COMP0169: Machine Learning for Visual Computing

Backprop and Overfitting

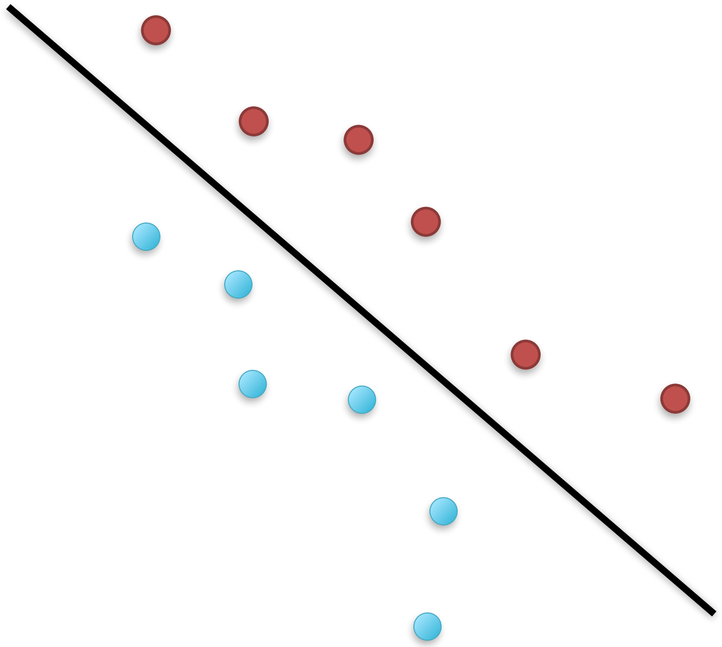
UCL

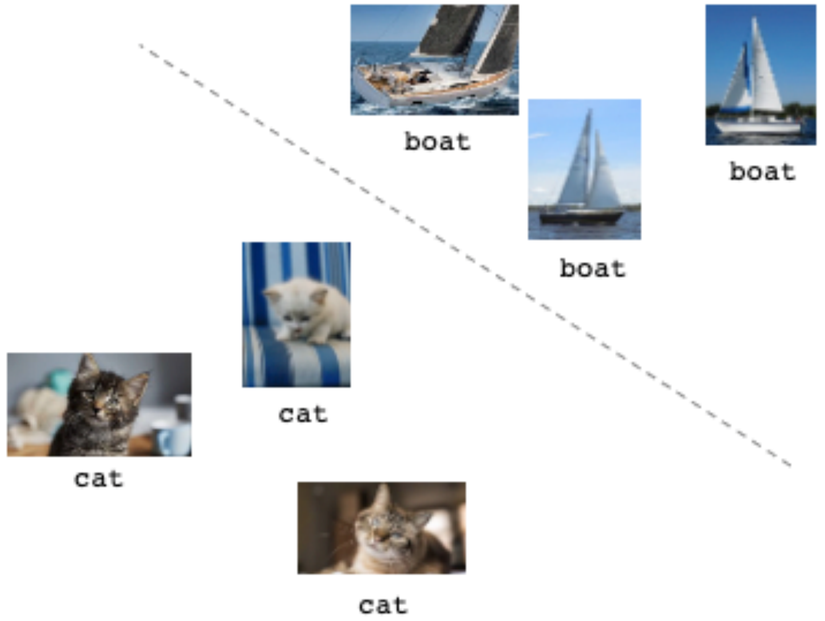
Lectures will be Recorded

Recap

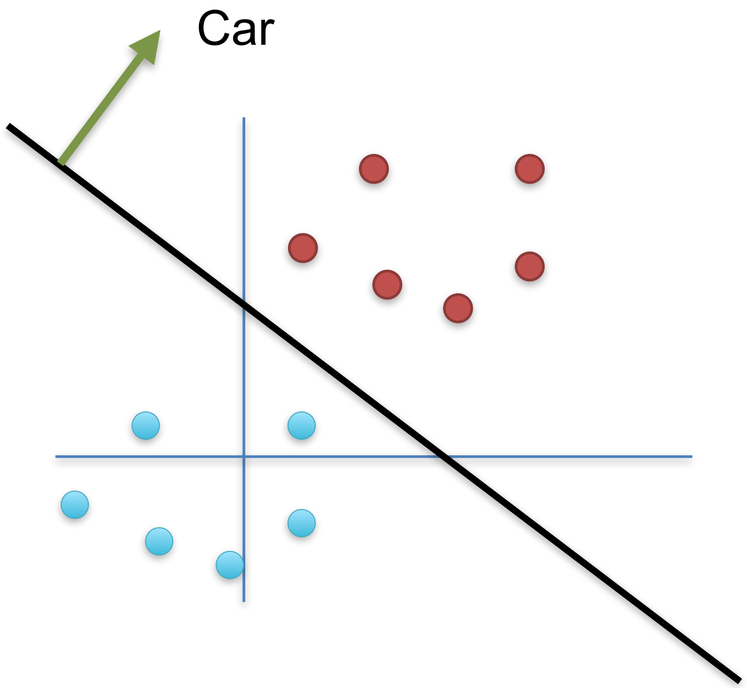
- PCA
- Normal Equation (linear and polynomial fits)
- Linear Classifiers
  - Linear Regression
  - Perceptron
  - Logistic regression

Simple Classifier





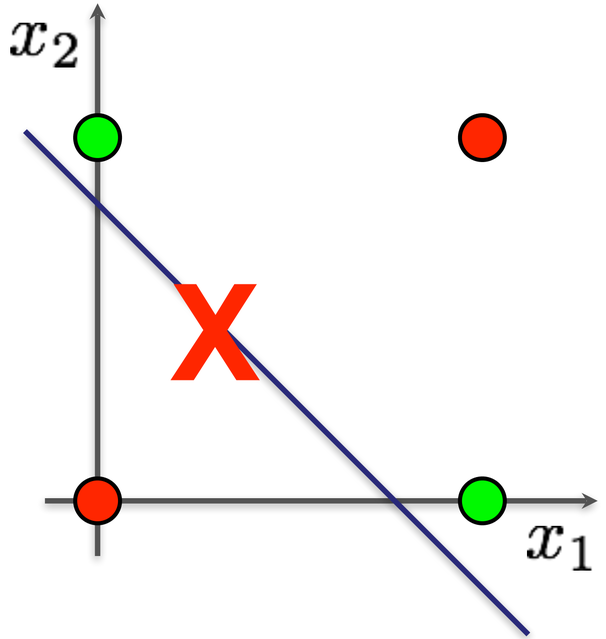
Classifier Geometric Viewpoint



XOR Problem

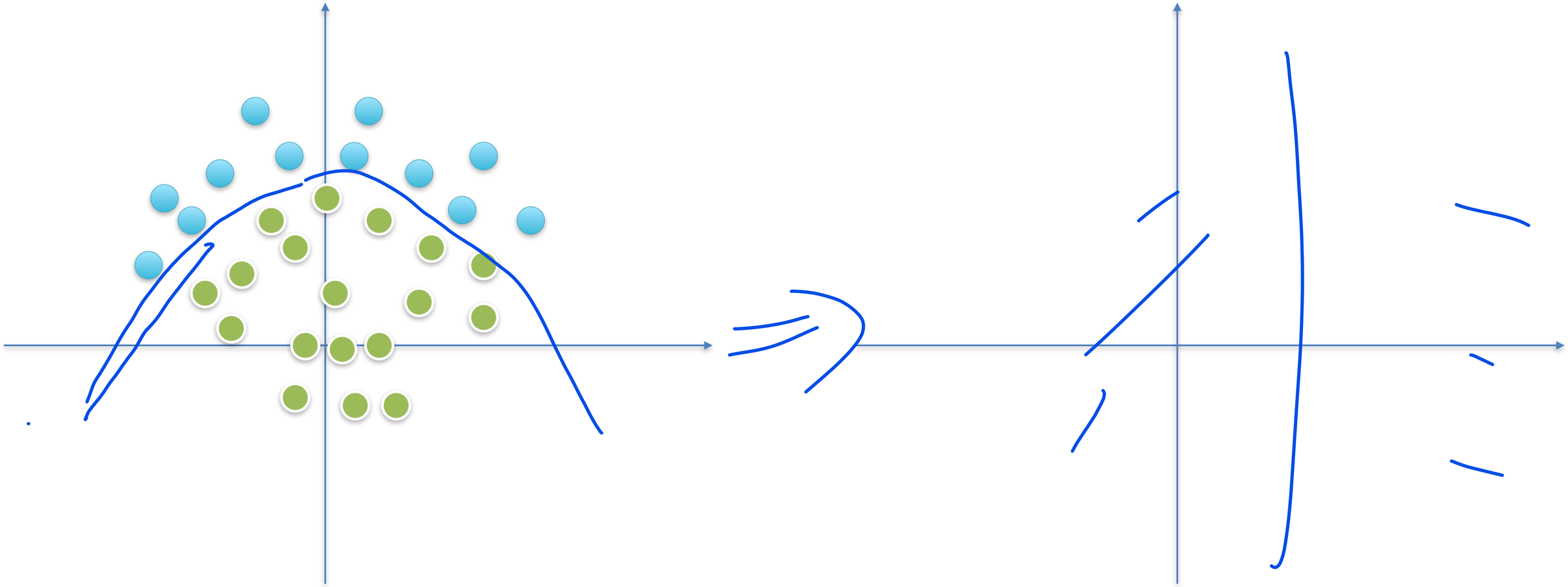
$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2) = \mathcal{H}(\mathbf{w}^T \mathbf{x})$$

What are Neural Networks Doing?

