Lecture 10: Backpropagation

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

This lecture uses material from:

- A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey
- Stanford CS231n
- *Deep Learning* by I. Goodfellow

Important components of a CNN

deep imaging

CNN Architecture

Architecture choices

- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

Loss function & optimization

- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

Optimization choices

Other specifics: Pre-processing**,** initialization**,** dropout, batch normalization, augmentation

Important components of a CNN

deep imaging

Architecture choices

CNN Architecture

- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

Loss function & optimization

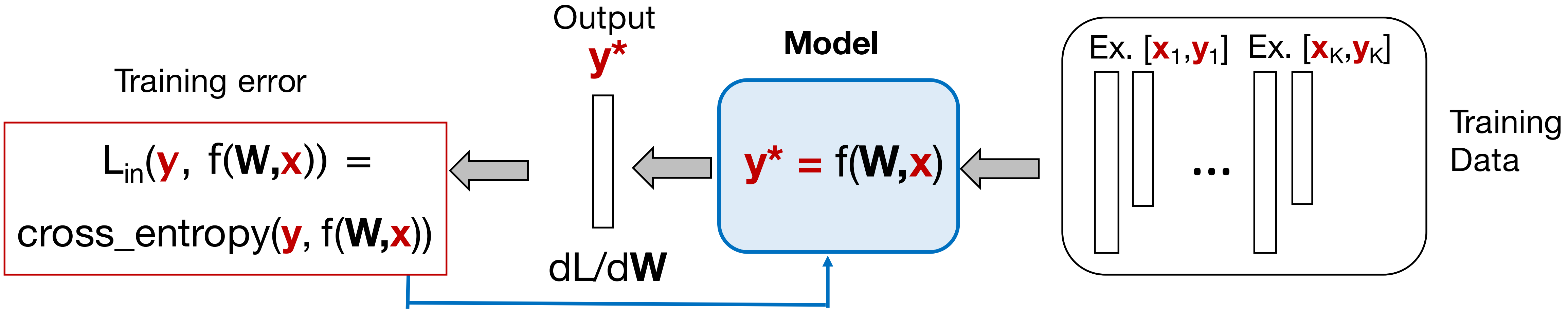
- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

Optimization choices

How does the optimizer actually work???

Other specifics: Pre-processing**,** initialization**,** dropout, batch normalization, augmentation

Our very basic convolutional neural network

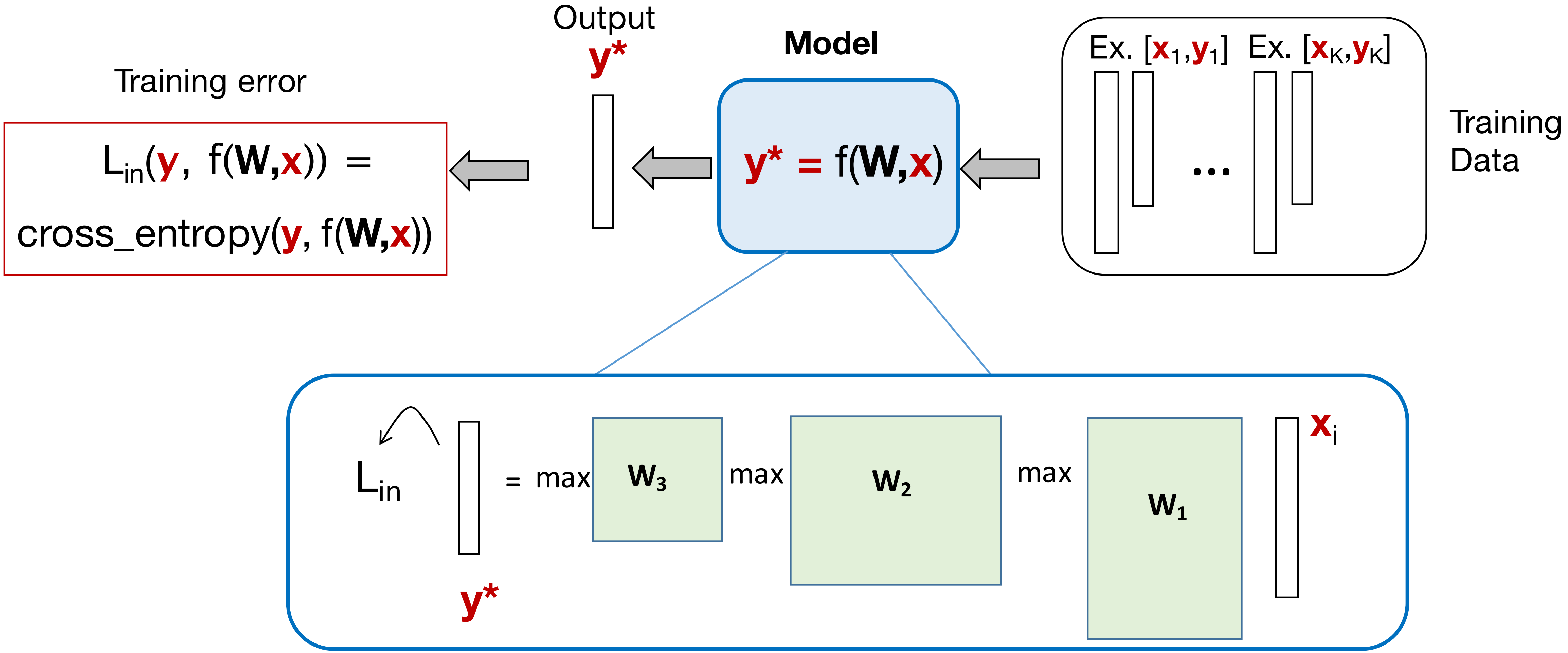


$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))}\right)$$

W_1 and W_2 are banded Toeplitz matrices, W_3 is a full matrix

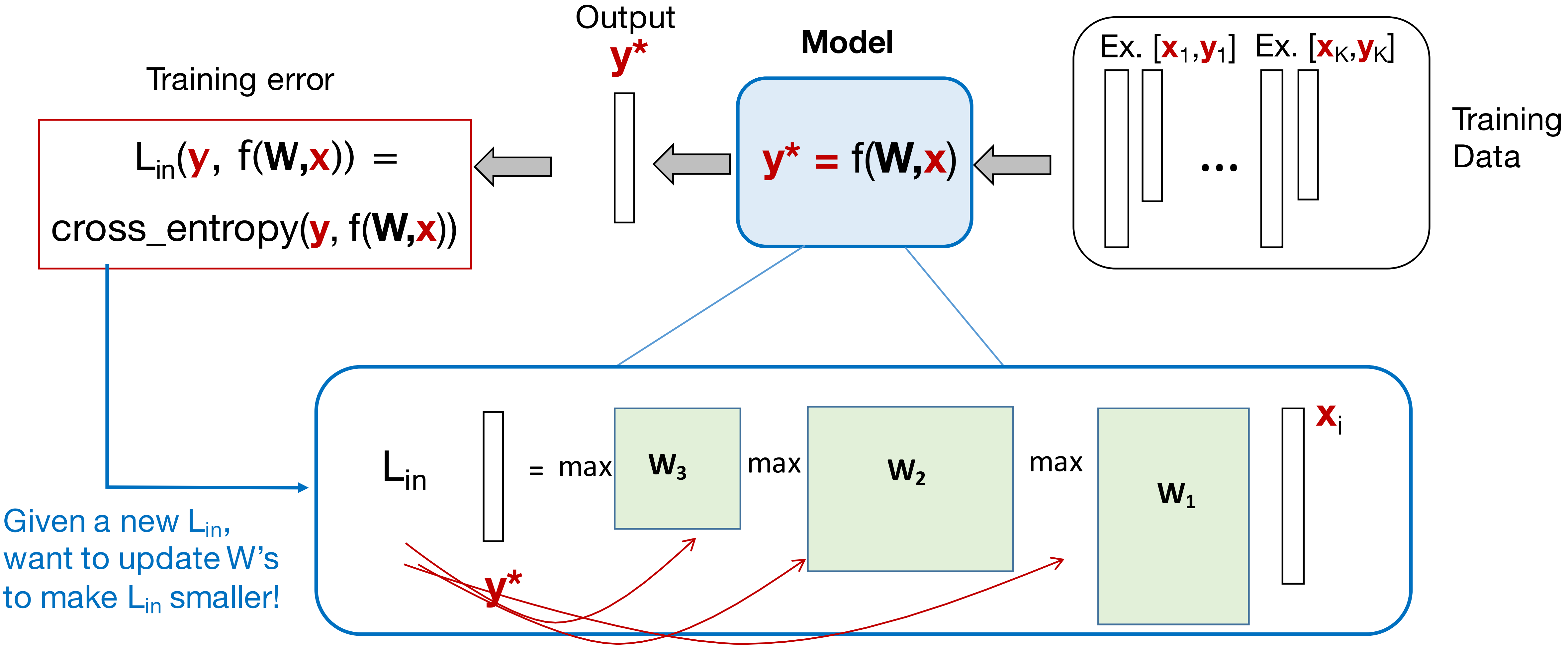
3-layer network

Our very basic convolutional neural network



Forward pass: from $\mathbf{x}_i$ and current **W**'s, find L_{in}

Our very basic convolutional neural network



Backwards pass: from new L_{in}, need to update **W**'s! How?

Review: how can we determine the optimal W?

deep imaging

- Here, let's assume we'll use the steepest descent algorithm to "go down the hill":

Review: how can we determine the optimal W?

- Here, let's assume we'll use the steepest descent algorithm to "go down the hill":

Input: labeled training examples [x_i,y_i] for i=1 to N, initial guess of **W**'s

while loss function is still decreasing:
  Compute loss function L(**W**,x_i,y_i)
  Update **W** to make L smaller:
    dL/d**W** = evaluate_gradient(**W**,x_i,y_i,L)
    W = **W** – step_size * dL/d**W**

```
while previous_step_size > precision and iters < max_iters
    prev_W = cur_W
    cur_W -= gamma * differential_dL(CNN_model, prev_W)
    previous_step_size = abs(cur_W - prev_W)
    iters+=1
```

Review: how can we determine the optimal W?

