Photogrammetry & Robotics Lab

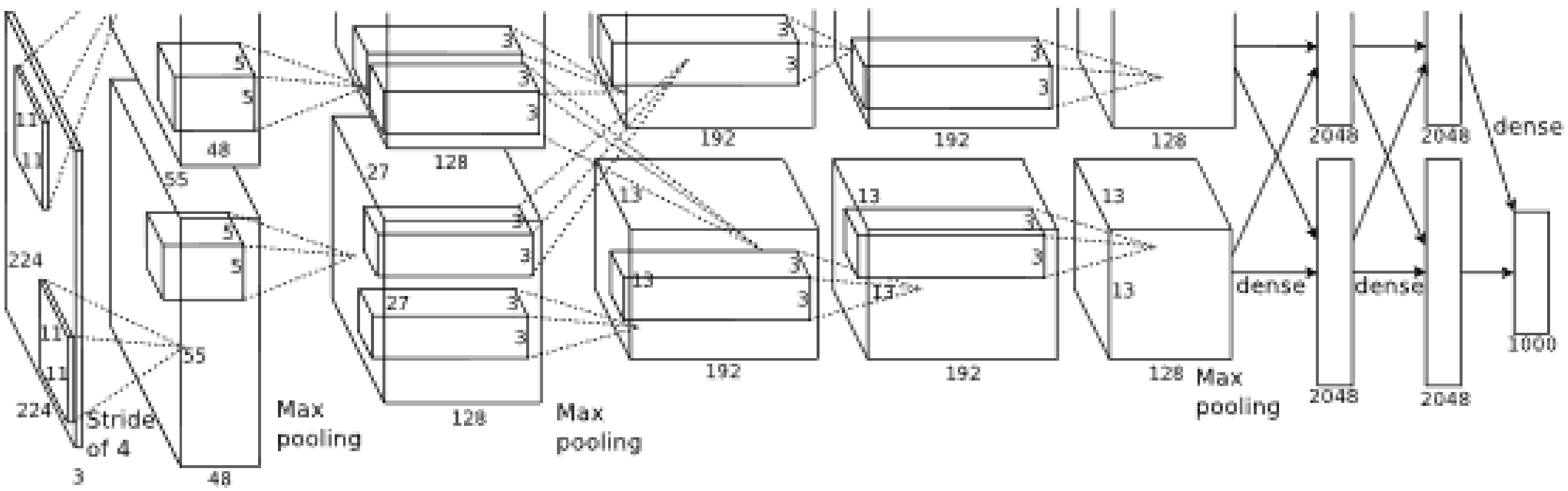
Machine Learning for Robotics and Computer Vision Tutorial

CNNs and Learning CNNs

Jens Behley

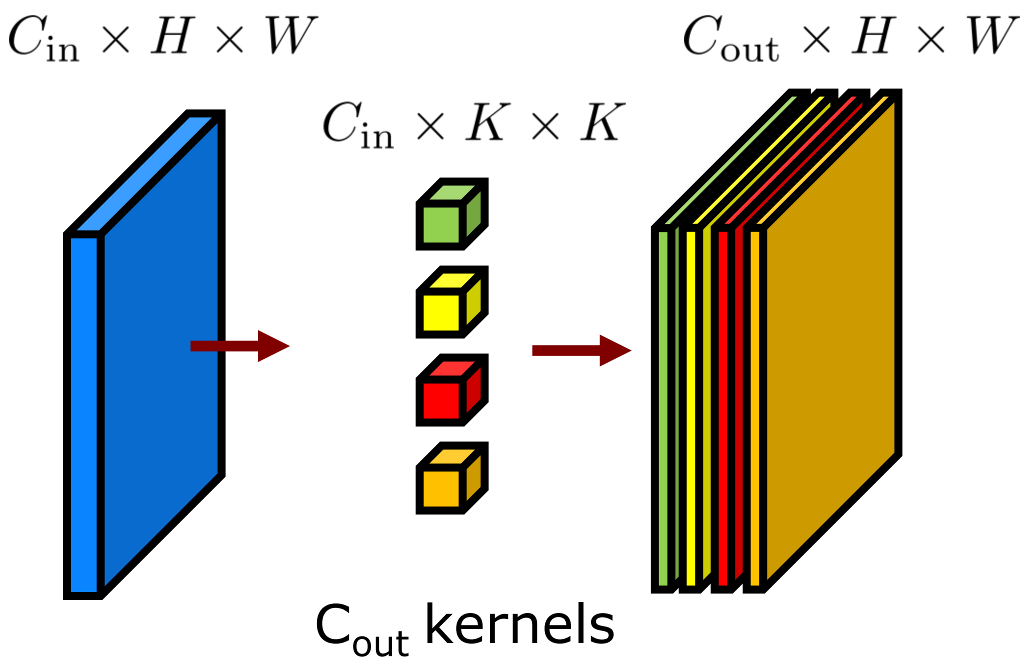
Recap CNNs

Recap: Main Building Blocks of CNNs



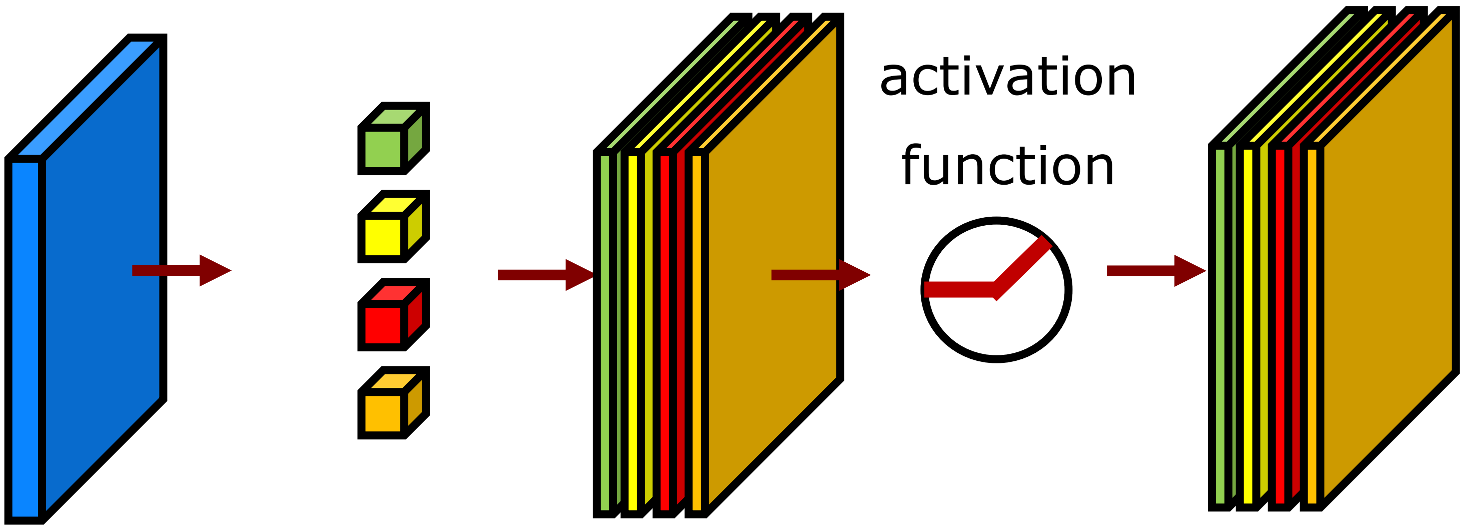
1. Convolutional Layers + Activation (ReLU)
2. Pooling Layers
3. Fully-connected Layers