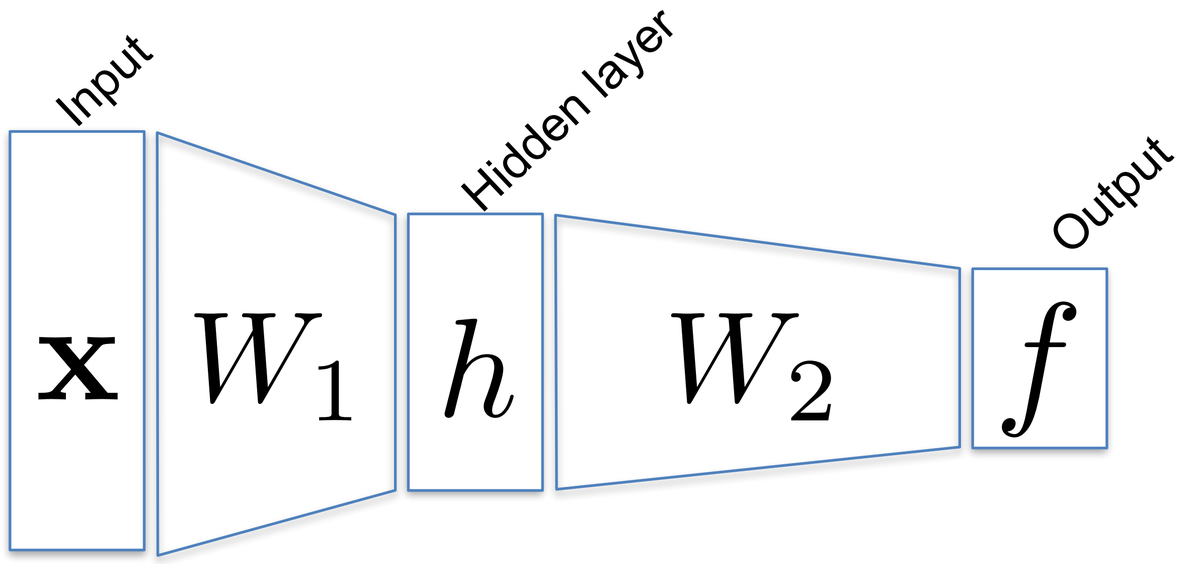


Gradient-based Optimisation

$$\min_{\theta} f_{\theta}(\{\mathbf{x}^{(i)}\}) \qquad \max_{\theta} f_{\theta}(\{\mathbf{x}^{(i)}\})$$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} f_{\theta} \qquad \theta \leftarrow \theta + \alpha \nabla_{\theta} f_{\theta}$$

2-layer Network

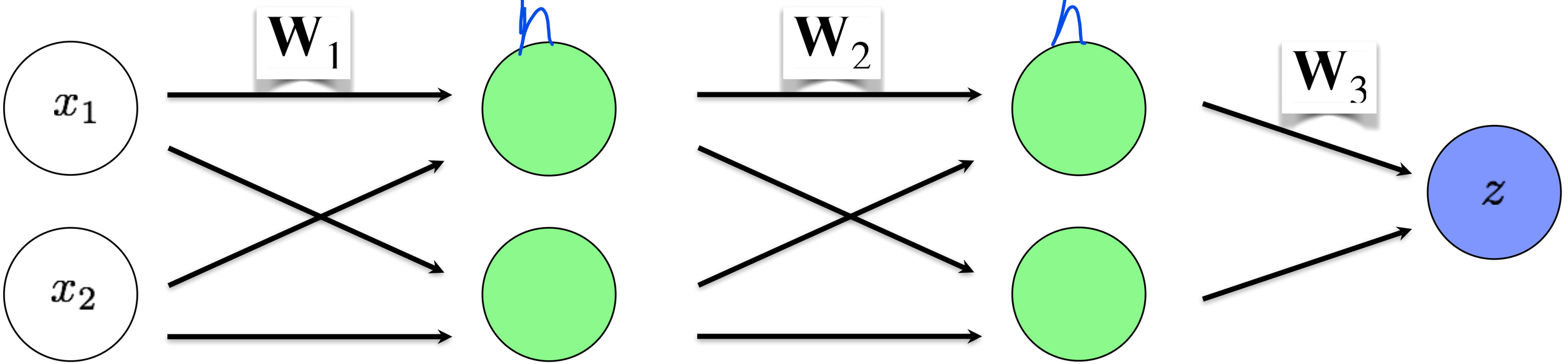


Activation Functions

- **ReLU(x)** $\mathrm{RELU}(x) = \max(0, x)$
- **Sigmoid** $\sigma(x) = \frac{1}{1+\exp(-x)}$ $\sigma'(x) = \sigma(x)(1-\sigma(x))$
- **Tanh** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Leaky ReLU** $\max(\lambda x, x) \;\; \lambda << 1$
- **ELU**

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

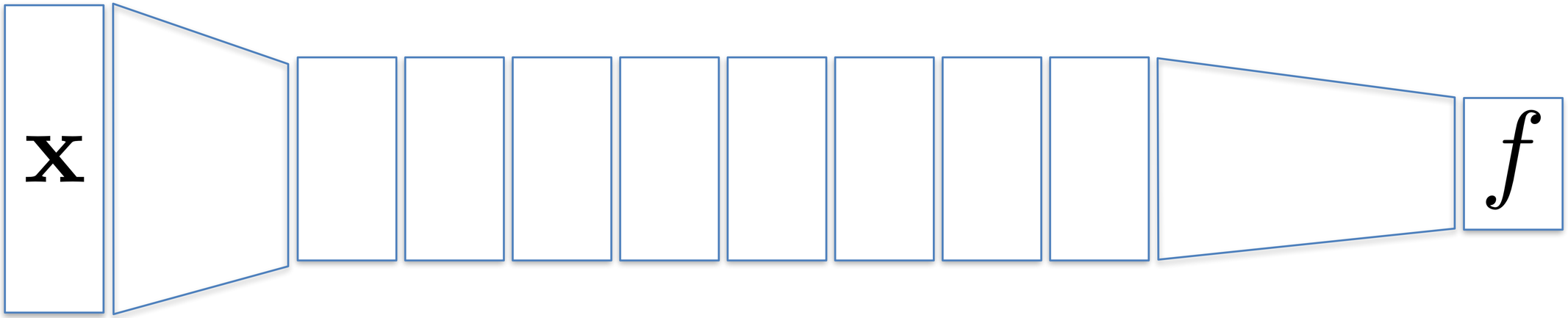
3-layer Neural Network



$$z = \mathscr{A}(\mathbf{W}_3\mathscr{A}(\mathbf{W}_2\mathscr{A}(\mathbf{W}_1\mathbf{x})))$$

$$\mathscr{A}(p) := \max(0,p)$$

Deep Networks



$$f = W_i \max(0, W_{i-1} \max(0, \cdots W_0 \mathbf{x})$$

Derivatives

- $y = f(x)$
- $y = f(\mathbf{x})$
- $y = f(\mathbf{A}\mathbf{x})$

Derivatives (Recap)

- $f(x) = ax \quad f'(x) = a$
- $f(x) = \frac{1}{x} \quad f'(x) = -\frac{1}{x^2}$
- $f(x) = x + a \quad f'(x) = 1$
- $f(x) = e^x \quad f'(x) = e^x$
- **Chain rule**

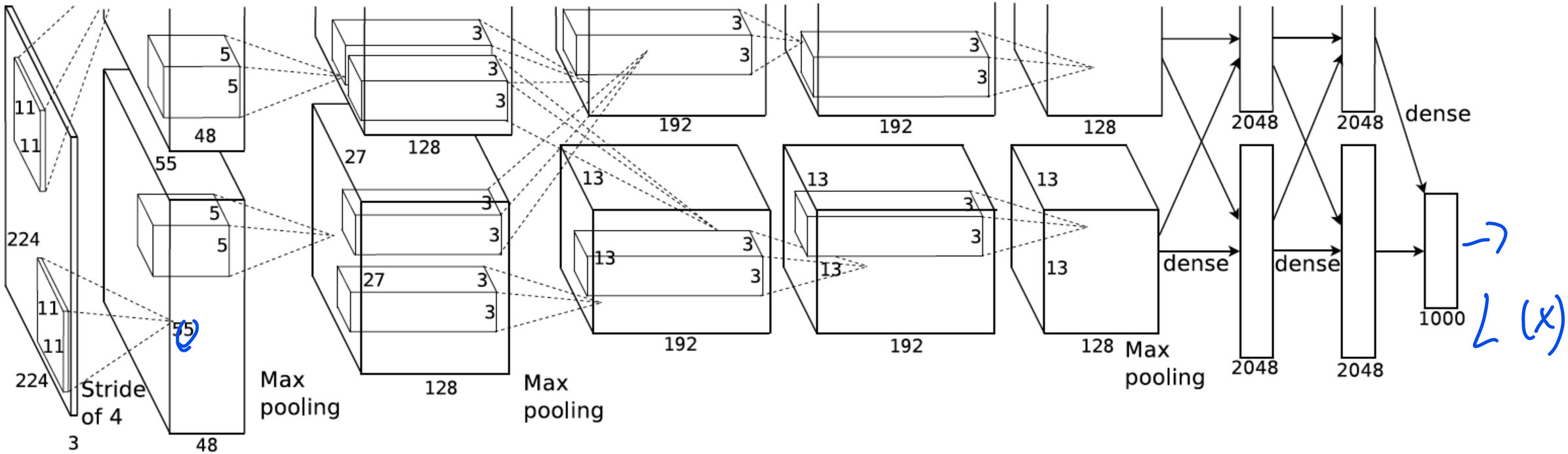
$f(x) = \frac{1}{1+e^{-x}}$

chain rule ⇒

$\frac{df}{dx} = \frac{-1}{(1+e^{-x})^2} \cdot (-e^{-x})$

$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$

Complex Networks (e.g., AlexNet)



