Backpropagation and Gradients

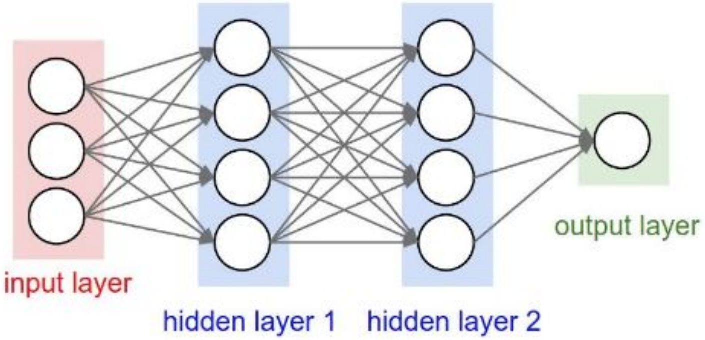
Agenda

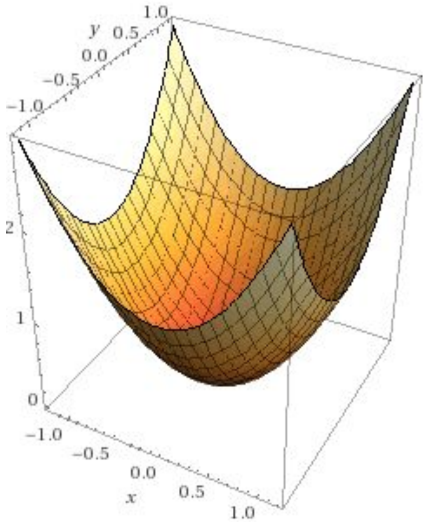
- Motivation
- Backprop Tips & Tricks
- Matrix calculus primer
- Example: 2-layer Neural Network

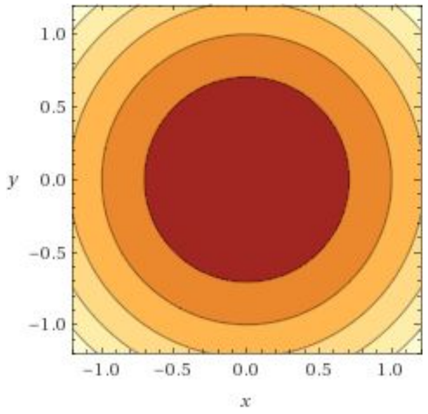
Motivation

Recall: Optimization objective is minimize loss

Goal: how should we tweak the parameters to decrease the loss slightly?



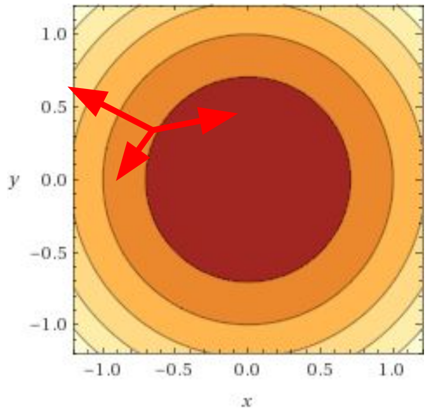




Plotted on WolframAlpha

Approach #1: Random search

Intuition: the way we tweak parameters is the *direction* we step in our optimization



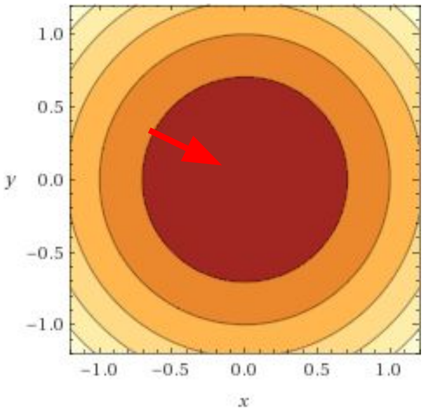
What if we randomly choose a direction?