- Here, let's assume we'll use the steepest descent algorithm to "go down the hill":

Input: labeled training examples [$\mathbf{x}_i$,$\mathbf{y}_i$] for i=1 to N, initial guess of **W**'s

while loss function is still decreasing:
    Compute loss function L(**W**,$\mathbf{x}_i$,$\mathbf{y}_i$)
    Update **W** to make L smaller:
        $\boxed{\text{dL/d}\mathbf{W} = \text{evaluate\_gradient}(\mathbf{W},\mathbf{x}_i,\mathbf{y}_i,L)}$
        W = **W** – step_size * dL/d**W**

Options to evaluate dL/d**W**:
1. Numerical gradient
2. Analytic gradient
3. Automatic differentiation

```
while previous_step_size > precision and iters < max_iters
    prev_W = cur_W
    cur_W -= gamma * differential_dL(CNN_model, prev_W)
    previous_step_size = abs(cur_W - prev_W)
    iters+=1
```

Review: how can we determine the optimal W?

- Here, let's assume we'll use the steepest descent algorithm to "go down the hill":

Input: labeled training examples [$\mathbf{x}_i$,$\mathbf{y}_i$] for i=1 to N, initial guess of **W**'s

while loss function is still decreasing:
  Compute loss function L(**W**,$\mathbf{x}_i$,$\mathbf{y}_i$)
  Update **W** to make L smaller:
    dL/d**W** = evaluate_gradient(**W**,$\mathbf{x}_i$,$\mathbf{y}_i$,L)
    W = **W** – step_size * dL/d**W**

Options to evaluate dL/d**W**:

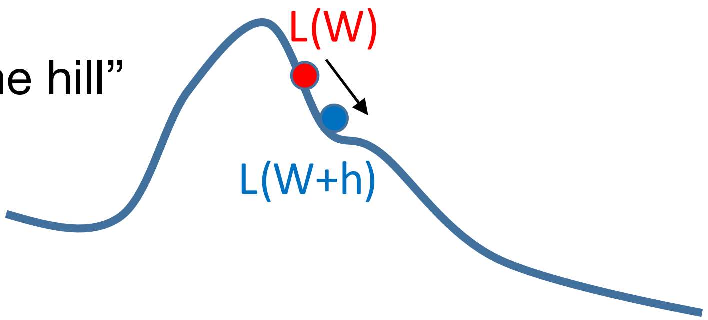
1. Numerical gradient
2. Analytic gradient
3. Automatic differentiation

```
while previous_step_size > precision and iters < max_iters
    prev_W = cur_W
    cur_W -= gamma * differential_dL(CNN_model, prev_W)
    previous_step_size = abs(cur_W - prev_W)
    iters+=1
```

*Note: Other gradient descent methods require the same fundamental calculation. So how gradient *is computed* is a different problem then how it is *used*

1. Numerical gradient, a simple example

"Go down the hill"



With a matrix, compute this for each entry:

$$\boxed{\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}}$$

<u>Example:</u>

W = [1,2;3,4]
L(W, x, y) = 12.79

W_1+h = [1.001,2;3,4]
L(W_1+h, x, y) = 12.8

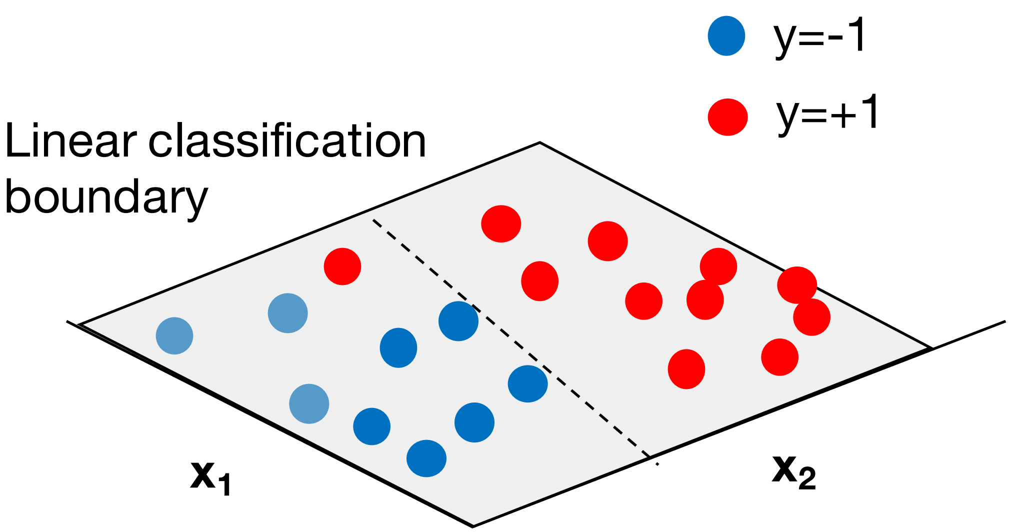
d$L(W_1)/dW_1$ = 12.8-12.79/.001

$\boxed{dL(W_1)/dW_1 = 10}$

Pros: Simple! Easy to code up!

Cons: Slow…really slow. And approximate

2. Analytic gradient, a simple example



$$L = \frac{1}{N}\sum_{i=1}^{N}(w^T x_i - y_i)^2$$

$$\nabla L(w) = \frac{2}{N} X^T (Xw - y)$$

Evaluate and use to update W

Pros: Fast and exact

Cons: Error prone, especially with deep networks…

deep imaging

3. Automatic differentiation – what we'll use without knowing it

Resources:

- Stanford CS231n, Lecture 4 notes and resources
  - http://cs231n.stanford.edu/syllabus

- I. Goodfellow et al., Deep Learning Chapter 6 Section 5
  - https://www.deeplearningbook.org/contents/mlp.html

- A. Baydin et al., "Automatic differentiation in machine learning: a survey"
  - https://arxiv.org/pdf/1502.05767.pdf

3. Automatic differentiation – what is it?

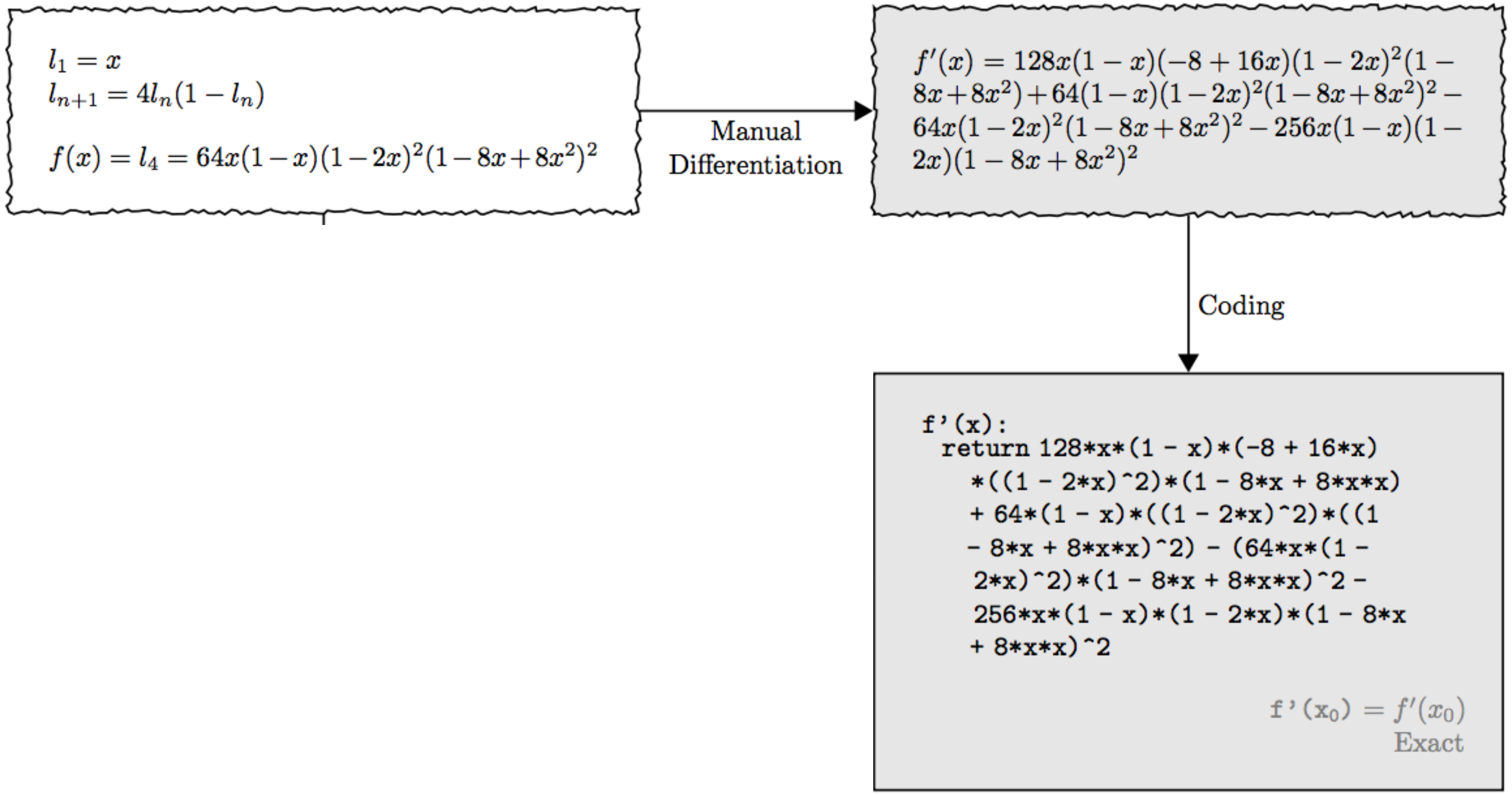
deep imaging

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- *Repeat process in lock-step with evaluation of final function*

