Reading data, performance and distributed learning

(TensorFlow case study)

Lecture plan:

- CNN leftovers (adversarial examples, style transfer, data augmentation)
- Performance issues (potential bottlenecks)
- Reading data
- Distributed learning (model parallelism, data parallelism)

Administrative:

- Assignment 2 is out, it requires much more work than assignment 1
- Assignment 3 is also out, Maciej Klimek will be here during lab session, you can ask questions.
- Due to assignments, there are no other tasks for today’s lab session.
- Oral exam on 21.06 (and possibly 22.06)

Adversarial examples

Adversarial examples

We have seen how to optimize over the input image to maximize any class score, can we use this approach to “fool” convnets?

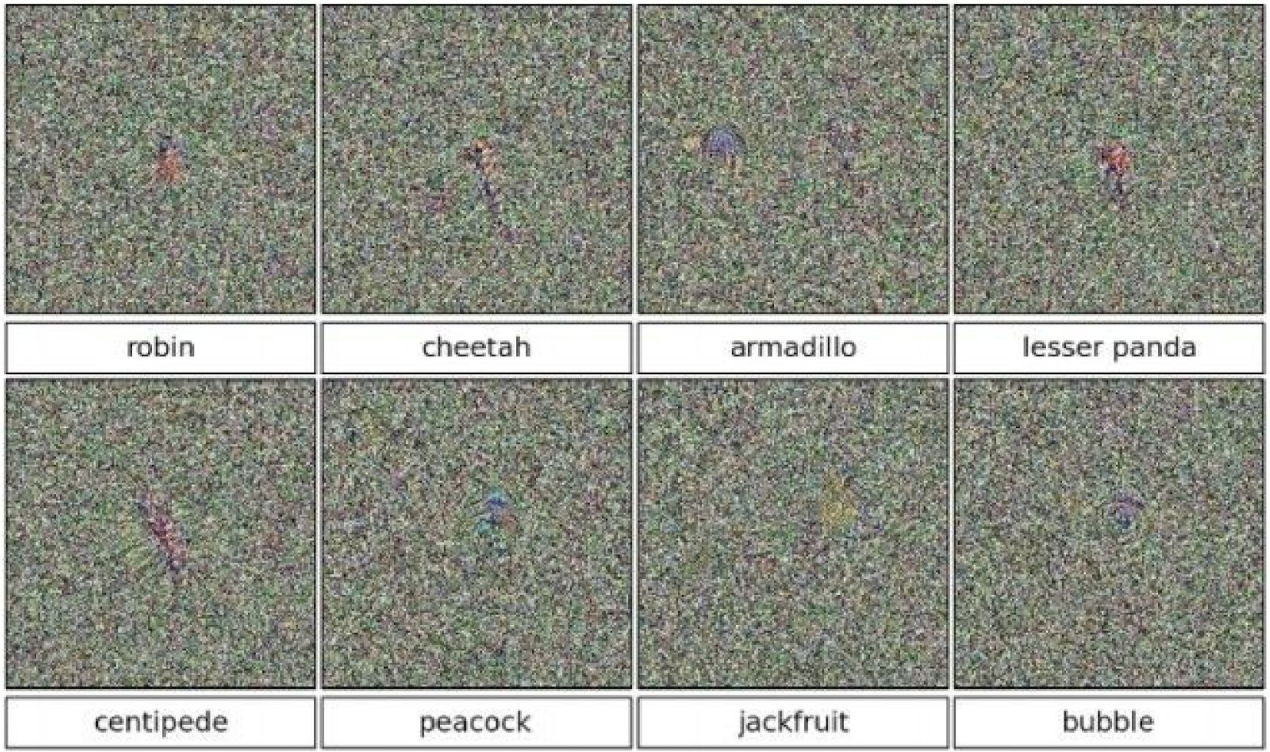
Answer: unfortunately yes.

[Intriguing properties of neural networks, Szegedy et al., 2013]

Slide taken from CS231n@stanford

[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]

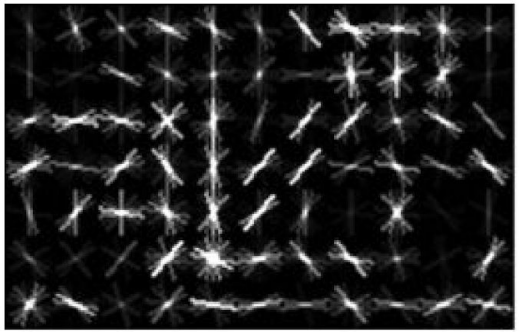
>99.6% confidences



Slide taken from CS231n@stanford

These kinds of results were around even before ConvNets…

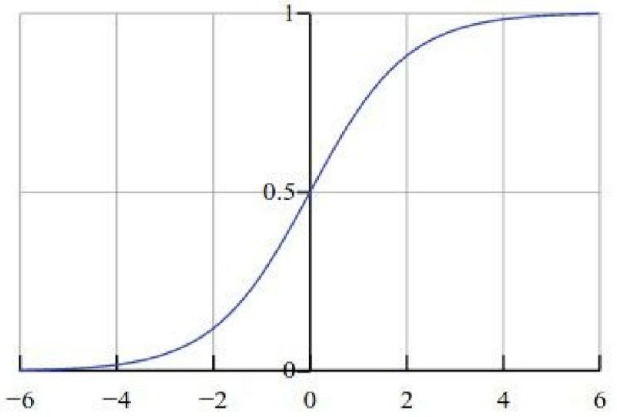
[Exploring the Representation Capabilities of the HOG Descriptor, Tatu et al., 2011]



Identical HOG represention

Lets fool a binary linear classifier: (logistic regression)

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$



Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y = 0 \mid x; w, b) = 1 - P(y = 1 \mid x; w, b)$. Hence, an example is classified as a positive example (y = 1) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$.

Slide taken from CS231n@stanford

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Slide taken from CS231n@stanford

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	?	?	?	?	?	?	?	?	?	?

← input example

← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Slide taken from CS231n@stanford

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

i.e. we improved the class 1 probability from 5% to 88%

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Slide taken from CS231n@stanford

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

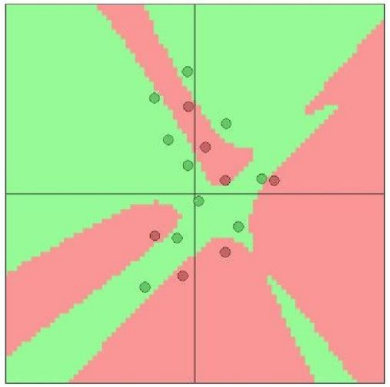
i.e. we improved the class 1 probability from 5% to 88%

This was only with 10 input dimensions. A 224x224 input image has 150,528.

(It's significantly easier with more numbers, need smaller nudge for each)

Slide taken from CS231n@stanford

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES
[Goodfellow, Shlens & Szegedy, 2014]



“primary cause of neural networks’ vulnerability to adversarial perturbation is their **linear nature**“
(and very high-dimensional, sparsely-populated input spaces)

In particular, this is not a problem with Deep Learning, and has little to do with ConvNets specifically. Same issue would come up with Neural Nets in any other modalities.

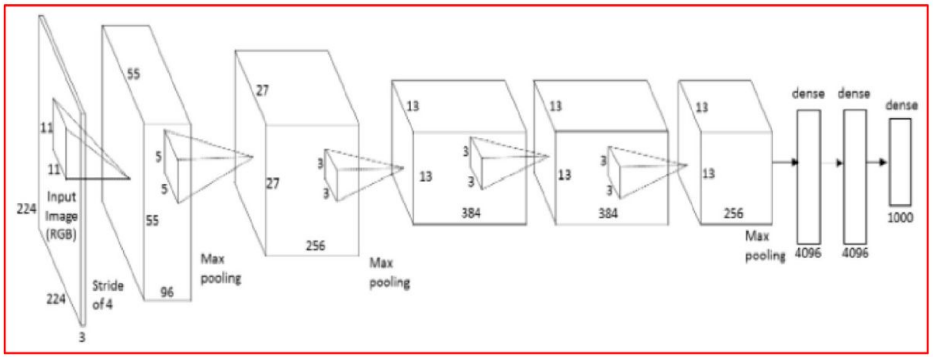
Style transfer

NeuralStyle

[A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, 2015]
good implementation by Justin in Torch:
https://github.com/jcjohnson/neural-style

Slide taken from CS231n@stanford

Step 1: Extract **content targets** (ConvNet activations of all layers for the given content image)



content activations

e.g.
at CONV5_1 layer we would have a [14x14x512] array of target activations

Slide taken from CS231n@stanford

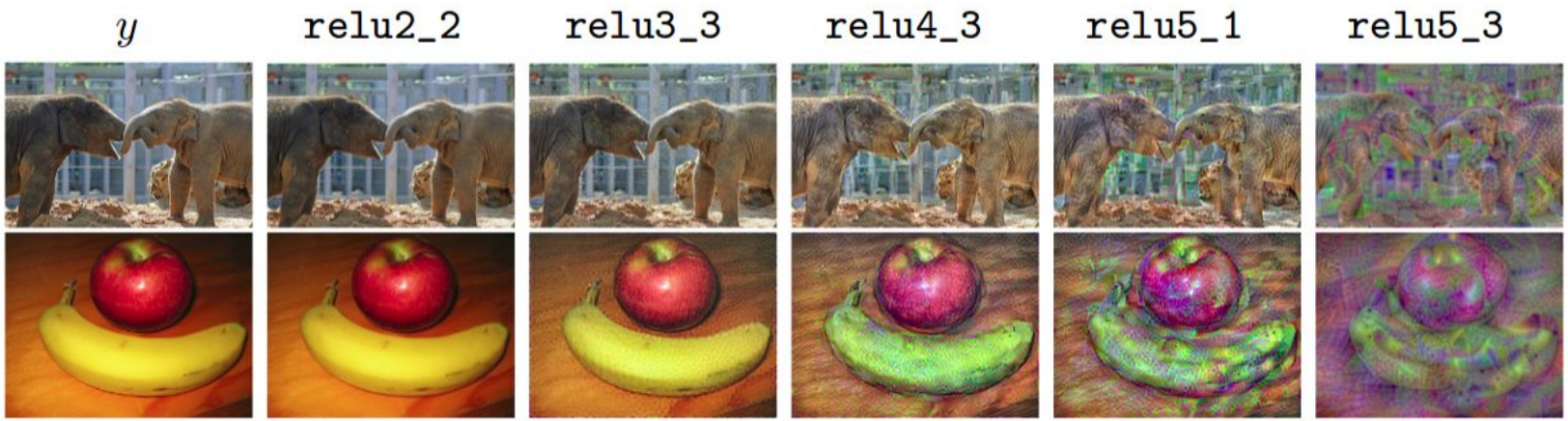
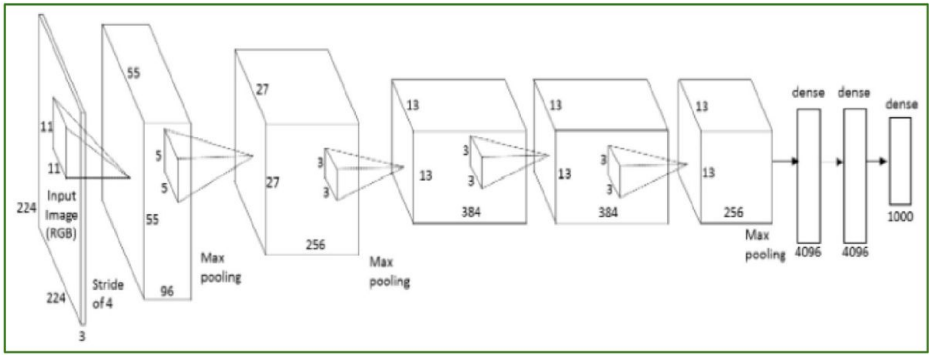


