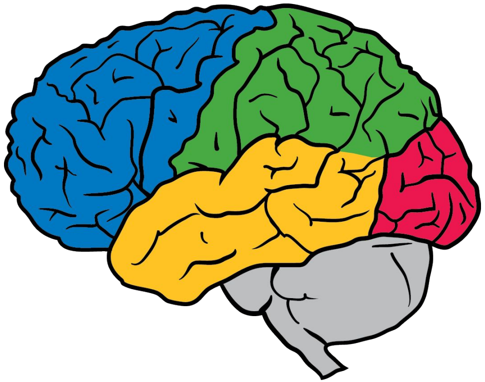


Large Scale Distributed Systems for Training Neural Networks

Jeff Dean & Oriol Vinyals
Google

Google Brain team in collaboration with many other teams

Background

Google Brain project started in 2011, with a focus on pushing state-of-the-art in neural networks. Initial emphasis:

- use large datasets, and
- large amounts of computation

to push boundaries of what is possible in perception and language understanding

Overview

- Cover our experience from past ~5 years
    - **Research**: speech, images, video, robotics, language understanding, NLP, translation, optimization algorithms, unsupervised learning, ...
    - **Production**: deployed systems for advertising, search, GMail, Photos, Maps, YouTube, speech recognition, image analysis, user prediction, ...
- Focus on neural nets, but many techniques more broadly applicable

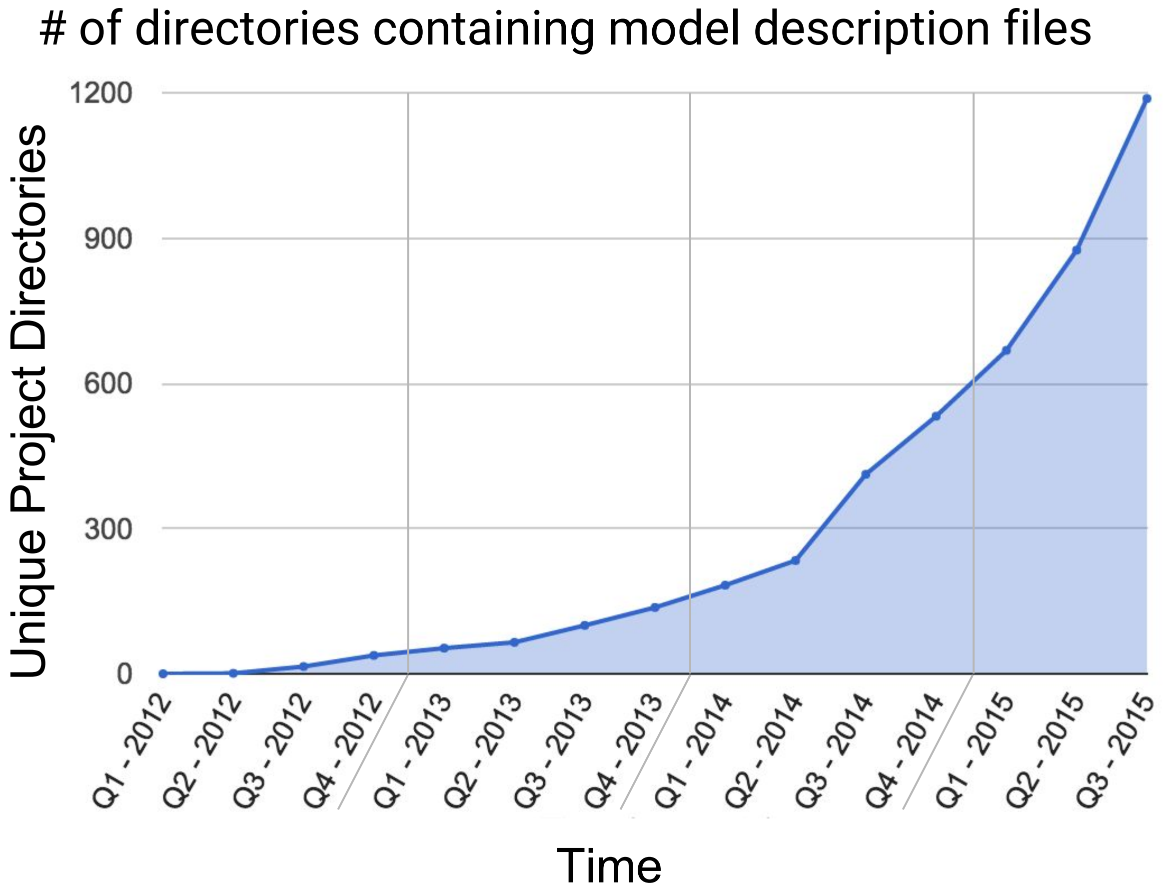
Overview

- Demonstrate *TensorFlow*, an open source machine learning system
  - Our primary research and production system
  - Show real examples
  - Explain what's happening underneath the covers

Outline

- Introduction to Deep Learning
- TensorFlow Basics
  - Demo
  - Implementation Overview
- Scaling Up
  - Model Parallelism
  - Data Parallelism
  - Expressing these in TensorFlow
- More complex examples
  - CNNs / Deep LSTMs

Growing Use of Deep Learning at Google



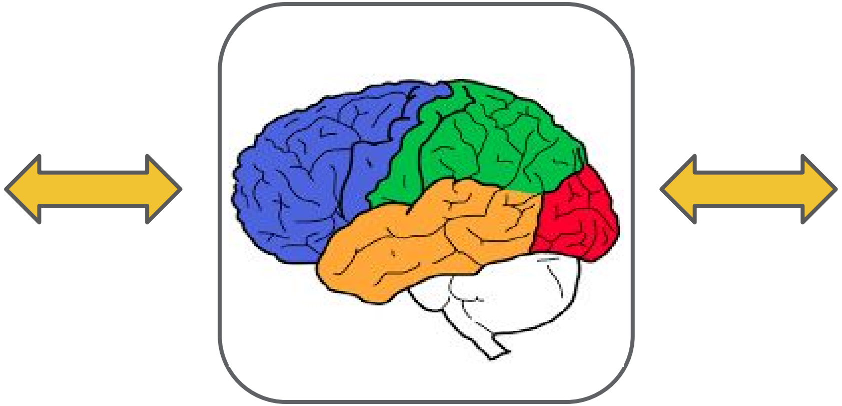
Across many products/areas:

Android
Apps
drug discovery
Gmail
Image understanding
Maps
Natural language understanding
Photos
Robotics research
Speech
Translation
YouTube
… many others ...

Deep Learning

Universal Machine Learning

Speech
Text
Search
Queries
Images
Videos
Labels
Entities
Words
Audio
Features



Speech
Text
Search
Queries
Images
Videos
Labels
Entities
Words
Audio
Features

Deep Learning

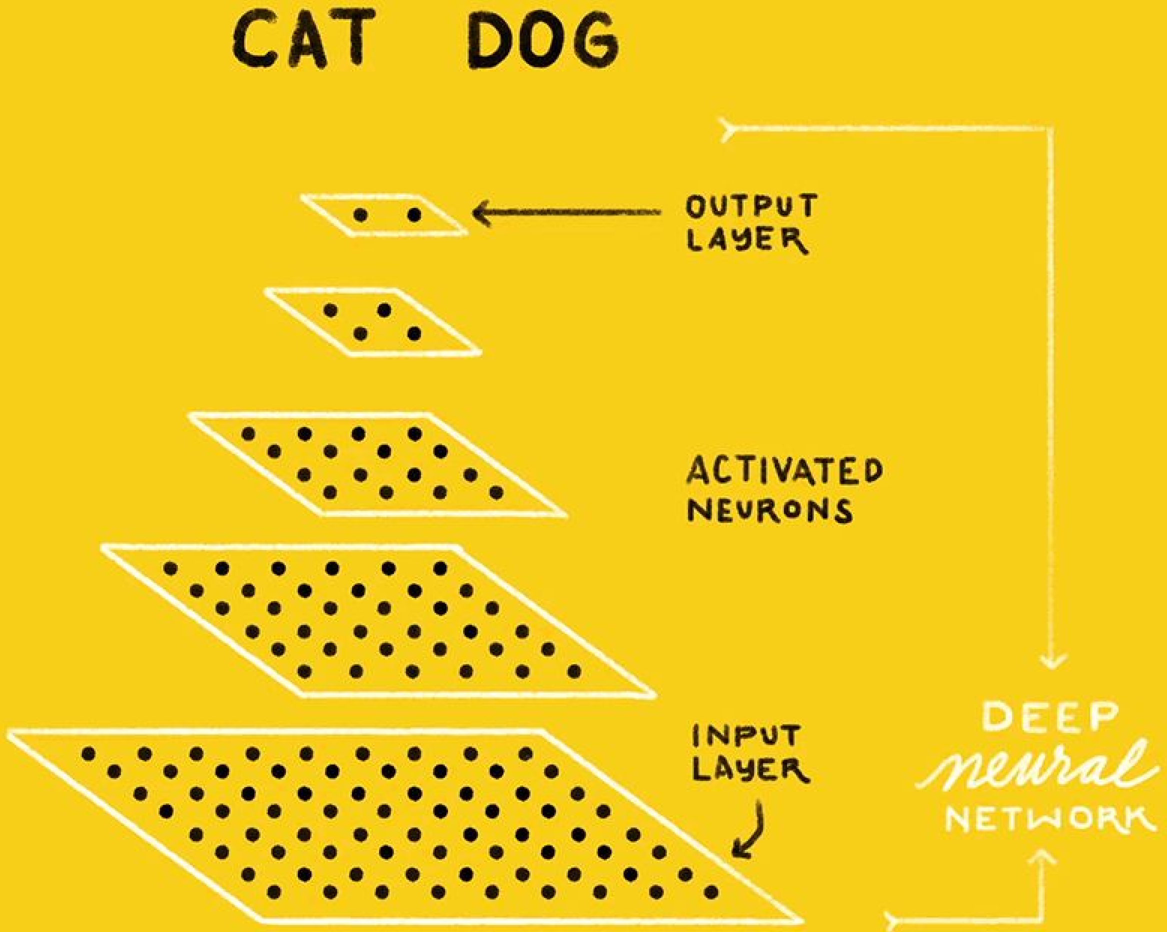
Universal Machine Learning

...that works better than the alternatives!

Current State-of-the-art in:

Speech Recognition
Image Recognition
Machine Translation
Molecular Activity Prediction
Road Hazard Detection
Optical Character Recognition
...

IS THIS A CAT or DOG?



Some More Benefits

Deals very naturally w/sequence data (text, speech, video...)

Very effective at transfer learning across tasks

Very easy to get started with a commodity GPU

A common ‘language’ across great many fields of research

Two Generations of Distributed ML Systems

1st generation - DistBelief (Dean *et al.*, NIPS 2012)

- Scalable, good for production, but not very flexible for research

2nd generation - TensorFlow (see tenorflow.org and whitepaper 2015, tensorflow.org/whitepaper2015.pdf)

- Scalable, good for production, but also flexible for variety of research uses
- Portable across range of platforms
- Open source w/ Apache 2.0 license

Need Both Large Datasets & Large, Powerful Models

"Scaling Recurrent Neural Network Language Models", Williams et al. 2015

arxiv.org/pdf/1502.00512v1.pdf

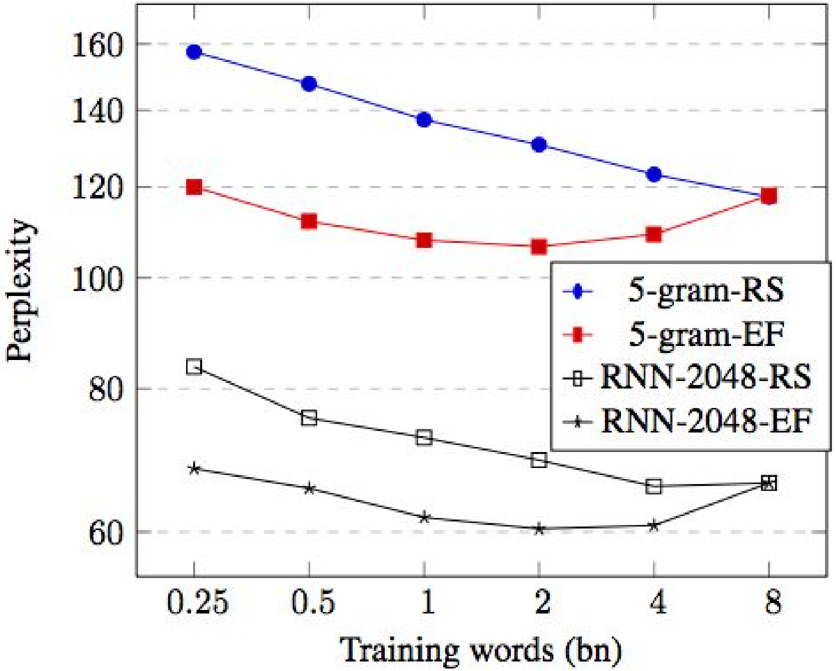


Fig. 2. Performance of 5-grams against *nstate* 2048 RNNs with increasing training data size. We test on Randomly Selected (RS) splits and Entropy Filtered (EF) splits of the 8bn corpus.

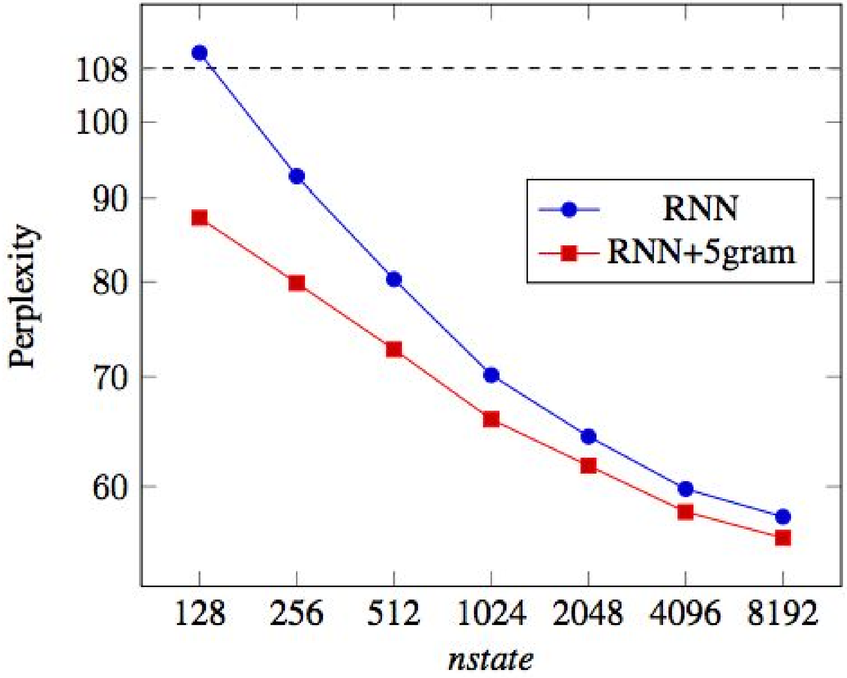


Fig. 3. Scaling *nstate* trained on 1bn words of the entropy filtered 8bn corpus. Dashed line is the 5-gram baseline.

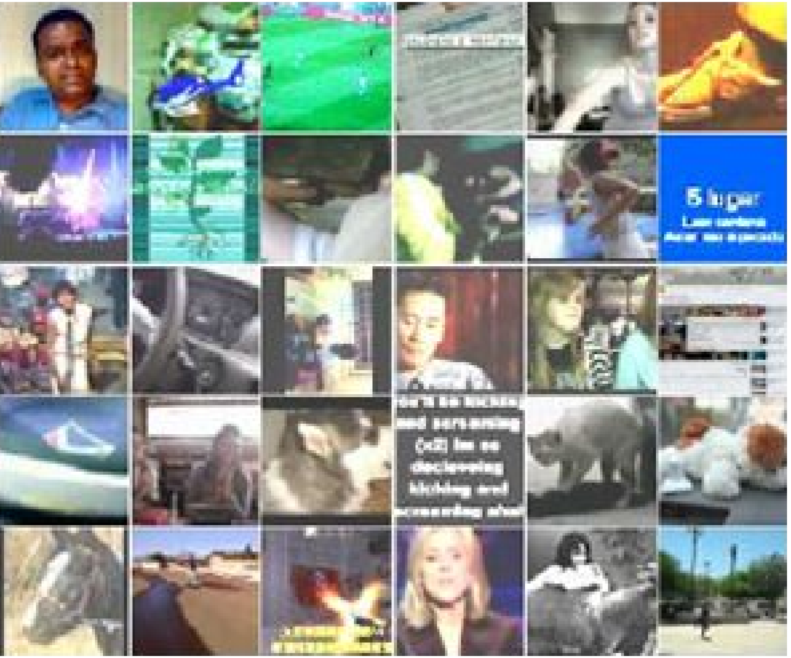
Large Datasets + Powerful Models

- Combination works incredibly well
- Poses interesting systems problems, though:
  - Need lots of computation
  - Want to train and do experiments quickly
  - Large-scale parallelism using distributed systems really only way to do this at very large scale
  - Also want to easily express machine learning ideas

Basics of Deep Learning

- Unsupervised cat
- Speech
- Vision
- General trend is towards more complex models:
  - Embeddings of various kinds
  - Generative models
  - Layered LSTMs
  - Attention

Learning from Unlabeled Images



- Train on 10 million images (YouTube)
- 1000 machines (16,000 cores) for 1 week.
- 1.15 billion parameters

Learning from Unlabeled Images

Top 48 stimuli from the test set

Optimal stimulus
by numerical optimization

Learning from Unlabeled Images

Top 48 stimuli from the test set

Optimal stimulus
by numerical optimization

Adding Supervision

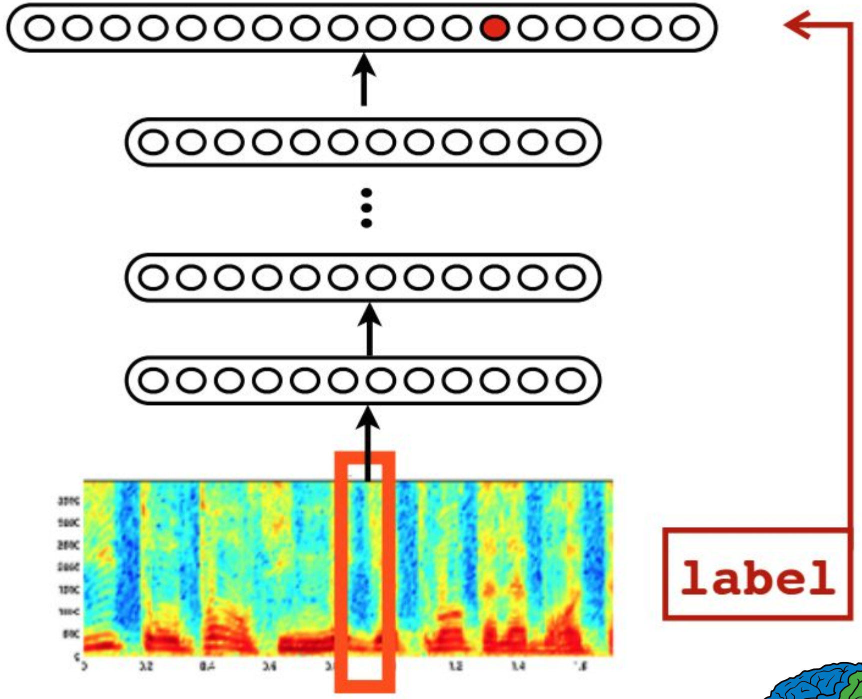
Top stimuli for selected neurons.

Speech: Feedforward Acoustic Models

Model speech frame-by-frame, independently

Simple fully-connected networks

Deep Neural Networks for Acoustic Modeling in Speech Recognition
Hinton *et al.* IEEE Signal Processing Magazine, 2012

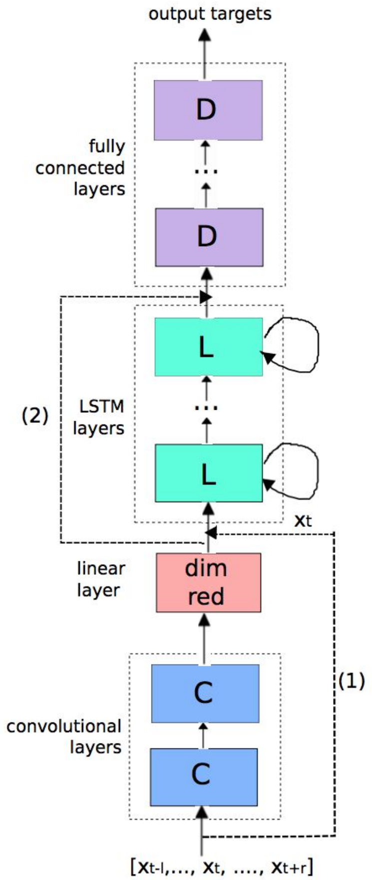


CLDNNs

Model frequency invariance using 1D convolutions

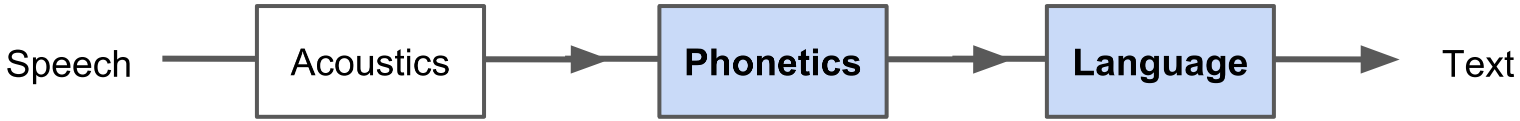
Model time dynamics using an LSTM

Use fully connected layers on top to add depth



Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks
Sainath *et al.* ICASSP'15

Trend: LSTMs end-to-end!



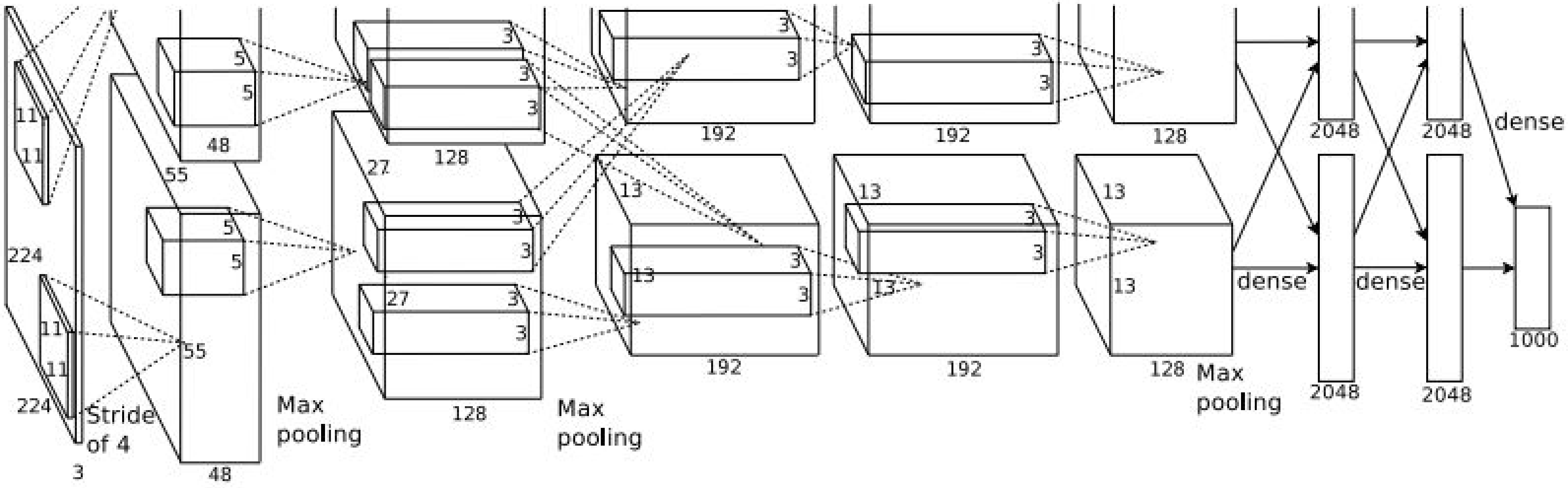
Train recurrent models that also incorporate **Lexical** and **Language Modeling:**

Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition, H. Sak *et al.* 2015

Deep Speech: Scaling up end-to-end speech recognition, A. Hannun *et al.* 2014

Listen, Attend and Spell, W. Chan *et al.* 2015

CNNs for Vision: AlexNet

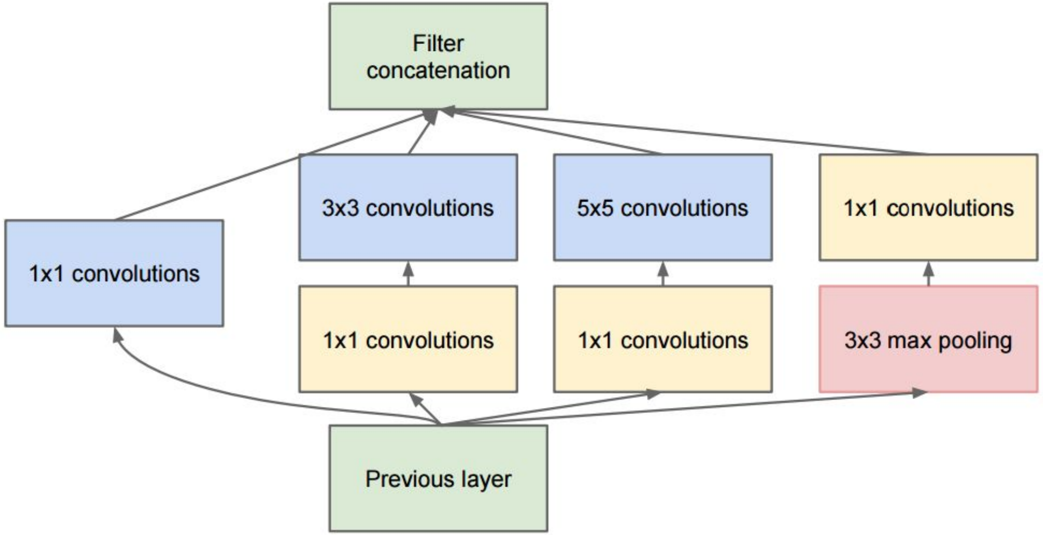


ImageNet Classification with Deep Convolutional Neural Networks

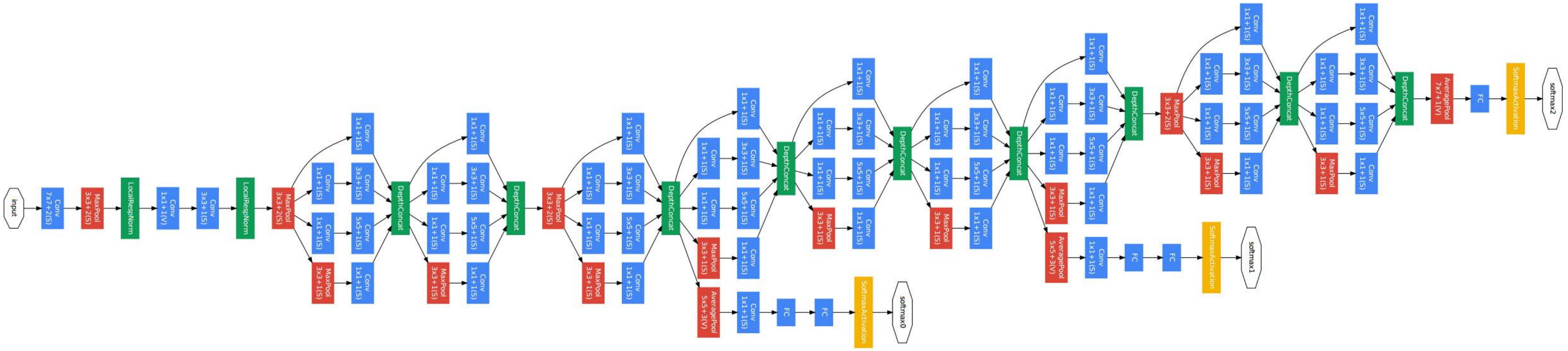
Krizhevsky, Sutskever and Hinton, NIPS 2012

The Inception Architecture (GoogLeNet, 2015)

Basic module, which is then replicated many times



The Inception Architecture (GoogLeNet, 2015)



Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

ArXiv 2014, CVPR 2015

Inception-v3 (December 2015)

Rethinking the Inception Architecture for Computer Vision

Christian Szegedy
Google Inc.
szegedy@google.com

Vincent Vanhoucke
vanhoucke@google.com

Sergey Ioffe
sioffe@google.com

Jonathon Shlens
shlens@google.com

Zbigniew Wojna
University College London
zbigniewwojna@gmail.com

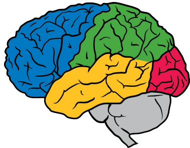
http://arxiv.org/abs/1512.00567

Rapid Progress in Image Recognition

Team	Year	Place	Error (top-5)	Params
XRCE (pre-neural-net explosion)	2011	1st	25.8%	
Supervision (AlexNet)	2012	1st	16.4%	60M
Clarifai	2013	1st	11.7%	65M
MSRA	2014	3rd	7.35%	
VGG	2014	2nd	7.32%	180M
GoogLeNet (Inception)	2014	1st	6.66%	5M
Andrej Karpathy (human)	2014	N/A	5.1%	100 trillion?
BN-Inception (Arxiv)	2015	N/A	4.9%	13M
Inception-v3 (Arxiv)	2015	N/A	3.46%	25M

ImageNet challenge classification task

Models with small number of parameters fit easily in a mobile app (8-bit fixed point)



Today’s News: Pre-trained Inception-v3 model released

http://googleresearch.blogspot.com/2015/12/how-to-classify-images-with-tensorflow.html

Dear TensorFlow community,

Today we are releasing our best image classifier trained on ImageNet data. As described in our recent Arxiv preprint at http://arxiv.org/abs/1512.00567, an ensemble of four of these models achieves 3.5% top-5 error on the validation set of the ImageNet whole image ILSVRC2012 classification task (compared with our ensemble from last year that won the 2014 ImageNet classification challenge with a 6.66% top-5 error rate).