Approach #2: Numerical gradient

Intuition: gradient describes rate of change of a function with respect to a variable surrounding an infinitesimally small region

Finite Differences:

$$\frac{f(x+h) - f(x)}{h}$$

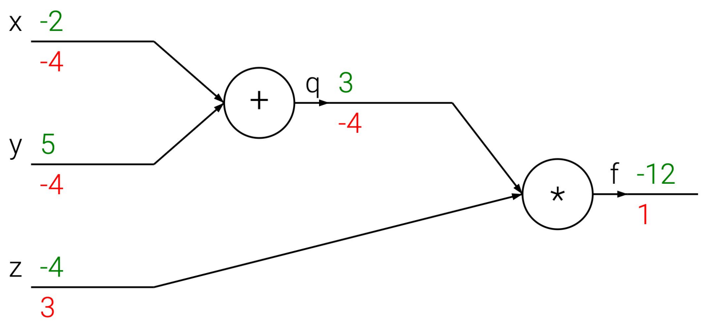


Challenge: how do we compute the gradient independent of each input?

Approach #3: Analytical gradient

Recall: chain rule

$$z = f(y), y = g(x)$$
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$



Assuming we know the structure of the computational graph beforehand…

Intuition: upstream gradient values propagate backwards -- we can reuse them!

What about autograd?

- Deep learning frameworks can automatically perform backprop!

- Problems might surface related to underlying gradients when debugging your models

“Yes You Should Understand Backprop”

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Problem Statement

$$Loss = f(x, y; \theta)$$

Given a function ***f*** with respect to inputs ***x***, labels ***y***, and parameters ***θ*** compute the gradient of ***Loss*** with respect to ***θ***

Backpropagation

$$Loss = ((\sigma(xW_1 + b_1)W_2 + b_2) - y)^2$$

An algorithm for computing the gradient of a **compound** function as a series of **local, intermediate gradients**

Backpropagation

$$Loss = ((\sigma(xW_1 + b_1)W_2 + b_2) - y)^2$$

1. Identify intermediate functions (forward prop)
2. Compute local gradients
3. Combine with upstream error signal to get full gradient

Modularity - Simple Example

Compound function

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

Intermediate Variables

(forward propagation)

$$q = x + y$$

$$f = qz$$

Modularity - Neural Network Example

Compound function

$$Loss = ((\sigma(xW_1 + b_1)W_2 + b_2) - y)^2$$

Intermediate Variables

(forward propagation)

$$h_1 = xW_1 + b_1$$

$$z_1 = \sigma(h_1)$$

$$z_2 = z_1 W_2 + b_2$$

$$Loss = (z_2 - y)^2$$

Intermediate **Variables**

(forward propagation)

$$h_1 = xW_1 + b_1$$

$$z_1 = \sigma(h_1)$$

$$z_2 = z_1 W_2 + b_2$$

$$Loss = (z_2 - y)^2$$

Intermediate **Gradients**

(backward propagation)