Chain Rule

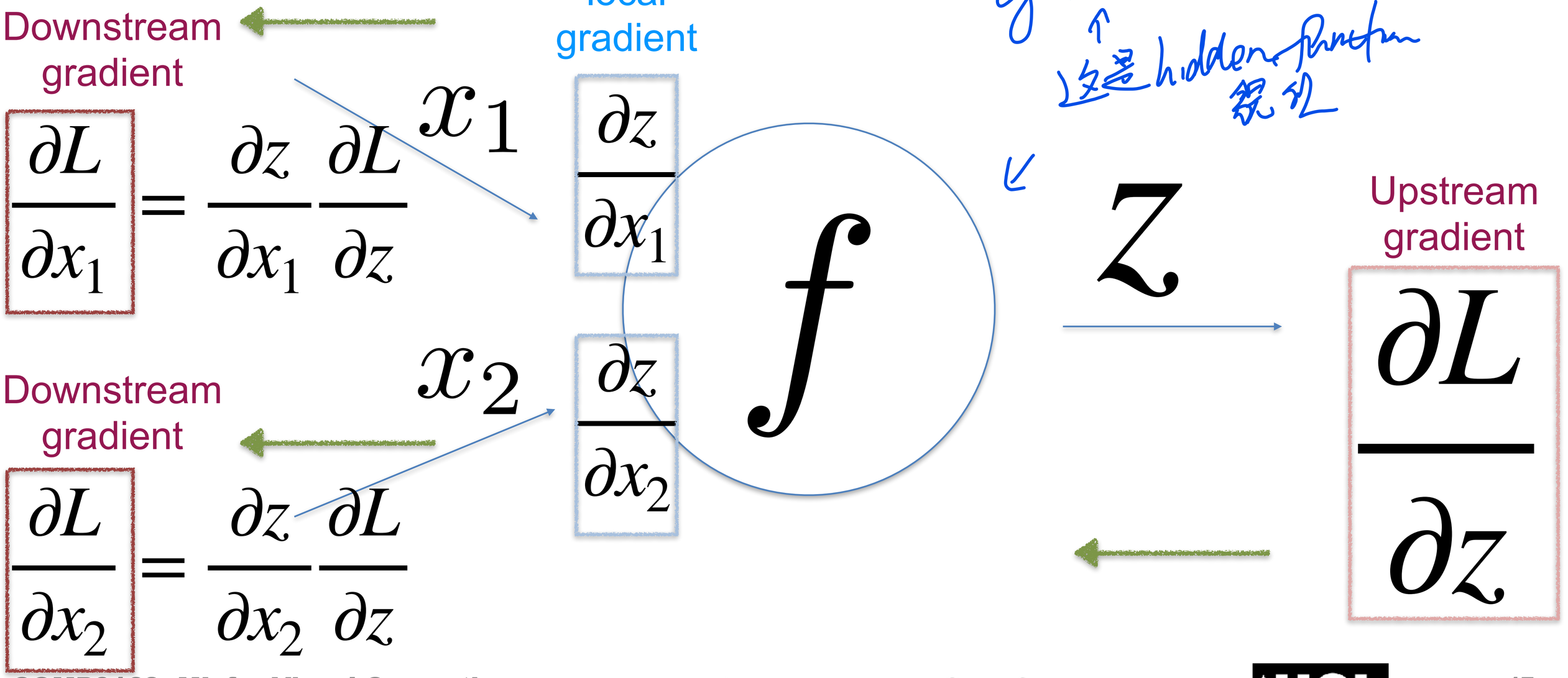
Single variable

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)}\frac{dg(x)}{dx} = \frac{df(g)}{dg}\frac{dg}{dx} = f'(g(x))g'(x)$$

Multiple variables

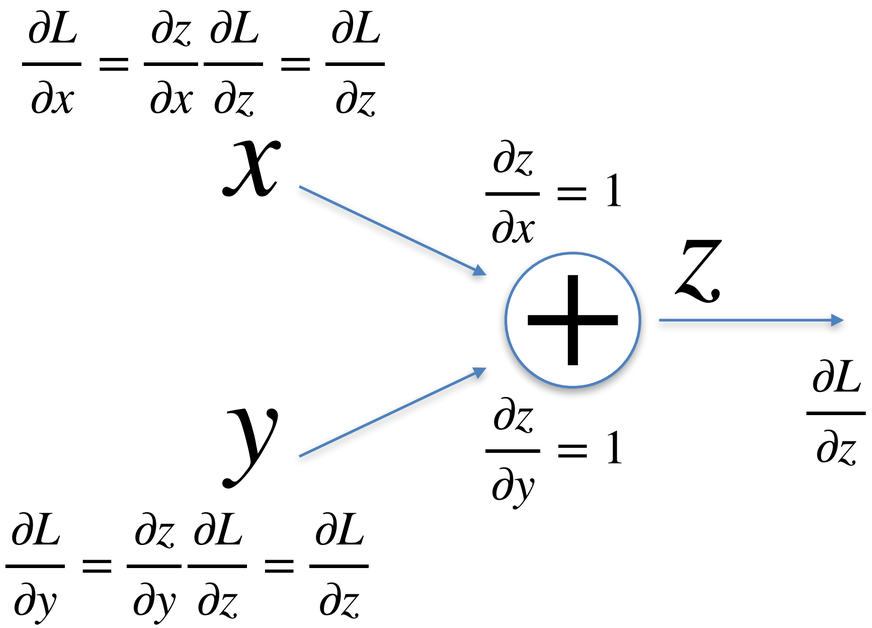
$$\frac{df(x(t),y(t))}{dt} = \frac{\partial f(x(t),y(t))}{\partial x(t)}\frac{dx(t)}{dt} + \frac{\partial f(x(t),y(t))}{\partial y(t)}\frac{dy(t)}{dt} = \frac{\partial f(x,y)}{\partial x}\frac{dx}{dt} + \frac{\partial f(x,y)}{\partial y}\frac{dy}{dt}$$

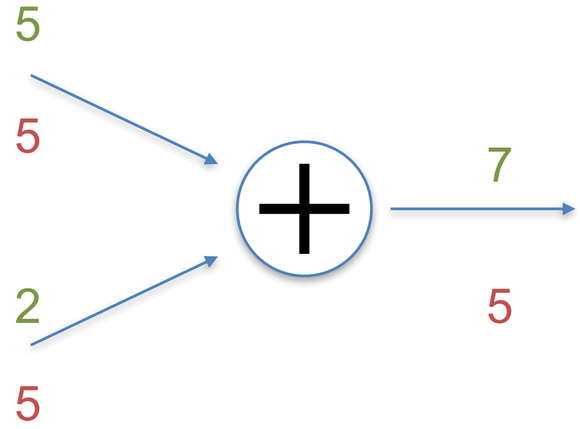
Basic Block


Downstream gradient
local gradient
$\frac{\partial L}{\partial x_1} = \frac{\partial z}{\partial x_1}\frac{\partial L}{\partial z}$
x_1
$\frac{\partial z}{\partial x_1}$
f
z
Upstream gradient
$\frac{\partial L}{\partial z}$
Downstream gradient
x_2
$\frac{\partial z}{\partial x_2}$
$\frac{\partial L}{\partial x_2} = \frac{\partial z}{\partial x_2}\frac{\partial L}{\partial z}$