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

3. Automatic differentiation – what is it?

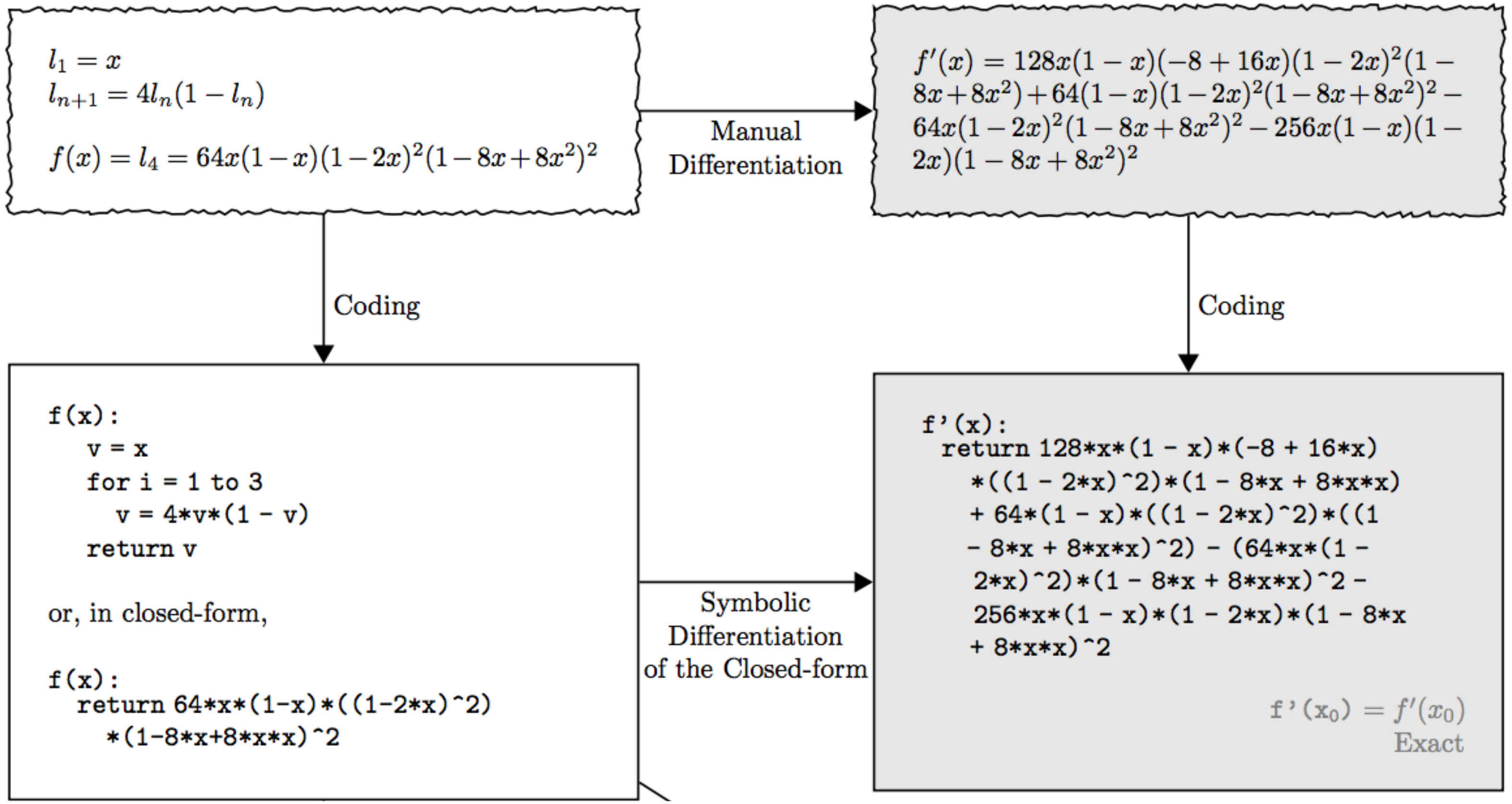
- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- *Repeat process in lock-step with evaluation of final function*



A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

3. Automatic differentiation – what is it?

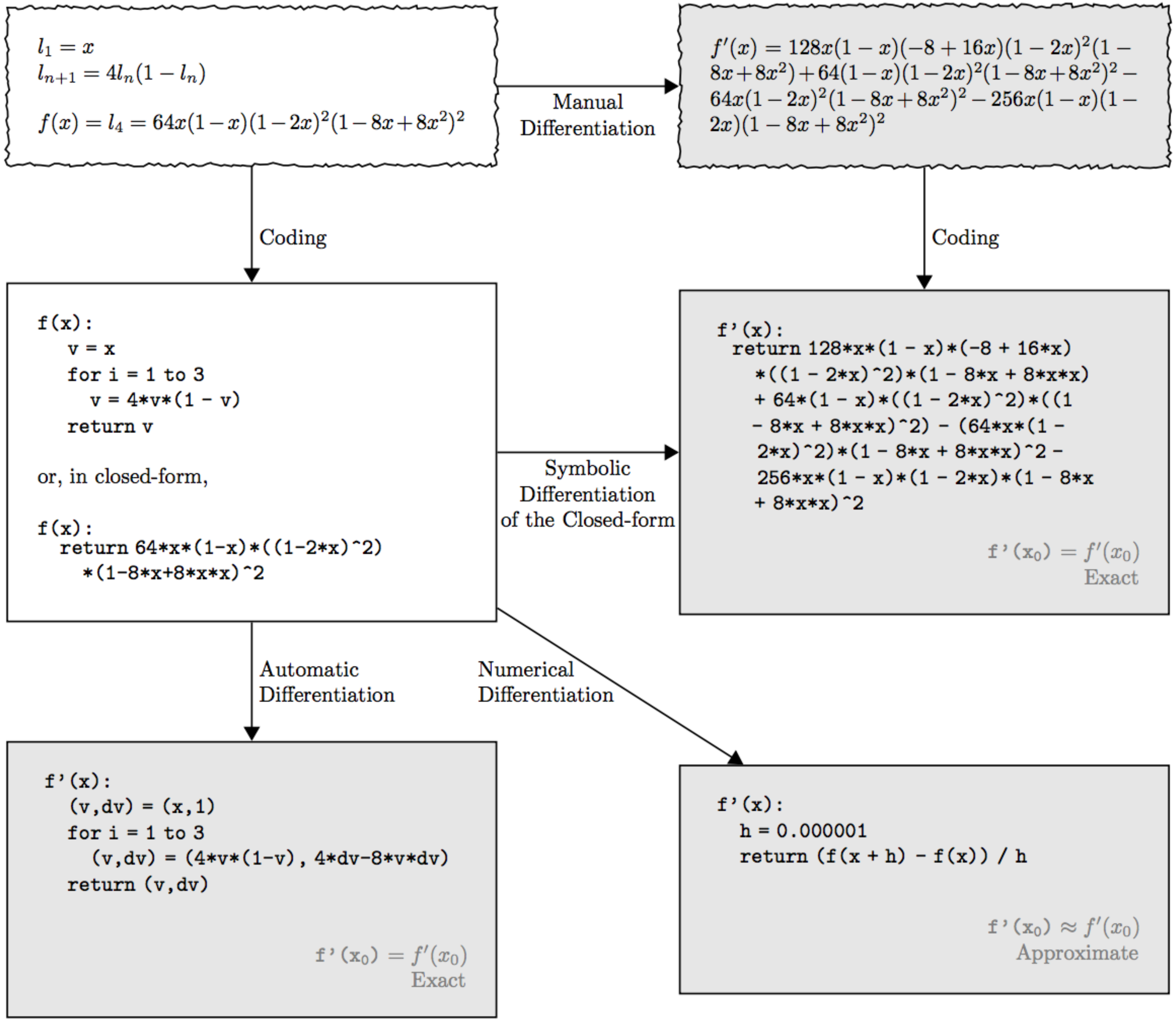
- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- *Repeat process in lock-step with evaluation of final function*



A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

3. Automatic differentiation – what is it?

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- *Repeat process in lock-step with evaluation of final function*