Recap: Convolutional Layer



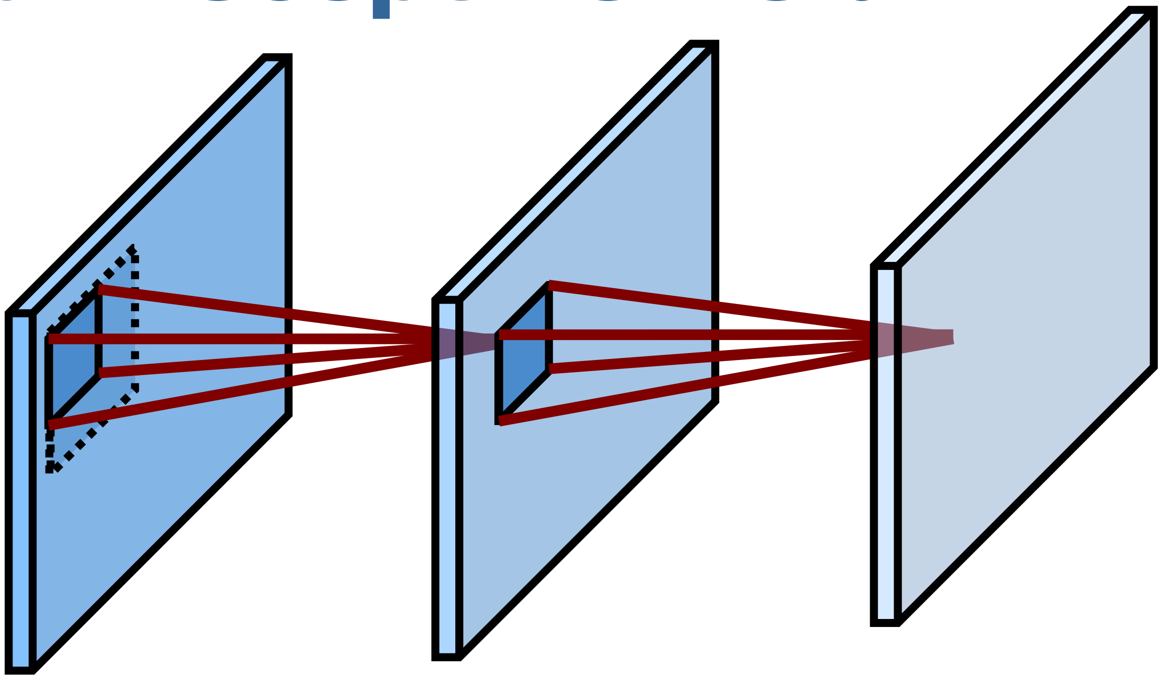
- Use multiple kernels to produce C_{out} maps
- Non-linear activation function applied element-wise on convolution results

Recap: ConvLayer + Activation Function



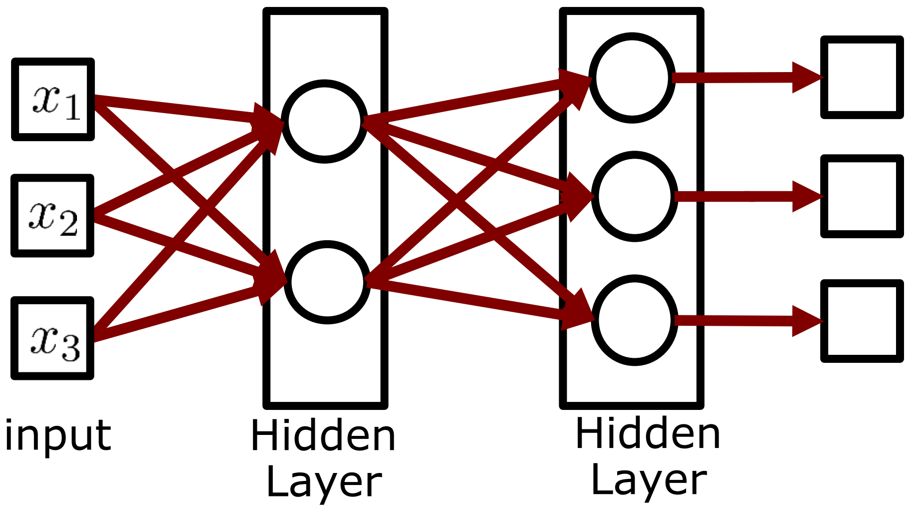
- Activation function (such as ReLU) applied after each convolutional layer
- Usually only implicit in the graphical representation

Recap: Receptive field



- Location in deeper layers take inputs of window of earlier layers
- Deeper layers “see” more from earlier layers

Recap: Fully-Connected Layer



$$\mathbf{x}_{\text{out}} = \mathbf{W}\mathbf{x}_{\text{in}}$$

$$\mathbf{x}_{\text{out}} \in \mathbb{R}^{C_{\text{in}}}$$
$$\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$$
$$\mathbf{x}_{\text{in}} \in \mathbb{R}^{C_{\text{in}}}$$

- Linear Weight matrix that takes all input values and produces C_{out} output values

Questions?

Recap: Loss Minimization

- Loss $\ell(y_i, f(\mathbf{x}_i; \theta)) \in \mathbb{R}$ determines difference between prediction $f(\mathbf{x}_i; \theta)$ and target value y_i

$$L(\theta) = \frac{1}{N} \sum_i \ell(y_i, f(\mathbf{x}_i; \theta))$$

- Typical loss functions
  - Regression: **L2 loss**
  - Classification: **Cross-entropy loss**

Recap: L2 Loss

- L2-loss is defined as

$$\ell(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i, \theta))^2$$

- Intuitively, predicted values $f(\mathbf{x})$ should be close to target values
- Equivalent to negative log-likelihood with normal distributed error (see **regression lecture**)

MSELOSS

L2 Loss in PyTorch

CLASS `torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean')` [SOURCE]

Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then:

Recap: Cross Entropy Loss

- As before, for classification:

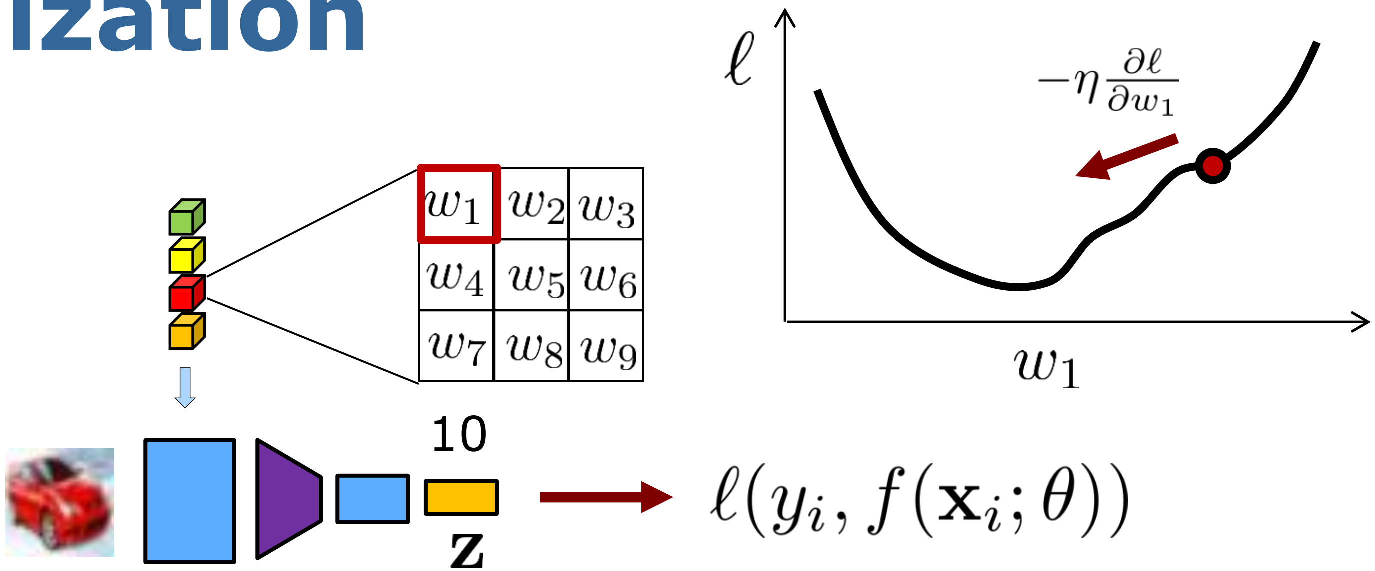
$$P(y = j|\mathbf{x}) = \mathrm{softmax}_j(f_1(\mathbf{x}), \dots, f_C(\mathbf{x}))$$
$$= \frac{\exp(f_j(\mathbf{x}))}{\sum_k \exp(f_k(\mathbf{x})))}$$

- The negative log-likelihood is then:

$$\ell(j, f(\mathbf{x})) = -\log \frac{\exp(f_j(\mathbf{x}))}{\sum_k \exp(f_k(\mathbf{x}))}$$
$$= -f_j(\mathbf{x}) + \log(\textstyle\sum_k \exp(f_k(\mathbf{x}))$$

- But why is this called *cross entropy* then?

