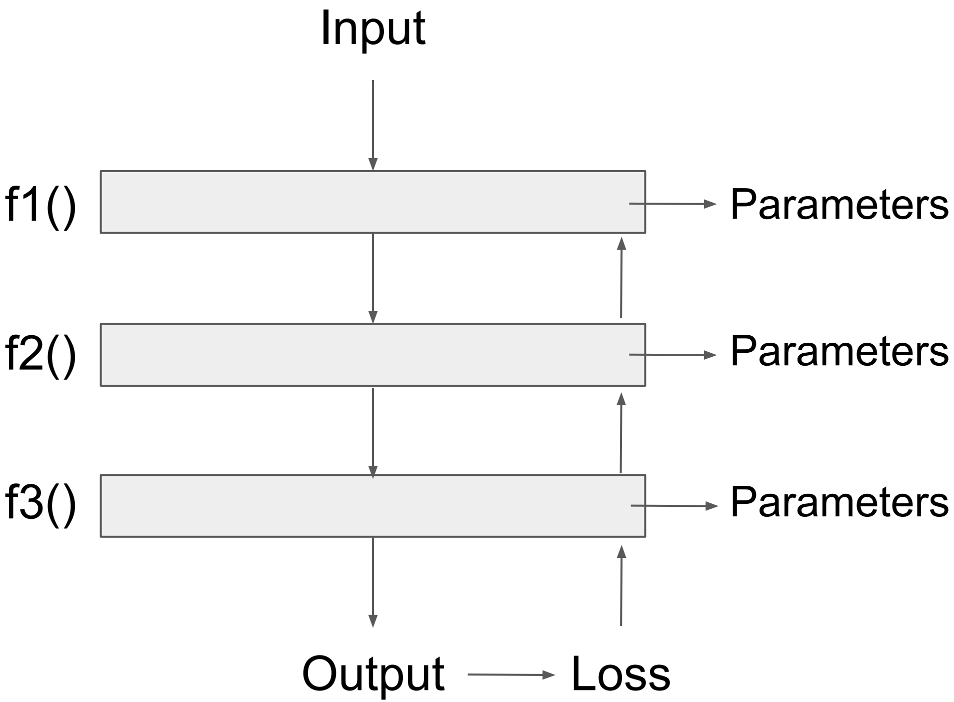
Autodiff Bootcamp: new_grad

Kinori Rosnow, Anurag Katakkar, Shriti Priya, David Park

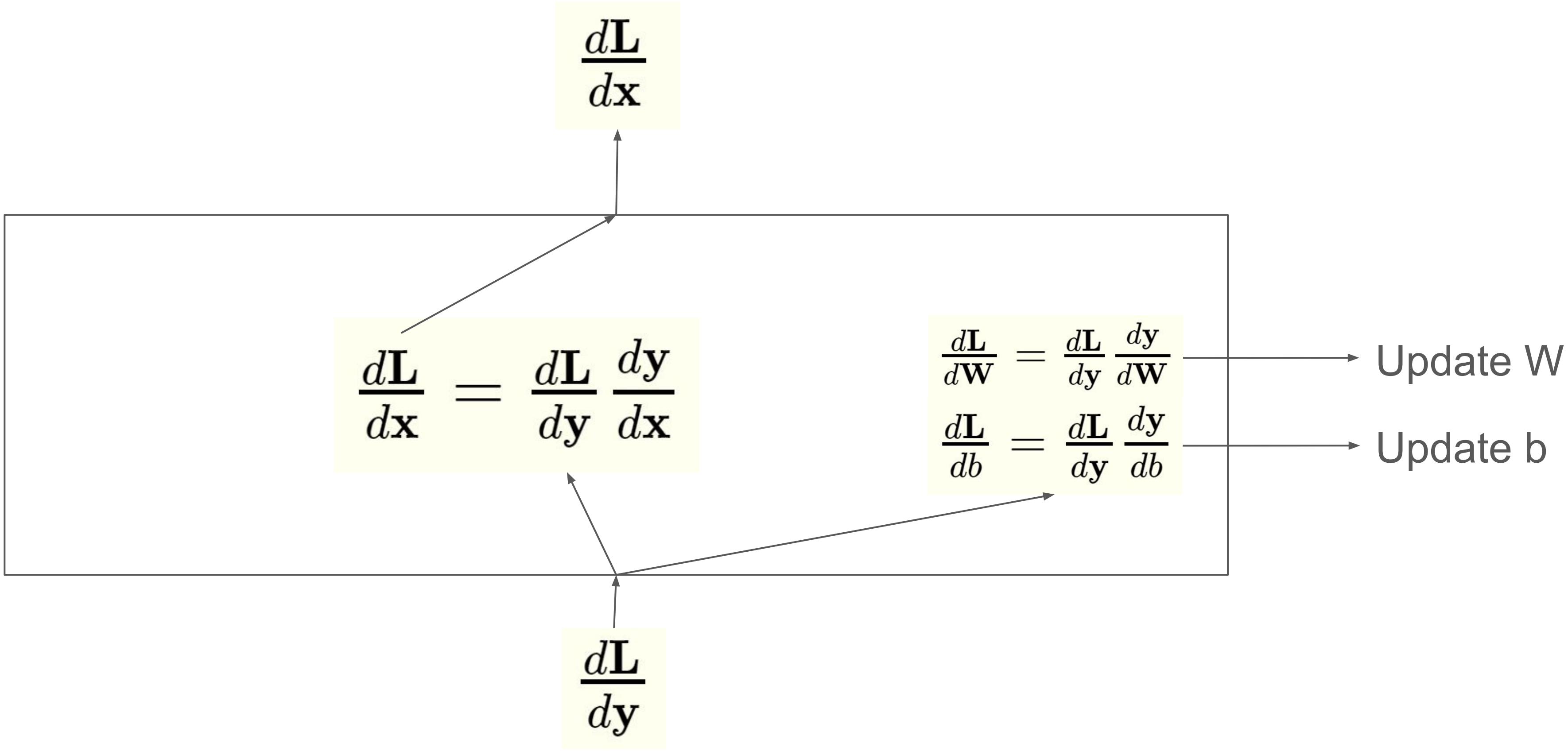
Backpropagate Loss

1. Forward
2. Calculate Loss
3. Pass Gradient with respect to output
4. Update Parameters
5. Continue



Output = f3(f2(f1(x)))

Single Layer Backward: Linear

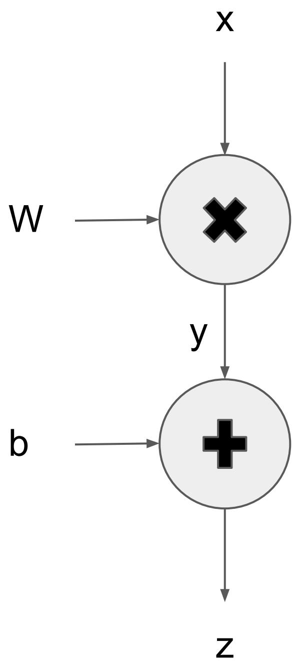


How does Pytorch take derivatives and backpropagate?