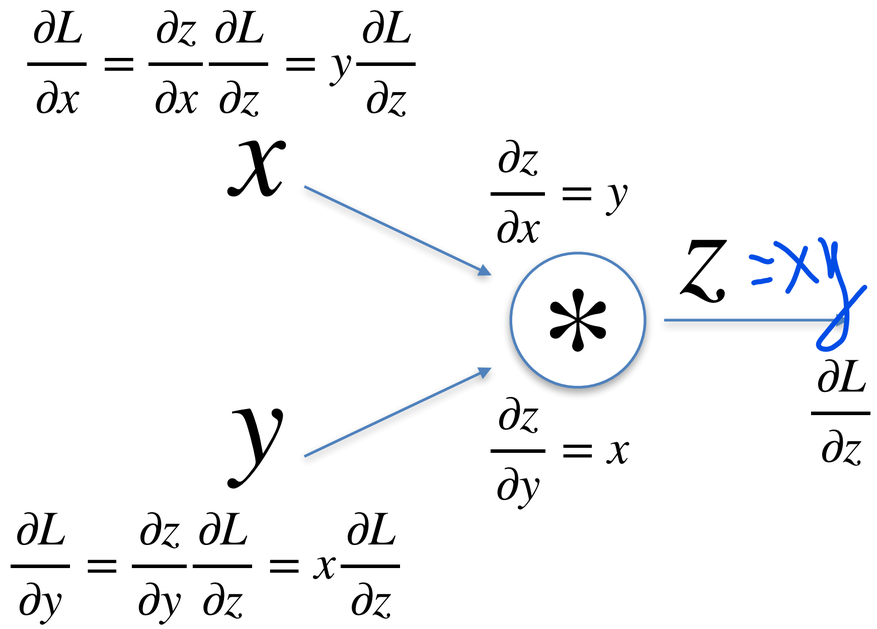
Patterns in Backpropagation: Add Block

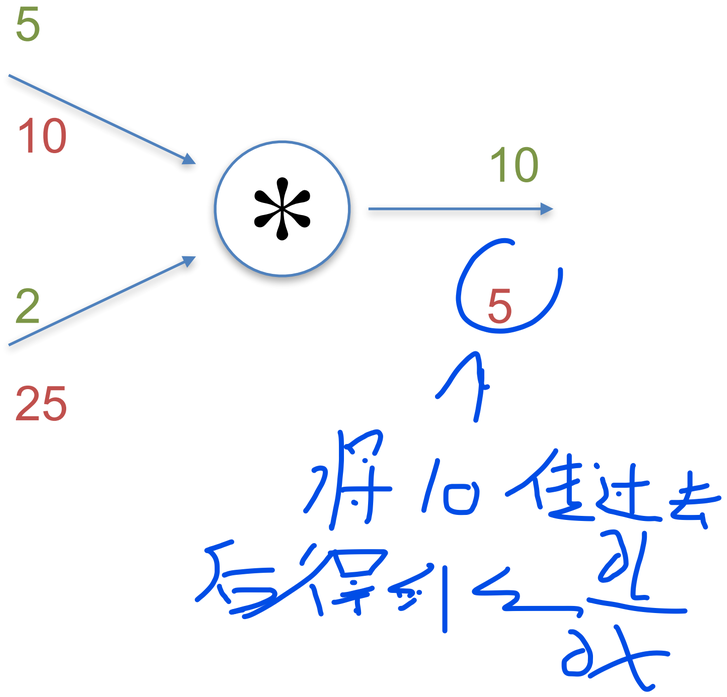
假设 $z = x_1 + x_2$





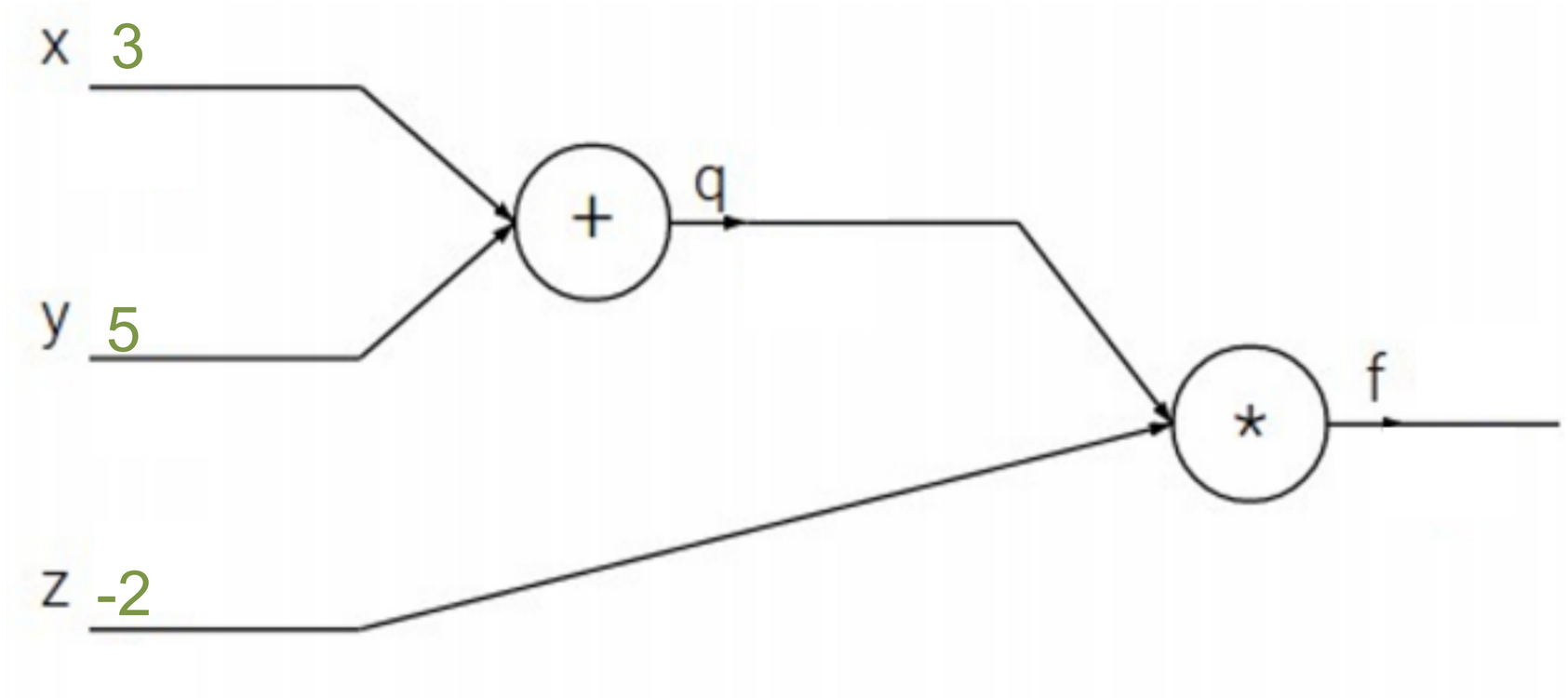
Patterns in Backpropagation: Multiply Block





