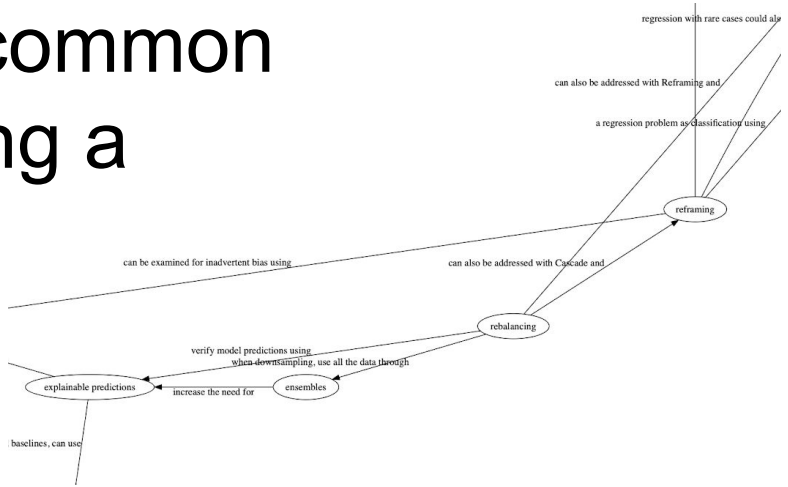
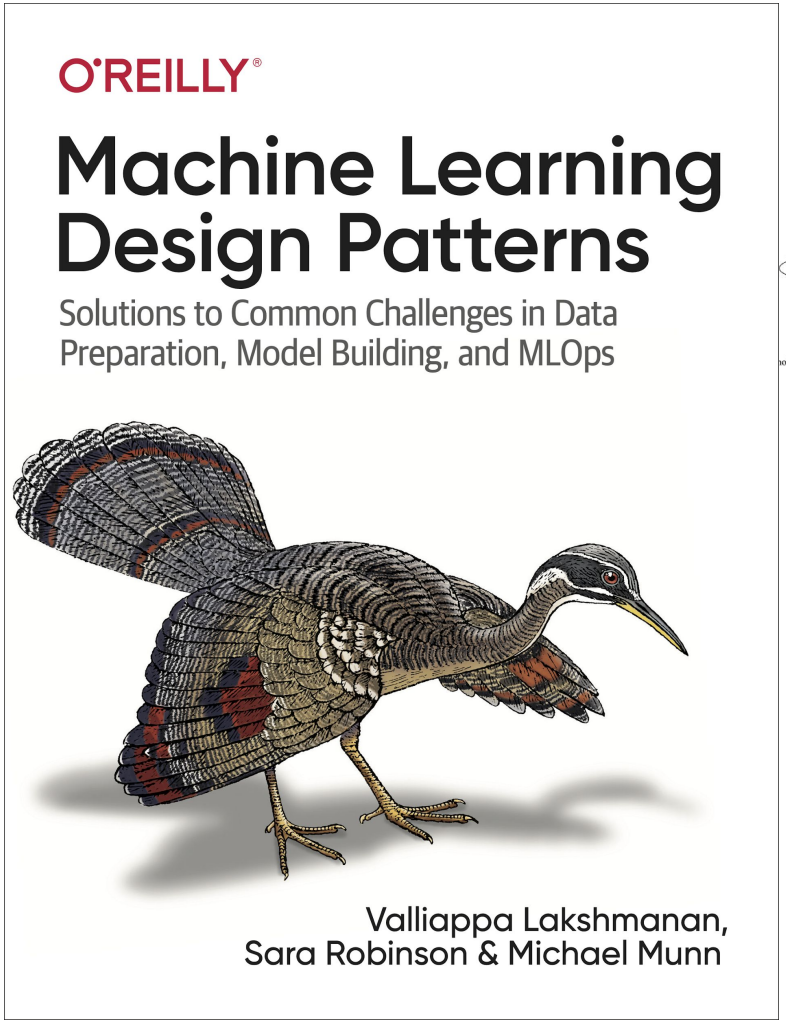
ML Design Patterns for ML developers

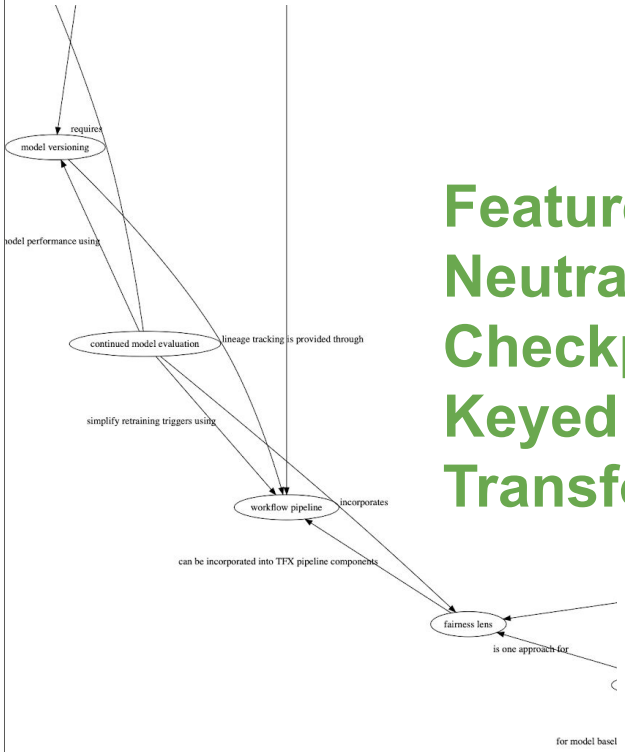
Valliappa (Lak) Lakshmanan
Director of Analytics & AI Solutions,
Google Cloud
Twitter: @lak_gcp
lak@google.com

Google Cloud

Design patterns are formalized best practices to solve common problems when designing a software system.





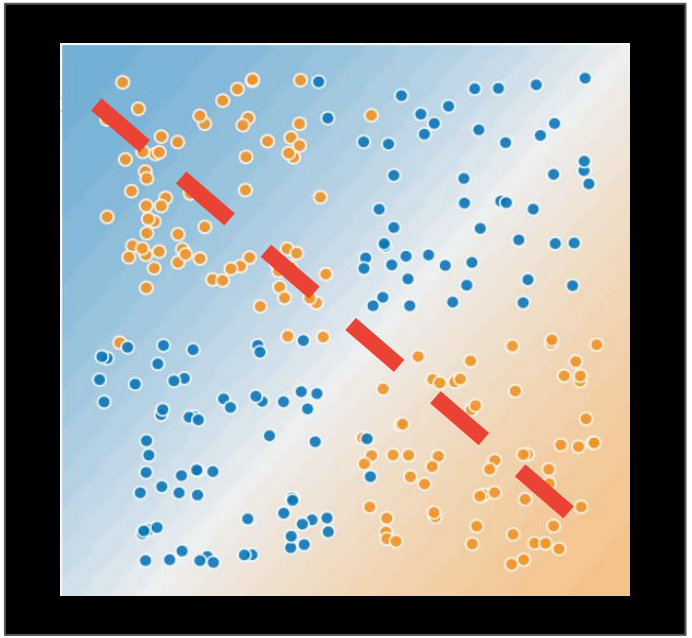


Feature Cross
Neutral Class
Checkpoints
Keyed Predictions
Transform

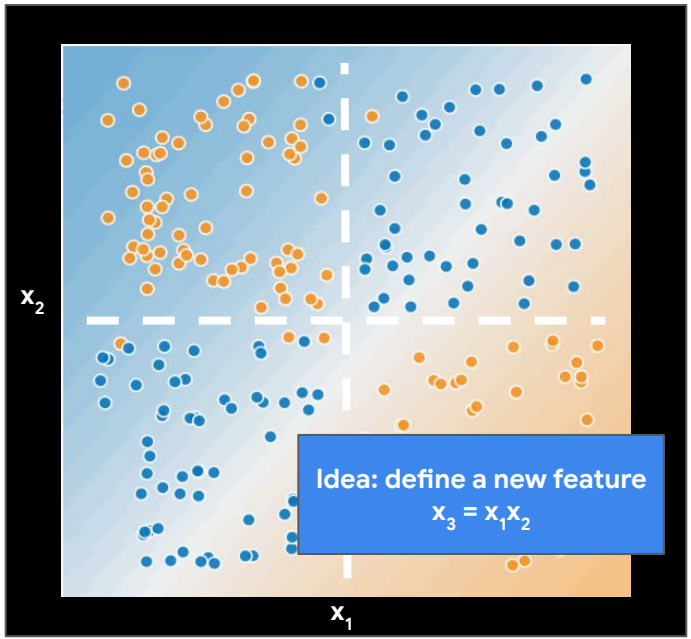
https://github.com/GoogleCloudPlatform/ml-design-patterns

