# Lecture 6: Backpropagation

Reminder: A2

- Use SGD to train linear classifiers and fully-connected networks
- Today's lecture can help you compute derivatives in A2
- Due Friday January 28, 11:59pm ET

## Autograder

- A1: 10 submissions / day
- A2: 5 submissions / day
- A3+: 3 submissions / day

Autograder is a val set, not a training set! You shouldn't be using autograder to debug; you should be testing your code on the side

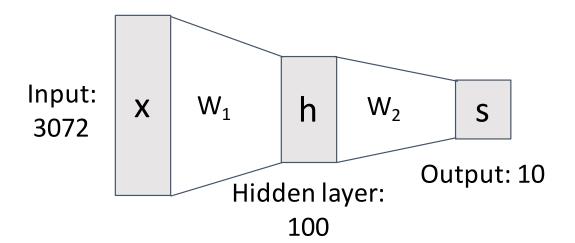
Extra incentive to start early

### Last time: Neural Networks

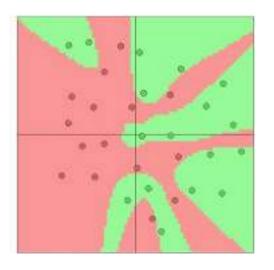
#### **Universal Approximation**

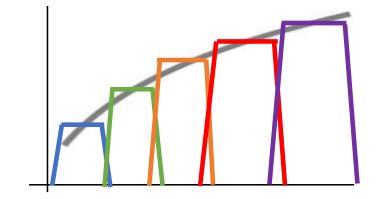
From linear classifiers to fully-connected networks

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$

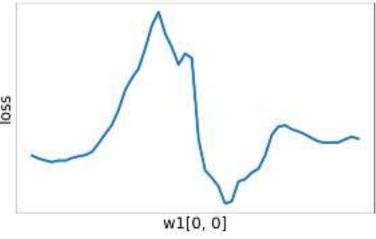


#### **Space Warping**





#### Nonconvex



## Problem: How to compute gradients?

$$s = W_2 \max(0, W_1 x + b_1) + b_2$$

Nonlinear score function

$$L_i = \sum_{i \neq v_i} \max(0, s_j - s_{y_i} + 1)$$

Per-element data loss

$$R(W) = \sum_{k} W_k^2$$

L2 Regularization

$$L(W_1, W_2, b_1, b_2) = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$
 Total loss

If we can compute  $\frac{\partial L}{\partial W_1}$ ,  $\frac{\partial L}{\partial W_2}$ ,  $\frac{\partial L}{\partial b_1}$ ,  $\frac{\partial L}{\partial b_2}$  then we can optimize with SGD

## (Bad) Idea: Derive $abla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_{i} = \sum_{j \neq y_{i}} \max(0, s_{j} - s_{y_{i}} + 1)$$

$$= \sum_{j \neq y_{i}} \max(0, W_{j,:} \cdot x + W_{y_{i},:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_{i} + \lambda \sum_{k} W_{k}^{2}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_{i}} \max(0, W_{j,:} \cdot x + W_{y_{i},:} \cdot x + 1) + \lambda \sum_{k} W_{k}^{2}$$

**Problem**: Very tedious: Lots of matrix calculus, need lots of paper

**Problem**: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

**Problem**: Not feasible for very complex models!

$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

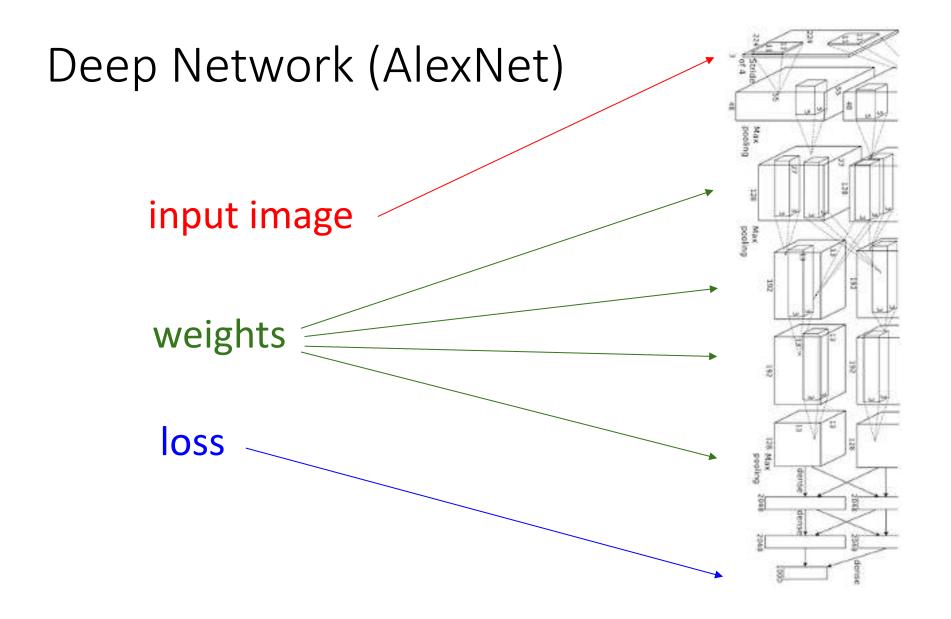
## Better Idea: Computational Graphs

$$S = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$* \quad s \text{ (scores)} \quad \underset{loss}{\overset{\text{hinge}}{\longrightarrow}} \quad L$$

$$R$$

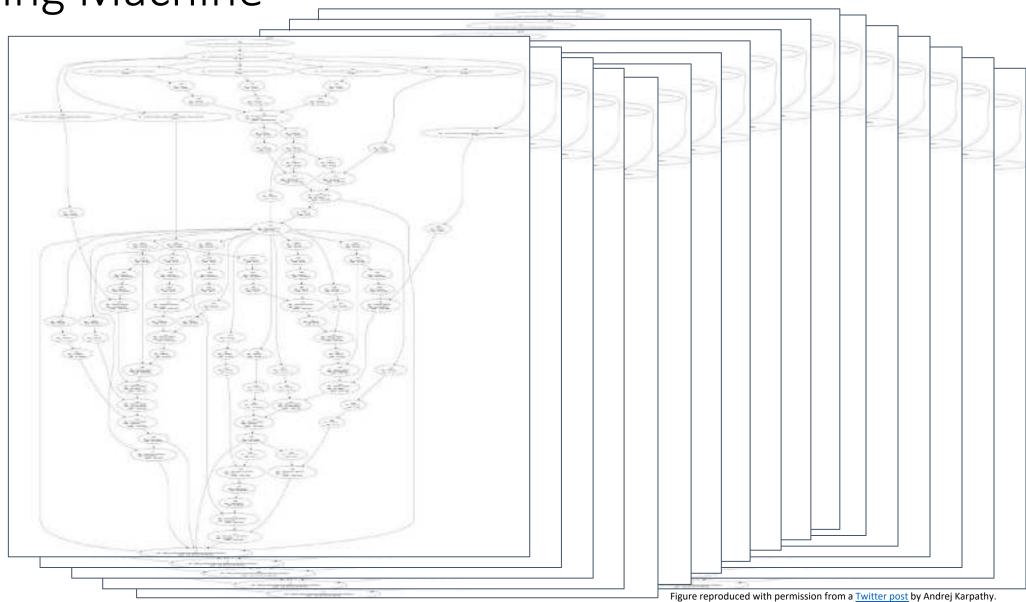


Neural Turing Machine input image loss

Figure reproduced with permission from a <u>Twitter post</u> by Andrej Karpathy.

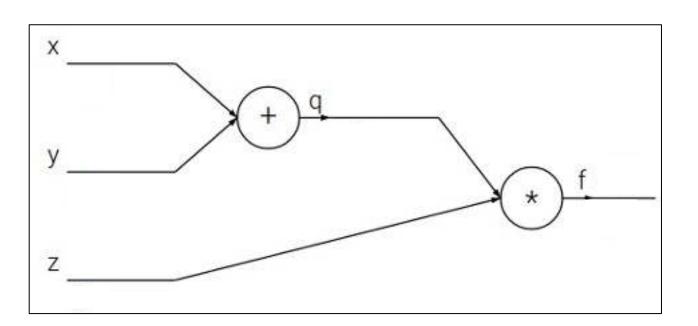
Graves et al, arXiv 2014

Neural Turing Machine

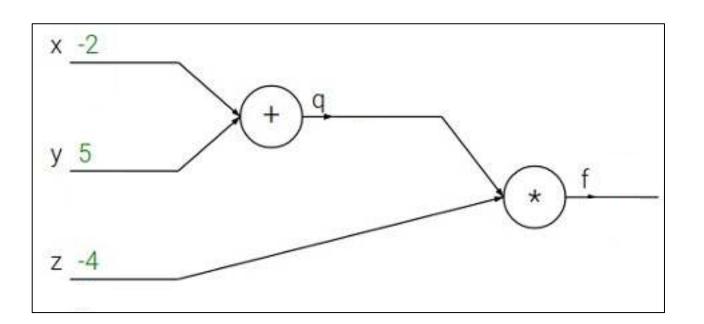


Graves et al, arXiv 2014

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



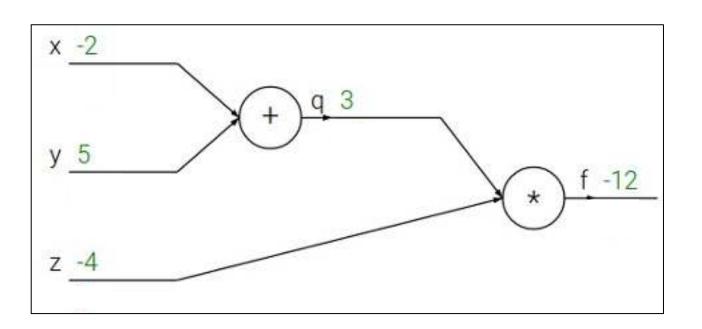
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4



$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

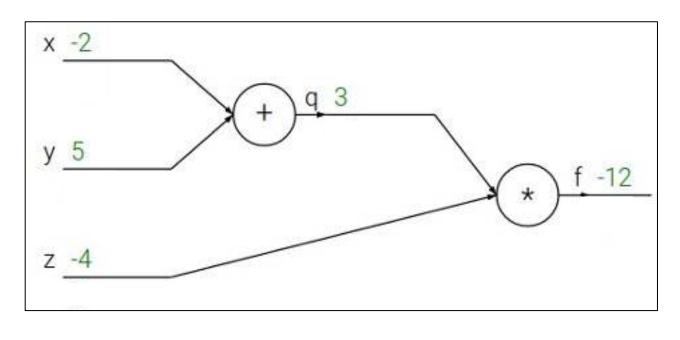
$$q = x + y$$
  $f = q \cdot z$ 



$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 



$$q = x + y$$
  $f = q \cdot z$ 



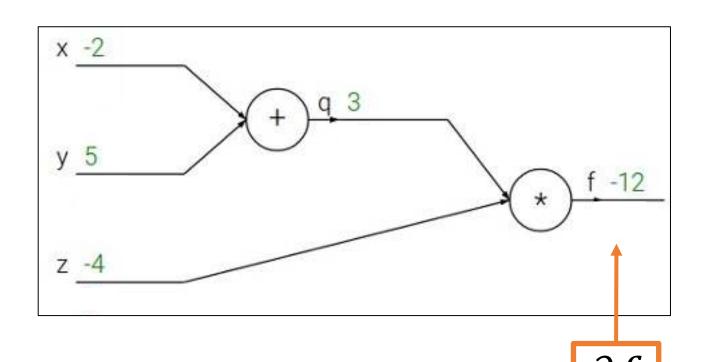
Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

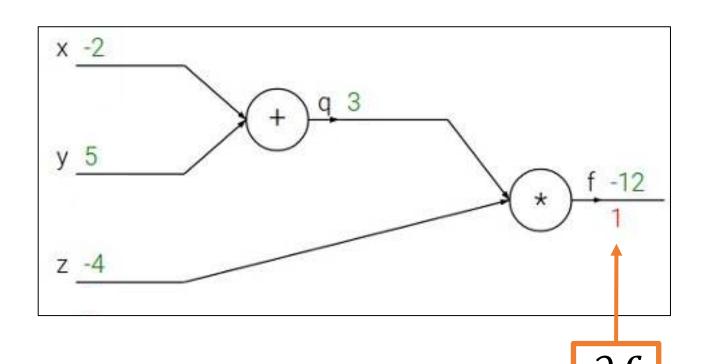


$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

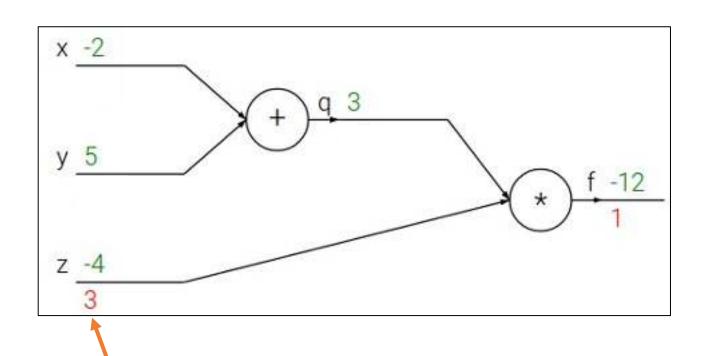


$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



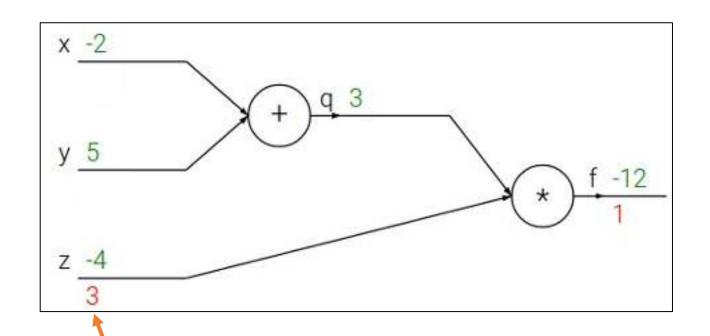
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

#### 1. Forward pass: Compute outputs

$$q = x + y$$

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



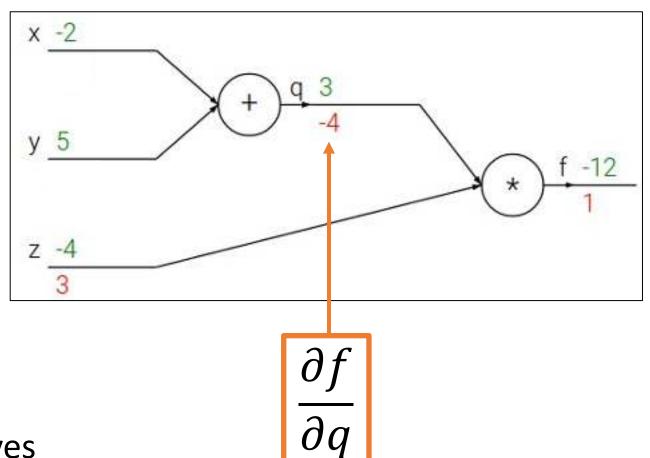
$$\frac{\partial f}{\partial z} = q$$

$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

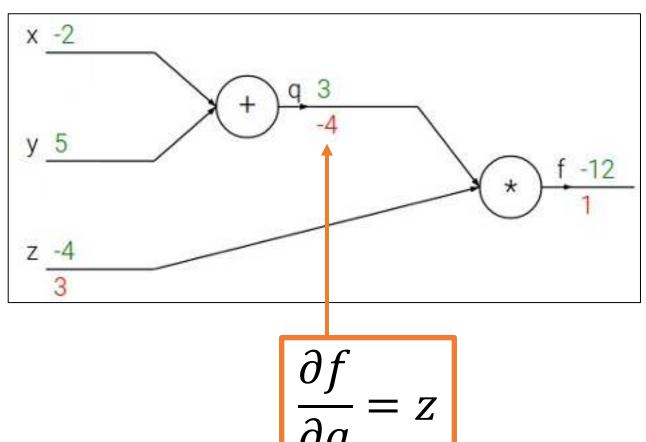


$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

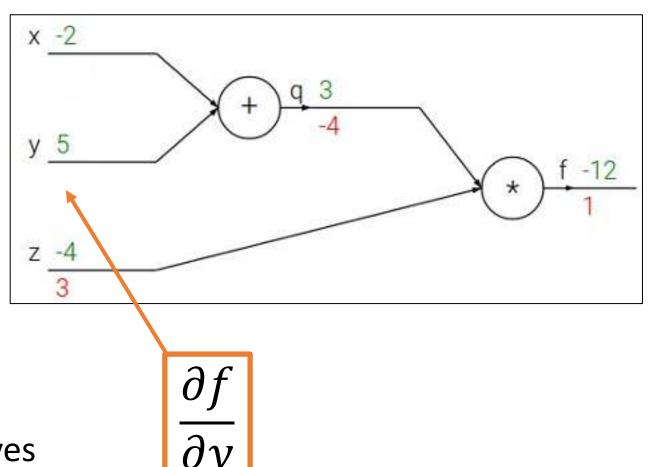


$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



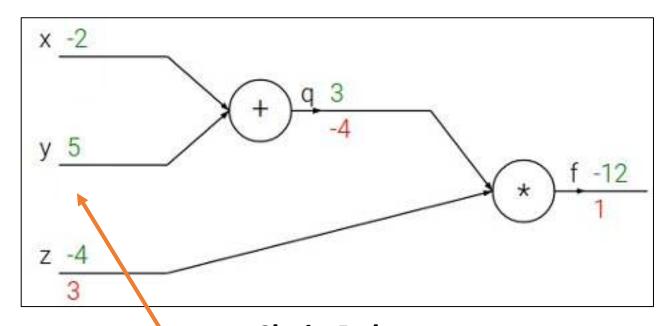
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