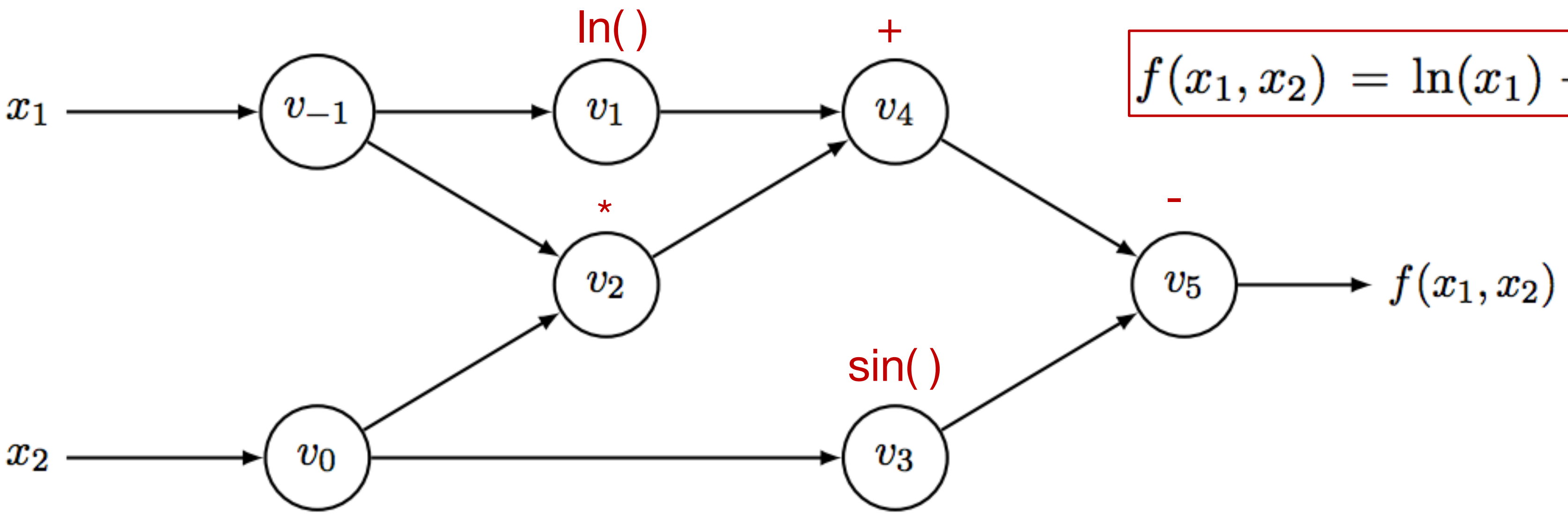


A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

Automatic differentiation on computational graphs

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$

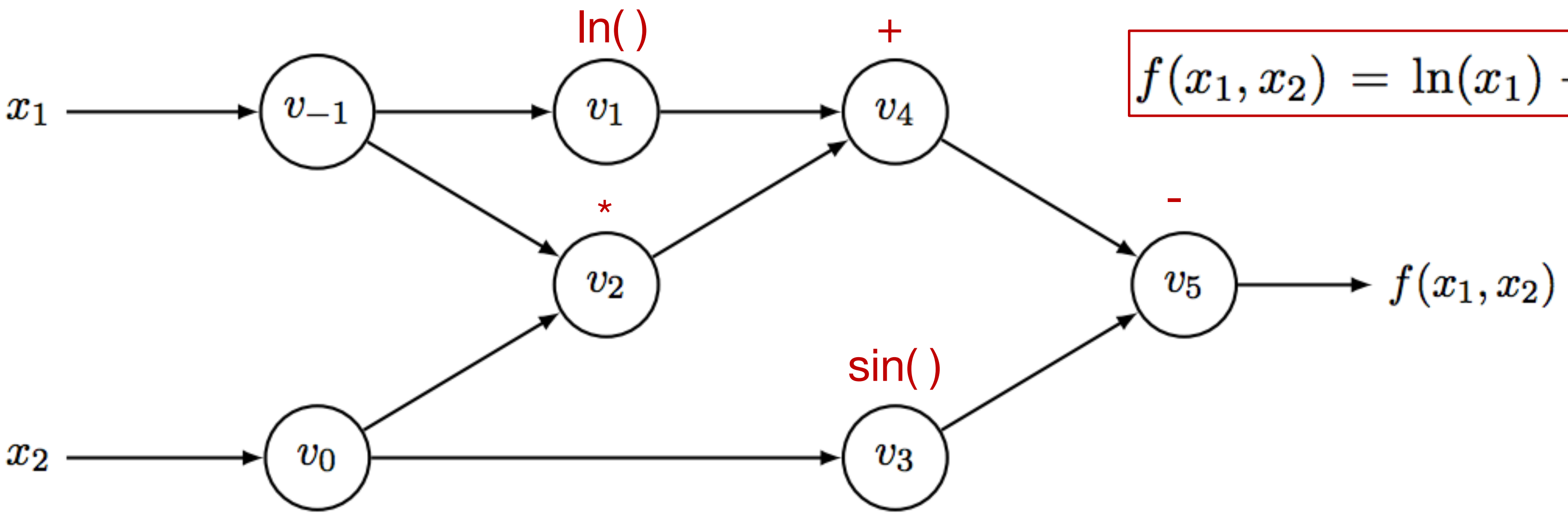


Automatic differentiation on computational graphs

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging

$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$



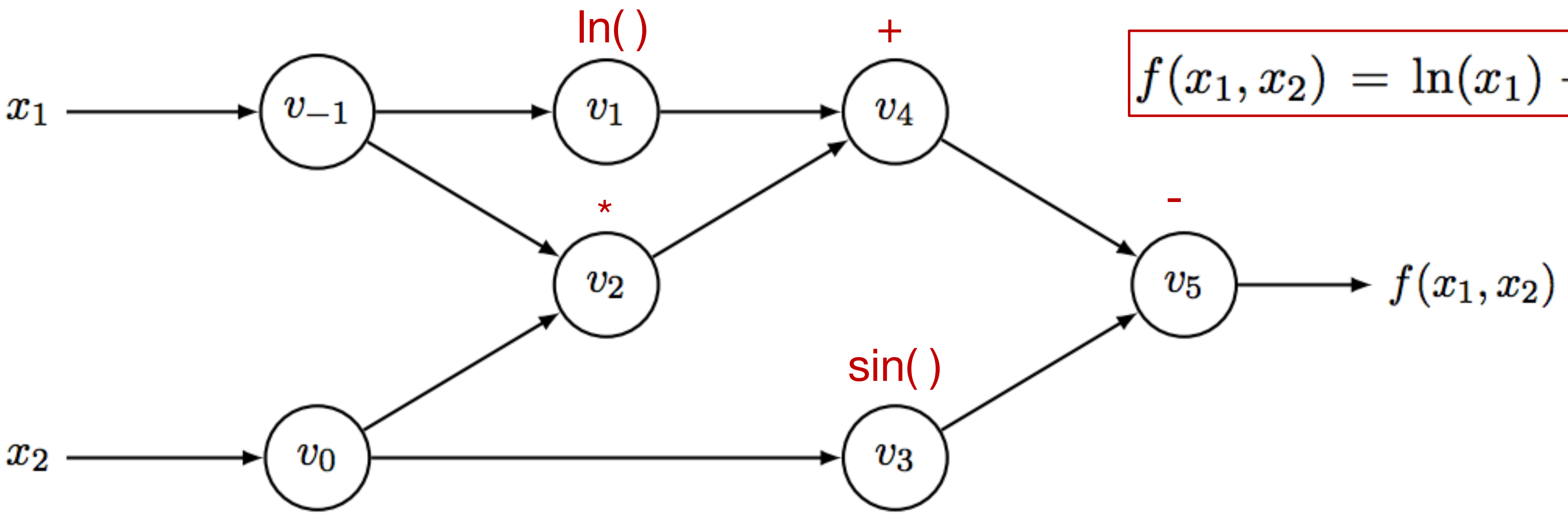
To both determine f and find df/dx_i :

- Create graph of local operations
- Compute analytic (symbolic) gradient at each node (unit) in graph
- Use inter-relationships to establish final desired gradient, df/dx_1
  - Forward differentiation
  - Backwards differentiation = Backpropagation

Automatic differentiation on computational graphs

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging



$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$

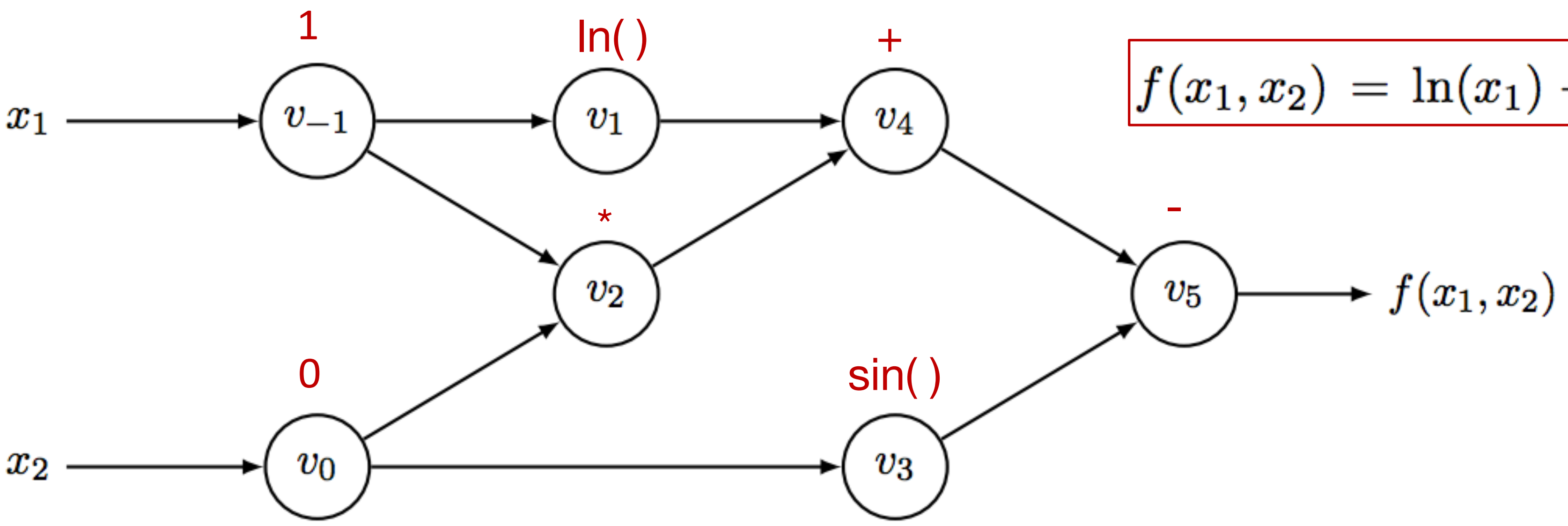
Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging

Forward automatic differentiation

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$



Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Forward Tangent (Derivative) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1} / v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$

