Backpropagation and Automatic Differentiation

EEP 596 Computer Vision: Classical and Deep Methods (Autumn 2025)

UNIVERSITY of WASHINGTON

Stan Birchfield

NVIDIA

1

Recap

- We just looked at ways to minimize the loss function
- Stochastic gradient descent is the most popular (mini-batch gradient descent)
- How to calculate gradients?
- Answer: backpropagation
- That is the focus of this lecture

2

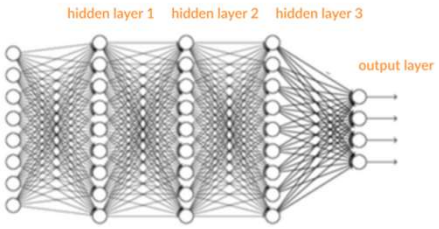
Backpropagation

Nouns

Neural network has layers
Layers have units
Each unit is a perceptron
Each perceptron implements

$\phi(\mathbf{w}^T\mathbf{x} + b)$

where $\phi()$ is nonlinear activation function



Verbs

Pass training data through network, left to right (forward pass)
Compare output to ground truth, yields error
Compute loss
Pass loss backward through network via gradients (backpropagation)
This yields $\Delta\mathbf{w}$ and Δb for each weight and bias
Update each weight and bias according to stochastic gradient descent (SGD):

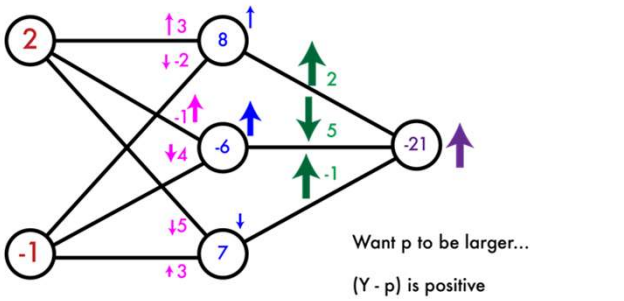
$w_{i+1} \leftarrow w_i - \alpha \cdot \partial L / \partial w$

where L is the loss, α is the learning rate

3

Backpropagation: just taking derivatives

This is the backpropagation algorithm. It's really just an easy way to calculate partial derivatives in a neural network. We forward-propagate information through the network, calculate our error, then backpropagate that error through network to calculate weight updates.



https://danielgordon10.github.io/other/dlclass.html

4

Backpropagation

To train neural network:

1) Define loss function
2) Backpropagate to get gradients
3) Update weights using SGD (which needs the gradients)

What is backpropagation?

Use chain rule to adjust network weights based on loss

Do you remember the chain rule of differentiation?

5

Chain rule

Function $h(x) = (\sin x)^2$

Derivative? $\frac{d}{dx}[h(x)] = ?$

What we know $\frac{d}{dx}[x^2] = 2x$ $\frac{d}{da}[a^2] = 2a$ $\frac{d}{d(foo)}[(foo)^2] = 2(foo)$ $\frac{d}{dx}[\sin x] = \cos x$

Chain rule $\frac{d}{dx}[h(x)] = \frac{d}{d(foo)}[h(x)] \cdot \frac{d}{dx}[(foo)]$

Put it all together

$$\frac{d}{dx}[h(x)] = \frac{d}{dx}[(\sin x)^2]$$ Definition of *h*

$$= \frac{d[(\sin x)^2]}{d(foo)} \cdot \frac{d[(foo)]}{dx}$$ Chain rule

$$= \frac{d[(\sin x)^2]}{d(\sin x)} \cdot \frac{d[(\sin x)]}{dx}$$ Substitute sin *x* for *foo*

