Course Notes: Deep Learning for Visual Computing

Peter Wonka

November 8, 2023

Contents

1 Backpropagation

1.1 Derivative

- Input:
  - function $f(x) : \mathbb{R} \to \mathbb{R}$,
  - $y = f(x)$
- Derivative:

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

- Example:

$$f(x) = 2x^2, \quad \frac{df(x)}{dx} = 4x$$

- Interpretation:
  How does the output $f(x)$ change when adding a small value to the input x
- Notation: wiki
- **Leibniz notation**:

- Minimal version

$$\frac{dy}{dx}, \frac{df}{dx}$$

- Making the input variable explicit

$$\frac{df(x)}{dx}, \frac{df}{dx}(x), \frac{d}{dx}f(x)$$

- Derivative at a point

$$\frac{dy}{dx}\Big|_{x=a}, \frac{dy}{dx}(a), \frac{d}{dx}f(a), \ldots$$

- **Lagrange notation**: $f'(x)$
- **Euler notation**: D is an operator mapping a function to another function

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

1.2 Chain Rule

- $h(x) = g \circ f = g(f(x))$
  - $y = g(u), u = f(x)$
- Chain rule (Leibniz):

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

- Chain rule (Lagrange):

$$h'(x) = g'(f(x))f'(x)$$

1.3 Partial Derivatives

- Literature: Boyd Course Notes
- Input:
  - function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$
  - $f(\mathbf{x}) = f(x_1, \ldots, x_n) = \ldots$
- Partial derivatives:

$$\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n},$$

- Example:

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

-

1.4 Derivatives with Multiple Dimensions

- functions that have multiple inputs and multiple outputs
- $f : \mathbb{R}^n \to \mathbb{R}^m$
- Derivative: $Df(x) \in \mathbb{R}^{m \times n}$:

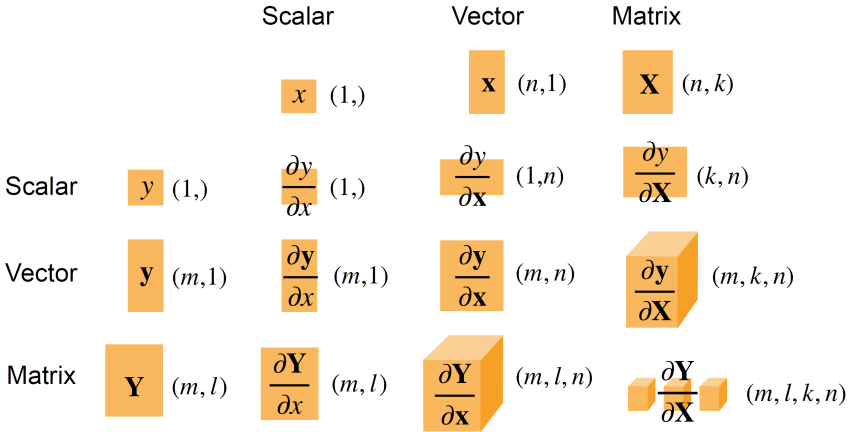
$$(Df(x))_{ij} = \frac{\partial f_i}{\partial x_j}\bigg|_x, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

1.5 Notational Conventions

- Warning: there are competing conventions for the layout
- **numerator-layout**:
  - Derivate of a function $\mathbb{R}^n \to \mathbb{R}$ is a row vector (even if the input is a column vector)
  - Derivative of a function $\mathbb{R} \to \mathbb{R}^n$ is a column vector
- **denominator-layout**:
  - Transpose of numerator-layout
- **mixed forms**
  - can be confusing

1.6 Numerator Layout

- Inputs: $x, \mathbf{x}, X$; Outputs: $y, \mathbf{y}, Y$



1.7 Practical Considerations

- **Graph drawing**: derivatives travel on graph edges as tensors(scalars, vectors, matrices) of the same shape as data traveling on an edge
- **Implementation**: derivatives (during backpropagation) should be stored in the same tensor data structure as the data

1.8 Examples

- Numerator-layout
- **Linear layer**, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, n inputs and m outputs:

$$f(\mathbf{x}) = \boldsymbol{A}\mathbf{x}, \quad Df(\mathbf{x}) = \boldsymbol{A} \tag{1.1}$$

$$\mathbf{y} = \boldsymbol{A}\mathbf{x}, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \boldsymbol{A} \tag{1.2}$$

  - $A(i, j)$: How much does output i depend on input j?
- **Squared-norm**:

$$f(\mathbf{x}) = \|\mathbf{x}\|_2^2, \quad Df(\mathbf{x}) = 2\mathbf{x}^T \tag{1.3}$$

- **ReLU**: (a condition evaluates to 1 if true)

$$f(\mathbf{x}) = ReLU(\mathbf{x}), \quad Df(\mathbf{x}) = \text{diag}(x_1 \geq 0, \ldots, x_n \geq 0) \tag{1.4}$$

- **Mean**: (usefule for normalization)

$$\mu(\mathbf{x}) = f(\mathbf{x}) = \frac{1}{N}\sum_i x_i, \quad Df(\mathbf{x}) = (\frac{1}{N}, \ldots, \frac{1}{N}) \tag{1.5}$$

- **Variance**:

$$f(\mathbf{x}) = \frac{1}{N}\sum_i (x_i - \mu(\mathbf{x}))^2, \quad Df(\mathbf{x}) = \frac{2}{N}(\mathbf{x}^T - \mu \mathbb{1}_n^T) \tag{1.6}$$

  - $\mathbb{1}_n$ is a column vector of n 1s

$$\frac{\partial}{\partial x_i}\frac{1}{N}\sum_j (x_j - \mu)^2 = \frac{1}{N}\sum_j 2(x_j - \mu)\frac{\partial (x_j - \mu)}{\partial x_i} \tag{1.7}$$

$$= -\frac{1}{N^2}\sum_j 2(x_j - \mu) + \frac{1}{N}2(x_i - \mu) \tag{1.8}$$

$$= \frac{2(x_i - \mu)}{N} \tag{1.9}$$

- Example:

$$f(\mathbf{x}) = \begin{pmatrix} x_1 - x_2^2 \\ x_1 x_3 \end{pmatrix}, \quad Df(\mathbf{x}) = \begin{pmatrix} 1 & -2x_2 & 0 \\ x_3 & 0 & x_1 \end{pmatrix}$$