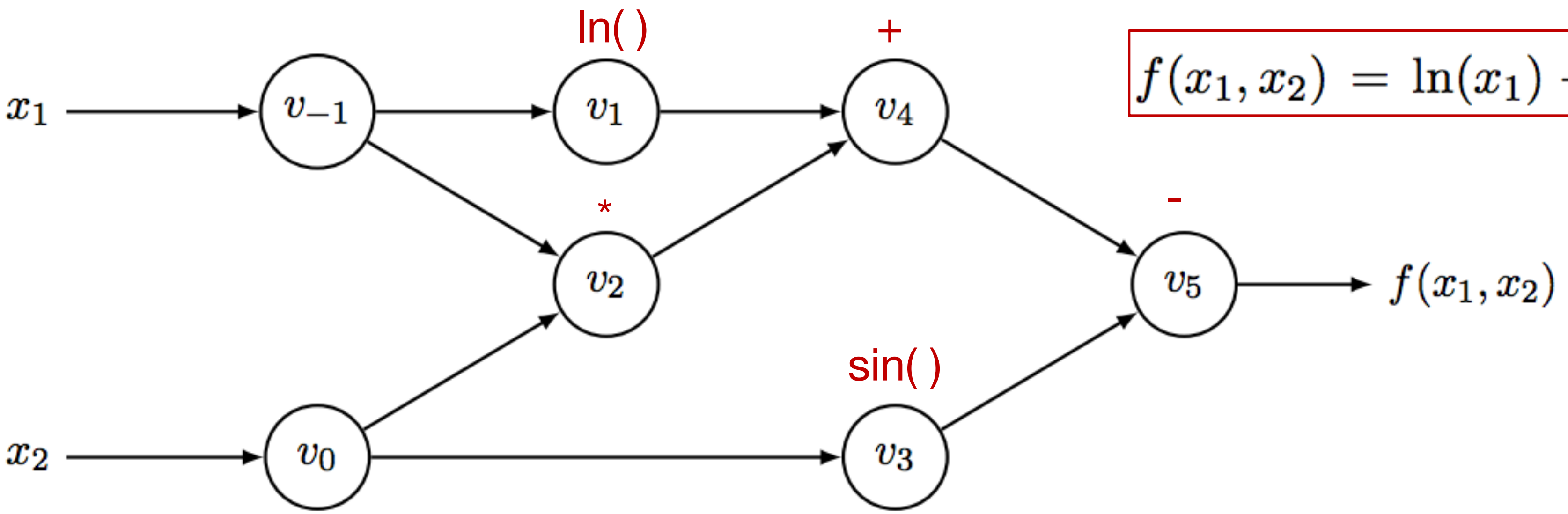
Set to 1 because we want df/dx_1

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging

Forward automatic differentiation

$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$



Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

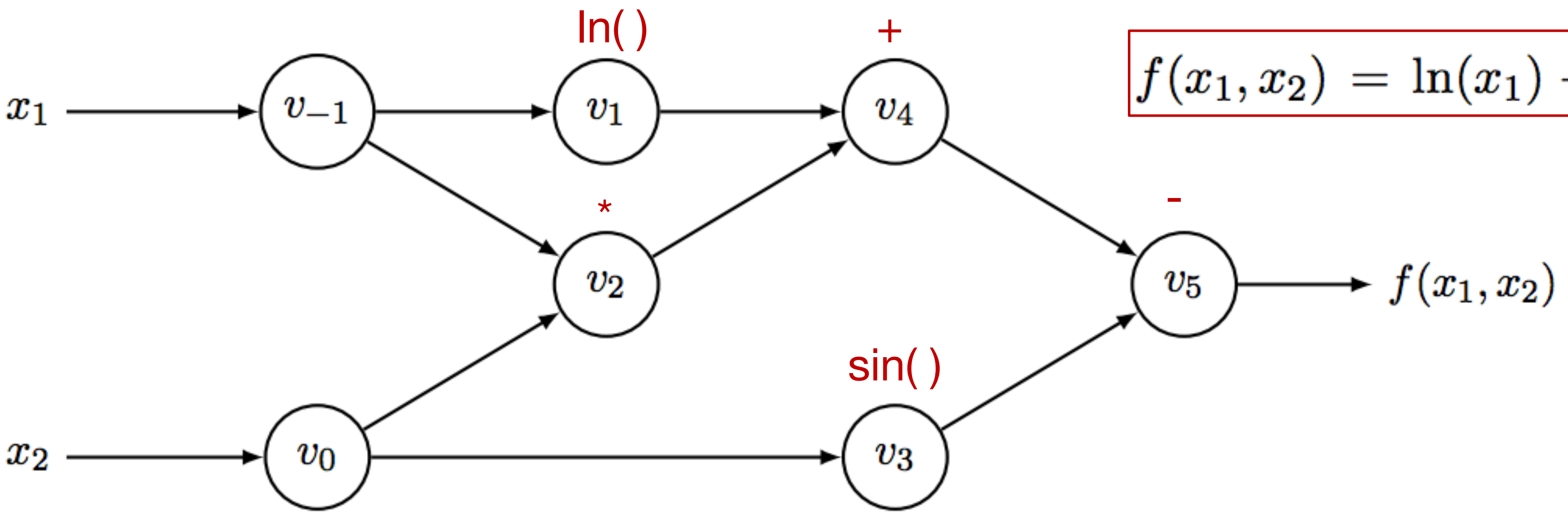
Forward Tangent (Derivative) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\boldsymbol{= 5.5}$

Compute local derivative for *all inputs and accumulate* with chain rule

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

Forward automatic differentiation

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$



Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

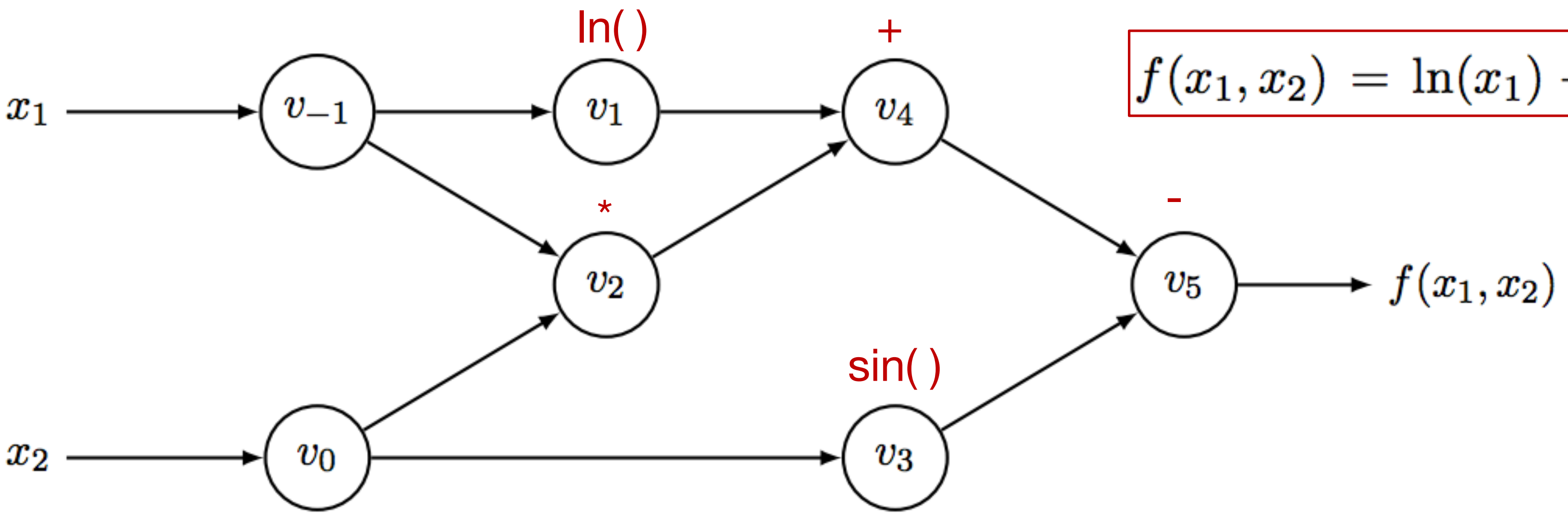
Forward Tangent (Derivative) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$

Leads to final desired df/dx_1

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging

Forward automatic differentiation



$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$

Forward Primal Trace

$$\begin{aligned}
v_{-1} &= x_1 && = 2 \\
v_0 &= x_2 && = 5 \\
\hline
v_1 &= \ln v_{-1} && = \ln 2 \\
v_2 &= v_{-1} \times v_0 && = 2 \times 5 \\
v_3 &= \sin v_0 && = \sin 5 \\
v_4 &= v_1 + v_2 && = 0.693 + 10 \\
v_5 &= v_4 - v_3 && = 10.693 + 0.959 \\
\hline
y &= v_5 && = 11.652
\end{aligned}$$

Forward Tangent (Derivative) Trace

$$\begin{aligned}
\dot{v}_{-1} &= \dot{x}_1 && = 1 \\
\dot{v}_0 &= \dot{x}_2 && = 0 \\
\hline
\dot{v}_1 &= \dot{v}_{-1}/v_{-1} && = 1/2 \\
\dot{v}_2 &= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1} && = 1 \times 5 + 0 \times 2 \\
\dot{v}_3 &= \dot{v}_0 \times \cos v_0 && = 0 \times \cos 5 \\
\dot{v}_4 &= \dot{v}_1 + \dot{v}_2 && = 0.5 + 5 \\
\dot{v}_5 &= \dot{v}_4 - \dot{v}_3 && = 5.5 - 0 \\
\hline
\boldsymbol{\dot{y}} &= \boldsymbol{\dot{v}_5} && \mathbf{= 5.5}
\end{aligned}$$