Fig. 3. Similar to [6], we use optimization to find an image $\hat{y}$ that minimizes the feature reconstruction loss $\ell^{\phi,j}_{feat}(\hat{y}, y)$ for several layers j from the pretrained VGG-16 loss network ϕ. As we reconstruct from higher layers, image content and overall spatial structure are preserved, but color, texture, and exact shape are not.

https://arxiv.org/abs/1603.08155

Step 2: Extract **style targets** (Gram matrices of ConvNet activations of all layers for the given style image)



style gram matrices

$$G = V^{\mathrm{T}} V$$

e.g.
at CONV1 layer (with [224x224x64] activations) would give a [64x64] Gram matrix of all pairwise activation covariances (summed across spatial locations)

Slide taken from CS231n@stanford

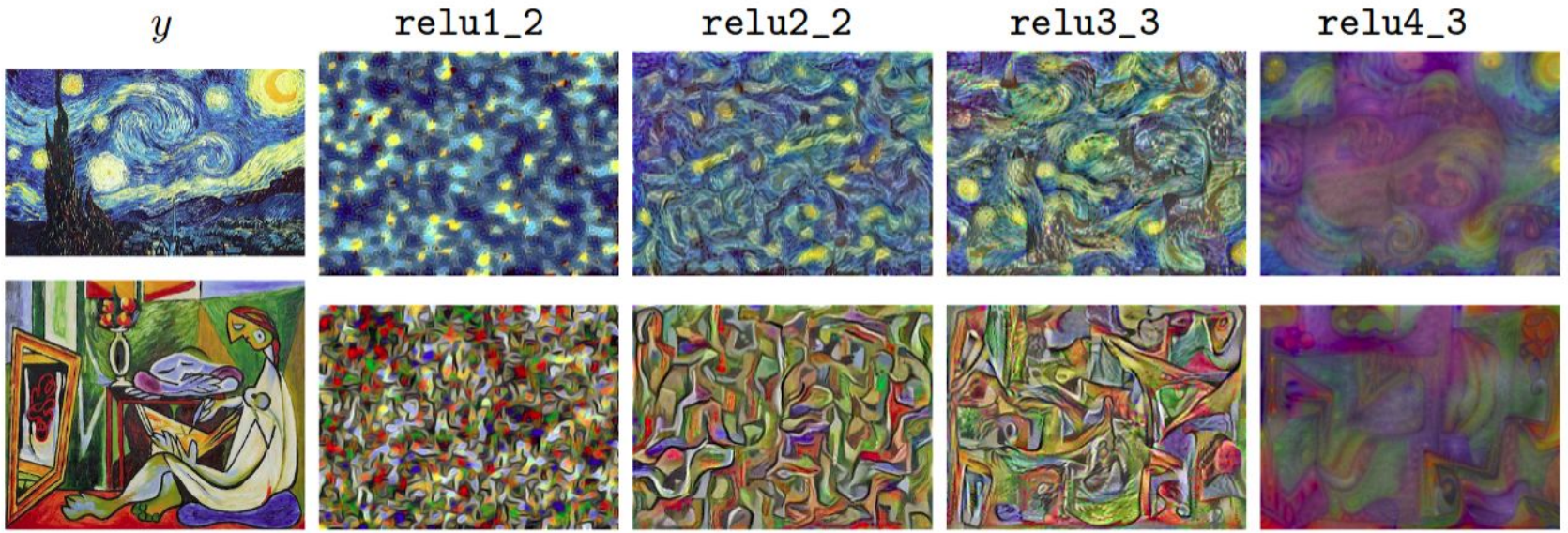


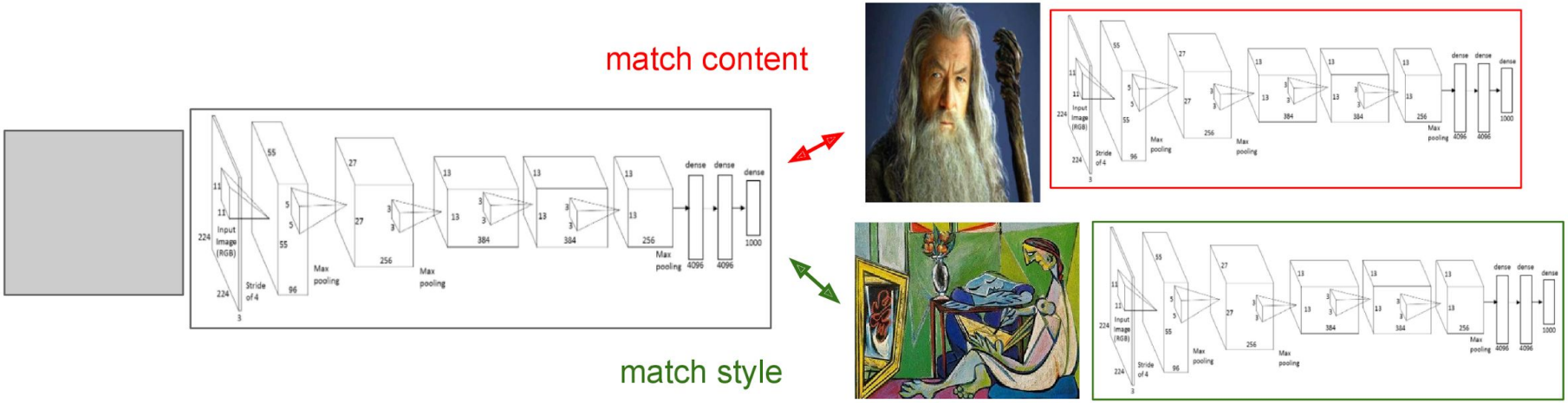
Fig. 4. Similar to [10], we use optimization to find an image $\hat{y}$ that minimizes the style reconstruction loss $\ell^{\phi,j}_{style}(\hat{y}, y)$ for several layers j from the pretrained VGG-16 loss network ϕ. The images $\hat{y}$ preserve stylistic features but not spatial structure.

https://arxiv.org/abs/1603.08155

Step 3: Optimize over image to have:

- The **content** of the content image (activations match content)
- The **style** of the style image (Gram matrices of activations match style)

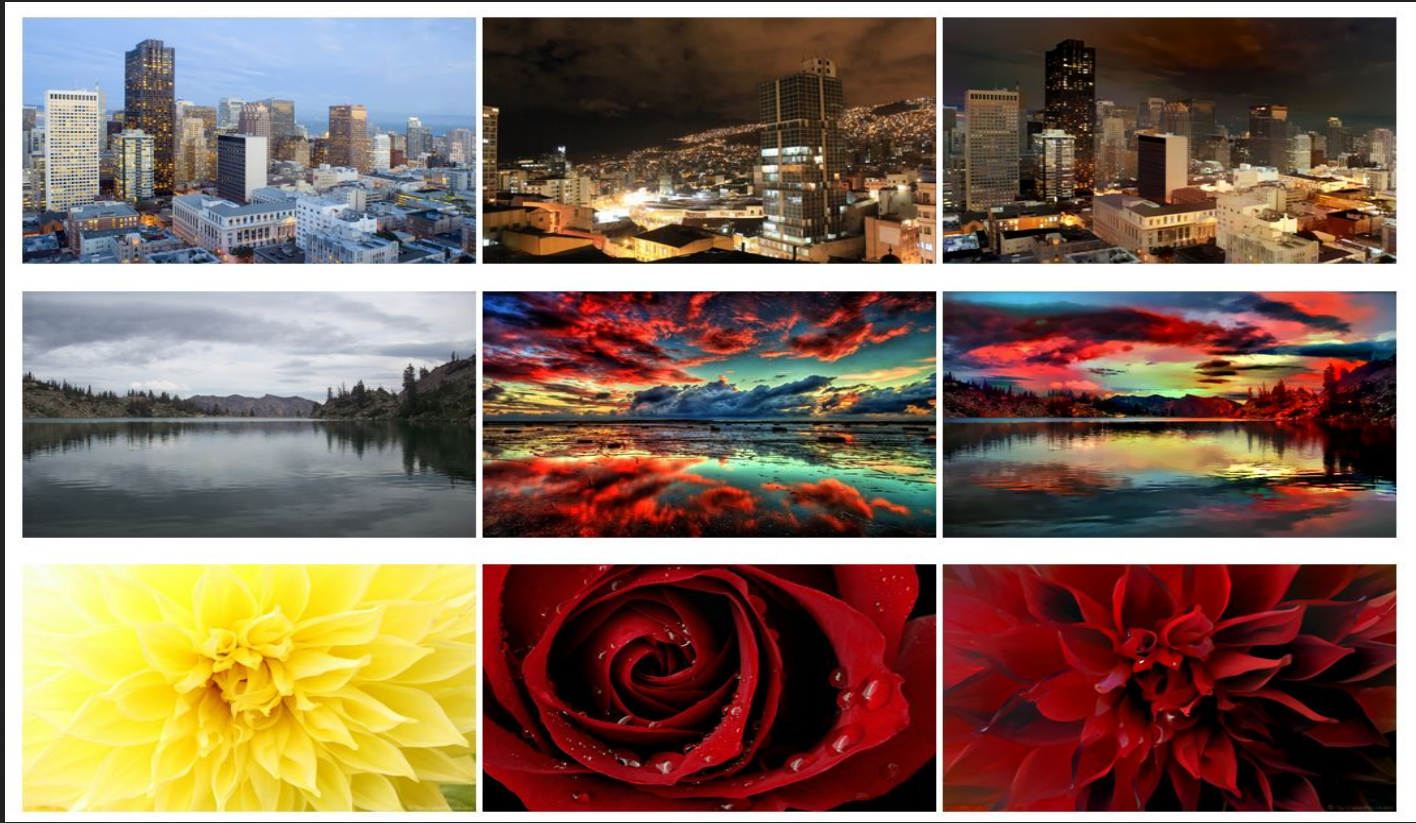
$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$



