An introduction to Tensorflow

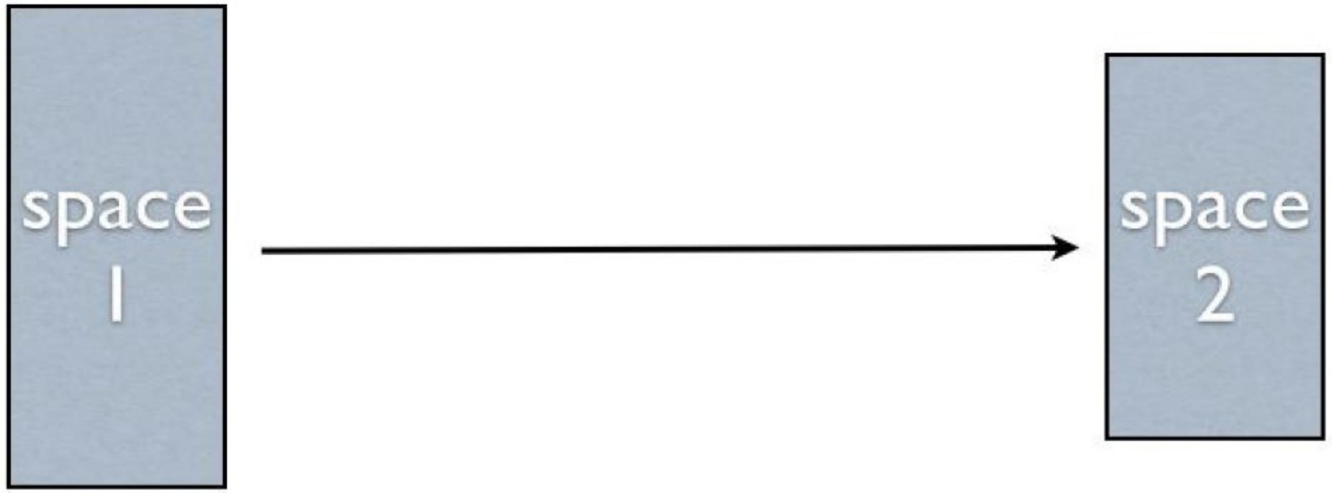
Jinming Hu & Simei He

2021.4.24

An intro to DL

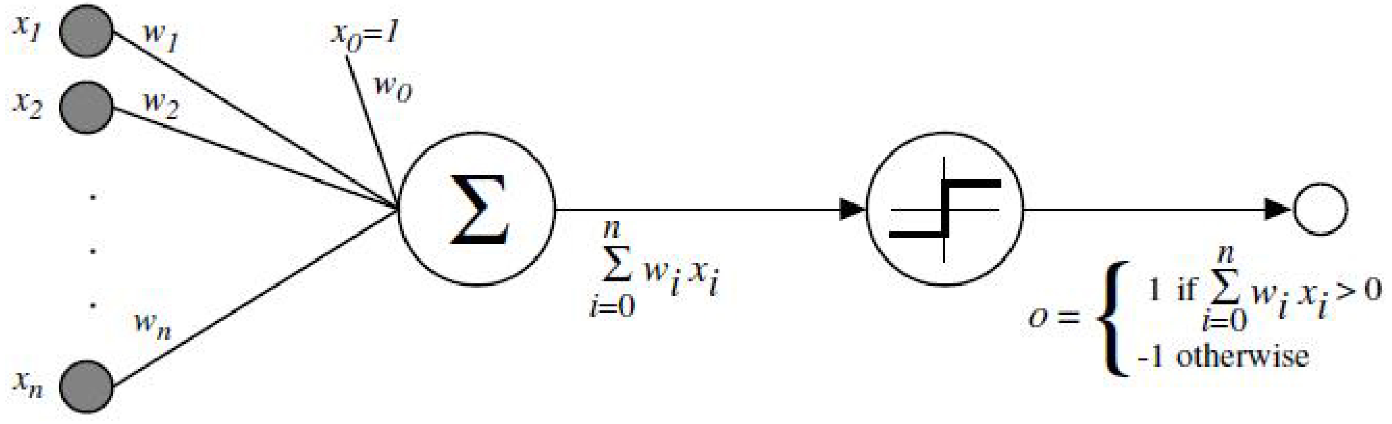
What is machine learning?

- Roughly speaking, learn a mapping function from data
- inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$
    - e.g image classification -> map images to labels



What is deep learning/neural network?

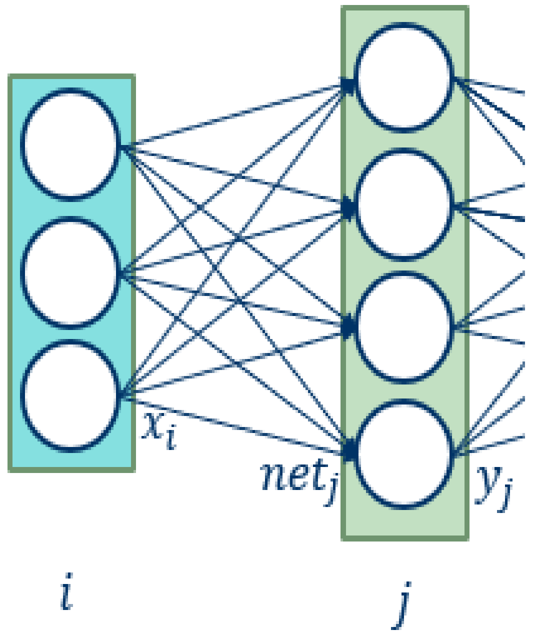
- Consist of many **layers** of **neurons**
- For neurons, $f(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{w}^T \boldsymbol{x} + b)$



x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

What is a layer of neural network?

