Neural Network and Backpropagation Review

Natural Language Processing

(based on revision of Chris Manning Lectures)

Announcement

- **TA announcements (if any)...**

Suggested Readings

1. Stanford matrix calculus notes
2. Stanford review of differential calculus
3. Stanford CS231n notes on network architectures
4. Stanford CS231n notes on backprop
5. Stanford derivatives, Backpropagation, and Vectorization
6. Learning Representations by Backpropagating Errors (seminal Rumelhart et al. backpropagation paper)

Name Entity Recognition

Named Entity Recognition (NER)

- **NER**: find and classify names in text, for example:

Last night , Paris Hilton wowed in a sequin gown .
PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .
PER PER LOC LOC LOC DATE DATE

- Uses
  - Tracking mentions of particular entities in documents
  - For question-answering, answers are usually named entities
- Often followed by Named Entity Linking/Canonicalization into Knowledge Base

Simple NER: Window classification using binary logistic classifier

- **Idea:** classify each word in its context window of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window

Example: Classify “Paris” as +/- `LOC` in context of sentence with window length 2:

the museums in Paris are amazing to see

$$X_{window} = [X_{museums} \quad X_{in} \quad X_{Paris} \quad X_{are} \quad X_{amazing}]^T$$

Resulting vector $X_{window} \in R^{5d}$, a column vector

To classify all words: run classifier for each class on vector on each word in the sentence

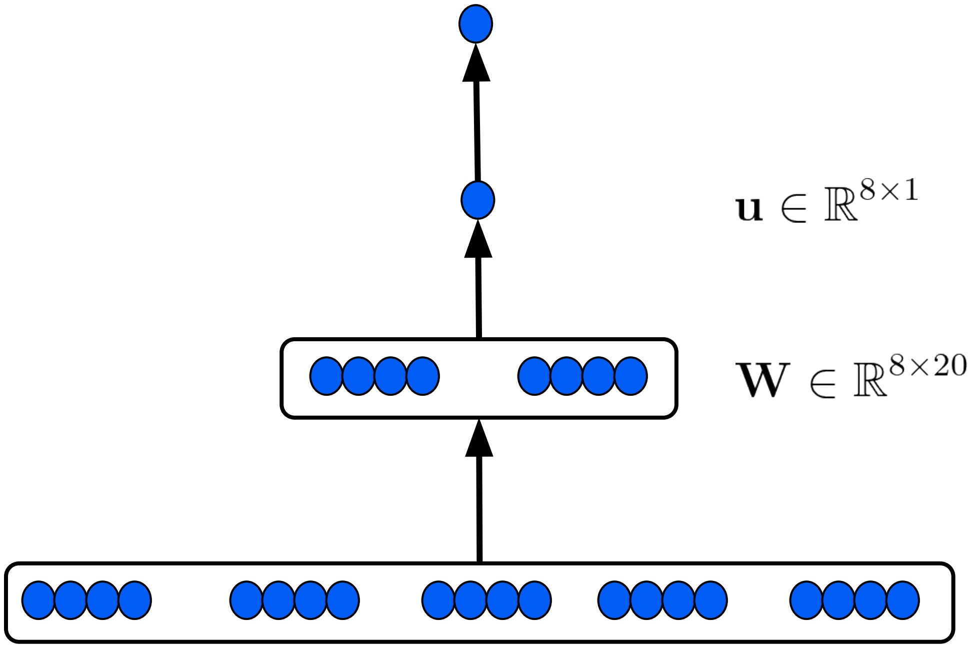
Simple NER: Window classification using binary logistic classifier

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

$$s = \mathbf{u}^{\top}\mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x} \quad (input)$$



$\mathbf{u} \in \mathbb{R}^{8\times 1} \quad s \in \mathbb{R}^{1}$

$\mathbf{W} \in \mathbb{R}^{8\times 20} \quad \mathbf{b} \in \mathbb{R}^{8} \quad \mathbf{h} \in \mathbb{R}^{8}$

$\mathbf{x} \in \mathbb{R}^{20}$

$X_{window} = [X_{museums} \quad X_{in} \quad X_{Paris} \quad X_{are} \quad X_{amazing}]^T$