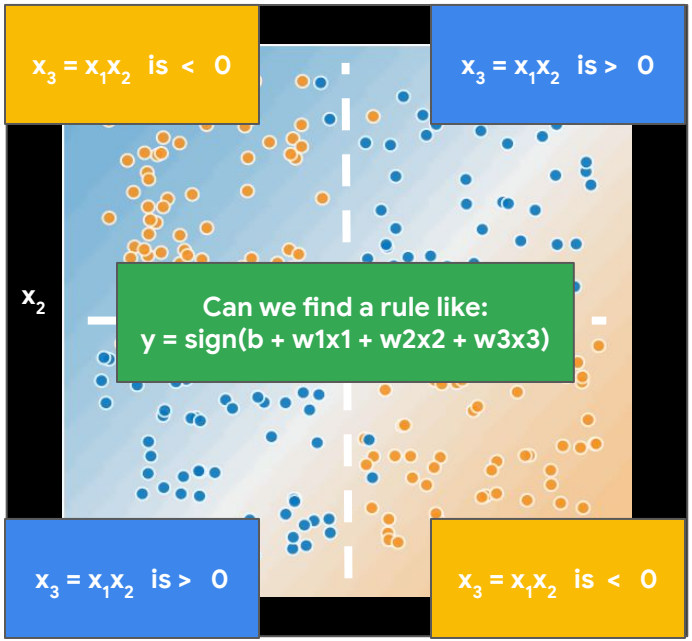
Feature Cross

Can you draw a line to separate the two classes in this problem?

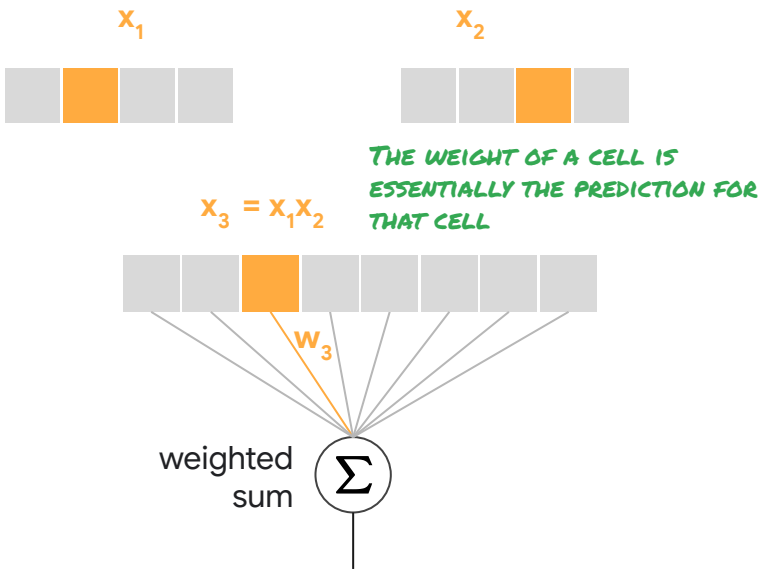


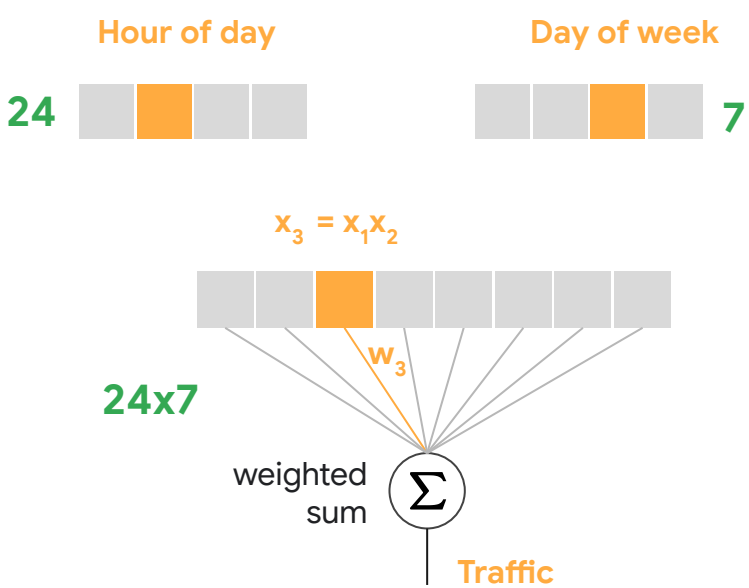
The feature cross provides a way to combine features in a linear model





Using feature crosses, you get to treat each scenario independently





Implementing a feature cross

1

```
CONCAT(CAST(day_of_week AS STRING),
       CAST(hour_of_week AS STRING))
  AS day_X_hour
```

2

Bucketize Numeric features before crossing:

```
# Create a bucket feature column for latitude.
latitude_as_numeric = fc.numeric_column("latitude")
lat_bucketized = fc.bucketized_column(latitude_as_numeric,
                                      lat_boundaries)
# Create a bucket feature column for longitude.
longitude_as_numeric = fc.numeric_column("longitude")
lon_bucketized = fc.bucketized_column(longitude_as_numeric,
                                      lon_boundaries)

# Create a feature cross of latitude and longitude
lat_x_lon = fc.crossed_column([lat_bucketized, lon_bucketized],
                              hash_bucket_size=nbuckets**4)

crossed_feature = fc.indicator_column(lat_x_lon)
```

3

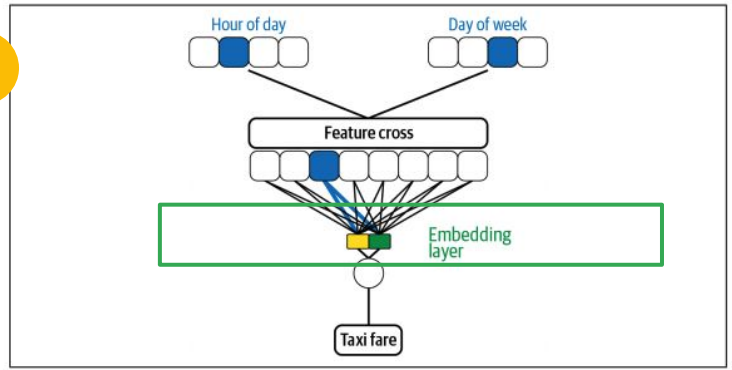


Figure 2-18. An embedding layer is a useful way to address the sparsity of a feature cross.

4

```
OPTIONS(input_label_cols=['fare_amount'],
        model_type='linear_reg', l2_reg=0.1)
AS
SELECT * FROM mlpatterns.taxi_data
```

There are two feature crosses here: one in time (of day of week and hour of day) and the other in space (of the pickup and dropoff locations). The location, in particular, is very high cardinality, and it is likely that some of the buckets will have very few examples.

