Backpropagation

CS5330, Huaizu Jiang

Fall 2021, Northeastern University

Recap

Batch Gradient Descent

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

Problem: Full sum is expensive when N is large!

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

Solution: Approximate sum using a **minibatch** of examples, e.g. *B*=32

Slide credit: D Fouhey & J Johnson

Stochastic Gradient Descent (SGD)

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

Problem: Full sum is expensive when N is large!

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

Solution: Approximate sum using a **minibatch** of examples, e.g. *B*=32

```
# Stochastic gradient descent
w = initialize_weights()
for t in range(num_steps):
  minibatch = sample_data(data, batch_size)
  dw = compute_gradient(loss_fn, minibatch, w)
  w -= learning_rate * dw
```

Hyperparameters:

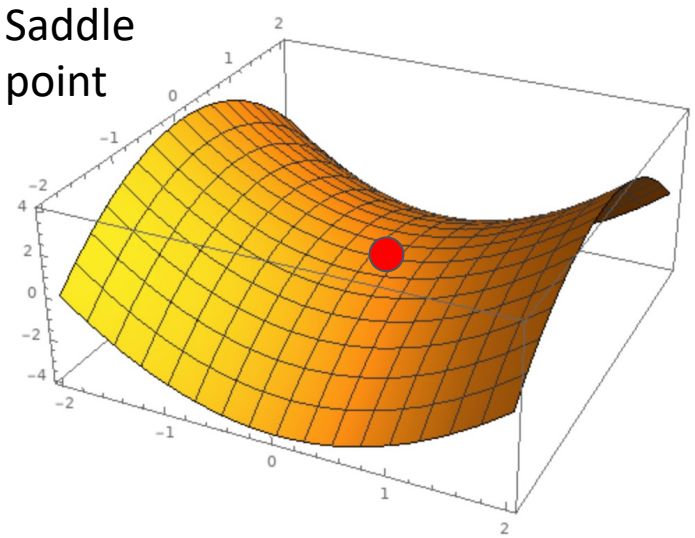
- Weight initialization
- Number of steps
- Learning rate
- Batch size
- Data sampling

Slide credit: D Fouhey & J Johnson

Problems with SGD

What if the loss function has a **local minimum** or **saddle point**?

Gradient is zero,
SGD gets stuck



Slide credit: D Fouhey & J Johnson

Problems with SGD

Our gradients come from minibatches so they can be noisy!

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

Slide credit: D Fouhey & J Johnson

SGD

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: D Fouhey & J Johnson

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

SGD + Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

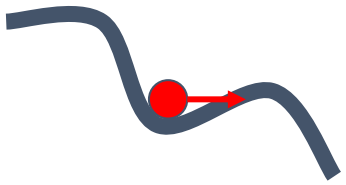
- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically $\rho = 0.9$ or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

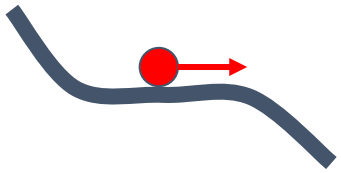
Slide credit: D Fouhey & J Johnson

SGD + Momentum

Local Minima



Saddle points



Gradient Noise

SGD SGD+Momentum

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: D Fouhey & J Johnson

Optimizing Everything

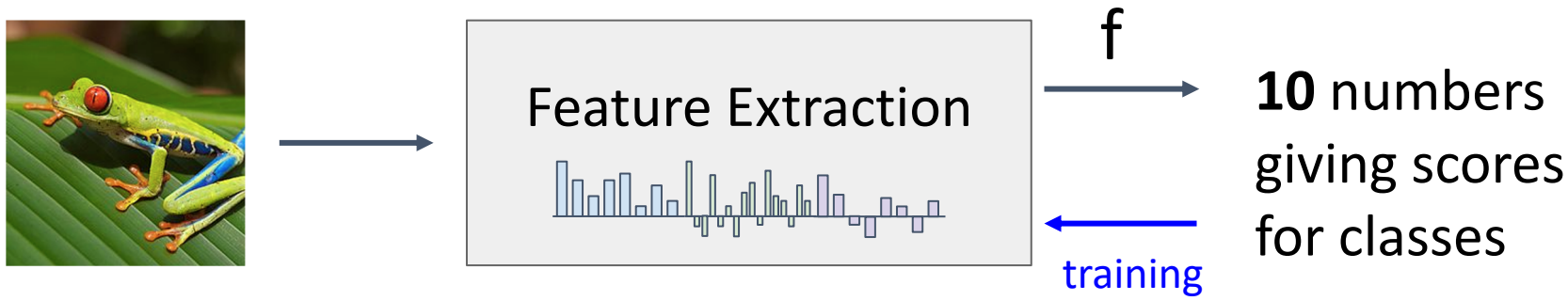
$$L(\boldsymbol{W}) = \boldsymbol{\lambda}\|\boldsymbol{W}\|_{\boldsymbol{2}}^{\boldsymbol{2}} + \sum_{i=1}^{n} -\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k))}\right)$$

$$L(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}}\right)^2$$

- (Automatically) Optimize **w** on training set with SGD to maximize training accuracy
- (Manually) Optimize λ with random/grid search to maximize validation accuracy

Slide credit: D Fouhey & J Johnson

Image Features vs Neural Networks

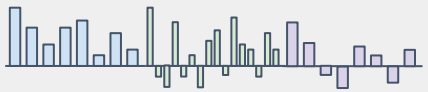


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012

Slide credit: D Fouhey & J Johnson

Image Features vs Neural Networks

Feature Extraction

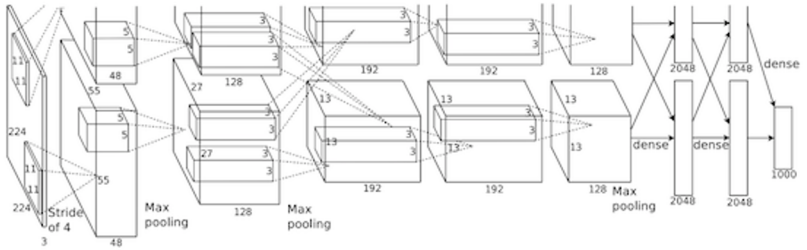


f

10 numbers giving scores for classes

training

Deep Neural Network



10 numbers giving scores for classes

training

Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

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

In practice we add a learnable bias
+b after each matrix multiply

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

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

2-layer Neural Net:

$$s = \mathrm{W}_2 \max(0, W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D}$$

$$W_2 \in \mathbb{R}^{C \times H}$$

In practice we add a learnable bias
+b after each matrix multiply

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

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

2-layer Neural Net:

$$s = \mathrm{W}_2 \max(0, W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D}$$

$$W_2 \in \mathbb{R}^{C \times H}$$

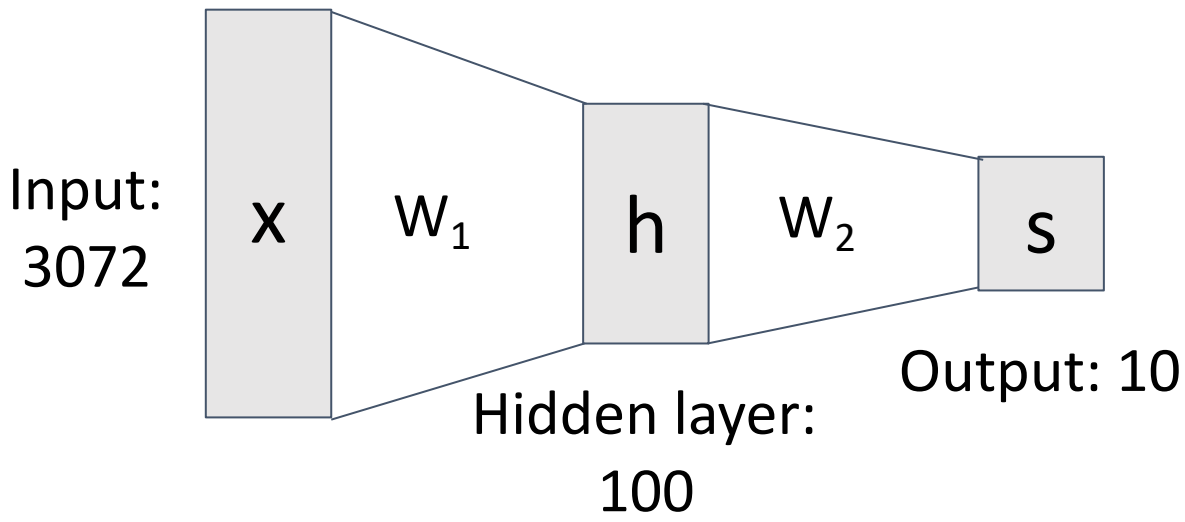
3-layer Neural Net:

$$s = \mathrm{W}_3 \max(0, W_2 \max(0, W_1 x))$$

Slide credit: D Fouhey & J Johnson

Neural Networks

Two-Layer Neural Network: $s = \mathrm{W}_2 \max(0, W_1 x)$



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

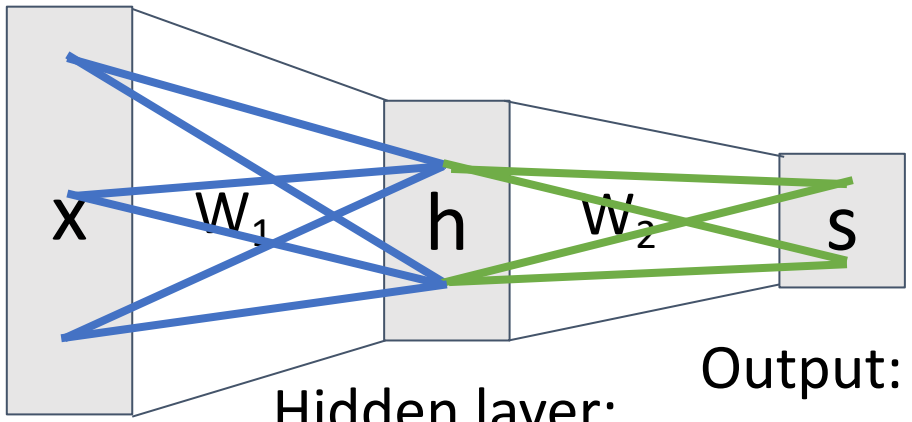
Slide credit: D Fouhey & J Johnson

Neural Networks

Two-Layer Neural Network: $s = W_2 \max(0, W_1 x)$

Element (i, j) of W_1 gives the effect on h_i from x_j

Element (i, j) of W_2 gives the effect on s_i from h_j



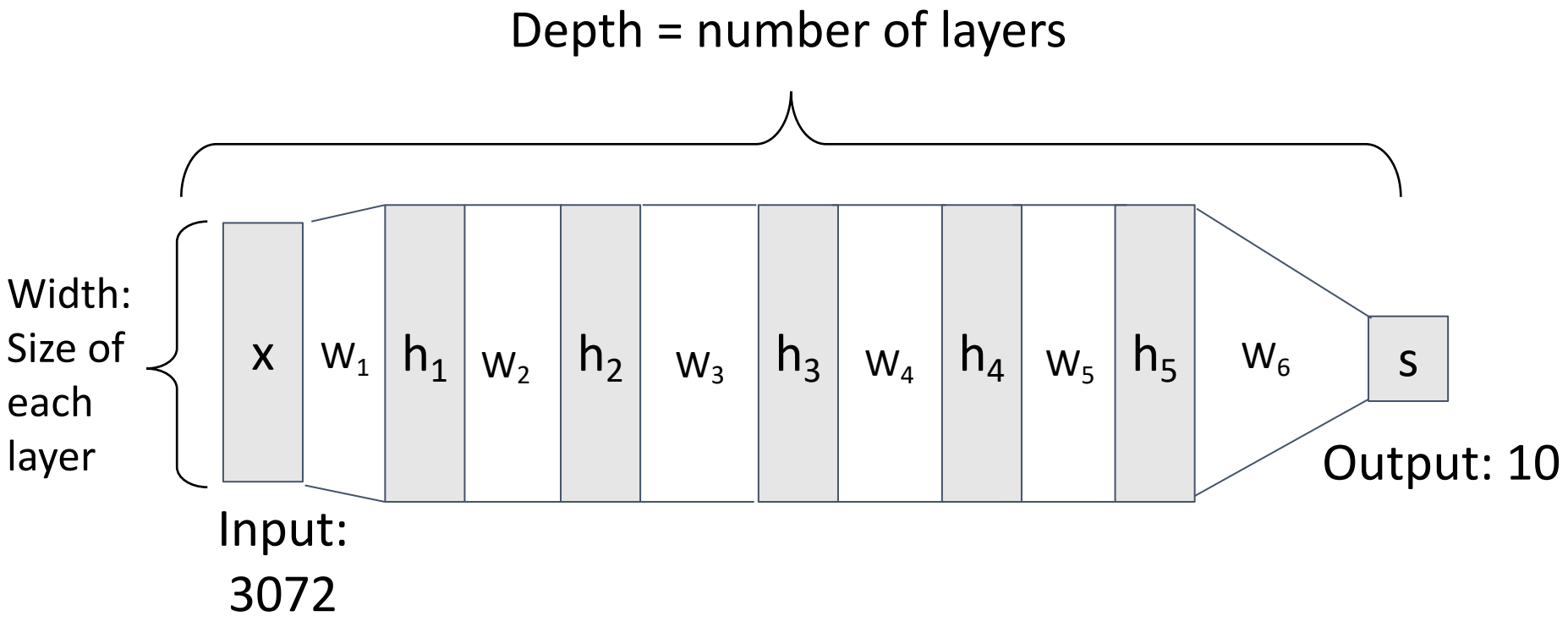
All elements of x affect all elements of h

All elements of h affect all elements of s

"Fully-Connected" neural network
Also "Multi-Layer Perceptron" (MLP)

Slide credit: D Fouhey & J Johnson

Deep Neural Networks



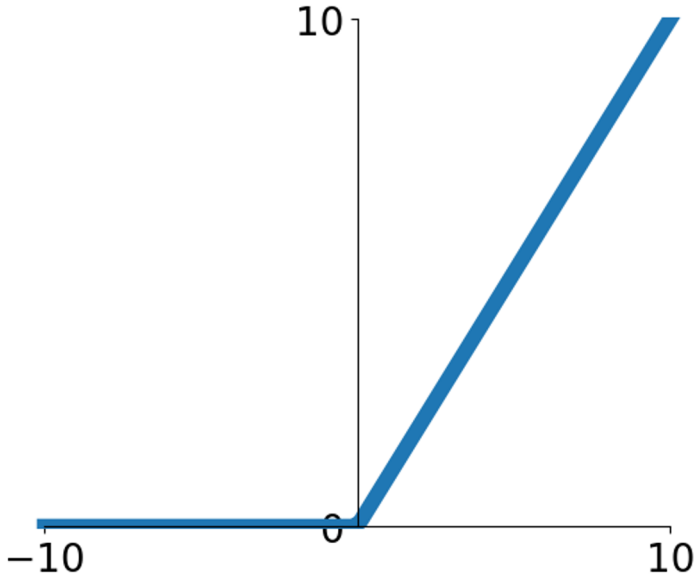
$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



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

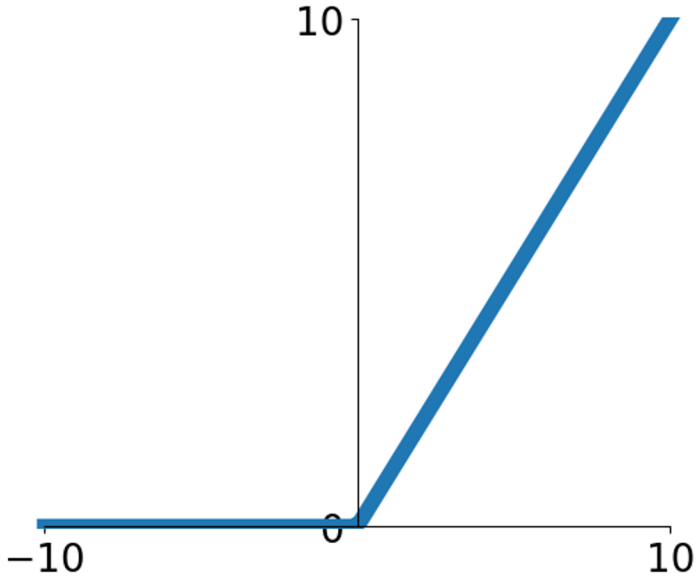
This is called the **activation function** of the neural network

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



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

This is called the **activation function** of the neural network

Q: What happens if we build a neural network with no activation function?

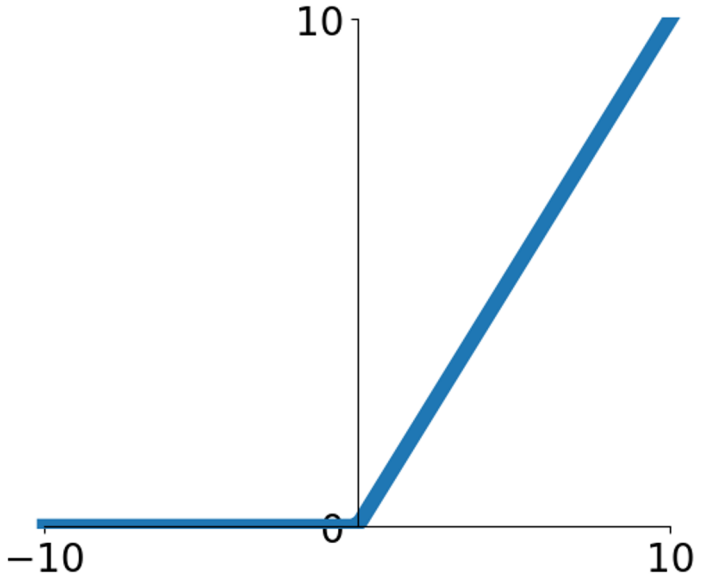
$$s = W_2 W_1 x$$

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



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

This is called the **activation function** of the neural network

Q: What happens if we build a neural network with no activation function?

$$s = W_2 W_1 x$$

A: We get a linear classifier!

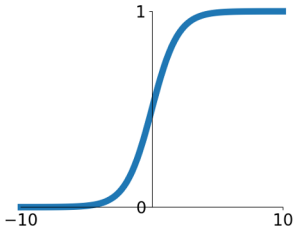
$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times D}$$
$$s = W_3 x$$

Slide credit: D Fouhey & J Johnson

Activation Functions

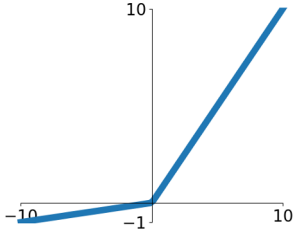
Sigmoid

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



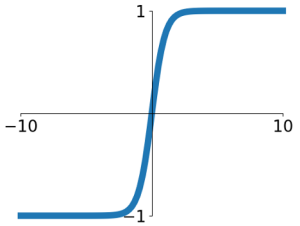
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

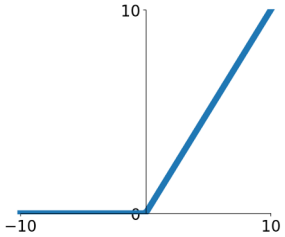


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

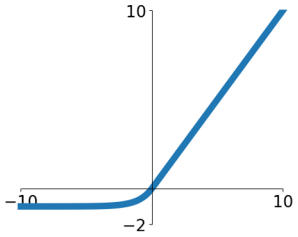
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



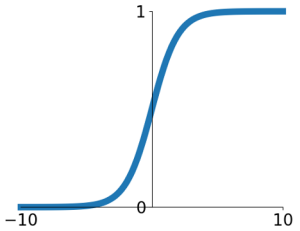
Slide credit: D Fouhey & J Johnson

Activation Functions

ReLU is a good default choice

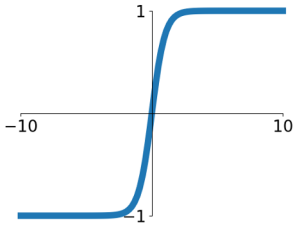
Sigmoid

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



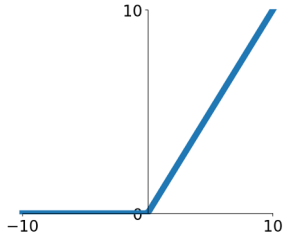
tanh

$$\tanh(x)$$



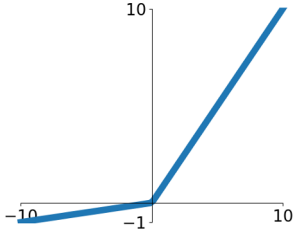
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

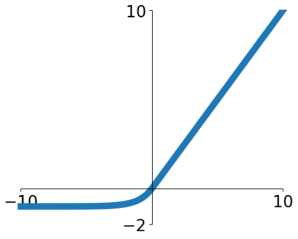


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Slide credit: D Fouhey & J Johnson

Today's class

Backpropagation: how to compute gradients and optimize parameters of neural networks

Problem: How to compute gradients?

$$s = W_2 \max(0, W_1 x) + b_2$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max\left(0, s_j - s_{y_i} + 1\right)$$ Per-element data loss

$$R(W) = \sum_k W_k^2$$ L2 Regularization

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

What do we need to optimize with SGD?

Compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$

Slide credit: D Fouhey & J Johnson

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

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

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

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

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

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

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

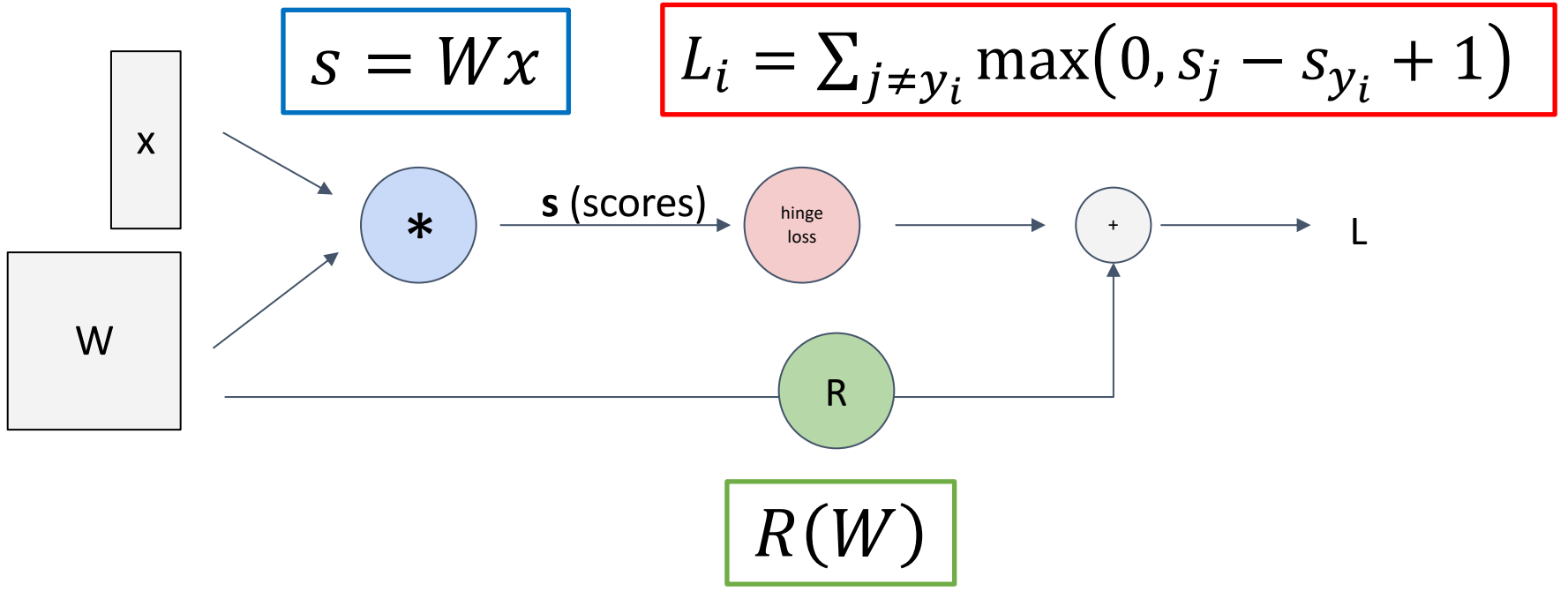
Problem: Very tedious: Lots of matrix calculus, need lots of paper

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

Problem: Not feasible for very complex models!