Maximum Margin Objective Function

- Let's called the score computed for the "**true**" labeled window "*Museums in Paris are amazing*" as *s* where $s = \mathbf{u}^\top f(\mathbf{W}\mathbf{x} + \mathbf{b})$
- Let's called the score for "**false**" label window, e.g., "*Not all museums in Paris*" as s_c where $s_c = \mathbf{u}^\top f(\mathbf{W}\mathbf{x}_c + \mathbf{b})$
- We want to maximize (s - s_c) or **minimize (*s_c* - *s*)**
- We want to further ensure that error is only computed if $s_c > s$ or $s_c - s > 0$. We only care that the "true" data point have higher score, thus, the objective function is **min J = max(*s_c* - *s*, 0)**
- This is a big risky. To create a margin of safety, we want the "true" labeled data point to score higher than the "false" labeled data by some margin Δ. In other words, we want error to be ($s - s_c < \Delta$). If the Δ = 1, then the objective function is **min J** = **max(1 + *s_c* - *s*, 0)**

Matrix Calculus

Computing Gradients by Hand

Matrix calculus: Fully vectorized gradients

- "Multivariable calculus is just like single-variable calculus if you use matrices"
- Much faster and more useful than non-vectorized gradients
- Support by NumPy and PyTorch
- But doing a non-vectorized gradient can be good for intuition
- Learning them allows you to deeply understand gradient-related problems, e.g., vanishing gradients

Gradients

- Given a function with 1 output and 1 input $f(x) = x^3$
- It's gradient (slope) is its derivative $\frac{df}{dx} = 3x^2$
- "How much will the output change if we change the input a bit?"
  - At x = 1, it changes about 3 times as much: $1.03^3 = 1.03$
  - At x = 4, it changes about 48 times as much: $4.01^3 = 64.48$

Gradients

- Given a function with 1 output and *n* inputs

$$f(\mathbf{x}) = f(x_1, x_2, \cdots, x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right]$$

Jacobian Matrix: Generalization of the Gradient

- Given a function with ***m* outputs** and *n* inputs

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2, \cdots, x_n), \cdots, f_m(x_1, x_2, \cdots, x_n)]$$

- It’s Jacobian is an ***m* x *n*** matrix of partial derivatives

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \ddots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \qquad \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

Chain Rule

- For composition of one-variable function: **multiply derivatives**

$$z = 3y$$
$$y = x^2$$
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x) = 6x$$

- For multiple variables at once: **multiply Jacobians**

$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \cdots$$

Example Jacobian: Elementwise activation function

$$\mathbf{h} = f(\mathbf{z}) \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \qquad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$

To figure it out, it's useful to think about single-variable calculus

$$h_i = f(z_i)$$

The derivative is simply:

$$\left(\frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i)$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

Thus, if we take all derivatives:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \operatorname{diag}(f'(\mathbf{z}))$$

Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}$$

$$\frac{\partial}{\partial \mathbf{b}}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

$$\frac{\partial}{\partial \mathbf{u}}(\mathbf{u}^\top \mathbf{h}) = \mathbf{h}^\top$$

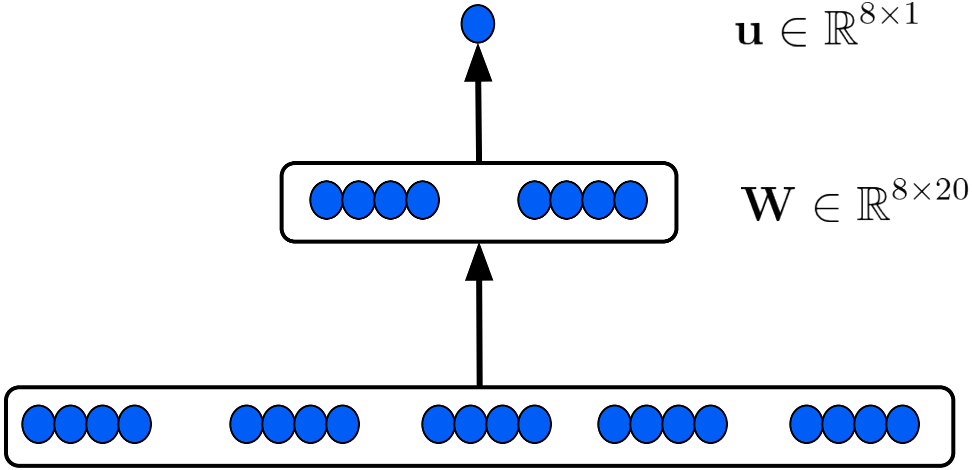
Back to our Neural Net!