$$= 2(\sin x) \cdot (\cos x)$$ Apply known derivatives

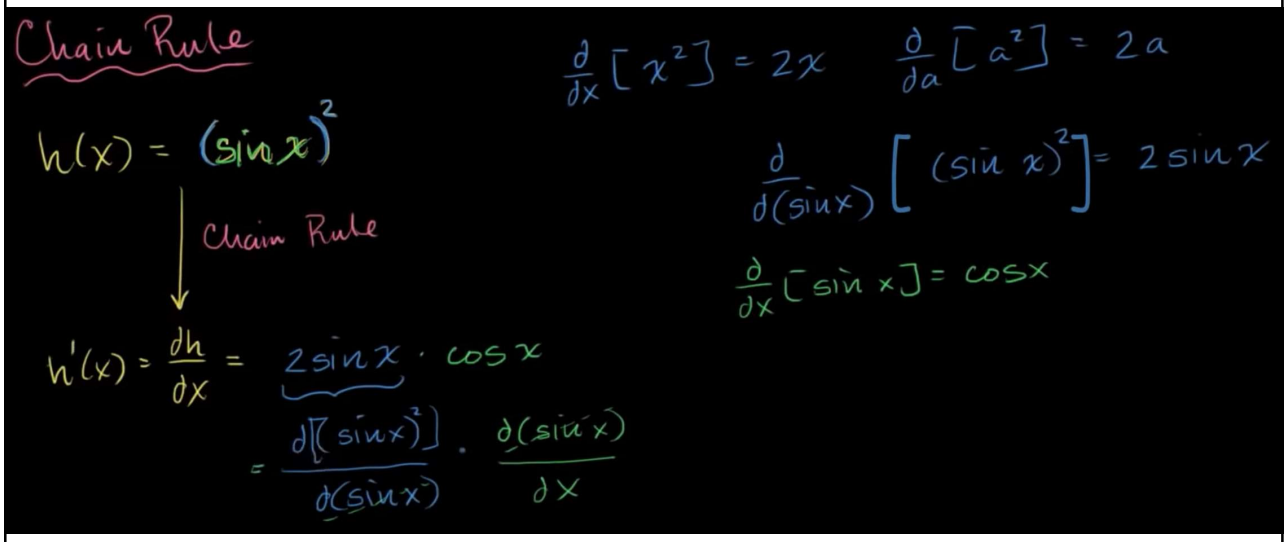
$$= 2 \sin x \cos x$$ Simplify

$$\frac{\partial w}{\partial t} = \sum_i \left(\frac{\partial w}{\partial u_i} \cdot \frac{\partial u_i}{\partial t} \right) = \frac{\partial w}{\partial u_1} \cdot \frac{\partial u_1}{\partial t} + \frac{\partial w}{\partial u_2} \cdot \frac{\partial u_2}{\partial t} + \cdots$$

https://www.khanacademy.org/math/ap-calculus-ab/ab-differentiation-2-new/ab-3-1a/v/chain-rule-introduction

6

Chain rule



https://www.khanacademy.org/matkh/ap-calculus-ab/ab-differentiation-2-new/ab-3-1a/v/chain-rule-introduction

7

Derivative

1D function $$f(x) = 4x$$

Derivative $$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Rearranged $$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Derivative tells us the amount the expression changes due to change in variable

$$\frac{df}{dx} = 4$$

If input *x* changes by 6, output *f(x)* changes by 6*4 = 24

https://cs231n.github.io/optimization-2/

8

Simple backprop

$\hat{y} = f(x) = w * x$
$df / dw = x$
Suppose ground truth labeled data is x=3, y=5
And suppose weight is initially w = 2
Learning rate $\alpha = 1$
Then:

Forward pass: $\hat{y} = f(x) = 2 * 3 = 6$
Error: $\hat{y} - y = 6 - 5 = 1$
Backward pass (backprop): $df / dw = 3$
Update: $\Delta w \approx (\hat{y} - y) / (df / dw) = 1 / 3 = 0.333$
$w \leftarrow w - \alpha \Delta w = 2 - 0.333 = 1.667$

Double-check: Next iteration, with updated weight,

Forward pass: $\hat{y} = 1.667 * 3 = 5$
Error: $\hat{y} - y = 5 - 5 = 0$

Derivation:

$df / dw \approx \Delta f / \Delta w$
$\Delta w \approx \Delta f / (df / dw)$
$\approx (\hat{y} - y) / (df / dw)$

We will focus primarily upon this step

Error minimized in one step!
(usually many iterations in practice)

9

Gradient

2D function $f(x, y) = xy$

Don't be confused by notation: Here x and y are both inputs

Partial derivatives $f(x, y) = xy \rightarrow \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$

At a certain point (x,y) $x = 4, y = -3$

$f(x, y) = -12 \quad \frac{\partial f}{\partial x} = -3 \quad \frac{\partial f}{\partial y} = 4$

Gradient $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$

https://cs231n.github.io/optimization-2/

10

Other examples

x and y are still inputs

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

indicator function

$$f(x,y) = \max(x,y) \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1(x \geq y) \qquad \frac{\partial f}{\partial y} = 1(y \geq x)$$

$$f(x) = \max(x,0) \quad \rightarrow \quad \frac{\partial f}{\partial x} = \begin{cases} 1 & x \geq 0 \\ 0 & \text{o.w.} \end{cases}$$

https://cs231n.github.io/optimization-2/

11

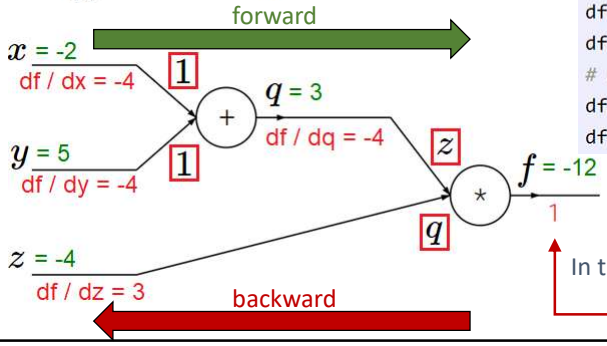
Chain rule

$$f(x,y,z) = (x+y)z$$

$$f = qz \quad q = x + y \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial q}{\partial x} = 1$$

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial q}{\partial y} = 1$$




```
# set some inputs
x = -2; y = 5; z = -4

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

# perform the backward pass (backprop) 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
dfdy = 1.0 * dfdq # dq/dy = 1    here is the chain rule!
```

The same math works for any input (x,y,z)
In practice, we usually only care about the weights
In this simple example, no distinction is made among inputs

Set this to 1

https://cs231n.github.io/optimization-2/

12

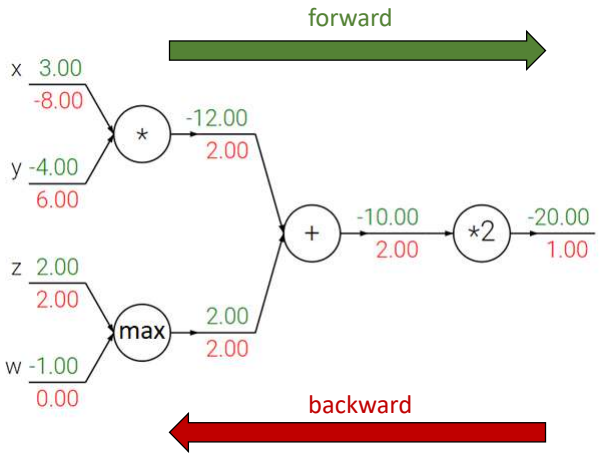
Common gates

Visualize with circuit diagram

Each "gate" represents computation

3 common gates:

- **add** – distributes gradient equally to all inputs
- **max** – distributes gradient to exactly one input
- **multiply** – gradient is multiplied by inputs (switched)



https://cs231n.github.io/optimization-2/

13

Example: Sigmoid

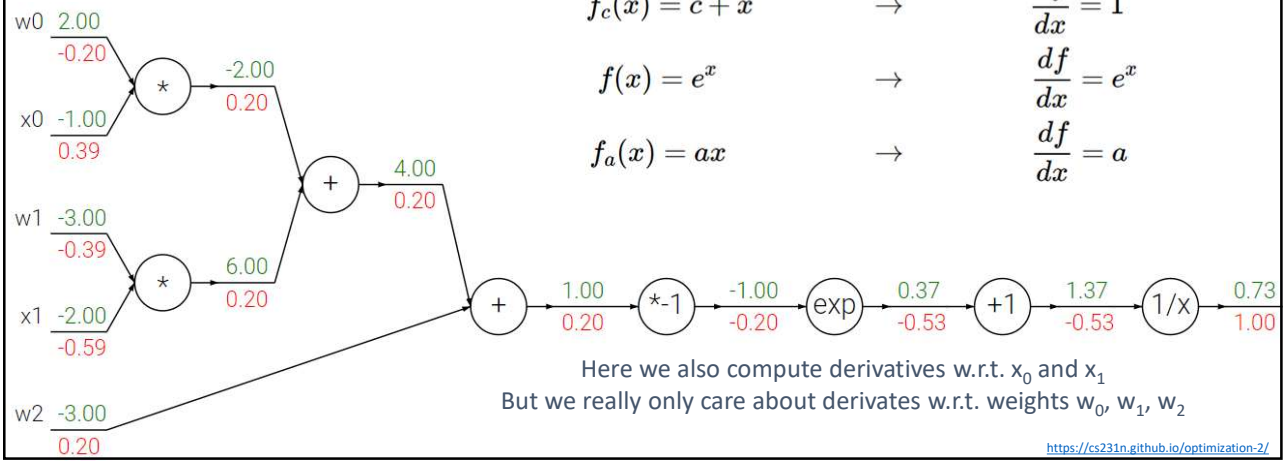
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$



Here we also compute derivatives w.r.t. x_0 and x_1
But we really only care about derivates w.r.t. weights w_0, w_1, w_2

https://cs231n.github.io/optimization-2/

14

Example: Sigmoid

Handy simplification:

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

$$\rightarrow \quad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

```
w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the sigmoid gradient derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the circuit
```

We only care about this

https://cs231n.github.io/optimization-2/

15

Backprop for matrix-matrix multiplication

$$\mathbf{D} = \mathbf{WX}$$

Perturb:

$$(\mathbf{W}+h)\mathbf{X} = \begin{bmatrix} w_{11}+h & w_{12}+h & \cdots & w_{1p}+h \\ w_{21}+h & w_{22}+h & & w_{2p}+h \\ & \vdots & & \\ w_{m1}+h & w_{m2}+h & \cdots & w_{mp}+h \end{bmatrix} \mathbf{X} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & & w_{2p} \\ & \vdots & & \\ w_{m1} & w_{m2} & \cdots & w_{mp} \end{bmatrix} \mathbf{X} + h\mathbf{X} = \mathbf{WX} + h\mathbf{X}$$

Differentiate:

$$\frac{\partial \mathbf{D}}{\partial \mathbf{W}} = \lim_{h\to 0} \frac{(\mathbf{W}+h)\mathbf{X} - \mathbf{WX}}{h} = \lim_{h\to 0} \frac{\mathbf{WX} + h\mathbf{X} - \mathbf{WX}}{h} = \lim_{h\to 0} \frac{h\mathbf{X}}{h} = \mathbf{X} \qquad \frac{\partial \mathbf{D}}{\partial \mathbf{X}} = \mathbf{W}$$

Treat as finite deltas:

$$\partial \mathbf{D}_{\{m\times n\}} = \mathbf{W}_{\{m\times p\}} \partial \mathbf{X}_{\{p\times n\}}$$

$$\partial \mathbf{D}_{\{m\times n\}} = \partial \mathbf{W}_{\{m\times p\}} \mathbf{X}_{\{p\times n\}}$$

Use dimensions to rearrange:

$$\partial \mathbf{W}_{\{m\times p\}} = \partial \mathbf{D}_{\{m\times n\}} \mathbf{X}^{\mathsf{T}}_{\{p\times n\}}$$

$$\partial \mathbf{X}_{\{p\times n\}} = \mathbf{W}^{\mathsf{T}}_{\{m\times p\}} \partial \mathbf{D}_{\{m\times n\}}$$

```
# forward pass
W = np.random.randn(5, 10)        m x p
X = np.random.randn(10, 3)        p x n
D = W.dot(X)                      m x n

# now suppose we had the gradient on D from above in the circuit
dD = np.random.randn(*D.shape) # same shape as D          m x n
dW = dD.dot(X.T) #.T gives the transpose of the matrix    m x p
dX = W.T.dot(dD)                                          p x n
```

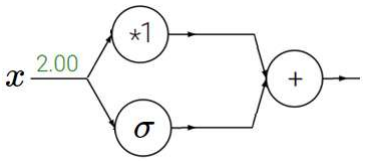
https://cs231n.github.io/optimization-2/

16

Exercise

Analyze this function

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



How to handle the fork?

