Possibly bugs in slides, will fix and reupload tonight

Applied Deep Learning

Lecture 6 • Oct 10th, 2019

Agenda

Concepts

- Distributed training
- Input pipeline performance
- Intro to Embeddings
- *Break / try the embedding projector*
- Background on TF1
- Dropout
- BatchNorm

News

Events

- PyTorch dev summit is today (videos will have the latest developments there if you're interested)

Papers

- Chester: A Web Delivered Locally Computed Chest X-Ray Disease Prediction System (trained with PyTorch, deployed with TF.js)

  Contributions / ideas: simplified deployment (just a URL). **Quick discussion**: what else? (and, what are considerations for this deployment strategy?)

News

Events

- PyTorch dev summit is today (videos will have the latest developments there if you're interested)

Papers

- Chester: A Web Delivered Locally Computed Chest X-Ray Disease Prediction System (trained with PyTorch, deployed with TF.js)

  Contributions / ideas: simplified deployment (just a URL). **Quick discussion**: what else? Easy demo / helps folks understand the limitations of the tool / think through use cases. Large model / slow downloads (150 MB, especially for developing world).

Administrative stuff

Review lecture before midterm?

- Yes def.

Demo day 12/5

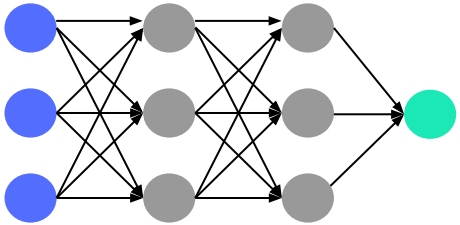
- Half of last day of class, regularly scheduled
- You do *not* have to have a completed project! Submission deadline is as late as possible to give you time.
- Just so we can see how it's going, help you if you're stuck.

Distributed training

Training loop

Training data

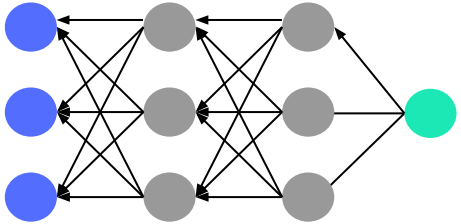
Input pipeline



Forward pass: compute predictions

Compute loss

Update parameters



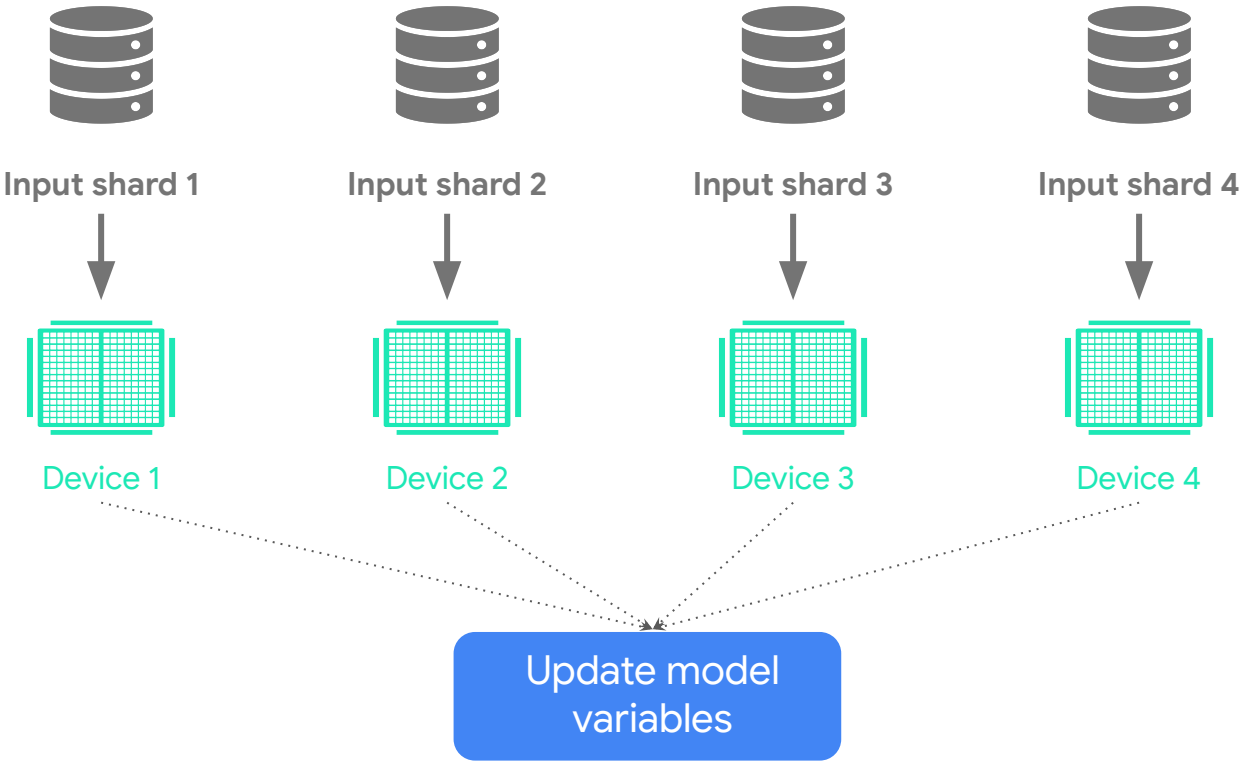
Backward pass: compute gradients

How can we go faster?

Given multiple cores, how can we use these to accelerate training?

“ “, how can we use these to train larger models?

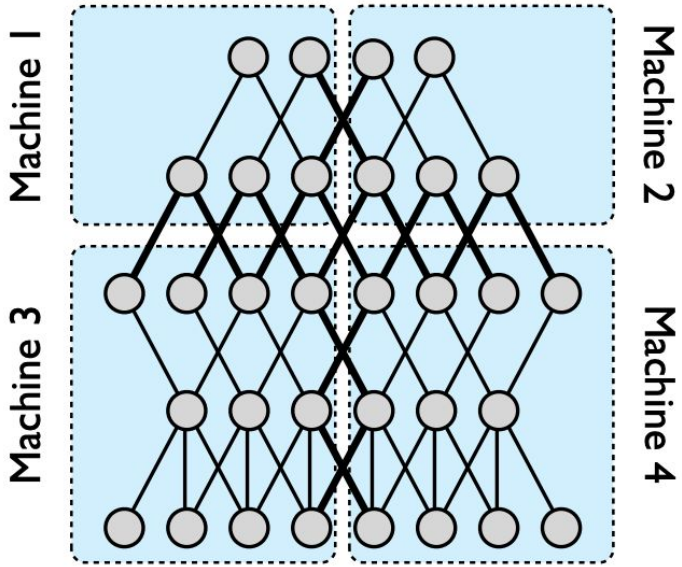
Data parallelism



Goal: effectively increase batch size without increasing computation time (larger batch → more accurate gradients).

Many possible strategies.

Model parallelism



Goal: handle layers that are too large to fit into memory (and/or accelerate training by dividing compute of a single layer across multiple machines).

Cons
Generally requires code changes.

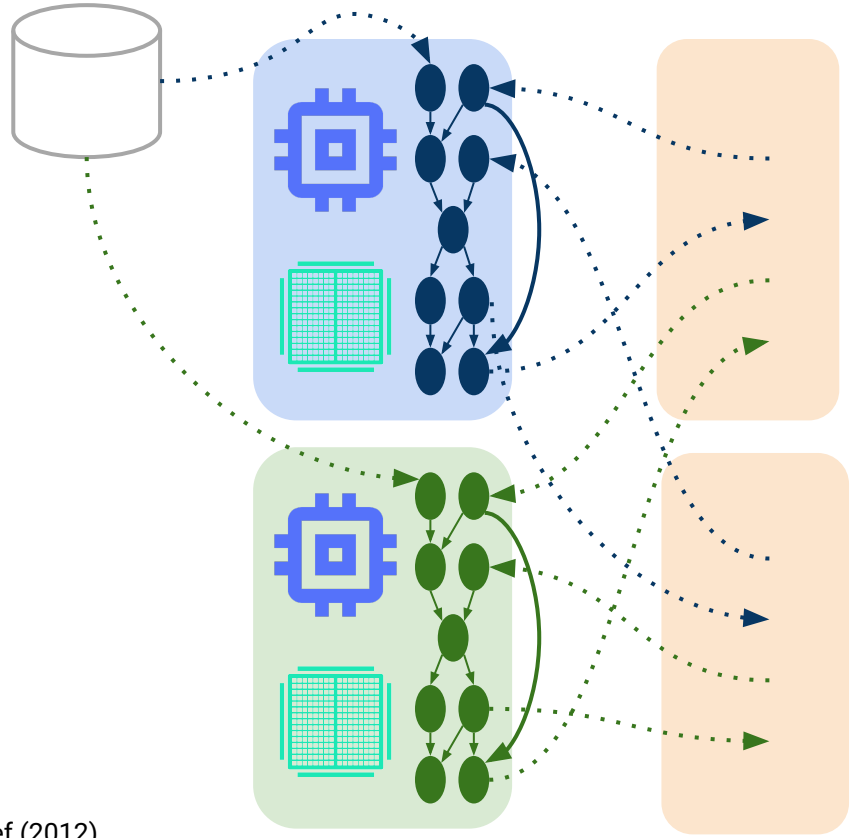
Large Scale Distributed Deep Networks, DistBelief (2012)

Early solutions

Parameter servers

Workers
Read input data, fetch parameters (weights) from server, compute forward and back pass, send gradient updates.

Quick discussion
Imagine you have 512 workers, and 8 parameter servers. How do you coordinate updates? Variable fetches?



Designed for the CPU era (lots of inexpensive cores available, but relatively slow). GPUs were used at the time, but were expensive and memory limited.

PS
Each variable has a home on a single parameter server.

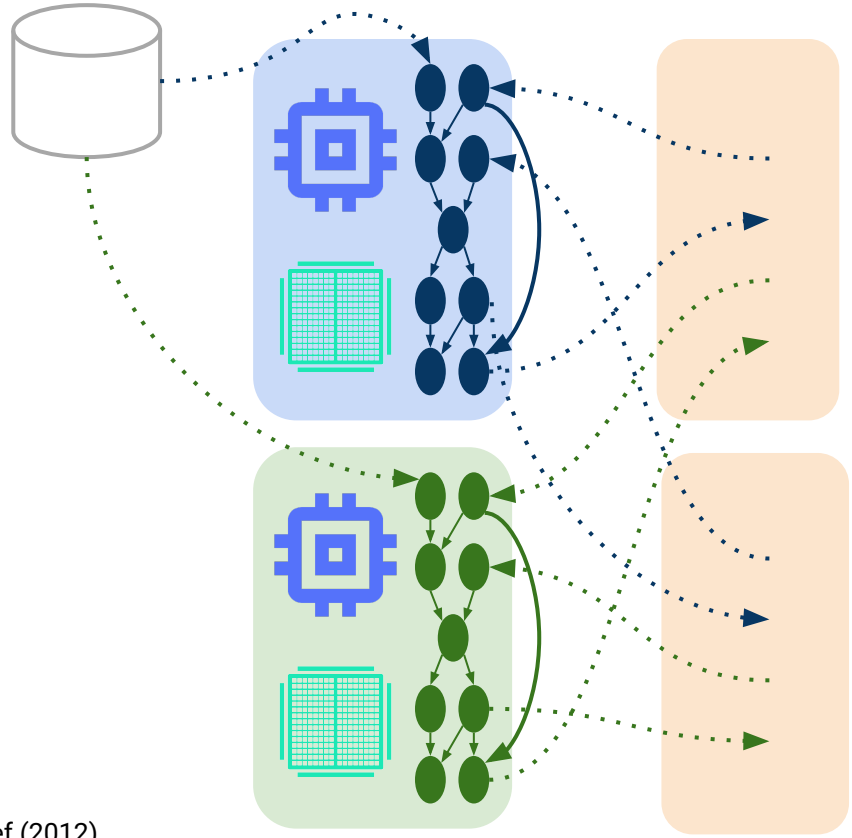
Large Scale Distributed Deep Networks, DistBelief (2012)
Diagram from a helpful talk by Josh Levenberg, Aug 2019

Parameter servers

Workers
Read input data, fetch parameters (weights) from server, compute forward and back pass, send gradient updates.

Solution
Workers are independent. Updates made asynchronously.

Quick discussion
What could go wrong?



PS
Each variable has a home on a single parameter server.