In this release, we are supplying code and data files containing the trained model parameters for running the image classifier on:

- Both desktop and mobile environments
- Employing either a C++ or Python API.

In addition, we are providing a tutorial that describes how to use the image recognition system for a variety of use-cases.

http://www.tensorflow.org/tutorials/image_recognition/index.html

```
bazel build tensorflow/examples/label_image/...
```

That should create a binary executable that you can then run like this:

```
bazel-bin/tensorflow/examples/label_image/label_image
```

This uses the default example image that ships with the framework, and should output something similar to this:

```
I tensorflow/examples/label_image/main.cc:200] military uniform (866): 0.647296
I tensorflow/examples/label_image/main.cc:200] suit (794): 0.0477196
I tensorflow/examples/label_image/main.cc:200] academic gown (896): 0.0232411
I tensorflow/examples/label_image/main.cc:200] bow tie (817): 0.0157356
I tensorflow/examples/label_image/main.cc:200] bolo tie (940): 0.0145024
```

In this case, we're using the default image of Admiral Grace Hopper, and you can see the network correctly identifies she's wearing a military uniform, with a high score of 0.6.

What do you want in a research system?

- **Ease of expression**: for lots of crazy ML ideas/algorithms
- **Scalability**: can run experiments quickly
- **Portability**: can run on wide variety of platforms
- **Reproducibility**: easy to share and reproduce research
- **Production readiness**: go from research to real products

TensorFlow:
Second Generation Deep Learning System

TensorFlow

TensorFlow

If we like it, wouldn't the rest of the world like it, too?

Open sourced single-machine TensorFlow on Monday, Nov. 9th

- Flexible Apache 2.0 open source licensing
- Updates for distributed implementation coming soon

http://tensorflow.org/

Motivations

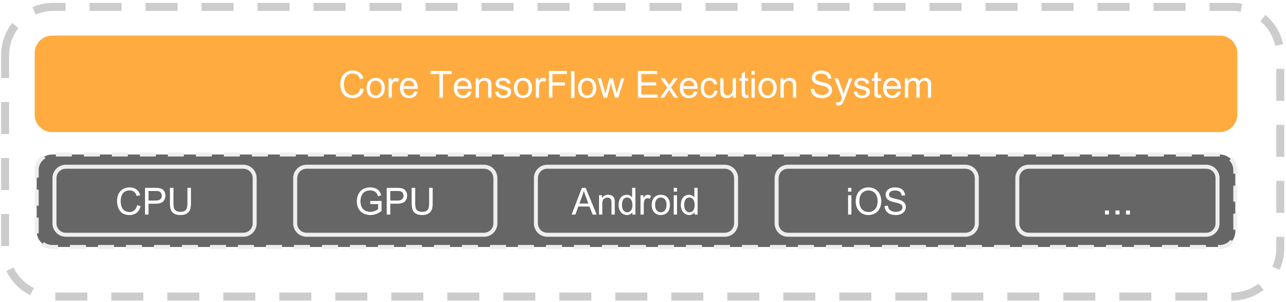
DistBelief (1st system):

- Great for scalability, and production training of basic kinds of models
- Not as flexible as we wanted for research purposes

Better understanding of problem space allowed us to make some dramatic simplifications

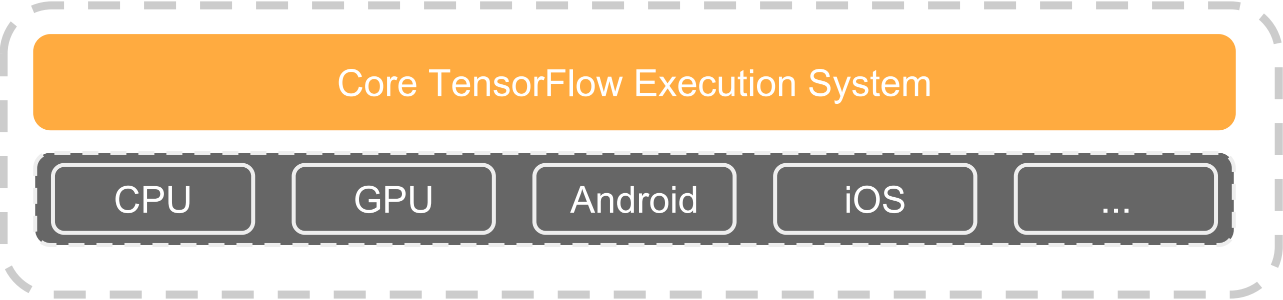
TensorFlow: Expressing High-Level ML Computations

- Core in C++



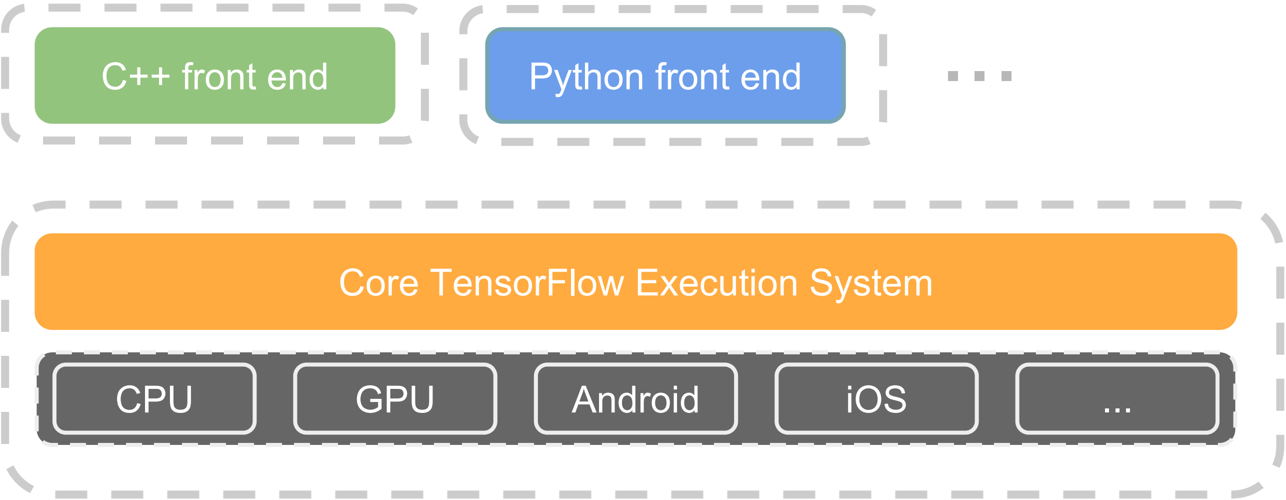
TensorFlow: Expressing High-Level ML Computations

- Core in C++
- Different front ends for specifying/driving the computation
    - Python and C++ today, easy to add more



TensorFlow: Expressing High-Level ML Computations

- Core in C++
- Different front ends for specifying/driving the computation
    - Python and C++ today, easy to add more



Portable

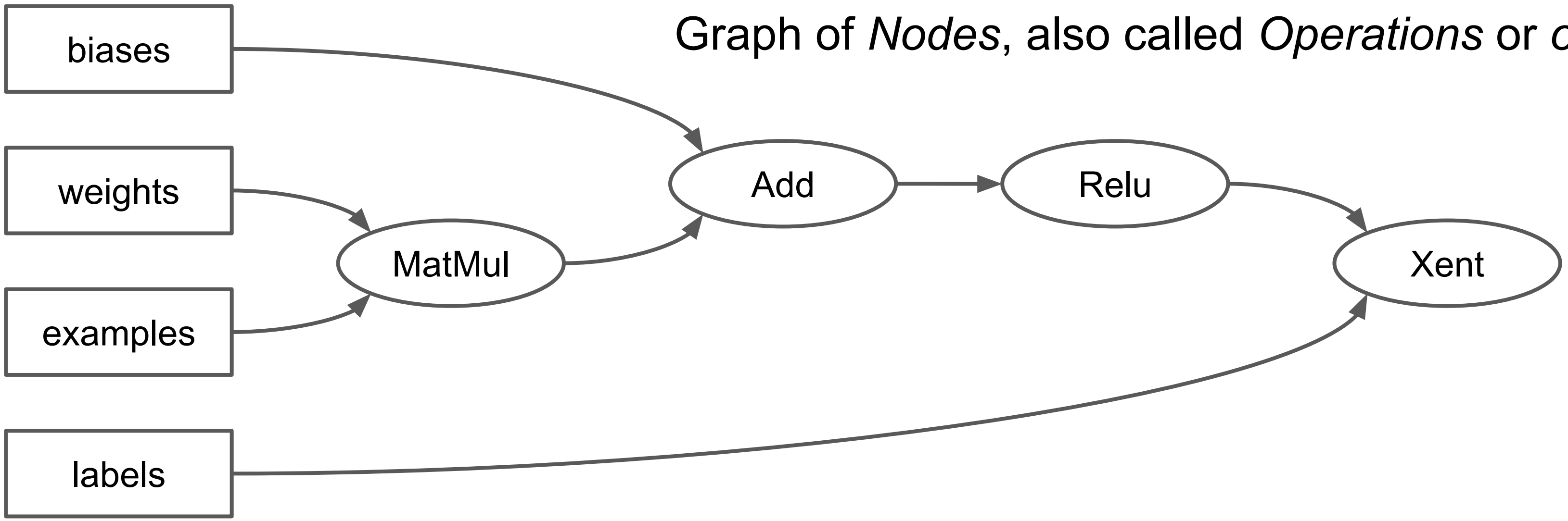
Automatically runs models on range of platforms:

from **phones** ...

to **single machines** (CPU and/or GPUs) …

to **distributed systems** of many 100s of GPU cards

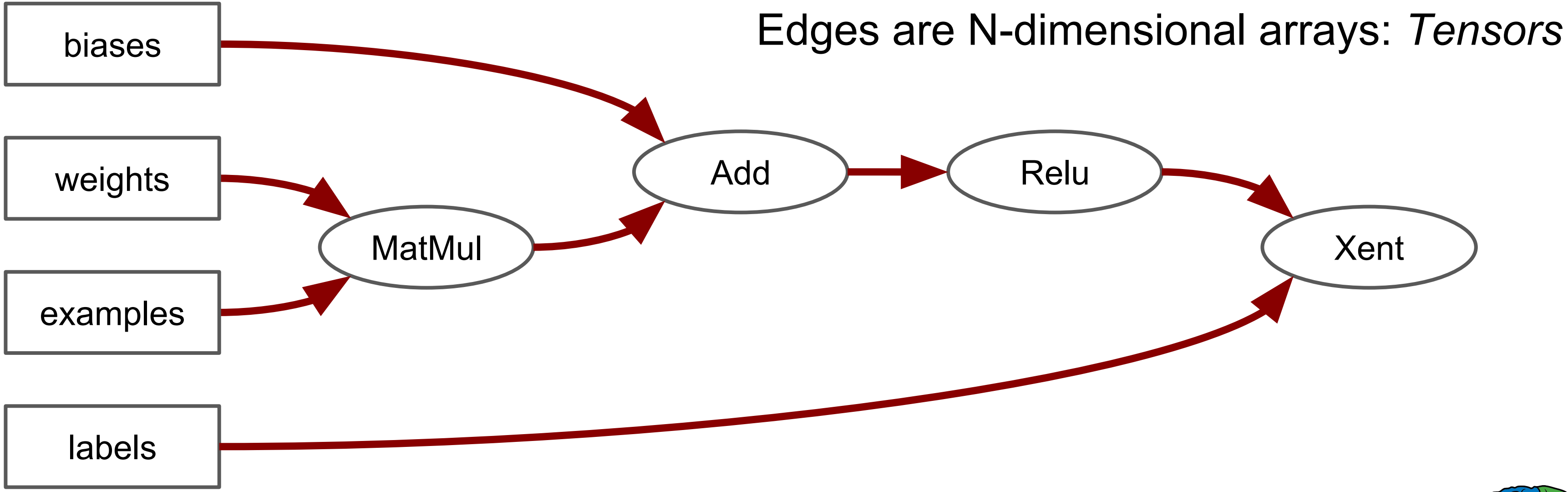
Computation is a dataflow graph



Graph of *Nodes*, also called *Operations* or *ops.*

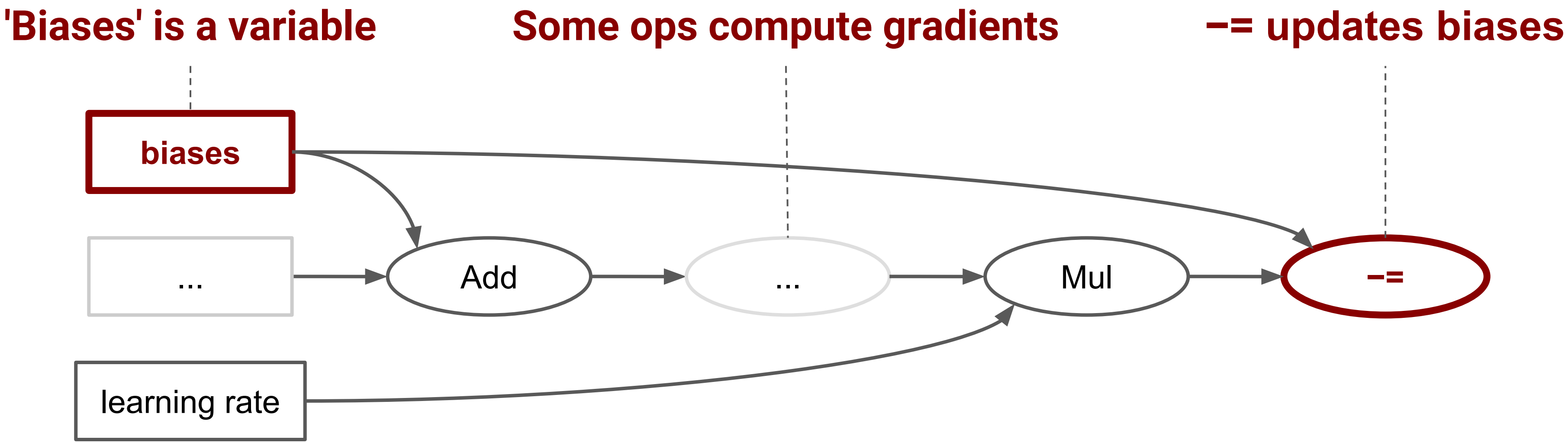
Computation is a dataflow graph

with tensors



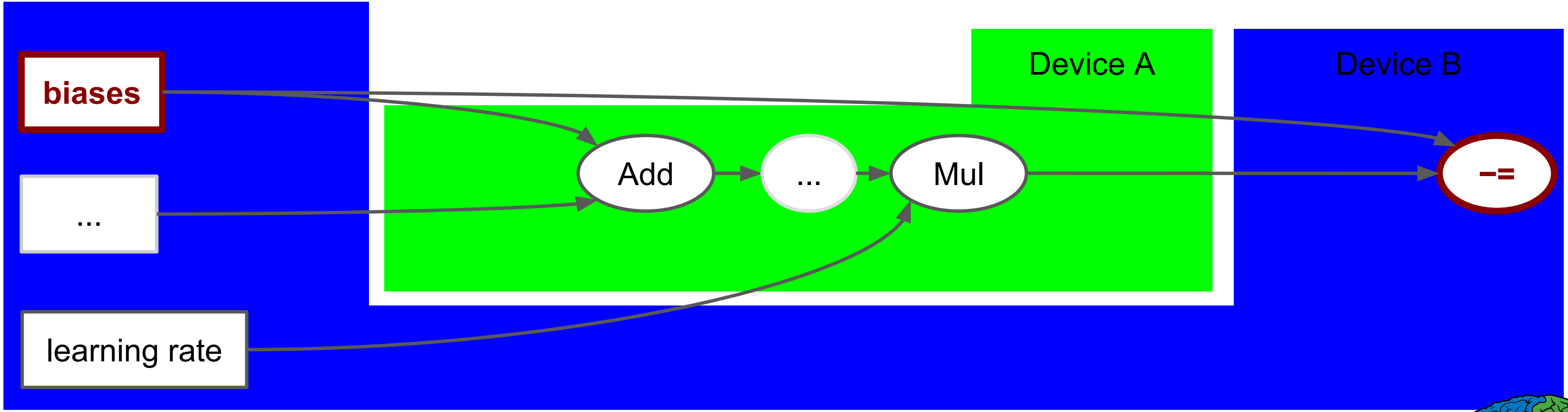
Computation is a dataflow graph

with state



Computation is a dataflow graph

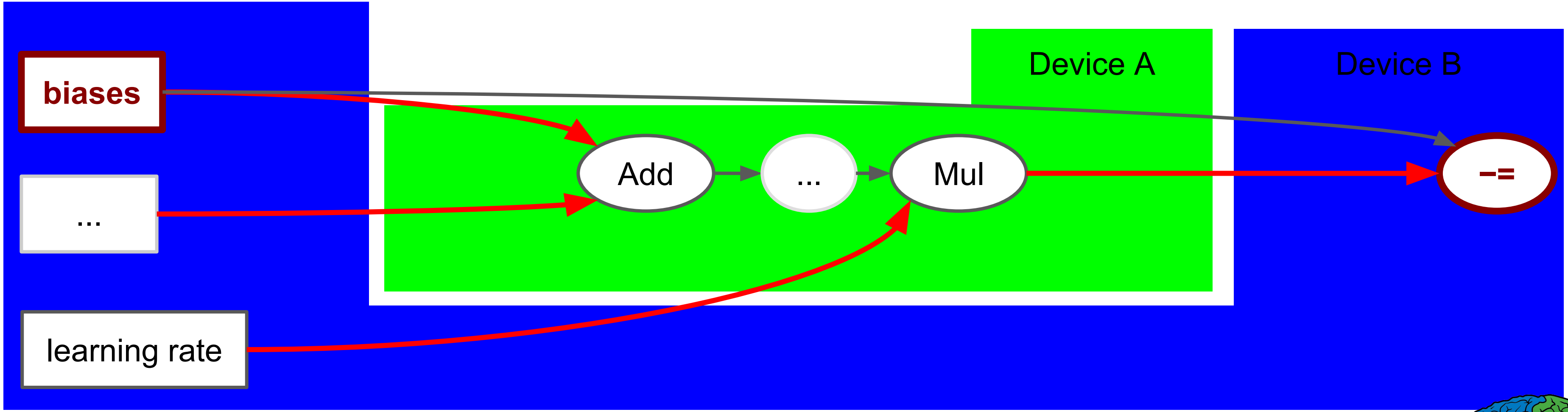
distributed



Devices: Processes, Machines, GPUs, etc

Send and Receive Nodes

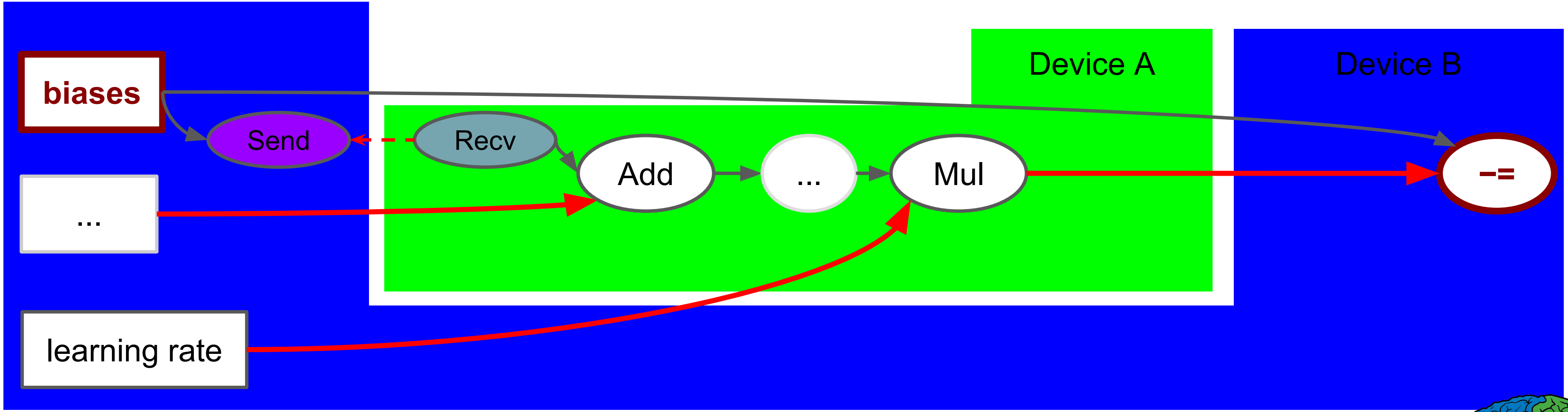
distributed



Devices: Processes, Machines, GPUs, etc

Send and Receive Nodes

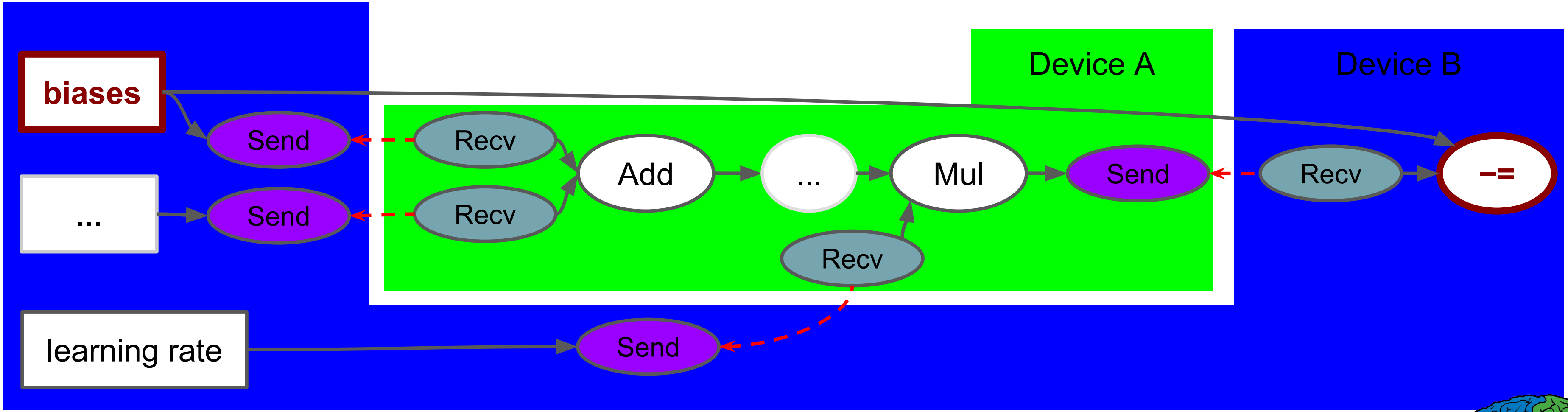
distributed



Devices: Processes, Machines, GPUs, etc

Send and Receive Nodes

distributed



Devices: Processes, Machines, GPUs, etc

Send and Receive Implementations

- Different implementations depending on source/dest devices
- e.g. GPUs on same machine: **local GPU → GPU copy**
- e.g. CPUs on different machines: **cross-machine RPC**
- e.g. GPUs on different machines: **RDMA**

