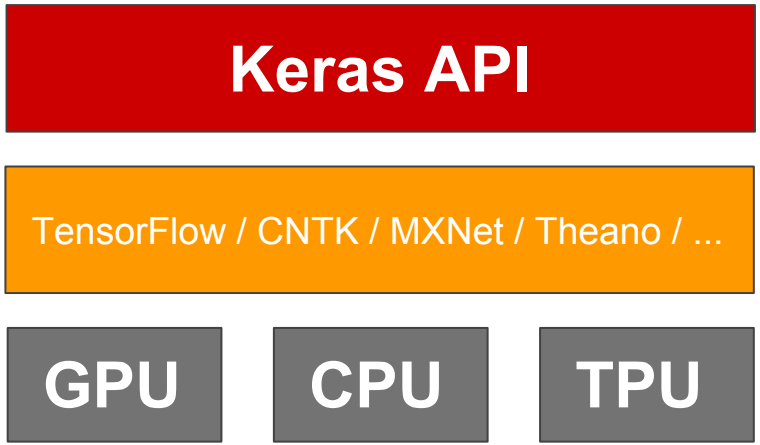
Large-scale deep learning with Keras

Francois Chollet
March 24th, 2018

Outline

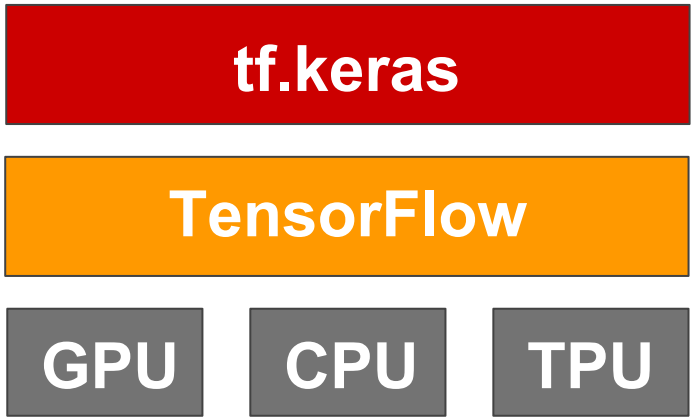
- Introduction: what's Keras?
- Overview of distributed training, multi-GPU training, & TPU training options
- Example: building a video captioning model with distributed training on Google Cloud

Keras: an API for specifying & training differentiable programs



Keras is the high-level model-building API of TensorFlow

- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- Better optimized for TF
- Better integration with TF-specific features
  - Estimator API
  - Eager execution
  - etc.



What’s special about Keras?

- Large adoption in the industry and research community.
- A focus on user experience.
- Multi-backend, multi-platform.
- Easy productization of models.

250,000

Keras developers

Industry traction

etc...

Distributed, multi-GPU, & TPU training

Distributed Keras

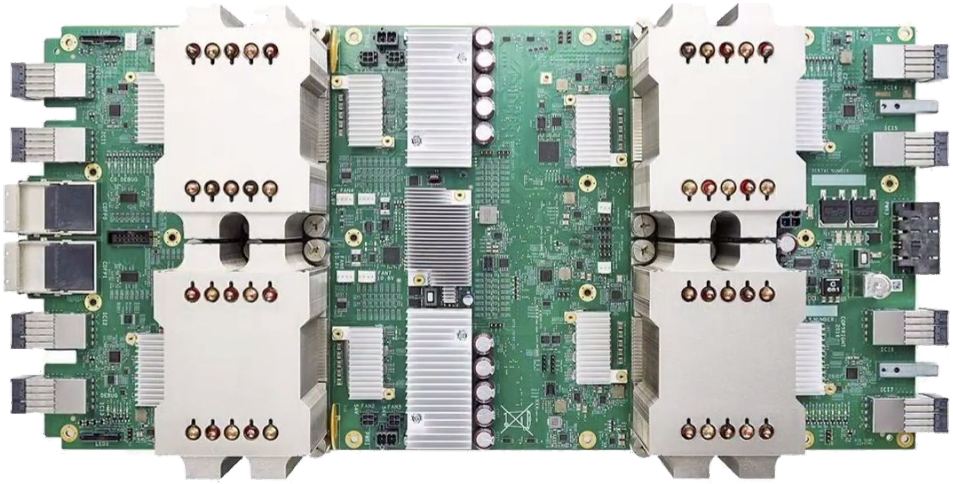
- Uber’s Horovod
- TF Estimator API (TF built-in option - only tf.keras)
- Keras on Spark
  - Dist-Keras (from CERN)
  - Elephas