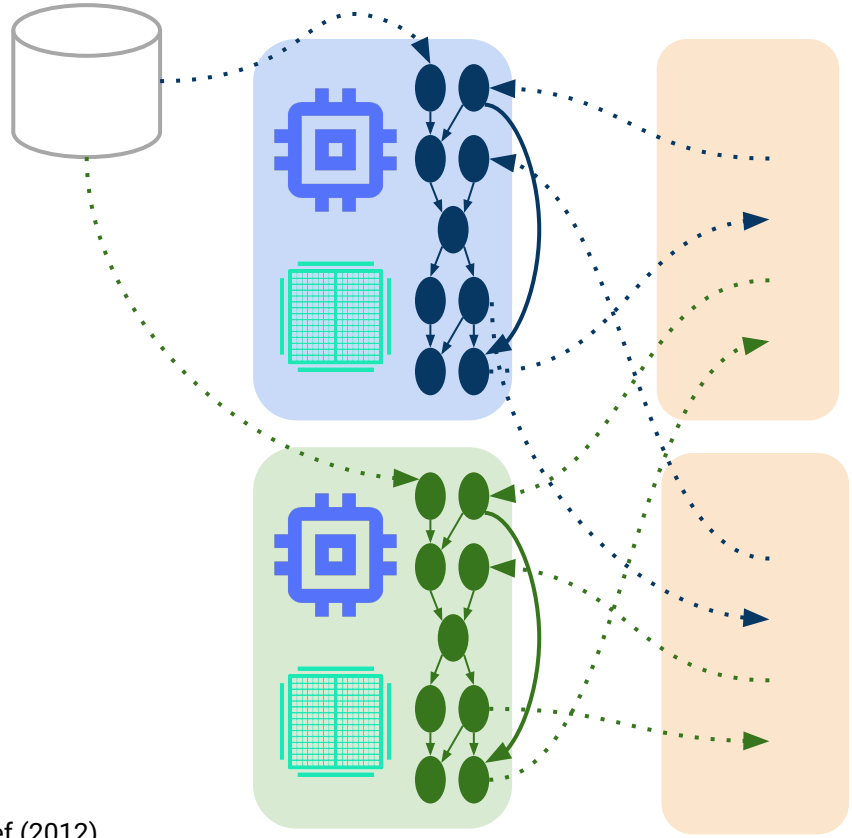
Large Scale Distributed Deep Networks, DistBelief (2012)
Diagram from a helpful talk by Josh Levenberg, Aug 2019

Parameter servers

Workers
Read input data, fetch parameters (weights) from server, compute forward and back pass, send gradient updates.

Solution
Workers are independent. Updates made asynchronously.

Quick discussion
What could go wrong?
Good chance a worker is operating on stale data.



Quick discussion
What could go right?
(What does this scheme allow?)

PS
Each variable has a home on a single parameter server.

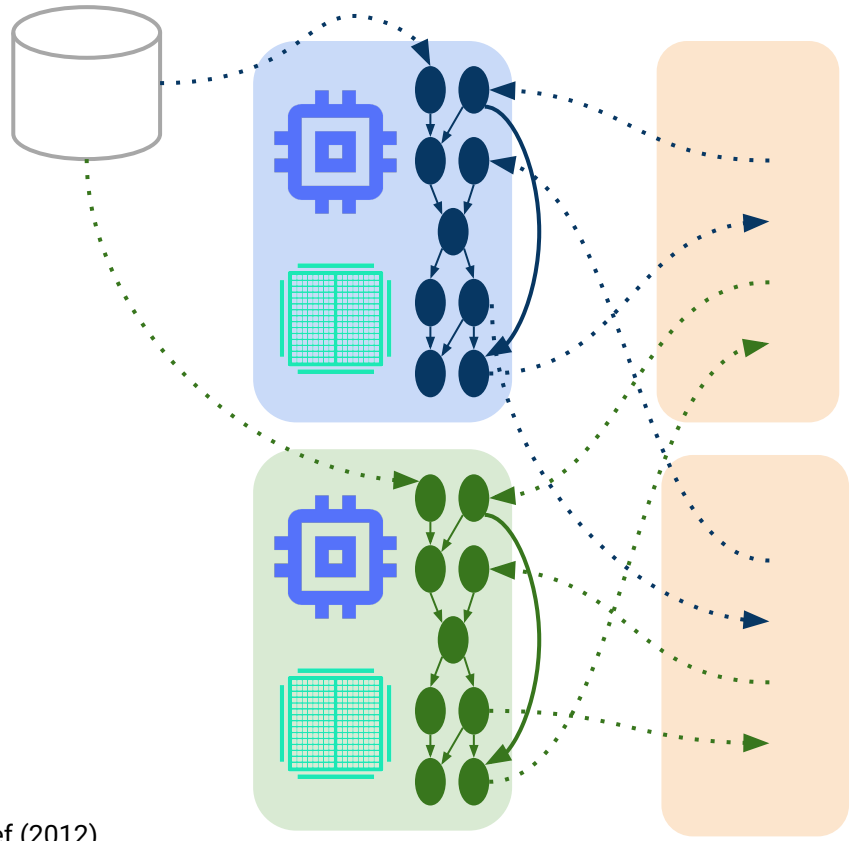
Large Scale Distributed Deep Networks, DistBelief (2012)
Diagram from a helpful talk by Josh Levenberg, Aug 2019

Parameter servers

Workers
Read input data, fetch parameters (weights) from server, compute forward and back pass, send gradient updates.

Solution
Workers are independent. Updates made asynchronously.

Quick discussion
What could go wrong?
Good chance a worker is operating on stale data.



Quick discussion
What could go right? (What does this scheme allow?)

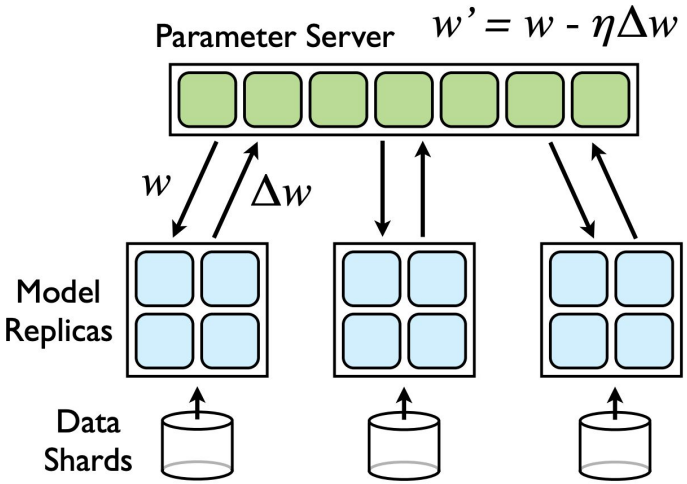
Benefits: Loose coupling allows workers to be preempted by higher priority jobs (not a problem if a single worker is disconnected) / great for large, long running jobs.

But…

Drawbacks: complexity (both in terms of engineering and infrastructure), probably not something a single researcher will be able to spin up outside of a large team.

Large Scale Distributed Deep Networks, DistBelief (2012)
Diagram from a helpful talk by Josh Levenberg, Aug 2019

Downpour SGD



Notes

Asynchronous updates via parameter servers.

- Divide training data into subsets.
- Run a copy (replica) of the model on each worker.

Workers

- Workers communicate updates through a centralized parameter server.
- Works take a batch of data as input, fetch the latest version of the parameters (weights) from a server, calculate gradients, and send the updates they want to make back to the server.

Parameter servers (PS)

- PS keeps the current state of all parameters for the model, sharded across multiple machines.
- If 10 machines, each PS has 1/10 of the variables (and is responsible for storing and applying updates to that subset).

Drawbacks: workers may be operating on stale data (a model replica is almost certainly computing its gradients based on a set of parameters that are slightly out of date, in that some other model replica will likely have updated the parameters on the parameter server in the meantime). Empircally worked well.

Large Scale Distributed Deep Networks, DistBelief (2012)

Modern approach

Horovod

Synchronous data parallel training, based on earlier work from Baidu.

Key contributions

- Network efficient all-reduce
- Simplified developer tools
- Used NCCL (NVIDIA's collective communication library)

Trivia (does anyone know the etymology of the name?)

https://eng.uber.com/horovod/ (2017)
Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations (2009)

Mirrored strategy

Notes

Current best practice.

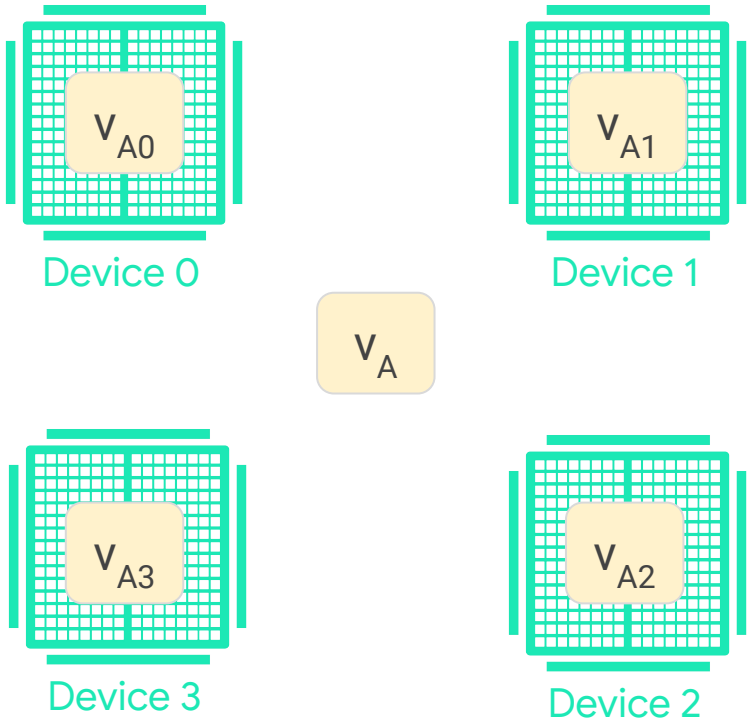
- Ideal for one-machine, multi-GPU (also, there's experimental support for multi-machine, multi-GPU in TF2).

Unlike downpour, does not affect SGD algorithm.

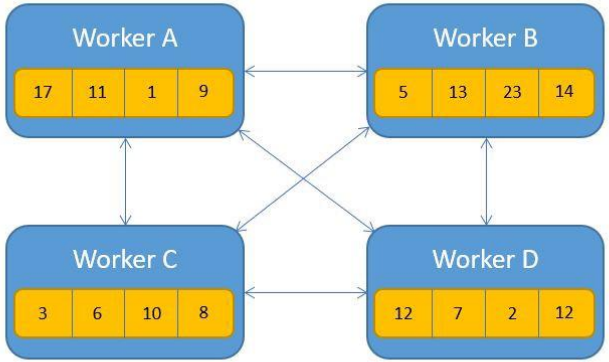
- Each device has a copy of the model and parameters.
- Devices are run in lock-step (variables and updates are synchronized at each step)

The trick is to do this efficiently (the naive approach is to have each GPU run a forward pass, backward pass, then average the gradients and make updates).

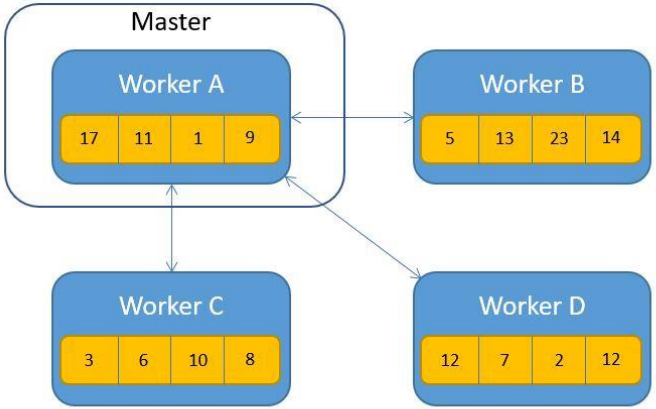
Quick discussion: how can we go faster?