2. Backward pass: Compute derivatives

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



#### **Chain Rule**

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

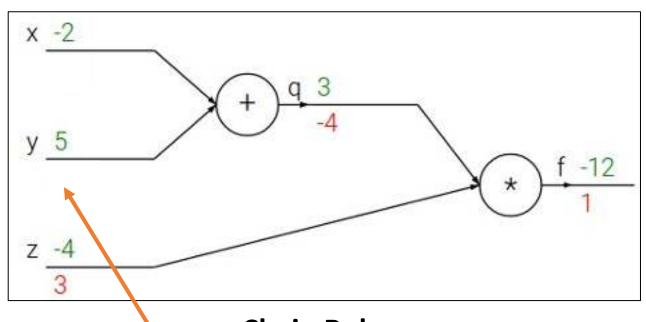
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

2. Backward pass: Compute derivatives

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



**Chain Rule** 

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{1}{2} + \frac{1}{2} + \frac{1}$$

**Gradient** 

Gradient

Gradient

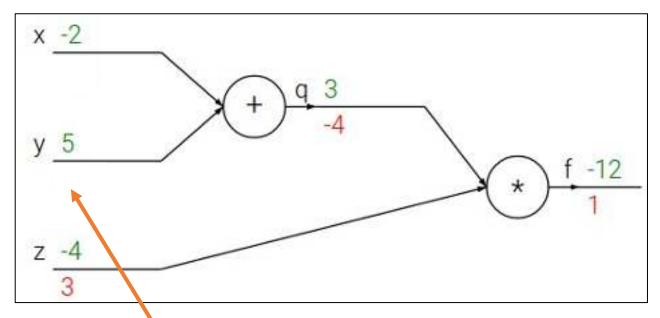
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

#### 2. Backward pass: Compute derivatives

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



#### **Chain Rule**

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

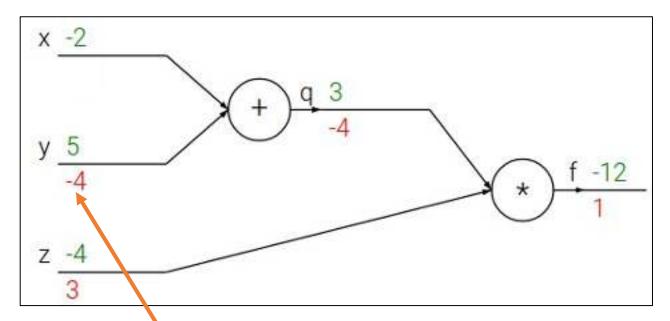
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

#### 2. Backward pass: Compute derivatives

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



#### **Chain Rule**

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

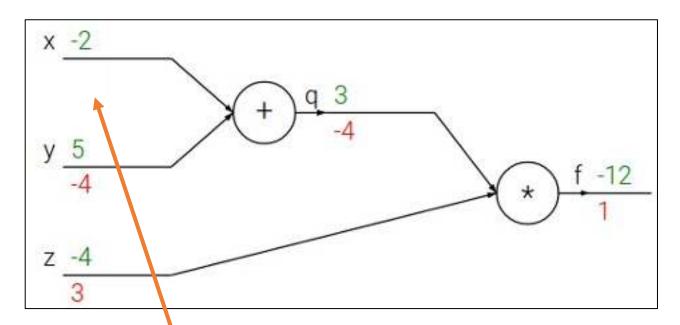
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$ 

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

#### 2. Backward pass: Compute derivatives

Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 



#### **Chain Rule**

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial x} = 1$$

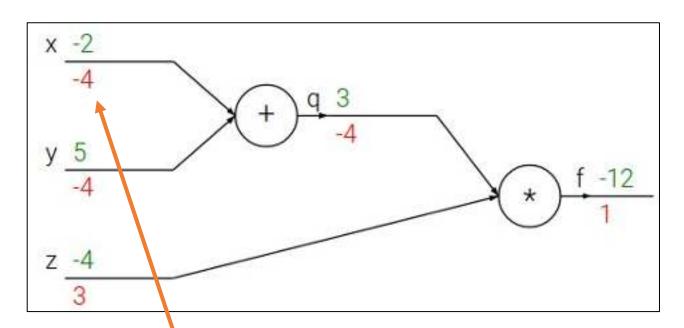
$$f(x, y, z) = (x + y) \cdot z$$
  
e.g. x = -2, y = 5, z = -4

#### 1. Forward pass: Compute outputs

$$q = x + y$$
  $f = q \cdot z$ 

#### 2. Backward pass: Compute derivatives

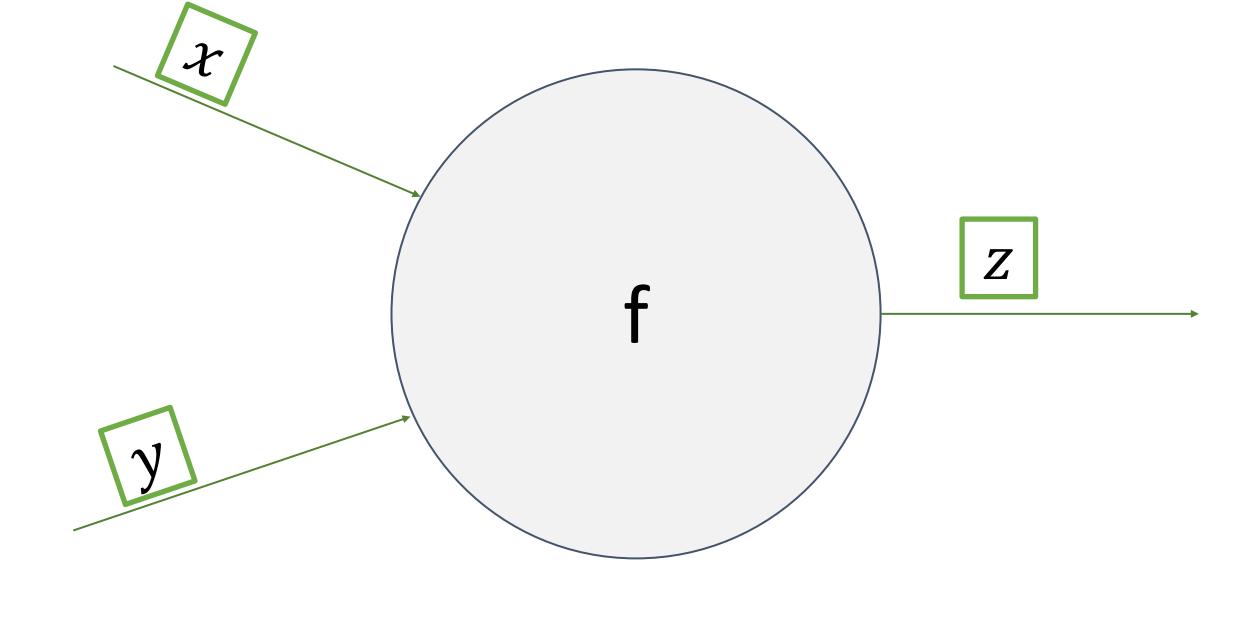
Want: 
$$\frac{\partial f}{\partial x}$$
,  $\frac{\partial f}{\partial y}$ ,  $\frac{\partial f}{\partial z}$ 

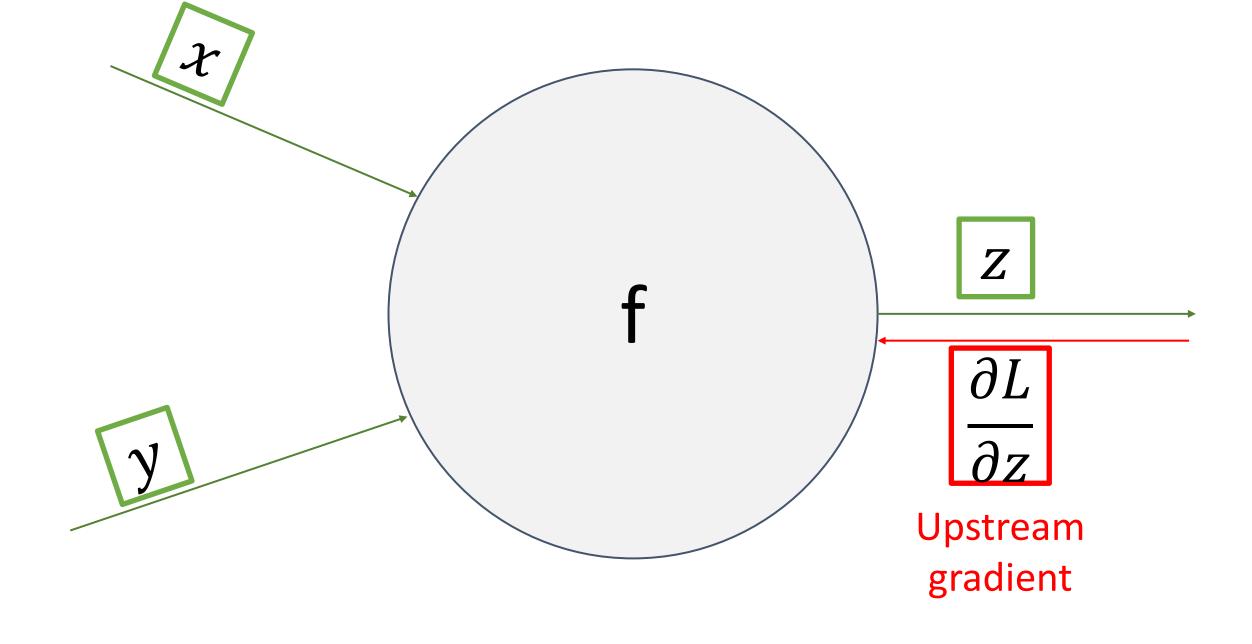


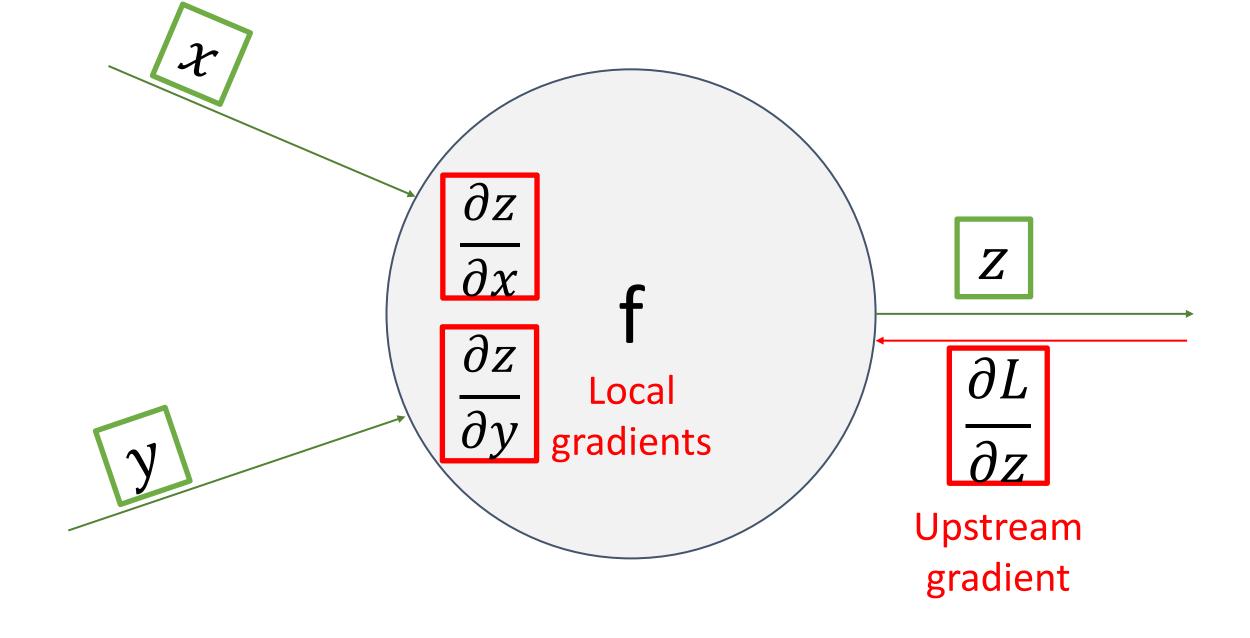
#### **Chain Rule**

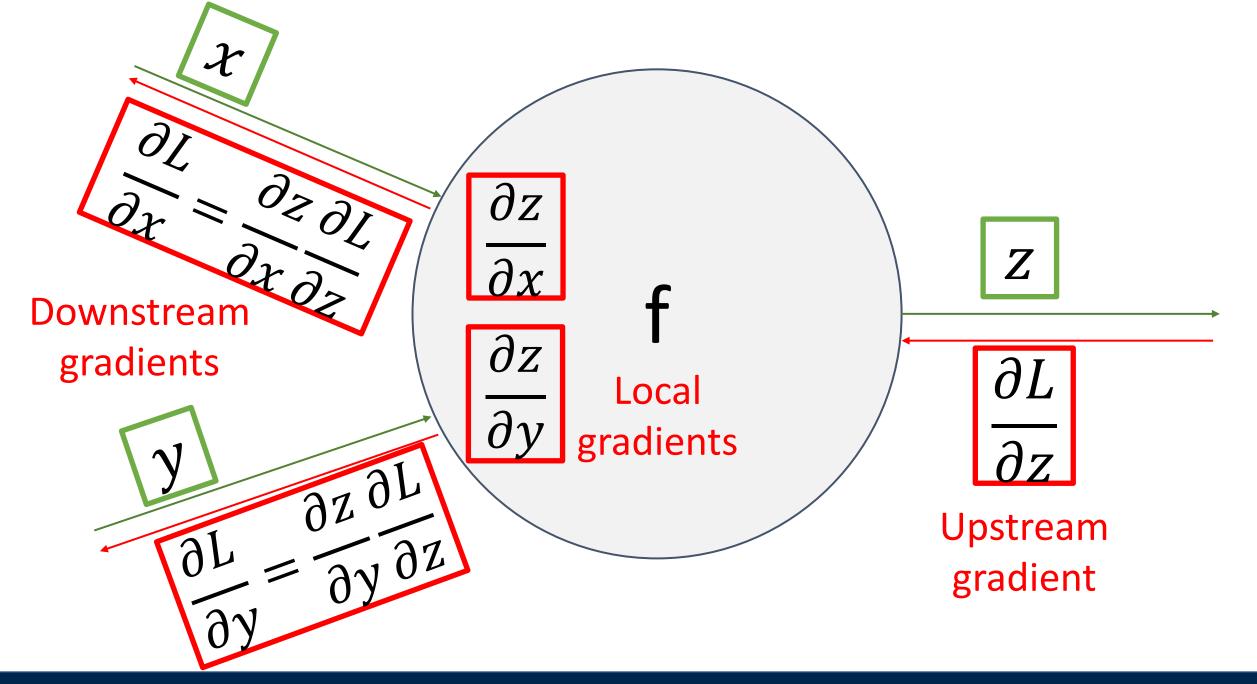
$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial x} = 1$$