Slide credit: D Fouhey & J Johnson

Better Idea: **Computational Graphs**



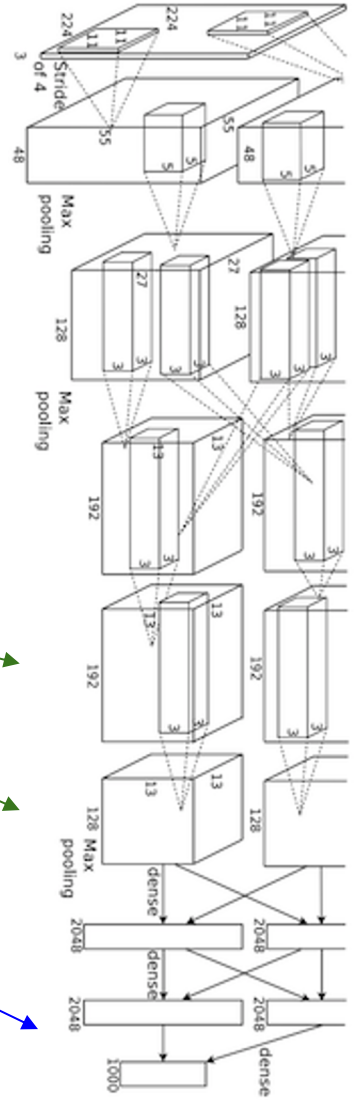
Slide credit: D Fouhey & J Johnson

Deep Network (AlexNet)

input image

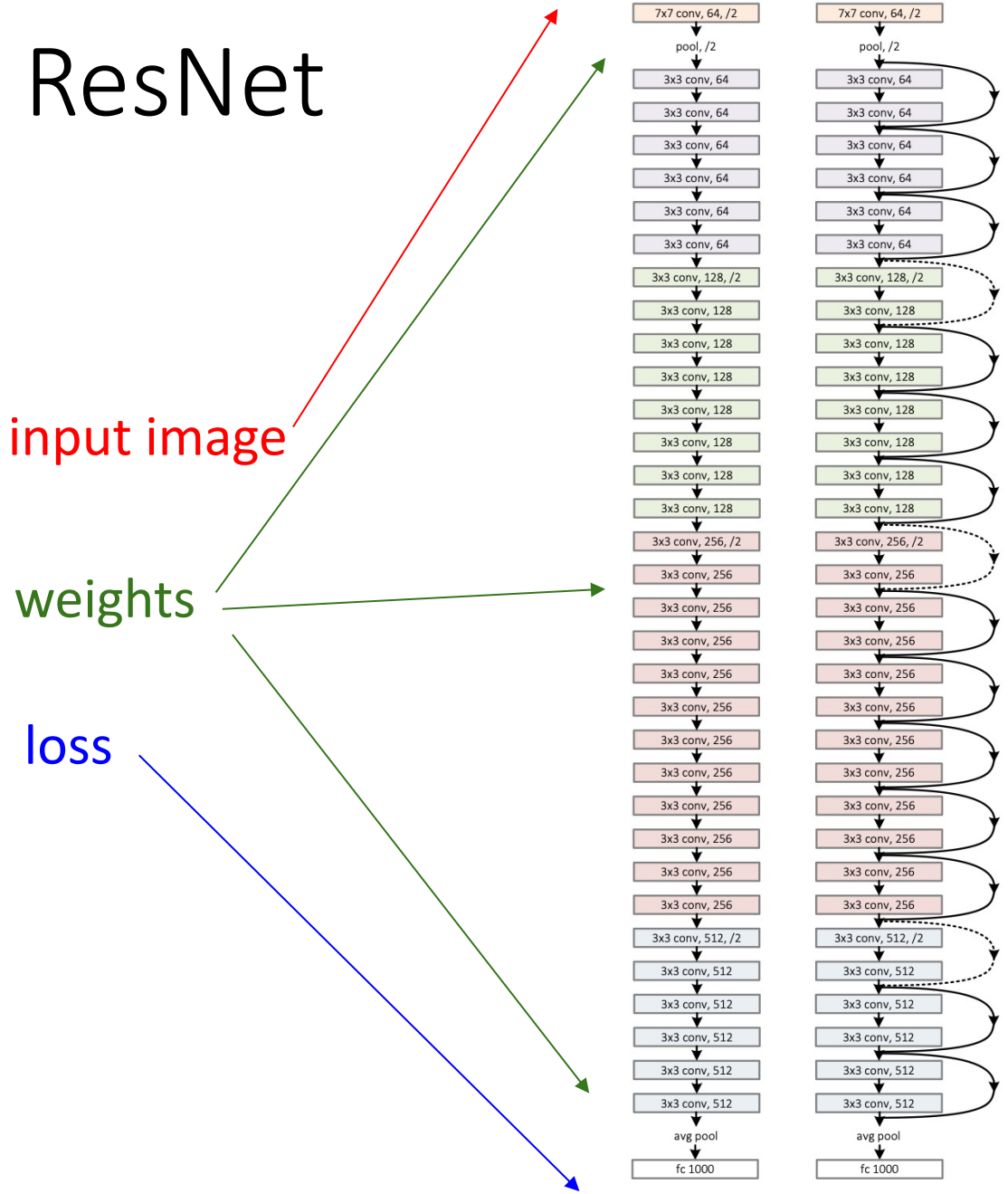
weights

loss



Slide credit: D Fouhey & J Johnson

VGG and ResNet



He et al, CVPR 2016

Overview of where we're going

- We want to **evaluate** the gradient of a Loss function L(x,W,...), with respect to the parameters (weights) of a neural network, at the "point" represented by the arguments to the function (x,W,...).
  - We are **not interested** in an **algebraic expression** for the gradient
  - but rather only in the **evaluation of that gradient at the current value of the function** arguments.

Slide credit: E Learned-Miller

Consider the function

$$z(x, y) = x^2 + y^2,$$

and suppose we are interested in evaluating the gradient of this function at the point

$$(x, y) = (5, 3).$$

Evaluate the gradient:

$$\frac{\partial z}{\partial x} = 2x.$$

$$\frac{\partial z}{\partial y} = 2y.$$

The algebraic expression of the gradient is just the collection of these partials into a "vector":

$$\nabla z = \begin{bmatrix} 2x \\ 2y \end{bmatrix}.$$

Don't care about this

The evaluation of this gradient at the point $(x, y) = (5, 3)$ is simply

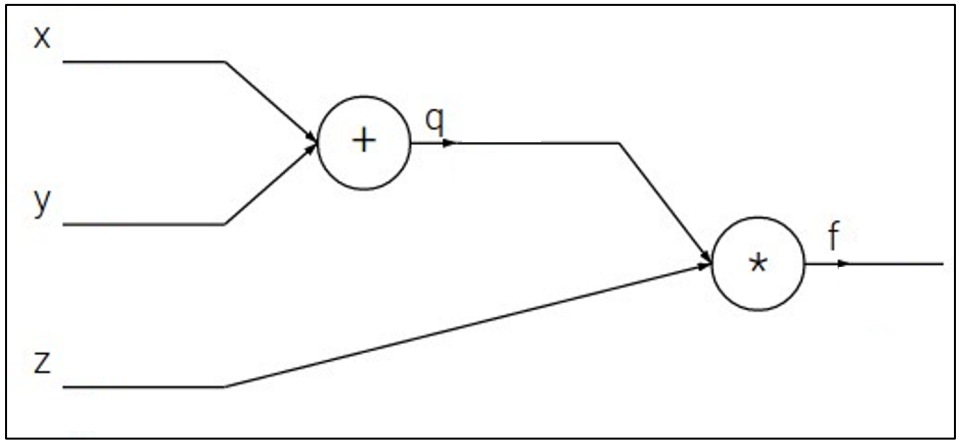
$$\nabla z(5, 3) = \begin{bmatrix} 2 \times 5 \\ 2 \times 3 \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \end{bmatrix}.$$

Do care about this

Slide credit: E Learned-Miller

Backpropagation: Simple Example

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

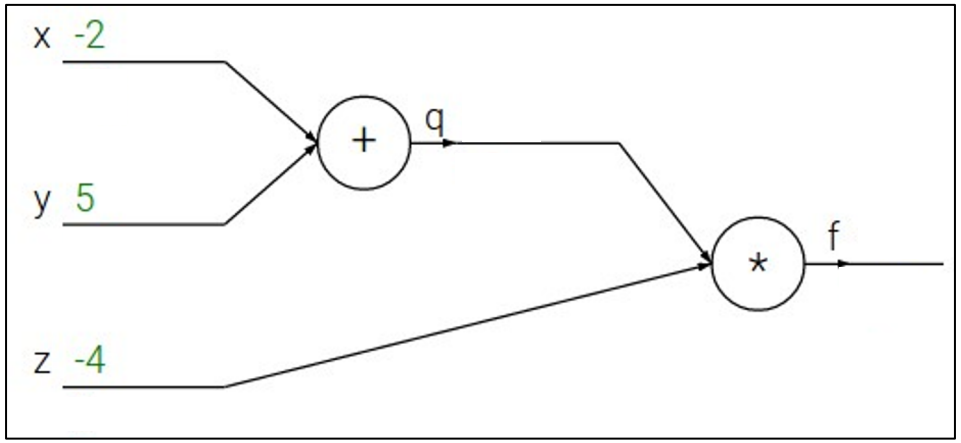


Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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

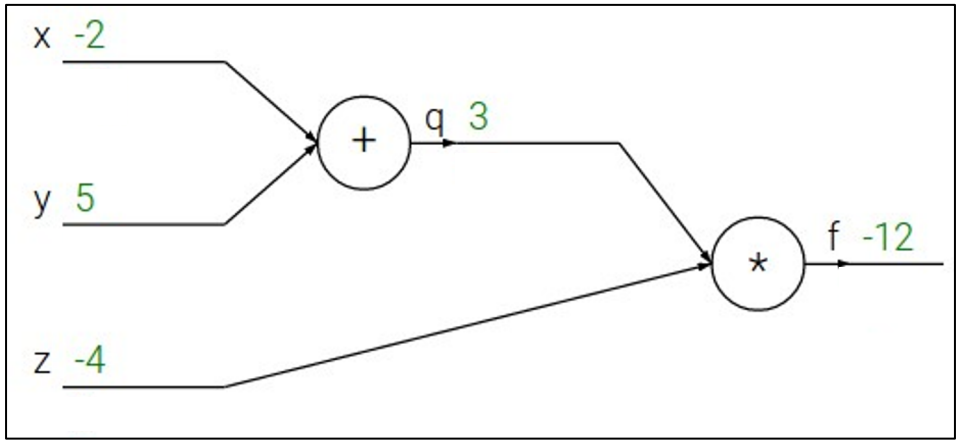


Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

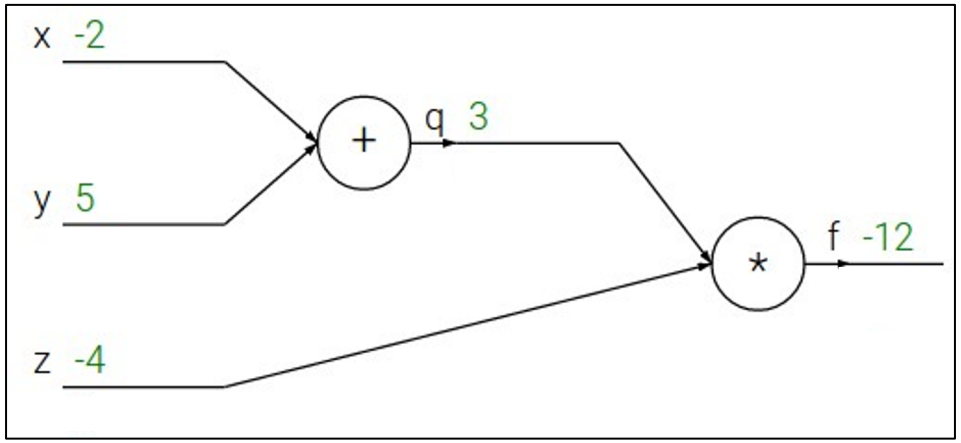
$$q = x + y \quad f = q \cdot z$$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

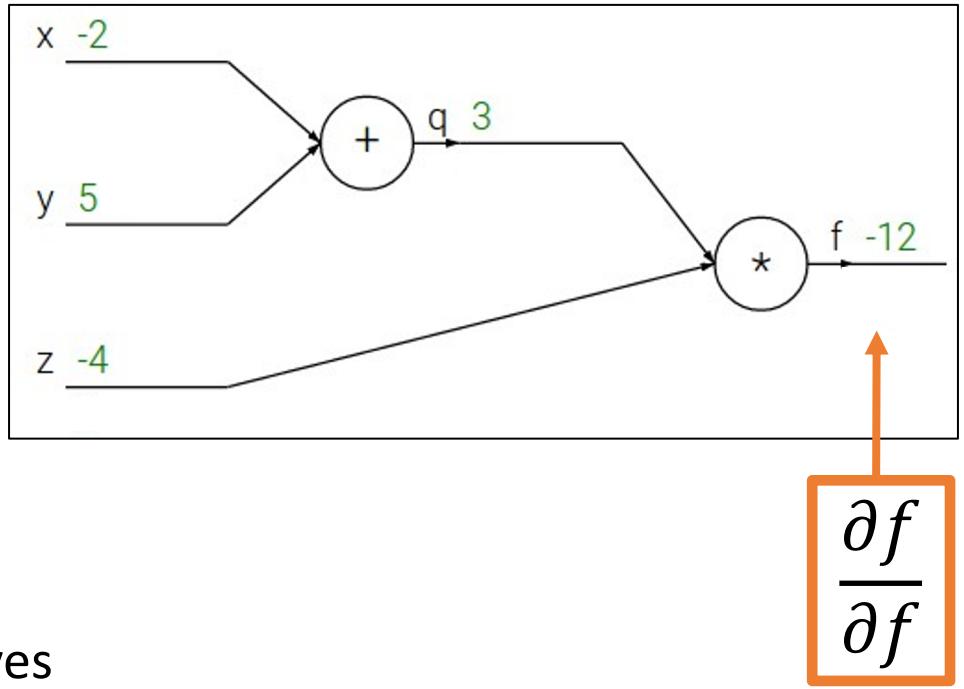
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

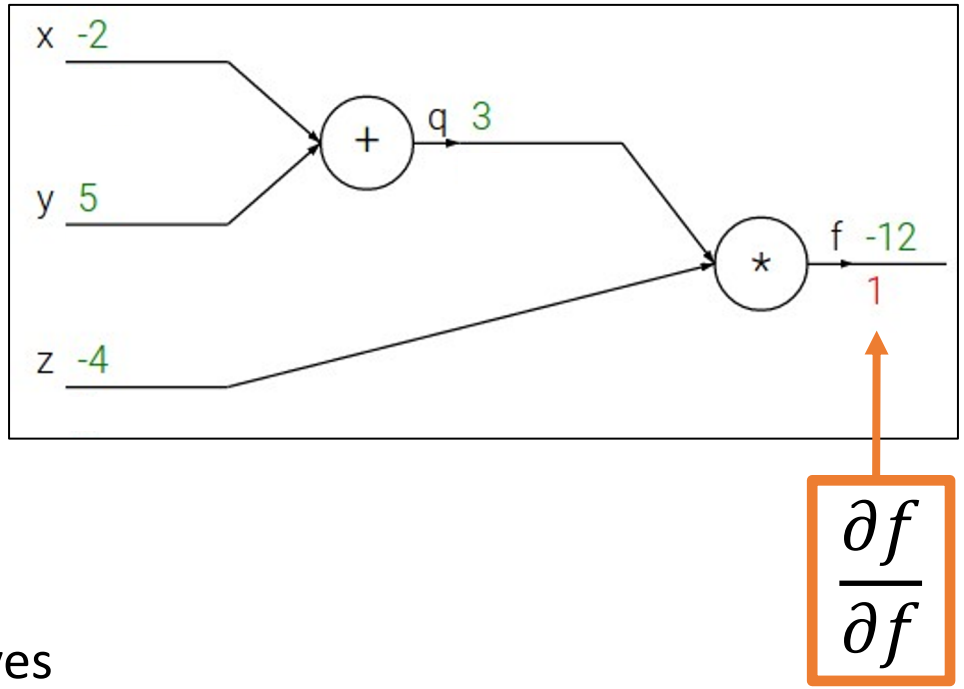
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

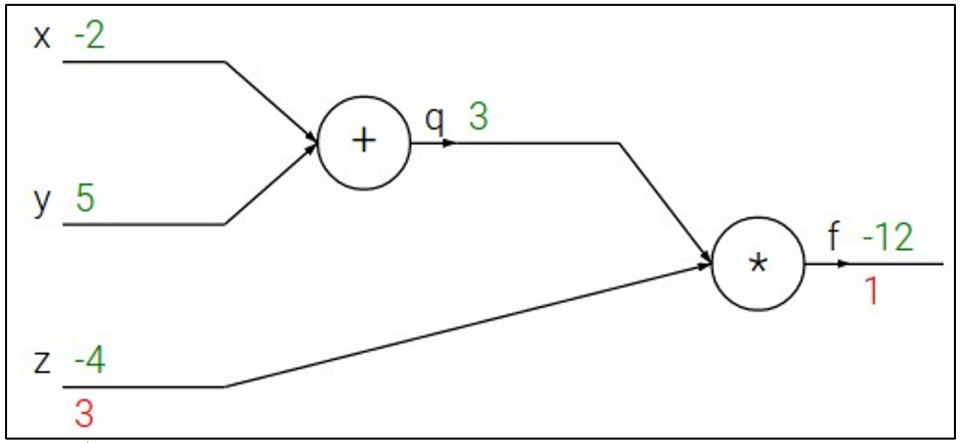
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

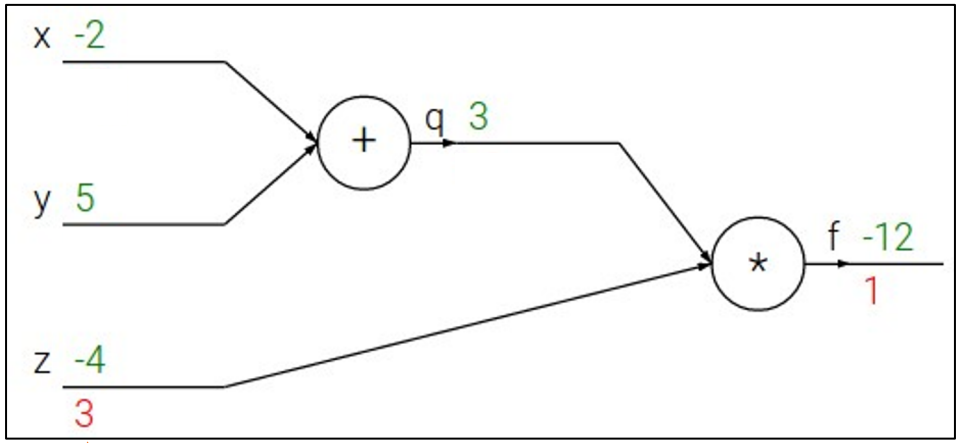
$\frac{\partial f}{\partial z}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

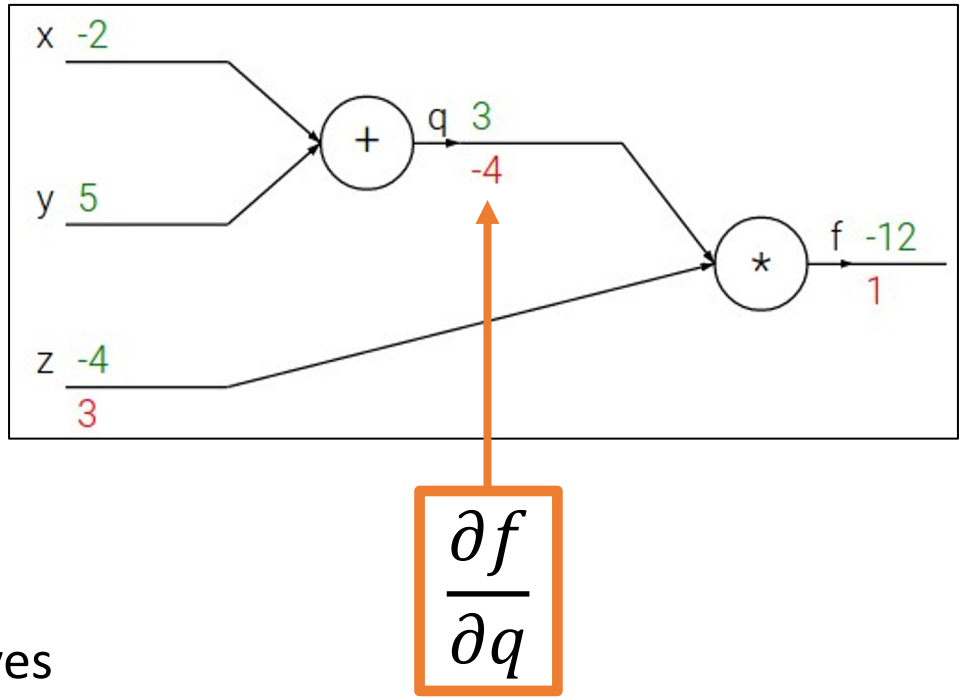
$$\frac{\partial f}{\partial z} = q$$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

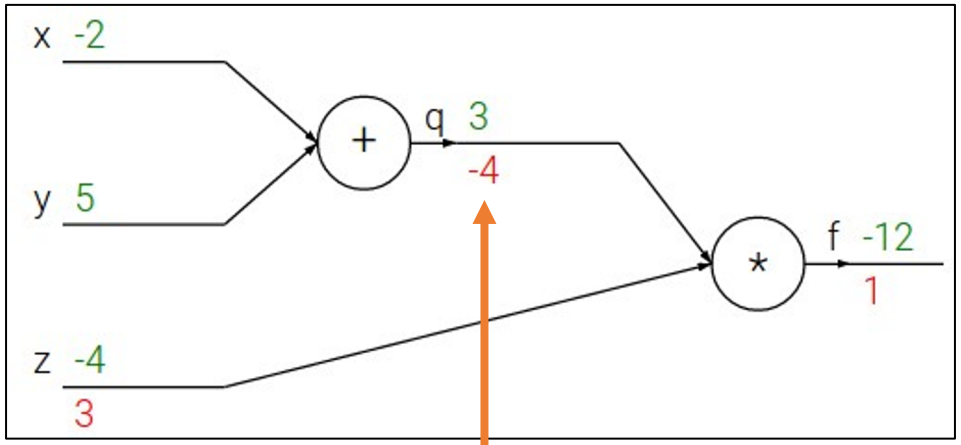
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

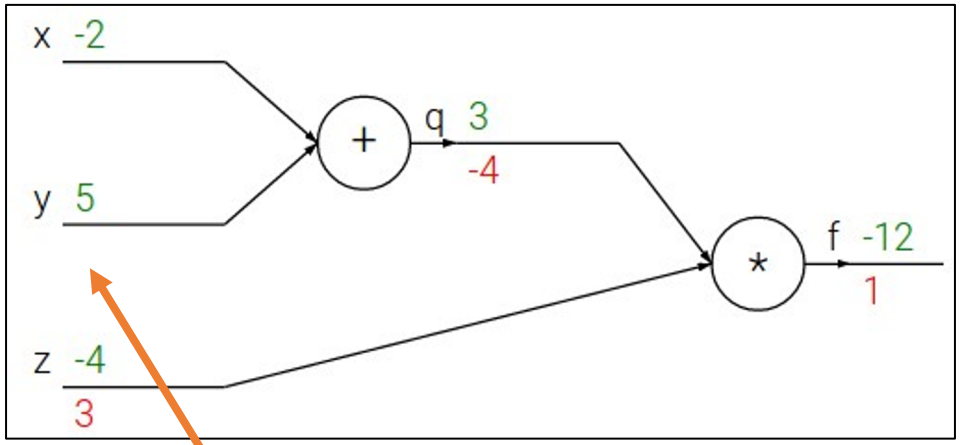
$$\frac{\partial f}{\partial q} = z$$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

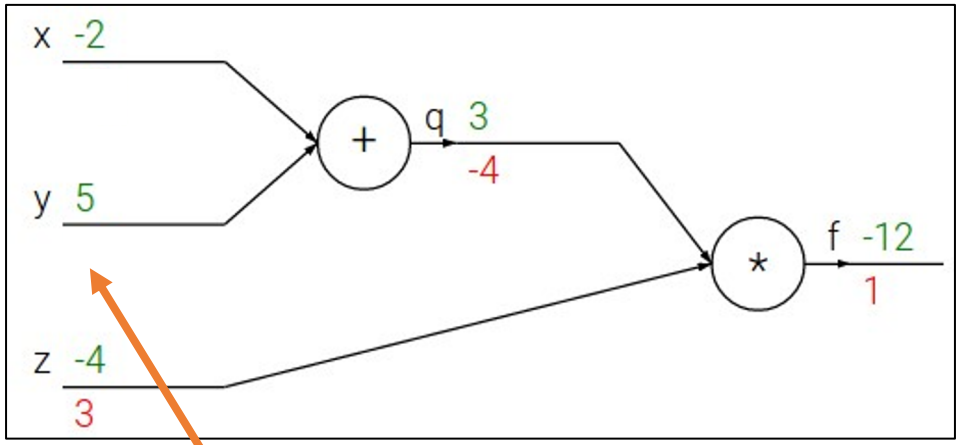
$\frac{\partial f}{\partial y}$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

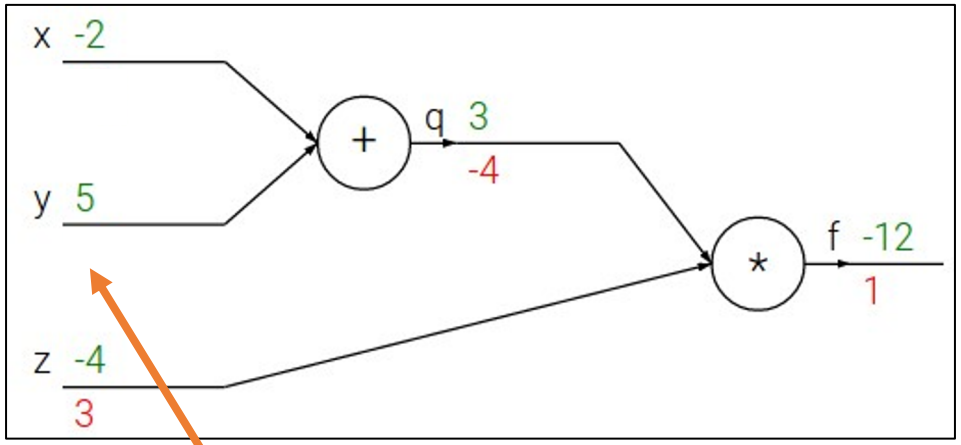
$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