Example: two inefficient strategies for all-reduce



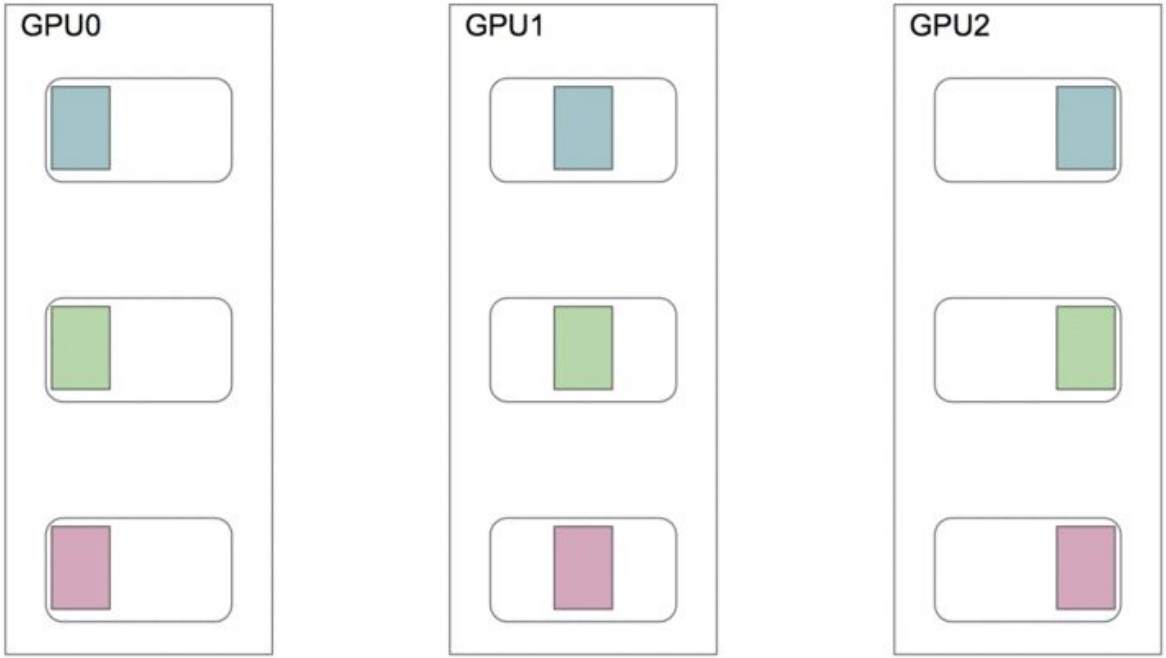
All-to-all



All-to-master

Images from A Visual intuition on ring-Allreduce for distributed Deep Learning

Ring all-reduce

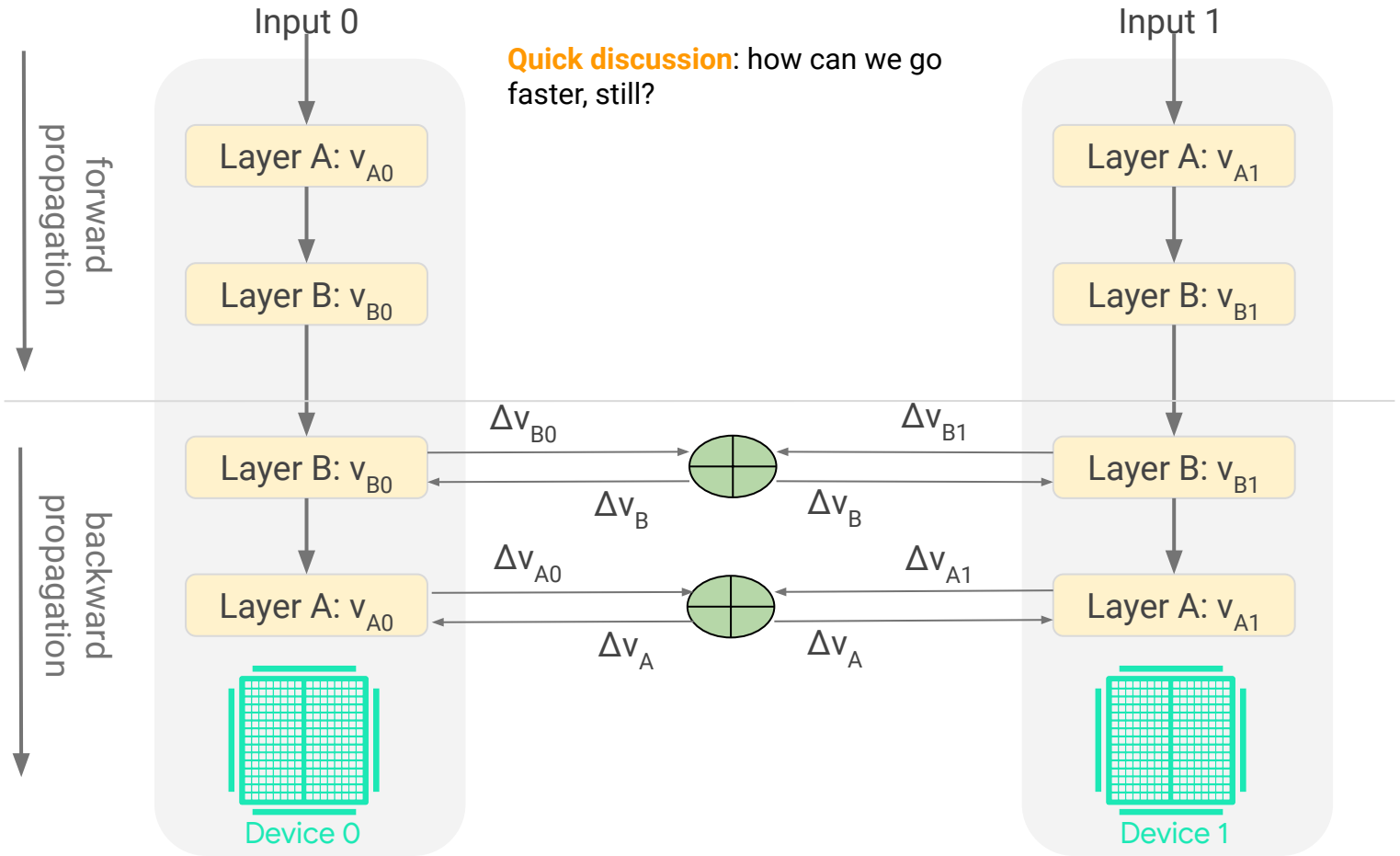


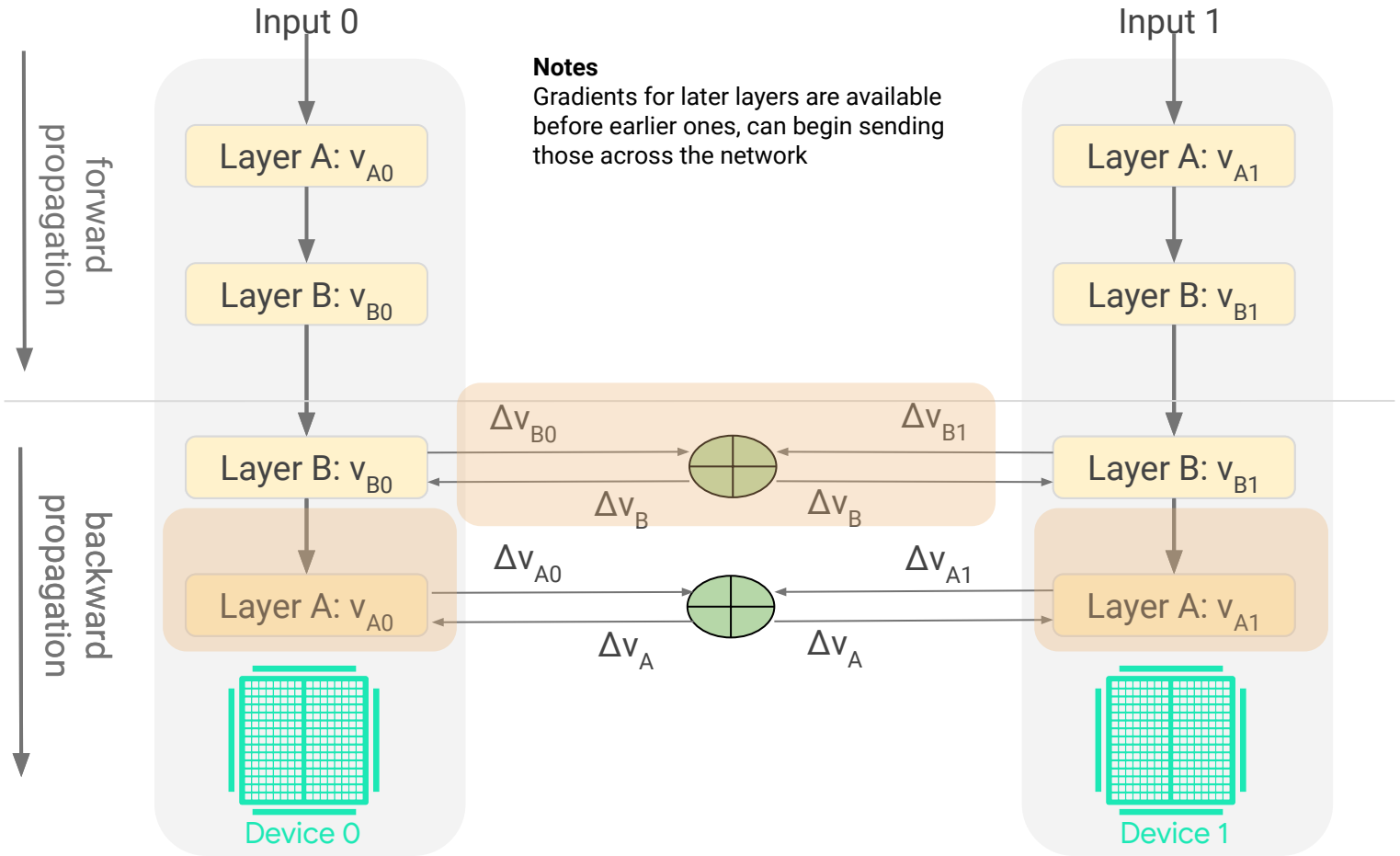
Notes

“all” = from every device, to every device.

“reduce” = sum or mean.

Network efficient.


Input 0
Input 1
Quick discussion: how can we go faster, still?
forward propagation
backward propagation
Layer A: v_{A0}
Layer B: v_{B0}
Layer A: v_{A1}
Layer B: v_{B1}
Δv_{B0}
Δv_{B1}
Δv_B
Δv_B
Δv_{A0}
Δv_{A1}
Δv_A
Δv_A
Device 0
Device 1



Notes
Gradients for later layers are available before earlier ones, can begin sending those across the network

Implementation in tf.distribute

```
# No code changes to model necessary

  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(10, activation='softmax')
  ])

  model.compile(loss='sparse_categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])

model.fit(train_dataset, epochs=10)
```

```
mirrored_strategy = tf.distribute.MirroredStrategy() # You can also pass an all-reduce strategy

# In general, use the largest batch size that fits the GPU memory
# and tune the learning rate accordingly.
BATCH_SIZE_PER_REPLICA = 64
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_sync

with strategy.scope():
  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(10, activation='softmax')
  ])

  model.compile(loss='sparse_categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])

model.fit(train_dataset, epochs=10)
```

With multiple machines

Cluster config

```
os.environ['TF_CONFIG'] = json.dumps({
    'cluster': {
        # would be other machines on the network
        'worker': ["localhost:12345", "localhost:23456"]
    },
    'task': {'type': 'worker', 'index': 0}
})
```

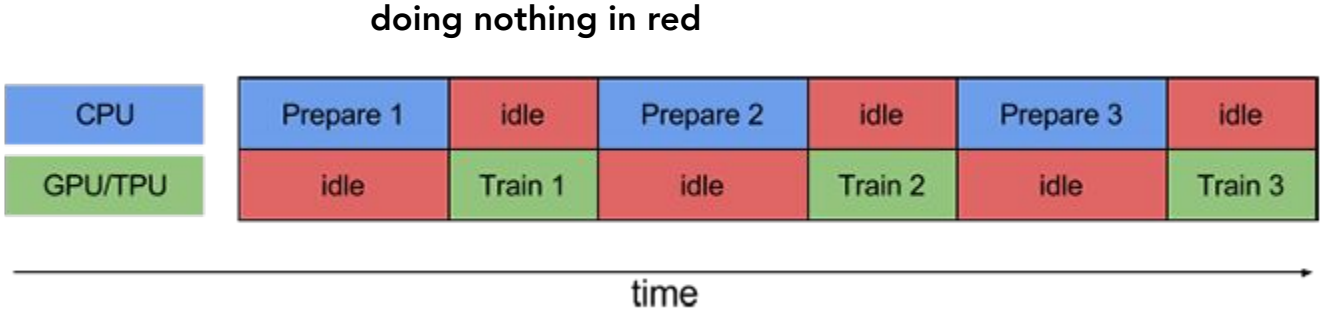
Notes
An environmental variable that needs to be set on each machine. Each machine has an identical cluster spec, the only difference is the index below. Worker “0” is the “chief” (and responsible for extra work, like saving checkpoints, and/or logs for TensorBoard).

tensorflow.org/tutorials/distribute/multi_worker_with_keras

Input pipeline performance counts

It doesn’t matter how fast your GPUs are if they’re sitting around waiting for data

Example 1



Scenario: a CPU is preparing data (say, reading images off disk, and doing some preprocessing - normalization, data augmentation, etc) before sending them to a GPU.

Where is the slowdown?

tensorflow.org/guide/data_performance

Example 1

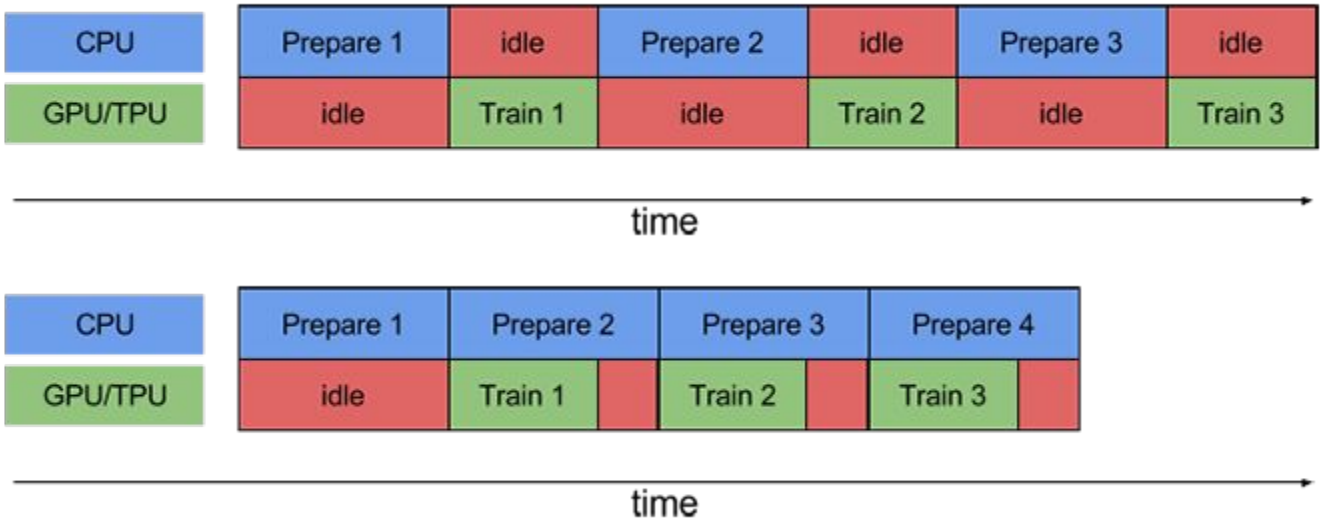
Scenario: a CPU is preparing data (say, reading images off disk, and doing some preprocessing - normalization, data augmentation, etc) before sending them to a GPU.