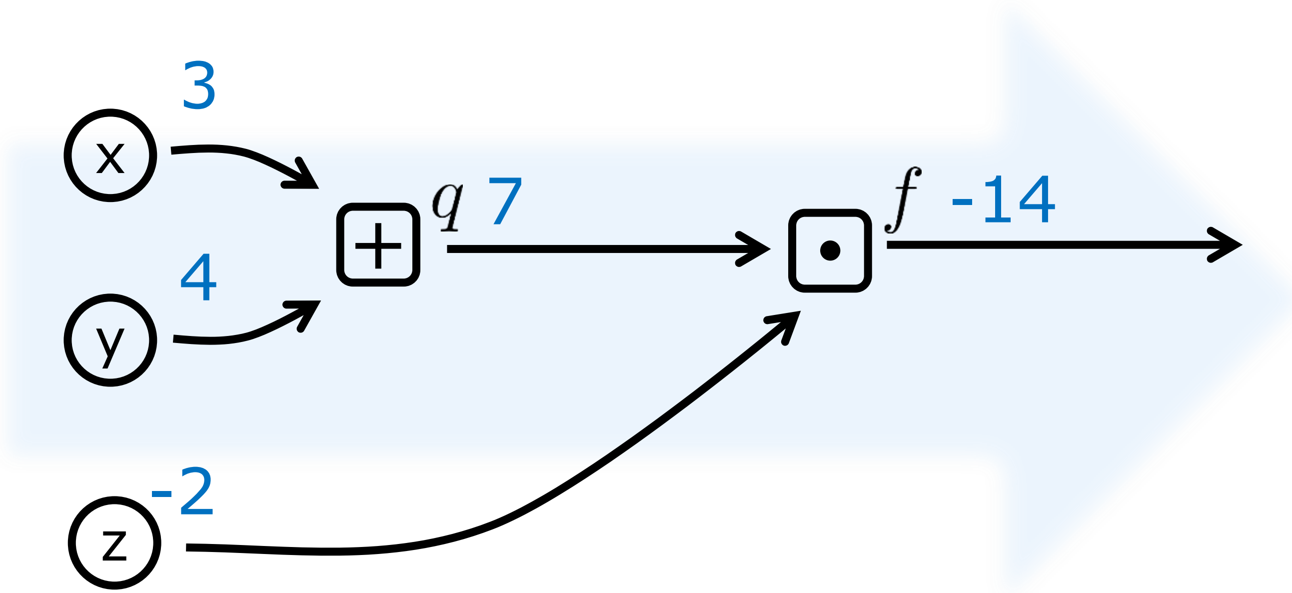
Recap: Gradient-based Optimization



- As before, **partial derivatives** tell us in which direction to change parameters such that loss is minimized

Backpropagation: Forward Pass

- For a given input, we first compute all activations in the forward pass

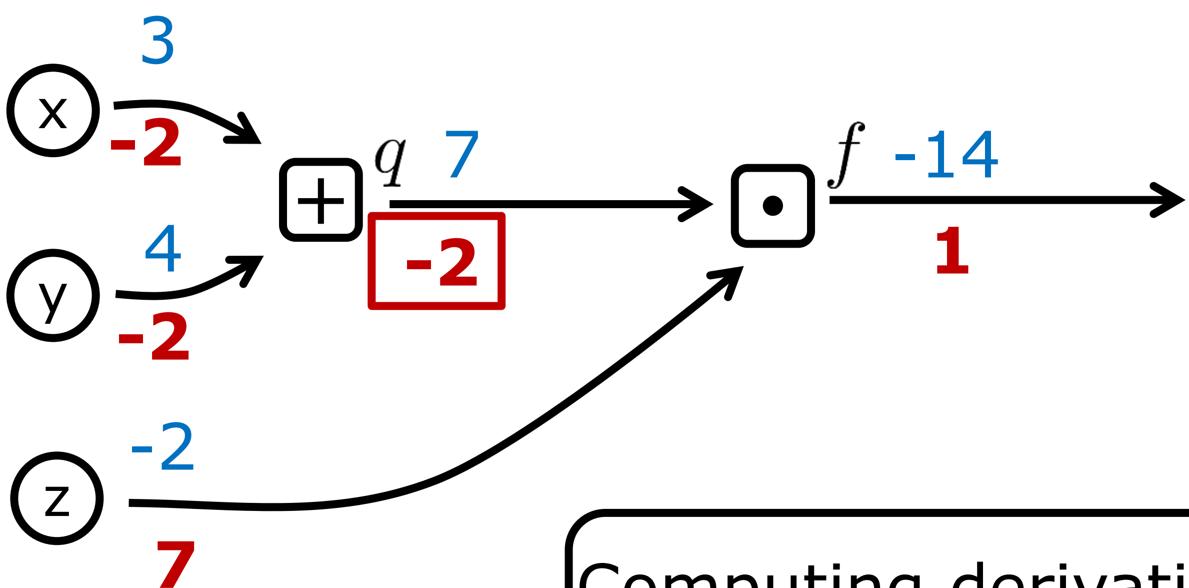


Slides inspired by Justin Johnson's lecture on backprop

Backpropagation: Backward Pass

$$\boxed{q = x + y}$$
$$f = q \cdot z$$

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \cdot \frac{\partial f}{\partial q} = 1 \cdot -2$$