Extensible

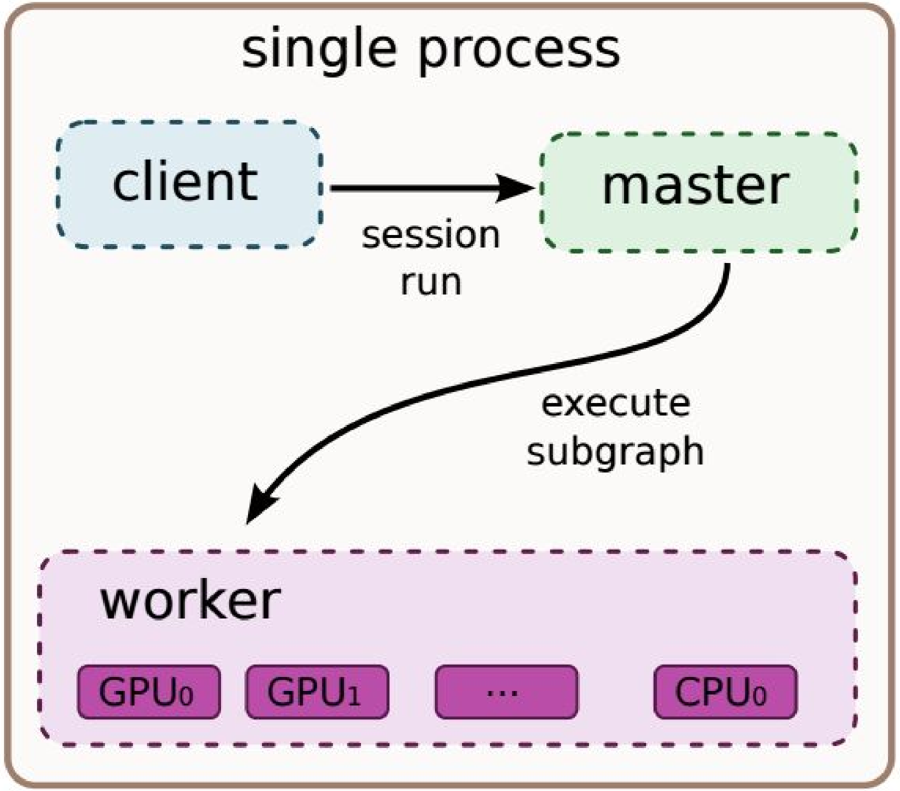
- Core system defines a number of standard ***operations*** and ***kernels*** (device-specific implementations of operations)
- Easy to define new operators and/or kernels

Session Interface

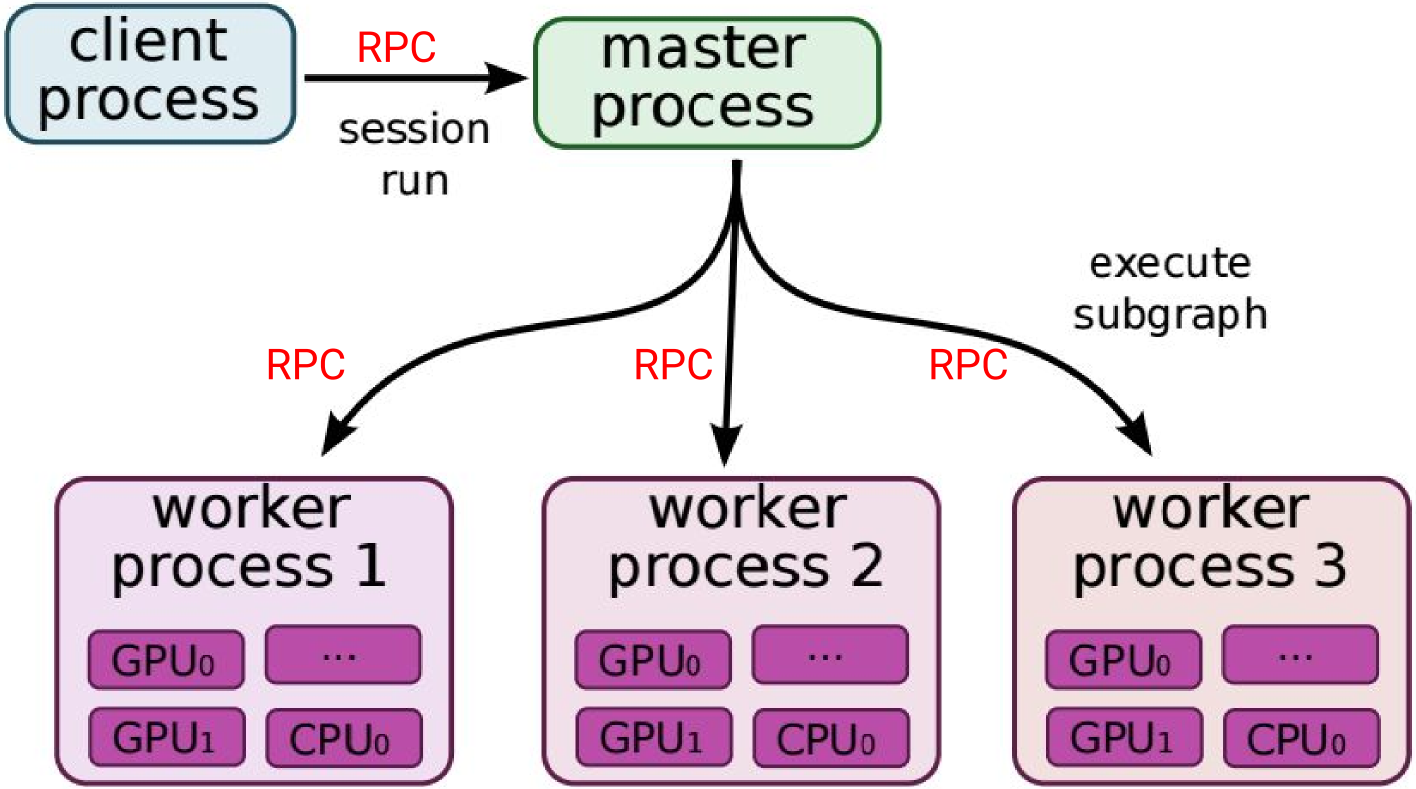
- `Extend`: add nodes to computation graph
- `Run`: execute an arbitrary subgraph
  - optionally feeding in Tensor inputs and retrieving Tensor output

Typically, setup a graph with one or a few `Extend` calls and then `Run` it thousands or millions or times

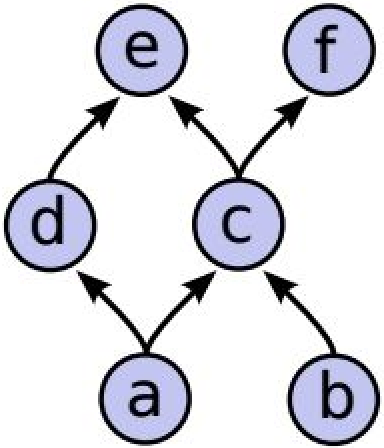
Single Process Configuration



Distributed Configuration

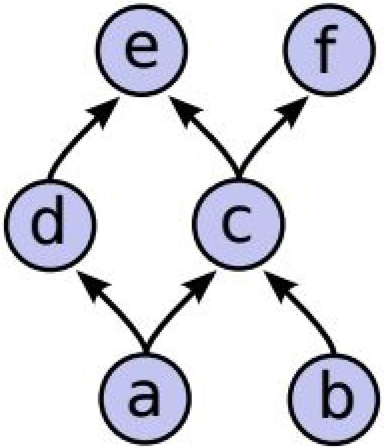


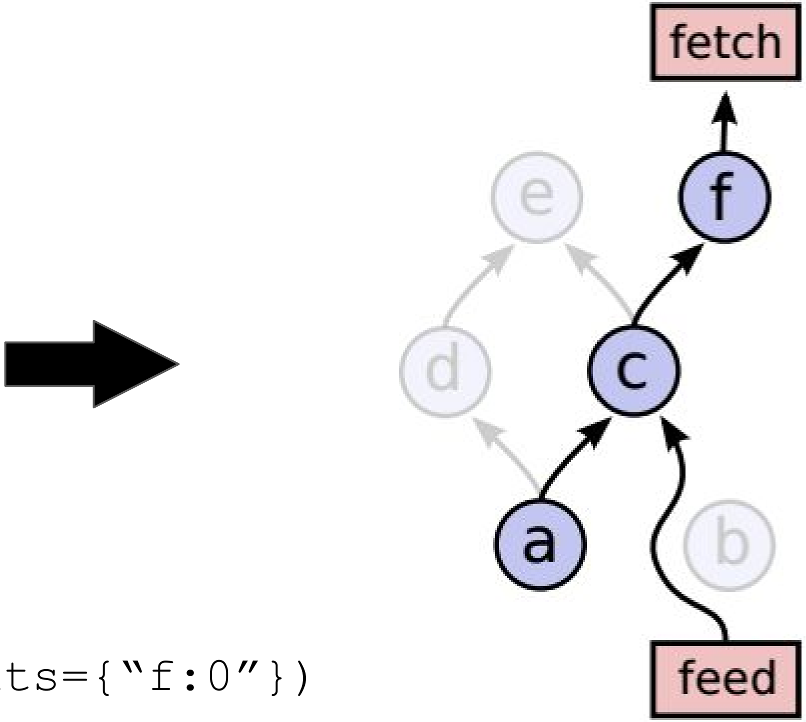
Feeding and Fetching




```
Run(input={"b": ...}, outputs={"f:0"})
```

Feeding and Fetching






```
Run(input={"b": ...}, outputs={"f:0"})
```

Example: Power method for Eigenvectors

- Simple 5x5 matrix, compute result, iterated K times
- TensorBoard graph visualization

Under the hood: Power method

- Operators
- Kernel implementations for different devices
- Run call
- Tensor memory management

Example: Symbolic differentiation

- $f(x) = x^T * W * x$; now minimize
- Show $df/dx = 2*Wx$ in graph

TensorFlow Single Device Performance

Initial measurements done by Soumith Chintala

Benchmark	Forward	Forward+Backward
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	70 ms	231 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	**96 ms**	**326 ms**

See https://github.com/soumith/convnet-benchmarks/issues/66

Two main factors:

(1) various overheads (nvcc doesn't like 64-bit tensor indices, etc.)

(2) versions of convolutional libraries being used (cuDNNv2 vs. v3, etc.)

TensorFlow Single Device Performance

Prong 1: Tackling sources of overhead

Benchmark	**Forward**	**Forward+Backward**
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	70 ms	231 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	96 ms	326 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (our machine)	97 ms	336 ms

TensorFlow Single Device Performance

Prong 1: Tackling sources of overhead

Benchmark	**Forward**	**Forward+Backward**
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	70 ms	231 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	96 ms	326 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (our machine)	**97 ms**	**336 ms**
AlexNet - cuDNNv2 on TensorFlow 0.6 (our machine: soon)	**70 ms (+39%)**	**230 ms (+31%)**

TensorFlow Single Device Performance

Prong 1: Tackling sources of overhead

Benchmark	Forward	Forward+Backward
AlexNet - cuDNNv3 on Torch (Soumith)	32 ms	96 ms
AlexNet - Neon (Soumith)	32 ms	101 ms
AlexNet - cuDNNv2 on Torch (Soumith)	**70 ms**	**231 ms**
AlexNet - cuDNNv2 on TensorFlow 0.5 (Soumith)	96 ms	326 ms
AlexNet - cuDNNv2 on TensorFlow 0.5 (our machine)	97 ms	336 ms
AlexNet - cuDNNv2 on TensorFlow 0.6 (our machine: soon)	**70 ms (+39%)**	**230 ms (+31%)**

TensorFlow Single Device Performance

TF 0.5 vs. 0.6 release candidate measurements (on our machine w/ Titan-X)

Benchmark	Forward	Forward+Backward
AlexNet - cuDNNv2 on TensorFlow 0.5	97 ms	336 ms
AlexNet - cuDNNv2 on TensorFlow 0.6 (soon)	**70 ms (+27%)**	**230 ms (+31%)**
OxfordNet - cuDNNv2 on TensorFlow 0.5	573 ms	1923 ms
OxfordNet - cuDNNv2 on TensorFlow 0.6 (soon)	**338 ms (+41%)**	**1240 ms (+36%)**
Overfeat - cuDNNv2 on TensorFlow 0.5	322 ms	1179 ms
Overfeat - cuDNNv2 on TensorFlow 0.6 (soon)	**198 ms (+39%)**	**832 ms (+29%)**

TensorFlow Single Device Performance

Prong 2: Upgrade to faster core libraries like cuDNN v3 (and/or the upcoming v4)

Won’t make it into 0.6 release later this week, but likely in next release

Single device performance important, but

....

biggest performance improvements come from large-scale distributed systems with model and data parallelism

Experiment Turnaround Time and Research Productivity

- **Minutes, Hours**:
  - **Interactive research! Instant gratification!**
- **1-4 days**
  - Tolerable
  - Interactivity replaced by running many experiments in parallel
- **1-4 weeks**
  - High value experiments only
  - Progress stalls
- **>1 month**
  - Don't even try

Transition

- How do you do this at scale?
- How does TensorFlow make distributed training easy?

Model Parallelism

- Best way to decrease training time: **decrease the step time**
- Many models have lots of inherent parallelism
- Problem is distributing work so communication doesn’t kill you
    - local connectivity (as found in CNNs)
    - towers with little or no connectivity between towers (e.g. AlexNet)
    - specialized parts of model active only for some examples

Exploiting Model Parallelism

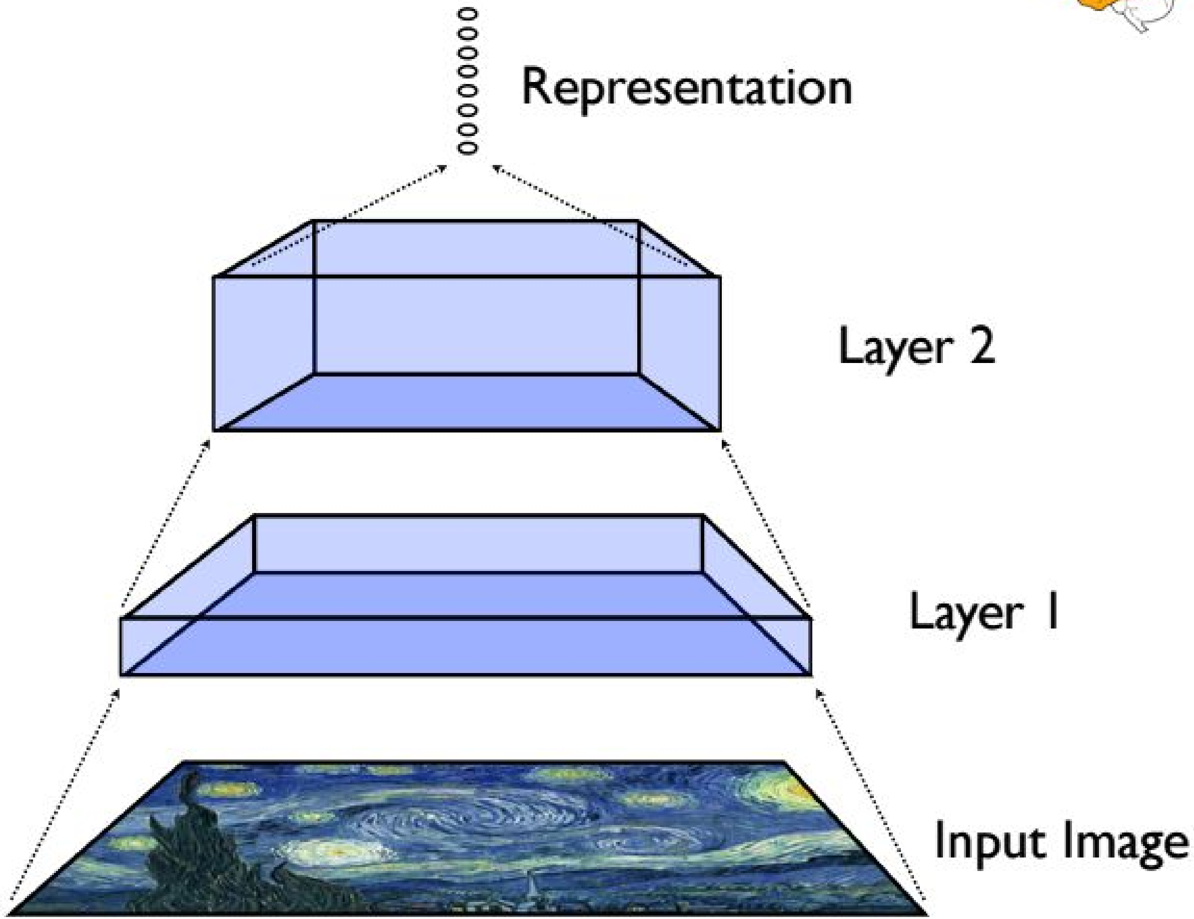
On a single core: Instruction parallelism (SIMD). Pretty much free.

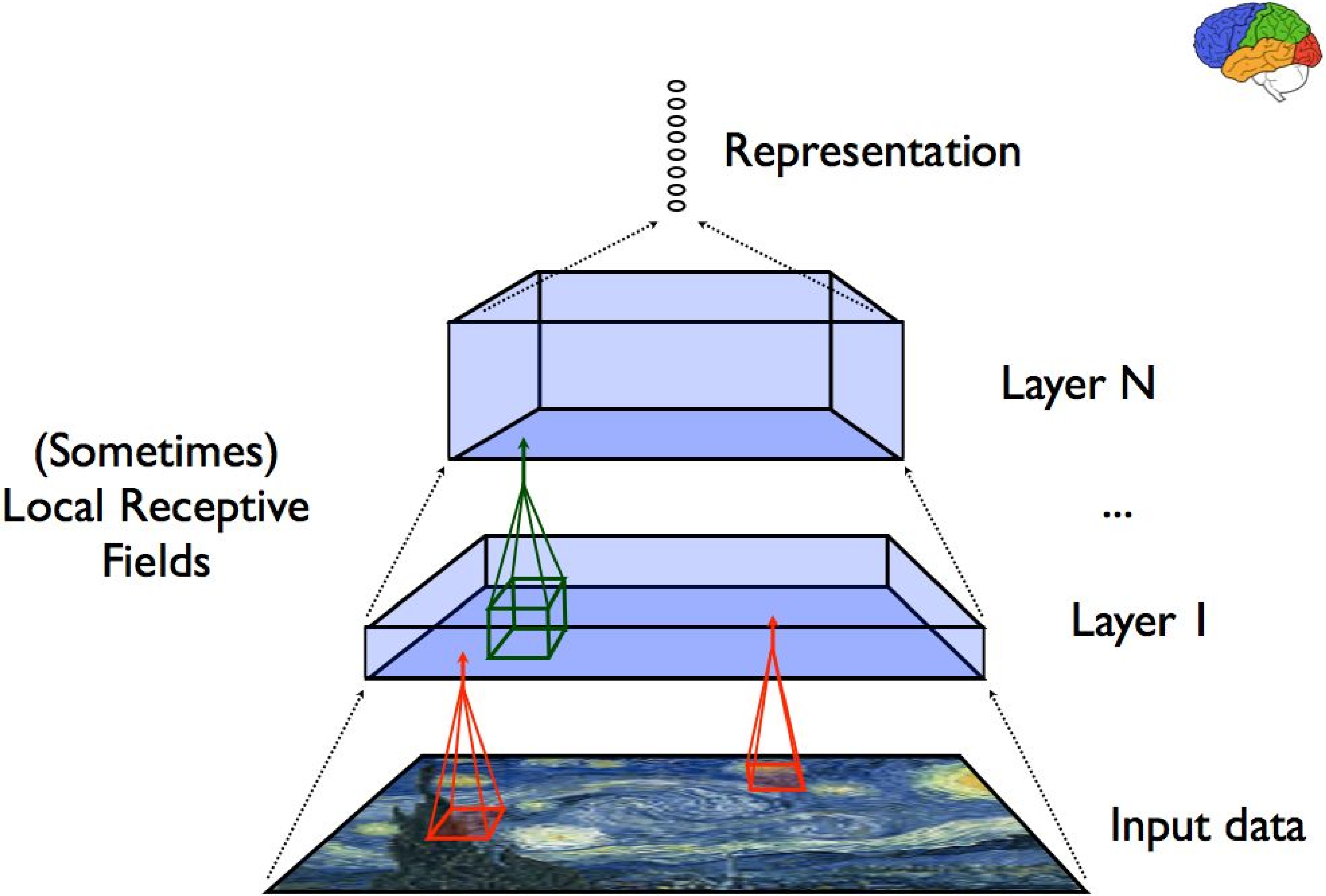
Across cores: thread parallelism. Almost free, unless across sockets, in which case inter-socket bandwidth matters (QPI on Intel).

Across devices: for GPUs, often limited by PCIe bandwidth.

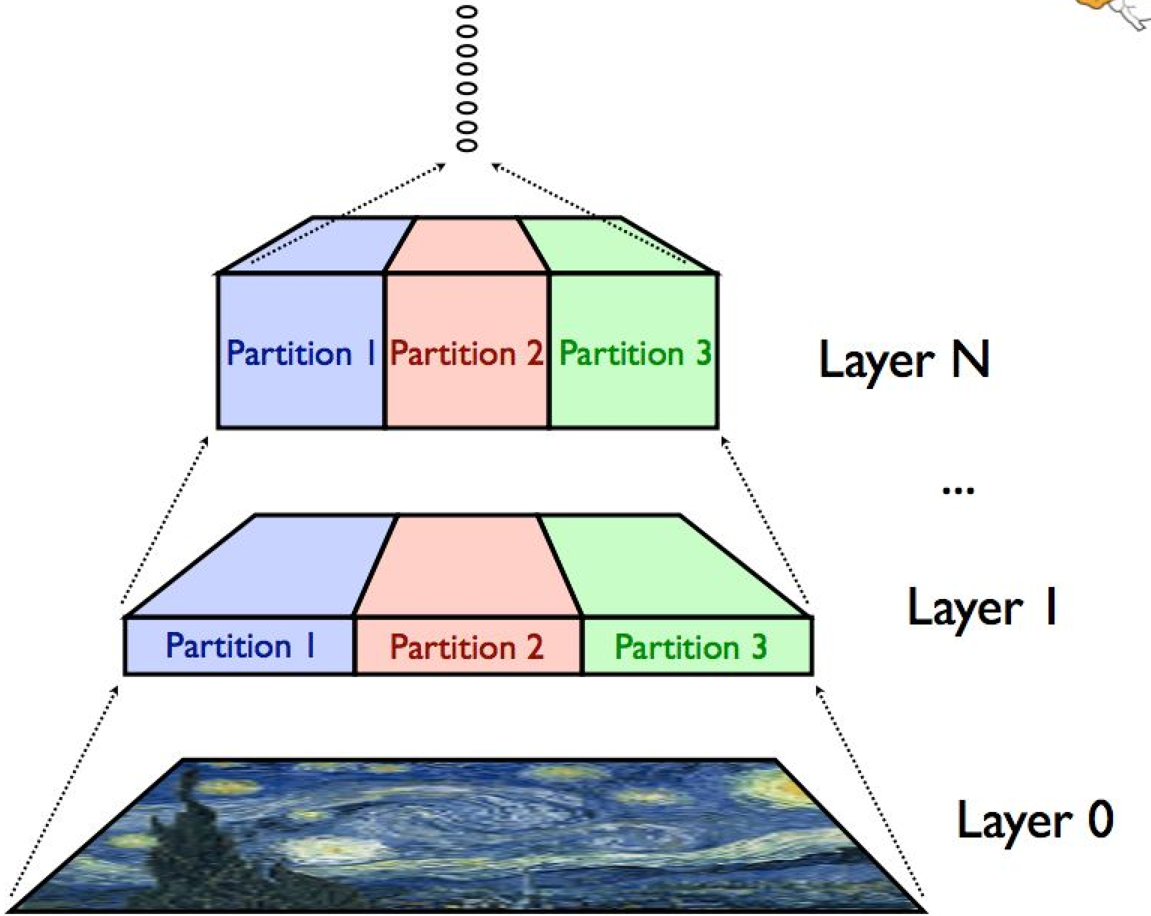
Across machines: limited by network bandwidth / latency