1.9 Chain Rule with Multiple Dimensions

- Given $g : \mathbb{R}^m \to \mathbb{R}^k$, $f : \mathbb{R}^n \to \mathbb{R}^m$, and $\mathbf{a} \in \mathbb{R}^n$

$$D(g \circ f)(\mathbf{a}) = (Dg)(f(\mathbf{a})) \circ (Df)(\mathbf{a})$$

- Jacobian notation:

$$J_{g \circ f} = J_g(f(\mathbf{a}))J_f(\mathbf{a})$$

1.10 Gradient

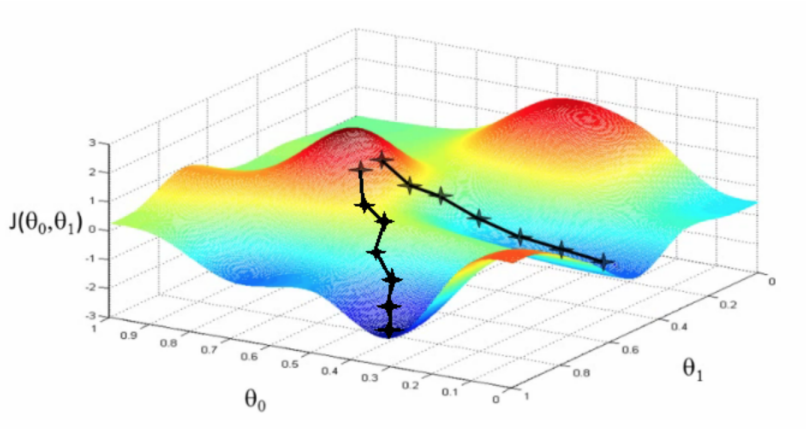
- function with multiple inputs and one output
- Gradient: vector of partial derivatives
  - Notation: $\nabla f(x)$
  - Example:

$$f(x, y) = xy, \quad \nabla f = \begin{pmatrix} y \\ x \end{pmatrix}$$

  - $\nabla f = (Df)^T$
    - **Derivative is a row vector and gradient a column vector**
    - Often, gradient and derivative are treated the same
  - Interpretation:
    - You are at a point on an n-dimensional mountain. The gradient points into the direction of the steepest ascent.

- You are at at point in an n-dimensional room. The (temperature) gradient points in the direction of maximum temperature increase

- Optimization and Visualization:
  - We want to iteratively go in the negative gradient direction (gradient descent)


J(θ0,θ1)
θ0
θ1

1.11 Jacobian

- **Jacobian**: derivative in matrix form for a function with multiple inputs and multiple outputs

1.12 Automatic Differentiation

- Terms: **automatic differentiation**, **AD**, **algorithmic differentiation**, **computational differentiation**, **auto-differentiation**, **autodiff**
- set of techniques to numerically evaluate the derivative of a function specified by a computer program (at a particular point)
- **Symbolic differentiation:** e.g. Mathematica, generate symbolic expressions for the derivative
- **Numerical differentiation** using finite differences
  - perturb the input by some δ and see how the output changes

1.13 Forward vs. Backward

- Motivating example:

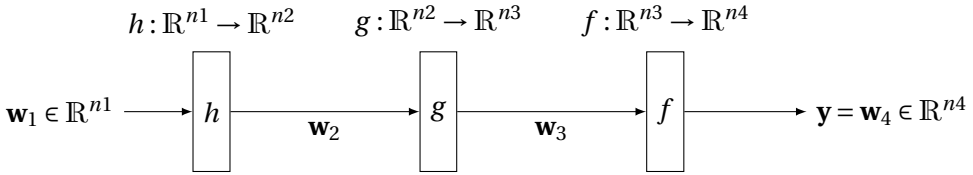
$$\mathbf{y} = f(g(h(\mathbf{x}))) \tag{1.10}$$

$$\mathbf{w}_1 = \mathbf{x} \in \mathbb{R}^{n1} \tag{1.11}$$

$$\mathbf{w}_2 = h(\mathbf{w}_1) \tag{1.12}$$

$$\mathbf{w}_3 = g(\mathbf{w}_2) \tag{1.13}$$

$$y = \mathbf{w}_4 = f(\mathbf{w}_3) \in \mathbb{R}^{n4} \tag{1.14}$$



$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1} = \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_3}\frac{\partial \mathbf{w}_3}{\partial \mathbf{w}_2}\frac{\partial \mathbf{w}_2}{\partial \mathbf{w}_1} = \frac{\partial f(\mathbf{w}_3)}{\partial \mathbf{w}_3}\frac{\partial g(\mathbf{w}_2)}{\partial \mathbf{w}_2}\frac{\partial h(\mathbf{w}_1)}{\partial \mathbf{w}_1} \tag{1.15}$$

- Derivatives are associate; We can multiply from the beginning or the end

- Derivatives can be large
- Example: large conv layer
  - $\mathbf{w}_1$ is a tensor $1024 \times 1024 \times 3$
  - $\mathbf{w}_2$ is a tensor $1024 \times 1024 \times 64$
  - Need $1024^4 \times 3 \times 64$ scalar derivatives
  - Not possible to store the derivative of a single layer explicitly

- **Forward accumulation:**
  - proceed right to left in the **equation**; $i = max$
  - proceed left to right in the **graph**

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{x}} = \frac{\partial \mathbf{w}_i}{\partial \mathbf{w}_{i-1}} \frac{\partial \mathbf{w}_{i-1}}{\partial \mathbf{x}} \tag{1.16}$$

- **Backward accumulation:**
  - proceed left to right in the **equation**; $i = 1$
  - proceed right to left in the **graph**

$$\frac{\partial \mathbf{y}}{\partial \mathbf{w}_i} = \frac{\partial \mathbf{y}}{\partial \mathbf{w}_{i+1}} \frac{\partial \mathbf{w}_{i+1}}{\partial \mathbf{w}_i} \tag{1.17}$$

- Observation:
  - In NN the loss function only has 1 output, e.g. $n4 = 1$, therefore backward accumulation works great
  - Only need to store derivatives / gradients of the same size as the data on edges in the graph
  - Disadvantage: need to store a lot of intermediate outputs, e.g. all outputs of all nodes for all samples in the batch
  - For different functions, e.g. with a single input so that $n1 = 1$, forward accumulation is great