- Multiple neurons together, with the same input:

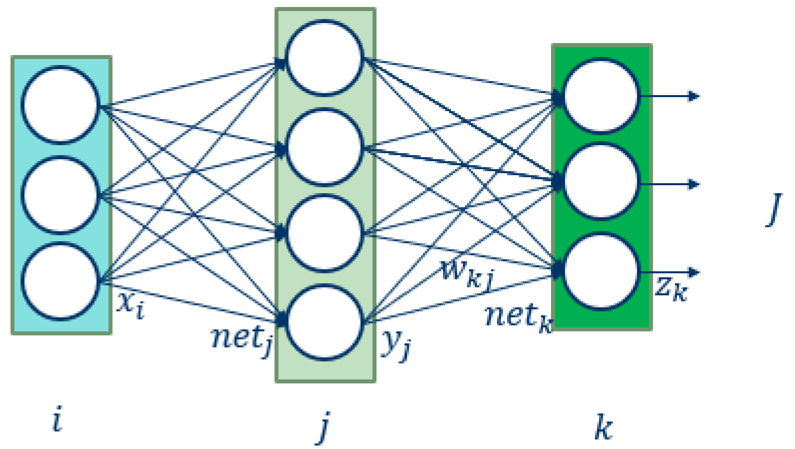


$$f(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

$$\boldsymbol{y} = F(\boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

What is neural network?

- Multiple layers stacked together



$$\boldsymbol{y} = F(\boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{z} = F(\boldsymbol{y}) = \boldsymbol{G}(\boldsymbol{W}^T\boldsymbol{y} + \boldsymbol{b})$$

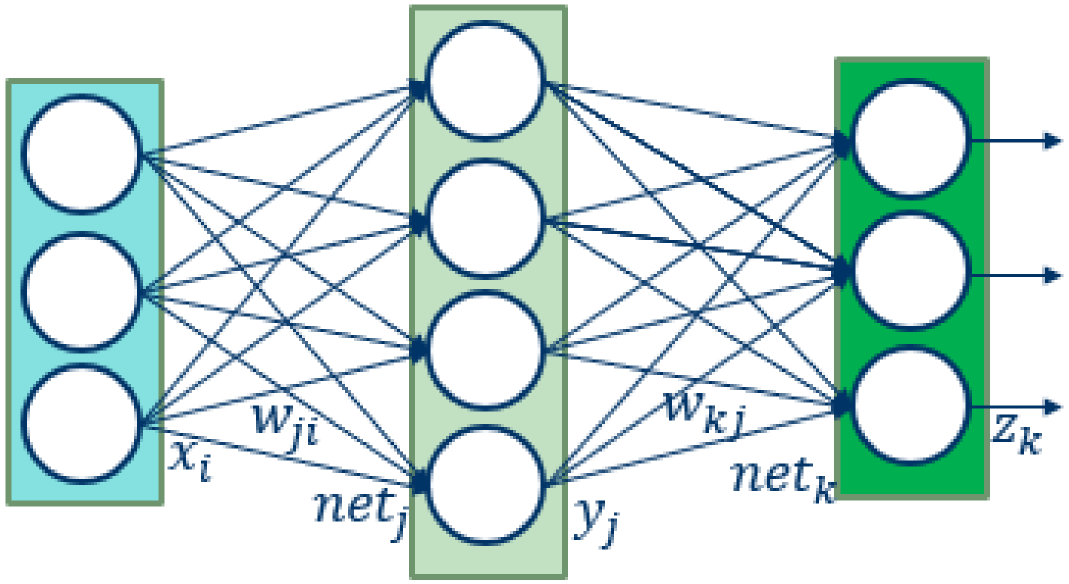
How to get the output with an existing NN?

- This is called predicting/testing.
- Just do some matrix multiplication:

$$\boldsymbol{y} = F(\boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

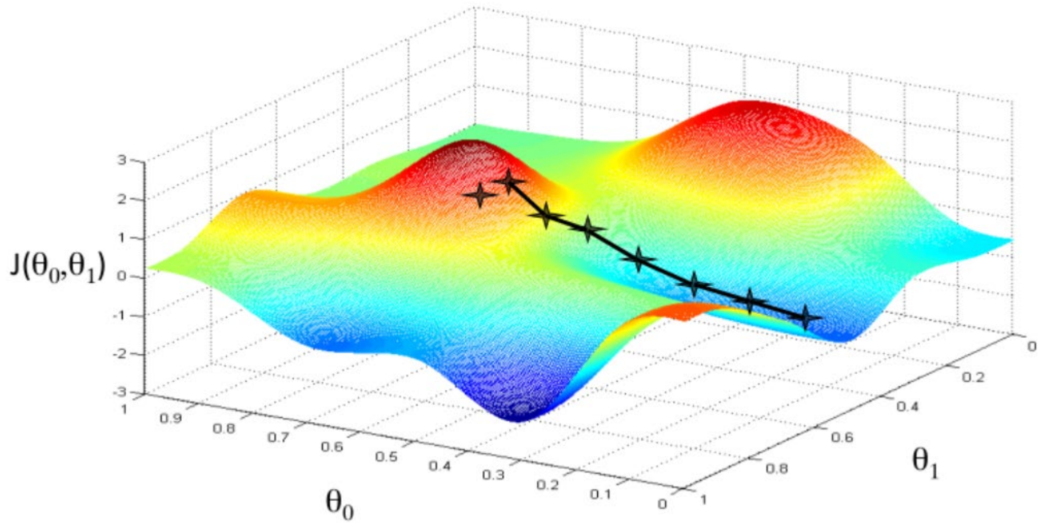
$$\boldsymbol{z} = F(\boldsymbol{y}) = \boldsymbol{G}(\boldsymbol{W}^T\boldsymbol{y} + \boldsymbol{b})$$

This is called feedforward



How are the parameters learned?