Where is the slowdown?

Producer / consumer, but synchronous (CPU prepares data, GPU runs a forward / backward pass. CPU prepares next batch, etc.)

tensorflow.org/guide/data_performance

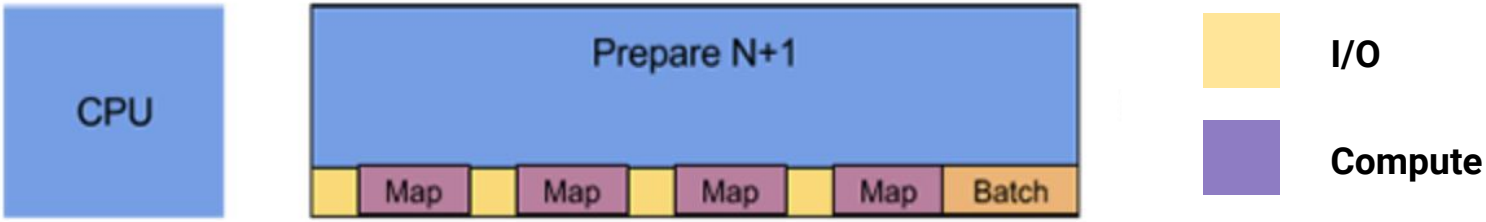
Example 1



Prefecting. Decouples producer / consumer, so the CPU can begin preparing the next batch while the GPU is classifying on the previous one.

tensorflow.org/guide/data_performance

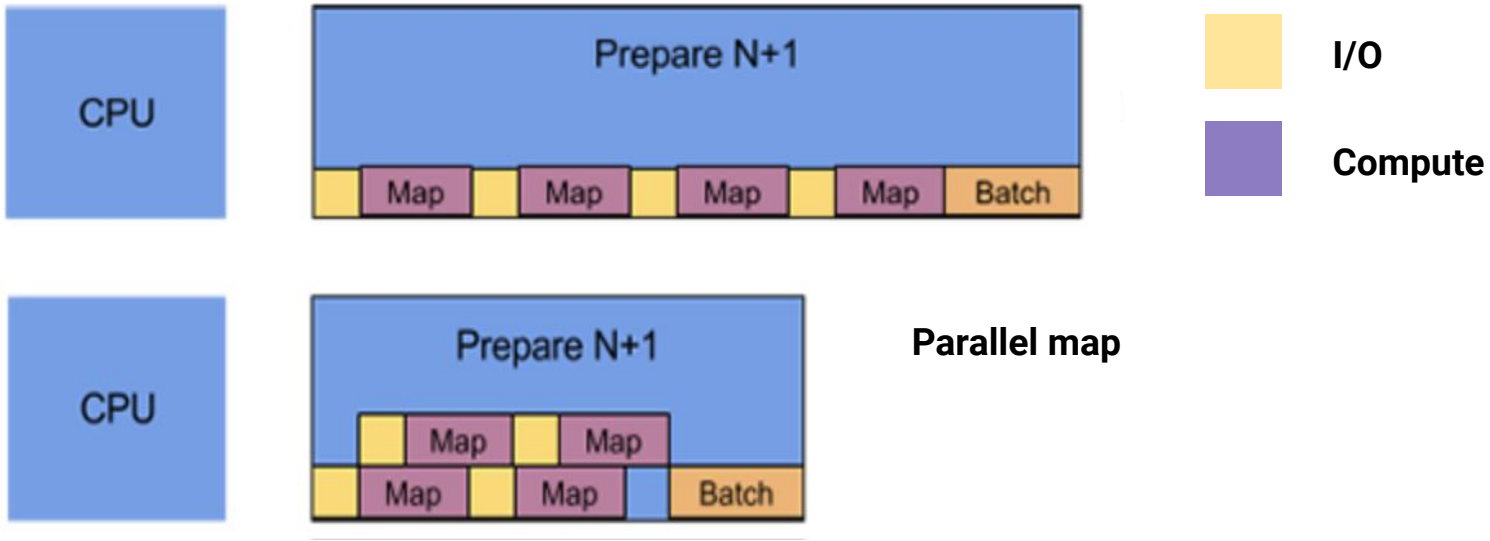
Example 2



Just looking at the CPU now. Say we have a single CPU with 4 cores. As before, it's reading images off disk, and doing some preprocessing (normalization, data augmentation, etc) before sending them to a GPU.

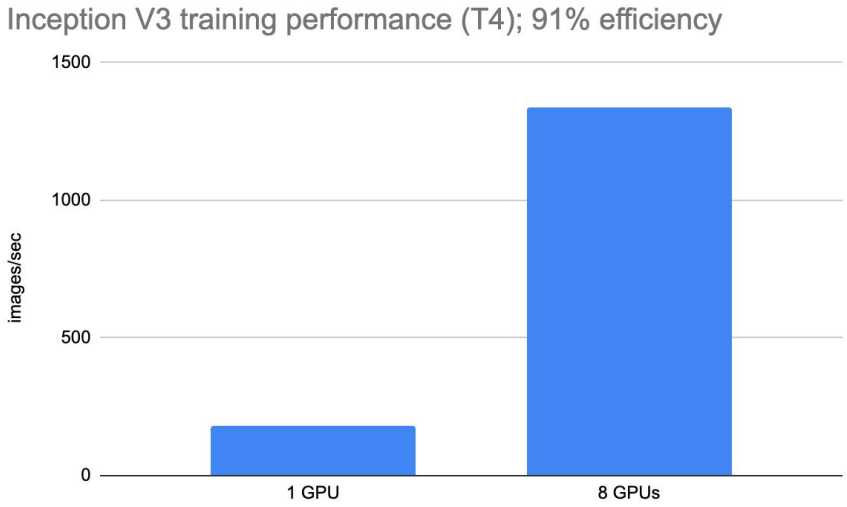
Where is the slowdown?

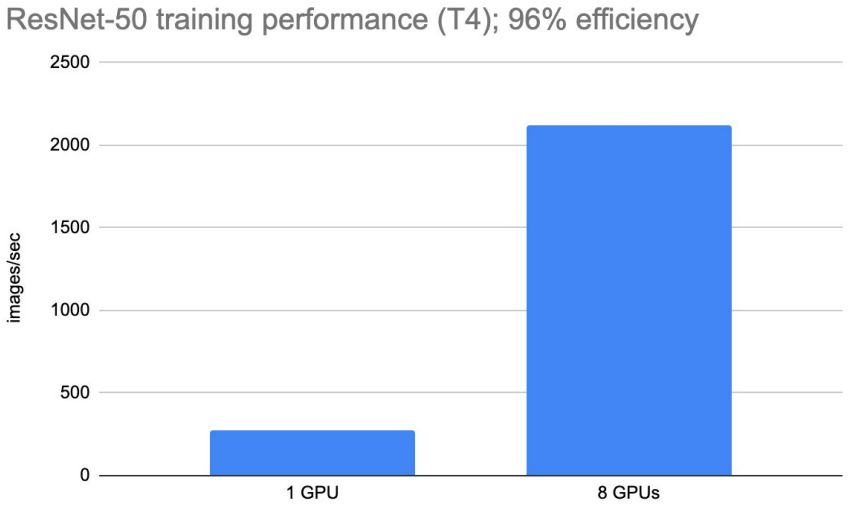
Example 2


CPU
Prepare N+1
Map
Map
Map
Map
Batch
CPU
Prepare N+1
Map
Map
Map
Map
Batch
I/O
Compute
Parallel map

```
# Recall example in HW2 (benchmarking data input pipelines)
# Best practice: benchmark your pipeline before spinning up multiple GPUs
# note for distributed training, sharding is handled by model.fit ()

files = tf.data.Dataset.list_files(file_pattern)
dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
dataset = dataset.map(parser_fn, num_parallel_calls=40)
dataset = dataset.batch(batch_size)
dataset = dataset.prefetch(1)
```

For reference (how much input pipeline performance do you need?)





Notes: T4s are the same GPUs you'll find on some Colab instances. Even with relatively common hardware, you can see a naive input pipeline may already become a bottleneck.

developer.nvidia.com/deep-learning-performance-training-inference

A word on benchmarks

Benchmarking distributed training is difficult. **Depends on…**

- ?

A word on benchmarks

Benchmarking distributed training is difficult. **Depends on…**

- Type of model
- Type of GPU
- Network
- Synthetic data or real data
- Etc

For exotic hardware

Mesh TensorFlow

Imagine you have a “mesh” of fast compute units, with high / fast network connectivity. E.g., hundreds of cores on a single chip (or a TPU).

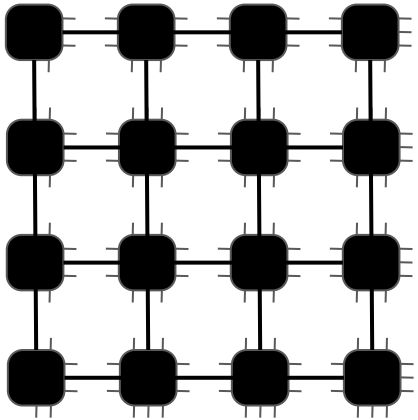
How could we use this design to simplify distributed computing?

Advantages of data parallelism

- Every processor involved in every operation
- Single program multiple devices
- Collective communication (all-reduce)

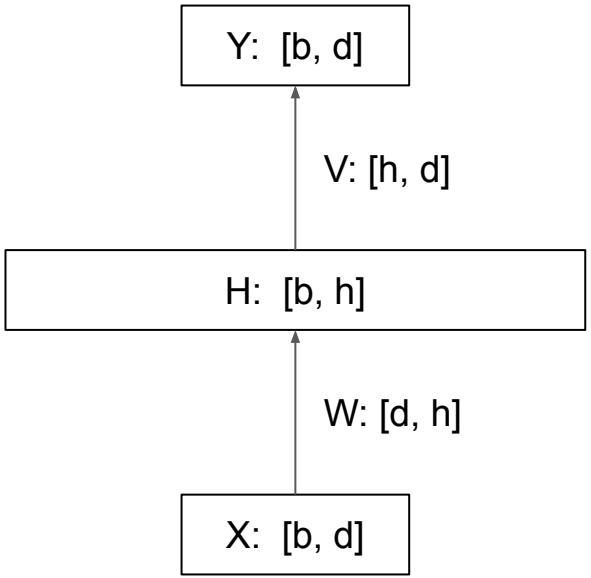
Advantages of model parallelism

- Larger models are possible (not limited by memory of single device)
- **But, complex… is there a solution with a mesh?**



An example NN

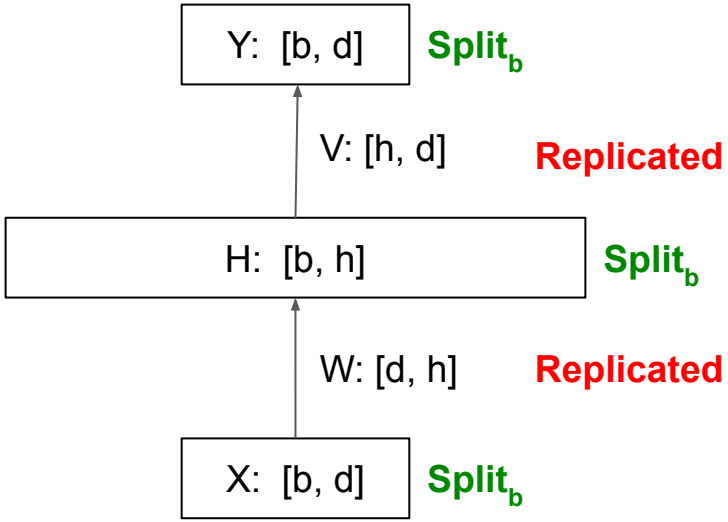
$$Y = (H = \text{Relu}(XW))V$$

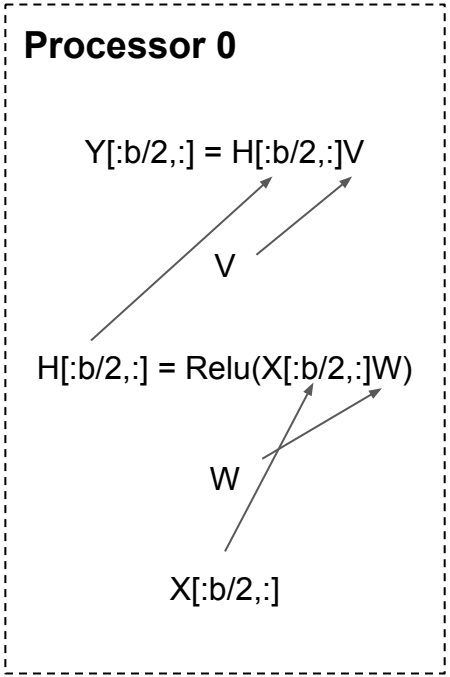


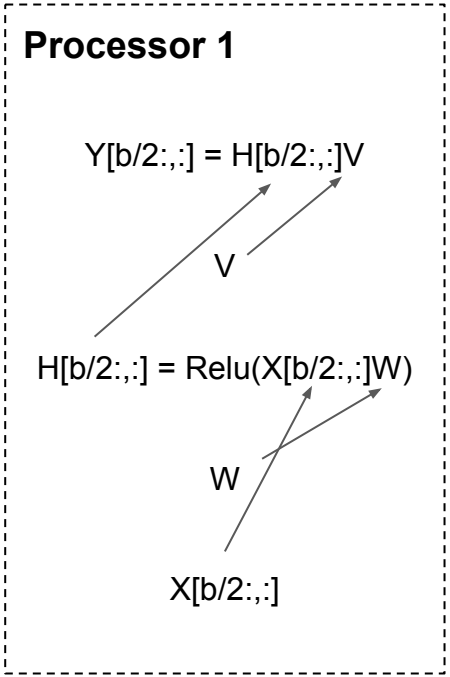
Diagrams from Noam Shazeer, March 2019

Data Parallelism: split dimension “b”