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

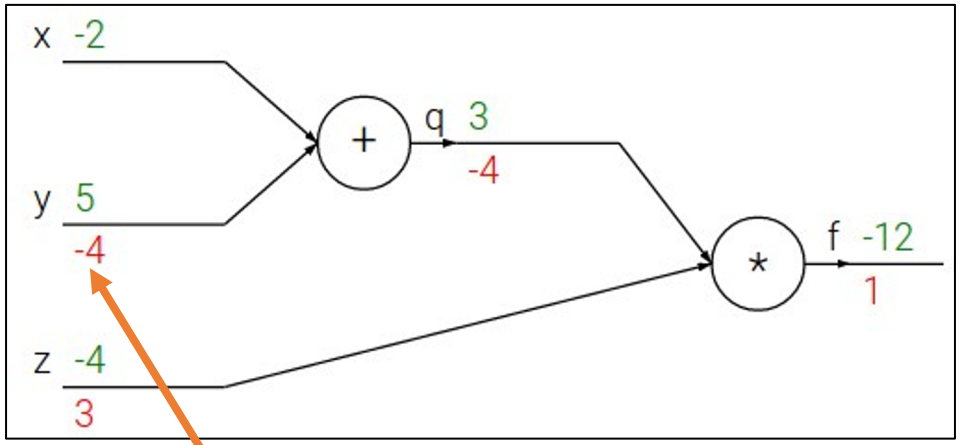
Downstream Gradient **Local Gradient** **Upstream Gradient**

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

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

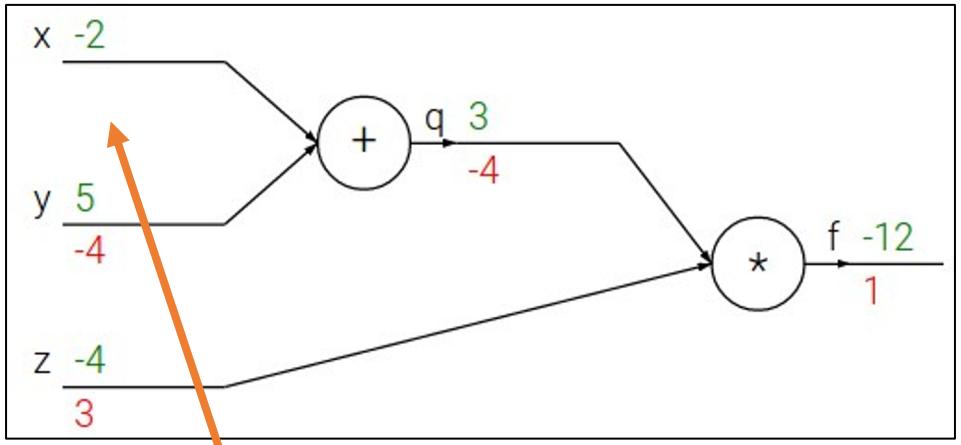
Downstream Gradient **Local Gradient** **Upstream Gradient**

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

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

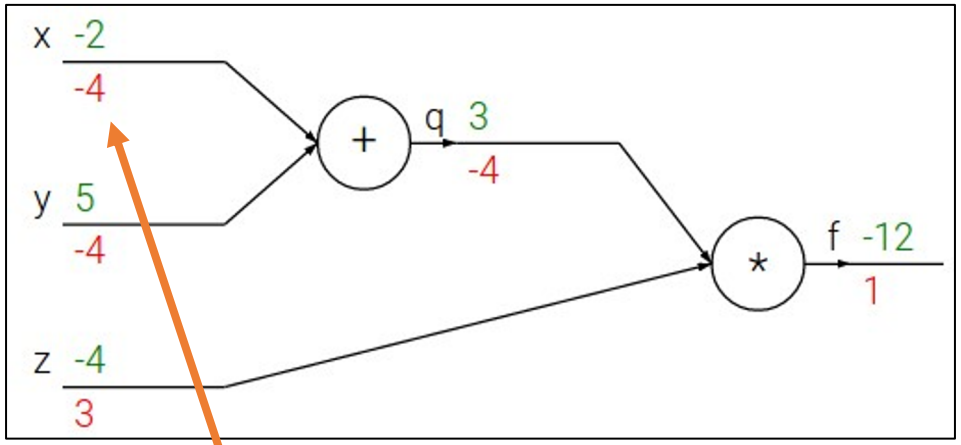
Downstream Gradient **Local Gradient** **Upstream Gradient**

Slide credit: D Fouhey & J Johnson

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

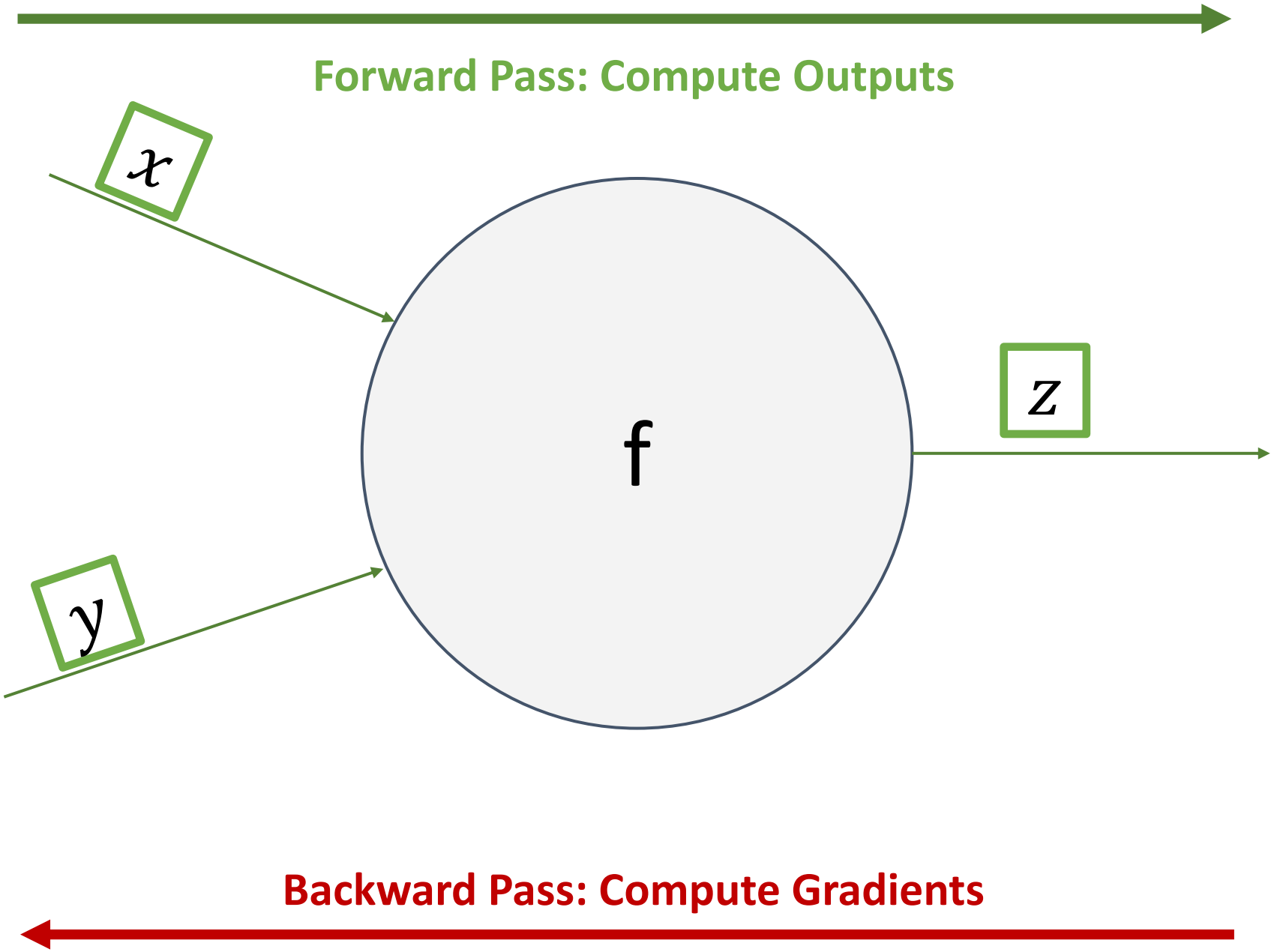
Chain Rule

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

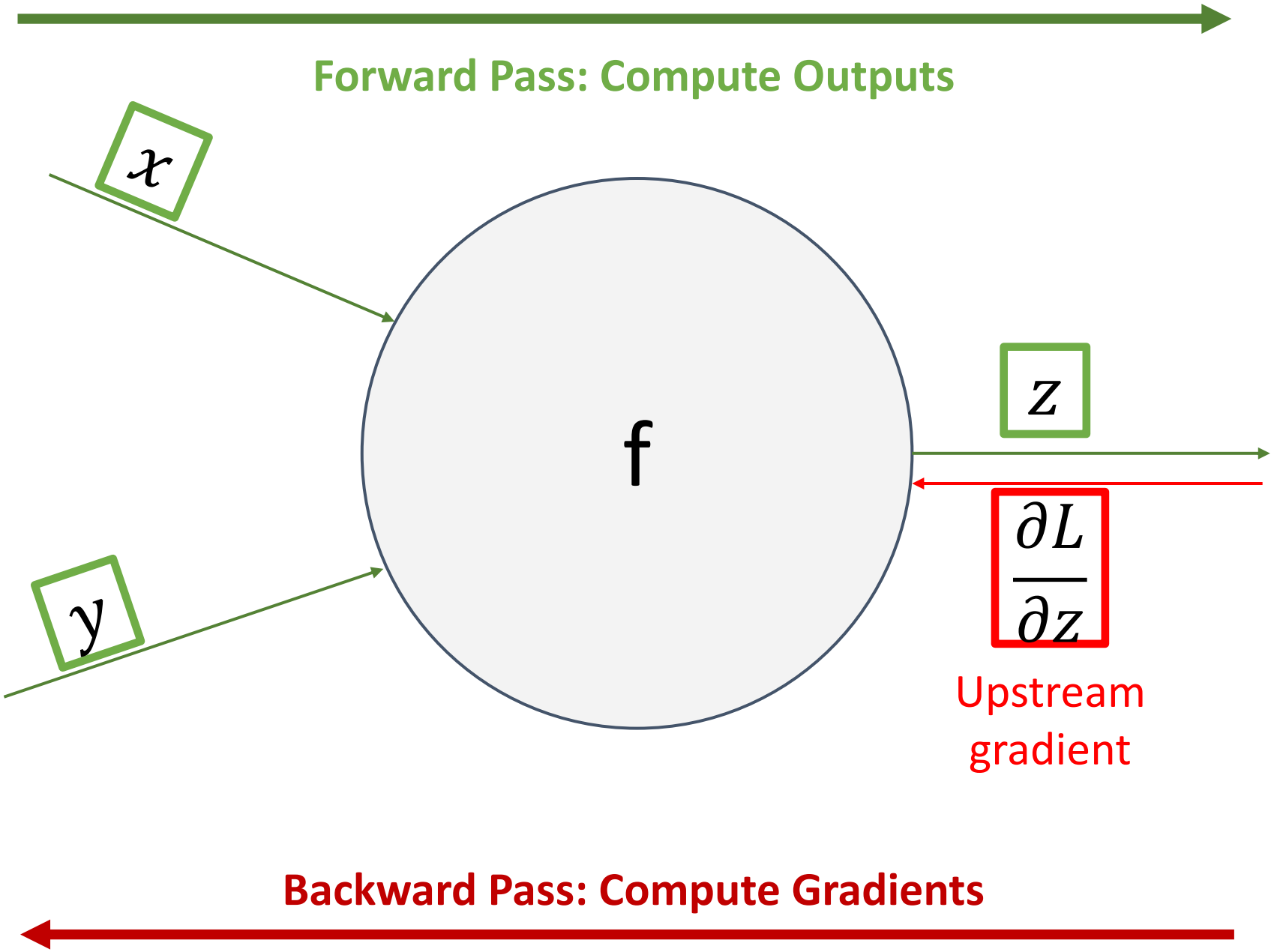
$$\frac{\partial q}{\partial x} = 1$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

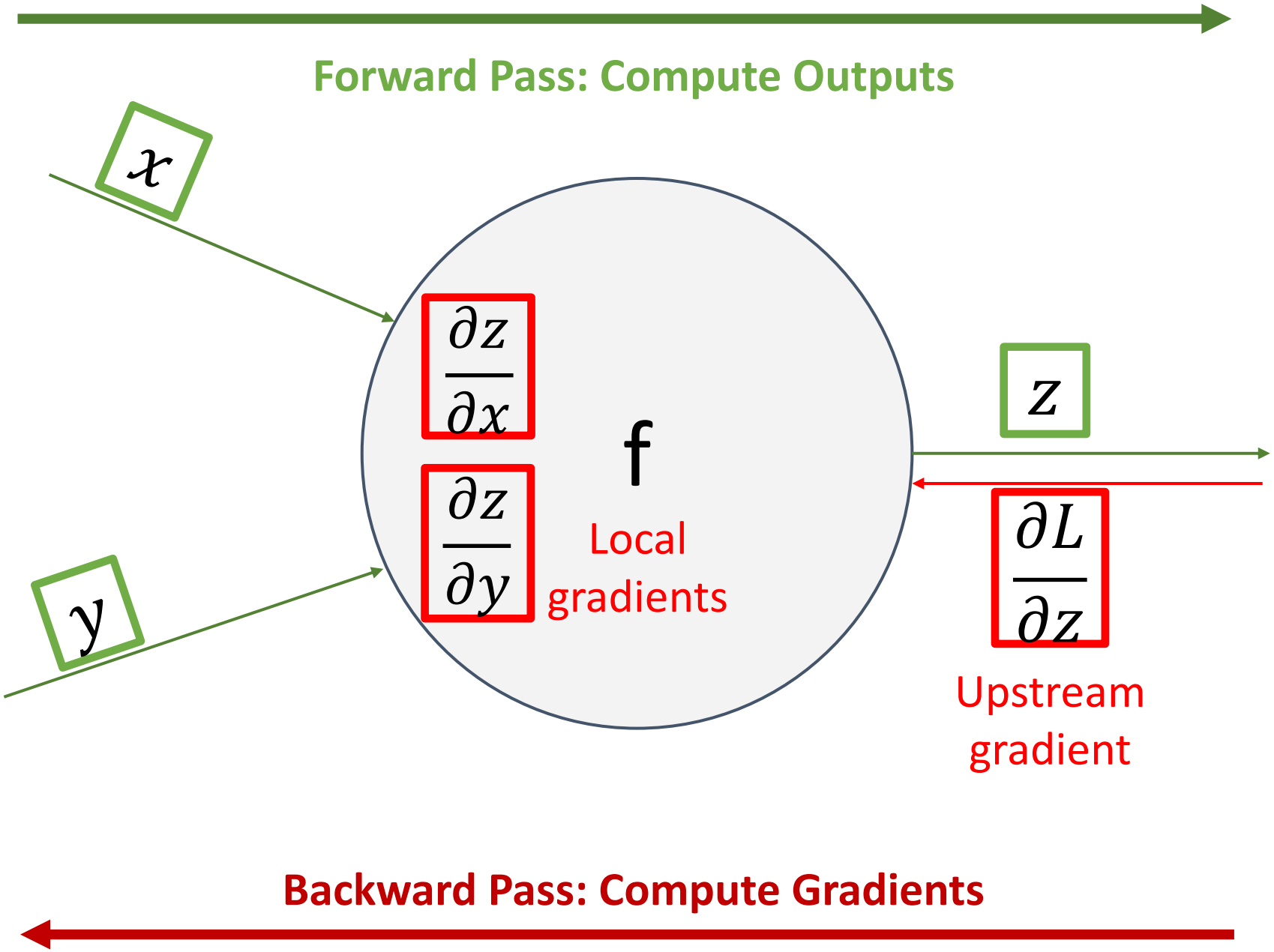
Slide credit: D Fouhey & J Johnson



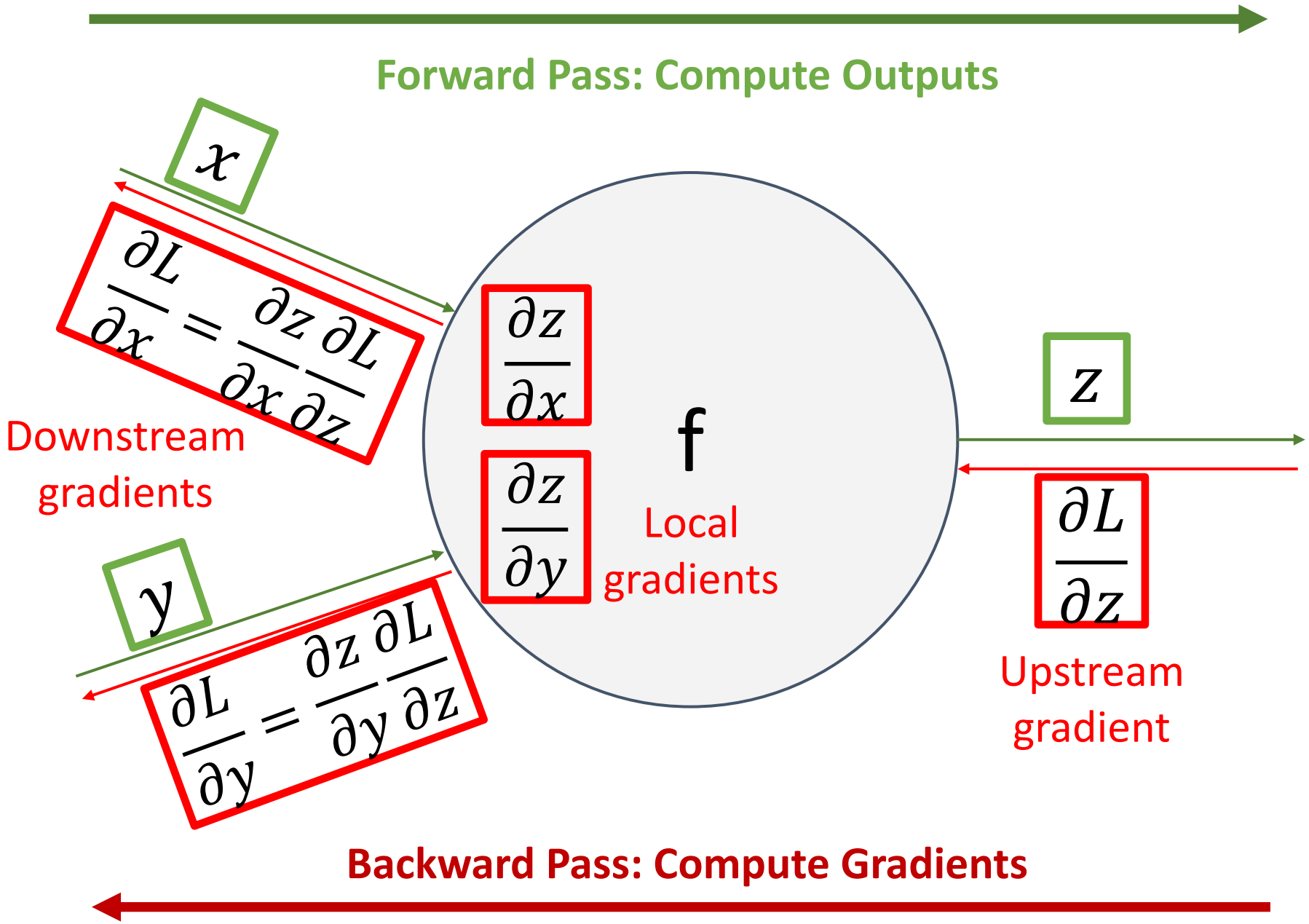
Slide credit: D Fouhey & J Johnson



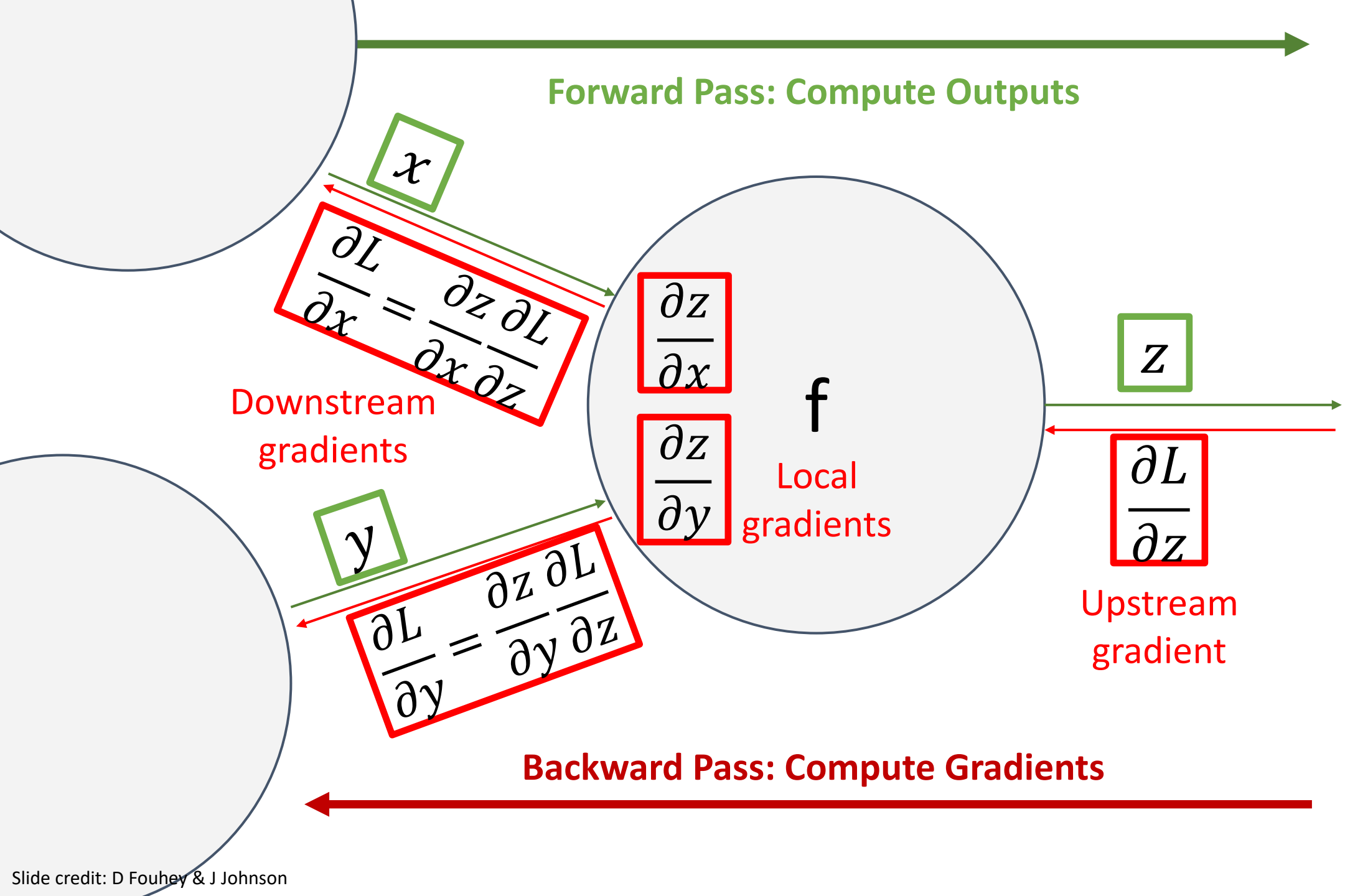
Slide credit: D Fouhey & J Johnson



Slide credit: D Fouhey & J Johnson

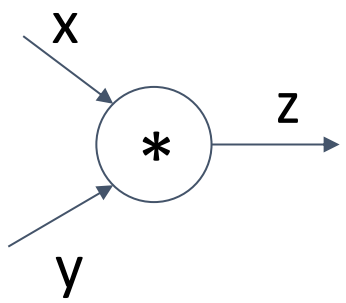


Slide credit: D Fouhey & J Johnson



Slide credit: D Fouhey & J Johnson

Example: PyTorch Autograd Functions with Modular API



(x,y,z are scalars)

```
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 stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Slide credit: D Fouhey & J Johnson

Example: PyTorch operators

pytorch / pytorch

Watch 1,221 | Unstar 26,770 | Fork 6,340

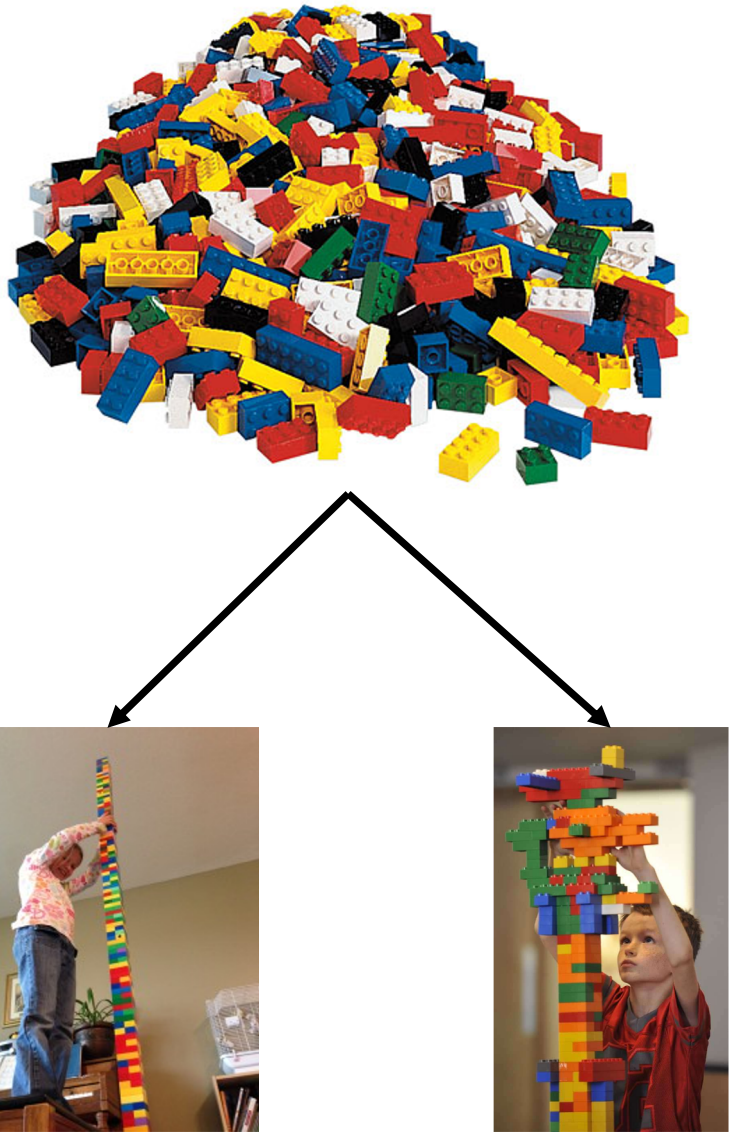
Code | Issues 2,286 | Pull requests 561 | Projects 4 | Wiki | Insights

Tree: 517c7c9861 ▾ pytorch / aten / src / THNN / generic / | Create new file | Upload files | Find file | History

ezyang and facebook-github-bot Canonicalize all includes in PyTorch. (#14849) ··· Latest commit 517c7c9 on Dec 8, 2018

..

AbsCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
BCECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ClassNLLCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Col2Im.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ELU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
FeatureLPPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
GatedLinearUnit.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
HardTanh.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Im2Col.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
IndexLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LeakyReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LogSigmoid.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MSECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiLabelMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
RReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago



Slide credit: D Fouhey & J Johnson, E Learned-Miller

Building NNs using PyTorch with AutoGrad

