Lecture 4:
Neural Networks and Backpropagation

Administrative: Project Proposal

Due Fri 4/25

TA expertise is posted on the webpage.

(http://cs231n.stanford.edu/office_hours.html)

Administrative: Discussion Section

Discussion section tomorrow

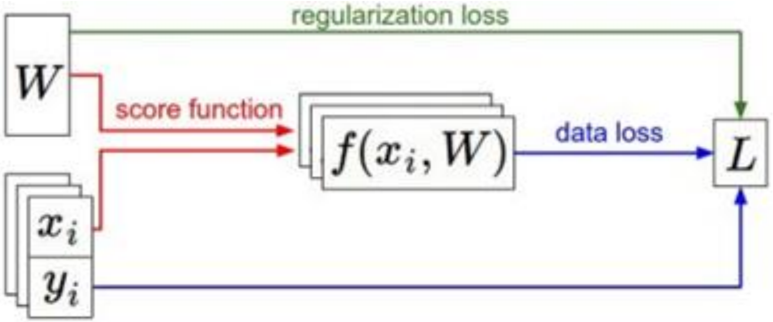
(led by Matthew Jin, With Emily Jin's help):

Backpropagation

Recap

Recap

- We have some dataset of (x,y)
- We have a score function: $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a loss function:

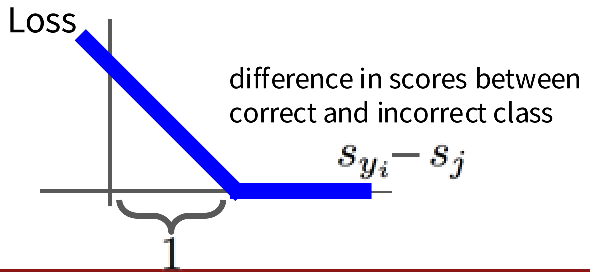


$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ Softmax

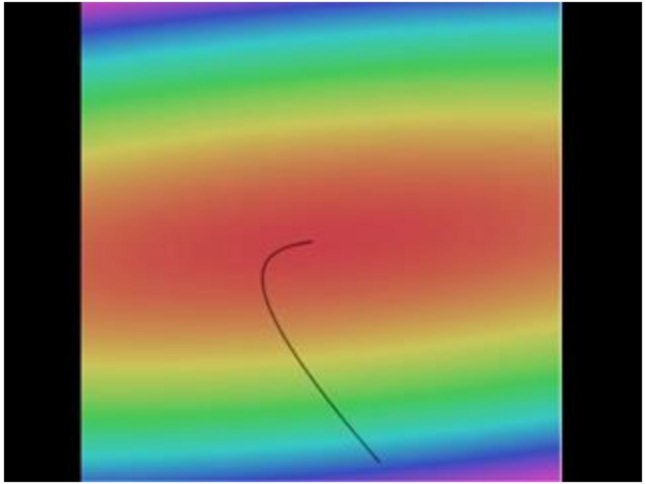
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SVM/hinge Loss (refer to Lecture 2 reading assignment)



