Explore the dataset
- 2 Create the dataset
- 3 **Build the model**
- 4 Operationalize the model



TensorFlow is an open-source high-performance library for numerical computation that uses directed graphs



Nodes represent mathematical operations.

Edges represent arrays of data.



A tensor is an N-dimensional array of data



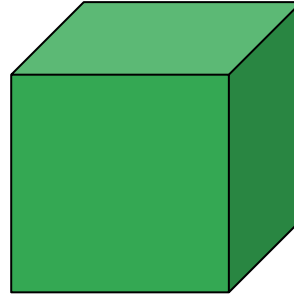
Rank 0
Tensor
scalar



Rank 1
Tensor
vector



Rank 2
Tensor
matrix



Rank 3
Tensor



Rank 4
Tensor



TensorFlow toolkit hierarchy

tf.estimator	High-level API for distributed training			
tf.layers, tf.losses, tf.metrics	Components useful when building custom NN models			
Core TensorFlow (Python)	Python API gives you full control			
Core TensorFlow (C++)	C++ API is quite low level			
CPU	GPU	TPU	Android	TF runs on different hardware



Cloud ML Engine



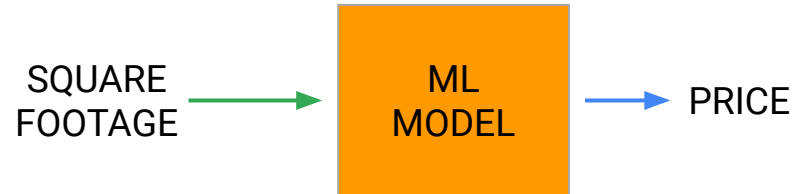
Working with Estimator API

Set up machine learning model:

- 1 Regression or classification?
- 2 What is the label?
- 3 What are the features?

Carry out ML steps:

- 1 Train the model.
- 2 Evaluate the model.
- 3 Predict with the model.



Structure of an Estimator API ML model

```
import tensorflow as tf
#Define input feature columns
featcols = [
    tf.feature_column.numeric_column("sq_footage") ]
```

```
#Instantiate Linear Regression Model
model = tf.estimator.LinearRegressor(featcols, './model_trained')
```

```
#Train
def train_input_fn():
    ...
    return features, labels
model.train(train_input_fn, steps=100)
```

```
#Predict
def pred_input_fn():
    ...
    return features
out = model.predict(pred_input_fn)
```

SQUARE
FOOTAGE

ML
MODEL

PRICE



Encoding categorical data to supply to a DNN

1a. If you know the complete vocabulary beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('zipcode',  
    vocabulary_list = ['83452', '72345', '87654', '98723', '23451']),
```

1b. If your data is already indexed; i.e., has integers in [0-N):

```
tf.feature_column.categorical_column_with_identity('stateId',  
    num_buckets = 50)
```

2. To pass in a categorical column into a DNN, one option is to one-hot encode it:

```
tf.feature_column.indicator_column( my_categorical_column )
```



To read CSV files, create a TextLineDataset giving it a function to decode the CSV into features, labels

```
CSV_COLUMNS = ['sqfootage', 'city', 'amount']
LABEL_COLUMN = 'amount'
DEFAULTS = [[0.0], ['na'], [0.0]]

def read_dataset(filename, mode, batch_size=512):
    def decode_csv(value_column):
        columns = tf.decode_csv(value_column, record_defaults=DEFAULTS)
        features = dict(zip(CSV_COLUMNS, columns))
        label = features.pop(LABEL_COLUMN)
        return features, label

    dataset = tf.data.TextLineDataset(filename).map(decode_csv)

    ...
    return ...
```



Shuffling is important for distributed training

```
def read_dataset(filename, mode, batch_size=512):  
    ...  
    dataset = tf.data.TextLineDataset(filename).map(decode_csv)  
    if mode == tf.estimator.ModeKeys.TRAIN:  
        num_epochs = None # indefinitely  
        dataset = dataset.shuffle(buffer_size=10*batch_size)  
    else:  
        num_epochs = 1 # end-of-input after this  
        dataset = dataset.repeat(num_epochs).batch(batch_size)  
  
    return dataset.make_one_shot_iterator().get_next()
```



Estimator API comes with a method that handles distributed training and evaluation