```python
import torch.nn as nn

class MLP(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim):
        super(MLP, self).__init__()

        # hidden layer
        self.h_mult = nn.Linear(input_dim, hidden_dim)
        self.h_relu = nn.ReLU()

        # output layer
        self.out = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h = self.h_mult(x)
        h = self.h_relu(h)
        y = self.out(h)
        return y
```

So far: Backprop with scalars

What about vector-valued functions?

Slide credit: D Fouhey & J Johnson

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?

Slide credit: D Fouhey & J Johnson

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?

Slide credit: D Fouhey & J Johnson

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**:

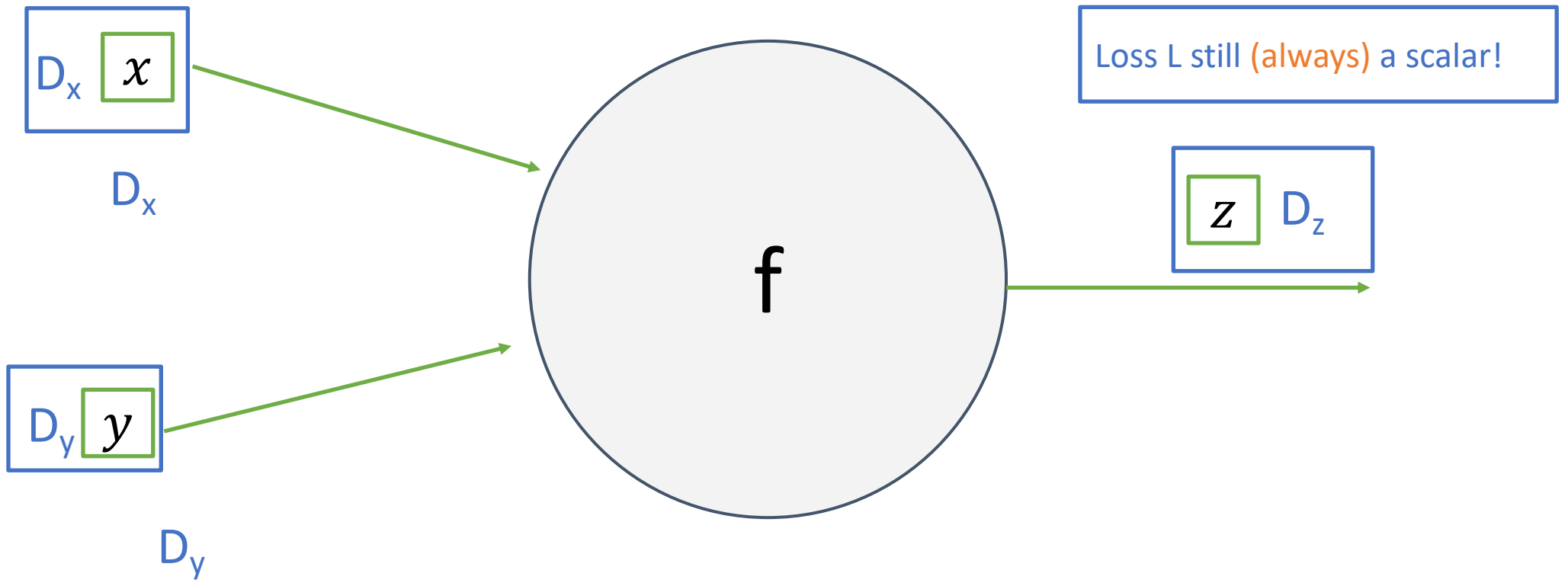
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$$

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

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

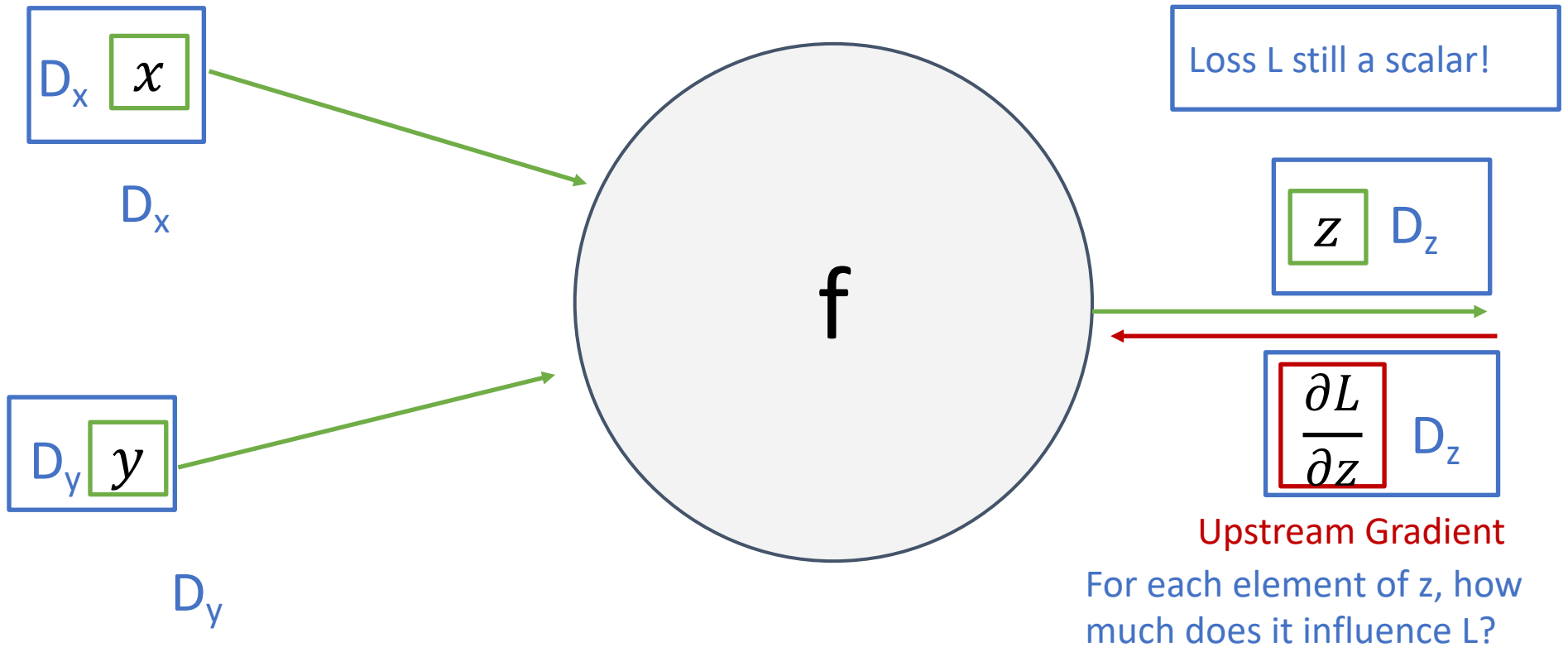
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



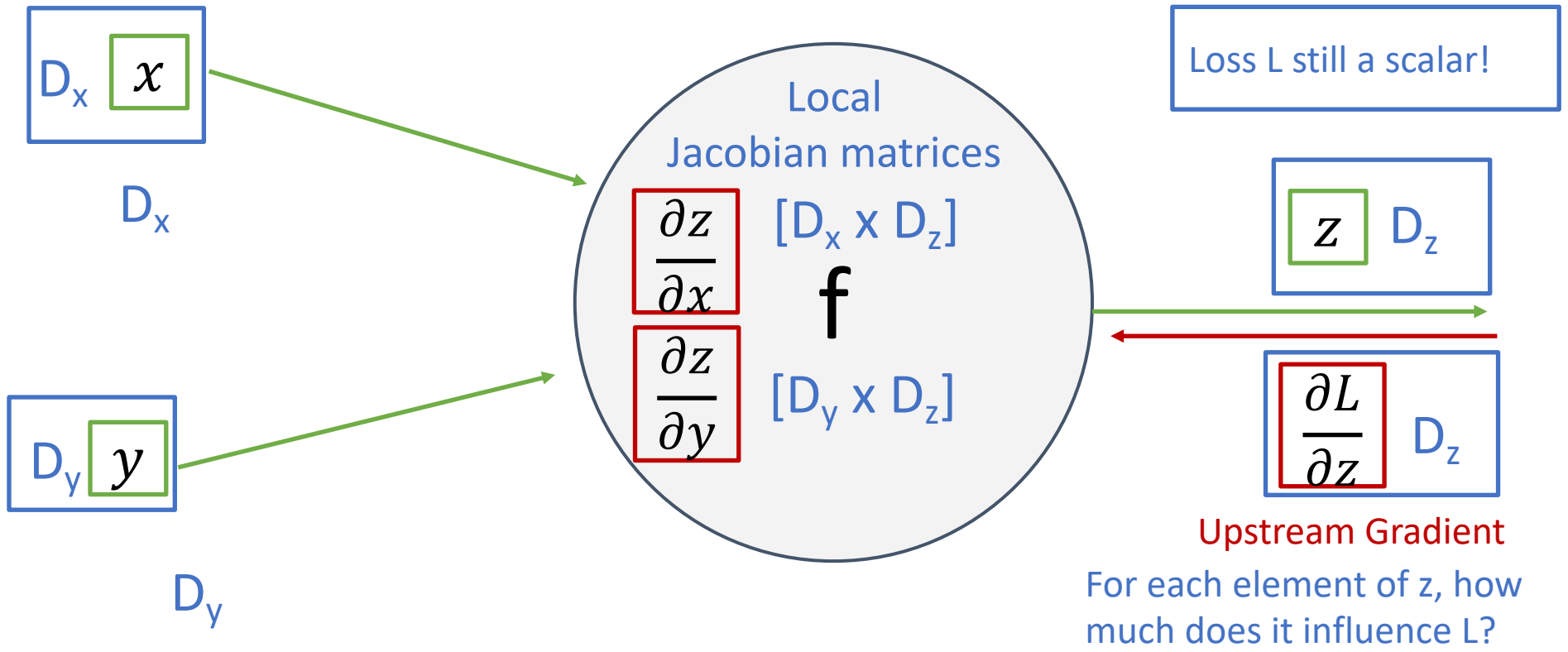
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



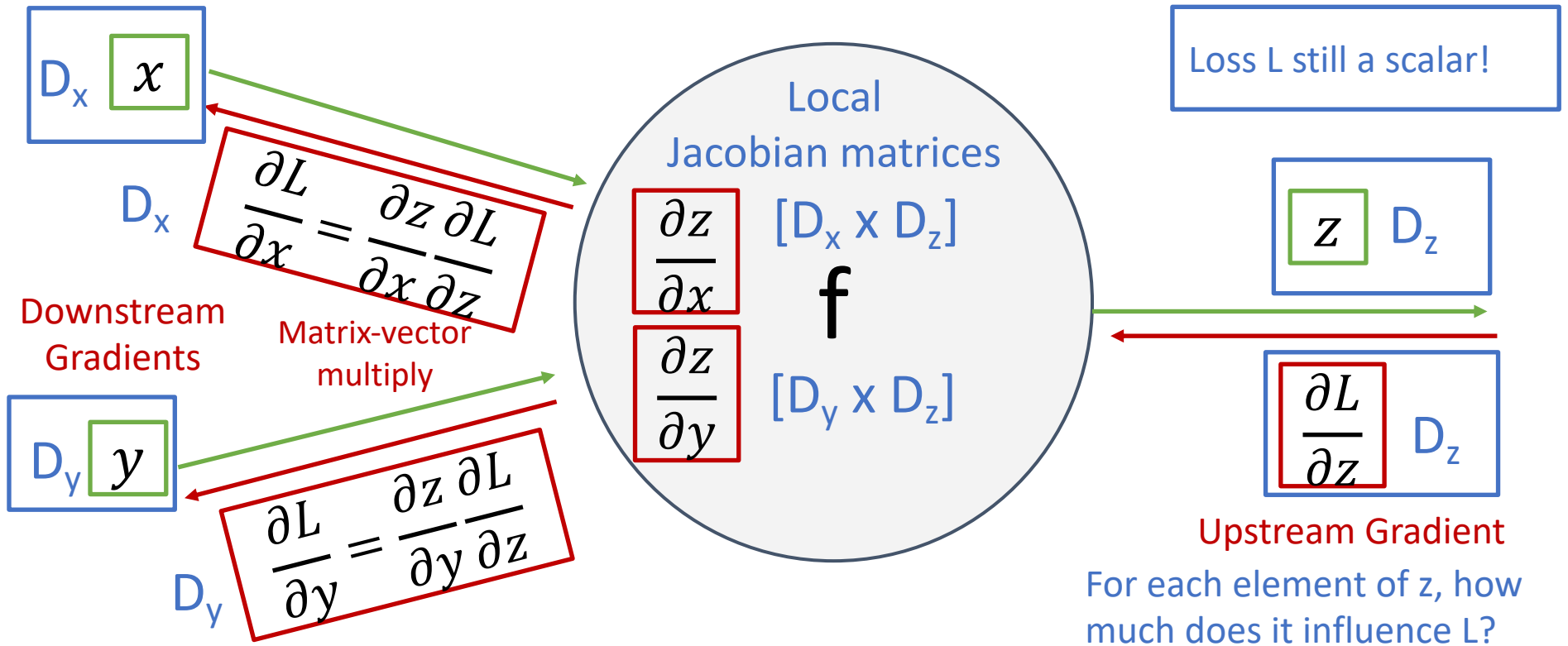
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



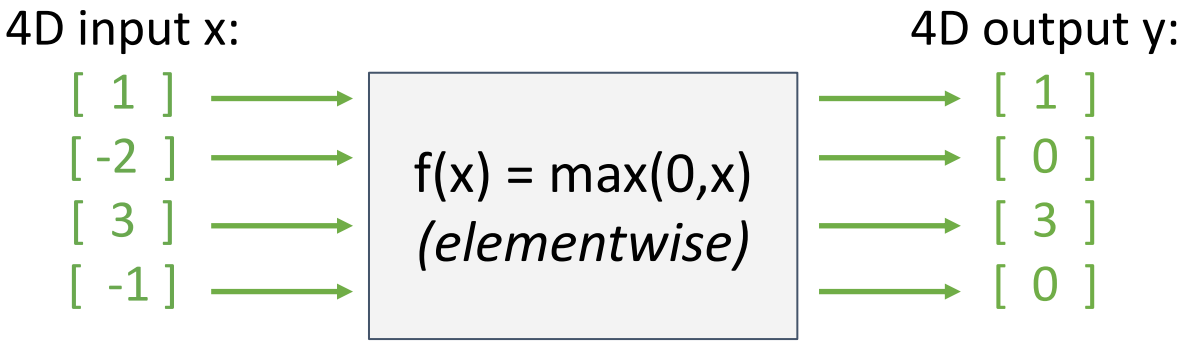