There’s also built-in support for single-node, multi-GPU training

TPU support

Only tf.keras

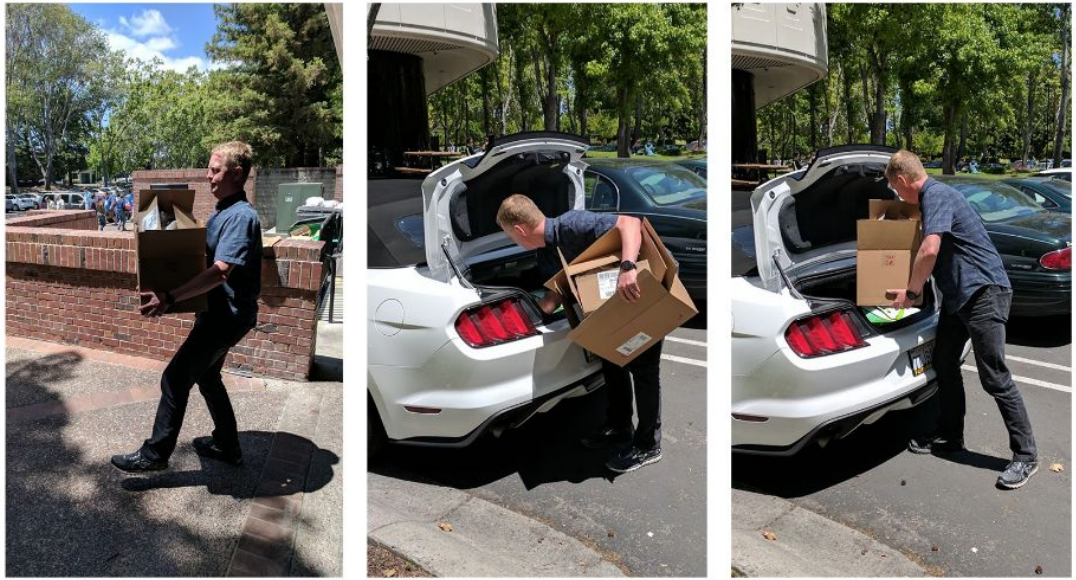
Via Estimator API



Example: building a video captioning model

with distributed training via the TF Estimator API

Toy video-QA problem



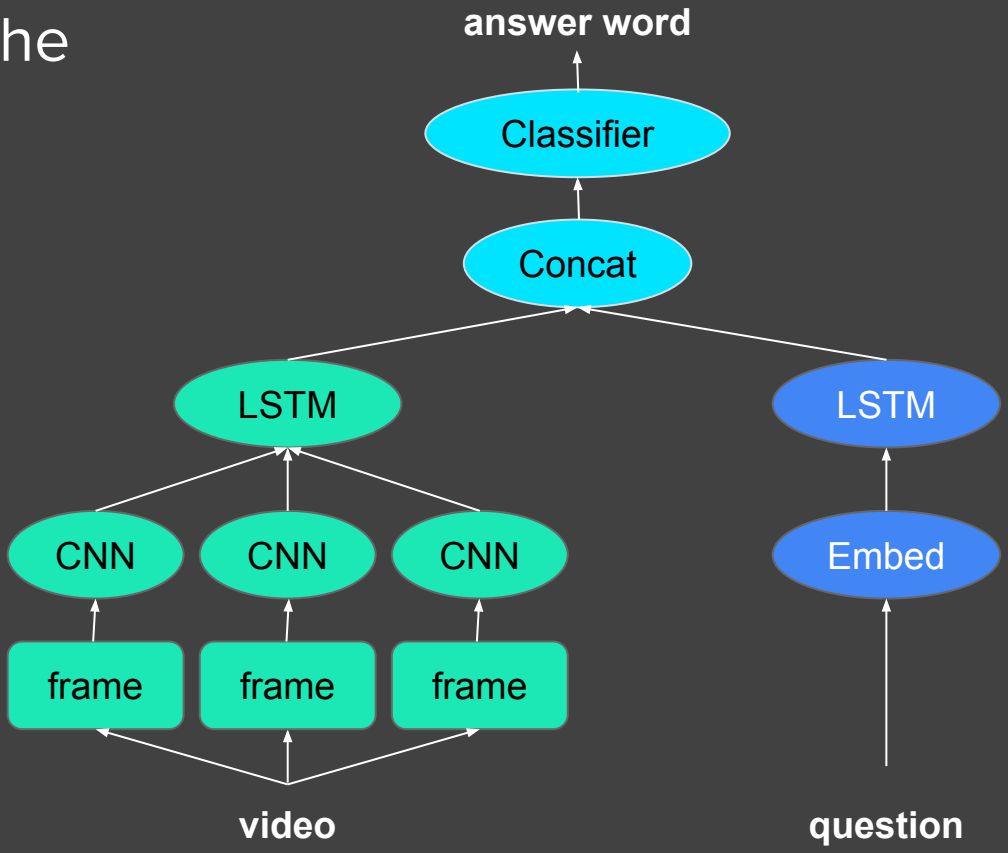
> ***“What is the man doing?”***
> packing