$$Y = (H = \text{Relu}(XW))V$$



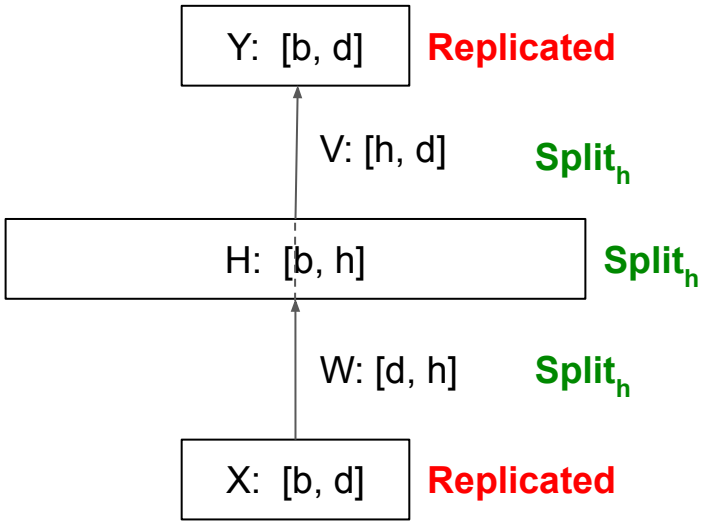


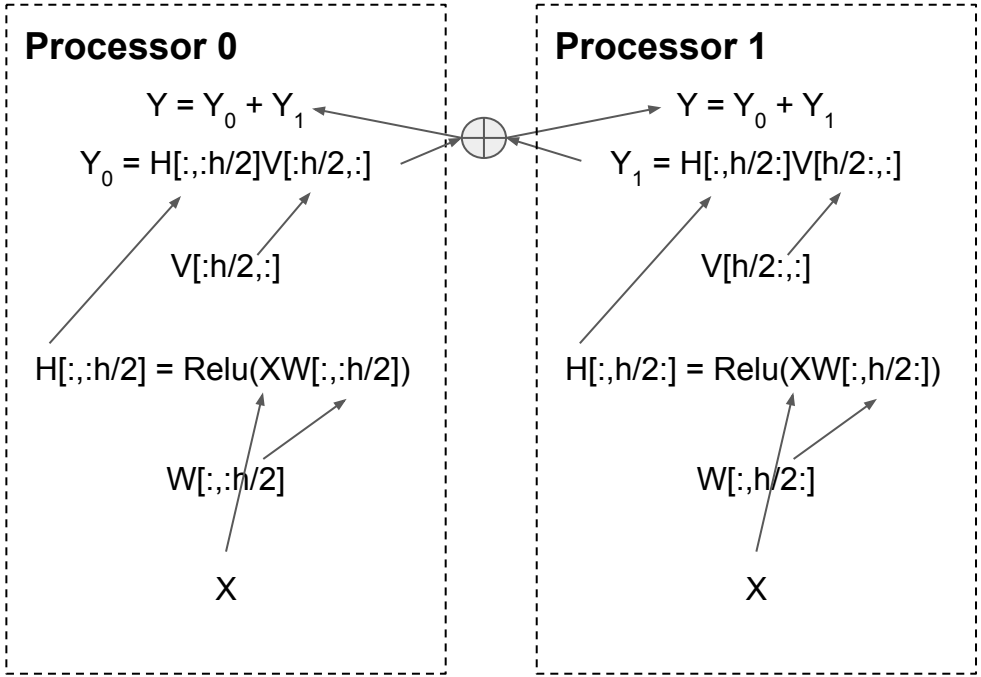


Diagrams from Noam Shazeer, March 2019

Model Parallelism: split dimension “h”

$Y = (H = Relu(XW))V$

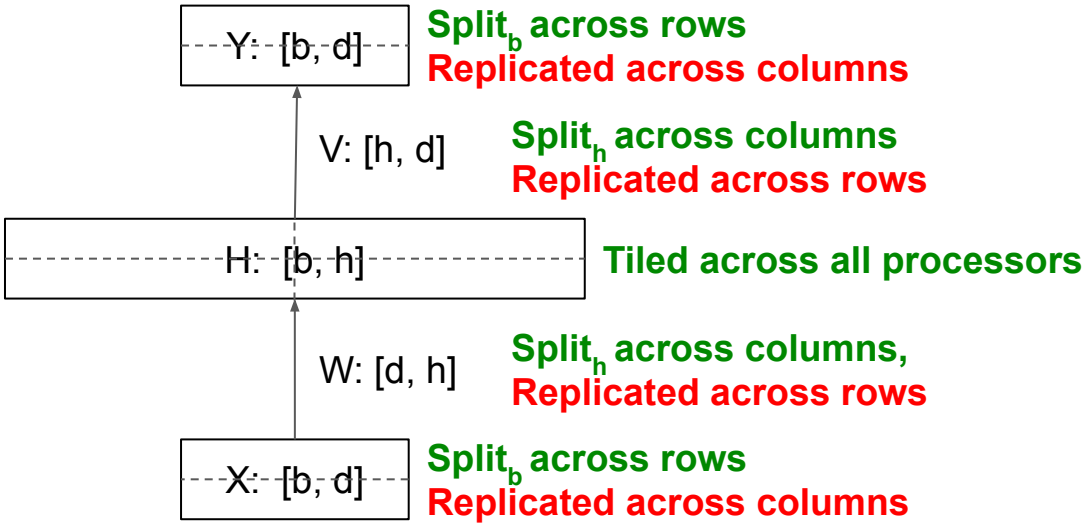


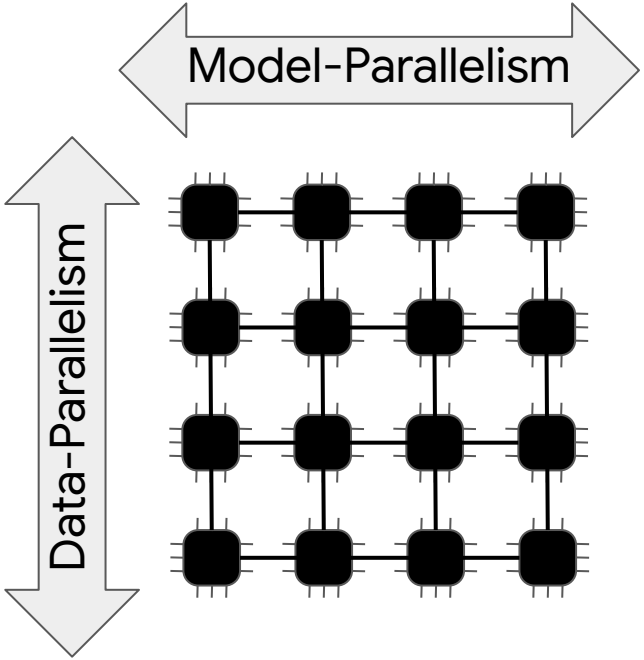


Diagrams from Noam Shazeer, March 2019

Data and Model Parallelism on 2D Mesh

Split batch “b” across rows of processors, “h” across columns of processors



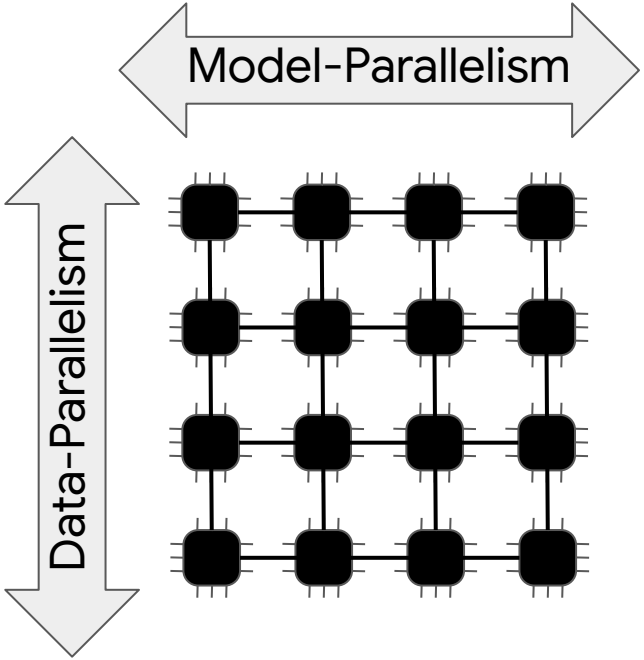


Diagrams from Noam Shazeer, March 2019

Data and Model Parallelism on 2D Mesh

Split batch “b” across rows of processors, “h” across columns of processors

Notes
Unlike previous model-parallel approaches, does not require code changes. At the moment, still requires exotic hardware.



Diagrams from Noam Shazeer, March 2019

Additional References

Practical guides
tensorflow.org/tutorials/distribute/keras
tensorflow.org/guide/distributed_training

Two helpful talks
Talk by Josh Levenberg
Talk by Noam Shazeer

Mesh TensorFlow
https://github.com/tensorflow/mesh

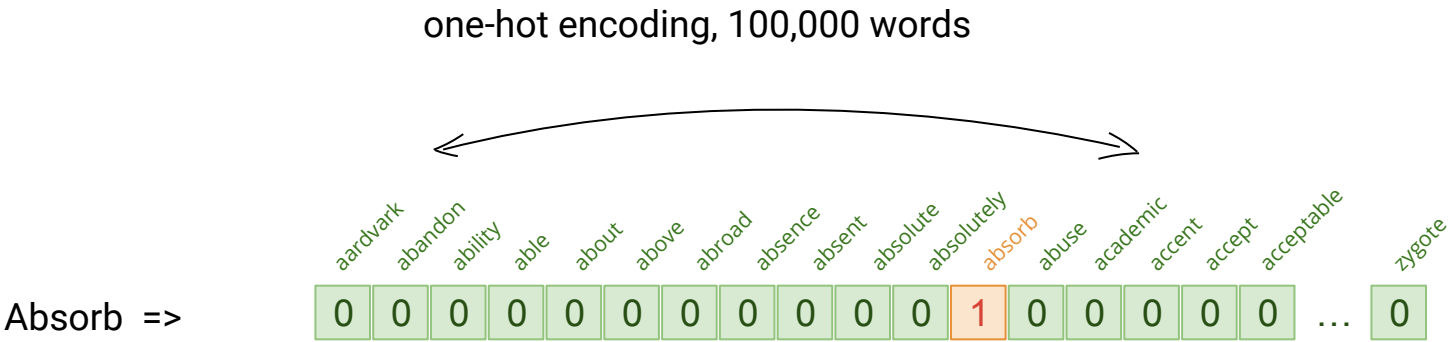
Quick intro to embeddings

You've already used them / not just for words

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np
model = VGG16(weights='imagenet', include_top=False)
img_path = 'elephant.jpg'

img = image.load_img(img_path, target_size=(224, 224)) # 224 * 224 * 3 = 150,528
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
features = model.predict(x)

# not exactly a low dimension embedding
print(features.shape) # 7 * 7 * 512 = 25,088
```

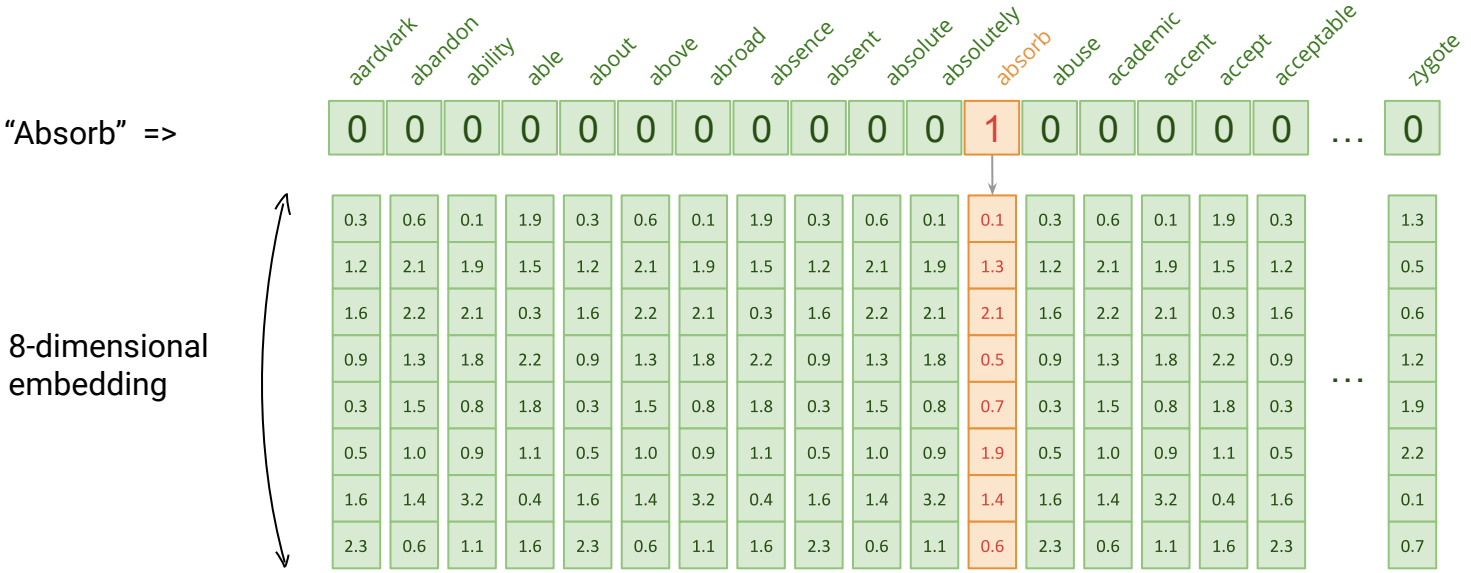
One-hot encodings

- Imagine our vocabulary is 100,000 words. A one-hot encoding results in **sparse**, **high-dimensional vector**.



Doesn't capture the relationships between words / cannot be adjusted by the classifier.

Embeddings



Notes: the embedding layer is basically a lookup table. Embedding weights begin randomly and are adjusted by the classifier by backprop (exactly as in a Dense layer). Size of the embedding is a hyperparameter. How is the lookup performed? Using a dictionary mapping from integer-encoded words -> embeddings. This is more efficient than an approach you may see written in math notation (where a one-hot vector is multiplied against a matrix to "select" a column).

Embeddings

Dense, lower-dimensional vectors learned from data.

Common sizes

- **8** to **1024**

Parameters in an Embedding layer

```
from tensorflow.keras.layers import Embedding

vocab_size, embed_dim, = 1000, 64

model = Sequential()
model.add(Embedding(vocab_size, embed_dim))
model.summary()
```

Parameters in an Embedding layer

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 64)          64000
=================================================================
Total params: 64,000
Trainable params: 64,000
Non-trainable params: 0
_________________________________________________________________
```

Shape: batch size, sentence length (in words), embedding dimension

As you would expect: vocabulary_size * embeddding_dimension.