Finding the best W: Optimize with Gradient Descent




```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Gradient descent

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

Numerical gradient: slow ☹, approximate ☹, easy to write ☺
Analytic gradient: fast ☺, exact ☺, error-prone ☹

In practice: Derive analytic gradient, check your implementation with numerical gradient

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

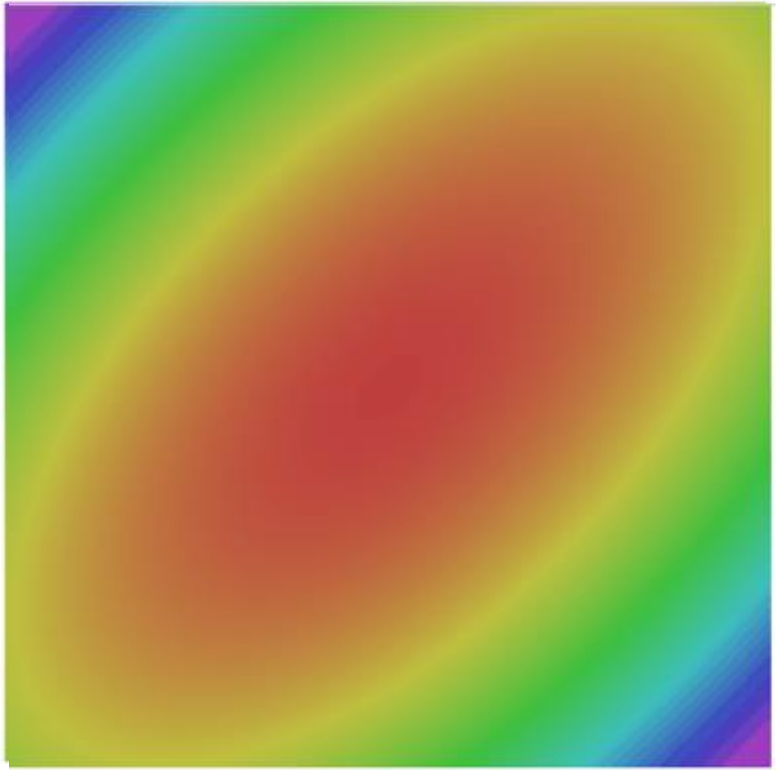
Full sum is expensive when N is large!

Approximate sum using a minibatch of examples 32 / 64 / 128 / 256

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Last time: fancy optimizers



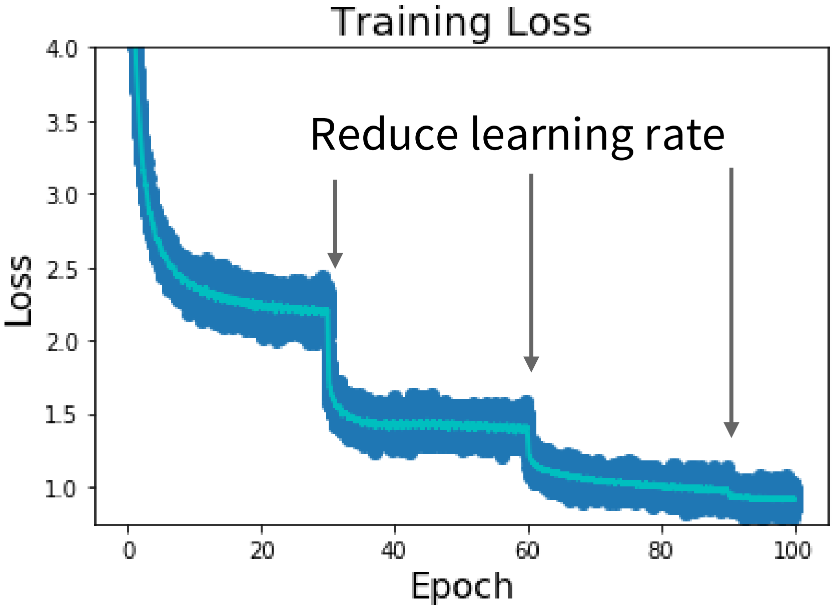
SGD

SGD+Momentum

RMSProp

Adam

Last time: learning rate scheduling



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Today:

Deep Learning

DALL-E 2

“Teddy bears working on new AI research on the moon in the 1980s.”

“Rabbits attending a college seminar on human anatomy.”

“A wise cat meditating in the Himalayas searching for enlightenment.”

Image source: Sam Altman, https://openai.com/dall-e-2/, https://twitter.com/sama/status/1511724264629678084

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

an espresso machine that makes coffee from human souls, artstation

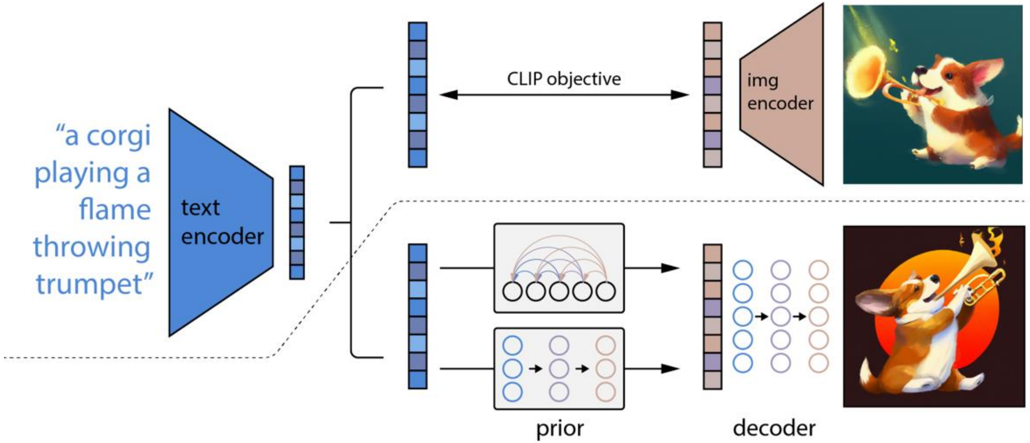
panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

a dolphin in an astronaut suit on saturn, artstation

a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese

a teddybear on a skateboard in times square



Ramesh et al., Hierarchical Text-Conditional Image Generation with CLIP Latents, 2022.

DALL-E 3

In a fantastical setting, a highly detailed furry humanoid skunk with piercing eyes confidently poses in a medium shot, wearing an animal hide jacket. The artist has masterfully rendered the character in digital art, capturing the intricate details of fur and clothing texture.

Betker, James, et al. "Improving image generation with better captions." Computer Science. https://cdn.openai.com/papers/dall-e-3.pdf (2023).

DALL-E 3

An illustration from a graphic novel. A bustling city street under the shine of a full moon. The sidewalks bustling with pedestrians enjoying the nightlife. At the corner stall, a young woman with fiery red hair, dressed in a signature velvet cloak, is haggling with the grumpy old vendor. The grumpy vendor, a tall, sophisticated man wearing a sharp suit, who sports a noteworthy mustache is animatedly conversing on his steampunk telephone.

Betker, James, et al. "Improving image generation with better captions." Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf (2023).

GPT-4

User What is unusual about this image?

Source: Barnorama

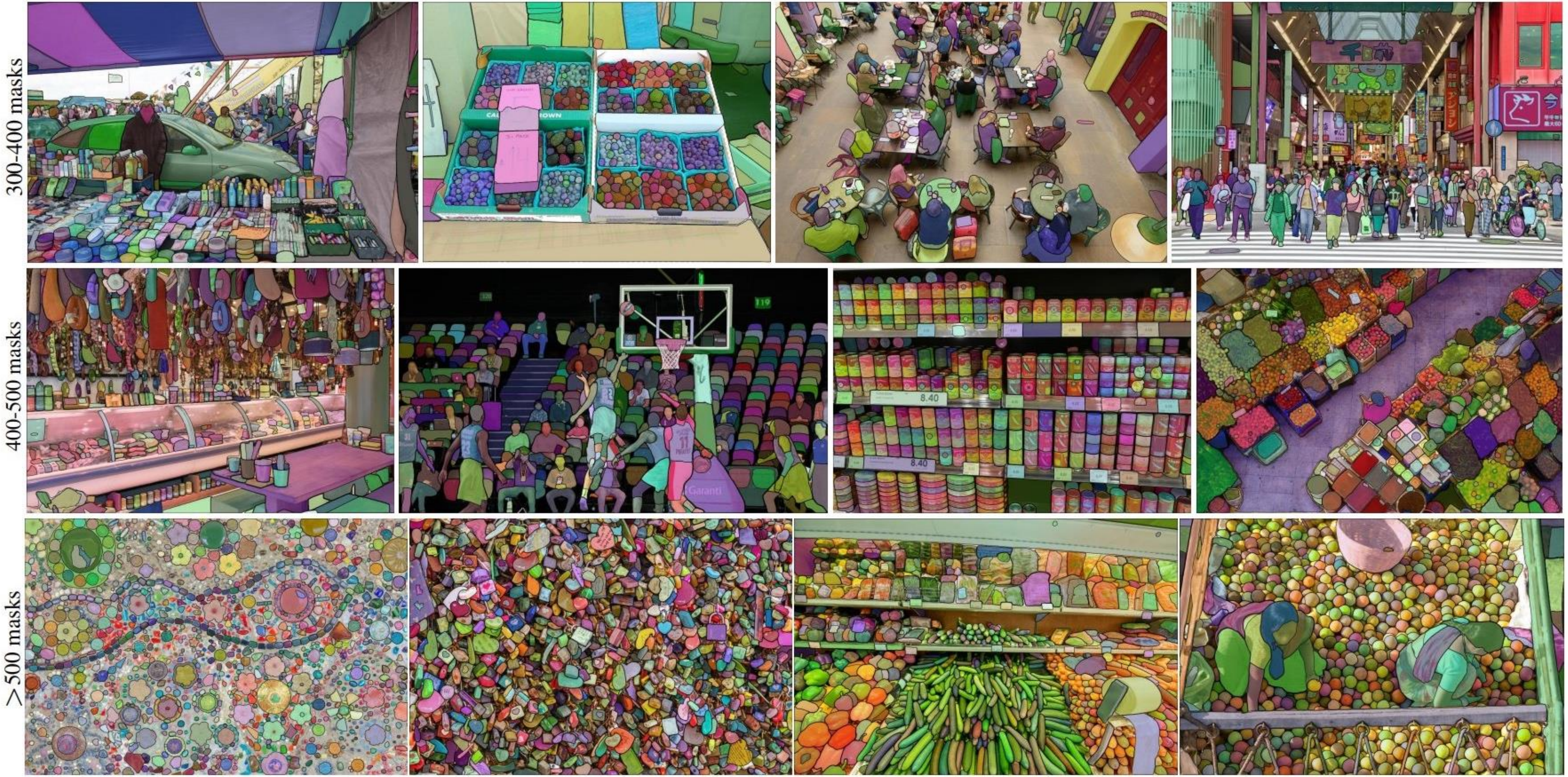
GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

User Can you explain this meme?

GPT-4 This meme is a joke that combines two unrelated things: pictures of the earth from space and chicken nuggets.
The text of the meme suggests that the image below is a beautiful picture of the earth from space. However, the image is actually of chicken nuggets arranged to vaguely resemble a map of the world.
The humor in this meme comes from the unexpected juxtaposition of the text and the image. The text sets up an expectation of a majestic image of the earth, but the image is actually something mundane and silly.

Image source: https://openai.com/research/gpt-4

Segment Anything Model (SAM)



Sora

- Animating Images (generated by DALL-E)

- Video-to-video editing

A Shiba Inu dog wearing a beret and black turtleneck.

put the video in space with a rainbow road

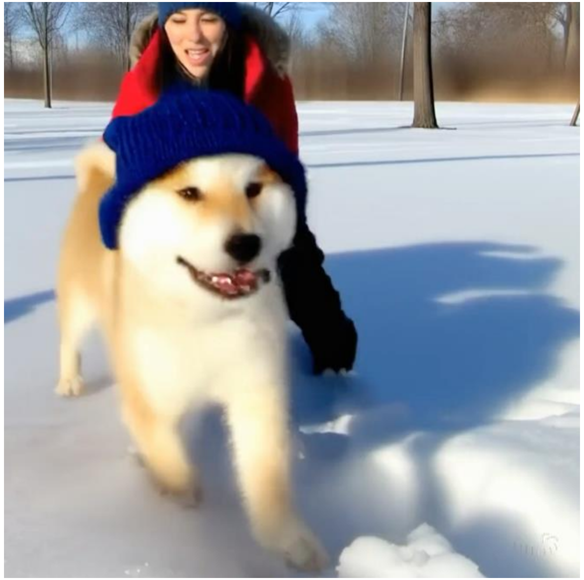
change the video setting to be different than a mountain? perhaps joshua tree

https://openai.com/research/video-generation-models-as-world-simulators

Sora

- More compute

Base Compute



4x Compute

32x Compute

https://openai.com/research/video-generation-models-as-world-simulators

Neural Networks

Neural networks: the original linear classifier

(Before) Linear score function: $$f = Wx$$

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

Neural networks: 2 layers

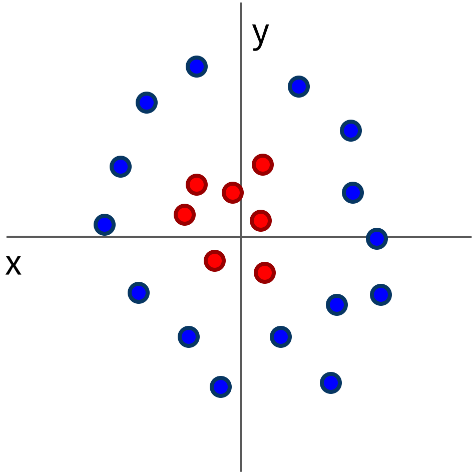
(Before) Linear score function: $$f = Wx$$

(Now) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

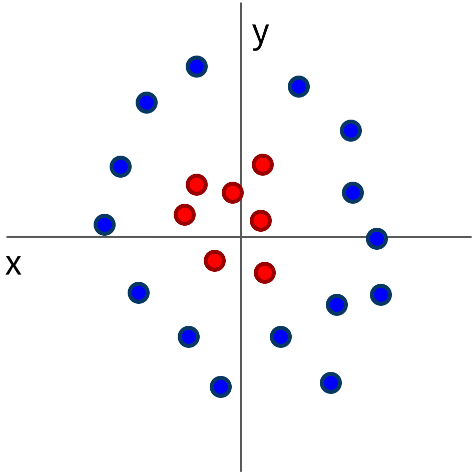
(In practice we will usually add a learnable bias at each layer as well)

Why do we want non-linearity?

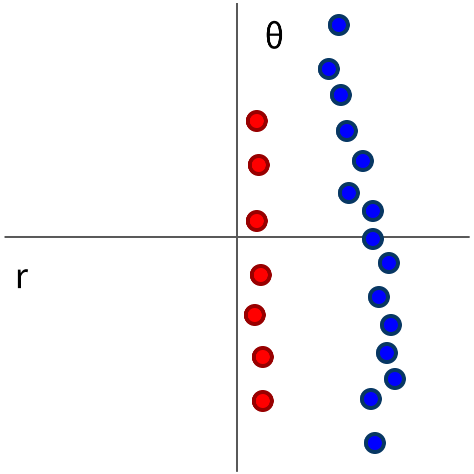


Cannot separate red and blue points with linear classifier

Why do we want non-linearity?



$$f(x, y) = (r(x, y), \theta(x, y))$$



Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier

Neural networks: also called fully connected network

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

"Neural Network" is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

(In practice we will usually add a learnable bias at each layer as well)

Neural networks: 3 layers

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

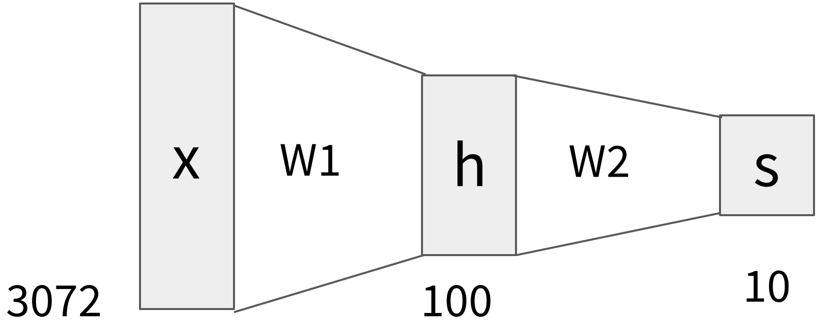
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

(In practice we will usually add a learnable bias at each layer as well)

Neural networks: hierarchical computation

(Before) Linear score function: $$f = Wx$$

(Now) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$

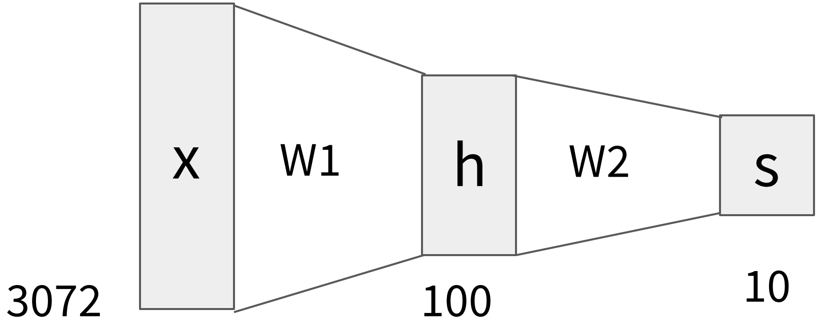


$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural networks: learning 100s of templates

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Learn 100 templates instead of 10.

Share templates between classes

Neural networks: why is max operator important?

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \boxed{\max(0,} W_1 x)$

The function $\max(0, z)$ is called the activation function.

Q: What if we try to build a neural network without one?

$$f = W_2 W_1 x$$

Neural networks: why is max operator important?

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \boxed{\max(0,} W_1 x)$

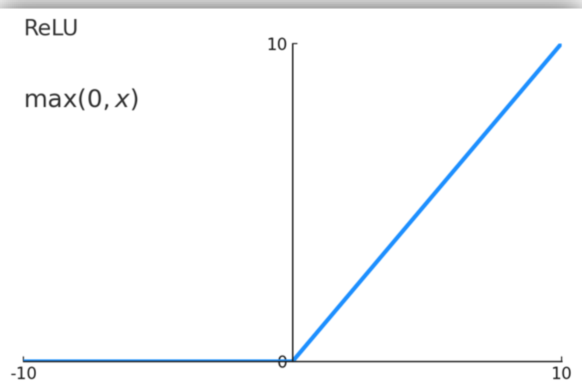
The function $\max(0, z)$ is called the activation function.
Q: What if we try to build a neural network without one?

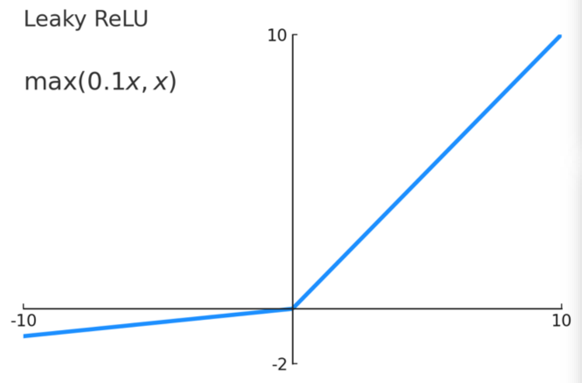
$$f = W_2 W_1 x \qquad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

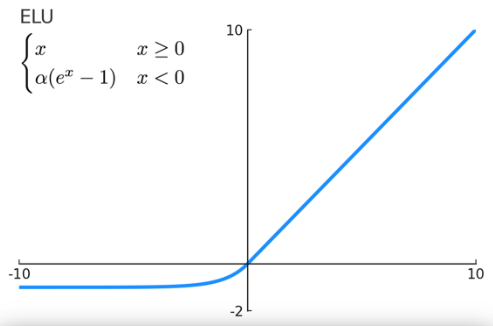
A: We end up with a linear classifier again!

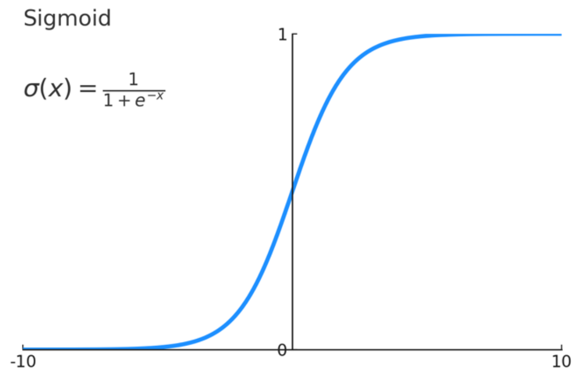
Activation functions

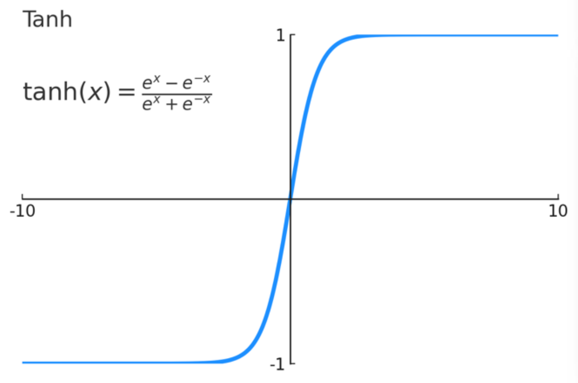
ReLU is a good default choice for most problems

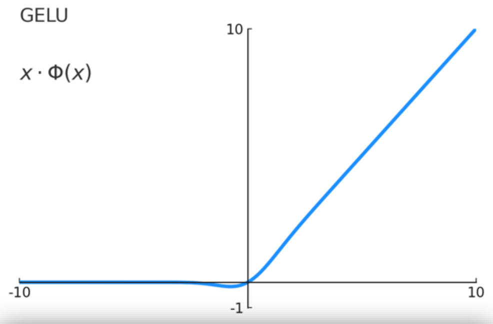


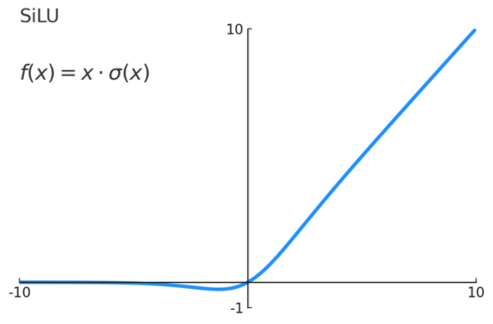




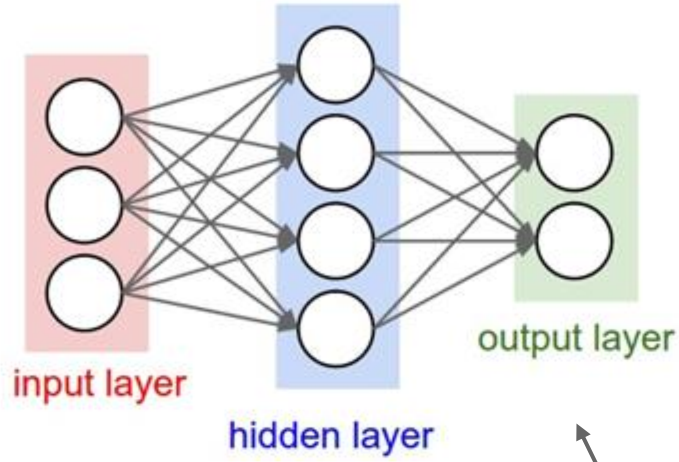


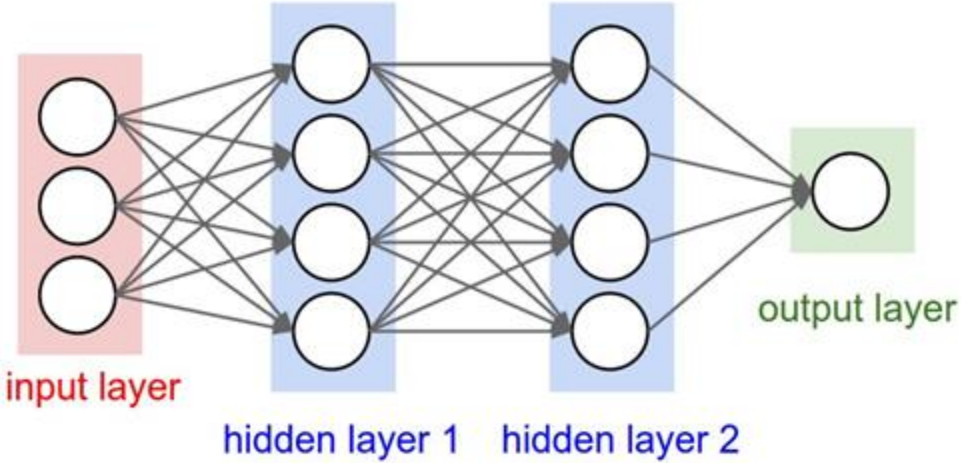






Neural networks: Architectures



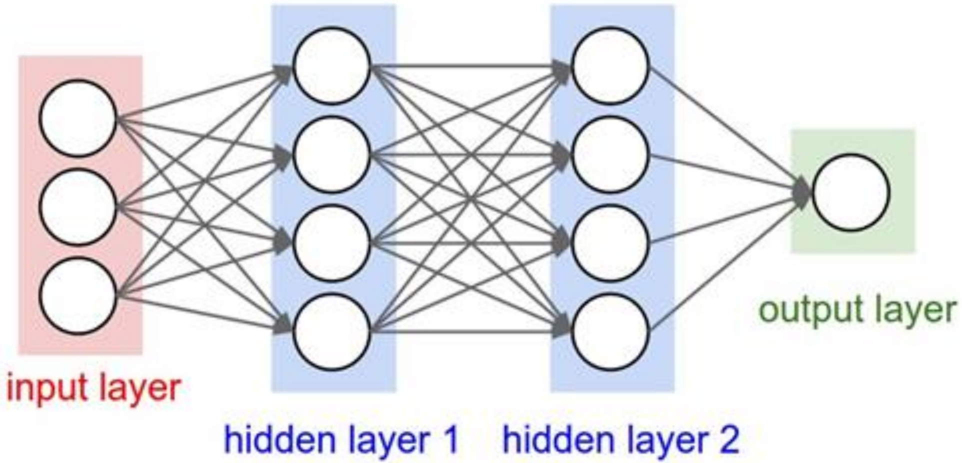


“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

Example feed-forward computation of a neural network




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Forward pass

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Forward pass

Calculate the analytical gradients

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

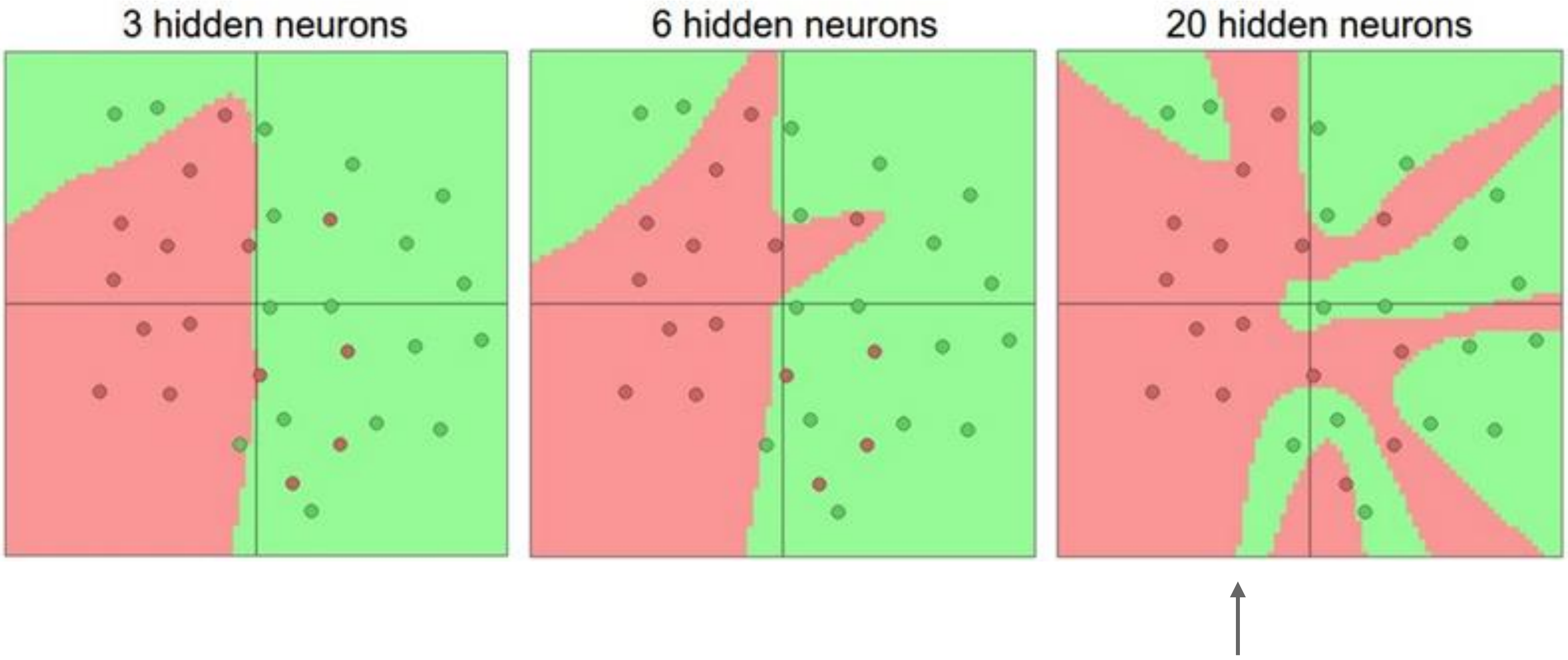
Define the network

Forward pass

Calculate the analytical gradients

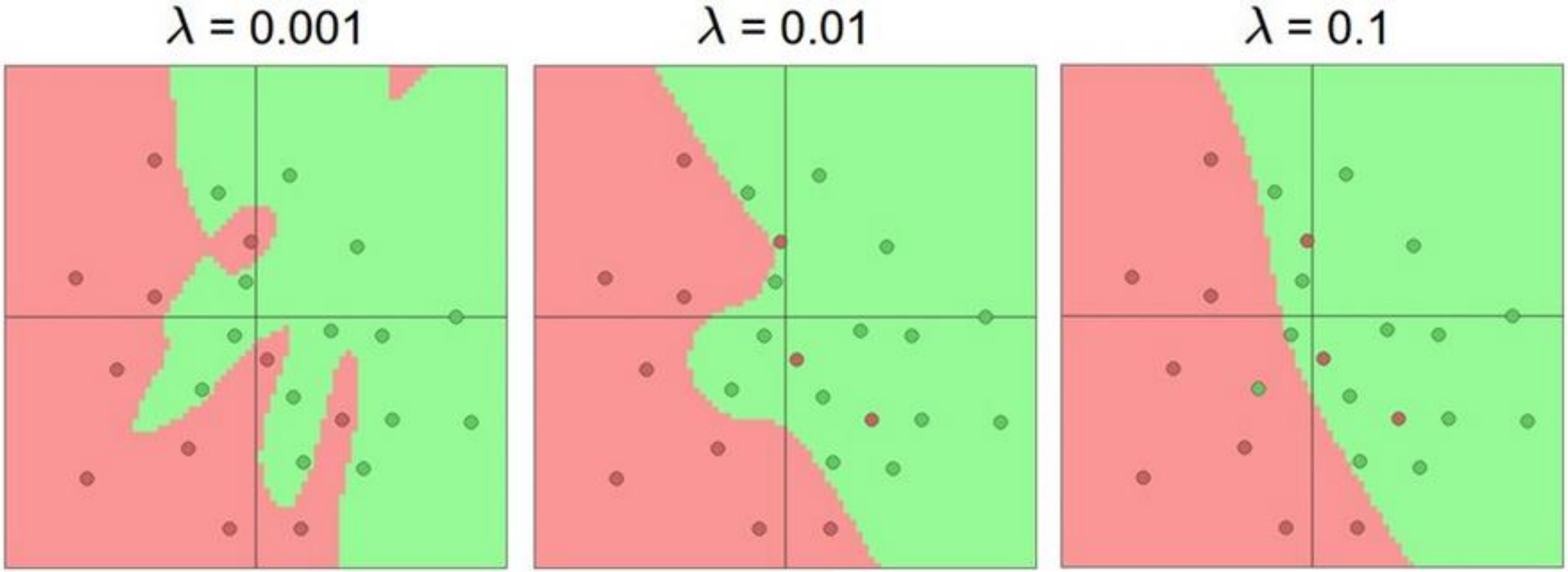
Gradient descent

Setting the number of layers and their sizes



more neurons = more capacity

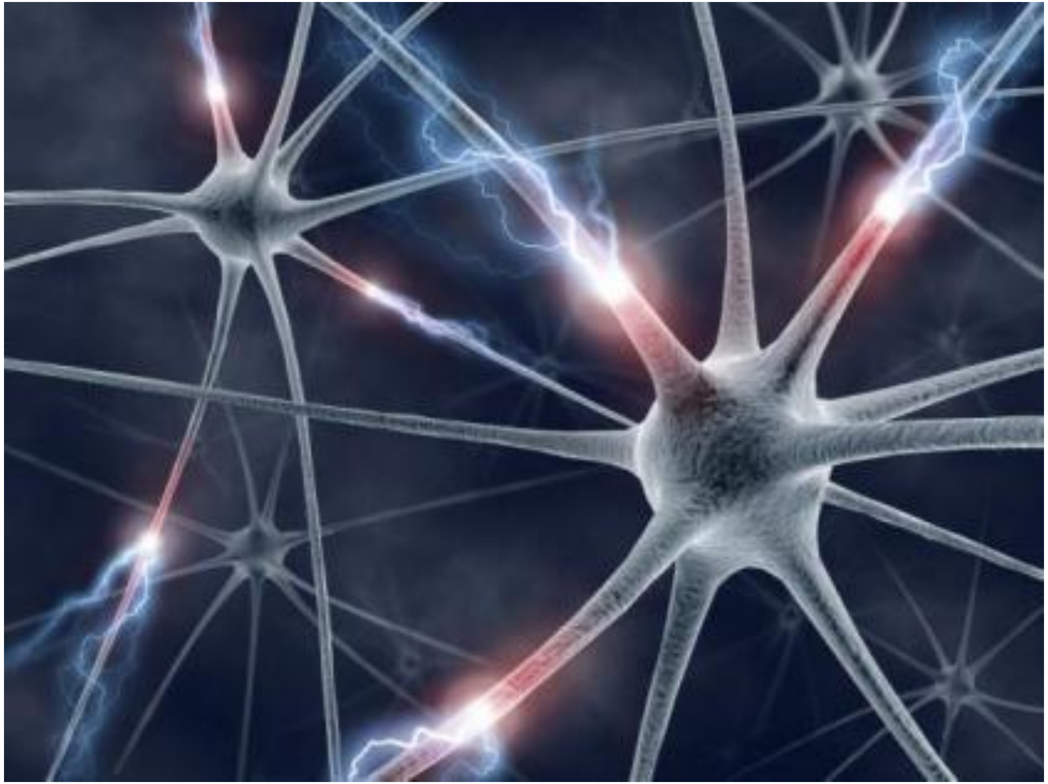
Do not use size of neural network as a regularizer. Use stronger regularization instead:

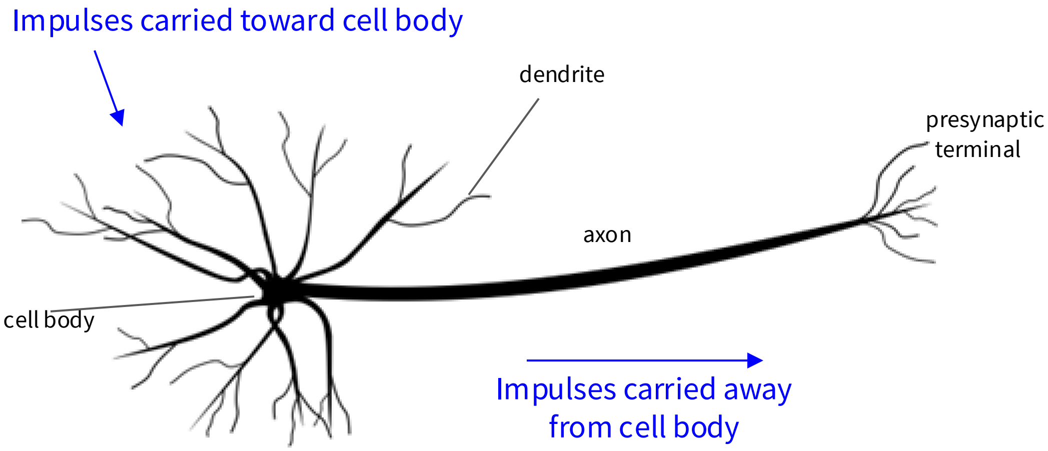


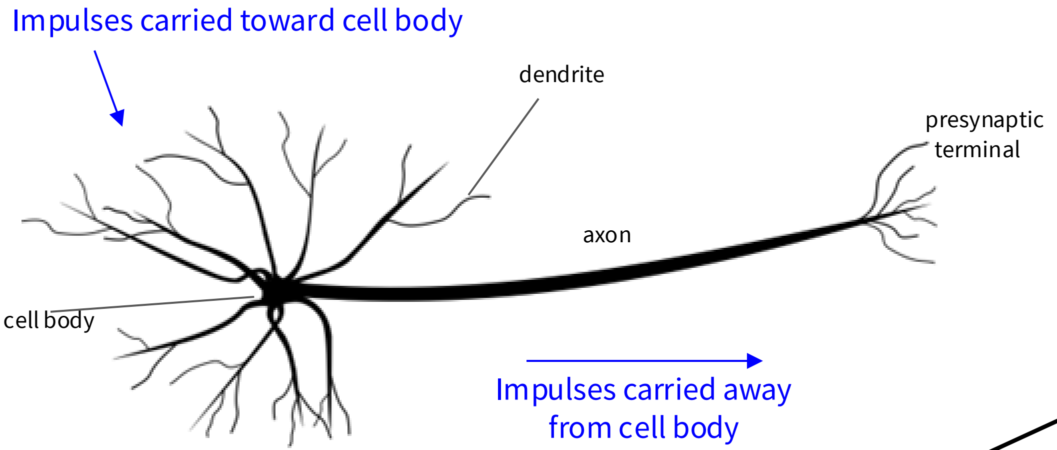
(Web demo with ConvNetJS:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

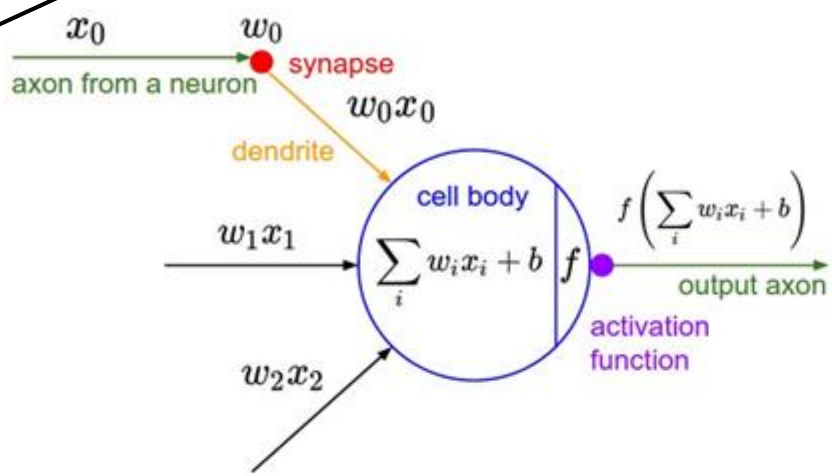
TensorFlow Play Ground: https://playground.tensorflow.org/

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

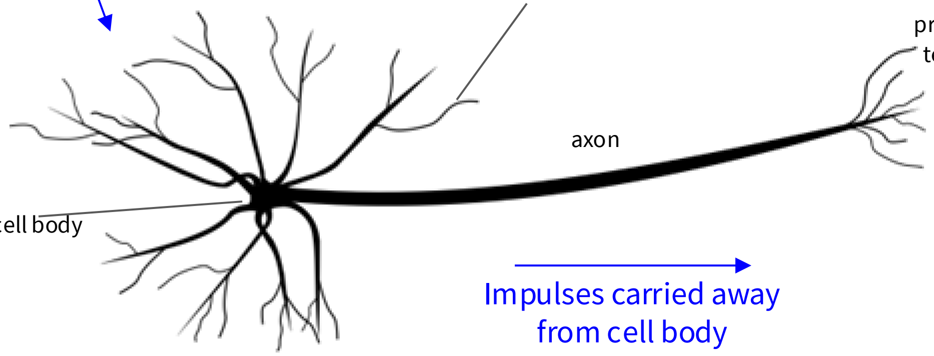



Impulses carried toward cell body
dendrite
presynaptic terminal
axon
cell body
Impulses carried away from cell body

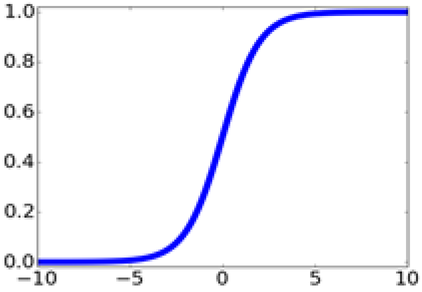

Impulses carried toward cell body
dendrite
presynaptic terminal
axon
cell body
Impulses carried away from cell body


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation function
$w_2 x_2$

Impulses carried toward cell body

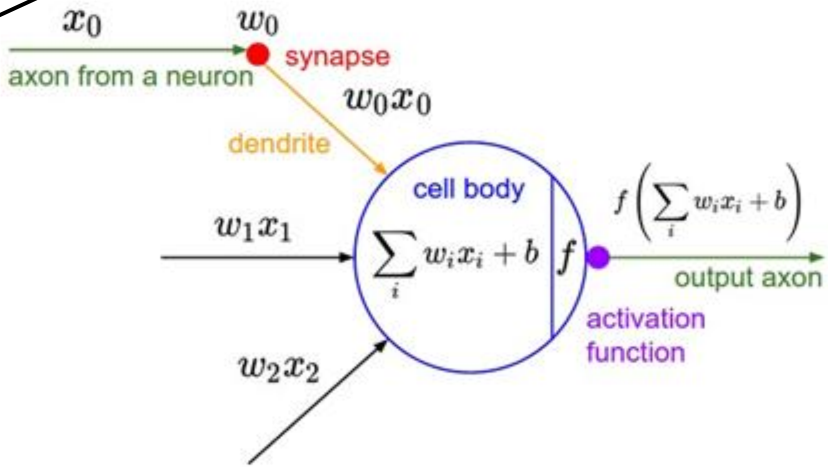


Impulses carried away from cell body

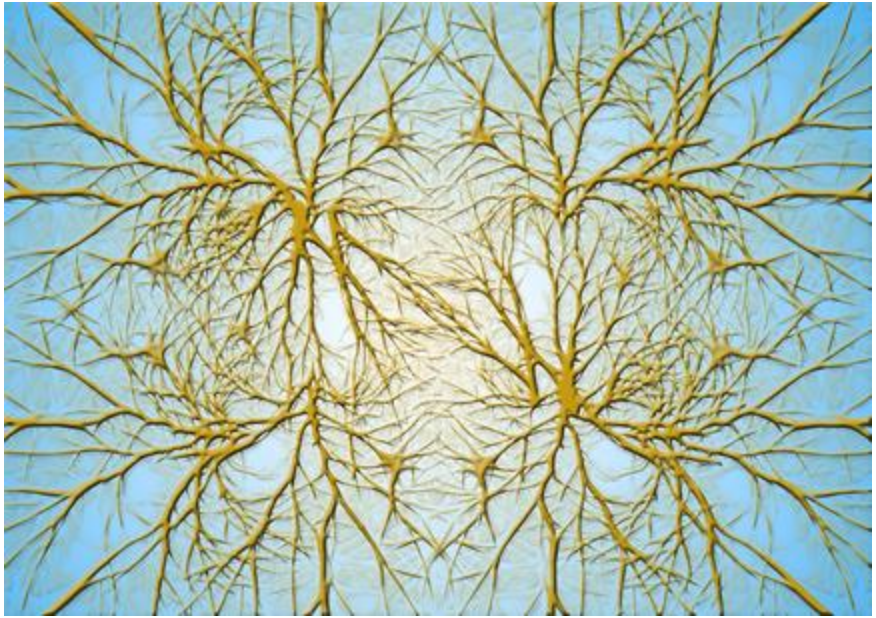


sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

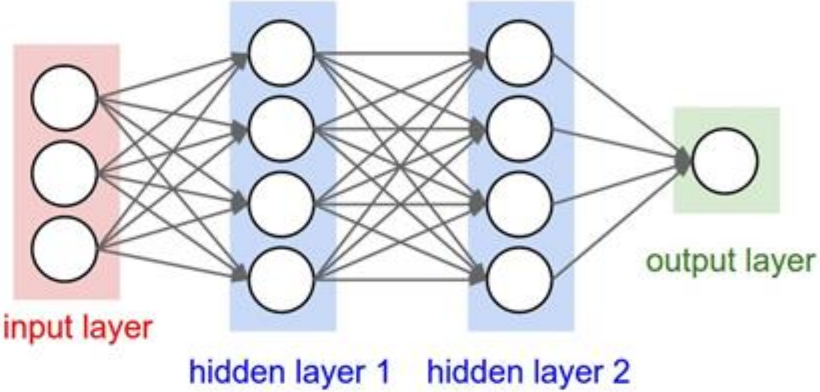


Biological Neurons:
Complex connectivity patterns

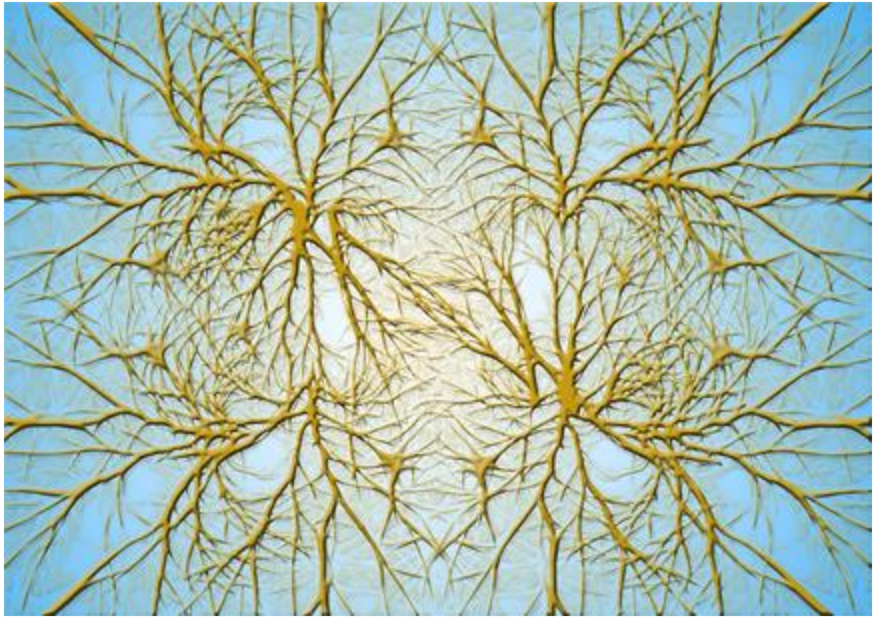


This image is CC0 Public Domain

Neurons in a neural network:
Organized into regular layers for computational efficiency

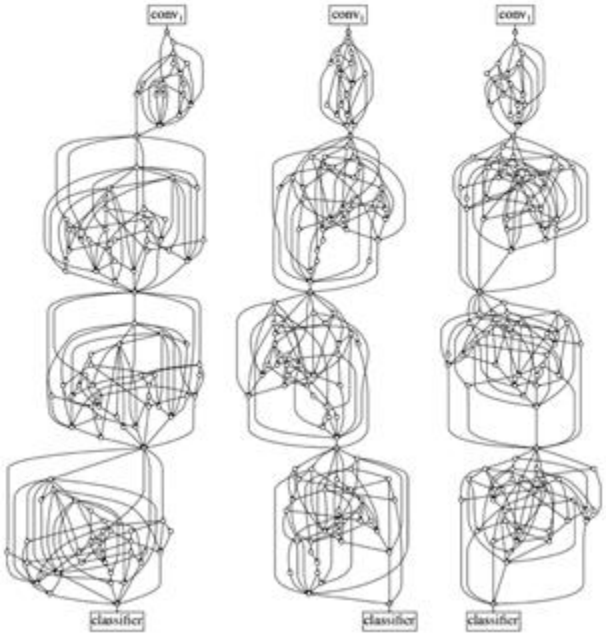


Biological Neurons:
Complex connectivity patterns



This image is CC0 Public Domain

But neural networks with random connections can work too!



Xie et al, "Exploring Randomly Wired Neural Networks for Image Recognition", IEEE/CVF International Conference on Computer Vision 2019

Be very careful with your brain analogies!

Biological Neurons:

- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system

[Dendritic Computation. London and Hausser]

Plugging in neural networks with loss functions