1.14 Stanford Course Notation

- partial derivatives with respect to a vector or matrix:

$$\frac{\partial L}{\partial \mathbf{x}}, \quad \frac{\partial L}{\partial \boldsymbol{W}}$$

- gradient symbol, derivative symbol. Jacobian symbol mixed
- input vectors are sometimes shaped as matrices, derivative is computed, and derivative is reshaped in the same matrix form
  - works because loss L is only a scalar function

1.15 Gradient in Neural Networks

- A neural network computes a function: $f : \mathbb{R}^n \to \mathbb{R}^m$
  - Note, other choices besides $\mathbb{R}$ are possible for input and output
  - For example outputs could be m class probabilities
- Network function depends on input samples (images) as well as weights
- Example Notation:

$$f(\mathbf{x}), \quad f(\mathbf{x}; \mathbf{w}), \quad f(\mathbf{x}; \boldsymbol{W}), \quad f(\mathbf{x}; \boldsymbol{W}, \mathbf{b}), \quad f_{\mathbf{w}}(\mathbf{x})$$

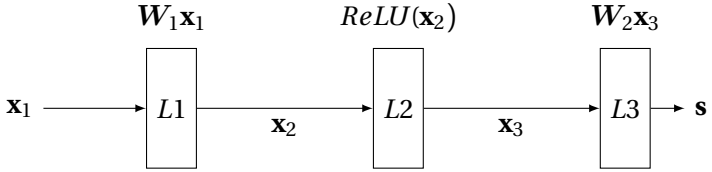
- We can compute the gradient with respect to the weights $\boldsymbol{W}$, input samples $\mathbf{x}$, or both
  - gradient with respect to weights is important for the core learning algorithms
- Example Notation:

$$\nabla f(\mathbf{x}; \mathbf{w}), \quad \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w}), \quad \nabla_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{W}, \mathbf{b})$$

  - Subscript denotes what variables the gradient is computed for

- We need a loss function: $L : \mathbb{R}^m \to \mathbb{R}$
  - Note: **only one ouput**
    - This is key for backpropagation!
    - Gradients / derivatives of a tensor $\boldsymbol{X}$ with respect to a scalar loss L only have as many elements as $\boldsymbol{X}$
  - Combines all goals into a single value
  - Loss can consider the final network output, the network weights, side branches, ...

1.16 Example Network



- Network with linear layer, ReLU, linear layer:

$$s = f(\mathbf{x}; \boldsymbol{W}_1, \boldsymbol{W}_2) = \boldsymbol{W}_2 \max(0, \boldsymbol{W}_1 \mathbf{x})$$

- weight decay to regularize weights

$$R(\boldsymbol{W}_1, \boldsymbol{W}_2) = R(\boldsymbol{W}) = \sum_k W_k^2$$

  - W_k is one weigth, k iterates over all weights

- Data loss function (SVM loss):

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

- Total loss = data loss + regularization

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

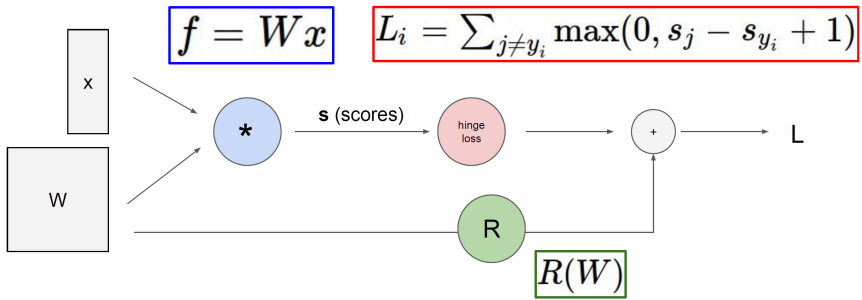
- To learn / optimize the weights $\boldsymbol{W}$ we need:

$$\frac{\partial L}{\partial \boldsymbol{W}_1}, \frac{\partial L}{\partial \boldsymbol{W}_2}$$

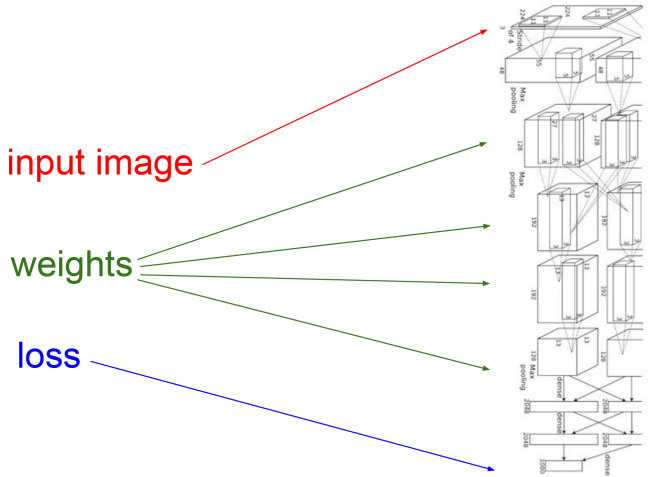
1.17 Problems and Solution

- Gradient $\nabla_{\boldsymbol{W}} L$ is difficult / tedious to compute by hand
  - Hard to change components of the network if gradient has to be recomputed / recoded every time.
  - e.g. change the loss function, change the regularization, add another class, add another layer
- Better Idea: Computational Graphs + Backpropagation