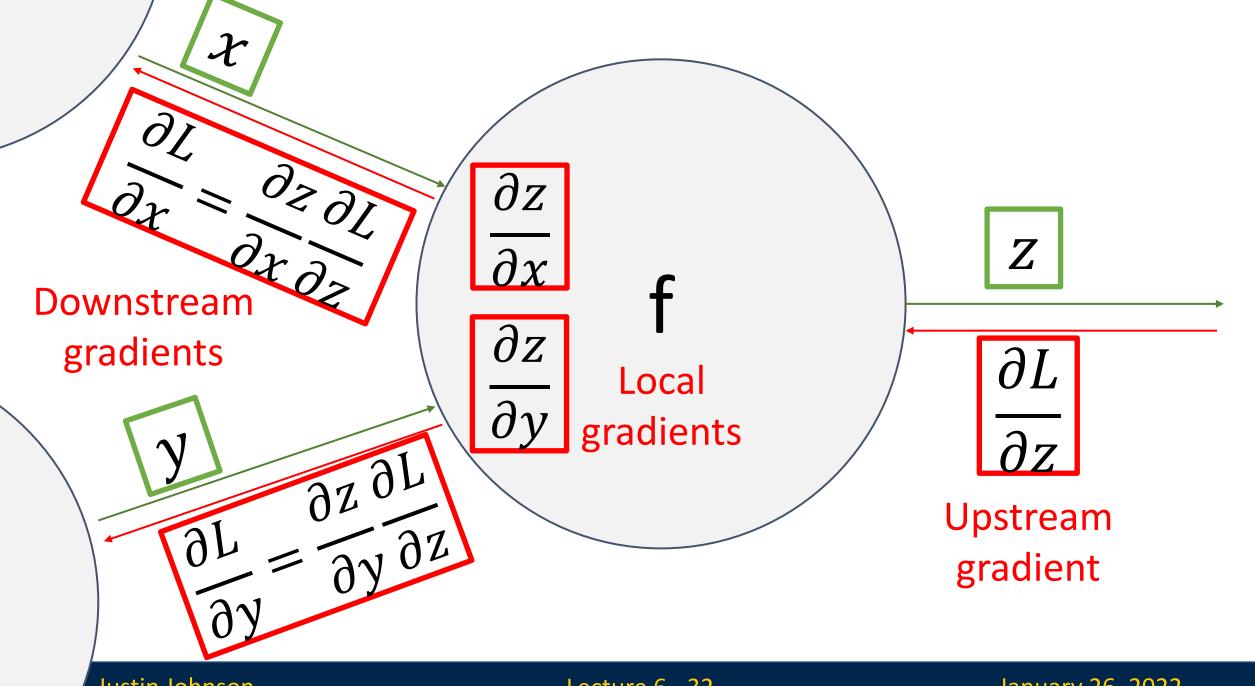






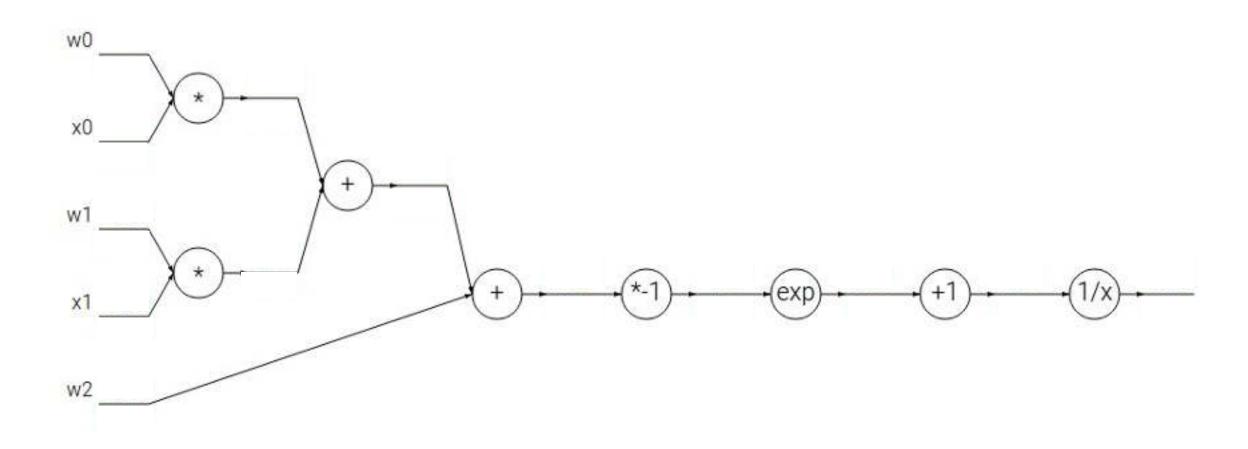


Justin Johnson Lecture 6 - 31 January 26, 2022



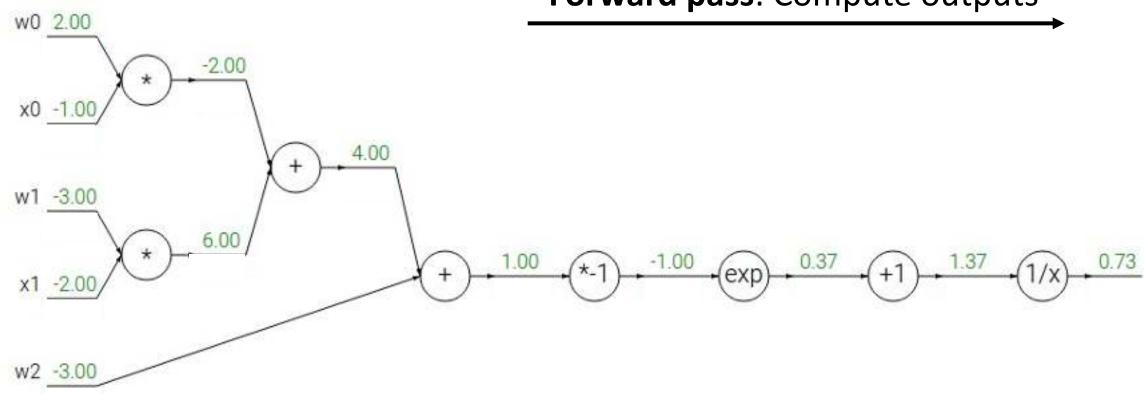
Justin Johnson Lecture 6 - 32 January 26, 2022

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



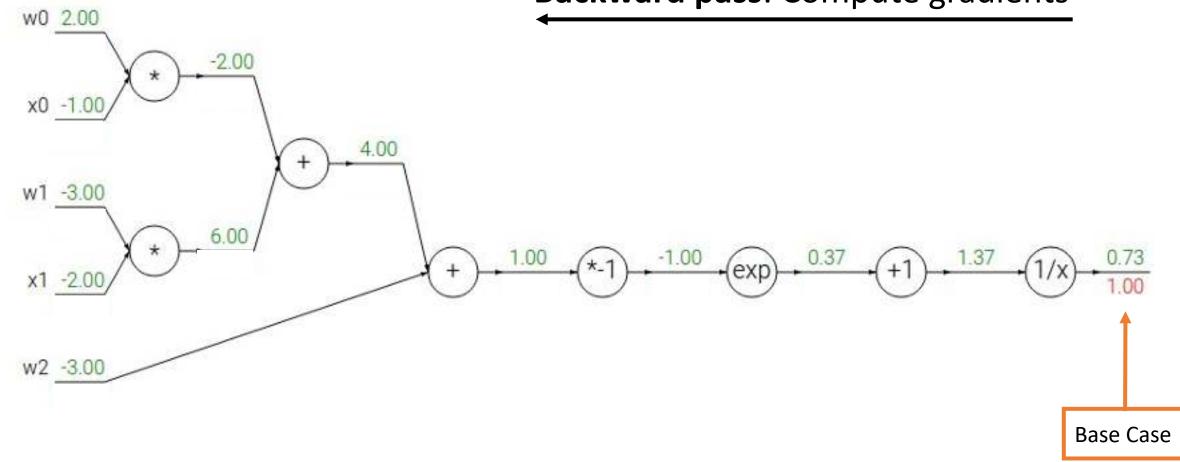
# Another Example $f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

Forward pass: Compute outputs

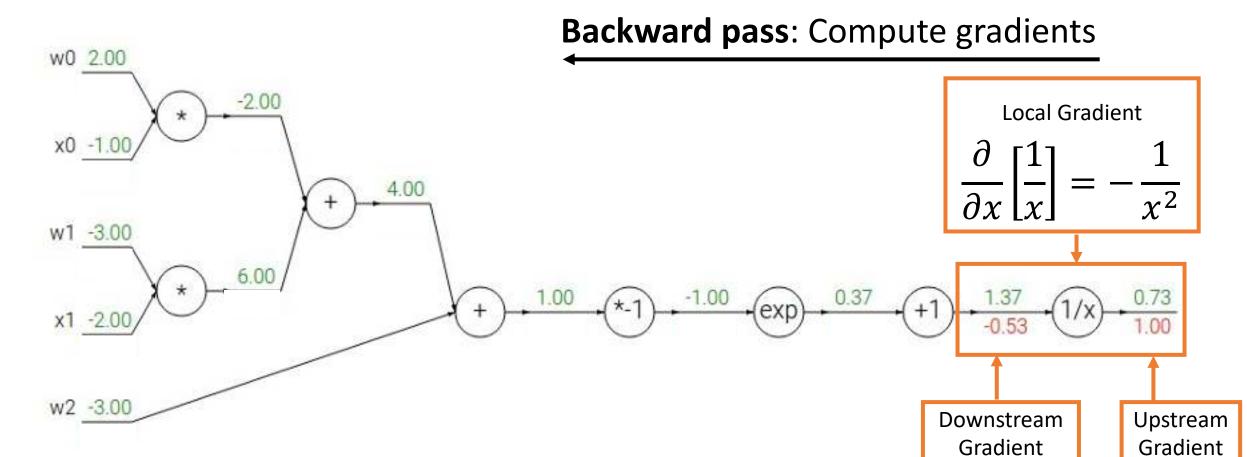


# Another Example $f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

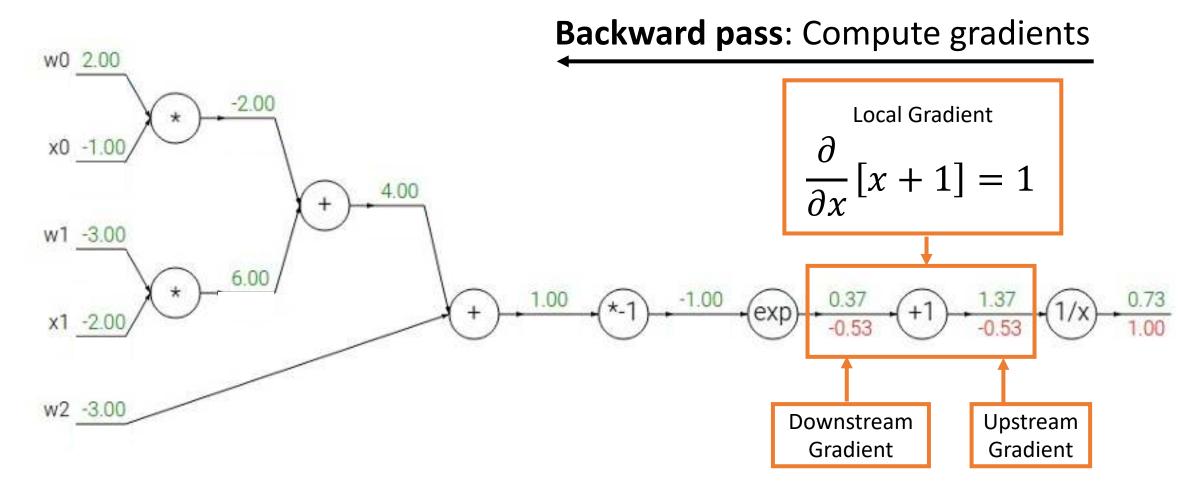
Backward pass: Compute gradients



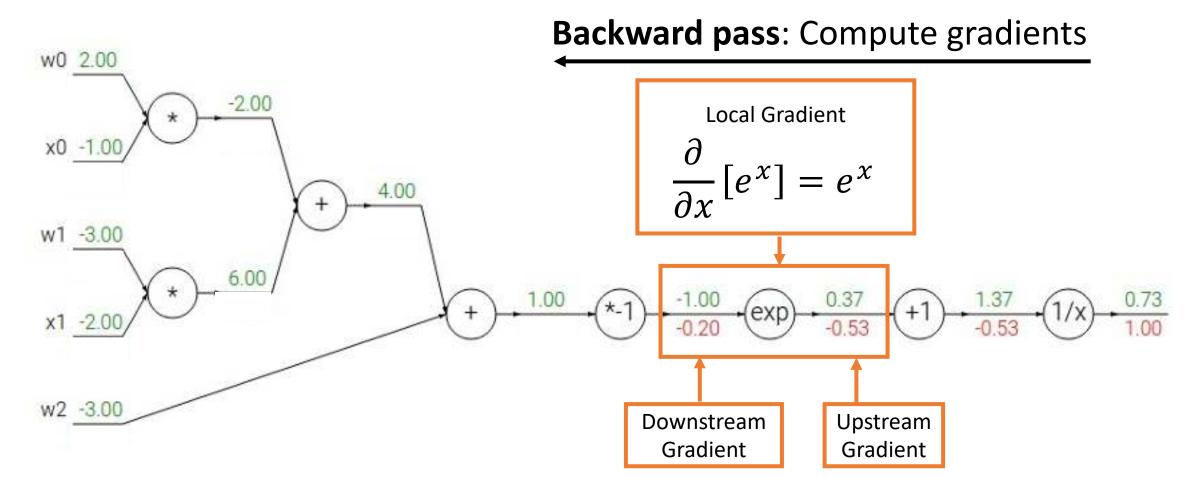
## Another Example $f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



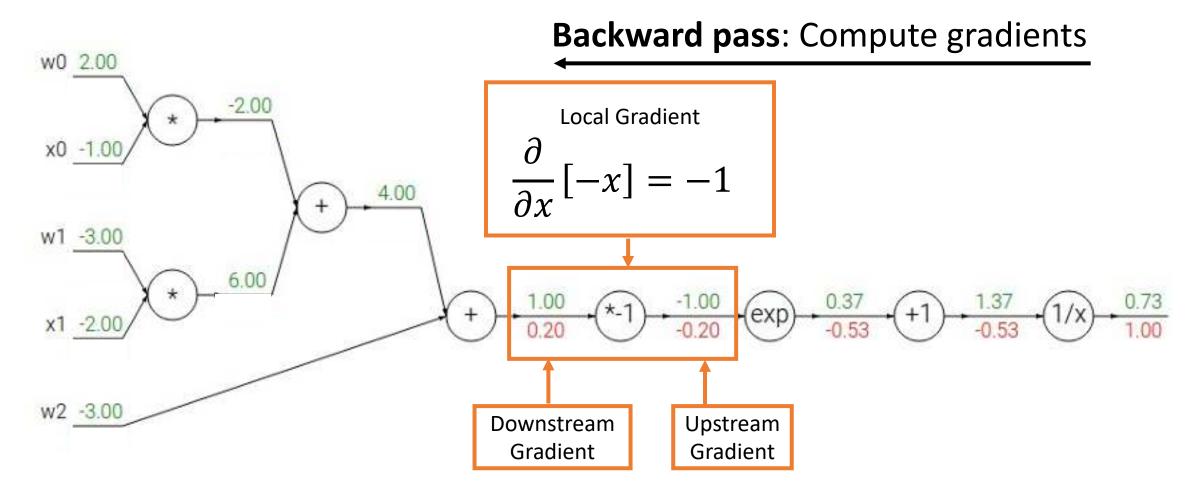
# Another Example $f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$



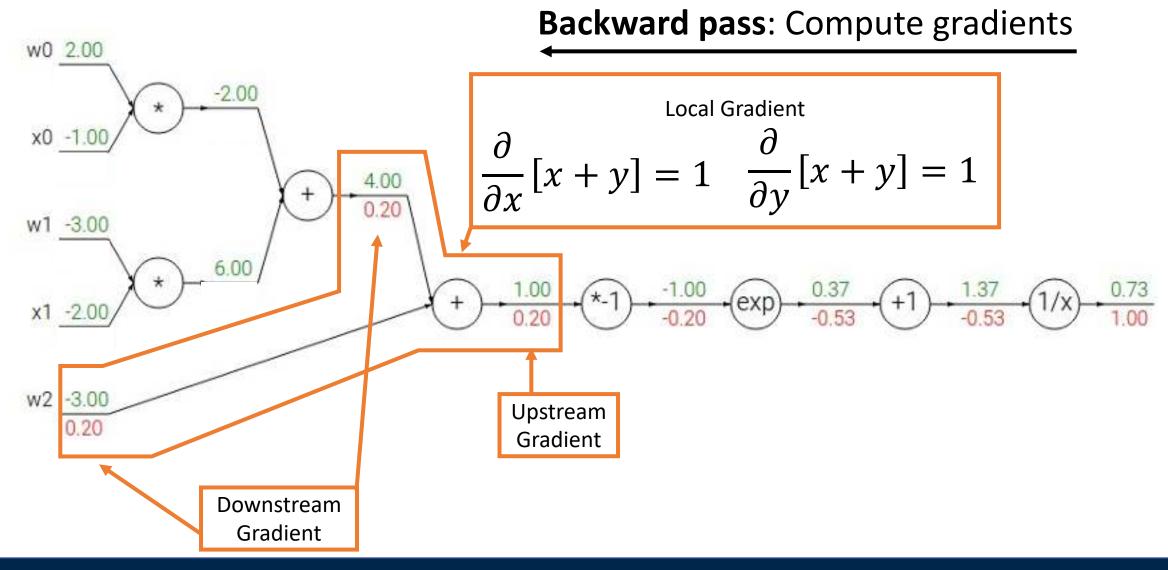
# Another Example $f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



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

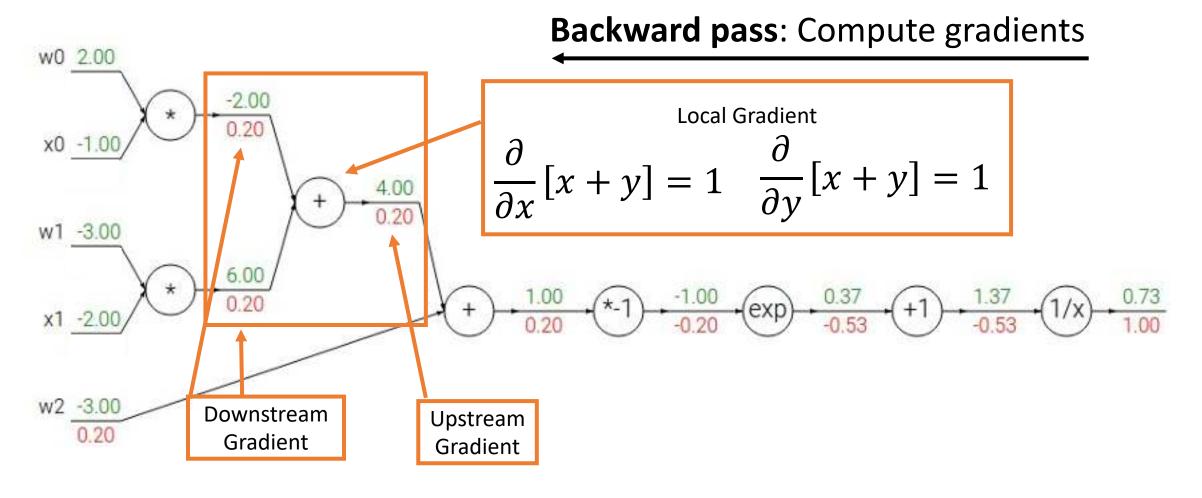


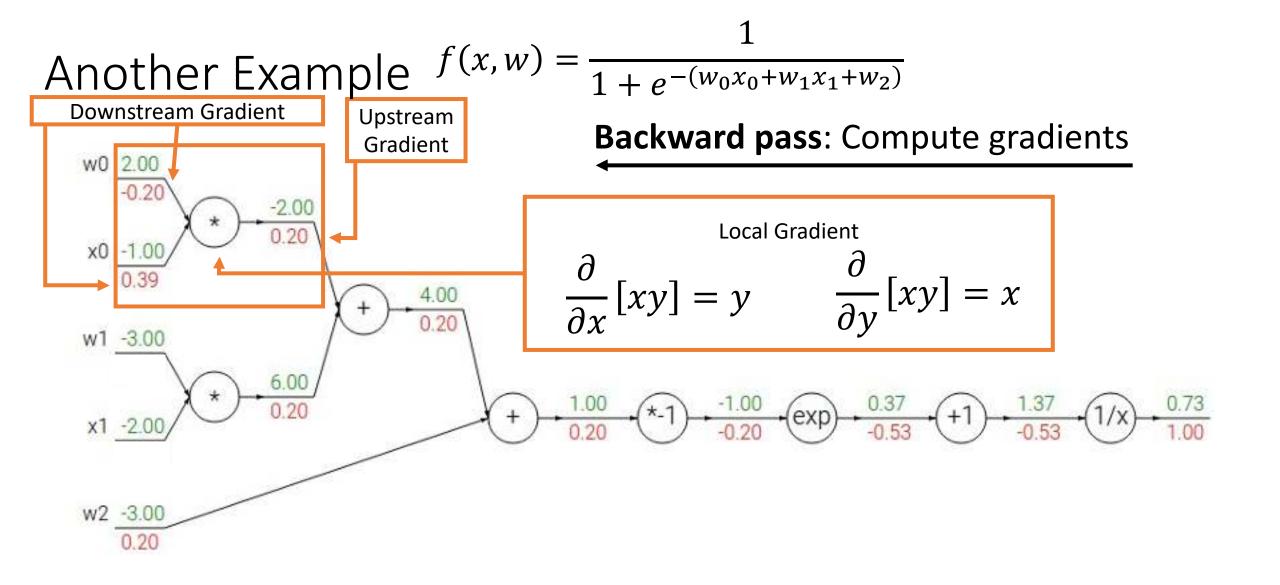
# Another Example $f(x,w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$