Embeddings (not just for words)

Quick discussion: what else could we embed?

Embeddings (not just for words)

Quick discussion: what else could we embed?

- Images
- Songs
- Behaviors (based on web activity)
- Conditions (based on sensor data)
- Sentences
- Etc.

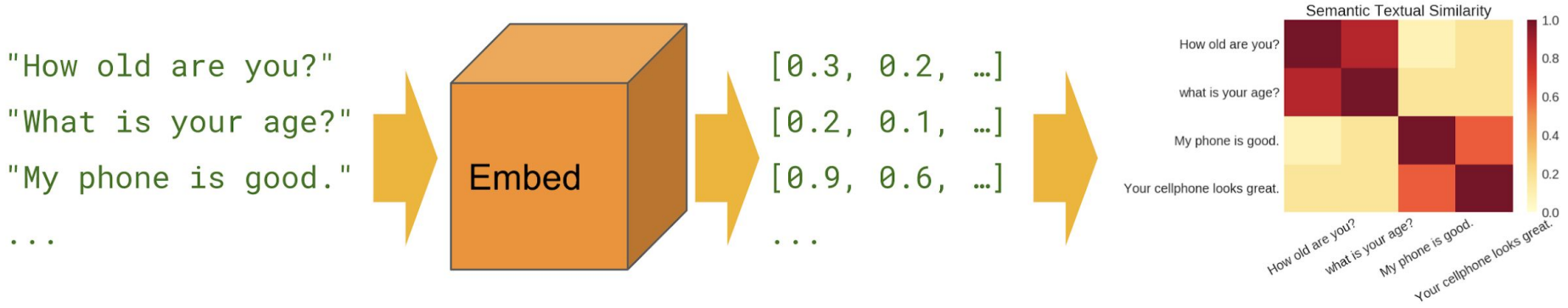
Example: pretrained word embeddings

```
%tensorflow_version 2.x

!pip install "tensorflow_hub>=0.5.0"


import tensorflow as tf
import tensorflow_hub as hub

module = "https://tfhub.dev/google/tf2-preview/nnlm-en-dim128/1"
embed = hub.KerasLayer(module)


embeddings = embed(["A long sentence.", "single-word", "http://example.com"])
print(embeddings.shape)  # (3,128)
```

tfhub.dev/google/tf2-preview/nnlm-en-dim128/1 (doc is important to understand how word embeddings are combined into one)

What else is available

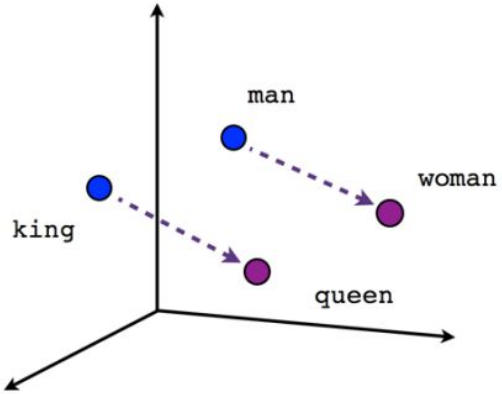
TF Hub has a bunch of word embedding models (currently being updated for TF2, a few work now). Also some interesting things: https://tfhub.dev/google/universal-sentence-encoder/2



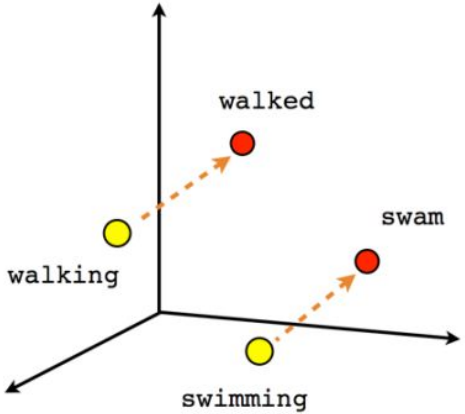
Universal Sentence Encoder

Vector arithmetic

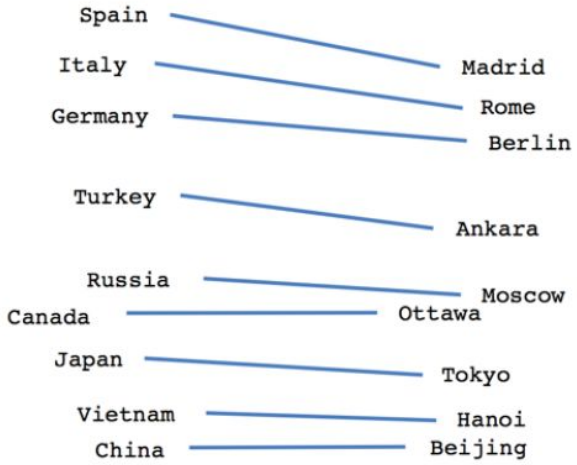
Notes: YMMV. Embeddings trained on small datasets are not likely to be as interpretable.



Male-Female



Verb tense



Country-Capital

https://www.aclweb.org/anthology/N13-1090

Transfer learning

- **Quick discussion**: how might this work with word embeddings?
- How well is it likely to work?

Transfer learning

- **Quick discussion**: how might this work with word embeddings?
- How well is it likely to work?

Notes:

- **Surprisingly, I haven't had great results with it personally (similar experience to this experiment). Though this may change when using larger embeddings.**
- **Helpful example code in Deep Learning with Python.**
- **You can check http://tensorflow.org/hub for the state of the art (although skip the TF v1 code, wait a few weeks for a hopeful upgrade to TF2).**

Break / try the embedding projector

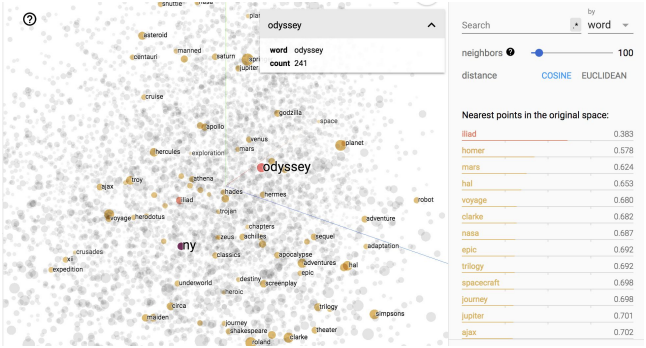
This tutorial contains instructions to train and upload embeddings

tensorflow.org/tutorials/text/word_embeddings

I wrote this a while back, if you have any suggestions, please LMK. Ideas for new tutorials:

- **Use TF Hub to retrieve embeddings for a few images (cats and dogs); project them and cluster**

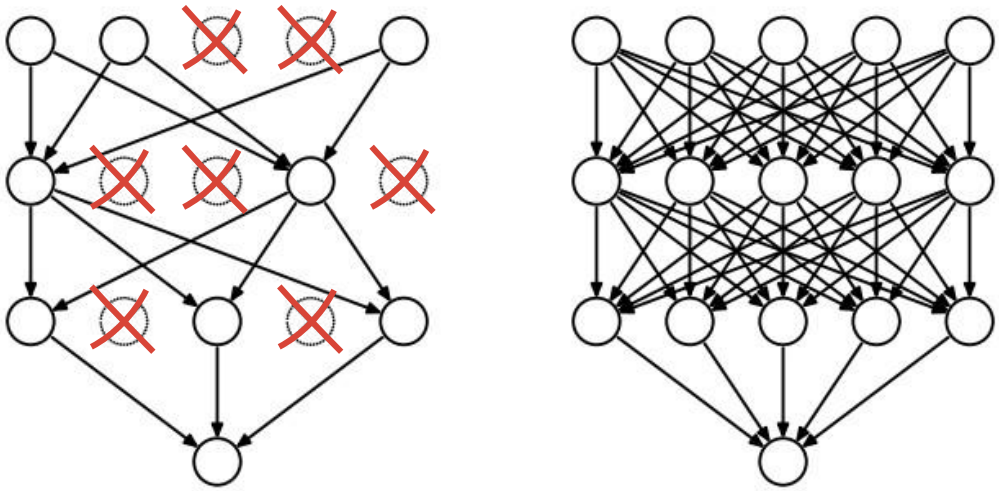
- Likewise, but for sentence embeddings



Dropout

Dropout

Note: used to drop neurons in hidden layers (not usually inputs to the network as shown in the diagram).



Dropout rate is the fraction of the activations that are zeroed out; it's usually set between 0.2 and 0.5 (left).

No activations are dropped at testing time (right).

Quick discussion: Does anyone who hasn't seen this before have an idea why it might work?

Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)

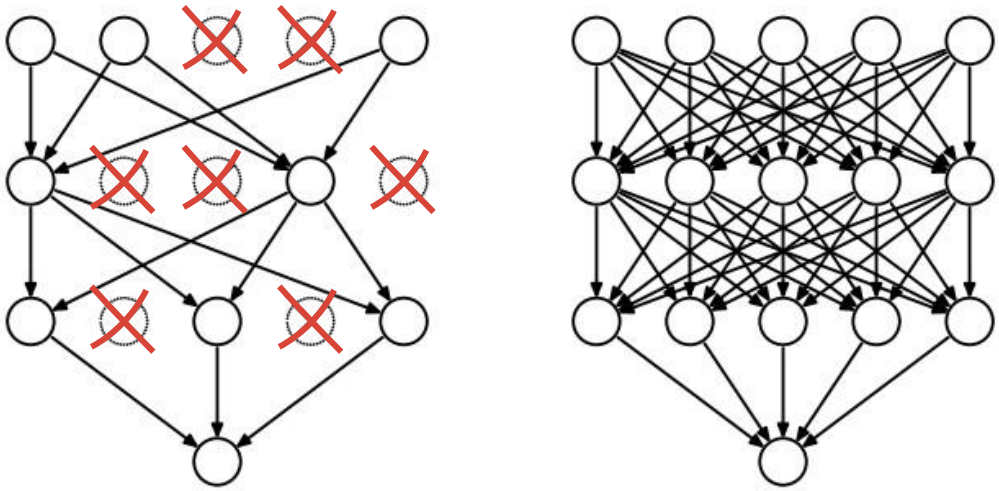
Reddit AMA with Hinton

“...I went to my bank. The tellers kept changing and I asked one of them why. He said he didn't know but they got moved around a lot. I figured it must be because it would require cooperation between employees to successfully defraud the bank. This made me realize that randomly removing a different subset of neurons on each example would prevent conspiracies and thus reduce overfitting.”

Reddit

Dropout

Note: used to drop neurons in hidden layers (not usually inputs to the network as shown in the diagram).



Dropout rate is the fraction of the activations that are zeroed out; it's usually set between 0.2 and 0.5 (left).

No activations are dropped at testing time (right).

Answer: a) Forces the model to learn redundant representations / reduces it's capacity / can only "remember" most important patterns. b) A little bit like ensembling (we're loosely training a different model on each iteration).

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Implementation (Sequential)

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(rate=0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Common pattern: use following Dense or Flatten layers.

How many Dropout layers should you use? Hyperparameter.

Best bet: start with just one, right before the output layer.

Keras Layers - Dropout

Implementation (Subclassing)

```
class MyModel(tf.keras.Model):
  def __init__(self):
    self.dense1 = layers.Dense(100)
    self.dropout = layers.Dropout(0.2)
    # ...

  def call(self, x, training=True):
    x = self.dense1(x)
    x = self.dropout(x, training=training)
    # ...
    return x

preds = model(data, training=False) # at test time
```

Tip: If you're using Dropout with the Subclassing API, remember to pass a parameter to let your model know whether it's train or test time.

Keras Layers - Dropout

Batch norm

Intuition

Consider this network

$$\text{loss} = F_2(F_1(x, \Theta_1), \Theta_2)$$

Observation: as we update Θ_1 we change the distribution F_2 sees as input.