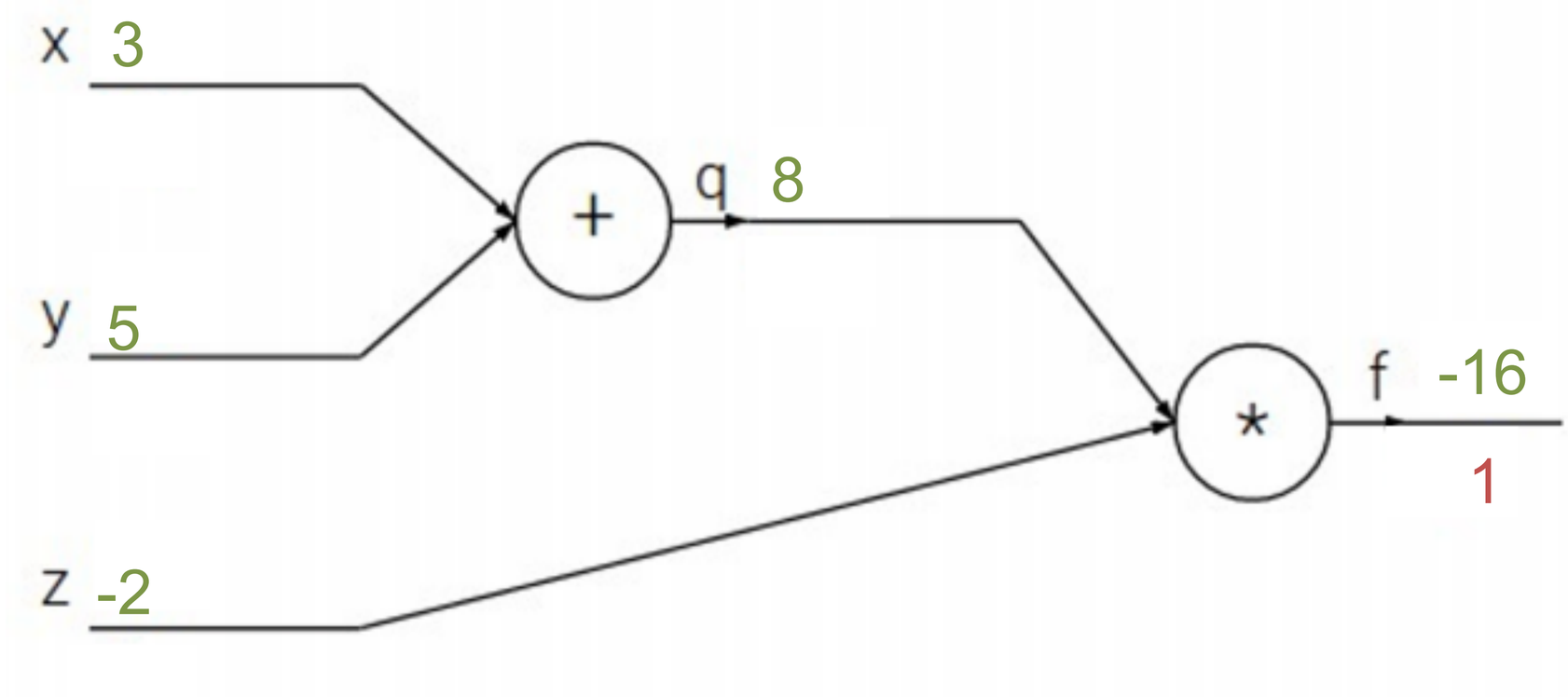
Example 1

$$f(x, y, z) = (x + y)z = q \cdot z$$



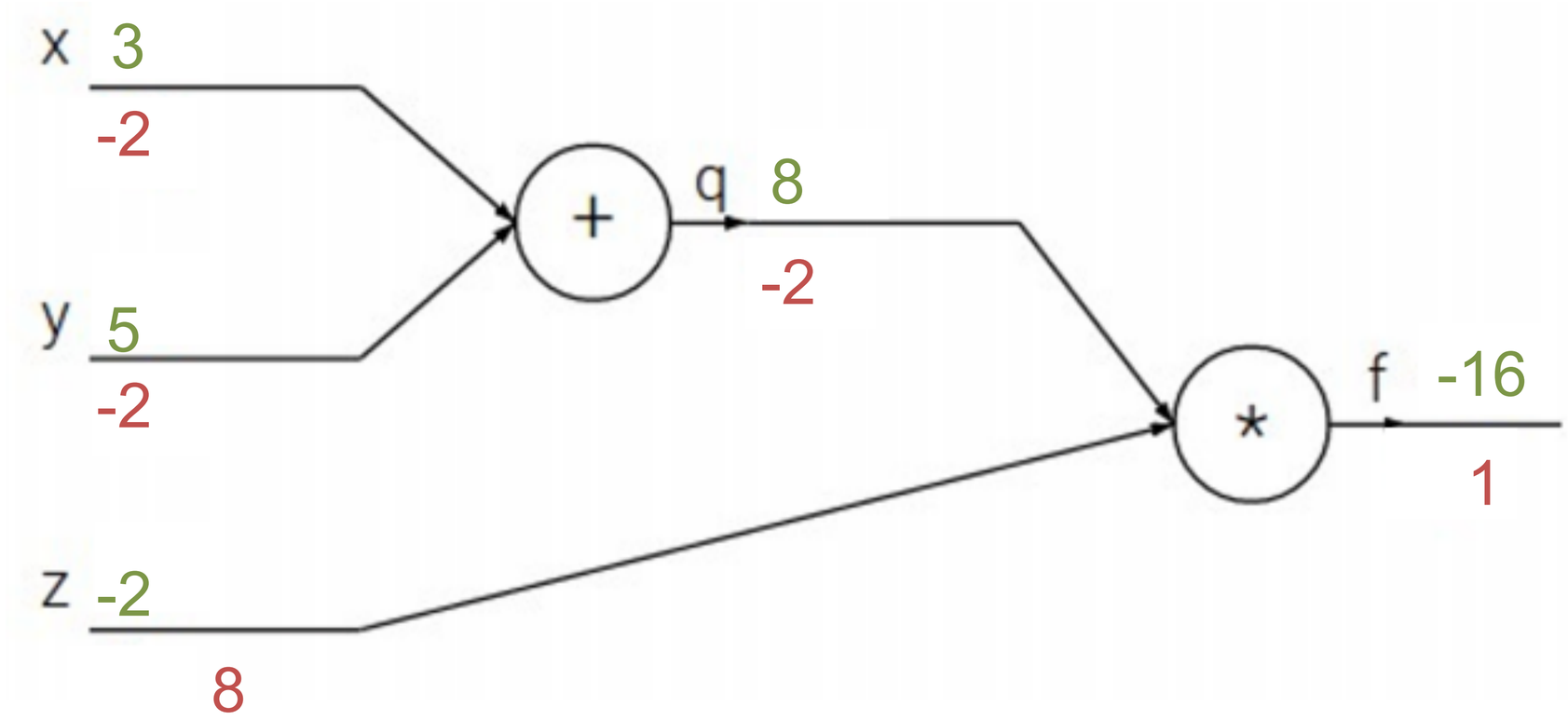
Example 1

$$f(x, y, z) = (x + y)z = q \cdot z$$



Example 1

$$f(x, y, z) = (x + y)z = q \cdot z$$

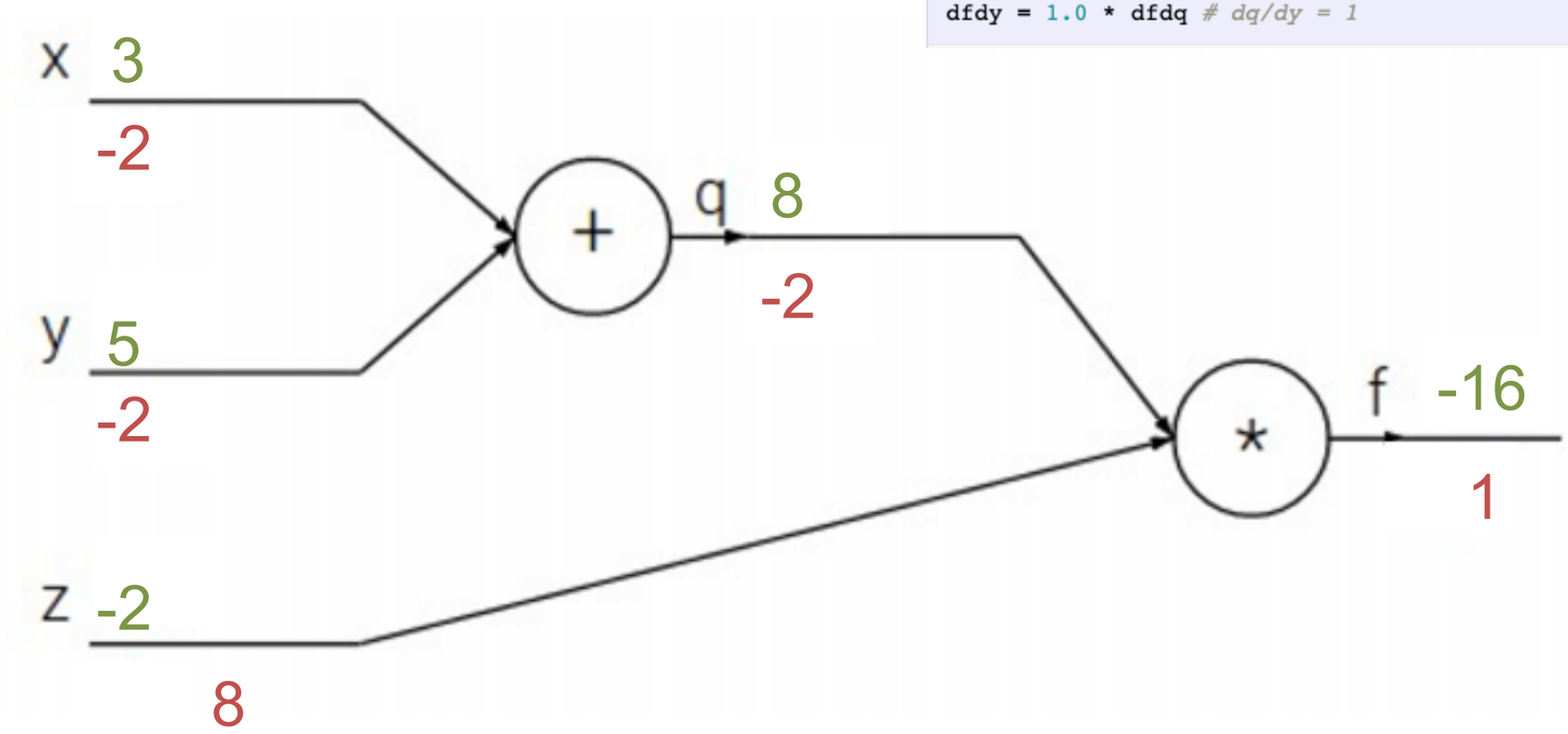


Example 1

$$f(x, y, z) = (x + y)z = q \cdot z$$

```
# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

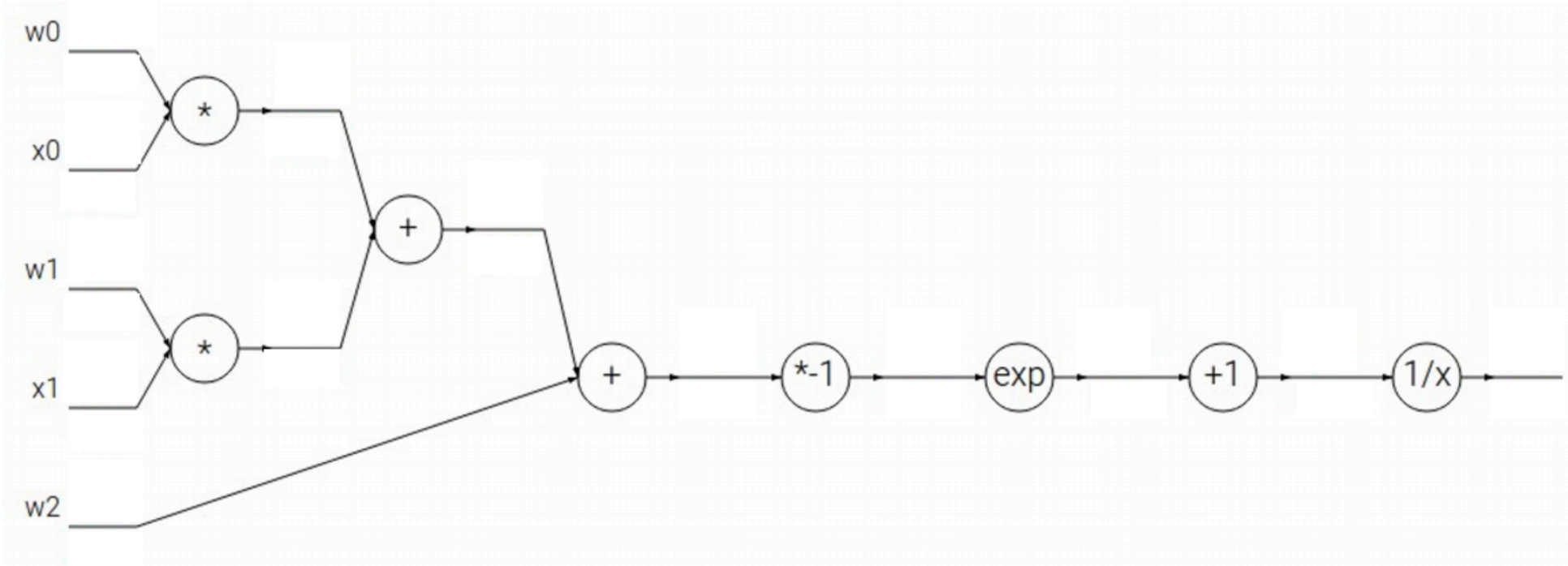


Example 2

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

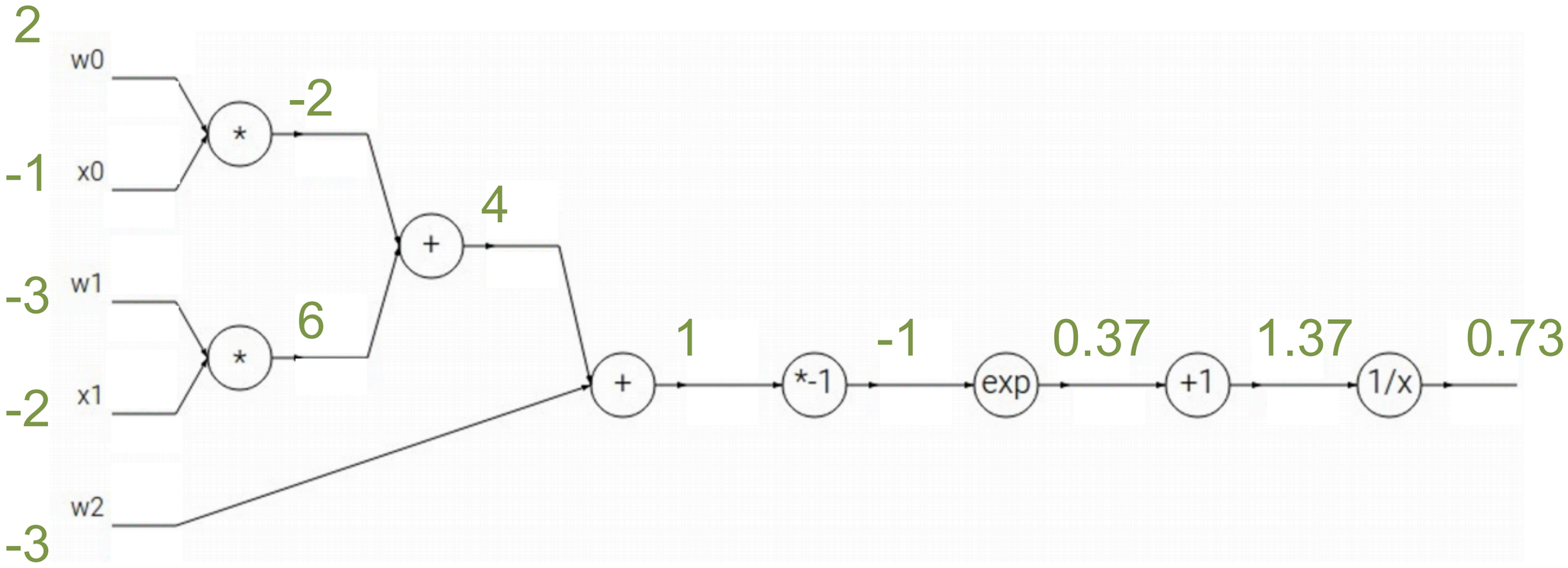
Example 2

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

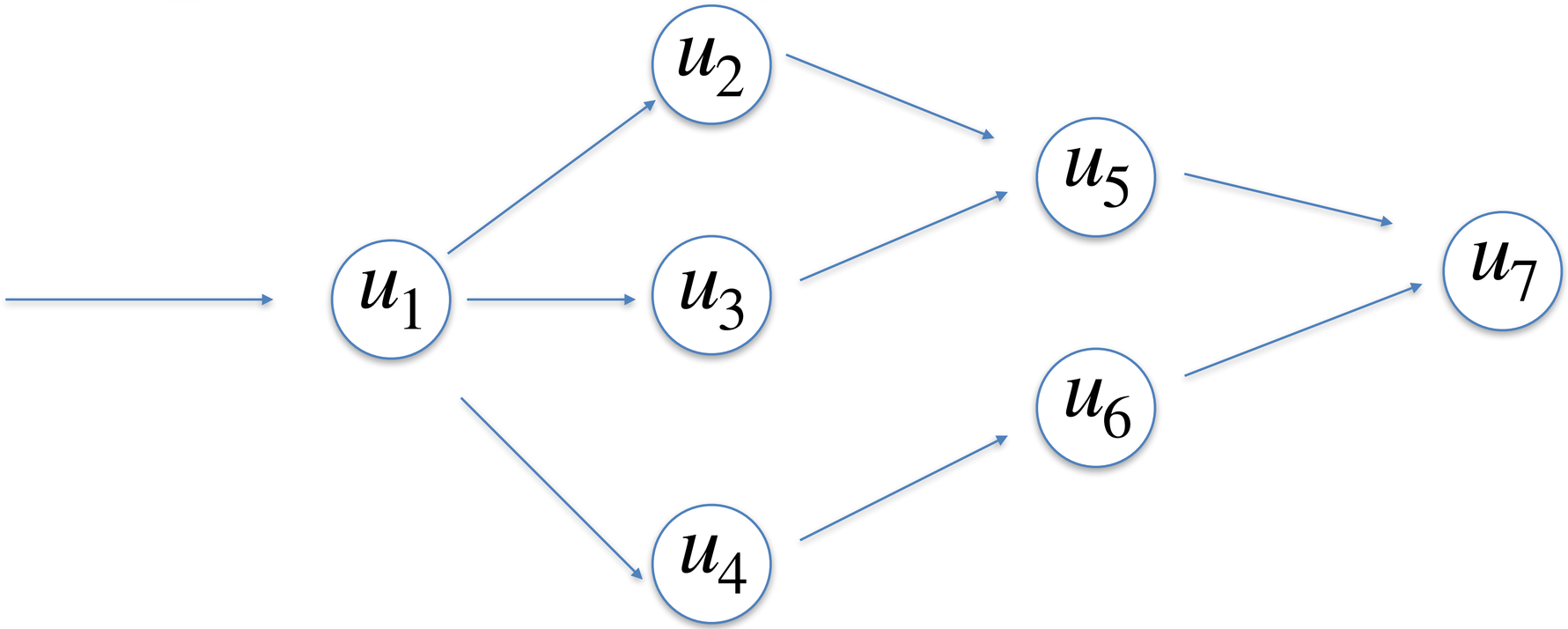


Example 2

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



Topological Sort (for Directed Graph)



For every edge u_{ij}, node u_i should come (be visited) before node u_j.

$\{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$

$\{u_1, u_4, u_2, u_3, u_6, u_5, u_7\}$

Backprop Modular API

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