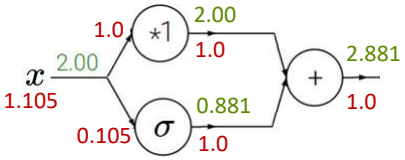
https://cs231n.github.io/optimization-2/

17

Answer

Analyze this function

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

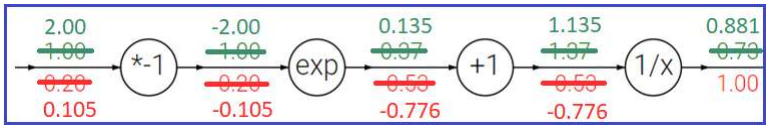


Answer = ∂f/ ∂x = 1.105

σ(2) = 1/(1+exp(-2)) = 0.881

σ(2) · (1- σ(2)) = 0.881 · (1-0.881) = 0.105

double-check:



https://cs231n.github.io/optimization-2/

18

Auto differentiation

Chain rule – how to decompose the derivative of complex function into derivatives of simple functions

Automatic differentiation (AD) – how to apply the chain rule computationally

- Forward-mode AD – compute derivative wrt fixed input
- Reverse-mode AD – compute derivative of fixed output

Backprop – how to apply reverse-mode AD to graph arising from neural network

- With scalar notation – simpler to explain and follow intuitively
- With matrix-vector notation – easier to keep track of notation

Stochastic Gradient Descent (SGD) –use backprop to update weights of neural network

19

Auto differentiation

Forward-mode AD

How to evaluate expression? $z = x \cdot y + \sin(x)$

Easy. Write code.

```
z = x * y + sin(x)
```

How to evaluate derivatives? $\frac{\partial z}{\partial x} = ?$ $\frac{\partial z}{\partial y} = ?$

Break it down

$$a = x \cdot y$$
$$b = \sin(x)$$
$$z = a + b$$

Known derivatives

$$\frac{\partial a}{\partial x} = y \qquad \frac{\partial b}{\partial x} = \cos(x)$$

Apply chain rule

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial z}{\partial b}\frac{\partial b}{\partial x} = 1\,y + 1\cos(x)$$

```
da = y
db = cos(x)
dz = da + db
```

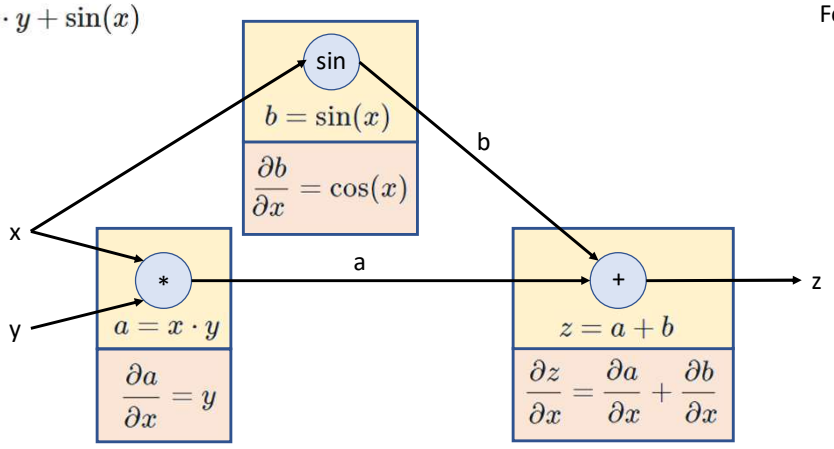
https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation

20

Auto differentiation

$z = x \cdot y + \sin(x)$

Forward-mode AD



Computation graph

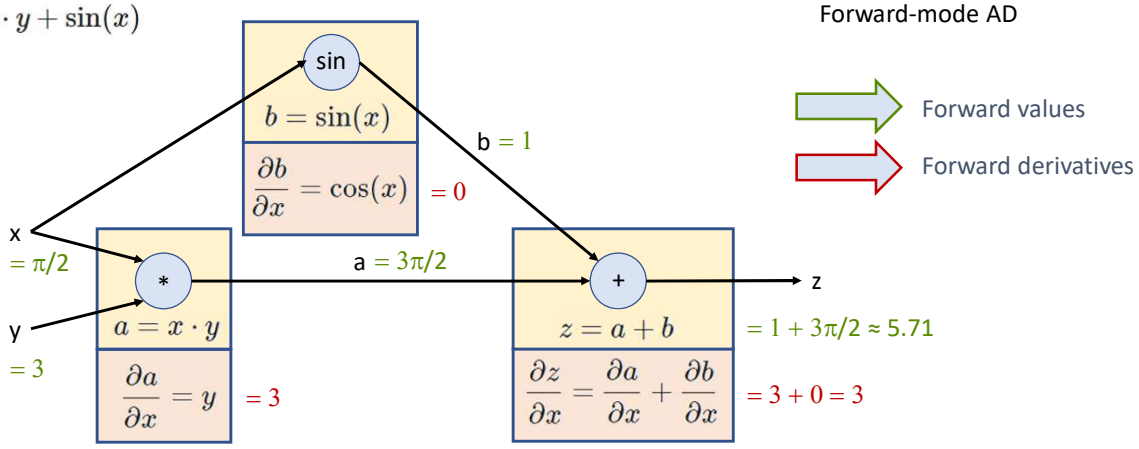
All local computations!