Notes

As we train a network with SGD, keep in mind that:

1. Inputs to each layer are affected by the parameters of all preceding layers.
2. Adjusting the values of Θ_1 changes the distribution of inputs to F_2.
3. Subsequent layers must continuously adapt to new distributions of inputs → slower training.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Batch norm

Features

Batch (examples)

	F1	F2	F3	F4
E1	2	1	3	2
E2	4	1	0	4
E3	3	1	3	0
Mean	3.0	1.0	2.0	2.0
Std	0.81	0.0	1.41	1.63

Layer norm

	F1	F2	F3	F4	Mean	Std
E1	2	1	3	2	2.0	0.71
E2	4	1	0	4	2.25	1.78
E3	3	1	3	0	1.75	1.30

Layer Normalization

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Trends in framework design

Trivia

Acknowledgments

We would like to acknowledge Patrice Marcotte, Olivier Delalleau, Kyunghyun Cho, Guillaume Alain and Jason Yosinski for helpful discussions. Yann Dauphin shared his Parzen window evaluation code with us. We would like to thank the developers of Pylearn2 [11] and Theano [6, 1], particularly Frédéric Bastien who rushed a Theano feature specifically to benefit this project. Ar-

Theano: A Python framework for fast computation of mathematical expressions (2010)
Generative Adversarial Nets (2014)

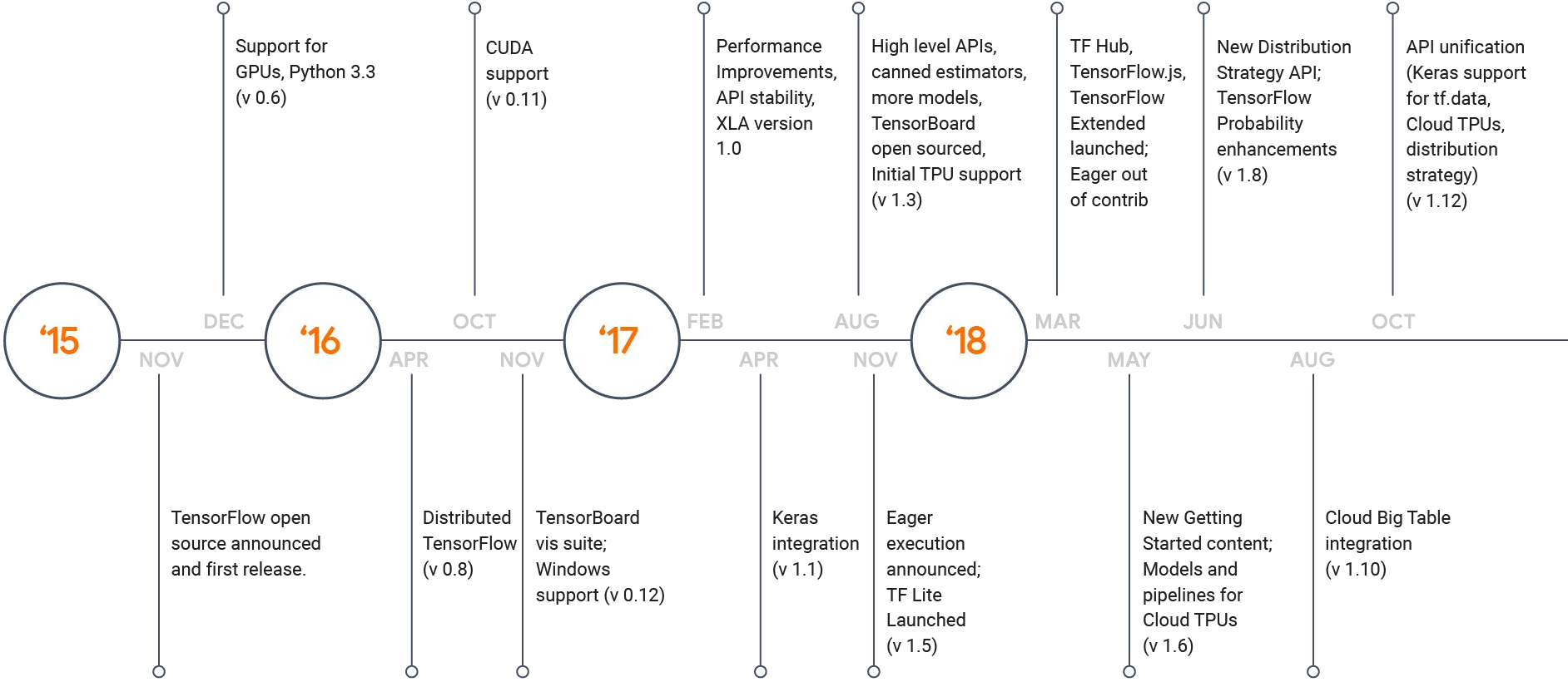
Trivia

Acknowledgments

We would like to acknowledge Patrice Marcotte, Olivier Delalleau, Kyunghyun Cho, Guillaume Alain and Jason Yosinski for helpful discussions. Yann Dauphin shared his Parzen window evaluation code with us. We would like to thank the developers of Pylearn2 [11] and Theano [6, 1], particularly Frédéric Bastien who rushed a Theano feature specifically to benefit this project. Ar-

particularly Frédéric Bastien who rushed a Theano feature specifically to benefit this project. Arnaud Bergeron provided much-needed support with LaTeX typesetting. We would also like to thank CIFAR, and Canada Research Chairs for funding, and Compute Canada, and Calcul Québec for providing computational resources. Ian Goodfellow is supported by the 2013 Google Fellowship in Deep Learning. Finally, we would like to thank Les Trois Brasseurs for stimulating our creativity.

Theano: A Python framework for fast computation of mathematical expressions (2010)
Generative Adversarial Nets (2014)


'15
NOV
TensorFlow open source announced and first release.
DEC
Support for GPUs, Python 3.3 (v 0.6)
'16
APR
Distributed TensorFlow (v 0.8)
OCT
CUDA support (v 0.11)
NOV
TensorBoard vis suite; Windows support (v 0.12)
'17
APR
Keras integration (v 1.1)
FEB
Performance Improvements, API stability, XLA version 1.0
AUG
High level APIs, canned estimators, more models, TensorBoard open sourced, Initial TPU support (v 1.3)
NOV
Eager execution announced; TF Lite Launched (v 1.5)
'18
MAY
New Getting Started content; Models and pipelines for Cloud TPUs (v 1.6)
MAR
TF Hub, TensorFlow.js, TensorFlow Extended launched; Eager out of contrib
AUG
Cloud Big Table integration (v 1.10)
JUN
New Distribution Strategy API; TensorFlow Probability enhancements (v 1.8)
OCT
API unification (Keras support for tf.data, Cloud TPUs, distribution strategy) (v 1.12)

Converging trends

- PyTorch added static graphs (via TorchScript)
- TF2 added define-by-run
- TF-Hub <-> PyTorch Hub
- Today, PyTorch added mobile support, etc

Takeaways

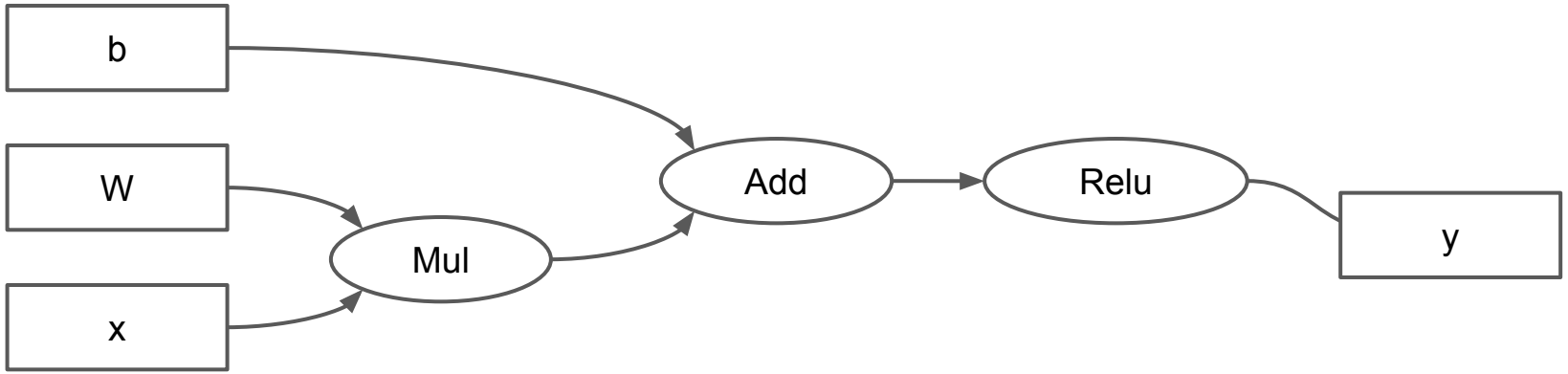
Focus on doing good work, you should be comfortable migrating from one to another without much trouble (personally, I need to begin learning Swift).

TensorFlow 1.0

Just for reference

- There’s value in understanding older code (in case you’d like to reuse some of many papers written with it), and so you can appreciate why frameworks like Keras and PyTorch became so popular, so quickly.

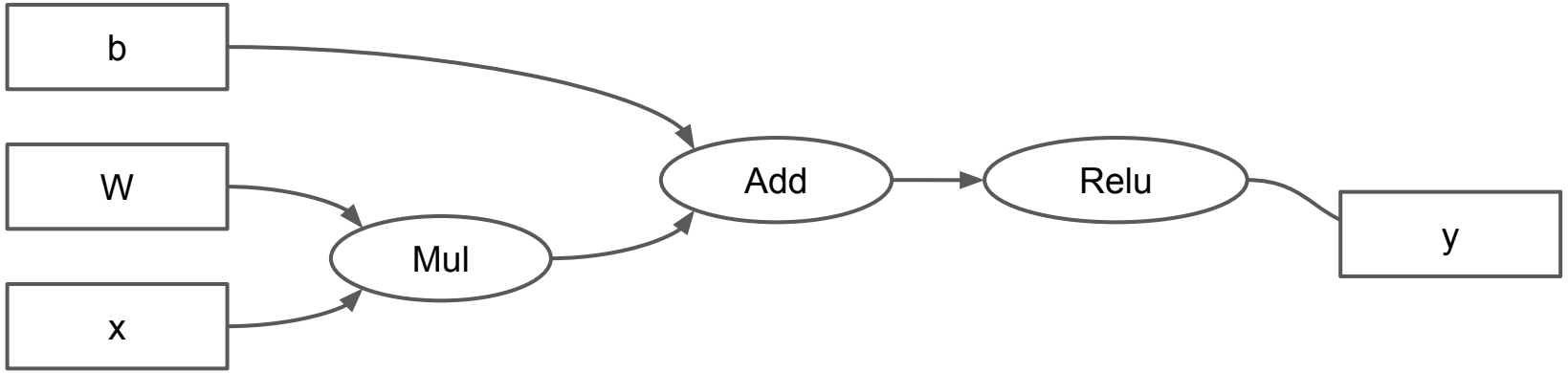
Why graphs



Why graphs

Notes

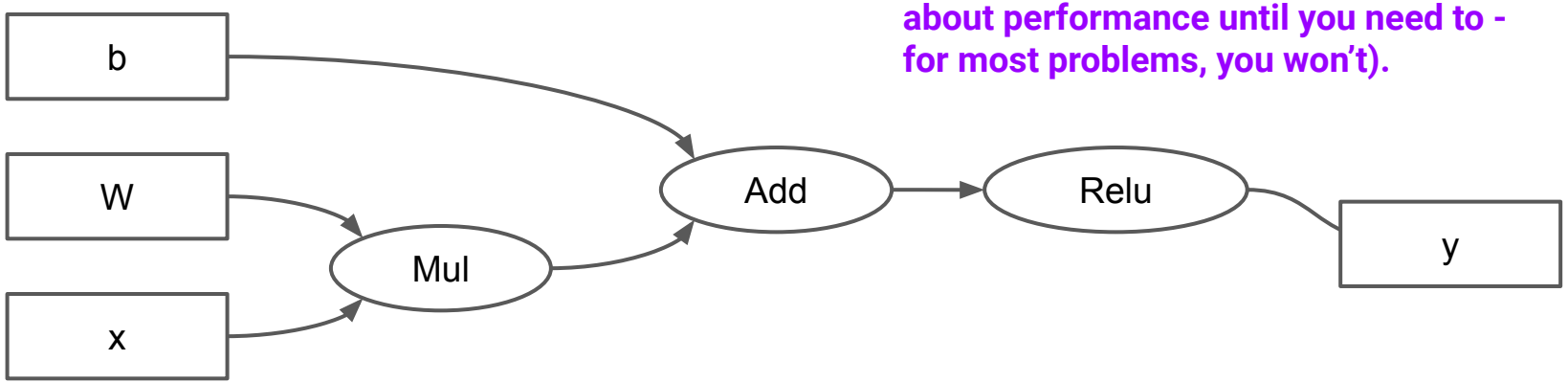
- Portability: develop in Python, but deploy to devices (like iOS, or Android) that don't support it.