- This is called training
- Get some "ground truth" labeled data, a set of $(\boldsymbol{x}, \boldsymbol{y})$ i.e. training data
- Feedforward: $\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{x})$, calculate loss: $\boldsymbol{L}(\boldsymbol{y}', \boldsymbol{y})$
- Gradient Descent $\boldsymbol{W} = \boldsymbol{W} - \lambda \frac{\partial L}{\partial \boldsymbol{W}}$



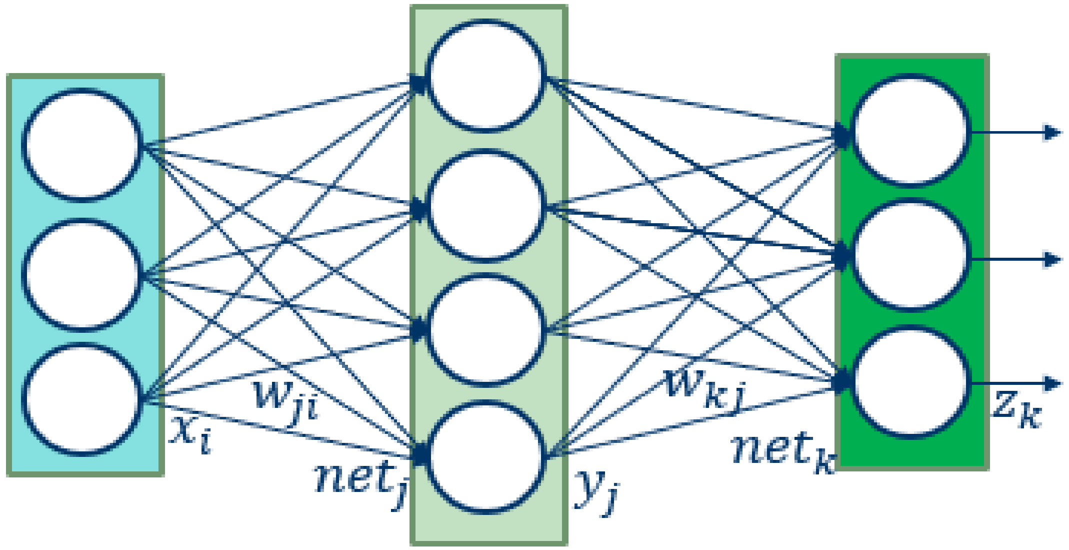
How are the parameters learned?

- **Backward:** SGD $\boldsymbol{W} = \boldsymbol{W} - \lambda \frac{\partial L}{\partial \boldsymbol{W}}$
- How to get gradient?
- Backpropagation.

$$\frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}}$$

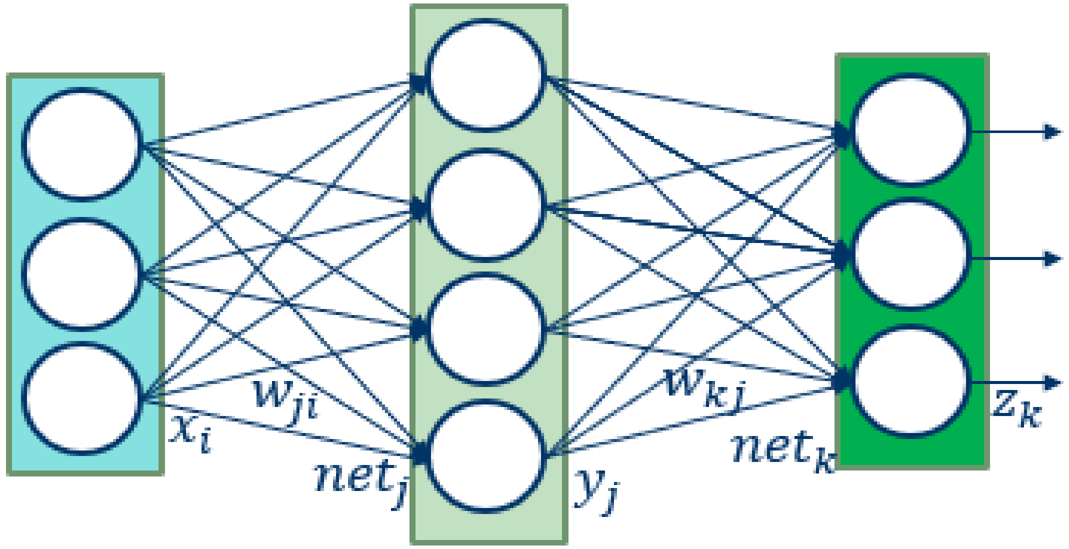
$$= (z_k - t_k) \cdot f'(net_k) \cdot y_j$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{net}} \cdot \frac{\partial \boldsymbol{net}}{\partial \boldsymbol{W}}$$