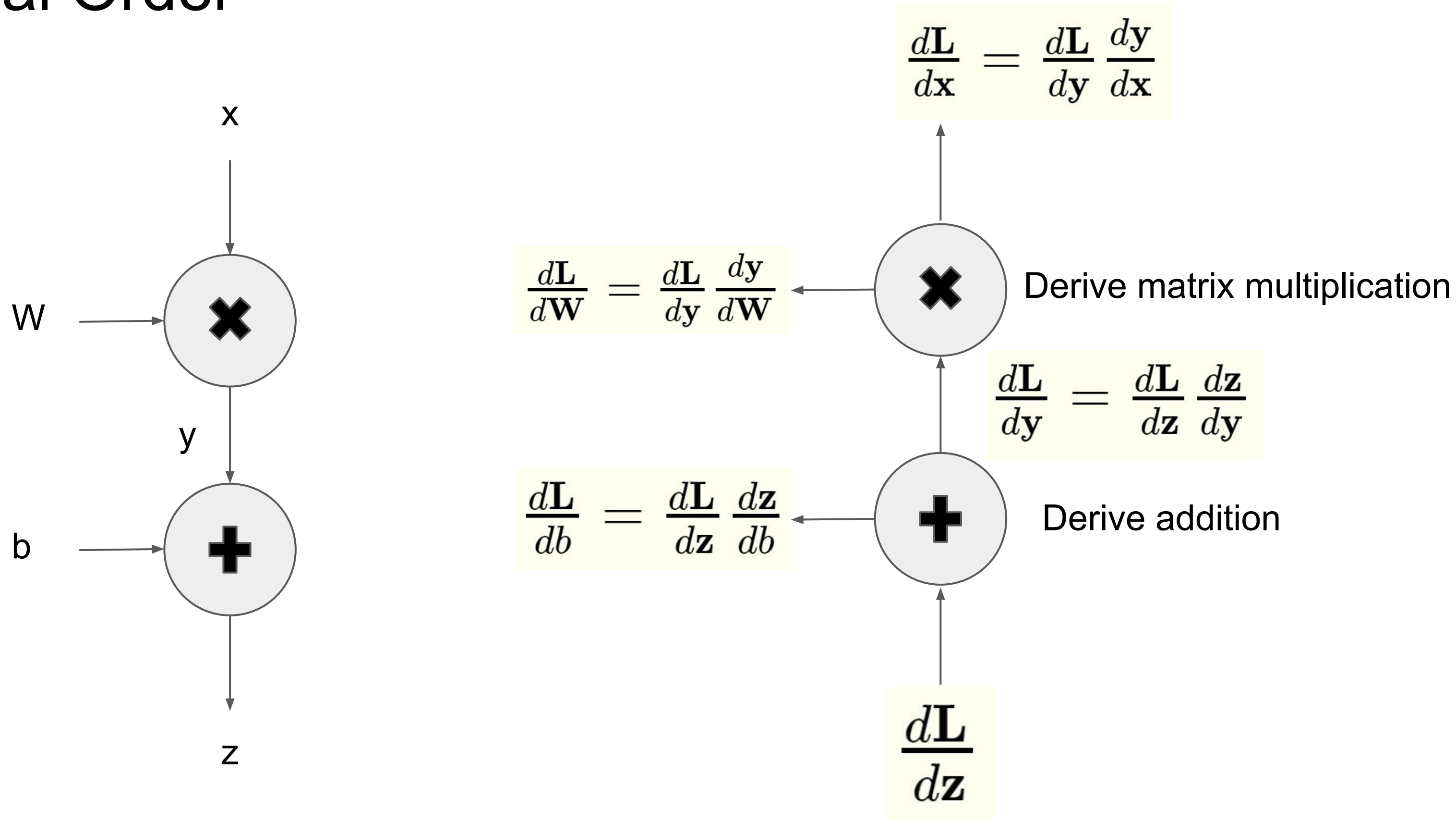
Auto-differentiation:

- All of the functions can be rewritten into basic operations
  - True for all computer based calculations
- Sequence of operations instead of a layers
- Each operation is differentiable

$$\mathbf{z} = \mathbf{W}\mathbf{x} + b \longrightarrow \begin{array}{l} \mathbf{y} = \mathbf{W}\mathbf{x} \\ \mathbf{z} = \mathbf{y} + b \end{array}$$