For this reason, it is advisable to pair feature crosses with L1 regularization, which encourages sparsity of features, or L2 regularization, which limits overfitting. This

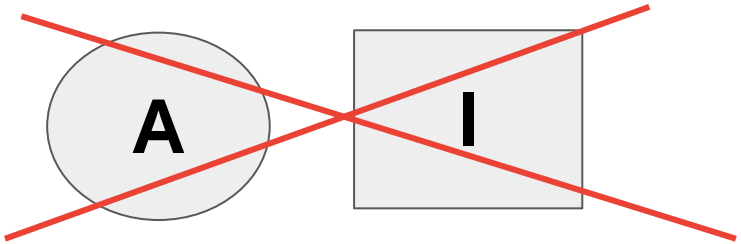
Neutral Class

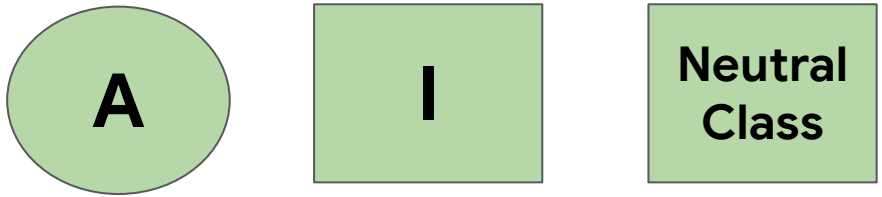
Imagine you are training a model that provides guidance on pain relievers

In historical dataset ...

If patient has risk of stomach problem	Acetaminophen
If patient has risk of liver damage	Ibuprofen
Otherwise	Pretty arbitrary

Change the problem from binary classification





provided there is random assignment at work, the Neutral Class design pattern can help us avoid losing model accuracy because of arbitrarily labeled data.

Problem framing has to be done when data is collected

Imagine a different scenario. Suppose the electronic record that captures the doctor's prescriptions also asks them whether the alternate pain medication would be acceptable. If the doctor prescribes acetaminophen, the application asks the doctor whether the patient can use ibuprofen if they already have it in their medicine cabinet.

Based on the answer to the second question, we have a neutral class. The prescription might still be written as "acetaminophen," but the record captures that the doctor was neutral for this patient. Note that this fundamentally requires us to design the data collection appropriately—we cannot manufacture a neutral class after the fact. We have to correctly design the machine learning problem. Correct design, in this case, starts with how we pose the problem in the first place.

Example: does a baby need immediate attention?

```
CREATE OR REPLACE MODEL mlpatterns.neutral_2classes
OPTIONS(model_type='logistic_reg', input_label_cols=['health']) AS

SELECT
  IF(apgar_1min >= 9, 'Healthy', 'NeedsAttention') AS health,
  plurality,
  mother_age,
  gestation_weeks,
  ever_born
FROM `bigquery-public-data.samples.natality`
WHERE apgar_1min <= 10
```

BINARY CLASSIFICATION

We are thresholding the Apgar score at 9 and treating babies whose Apgar score is 9 or 10 as healthy, and babies whose Apgar score is 8 or lower as requiring attention. The accuracy of this binary classification model when trained on the natality dataset and evaluated on held-out data is 0.56.

Yet, assigning an Apgar score involves a number of relatively subjective assessments, and whether a baby is assigned 8 or 9 often reduces to matters of physician preference. Such babies are neither perfectly healthy, nor do they need serious medical intervention. What if we create a neutral class to hold these "marginal" scores? This requires creating three classes, with an Apgar score of 10 defined as healthy, scores of 8 to 9 defined as neutral, and lower scores defined as requiring attention:

```
CREATE OR REPLACE MODEL mlpatterns.neutral_3classes
OPTIONS(model_type='logistic_reg', input_label_cols=['health']) AS

SELECT
  IF(apgar_1min = 10, 'Healthy',
     IF(apgar_1min >= 8, 'Neutral', 'NeedsAttention')) AS health,
  plurality,
  mother_age,
  gestation_weeks,
  ever_born
FROM `bigquery-public-data.samples.natality`
WHERE apgar_1min <= 10
```

NEUTRAL CLASS

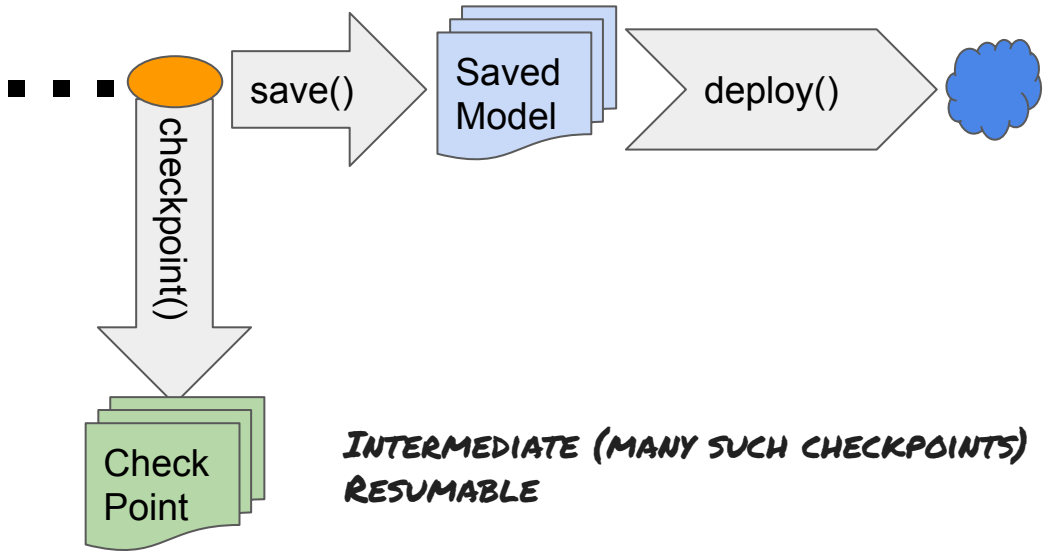
This model achieves an accuracy of 0.79 on a held-out evaluation dataset, much higher than the 0.56 that was achieved with two classes.

Other situations where Neutral Class comes in handy

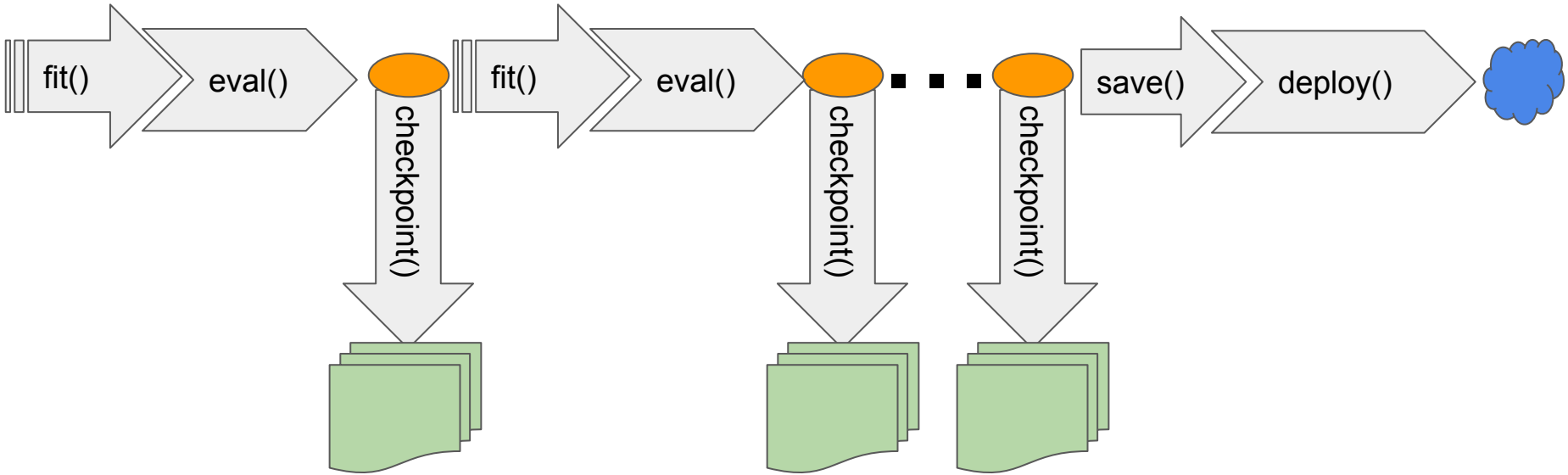
1. Disagreement among human experts
2. Customer satisfaction scores: treat 1-4 as bad, 8-10 as good and 5-7 as neutral
3. Improving embeddings, by training only on confident predictions
4. Improving decisions (eg. stock market trading) by trading only on confident predictions