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



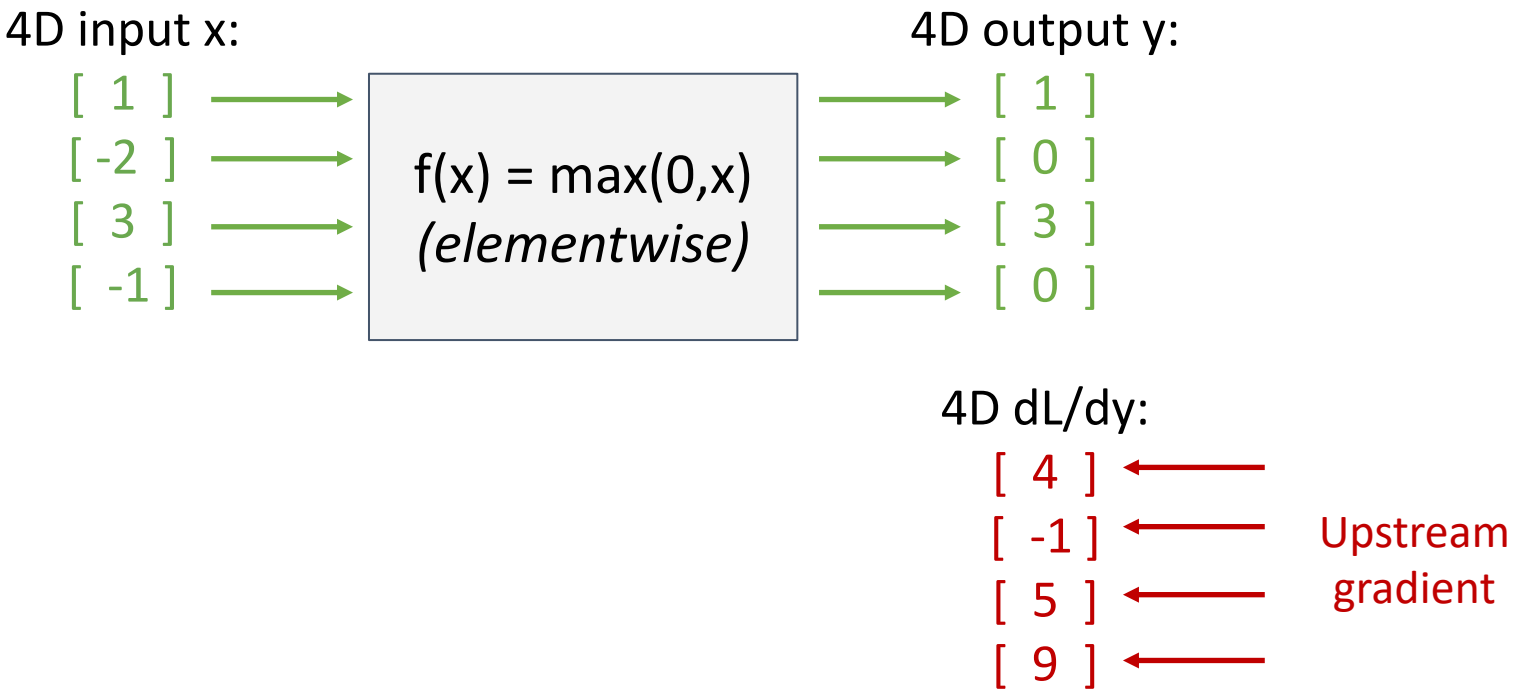
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



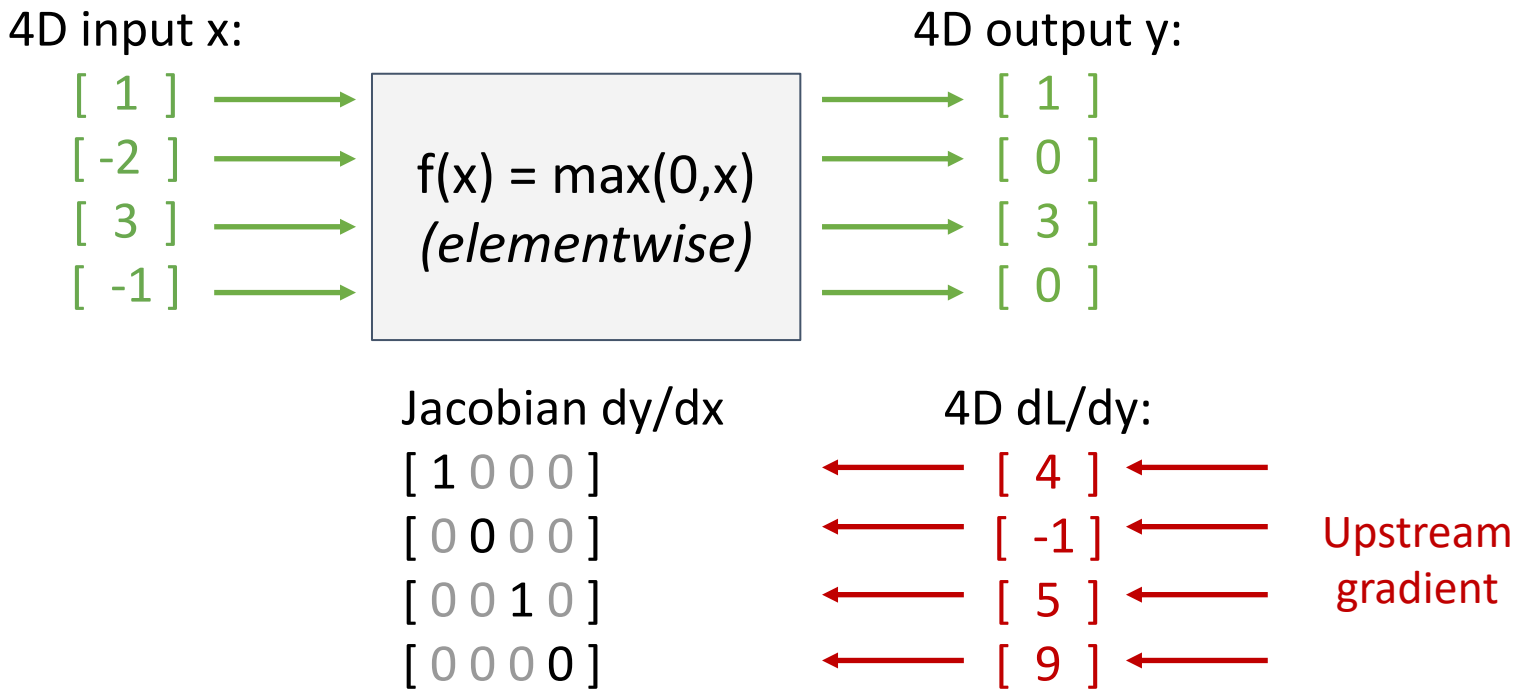
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



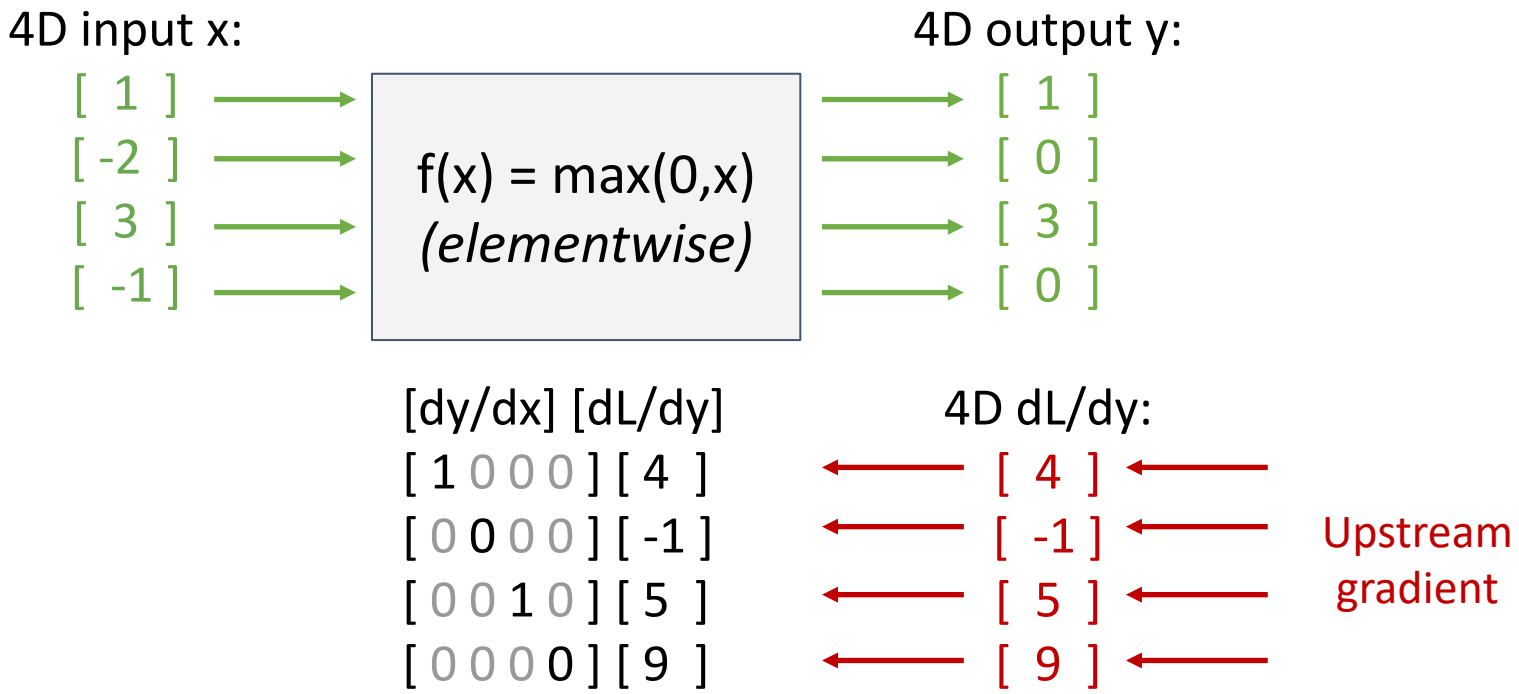
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



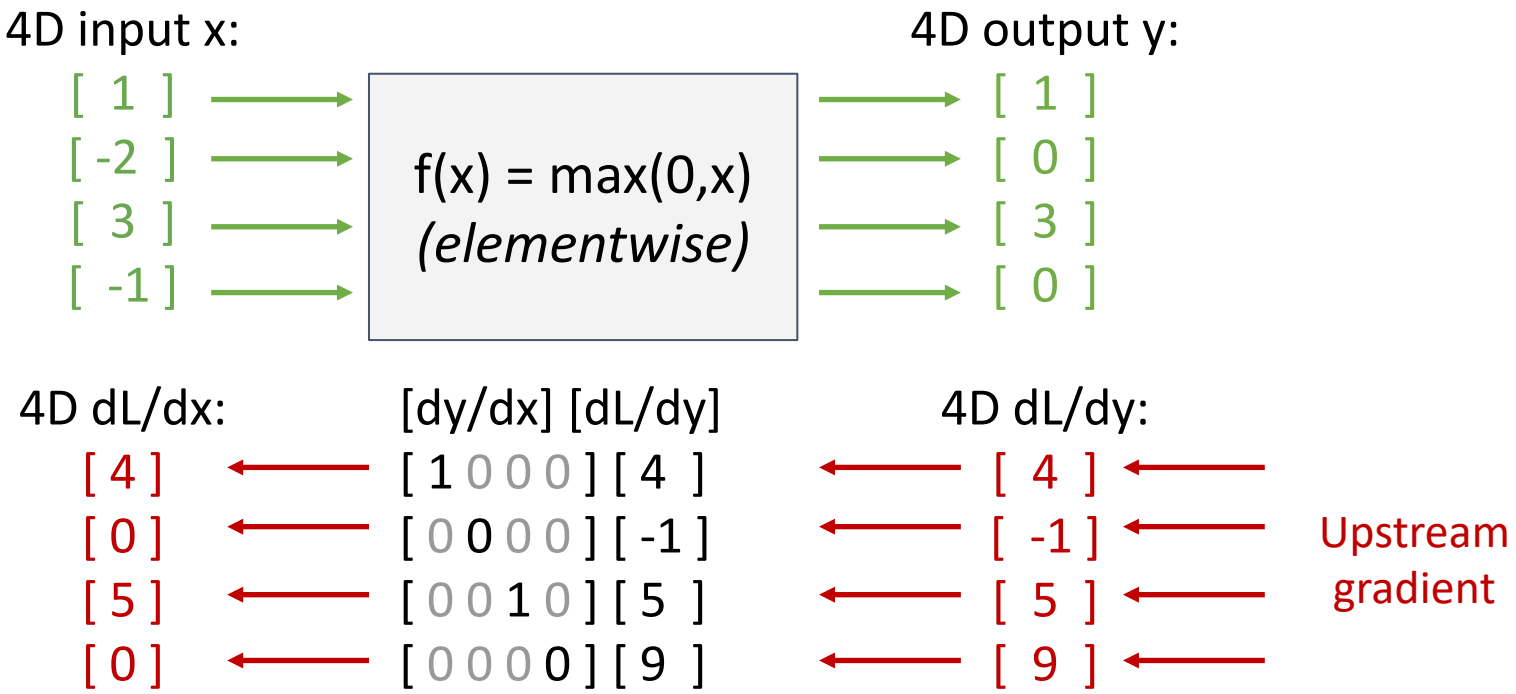
Slide credit: D Fouhey & J Johnson

Backprop with Vectors



Slide credit: D Fouhey & J Johnson

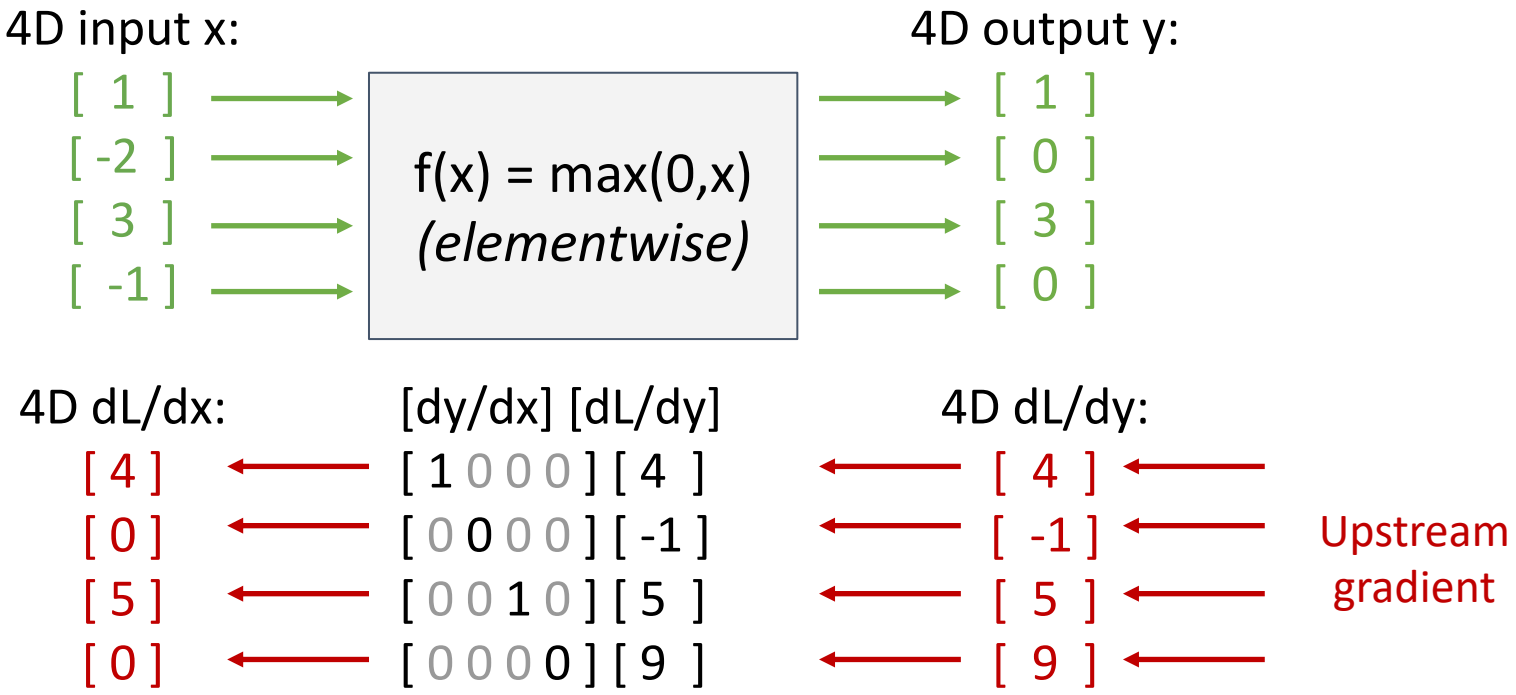
Backprop with Vectors



Slide credit: D Fouhey & J Johnson

Backprop with Vectors

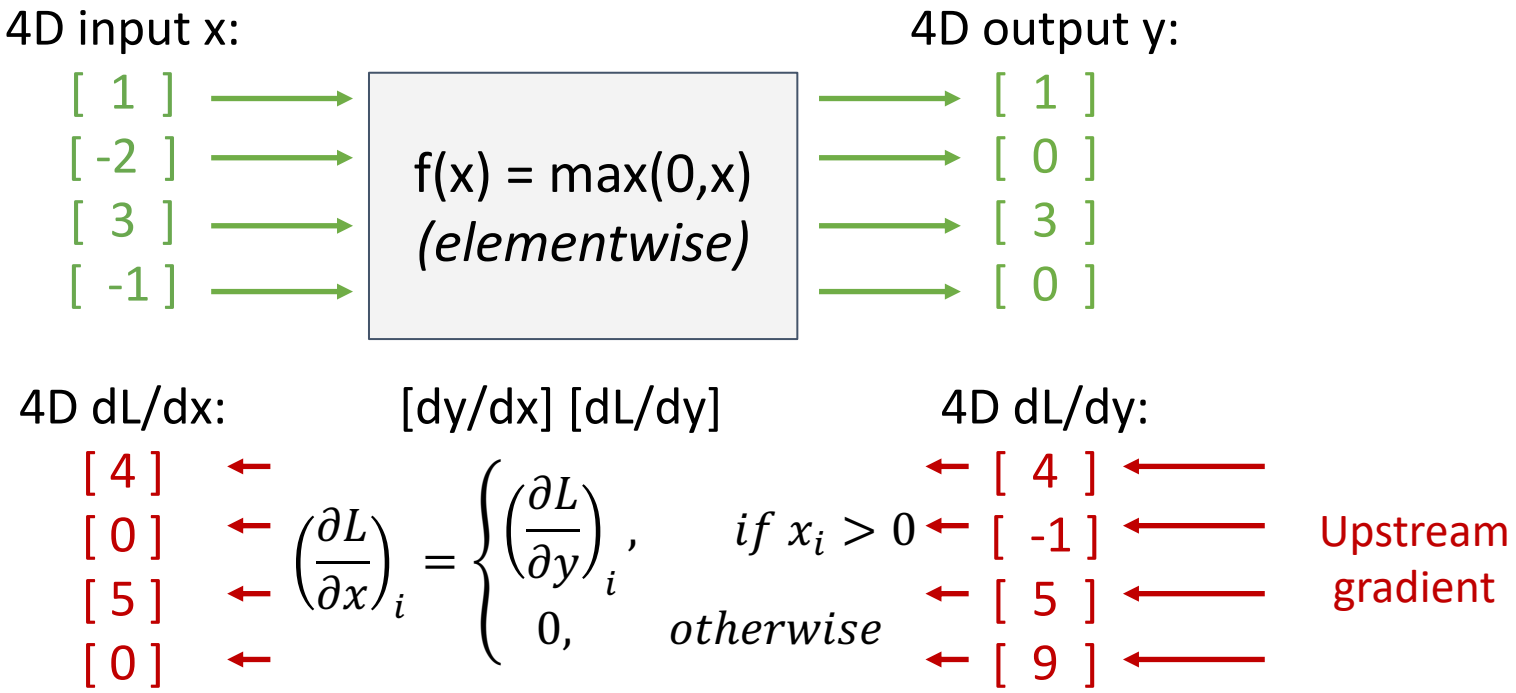
Jacobian is **sparse**: off-diagonal entries all zero! Never **explicitly** form Jacobian; instead use **implicit** multiplication



Slide credit: D Fouhey & J Johnson

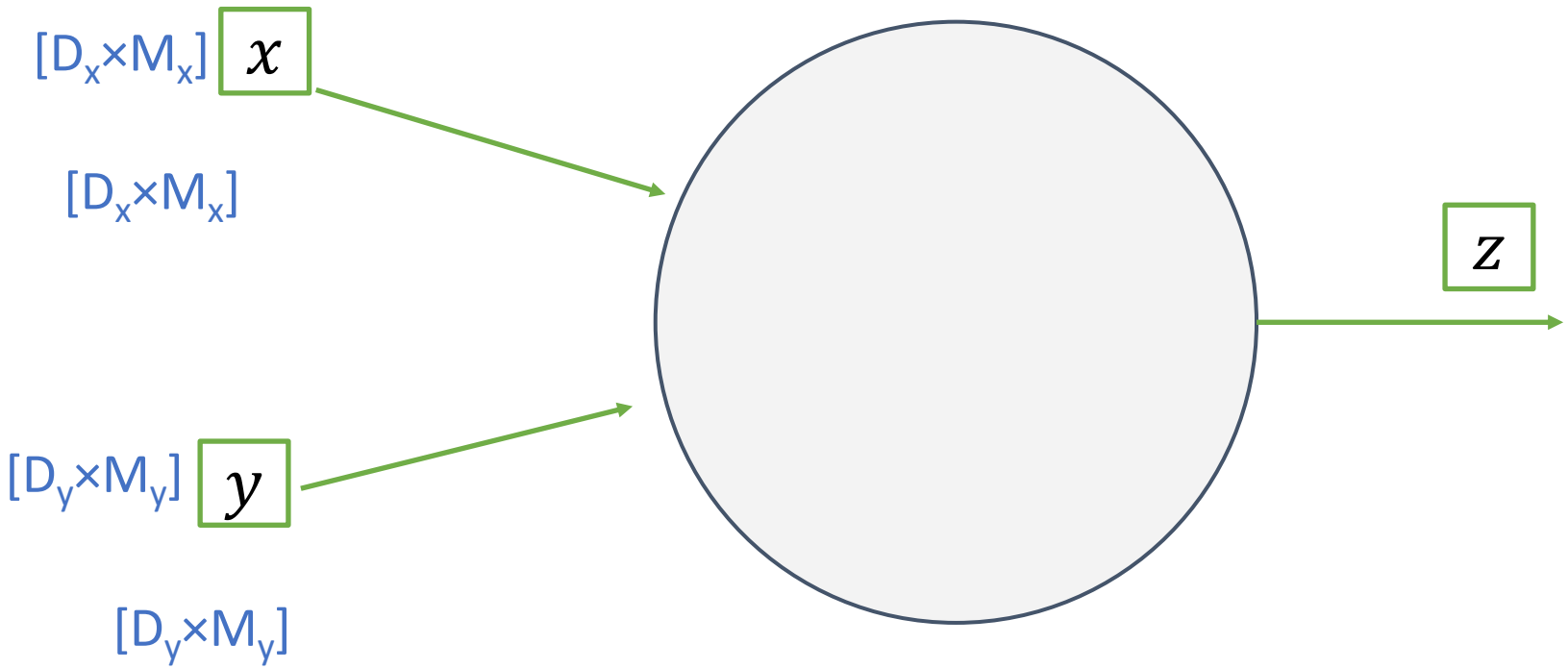
Backprop with Vectors

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



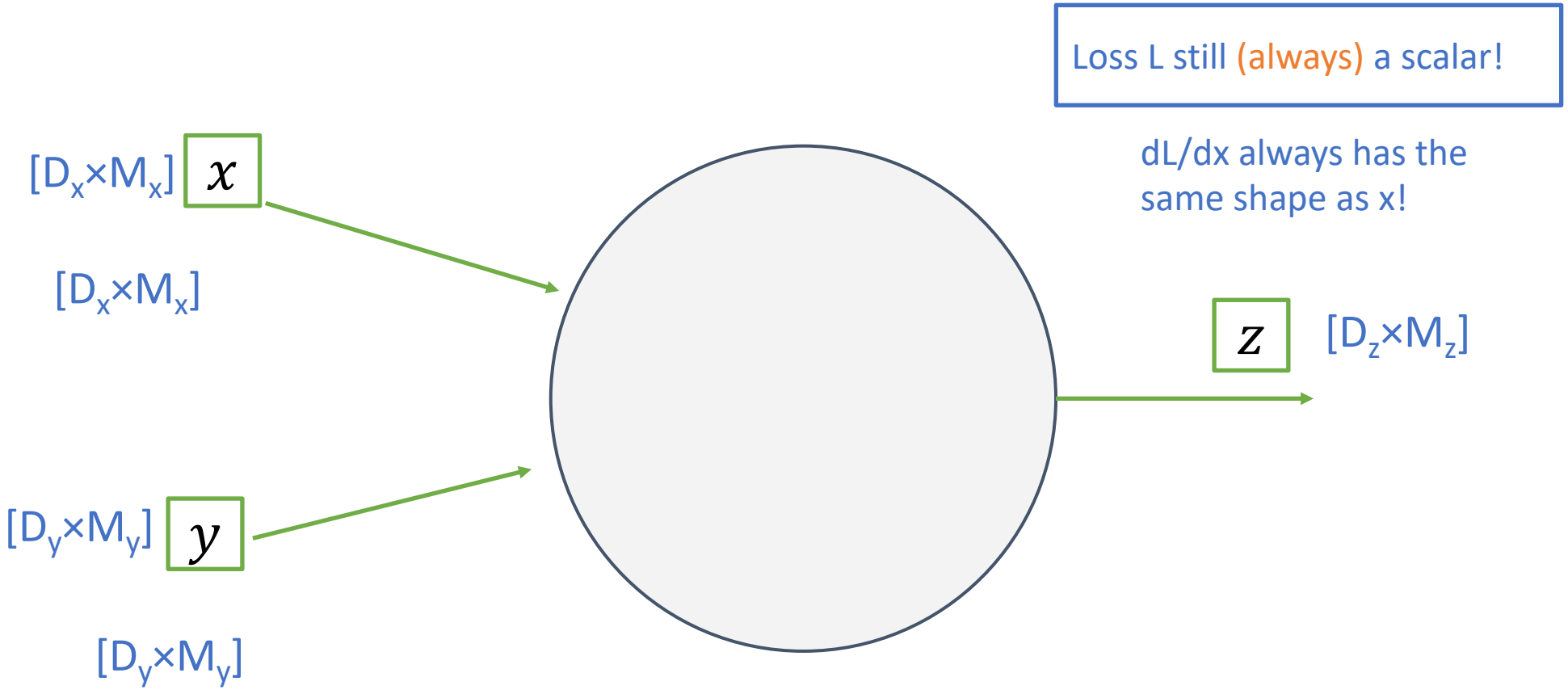
Slide credit: D Fouhey & J Johnson

Backprop with Matrices (or Tensors):



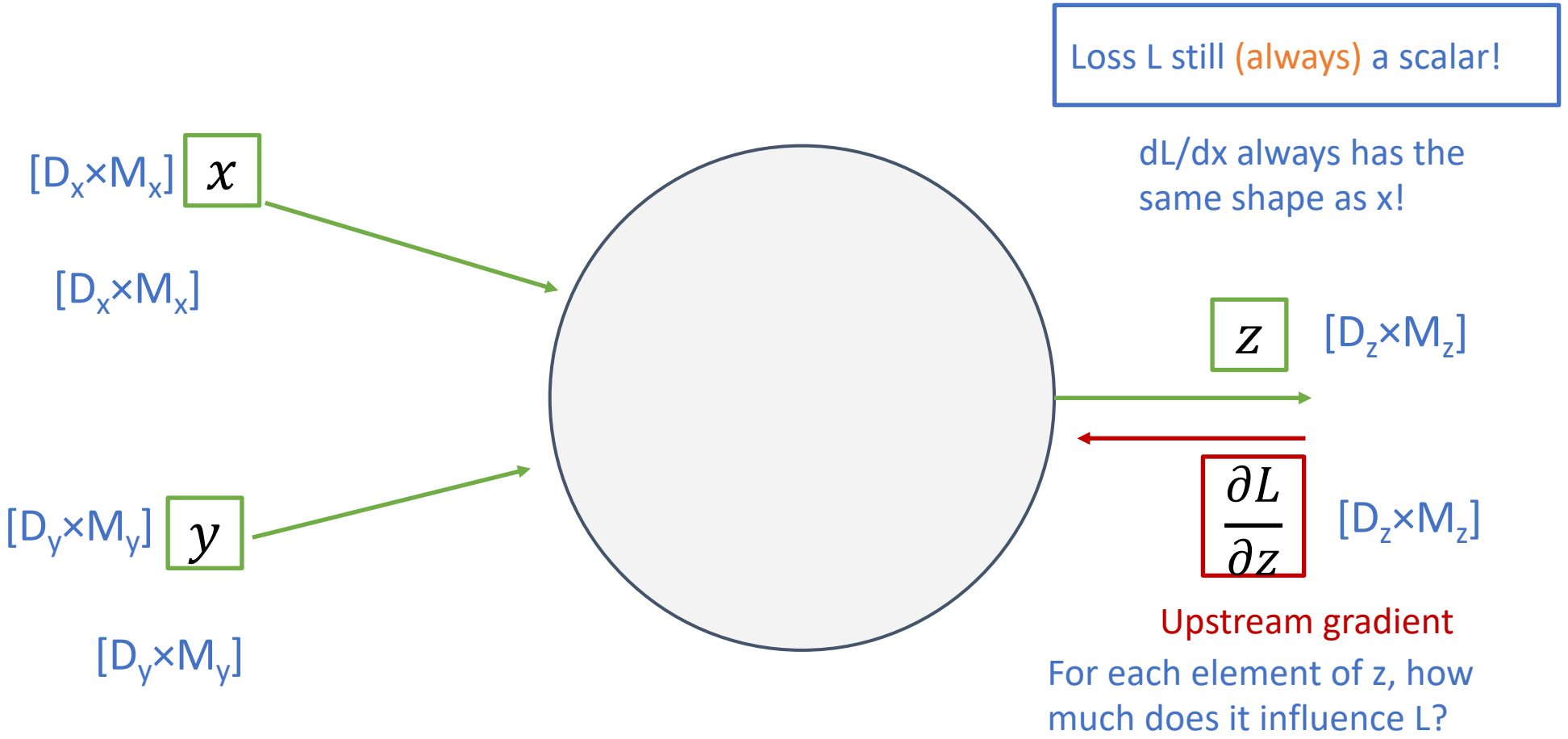
Slide credit: D Fouhey & J Johnson

Backprop with Matrices (or Tensors):



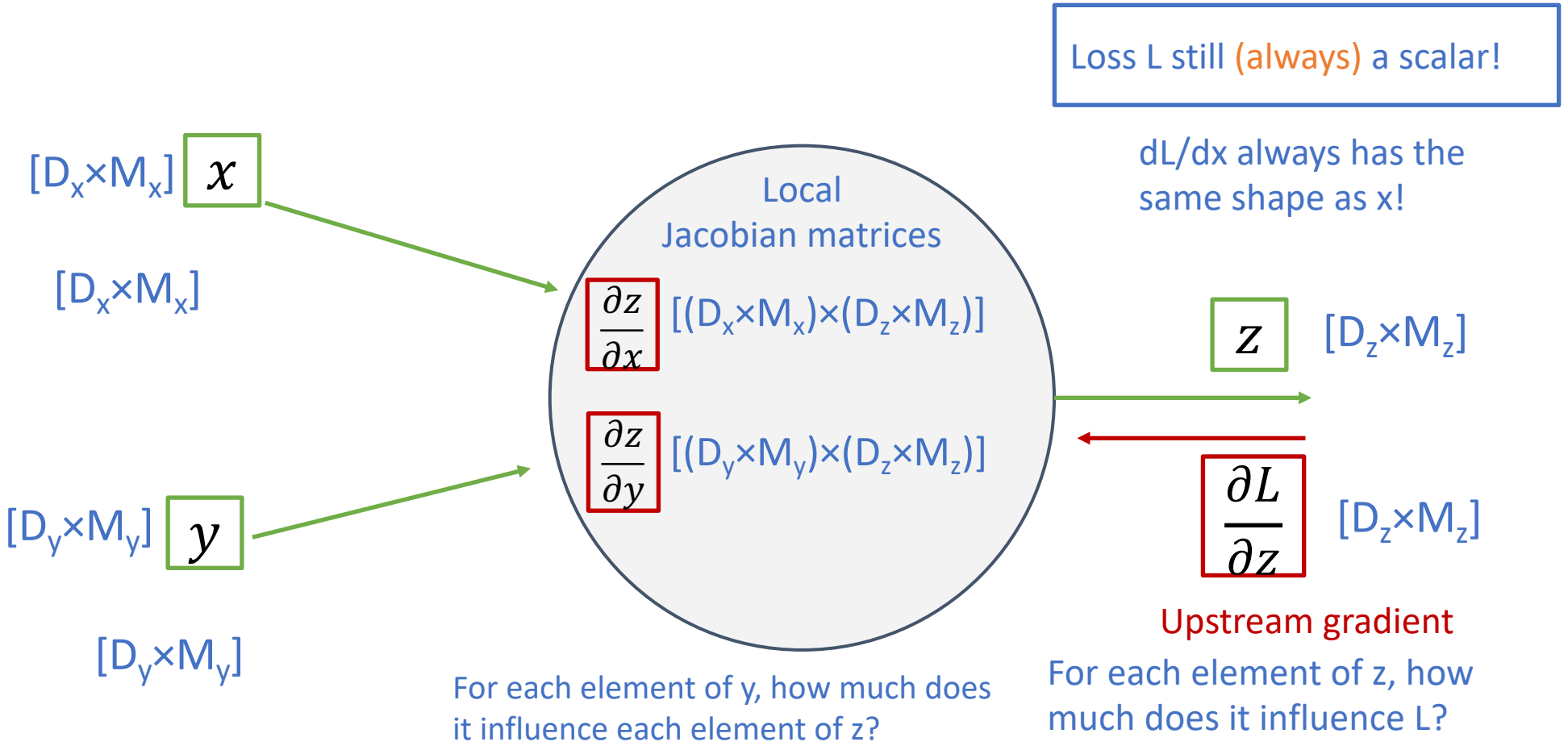
Slide credit: D Fouhey & J Johnson

Backprop with Matrices (or Tensors):