$$\frac{\partial h_1}{\partial x} = W_1^T$$

$$\frac{\partial z_1}{\partial h_1} = \sigma'(h_1) = z_1 \circ (1 - z_1)$$

$$\frac{\partial z_2}{\partial z_1} = W_2^\top$$

$$\frac{\partial Loss}{\partial z_2} = 2(z_2 - y)$$

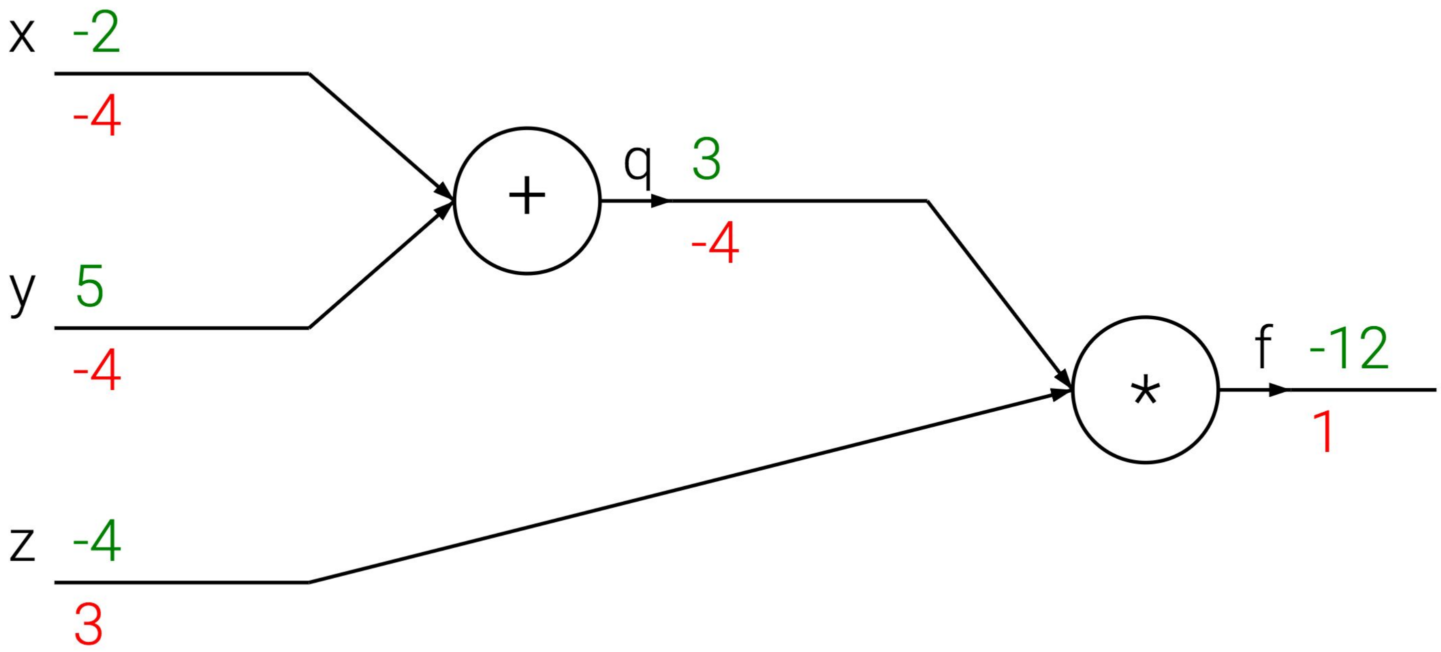
Chain Rule Behavior

$$\frac{d((f \circ g)(x))}{dx} = \frac{d(f(g(x)))}{d(g(x))} \frac{d(g(x))}{dx}$$

Key chain rule intuition:

Slopes multiply

Circuit Intuition



Matrix Calculus Primer

Scalar-by-Vector

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} \cdots \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Vector-by-Vector

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Matrix Calculus Primer

Scalar-by-Matrix

$$\frac{\partial y}{\partial A} = \begin{bmatrix} \frac{\partial y}{\partial A_{11}} & \frac{\partial y}{\partial A_{12}} & \cdots & \frac{\partial y}{\partial A_{1n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial A_{m1}} & \frac{\partial y}{\partial A_{m2}} & \cdots & \frac{\partial y}{\partial A_{mn}} \end{bmatrix}$$

Vector-by-Matrix

$$\frac{\partial y}{\partial A_{ij}} = \frac{\partial y}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial A_{ij}}$$

Vector-by-Matrix Gradients

$$\frac{\partial J}{\partial A_{ij}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial A_{ij}}$$

Let $\mathbf{z} = A\mathbf{x}$

$$\frac{\partial \mathbf{z}}{\partial A_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{'th element}$$

$$\frac{\partial J}{\partial A_{ij}} = \delta_i \mathbf{x}_j$$

$$\Rightarrow \frac{\partial J}{\partial A} = \delta^\top \mathbf{x}$$

$$\boxed{\begin{aligned} z &= Wx \\ \frac{\partial z}{\partial x} &= W \end{aligned}}$$

$$\boxed{\begin{aligned} z &= x \\ \frac{\partial z}{\partial x} &= I \end{aligned}}$$

$$\boxed{\begin{aligned} z &= xW \\ \frac{\partial z}{\partial x} &= W^\top \end{aligned}}$$

$$\boxed{\begin{gathered} z = Wx \quad \delta = \frac{\partial J}{\partial z} \\ \frac{\partial J}{\partial W} = \delta^\top x \end{gathered}}$$

$$\boxed{\begin{gathered} z = xW \quad \delta = \frac{\partial J}{\partial z} \\ \frac{\partial J}{\partial W} = x^\top \delta \end{gathered}}$$

Backpropagation Shape Rule

When you take gradients against a **scalar**

↓

The gradient at each intermediate step has **shape of denominator**

$$X \in \mathbb{R}^{m \times n} \iff \delta_X = \frac{\delta Scalar}{\delta X} \in \mathbb{R}^{m \times n}$$

