Where We Are

Machine Learning Systems	2012 - Now
Big Data	2010 - Now
Cloud	2000 - 2016
Foundations of Data Systems	1980 - 2000

Logistics

- Exam date:
  - Final Exam date (tentative): **Friday, March 22, 8 - 11 am, PT**
  - TAs and I are still debating between classroom or canvas
    - Will update you by this Wed (3/13)
  - All Multiple choice questions
- Course Evaluation
  - It is important for both me, yourself, and TAs
  - Please participate to get your extra credits ☺

ML System history

- ML Systems evolve as more and more ML components (models/optimization algorithms) are unified

Ad-hoc: diverse model family, optimization algos, and data

Opt algo: iterative-convergent

Model family: neural nets

Model: CNNs/transformers/GNNs

LLMs: transformer decoders

Today: NN, data flow graph, and data parallelism

Recap: Parameter Server

- Pros?
  - General: Abstract iterative-convergent algo
  - Relax Consistency: stale synchronous
  - Nice interface like map-reduce
- Cons?
  - Extension to GPUs?
  - Strong assumption on communication bottleneck

The Second Unification: Neural Networks

Imagenet classification with deep convolutional neural networks [PDF] neurips.cc

A Krizhevsky, I Sutskever… - Advances in neural …, 2012 - proceedings.neurips.cc

We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the …

Save Cite Cited by 126745 Related articles All 102 versions

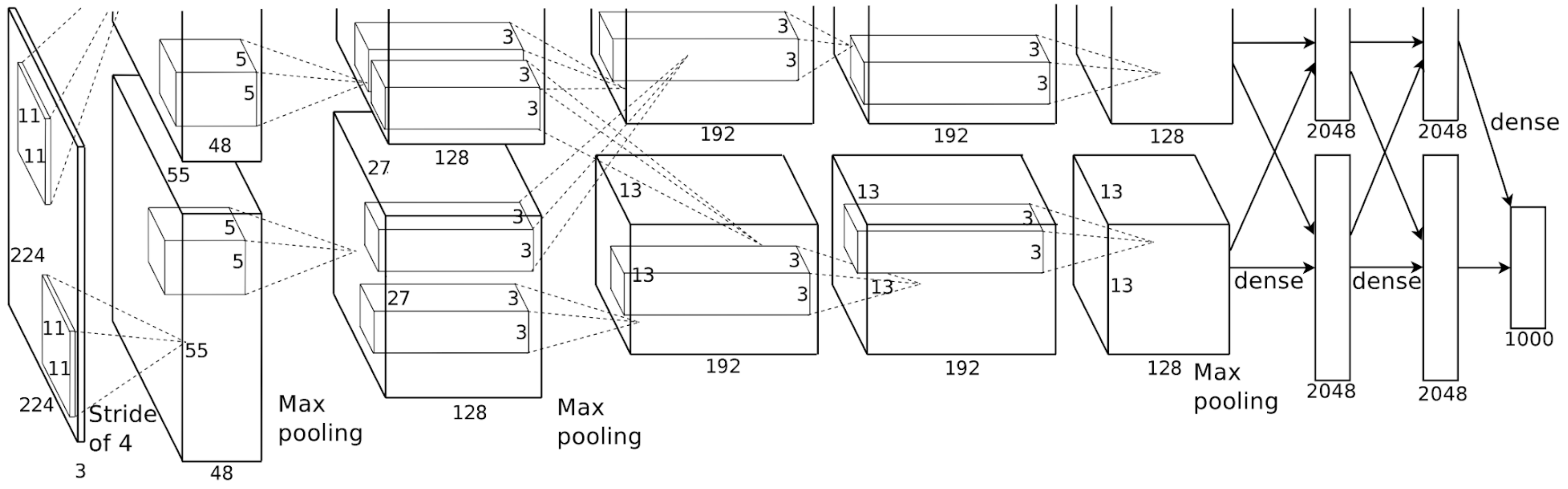


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure from AlexNet
[Krizhevsky et al., NeurIPS 2012], [Krizhevsky et al., preprint, 2014]

Why DL Emerged and Succeeded(but failed before)?

- It beats the previous state-of-the-art method by 10 points
  - Every year we see 1 point improvement in the past 10 years
- It scales with the size of data
  - Train with the entire ImageNet data
- It is simple: optimized by 1^{st}-order method SGD
- Its computation pattern aligns with hardware (GPU/accelerators)