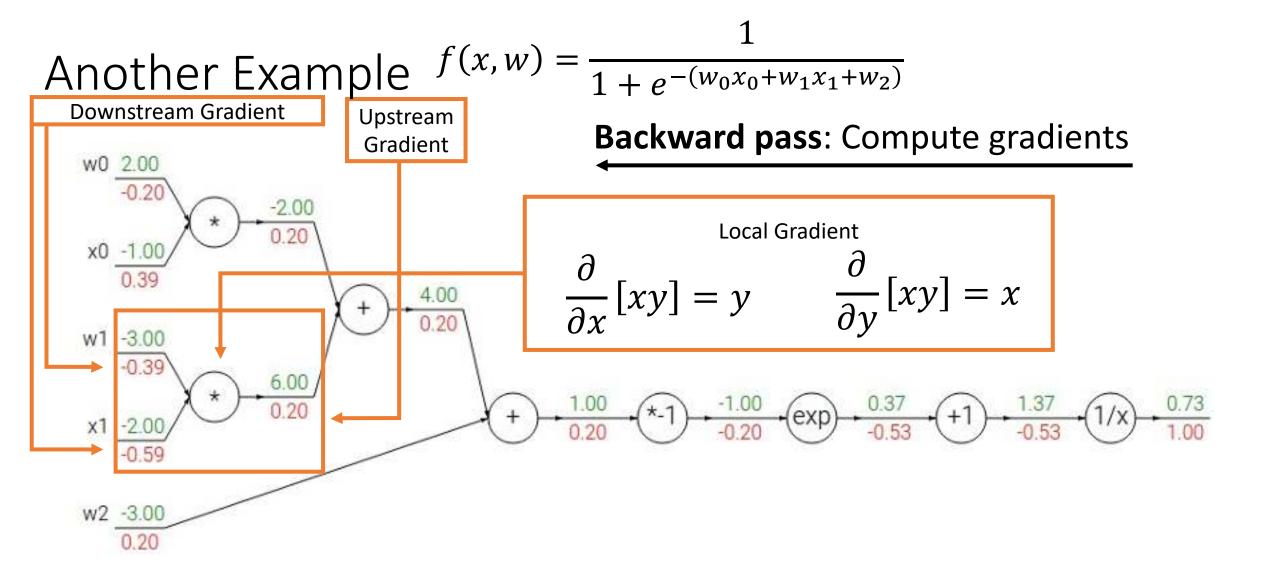


Justin Johnson Lecture 6 - 40 January 26, 2022

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

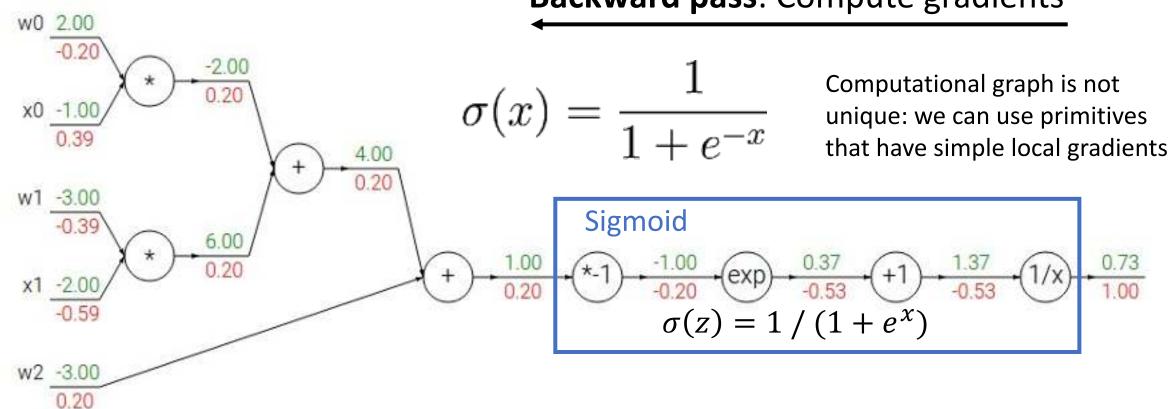






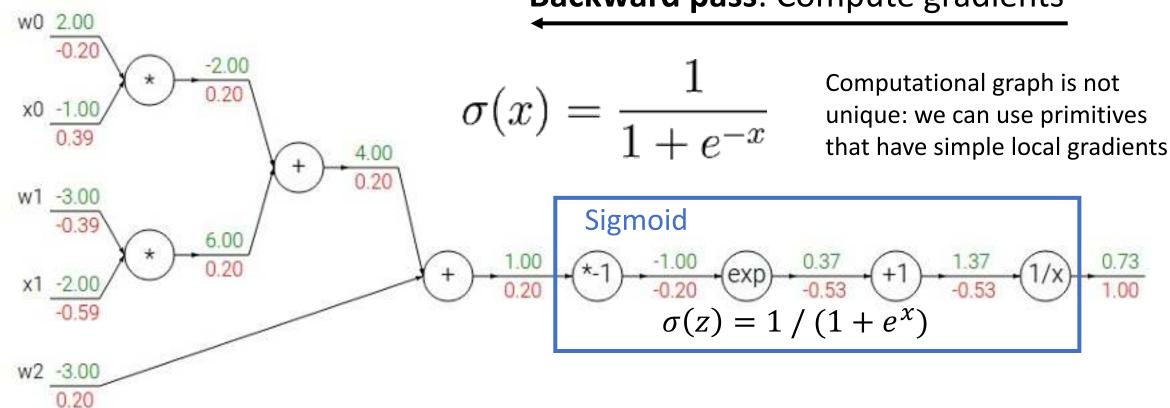
# Another Example $f(x,w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} = \sigma(w_0x_0 + w_1x_1 + w_2)$

#### Backward pass: Compute gradients



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

#### Backward pass: Compute gradients



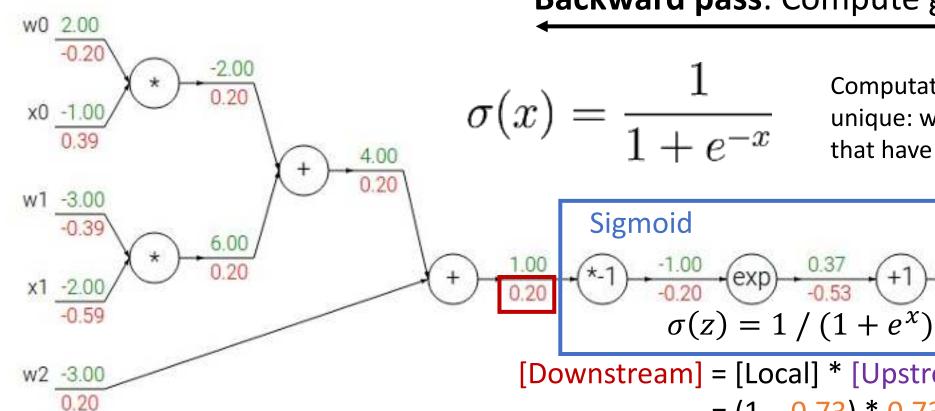
Sigmoid local gradient: 
$$\frac{\partial}{\partial x} \left[ \sigma(x) \right] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = \left( 1 - \sigma(x) \right) \sigma(x)$$

Justin Johnson Lecture 6 - 45 January 26, 2022

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

$$\sigma(w_0 x_0 + w_1 x_1 + w_2)$$

#### **Backward pass**: Compute gradients



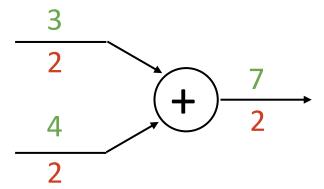
Computational graph is not unique: we can use primitives that have simple local gradients

[Downstream] = [Local] \* [Upstream]

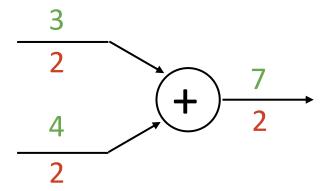
$$= (1 - 0.73) * 0.73 * 1.0 = 0.2$$

Sigmoid local gradient: 
$$\frac{\partial}{\partial x} \left[ \sigma(x) \right] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = \left( 1 - \sigma(x) \right) \sigma(x)$$

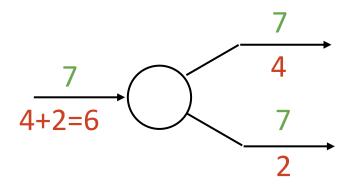
add gate: gradient distributor



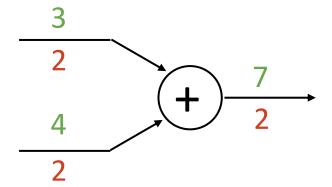
add gate: gradient distributor



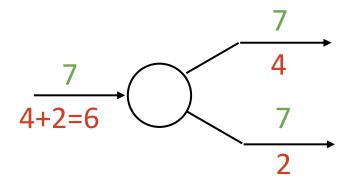
copy gate: gradient adder



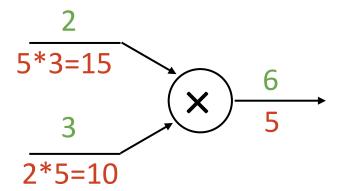
add gate: gradient distributor



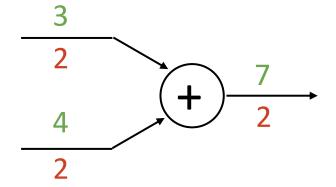
copy gate: gradient adder



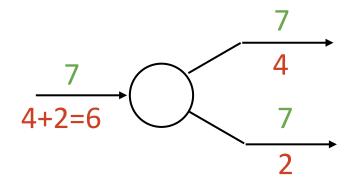
mul gate: "swap multiplier"



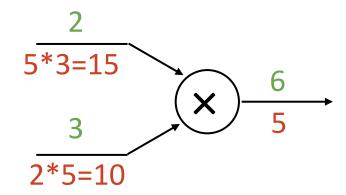
add gate: gradient distributor



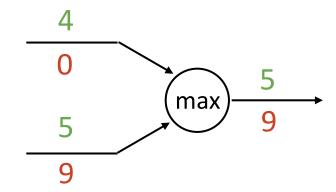
copy gate: gradient adder



mul gate: "swap multiplier"

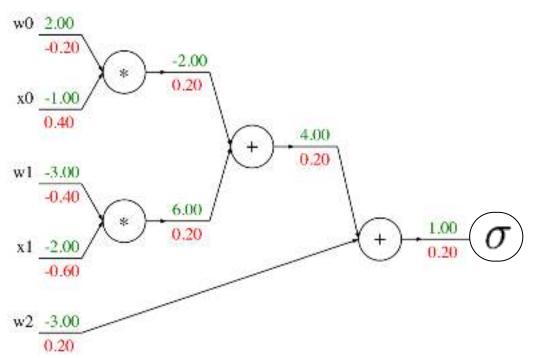


max gate: gradient router



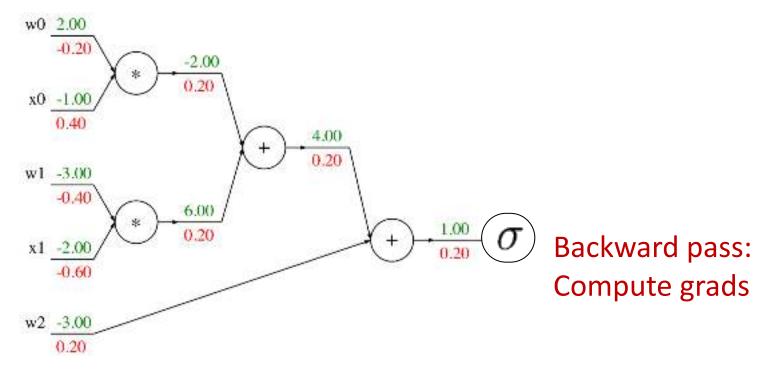
"Flat" gradient code:

Forward pass:



"Flat" gradient code:

Forward pass:

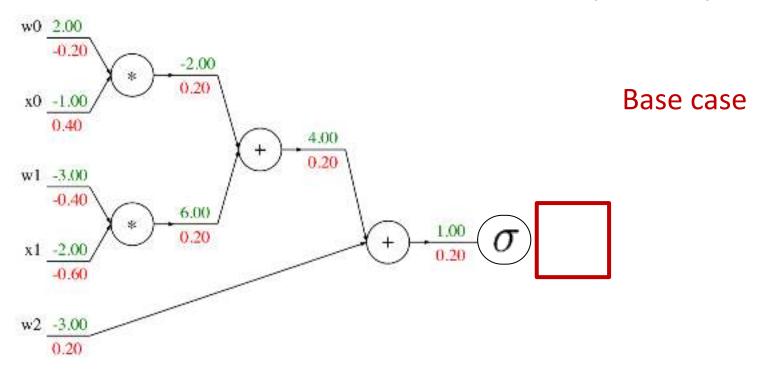


```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

"Flat" gradient code:

Forward pass:

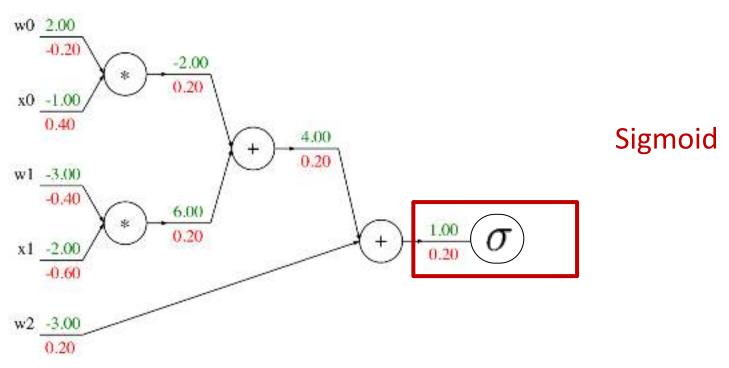


```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

"Flat" gradient code:

Forward pass:



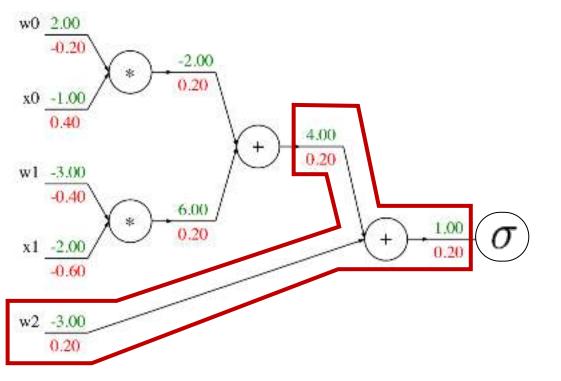
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

"Flat" gradient code:

Forward pass:

Compute output



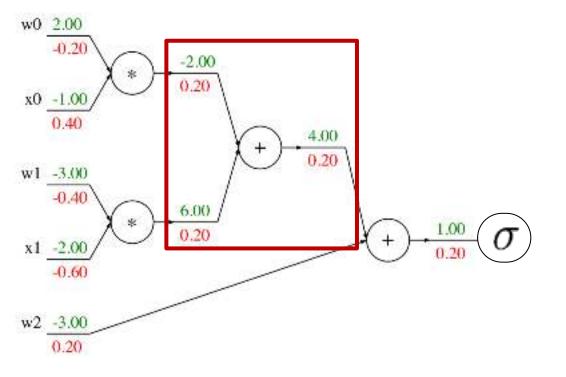
Add

```
def f(w0, x0, w1, x1, w2):
 50 = w0 * x0
 s1 = w1 * x1
 s2 = s0 + s1
 s3 = s2 + w2
 L = sigmoid(s3)
 grad_L = 1.0
 grad_s3 = grad_L * (1 - L) * L
 grad_w2 = grad_s3
 grad_s2 = grad_s3
 grad_s0 = grad_s2
 grad_s1 = grad_s2
 grad_w1 = grad_s1 * x1
 grad_x1 = grad_s1 * w1
 grad_w0 = grad_s0 * x0
 grad_x0 = grad_s0 * w0
```