Model Parallelism

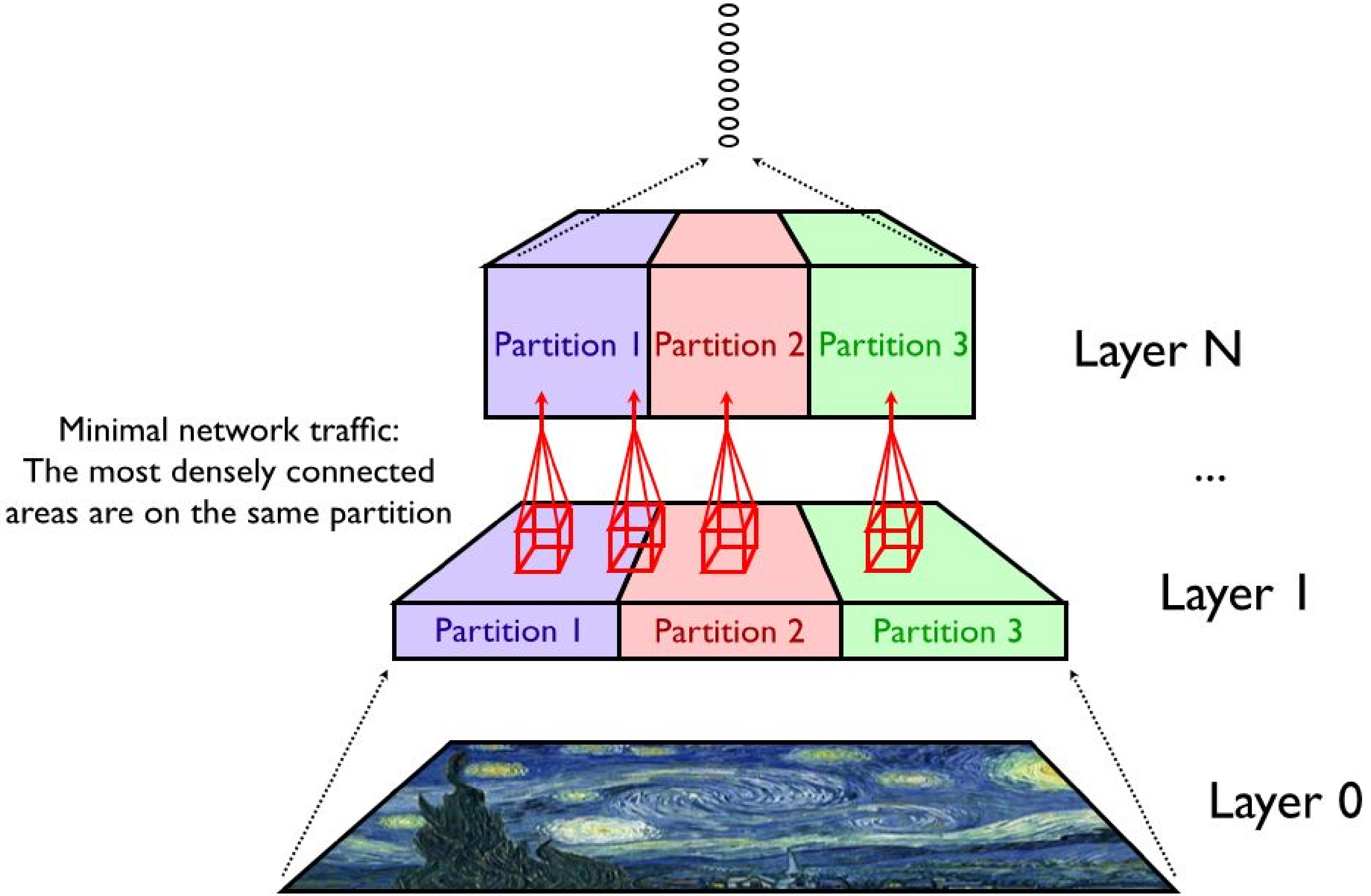



Representation
Layer N
(Sometimes)
Local Receptive
Fields
...
Layer 1
Input data

Model Parallelism: Partition model across machines



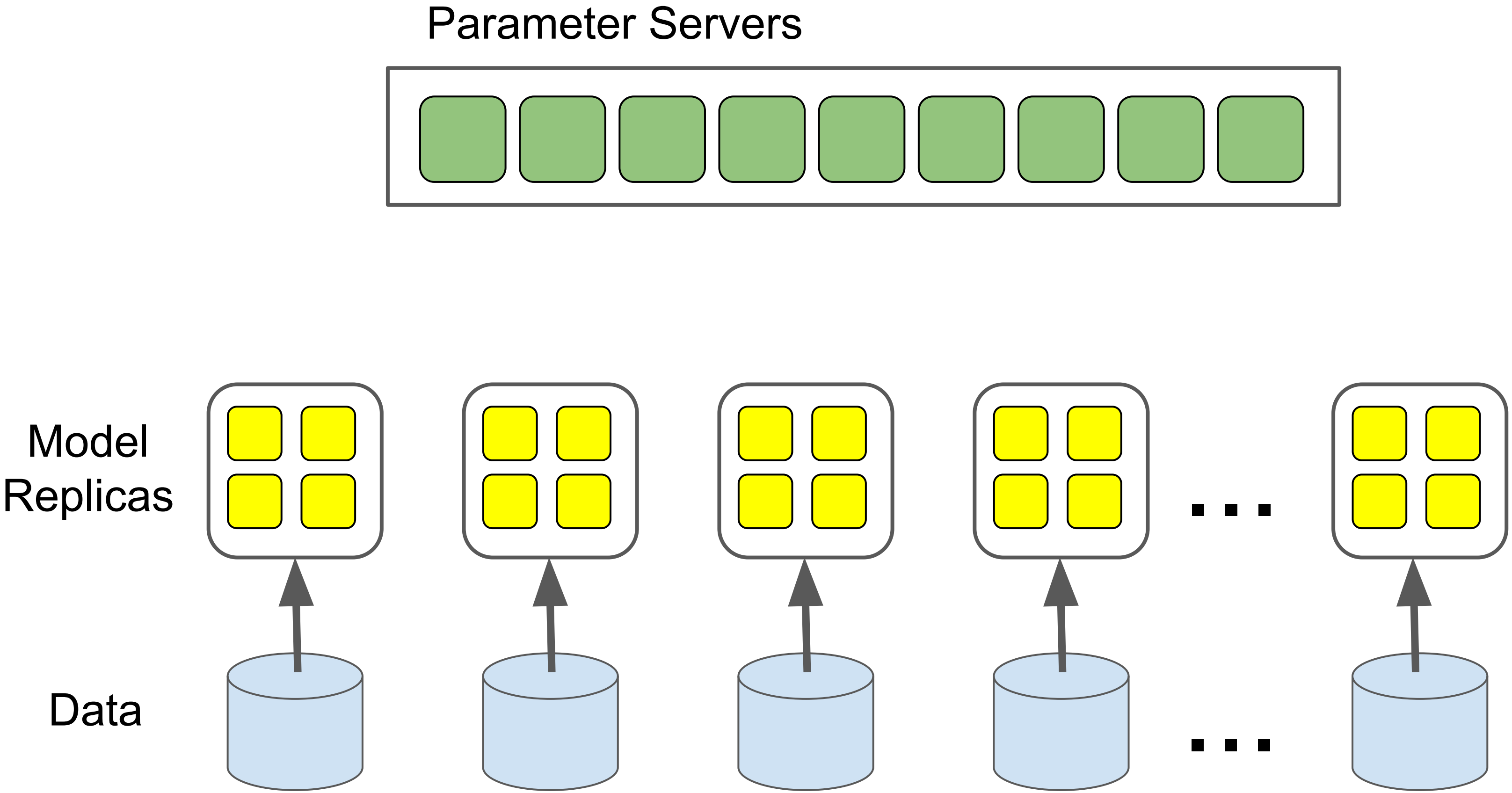
Model Parallelism: Partition model across machines



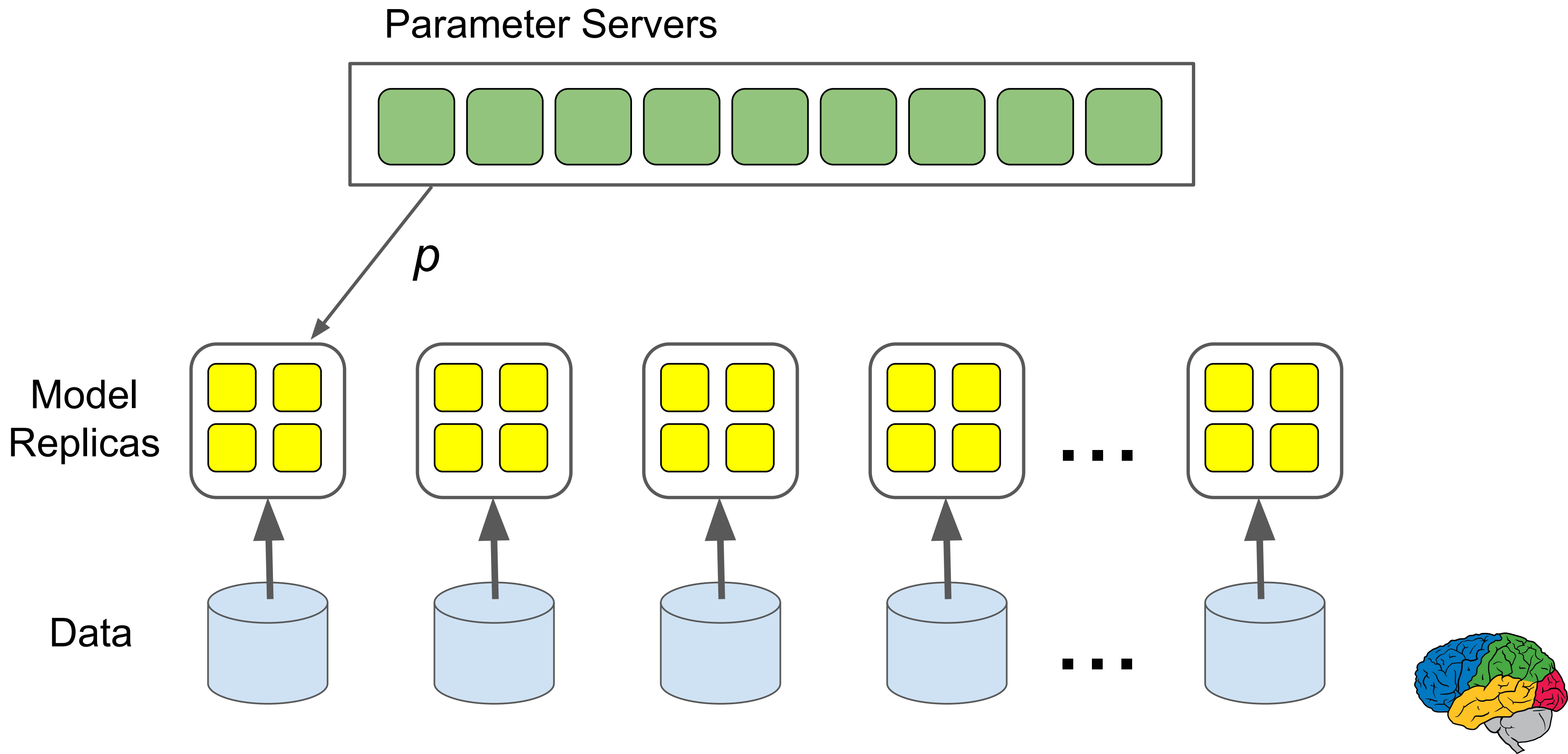
Data Parallelism

- Use multiple model replicas to process different examples at the same time
  - All collaborate to update model state (parameters) in shared parameter server(s)
- Speedups depend highly on kind of model
  - Dense models: 10-40X speedup from 50 replicas
  - Sparse models:
    - support many more replicas
    - often can use as many as 1000 replicas

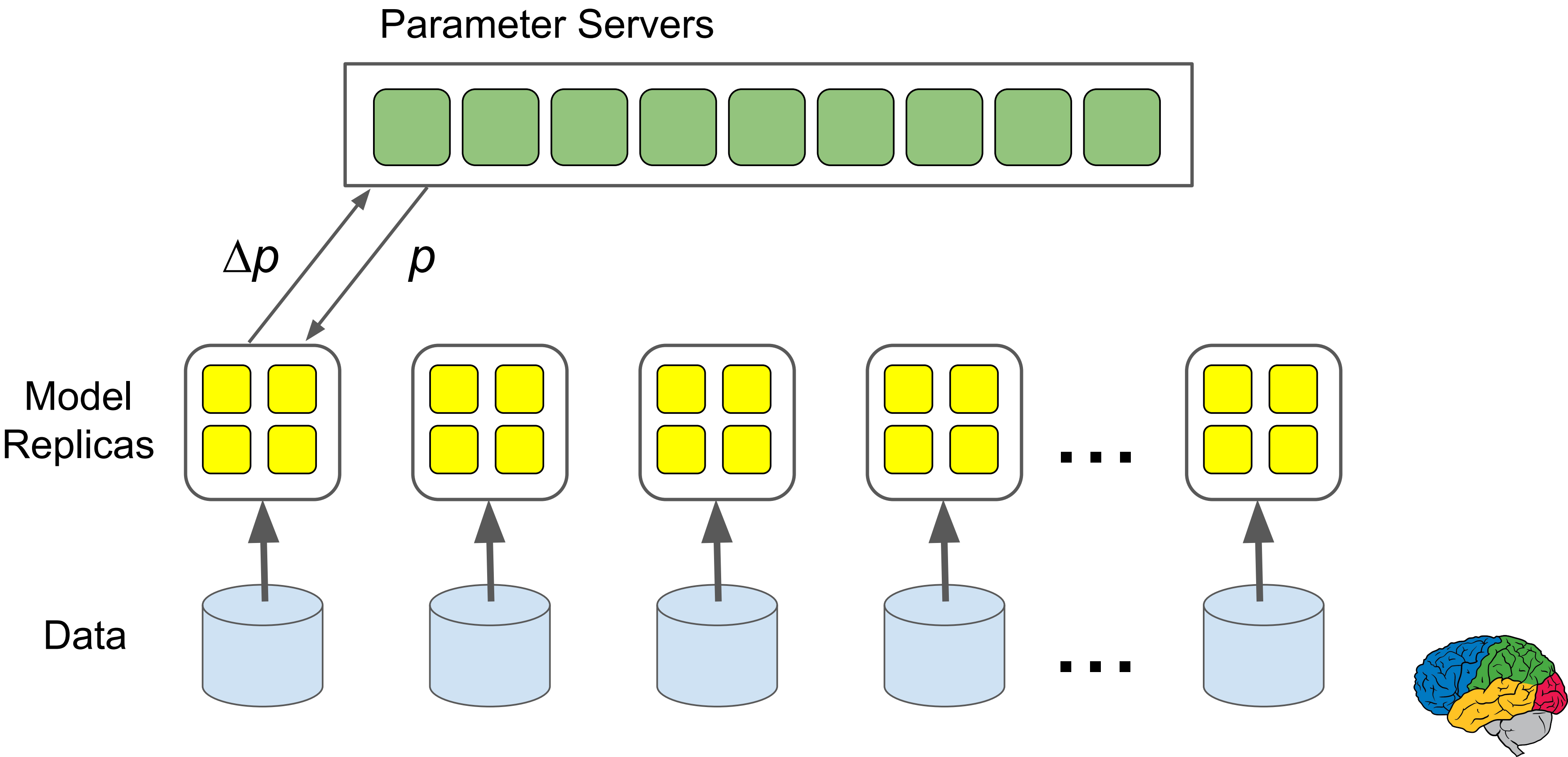
Data Parallelism



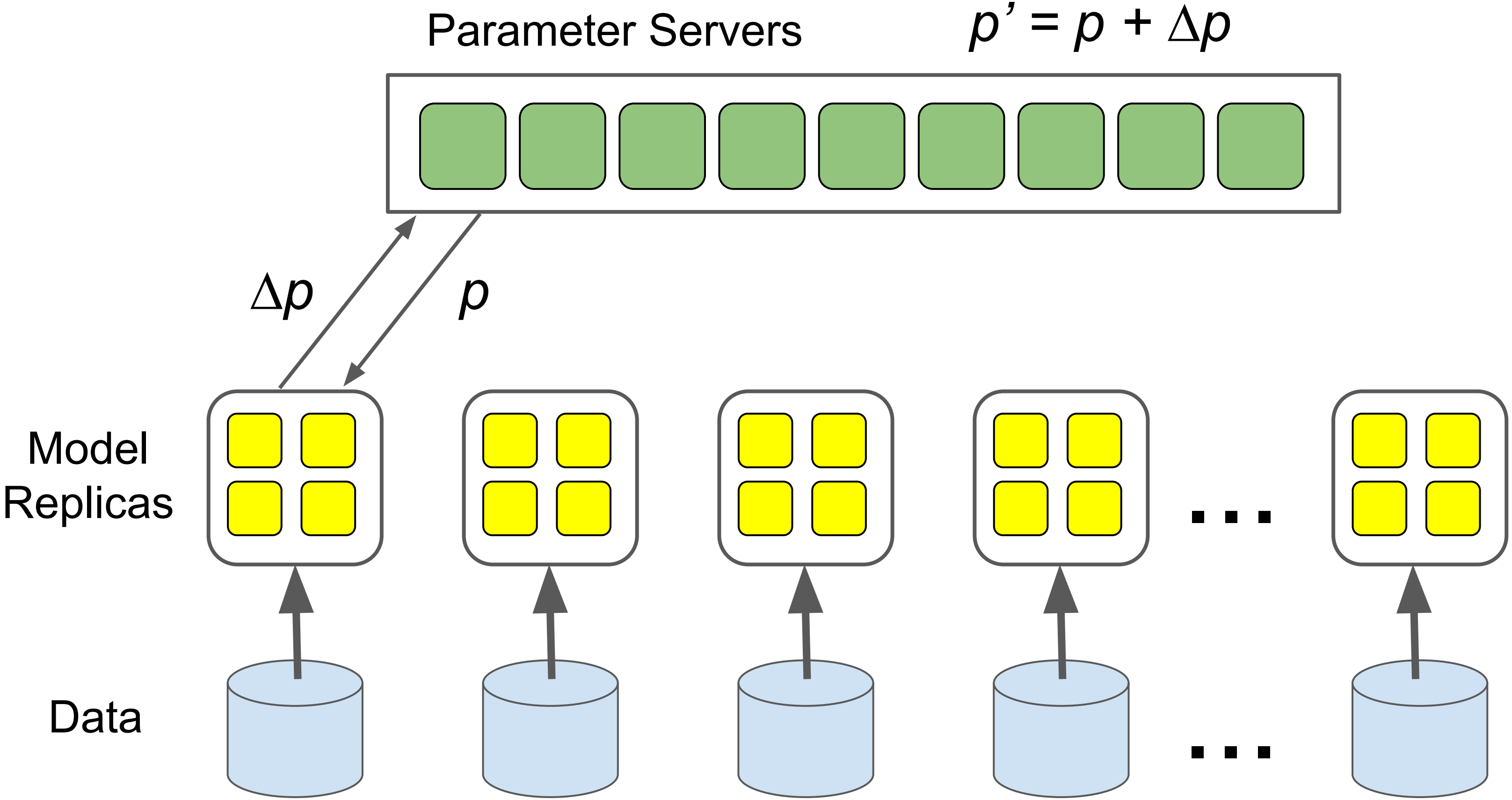
Data Parallelism



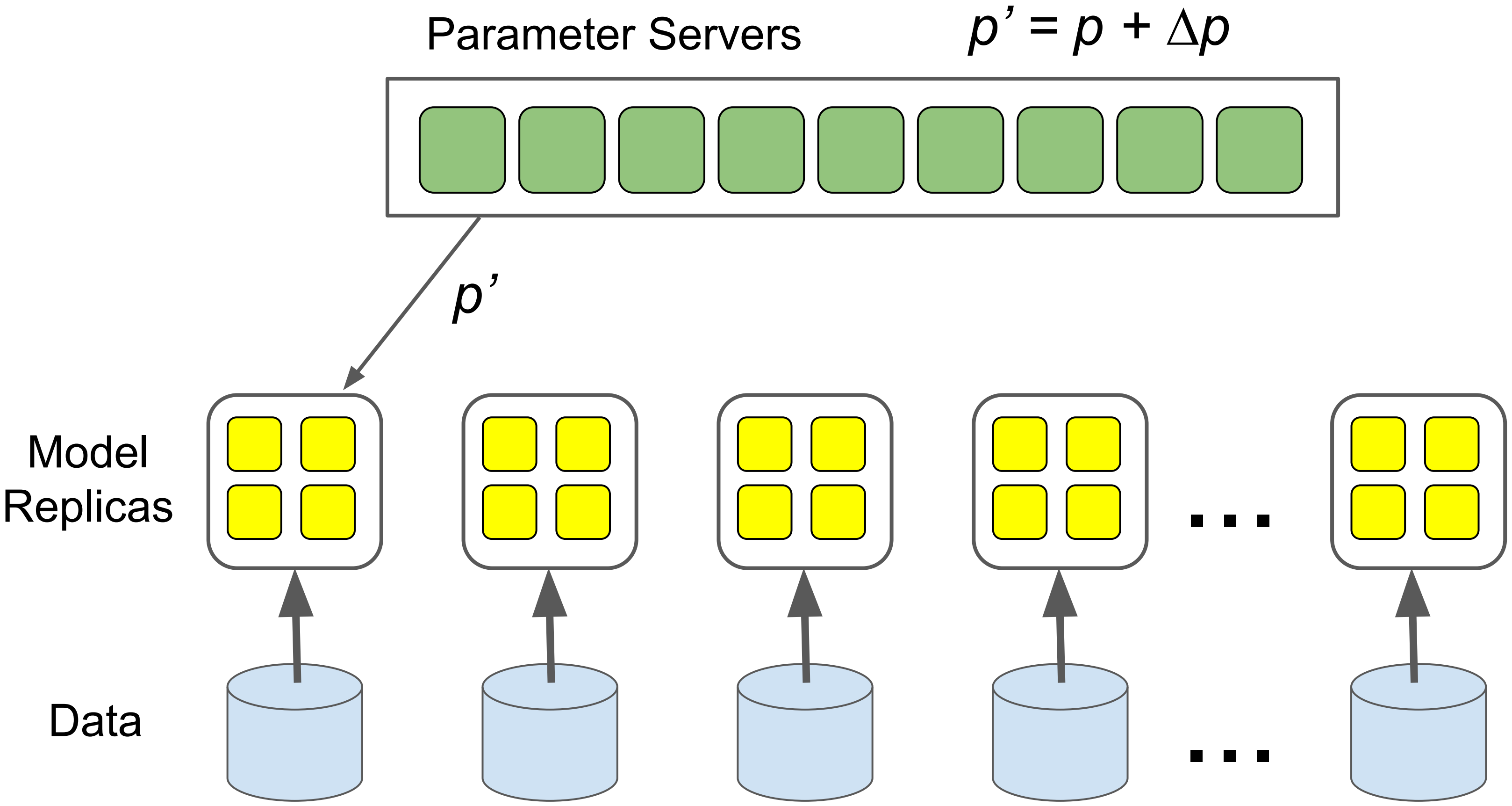
Data Parallelism



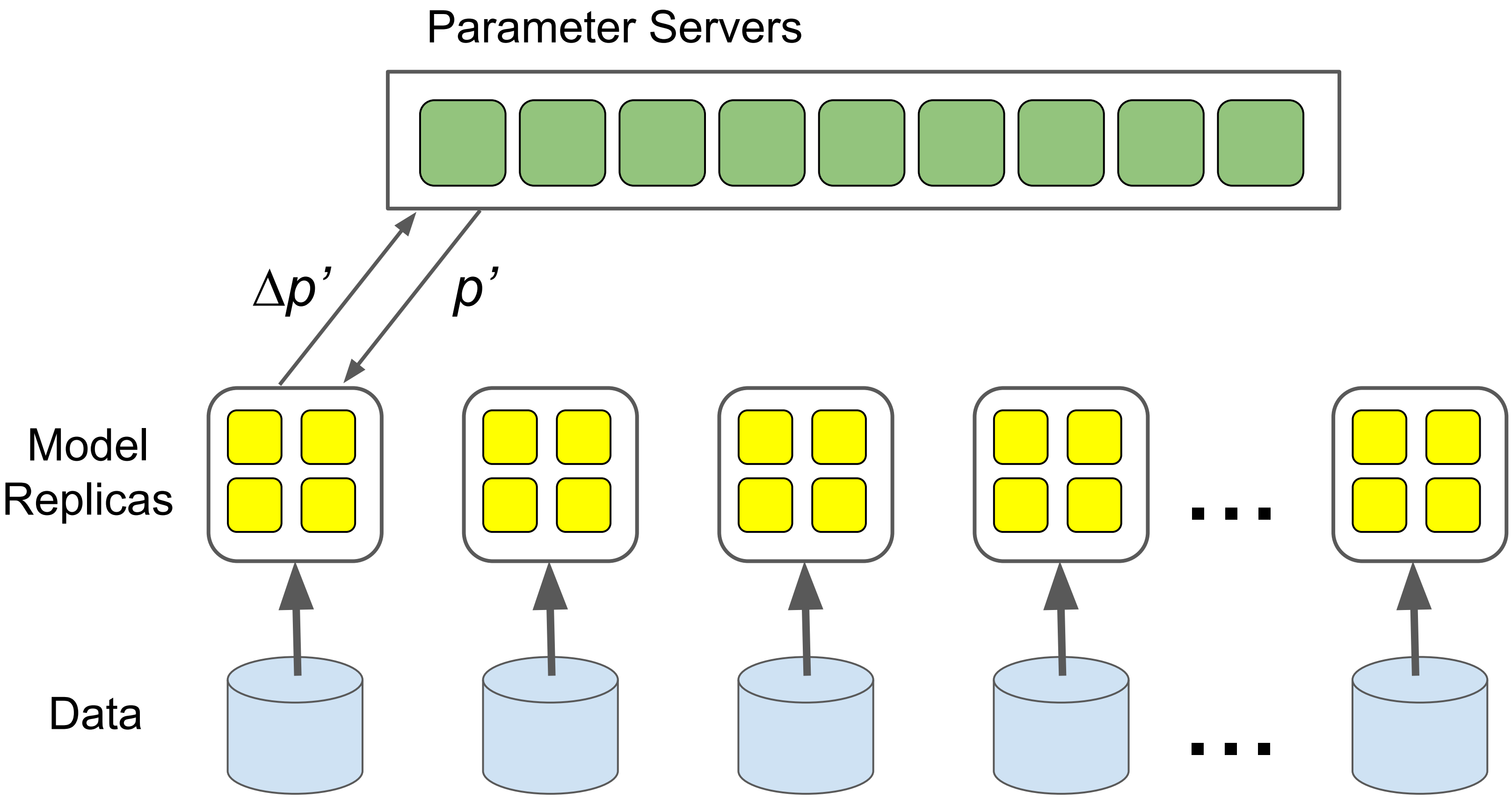
Data Parallelism



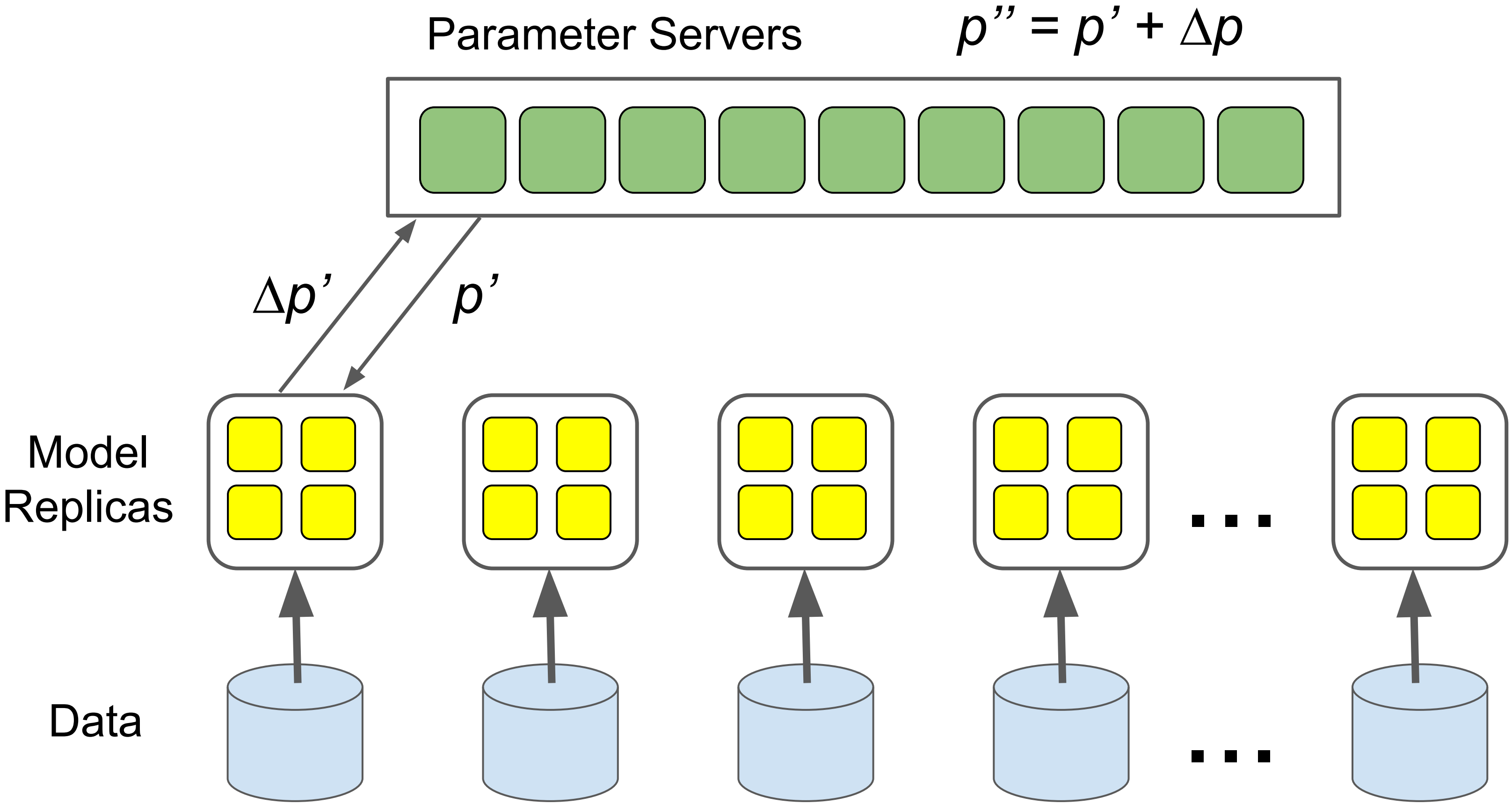
Data Parallelism



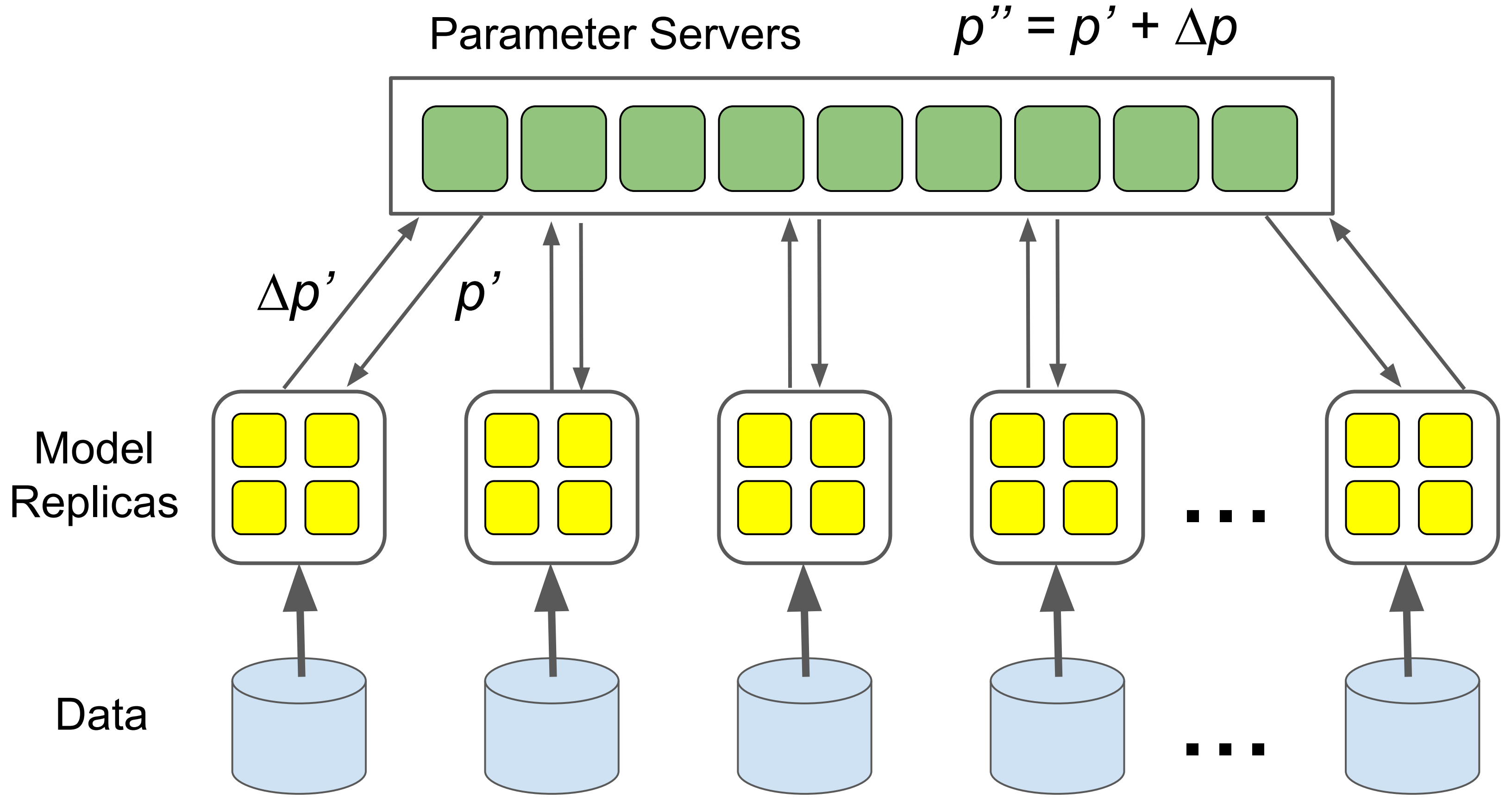
Data Parallelism



Data Parallelism



Data Parallelism



Data Parallelism Choices

Can do this **synchronously**:

- **N replicas** equivalent to an **N times larger batch size**
- Pro: No gradient staleness
- Con: Less fault tolerant (requires some recovery if any single machine fails)

Can do this **asynchronously**:

- Pro: Relatively fault tolerant (failure in model replica doesn't block other replicas)
- Con: Gradient staleness means each gradient less effective

(Or **hybrid**: M asynchronous groups of N synchronous replicas)

Data Parallelism Considerations

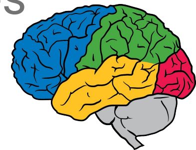
Want model computation time to be large relative to time to send/receive parameters over network

Models with fewer parameters, that reuse each parameter multiple times in the computation

- Mini-batches of size *B* reuse parameters *B* times

Certain model structures **reuse each parameter** many times within each example:

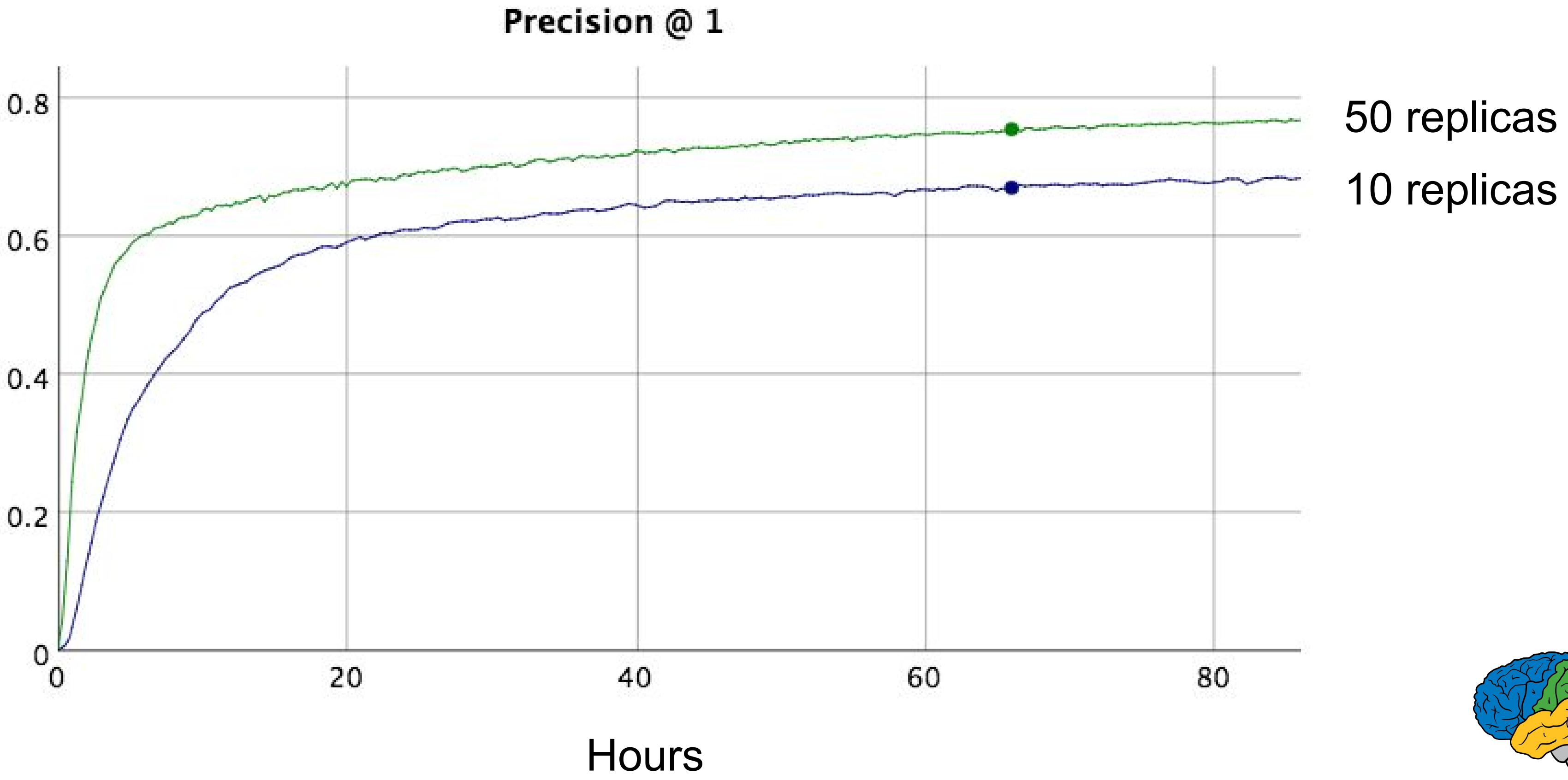
- **Convolutional models** tend to reuse hundreds or thousands of times per example (for different spatial positions)
- **Recurrent models** (LSTMs, RNNs) tend to reuse tens to hundreds of times (for unrolling through *T* time steps during training)



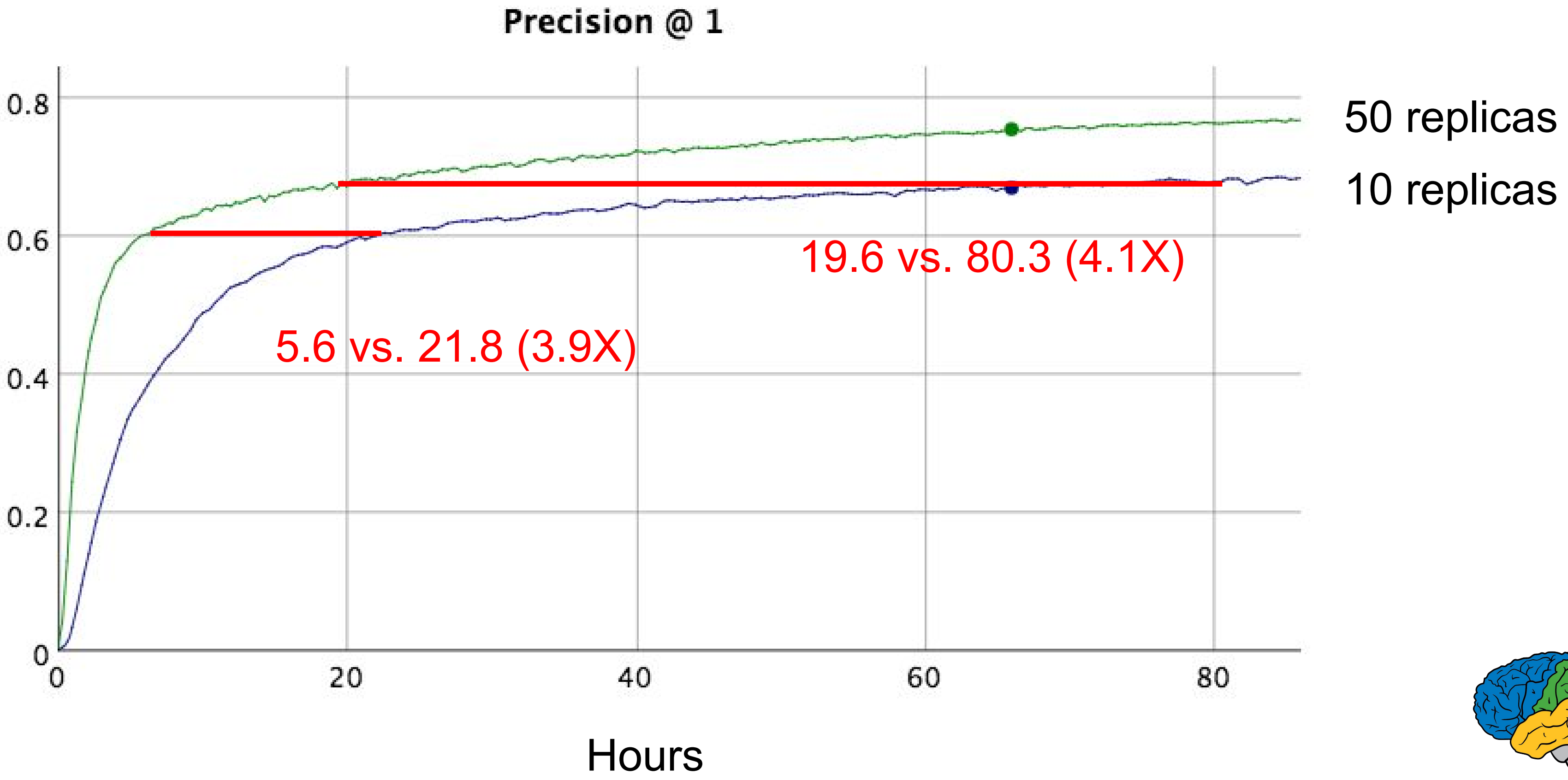
Success of Data Parallelism

- Data parallelism is **really important** for many of Google's problems (very large datasets, large models):
    - RankBrain uses 500 replicas
    - ImageNet Inception training uses 50 GPUs, ~40X speedup
    - SmartReply uses 16 replicas, each with multiple GPUs
    - State-of-the-art on LM "One Billion Word" Benchmark model uses both data and model parallelism on 32 GPUs

10 vs 50 Replica Inception Synchronous Training



10 vs 50 Replica Inception Synchronous Training



Using TensorFlow for Parallelism

Trivial to express both model parallelism as well as data parallelism

- Very minimal changes to single device model code

Devices and Graph Placement

- Given a graph and set of devices, TensorFlow implementation must decide which device executes each node

Full and Partial Device Constraints (Hints)

Devices are named hierarchically:

```
/job:localhost/device:cpu:0
/job:worker/task:17/device:gpu:3
/job:parameters/task:4/device:cpu:0
```

Client can specify full or partial constraints for nodes in graph:

"Place this node on `/job:localhost/device:gpu:2`"

"Place this node on `/device:gpu:*`"

Placement Algorithm

Given hints, plus a cost model (node execution time estimates and Tensor size estimates), make placement decisions

- Current relatively simple greedy algorithm
- Active area of work

Show CIFAR10 placement TensorBoard.

Example: LSTM **[Hochreiter et al, 1997]**

- From research paper to code

$$i_t = W_{ix}x_t + W_{ih}h_{t-1} + b_i$$

$$j_t = W_{jx}x_t + W_{jh}h_{t-1} + b_j$$

$$f_t = W_{fx}x_t + W_{fh}h_{t-1} + b_f$$

$$o_t = W_{ox}x_t + W_{oh}h_{t-1} + b_o$$

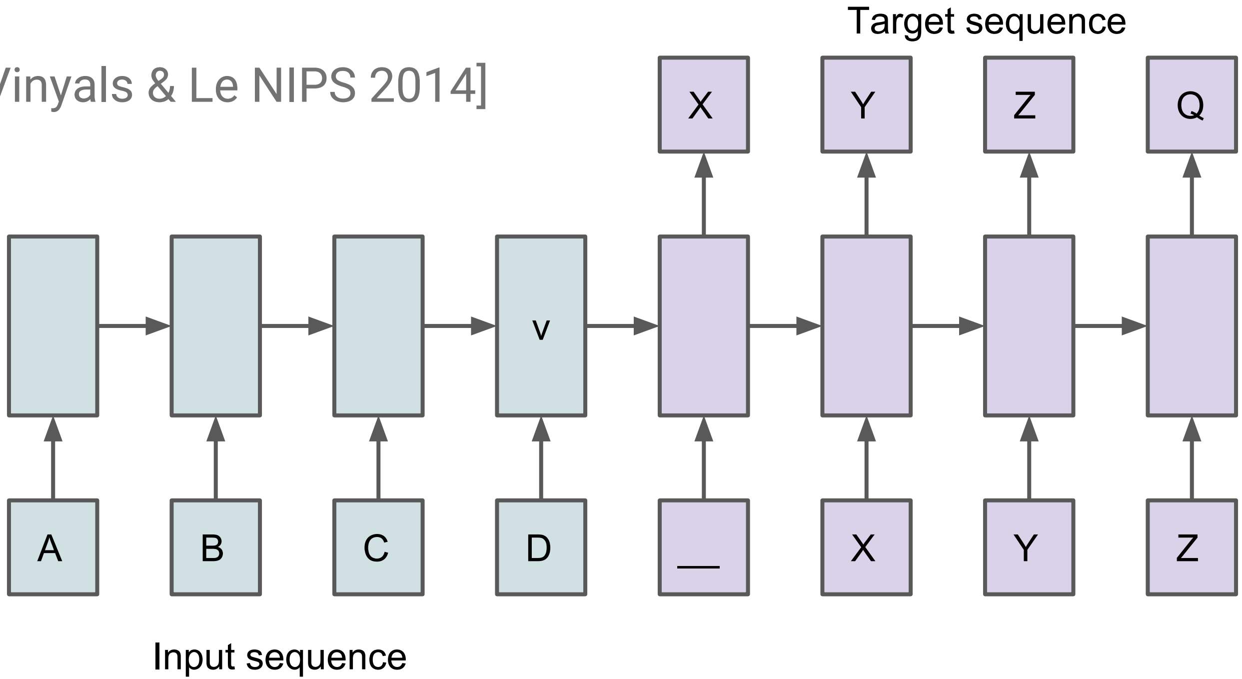
$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot \tanh(j_t)$$

$$h_t = \sigma(o_t) \odot \tanh(c_t)$$

```
def __call__(self, inputs, state, scope=None):
  """Long short-term memory cell (LSTM)."""
  with vs.variable_scope(scope or type(self).__name__):  # "BasicLSTMCell"
    # Parameters of gates are concatenated into one multiply for efficiency.
    c, h = array_ops.split(1, 2, state)
    concat = linear([inputs, h], 4 * self._num_units, True)

    # i = input_gate, j = new_input, f = forget_gate, o = output_gate
    i, j, f, o = array_ops.split(1, 4, concat)

    new_c = c * sigmoid(f + self._forget_bias) + sigmoid(i) * tanh(j)
    new_h = tanh(new_c) * sigmoid(o)

  return new_h, array_ops.concat(1, [new_c, new_h])
```

Sequence-to-Sequence Model

[Sutskever & Vinyals & Le NIPS 2014]



$$P(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Sequence-to-Sequence

- Active area of research
- Many groups actively pursuing RNN/LSTM
    - Montreal
    - Stanford
    - U of Toronto
    - Berkeley
    - Google
    - ...
- Further Improvements
    - Attention
    - NTM / Memory Nets
    - ...

Sequence-to-Sequence

- **Translation:** [Kalchbrenner *et al.*, EMNLP 2013][Cho *et al.*, EMLP 2014][Sutskever & Vinyals & Le, NIPS 2014][Luong *et al.*, ACL 2015][Bahdanau *et al.*, ICLR 2015]
- **Image captions:** [Mao *et al.*, ICLR 2015][Vinyals *et al.*, CVPR 2015][Donahue *et al.*, CVPR 2015][Xu *et al.*, ICML 2015]
- **Speech:** [Chorowsky *et al.*, NIPS DL 2014][Chan *et al.*, arxiv 2015]
- **Language Understanding:** [Vinyals & Kaiser *et al.*, NIPS 2015][Kiros *et al.,* NIPS 2015]
- **Dialogue:** [Shang *et al.*, ACL 2015][Sordoni *et al.*, NAACL 2015][Vinyals & Le, ICML DL 2015]
- **Video Generation:** [Srivastava *et al.*, ICML 2015]
- **Algorithms:** [Zaremba & Sutskever, arxiv 2014][Vinyals & Fortunato & Jaitly, NIPS 2015][Kaiser & Sutskever, arxiv 2015][Zaremba *et al.*, arxiv 2015]

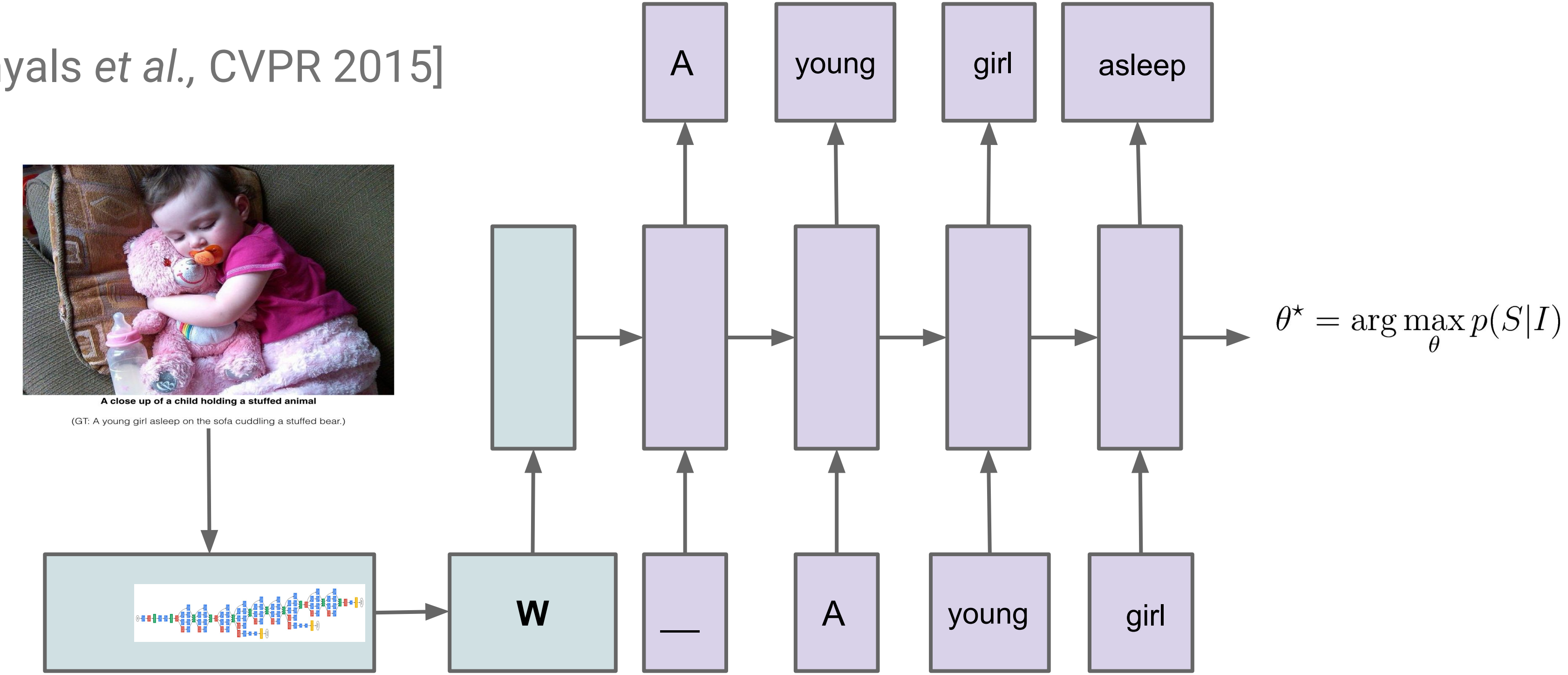
How to do Image Captions?

P(English | French)

P(English | Image)

How?

[Vinyals *et al.*, CVPR 2015]



Human: A young girl asleep on the sofa cuddling a stuffed bear.

NIC: A close up of a child holding a stuffed animal.

NIC: A baby is asleep next to a teddy bear.

(Recent) Captioning Results

Source: http://mscoco.org/dataset/#leaderboard-cap

Method	Meteor	CIDEr	LSUN	LSUN (2)
Google NIC	**0.346 (1)**	**0.946 (1)**	**0.273 (2)**	**0.317 (2)**
MSR Capt	0.339 (2)	0.937 (2)	0.250 (3)	0.301 (3)
UCLA/Baidu v2	0.325 (5)	0.935 (3)	0.223 (5)	0.252 (7)
MSR	0.331 (4)	0.925 (4)	**0.268 (2)**	**0.322 (2)**
MSR Nearest	0.318 (10)	0.916 (5)	0.216 (6)	0.255 (6)
Human	0.335 (3)	0.910 (6)	**0.638 (1)**	**0.675 (1)**
UCLA/Baidu v1	0.320 (8)	0.896 (7)	0.190 (9)	0.241 (8)
LRCN Berkeley	0.322 (7)	0.891 (8)	0.246 (4)	0.268 (5)
UofM/Toronto	0.323 (6)	0.878 (9)	0.262 (3)	0.272 (4)

Human: A close up of two bananas with bottles in the background.

BestModel: A bunch of bananas and a bottle of wine.

InitialModel: A close up of a plate of food on a table.

Human: A view of inside of a car where a cat is laying down.

BestModel: A cat sitting on top of a black car.

InitialModel: A dog sitting in the passenger seat of a car.

Human: A brown dog laying in a red wicker bed.

BestModel: A small dog is sitting on a chair.

InitialModel: A large brown dog laying on top of a couch.

Human: A man outside cooking with a sub in his hand.

BestModel: A man is holding a sandwich in his hand.

InitialModel: A man cutting a cake with a knife.

Human: Someone is using a small grill to melt his sandwich.

BestModel: A person is cooking some food on a grill.

InitialModel: A pizza sitting on top of a white plate.

Human: A woman holding up a yellow banana to her face.

BestModel: A woman holding a banana up to her face.

InitialModel: A close up of a person eating a hot dog.

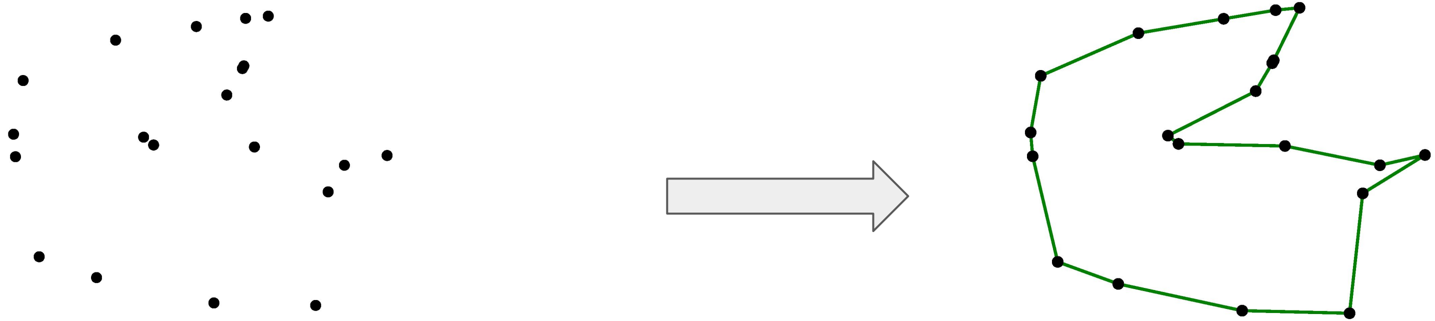
Human: A blue , yellow and red train travels across the tracks near a depot.

BestModel: A blue and yellow train traveling down train tracks.

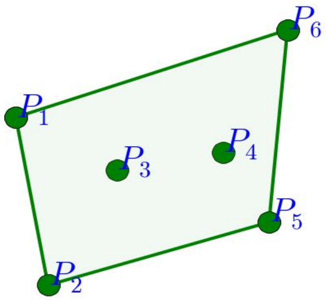
InitialModel: A train that is sitting on the tracks.