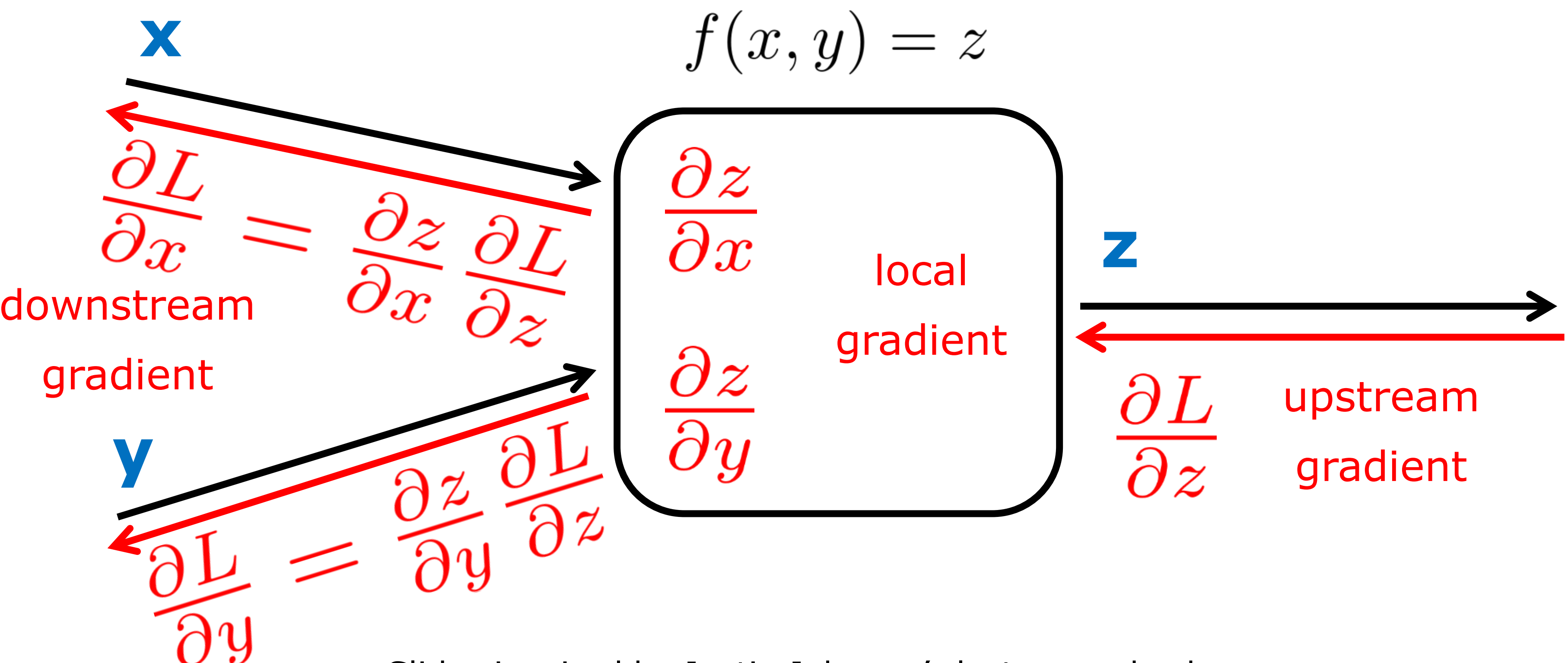


Computing derivatives only involves **local** information

Slides inspired by Justin Johnson's lecture on backprop

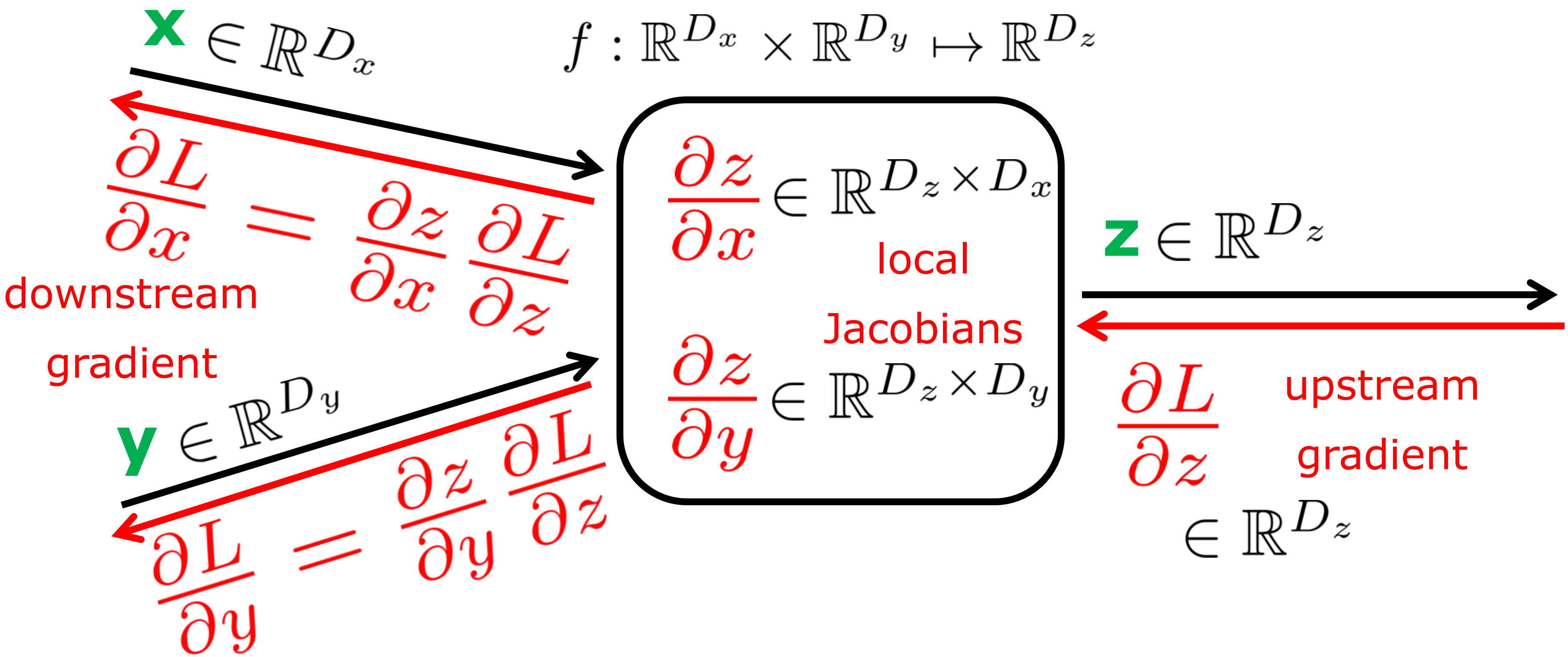
Local connectivity

- For each compute node only the incident and outgoing edges are relevant
- Gradient "messages" are passed along the edges



Slides inspired by Justin Johnson's lecture on backprop

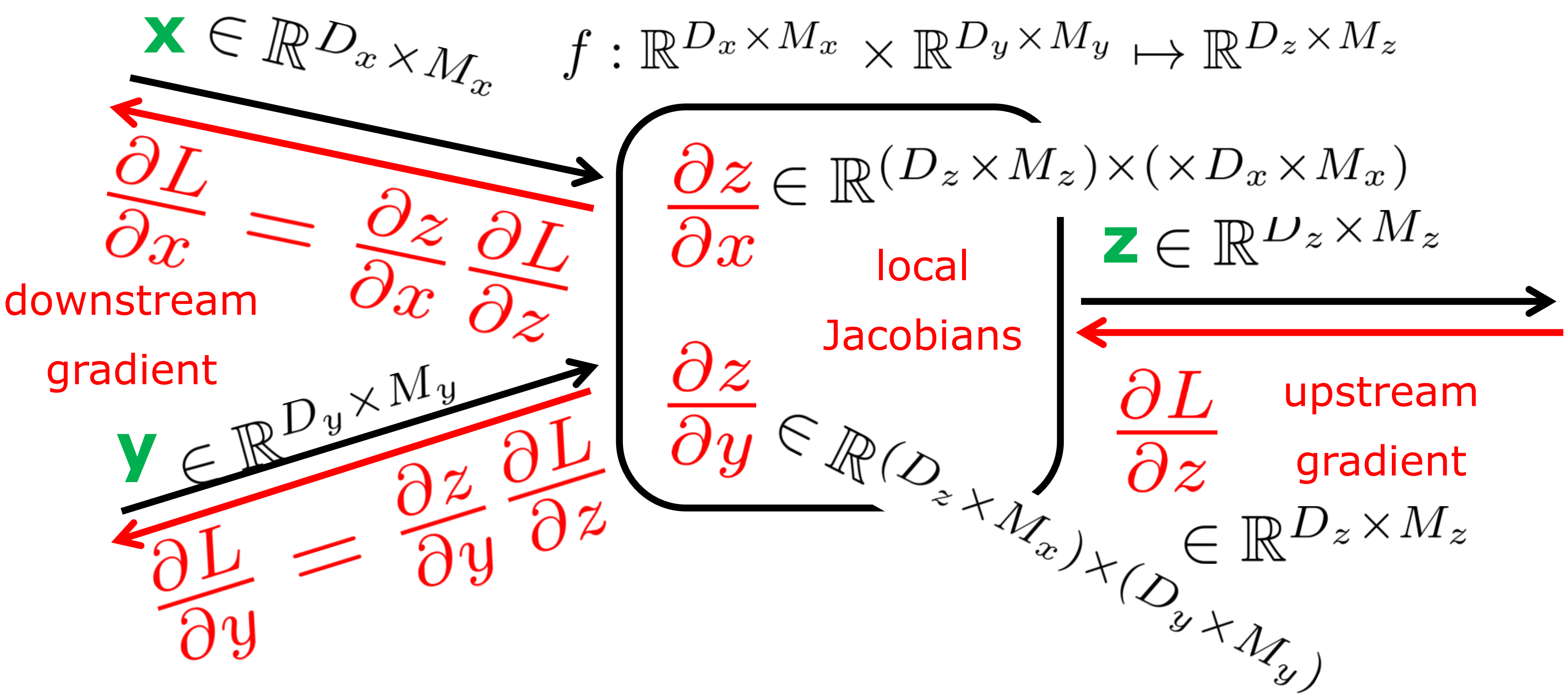
Backpropagation with vectors



- **Important:** For matrix-vector multiplication in the downstream gradient, we use transpose of Jacobian

Slides inspired by Justin Johnson's lecture on backprop

Backpropagation with matrices



- With matrices it get's a bit more involved, since then the Jacobian is a 4 dimensional tensor…

Example: Matrix Multiplication

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 $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

Example: Matrix Multiplication

x: [N×D]
[$\boxed{2}$ 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[$\boxed{?}$? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$
[? ? ? ?]
[? ? ? ?]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} \boxed{-1} & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,1}/dx_{1,1}$ $\begin{bmatrix} \boxed{?} & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} \boxed{-1} & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{cc} dy_{1,1}/dx_{1,1} & \begin{bmatrix} \boxed{?} & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \end{array}$$

$dL/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} \boxed{-1} & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$dL/dx_{1,1}$

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

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{cc} dy_{1,1}/dx_{1,1} & \begin{bmatrix} \boxed{3} & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \end{array}$$

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

$\Rightarrow dy_{1,1}/dx_{1,1} = \boxed{w_{1,1}}$

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & \boxed{-1} & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$$dy_{1,2}/dx_{1,1} \quad \begin{bmatrix} 3 & \boxed{?} & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

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 $y = xw$

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

y: [N×M]

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ [3 ? ? ?]
[? ? ? ?]