"Flat" gradient code:

Forward pass:

Compute output



Add

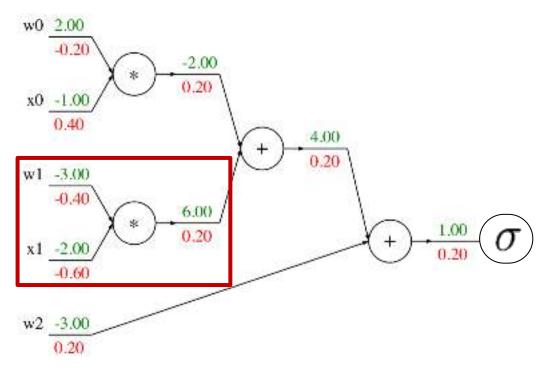
```
def f(w0, x0, w1, x1, w2):
 50 = w0 * x0
 s1 = w1 * x1
 s2 = s0 + s1
 s3 = s2 + w2
 L = sigmoid(s3)
 grad_L = 1.0
 grad_s3 = grad_L * (1 - L) * L
 grad_w2 = grad_s3
 grad_s2 = grad_s3
 grad_s0 = grad_s2
 grad_s1 = grad_s2
 grad_w1 = grad_s1 * x1
 grad_x1 = grad_s1 * w1
 grad_w0 = grad_s0 * x0
 grad_x0 = grad_s0 * w0
```

Justin Johnson Lecture 6 - 56 January 26, 2022

"Flat" gradient code:

Forward pass:

Compute output



Multiply

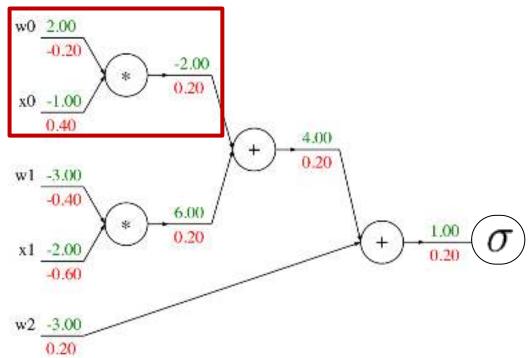
```
def f(w0, x0, w1, x1, w2):
 s0 = w0 * x0
 s1 = w1 * x1
 s2 = s0 + s1
 s3 = s2 + w2
 L = sigmoid(s3)
 grad_L = 1.0
 grad_s3 = grad_L * (1 - L) * L
 grad_w2 = grad_s3
 grad_s2 = grad_s3
 grad_s0 = grad_s2
 grad_s1 = grad_s2
 grad_w1 = grad_s1 * x1
 grad_x1 = grad_s1 * w1
 grad_w0 = grad_s0 * x0
 grad_x0 = grad_s0 * w0
```

Justin Johnson Lecture 6 - 57 January 26, 2022

"Flat" gradient code:

Forward pass:

Compute output



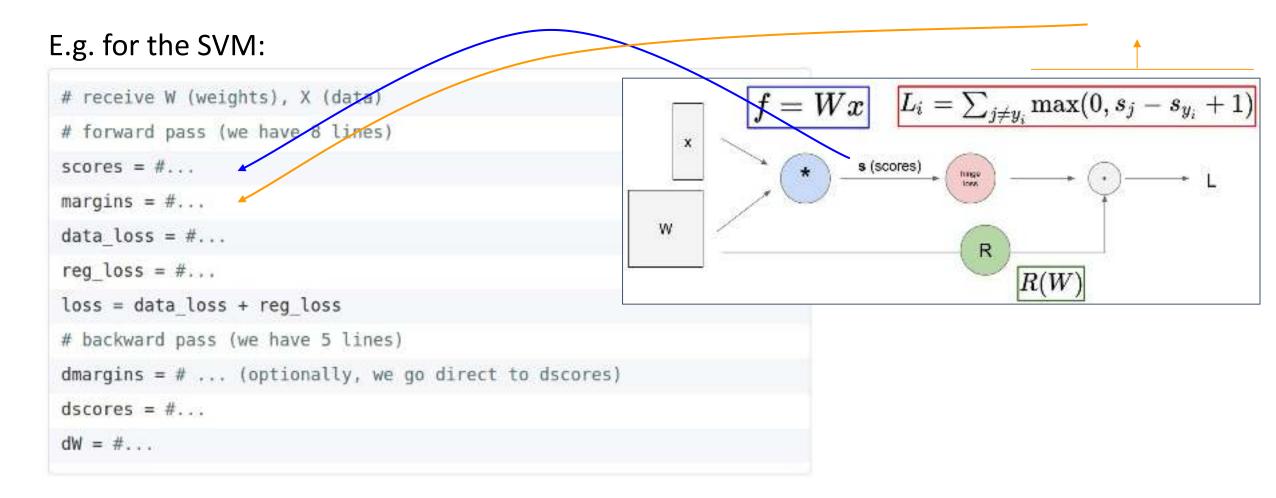
Multiply

```
def f(w0, x0, w1, x1, w2):
 s0 = w0 * x0
 s1 = w1 * x1
 s2 = s0 + s1
 s3 = s2 + w2
 L = sigmoid(s3)
 grad_L = 1.0
 grad_s3 = grad_L * (1 - L) * L
 grad_w2 = grad_s3
 grad_s2 = grad_s3
 grad_s0 = grad_s2
 grad_s1 = grad_s2
 grad_w1 = grad_s1 * x1
 grad_x1 = grad_s1 * w1
 grad_w0 = grad_s0 * x0
 grad_x0 = grad_s0 * w0
```

Justin Johnson Lecture 6 - 58 January 26, 2022

# "Flat" Backprop: Do this for Assignment 2!

Your gradient code should look like a "reversed version" of your forward pass!



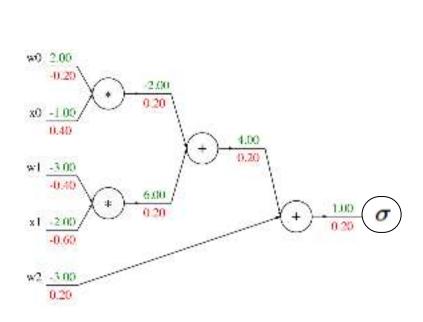
# "Flat" Backprop: Do this for Assignment 2!

Your gradient code should look like a "reversed version" of your forward pass!

#### E.g. for two-layer neural net:

```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = \#... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1, dW2, db2 = #...
dW1, db1 = #...
```

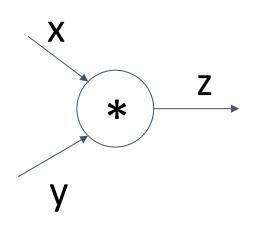
## Backprop Implementation: Modular API



#### Graph (or Net) object (rough pseudo code)

```
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()):
            qate.backward() # little piece of backprop (chain rule applied)
        return inputs gradients
```

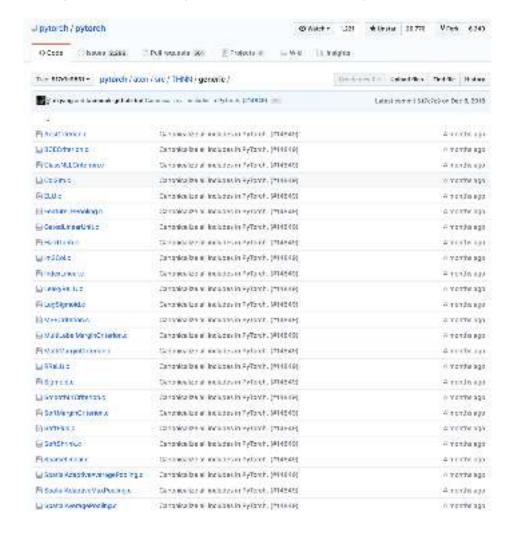
# Example: PyTorch Autograd Functions



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
 @staticmethod
 def forward(ctx, x, y):
                                              Need to stash some
    ctx.save_for_backward(x, y)
                                              values for use in
    z = x * y
                                              backward
    return z
 @staticmethod
                                              Upstream
  def backward(ctx, grad_z):
                                              gradient
    x, y = ctx.saved_tensors
    grad_x = y * grad_z # dz/dx * dL/dz
                                              Multiply upstream
    grad_y = x * grad_z # dz/dy * dL/dz
                                              and local gradients
    return grad_x, grad_y
```

# Example: PyTorch operators



of Souther Condition Printers	Cancelica be although dea in PyTorbe self-(EHZ)	4 norths app
⊒ SpartialConvolus street-c	Campaign the attinicia dear in Pyttorch, cor 48 (6)	4 months ago
Espainac evacewainers	(product two strong state in express in the rest	4 Perfenge
Specialis managements of	- conductive and some in replace is not shared	director less
Standard or Value of the Contract of the Contr	Carrier as horizer e feter i 104545)	4 months ago
Specialis District Countries	Carrier to eliminate in Pattern 19145451	4 membrane
A Son half-hild move in gra-	Care des level trackes à Estant (#16545)	4 months also
- Contribution in debutings.	Carcellos Sentinoladas in Tytos n. (414540)	4 months ago
al SantaRep ou enfeedings	Carcelos be ellingiadas in Pythoch (#1888)	finertia ago
ja Specialis z kenping kilmons	Cancellos Depático Lobar In Pylloros, en 4946)	9 months ago
State of the second sec	Carcaliza de Jalian siden in Pyronin 1 (1) 4 6 4 6	4 months ago
ROSE .	Concess Red Concessor in Pytonia Concessor	4 rocety age
Studie	Concern service describes a Parlace (404848)	discribina.
Siterone bekombaning	Careno, www.freschole.fotoen (414545)	4 rento-as
A Tempore Suplication Subtraction	Care sick love threliades in PyTaren, (414645)	depertuage
- Tempore Toy Conyoutlorus	Carcelos benéficiados in Pytoren (414540)	4 months ago
a Tampore upčernotingi, nesto	Carpellos be efficiadas in Pythron, 1814 6464	f nonta spc
⊒ fartoora upolimplingsvaryetik.	Cancellos de altinos des in Pylloren (m. 1816)	4 north age
Д метеринация мерторичества.	Contract bediting size in eyear 1, 40 keyes	4 months ago
Systematic style in Vinciologic	contains position where Epison's Octobers	# months ago
Spannan Scoutter of a	Carriero so of recicles e Patricia (1/14848)	4 (4) (4)
Nivokeapatteers and 99 a	Carcaro profire bides a Pataco 14/48458	denote an
H Volument: OffstedConvolutions	Care nice by 41 for ladys in TyTexts 141 (5140)	4 months was
er Volume ark Olivered He dinasing c	Carporida de altriciades in Pytanon 14/48464	4 months ago
g Voluments instricted with piece ing a	Cancelos be eitherades in Pytteron, or 48484	4 months ago
g voluments with innocents into a	Cancellar be a time under in Pyrana 1 (in 1919)	A months ago
All wolk internal construction providings.	CWC 972 ON JUDGE OF PYTON THE CHARLES	4 thérethé ngo
(Switzmance)physiocentricity is	minimum des displacies à Parlair e séculage)	A FRICTA SEC
Spakering Seed approved	Carrier Vocabulate + Pitter + 10 4549	4 mento acc
STANDARD ASSESS OF TAXABLE	Carcello localitacidas à l'effact (9/4545)	Arenteup
Af nor coming by 1	instance authorize thispoins based on opernois (#252))	I micros wys
-specific shape.h	Use Integer math to compute output also of pooling operations (ef 4400)	finenta esc
a) untold.c	Canadica be attracted as in Pythoran (at 4845)	1 horra spc

```
#ifndef TH_GENERIC_FILE
    #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
    #else
     void THNN_(Sigmoid_updateOutput)(
               THNNState *state,
              THTensor *input,
              THTensor *output)
9
       THTensor_(sigmoid)(output, input);
12
     void THNN_(Sigmoid_updateGradInput)(
14
               THNNState *state,
              THTensor *gradOutput,
               THTensor *gradInput,
16
              THTensor *output)
18
19
       THNN_CHECK_NELEMENT(output, gradOutput);
       THTensor_(resizeAs)(gradInput, output);
20
21
       TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
        scalar_t z = *output_data;
        *gradInput_data = *gradOutput_data * (1. - z) * z;
24
      );
25
26
    #endif
```

### PyTorch sigmoid layer

<u>Source</u>

```
#ifndef TH_GENERIC_FILE
    #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
    #else
    void THNN_(Sigmoid_updateOutput)(
                                                           Forward
              THNNState *state,
              THTensor *input,
              THTensor *output)
9
      THTensor_(sigmoid)(output, input);
10
    void THNN_(Sigmoid_updateGradInput)(
              THNNState *state,
              THTensor *gradOutput,
              THTensor *gradInput,
              THTensor *output)
19
      THNN_CHECK_NELEMENT(output, gradOutput);
      THTensor_(resizeAs)(gradInput, output);
20
      TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
        scalar_t z = *output_data;
        *gradInput_data = *gradOutput_data * (1. - z) * z;
      );
    #endif
```

11

14

16

18

21

24

25

#### PyTorch sigmoid layer

Source

```
#ifndef TH_GENERIC_FILE
    #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
    #else
     void THNN_(Sigmoid_updateOutput)(
                                                           Forward
               THNNState *state,
              THTensor *input,
               THTensor *output)
9
      THTensor_(sigmoid)(output, input);
11
     void THNN (Sigmoid updateGradInput)(
14
               THNNState *state,
              THTensor *gradOutput,
               THTensor *gradInput,
16
               THTensor *output)
18
      THNN_CHECK_NELEMENT(output, gradOutput);
19
      THTensor_(resizeAs)(gradInput, output);
20
21
      TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22
        scalar_t z = *output_data;
        *gradInput data = *gradOutput data * (1. - z) * z;
      );
25
26
    #endif
```

#### PyTorch sigmoid layer

```
static void sigmoid_kernel(TensorIterator& iter) {
   AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
      unary_kernel_vec(
        iter,
      [=](scalar_t a) -> scalar_t { return (1 / (1 + std::exp((-a)))); },
      [=](Vec256<scalar_t> a) {
        a = Vec256<scalar_t>((scalar_t)(0)) - a;
        a = a.exp();
        a = vec256<scalar_t>((scalar_t)(1)) + a;
        a = a.reciprocal();
        return a;
      });
   Forward actually defined elsewhere...
```