Pointer Networks Teaser

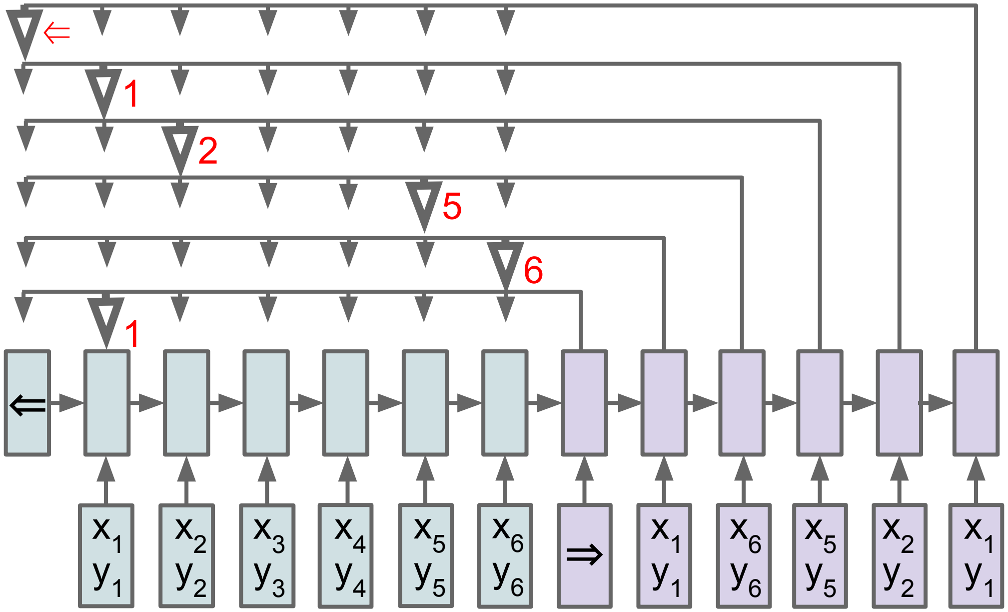
- Goal: Mappings where outputs are (sub)sets of inputs
- Travelling Salesman Problem

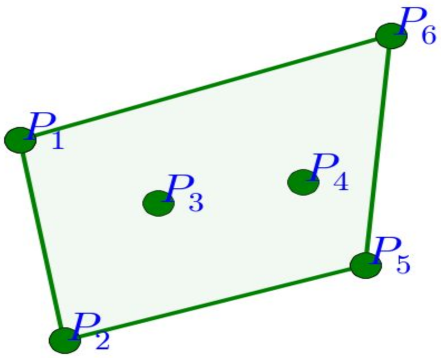


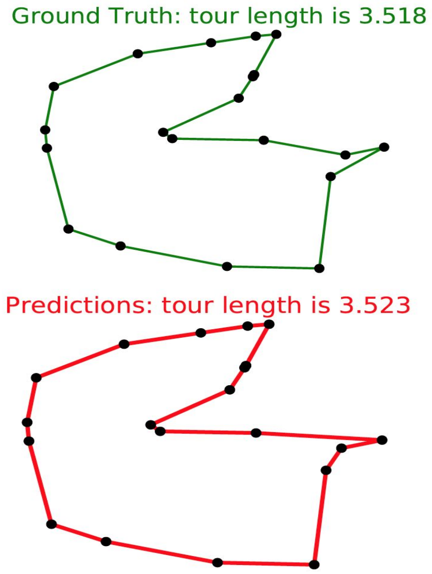
- Convex Hulls



Pointer Networks







Poster => Wed. 210C #22

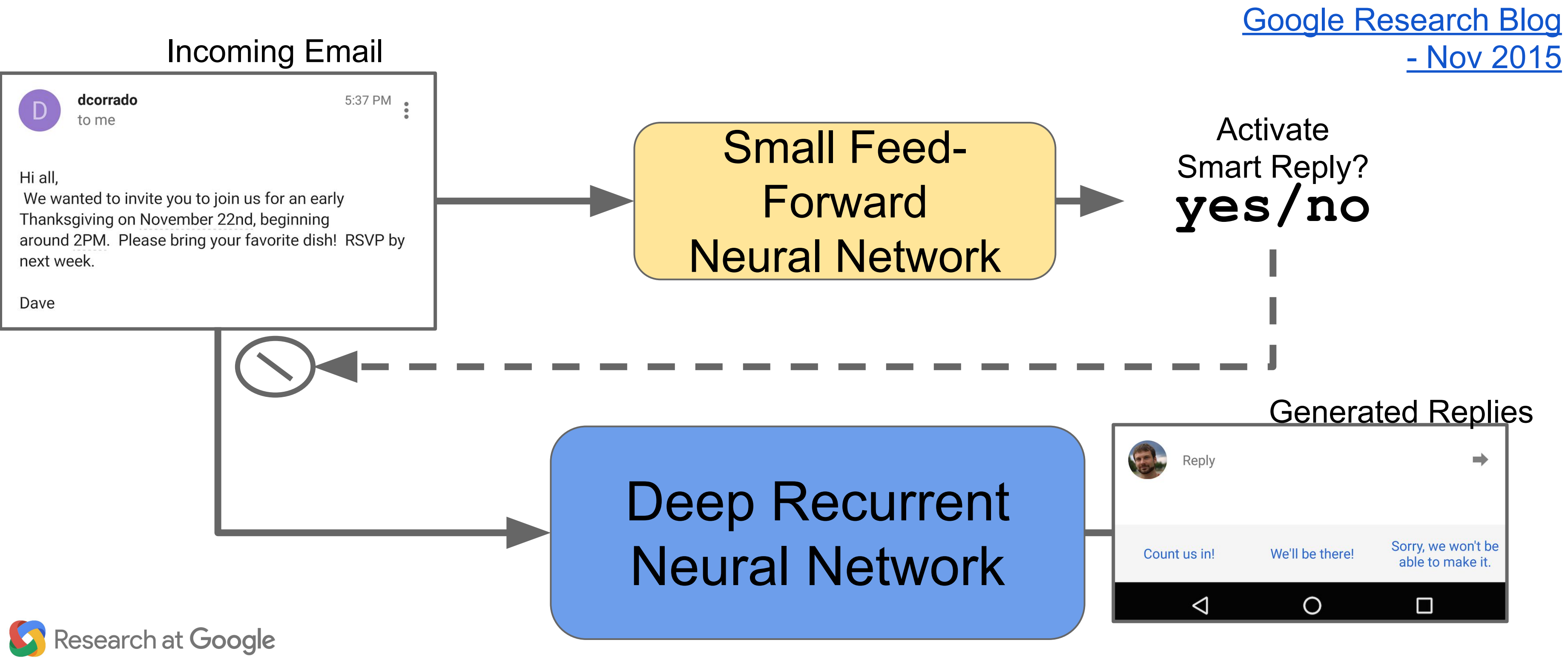
Neural Conversational Models

- Take movie subtitles (~900M words) or IT HelpDesk chats
- Predict the next dialog from history

i got to go .
no .
i get too emotional when i drink .
have another beer . i 've got to get up early .
no , you don 't . sit down .
i get too emotional when i drink .
will you have another beer ?
i 've got to go !
why ?
i got to get up early in the morning .
you 're drunk .
and emotional !
you got to go .

[Vinyals & Le ICML DL Workshop 2015]

Smart Reply



Google Research Blog - Nov 2015
Incoming Email
dcorrado
to me
5:37 PM
Hi all,
We wanted to invite you to join us for an early Thanksgiving on November 22nd, beginning around 2PM. Please bring your favorite dish! RSVP by next week.
Dave
Small Feed-Forward Neural Network
Activate Smart Reply?
yes/no
Deep Recurrent Neural Network
Generated Replies
Reply
Count us in!
We'll be there!
Sorry, we won't be able to make it.
Research at Google

Example: LSTM

```
for i in range(20):
    m, c = LSTMCell(x[i], mprev, cprev)
    mprev = m
    cprev = c
```

Example: Deep LSTM

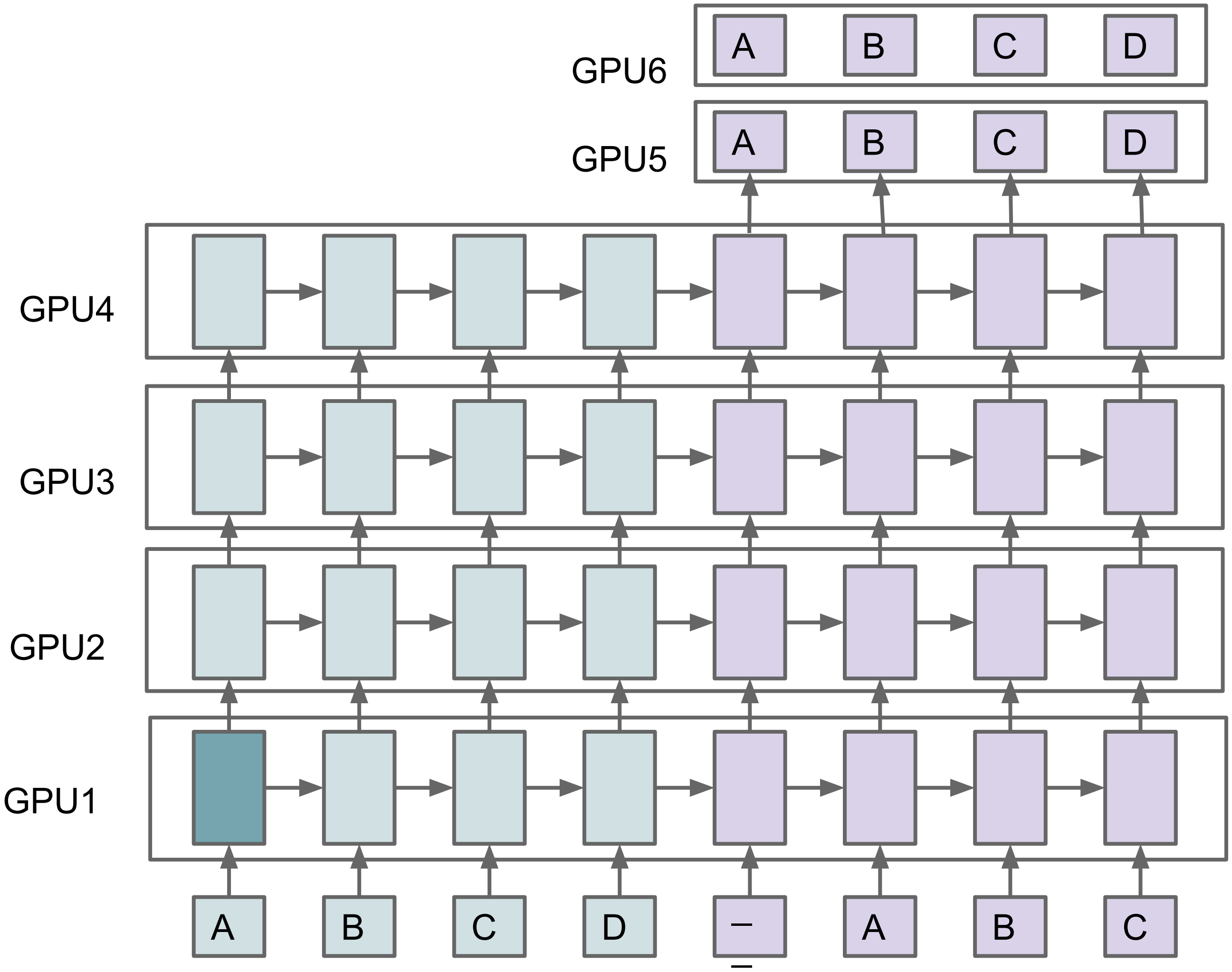
```
for i in range(20):
  for d in range(4): # d is depth
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

Example: Deep LSTM

```
for i in range(20):
  for d in range(4): # d is depth
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

Example: Deep LSTM

```
for i in range(20):
  for d in range(4): # d is depth
    with tf.device("/gpu:%d" % d):
      input = x[i] if d is 0 else m[d-1]
      m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
      mprev[d] = m[d]
      cprev[d] = c[d]
```

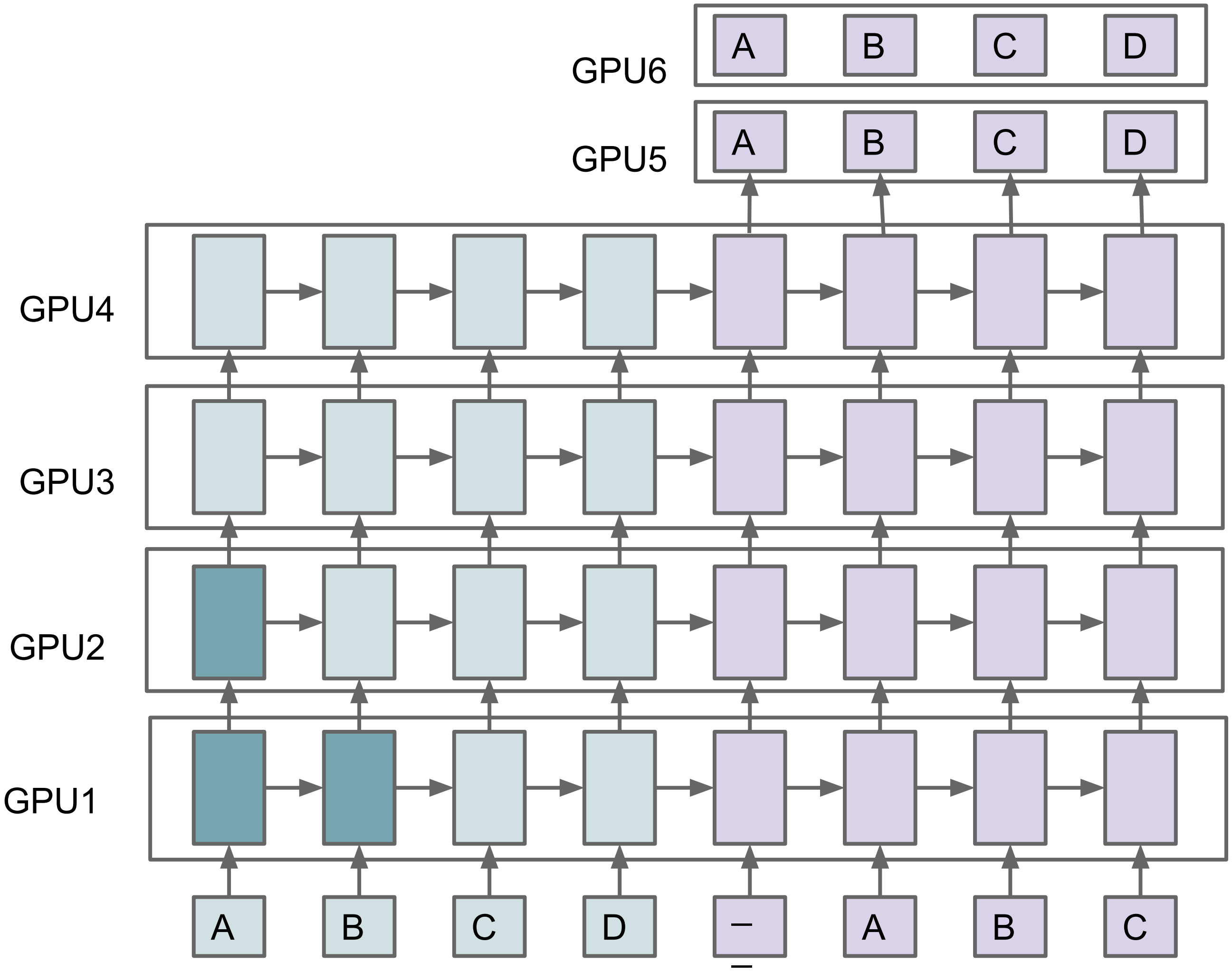


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

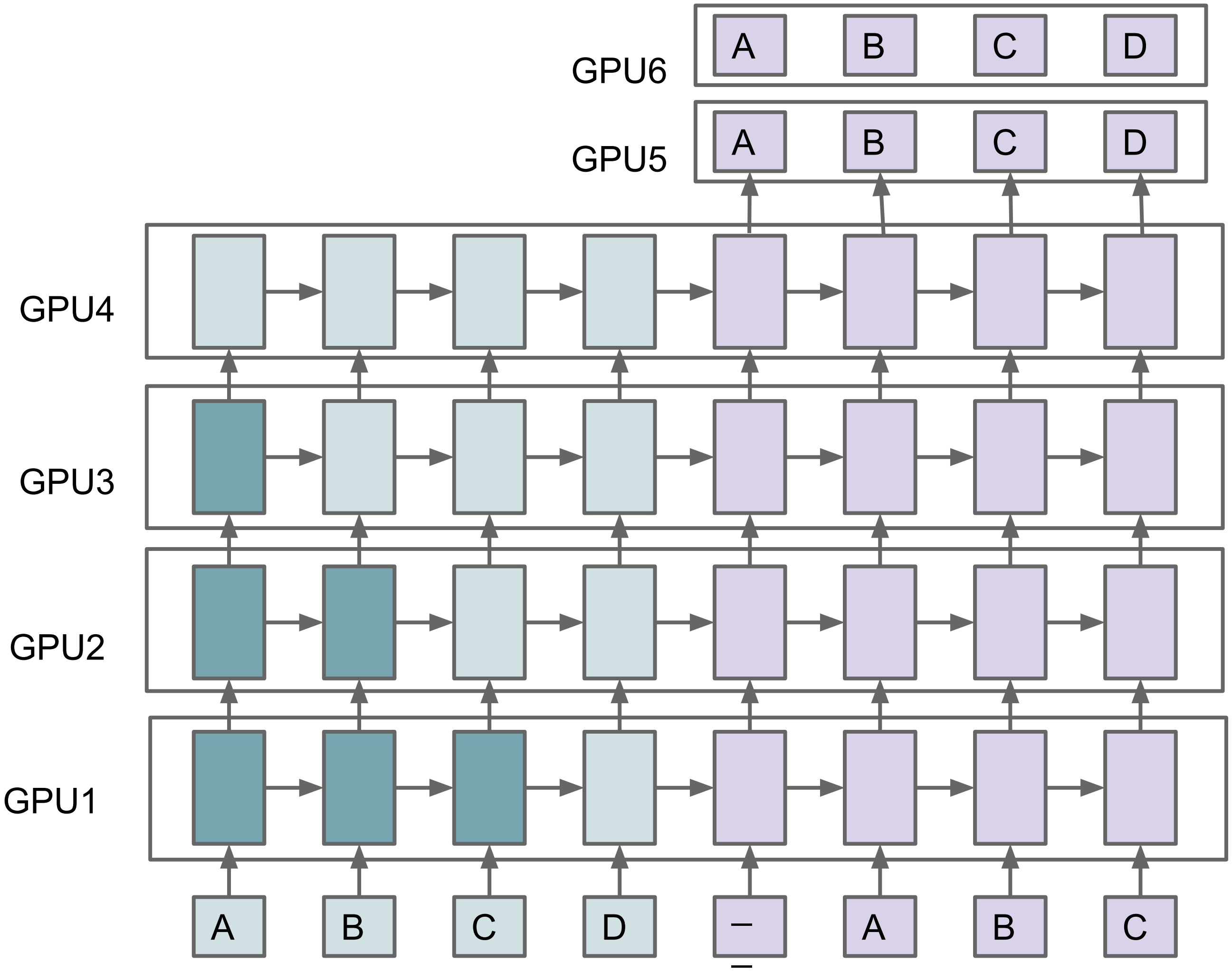


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

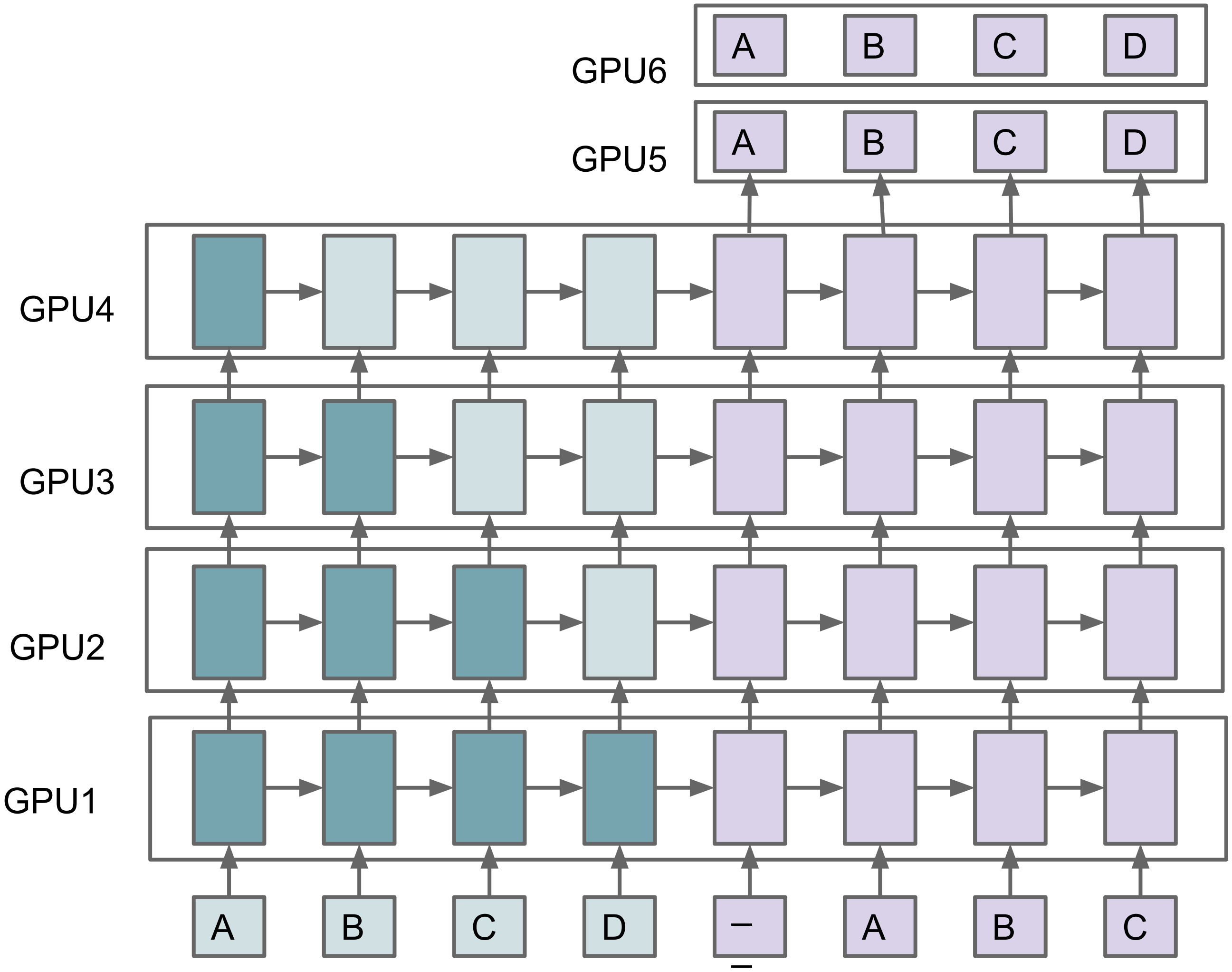


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

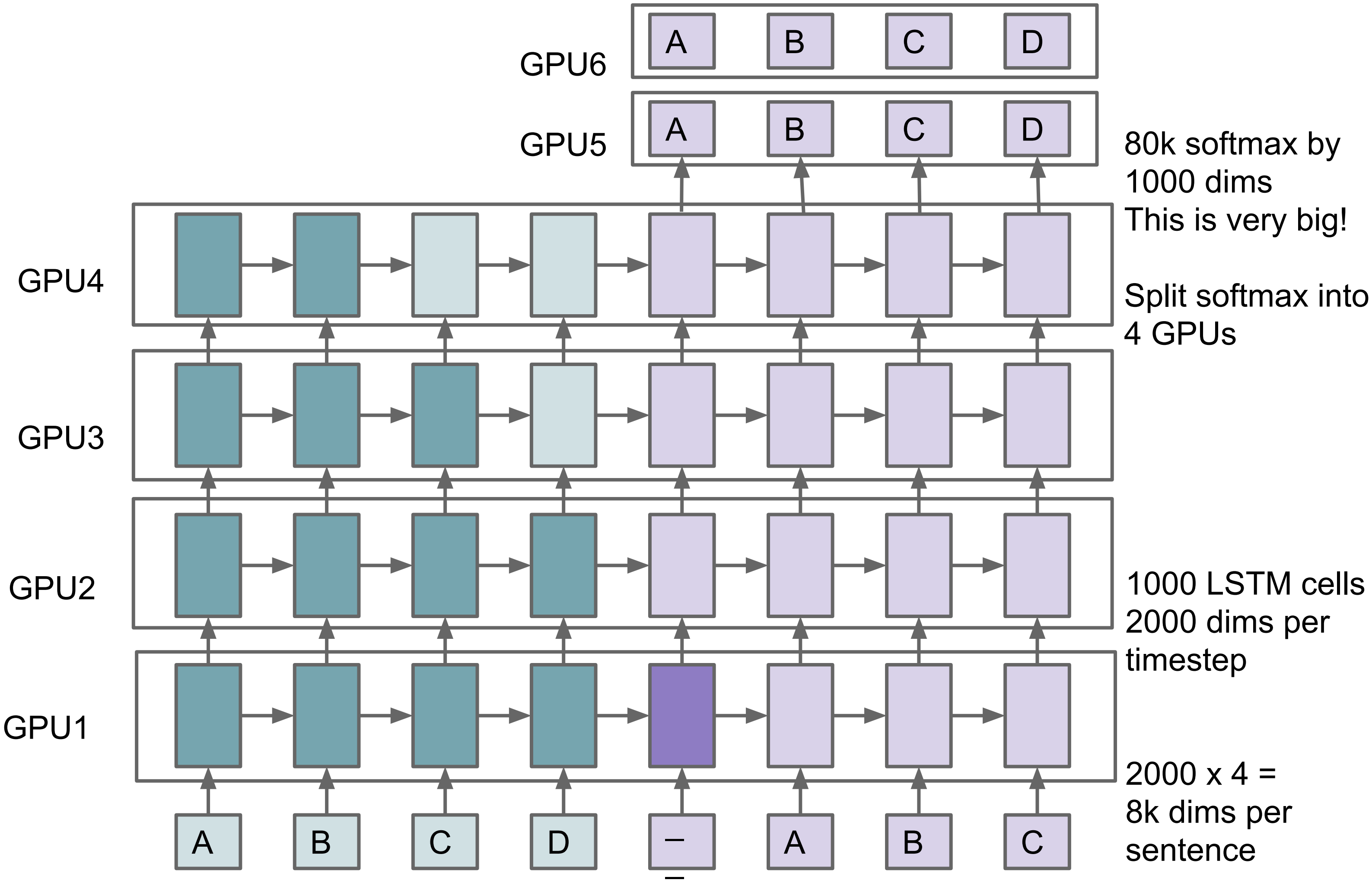


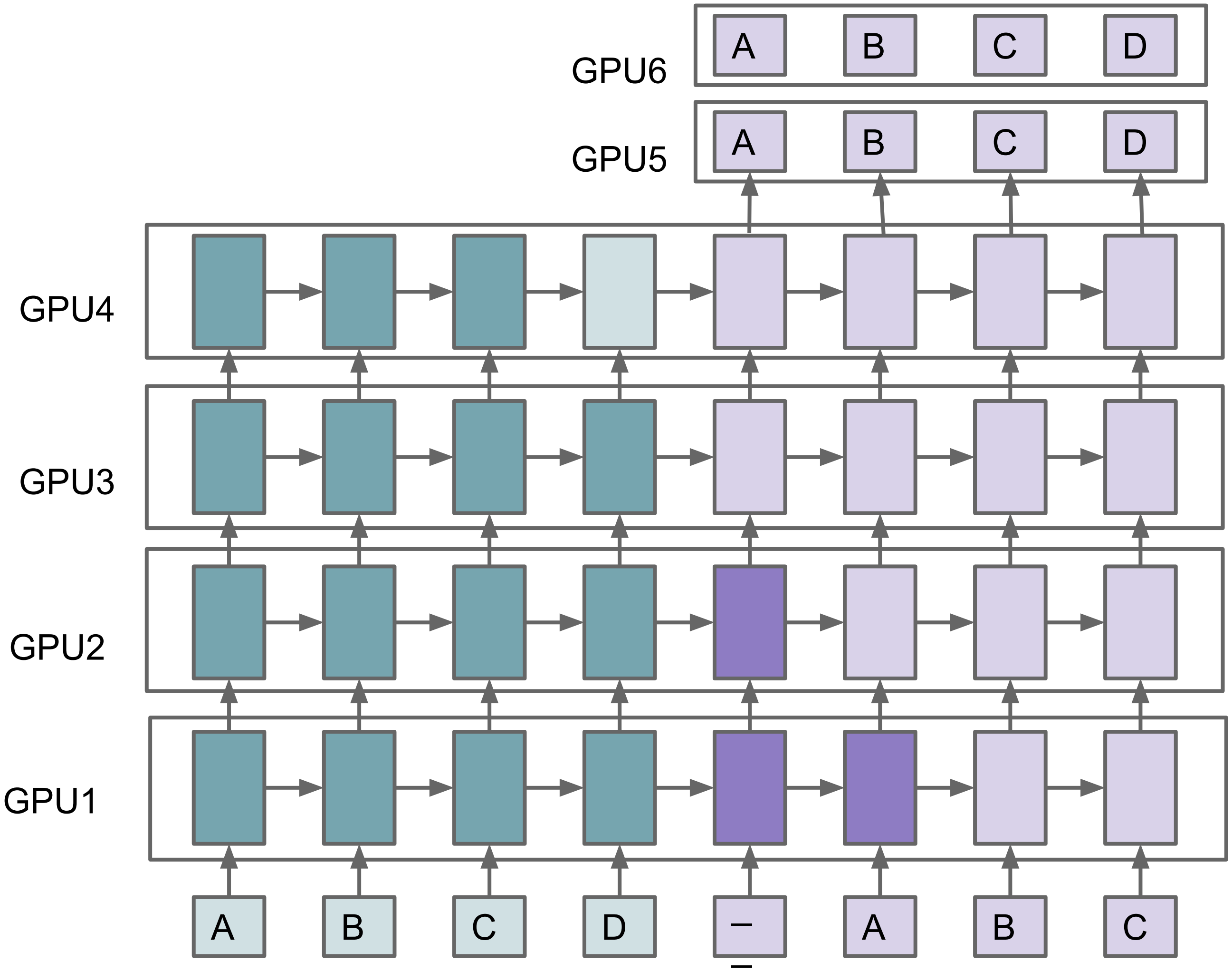
80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
Split softmax into
4 GPUs
GPU4
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
_
A
B
C