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & \boxed{-1} & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$

$$\begin{bmatrix} 3 & \boxed{2} & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$dL/dx_{1,1}$

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

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

$\Rightarrow dy_{1,2}/dx_{1,1} = w_{1,2}$

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{r l} dy_{1,2}/dx_{1,1} & \begin{bmatrix} 3 & 2 & \boxed{1 \; -1} \\ ? & ? & ? & ? \end{bmatrix} \end{array}$$

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ \boxed{5} & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{ll} dy_{1,2}/dx_{1,1} & [\ 3\ \ 2\ \ 1\ \ -1\ ] \\ & [\ \boxed{?}\ \ ?\ \ ?\ \ ?\ ] \end{array}$$

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

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

Example: Matrix Multiplication

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 $y = xw$

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

y: [N×M]

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ [3 2 1 -1]

[0 ? ? ?]

$dL/dx_{1,1}$

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

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

$\Rightarrow dy_{2,1}/dx_{1,1} = 0$

Example: Matrix Multiplication

x: [N×D]
[$\boxed{2}$ 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[$\boxed{?}$? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$

[3 2 1 -1]
[0 $\boxed{0 \ 0 \ 0}$]

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

Example: Matrix Multiplication

x: [N×D]
[$\boxed{2}$ 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[$\boxed{?}$? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$
[3 2 1 -1]
[0 0 0 0]

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

Example: Matrix Multiplication

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 $y = xw$

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

y: [N×M]

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

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

dL/dx: [N×D]

[0 ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

[3 2 1 -1]
[0 0 0 0]

$$\frac{\partial L}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial L}{\partial y}$$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} 0 & ? & ? \\ ? & ? & \boxed{-30} \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{2,3}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

$dL/dx_{2,3}$

$= (dy/dx_{2,3}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 $\boxed{2}$]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[$\boxed{3\ 2\ 1\ -2}$]

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[$\boxed{-8\ 1\ 4\ 6}$]

dL/dx: [N×D]
[0 ? ?]
[? ? $\boxed{-30}$]

Local Gradient Slice:

$dy/dx_{2,3}$
[0 0 0 0]
[$\boxed{3\ 2\ 1\ -2}$]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

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

dL/dx: [N×D]

$$\begin{bmatrix} 0 & 16 & -9 \\ -24 & 9 & -30 \end{bmatrix}$$

$dL/dx_{i,j}$

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

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

Example: Matrix Multiplication

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 $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

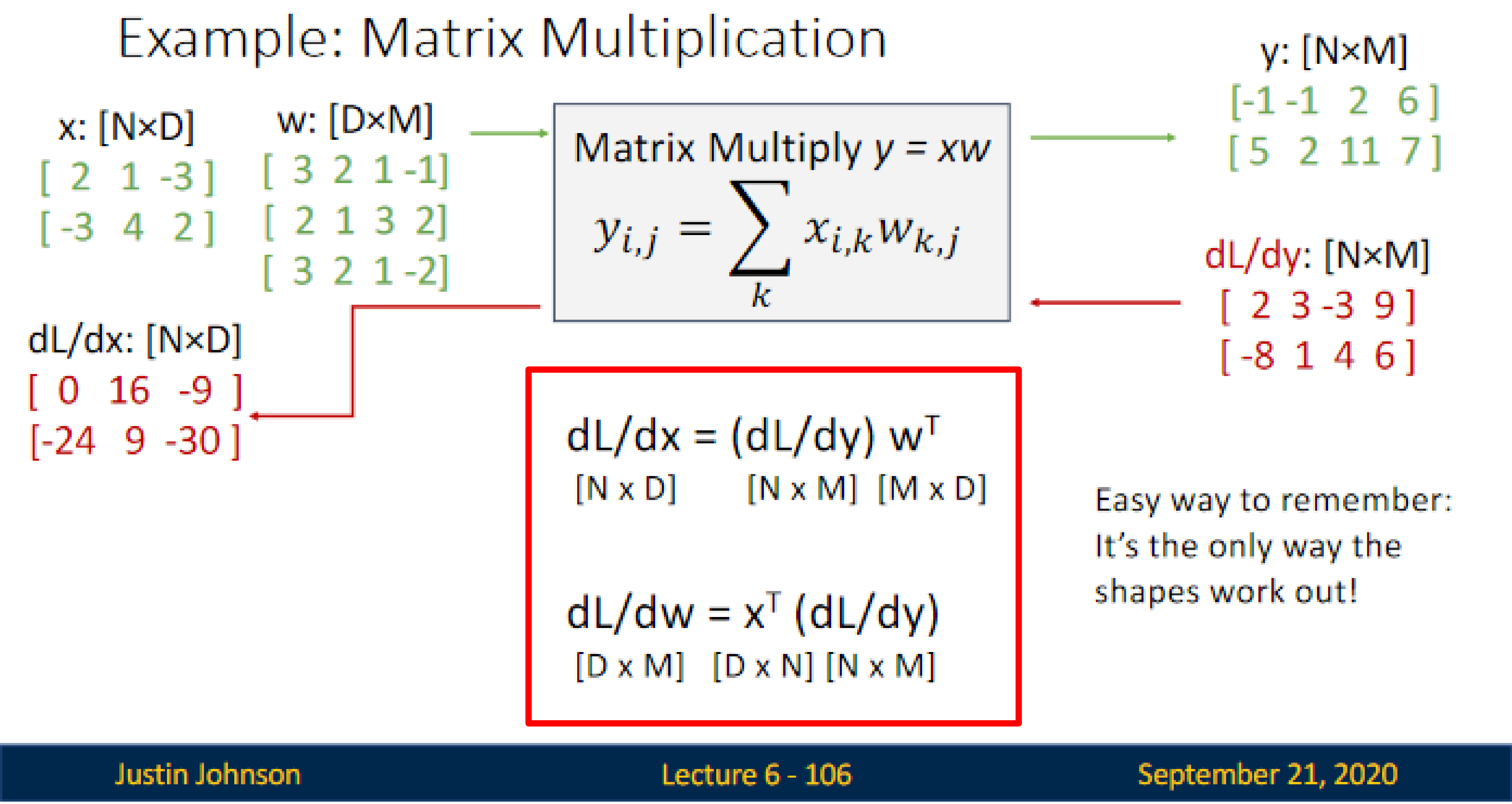
dL/dx: [N×D]
[0 16 -9]
[-24 9 -30]

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

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

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

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



- See also
  https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/linear-backprop.html
  for more details.

Questions?

Learning Rate

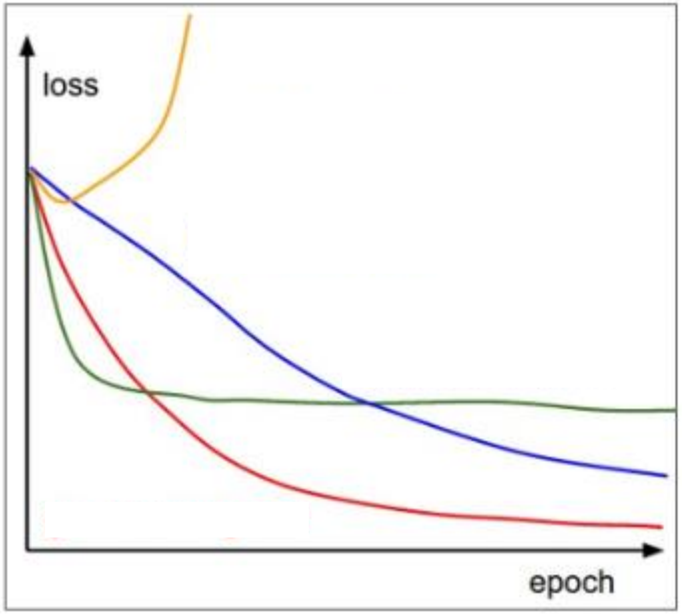
```
while not converged:
```

Compute $\frac{dL}{d\theta}$ for current batch

$$\theta_t = \theta_{t-1} - \eta \frac{dL}{d\theta}$$

- (Most) important hyperparameter: learning rate (even with ADAM needs to be adapted)
- Parameter-adaptive methods (say ADAM) it is less critical to find a specific "good" learning rate
- How to choose a "good" learning rate?