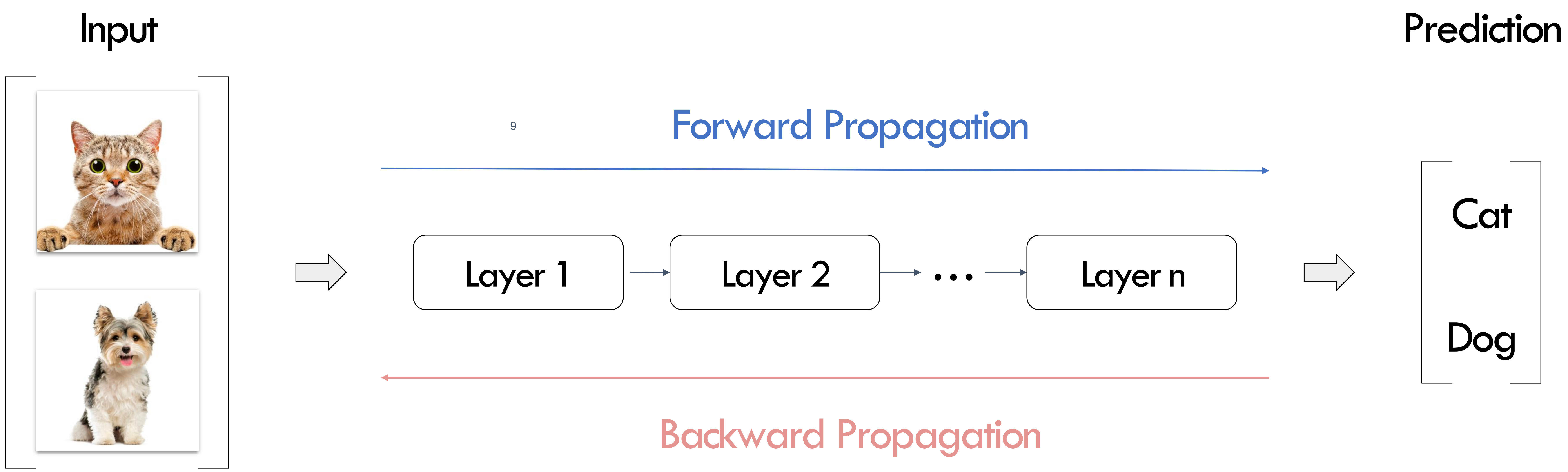
Deep learning Characteristics

- Iterative-convergent?
  - Yes: SGD
- Model still diverse?
  - No: much less diverse than the entire spectrum of ML
  - Yes: Still many flavors of NNs and needs sufficiently expressive lib to program various architectures
- Compute very intensive?
  - Yes: GPU becomes a must
- Model very large?
  - No: It starts with a relatively small model (2012)
  - Yes: It becomes large when people discover the transformer architecture
- Existing data systems to program NNs?
  - Map-reduce: not for iterative-convergence
  - Spark: op lib is very corase grained and not for neural network ops
  - P: programming model offers too many flexibility which renders it not so helpful

Outline

- **Deep Learning as Dataflow Graphs**
- Auto-differentiation Libraries
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism