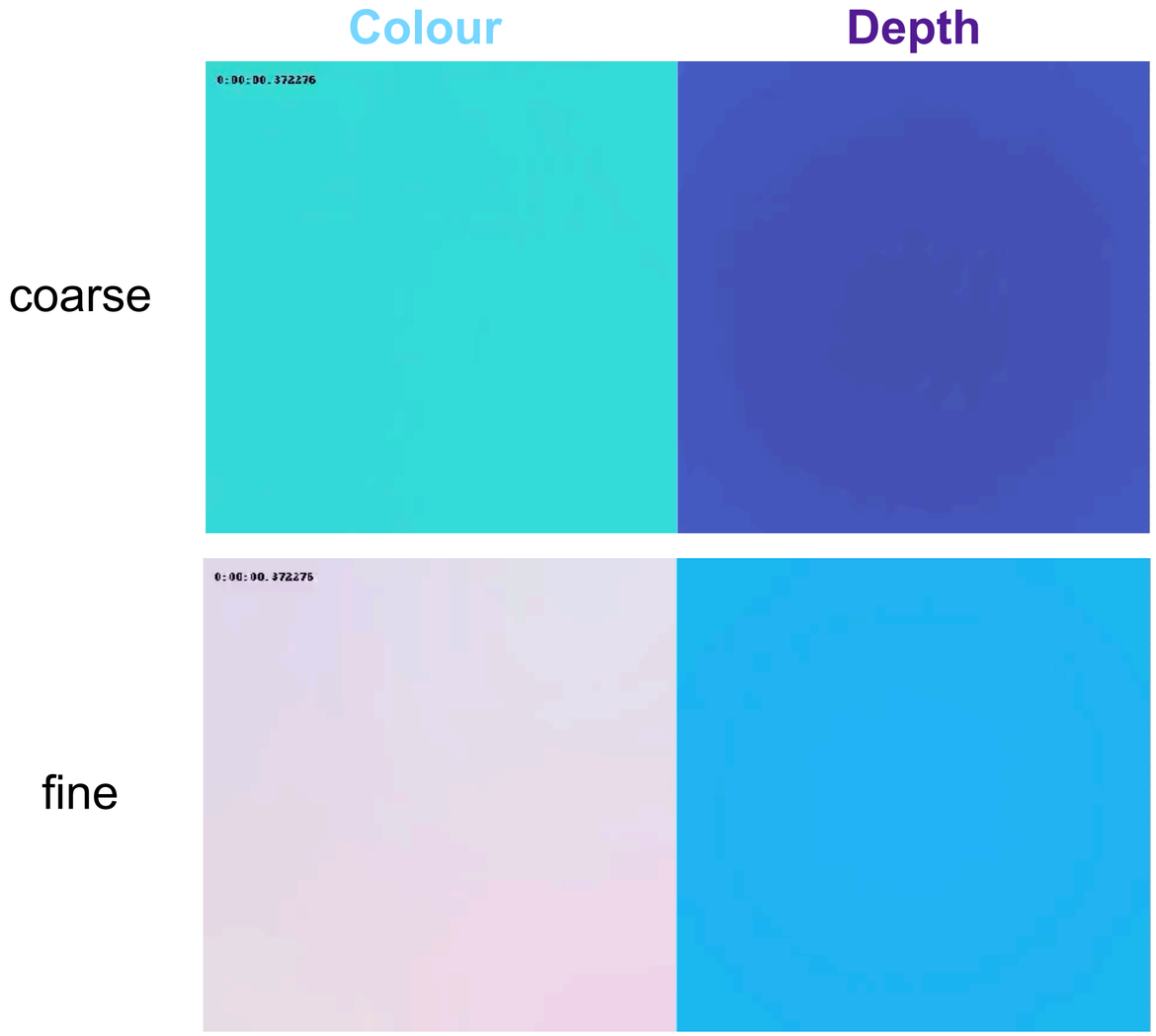
Fast and Realistic Reconstruction

High-quality Novel View Synthesis

Scene: Blender Drums
PSNR: 23.82

Ref: Yen-Chen, Lin. ‘NeRF-pytorch’. *GitHub repository* 2020.

Quality versus Speed



Quality Tradeoff

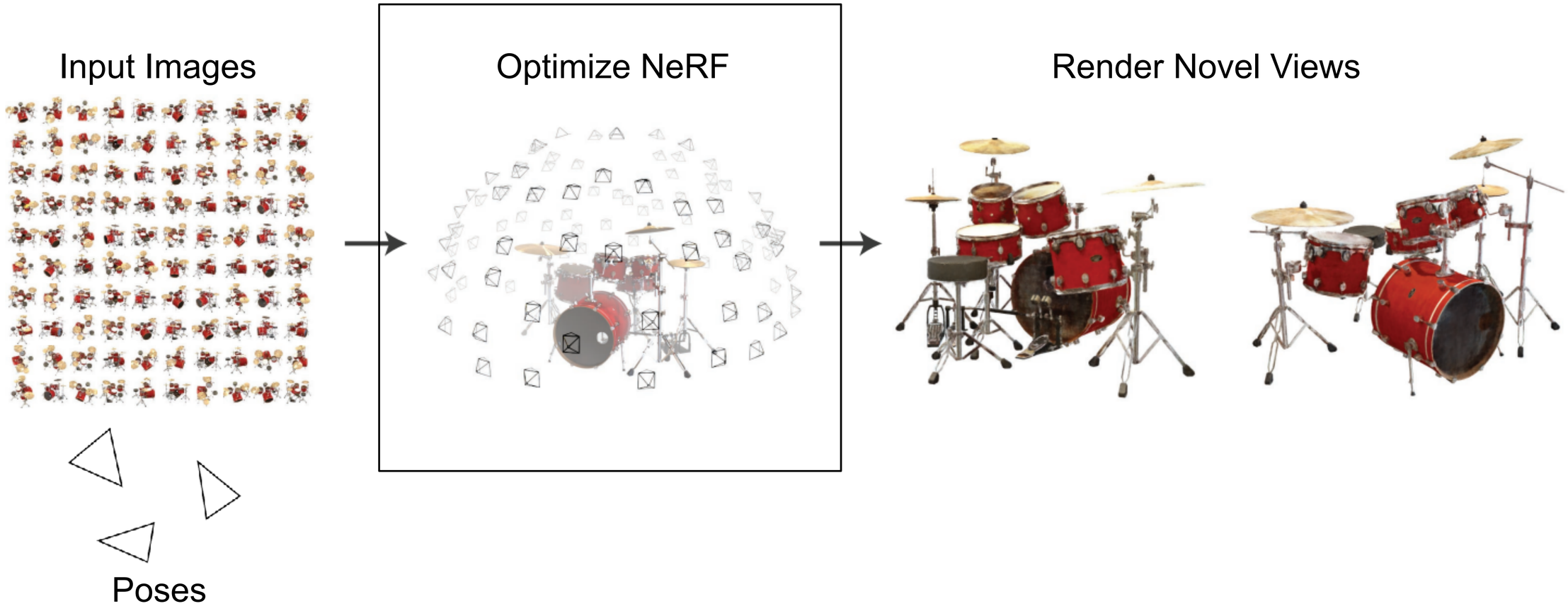
NeRF

Scene: Blender Drums
PSNR: 23.82

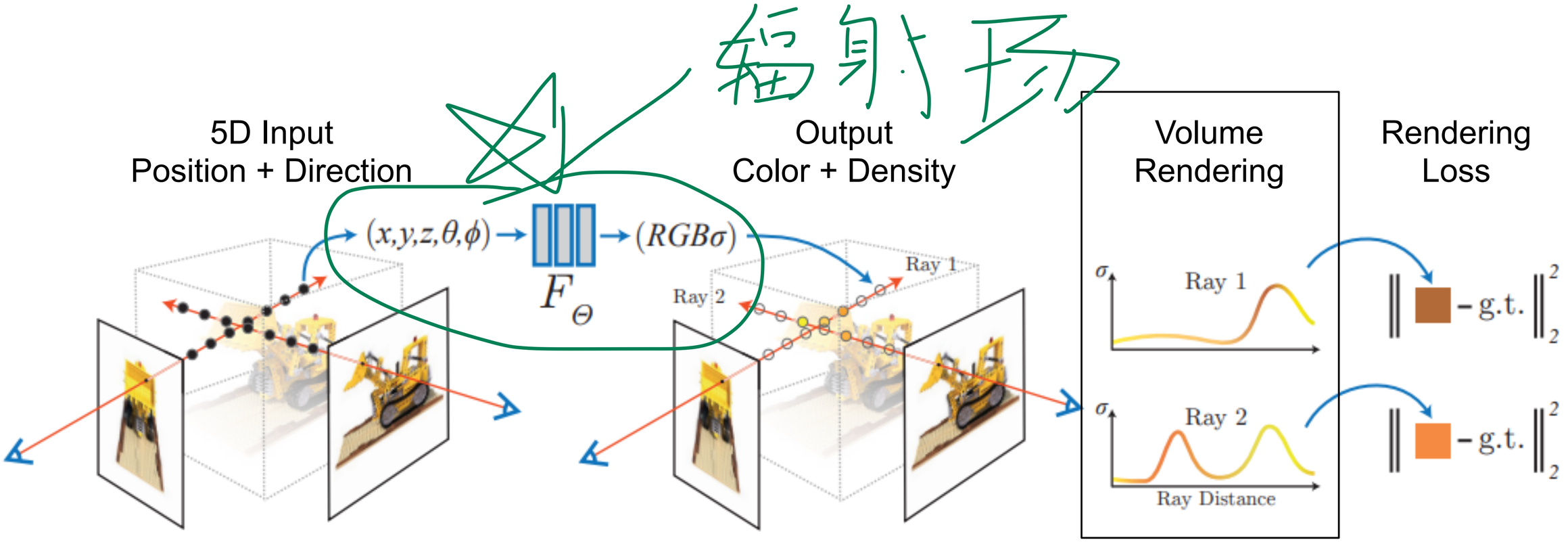
Traditional Grid

Scene: Blender Drums
PSNR: 20.70

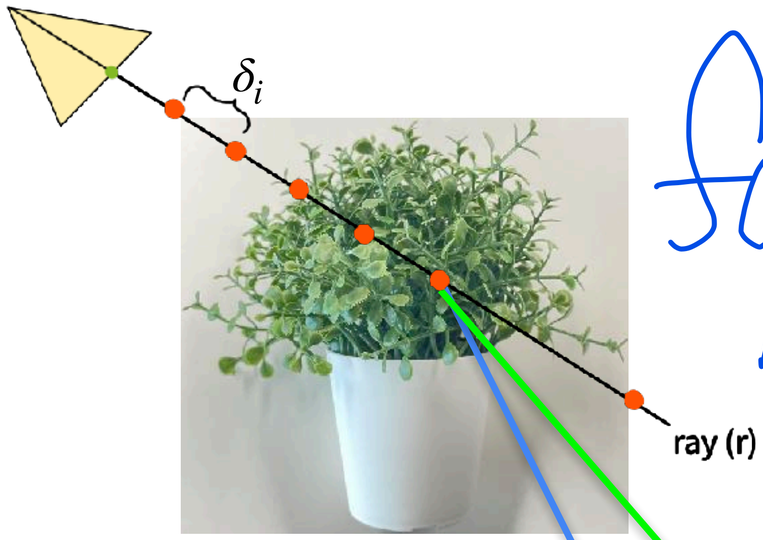
Nerf Formulation



NeRF 3D Representation and Optimization



Differentiable Volumetric Rendering



$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - exp(-\sigma_i\delta_i))c_i$$

where, $T_i = exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$

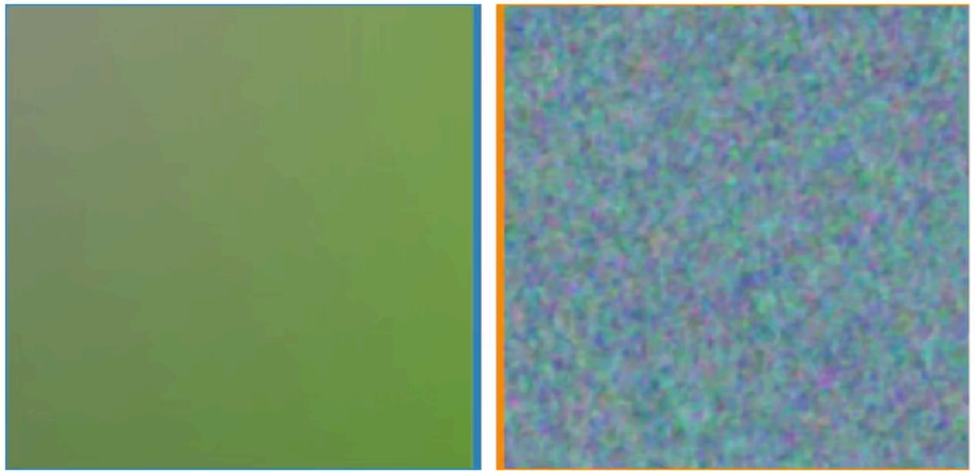
[1] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." European conference on computer vision. Springer, Cham, 2020.

[2] Kajiya, James T., and Brian P. Von Herzen. "Ray tracing volume densities." ACM SIGGRAPH computer graphics 18.3 (1984): 165-174.

Positional Encoding