Checkpoints

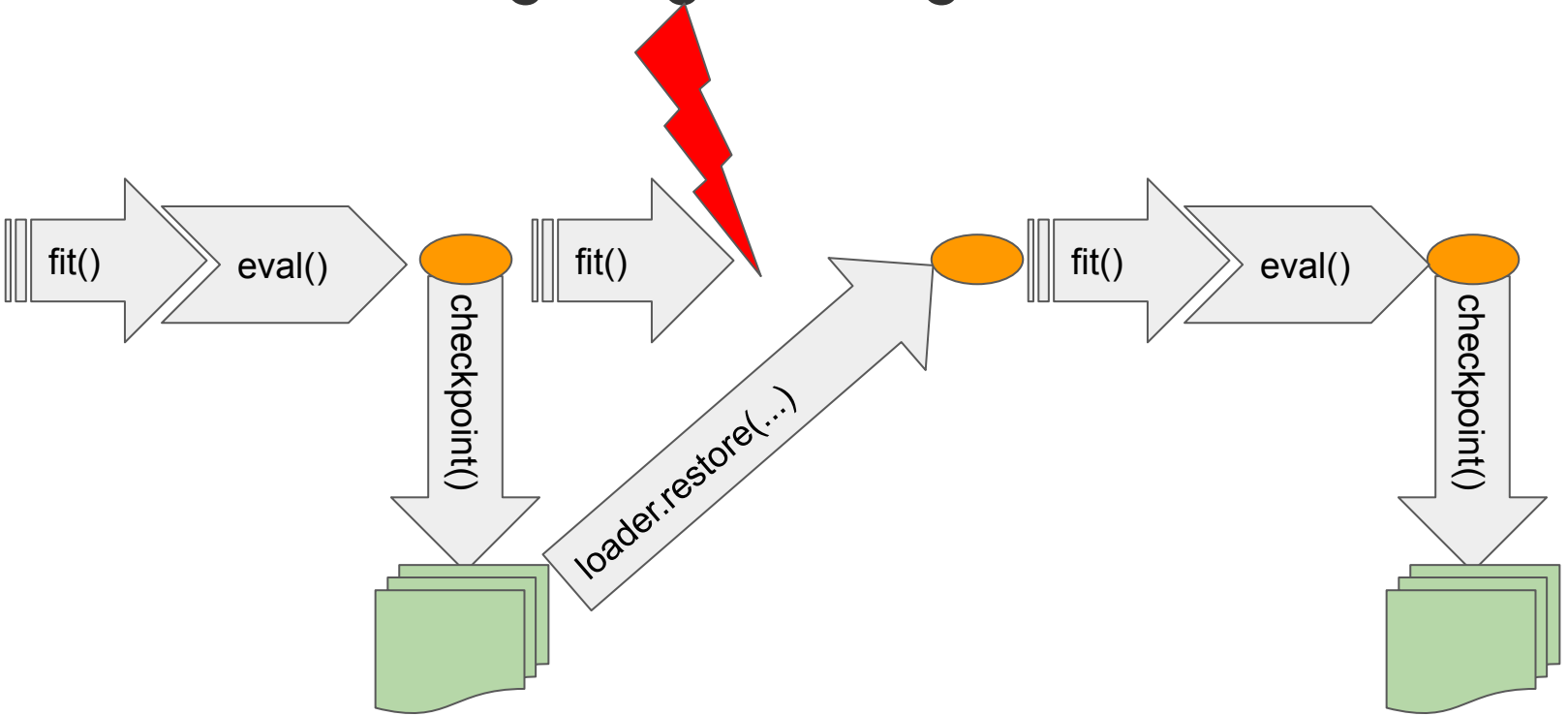
Checkpoints != Saved Models



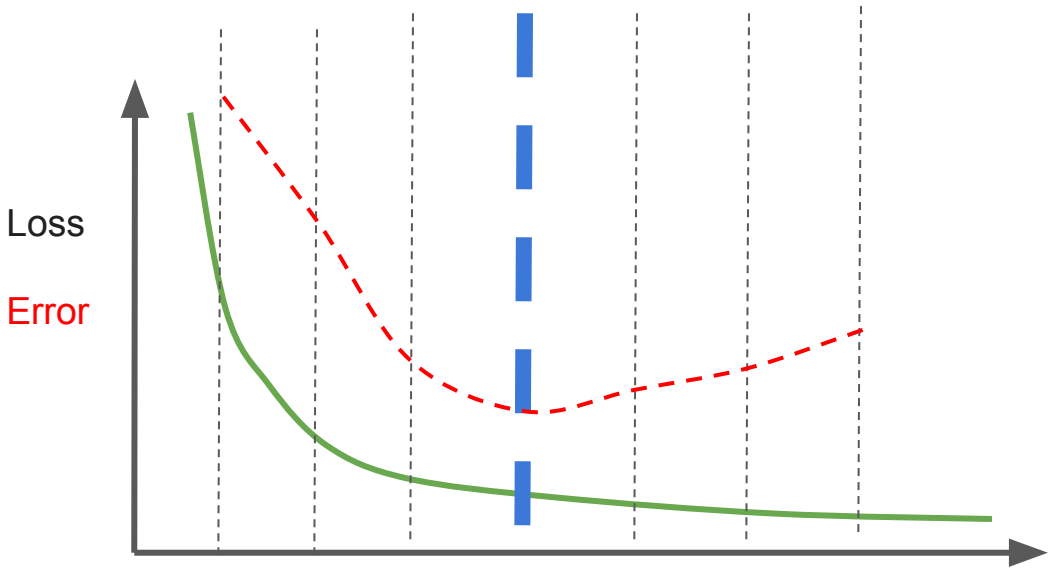
Checkpoints



1. Resilience during long training runs

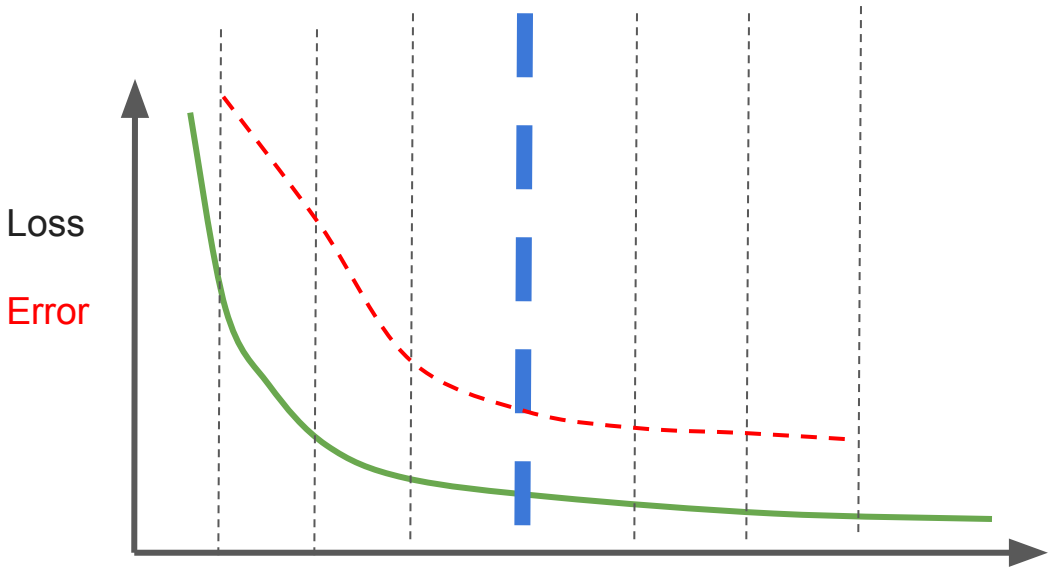


2. Generalization (early stopping)



Better approach is to add regularization

3. Fine-tuning



Checkpointing in Keras

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=NUM_EVALS,
                    steps_per_epoch=steps_per_epoch,
                    verbose=2, # 0=silent, 1=progress bar, 2=one line per epoch
                    callbacks=[cp_callback])
model.save(...)
```

tf.dataset is an out-of-memory iterable

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=15,
                    batch_size=128,
                    callbacks=[cp_callback])
```

But epochs are problematic

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=15,
                    batch_size=128,
                    callbacks=[cp_callback])
```

Integer epochs can be expensive when dataset is millions of examples

Resilience problems when an epoch takes hours to process

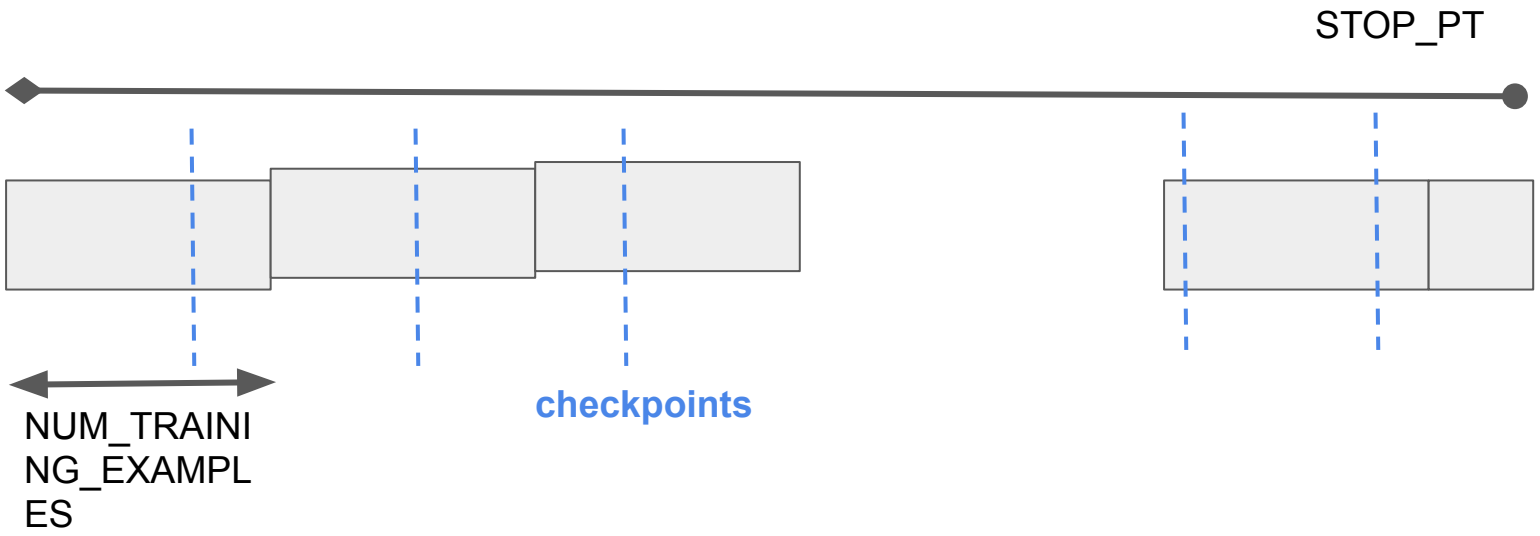
Datasets grow over time.

Datasets grow over time??.

If you get 100,000 more examples and you train the model and get a higher error, is it because you need to do an early stop or is the new data corrupt in some way? You can't tell because the prior training was on 15 million examples and the new one is one is on 16.5 million examples.

Virtual Epochs

```
trainds = trainds.repeat()
```



Keeping number of steps fixed is a partial answer

```
NUM_STEPS = 143000
BATCH_SIZE = 50 # ~~100~~
NUM_CHECKPOINTS = 15
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=NUM_CHECKPOINTS,
                    steps_per_epoch=NUM_STEPS // NUM_CHECKPOINTS,
                    batch_size=BATCH_SIZE,
                    callbacks=[cp_callback])
```

FAILS IF YOU DO HYPER-PARAMETER TUNING OF BATCH SIZE

Solution: Keep as constant the number of examples you show the model