```
estimator = tf.estimator.LinearRegressor(  
    model_dir=output_dir,  
    feature_columns=feature_cols)  
  
...  
  
tf.estimator.train_and_evaluate(estimator,  
    train_spec,  
    eval_spec)
```

PASS IN:

1. ESTIMATOR
2. TRAIN SPEC
3. EVAL SPEC

Distribute the graph

Share variables

Evaluate occasionally

Handle machine failures

Create checkpoint files

Recover from failures

Save summaries for TensorBoard



TrainSpec consists of the things that used to be passed into the train() method

```
train_spec = tf.estimator.TrainSpec(  
    input_fn=read_dataset('gs://.../train*'),  
    mode=tf.contrib.learn.ModeKeys.TRAIN),  
    max_steps=num_train_steps)  
...  
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

Think “steps”, not “epochs,” with production-ready, distributed models.

1. Gradient updates from slow workers could get ignored.
2. When retraining a model with fresh data, we'll resume from earlier number of steps (and corresponding hyper-parameters).



EvalSpec controls the evaluation and the checkpointing of the model because they happen at the same time

```
exporter = ...
eval_spec=tf.estimator.EvalSpec(
    input_fn=read_dataset('gs://.../valid*',
                           mode=tf.contrib.learn.ModeKeys.EVAL),
    steps=None,
    start_delay_secs=60, # start evaluating after N seconds
    throttle_secs=600,  # evaluate every N seconds
    exporters=exporter)

tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```



Lab

Creating a TensorFlow model

In this lab, you use the Estimator API to build linear and deep neural network models, use the Estimator API to build wide and deep model, and monitor training using TensorBoard.



Two types of features: Dense and sparse

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  → "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    → "employeeId": 72365,  
    → "waitTime": 1.4,  
    "customerRating": 4  
  },  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7  
  }  
},
```

price	8345	72345	87654	98723	wait
2.50	0	1	0	0	1.4
↑	↑	↑	↑	↑	↑

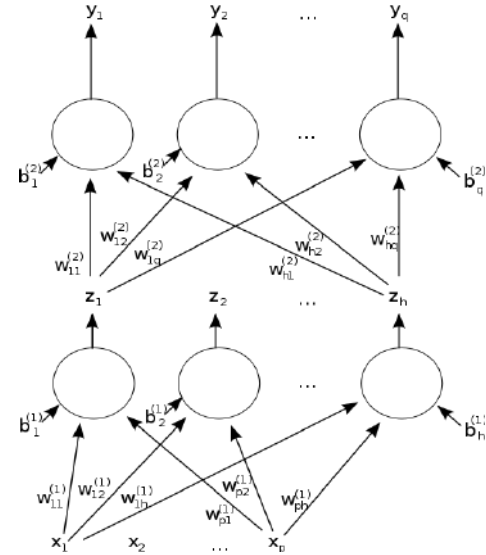


DNNs good for dense, highly-correlated inputs

pixel_values (



)