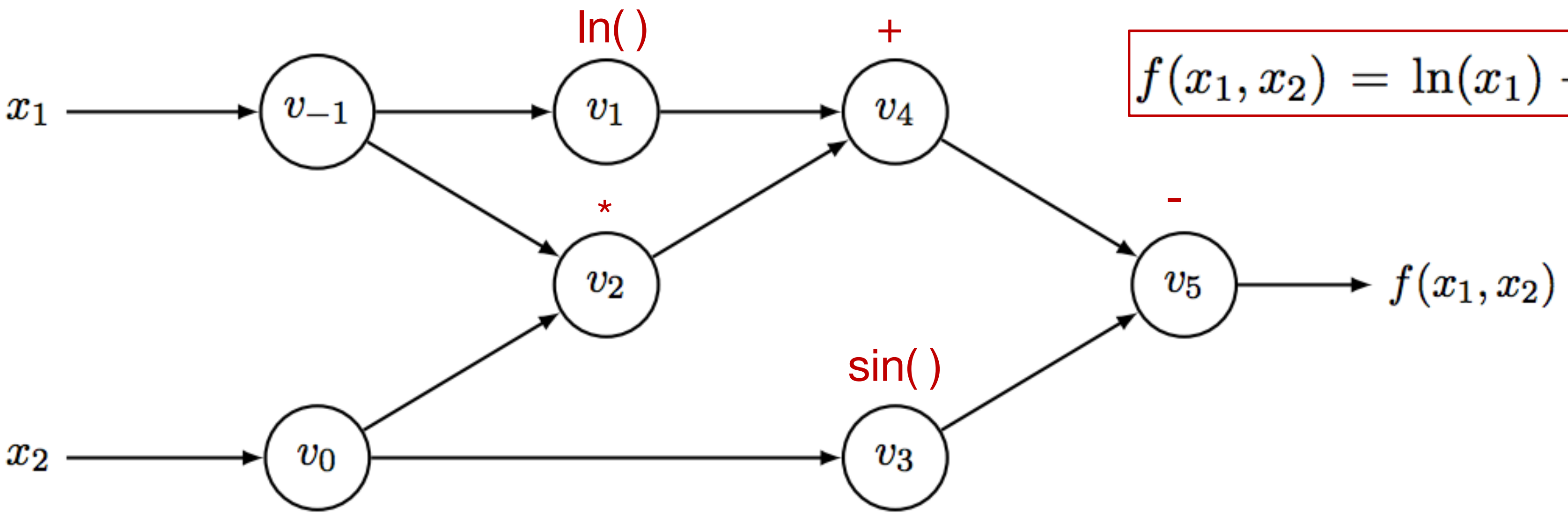
<u>Problem</u>:

- For N inputs, need to compute this N times, setting x_i to 1 each time…

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey

deep imaging

Forward automatic differentiation



$$\boxed{f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$

Forward Primal Trace

$$\begin{array}{lll}
v_{-1} & = x_1 & = 2 \\
v_0 & = x_2 & = 5 \\
\hline
v_1 & = \ln v_{-1} & = \ln 2 \\
v_2 & = v_{-1} \times v_0 & = 2 \times 5 \\
v_3 & = \sin v_0 & = \sin 5 \\
v_4 & = v_1 + v_2 & = 0.693 + 10 \\
v_5 & = v_4 - v_3 & = 10.693 + 0.959 \\
\hline
y & = v_5 & = 11.652
\end{array}$$

Forward Tangent (Derivative) Trace

$$\begin{array}{lll}
\dot{v}_{-1} & = \dot{x}_1 & = 1 \\
\dot{v}_0 & = \dot{x}_2 & = 0 \\
\hline
\dot{v}_1 & = \dot{v}_{-1}/v_{-1} & = 1/2 \\
\dot{v}_2 & = \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1} & = 1 \times 5 + 0 \times 2 \\
\dot{v}_3 & = \dot{v}_0 \times \cos v_0 & = 0 \times \cos 5 \\
\dot{v}_4 & = \dot{v}_1 + \dot{v}_2 & = 0.5 + 5 \\
\dot{v}_5 & = \dot{v}_4 - \dot{v}_3 & = 5.5 - 0 \\
\hline
\dot{\boldsymbol{y}} & = \dot{\boldsymbol{v}}_5 & = \mathbf{5.5}
\end{array}$$

Problem:

- For N inputs, need to compute this N times, setting x_i to 1 each time…

Solution:

Work backwards from end to start with back-propagation

Backpropagation explanation from Stanford CS231N Slides

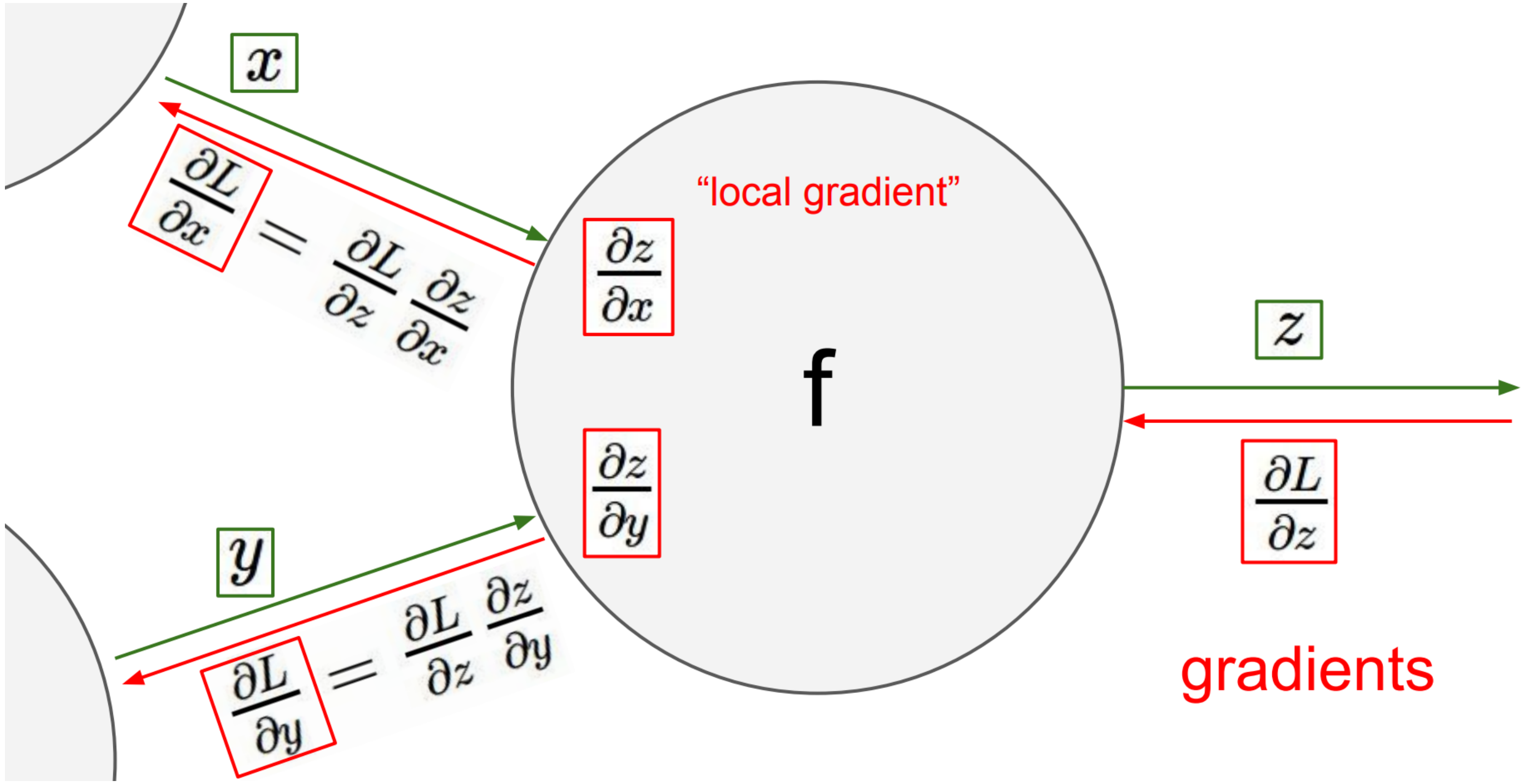
Go over slides 12-44 here: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

Other useful info here: http://cs231n.github.io/optimization-2/

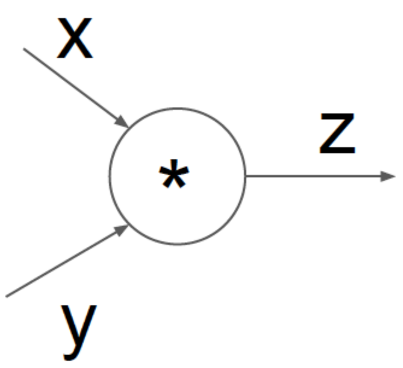
Backpropagation explanation from Stanford CS231N Slides