Let's find $\dfrac{\partial s}{\partial \mathbf{b}}$

$$s = \mathbf{u}^\top \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x} \quad (input)$$



$\mathbf{u} \in \mathbb{R}^{8\times 1}$ $\quad s \in \mathbb{R}^{1}$

$\mathbf{W} \in \mathbb{R}^{8\times 20}$ $\quad \mathbf{b} \in \mathbb{R}^{8}$ $\quad \mathbf{h} \in \mathbb{R}^{8}$

$\mathbf{x} \in \mathbb{R}^{20}$

$X_{window} = [X_{museums} \quad X_{in} \quad X_{Paris} \quad X_{are} \quad X_{amazing}]^T$

Apply the chain rule

$$\boxed{s = \mathbf{u}^\top \mathbf{h}}$$

$$\boxed{\mathbf{h} = f(\mathbf{z})}$$

$$\boxed{\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}}$$

$$\mathbf{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \mathbf{b}} = \underbrace{\boxed{\frac{\partial s}{\partial \mathbf{h}}}}_{\mathbf{u}^\top} \underbrace{\boxed{\frac{\partial \mathbf{h}}{\partial \mathbf{z}}}}_{\text{diag}(f'(\mathbf{z}))} \underbrace{\boxed{\frac{\partial \mathbf{z}}{\partial \mathbf{b}}}}_{I}$$

$$= \mathbf{u}^\top \circ f'(\mathbf{z})$$

$$\in \mathbb{R}^{1\times 8}$$

Hadamard product
(element wise product)

Re-using computation

Let's find $\frac{\partial s}{\partial \mathbf{W}}$

Using the chain rule again:

$$\frac{\partial s}{\partial \mathbf{W}} = \boxed{\frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

$$\frac{\partial s}{\partial \mathbf{b}} = \boxed{\frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

$$\delta = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \in \mathbb{R}^{1 \times 8}$$

δ is the local error signal

Let's avoid duplicated computation.....

Derivative with respect to Matrix: Output shape

Let's find $\frac{\partial s}{\partial \mathbf{W}}$ look like? $\mathbf{W} \in \mathbb{R}^{n \times m}$

- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to then do $\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_\theta J(\theta)$

Instead, we use the shape convention, i.e., **the shape of the gradient is the shape of the parameters**

- So $\frac{\partial s}{\partial \mathbf{W}}$ is *n* by *m*: $\begin{bmatrix} \frac{\partial s}{\partial w_{11}} & \ddots & \frac{\partial s}{\partial w_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial w_{n1}} & \cdots & \frac{\partial s}{\partial w_{nm}} \end{bmatrix}$

Derivative with respect to Matrix

Since $\frac{\partial s}{\partial \mathbf{W}} = \delta \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$ thus, plugging what we have already learned, we get

$$\frac{\partial s}{\partial \mathbf{W}} = \delta^\top \quad \mathbf{x}^\top$$

$$[n \times m] \ [n \times 1][1 \times m]$$

$$= \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, \cdots, x_m] = \begin{bmatrix} \delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m \end{bmatrix}$$

- Shape checking is a useful trick for checking your work!

What shape should derivatives be?

Similarly $\frac{\partial s}{\partial \mathbf{b}} = \mathbf{h}^\top \circ f'(\mathbf{z}) \in \mathbb{R}^{1 \times 8}$ is a row vector

- But shape convention says our gradient should be a column vector because *b* is a column vector

Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementation easy)

- **Always use shape convention**
  - Use Jacobian form and then reshape to follow the shape convention at the end, e.g., here we simply perform a transpose should be enough to make this gradient a column vector.

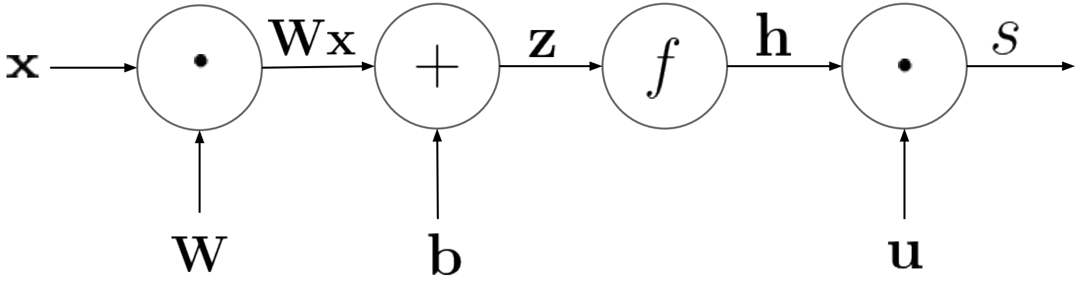
Computation graphs and Backpropagation

Computation graphs and backpropagation

$$s = \mathbf{u}^\top \mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x} \quad \text{(input)}$$

Software represents our neural network equations as **graph**

- **Why**: reusing computations (which we hinted earlier)