1024² input
nodes

10 hidden nodes
→ 10 image
features

https://commons.wikimedia.org/wiki/File:Two_layer_ann.svg

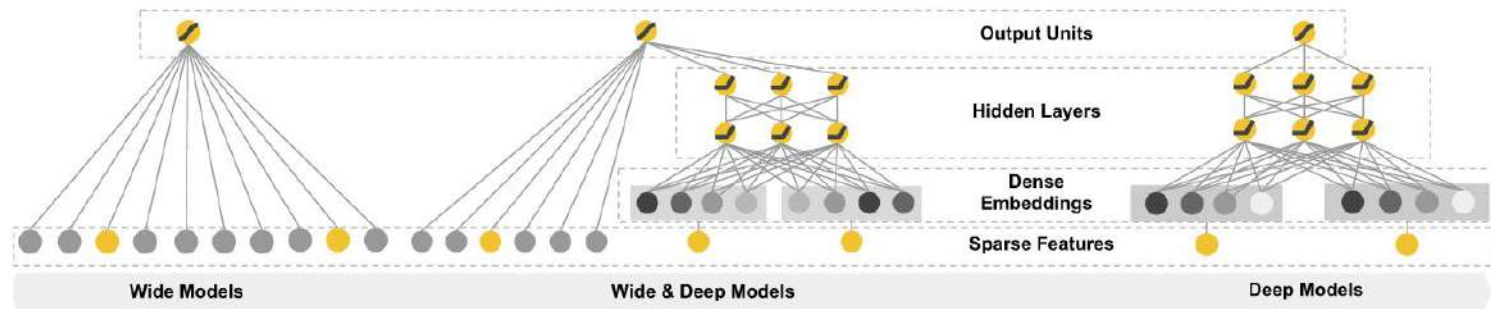


Linear models are better at handling sparse, independent features

```
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```



Wide-and-deep models let you handle both



Wide and Deep

memorization
relevance



generalization
diversity

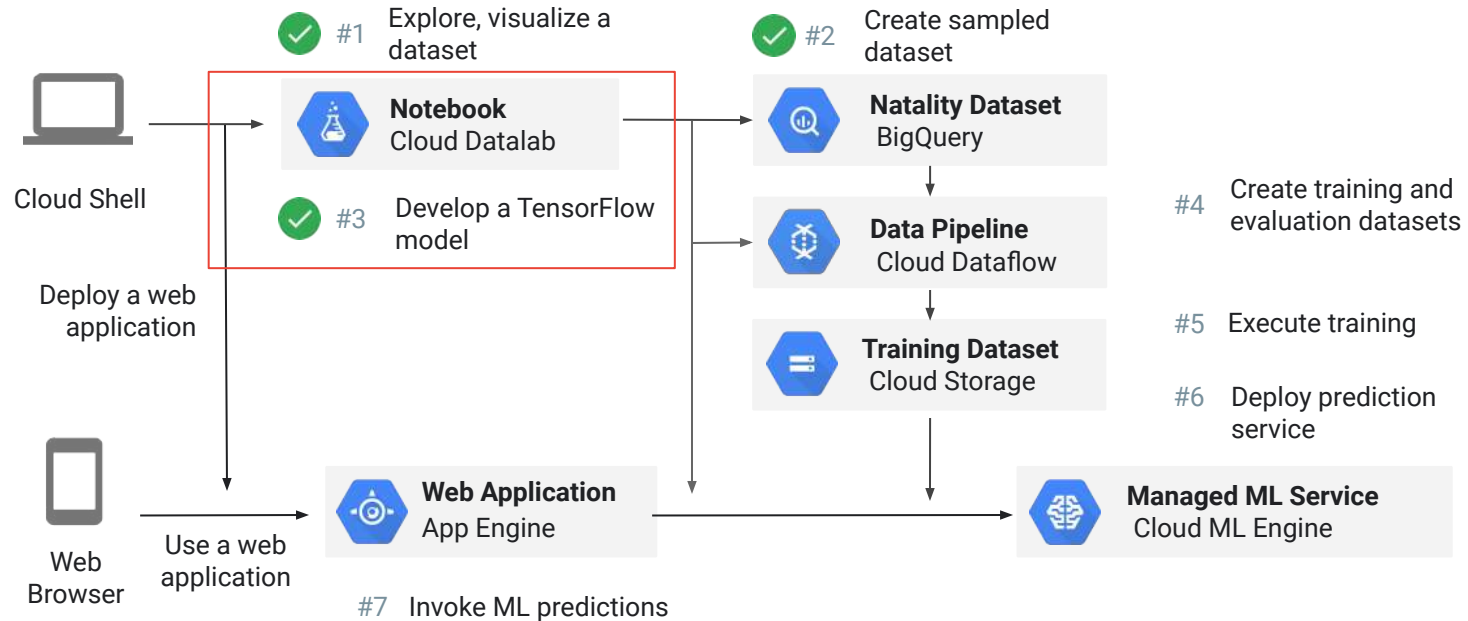


Wide-and-deep network in Estimator API

```
model = tf.estimator.DNNLinearCombinedClassifier(  
    model_dir=...,  
    linear_feature_columns=wide_columns,  
    dnn_feature_columns=deep_columns,  
    dnn_hidden_units=[100, 50])
```



The end-to-end process



cloud.google.com