Slide taken from CS231n@stanford

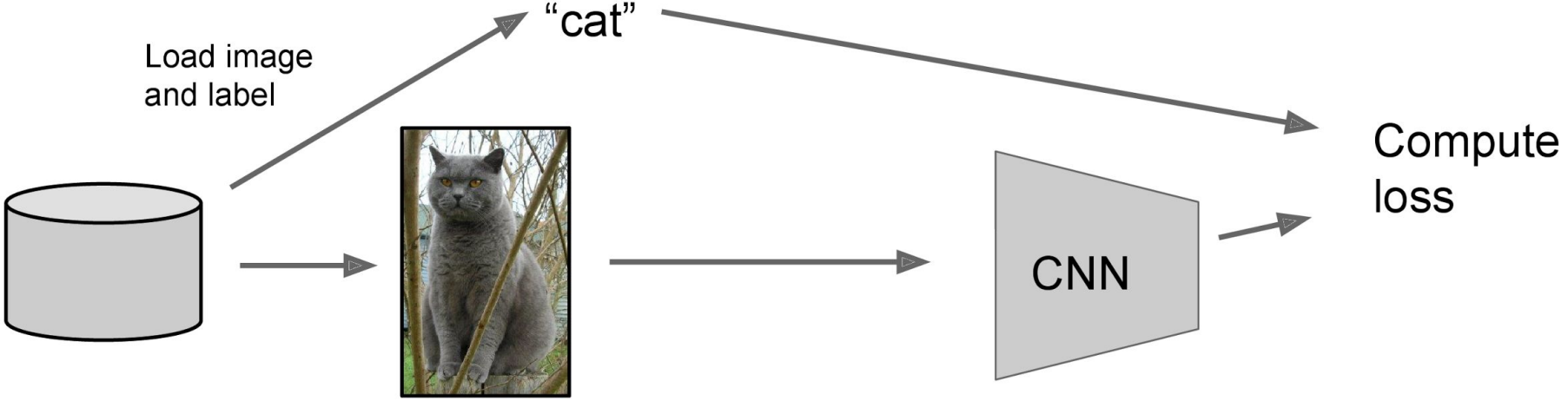
Deep Photo Style Transfer

Fujun Luan, Sylvain Paris, Eli Shechtman, Kavita Bala (03.2017) https://github.com/luanfujun/deep-photo-styletransfer



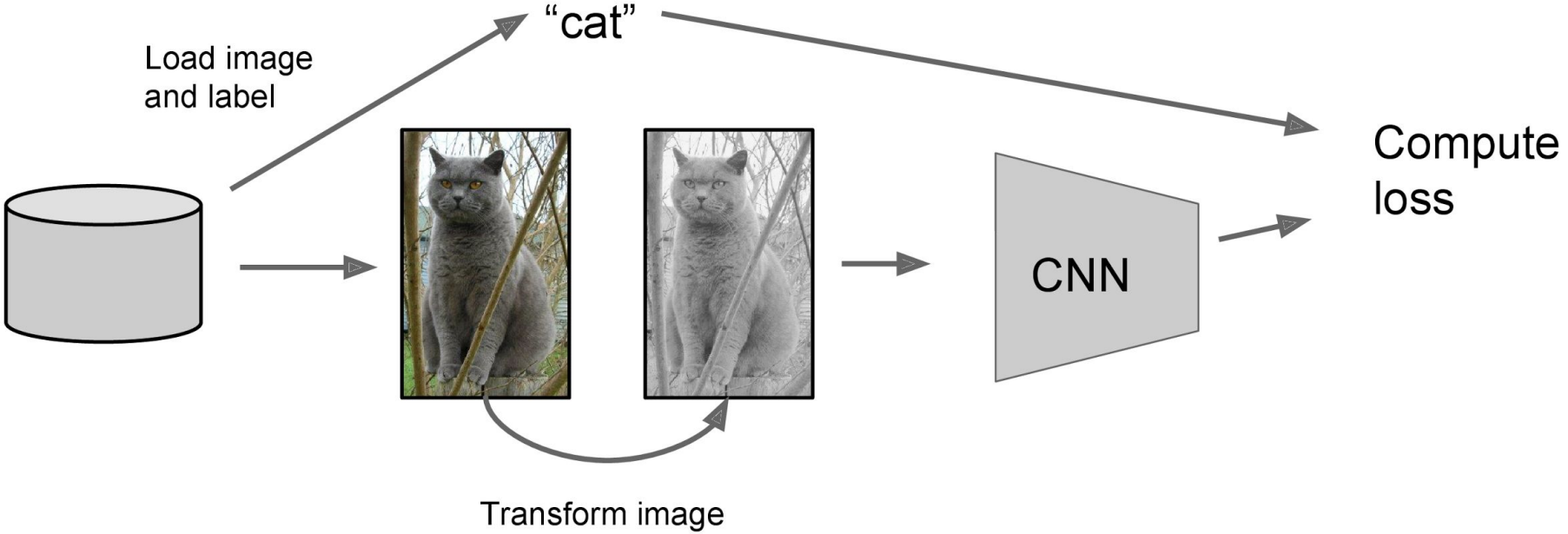
Data augmentation

Data Augmentation



Slide taken from CS231n@stanford

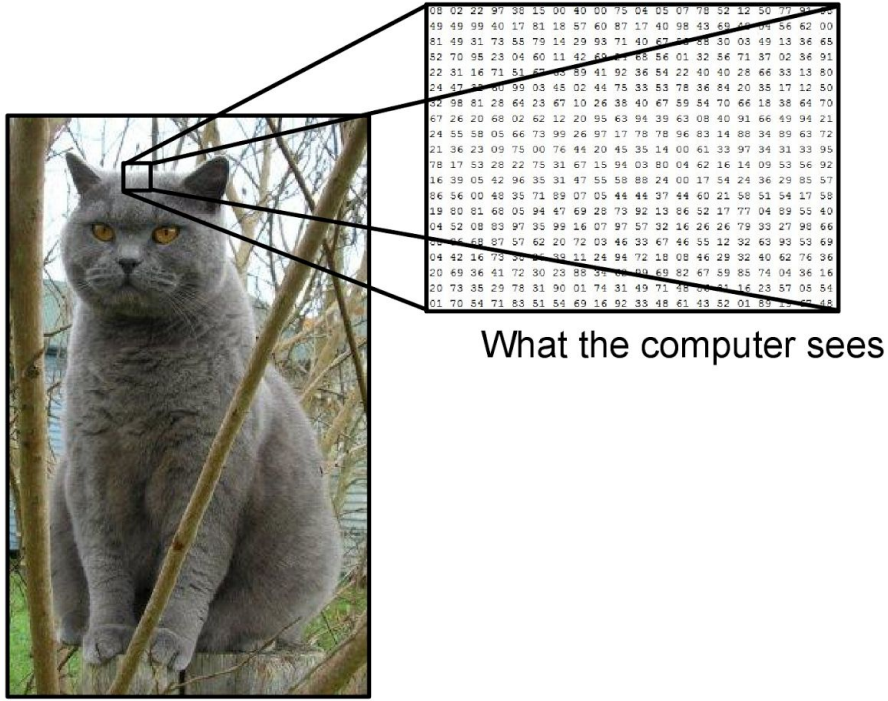
Data Augmentation



Slide taken from CS231n@stanford

Data Augmentation

- Change the pixels without changing the label
- Train on transformed data
- VERY widely used

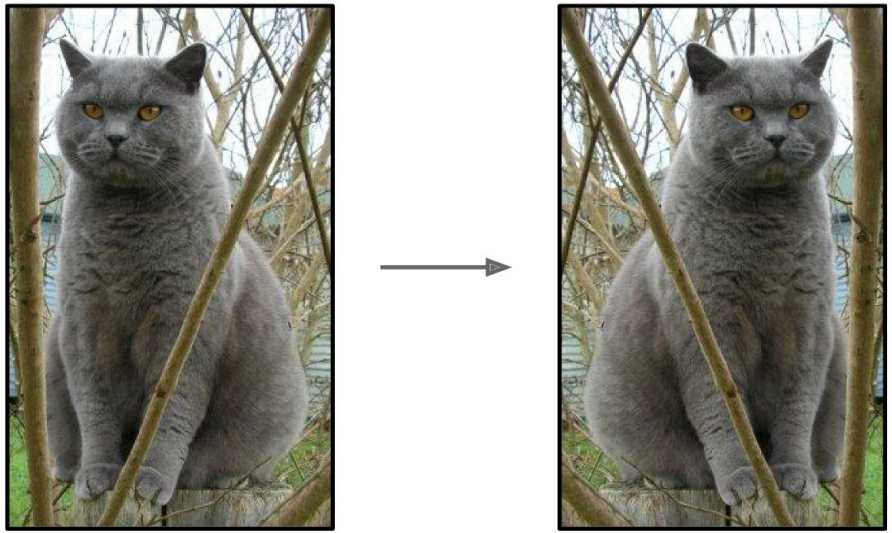


What the computer sees

Slide taken from CS231n@stanford

Data Augmentation

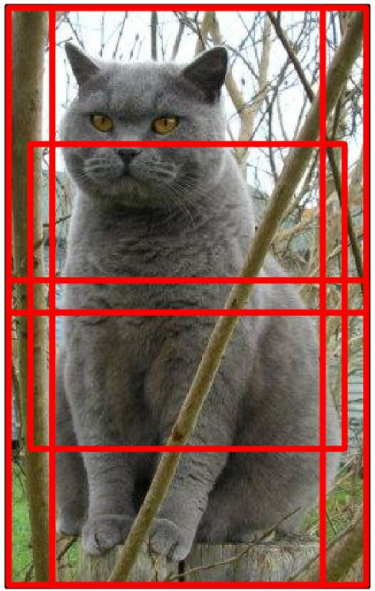
1. Horizontal flips



Slide taken from CS231n@stanford

Data Augmentation

2. Random crops/scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

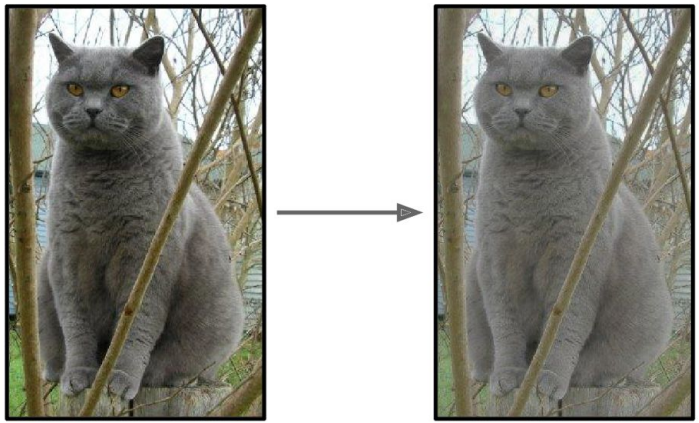
ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation

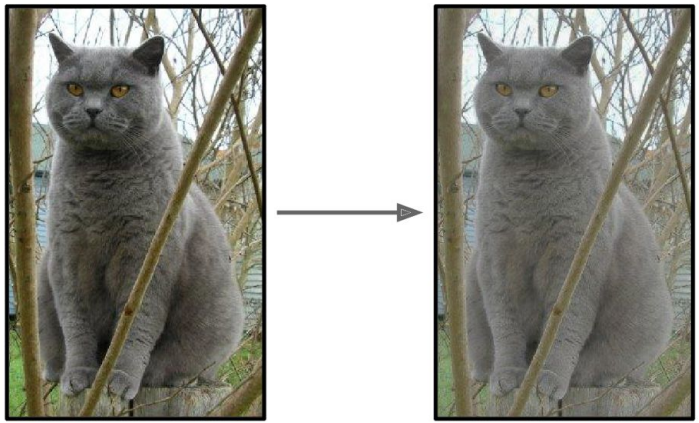
3. Color jitter

Simple:
Randomly jitter contrast



Data Augmentation
3. Color jitter

Simple:
Randomly jitter contrast



Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Data Augmentation

4. Get creative!

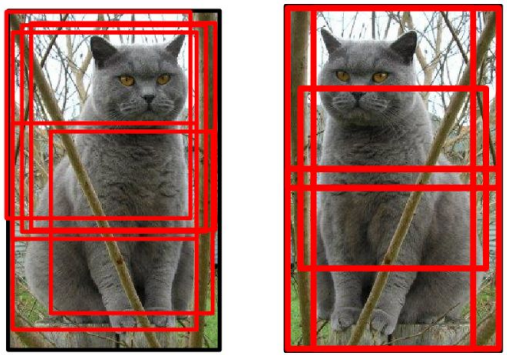
Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, …

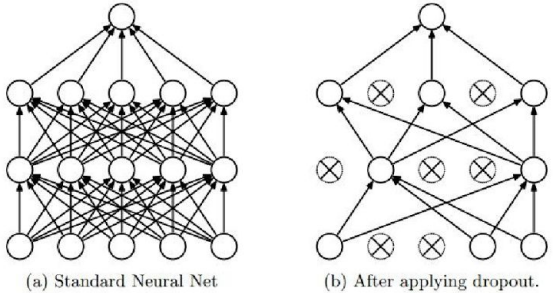
Slide taken from CS231n@stanford

A general theme:

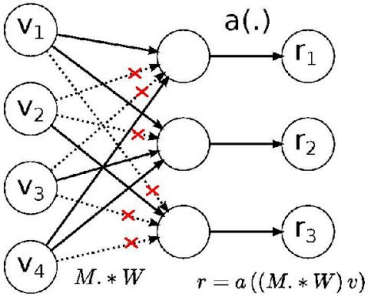
1. **Training**: Add random noise
2. **Testing**: Marginalize over the noise



Data Augmentation



Dropout



DropConnect

Batch normalization, Model ensembles

Slide taken from CS231n@stanford

Data Augmentation: Takeaway

- Simple to implement, use it
- Especially useful for small datasets
- Fits into framework of noise / marginalization

End of convnets material

Performance

Bottleneck

Depending on application, your bottleneck might be in different place:

- GPU
- CPU-GPU communication
- GPU-GPU communication (distributed learning)
- CPU (image preprocessing)
- I/O (non-sequential reads from SATA drive)

Your goal: make the GPU 100% busy.

How to store my images?

When the input to the neural network is an image, how the images should be stored?

- In one big array that fits into memory.
  Works only for small datasets

- Compressed format (JPEG).
  Uncompressing often a bottleneck, use multithreaded batch preparation.
  Worth considering packing samples into larger (few MBs) files (e.g., TFRecord in TensorFlow).

- Uncompressed format (BMP).
  SSD disk required.

Latency comparison

L1 cache reference	0.5 ns			
Branch mispredict	5 ns			
L2 cache reference	7 ns			14x L1 cache
Mutex lock/unlock	25 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	3 us		
Send 1K bytes over 1 Gbps network	10,000 ns	10 us		
Read 4K randomly from SSD	150,000 ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from SSD	1,000,000 ns	1,000 us	1 ms	~1GB/sec SSD, 4X memory
Disk seek	10,000,000 ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms	80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000 ns	150,000 us	150 ms	

https://gist.github.com/jboner/2841832

Reading data
(TensorFlow case study)

Reading data

There are three main methods of getting data into a TensorFlow program:

- Preloaded data: a constant or variable in the TensorFlow graph holds all the data (for small data sets).
  Note: most of the times you will not want to use that.

- Feeding: Python code provides the data when running each step.
  Note: slow when client and workers are on different machines.

- Reading from files: an input pipeline reads the data from files at the beginning of a TensorFlow graph.
  Note: uses asynchronous tensors computation by using queues. Tricky to debug.

From Tensorflow docs

Input Formats

From fastest to slowest

1. tf.Example and tf.SequenceExample protocol buffers in TFRecords files
2. Native TensorFlow ops to read CSV, JSON
3. Feed data directly from Python
    - Easiest to experiment with
    - Also useful for settings like reinforcement learning

Figure from TensorFlow Dev Summit

Distributed learning

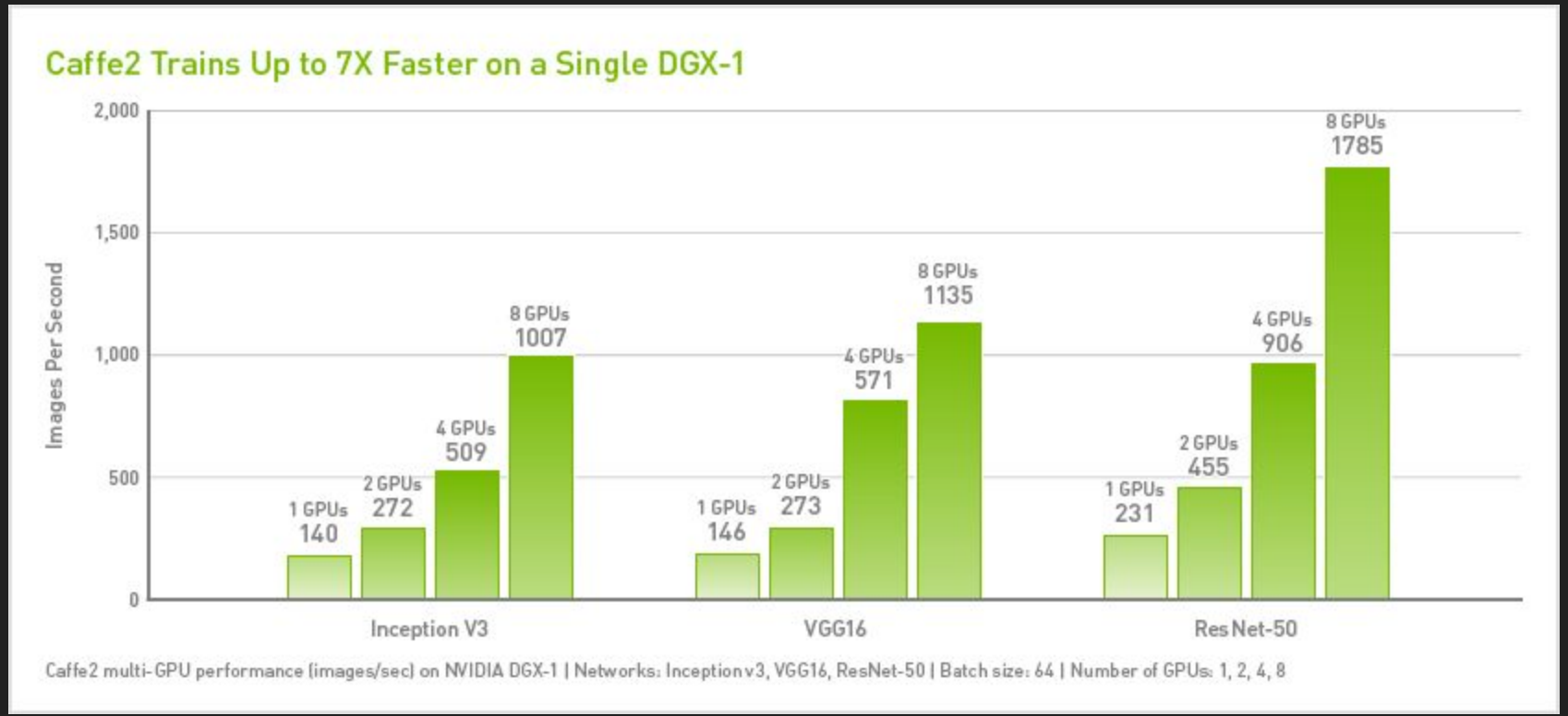


Figure from NVidia

Model parallelism

Each device calculates some part of the computational graph.