- Simple example network

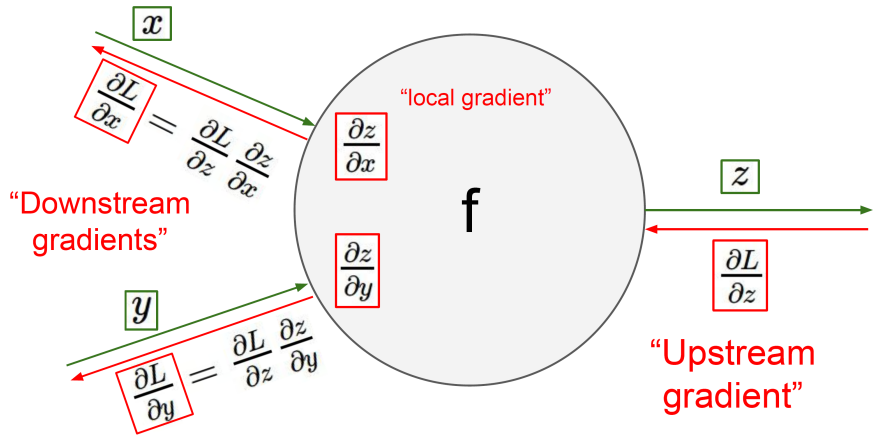


- Example Network: AlexNet



1.18 Backpropagation

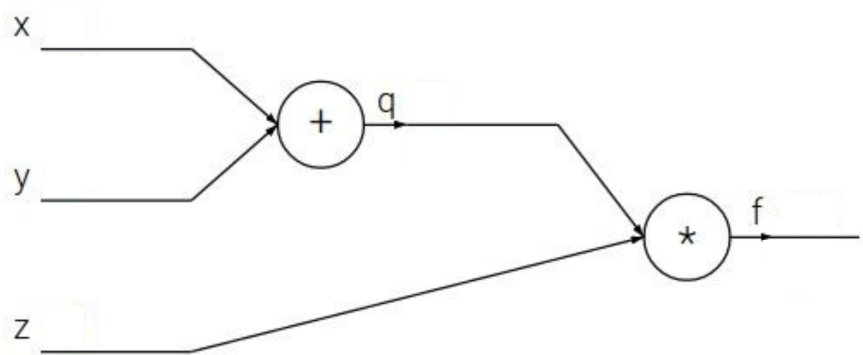
- Visualization:



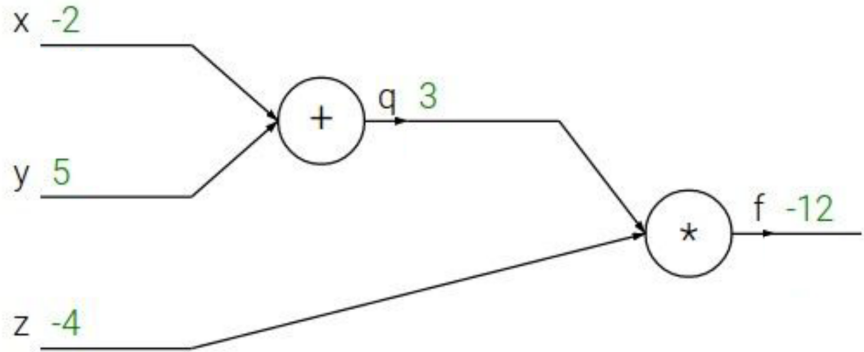
- Chain Rule:
  - **downstream gradient = upstream gradient × local gradient**
- Each edge is associated with a variable:
  - inputs $\mathbf{x}$ from a sample
  - network weights $\mathbf{w}$ (parameters)
  - computation of an intermediate node
- Forward pass:
  - Compute the values of intermediate nodes for each edge
- Backward pass:
  - compute local gradients for each node
  - compute a gradient for each edge using chain rule
  - if edge has variable v in forward pass, the backward pass computes $\frac{\partial L}{\partial v}$

1.19 Simple Backpropagation Example P1

- Example function: $f(x, y, z) = (x + y)z$
- Example function as graph:
  - intermediate function: $q = x + y$



1.20 Simple Backpropagation Example P2



- Example inputs:
  - $x = -2$, $y = 5$, $z = -4$ (shown in green)
- Forward Computation and Symbolic Local Gradients:

- q-node

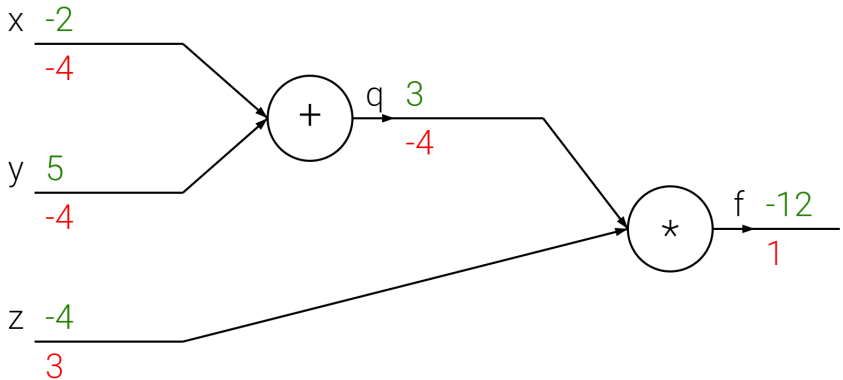
$$q = x + y = (-2) + 5 = 3, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

- f-node

$$f = qz = 3(-4) = -12, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

1.21 Simple Backpropagation Example P3

- Solution Visualization:
  - gradients shown in red on edges



- f-edge (initialization):
  - 1 is sent back on the last edge

$$\frac{\partial f}{\partial f} = 1$$

- f-node: $f = qz$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial q} = 1 \times z = -4, \quad \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = 1 \times q = 3$$

  - Note: we need to save q from forward pass
- q-node: $q = x + y$

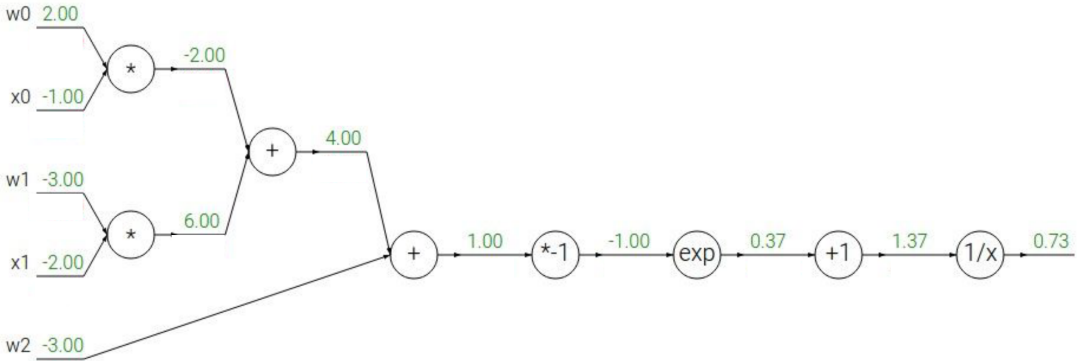
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = -4 \times 1 = -4, \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = -1 \times 1 = -4$$