```
return (1 / (1 + std::exp((-a))));
```

Source

```
#ifndef TH_GENERIC_FILE
    #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
    #else
     void THNN_(Sigmoid_updateOutput)(
                                                           Forward
              THNNState *state,
              THTensor *input,
              THTensor *output)
9
      THTensor_(sigmoid)(output, input);
    void THNN_(Sigmoid_updateGradInput)(
13
14
               THNNState *state,
              THTensor *gradOutput,
15
              THTensor *gradInput,
16
              THTensor *output)
18
19
      THNN_CHECK_NELEMENT(output, gradOutput);
      THTensor_(resizeAs)(gradInput, output);
20
21
      TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22
        scalar_t z = *output_data;
        *gradInput_data = *gradOutput_data * (1. - z) * z;
24
```

#### PyTorch sigmoid layer

**Backward** 

$$(1-\sigma(x))\,\sigma(x)$$

<u>Source</u>

25

So far: backprop with scalars

What about vector-valued functions?

# Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

## Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N,$$

$$\left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will y change?

### Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N, \\ \left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

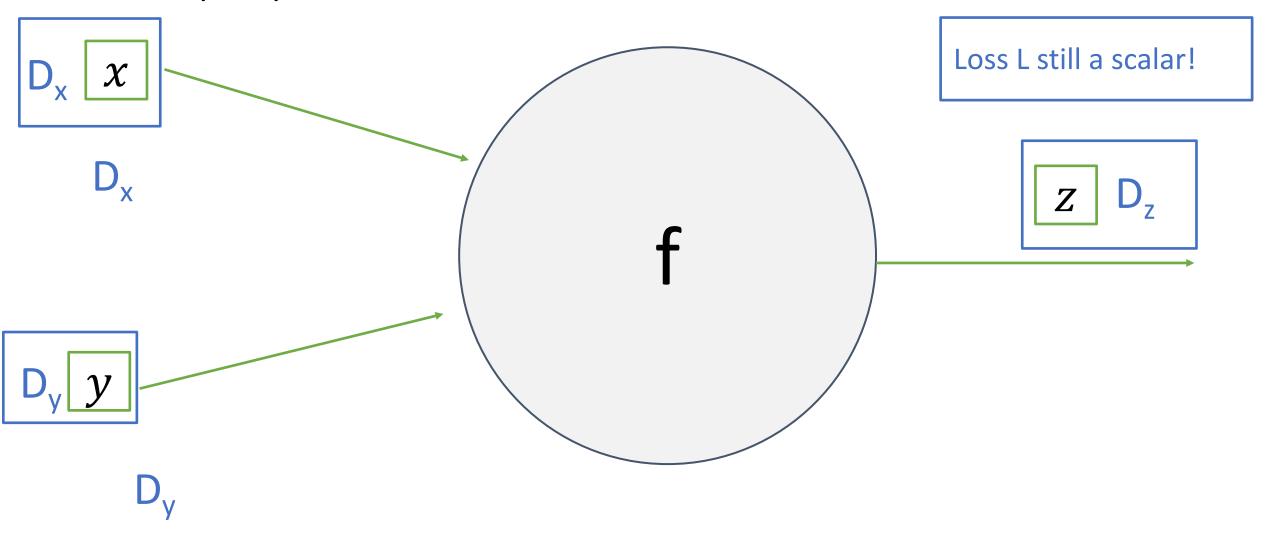
Derivative is **Jacobian**:

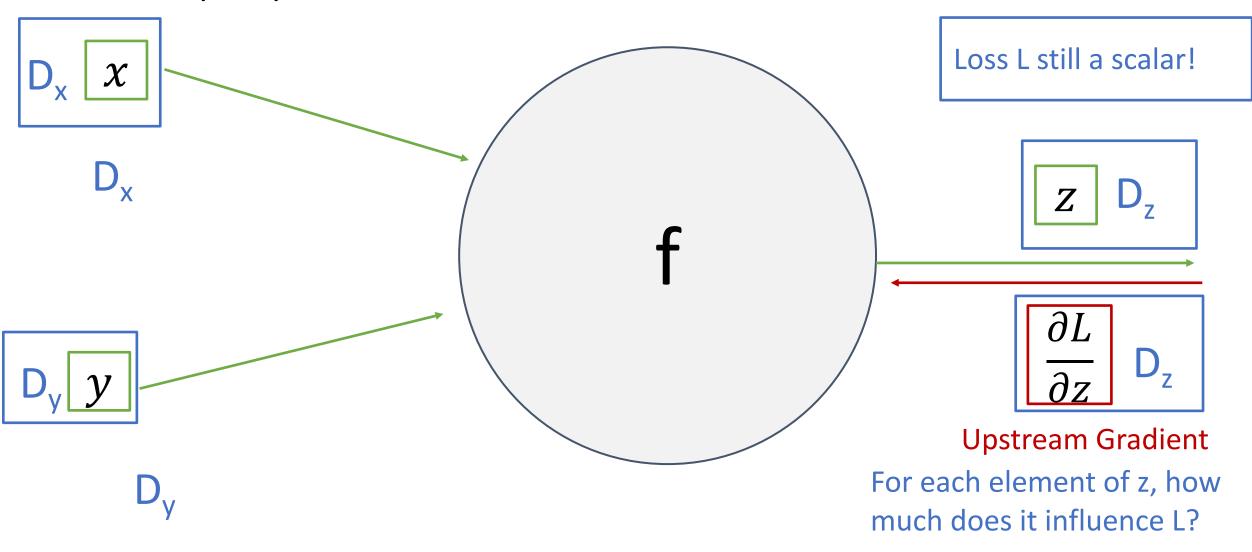
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$$

$$\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

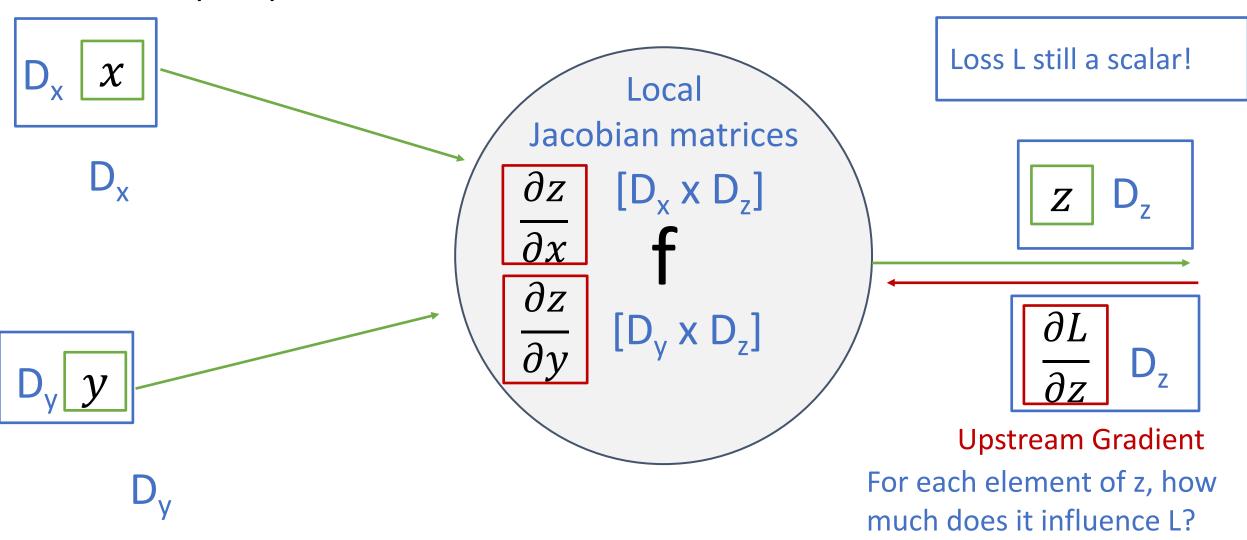
For each element of x, if it changes by a small amount then how much will each element of y change?

# Backprop with Vectors

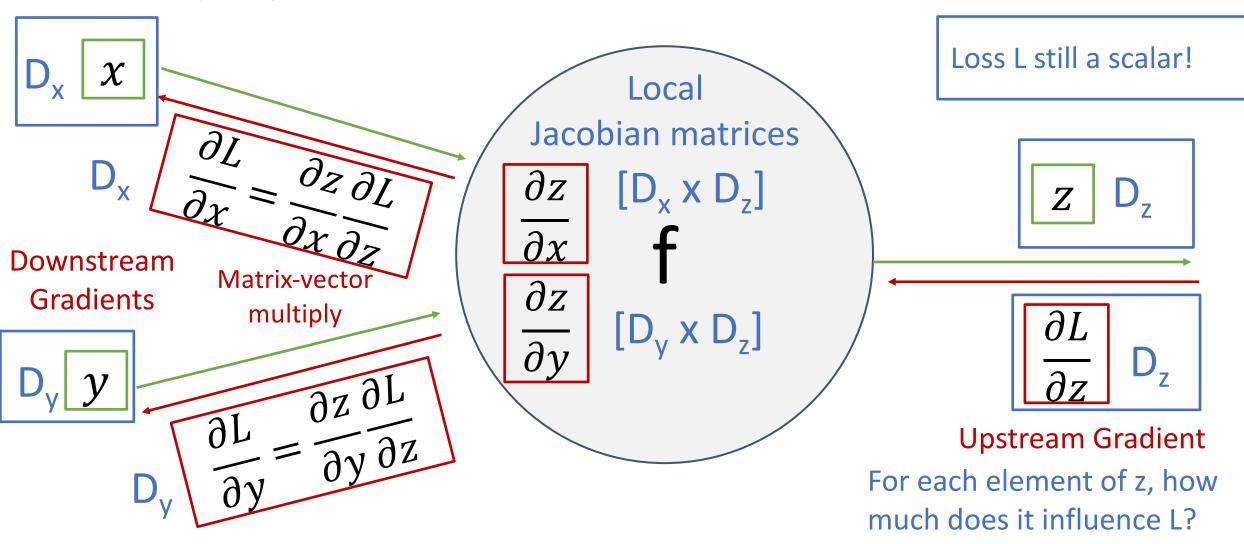




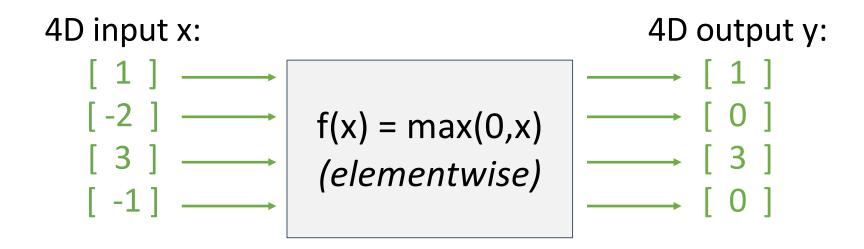
Justin Johnson Lecture 6 - 73 January 26, 2022

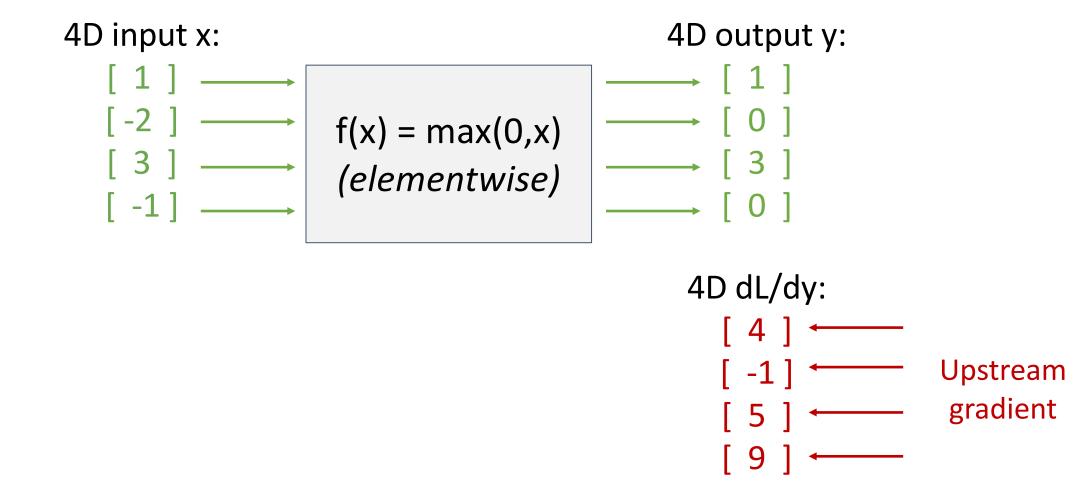


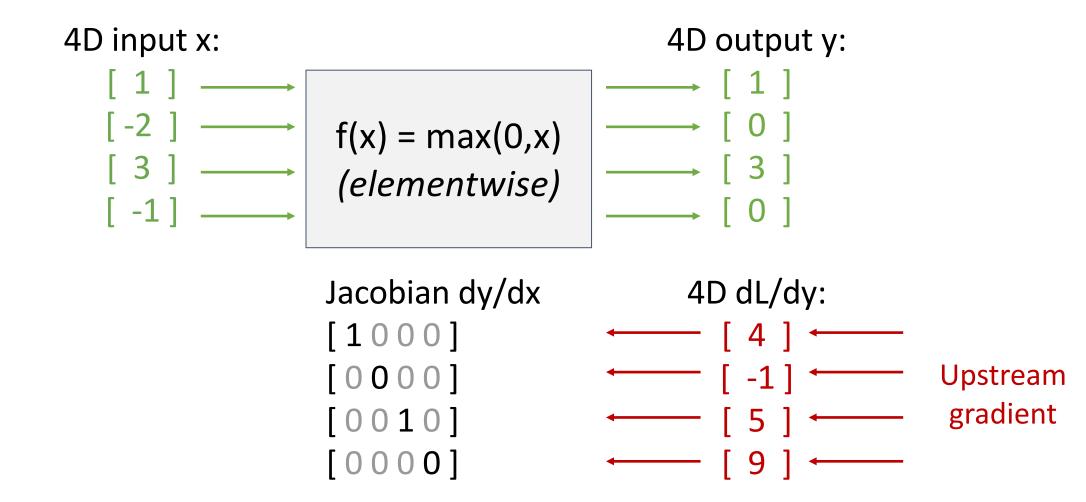
Justin Johnson Lecture 6 - 74 January 26, 2022

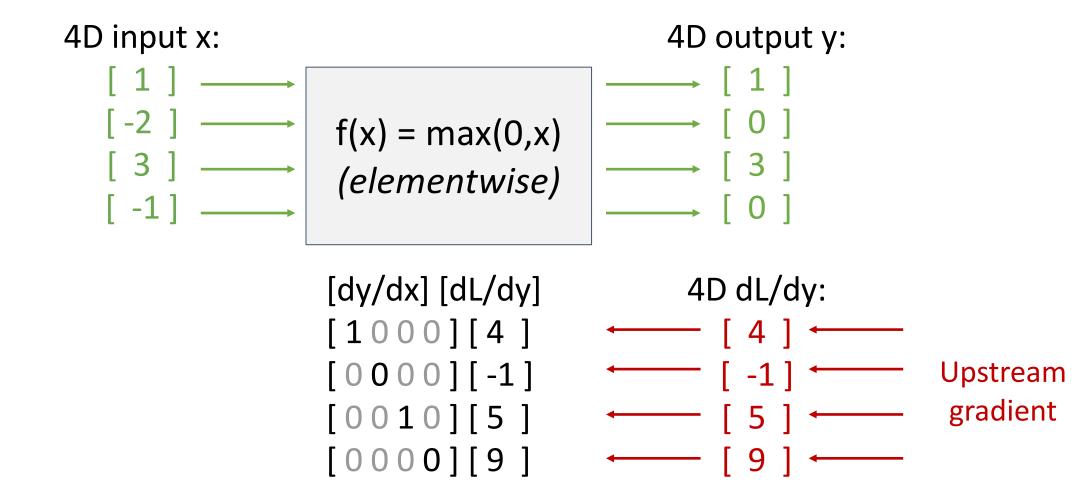


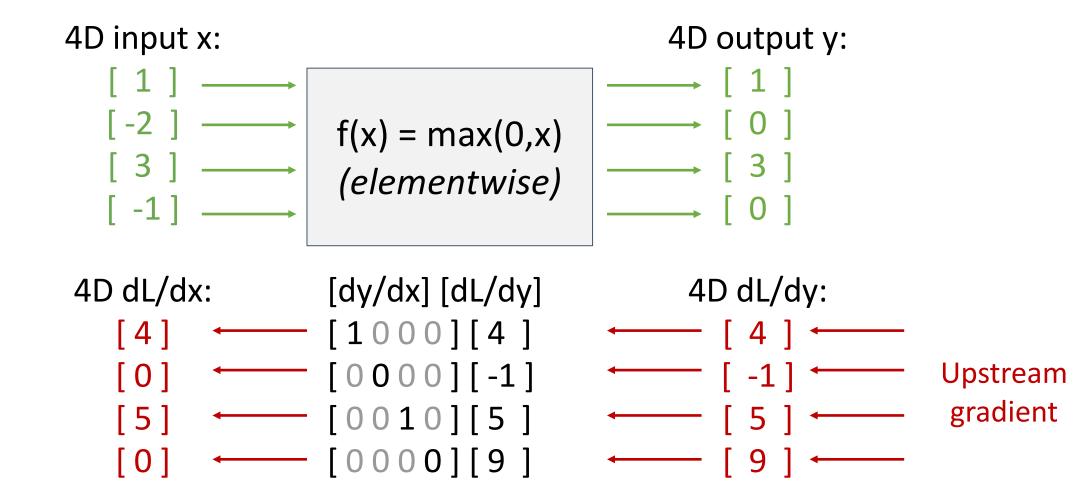
Justin Johnson Lecture 6 - 75 January 26, 2022



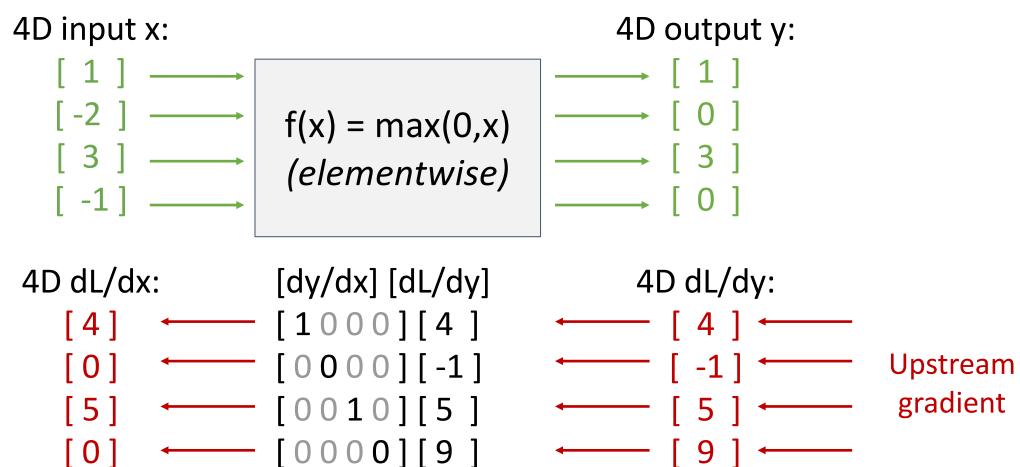








Jacobian is **sparse**: off-diagonal entries all zero! Never **explicitly** form Jacobian; instead use **implicit** multiplication



Jacobian is **sparse**: off-diagonal entries all zero! Never **explicitly** form Jacobian; instead use **implicit** multiplication

#### 4D input x:

$$\begin{bmatrix} -2 \end{bmatrix} \longrightarrow f(x)$$

$$f(x) = \max(0,x)$$

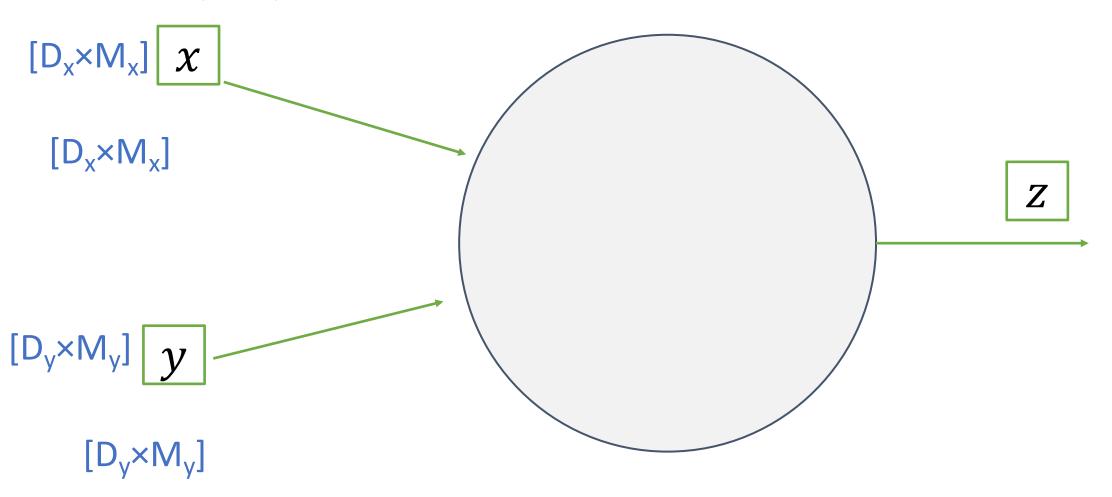
$$(elementwise)$$

$$[0]$$

$$[3]$$

4D 
$$dL/dx$$
:  $[dy/dx] [dL/dy]$ 

$$\begin{bmatrix} 4 \end{bmatrix} \leftarrow \begin{bmatrix} \frac{\partial L}{\partial x} \end{bmatrix}_i = \begin{cases} \left(\frac{\partial L}{\partial y}\right)_i, & if \ x_i > 0 \leftarrow [-1] \leftarrow \\ 0, & otherwise \end{cases}$$
 Upstream gradient 
$$\begin{bmatrix} 0 \end{bmatrix} \leftarrow \begin{bmatrix} 9 \end{bmatrix} \leftarrow \begin{bmatrix} 9$$



 $[D_x \times M_x]$ 

Loss L still a scalar!



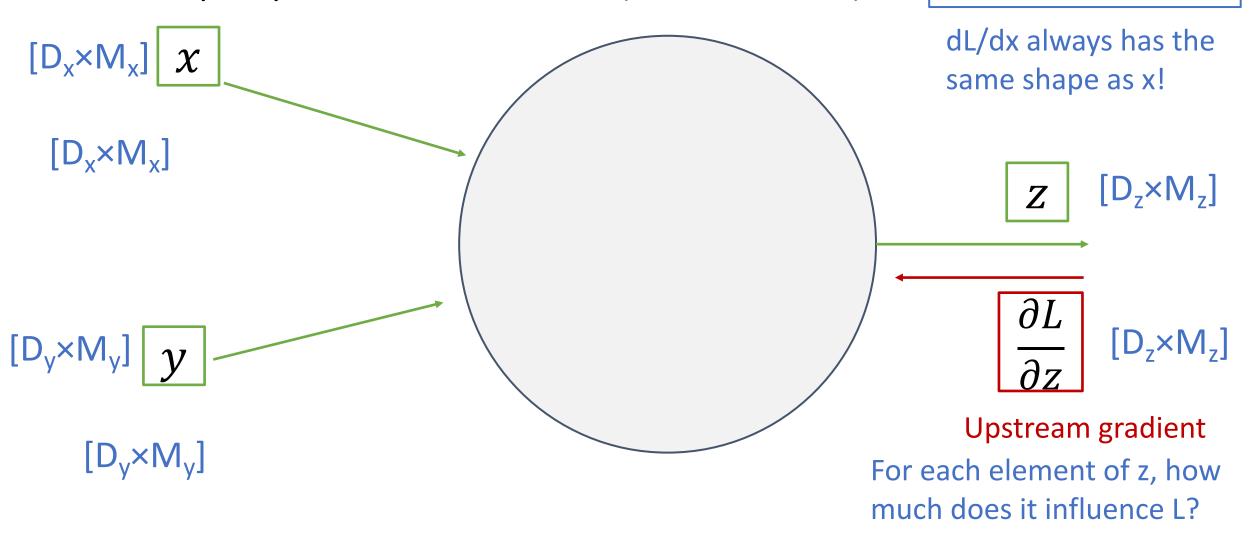
$$\begin{bmatrix} D_{x} \times M_{x} \end{bmatrix}$$

$$\begin{bmatrix} D_{y} \times M_{y} \end{bmatrix} y$$

$$\begin{bmatrix} D_{y} \times M_{y} \end{bmatrix}$$

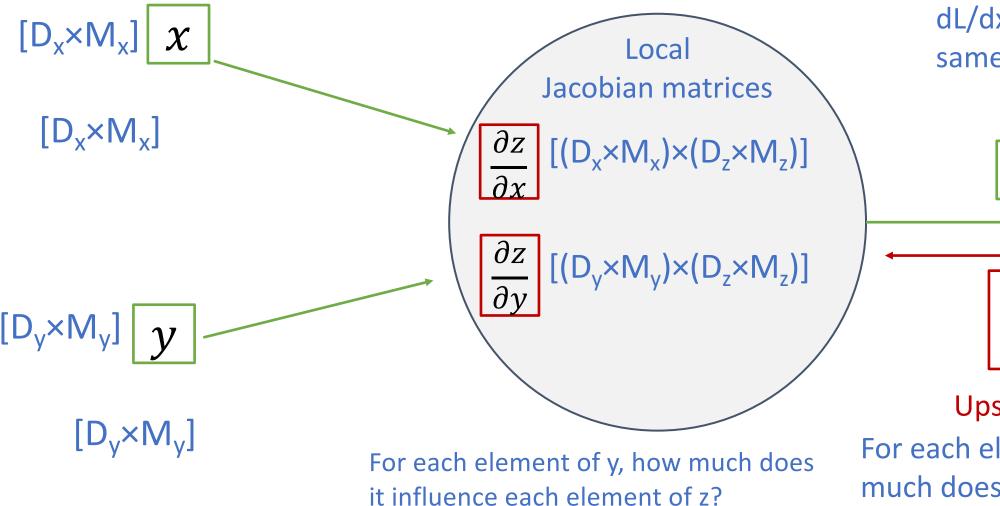
Justin Johnson Lecture 6 - 84 January 26, 2022

Loss L still a scalar!

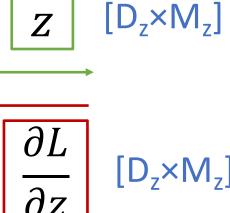


Justin Johnson Lecture 6 - 85 January 26, 2022

Loss L still a scalar!



dL/dx always has the same shape as x!

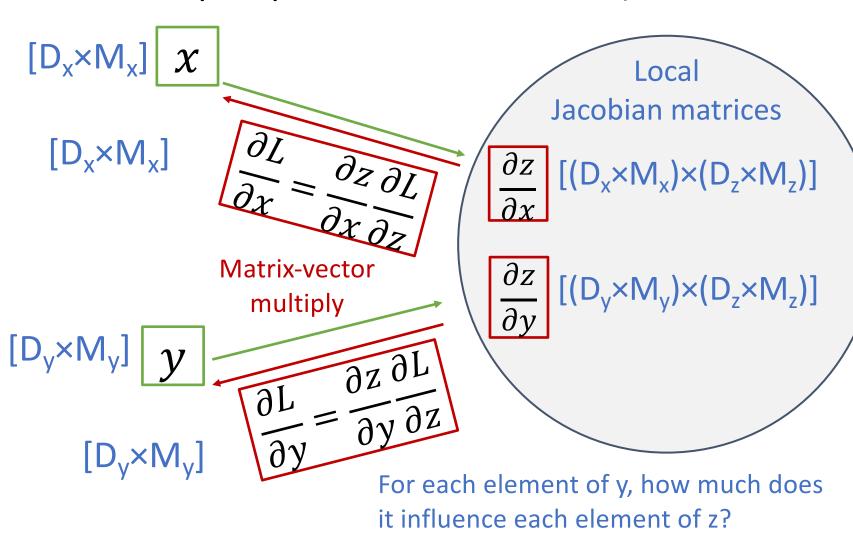


Upstream gradient

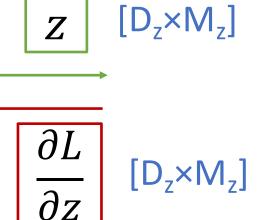
For each element of z, how much does it influence L?

Justin Johnson Lecture 6 - 86 January 26, 2022

Loss L still a scalar!



dL/dx always has the same shape as x!



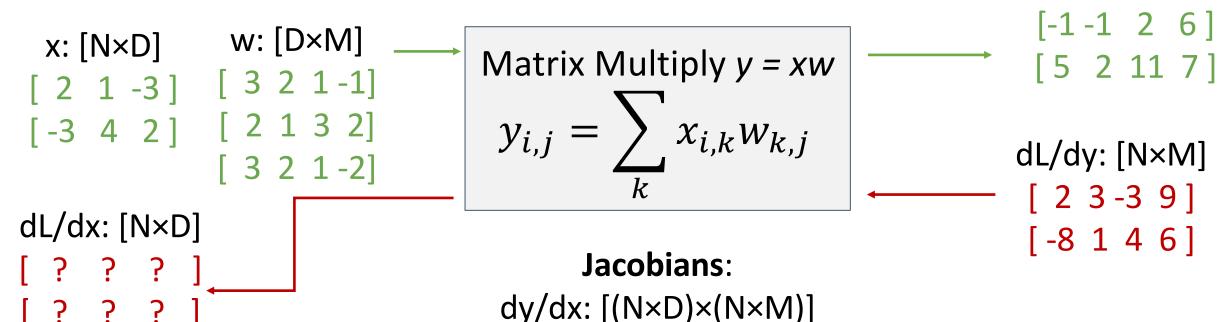
Upstream gradient

For each element of z, how much does it influence L?

Justin Johnson Lecture 6 - 87 January 26, 2022

x: [N×D] w: [D×M]   
[ 2 1 -3 ] [ 3 2 1 -1]   
[ -3 4 2 ] [ 2 1 3 2]   
[ 3 2 1 -2] 
$$y_{i,j} = \sum_{k} x_{i,k} w_{k,j}$$

y: [N×M]



y: [N×M]

For a neural net we may have N=64, D=M=4096
Each Jacobian takes 256 GB of memory! Must work with them implicitly!

 $dy/dw: [(D\times M)\times (N\times M)]$ 