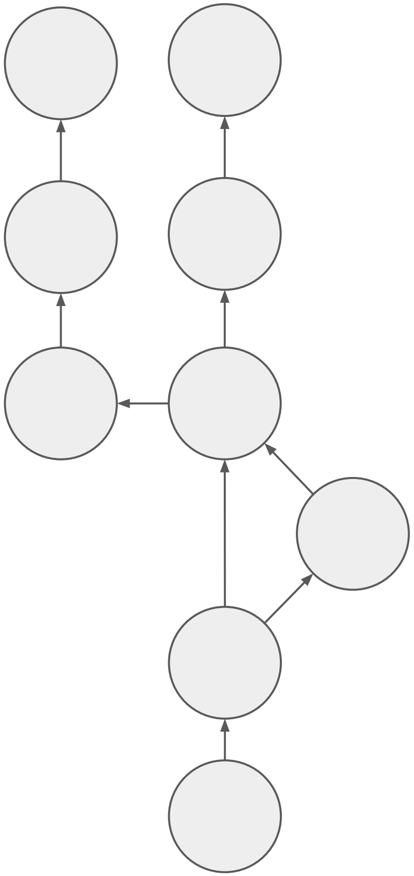


Operational Order

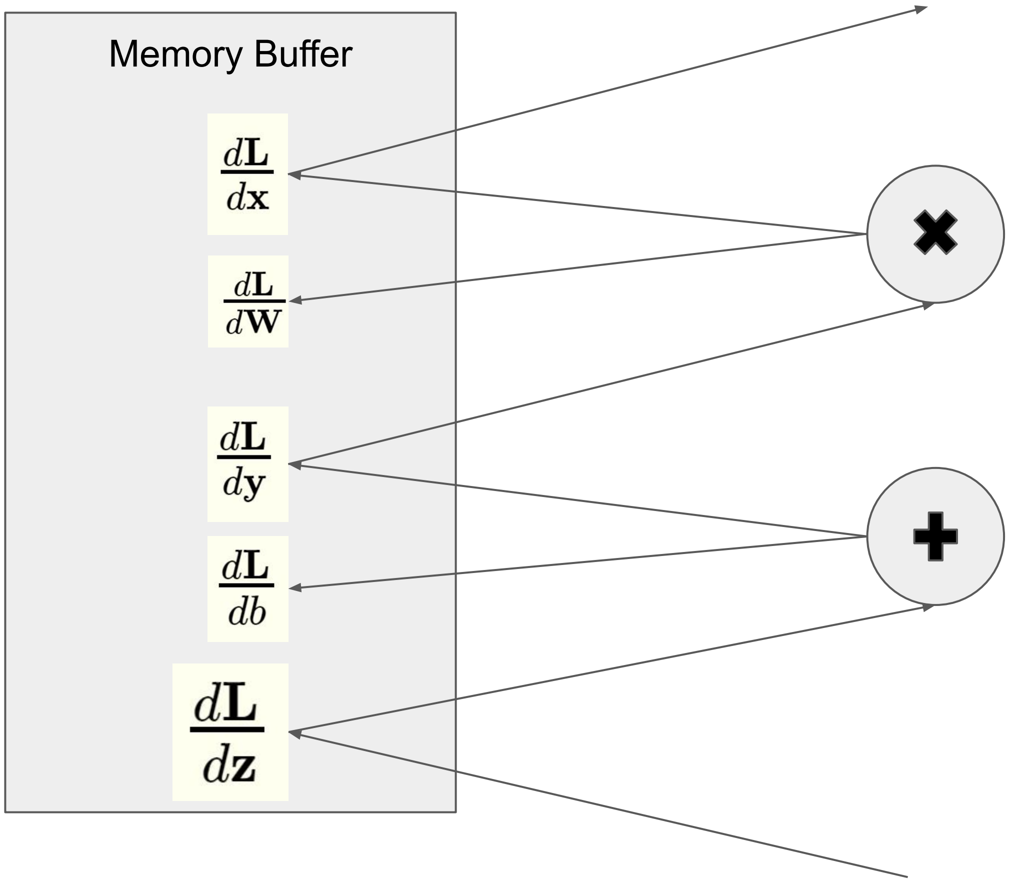


Deep Learning Computation Actually

- Operations are monotonically ordered
- 2 methods for backprop
  - Traverse directed acyclic graph (DAG)
  - Take advantage of ordering - clever gradient storage
- Pytorch's Autograd - tensor class
  - Computational DAG
  - Backpropagation = graph traversal
- new_grad - memory buffer class
  - Computational list
  - Backpropagation = iterate backwards



Operation List Implementation



Derive matrix multiplication

$$\frac{d\mathbf{L}}{d\mathbf{x}} = \frac{d\mathbf{L}}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}}$$

$$\frac{d\mathbf{L}}{d\mathbf{W}} = \frac{d\mathbf{L}}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{W}}$$

Derive addition

$$\frac{d\mathbf{L}}{d\mathbf{y}} = \frac{d\mathbf{L}}{d\mathbf{z}} \frac{d\mathbf{z}}{d\mathbf{y}}$$

$$\frac{d\mathbf{L}}{db} = \frac{d\mathbf{L}}{d\mathbf{z}} \frac{d\mathbf{z}}{db}$$