Dimension Balancing

$$Z = XW \qquad [m \times w] \qquad \frac{\partial Loss}{\partial Z} = \delta$$

$$W \qquad [n \times w] \qquad \frac{\partial Loss}{\partial W} = \ ?$$

$$X \qquad [m \times n] \qquad \frac{\partial Loss}{\partial X} = \ ?$$

Dimension Balancing

$$Z = XW \qquad [m \times w] \qquad \frac{\partial Loss}{\partial Z} = \delta$$

$$W \qquad [n \times w] \qquad \frac{\partial Loss}{\partial W} = X^\top \delta$$

$$X \qquad [m \times n] \qquad \frac{\partial Loss}{\partial X} = \delta W^T$$

Dimension Balancing

Dimension balancing is the “cheap” but **efficient** approach to gradient calculations in most practical settings

Read *gradient computation notes* to understand how to derive matrix expressions for gradients from **first principles**

Activation Function Gradients

$z = \sigma(h)$ is an element-wise function on each index of ***h*** (scalar-to-scalar)

Officially, $$\frac{\partial z}{\partial h} = \begin{bmatrix} z_1(1-z_1) & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & z_n(1-z_n) \end{bmatrix}$$

Diagonal matrix represents that z_i and h_j have no dependence if $i \neq j$

Activation Function Gradients

Element-wise multiplication (hadamard product) corresponds to **matrix product with a diagonal matrix**

$$\frac{\partial Loss}{\partial h} = \frac{\partial Loss}{\partial z} \begin{bmatrix} z_1(1-z_1) & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & z_n(1-z_n) \end{bmatrix}$$

$$= \frac{\partial Loss}{\partial z} \circ (z \circ (1-z))$$

Backprop Menu for Success

1. Write down variable graph
2. Compute derivative of cost function
3. Keep track of error signals
4. Enforce shape rule on error signals
5. Use matrix balancing when deriving over a linear transformation

As promised: A matrix example...

$$z_1 = XW_1$$

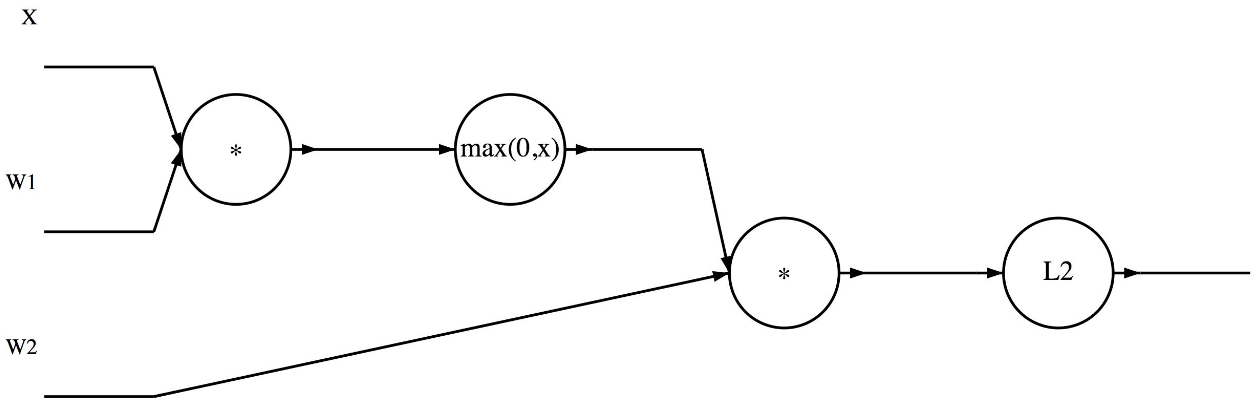
$$h_1 = \text{ReLU}(z_1)$$

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

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

$$\frac{\partial L}{\partial W_2} = \quad ?$$

$$\frac{\partial L}{\partial W_1} = \quad ?$$



As promised: A matrix example...

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

$$\frac{\partial L}{\partial \hat{y}} = 2\hat{y}$$
$$\boxed{\frac{\partial L}{\partial W_2} = h_1^\top \frac{\partial L}{\partial \hat{y}}}$$
$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} W_2^\top$$
$$\frac{\partial h_1}{\partial z_1} = \frac{\partial L}{\partial h_1} \circ I[h_1 > 0]$$
$$\boxed{\frac{\partial z_1}{\partial W_1} = x^\top \frac{\partial h_1}{\partial z_1}}$$

```
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)
```