Why graphs

Notes

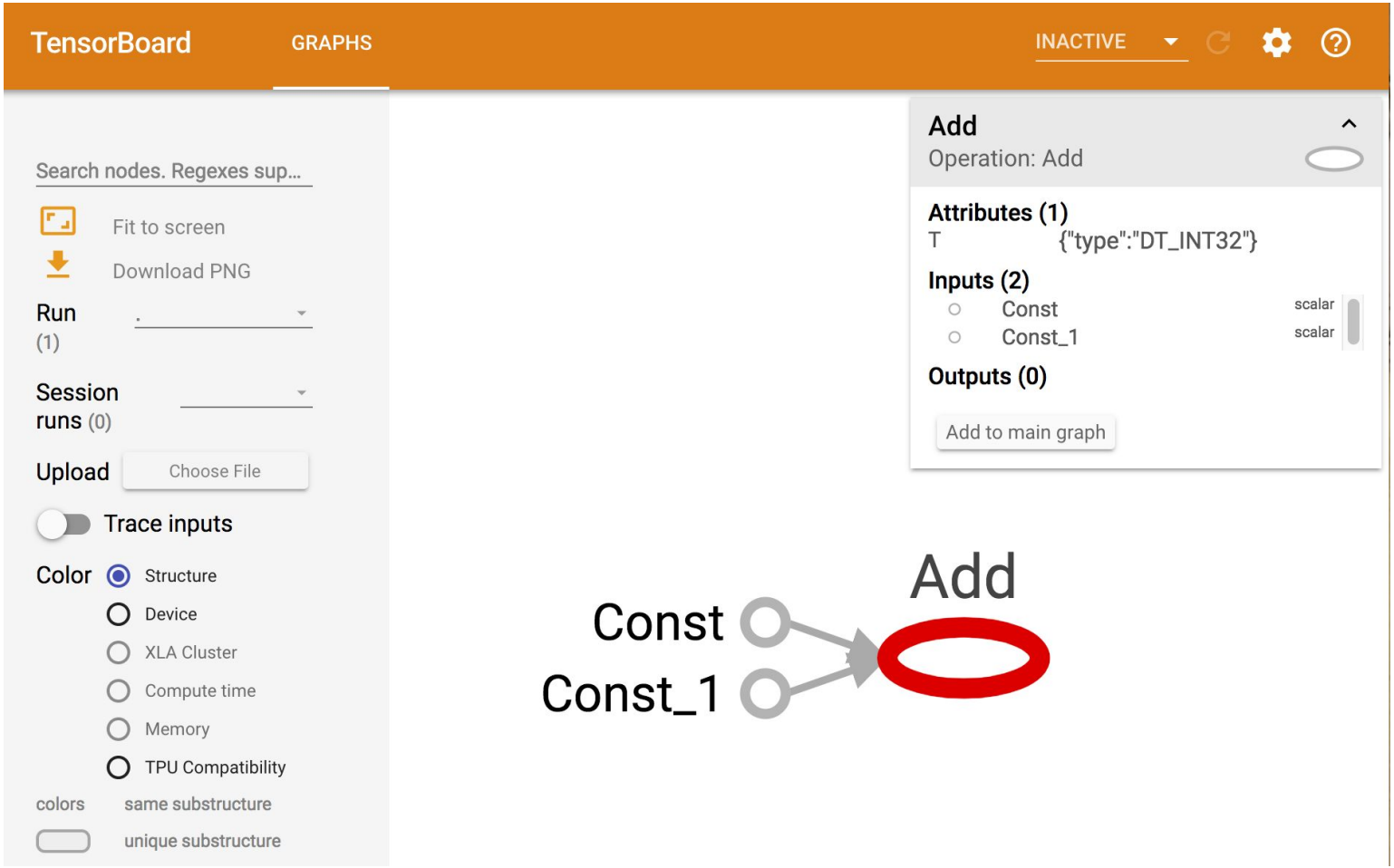
- Portability: develop in Python, but deploy to devices (like iOS, or Android) that don't support it.
- Performance: graphs can be optimized and distributed (as always, don't worry about performance until you need to - for most problems, you won't).



What's a graph?

1.0 code

```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)

                                    # $ tensorboard --logdir=graphs/
                                    # then browse to http://localhost:6006

writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
with tf.Session() as sess:
    print(sess.run(x))
writer.close()
```


TensorBoard
GRAPHS
INACTIVE
Search nodes. Regexes sup...
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
Add
Operation: Add
Attributes (1)
T
{"type":"DT_INT32"}
Inputs (2)
Const
scalar
Const_1
scalar
Outputs (0)
Add to main graph
Add
Const
Const_1

1.0 code

Tensors in graph mode are symbolic

1.0 code

```
import tensorflow as tf
w = tf.Variable(10)
print(w)
# <tf.Variable shape=() dtype=int32_ref>
```

Note: printing w does not print 10!

Sessions

1.0 code

```
import tensorflow as tf
w = tf.Variable(10)
with tf.Session() as sess:
    print(sess.run(w))
```

FailedPreconditionError: Attempting to use uninitialized value

Variables must be initialized before use

1.0 code

```
import tensorflow as tf
w = tf.Variable(10)
with tf.Session() as sess:
    sess.run(w.initializer)
    print(sess.run(w))     # 10
```

Why doesn't this work?

1.0 code

```
import tensorflow as tf
w = tf.Variable(10)
with tf.Session() as sess:
    sess.run(w.initializer)
    w.assign(100)
    print(sess.run(w)) # 10
```

The op needs to be executed first

1.0 code

```
import tensorflow as tf
w = tf.Variable(10)
with tf.Session() as sess:
    sess.run(w.initializer)
    assign_op = w.assign(100)
    sess.run(assign_op)
    print(sess.run(w)) # 100
```

Not deterministic

1.0 code

```
X, y = tf.Variable(1.0), tf.Variable(1.0)
add_op = x.assign(x + y)
div_op = y.assign(y / 2)
init = tf.global_variables_initializer()
with tf.Session() as sess:
 init.run()
 for iteration in range(50):
        sess.run([add_op, div_op])
 print(sess.run(w)) # run 1: 2.0, run 2: 2.75
```

No dependency seen between operations, run in parallel. Would need to be explicitly called out.

Feeding and fetching

1.0 code

```
# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
add_op = tf.add(a,b)

with tf.Session() as sess:
    print(sess.run(add_op, {a: [1, 2, 3]})) # [6, 7, 8]
```

If you have a bunch of data to feed, you can do so in a loop - causes performance penalties (from Python -> TF -> Python -> TF)

XLA

```
tmp = tf.add(x, y)
result = tf.multiply(tmp, z)
```

```
for (i = 0; i < element_count; i++) {
  tmp[i] = x[i] + y[i];
}

for (i = 0; i < element_count; i++) {
  result = tmp[i] * z[i];
}
```

```
for (i = 0; i < element_count; i++) {
  result = (x[i] + y[i]) * z[i];
}
```

Compress these two loops into one.

Eager (aka normal) execution

1.0 code

- Define-by-run
- Easier to develop and debug

See also RFC for TensorFlow 2.0: Functions, not Sessions.

Notes: Writing graph-level code directly is a bit like working with a compiler. With eager mode, TensorFlow felt more like working with NumPy.

Eager

1.0 code

```
import tensorflow as tf


# must be called once at program startup
tf.enable_eager_execution()
```

Multiplying a matrix by itself

```
a = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])
print(tf.matmul(a, a))

tf.Tensor(
[[ 7. 10.]
 [15. 22.]], shape=(2, 2), dtype=float32)
```

No sessions, no graphs, no placeholders!

NumPy compatibility

```
a = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])

foo = tf.matmul(a, a)

print(foo.numpy())

array([[ 7., 10.],
       [15., 22.]], dtype=float32)
```

Works both ways

```
a = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])

foo = tf.matmul(a, a)
bar = foo.numpy()
tf.reduce_sum(bar)
```

TensorFlow operations work on NumPy data

```
<tf.Tensor: id=58, shape=(), dtype=float32, numpy=54.0>
```

NumPy operations accept tf.Tensor arguments

```
a = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])

print(type(a)) # <type 'EagerTensor'>
bar = np.dot(a,a)
print(type(bar)) # <type 'numpy.ndarray'>
```

Acceleration

Note: automatic device placement is supported for most ops (so you do need to specify whether to use a GPU in practice).

```
n = 1000
def time_matmul(x):
  %timeit tf.matmul(x, x)

# Force execution on CPU
with tf.device("CPU:0"):
  x = tf.random_uniform([n, n])
  assert x.device.endswith("CPU:0")
  time_matmul(x)
```

18 ms per loop

```
# Force execution on GPU #0
if tf.test.is_gpu_available():
  with tf.device("GPU:0"): # Or GPU:N
    x = tf.random_uniform([n, n])
    assert x.device.endswith("GPU:0")
    time_matmul(x)
```

1.1 ms per loop

Gradients

Every operation (like tf.multiply, and more complex ones, like tf.nn.relu) have an associated gradient function.

To take the gradient of a user defined function:

- Operations recorded on a tape.
- Tape is played back in reverse.
- Grad functions used to compute the gradient.

Example

```
@ops.RegisterGradient("Square")
def _SquareGrad(op, grad):
  x = op.inputs[0]
  y = constant_op.constant(2.0, dtype=x.dtype)
  return math_ops.multiply(grad, math_ops.multiply(x, y))
```

Some details omitted

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/math_grad.py

Derivative of a function

1.0 code

```
import tensorflow as tf
tf.enable_eager_execution()
tfe = tf.contrib.eager

def f(x):
  return tf.square(x)

grads = tfe.gradients_function(f)
grads(3.0) # 6.0
```

Gradient tapes

1.0 code

```
c = tfe.Variable([[2.0]])
d = tfe.Variable([[3.0]])
with tf.GradientTape() as tape:
  loss = c * d

grad = tape.gradient(loss, d)
print(grad) # 2.0
```

Traininable variables (created by tf.contrib.eager.Variable) are automatically watched.

Can also manually ‘watch’ other tensors with tape.watch()

You could switch between eager and graph mode

1.0 code

```
import tensorflow as tf
tf.enable_eager_execution()

print(tf.executing_eagerly()) # True

graph = tf.Graph()
with graph.as_default():
    print(tf.executing_eagerly()) # False
```

But don't. Just stay in eager unless you have a special reason not to.

You could also compile functions

1.0 code

```
import tensorflow as tf
tf.enable_eager_execution()

@tf.contrib.eager.defun
def square_sum(x, y):
    return tf.square(x+y)

result = square_sum(2., 3.)
print(result.numpy()) # 25
```

TensorFlow 2.0

Removed

- session.run
- placeholders
- tf.control_dependencies
- tf.global_variables_initializer
- tf.cond, tf.while_loop

Added

- tf.function, AutoGraph

This code runs “eagerly” -- or as you would expect in regular Python. This is the default in TensorFlow 2.0 unless you explicitly specify otherwise.

```
def add(a, b):
  return a + b

add(tf.ones([2, 2]), tf.ones([2, 2])) # [[2., 2.], [2., 2.]]
```

```
# A function is like an op


@tf.function
def add(a, b):
  return a + b


add(tf.ones([2, 2]), tf.ones([2, 2])) # [[2., 2.], [2., 2.]]
```

Adding this annotation causes code in the function be “compiled” and run in graph mode. For certain functions, this can accelerate it significantly, and makes it possible to deploy to devices without a Python interpreter.

```
# Let's make this faster
lstm_cell = tf.keras.layers.LSTMCell(10)



def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2

# warm up
lstm_cell(input, state); fn(input, state)

# "benchmark"
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
```

Always take benchmarks with a grain of salt (they depend on the code, hardware, network, etc). This is a simple example just to show the idea.

```python
# Let's make this faster
lstm_cell = tf.keras.layers.LSTMCell(10)

@tf.function
def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2

# warm up
lstm_cell(input, state); fn(input, state)

# "benchmark"
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
timeit.timeit(lambda: fn(input, state), number=10) # 0.004
```

Always take benchmarks with a grain of salt (they depend on the code, hardware, network, etc). This is a simple example just to show the idea.

+/- 10x improvement (don't take that to imply this will always be the case!)

```
# tf.function is polymorphic

@tf.function
def add(a, b):
  return a + b

add(tf.ones([2, 2]), tf.ones([2, 2])) # [[2., 2.], [2., 2.]]
```

This code will work with ints, floats, etc.

```
# tf.function handles complex control flow

@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    x = tf.tanh(x)
  return x

f(tf.random.uniform([10]))  # Works!
```

Autograph (or, how to never write assembly-like code again)

```
print(tf.autograph.to_code(f)) # Autograph in action
```

AutoGraph is a Python-Python compiler, which produces code the TF-backend can compile / optimize / and accelerate in C++.

This is not a complex system at all...

```
print(tf.autograph.to_code(f)) # Autograph in action
...
def tf__f(x):
  def loop_test(x_1):
    with ag__.function_scope('loop_test'):
      return ag__.gt(tf.reduce_sum(x_1), 1)
  def loop_body(x_1):
    with ag__.function_scope('loop_body'):
      with ag__.utils.control_dependency_on_returns(tf.print(x_1)):
        tf_1, x = ag__.utils.alias_tensors(tf, x_1)
        x = tf_1.tanh(x)
        return x,
  x = ag__.while_stmt(loop_test, loop_body, (x,), (tf,))
  return x
```

AutoGraph is a Python-Python compiler, which produces code the TF-backend can compile / optimize / and accelerate in C++.

This is not a complex system at all...

```python
# Controlling autograph


@tf.function
def f(x):
  for i in range(10): # Static python loop, not converted
    do_stuff()
  for i in tf.range(10): # Depends on a tensor, converted
    do_stuff()
```

Python statements like “if” and “while” are converted automatically. For other things, use tf.* when in doubt.

For next time

Reading

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting
- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- https://eng.uber.com/horovod/