```
w: [D×M]
 x: [N \times D]
                                Matrix Multiply y = xw
 [2] 1 -3] [3 2 1-1]
                                 y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
              [ 3 2 1-2]
dL/dx: [N×D]
                                  Local Gradient Slice:
                                         dy/dx_{1.1}
                                       [;;;]
dL/dx_{1.1}
                                        [;;;]
= (dy/dx_{1.1}) \cdot (dL/dy)
```

 $dL/dx_{1,1}$ 

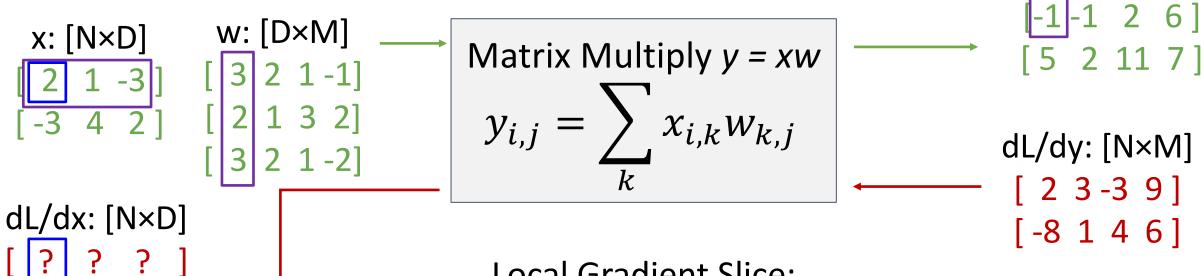
 $= (dy/dx_{1.1}) \cdot (dL/dy)$ 

```
[-1]-1 2 6]
           w: [D×M]
 x: [N×D]
                              Matrix Multiply y = xw
                                                                 [5 2 11 7]
 [2] 1 -3] [3 2 1-1]
                              y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
                                                                dL/dy: [N×M]
            [321-2]
                                                                 [23-39]
dL/dx: [N×D]
                                                                 [-8146]
                               Local Gradient Slice:
                                    dy/dx_{1.1}
                         dy_{1,1}/dx_{1,1} ????
```

y: [N×M]

 $dL/dx_{1,1}$ 

 $= (dy/dx_{1,1}) \cdot (dL/dy)$ 



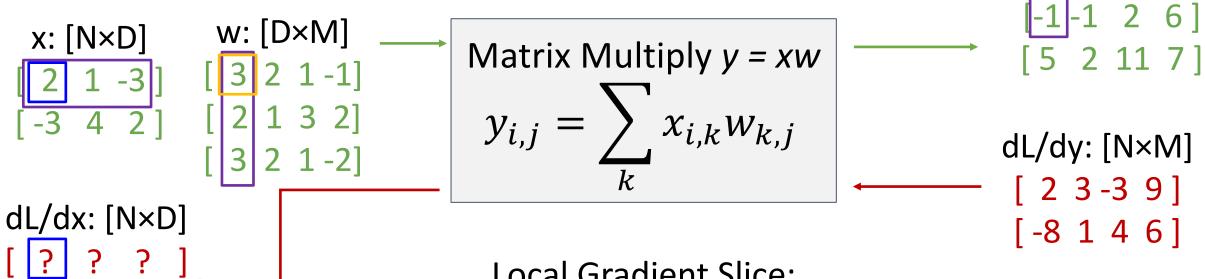
y: [N×M]

$$dy/dx_{1,1}$$

$$dy_{1,1}/dx_{1,1} ?????]$$

$$[?????]$$

$$y_{1,1} = x_{1,1}w_{1,1} + x_{1,2}w_{2,1} + x_{1,3}w_{3,1}$$



$$dL/dx_{1,1}$$
=  $(dy/dx_{1,1}) \cdot (dL/dy)$ 

y: [N×M]

$$dy/dx_{1,1}$$

$$dy/dx_{1,1}$$

$$[????]$$

$$y_{1,1} = x_{1,1}w_{1,1} + x_{1,2}w_{2,1} + x_{1,3}w_{3,1}$$

$$=> dy_{1,1}/dx_{1,1} = w_{1,1}$$

 $= (dy/dx_{1.1}) \cdot (dL/dy)$ 

```
w: [D×M]
 x: [N \times D]
                                 Matrix Multiply y = xw
 [2] 1 -3] [3 2 1-1]
                                  y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
              [321-2]
dL/dx: [N×D]
                                   Local Gradient Slice:
                                        dy/dx_{1.1}
                            dy_{1,2}/dx_{1,1} [3????]
dL/dx_{1,1}
```

Justin Johnson Lecture 6 - 95 January 26, 2022

y: [N×M]

[5 2 11 7]

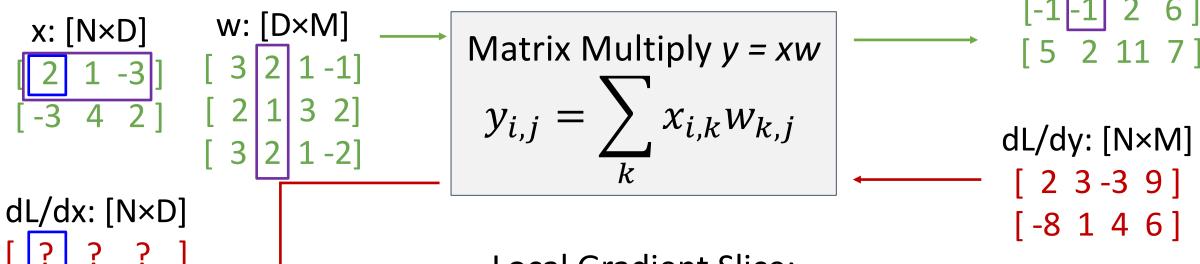
dL/dy: [N×M]

[23-39]

[-8146]

 $dL/dx_{1,1}$ 

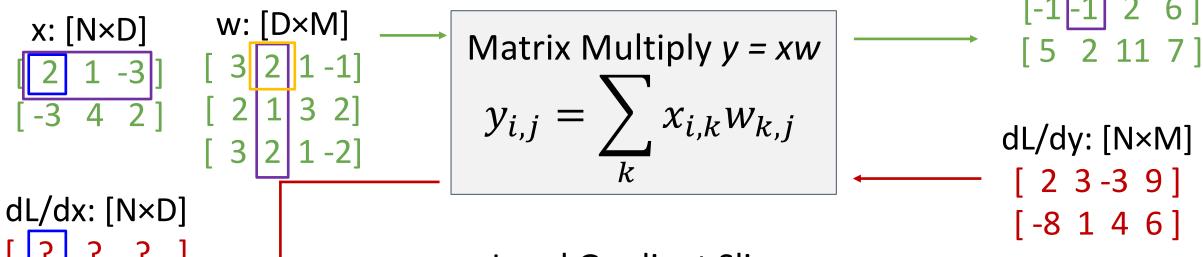
 $= (dy/dx_{1,1}) \cdot (dL/dy)$ 



#### **Local Gradient Slice:**

y: [N×M]

$$\frac{dy/dx_{1,1}}{dy_{1,2}/dx_{1,1}} \begin{bmatrix} 3 & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$
$$\begin{bmatrix} ? & ? & ? & ? \end{bmatrix}$$
$$y_{1,2} = x_{1,1}w_{1,2} + x_{1,2}w_{2,2} + x_{1,3}w_{3,2}$$



y: [N×M]

#### Local Gradient Slice:

 $dL/dx_{1.1}$ 

 $= (dy/dx_{1,1}) \cdot (dL/dy)$ 

$$\frac{dy/dx_{1,1}}{dy_{1,2}/dx_{1,1}} [32??]$$

$$[????]$$

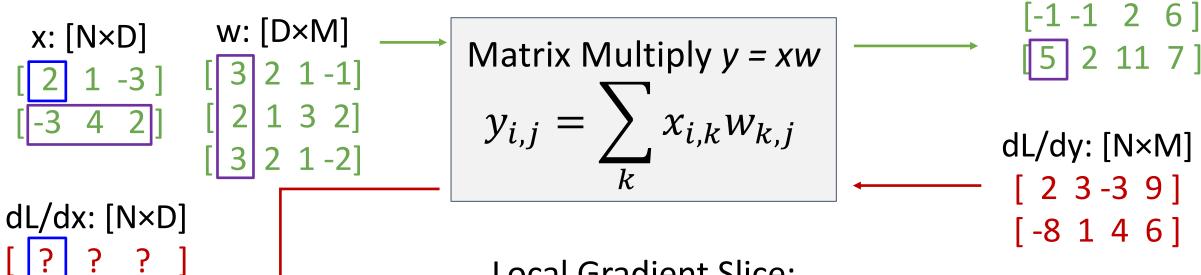
$$y_{1,2} = x_{1,1} w_{1,2} + x_{1,2} w_{2,2} + x_{1,3} w_{3,2}$$

$$=> dy_{1,2}/dx_{1,1} = w_{1,2}$$

y: [N×M]

$$dL/dx_{1,1}$$
=  $(dy/dx_{1,1}) \cdot (dL/dy)$ 

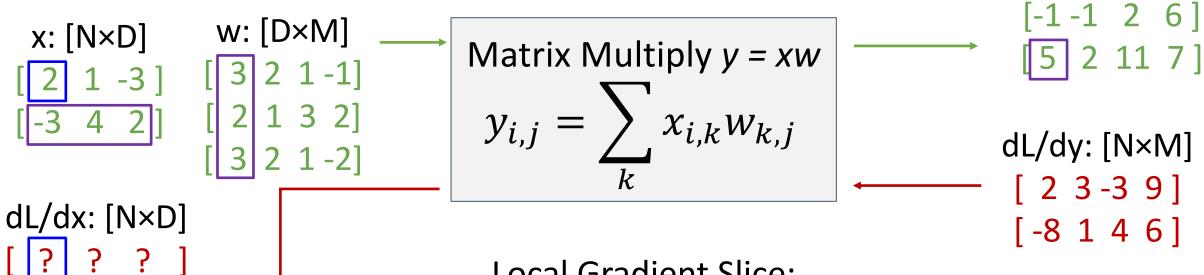
 $\frac{dy/dx_{1,1}}{dy_{1,2}/dx_{1,1}} [3 2 1 - 1]$  [????]



$$dL/dx_{1,1}$$
=  $(dy/dx_{1,1}) \cdot (dL/dy)$ 

y: [N×M]

$$\frac{dy/dx_{1,1}}{dy_{1,2}/dx_{1,1}} \begin{bmatrix} 3 & 2 & 1 & -1 \end{bmatrix}$$
$$\frac{?}{?} ? ? ? ]$$
$$y_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$$



$$dL/dx_{1,1}$$
=  $(dy/dx_{1,1}) \cdot (dL/dy)$ 

y: [N×M]

$$\frac{dy/dx_{1,1}}{dy_{1,2}/dx_{1,1}} \begin{bmatrix} 3 & 2 & 1 & -1 \end{bmatrix}$$

$$\frac{0}{2} ? ? ? ]$$

$$y_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$$

$$=> dy_{2,1}/dx_{1,1} = 0$$