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

No P.E. With P.E.



Source: 1. “Neural Radiance Fields” https://www.matthewtancik.com/nerf
2. “Fourier Feature Networks” https://bmild.github.io/fourfeat/index.html

SIGGRAPH 2022

ReLU Fields

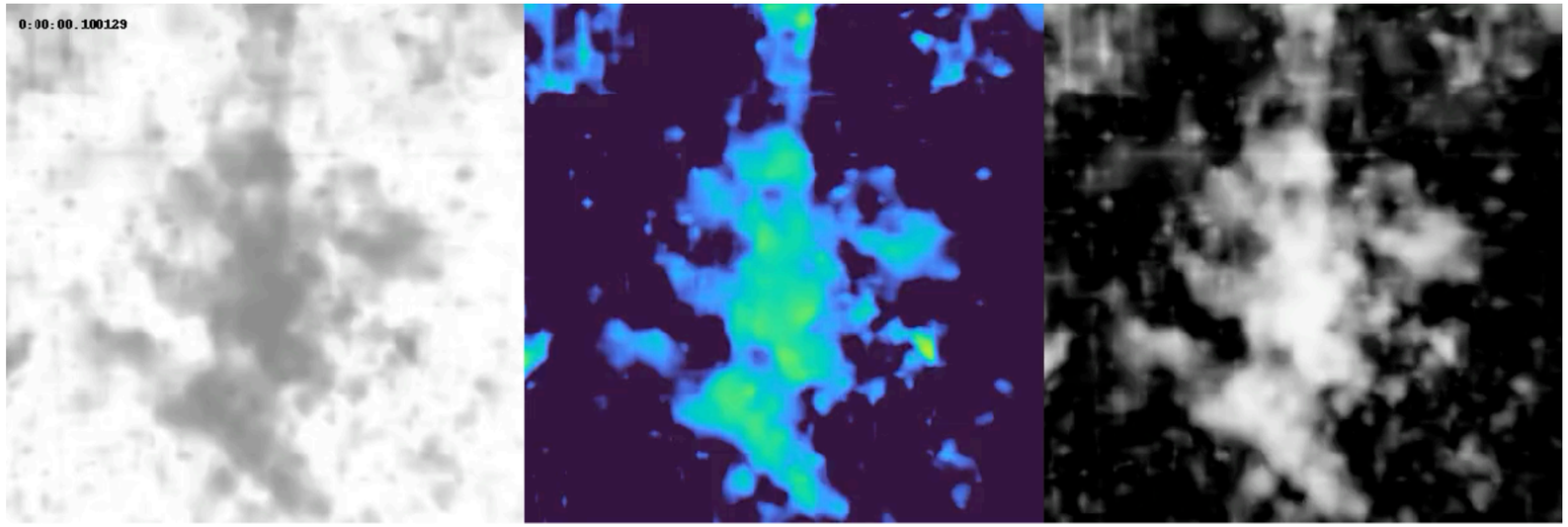
The little non-linearity that could




Animesh Karnewar[1] Tobias Ritschel[1] Oliver Wang[2] Niloy Mitra[2,1]

[1]University College London, [2]Adobe Research

ReLU Fields



Scene: Blender Drums

PSNR: 25.15

Applications

3D occupancy fields (geometry only)

3D Radiance Fields

Grid

PSNR: 28.53

train: 10m:02s
render: 99.1ms

NeRF

PSNR: 33.52

train: 11h:21m
render: 16363.0ms

ReLUField

PSNR: **35.72**

train: 10m:36s
render: 99.5ms

3D Radiance Fields

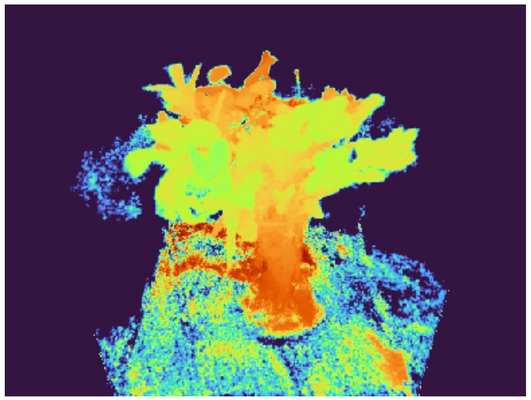
Radiance Fields (Real Capture)

Background

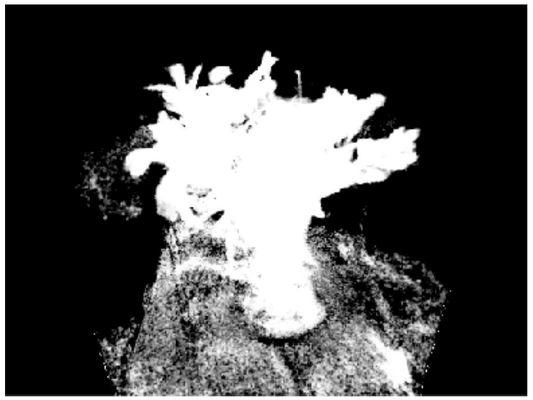
Full

Foreground

Depth

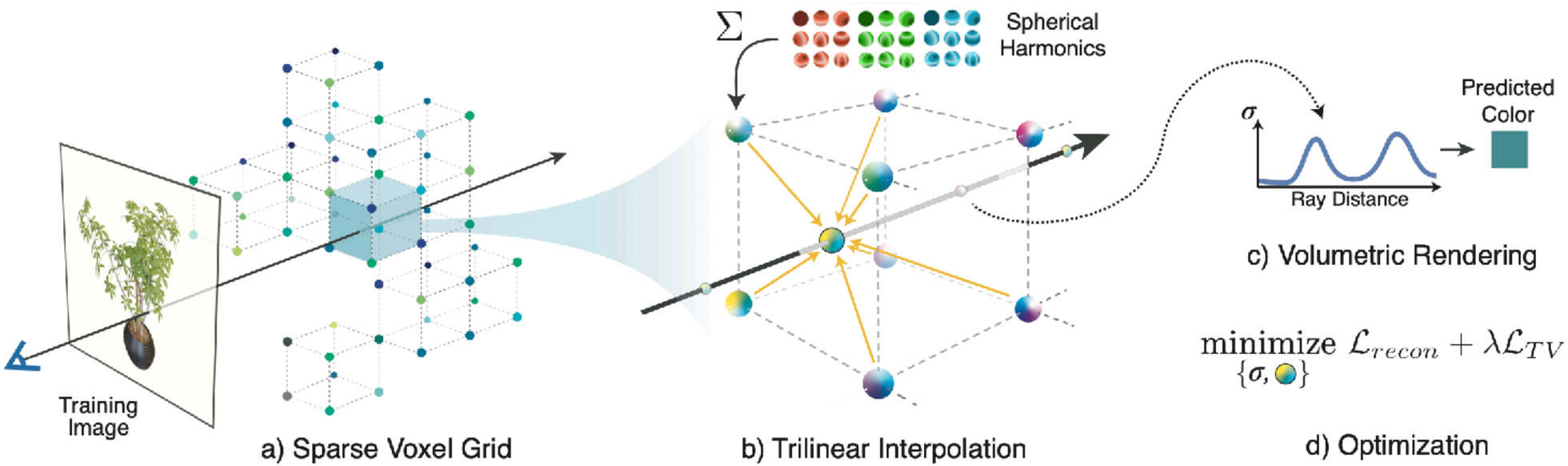


Acc

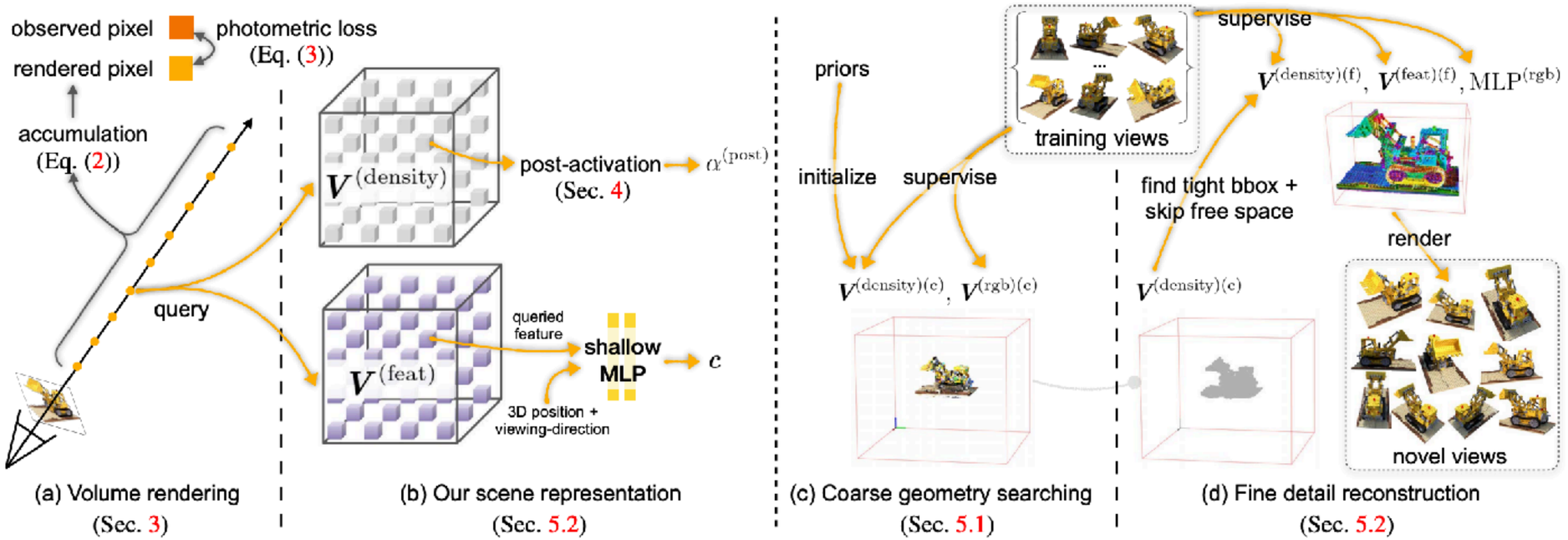