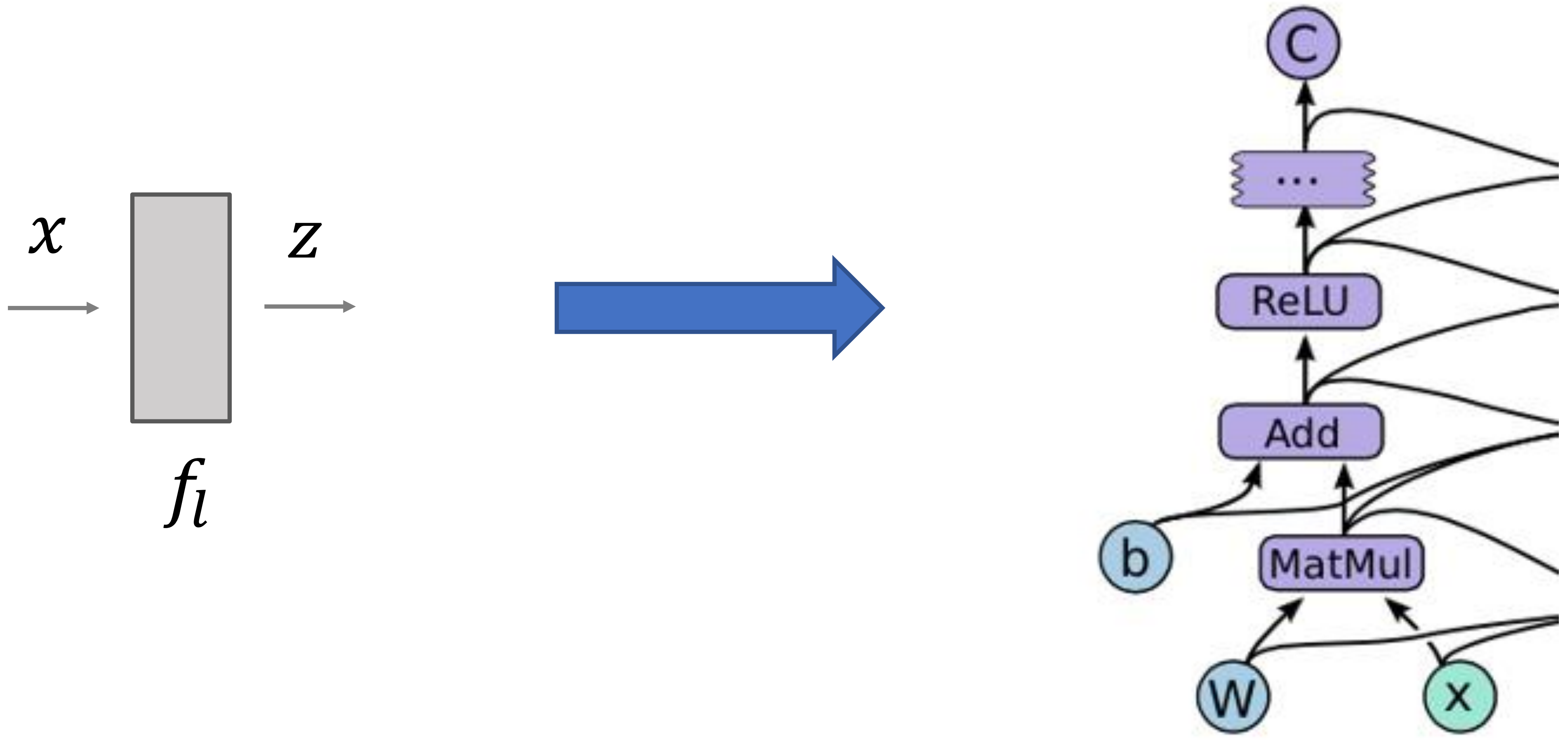
Background: DL Computation



$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update} \\ \text{(sgd, adam, etc.)}}}\Big(\theta^{(t)}, \underbrace{\nabla_L}_{\substack{\text{model} \\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)}, \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