Allows to train outrageously large models (100 billion parameters) https://arxiv.org/abs/1701.06538

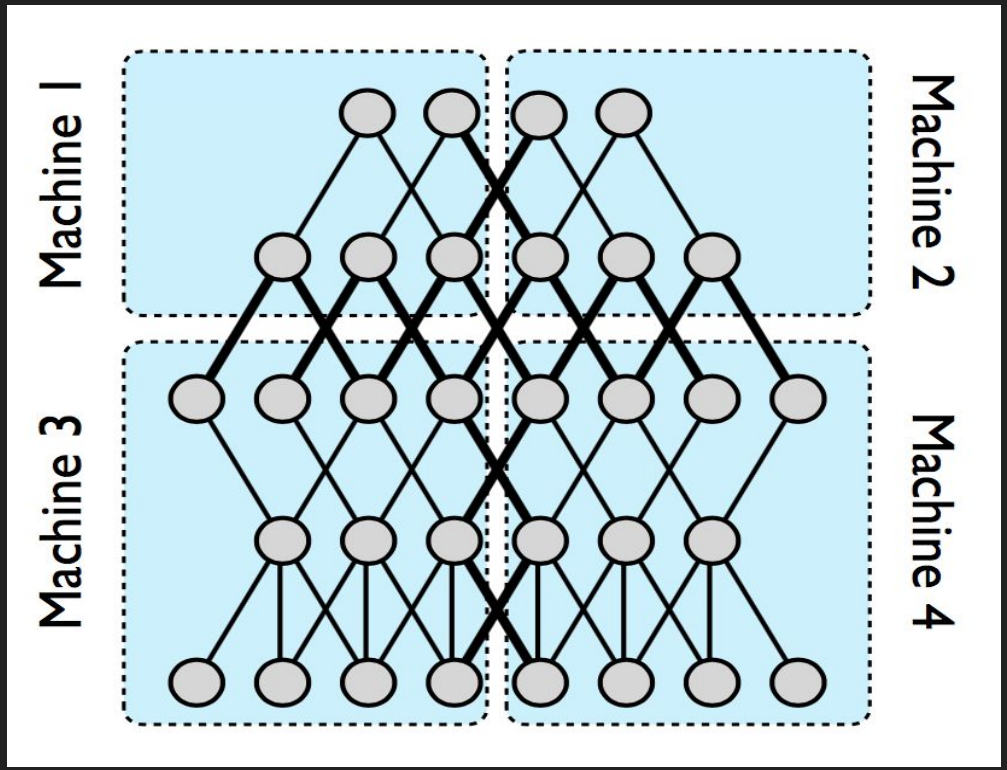


Figure from Google Brain slides

Model parallelism - CNNs

If we want to use model parallelism for convnets, is it a good idea to split:
Layer 1 to GPU:1
Layer 2 to GPU:2
Layer 3 to GPU:3
etc.?

Answer: NO! They would not be able to operate in parallel.

Model parallelism - CNNs

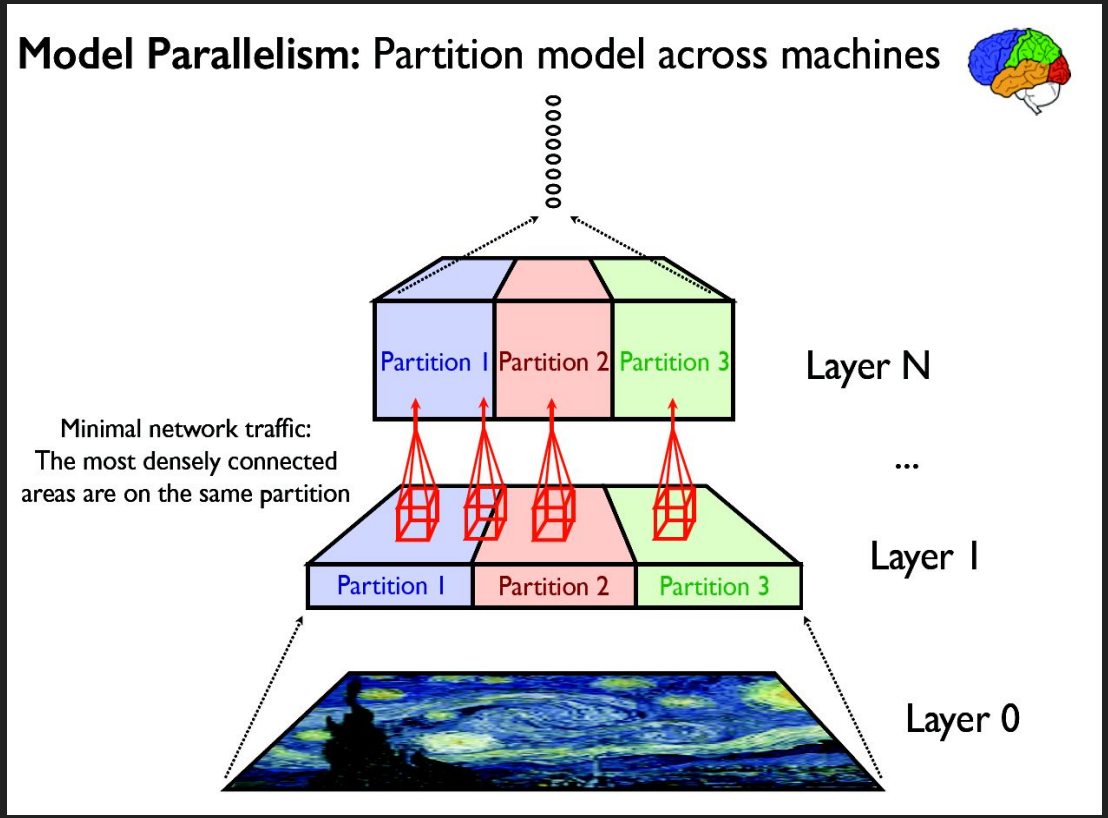


Figure from Google Brain slides

Model parallelism - LSTM

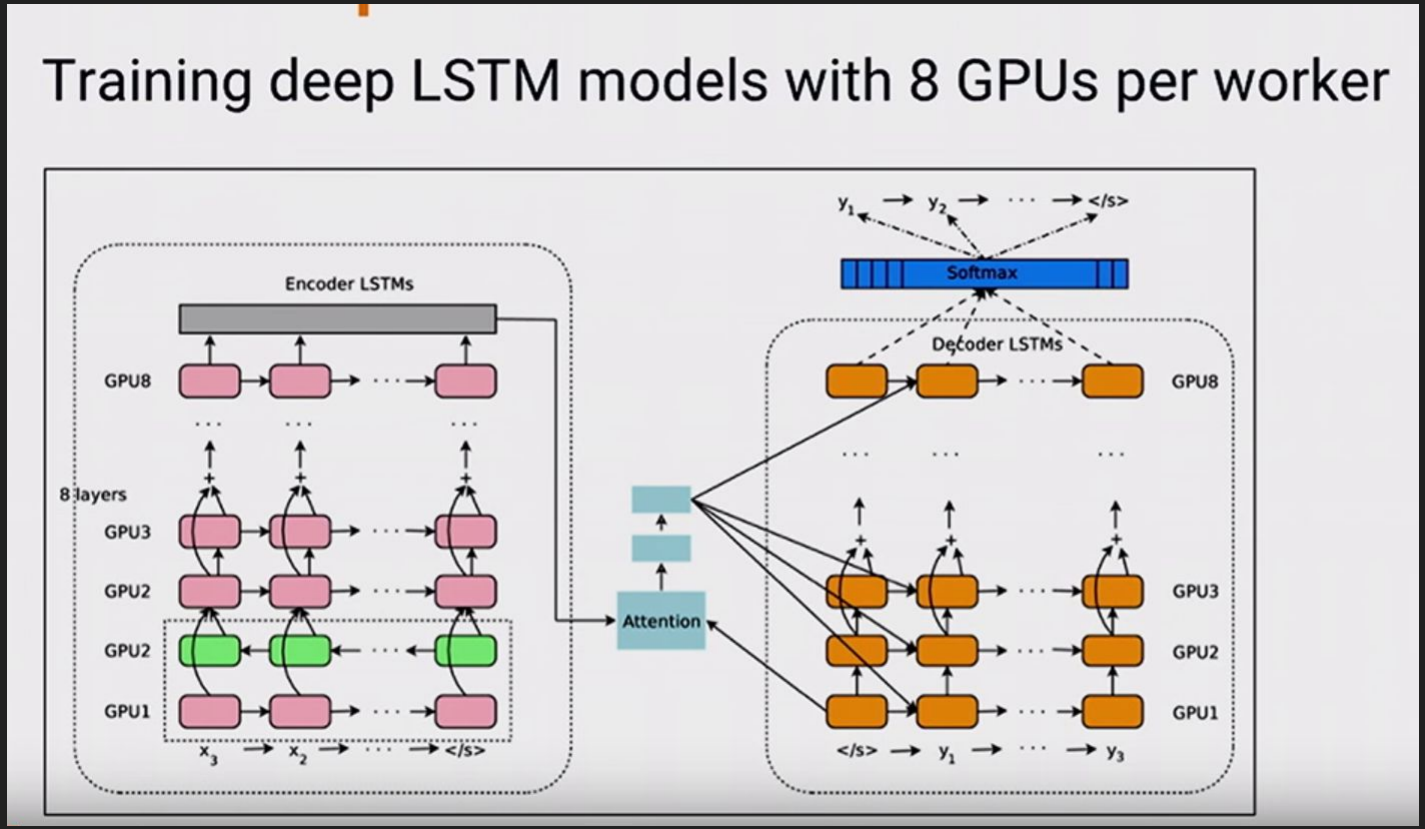


Figure from TensorFlow Dev Summit

Data parallelism

Split the batch into pieces, do forward and backward pass for each part of the batch and gather results