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

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

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

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

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

(Bad) Idea: Derive $\nabla_W L$ on paper

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

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

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

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

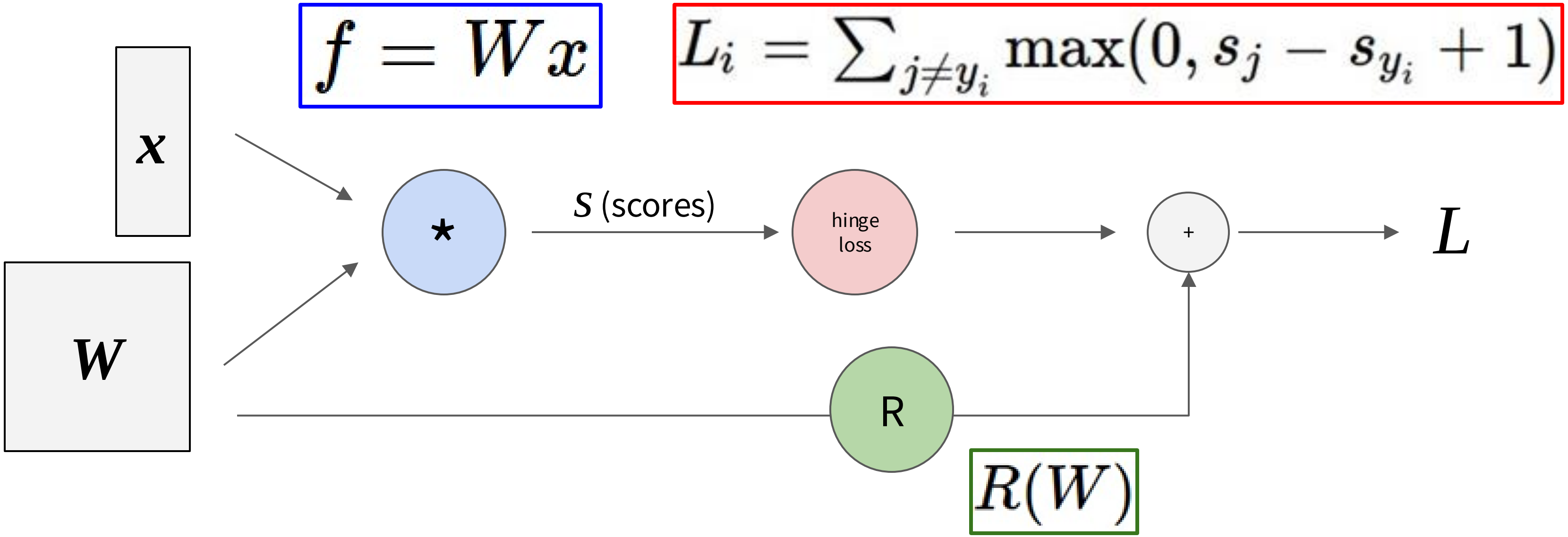
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

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 hinge? Need to re-derive everything from scratch!

Problem: Not feasible for very complex models!

Better Idea: Computational graphs + Backpropagation

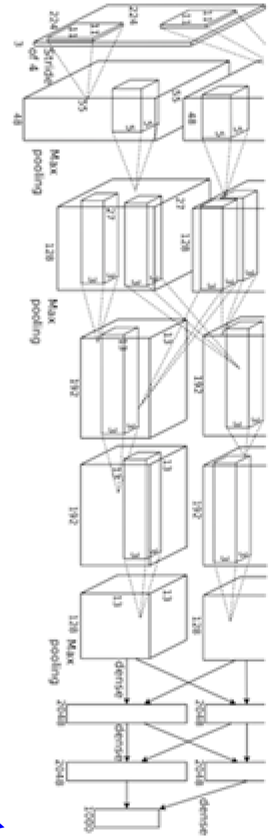


Convolutional network (AlexNet)

input image

weights

loss



Really complex neural networks!!

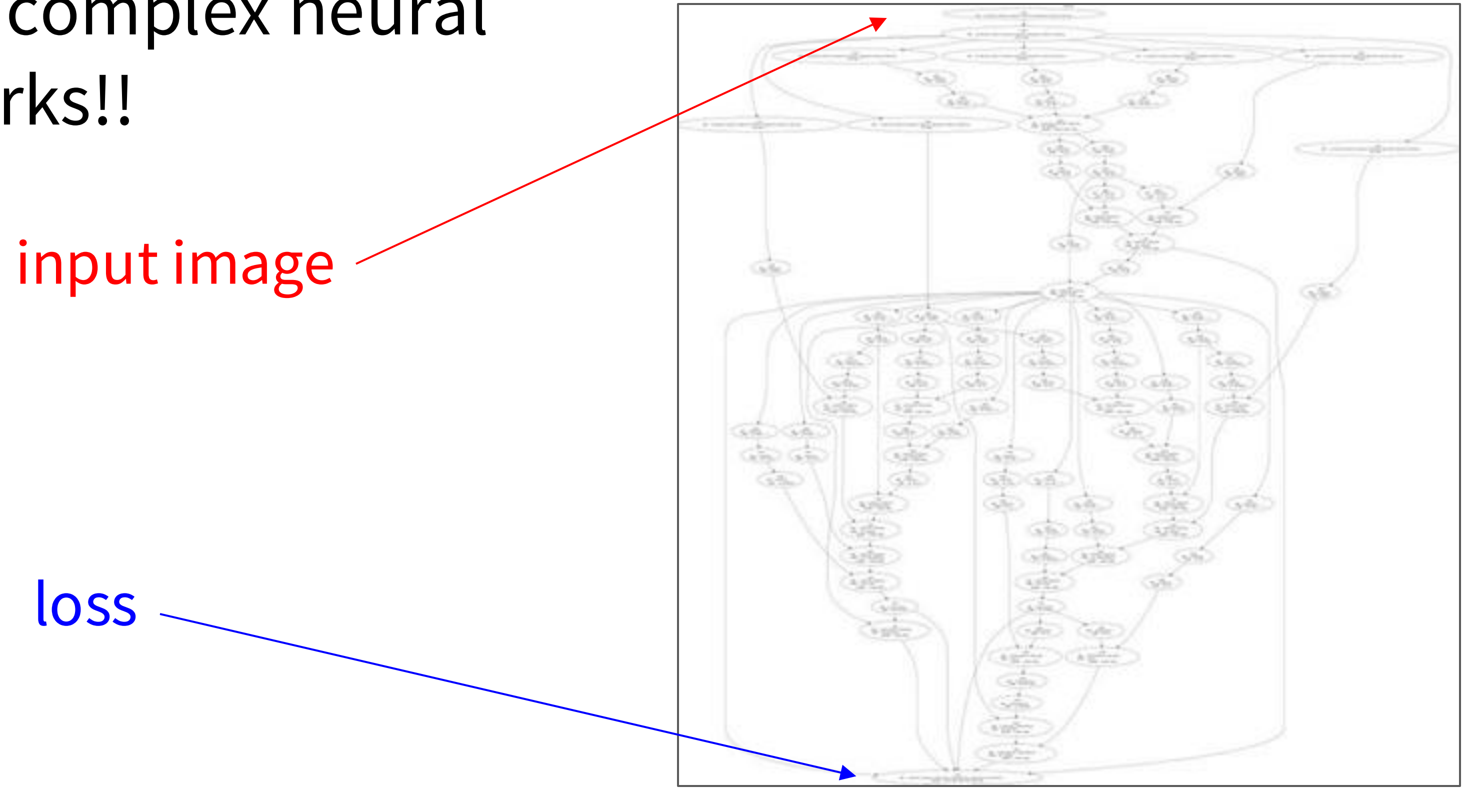


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

Neural Turing Machine

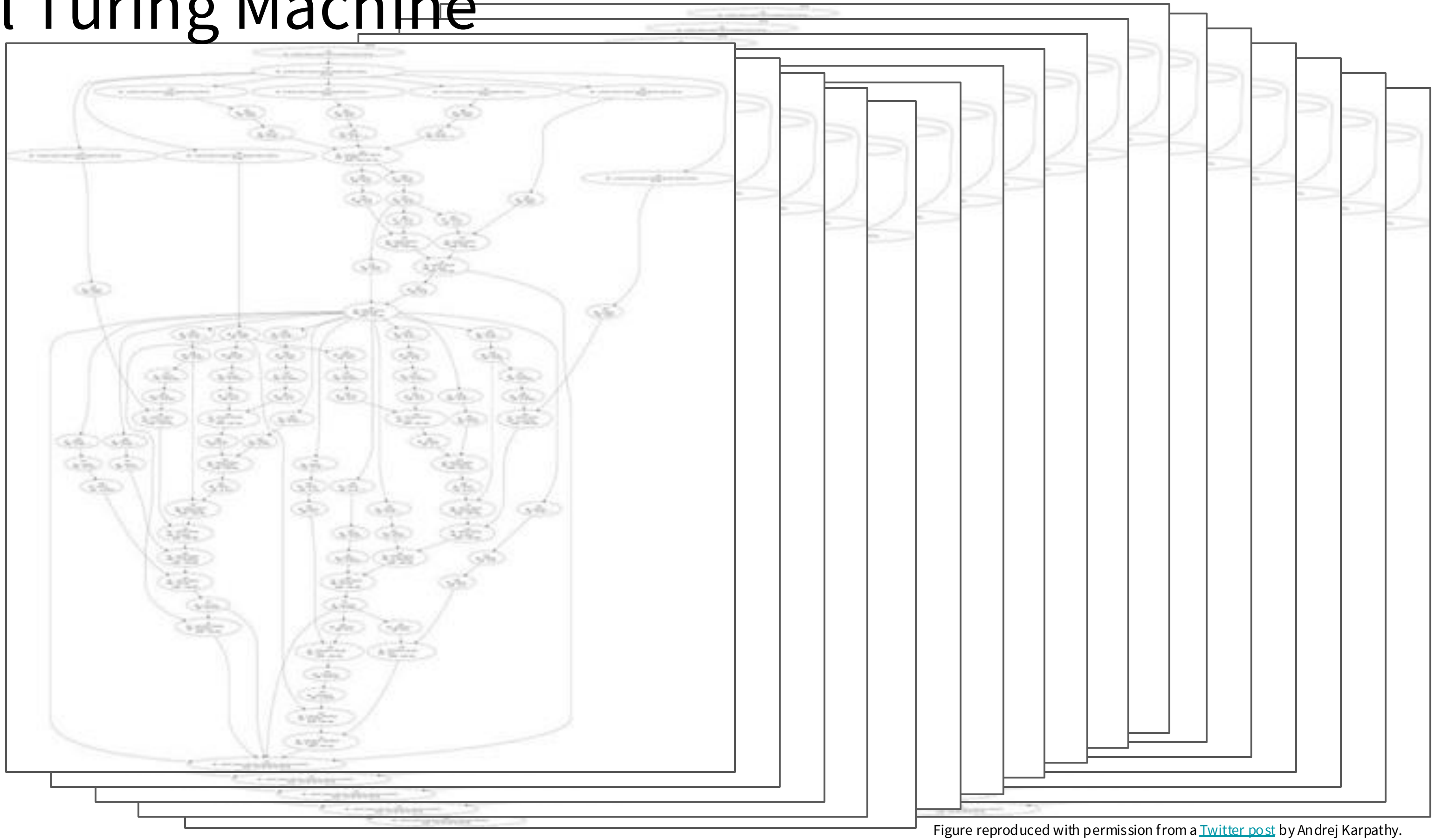


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

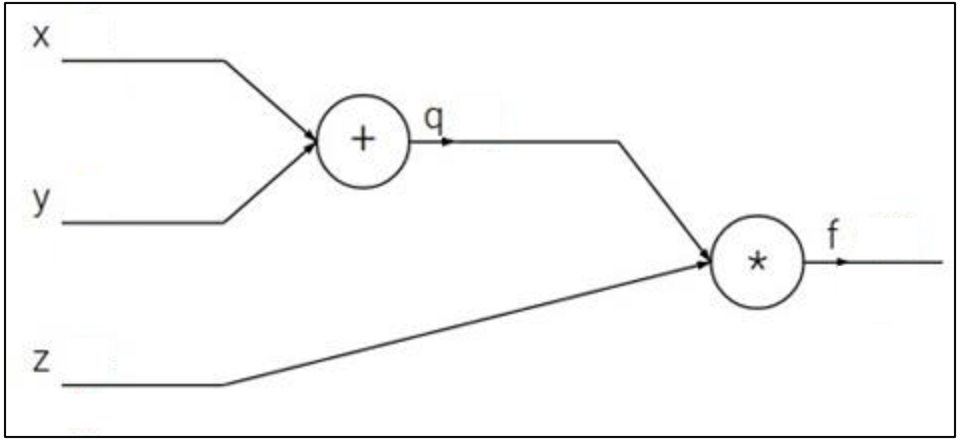
Solution: Backpropagation

Backpropagation: a simple example

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

Backpropagation: a simple example

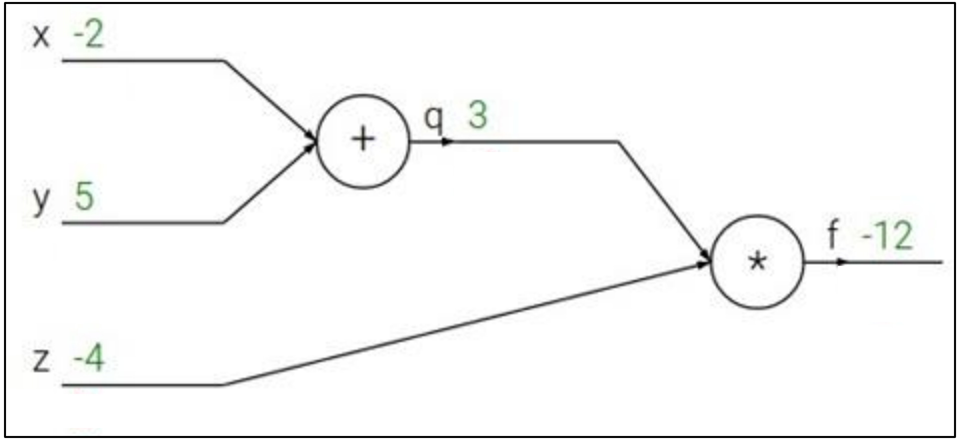
$$f(x, y, z) = (x + y)z$$



Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4

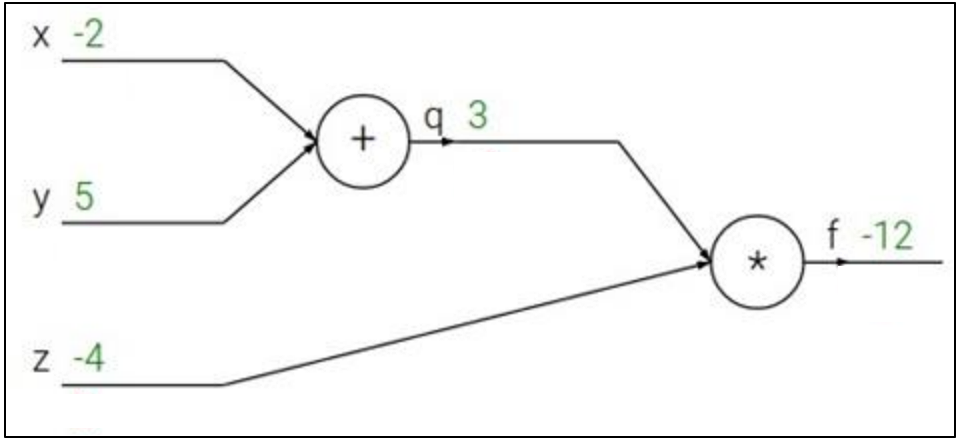


Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$



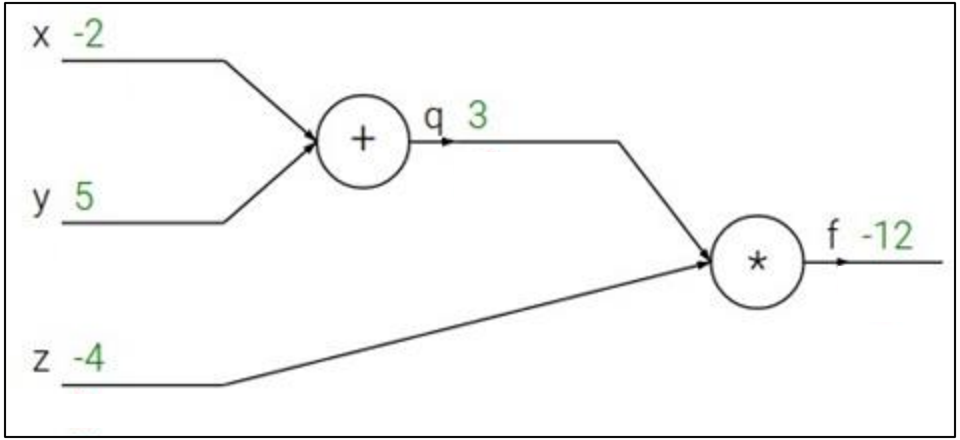
Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Backpropagation: a simple example

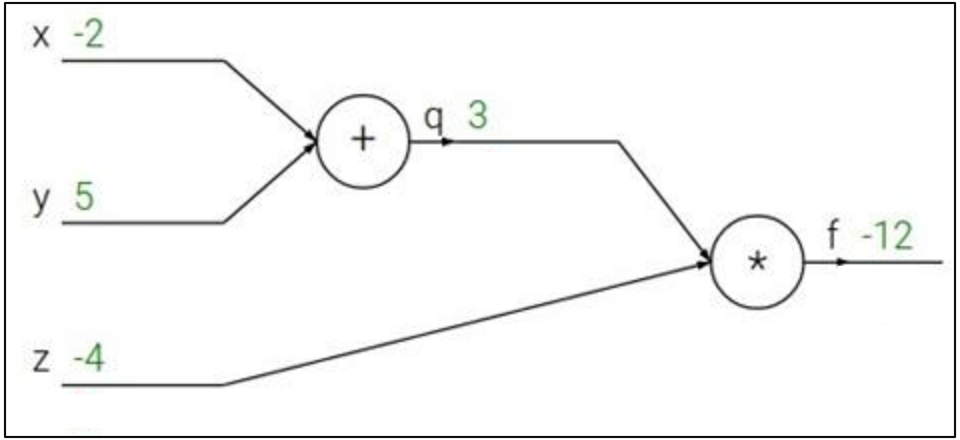
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Backpropagation: a simple example

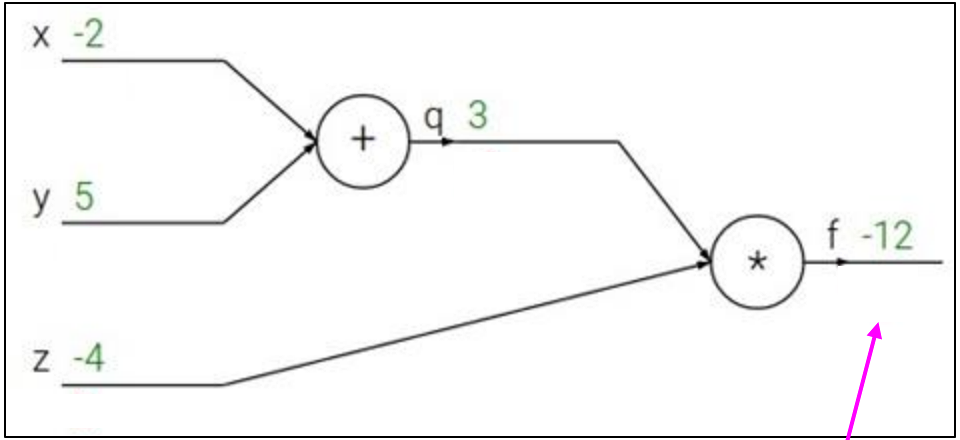
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



$\frac{\partial f}{\partial f}$

Backpropagation: a simple example

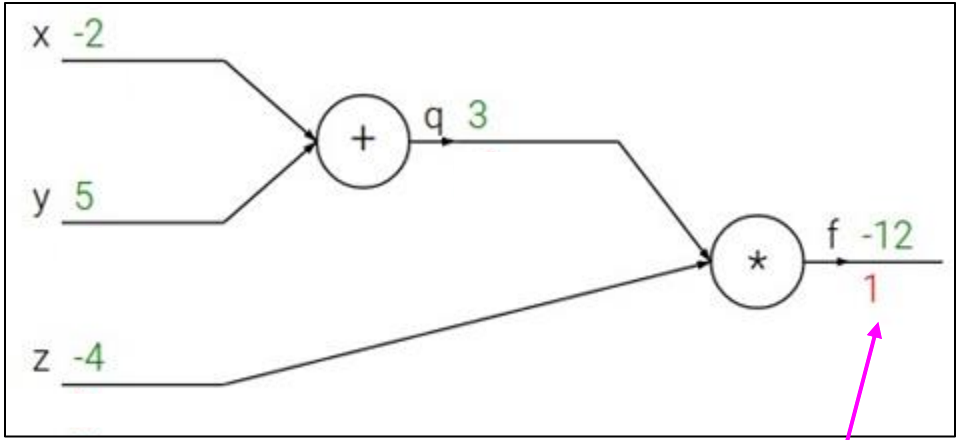
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



$\frac{\partial f}{\partial f}$

Backpropagation: a simple example

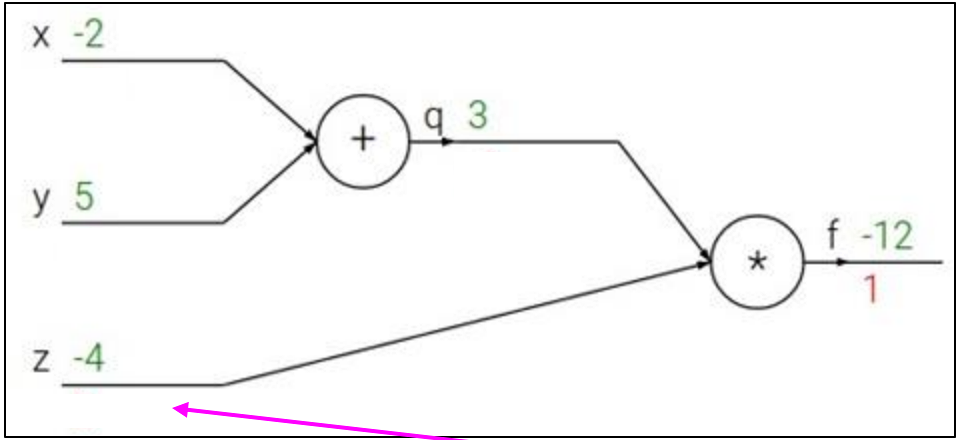
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Backpropagation: a simple example

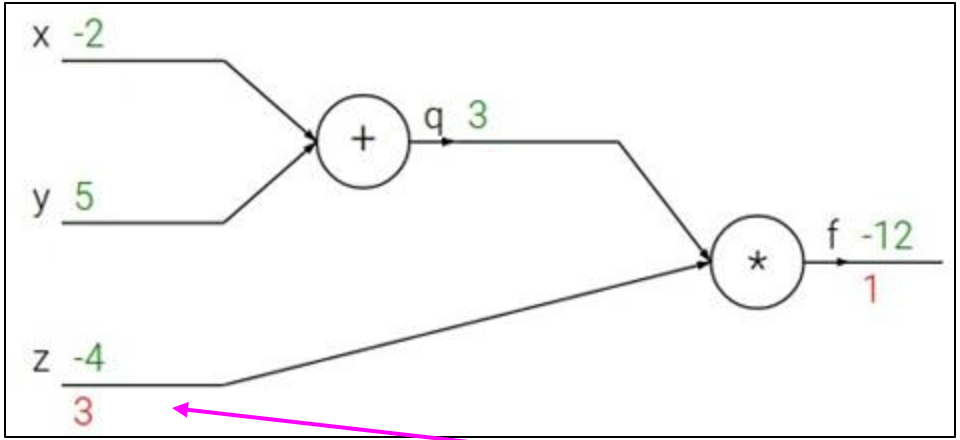
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Backpropagation: a simple example

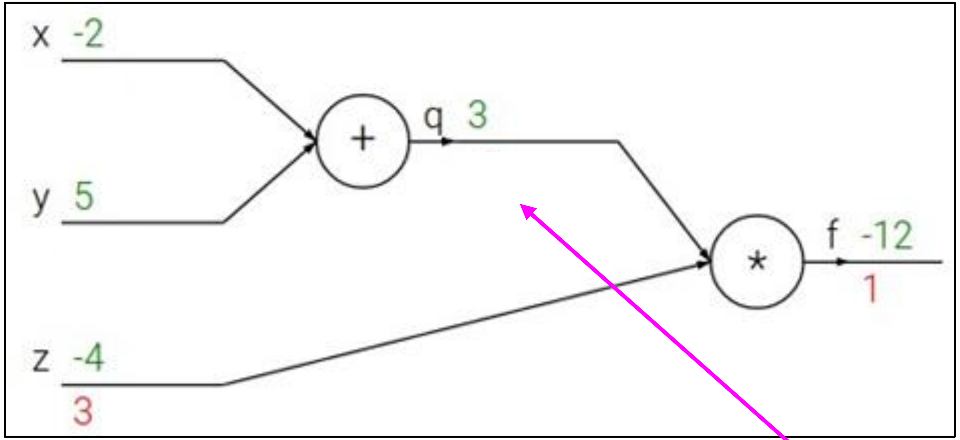
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Backpropagation: a simple example

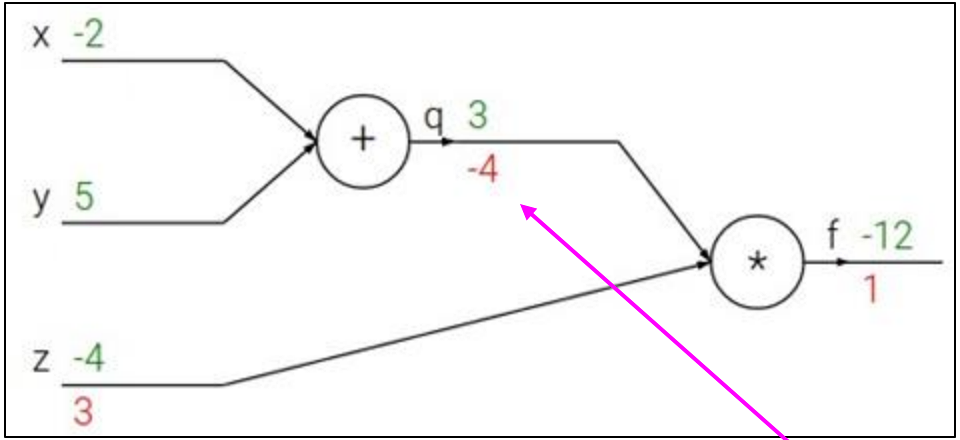
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Backpropagation: a simple example

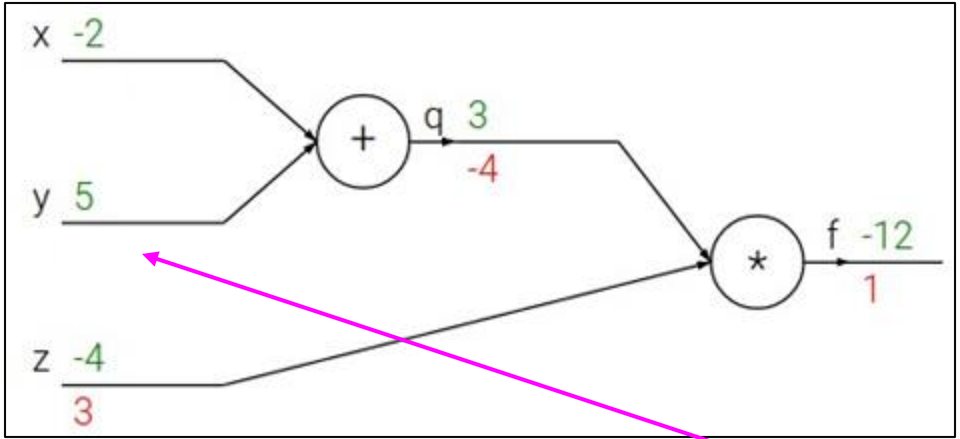
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

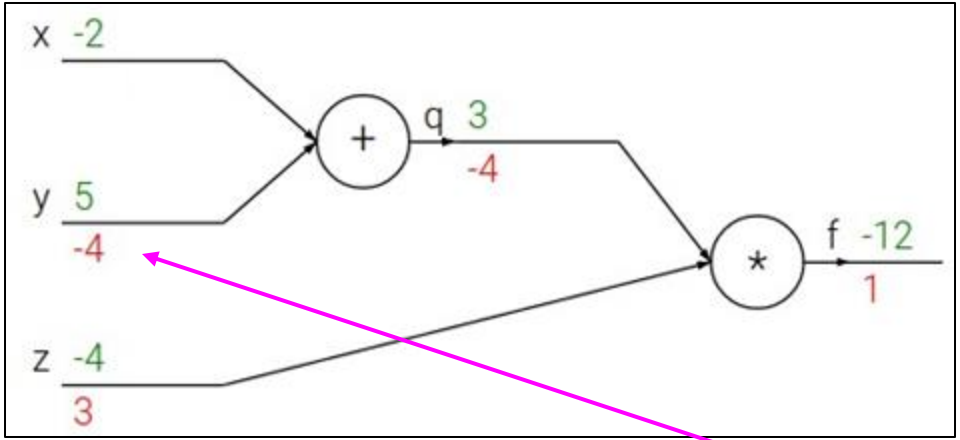
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

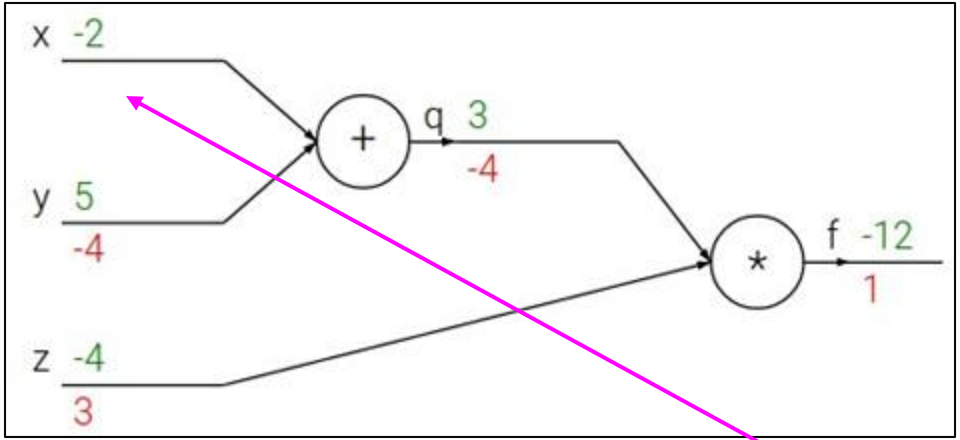
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

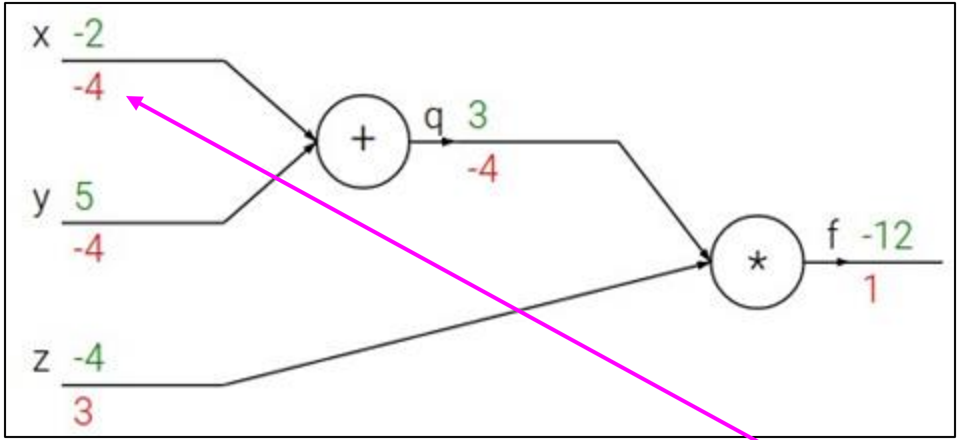
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



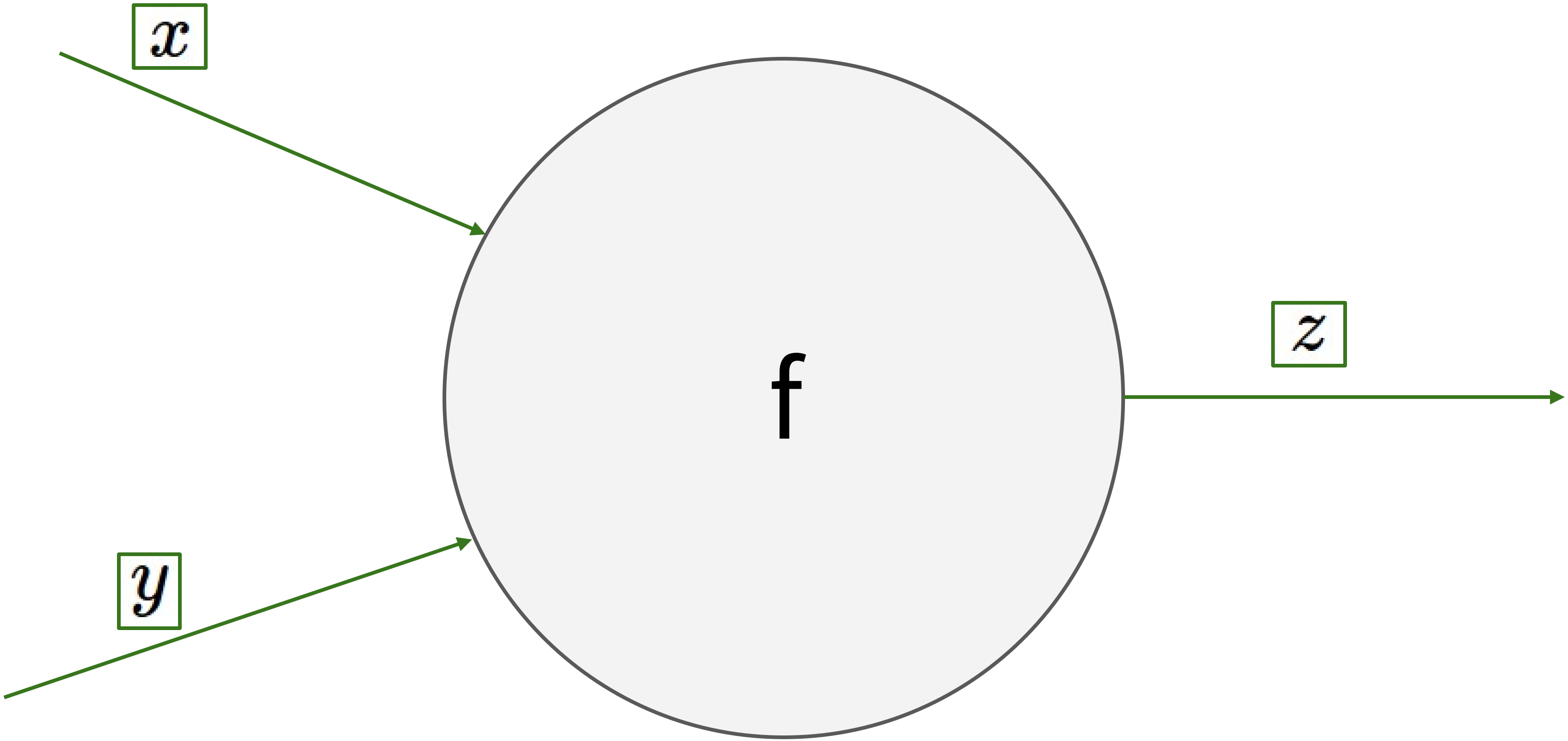
$\frac{\partial f}{\partial x}$

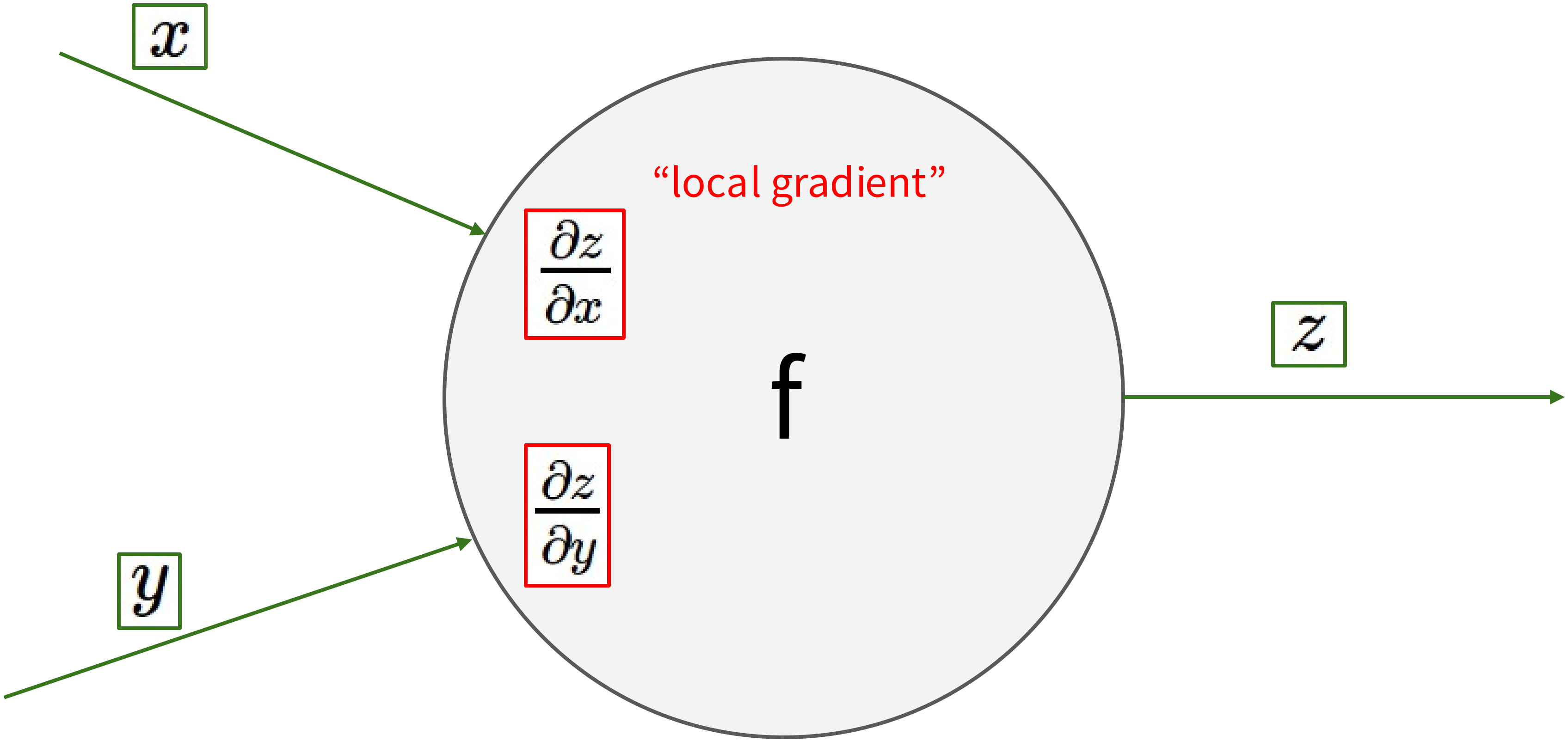
Chain rule:

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

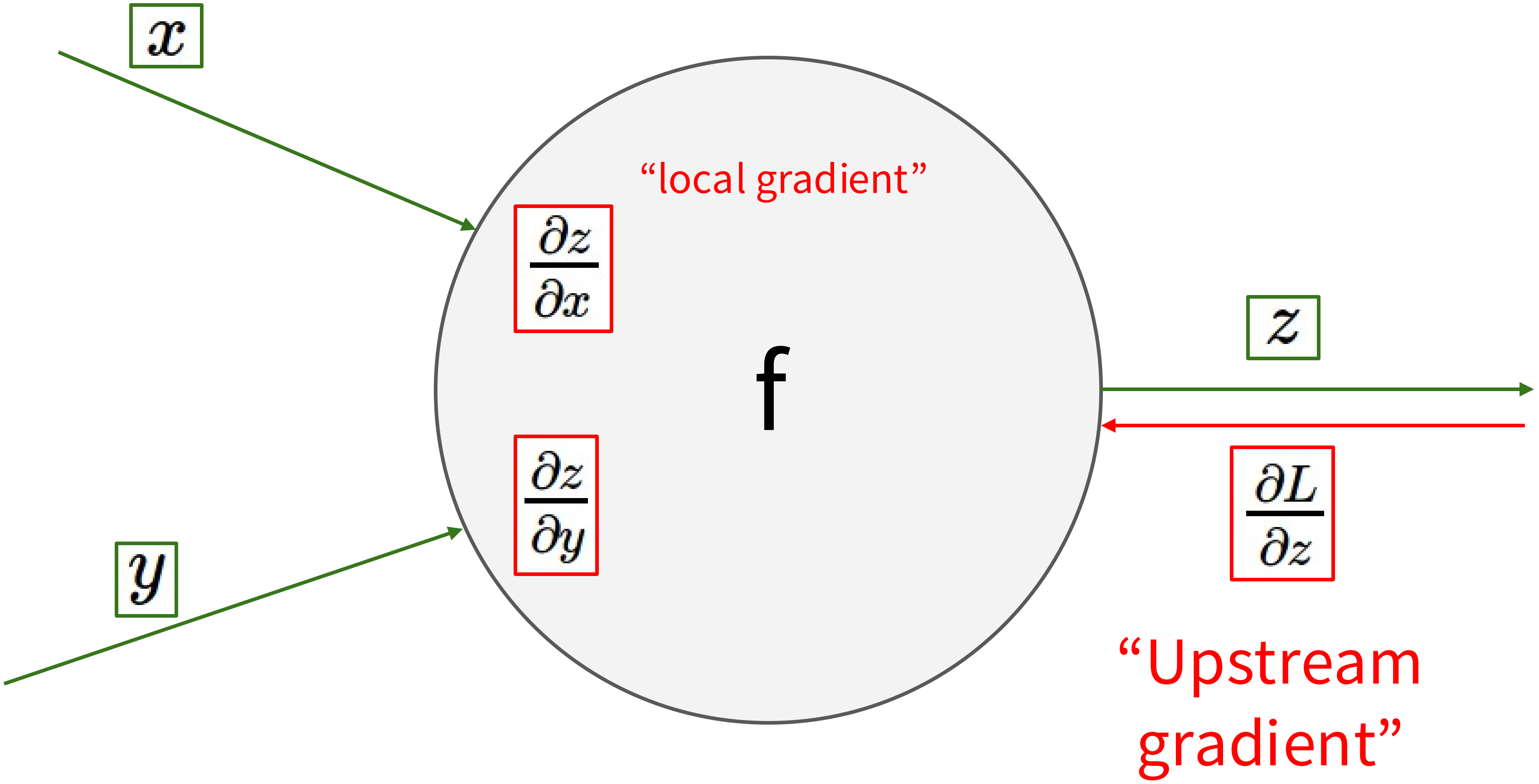
Upstream gradient

Local gradient

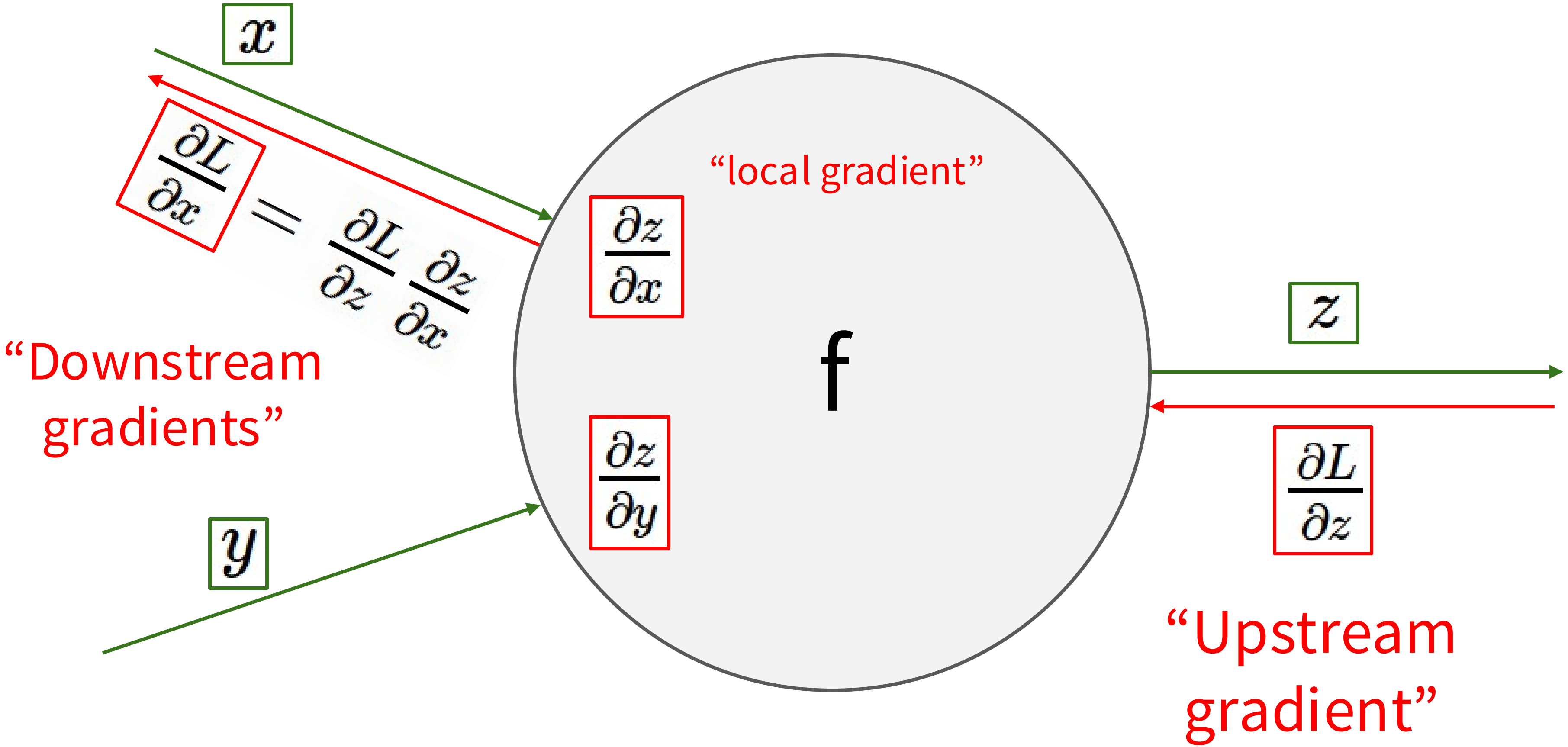

x
y
f
z



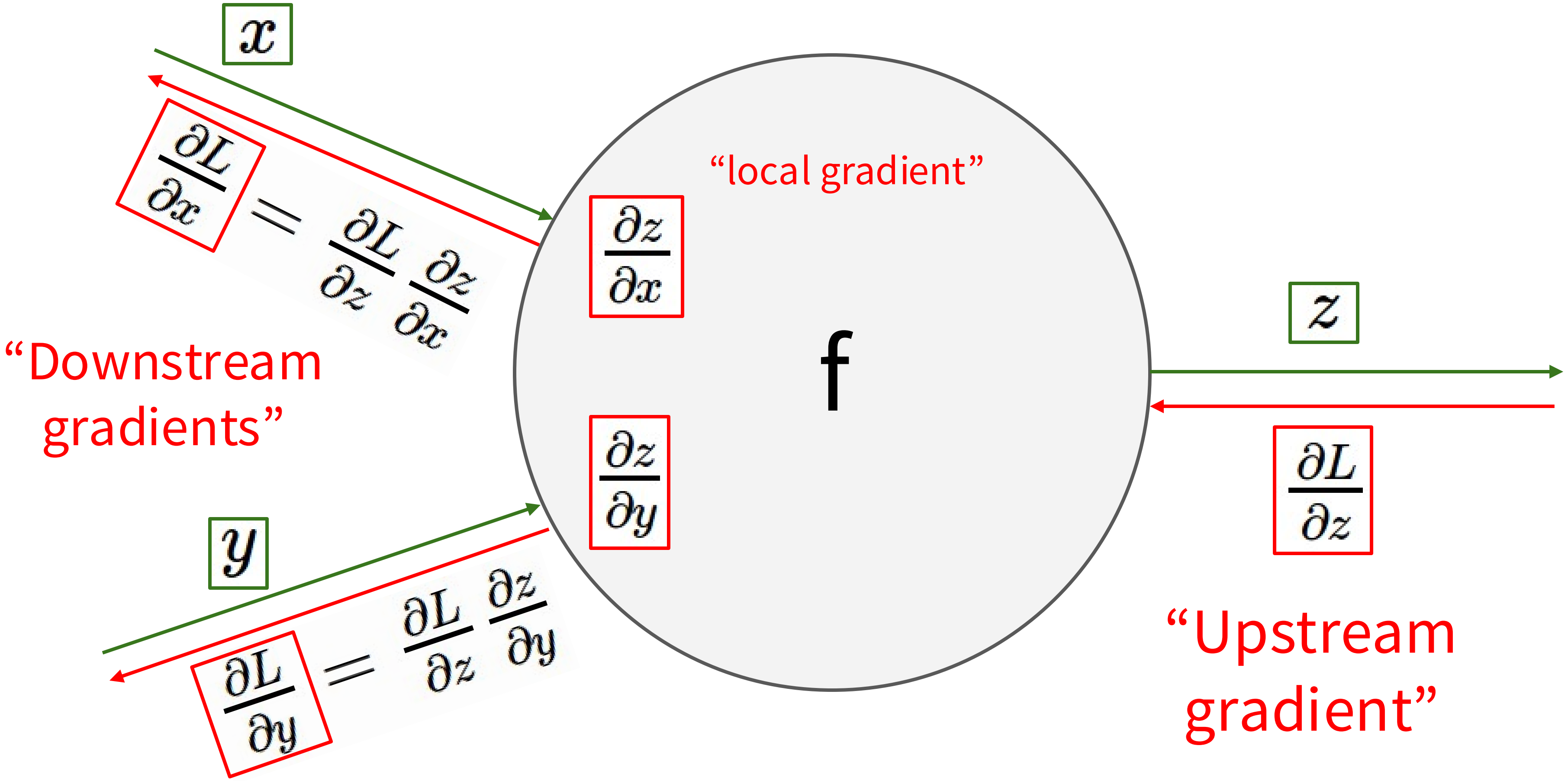
x
“local gradient”
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

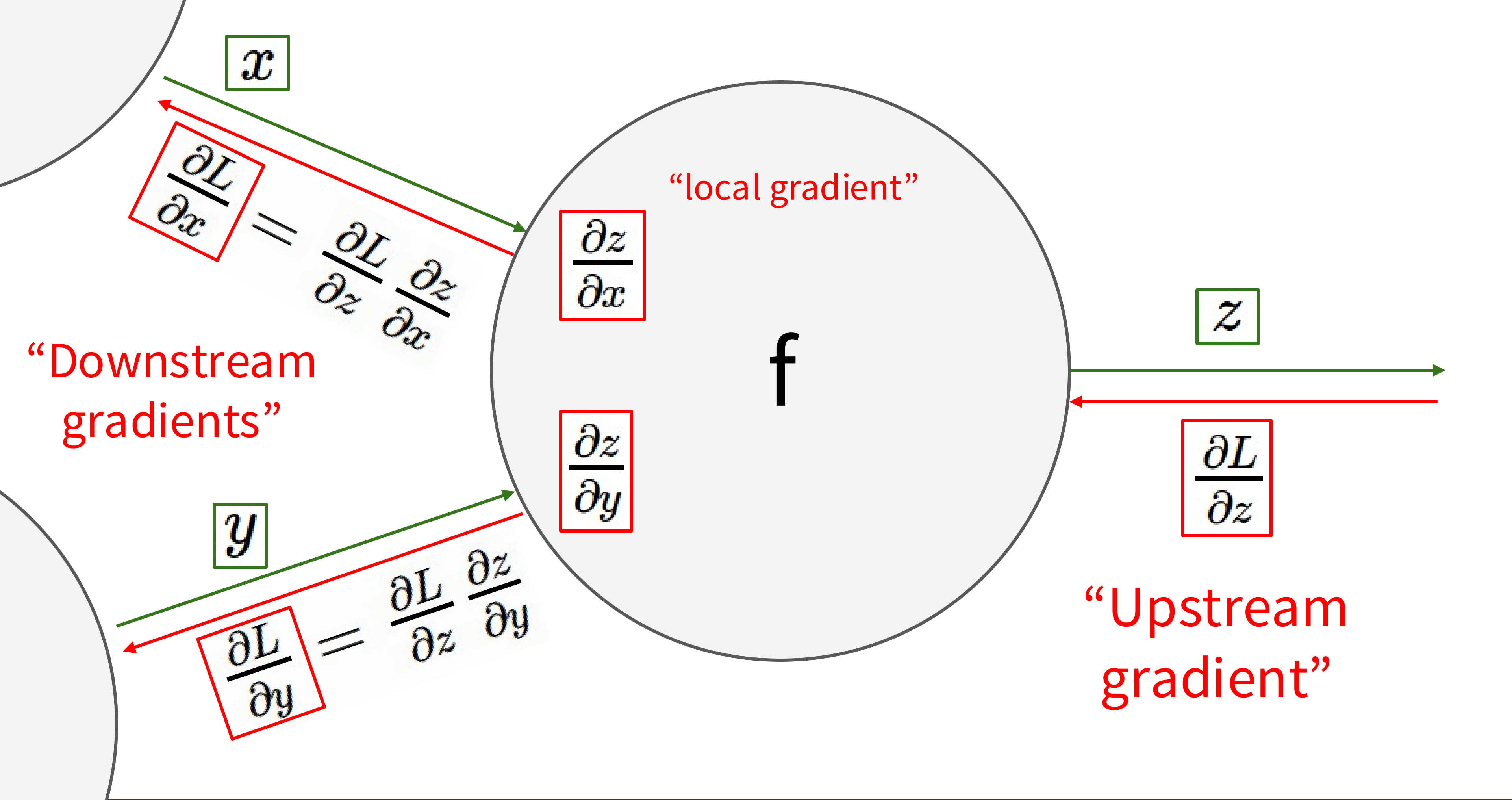


x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y
$\frac{\partial L}{\partial z}$
"Upstream gradient"



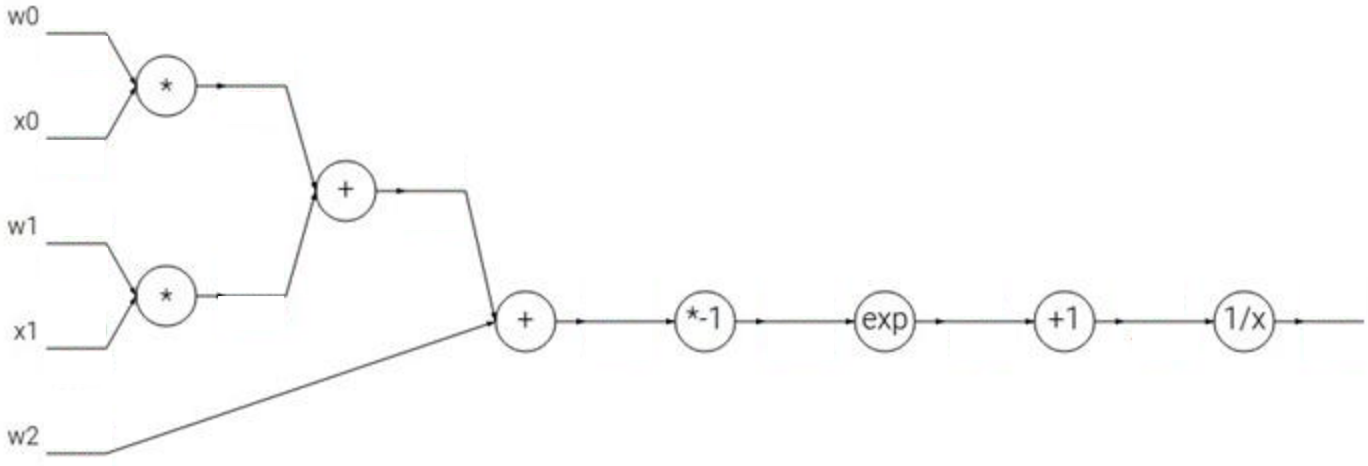
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
"Downstream gradients"
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
"Upstream gradient"


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

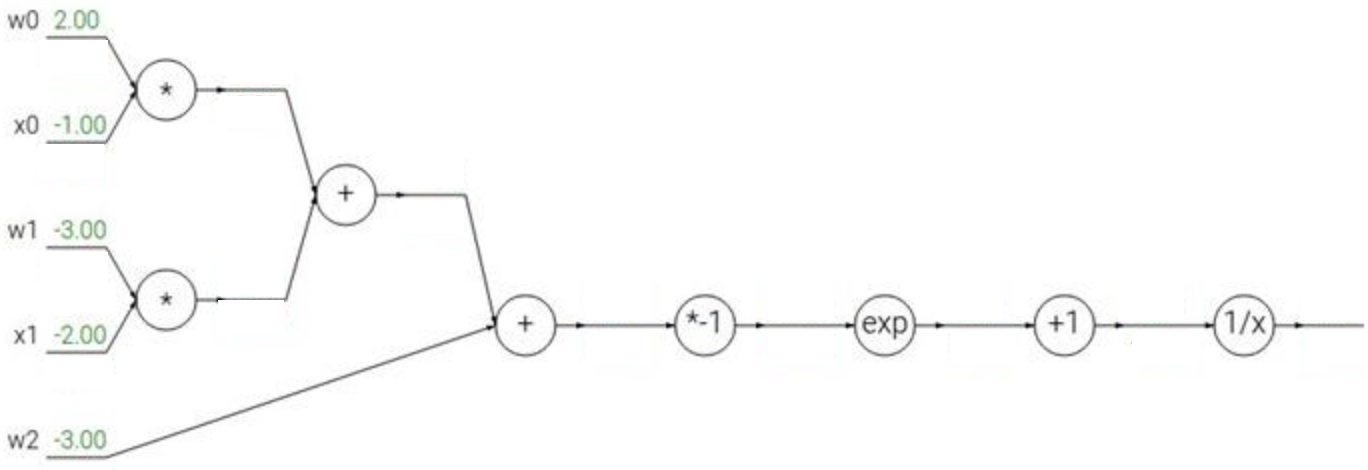
Another example:

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



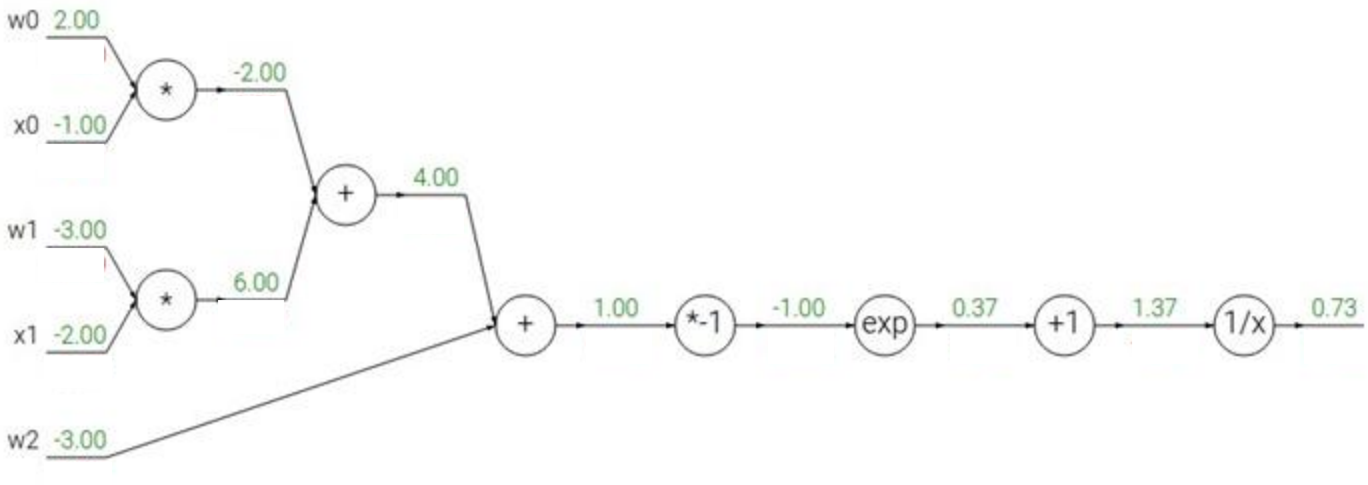
Another example:

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

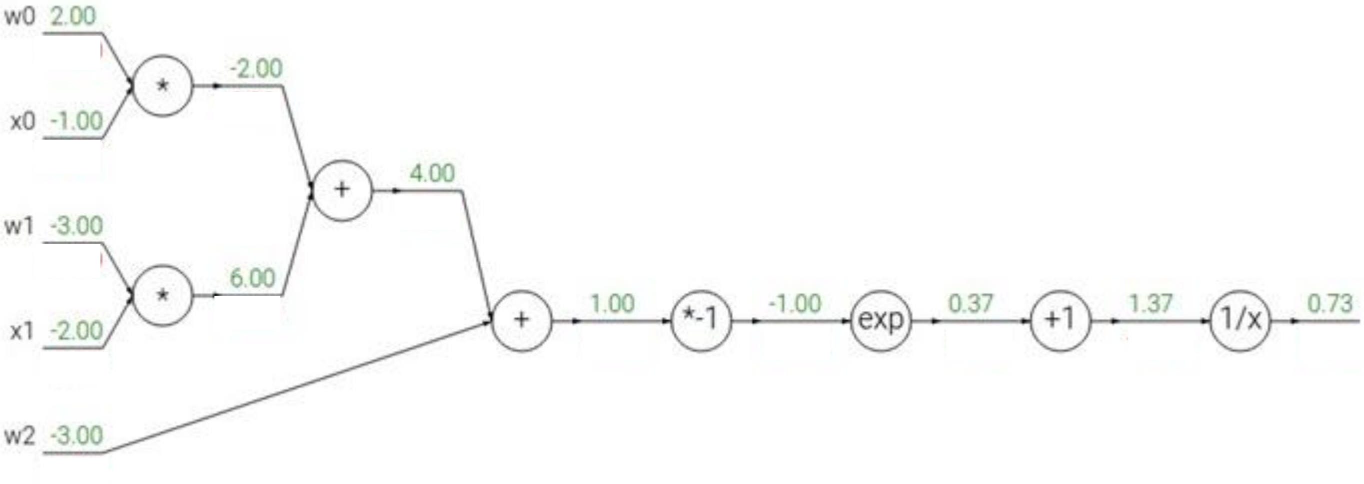


Another example:

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



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



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

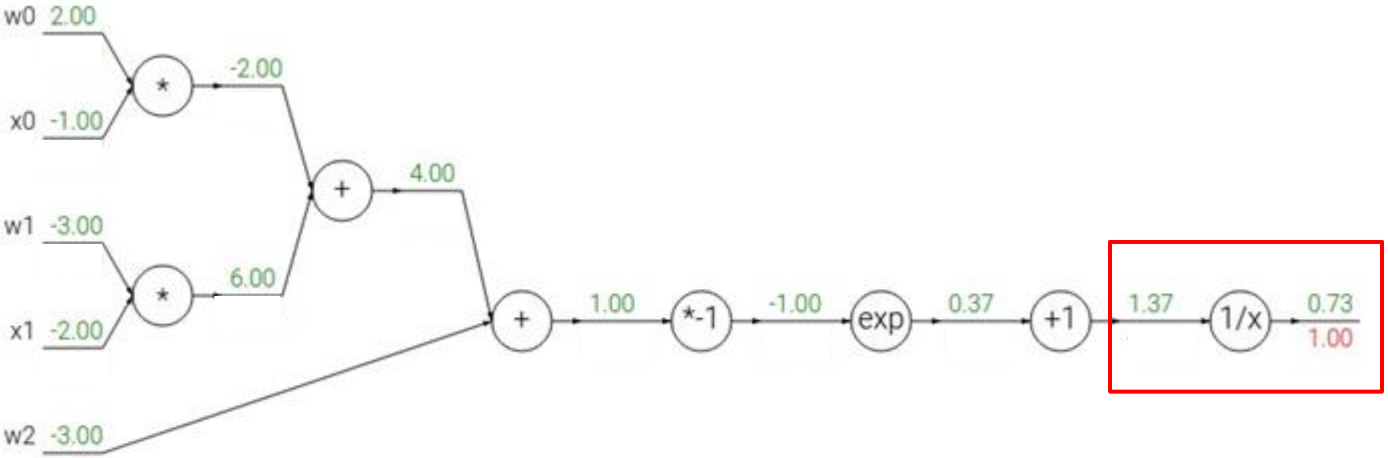
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

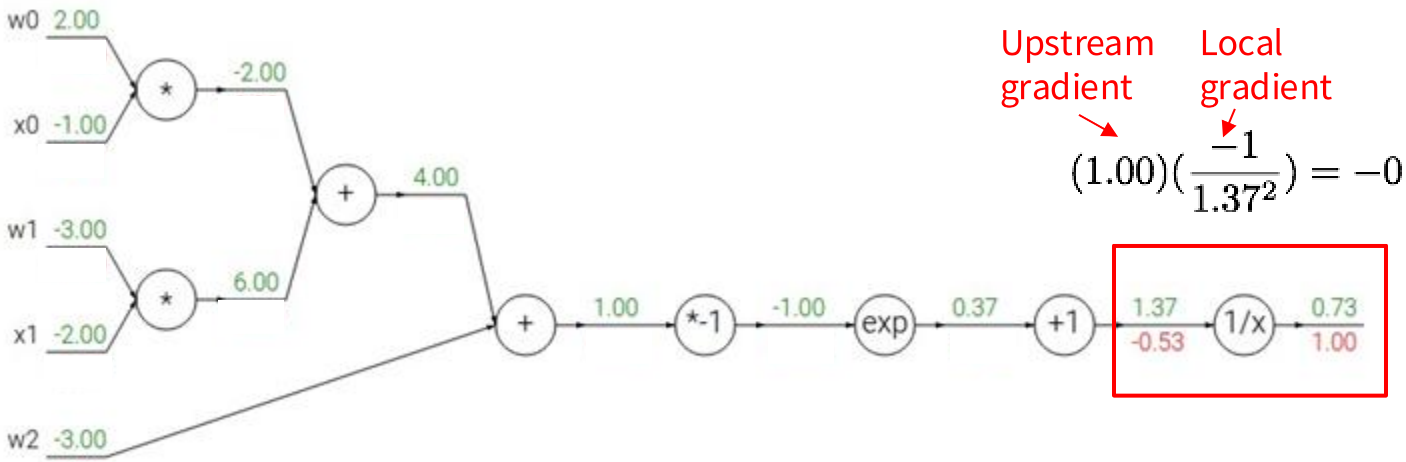
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

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