21

Auto differentiation

$z = x \cdot y + \sin(x)$

Forward-mode AD



Computation graph

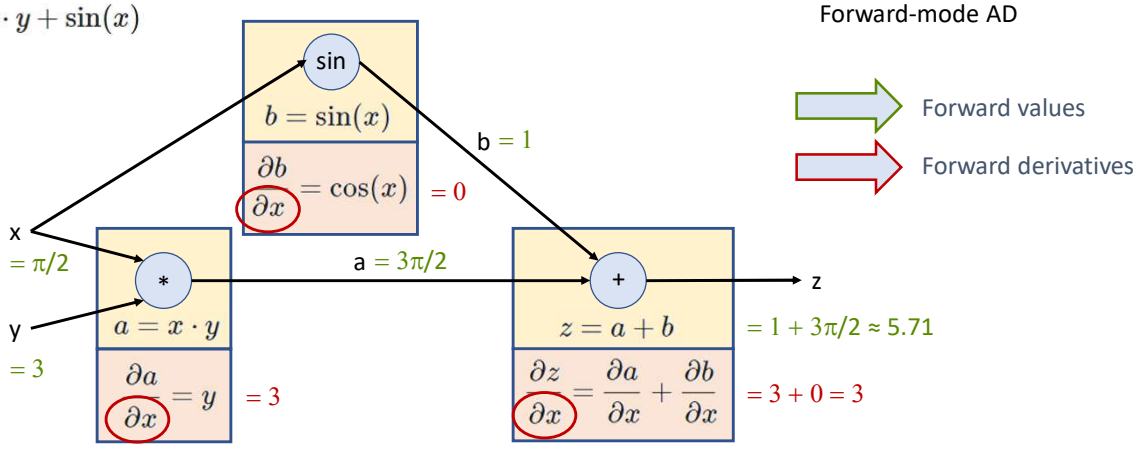
All local computations!

22

Auto differentiation

$z = x \cdot y + \sin(x)$

Forward-mode AD



Each node in computation graph
stores value associated with independent variable *x*

Need to repeat for all independent variables! (time consuming)

23

Auto differentiation

Forward-mode AD

How to evaluate expression? $z = x \cdot y + \sin(x)$

Easy. Write code.

```
z = x * y + sin(x)
```

How to evaluate derivatives? $\frac{\partial z}{\partial x} = ?$ $\frac{\partial z}{\partial y} = ?$

Break it down

$a = x \cdot y$
$b = \sin(x)$
$z = a + b$

Known derivatives

$\frac{\partial a}{\partial x} = y$ $\frac{\partial b}{\partial x} = \cos(x)$

Apply chain rule

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial z}{\partial b}\frac{\partial b}{\partial x} = 1\,y + 1\cos(x)$$

```
da = y
db = cos(x)
dz = da + db
```

https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation

24

Auto differentiation

Reverse-mode AD

How to evaluate expression? $z = x \cdot y + \sin(x)$

Easy. Write code.

```
z = x * y + sin(x)
```

How to evaluate derivatives? $\frac{\partial z}{\partial x} = ?$ $\frac{\partial z}{\partial y} = ?$

Break it down

$$a = x \cdot y$$
$$b = \sin(x)$$
$$z = a + b$$

Known derivatives

$\frac{\partial a}{\partial x} = y$ $\frac{\partial b}{\partial x} = \cos(x)$

Apply chain rule

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial z}{\partial b}\frac{\partial b}{\partial x} = 1\,y + 1\cos(x)$$

```
gb = 1
ga = 1
gx = y * ga + cos(x) * gb
```

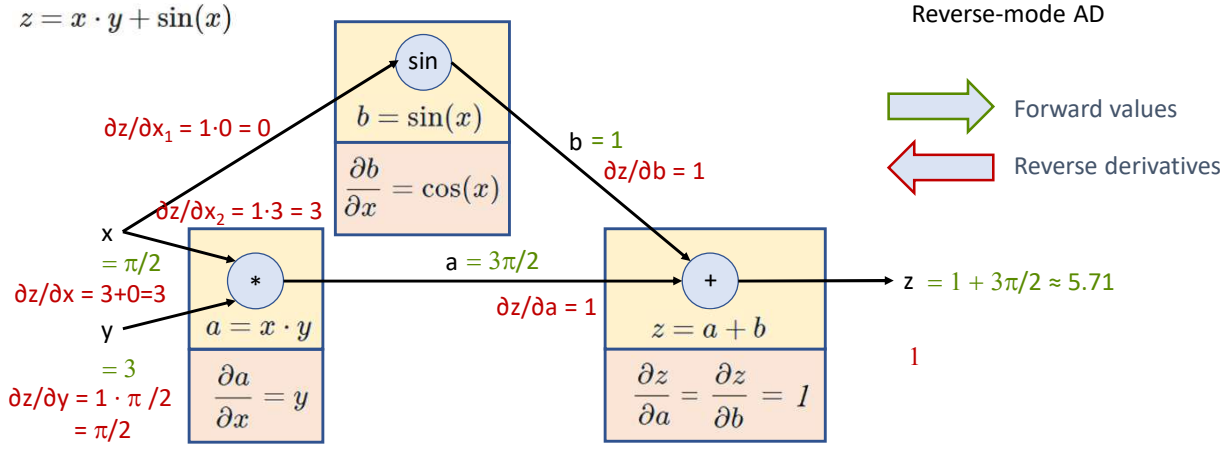
https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation