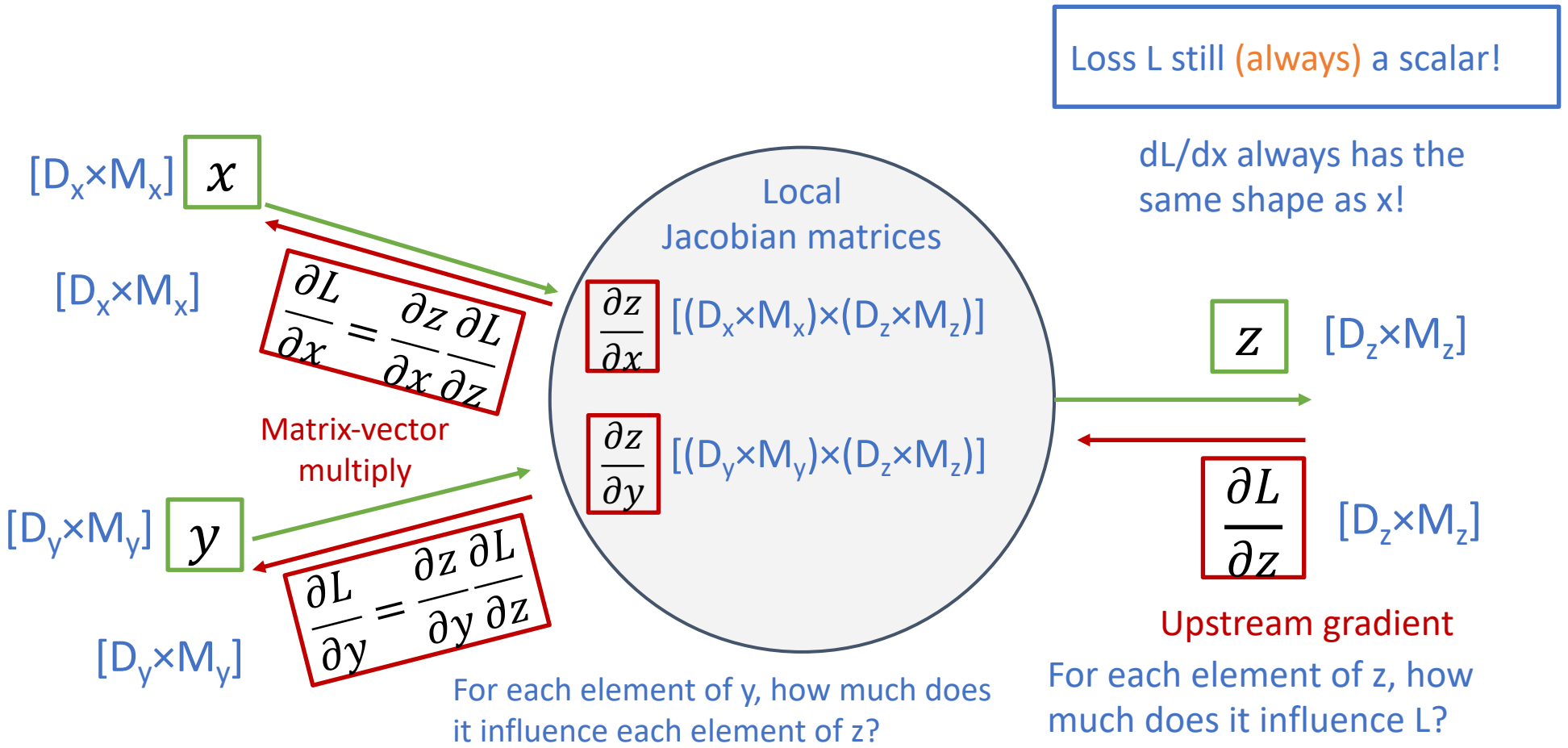
Slide credit: D Fouhey & J Johnson

Backprop with Matrices (or Tensors):



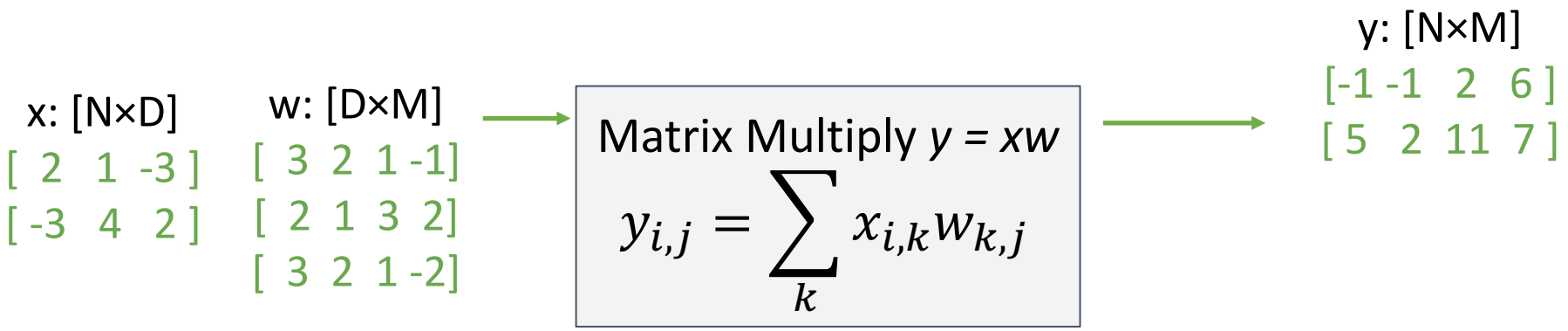
Slide credit: D Fouhey & J Johnson

Backprop with Matrices (or Tensors):



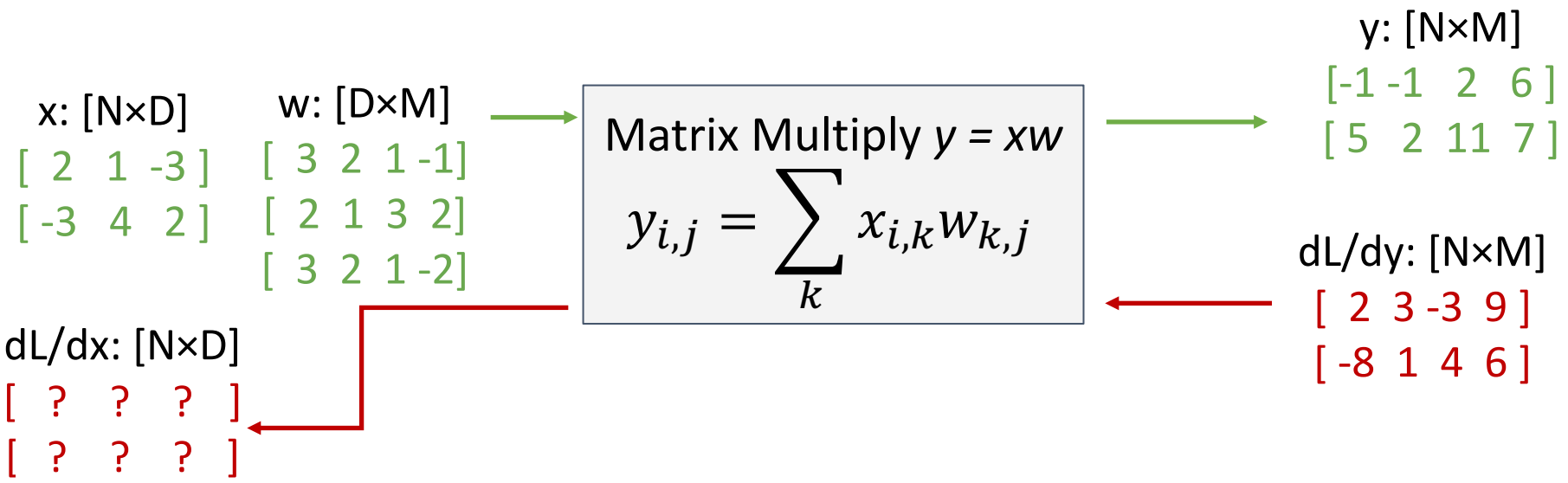
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



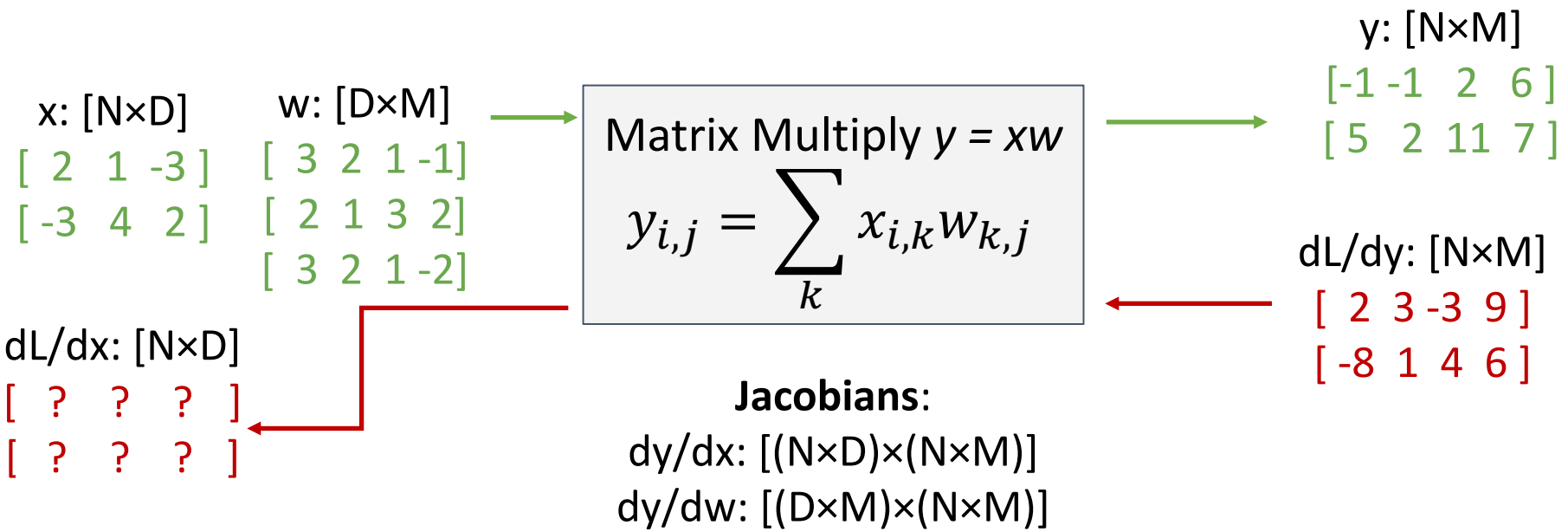
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



Slide credit: D Fouhey & J Johnson

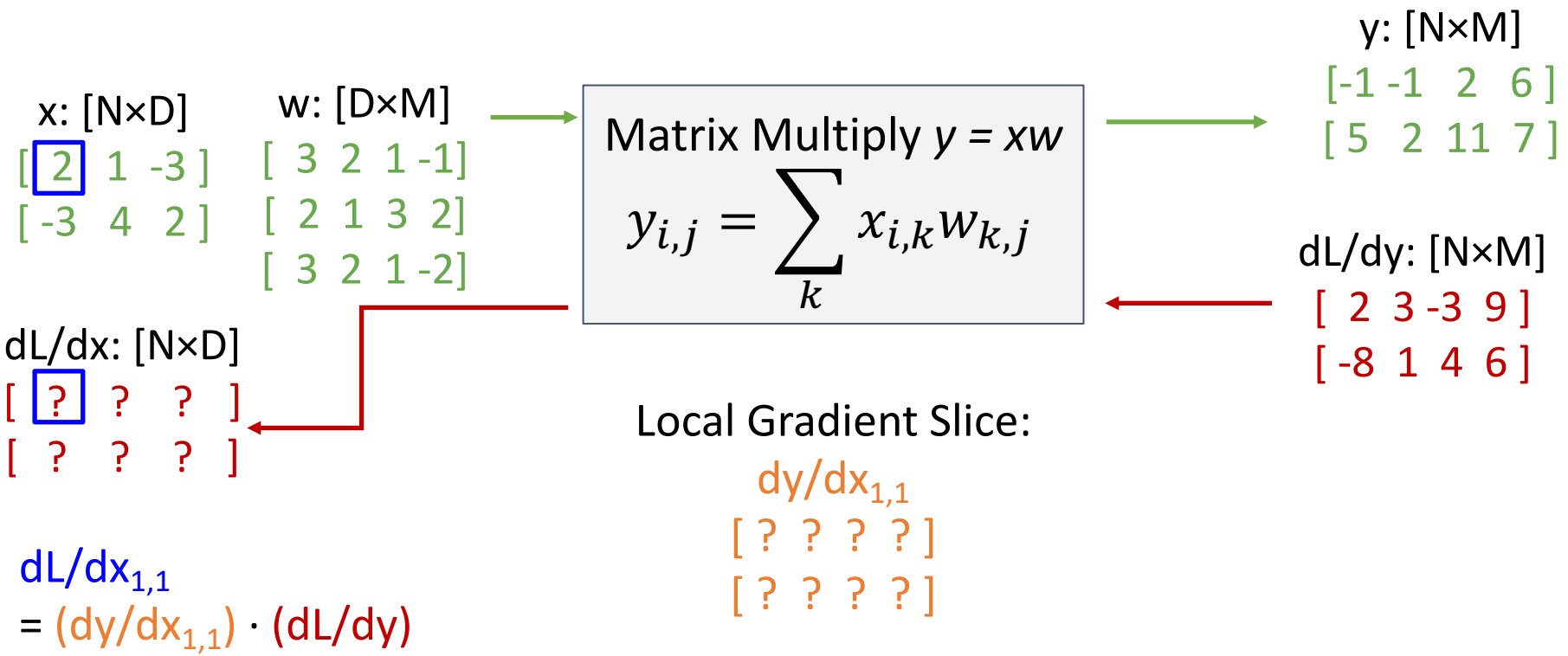
Example: Matrix Multiplication Operator



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

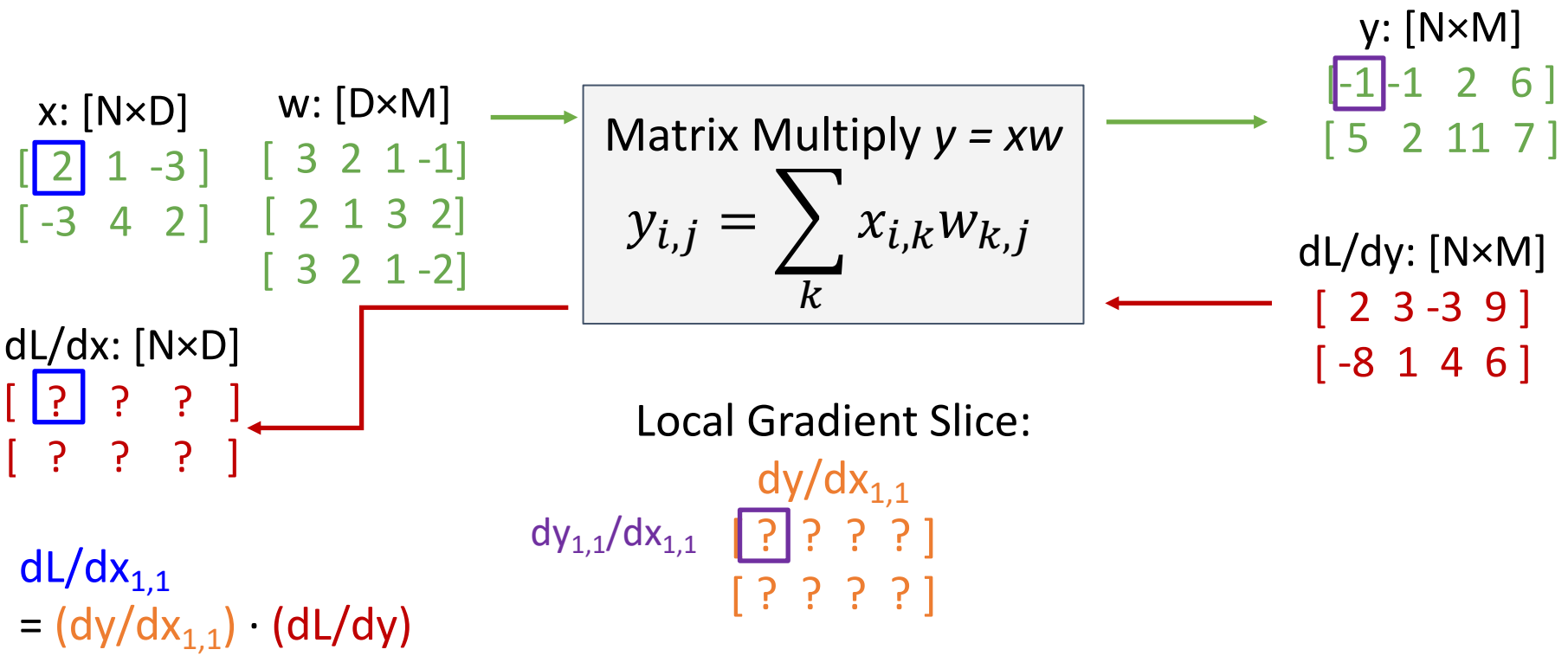
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



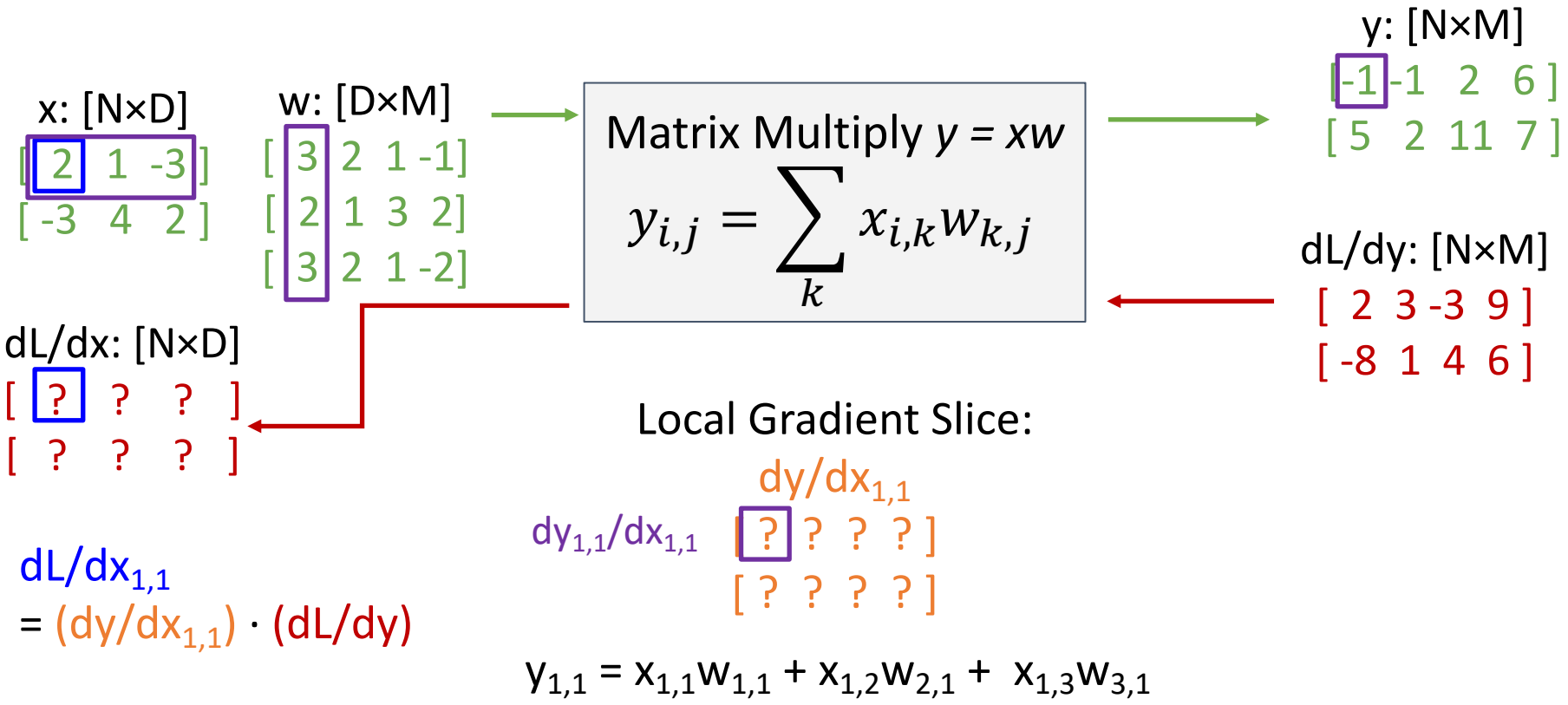
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



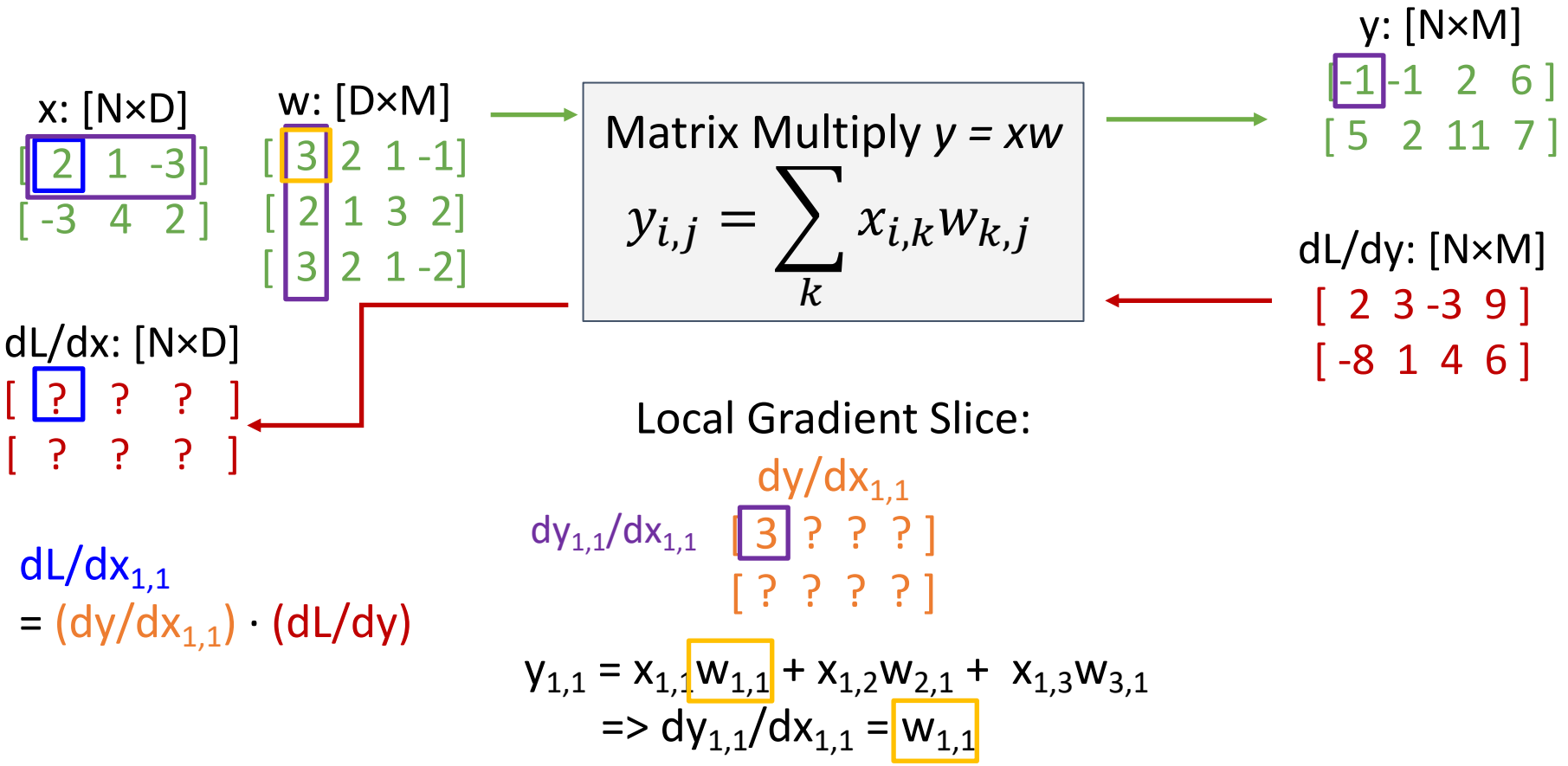
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



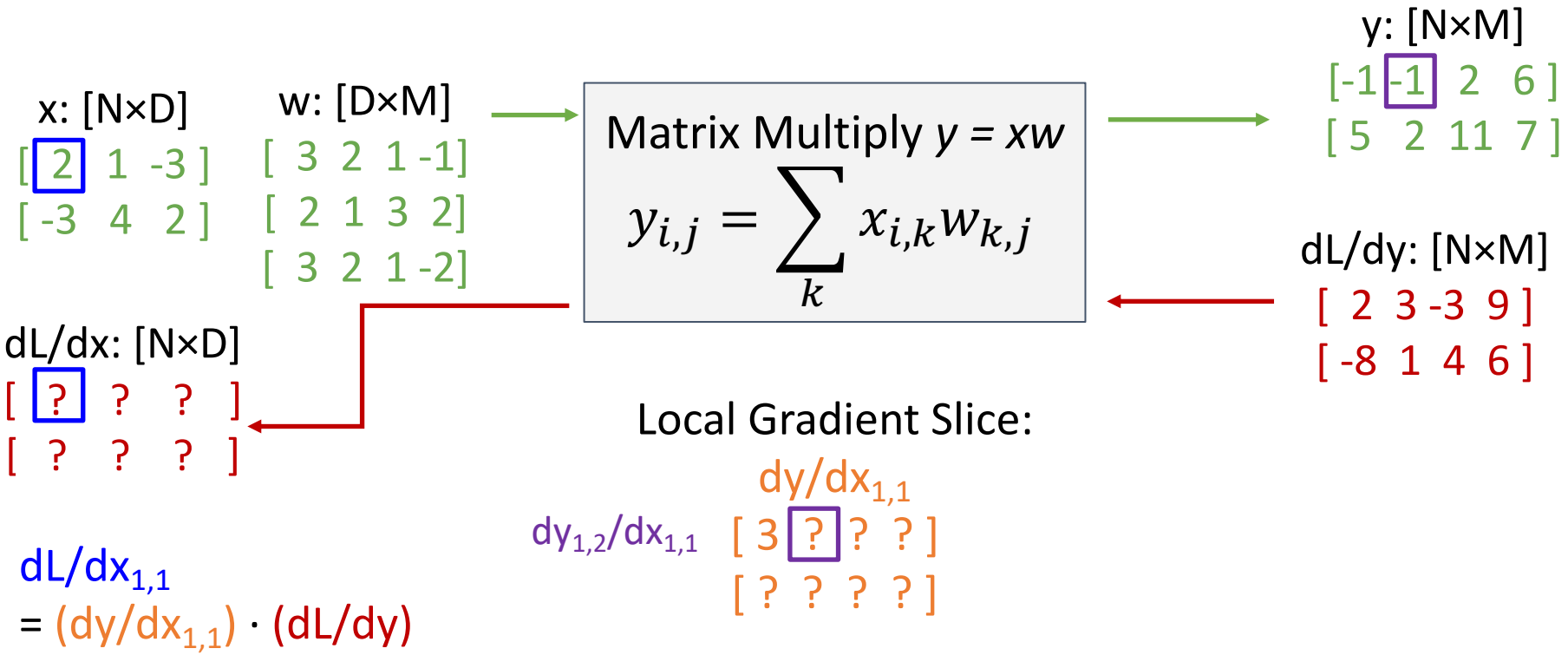
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



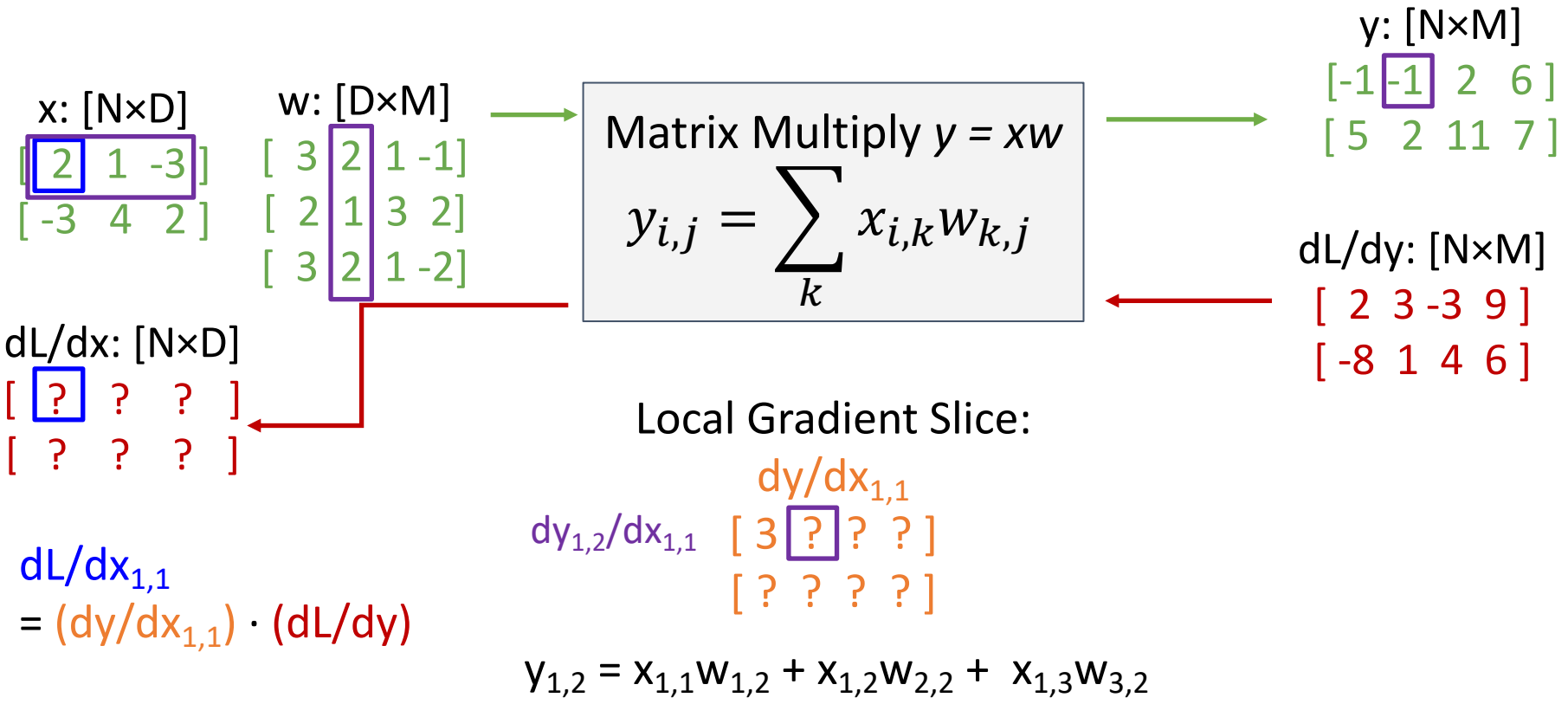
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



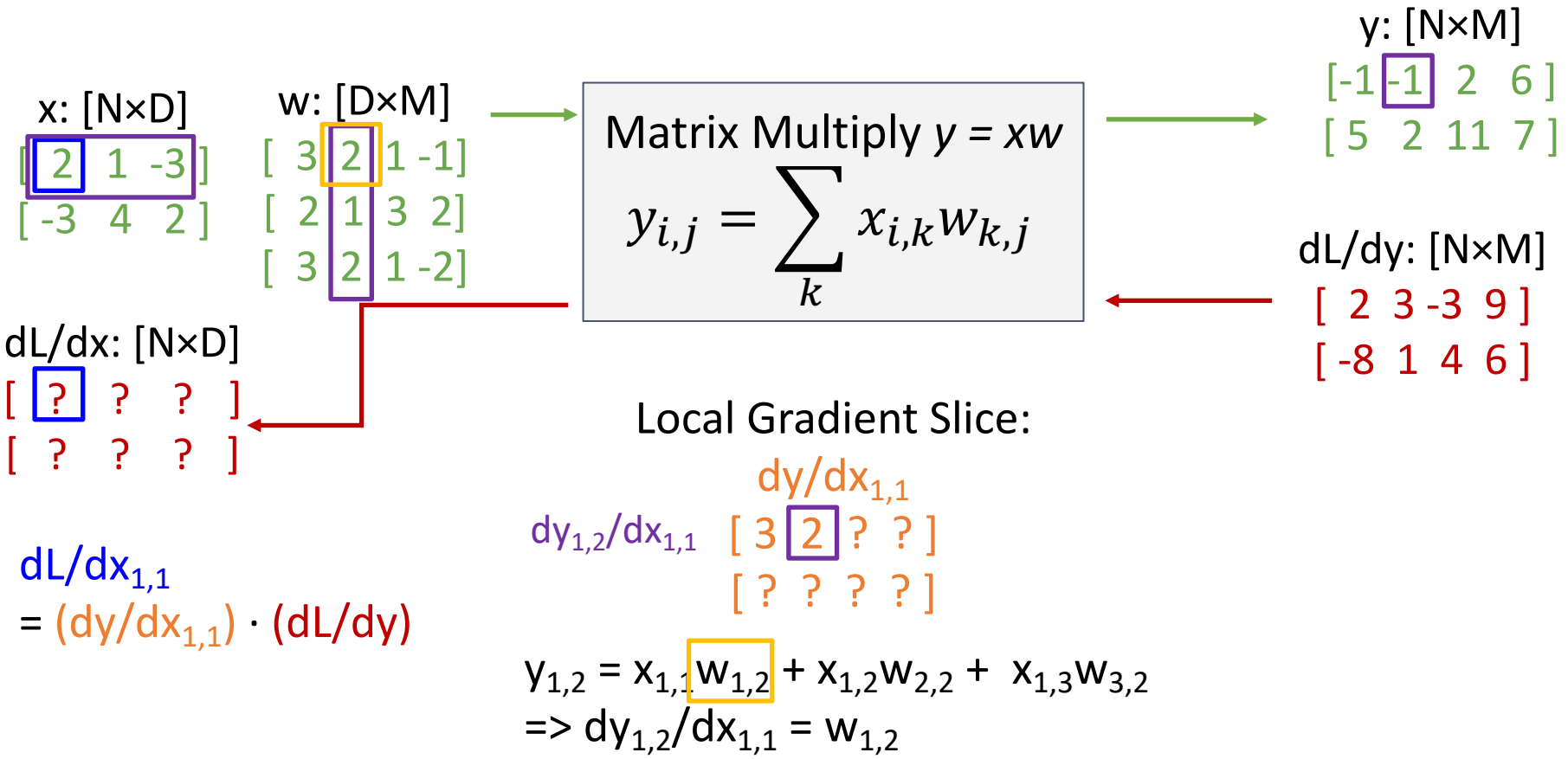
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



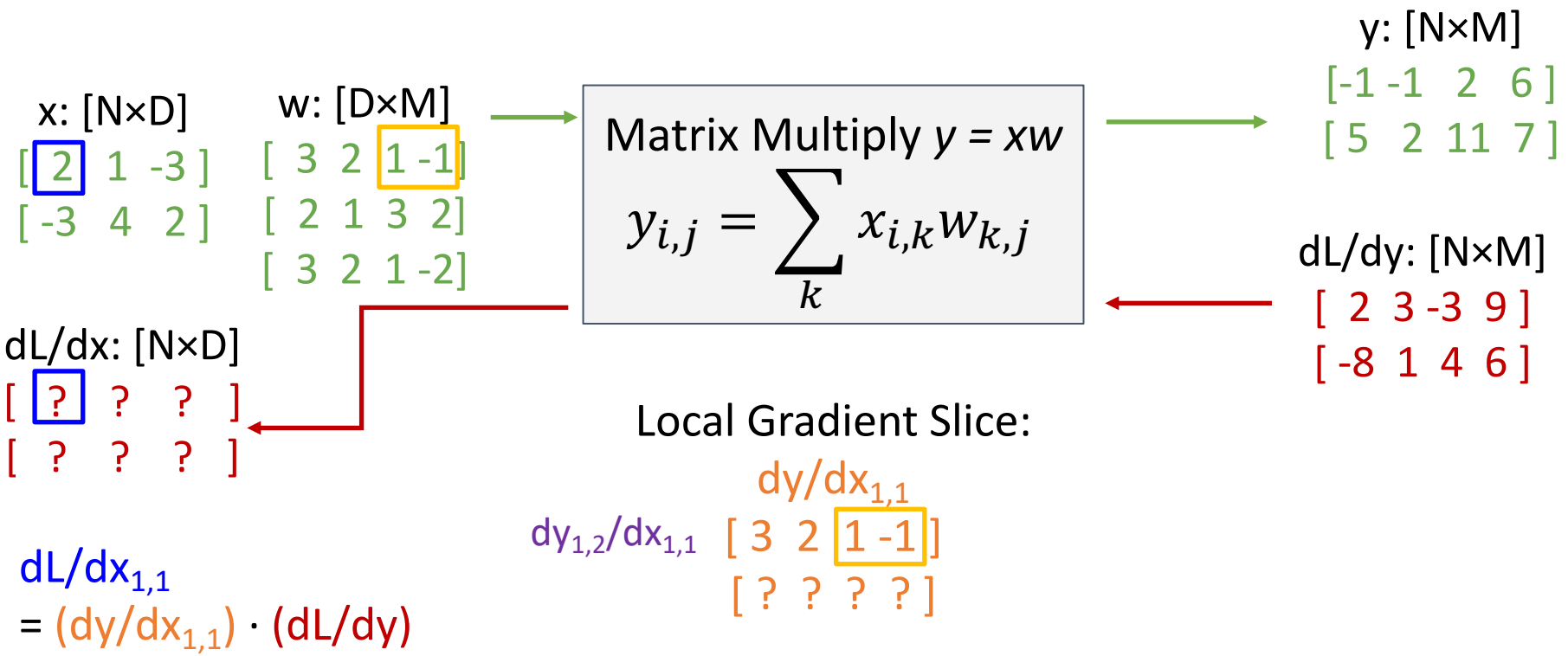
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



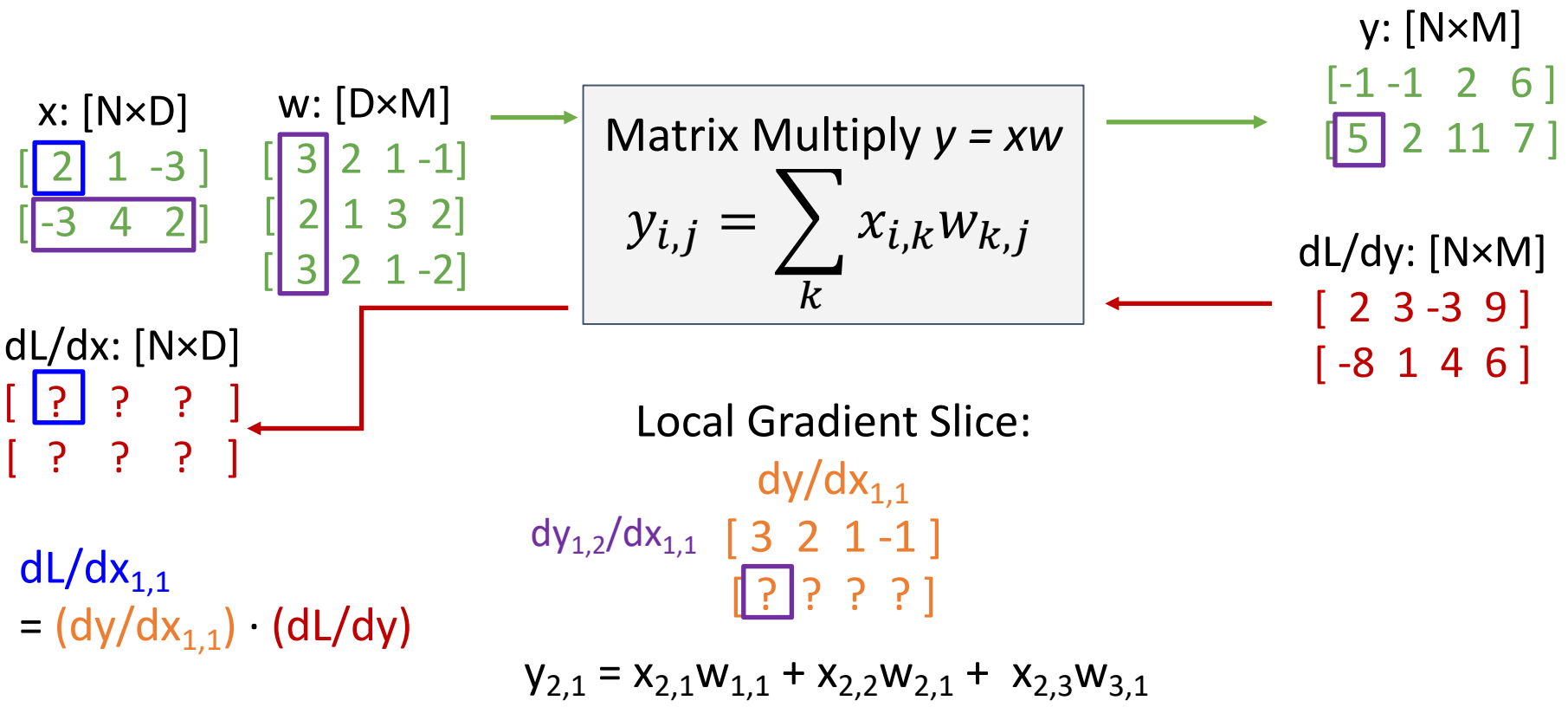