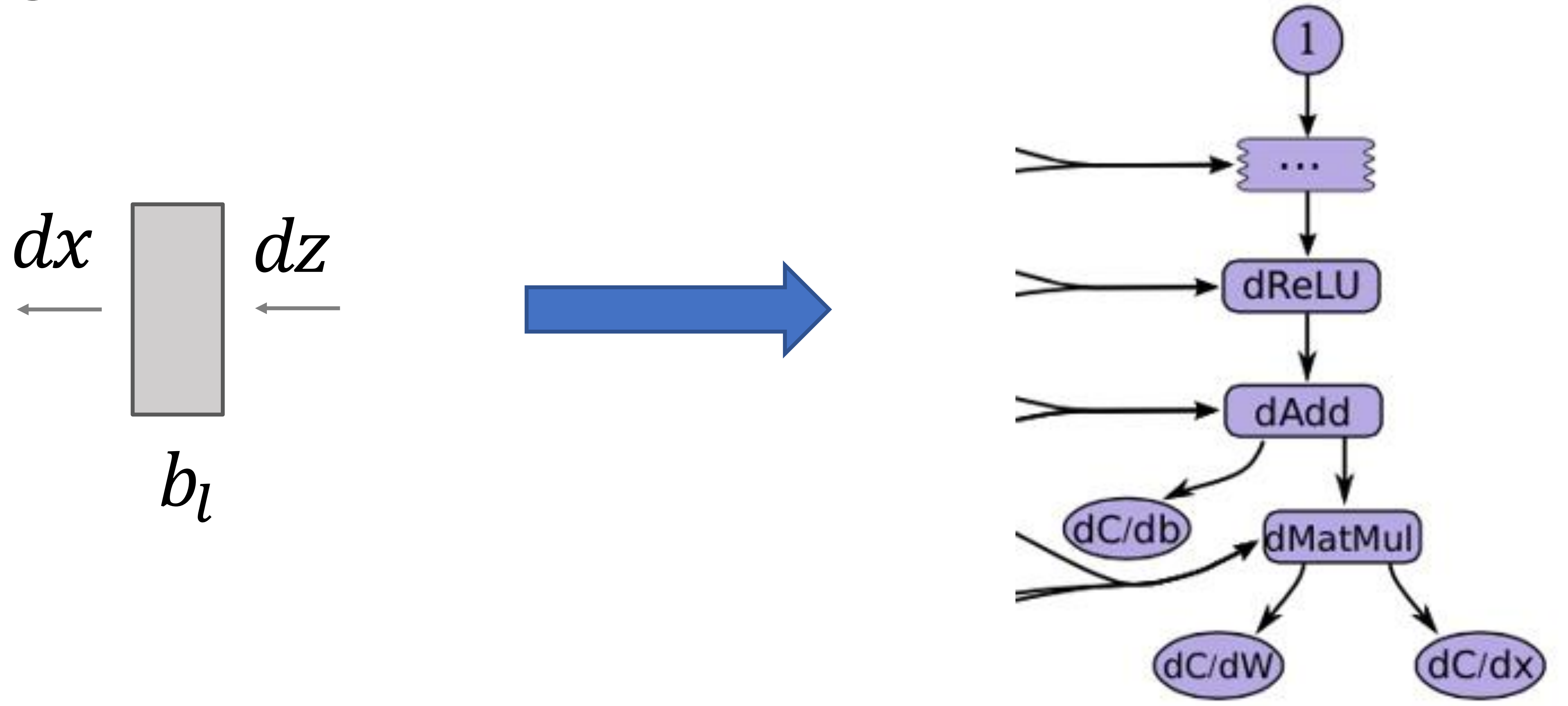
A Computational Layer in DL: forward

- A layer in a neural network is composed of a few finer computational operations
  - Consider: $\mathrm{z} = \mathrm{f_l}(\mathrm{x})$: $y = Wx + b, z = ReLU(y)$



A Computational Layer in DL: backward

- Denote the backward pass through a layer l as b_l
  - b_l derives the gradients of the input $x(\mathrm{d}x)$,given the gradient of z as $\mathrm{d}z$, as well as the gradients of the parameters W, b