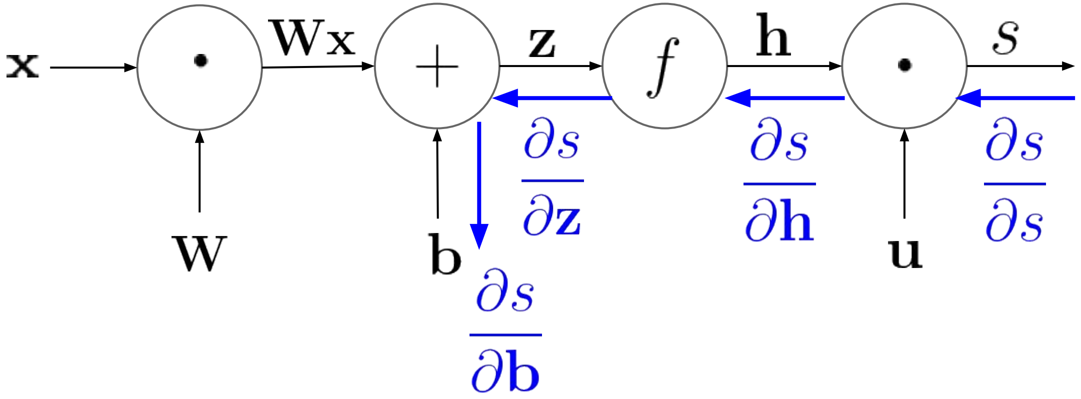


Computation graphs and backpropagation

$$s = \mathbf{u}^{\top}\mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x} \quad \text{(input)}$$

Then go backwards along edges

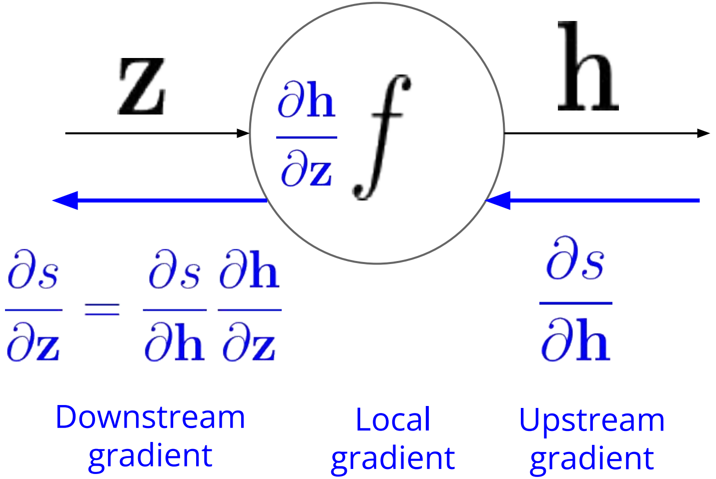
- Pass along **gradients**



Backpropagation: Single node

$$s = \mathbf{u}^\top \mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x} \quad \text{(input)}$$

- Node receives an “**upstream gradient**”
- Goal is to pass on the correct “**downstream gradient**”
  - [downstream gradient] = [upstream gradient] x [local gradient]
- Each node has a **local gradient**
  - The gradient of its output with respect to its input



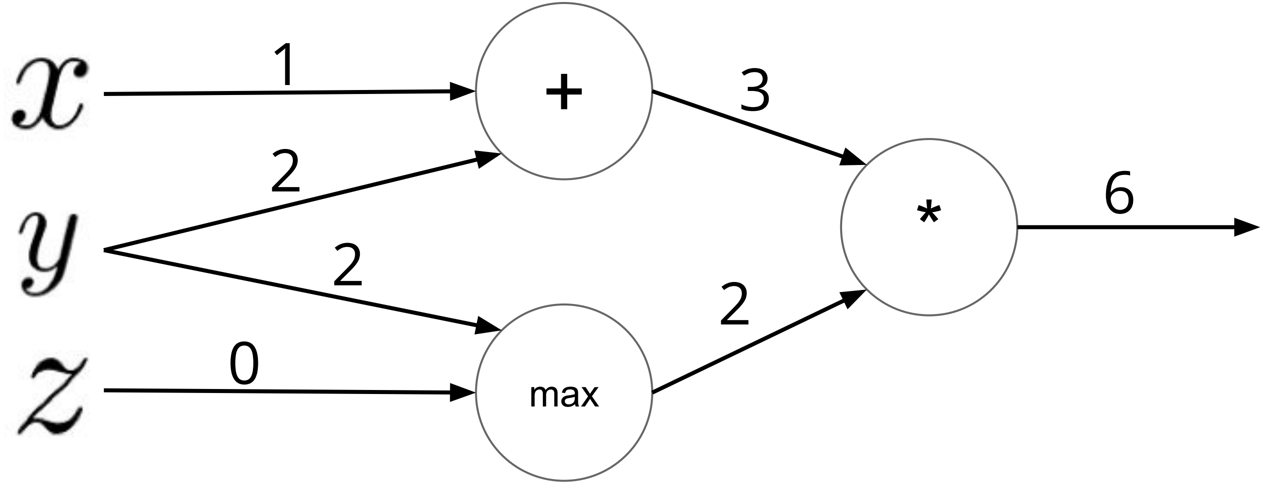
An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients



An Example

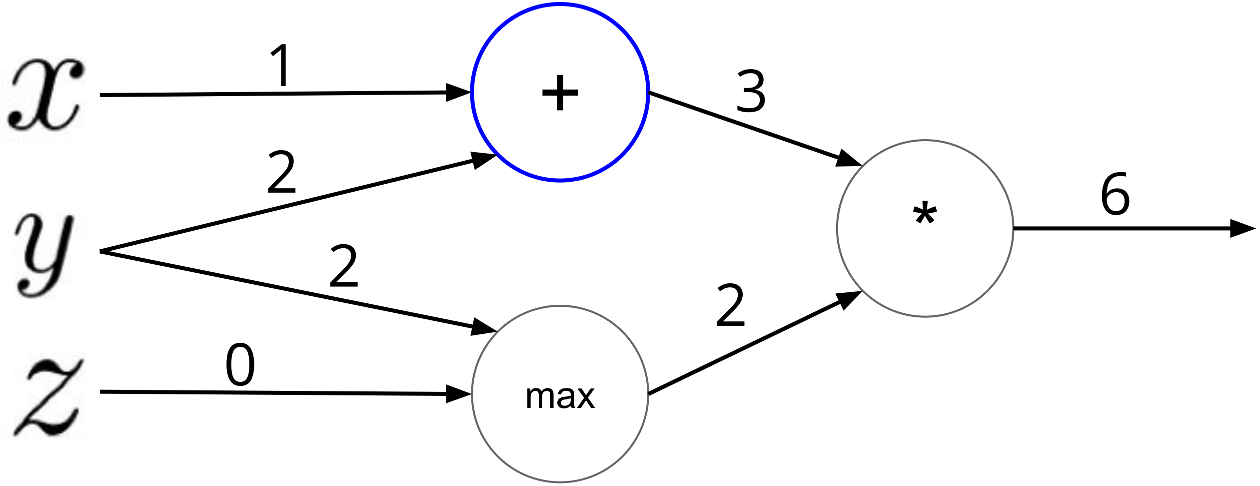
$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

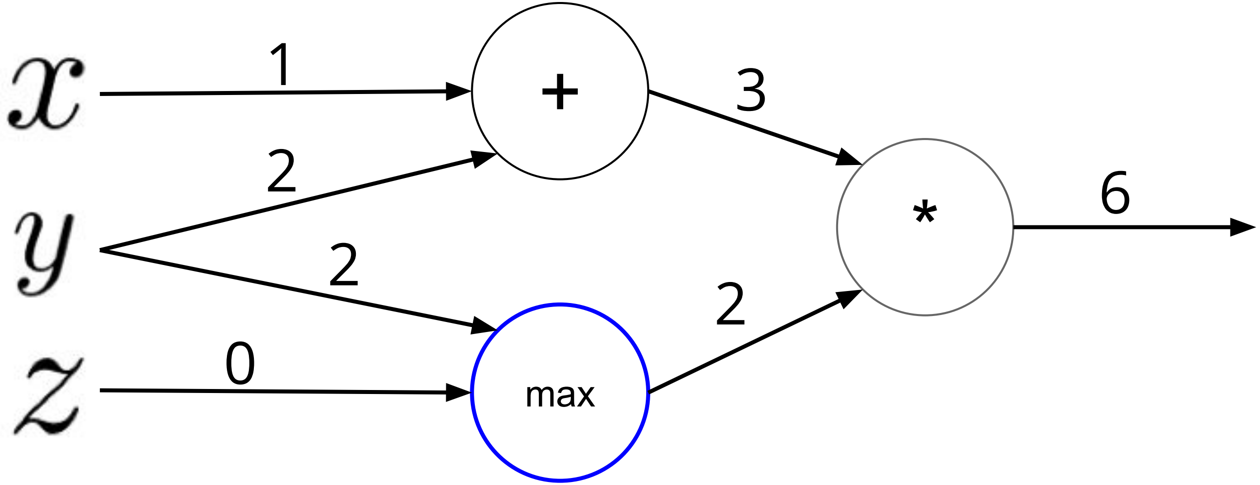
Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\boxed{\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0}$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