> ***“What color is his shirt?”***
> blue

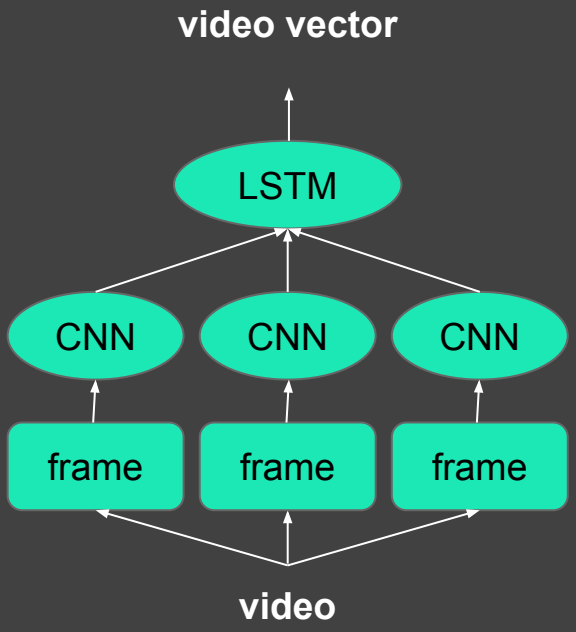
Overview of solution

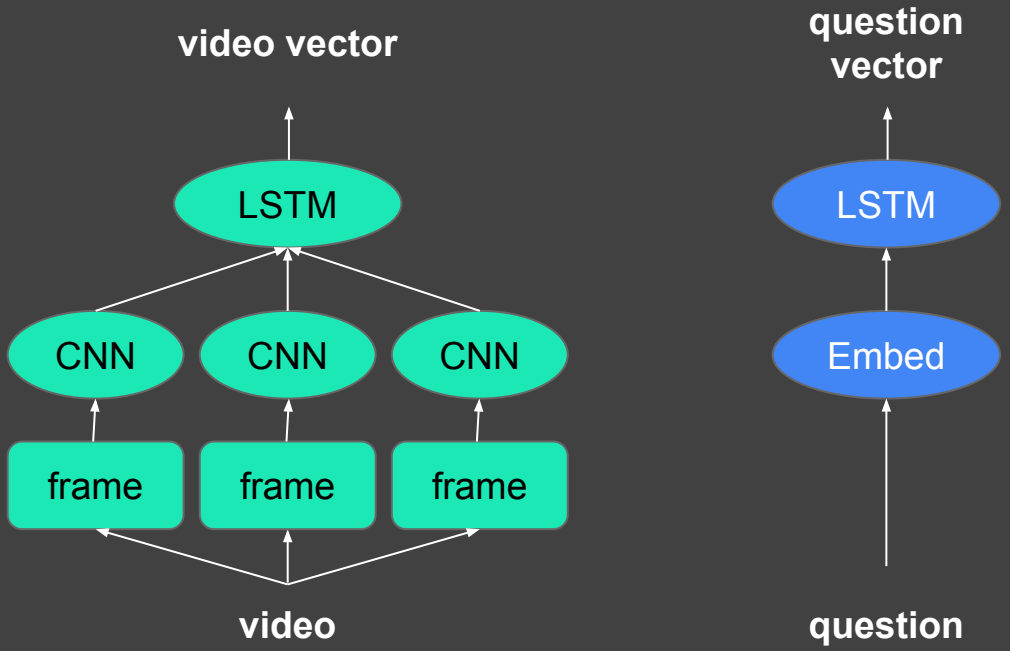
- Design network
- Write `model.py` implementing it (with tf.keras)
  - 15 lines for model definition
  - 12 lines for data/training handling
- Package it as a binary
- Upload binary to Google Cloud ML Engine
- Train on arbitrary number of GPUs using asynchronous data parallelism
  - From data stored on Google Cloud