  - $\mathrm{d}x$ will be the backward input of its previous layer $l - 1$
  - Backward pass can be thought as a backward dataflow where the gradient flow through the layer



A Layer as a Dataflow Graph

- Give the forward computation flow, gradients can be computed by auto differentiation
  - Automatically derive the backward gradient flow graph from the forward dataflow graph

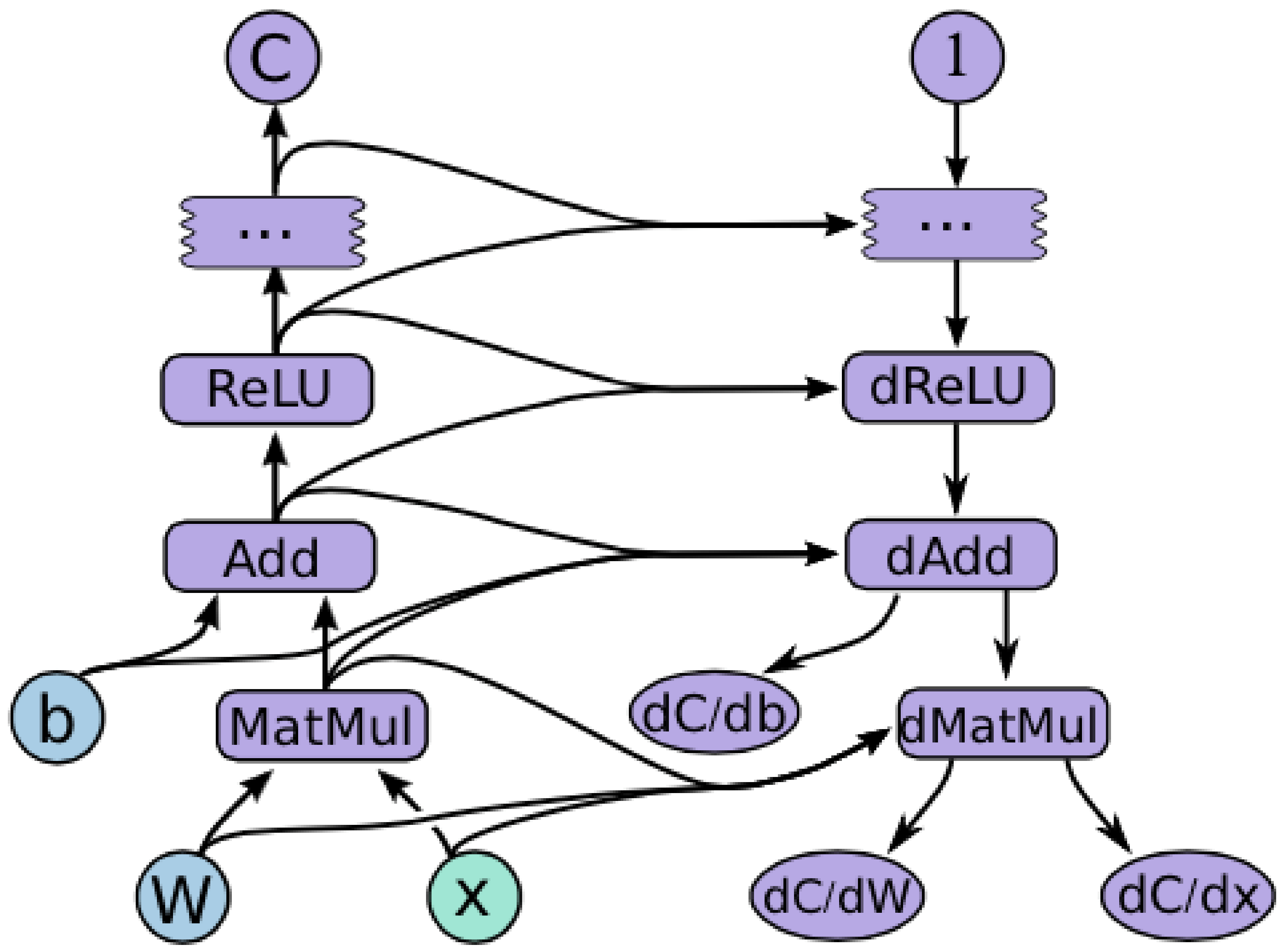
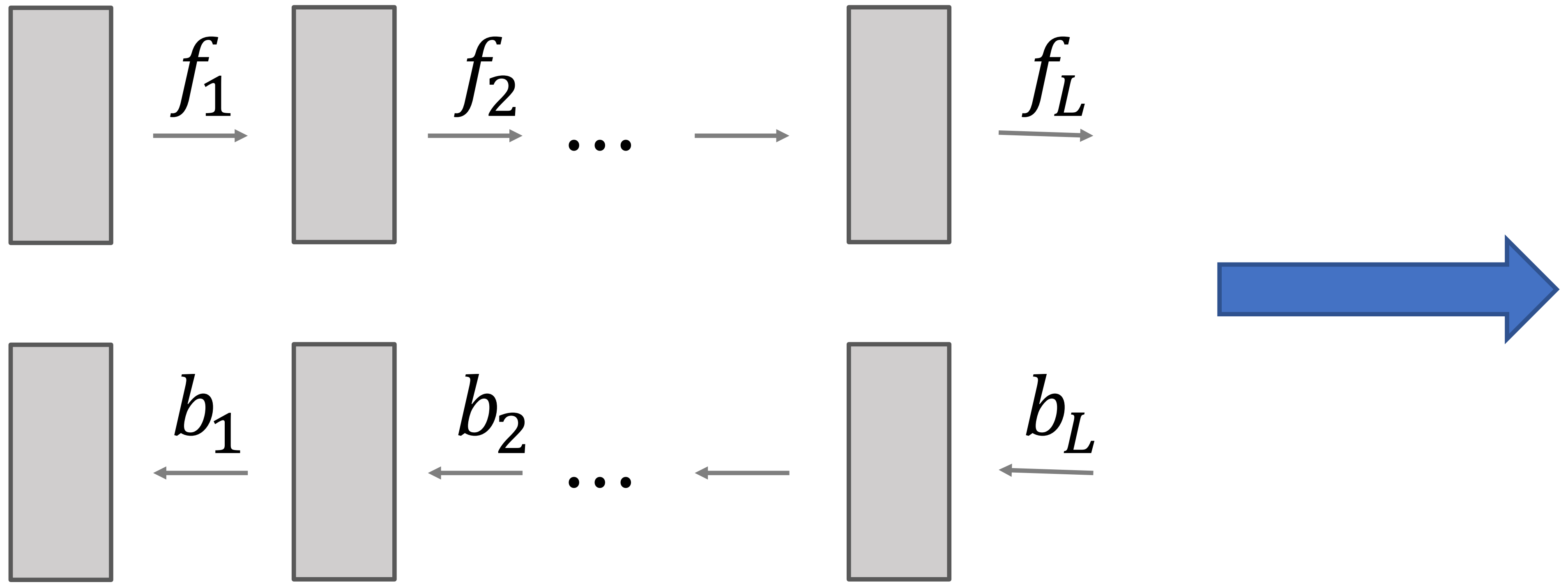