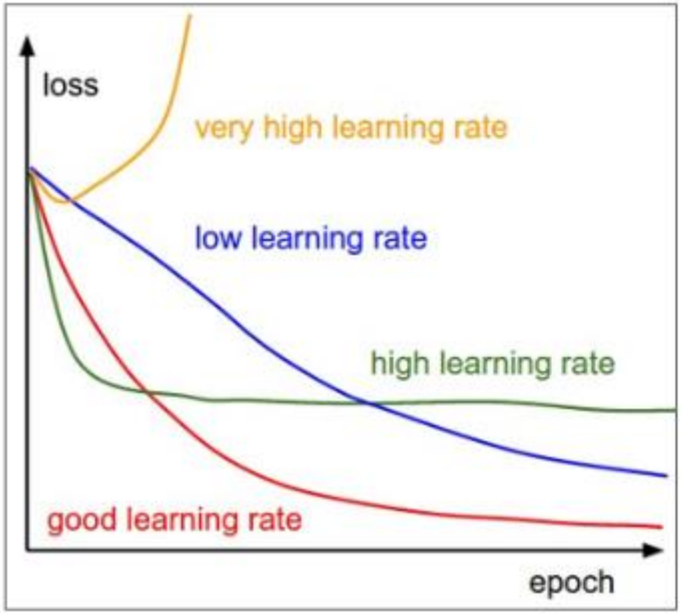
Initial Learning Rate



Which learning rate would you select?

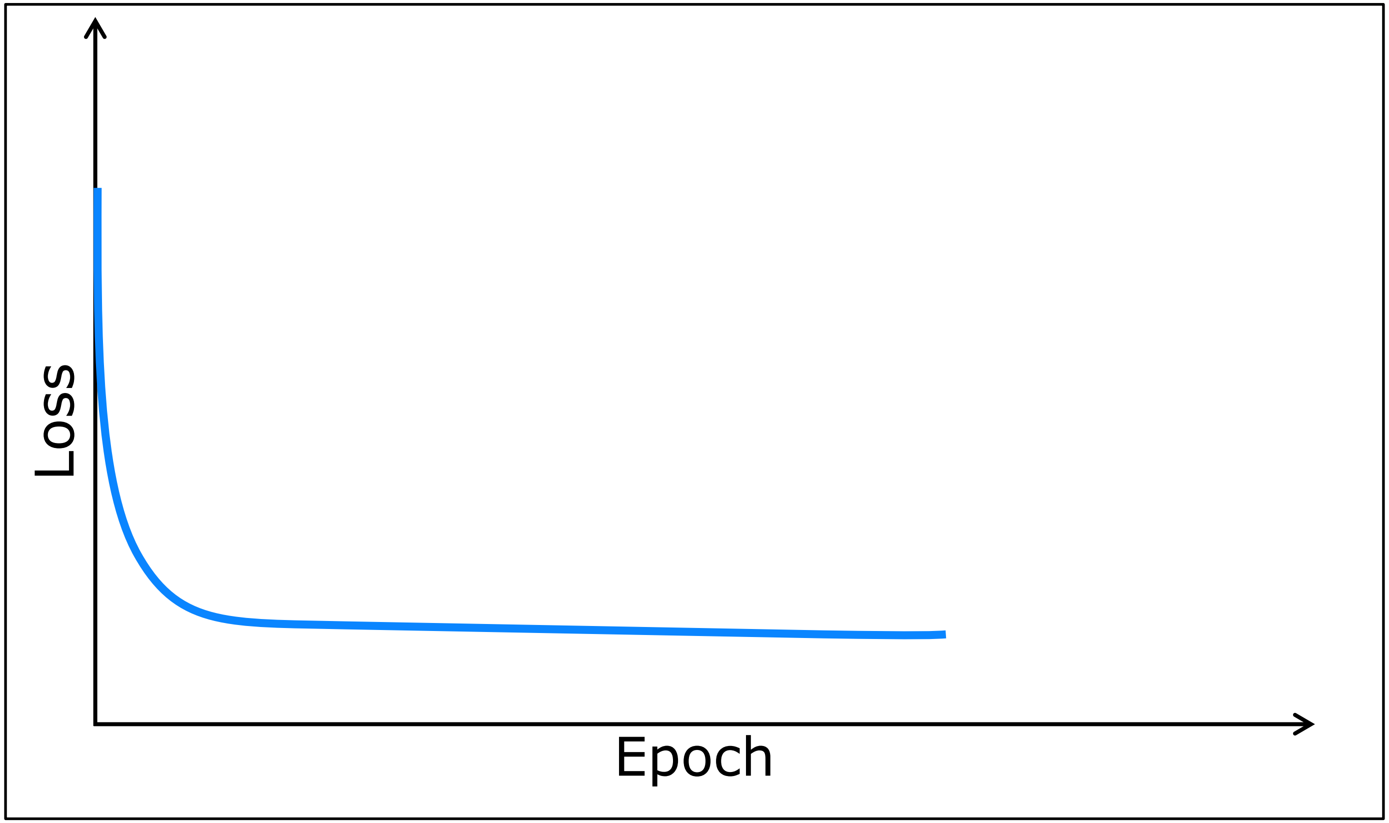
Figure from Justin Johnson's lecture.

Initial Learning Rate



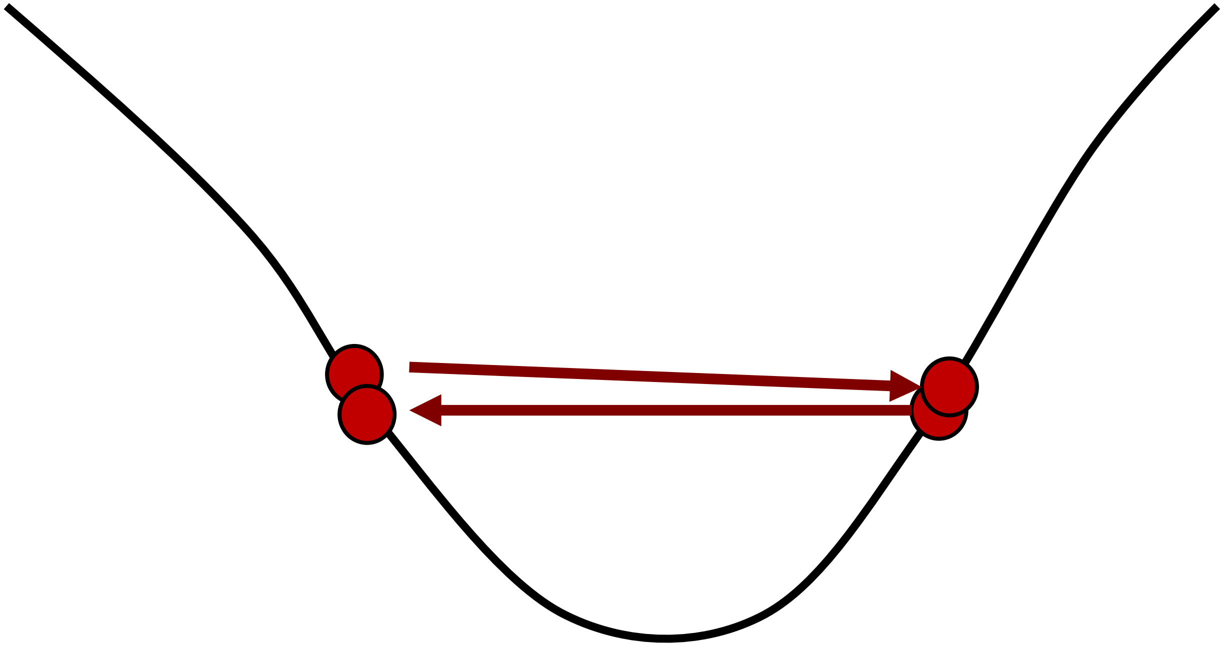
- Want to find learning rate that decreases loss in the beginning quickly
- Usual Receipt: Coarse search {0.1, 0.01, 0.001, 0.0001} and then more fine with best learning rate η_0, e.g. 0.8η_0, 0.9η_0, …, 2η_0

But...