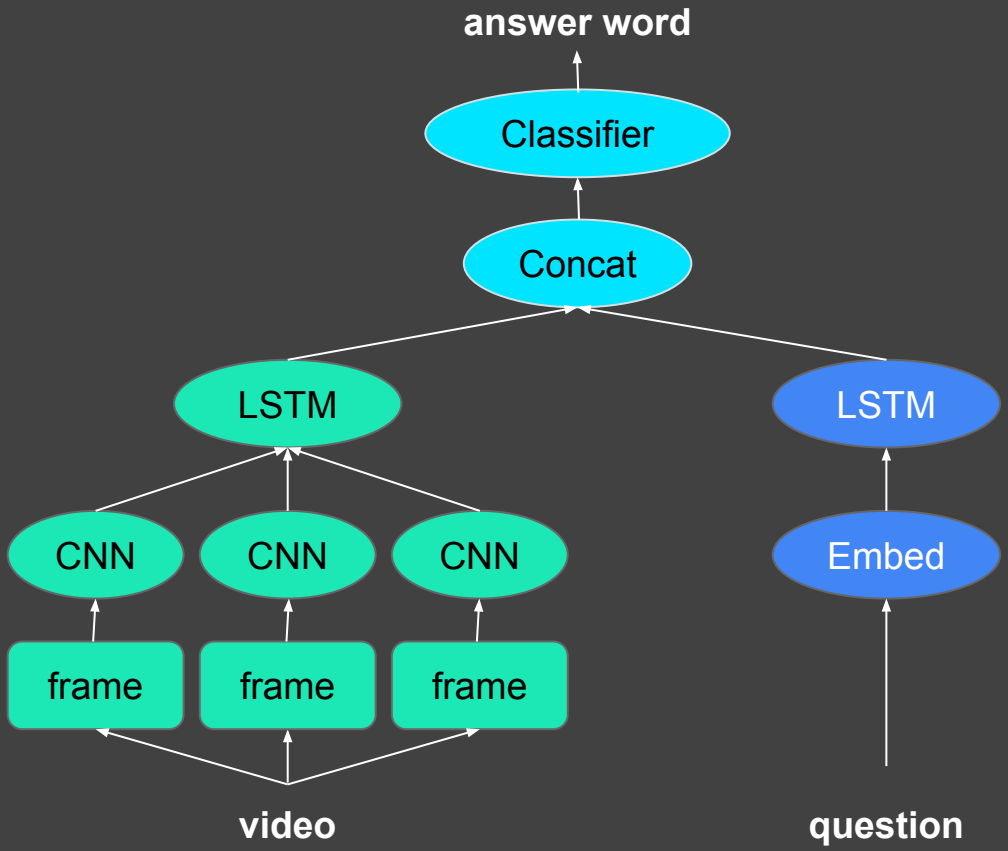
Designing the network

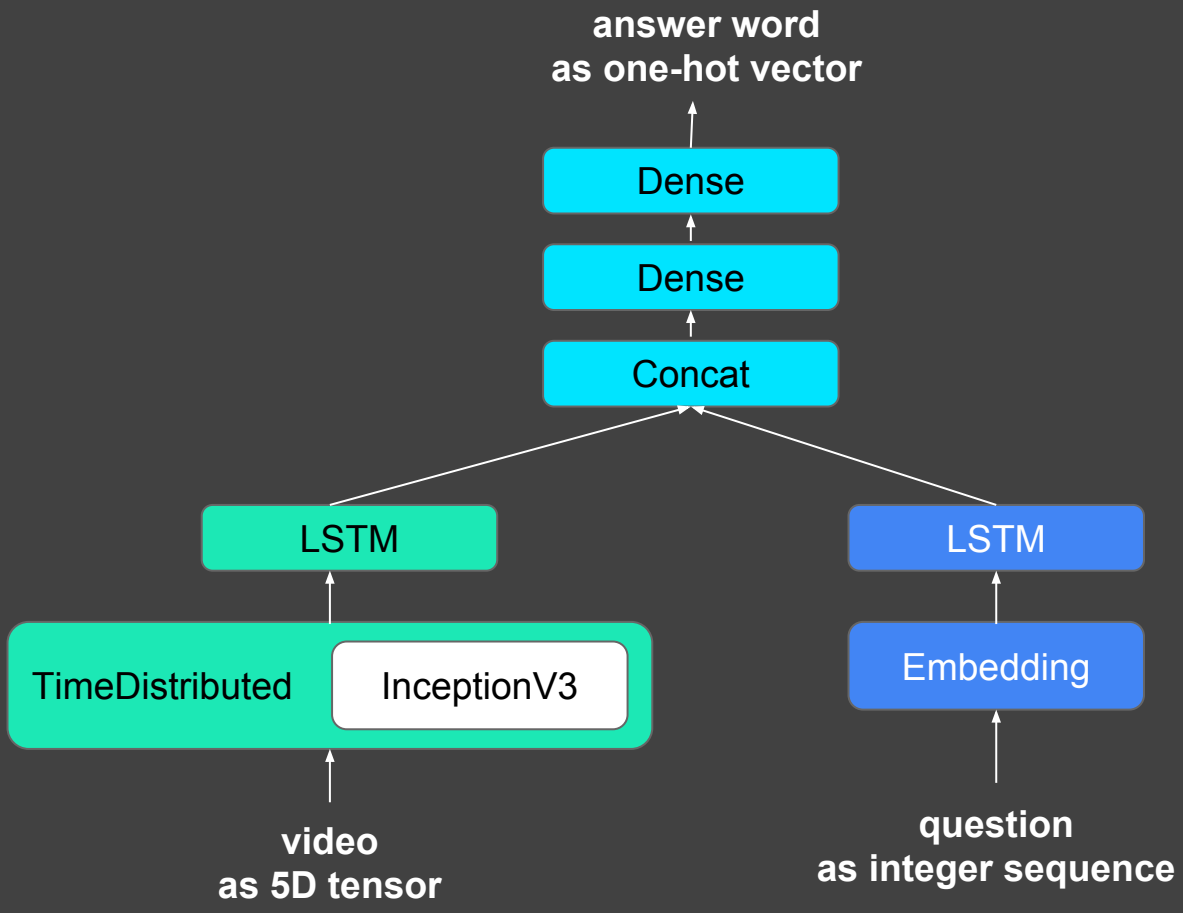


From frames to a vector




video vector
LSTM
CNN
CNN
CNN
frame
frame
frame
video
question vector
LSTM
Embed
question