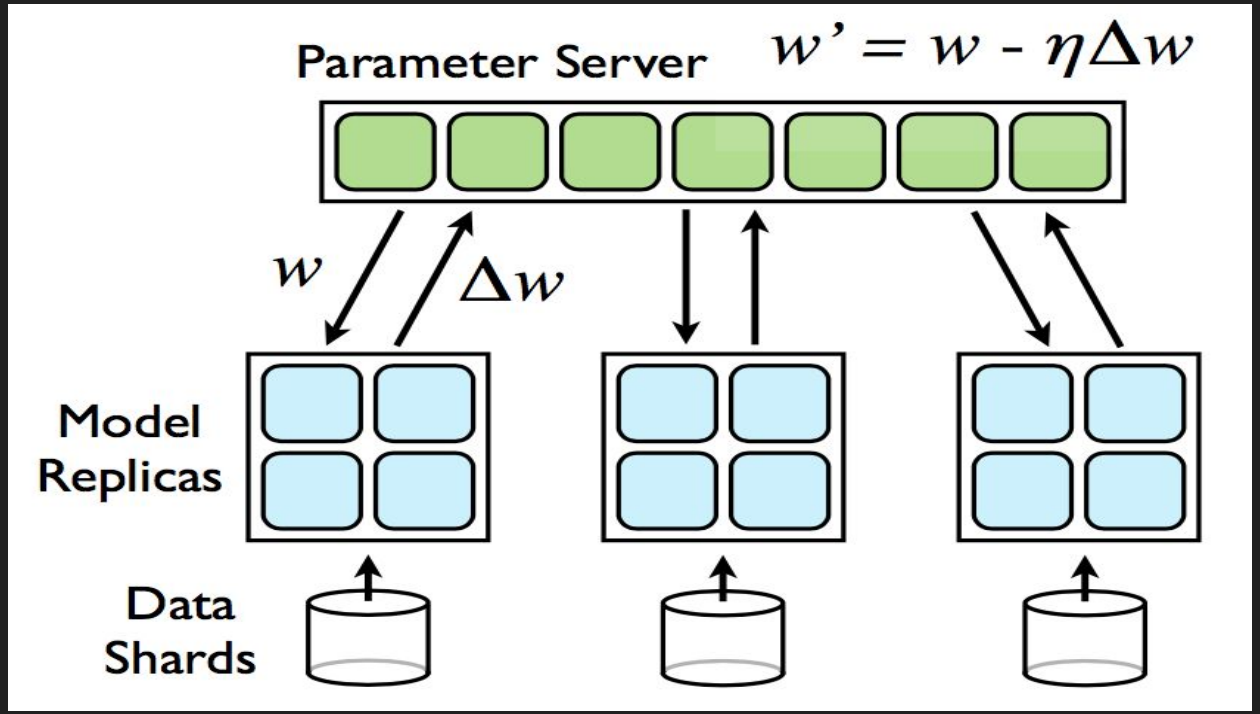
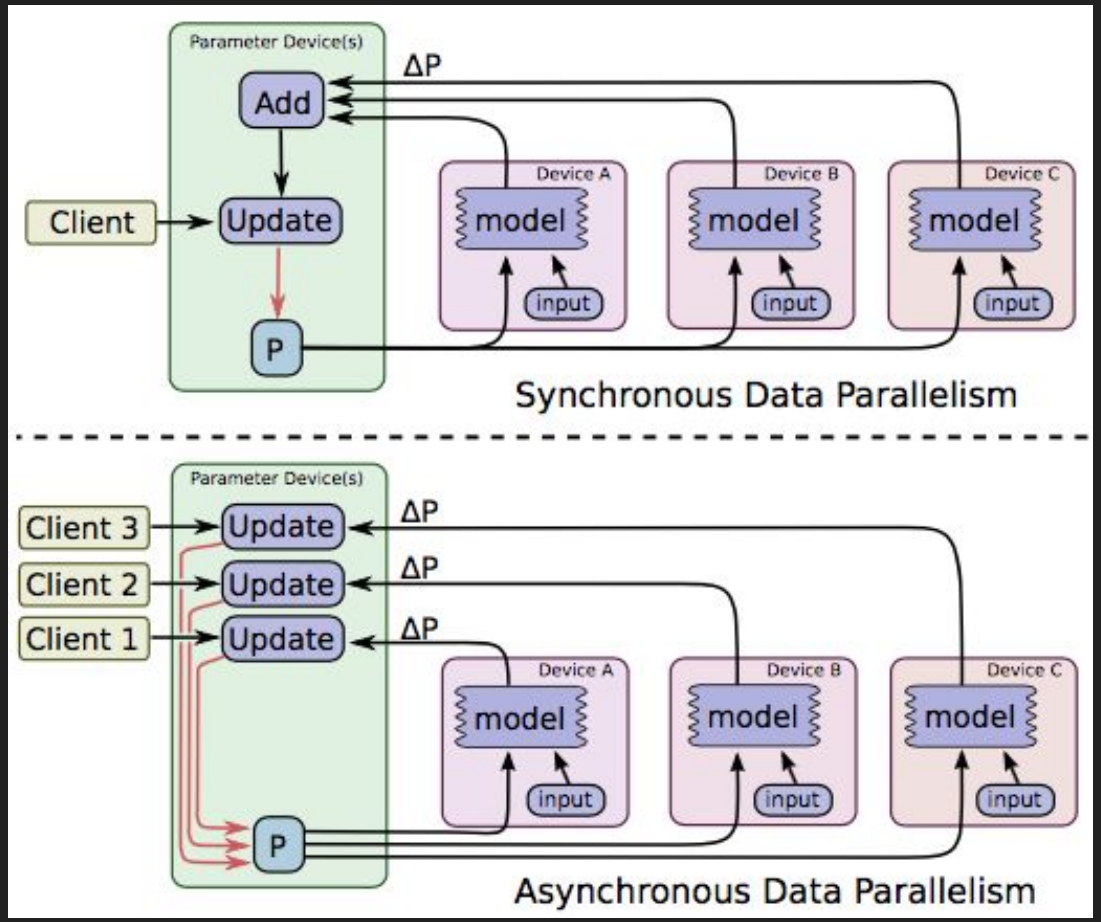


Figure from Google Brain slides

Data parallelism



- Synchronous - wait for all the computations and apply average gradient.

- Asynchronous - apply gradients as they arrive. Note: we are applying a gradient even though it has been computed for different parameters.

Figure from Google Brain slides

Distributed TensorFlow

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

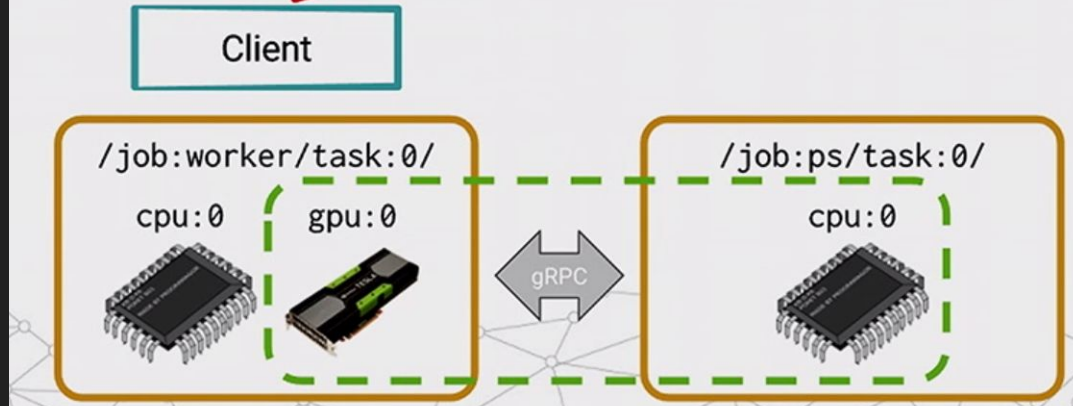


Figure from TensorFlow Dev Summit

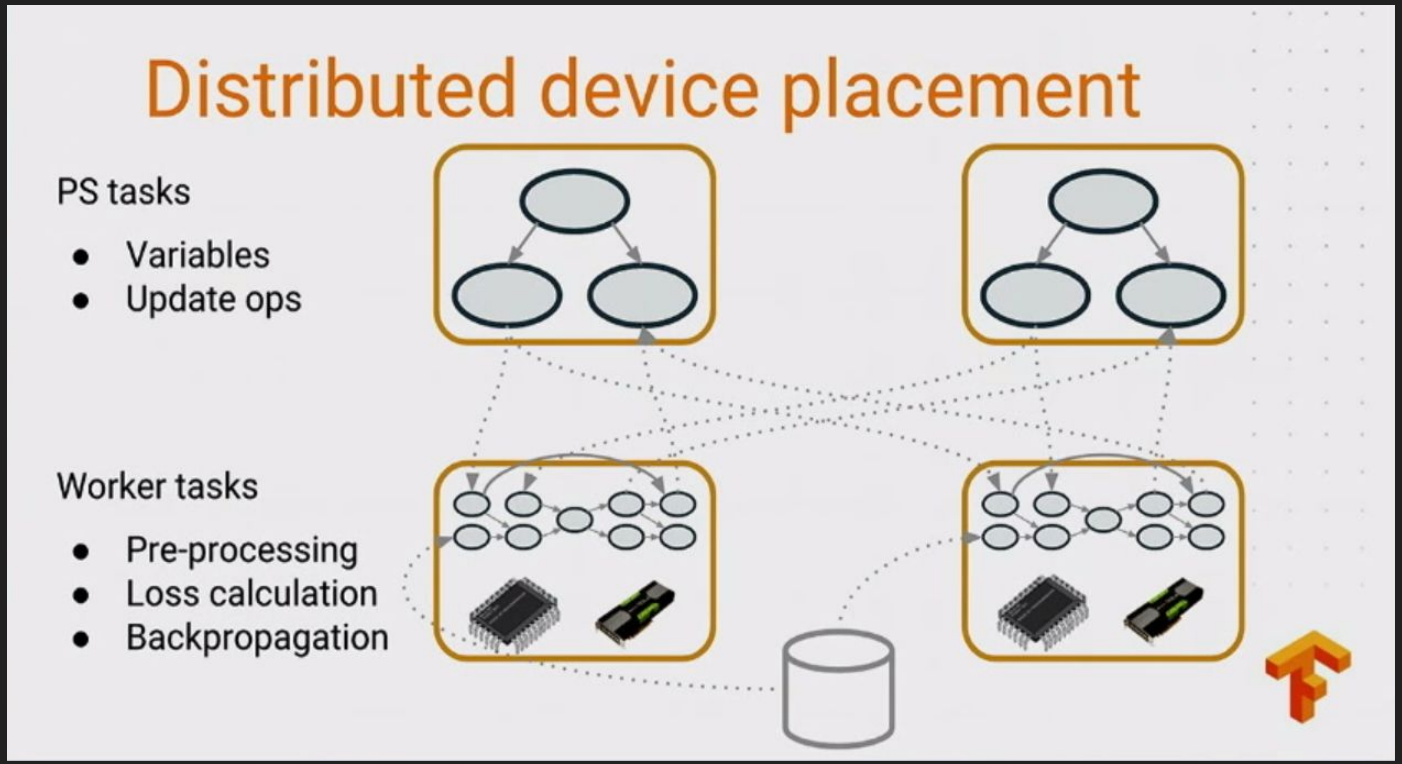


Figure from TensorFlow Dev Summit

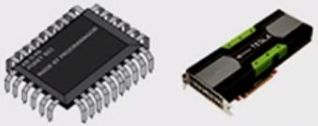
In-graph replication

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0, num_workers, input)
outputs = []
for i in range(num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" % i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(outputs)
```

Client

/job:worker/task:0/

cpu:0 gpu:0



/job:ps/task:0/

cpu:0

/job:worker/task:1/

gpu:0 cpu:0

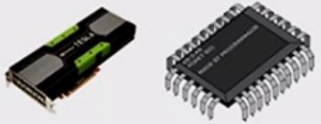


Figure from TensorFlow Dev Summit

Between-graph replication

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

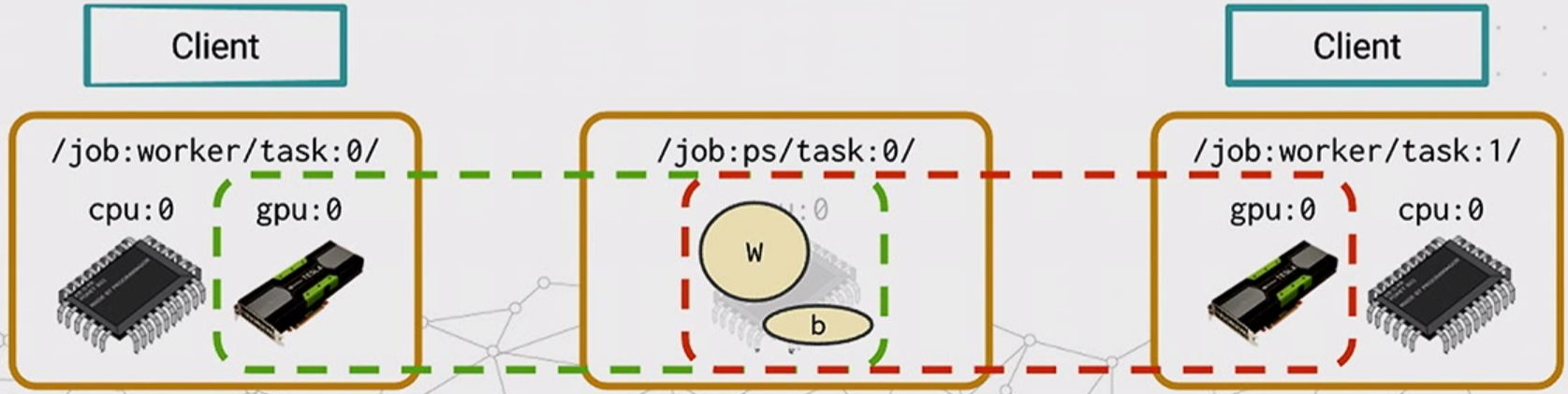


Figure from TensorFlow Dev Summit

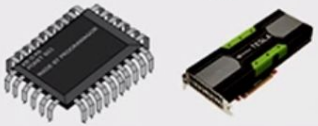
In-graph replication

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0, num_workers, input)
outputs = []
for i in range(num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" % i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(outputs)
```