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



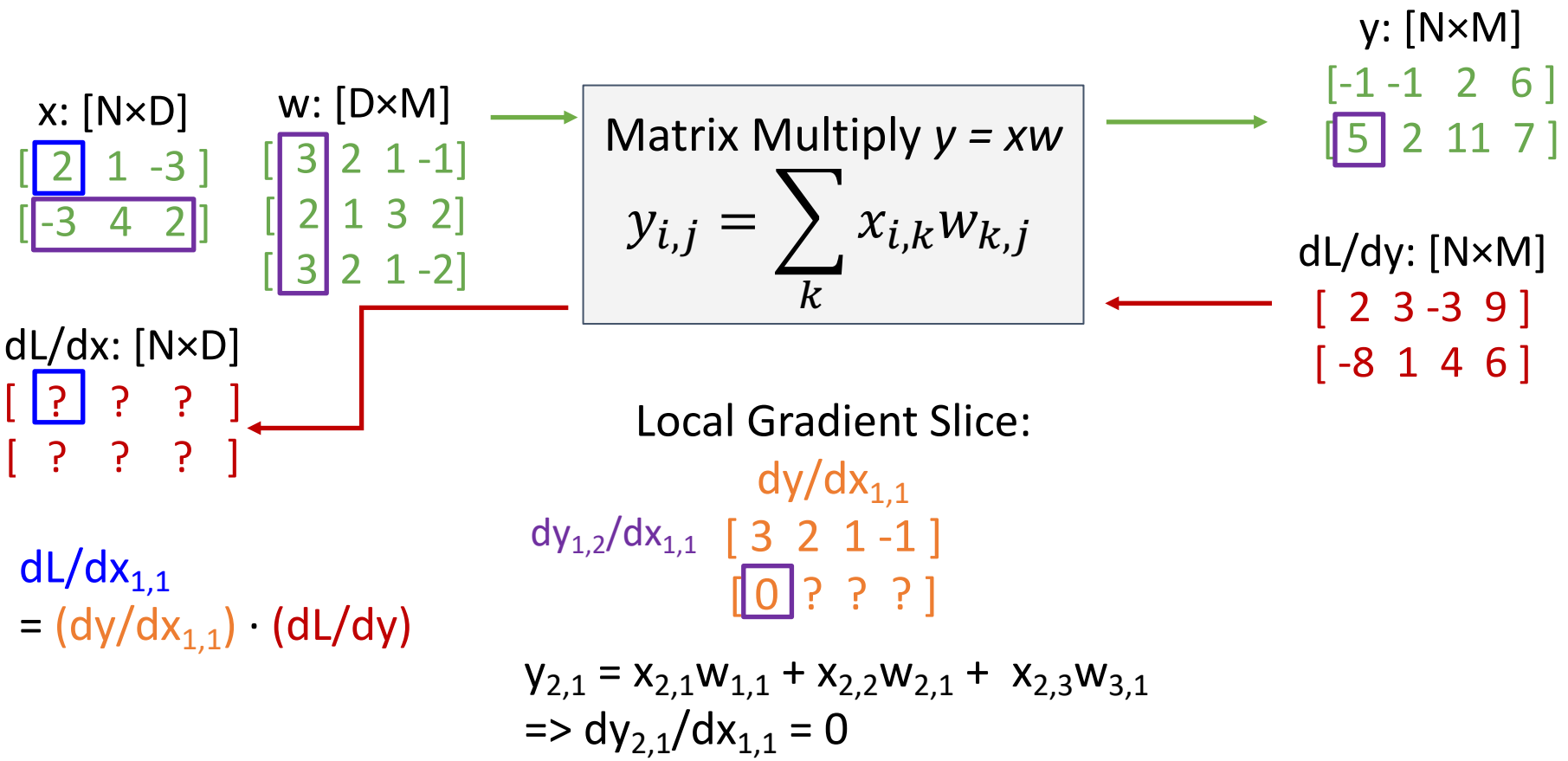
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



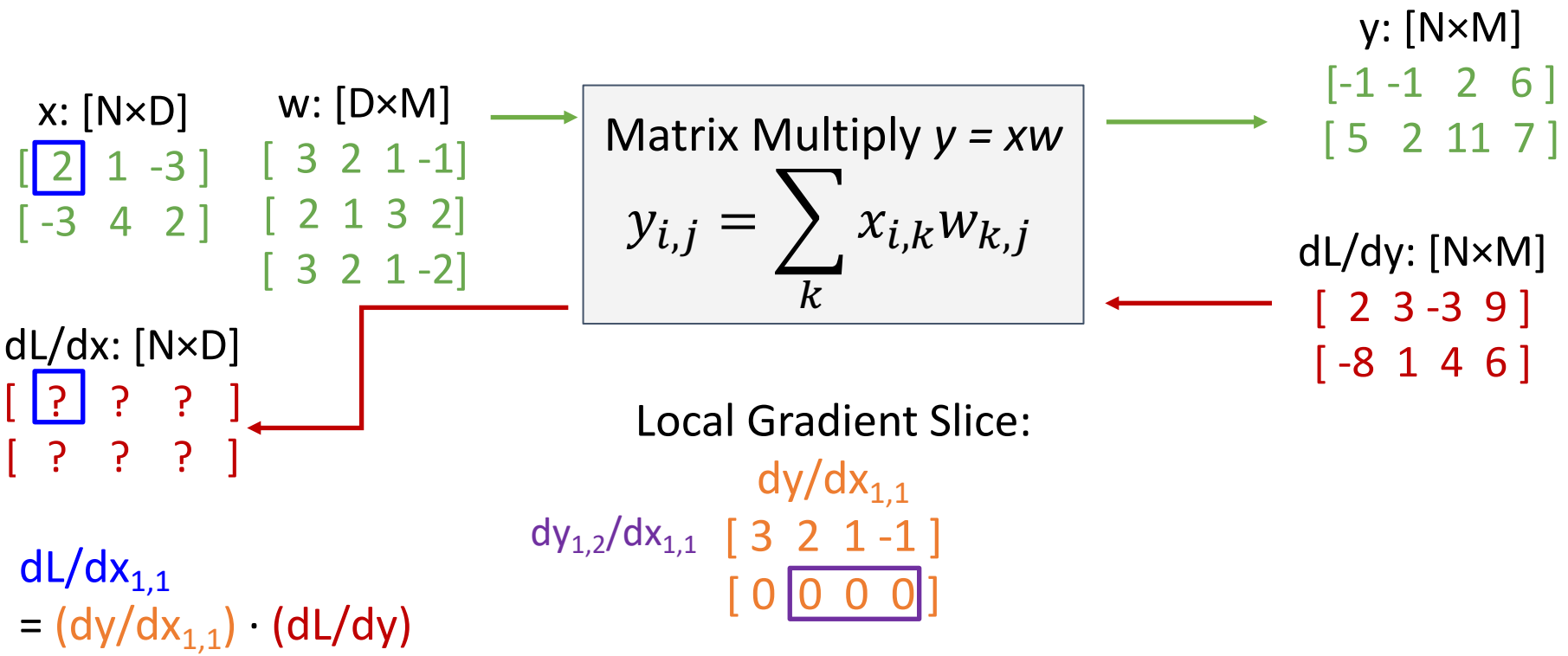
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



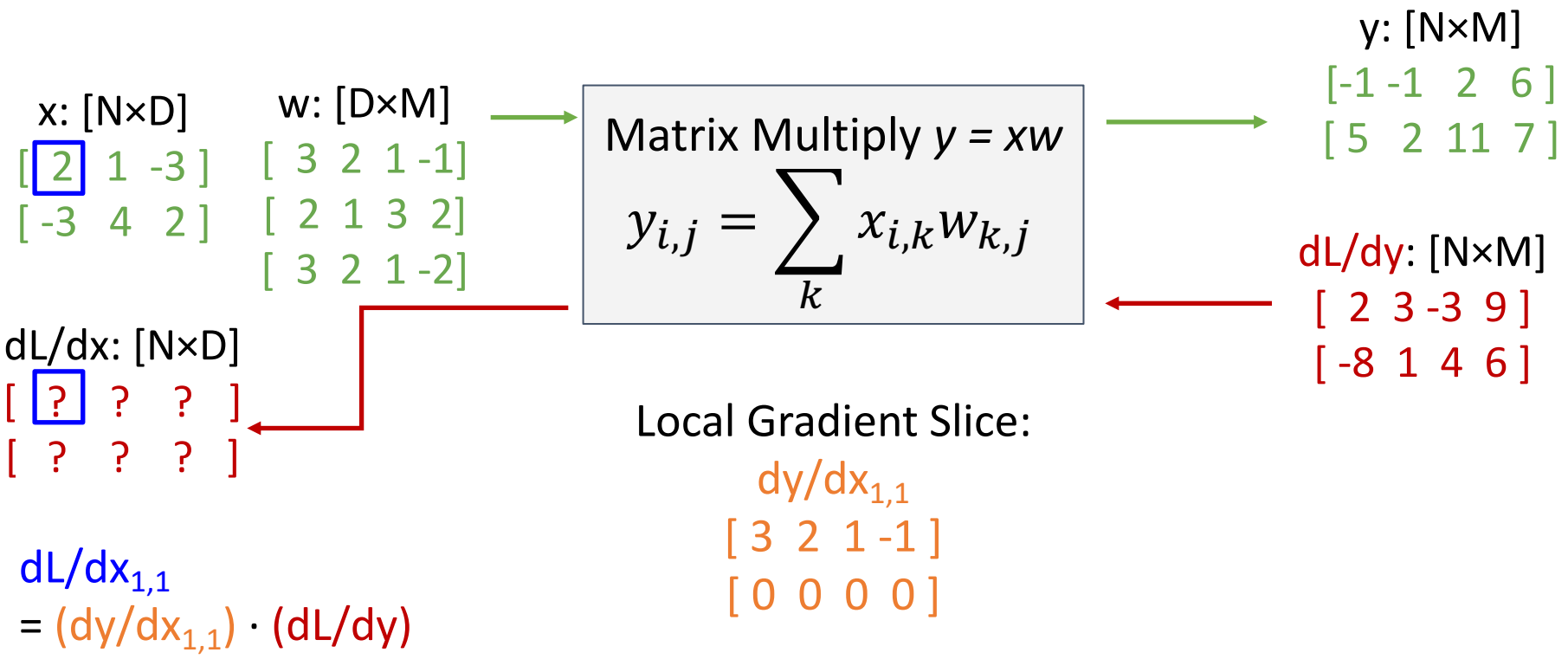
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



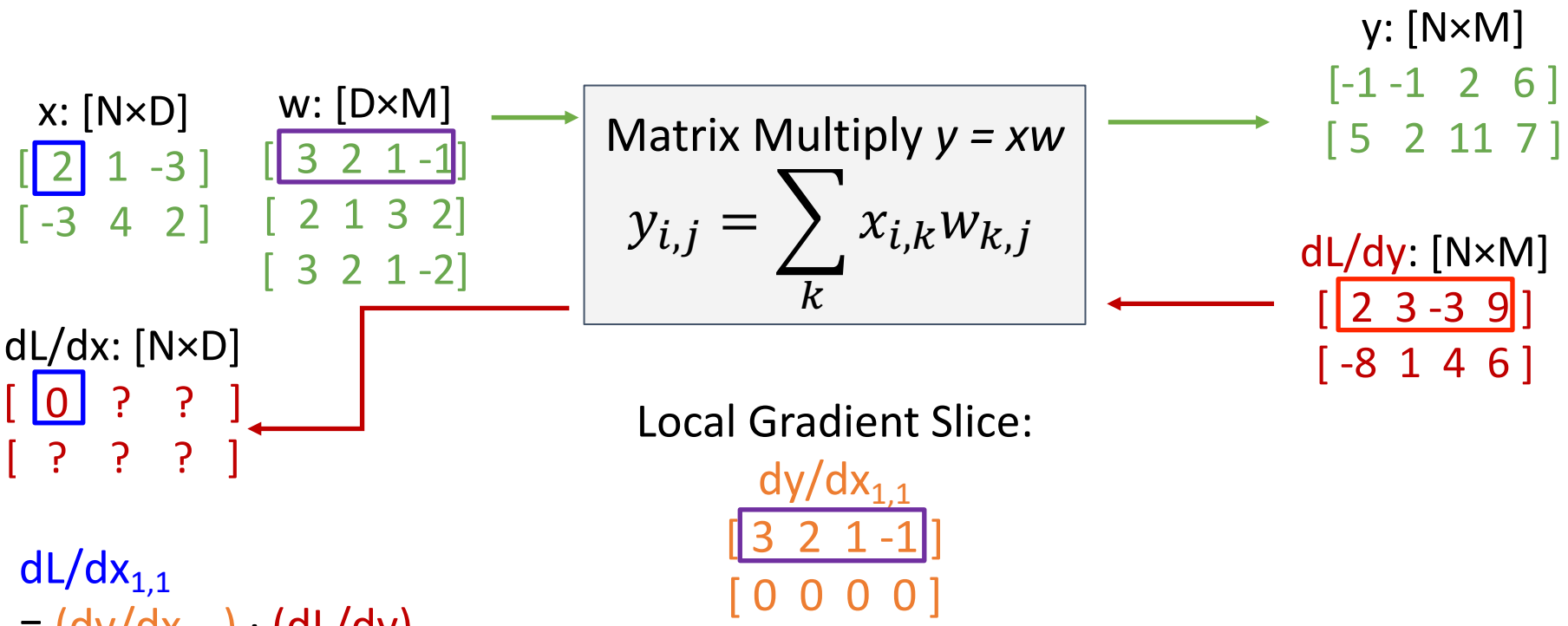
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



Slide credit: D Fouhey & J Johnson

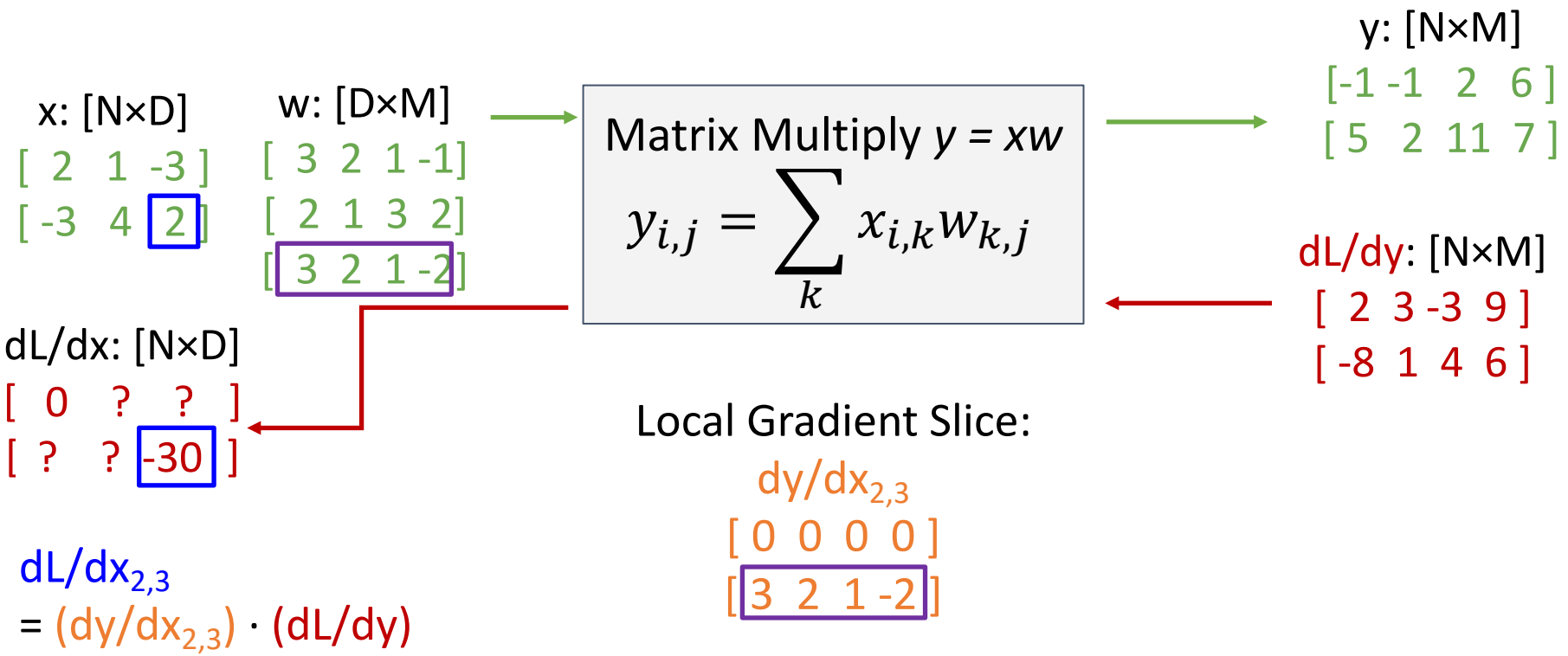
Example: Matrix Multiplication Operator



$dL/dx_{1,1}$
$= (dy/dx_{1,1}) \cdot (dL/dy)$
$= (w_{1,:}) \cdot (dL/dy_{1,:})$
$= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0$

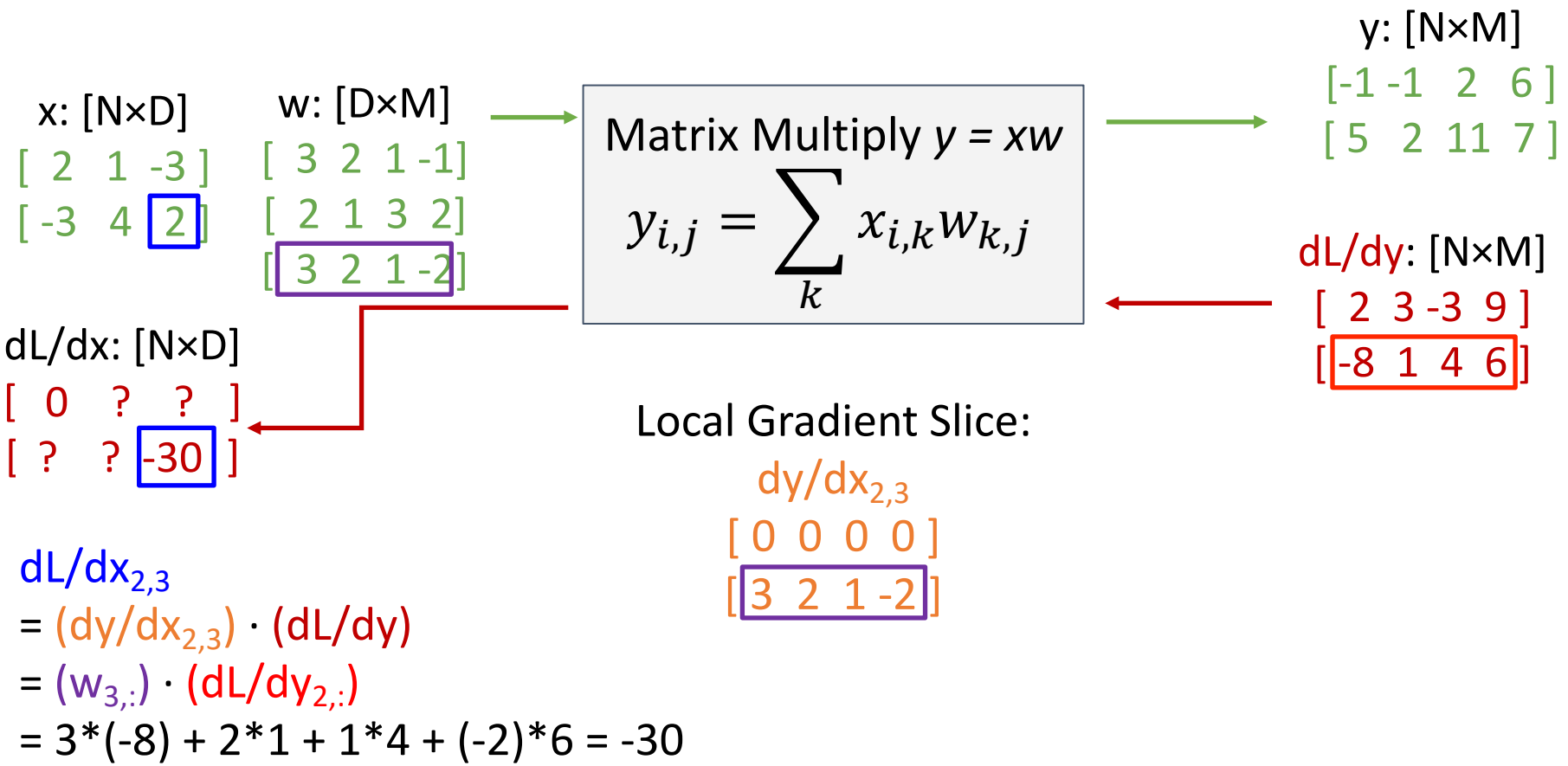
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



Slide credit: D Fouhey & J Johnson

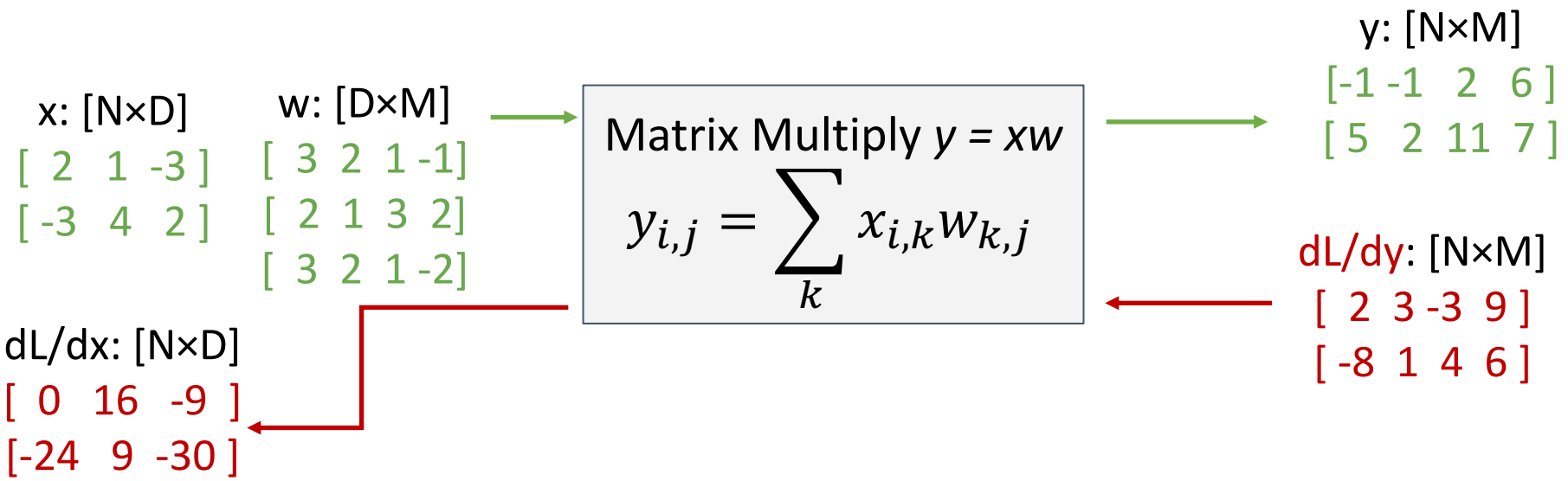
Example: Matrix Multiplication Operator



$dL/dx_{2,3}$
$= (dy/dx_{2,3}) \cdot (dL/dy)$
$= (w_{3,:}) \cdot (dL/dy_{2,:})$
$= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30$

Slide credit: D Fouhey & J Johnson

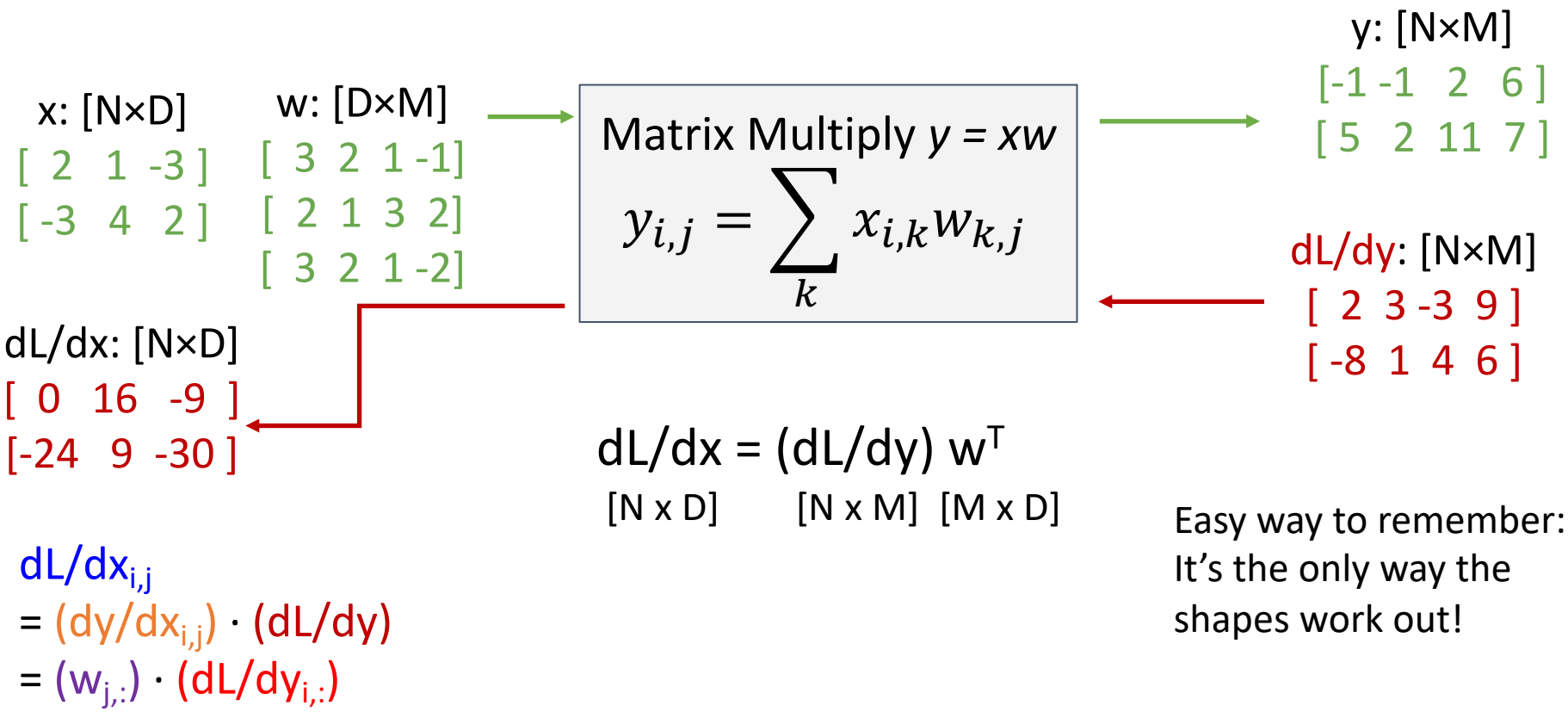
Example: Matrix Multiplication Operator



$dL/dx_{i,j}$
$= (dy/dx_{i,j}) \cdot (dL/dy)$
$= (w_{j,:}) \cdot (dL/dy_{i,:})$

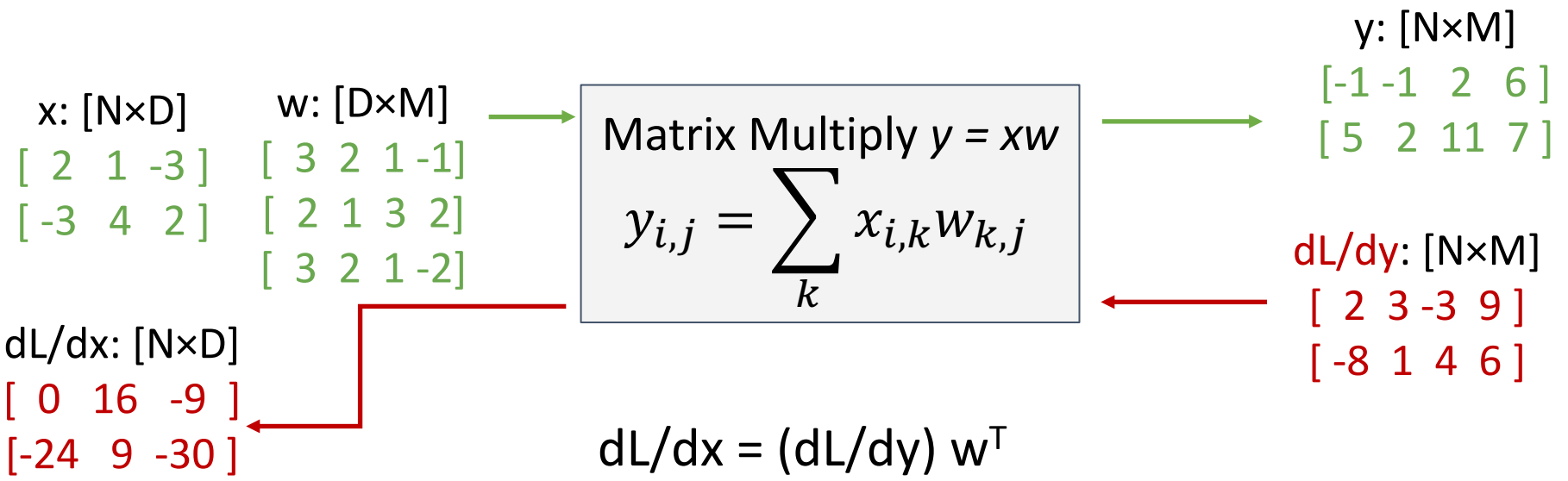
Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



Slide credit: D Fouhey & J Johnson

Example: Matrix Multiplication Operator



$dL/dx = (dL/dy)\, w^T$

[N x D] [N x M] [M x D]

$dL/dw = x^T\, (dL/dy)$

[D x M] [D x N] [N x M]

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

Slide credit: D Fouhey & J Johnson

Extra Resources

- Prof. Erik Learned-Miller's notes on vector, matrix, and tensor derivatives

  https://compsci682-fa21.github.io/docs/vecDerivs.pdf

- Prof. Justin Johnson's notes for a derivation of backprop for matrix multiplication

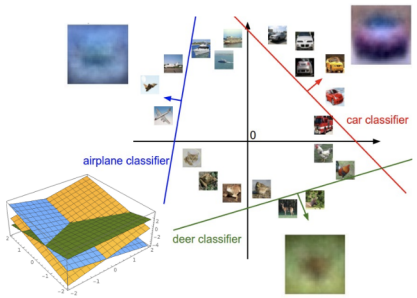
  https://web.eecs.umich.edu/~justincj/teaching/eecs442/notes/linear-backprop.html

Summary

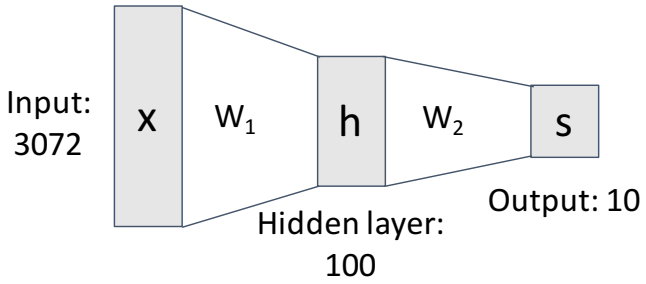
- **Computational graphs** are a useful data structure for organizing computation
- **Backpropagation** lets us compute compute gradients in a computational graph
- **Modular backprop** code composes pairs of forward and backward functions
- Backprop extends to vector and tensor-valued functions

Slide adapted from: D Fouhey & J Johnson

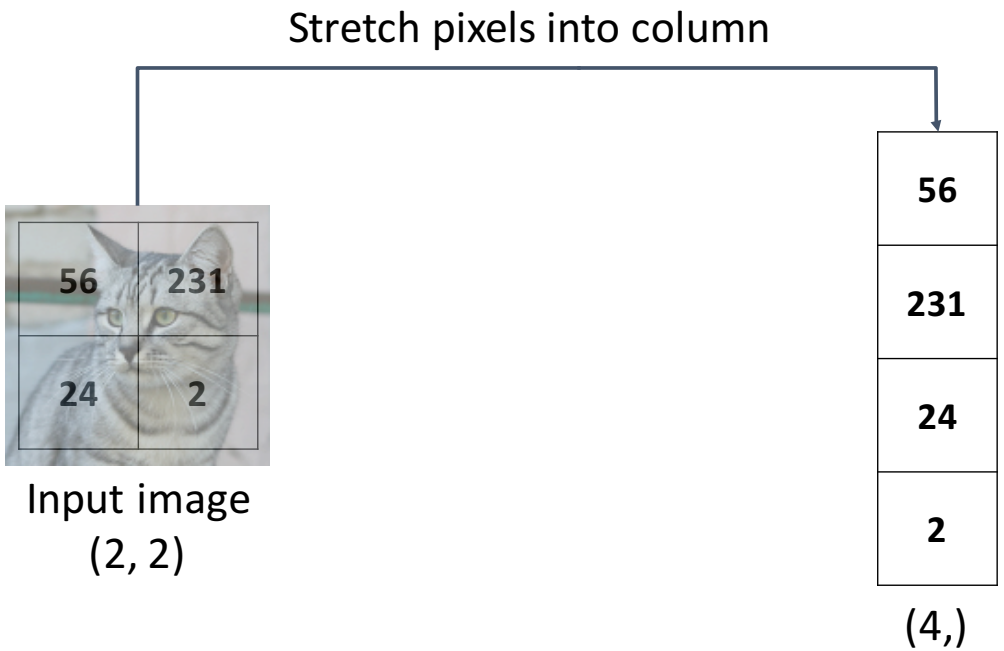
f(x,W) = Wx



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



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



Slide credit: D Fouhey & J Johnson

Next Class

Convolutional Neural Networks!