25

Auto differentiation

Reverse-mode AD

$z = x \cdot y + \sin(x)$

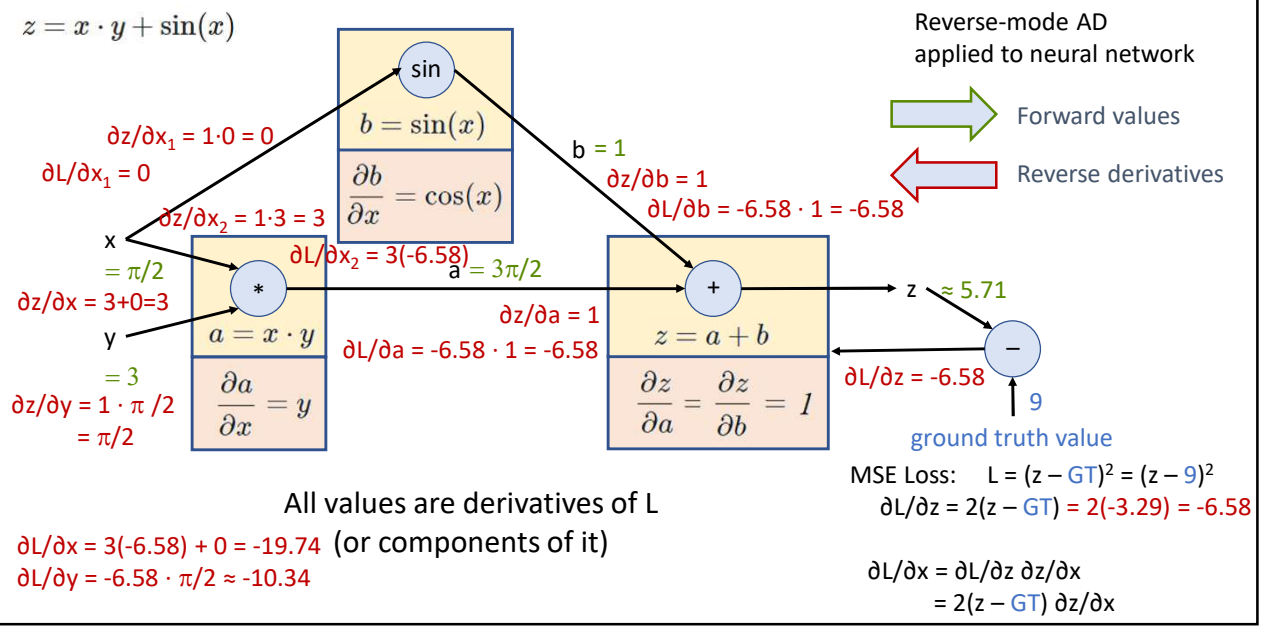


Each node in computation graph
stores value associated with dependent variable *z*

No need to repeat!