1.22 Second Backpropagation Example P1

- Function:

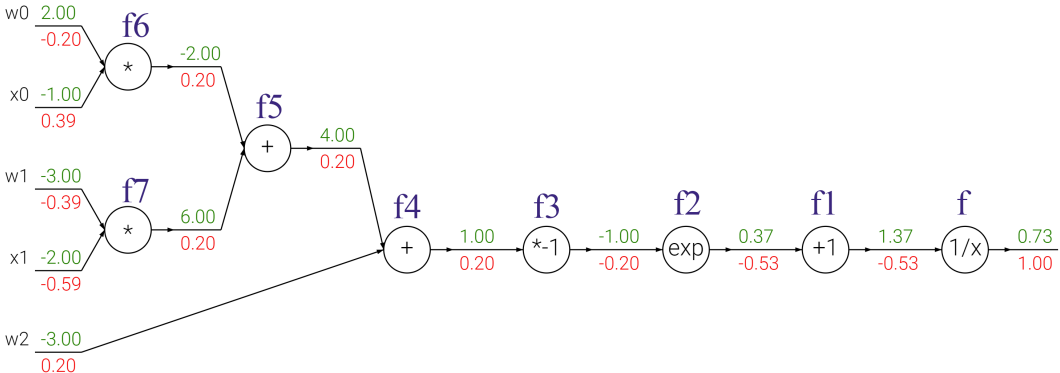
$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

- Forward Computation:
  - Assume $w_0 = 2, x_0 = -1, w_1 = -3, x_1 = -2, w_2 = -3$


w0 2.00
x0 -1.00
w1 -3.00
x1 -2.00
w2 -3.00
*
*
+
+
*-1
exp
+1
1/x
-2.00
6.00
4.00
1.00
-1.00
0.37
1.37
0.73

1.23 Second Backpropagation Example P2

- Backwards Computation:



- Initialization:

$$\frac{df}{df} = 1$$

- f-node:

$$f(f_1) = \frac{1}{f_1}, \quad \frac{df}{df_1} = \frac{df}{df}\frac{df}{df_1} = 1 \times \frac{-1}{f_1^2} = 1 \times \frac{-1}{1.37^2} = -0.53$$

- f_1-node:

$$f_1(f_2) = f_2 + 1, \quad \frac{df}{df_2} = \frac{df}{df_1}\frac{df_1}{df_2} = -0.53 \times 1 = -0.53$$

- f_2-node:

$$f_2(f_3) = e^{f_3}, \quad \frac{df}{df_3} = \frac{df}{df_2}\frac{df_2}{df_3} = -0.53 \times e^{f_3} = -0.53 \times e^{-1} = -0.2$$

- f_3-node:

$$f_3(f_4) = -f_4, \quad \frac{df}{df_4} = \frac{df}{df_3}\frac{df_3}{df_4} = -0.2 \times -1 = 0.2$$

- f_4-node:

$$f_4(f_5, w_2) = f_5 + w_2 \tag{1.18}$$

$$\frac{\partial f}{\partial w_2} = \frac{df}{df_4}\frac{\partial f_4}{\partial w_2} = 0.2 \times 1 = 0.2 \tag{1.19}$$

$$\frac{\partial f}{\partial f_5} = \frac{df}{df_4}\frac{\partial f_4}{\partial f_5} = 0.2 \times 1 = 0.2 \tag{1.20}$$

$$\tag{1.21}$$

- f_5-node:

$$f_5(f_6, f_7) = f_6 + f_7 \tag{1.22}$$

$$\frac{\partial f}{\partial f_6} = \frac{\partial f}{\partial f_5}\frac{\partial f_5}{\partial f_6} = 0.2 \times 1 = 0.2 \tag{1.23}$$

$$\frac{\partial f}{\partial f_7} = \frac{\partial f}{\partial f_5}\frac{\partial f_5}{\partial f_7} = 0.2 \times 1 = 0.2 \tag{1.24}$$

$$\tag{1.25}$$

- f_6-node:

$$f_6(w_0, x_0) = w_0 x_0 \tag{1.26}$$

$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial f_6}\frac{\partial f_6}{\partial w_0} = 0.2 \times x_0 = 0.2 \times -1 = -0.2 \tag{1.27}$$

$$\frac{\partial f}{\partial x_0} = \frac{\partial f}{\partial f_6}\frac{\partial f_6}{\partial x_0} = 0.2 \times w_0 = 0.2 \times 2 = 0.39 \tag{1.28}$$

$$\tag{1.29}$$

- f_7-node:

$$f_7(w_1, x_1) = w_1 x_1 \tag{1.30}$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial f_7}\frac{\partial f_7}{\partial w_1} = 0.2 \times x_1 = 0.2 \times -2 = -0.39 \tag{1.31}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial f_7}\frac{\partial f_7}{\partial x_1} = 0.2 \times w_1 = 0.2 \times -3 = -0.59 \tag{1.32}$$

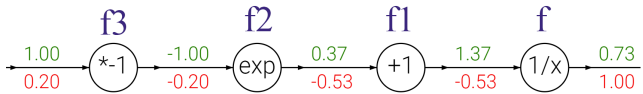
$$\tag{1.33}$$

1.24 Simplification of Sigmoid

- Computational graph representation is not unique
- Choose a graph where local gradients are easy to express
- Sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.34}$$

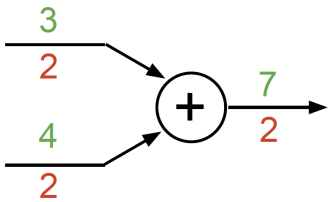
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x) \tag{1.35}$$



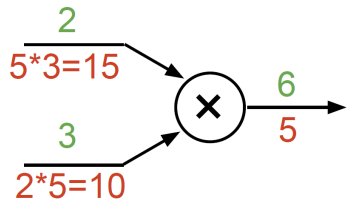
- Verification: $(1-0.73)0.73 = 0.2$

1.25 Patterns in Gradient Flow

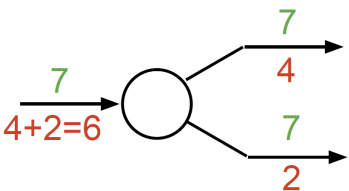
add gate: gradient distributor



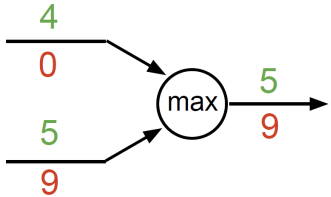
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router