```
NUM_TRAINING_EXAMPLES = 1000 * 1000
STOP_POINT = 14.3
TOTAL_TRAINING_EXAMPLES = int(STOP_POINT * NUM_TRAINING_EXAMPLES)
BATCH_SIZE = 100
NUM_CHECKPOINTS = 15
steps_per_epoch = (TOTAL_TRAINING_EXAMPLES //
                   (BATCH_SIZE*NUM_CHECKPOINTS))
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trainds,
                    validation_data=evalds,
                    epochs=NUM_CHECKPOINTS,
                    steps_per_epoch=steps_per_epoch,
                    batch_size=BATCH_SIZE,
                    callbacks=[cp_callback])
```

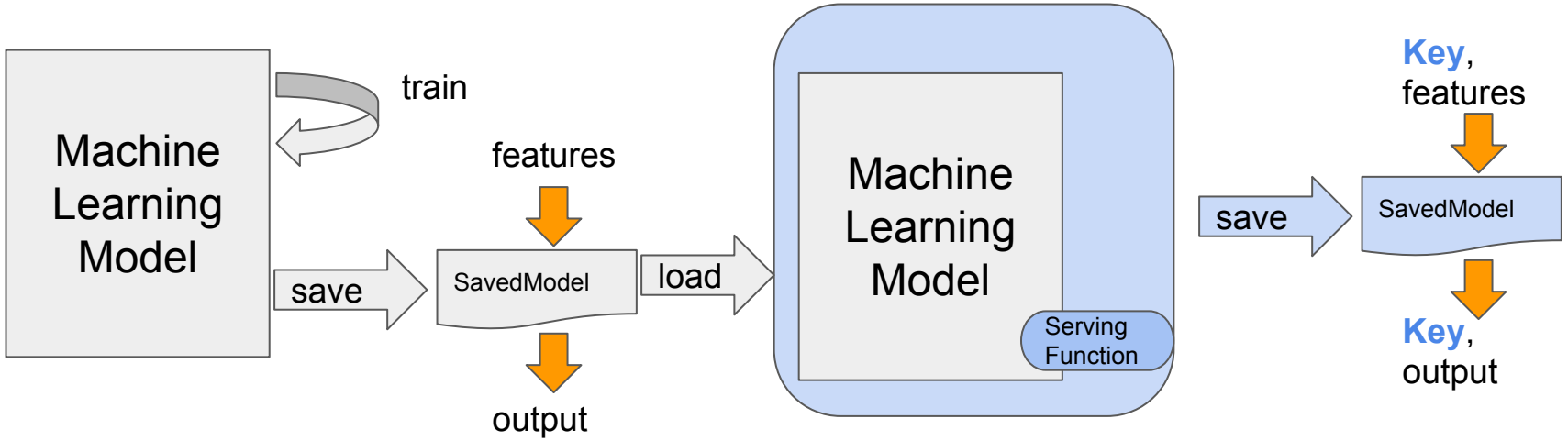
Keyed Predictions

04

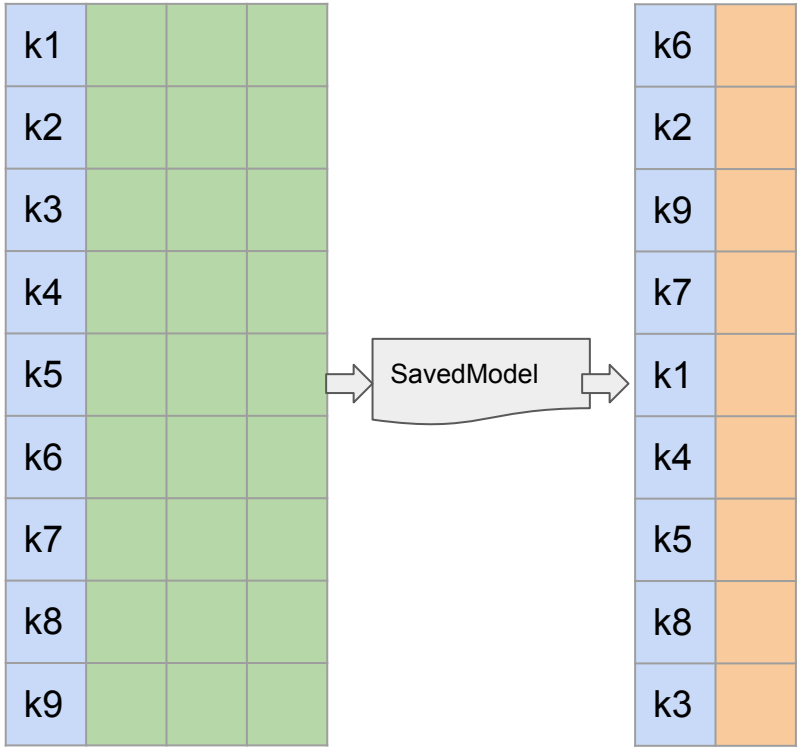
Pass-through keys in Keras

```
# Serving function that passes through keys
@tf.function(input_signature=[{
     'is_male': tf.TensorSpec([None,], dtype=tf.string, name='is_male'),
     'mother_age': tf.TensorSpec([None,], dtype=tf.float32, name='mother_age'),
     'plurality': tf.TensorSpec([None,], dtype=tf.string, name='plurality'),
     'gestation_weeks': tf.TensorSpec([None,], dtype=tf.float32, name='gestation_weeks'),
     'key': tf.TensorSpec([None,], dtype=tf.string, name='key')
}])
def my_serve(inputs):
   feats = inputs.copy()
   key = feats.pop('key')
   output = model(feats)
   return {'key': key, 'babyweight': output}