Photo from TensorFlow website

Combining Weight Update

- Gradients can be computed by auto differentiation
  - Automatically derive the gradient flow graph from the forward dataflow graph



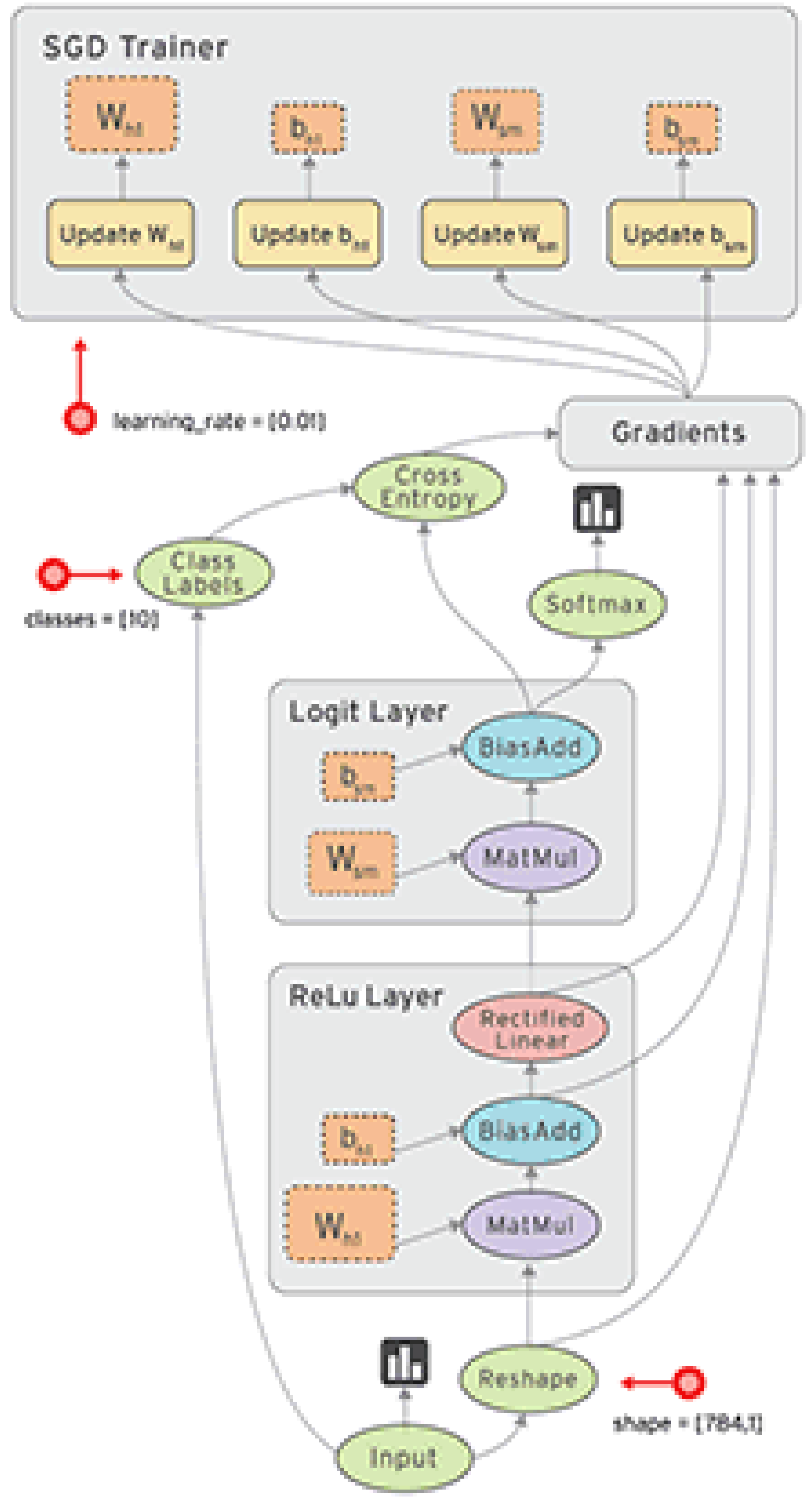


Photo from TensorFlow website

Practice

$$\theta^{(t+1)} = f\left(\theta^{(t)},\ \nabla_L\left(\theta^{(t)},\ D^{(t)}\right)\right)$$

$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x),\ y) \quad \theta = \{w_1, w_2\},\ D = \{(x, y)\}$$

$$f(\theta, \nabla_L) = \theta - \nabla_L$$

Forward	Backward	Weight update
$L(\cdot)$	$\nabla_L(\cdot)$	$f(\cdot)$

Practice

$$\theta^{(t+1)} = f\left(\theta^{(t)},\ \nabla_L\left(\theta^{(t)},\ D^{(t)}\right)\right)$$

$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x),\ y) \quad \theta = \{w_1, w_2\},\ D = \{(x, y)\}$$