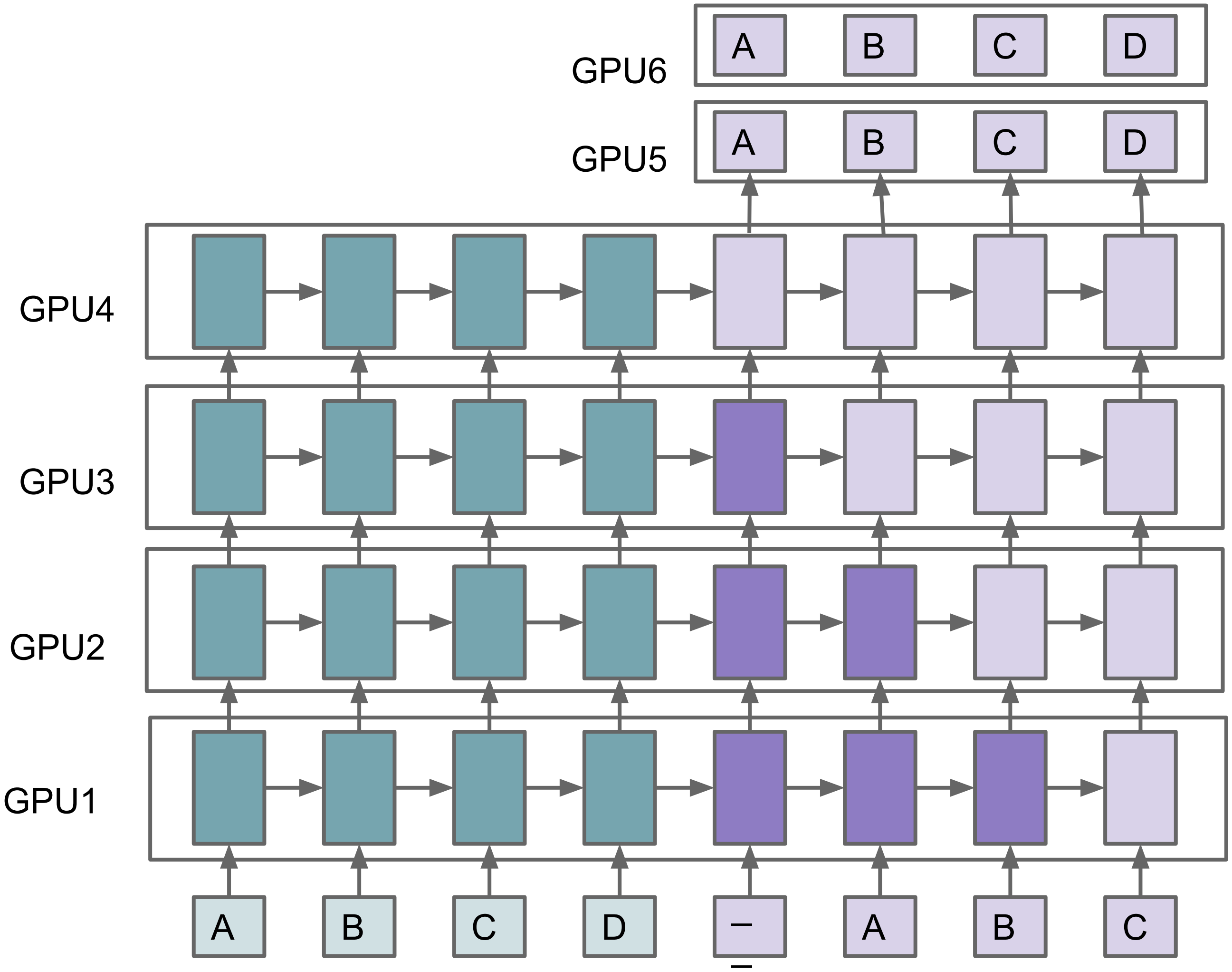


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

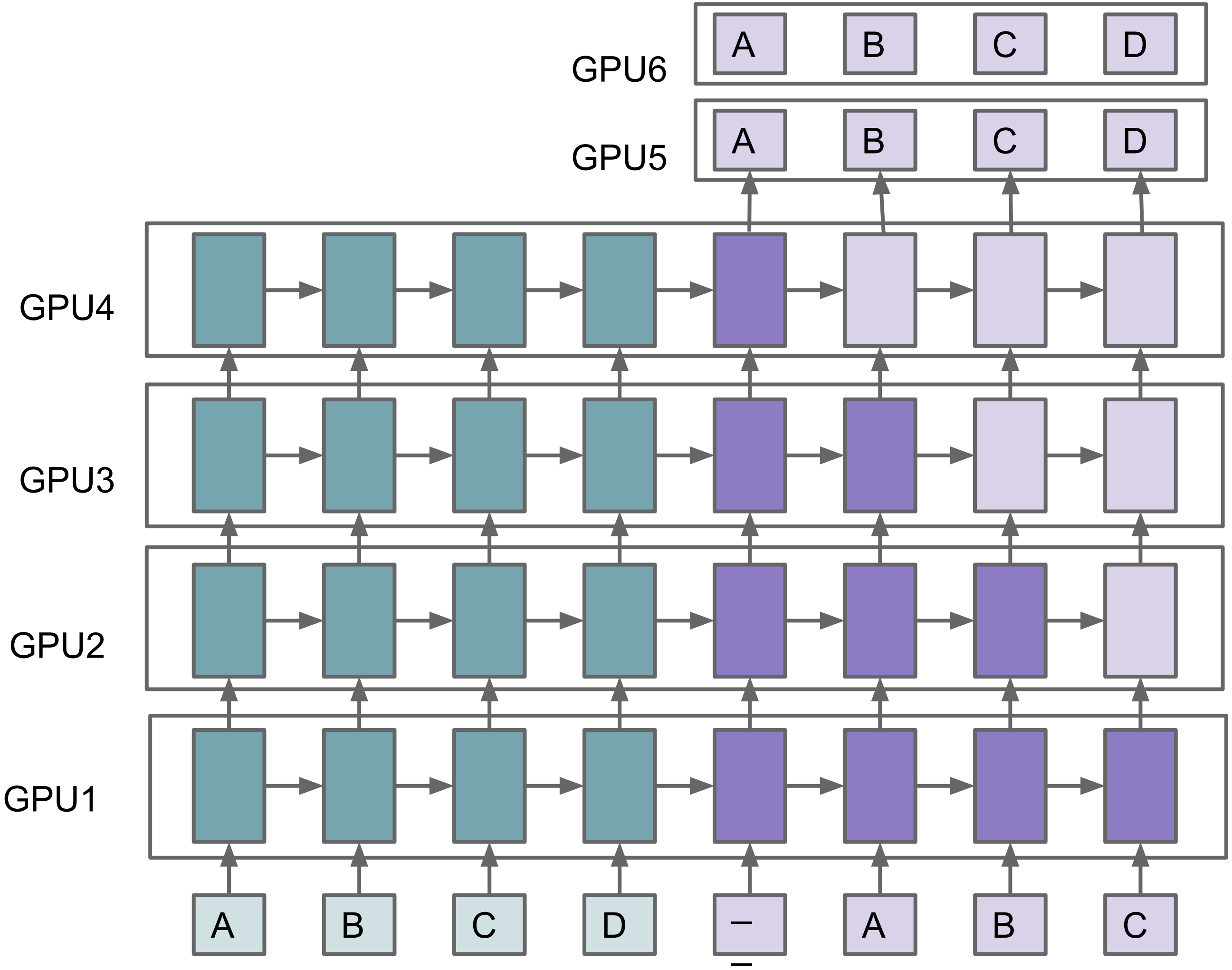


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

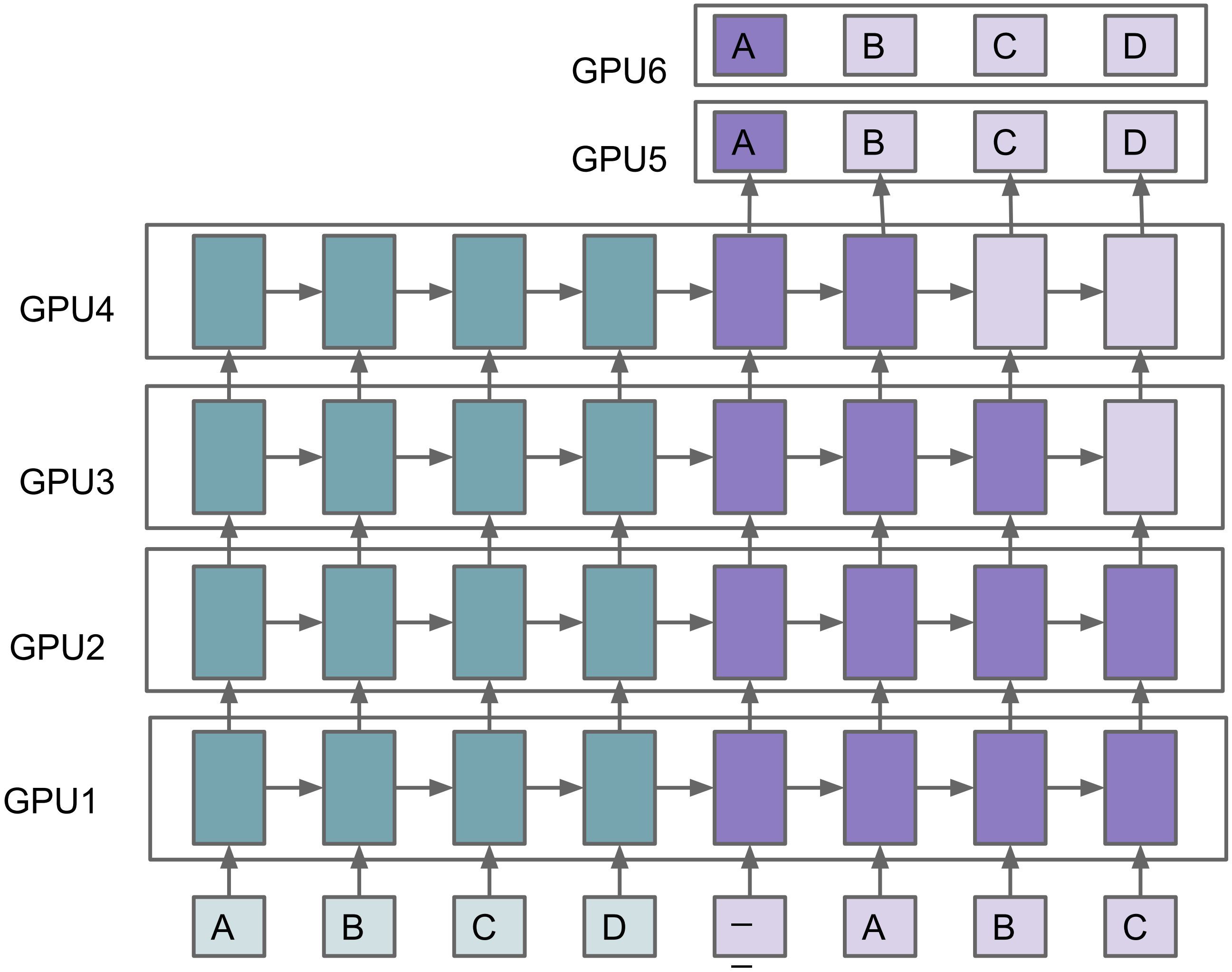


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

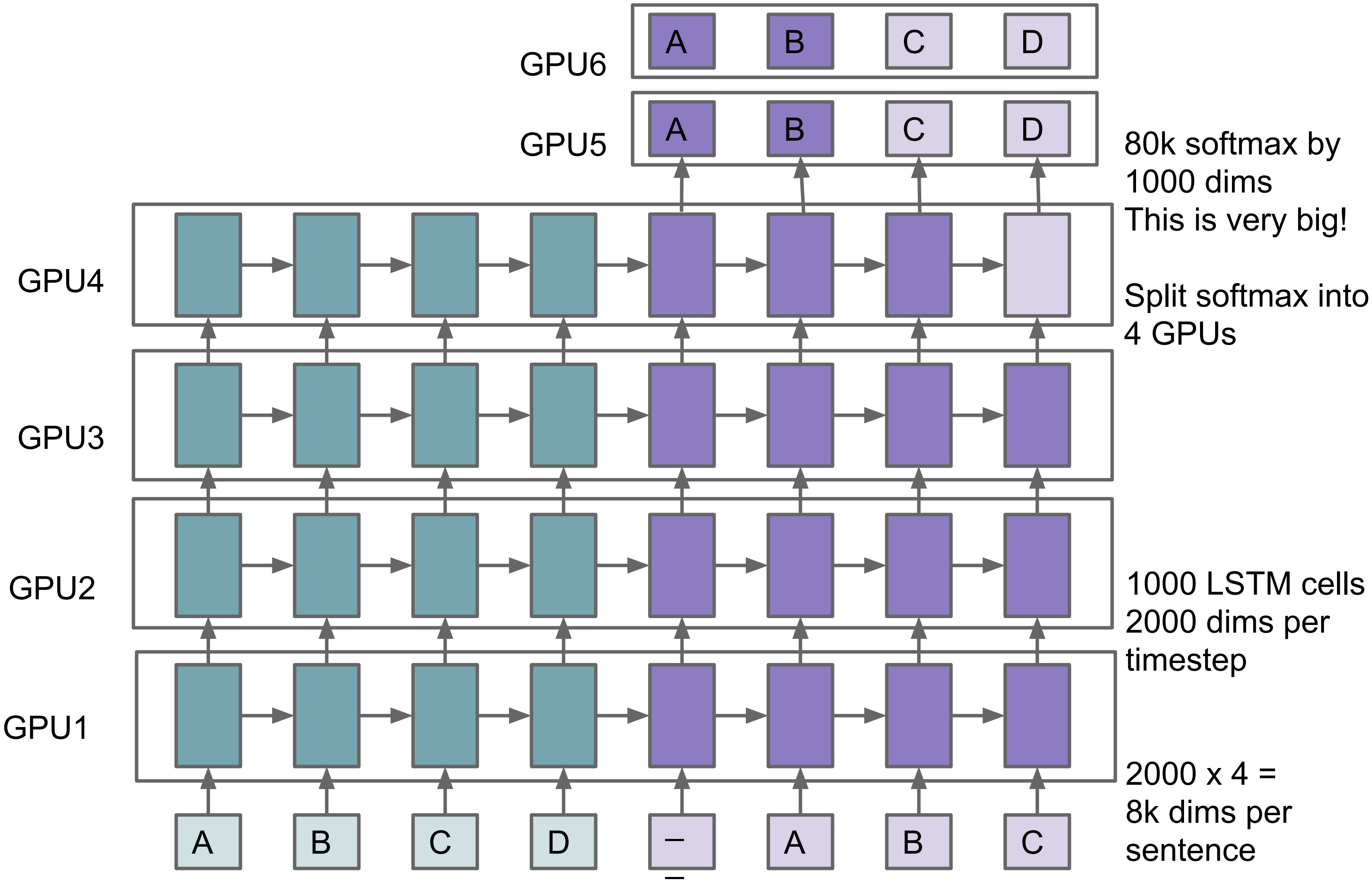


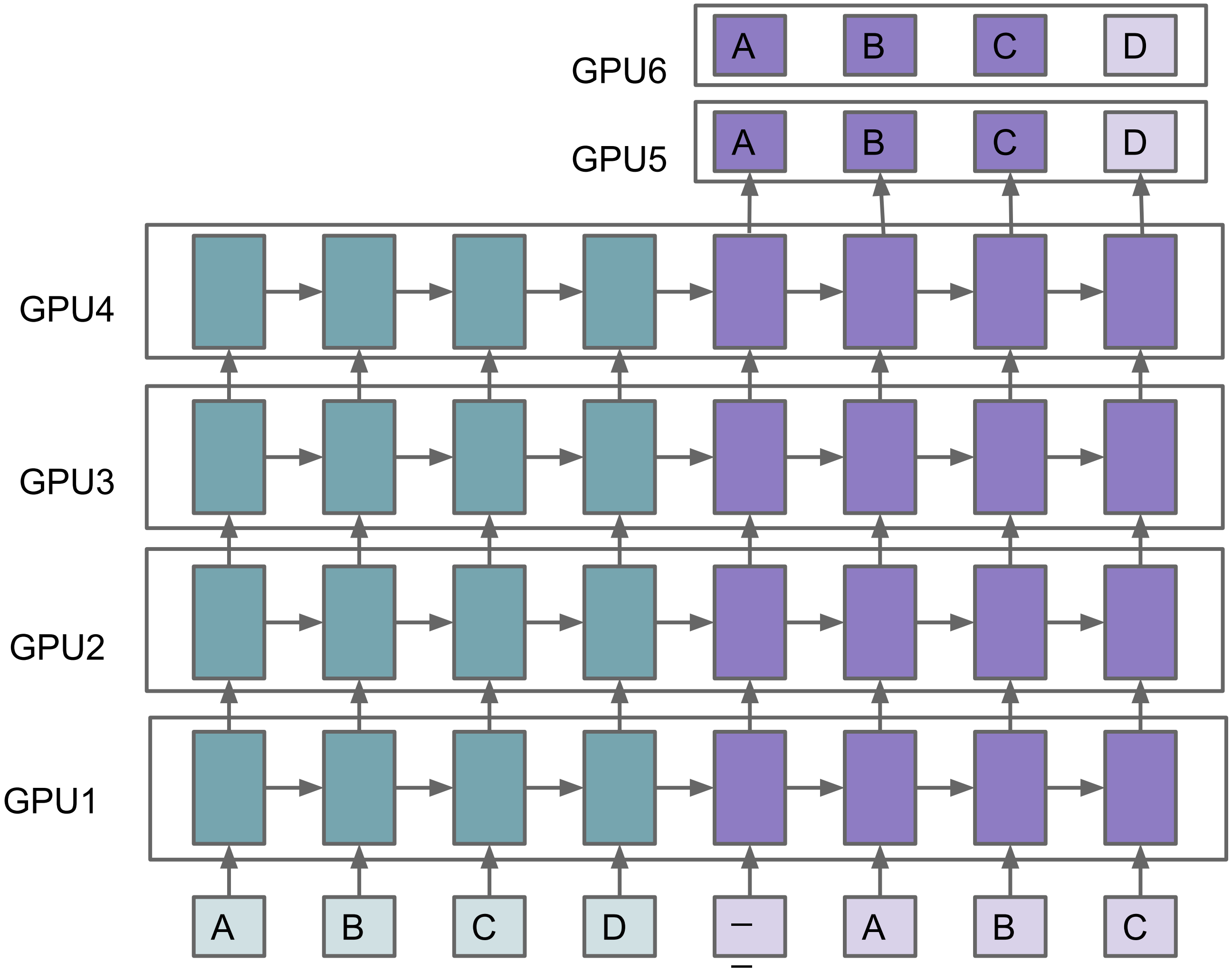
80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence


GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by
1000 dims
This is very big!
Split softmax into
4 GPUs
GPU4
GPU3
GPU2
1000 LSTM cells
2000 dims per
timestep
GPU1
2000 x 4 =
8k dims per
sentence
A
B
C
D
_
A
B
C

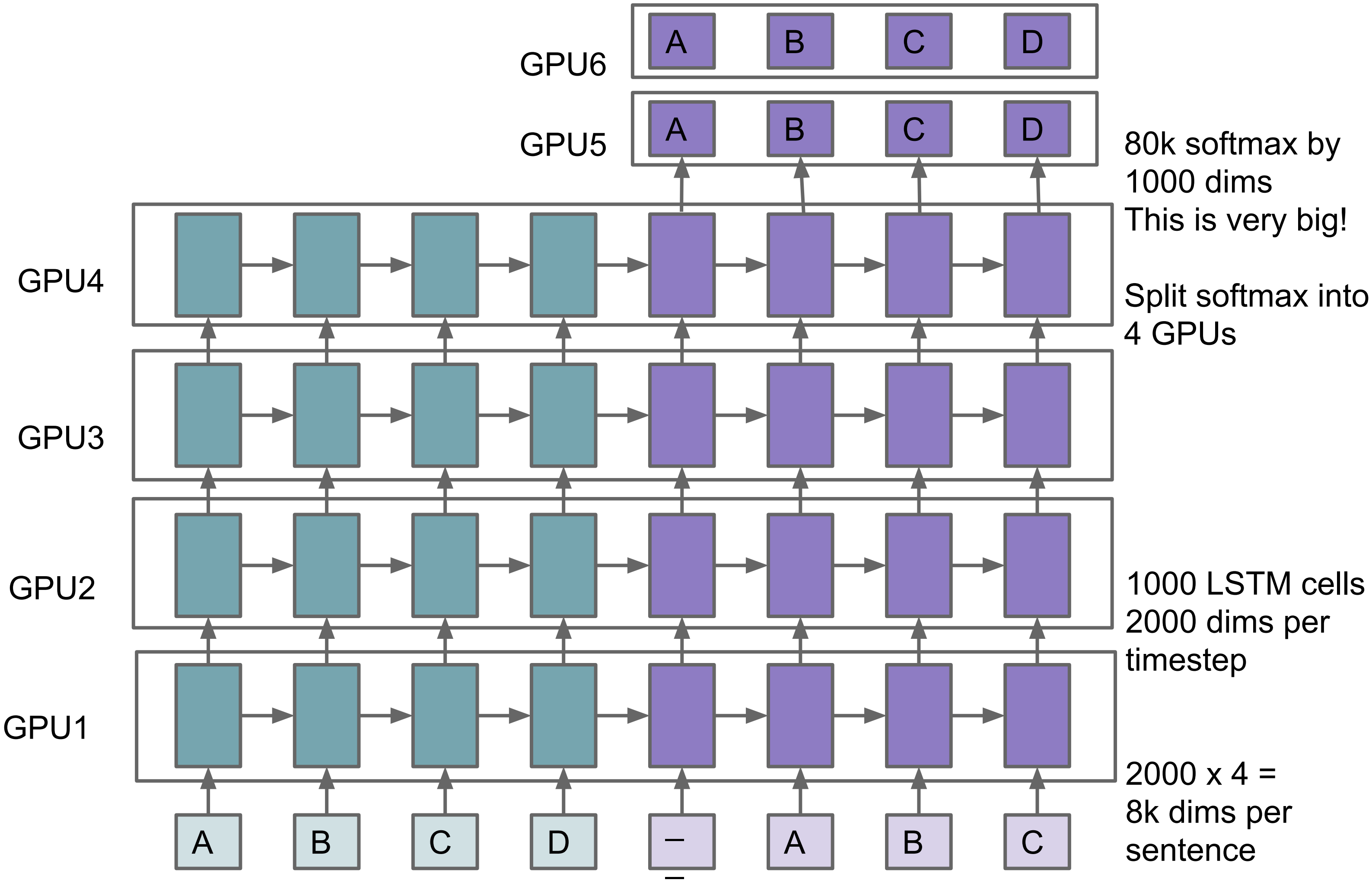


80k softmax by
1000 dims
This is very big!