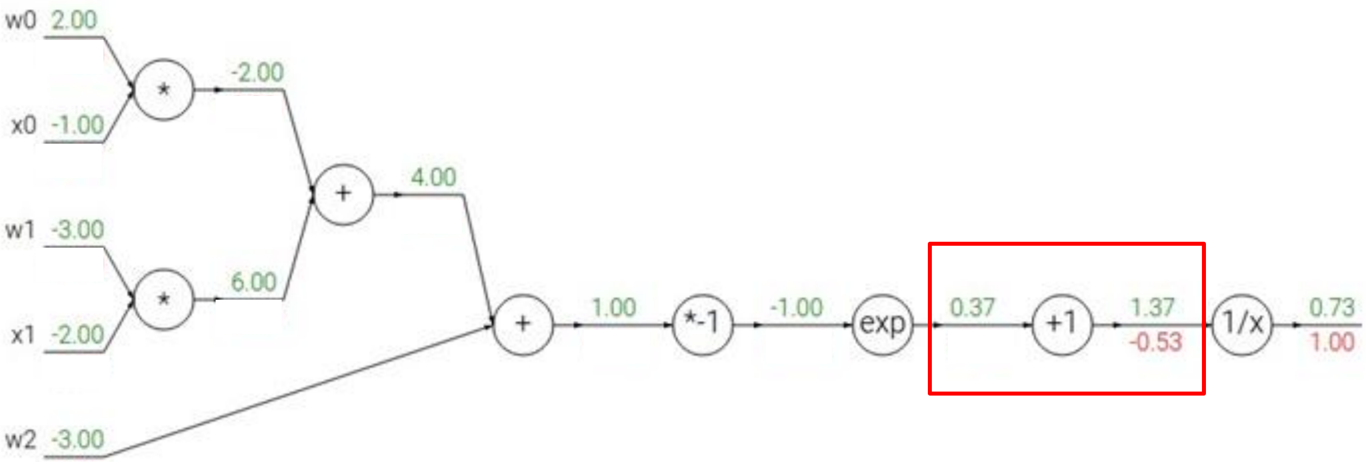
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

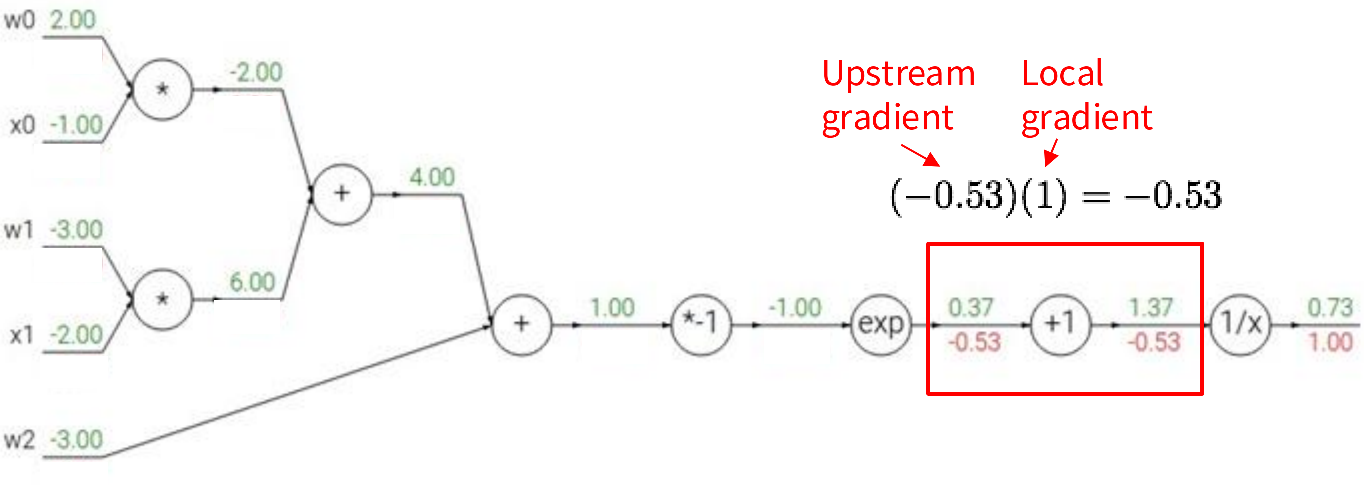
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

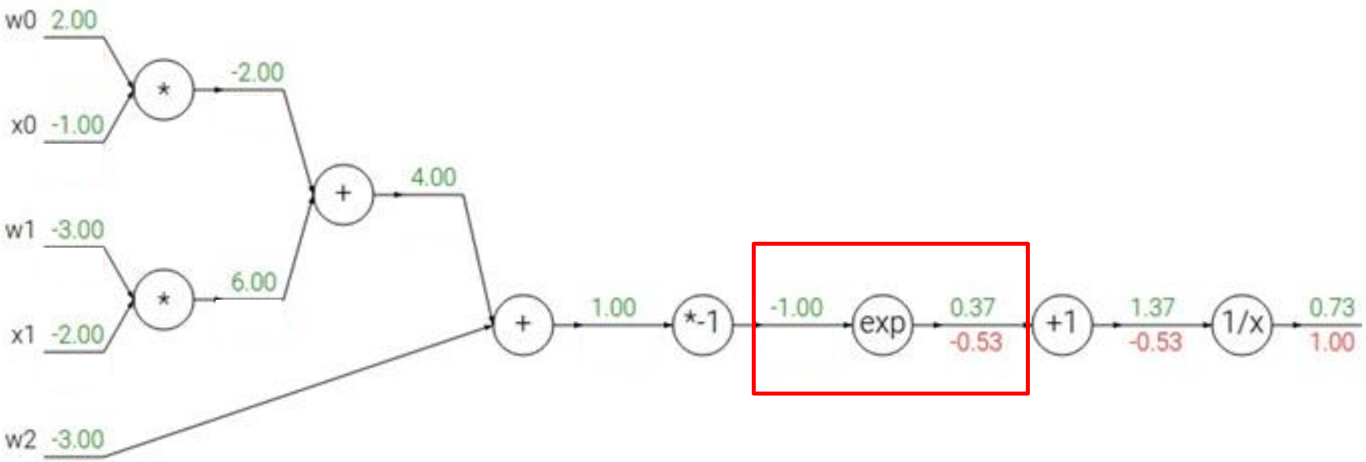
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$

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

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

Another example:

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



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

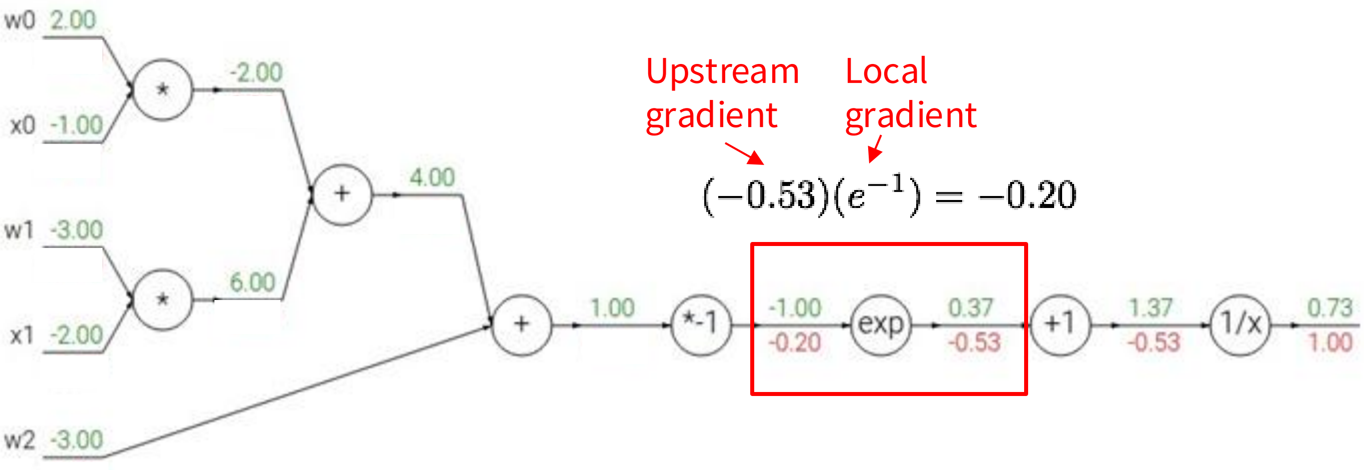
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



$$(-0.53)(e^{-1}) = -0.20$$

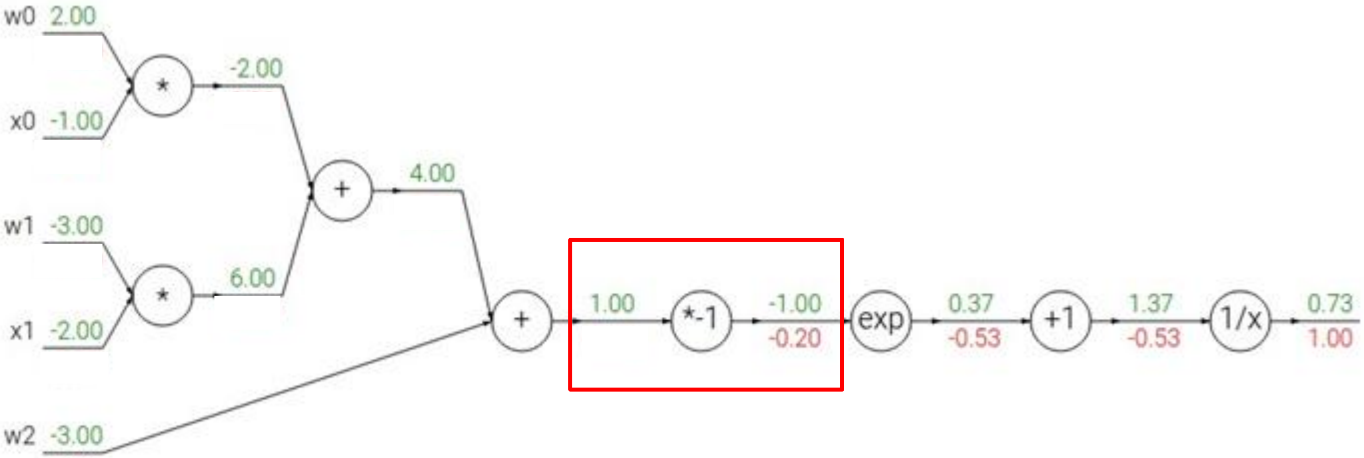
$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

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

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

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

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



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

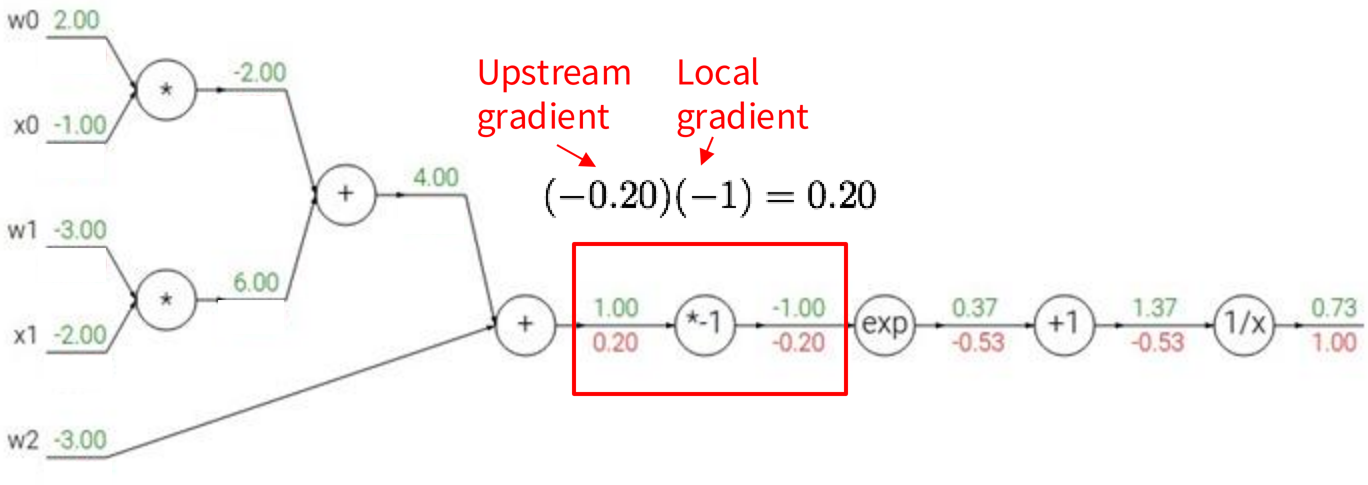
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

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

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

Another example:

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



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

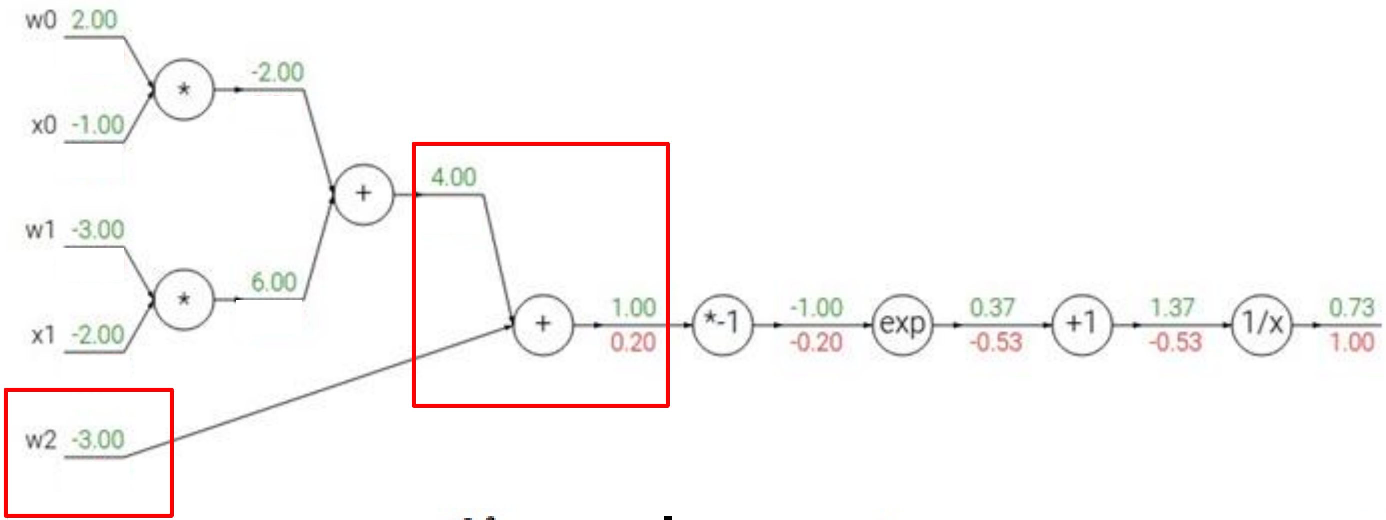
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

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

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

Another example:

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



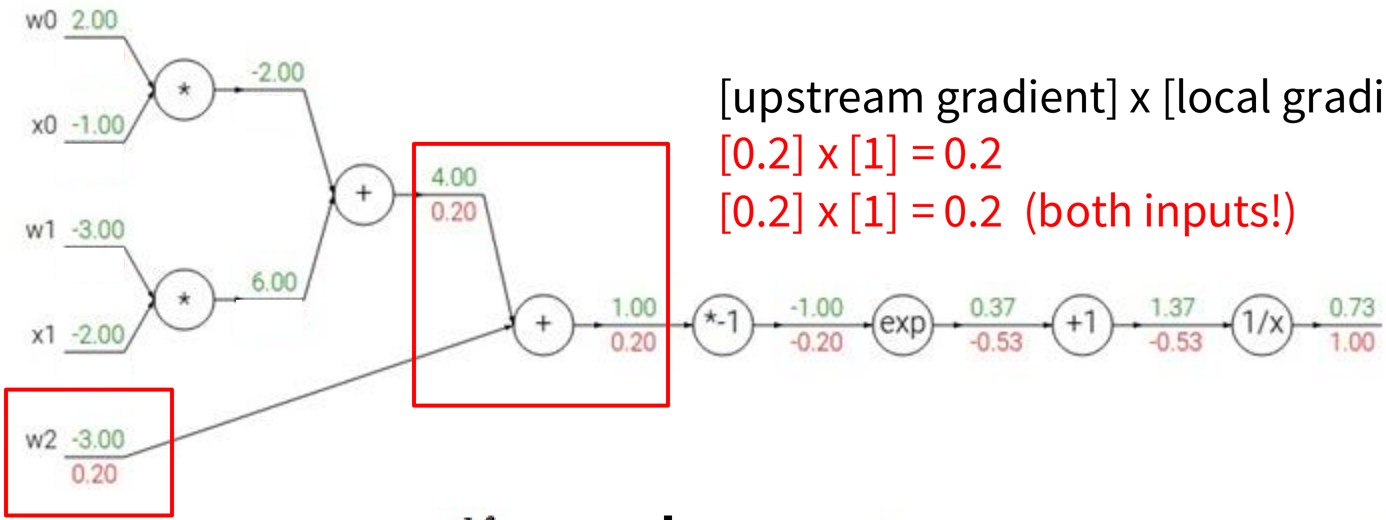
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

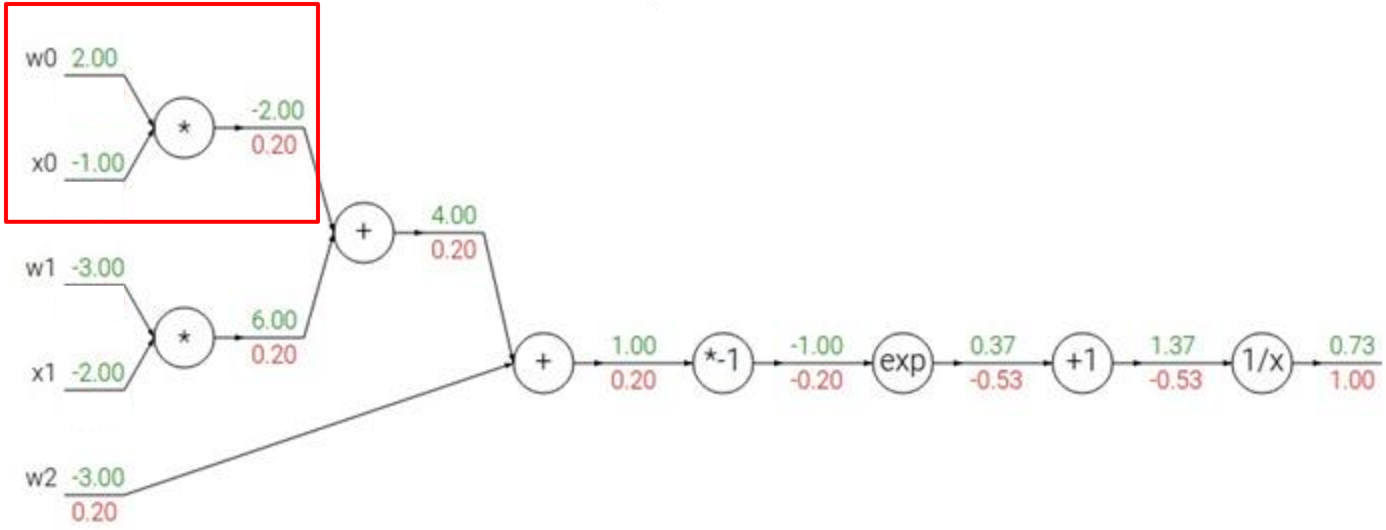
$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$

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

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

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

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

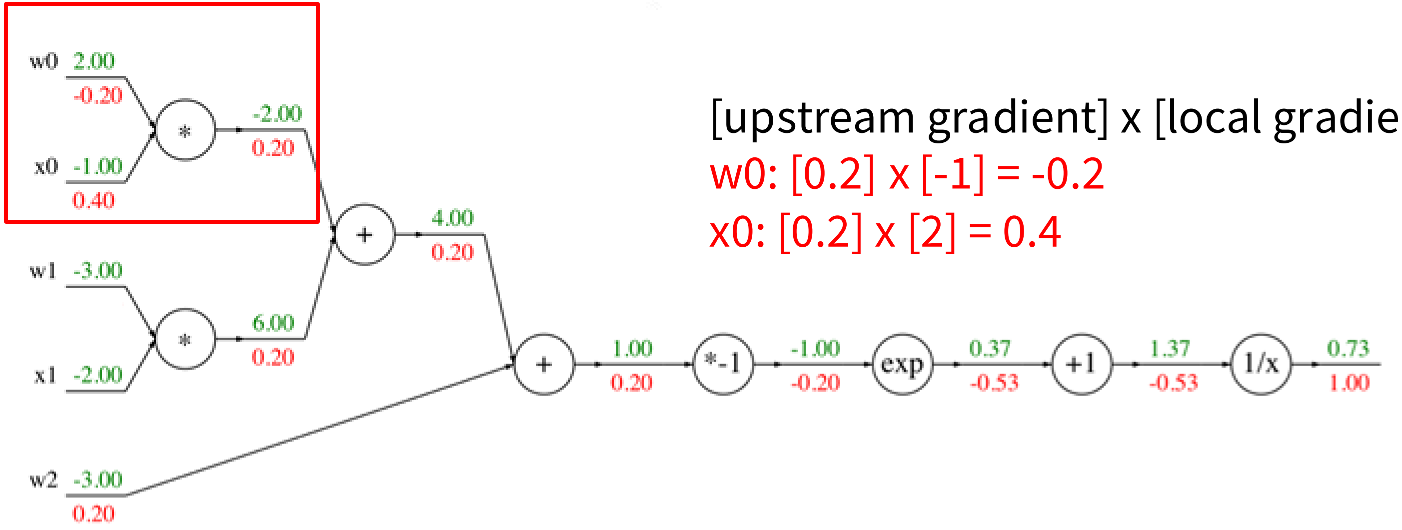


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \qquad\qquad f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

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



[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$	$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$	$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

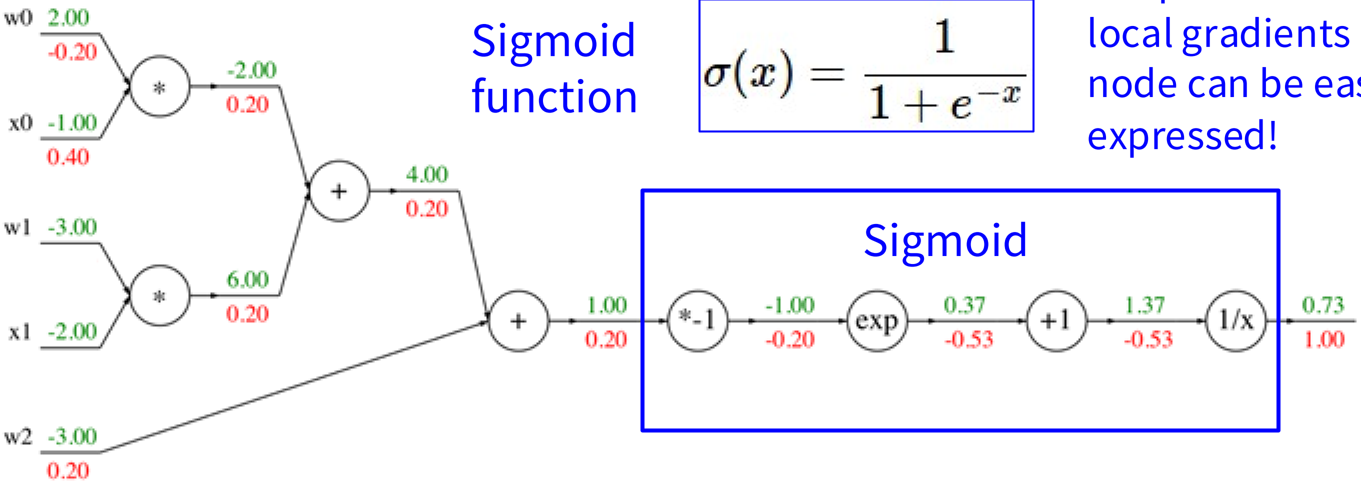
Another example:

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

Sigmoid function

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



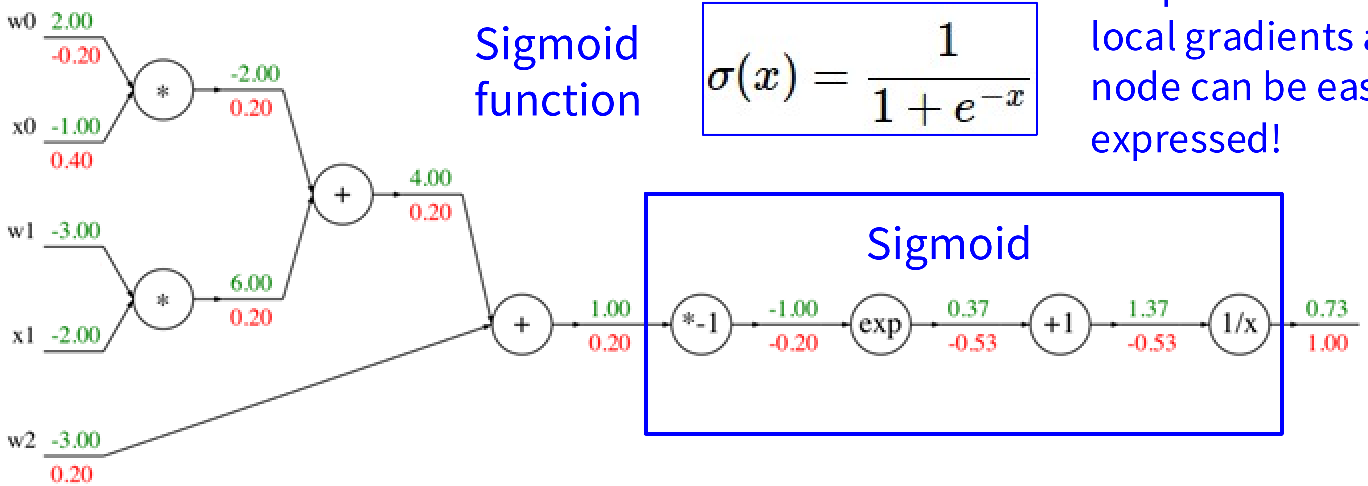
Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid function

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



Sigmoid local gradient:

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

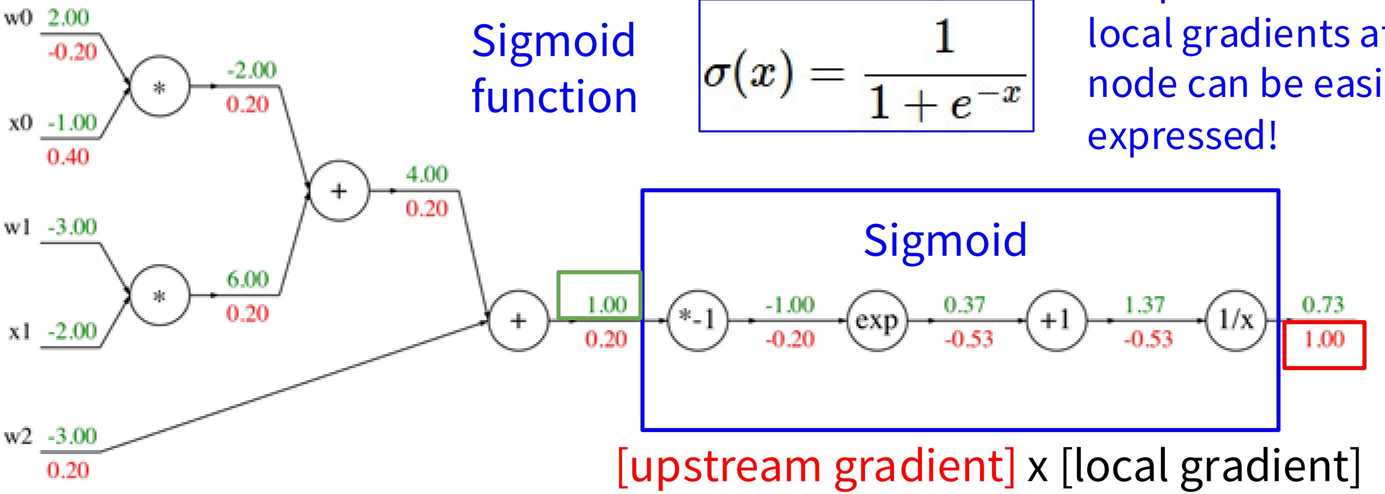
Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid function

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



[upstream gradient] x [local gradient]

[1.00] x [(1 - 1/(1+e^{-1})) (1/(1+e^{-1}))] = 0.2

Sigmoid local gradient:

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

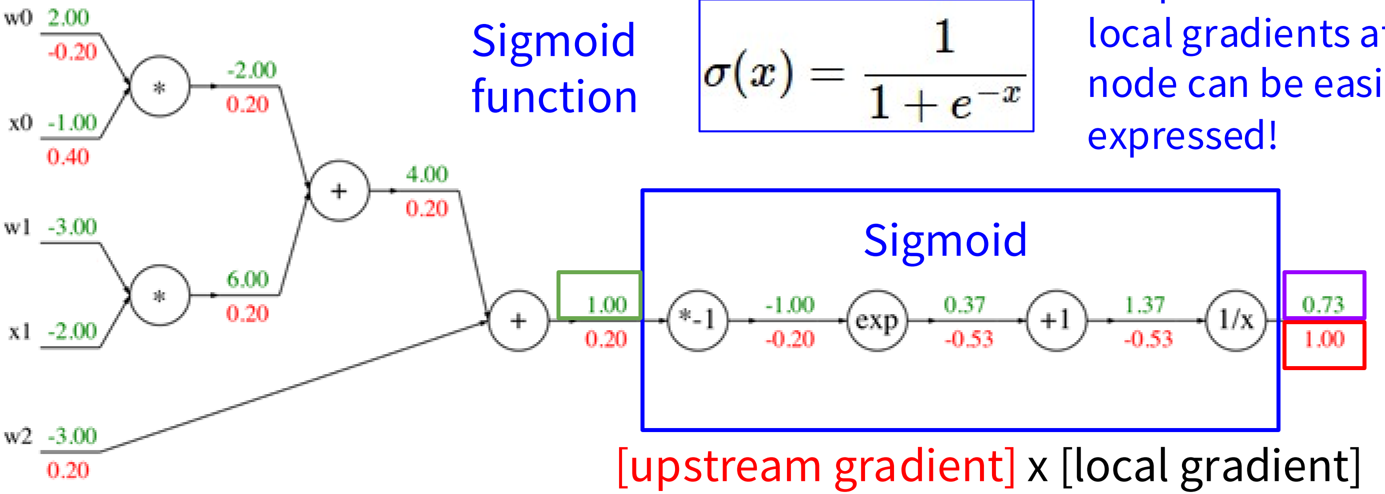
Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid function

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



[upstream gradient] x [local gradient]

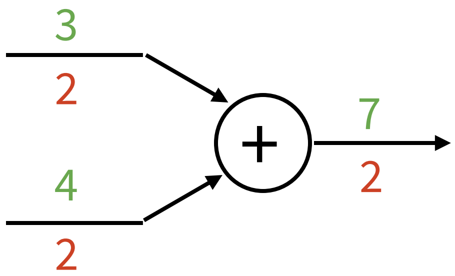
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

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

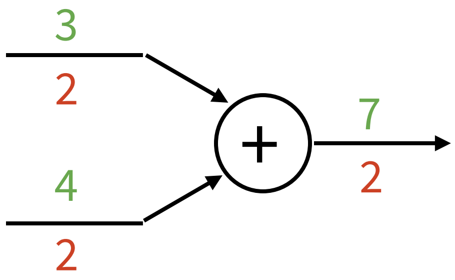
Patterns in gradient flow

add gate: gradient distributor

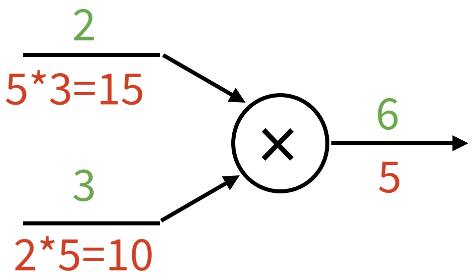


Patterns in gradient flow

add gate: gradient distributor

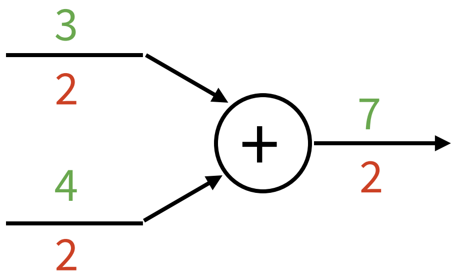


mul gate: "swap multiplier"

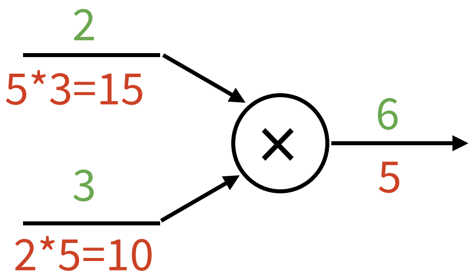


Patterns in gradient flow

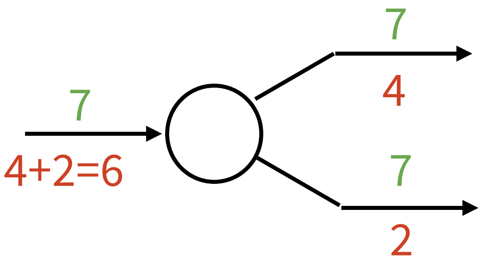
add gate: gradient distributor



mul gate: “swap multiplier”

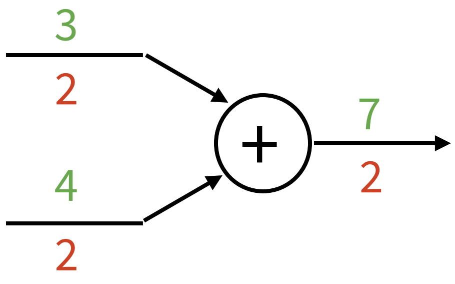


copy gate: gradient adder

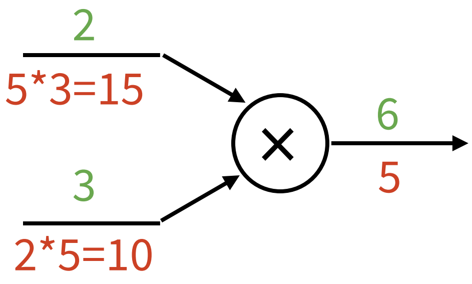


Patterns in gradient flow

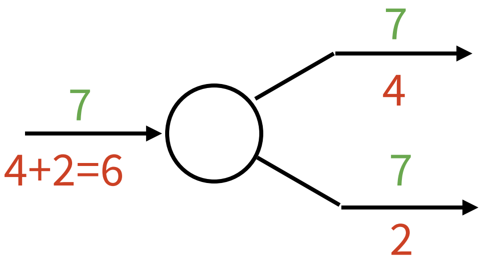
add gate: gradient distributor



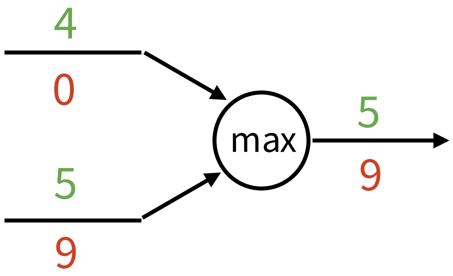
mul gate: “swap multiplier”



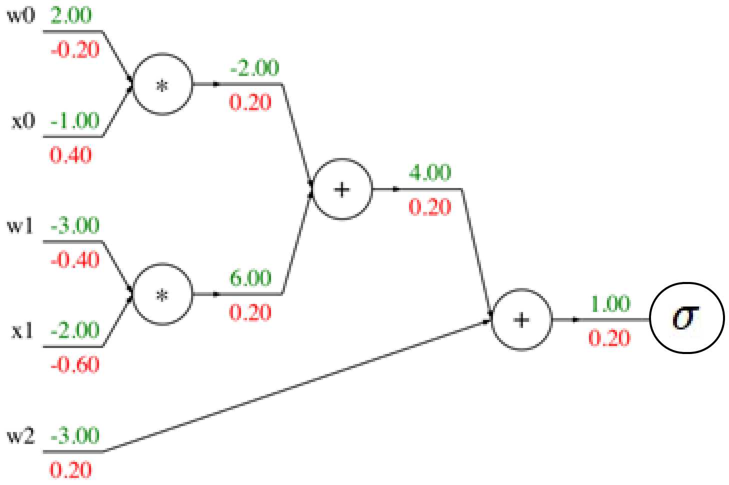
copy gate: gradient adder



max gate: gradient router



Backprop Implementation: "Flat" code



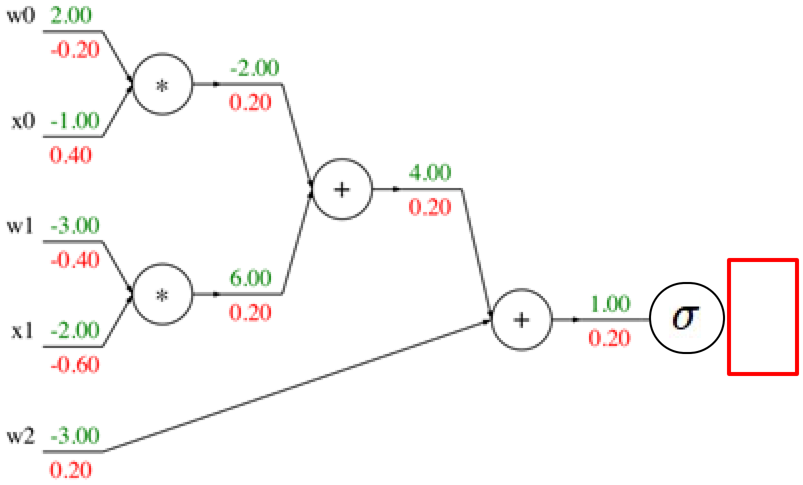
Forward pass:
Compute output

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

Backward pass:
Compute grads