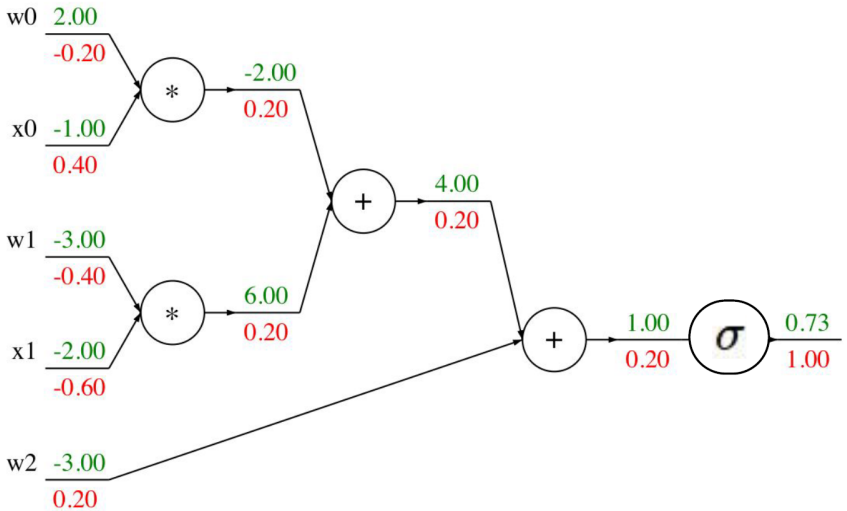


- add gate - gradient distributor: routes the gradient to all inputs
- multiply gate - swap multiplier: upstream gradient * other input value
- copy gate - gradient adder: sum over all upstream gradients
- max gate - gradient router: copy upstream gradient to the input edge that got selected as max, other inputs are 0

$$f = max(x_1, x_2), \quad \frac{\partial f}{\partial x_1} = [x_1 \geq x_2], \quad \frac{\partial f}{\partial x_2} = [x_2 \geq x_1]$$

  - We use [] to denote the Iverson bracket. It evaluates to 1 if the condition is true and 0 otherwise.

1.26 Code Example

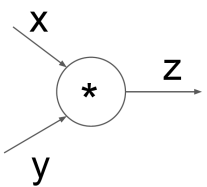


```
# Forward Pass: compute output
s0 = w0 * x0
s1 = w1 * x1
s2 = s0 + s1
s3 = s2 + w2
L = sigmoid(s3)

# Backward Pass: compute gradients
grad_L = 1.0

# Sigmoid
grad_s3 = grad_L * (1-L) - L

# Add gate
grad_w2 = grad_s3
grad_s2 = grad_s3

# Add gate
grad_s0 = grad_s2
```

```
20 grad_s1 = grad_s2
21
22 # Multiply gate
23 grad_w1 = grad_s1 * x1
24 grad_x1 = grad_s1 * w1
25
26 # Multiply gate
27 grad_w0 = grad_s0 * x0
28 grad_x0 = grad_s0 * w0
```

1.27 Structured Code Example Multiply

- Better to structure code
  - separate code for forward and backward pass



(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

1.28 Structured Code Example Sigmoid

```
1
2  #ifndef TH_GENERIC_FILE
3  #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
4  #else
5
6  void THNN_(Sigmoid_updateOutput)(
7            THNNState *state,
8            THTensor *input,
9            THTensor *output)
10 {
11   THTensor_(sigmoid)(output, input);
12 }
13
14 void THNN_(Sigmoid_updateGradInput)(
15           THNNState *state,
16           THTensor *gradOutput,
17           THTensor *gradInput,
18           THTensor *output)
```

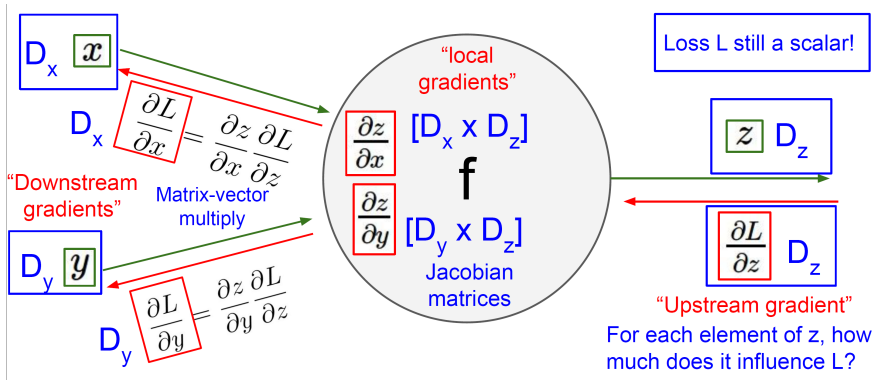
```
19 {
20   THNN_CHECK_NELEMENT(output, gradOutput);
21   THTensor_(resizeAs)(gradInput, output);
22   TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t,
         output,
23     scalar_t z = *output_data;
24     *gradInput_data = *gradOutput_data * (1. - z) * z;
25   );
26 }
27
28 #endif
```

1.29 Extending to multiple outputs

- $f : \mathbb{R} \rightarrow \mathbb{R}$
  - one input, one output → derivative is scalar
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$
  - many inputs, one output, → derivative is vector (gradient)
- $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
  - many inputs, many outputs, → derivative is matrix (Jacobian)
  - Example: $f : \mathbf{x} \rightarrow \mathbf{u}$