Treat intermediate nodes like a dummy variable z, for $L(w_1)$

Key Idea: $dL/dw_1 = (dL/dz)(dz/dw_1)$



Modularized implementation: forward / backward API



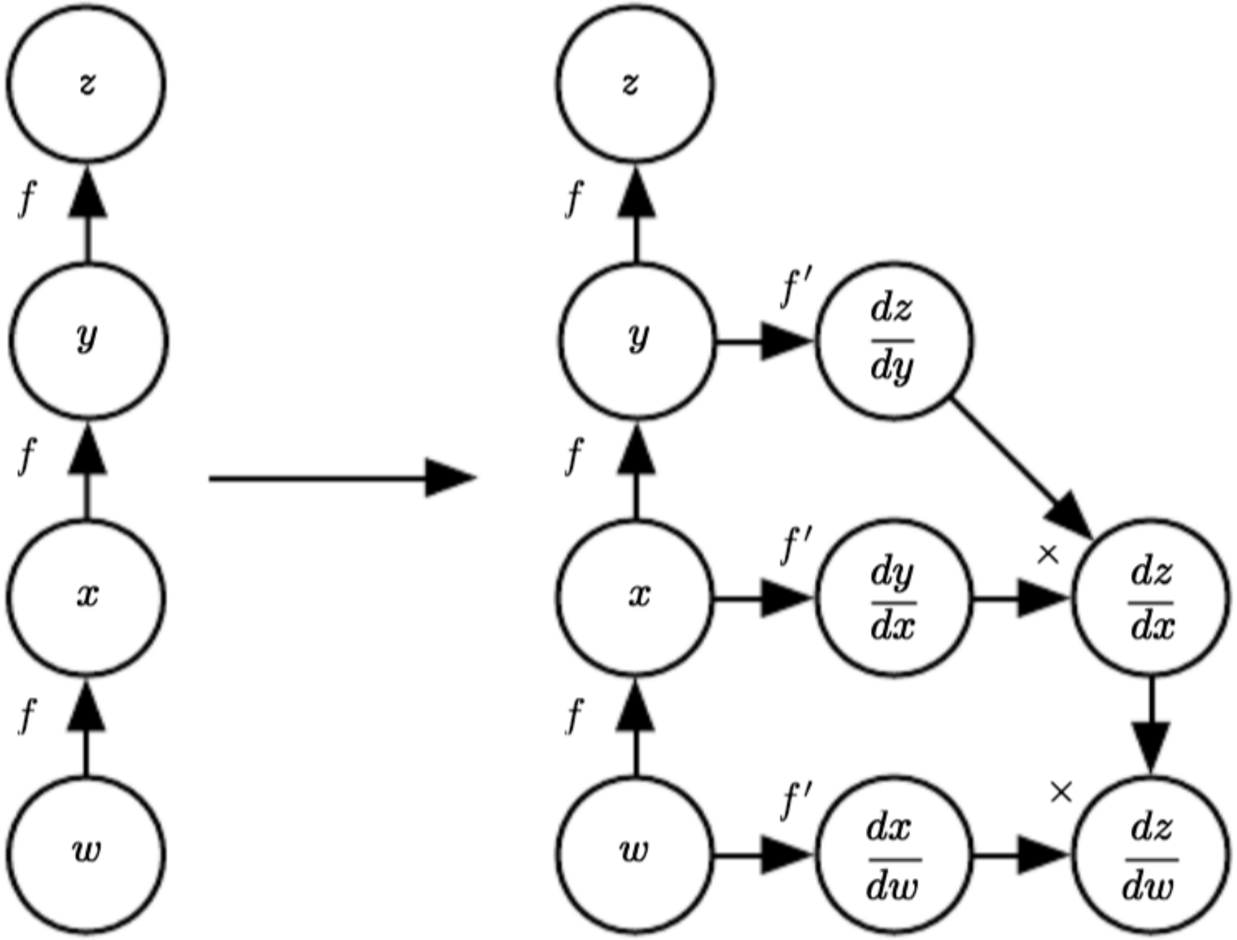
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Local gradient

Upstream gradient variable

How Tensorflow actually works: create a whole extra new graph



Final desired derivative

Last thing – matrix and vector derivatives

Here's a review:

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

Last thing – matrix and vector derivatives

Here's a review:

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

$$\mathbf{u}_3 = W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M$$

$$\frac{\partial u_3}{\partial v_2} = \frac{\partial}{\partial v_2}(W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M) = \frac{\partial}{\partial v_2}W_{3,2}v_2 = W_{3,2}$$

$$\frac{\partial u_i}{\partial v_j} = W_{i,j}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} = \mathbf{W}$$

- When confused, write out one entry, solve derivative and generalize
- Use dimensionality to help (if **x** has N elements, and **y** has M, then d**y**/d**x** must be NxM
- Take advantage of *The Matrix Cookbook*:
  - https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Let's go through an example:

deep imaging

$$L = \| \mathbf{W_2}\, \mathrm{ReLU}(\mathbf{W_1}\, \mathbf{X}) \|_2^2$$

(2-layer network with MSE where we neglect labels **y** for now)

$$dL/d\mathbf{W}_1 = ? \quad dL/d\mathbf{W}_2 = ?$$

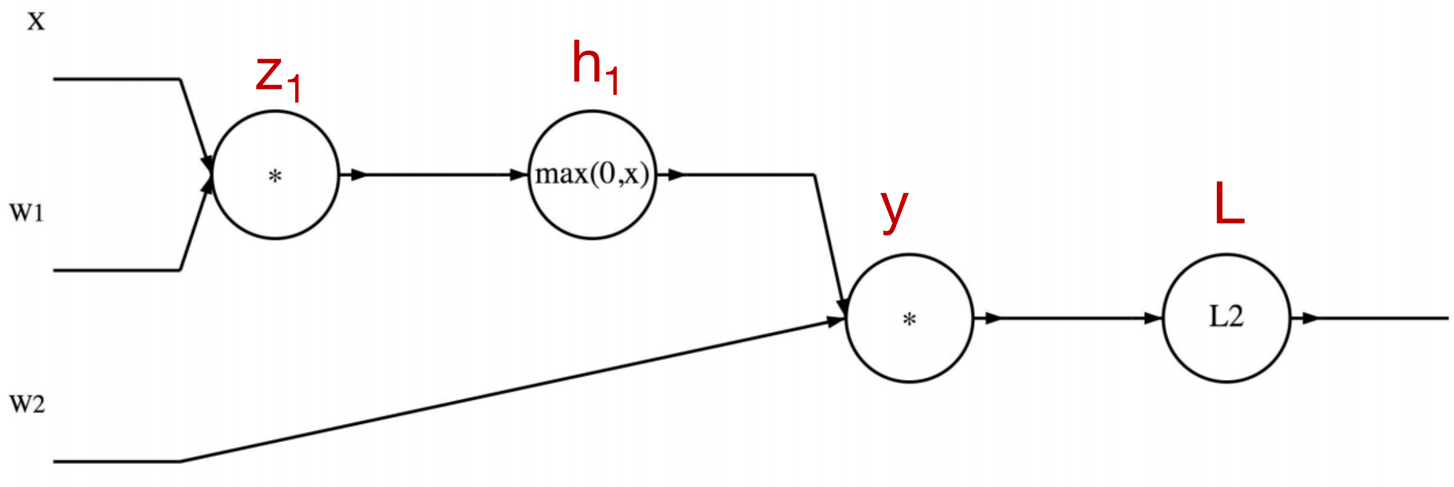
Let's go through an example:

$$L = \| \mathbf{W}_2 \, \mathrm{ReLU}(\mathbf{W}_1 \mathbf{X}) \|_2^2$$

(2-layer network with MSE where we neglect labels **y** for now)

$dL/d\mathbf{W}_1 = ?$ $\quad dL/d\mathbf{W}_2 = ?$

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$



Forward pass: solve for z_1, h_1, y and L

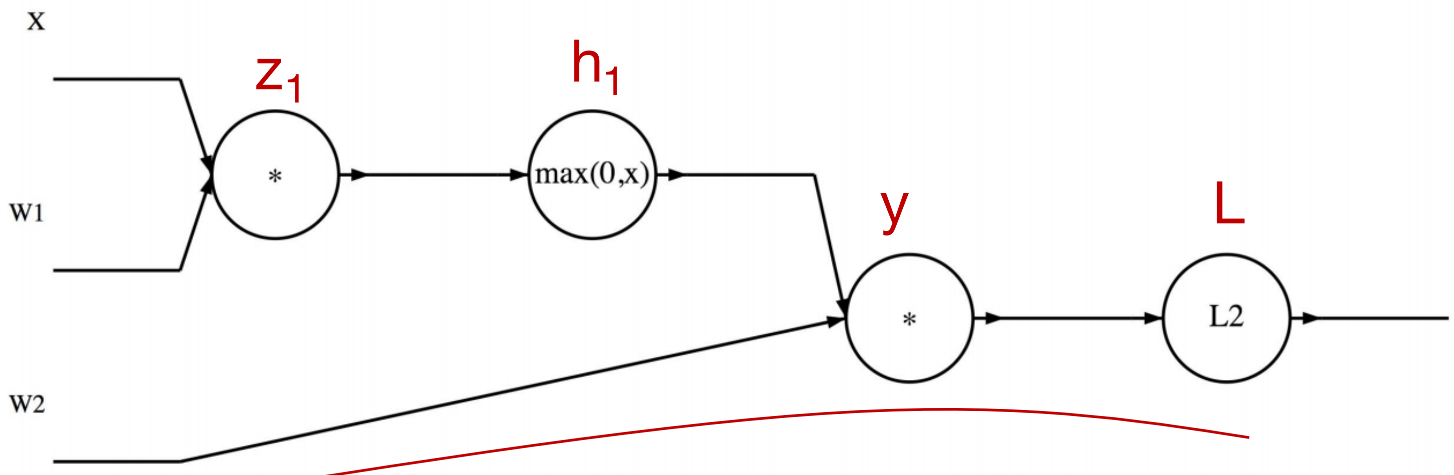
Let's go through an example:

$$L = \|\, \mathbf{W_2}\ \mathrm{ReLU}(\mathbf{W_1}\ \mathbf{X})\, \|_2^2$$

(2-layer network with MSE where we neglect labels **y** for now)

$dL/d\mathbf{W}_1$ = ? $\boxed{dL/d\mathbf{W}_2 = ?}$

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$



Backpropagation from L to W_2

$$\boxed{\frac{\partial L}{\partial W_2} = \frac{\partial \hat{y}}{\partial W_2}\frac{\partial L}{\partial \hat{y}}}$$