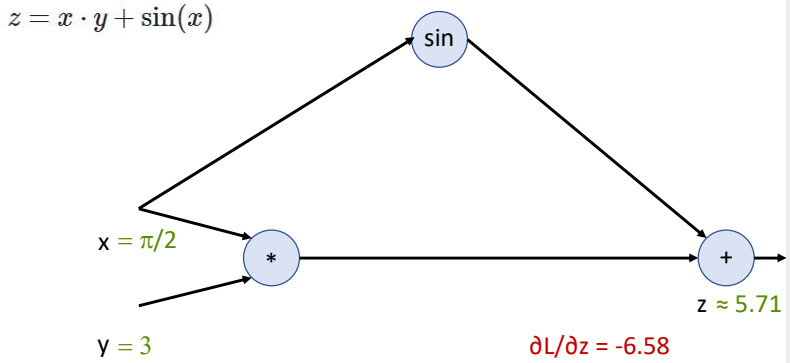
26

Backprop



27

Backprop

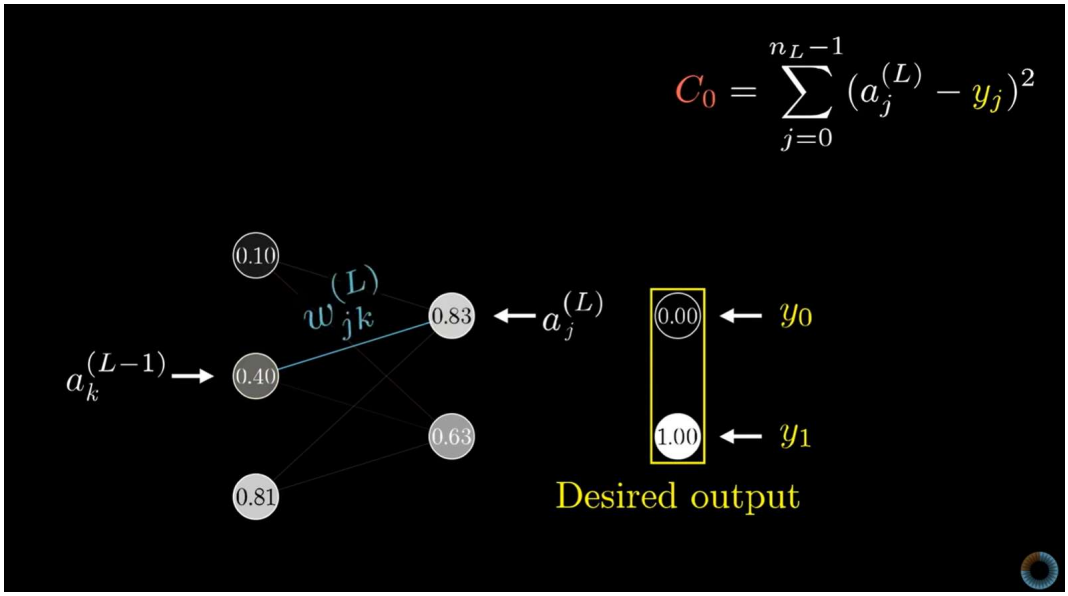



```
import torch
import numpy as np
x = torch.tensor(np.pi/2, requires_grad=True)
y = torch.tensor(3., requires_grad=True)
z = x*y + torch.sin(x)
z_gt = torch.tensor(9., requires_grad=True)
error = (z - z_gt)**2
print(x)    #tensor(1.5708, requires_grad=True)
print(y)    #tensor(3., requires_grad=True)
print(z)    #tensor(5.7124, grad_fn=<AddBackward0>)
print(error) #tensor(10.8084, grad_fn=<PowBackward0>)

z.retain_grad() #explicitly retain non-leaf grad
error.retain_grad()

error.backward()

print(x.grad) #tensor(-19.7257) = dL/dx
print(y.grad) #tensor(-10.3283) = dL/dy
print(z.grad) #tensor(-6.5752) = dL/dz
print(error.grad) #tensor(1.)

print(error.grad_fn) #<PowBackward0 object at 0x..>
print(z.grad_fn) #<AddBackward0 object at 0x..>
print(z.grad_fn.next[0][0]) #<MulBackward0 object at 0x..>
print(z.grad_fn.next[1][0]) #<SinBackward object at 0x..>
print(z.grad_fn.next[0][0].next[0][0]) #<AccumulateGrad object at 0x..>
print(z.grad_fn.next[1][0].next[0][0]) #<AccumulateGrad object at 0x..>

next = next_functions
```