Client

/job:worker/task:0/

cpu:0 gpu:0



/job:ps/task:0/

cpu:0

/job:worker/task:1/

gpu:0 cpu:0

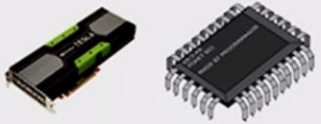


Figure from TensorFlow Dev Summit

Variable placement

```
with tf.device("/job:ps/task:0"):

  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

Figure from TensorFlow Dev Summit

Round-robin variables

```
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):

  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

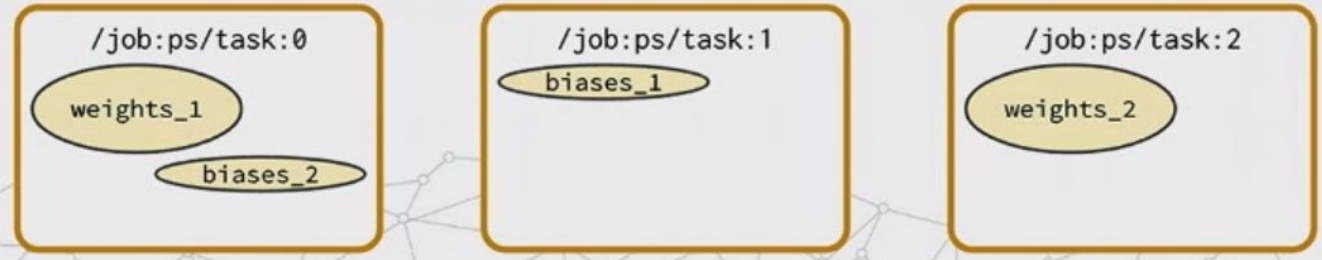


Figure from TensorFlow Dev Summit

Load balancing variables

```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(…)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):
  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

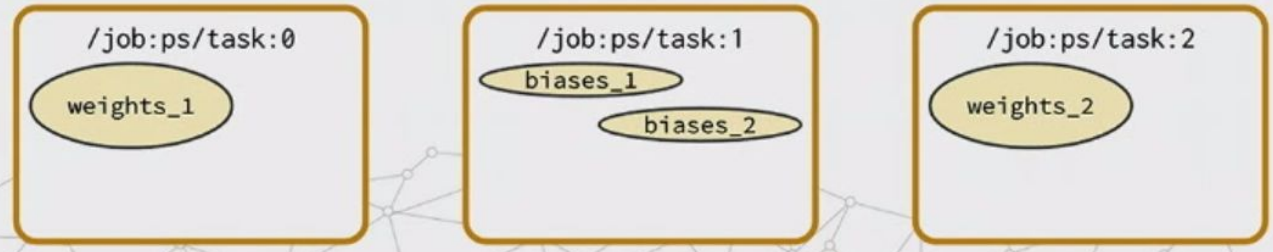


Figure from TensorFlow Dev Summit

Partitioned variables

```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(…)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):

  embedding = tf.get_variable(
      "embedding", [1000000000, 20],
      partitioner=tf.fixed_size_partitioner(3))
```

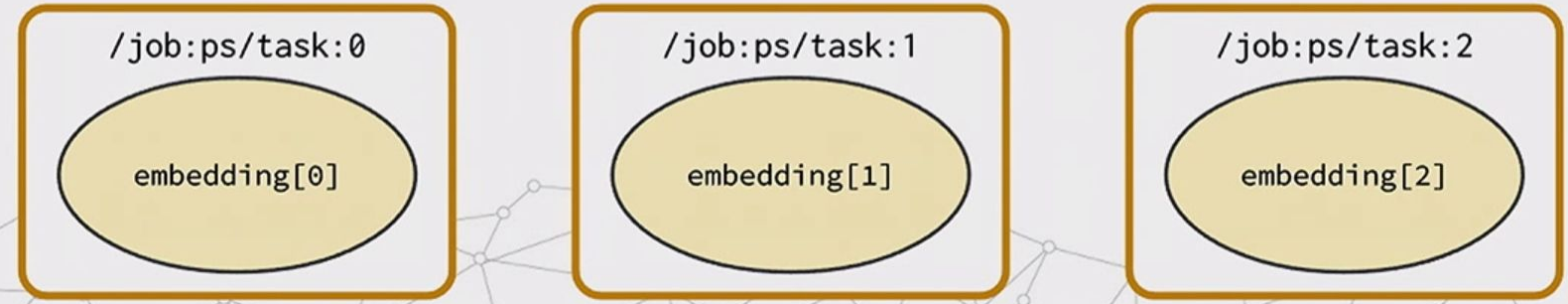


Figure from TensorFlow Dev Summit

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

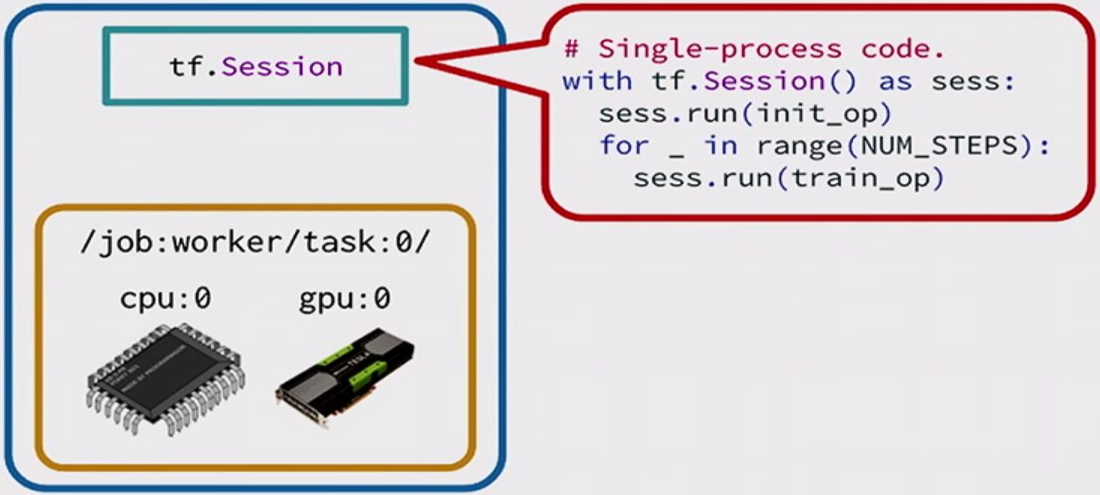


Figure from TensorFlow Dev Summit

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

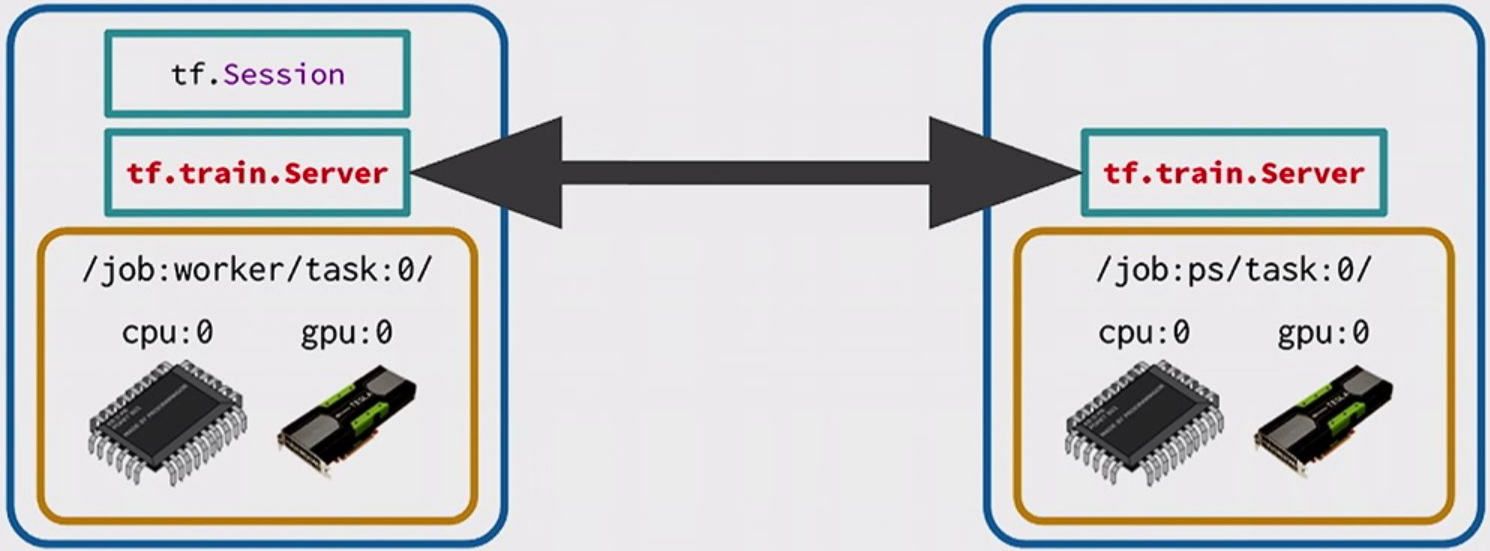


Figure from TensorFlow Dev Summit

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