tf.saved_model.save(model, EXPORT_PATH,
                    signatures={'serving_default': my_serve})
```

Keyed predictions



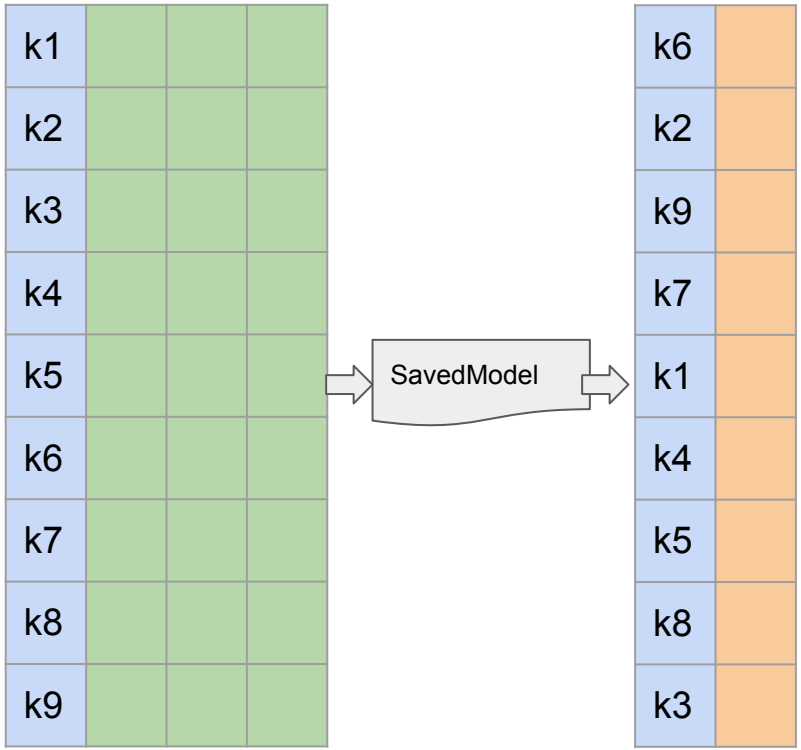
Why?



Batch predictions

Async Predictions

Why?

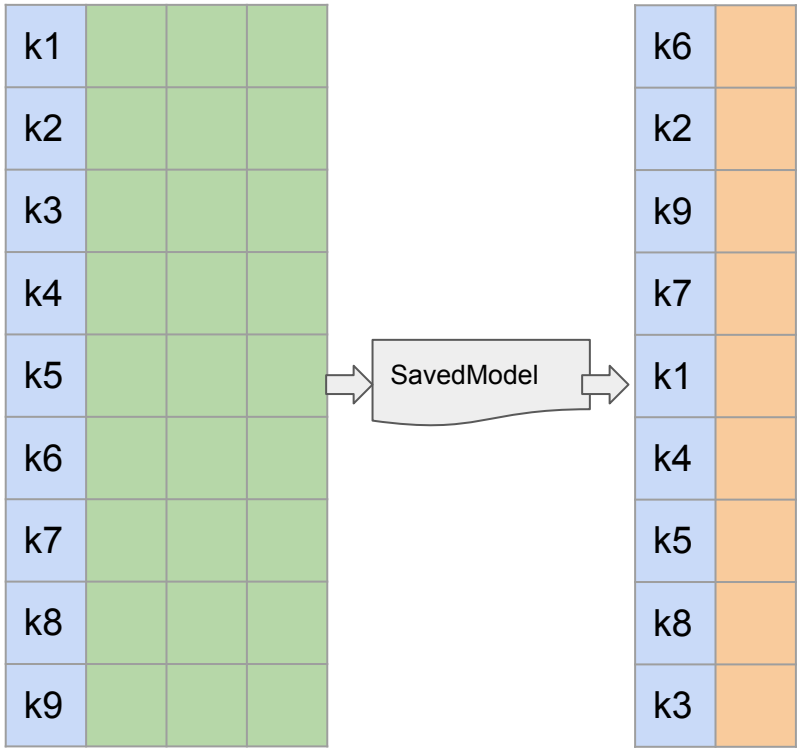


Batch predictions

Async Predictions

But ... why should the client supply the keys?

Why?



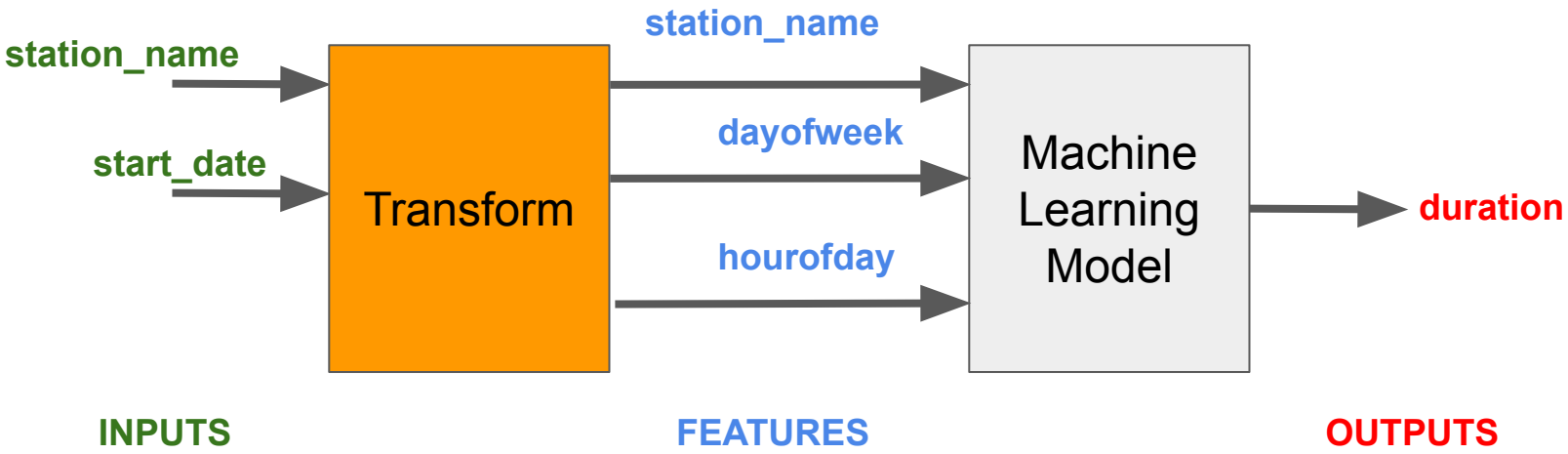
Batch predictions

Async Predictions

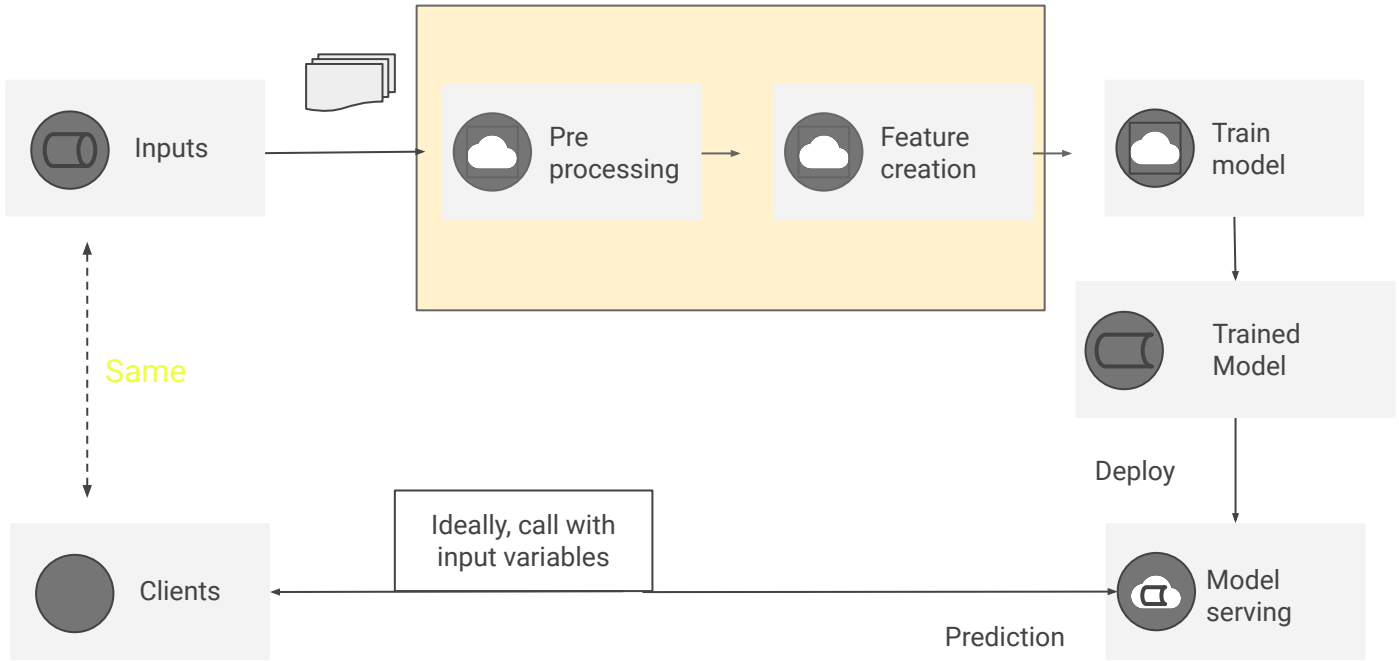
SLICED Evaluation

Transform

Imagine a model to predict the length of rentals

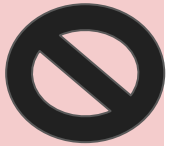


Who does transformation during prediction?



Ideally, client code does not have to know about all the transformations that were carried out

```
CREATE OR REPLACE MODEL ch09edu.bicycle_model
OPTIONS(input_label_cols=['duration'],
        model_type='linear_reg')
AS

SELECT
  duration
  , start_station_name
  , CAST(EXTRACT(dayofweek from start_date) AS STRING)
         as dayofweek
  , CAST(EXTRACT(hour from start_date) AS STRING)
         as hourofday
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
```

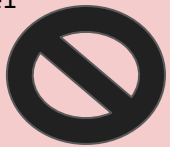


```
SELECT * FROM ML.PREDICT(MODEL ch09edu.bicycle_model,(
  350 AS duration