Gradients for scalar *L* will have same shape as denominator

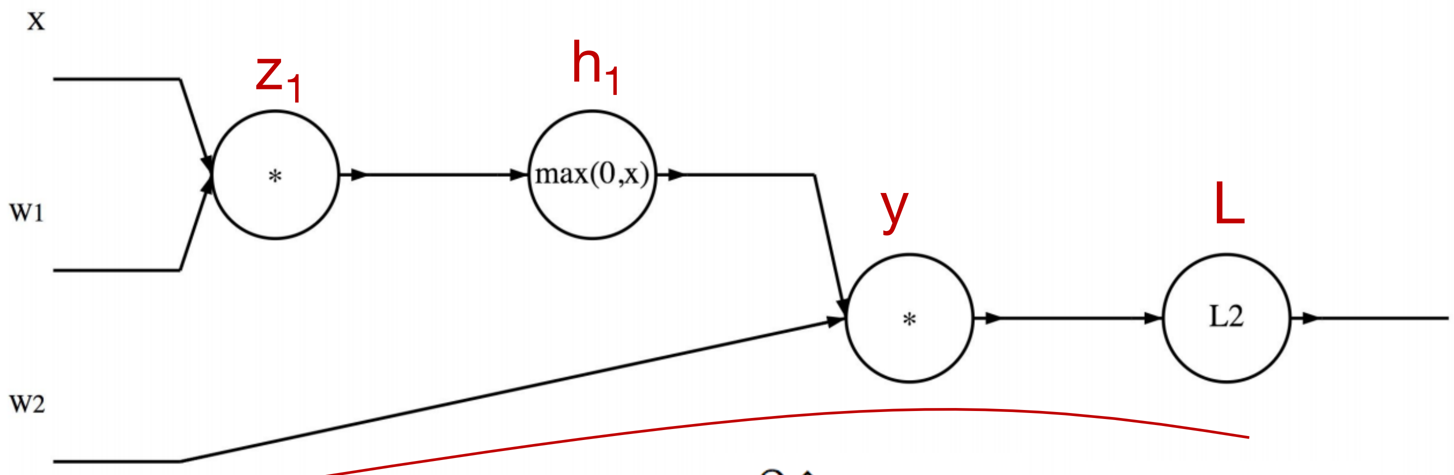
Let's go through an example:

$$L = \| \mathbf{W_2} \, \mathrm{ReLU}(\mathbf{W_1} \, \mathbf{X}) \|_2^2$$

(2-layer network with MSE where we neglect labels **y** for now)

$dL/d\mathbf{W}_1$ = ? $\boxed{dL/d\mathbf{W}_2 = ?}$

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$



$$\frac{\partial \hat{y}}{\partial W_2} = h_1^T \qquad \frac{\partial L}{\partial \hat{y}} = 2\hat{y}$$

Gradients for scalar L will have same shape as denominator

$$\boxed{\frac{\partial L}{\partial W_2} = \frac{\partial \hat{y}}{\partial W_2} \frac{\partial L}{\partial \hat{y}}}$$

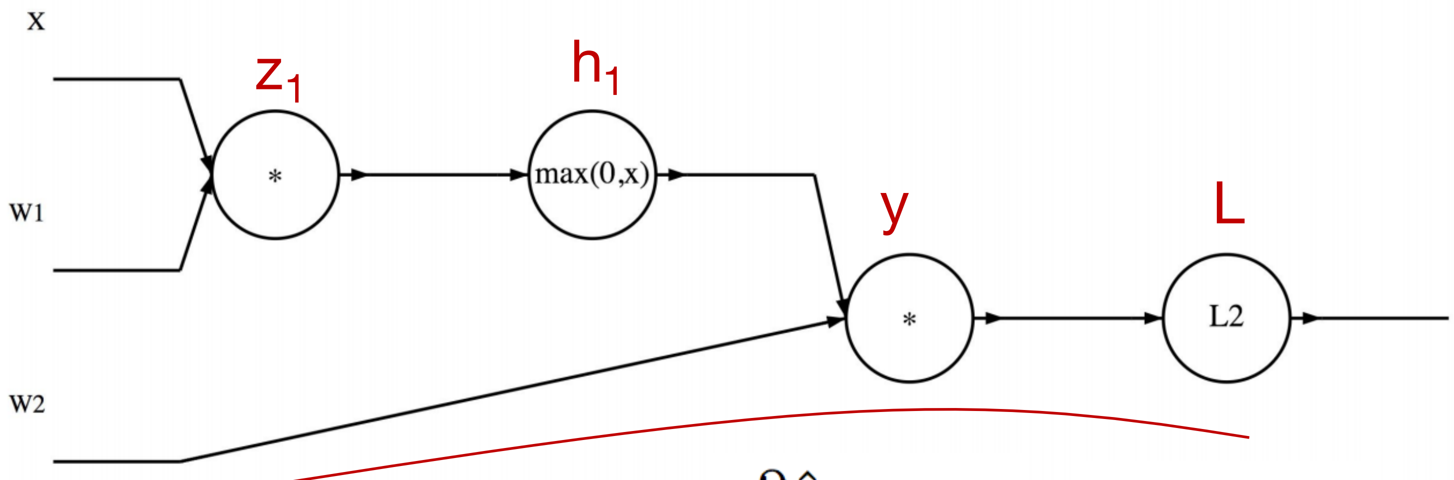
Let's go through an example:

L = || $\mathbf{W_2}$ ReLU($\mathbf{W_1}$ $\mathbf{X}$) $||_2^2$

(2-layer network with MSE where we neglect labels **y** for now)

dL/d$\mathbf{W}_1$ = ? $\boxed{\text{dL/d}\mathbf{W}_2 = ?}$

$$z_1 = XW_1$$
$$h_1 = \text{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$



$$\frac{\partial \hat{y}}{\partial W_2} = h_1^T \qquad \frac{\partial L}{\partial \hat{y}} = 2\hat{y}$$

$$\boxed{\frac{\partial L}{\partial W_2} = \frac{\partial \hat{y}}{\partial W_2} \frac{\partial L}{\partial \hat{y}} = 2h_1^T \hat{y}}$$

Let's go through an example:

L = || $\mathbf{W_2}$ ReLU($\mathbf{W_1}$ $\mathbf{X}$) $||_2^2$

(2-layer network with MSE where we neglect labels **y** for now)

$\boxed{dL/d\mathbf{W}_1 = ?}$ $dL/d\mathbf{W}_2 = ?$

$$z_1 = XW_1$$

$$h_1 = \mathrm{ReLU}(z_1)$$

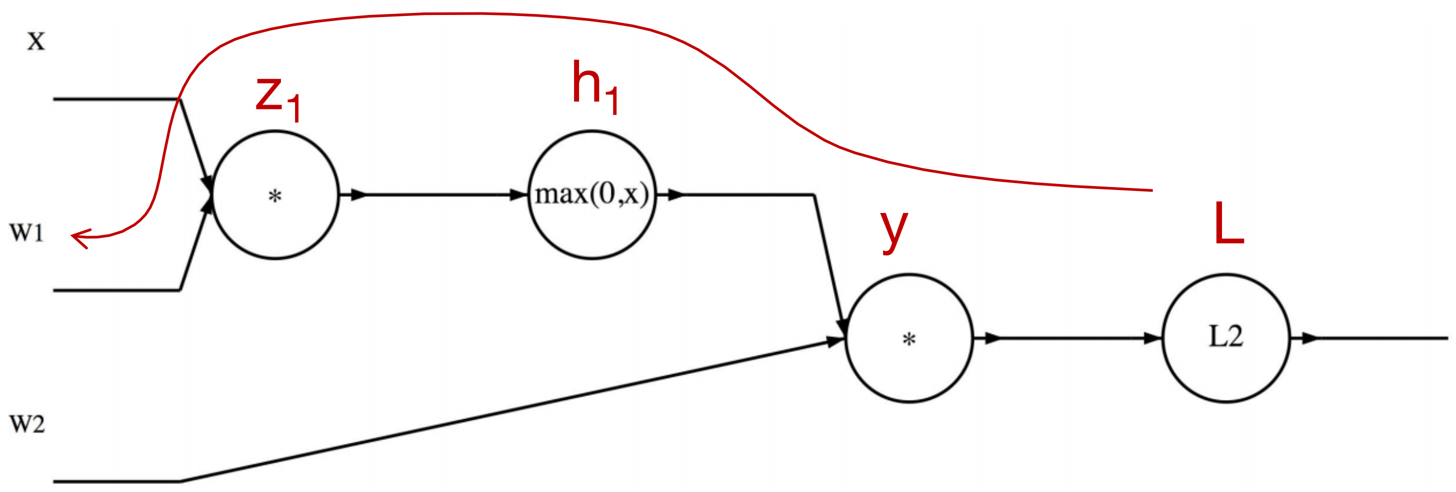
$$\hat{y} = h_1 W_2$$

$$L = ||\hat{y}||_2^2$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} W_2^\top$$

$$\frac{\partial h_1}{\partial z_1} = \frac{\partial L}{\partial h_1} \circ I[h_1 > 0]$$

$$\frac{dL}{dW_1} = \boxed{\frac{dh_1}{\partial W_1} = x^\top \frac{\partial h_1}{\partial z_1}}$$



Rely on dimensional analysis to select order of operations

Let's go through an example:

$$L = \| \mathbf{W_2}\, \mathrm{ReLU}(\mathbf{W_1}\, \mathbf{X}) \|_2^2$$

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$

$$\boxed{dL/d\mathbf{W}_1 = ?}$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} W_2^\top$$
$$\frac{\partial h_1}{\partial z_1} = \frac{\partial L}{\partial h_1} \circ I[h_1 > 0]$$
$$\frac{dL}{dW_1} = \boxed{\frac{dh_1}{\partial W_1} = x^\top \frac{\partial h_1}{\partial z_1}}$$

```
import numpy as np

# forward prop
z_1 = np.dot(X, W_1)
h_1 = np.maximum(z_1, 0)
y_hat = np.dot(h_1, W_2)
L = np.sum(y_hat**2)

# backward prop
dy_hat = 2.0*y_hat

dW2 = h_1.T.dot(dy_hat)

dh1 = dy_hat.dot(W_2.T)

dz1 = dh1.copy()
dz1[z1 < 0] = 0

dW1 = X.T.dot(dz1)
```

Summary

- Tensorflow: define variables, series of operations & a cost function
- When you hit enter, Tensorflow effectively forms two graphs
    - Forward graph to evaluate function at each node
    - **Backprop:** Backwards graph that includes *local* derivatives of each operation as symbolic functions, as well as connections
- Tensorflow will go through the forward graph & save numerical results, then the backwards graph, to update weights via local operations, to minimize cost function
- Uses more impressive operations to do this with vectors and matrices efficiently