```
    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 Implementation: “Flat” code



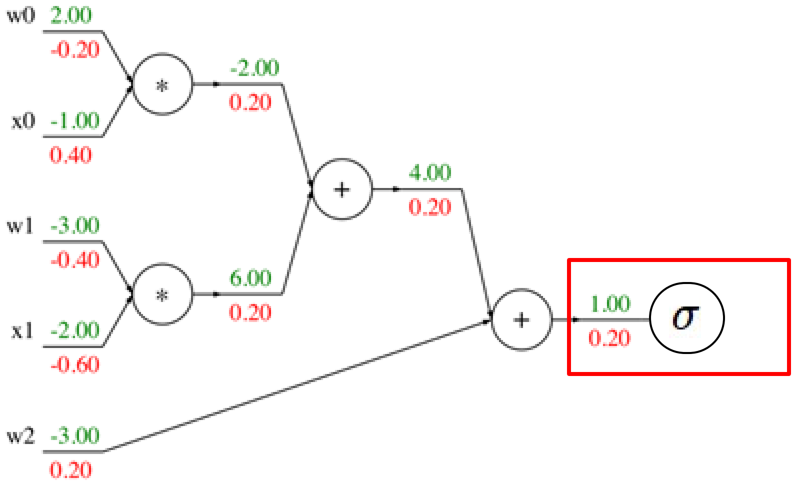
Forward pass: Compute output

Base case

```
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
```

Backprop Implementation: "Flat" code



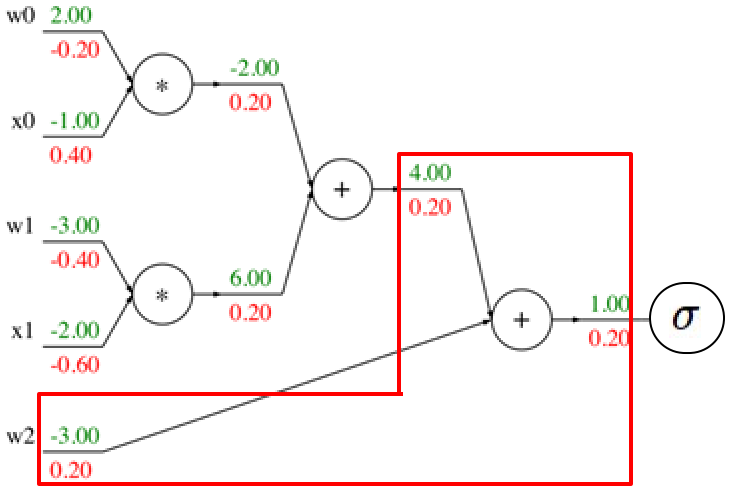
Forward pass:
Compute output

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

Sigmoid

```
    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 Implementation: "Flat" code



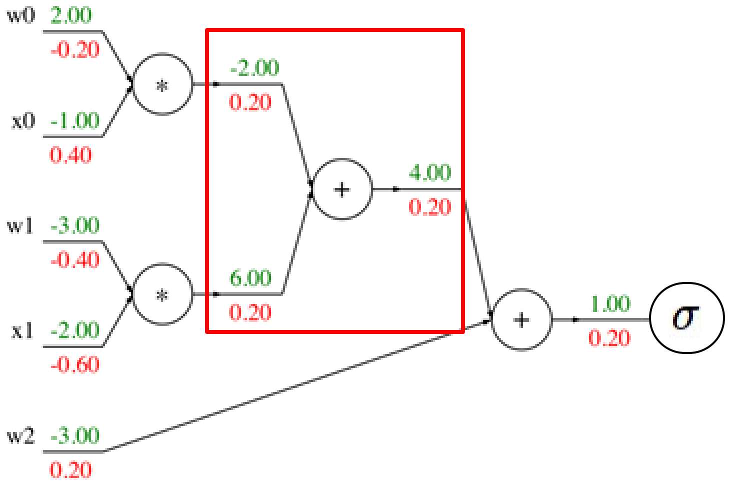
Forward pass: Compute output