  , 'Kings Cross' AS start_station_name
  , '3' as dayofweek
  , '18' as hourofday
))
```

Leading cause of training-serving skew

TRANSFORM ensures transformations are automatically applied during ML.PREDICT

```
CREATE OR REPLACE MODEL ch09edu.bicycle_model
OPTIONS(input_label_cols=['duration'],
        model_type='linear_reg')
AS

SELECT
  duration
  , start_station_name
  , CAST(EXTRACT(dayofweek from start_date) AS STRING)
         as dayofweek
  , CAST(EXTRACT(hour from start_date) AS STRING)
         as hourofday
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
```

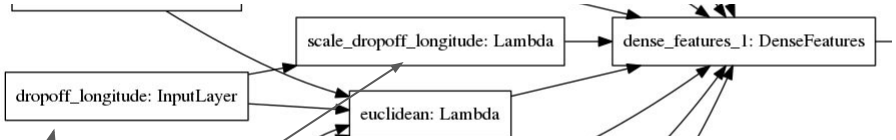


```
SELECT * FROM ML.PREDICT(MODEL ch09edu.bicycle_model,(
  350 AS duration
  , 'Kings Cross' AS start_station_name
  , '3' as dayofweek
  , '18' as hourofday
))
```

```
CREATE OR REPLACE MODEL ch09edu.bicycle_model
OPTIONS(input_label_cols=['duration'],
        model_type='linear_reg')
TRANSFORM(
  SELECT * EXCEPT(start_date)
  , CAST(EXTRACT(dayofweek from start_date) AS STRING)
         as dayofweek
  , CAST(EXTRACT(hour from start_date) AS STRING)
         as hourofday
)
AS
SELECT
  duration, start_station_name, start_date
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
```