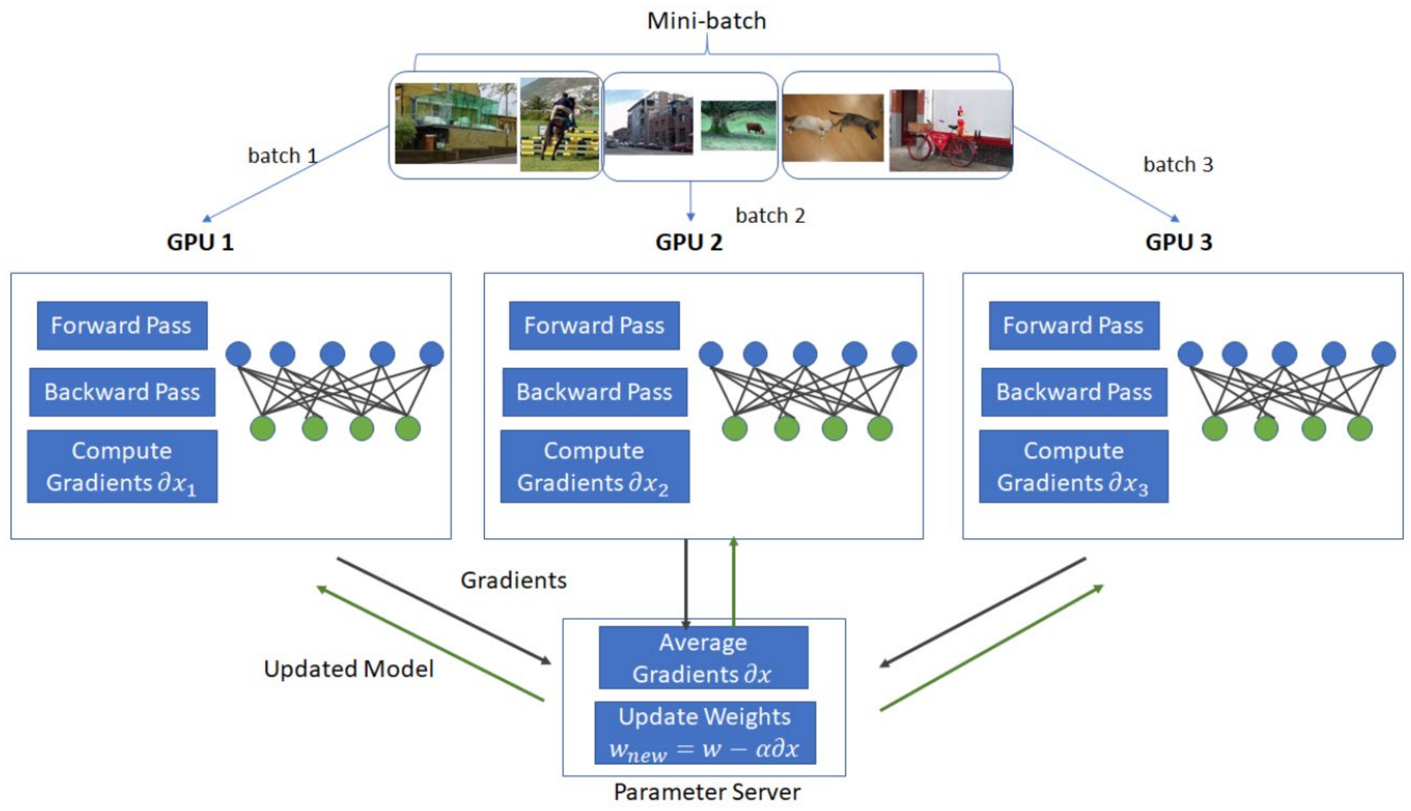
How are the parameters learned?

- For each **step/iteration**, we sample a **batch** of data from training data(usually sth like 64, 128, 256…)
- Feedforward + backward for this batch, get gradient for each neuron
- Apply some kinds of gradient descent for each neurons:
  - SGD: $\boldsymbol{W} = \boldsymbol{W} - \lambda \frac{\partial L}{\partial \boldsymbol{W}}$
  - Also some other methods



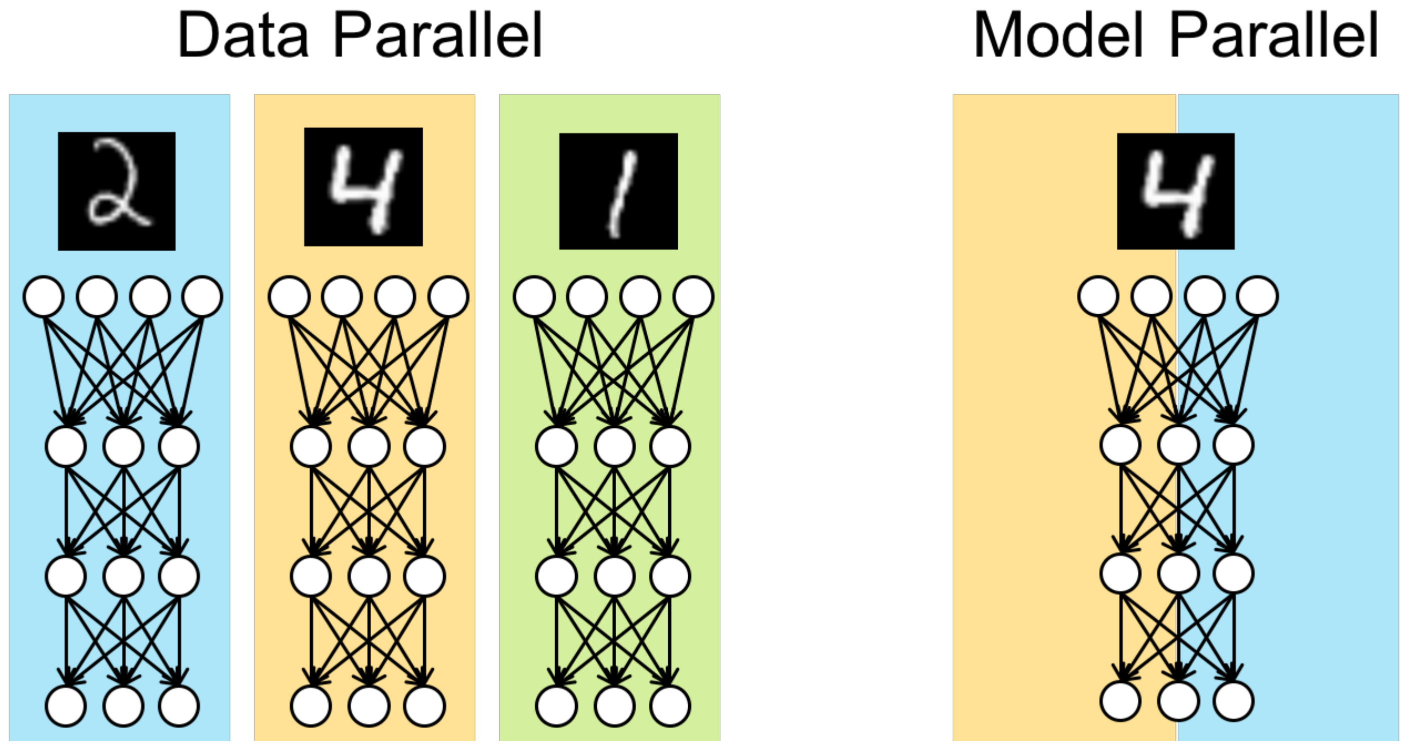
How to parallelize the computation?