answer word
Classifier
Concat
LSTM
LSTM
CNN
CNN
CNN
Embed
frame
frame
frame
video
question


answer word
as one-hot vector
Dense
Dense
Concat
LSTM
LSTM
TimeDistributed
InceptionV3
Embedding
video
as 5D tensor
question
as integer sequence

Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Turning frames into a vector, with pre-trained representations

```
from tensorflow import keras
from tensorflow.keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = keras.layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Turning a sequence of words into a vector

```
question = keras.Input(shape=(None,), dtype='int32', name='question')
embedded_words = keras.layers.Embedding(input_voc_size, 256)(question)
question_vector = keras.layers.LSTM(128)(embedded_words)
```

Predicting an answer word

```
x = keras.layers.concatenate([video_vector, question_vector])
x = keras.layers.Dense(128, activation=tf.nn.relu)(x)
predictions = keras.layers.Dense(output_voc_size,
                                 name='predictions')(x)
```

Setting up the training configuration

```
model = keras.models.Model([video, question], predictions)
model.compile(optimizer=tf.train.AdamOptimizer(3e-4),
              loss=keras.losses.categorical_crossentropy)

config = {output_dir: '...'}
estimator = keras.estimator.model_to_estimator(model,
                                               config=config)
```

Creating the input function with the TF Dataset API

```
def input_fn(filenames,
             epochs=100,
             batch_size=8):
  # Parse files and create dataset (omitted)
  dataset = tf.data.from_tensor_slices(...)

  dataset = dataset.repeat(epochs)
  dataset = dataset.batch(batch_size)
  iterator = dataset.make_one_shot_iterator()
  video, question, labels = iterator.get_next()
  return {'video': video, 'question': question}, labels
```

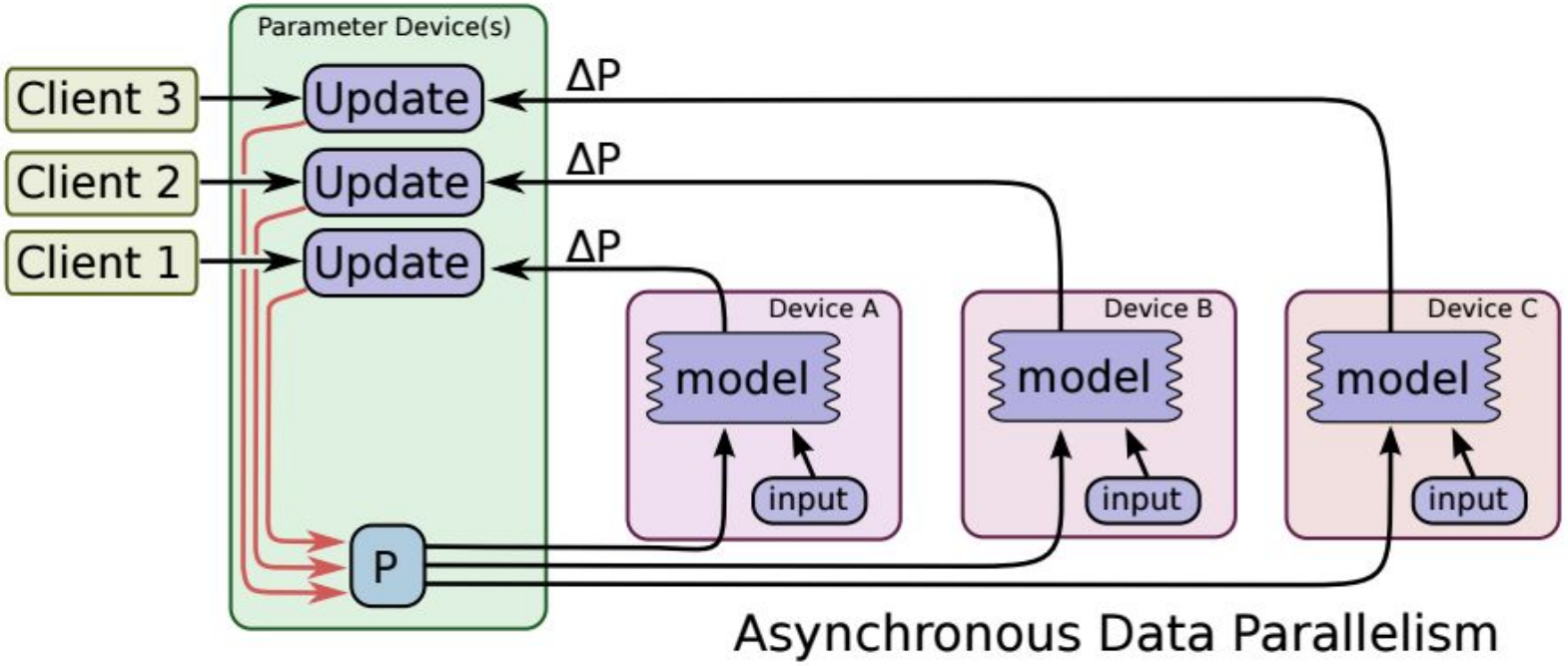
Distributed training and evaluation