```
SELECT * FROM ML.PREDICT(MODEL ch09edu.bicycle_model,(
  350 AS duration
  , 'Kings Cross' AS start_station_name
  , CURRENT_TIMESTAMP() as start_date
))
```

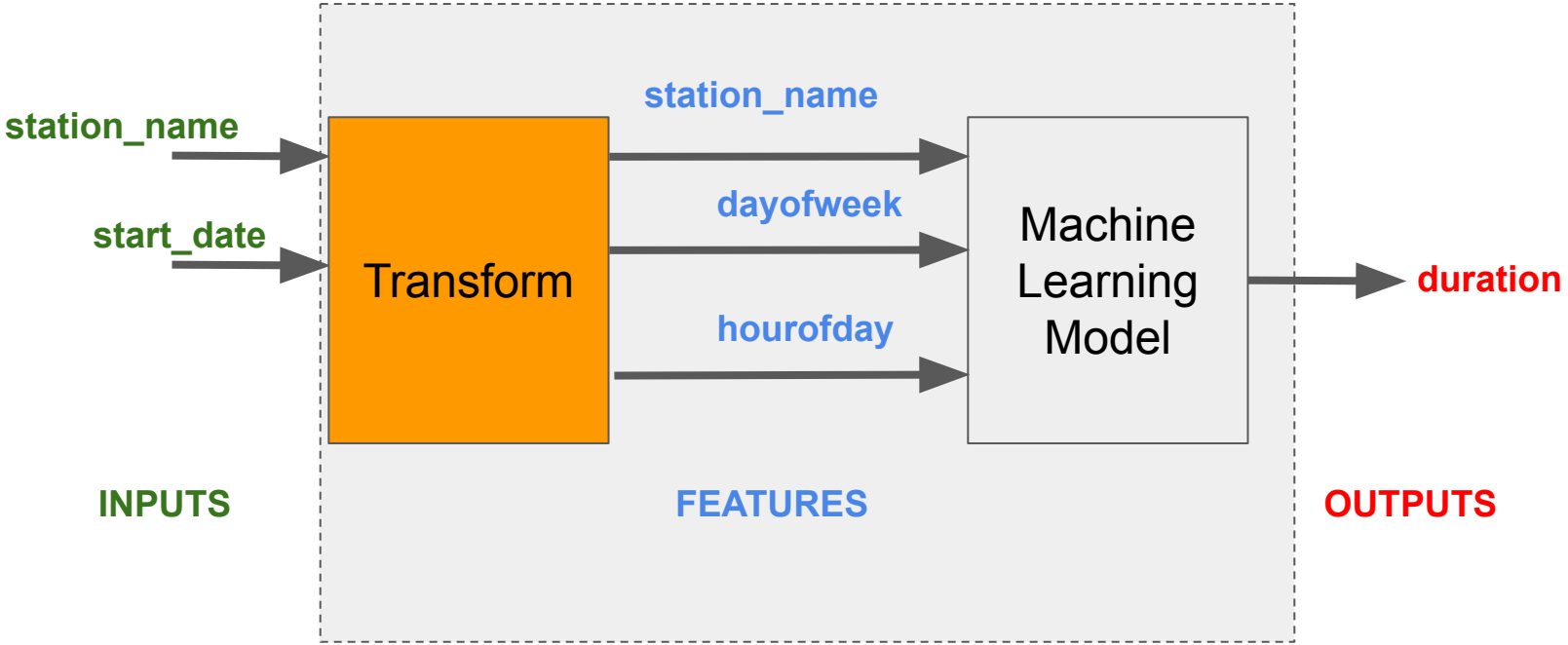
In TensorFlow/Keras, do transformations in Lambda Layers so that they are part of the model graph




```
for lon_col in ['pickup_longitude', 'dropoff_longitude']:  # in range -70 to -78
        transformed[lon_col] = tf.keras.layers.Lambda(
            lambda x: (x+78)/8.0,
            name='scale_{}'.format(lon_col)
        )(inputs[lon_col])
```

Moving an ML model to production is much easier if you keep inputs, features, and transforms separate

Transform pattern: the model graph should include the transformations

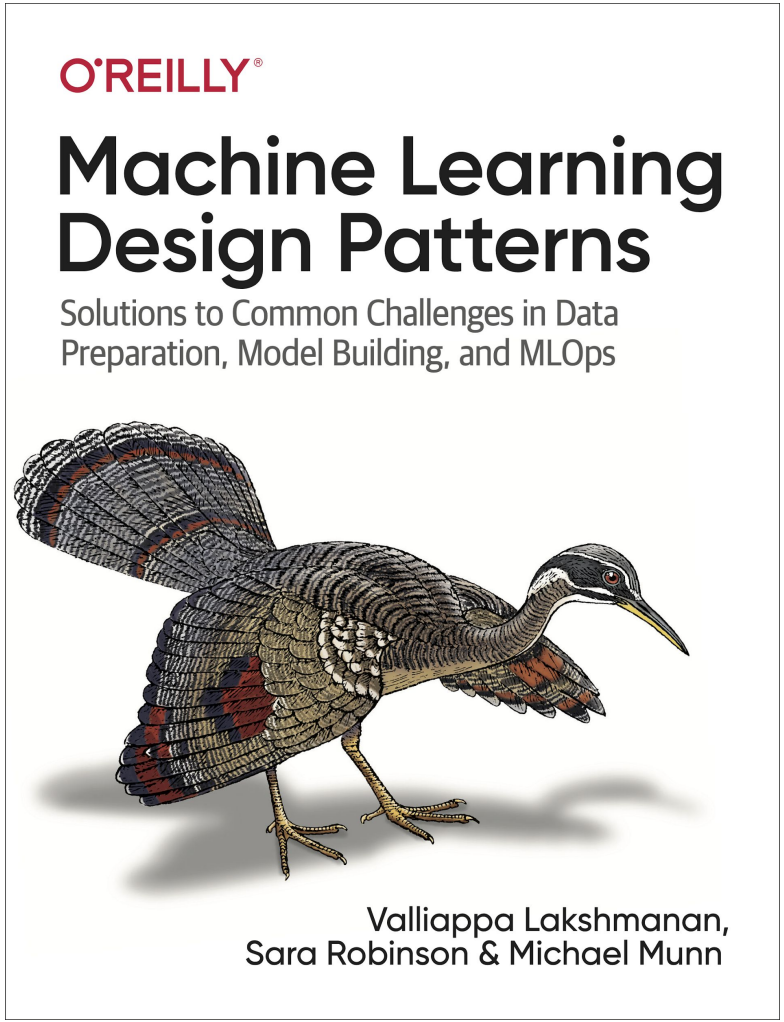


Summary

06

Summary

Chapter	Design pattern	Problem solved	Solution
Data Representation	Feature Cross	Model complexity insufficient to learn feature relationships	Help models learn relationships between inputs faster by explicitly making each combination of input values a separate feature
Problem Representation	Neutral Class	The class label for some subset of examples is essentially arbitrary.	Introduce an additional label for a classification model, disjoint from the current labels
Patterns that Modify Model Training	Checkpoints	Lost progress during long running training jobs due to machine failure	Store the full state of the model periodically, so that partially trained models are available and can be used to resume training from an intermediate point, instead of starting from scratch
Resilience	Keyed Predictions	How to map the model predictions that are returned to the corresponding model input when submitting large prediction jobs	Allow the model to pass through a client-supported key during prediction that can be used to join model inputs to model predictions
Reproducibility	Transform	The inputs to a model must be transformed to create the features the model expects and that process must be consistent between training and serving	Explicitly capture and store the transformations applied to convert the model inputs into features



Read the book
https://bit.ly/ml-design-patterns

Follow us on Twitter:
@lak_gcp
@SRobTweets

Read our blogs:
https://medium.com/@lakshmanok
https://sararobinson.dev/

https://github.com/GoogleCloudPlatform/ml-design-patterns

Thank you