- Data Parallel: computation for **every sample** in **one batch** can be parallelized


Mini-batch
batch 1
batch 2
batch 3
GPU 1
GPU 2
GPU 3
Forward Pass
Backward Pass
Compute Gradients ∂x_1
Compute Gradients ∂x_2
Compute Gradients ∂x_3
Gradients
Updated Model
Average Gradients ∂x
Update Weights $w_{new} = w - \alpha \partial x$
Parameter Server

How to parallelize the computation?

- Model parallel: computation for different parts of one neural network can be parallelized



So what do we want for a DL system?

- A lot of matrix computation, so we want for efficient matrix computation.
  - Thus run it efficiently on GPU
- Efficient to represent the computation for NN.
  - Use computational graph!
  - Automatically compute gradients.
- Parallelize computation easily.
  - Enable two types of parallelism efficiently
- Flexible to add new types of layers/optimization methods.
  - Can develop new layers/optimizations with basic operations
  - Use a handy wrapper language: usually python
- Once trained, deploy everywhere.
- ……

System Part

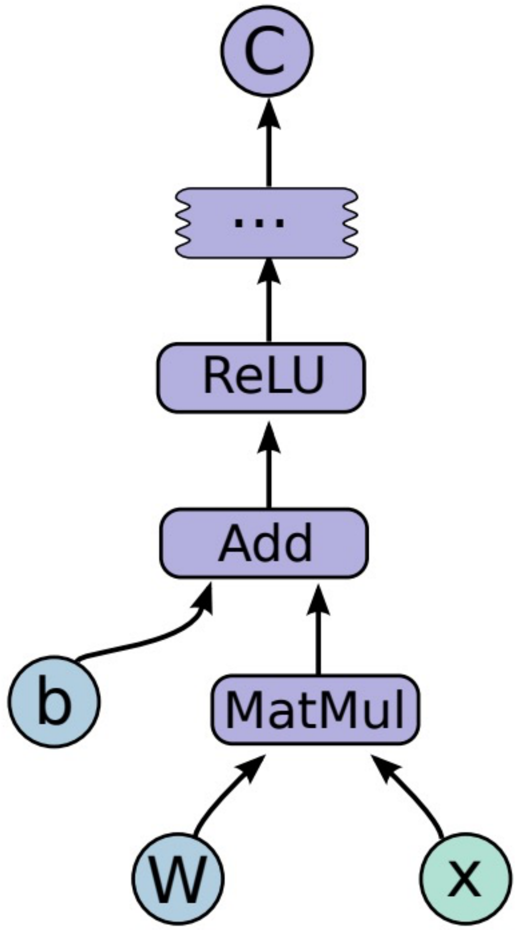
Design Goals

- To unify large-scale and small-scale machine learning abstraction
- To allow expressive paralleling progressing

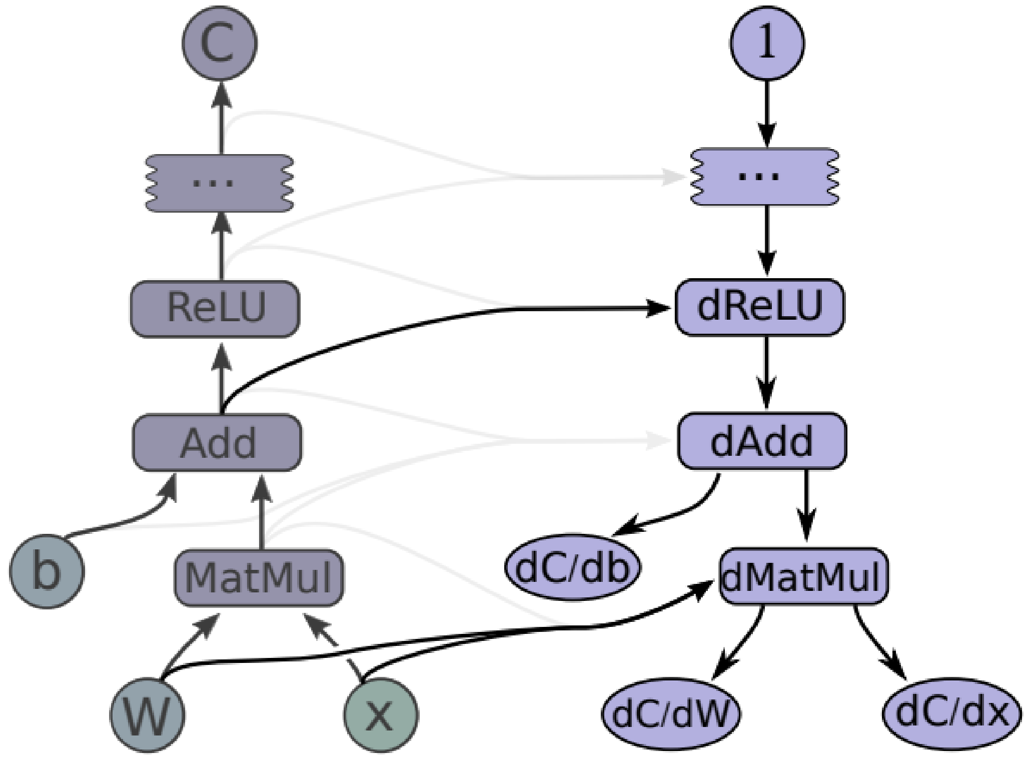
Programming Models

- Graph: stateful dataflow computation
- Node: operation
- Tensor: values that flow through normal