```
[-1 - 1 \ 2 \ 6]
            w: [D×M]
 x: [N×D]
                               Matrix Multiply y = xw
                                                                     [5 2 11 7]
 [2] 1 -3] [3 2 1-1]
                                y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
                                                                   dL/dy: [N×M]
             [321-2]
                                                                    [23-39]
dL/dx: [N×D]
                                                                    [-8146]
                                 Local Gradient Slice:
                                      dy/dx_{1.1}
                           dy_{1,2}/dx_{1,1} [3 2 1-1]
dL/dx_{1,1}
```

January 26, 2022

y: [N×M]

 $= (dy/dx_{1.1}) \cdot (dL/dy)$ 

```
w: [D×M]
 x: [N \times D]
                                  Matrix Multiply y = xw
 [2] 1 -3] [3 2 1-1]
                                   y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
               [ 3 2 1-2]
dL/dx: [N×D]
                                           dy/dx_{1.1}
                                         [3 2 1-1]
dL/dx_{1.1}
                                          [0 \ 0 \ 0 \ 0]
= (dy/dx_{1.1}) \cdot (dL/dy)
```

```
y: [N×M]
 [-1 - 1 \ 2 \ 6]
 [5 2 11 7]
dL/dy: [N×M]
[23-39]
[-8146]
```

```
w: [D×M]
  x: [N \times D]
                                      Matrix Multiply y = xw
                                       y_{i,j} = \sum x_{i,k} w_{k,j}
                [321-2]
dL/dx: [N×D]
                                               dy/dx_{1.1}
dL/dx_{1.1}
= (dy/dx_{1.1}) \cdot (dL/dy)
= (w_{1:}) \cdot (dL/dy_{1:})
= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0
```

[-8146]

```
w: [D×M]
 x: [N \times D]
             [ 3 2 1-1]
 [ 2 1 -3]
             [2132]
dL/dx: [N×D]
dL/dx_{2,3}
```

=  $(dy/dx_{2.3}) \cdot (dL/dy)$ 

Matrix Multiply y = xw

$$y_{i,j} = \sum_{k} x_{i,k} w_{k,j}$$

y: [N×M] [-1 -1 2 6] [5 2 11 7]

dL/dy: [N×M] ----- [ 2 3 -3 9 ] [-8 1 4 6]

```
w: [D×M]
 x: [N \times D]
                                   Matrix Multiply y = xw
              [321-1]
 [ 2 1 -3 ]
                                    y_{i,j} = \sum x_{i,k} w_{k,j}
             [ 2 1 3 2]
dL/dx: [N×D]
[ 0 ? ? ]
                                     Local Gradient Slice:
                                            dy/dx_{2.3}
dL/dx_{2.3}
= (dy/dx_{2.3}) \cdot (dL/dy)
= (w_{3:}) \cdot (dL/dy_{2:})
= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30
```

23-39]

```
w: [D×M]
 x: [N×D]
                               Matrix Multiply y = xw
[21-3][321-1]
                               y_{i,j} = \sum x_{i,k} w_{k,j}
[-3 4 2] [2 1 3 2]
             [321-2]
dL/dx: [N×D]
[ 0 16 -9 ]
[-24 9 -30]
dL/dx_{i,i}
= (dy/dx_{i,i}) \cdot (dL/dy)
```

 $= (w_{i::}) \cdot (dL/dy_{i::})$ 

```
y: [N×M]
[-1-1 2 6]
[5 2 11 7]
dL/dy: [N×M]
```

```
[-8 1 4 6]
```

[23-39]

```
w: [D×M]
 x: [N×D]
[21-3][321-1]
[-3 4 2] [2 1 3 2]
             [ 3 2 1-2]
dL/dx: [N×D]
[ 0 16 -9 ]
[-24 9 -30]
dL/dx_{i,i}
= (dy/dx_{i,i}) \cdot (dL/dy)
```

 $= (w_{i::}) \cdot (dL/dy_{i::})$ 

Matrix Multiply y = xw  $y_{i,j} = \sum_{k} x_{i,k} w_{k,j}$ 

$$dL/dx = (dL/dy) w^T$$
  
[N x D] [N x M] [M x D]

[-8146]

Easy way to remember: It's the only way the shapes work out!

```
w: [D×M]
 x: [N×D]
             [321-1]
 [ 2 1 -3]
              [ 2 1 3 2]
\begin{bmatrix} -3 & 4 & 2 \end{bmatrix}
               [ 3 2 1-2]
dL/dx: [N×D]
[ 0 16 -9 ]
[-24 9 -30]
```

Matrix Multiply y = xw  $y_{i,j} = \sum_{i} x_{i,k} w_{k,j}$ 

$$dL/dx = (dL/dy) w^T$$
  
[N x D] [N x M] [M x D]

$$dL/dw = x^{T} (dL/dy)$$
  
[D x M] [D x N] [N x M]

y: [N×M]

[-1-1 2 6]

[5 2 11 7]

dL/dy: [N×M]

[ 2 3-3 9]

[-8 1 4 6]

Easy way to remember: It's the only way the shapes work out!

Backpropagation: Another View

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} X_2 \xrightarrow{f_3} X_3 \xrightarrow{f_4} L$$
 $D_0 D_1 D_2 D_3 scalar$ 

The chain rule 
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left(\frac{\partial L}{\partial x_3}\right)$$

## Backpropagation: Another View

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} X_2 \xrightarrow{f_3} X_3 \xrightarrow{f_4} L$$
 $D_0 D_1 D_2 D_3 scalar$ 

Matrix multiplication is associative: we can compute products in any order

Thain rule 
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left(\frac{\partial L}{\partial x_3}\right) \left(\frac{\partial L}{\partial x_$$

## Reverse-Mode Automatic Differentiation

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} X_2 \xrightarrow{f_3} X_3 \xrightarrow{f_4} L$$
 $D_0 D_1 D_2 D_3 scalar$ 

Matrix multiplication is **associative**: we can compute products in any order Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule 
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left(\frac{\partial L}{\partial x_3}\right)$$

$$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$$

### Reverse-Mode Automatic Differentiation

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} X_2 \xrightarrow{f_3} X_3 \xrightarrow{f_4} L$$
 $D_0 D_1 D_2 D_3 scalar$ 

Matrix multiplication is **associative**: we can compute products in any order Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule 
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left(\frac{\partial L}{\partial x_3}\right)$$

Compute grad of scalar <u>output</u> w/respect to all vector <u>inputs</u>

$$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$$

What if we want grads of scalar input w/respect to vector outputs?

## Forward-Mode Automatic Differentiation

Chain rule 
$$\frac{\partial x_3}{\partial a} = \left(\frac{\partial x_0}{\partial a}\right) \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left(\frac{\partial x_3}{\partial x_2}\right) \left[D_0 \times D_1\right] \left[D_1 \times D_2\right] \left[D_2 \times D_3\right]$$

## Forward-Mode Automatic Differentiation

Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

Chain rule 
$$\frac{\partial x_3}{\partial a} = \left(\frac{\partial x_0}{\partial a}\right) \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right)$$

$$[D_0] [D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3]$$

## Forward-Mode Automatic Differentiation

Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

Beta implementation in PyTorch! <a href="https://pytorch.org/tutorials/intermediate/forward ad usage.html">https://pytorch.org/tutorials/intermediate/forward ad usage.html</a>

Chain rule 
$$\frac{\partial x_3}{\partial a} = \left(\frac{\partial x_0}{\partial a}\right) \left(\frac{\partial x_1}{\partial x_0}\right) \left(\frac{\partial x_2}{\partial x_1}\right) \left(\frac{\partial x_3}{\partial x_2}\right)$$

$$[D_0] [D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3]$$

You can also implement forward-mode AD using two calls to reverse-mode AD! (Inefficient but elegant)

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L$$
 $D_0 \qquad D_1 \qquad \text{scalar}$ 

$$\frac{\partial^2 L}{\partial x_0^2} \quad \begin{array}{l} \text{Hessian matrix} \\ \text{H of second} \\ \text{derivatives.} \end{array}$$
 
$$[\mathsf{D_0} \ \mathsf{x} \ \mathsf{D_0}]$$

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L$$
 $D_0 \qquad D_1 \qquad \text{scalar}$ 

$$\begin{array}{c} \partial^2 L & \text{Hessian matrix} \\ \overline{\partial x_0^2} & \text{H of second} \\ \text{derivatives.} \\ [\mathsf{D_0} \times \mathsf{D_0}] \end{array}$$

Hessian / vector multiply 
$$\frac{\partial^2 L}{\partial x_0^2} v$$
 
$$[\mathsf{D}_0 \times \mathsf{D}_0] \ [\mathsf{D}_0]$$

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L$$
 $D_0 \qquad D_1 \qquad \text{scalar}$ 

$$\begin{array}{c} \partial^2 L & \text{Hessian matrix} \\ \overline{\partial x_0^2} & \text{H of second} \\ \text{derivatives.} \\ [\mathsf{D_0} \ \mathsf{x} \ \mathsf{D_0}] \end{array}$$

**Hessian / vector multiply** 

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right] \stackrel{\text{(if v doesn't depend on } x_0)}{\det \text{(If p doesn't depend on } x_0)}$$

$$\left[ D_0 \times D_0 \right] \left[ D_0 \right]$$

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\cdot v} (dL/dx_0) \cdot v$$
 $D_0 \xrightarrow{D_1} scalar$ 

$$\begin{array}{ll} \partial^2 L & \text{Hessian matrix} \\ \overline{\partial x_0^2} & \text{H of second} \\ \text{derivatives.} \\ [\mathsf{D_0} \, \mathsf{x} \, \mathsf{D_0}] \end{array}$$

**Hessian / vector multiply** 

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right] \stackrel{\text{(if v doesn't depend on } x_0)}{\text{(if p doesn't depend on } x_0)}$$

$$\left[ D_0 \times D_0 \right] \left[ D_0 \right]$$

Justin Johnson Lecture 6 - 119 January 26, 2022

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\cdot v} (dL/dx_0) \cdot v$$
 $D_0 \xrightarrow{D_1} scalar \xrightarrow{D_1} D_0 scalar$ 

#### Backprop!

$$\frac{\partial^2 L}{\partial x_0^2}$$
 Hessian matrix H of second derivatives.  $[\mathsf{D}_0\,\mathsf{x}\,\mathsf{D}_0]$ 

**Hessian / vector multiply** 

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right] \stackrel{\text{(if v doesn't depend on } x_0)}{\text{(if p doesn't depend on } x_0)}$$

$$[D_0 \times D_0] [D_0]$$

Justin Johnson Lecture 6 - 120 January 26, 2022

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\cdot \vee} (dL/dx_0) \cdot v$$
 $D_0 \xrightarrow{D_1} scalar$ 

#### Backprop!

 $\frac{\partial^2 L}{\partial x_0^2}$  Hessian matrix derivatives.

$$[D_0 \times D_0]$$

**Hessian / vector multiply** 

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \right].$$

$$[D_0 \times D_0] [D_0]$$

This is implemented in PyTorch / Tensorflow!

$$[v]$$
 (if v doesn't depend on  $x_0$ )

$$X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\text{norm}} |dL/dx_0|^2$$
 $D_0 \text{ scalar}$ 

Backprop!

This is implemented in PyTorch / Tensorflow!

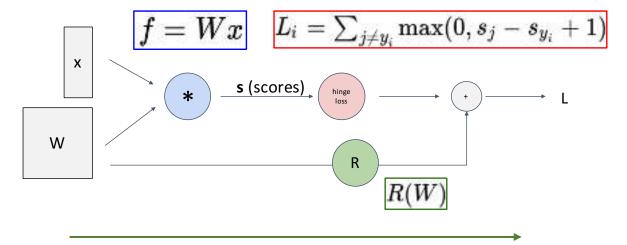
**Example:** Regularization to penalize the <u>norm</u> of the gradient

$$R(W) = \left\| \frac{\partial L}{\partial W} \right\|_{2}^{2} = \left( \frac{\partial L}{\partial W} \right) \cdot \left( \frac{\partial L}{\partial W} \right) \quad \frac{\partial}{\partial x_{0}} \left[ R(W) \right] = 2 \left( \frac{\partial^{2} L}{\partial x_{0}^{2}} \right) \left( \frac{\partial L}{\partial x_{0}} \right)$$

Gulrajani et al, "Improved Training of Wasserstein GANs", NeurIPS 2017

## Summary

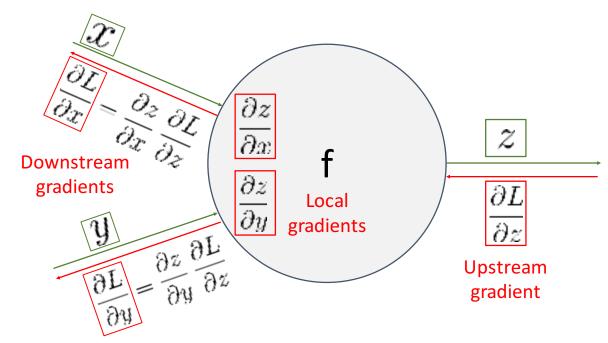
Represent complex expressions as **computational graphs** 



Forward pass computes outputs

Backward pass computes gradients

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients** 



## Summary

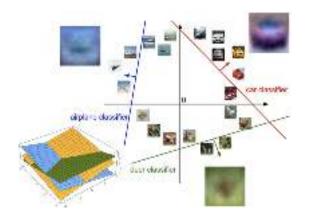
Backprop can be implemented with "flat" code where the backward pass looks like forward pass reversed (Use this for A2!)

```
def f(w0, x0, w1, x1, w2):
 s0 = w0 * x0
 s1 = w1 * x1
 s2 = s0 + s1
 53 = 52 + w2
 L = sigmoid(s3)
  grad L = 1.0
  grad_s3 = grad_L * (1 - L) * L
 grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad x0 = grad s0 * w0
```

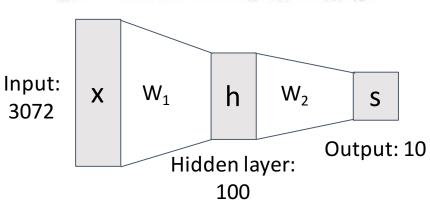
Backprop can be implemented with a modular API, as a set of paired forward/backward functions (We will do this on A3!)

```
class Multiply(torch.autograd.Function):
 @staticmethod
 def forward(ctx, x, y):
   ctx.save_for_backward(x, y)
   z = x * y
   return z
 @staticmethod
 def backward(ctx, grad_z):
   x, y = ctx.saved_tensors
   grad_x = y * grad_z # dz/dx * dL/dz
   grad_y = x * grad_z # dz/dy * dL/dz
    return grad_x, grad_y
```

#### f(x,W) = Wx



$$f = W_2 \max(0, W_1 x)$$



# **Problem**: So far our classifiers don't respect the spatial structure of images!

#### Stretch pixels into column



Justin Johnson Lecture 6 - 125 January 26, 2022

## Next time: Convolutional Neural Networks