Split softmax into
4 GPUs

1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence



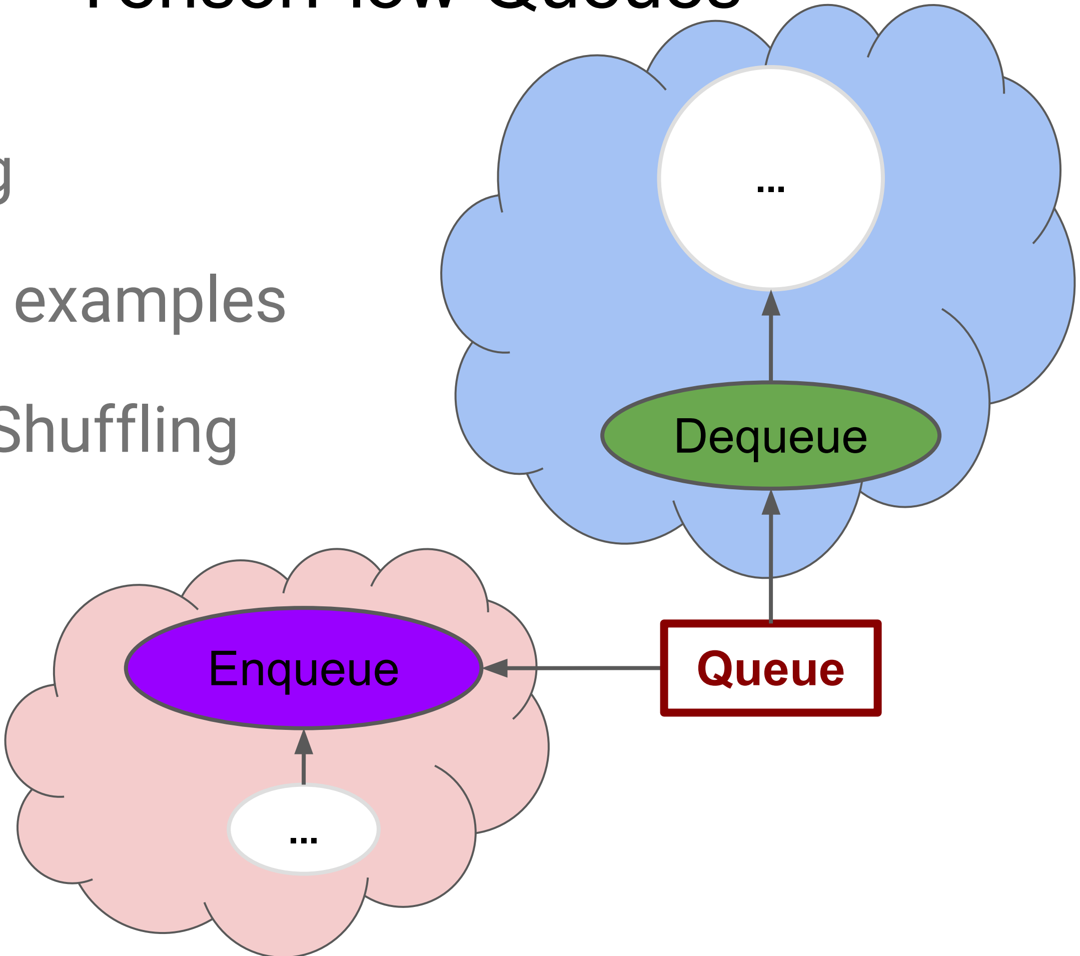
GPU6
A
B
C
D
GPU5
A
B
C
D
80k softmax by 1000 dims
This is very big!
Split softmax into 4 GPUs
GPU4
GPU3
GPU2
1000 LSTM cells
2000 dims per timestep
GPU1
2000 x 4 =
8k dims per sentence
A
B
C
D
_
A
B
C

TensorFlow Queues

Input prefetching

Grouping similar examples

Randomization/Shuffling



Example: Deep LSTMs

- Wrinkles
  - Bucket sentences by length using a queue per length
  - Dequeue when a full batch of same length has accumulated
  - N different graphs for different lengths
  - Alternative: while loop

Expressing Data Parallelism

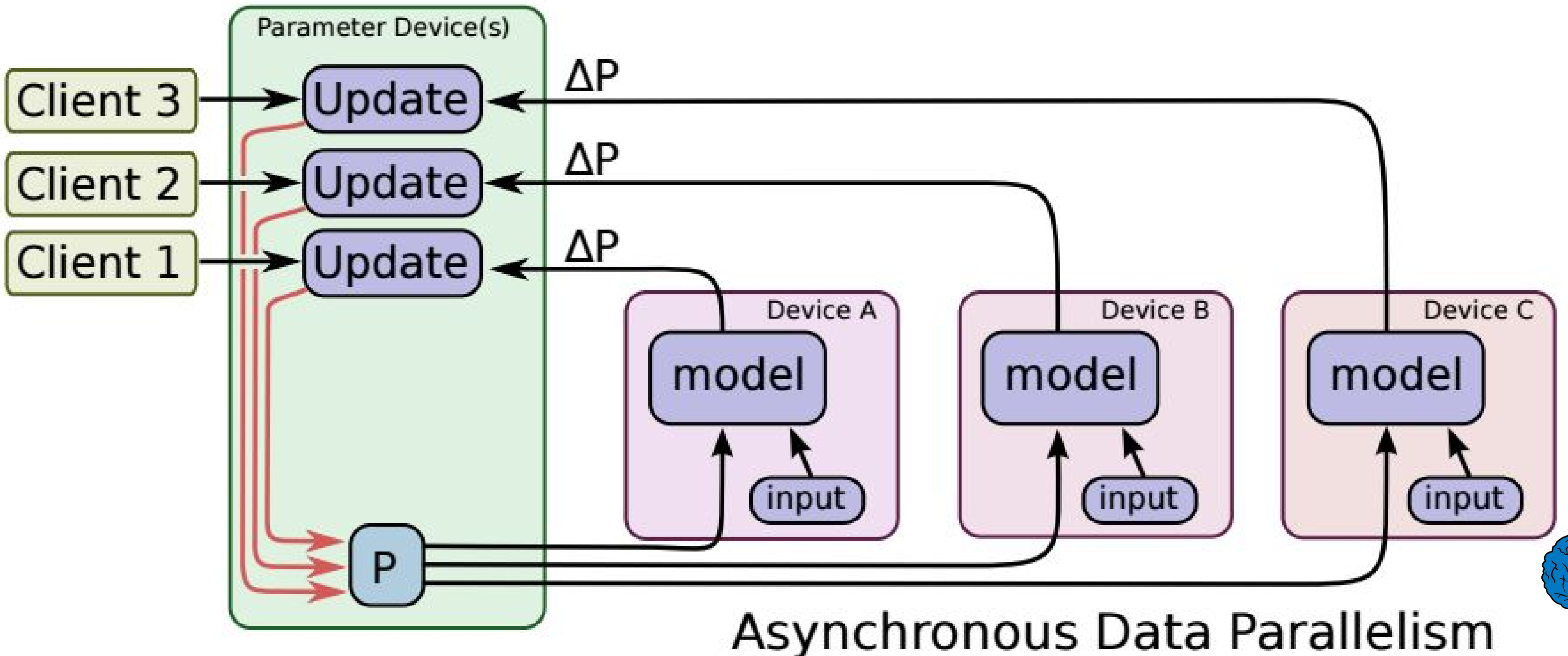
```
# We use the ReplicaDeviceSetter() device function to automatically
# assign Variables to the 'ps' jobs.
with tf.device("/cpu:0"):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # Get an initialized, and possibly recovered session.
    sess = tf.Session()

    # Train the model.
    for local_step in xrange(FLAGS.max_steps):
      _, loss, step = sess.run([model.train_op, model.loss, model.global_step])
      if local_step % 1000 == 0:
        print "step %d: %g" % (step, loss)
```

Expressing Data Parallelism

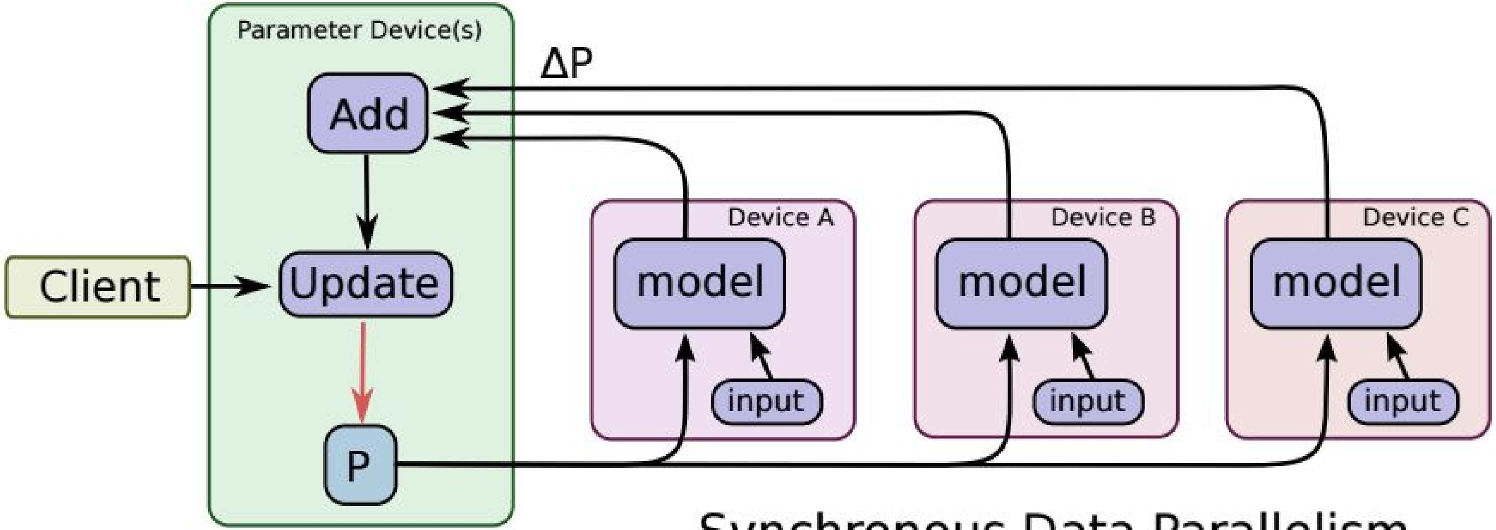
```
# We use the ReplicaDeviceSetter() device function to automatically
# assign Variables to the 'ps' jobs.
with tf.device(tf.ReplicaDeviceSetter(parameter_devices=10)):
    # Create the Mnist model.
    model = MnistModel(batch_size=16, hidden_units=200)

    # Create a Supervisor.  It will take care of initialization, summaries,
    # checkpoints, and recovery. When multiple replicas of this program are running,
    # the first one, identified by --task=0 is the 'chief' supervisor (e.g., initialization, saving)
    supervisor = tf.Supervisor(is_chief=(FLAGS.task == 0), saver=model.saver)

    # Get an initialized, and possibly recovered session.
    sess = supervisor.PrepareSession(FLAGS.master_job)

    # Train the model.
    for local_step in xrange(int32_max):
      _, loss, step = sess.run([model.train_op, model.loss, model.global_step])
      if step >= FLAGS.max_steps:
        break
      if local_step % 1000 == 0:
        print "step %d: %g" % (step, loss)
```

Asynchronous Training

- Unlike DistBelief, no separate parameter server system:
  - Parameters are now just stateful nodes in the graph



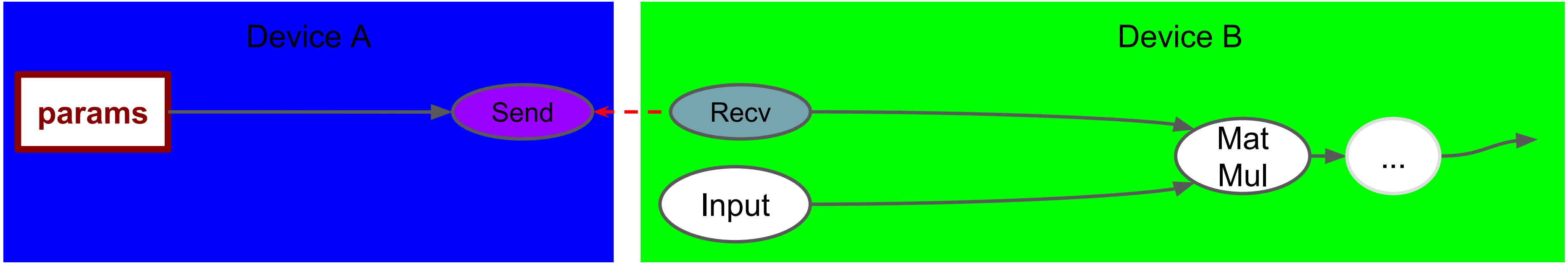
Synchronous Variant



Synchronous Data Parallelism

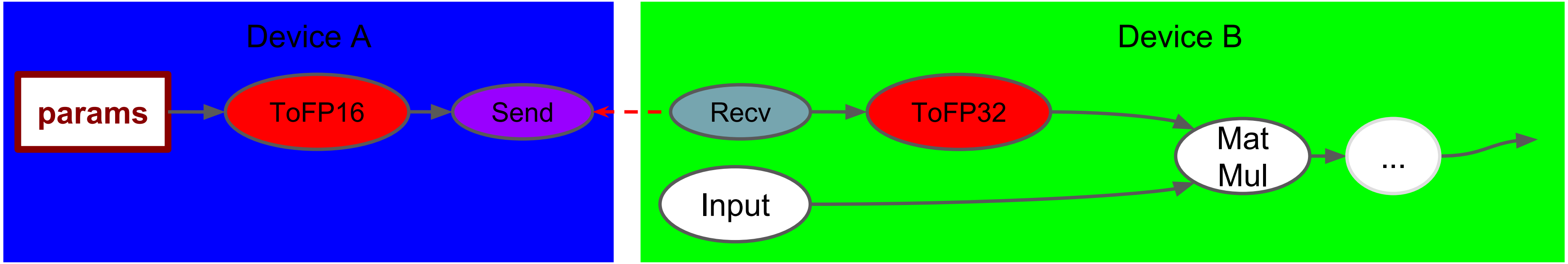
Network Optimizations

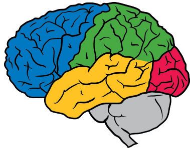
- Neural net training very tolerant of reduced precision
- e.g. drop precision to 16 bits across network



Network Optimizations

- Neural net training very tolerant of reduced precision
- e.g. drop precision to 16 bits across network





Quantization for Inference

- Need even less precision for inference
- 8-bit fixed point works well, but many ways of quantizing
- Critical for things like mobile devices
  - w/quantization, high-end smart phone can run Inception model at >6 frames per second (fps)

Open Source Status for Distributed TensorFlow

Multi GPU in single machine already in open source release

- See 4-GPU CIFAR10 training example in repository

Distributed implementation coming soon:

- GitHub tracking issue: github.com/tensorflow/tensorflow/issues/23

Concluding Remarks

- Model and Data Parallelism enable great ML work:
  - Neural Machine Translation: ~6x speedup on 8 GPUs
  - Inception / Imagenet: ~40x speedup on 50 GPUs
  - RankBrain: ~300X speedup on 500 machines
- A variety of different parallelization schemes are easy to express in TensorFlow

Concluding Remarks

- Open Sourcing of TensorFlow
  - Rapid exchange of research ideas (we hope!)
  - Easy deployment of ML systems into products
  - TensorFlow community doing interesting things!

A Few TensorFlow Community Examples

- DQN: github.com/nivwusquorum/tensorflow-deepq
- NeuralArt: github.com/woodrush/neural-art-tf
- Char RNN: github.com/sherjilozair/char-rnn-tensorflow
- Keras ported to TensorFlow: github.com/fchollet/keras
- Show and Tell: github.com/jazzsaxmafia/show_and_tell.tensorflow
- Mandarin translation: github.com/jikexueyuanwiki/tensorflow-zh

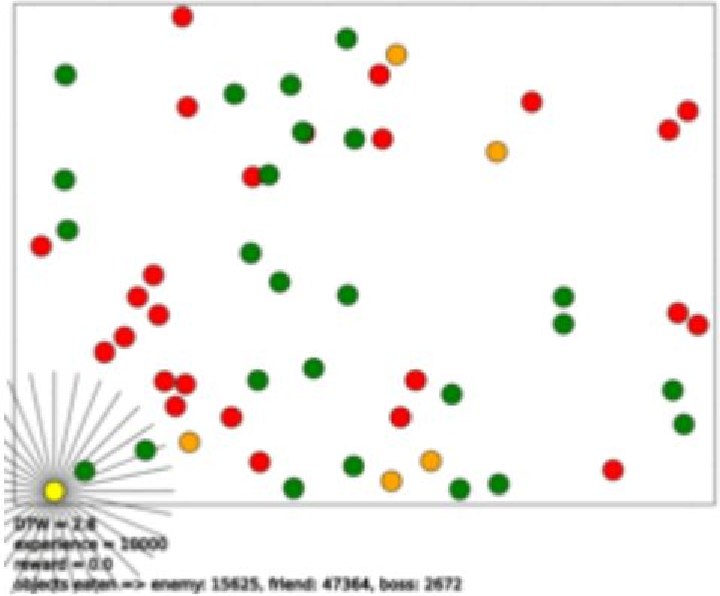
...

github.com/nivwusquorum/tensorflow-deepq

Reinforcement Learning using Tensor Flow

Quick start

Check out Karpathy game in `notebooks` folder.



The image above depicts a strategy learned by the DeepQ controller. Available actions are accelerating top, bottom, left or right. The reward signal is +1 for the green fellas, -1 for red and -5 for orange.

github.com/woodrush/neural-art-tf

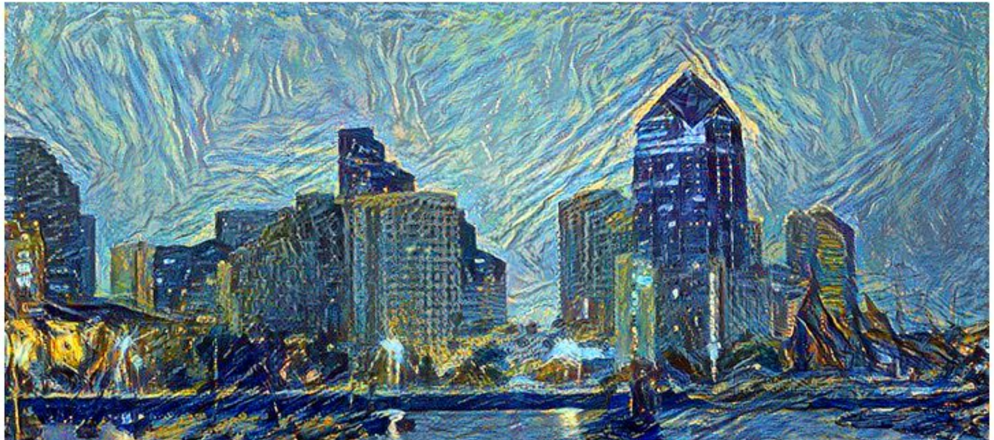
"Neural Art" in TensorFlow

An implementation of "A neural algorithm of Artistic style" in TensorFlow, for

- Introductory, hackable demos for TensorFlow, and
- Demonstrating the use of importing various Caffe cnn models (VGG and illustration2vec) in TF.

In this work, I put effort in putting the code simple as possible, for being a good introductory code to TF. For this reason, I also implemented very basic uses of TensorBoard (the visualizer). I also aimed on demonstrating the use of importing various Caffe models from *.caffemodel files into TensorFlow, especially models that seemed not to be imported by anybody yet in TF (as far as I know). Based on https://github.com/ethereon/caffe-tensorflow, I modified the importer so that it can import illustration2vec (http://illustration2vec.net/), which is another CNN available as a Caffe model. Using different CNNs yields different results, which reflects the characteristics of the model.

In the Neural Art problem setting, the weights of the CNN are fixed, and the input image into the CNN is the only "trainable" variable, making the code easy to understand (the optimized/trained image is the output image). I hope this example serves as a good introduction to TensorFlow as well as for entertainment purposes.



github.com/sherjilozair/char-rnn-tensorflow

char-rnn-tensorflow

Multi-layer Recurrent Neural Networks (LSTM, RNN) for character-level language models in Python using Tensorflow.

Inspired from Andrej Karpathy's char-rnn.

Requirements

- Tensorflow

Basic Usage

To train with default parameters on the tinyshakespeare corpus, run `python train.py`.

To sample from a checkpointed model, `python sample.py`.

github.com/fchollet/keras

Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running either on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: - **Python 2.7-3.5** with the Theano backend - **Python 2.7** with the TensorFlow backend

github.com/jazzsaxmafia/show_and_tell.tensorflow

Neural Caption Generator

- Implementation of "Show and Tell" http://arxiv.org/abs/1411.4555
  - Borrowed some code and ideas from Andrej Karpathy's NeuralTalk.
- You need flickr30k data (images and annotations)

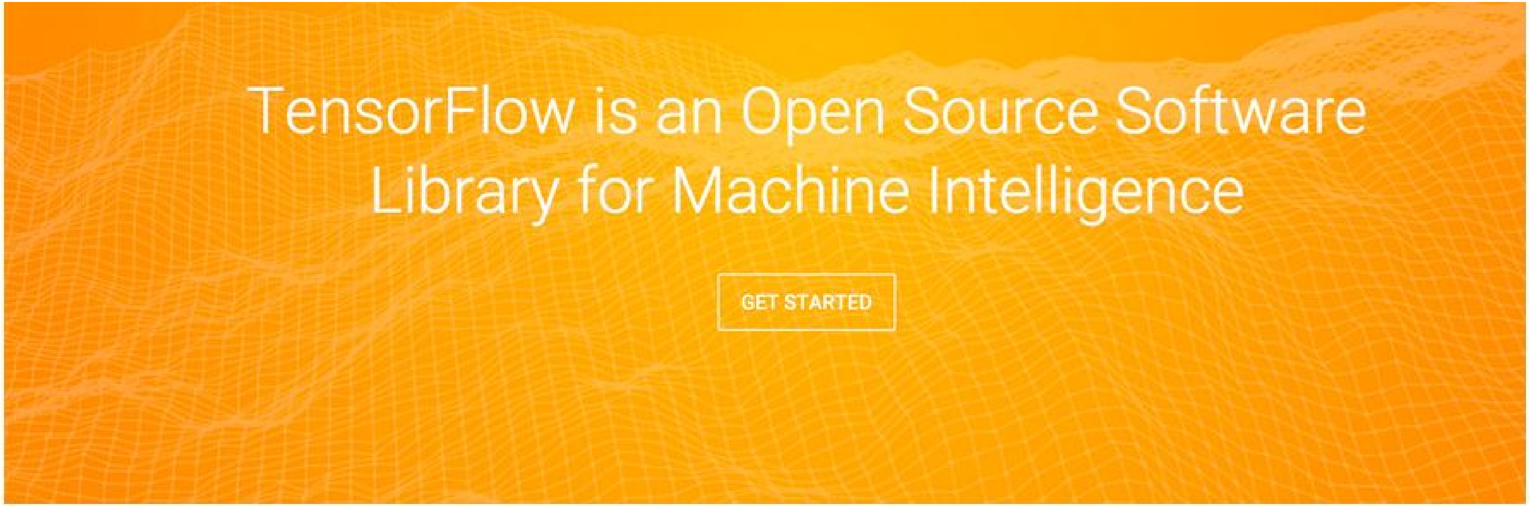
Code

- make_flickr_dataset.py : Extracting feats of flickr30k images, and save them in './data/feats.npy'
- model_tensorflow.py : TensorFlow Version
- model_theano.py : Theano Version

Usage

- Flickr30k Dataset Download
- Extract VGG Featues of Flicker30k images (make_flickr_dataset.py)
- Train: run train() in model_tensorflow.py or model_theano.py
- Test: run test() in model_tensorflow.py or model_theano.py.
  - parameters: VGG FC7 feature of test image, trained model path

github.com/jikexueyuanwiki/tensorflow-zh



你正在翻译的项目可能会比 Android 系统更加深远地影响着世界！

缘起

2015年11月9日，Google 官方在其博客上称，Google Research 宣布推出第二代机器学习系统 TensorFlow，针对先前的 DistBelief 的短板有了各方面的加强，更重要的是，它是开源的，任何人都可以用。

机器学习作为人工智能的一种类型，可以让软件根据大量的数据来对未来的情况进行阐述或预判。如今，领先的科技巨头无不在机器学习下予以极大投入。Facebook、苹果、微软，甚至国内的百度。Google 自然也在其中。「TensorFlow」是 Google

Google Brain Residency Program

New one year immersion program in deep learning research

Learn to conduct deep learning research w/experts in our team

- Fixed one-year employment with salary, benefits, ...
- Goal after one year is to have conducted several research projects
- Interesting problems, TensorFlow, and access to computational resources

Google Brain Residency Program

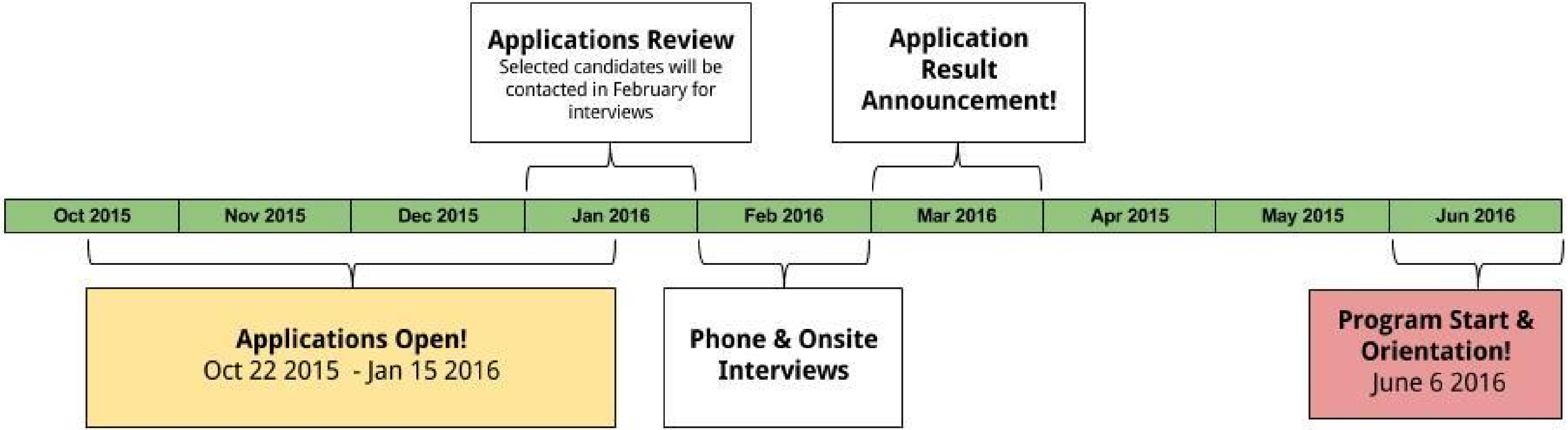
Who should apply?

- people with BSc, MSc or PhD, ideally in CS, mathematics or statistics
- completed coursework in calculus, linear algebra, and probability, or equiv.
- programming experience
- motivated, hard working, and have a strong interest in deep learning

Google Brain Residency Program

Program Application & Timeline

DEADLINE: January 15, 2016



Google
Brain Residency

Google Brain Residency Program

For more information:

g.co/brainresidency

Contact us:

brain-residency@google.com