$$f(\theta, \nabla_L) = \theta - \nabla_L$$

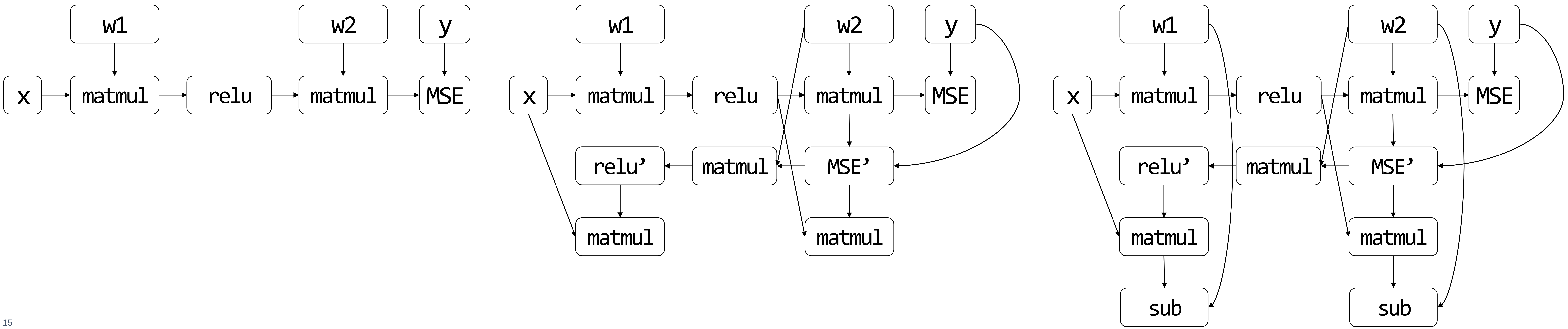
Operator / its output tensor

Data flowing direction

Forward

+Backward

+Weight update



Dataflow Graph Programming Model Today

- Define a neural network
  - Define operations and layers: fully-connected? Convolution? Recurrent?
  - Define the data I/O: read what data from where?
  - Define a loss function/optimization objective: L2 loss? Softmax? Ranking Loss?
  - Define an optimization algorithm: SGD? Momentum SGD? etc
- Auto-differential Libraries will then take over
  - Connect operations, data I/O, loss functions and trainer.
  - Build forward dataflow graph and backward gradient flow graphs.
  - Perform training and apply updates

Discussion:
Compare this vs. Spark, parameter server, MapReduce?

Outline

- Deep Learning as Dataflow Graphs
- **Auto-differentiable Libraries**
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism

Auto-differential Libraries

- Auto-differential Library automatically derives the gradients following the back-propagation rule.
- A lot of auto-differentiation libraries have been developed:

Caffe Caffe2 TensorFlow Microsoft CNTK

torch PYTORCH

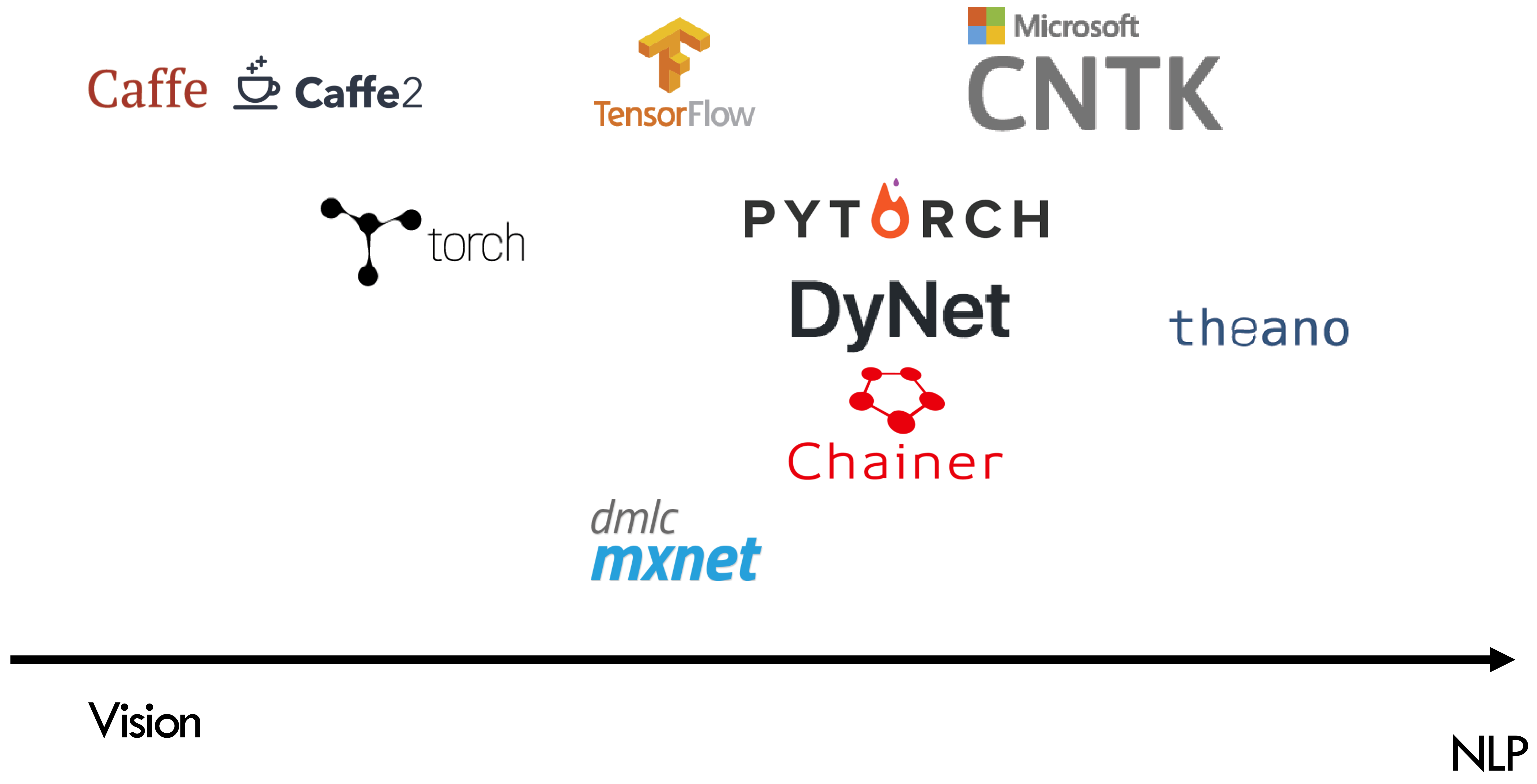
DyNet theano

Chainer

dmlc mxnet

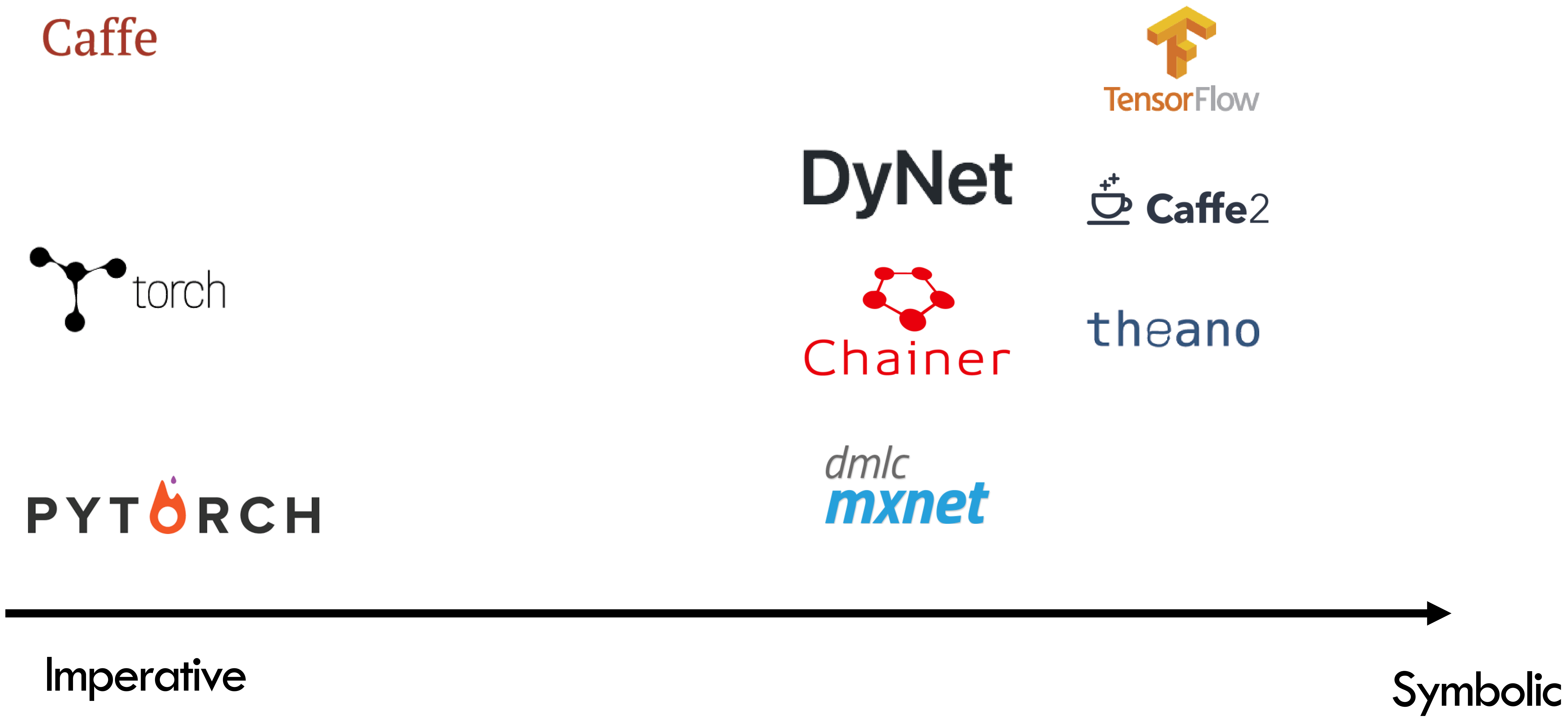
Deep Learning Toolkits

- They are roughly adopted by different domains



Symbolic vs. Imperative

- They are also designed differently
  - Symbolic v.s. imperative programming



Symbolic vs. Imperative

- Symbolic vs. imperative programming
  - Symbolic: write symbols to assemble the networks first, evaluate later
  - Imperative: immediate evaluation

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
# compiles the function
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

Symbolic

```
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

Imperative

Symbolic vs. Imperative

- Symbolic
  - Good
    - easy to optimize (e.g. distributed, batching, parallelization) for developers
    - More efficient
  - Bad
    - The way of programming might be counter-intuitive
    - Hard to debug for user programs
    - Less flexible: you need to write symbols before actually doing anything
- Imperative:
  - Good
    - More flexible: write one line, evaluate one line (that’s why we all like Python)
    - Easy to program and easy to debug: because it matches the way we use C++ or python
  - Bad
    - Less efficient
    - More difficult to optimize

Are All Models expressive in Dataflow Graph?