edges



Automatic Gradient Computation

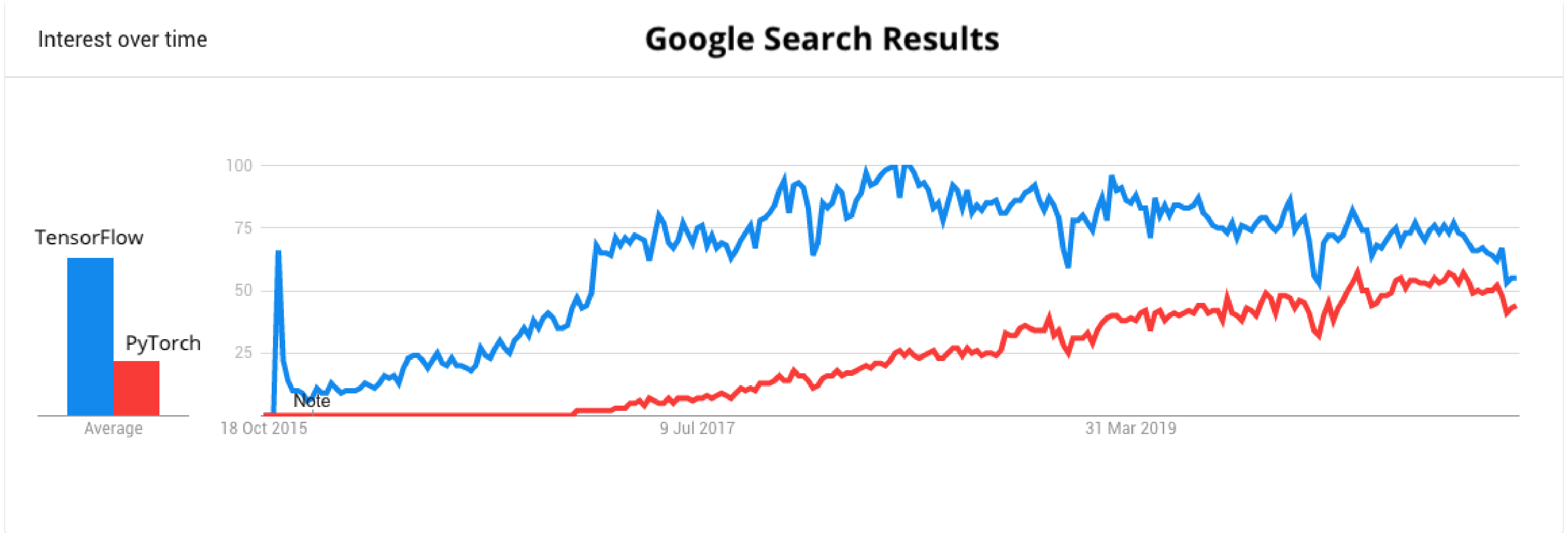


Graph Execution (TF1)

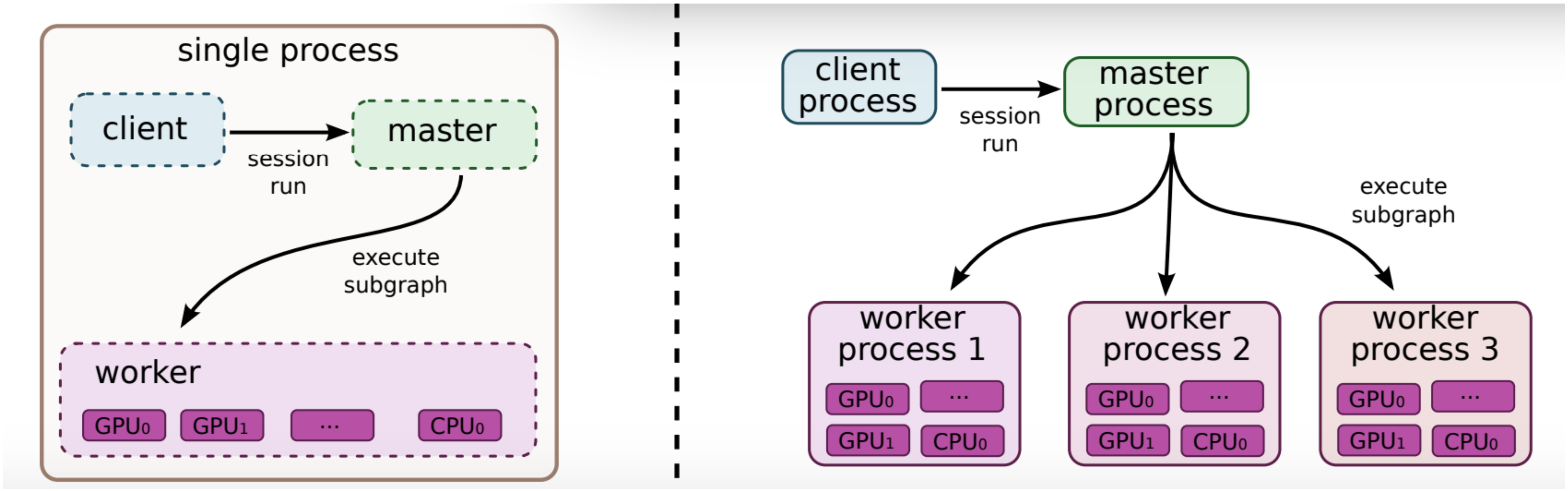
- Fast and efficient;
- Runs in parallel
- Easy to optimize
- With GPU & TPU acceleration capability