```
train_input = lambda: input_fn(TRAIN_FILES, batch_size=8)
train_spec = tf.estimator.TrainSpec(train_input, max_steps=10000)

eval_input = lambda: input_fn(EVAL_FILES, batch_size=8)
eval_spec = tf.estimator.TrainSpec(eval_input, steps=100)

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

Next: packaging and upload

We'll use a single `gcloud` command to:

- Package our code (and dependencies) into a binary
- Upload the binary to CMLE
- Specify the location of the training data (on GCS)
- Specify a number/type of workers & parameter servers
- Start distributed asynchronous training


Parameter Device(s)
Client 3
Client 2
Client 1
Update
Update
Update
ΔP
ΔP
ΔP
Device A
Device B
Device C
model
model
model
input
input
input
P
Asynchronous Data Parallelism

Start job on Google Cloud

First, we create a project folder:

```
trainer/
    ...model.py -> train_and_evaluate
    ...task.py -> parses arguments, calls model.py
```

```
gcloud ml-engine jobs submit training $JOB_NAME \
     --config scaling_config.yaml \
     --runtime-version 1.4 \
     --job-dir $GCS_JOB_DIR \
     --package-path trainer/ \
     --module-name trainer.task \
     --region us-central1 \
     --train-files $GCS_TRAIN_FILE \
     --eval-files $GCS_EVAL_FILE
```

Scaling configuration in scaling_config.yaml

```
trainingInput:
    scaleTier: CUSTOM
    masterType: standard_p100
    workerType: standard_p100
    parameterServerType: standard
    workerCount: 16
    parameterServerCount: 8
```

Main takeaways from this example

- Concise, easy model definitions with tf.keras
  - Including mix-and-matching existing pre-trained models)
- Concise, easy distributed training with TF Estimator API
  - Just configure and call `train_and_evaluate`
- CMLE gives you access to easy scaling of your training jobs
  - Just specify the number & type of workers, parameter servers
- The same code can be run on your own cluster (no lock-in)
  - Can also be run locally for debugging
- Alternatively, you can use Uber's Horovod

Thank you!