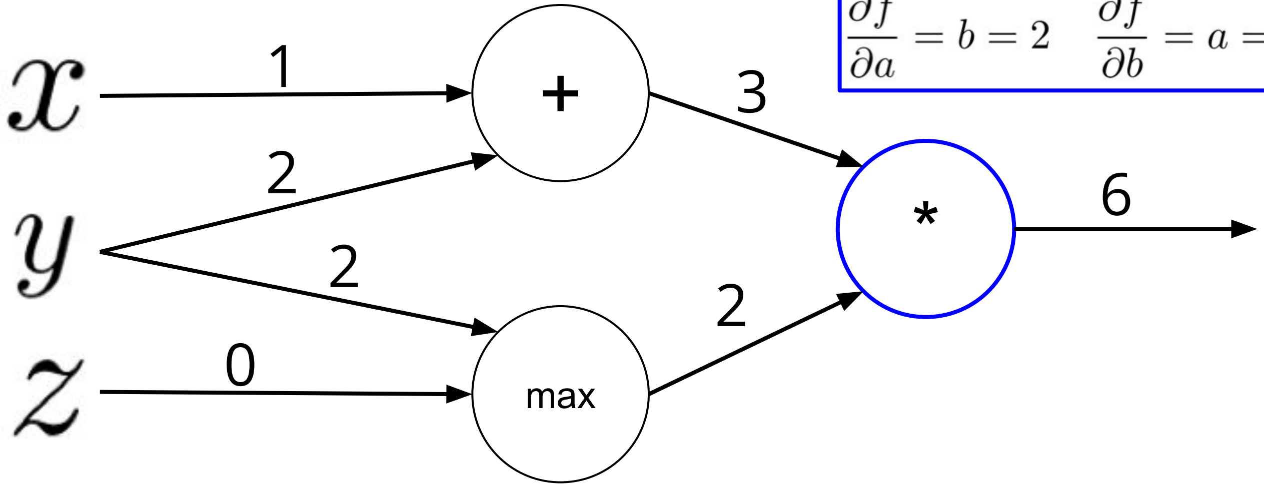
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\boxed{\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3}$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

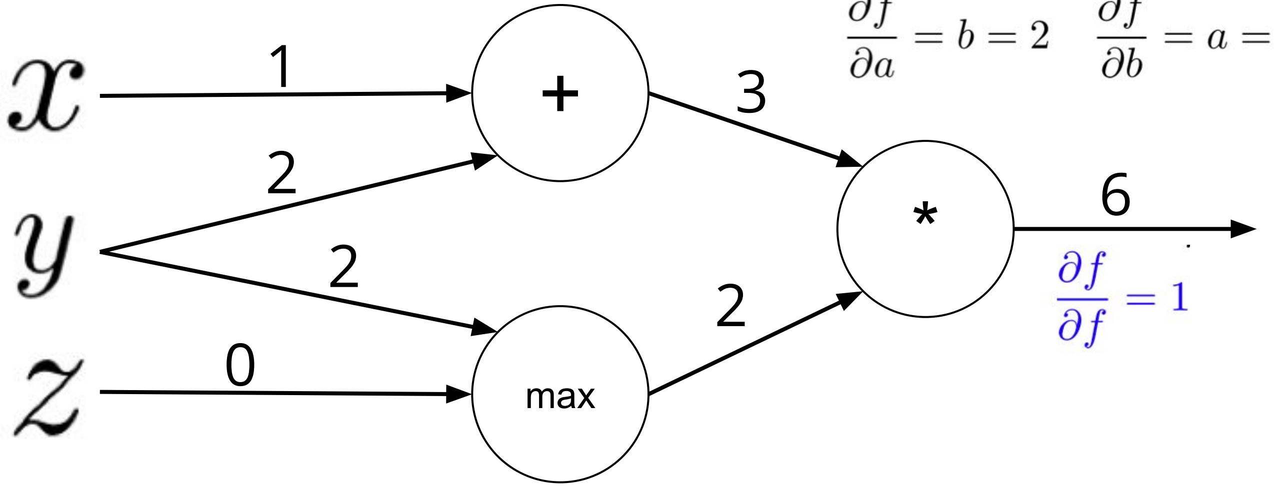
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