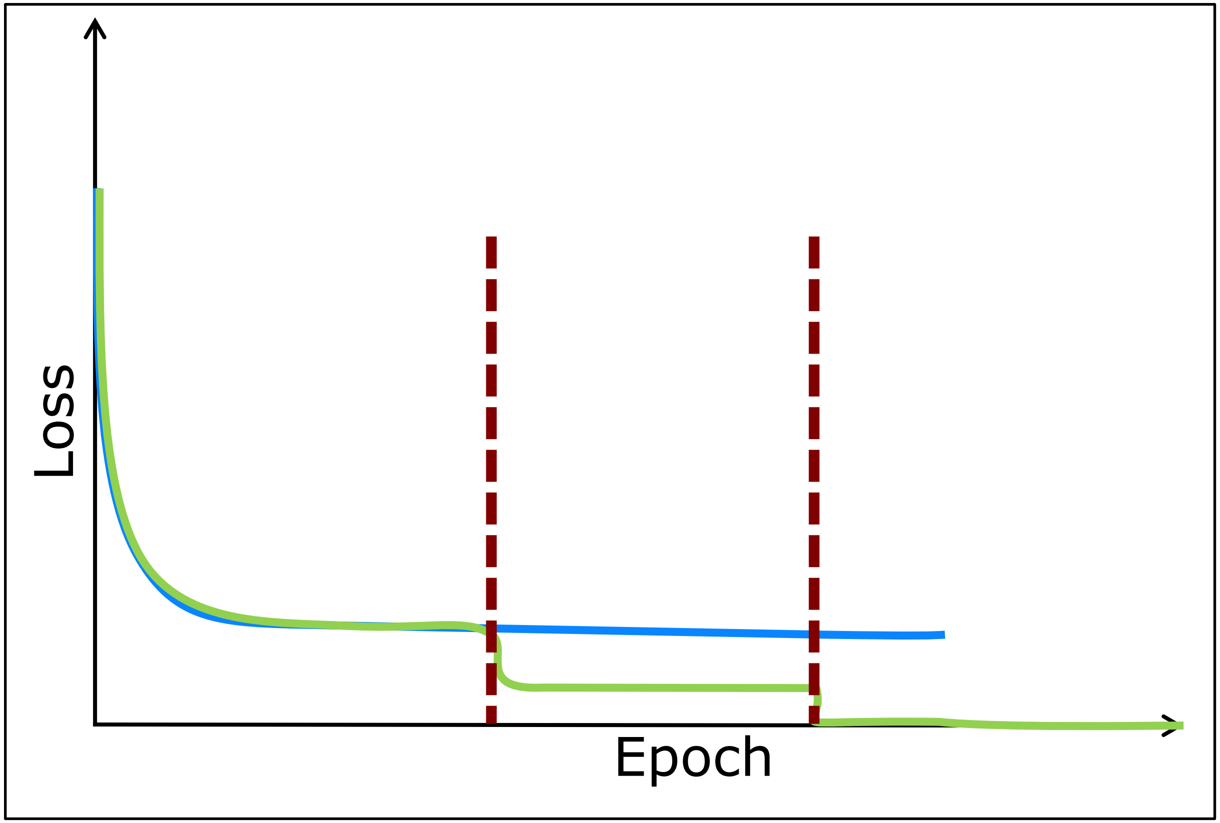
- Learning stalls after some epochs (= run through all images from the training set)

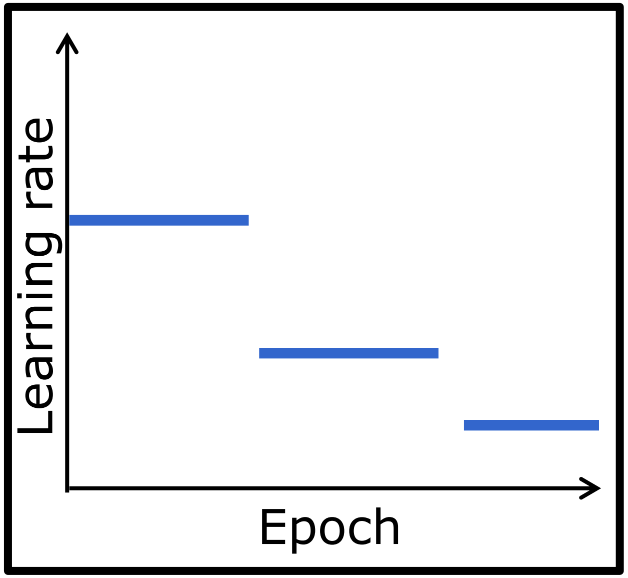
Possible Explanation



- Learning is not making progress as you are "dancing" around the local minimum
- What to do?

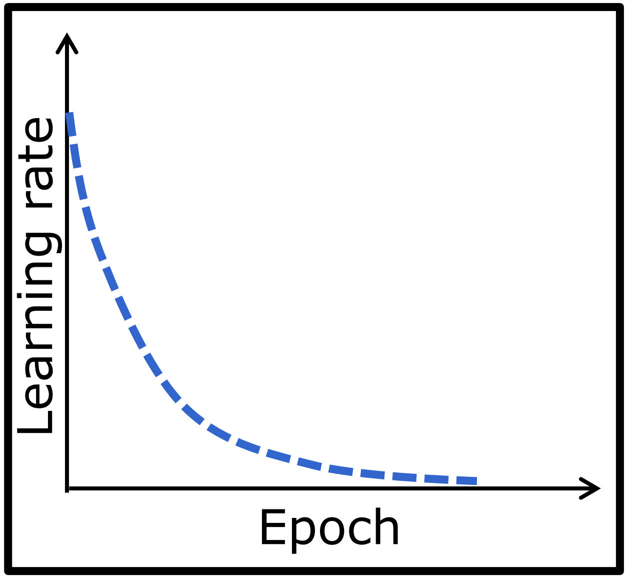
Step LR schedule





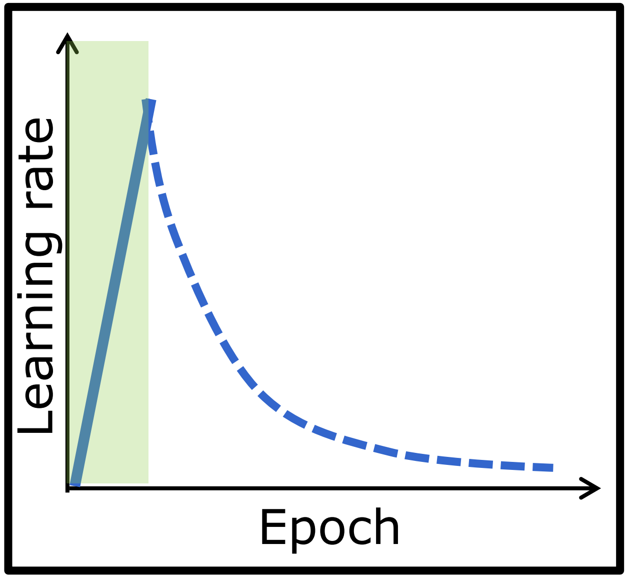
- Decrease learning rate by factor 10 after certain number epochs