```
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
```

Add gate

Backprop Implementation: "Flat" code



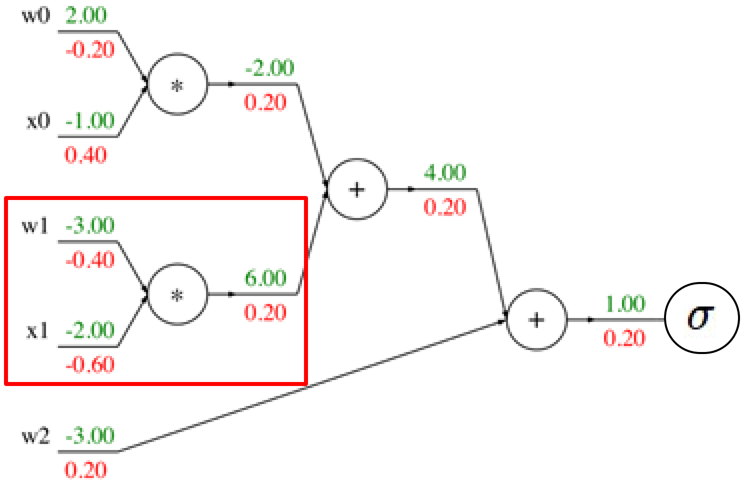
Forward pass:
Compute output

Add gate

```
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
```

Backprop Implementation: “Flat” code




```
def f(w0, x0, w1, x1, w2):
```

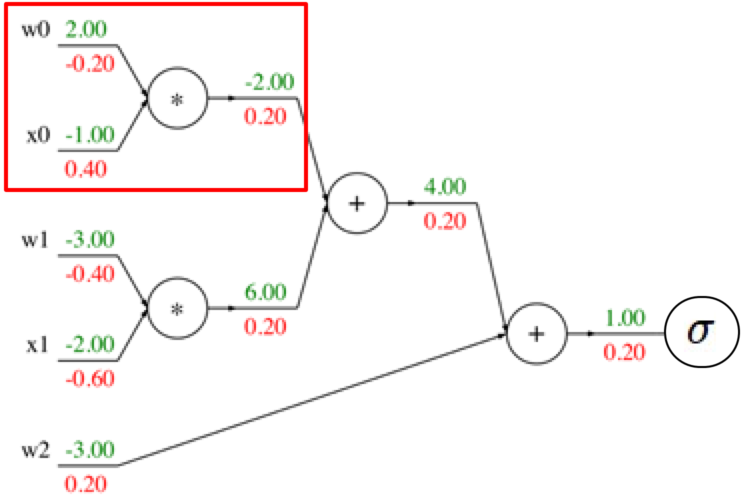
Forward pass:
Compute output

```
  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
```

Multiply gate

Backprop Implementation: "Flat" code



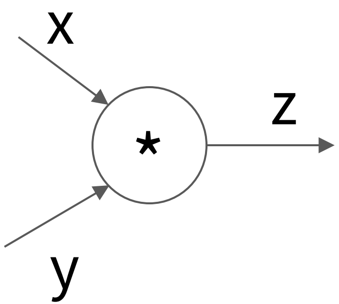
Forward pass:
Compute output

```
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
```

Multiply gate

Modularized implementation: forward / backward API



(x,y,z are scalars)

Gate / Node / Function object: Actual PyTorch code

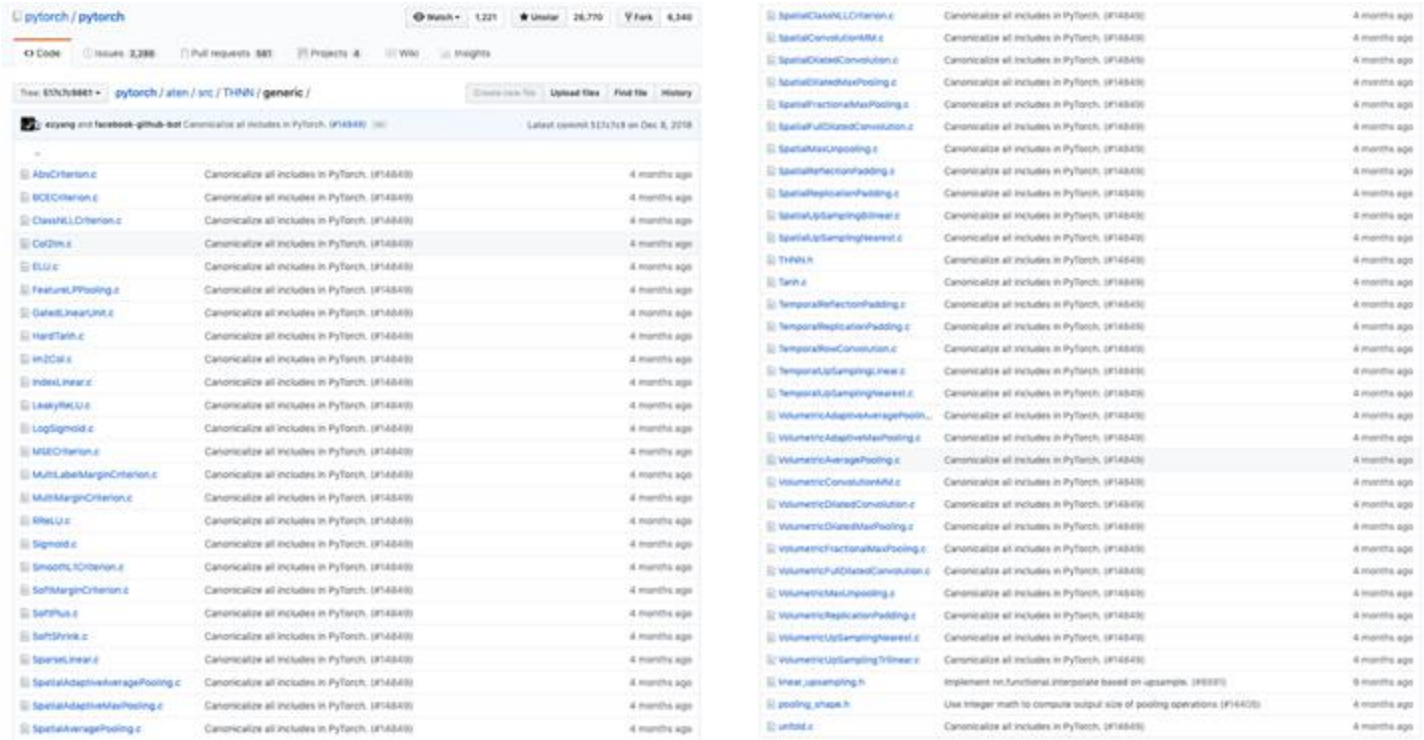
```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z  # dz/dx * dL/dz
    grad_y = x * grad_z  # dz/dy * dL/dz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

Need to cache some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Example: PyTorch operators



PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  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
```

Forward

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

Source

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  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
```

Forward

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

```
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

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  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
```