Attal, Benjamin, et al. "MatryODShka: Real-time 6DoF video view synthesis using multi-sphere images." *ECCV*, 2020.

Plenoxels


Training Image
a) Sparse Voxel Grid
Σ
Spherical Harmonics
b) Trilinear Interpolation
σ
Predicted Color
Ray Distance
c) Volumetric Rendering
minimize $\mathcal{L}_{recon} + \lambda \mathcal{L}_{TV}$
$\{\sigma, \}$
d) Optimization

DVGo


observed pixel
rendered pixel
photometric loss
(Eq. (3))
accumulation
(Eq. (2))
query
$V^{(\text{density})}$
post-activation
(Sec. 4)
$\alpha^{(\text{post})}$
$V^{(\text{feat})}$
queried feature
shallow MLP
c
3D position + viewing-direction
(a) Volume rendering
(Sec. 3)
(b) Our scene representation
(Sec. 5.2)
priors
initialize
supervise
training views
supervise
$V^{(\text{density})(\text{f})}$, $V^{(\text{feat})(\text{f})}$, $\text{MLP}^{(\text{rgb})}$
find tight bbox + skip free space
render
$V^{(\text{density})(\text{c})}$, $V^{(\text{rgb})(\text{c})}$
$V^{(\text{density})(\text{c})}$
novel views
(c) Coarse geometry searching
(Sec. 5.1)
(d) Fine detail reconstruction
(Sec. 5.2)