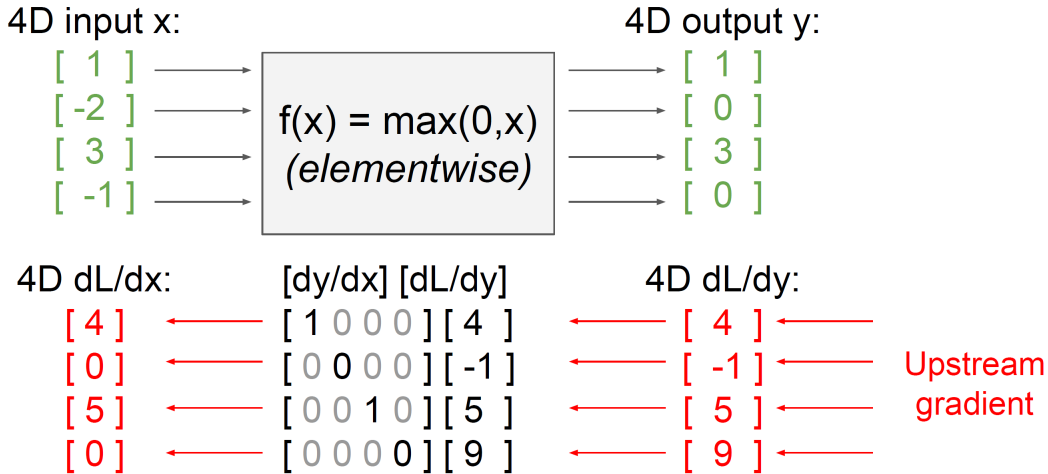
$$\begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} & \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} \end{bmatrix} \tag{1.36}$$

1.30 Vector Backpropagation



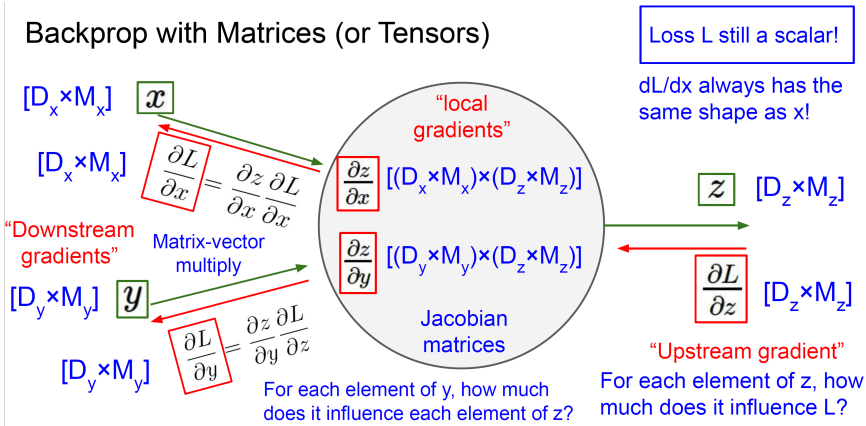
- Vector dimensions are specified by D_z, D_y, D_x
- Jacobians are of sizes $D_x \times D_z$ and $D_y \times D_z$

1.31 Vector Backpropagation Example



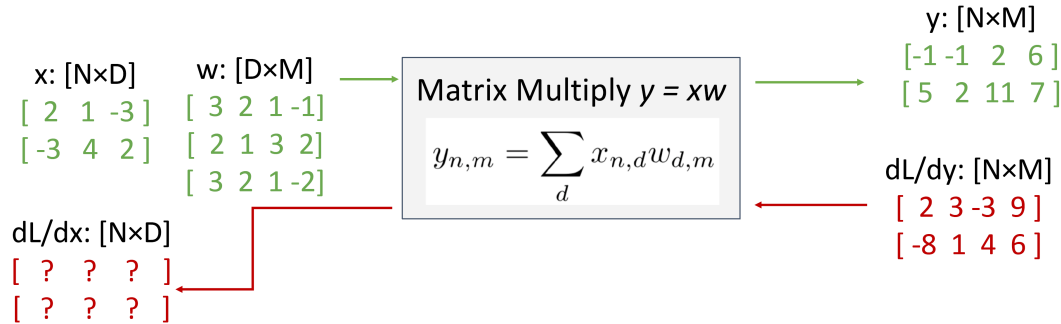
- Jacobian is sparse; too costly to define the complete Jacobian in code

1.32 Matrix Backpropagation



- Matrix dimensions are specified by $D_z \times M_z, D_y \times M_y, D_x \times M_x$
- Derivatives are of sizes $(D_x \times M_x) \times (D_z \times M_z)$ and $(D_y \times M_y) \times (D_z \times M_z)$

1.33 Matrix Backpropagation Example



- Derivative dy/dx has size $(N \times D) \times (N \times M)$
- Derivative dy/dw has size $(D \times M) \times (N \times M)$
- Network might have $N = 64, D = M = 4096$ - would take 256GB of memory for one Derivative if constructed explicitely

- Q: What parts of $\mathbf{y}$ are affected by one element of $\mathbf{x}$
- A: $x_{n,d}$ affects the whole row $\mathbf{y}_{n,:}$

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

- Q: How much does $x_{n,d}$ affect $y_{n,m}$?
- A: $w_{d,m}$
  - Example: $x_{1,2} = 1$ affects $y_{1,3} = 2$ with a factor of $w_{2,3} = 3$
- Backpropagation computation:
  - Equations are easy to remember: there is only one way to make shapes match up

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \boldsymbol{w}^T \tag{1.37}$$

$$\frac{\partial L}{\partial x} \in \mathbb{R}^{N \times D}, \quad \frac{\partial L}{\partial y} \in \mathbb{R}^{N \times M}, \quad \boldsymbol{w}^T \in \mathbb{R}^{M \times D} \tag{1.38}$$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m} \tag{1.39}$$

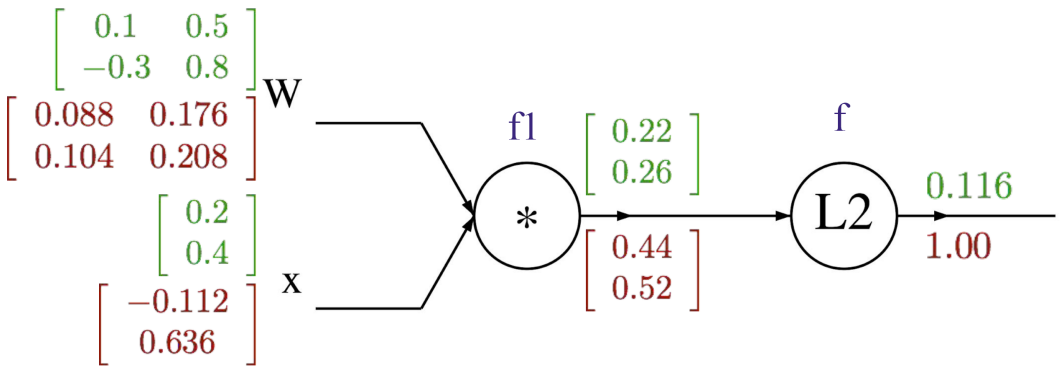
$$\frac{\partial L}{\partial w} = \boldsymbol{x}^T \frac{\partial L}{\partial y} \tag{1.40}$$

$$\frac{\partial L}{\partial w} \in \mathbb{R}^{D \times M}, \quad \boldsymbol{x}^T \in \mathbb{R}^{D \times N}, \quad \frac{\partial L}{\partial y} \in \mathbb{R}^{N \times M} \tag{1.41}$$