Forward

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

```
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...

Backward

$$\boxed{(1-\sigma(x))\,\sigma(x)}$$

Source

So far: backprop with scalars

What about vector-valued functions?

Recap: Vector derivatives

Scalar to Scalar

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

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

Scalar to Scalar

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

Regular derivative:

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

If $\boldsymbol{x}$ changes by a small amount, how much will $\boldsymbol{y}$ change?

Vector to Scalar

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

Derivative is Gradient:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

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

Recap: Vector derivatives

Scalar to Scalar

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

Regular derivative:

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

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

Vector to Scalar

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

Derivative is Gradient:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

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

Vector to Vector

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

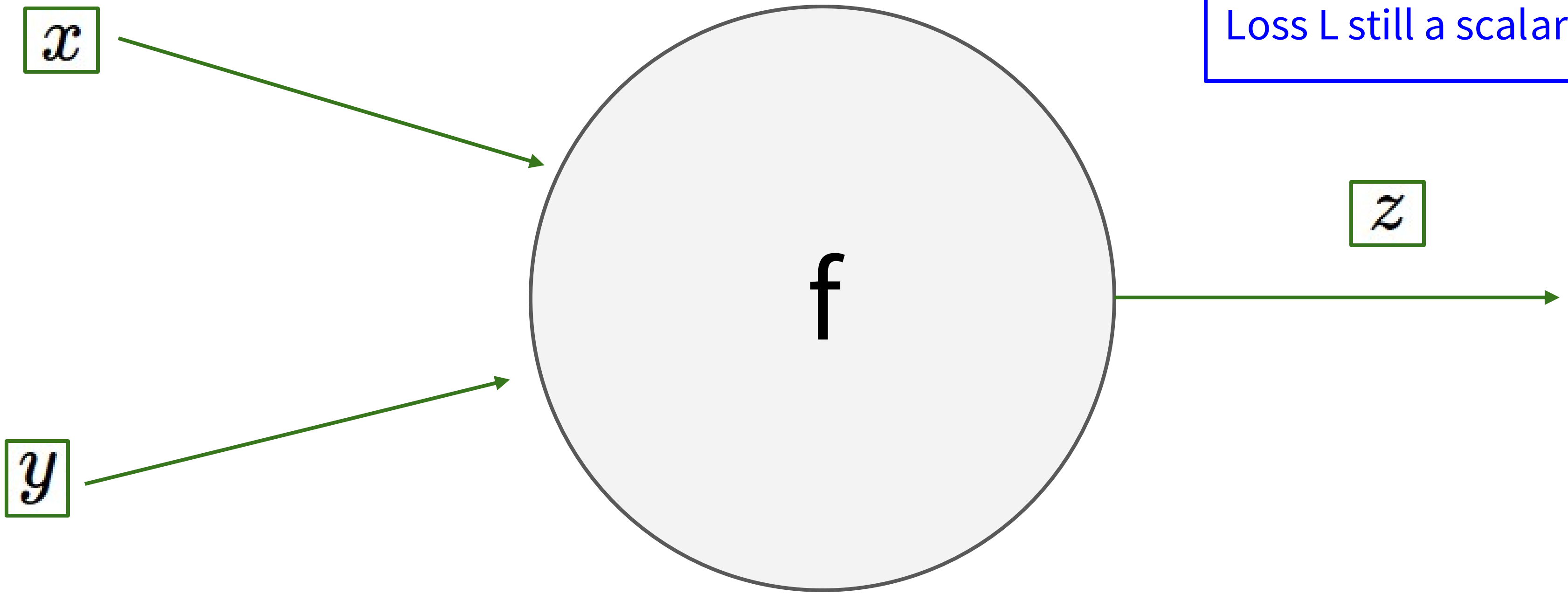
Derivative is Jacobian:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

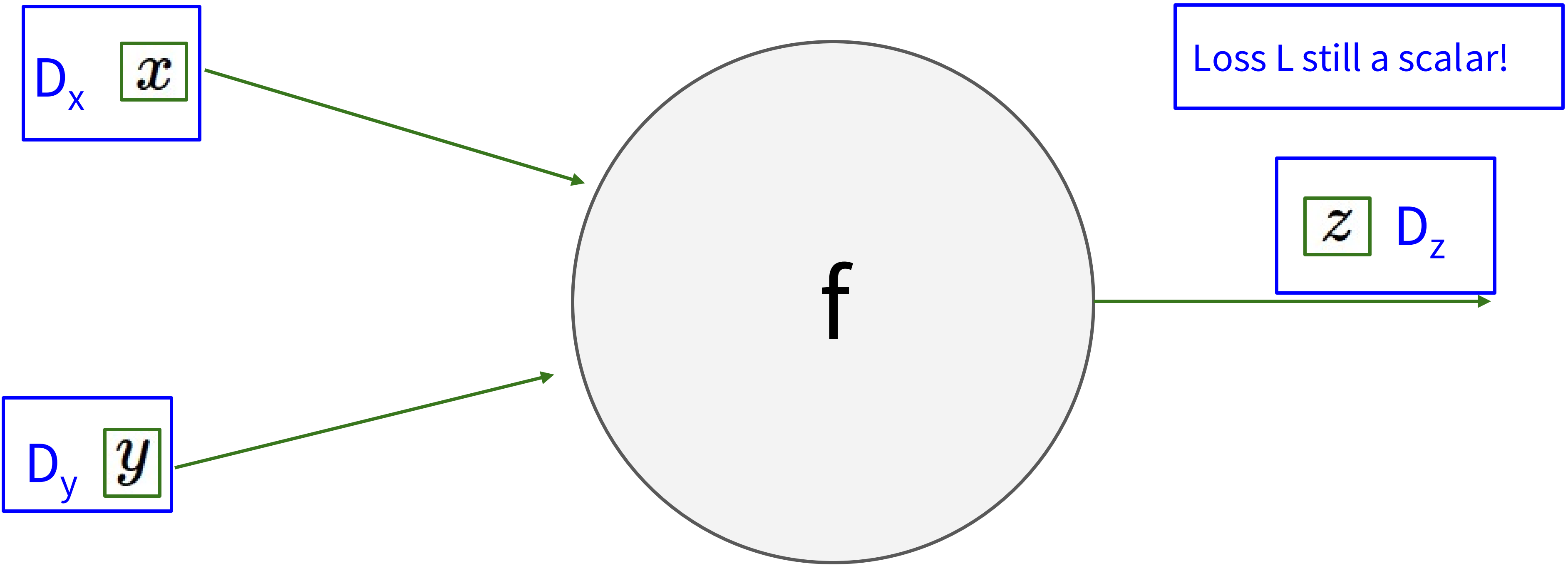
For each element of ***x***, if it changes by a small amount then how much will each element of ***y*** change?

Backprop with Vectors

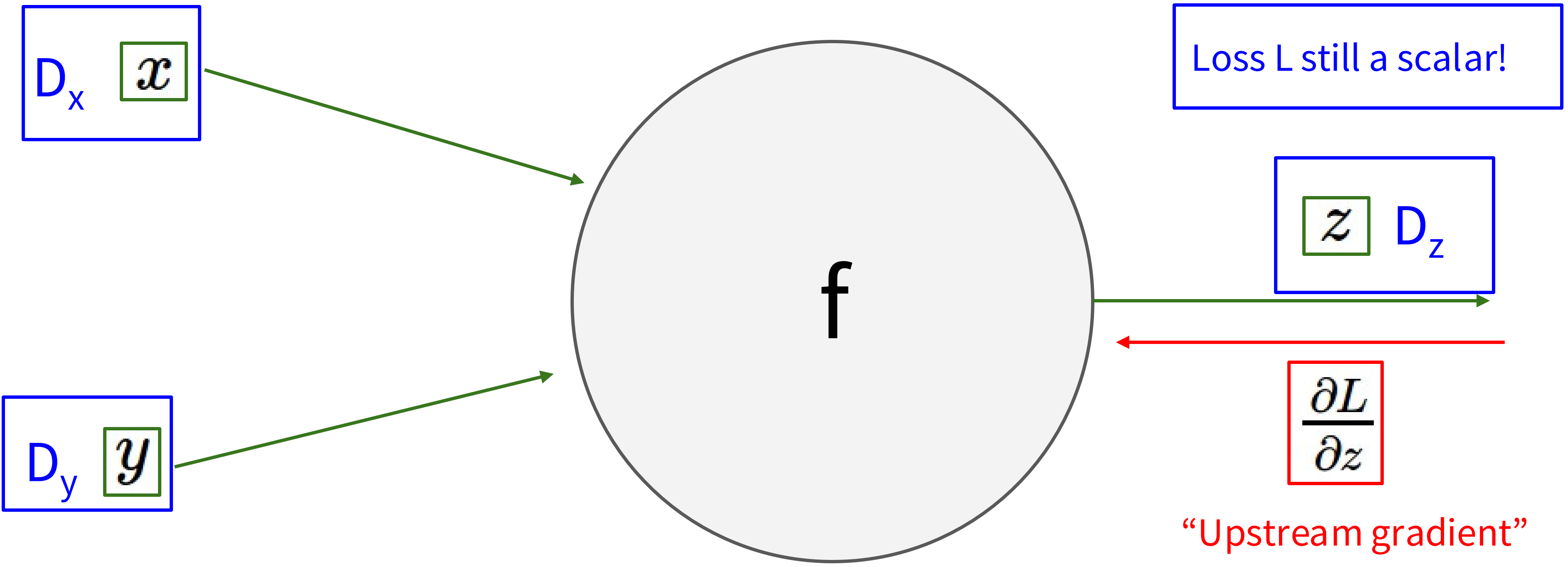
Loss L still a scalar!



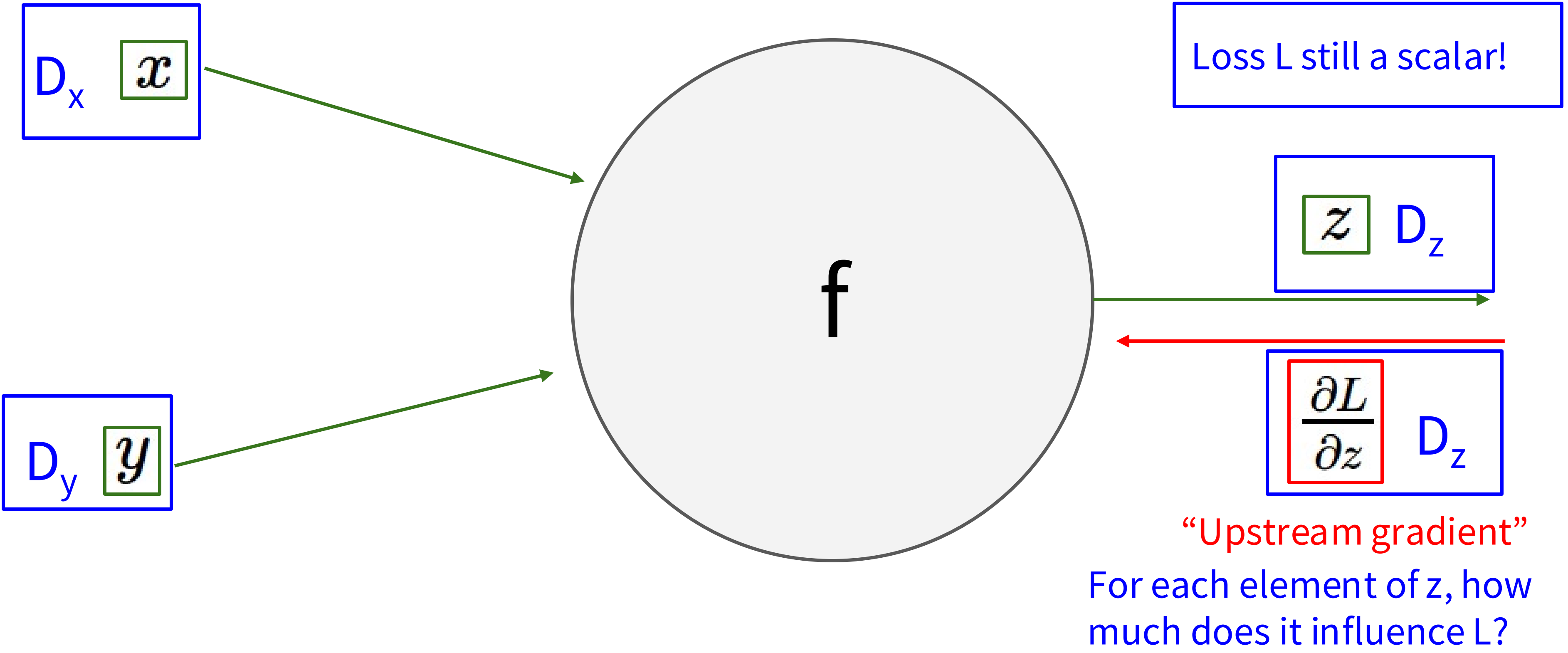
Backprop with Vectors



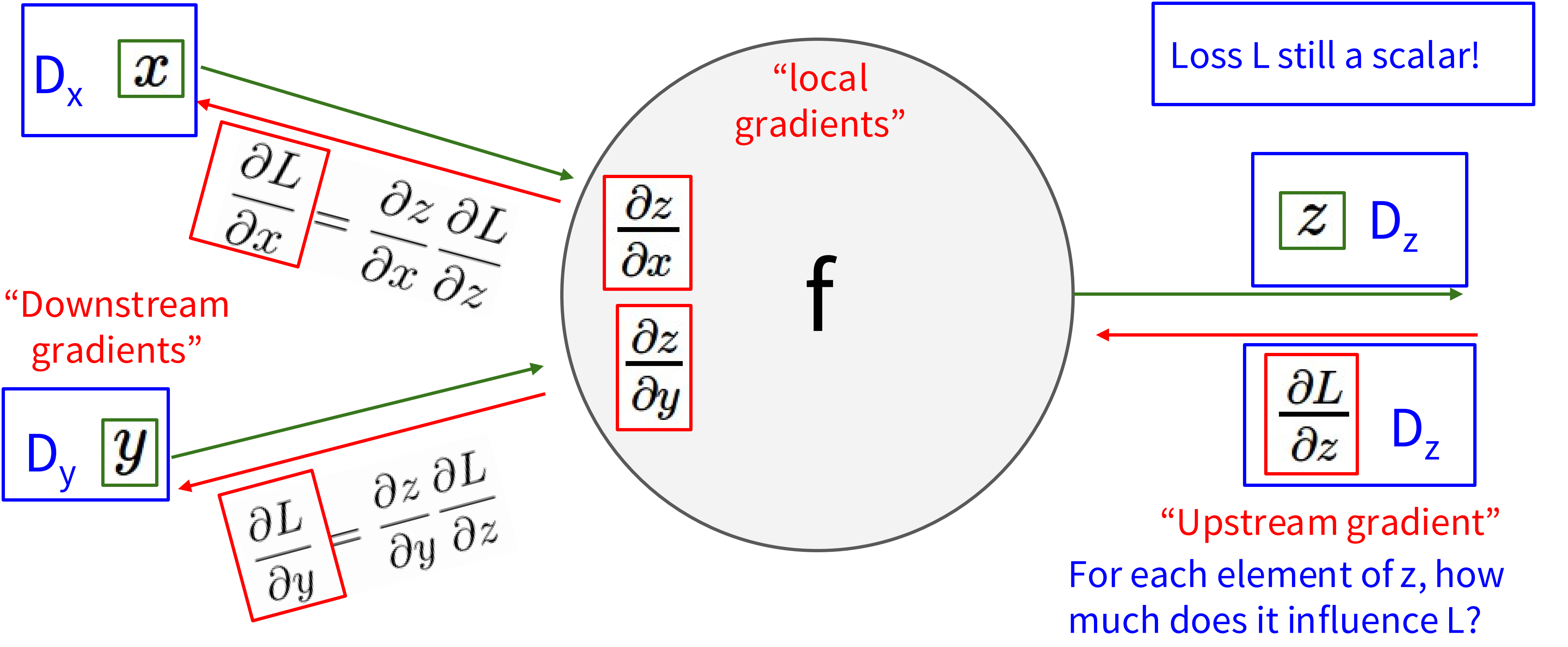
Backprop with Vectors



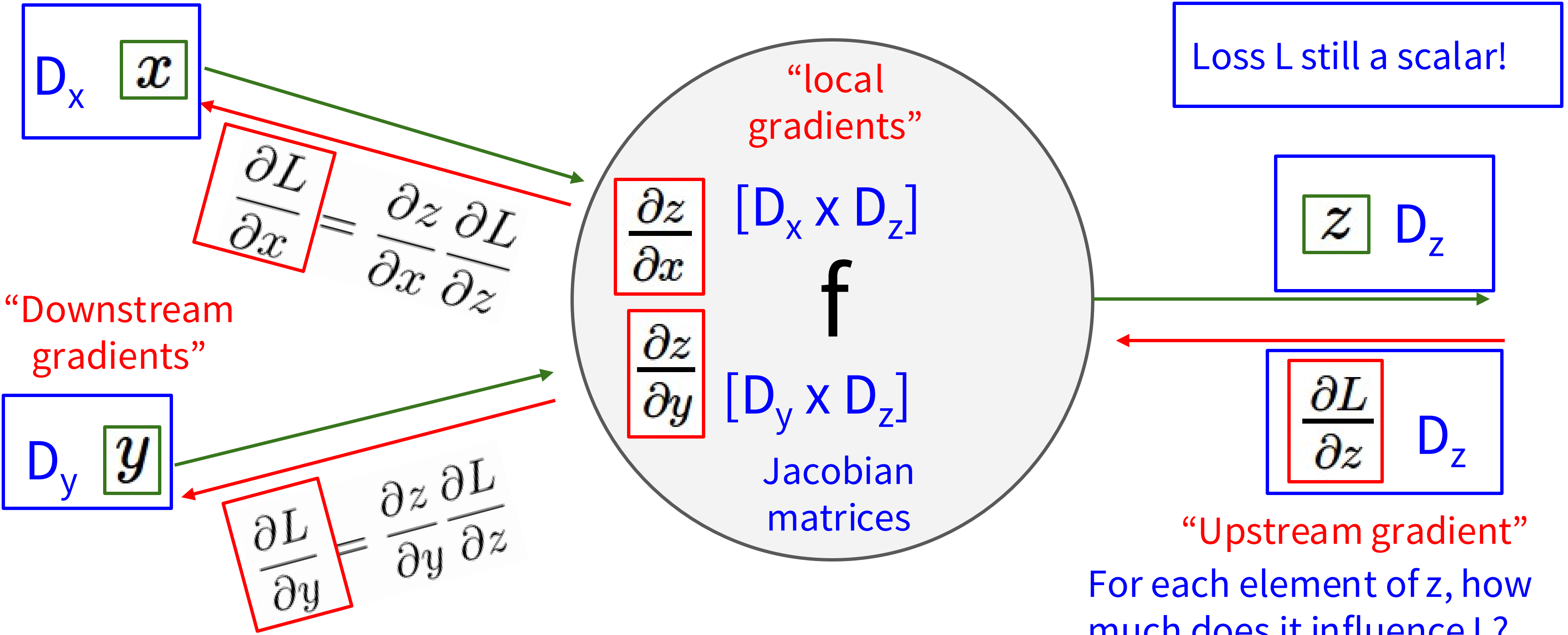
Backprop with Vectors



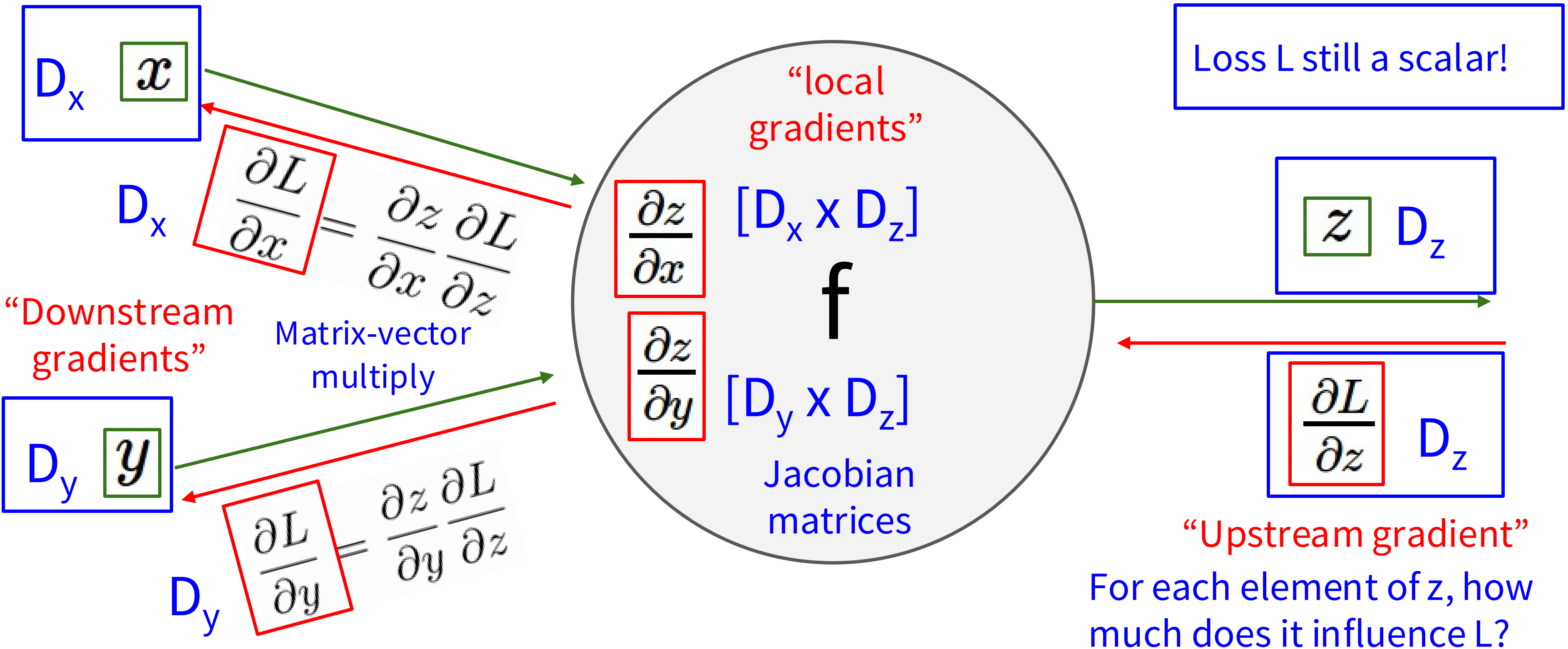
Backprop with Vectors



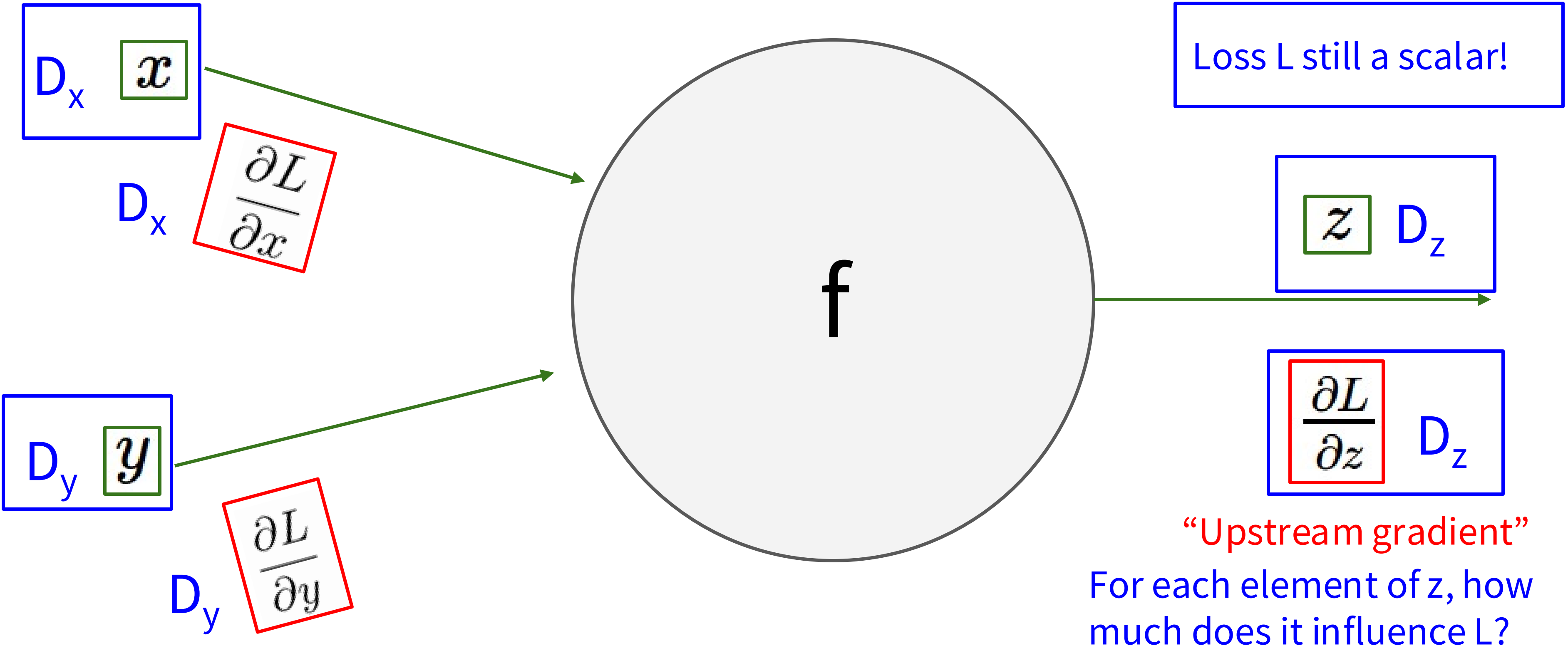
Backprop with Vectors



Backprop with Vectors



Gradients of variables wrt loss have same dims as the original variable



Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow \boxed{\begin{array}{c} f(x) = \max(0,x) \\ \text{(elementwise)} \end{array}} \longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D output z:

Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output z:

[1]
[0]
[3]
[0]

4D dL/dz:

[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$$

f(x) = max(0,x)
(elementwise)

4D output z:

$$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Jacobian dz/dx

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4D dL/dz:

$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix}1\\-2\\3\\-1\end{bmatrix}$$

f(x) = max(0,x)
(elementwise)

4D output z:

$$\begin{bmatrix}1\\0\\3\\0\end{bmatrix}$$

[dz/dx] [dL/dz]

$$\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}4\\-1\\5\\9\end{bmatrix}$$

4D dL/dz:

$$\begin{bmatrix}4\\-1\\5\\9\end{bmatrix}$$

Upstream gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \rightarrow \boxed{\begin{array}{c} f(x) = \max(0,x) \\ \text{(elementwise)} \end{array}} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D output z:

4D dL/dx:

$$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix} \leftarrow$$

[dz/dx] [dL/dz]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow$$

4D dL/dz:

$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow$$

Upstream gradient

Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output z:

[1]
[0]
[3]
[0]

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

4D dL/dx:

[4]
[0]
[5]
[0]

[dz/dx] [dL/dz]

[1 0 0 0] [4]
[0 0 0 0] [-1]
[0 0 1 0] [5]
[0 0 0 0] [9]

4D dL/dz:

[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output z:

[1]
[0]
[3]
[0]

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

4D dL/dx:

[4]
[0]
[5]
[0]

[dz/dx] [dL/dz]

$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial z}\right)_i & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

4D dL/dz:

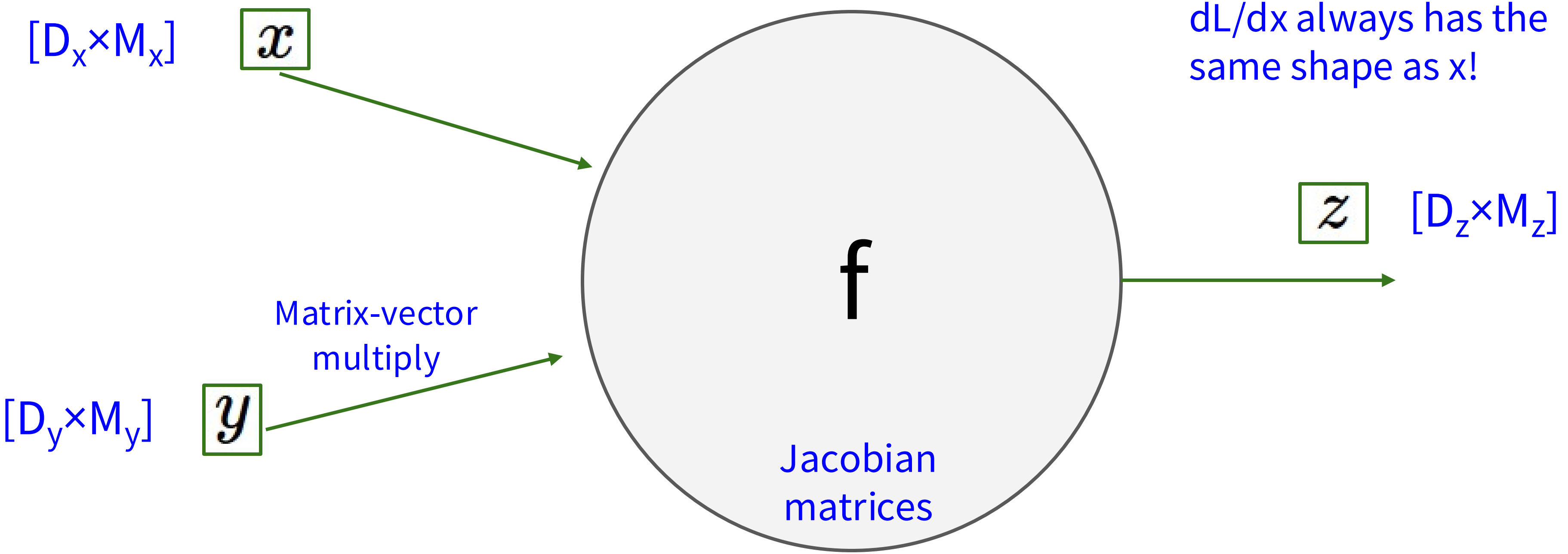
[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Matrices (or Tensors)

Loss L still a scalar!

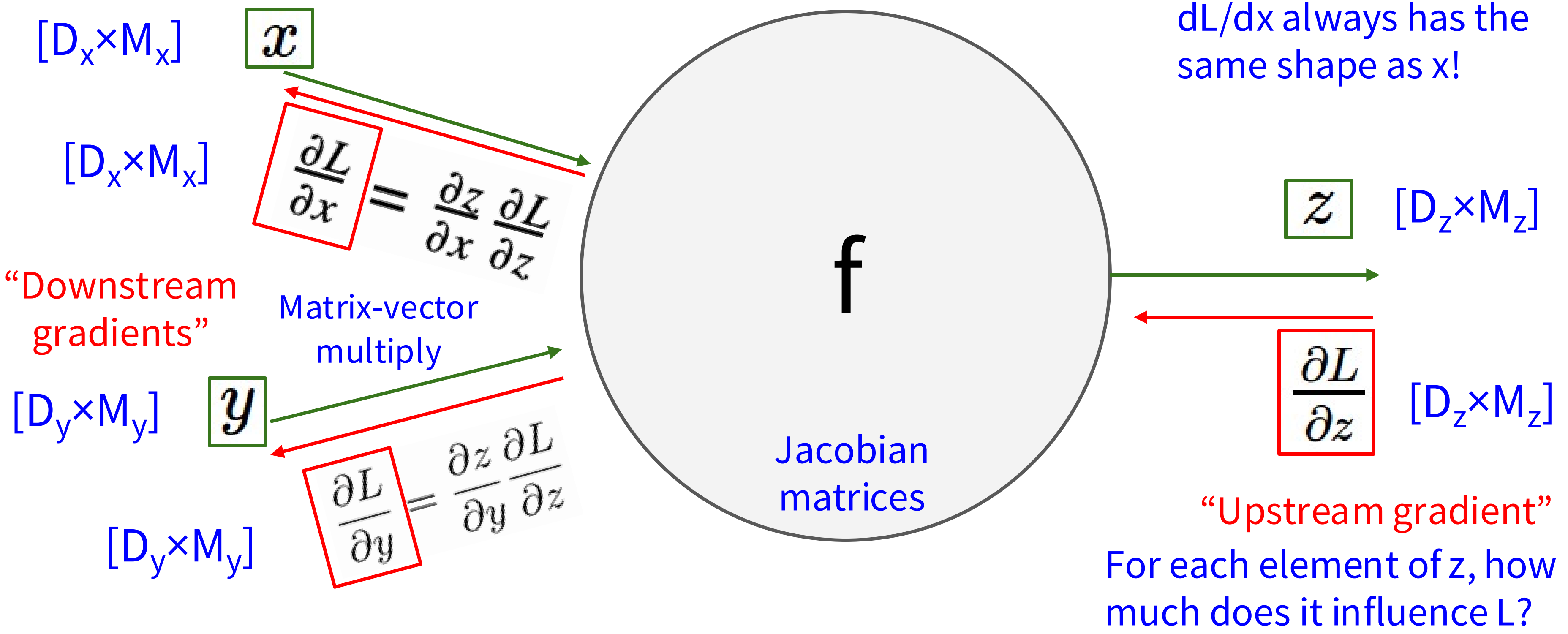
dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors)

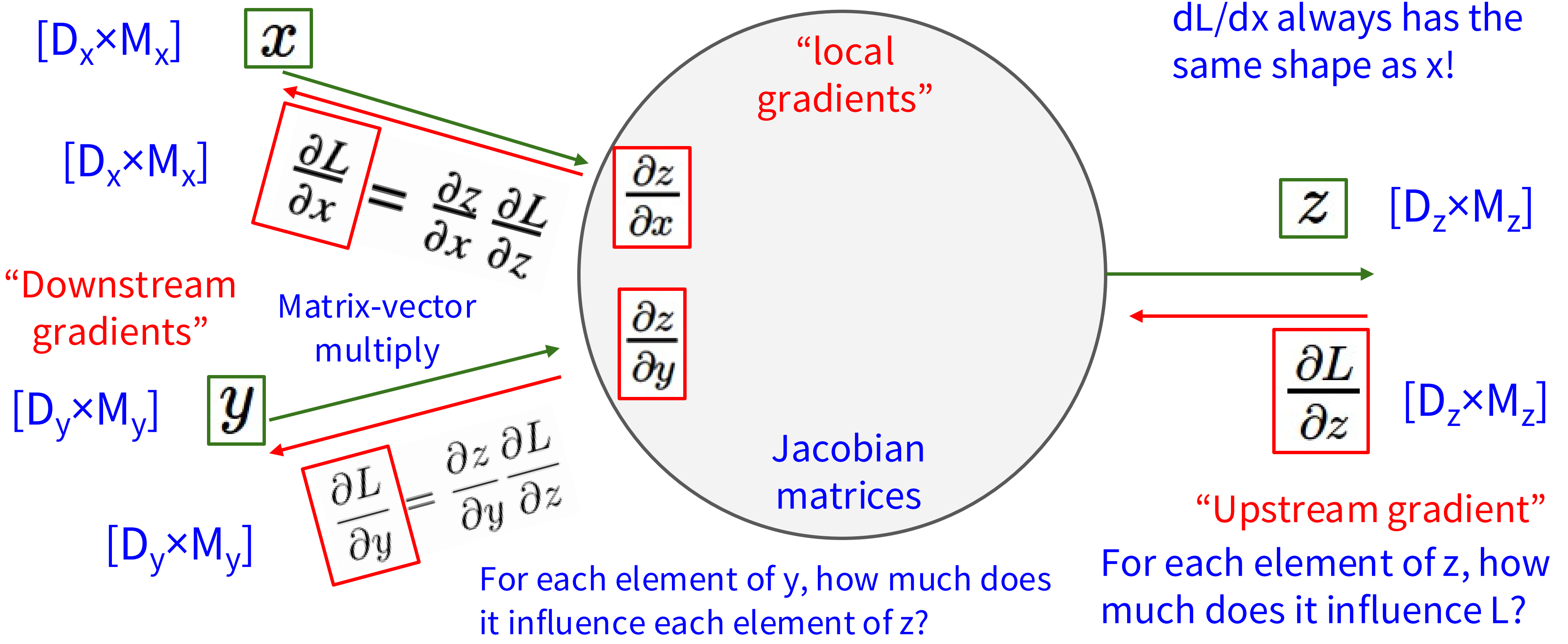
Loss L still a scalar!

dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors)

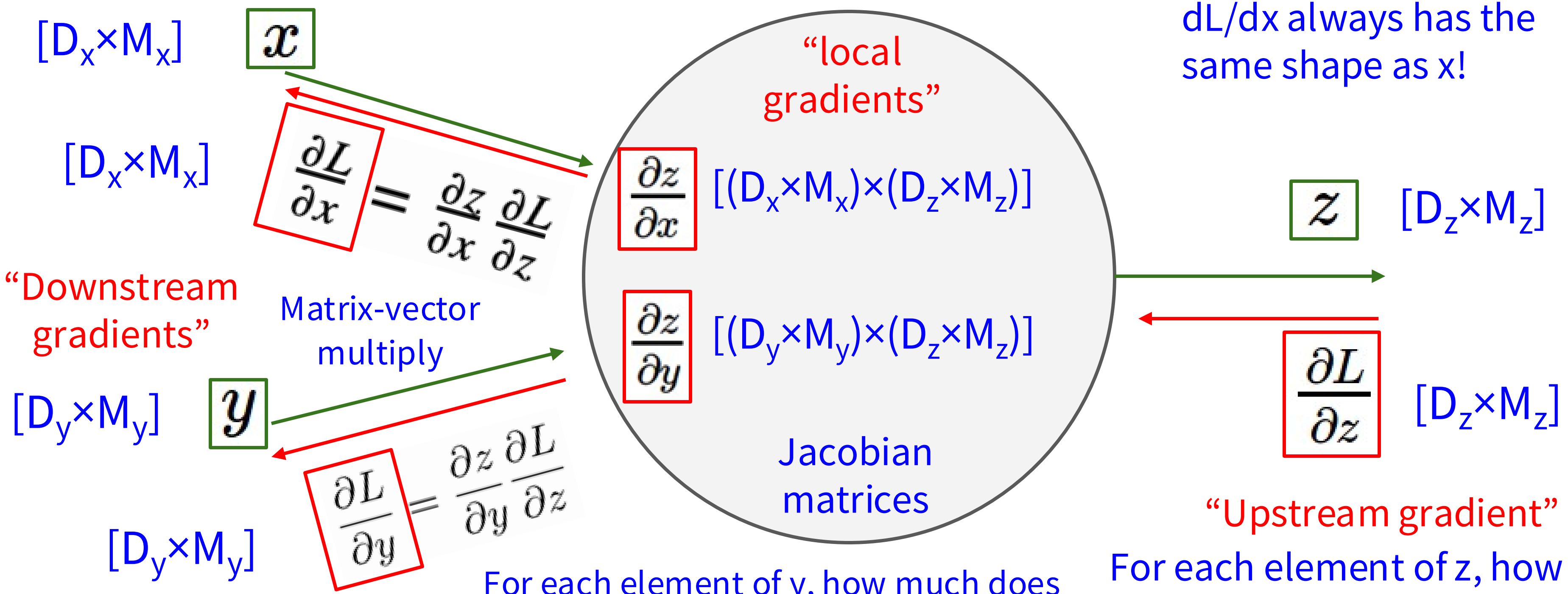
Loss L still a scalar!



Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!



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

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

Backprop with Matrices

x: [N×D]

[2 1 -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

[13 9 -2 -6]
[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

Also see derivation in the course notes:
http://cs231n.stanford.edu/handouts/linear-backprop.pdf

Backprop with Matrices

x: [N×D]
[2 1 -3]
[-3 4 2]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]
[13 9 -2 -6]
[5 2 17 1]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

Jacobians:
dy/dx: [(N×D)×(N×M)]
dy/dw: [(D×M)×(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!

Backprop with Matrices

x: [N×D]

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

Q: What parts of ***y*** are affected by one element of ***x***?

Backprop with Matrices

x: [N×D]

[2 $\boxed{1}$ -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$\boxed{[13\ 9\ -2\ -6]}$
[5 2 17 1]

dL/dy: [N×M]

[$\boxed{2\ 3\ -3\ 9}$]
[-8 1 4 6]

Q: What parts of y are affected by one element of x?

A: $\boxed{x_{n,d}}$ affects the whole row $y_{n,\cdot}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

Backprop with Matrices

x: [N×D]

[2 $\boxed{1}$ -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

[13 9 $\boxed{-2}$ -6]
[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

Q: What parts of y are affected by one element of x?

A: $\boxed{x_{n,d}}$ affects the whole row $y_{n,\cdot}$

Q: How much does $\boxed{x_{n,d}}$ affect $\boxed{y_{n,m}}$?

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

Backprop with Matrices

x: [N×D]

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$$\begin{bmatrix} 13 & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

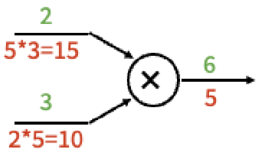
Q: What parts of y are affected by one element of x?

A: $\boxed{x_{n,d}}$ affects the whole row $y_{n,\cdot}$

Q: How much does $\boxed{x_{n,d}}$ affect $\boxed{y_{n,m}}$?

A: $\boxed{w_{d,m}}$

mul gate: "swap multiplier"



$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

Backprop with Matrices

x: [N×D]

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$$\begin{bmatrix} 13 & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

Q: What parts of y are affected by one element of x?

A: $\boxed{x_{n,d}}$ affects the whole row $y_{n,\cdot}$

Q: How much does $\boxed{x_{n,d}}$ affect $\boxed{y_{n,m}}$?

A: $\boxed{w_{d,m}}$

[N×D] [N×M] [M×D]

$$\boxed{\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T}$$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

Backprop with Matrices

x: [N×D]

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$$\begin{bmatrix} 13 & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

By similar logic:

[N×D] [N×M] [M×D]

$$\boxed{\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T}$$

[D×M] [D×N] [N×M]

$$\boxed{\frac{\partial L}{\partial w} = x^T \left(\frac{\partial L}{\partial y}\right)}$$

These formulas are easy to remember: they are the only way to make shapes match up!

Summary for today:

- (Fully-connected) Neural Networks are stacks of linear functions and nonlinear activation functions; they have much more representational power than linear classifiers
- backpropagation = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the forward() / backward() API
- forward: compute result of an operation and save any intermediates needed for gradient computation in memory
- backward: apply the chain rule to compute the gradient of the loss function with respect to the inputs

Next Time: Convolutional Neural Networks!