Eager Execution (TF2)

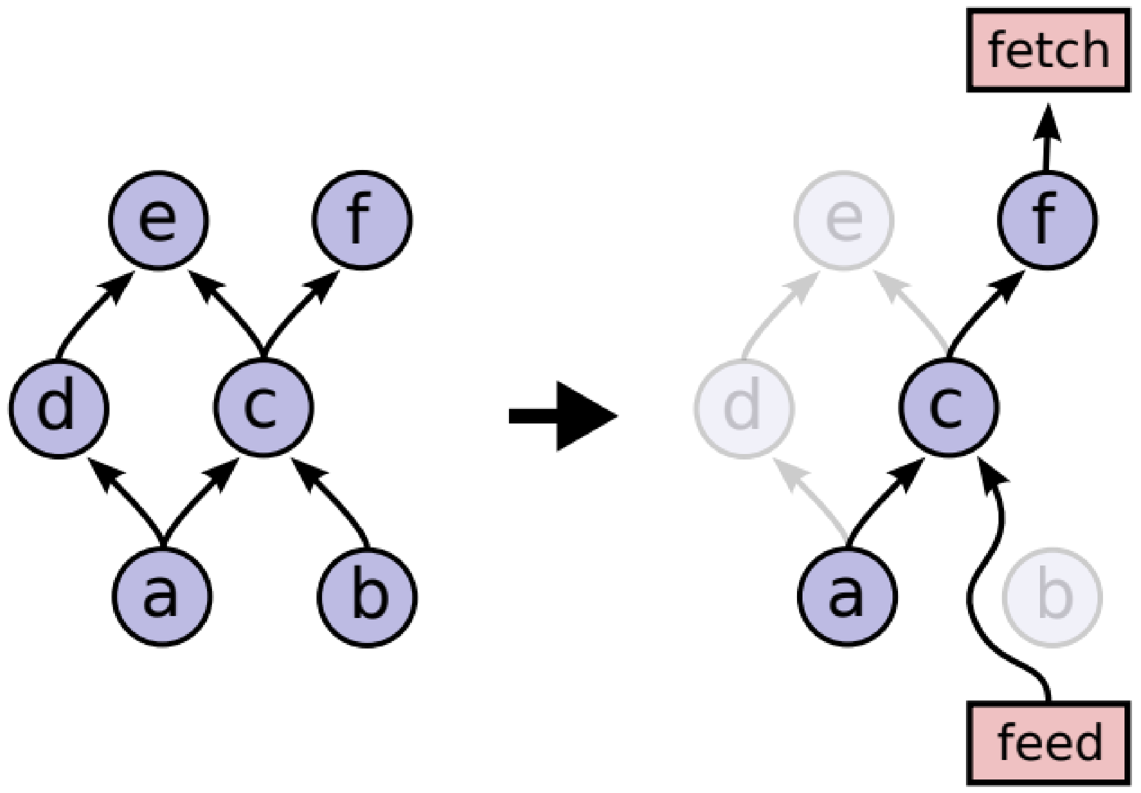
- PyTorch - dynamic computation graphs



Client, Master, Worker Layered Cake

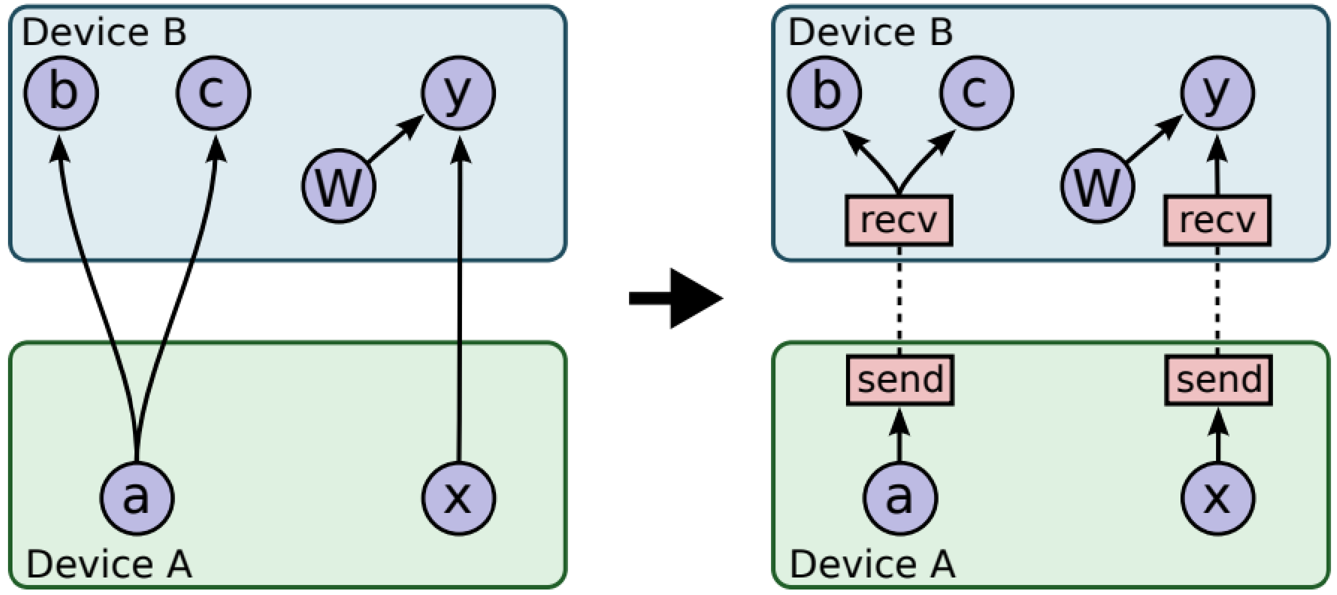


Partial Execution



Multi-Device Executions

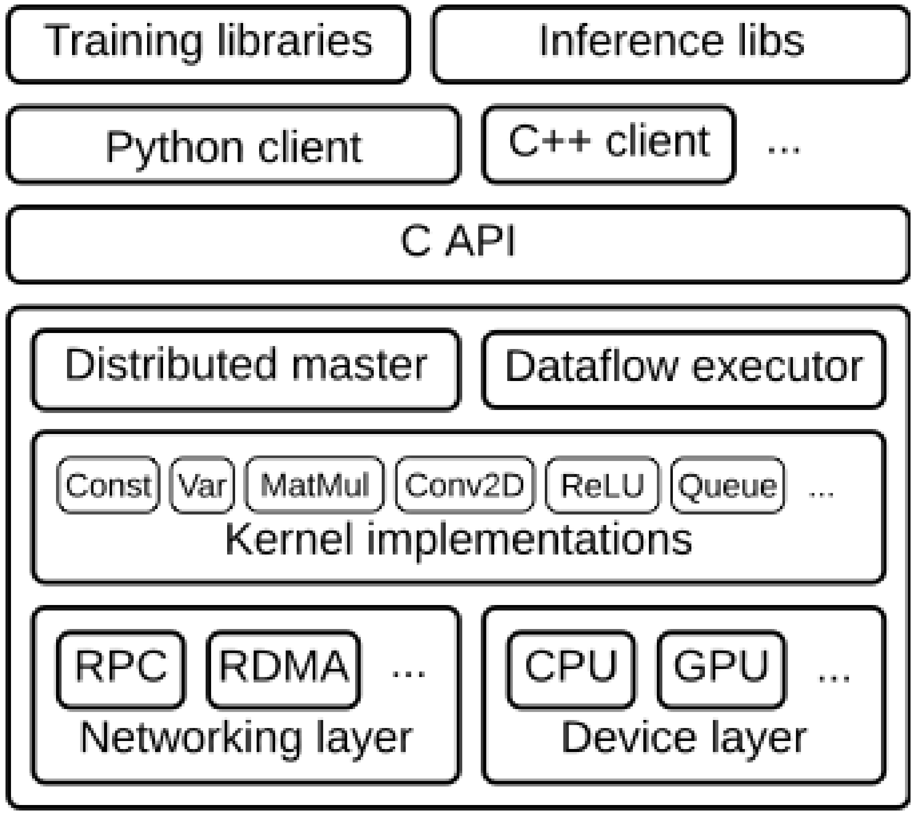
- Node placement: heuristics placement algo; device constraints
- Cross-device communication



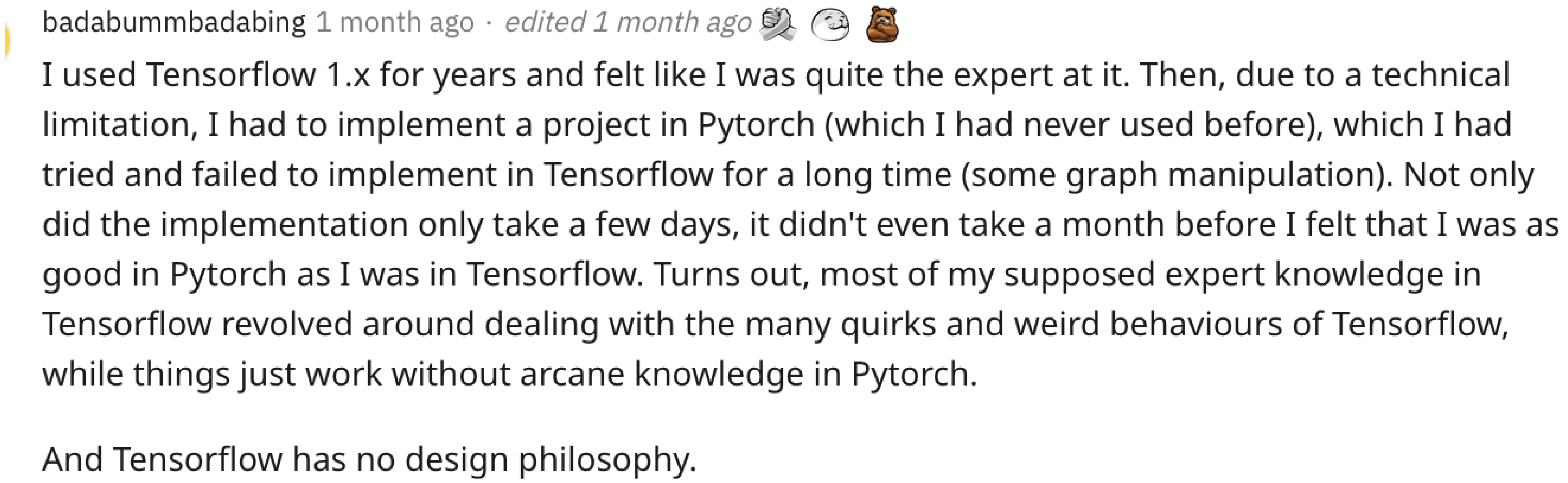
Distributed Executions

- Checkpoint support: to connect Variable node to Save node/Restore node

General Architecture



Eh?


badabummbadabing 1 month ago · *edited 1 month ago*

I used Tensorflow 1.x for years and felt like I was quite the expert at it. Then, due to a technical limitation, I had to implement a project in Pytorch (which I had never used before), which I had tried and failed to implement in Tensorflow for a long time (some graph manipulation). Not only did the implementation only take a few days, it didn't even take a month before I felt that I was as good in Pytorch as I was in Tensorflow. Turns out, most of my supposed expert knowledge in Tensorflow revolved around dealing with the many quirks and weird behaviours of Tensorflow, while things just work without arcane knowledge in Pytorch.

And Tensorflow has no design philosophy.