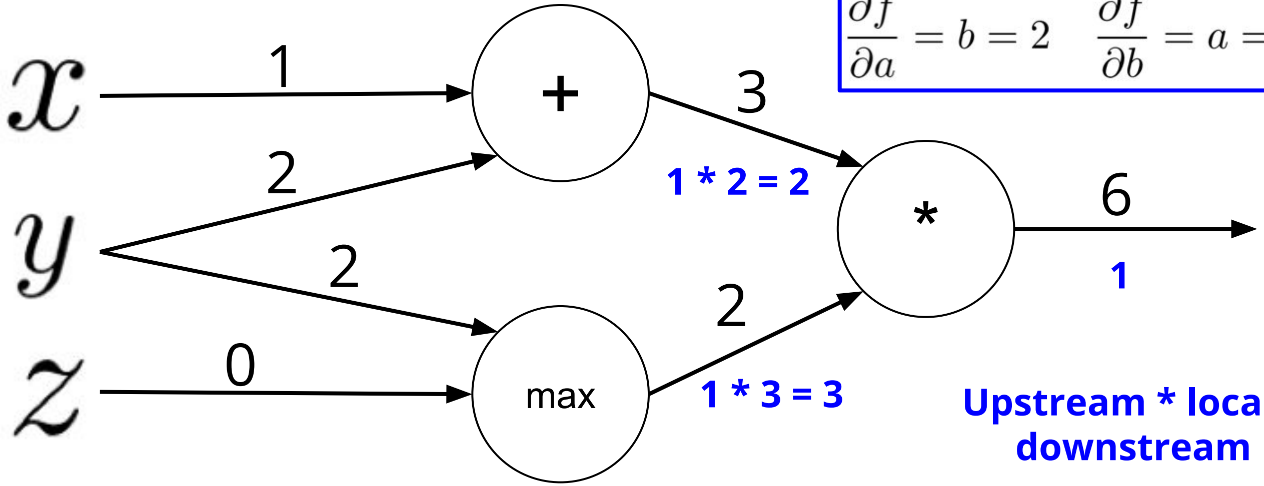
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\boxed{\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3}$$



Upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

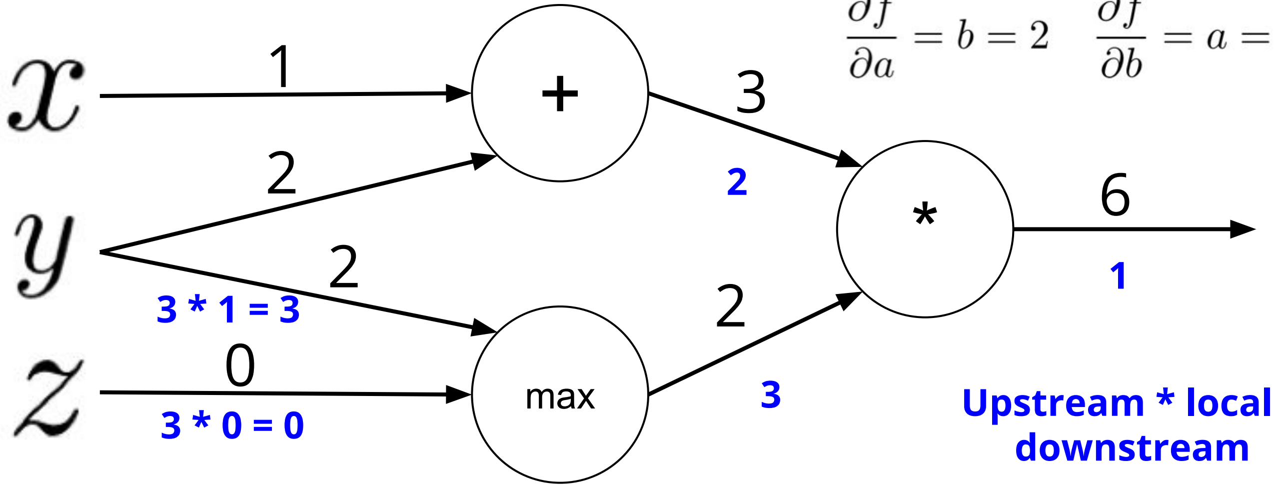
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\boxed{\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0}$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



Upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

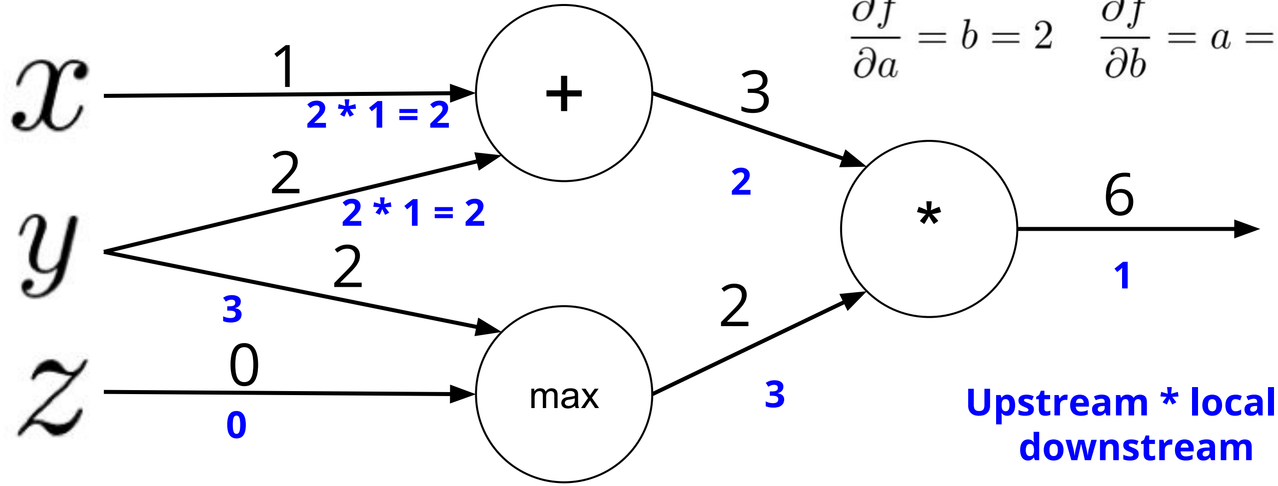
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\boxed{\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1}$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



Upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$\boxed{\begin{aligned} a &= x + y \\ b &= \max(y, z) \\ f &= ab \end{aligned}}$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

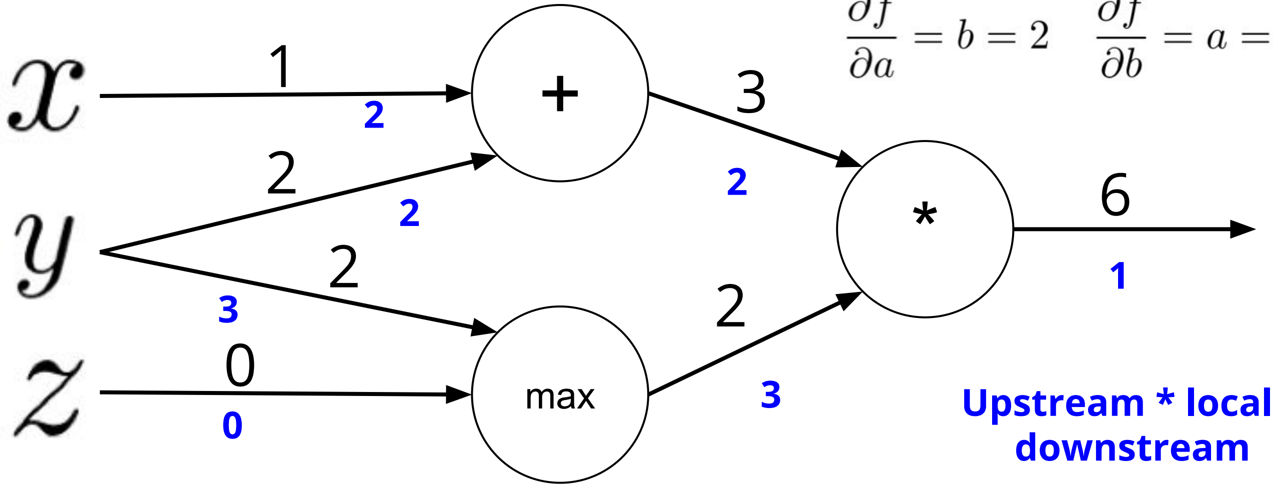
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$

$$\frac{\partial f}{\partial x} = 2$$

$$\frac{\partial f}{\partial y} = 3 + 2 = 5$$

$$\frac{\partial f}{\partial z} = 0$$



Upstream * local = downstream

Back-prop in general computation graph

1. Fprop: visit nodes in topological order
   a. Compute accordingly
2. Bprop:
   a. Initialize output gradient = 1
   b. Visit nodes in reverse order
   c. Pass along the gradients just like what we did

Done correctly, **big O() complexity of fprop and bprop is the same**

- In PyTorch, everything is done for you!
  - **So why study?** Very useful for debugging or creating your own theory / architecture.

Summary

- Performing **vectorized gradients** are much faster and more useful than **non-vectorized** gradients
  - To understand, it's useful to do single-variable calculus first
- For chain rule, the derivatives are simply the **multiplication of Jacobians**
- **Always follow shape convention**
  - That is, the gradient should be the same shape as the parameter itself
- Maintaining gradients in **graph form** allows us to backprop efficiently
  - Good new: PyTorch already does that for you!