1.34 Vectorized Example

- Function:

$$f(\mathbf{x}; \boldsymbol{W}) = \|\boldsymbol{W}\mathbf{x}\|_2^2$$



- Initialization:

$$\frac{df}{df} = 1$$

- f-node: $f(\mathbf{f_1}) = \|\mathbf{f_1}\|_2^2$

$$\frac{\partial f}{\partial \mathbf{f_1}} = \frac{df}{df}\frac{\partial f}{\partial \mathbf{f_1}} = 1 \times 2\mathbf{f_1}$$

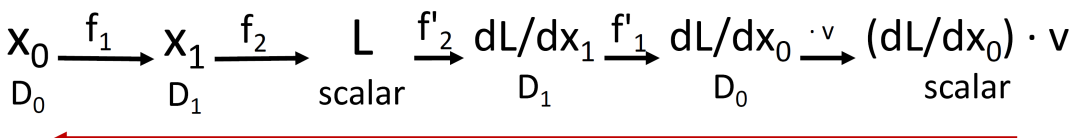
- $f1$-node: $f_1(\mathbf{x}, \boldsymbol{W}) = \boldsymbol{W}\mathbf{x}$

$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{f_1}}\frac{\partial f_1}{\partial \mathbf{x}} = 2\mathbf{f}_1\mathbf{x}^T \tag{1.42}$$

$$\frac{\partial f}{\partial \boldsymbol{W}} = \frac{\partial f}{\partial \mathbf{f_1}}\frac{\partial f_1}{\partial \boldsymbol{W}} = 2\boldsymbol{W}^T\mathbf{f}_1 \tag{1.43}$$

$$\tag{1.44}$$

1.35 Higher-Order Derivatives



Hessian / vector multiply

$\frac{\partial^2 L}{\partial x_0^2}$ **Hessian matrix** H of second derivatives.

$D_0 \times D_0$

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

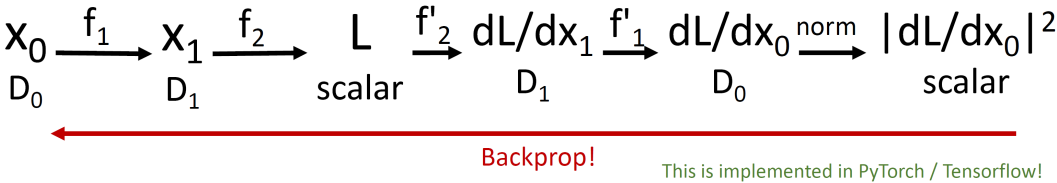
(if v doesn't depend on x_0)

$D_0 \times D_0 \quad D_0$

- Example: Gulrajani et al., Improved Training of Wasserstein GANs

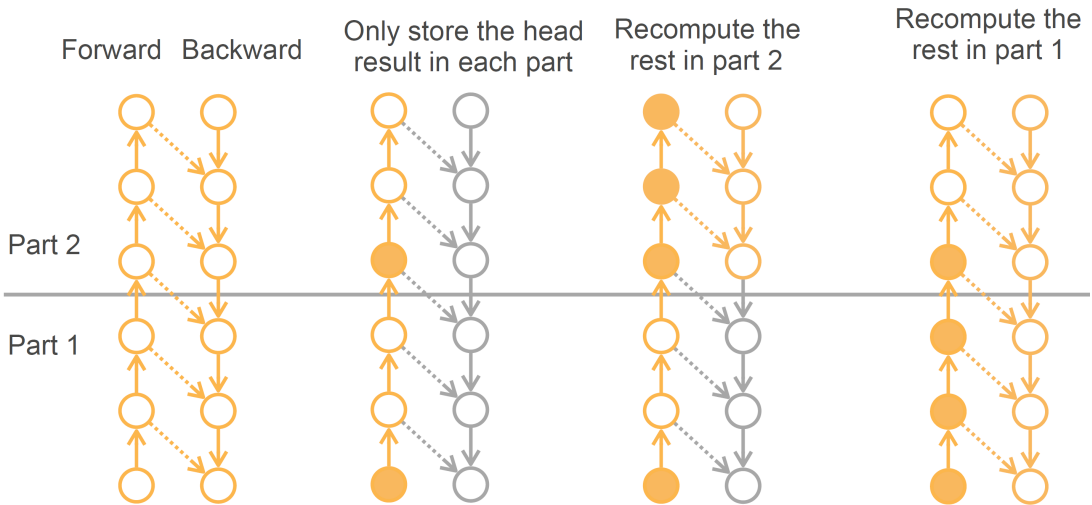
$$R(\boldsymbol{W}) = \left\| \frac{\partial L}{\partial \boldsymbol{W}} \right\|_2^2 = \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)^T \left(\frac{\partial L}{\partial \boldsymbol{W}} \right) \tag{1.45}$$

$$\frac{\partial}{\partial \mathbf{x}_0} R(\boldsymbol{W}) = 2 \left(\frac{\partial^2 L}{\partial \mathbf{x}_0^2} \right) \left(\frac{\partial L}{\partial \mathbf{x}_0} \right) \tag{1.46}$$



1.36 Memory Consumption of Backpropagation

- **Forward Pass**: Evaluate the graph, store intermediate resuls
  - memory consuming: scales linearly with batch size and # layers
- **Backward Pass**: Evaluate the graph in reverse order
  - can eliminate paths that are not needed
- **Rematerialization**:
  - Trade off computation for memory by recomputing values
  - E.g. ReLU is cheap to recompute, CONV layer is more expensive


Forward
Backward
Only store the head result in each part
Recompute the rest in part 2
Recompute the rest in part 1
Part 2
Part 1

1.37 Literature

- Backpropagation for a linear layer: cs231n handout