```
# Distributed code for a worker task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="worker", task_index=0)

with tf.Session(server.target) as sess:
  # …
```

Figure from TensorFlow Dev Summit

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

```
# Distributed code for a worker task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="worker", task_index=0)

with tf.Session(server.target) as sess:
  # …
```

`cluster` defines the set of processes

`server` represents a particular task in `cluster`

`sess` can run code on any device in `cluster`

Figure from TensorFlow Dev Summit

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

```
# Distributed code for a PS task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="ps", task_index=0)

# Wait for incoming connections forever.
server.join()
```

Figure from TensorFlow Dev Summit

> A distributed system is a system where I can’t get my work done because a computer has failed that I’ve never even heard of.
>
> — Leslie Lamport

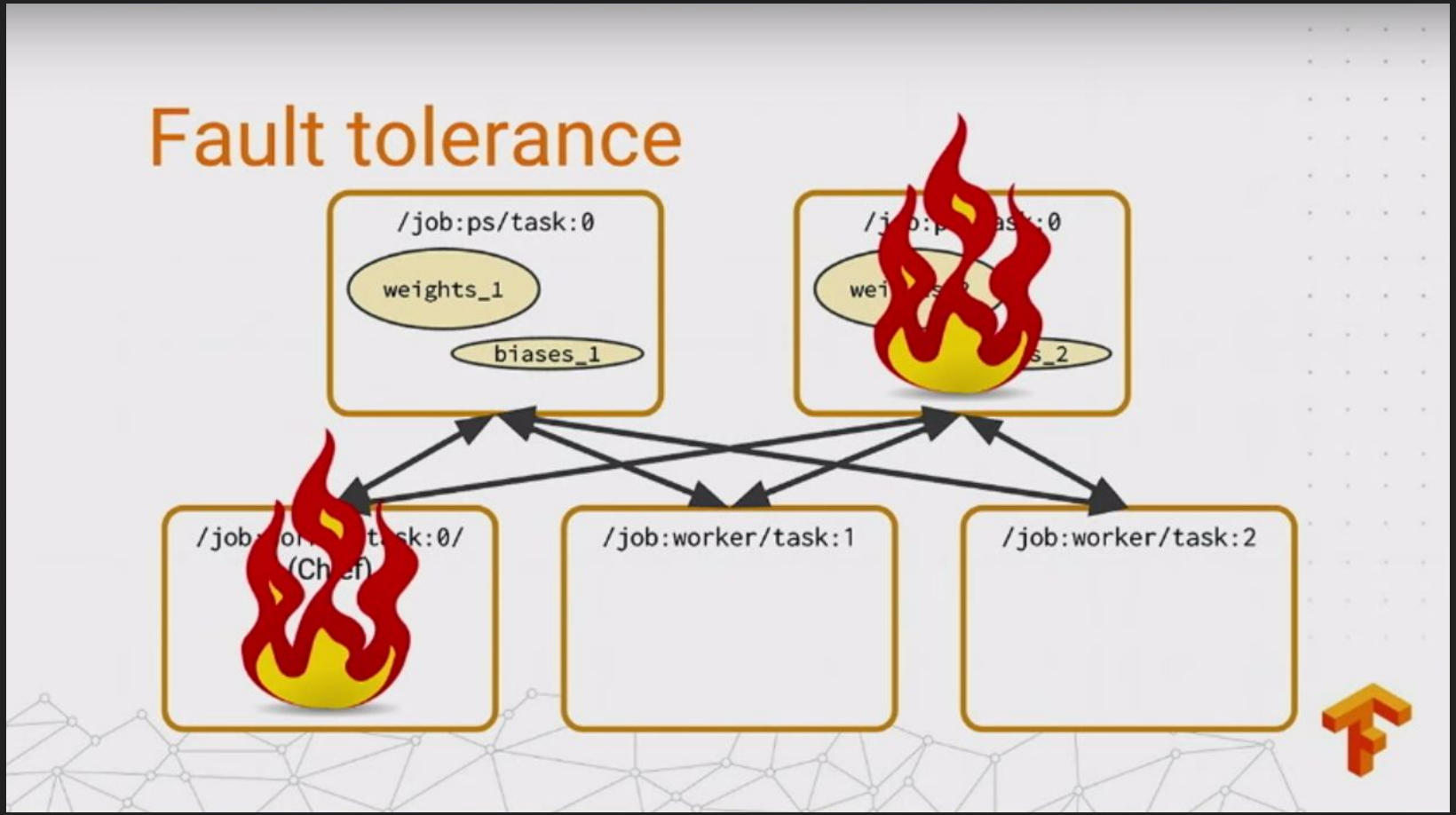


Figure from TensorFlow Dev Summit

Fault tolerance

`MonitoredTrainingSession` automates the recovery process

```
# Single-process code.
with tf.Session() as sess:
  sess.run(init_op)  # Or saver.restore(sess, …)
  for _ in range(NUM_STEPS):
    sess.run(train_op)
```

Figure from TensorFlow Dev Summit

Fault tolerance

MonitoredTrainingSession automates the recovery process

```
# Distributed code.
server = tf.train.Server(…)
is_chief = FLAGS.task_index == 0
with tf.train.MonitoredTrainingSession(server.target, is_chief) as sess:
  while not sess.should_stop():
    sess.run(train_op)
```

Automatically initializes and/or restores variables before returning

MonitoredSession.run() automatically recovers from PS failures, and can run additional code in *hooks*

Figure from TensorFlow Dev Summit

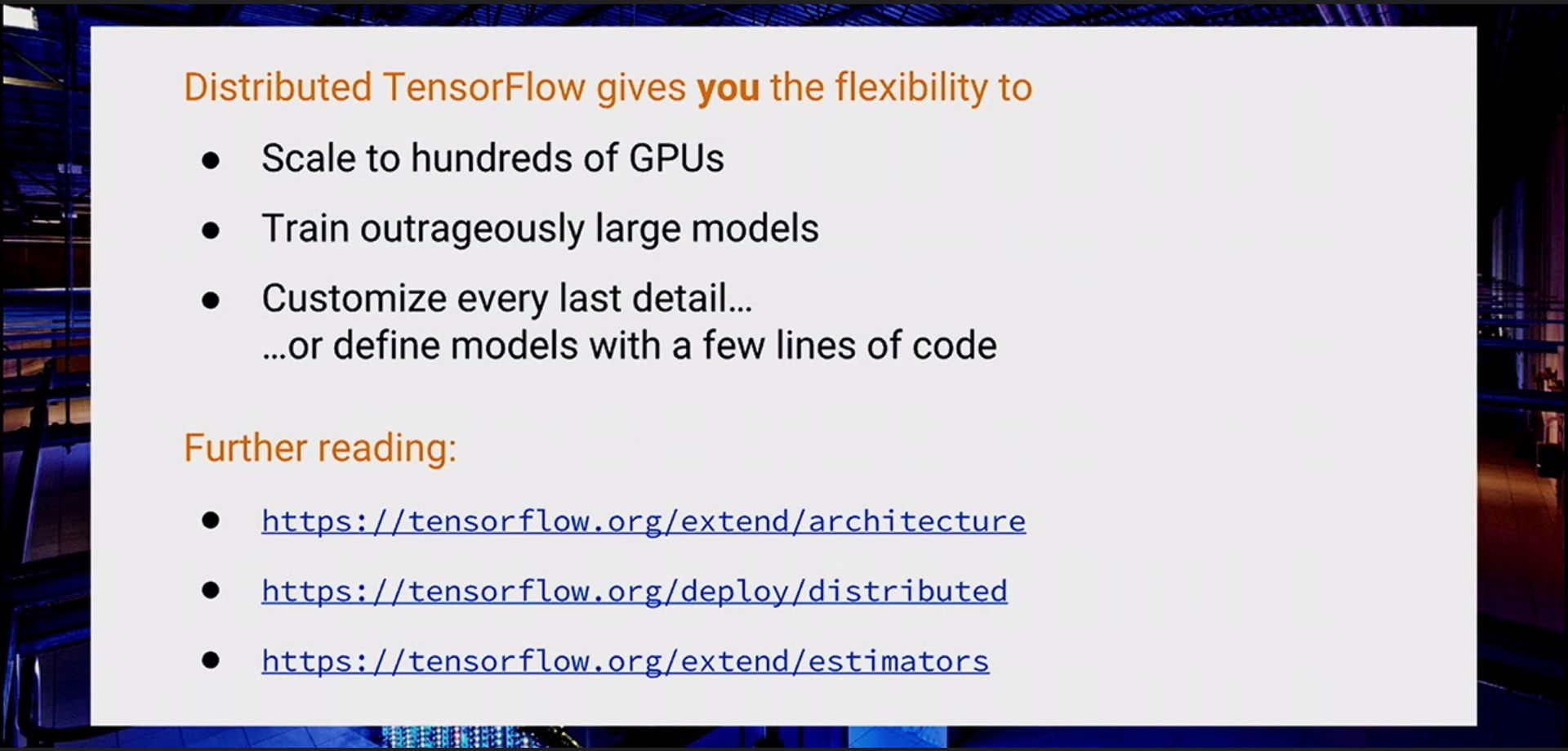


Figure from TensorFlow Dev Summit

Mind the transmission time for large modes.

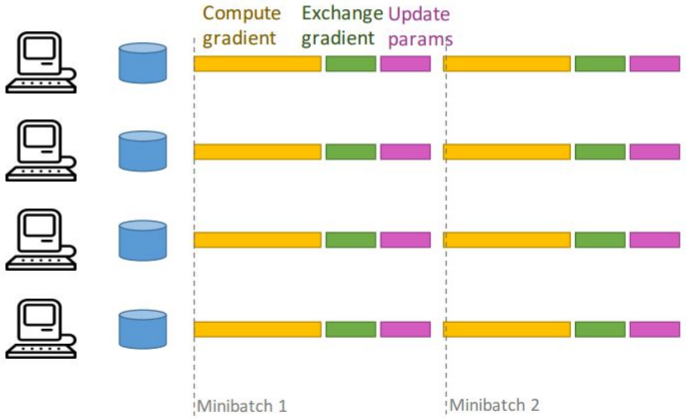
- GPUs have plenty of compute
- Yet, **bandwidth relatively limited**
- **PCIe** or (newer) **NVLINK**

Trend towards large models and datasets

- **Vision**: ImageNet (**1.8M images**)
- ResNet-152 model [He+15]:
  60M parameters (~240 MB)

- **Speech**: NIST2000 **2000 hours**
- LACEA [Yu+16]:
  65M parameters (~300 MB)

Gradient transmission is **expensive**.



The ZipML Framework for Training Models with End-to-End Low Precision:
The Cans, the Cannots, and a Little Bit of Deep Learning, https://arxiv.org/abs/1611.05402