Exponential Decaying LR



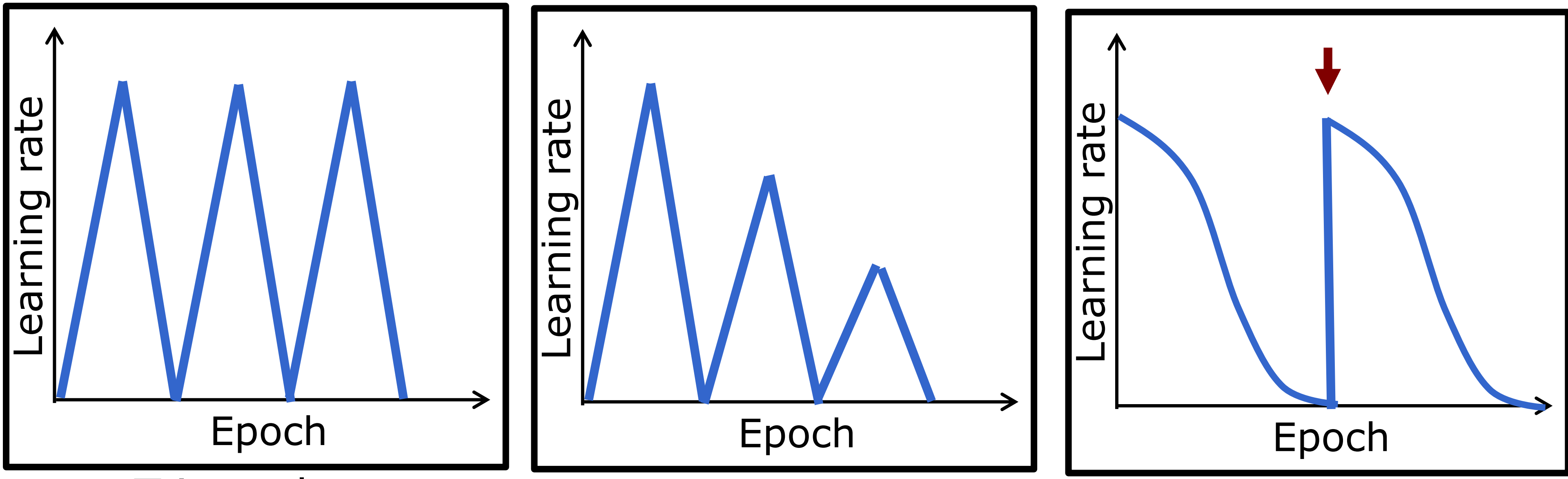
- Decay learning rate

Warm up



- In the beginning gradients noise and badly initialized network
- Start slow, increase until desired LR, then normal schedule.

Cyclic LR Schedule

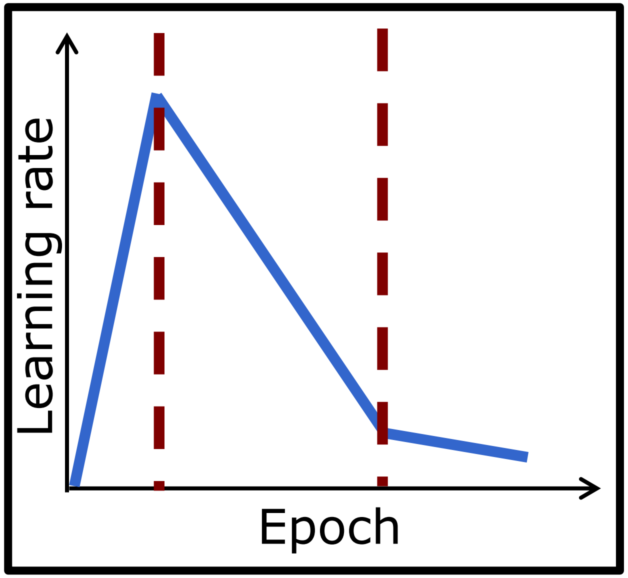


Triangular
Triangular Decay
Cosine with Restarts

- Also cyclic LR schedule sometimes advantageous
- Combination of decay + cyclic

[Smith, 2017][Loshchilov , 2016]

One-Cycle LR Schedule



- Instead of multiple cycles, in certain cases a single cycle (with good steps) showed faster convergence

[Smith et al. 2019]

LR Schedules in PyTorch

CLASS `torch.optim.lr_scheduler.StepLR(optimizer, step_size, gamma=0.1, last_epoch=-1, verbose=False)` [SOURCE]

CLASS `torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max, eta_min=0, last_epoch=-1, verbose=False)` [SOURCE]

CLASS `torch.optim.lr_scheduler.CyclicLR(optimizer, base_lr, max_lr, step_size_up=2000, step_size_down=None, mode='triangular', gamma=1.0, scale_fn=None, scale_mode='cycle', cycle_momentum=True, base_momentum=0.8, max_momentum=0.9, last_epoch=-1, verbose=False)` [SOURCE]

CLASS `torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, total_steps=None, epochs=None, steps_per_epoch=None, pct_start=0.3, anneal_strategy='cos', cycle_momentum=True, base_momentum=0.85, max_momentum=0.95, div_factor=25.0, final_div_factor=10000.0, three_phase=False, last_epoch=-1, verbose=False)` [SOURCE]

- PyTorch has already variants implemented.

Hyperparameters are a function of the Dataset

- While tuning parameters on a small subset of your data, these parameters (usually) don’t transfer to the full training set
- Even receipts that work for one dataset (especially tasks) don’t work so well for others!
- It’s mostly experience, trying things, having a gut feeling, …
- Therefore: Try things, break things, gather experience. (… that’s why often PhD students are hired by companies, they starred a lot a curves and have a bag of knowledge already acquired.)

Learn from others

3.4. Implementation

Our implementation for ImageNet follows the practice in [21, 41]. The image is resized with its shorter side randomly sampled in $[256, 480]$ for scale augmentation [41]. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted [21]. The standard color augmentation in [21] is used. We adopt batch normalization (BN) [16] right after each convolution and before activation, following [16]. We initialize the weights as in [13] and train all plain/residual nets from scratch. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60×10^4 iterations. We use a weight decay of 0.0001 and a momentum of 0.9. We do not use dropout [14], following the practice in [16].

In testing, for comparison studies we adopt the standard 10-crop testing [21]. For best results, we adopt the fully-convolutional form as in [41, 13], and average the scores at multiple scales (images are resized such that the shorter side is in $\{224, 256, 384, 480, 640\}$).

Paper learning on ImageNet

3.1. Implementation Details

We set hyper-parameters following existing Fast/Faster R-CNN work [12, 36, 27]. Although these decisions were made for object detection in original papers [12, 36, 27], we found our instance segmentation system is robust to them.

Training: As in Fast R-CNN, an RoI is considered positive if it has IoU with a ground-truth box of at least 0.5 and negative otherwise. The mask loss L_{mask} is defined only on positive RoIs. The mask target is the intersection between an RoI and its associated ground-truth mask.

We adopt image-centric training [12]. Images are resized such that their scale (shorter edge) is 800 pixels [27]. Each mini-batch has 2 images per GPU and each image has N sampled RoIs, with a ratio of 1:3 of positive to negatives [12]. N is 64 for the C4 backbone (as in [12, 36]) and 512 for FPN (as in [27]). We train on 8 GPUs (so effective mini-batch size is 16) for 160k iterations, with a learning rate of 0.02 which is decreased by 10 at the 120k iteration. We use a weight decay of 0.0001 and momentum of 0.9. With ResNeXt [45], we train with 1 image per GPU and the same number of iterations, with a starting learning rate of 0.01.

The RPN anchors span 5 scales and 3 aspect ratios, following [27]. For convenient ablation, RPN is trained separately and does not share features with Mask R-CNN, unless specified. For every entry in this paper, RPN and Mask R-CNN have the same backbones and so they are shareable.

Paper learning on MS COCO (Detection)

- Certain datasets have very refined training schedules that are historically grown
- Copy successful schemes

References on LR Schedules

- Cosine LR: Loshchilov et al. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv, 2016. https://arxiv.org/pdf/1608.03983.pdf
- Smith. Cyclical Learning Rates for Training Neural Networks, WACV, 2017. https://arxiv.org/pdf/1506.01186.pdf
- Smith. A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay, arxiv, 2018. https://arxiv.org/pdf/1803.09820.pdf
- Smith & Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, arxiv, 2019.

Still a "hot" topic

Published as a conference paper at ICLR 2020

AN EXPONENTIAL LEARNING RATE SCHEDULE FOR DEEP LEARNING

Zhiyuan Li
Princeton University
zhiyuanli@cs.princeton.edu

Sanjeev Arora
Princeton University and Institute for Advanced Study
arora@cs.princeton.edu

Thus the final training algorithm is roughly as follows: *Start from a convenient LR like* 0.1, *and grow it at an exponential rate with a suitable exponent. When validation loss plateaus, switch to an exponential growth of LR with a lower exponent. Repeat the procedure until the training loss saturates.*

- Recent work suggest that a network with batch normalization or weight decay can be trained with exponentially **increasing** learning rate to reach a global minimum

See you next week!