∂L/∂x = 3(-6.58) + 0 = -19.74
∂L/∂y = -6.58 · π/2 ≈ -10.34

28

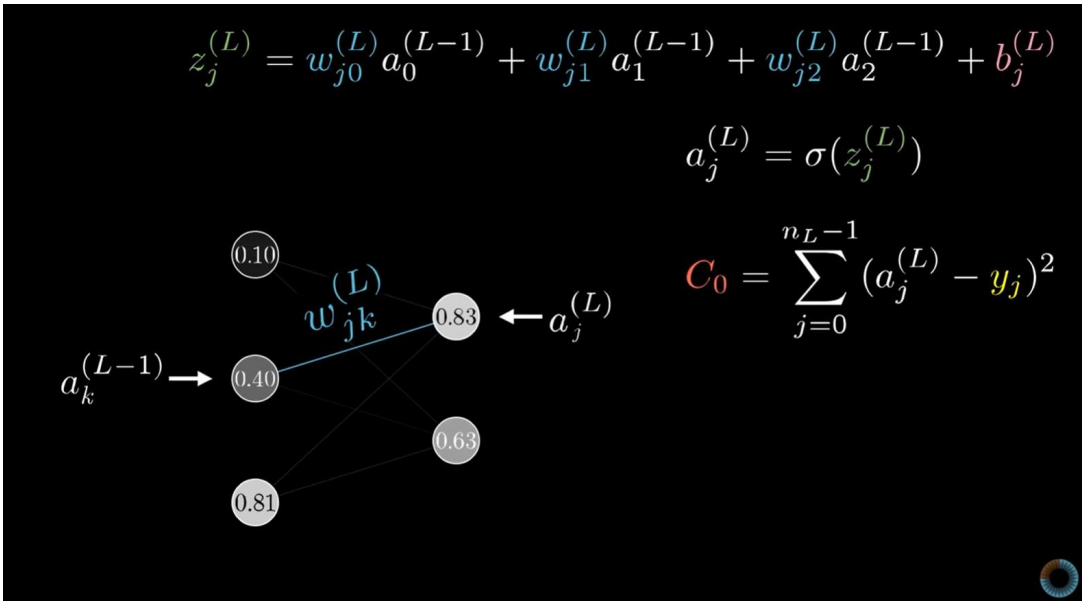
Backprop



3Blue1Brown, Backpropagation calculus: https://youtu.be/tIeHLnjs5U8

29

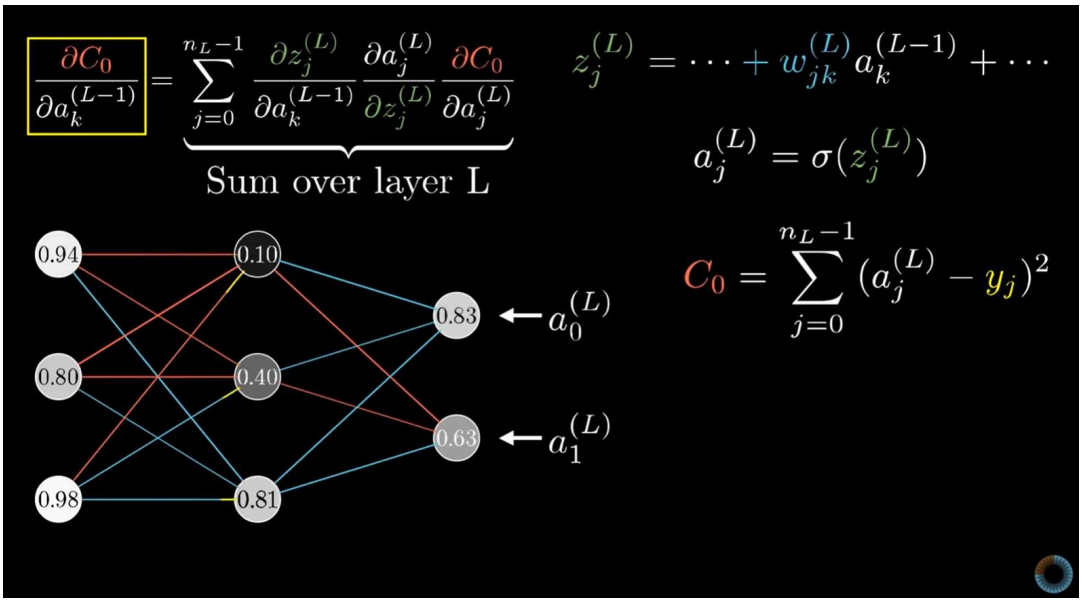
Backprop



3Blue1Brown, Backpropagation calculus: https://youtu.be/tIeHLnjs5U8

30

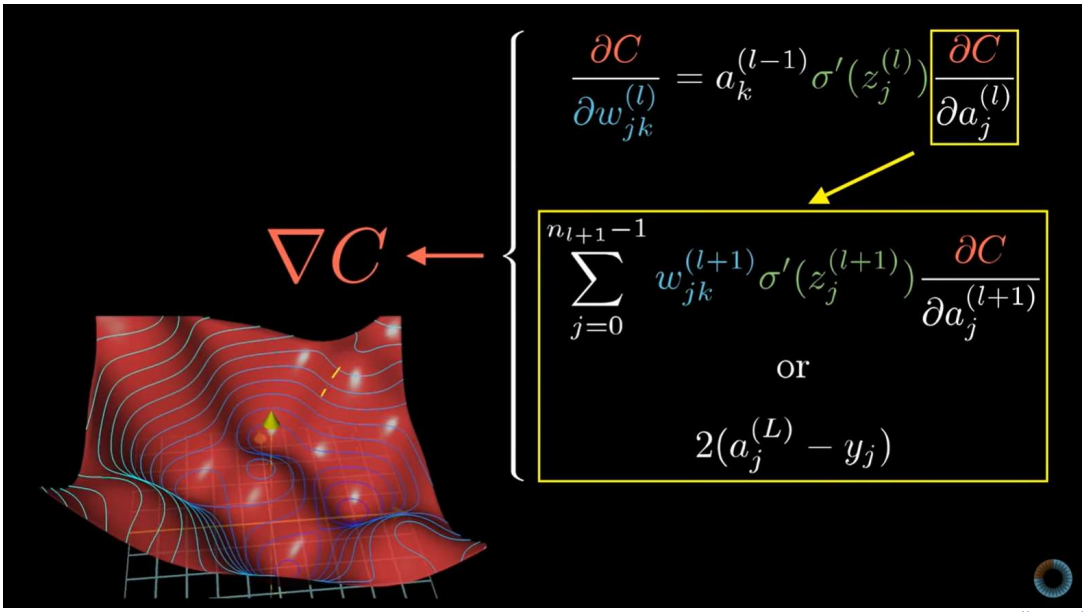
Backprop



3Blue1Brown, Backpropagation calculus: https://youtu.be/tIeHLnjs5U8

31

Backprop



3Blue1Brown, Backpropagation calculus: https://youtu.be/tIeHLnjs5U8

32

Autonomous drone trail following

Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness

Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, Stan Birchfield

- uses ResNet-18
- uses shifted ReLU (sReLU)
- no batch norm
- outputs orientation and lateral offset

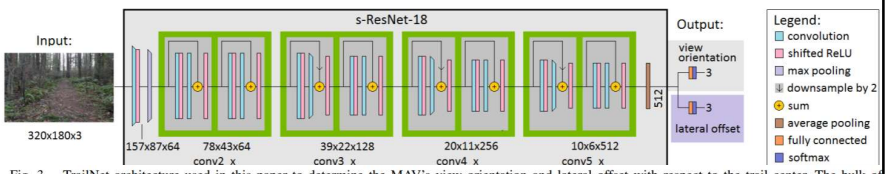


Fig. 3. TrailNet architecture used in this paper to determine the MAV's view orientation and lateral offset with respect to the trail center. The bulk of the network (labeled s-ResNet-18) is the standard ResNet-18 [14] architecture, but without batch normalization and with ReLU replaced by shifted ReLU. The first convolution layer is 7×7, whereas all the others are 3×3. Some layers downsample using stride of 2; all others use stride of 1. All weights except the lateral offset layer are trained with the IDSIA trail dataset, after which these weights are frozen, and the lateral offset layer weights are trained with our dataset. A final joint optimization step could be performed but was deemed to be not necessary.

https://arxiv.org/abs/1705.02550
https://www.youtube.com/watch?v=H7Ym3DMSGms
https://www.youtube.com/watch?v=USYlt9t0lZY

33