自然语言处理技术

Neural Net Fundamentals
神经网络基础

计算机科学与技术学院
杨沐昀 孙承杰

主要内容

- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

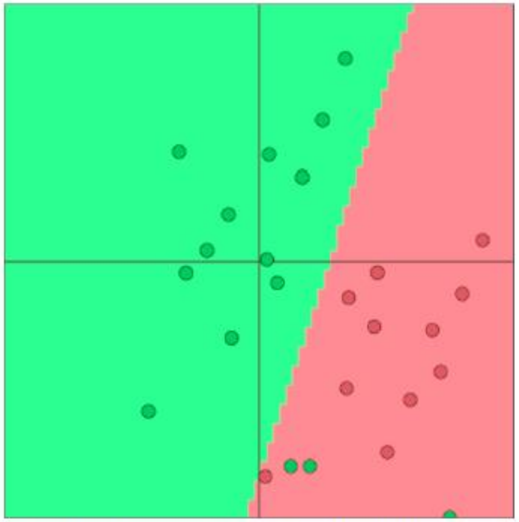
主要内容

- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

分类问题定义与表示

- We have a training dataset consisting of N samples$\{x_i,\ yi\}_{i=1}^{N}$
- x_i are inputs, e.g. words (indices or vectors!), sentences, documents, etc.
    - Dimension d
- y_i are labels (one of C classes) we try to predict, for example:
    - classes: sentiment, named entities
    - other words
    - later: multi-word sequences

分类方法



Visualizations with ConvNetJS by Karpathy!
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

- Training data:
  - Fixed 2D word vectors to classify
  - Using softmax/logistic regression
  - Linear decision boundary
- Traditional ML/Stats approach: assume x_i are fixed, train (i.e., set) softmax/logistic regression weights $W^{C \times d}$ to determine a decision boundary (hyperplane) as in the picture.
- Method: For each x, predict:

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

softmax分类方法

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C}\exp(W_c.x)}$$

可以将预测函数分解为两个步骤:

1. 将矩阵W 的第y行与向量x 相乘，计算f_c (c = 1, … , C)

$$W_y.x = \sum_{i=1}^{d} W_{yi}x_i = f_y$$

2.应用softmax函数得到归一化概率:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)} = \text{softmax}(f_y)$$

使用cross-entropy作为损失函数

- 对于每个训练样本 (x, y)， 我们的目标是最大化正确类别y的概率

- 或者说最小化正确类别y的负对数概率（negative log probability）

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

cross-entropy损失函数

- “cross entropy” 是一个信息论里的概念
- 如果正确的类别概率分布为p， 模型得到的类别概率分布为q， 则cross entropy可以被定义为:

$$H(p,q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

- 假设正确的类别概率分布为正确类别对应的概率为1，其他类别对应的概率为0，即p=[0,…,0,1,0,…0]。那么，cross entropy 中唯一留下的项就是正确类别的负对数概率。

cross-entropy损失函数

- 在整个数据集$\{xi,\ yi\}_{i=1}^{N}$ 上的cross-entropy损失函数可以表示为：

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

- 为了表示方便，我们将使用矩阵表示

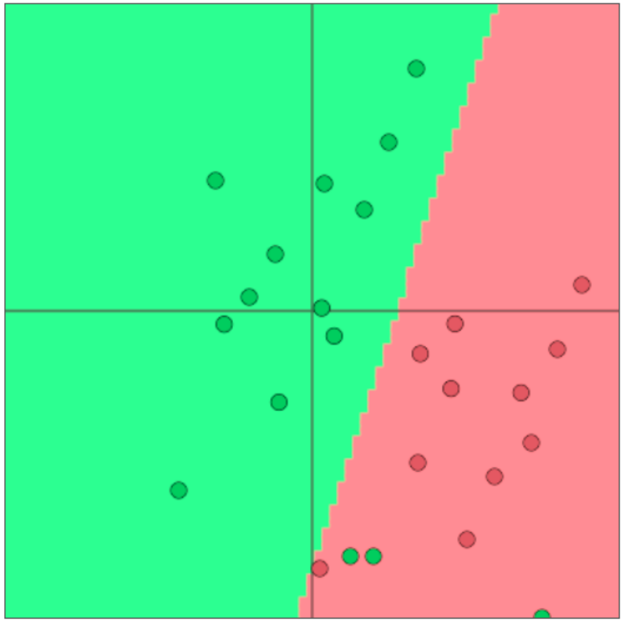
$$f = Wx$$

来代替之前的表示 $f_y = f_y(x) = W_{y\cdot}x = \sum_{j=1}^{d} W_{yj}x_j$

传统机器学习的优化方法

- 机器学习的参数θ 一般可以表示成如下形式:

$$\theta = \begin{bmatrix} W_{\cdot 1} \\ \vdots \\ W_{\cdot d} \end{bmatrix} = W(:) \in \mathbb{R}^{Cd}$$



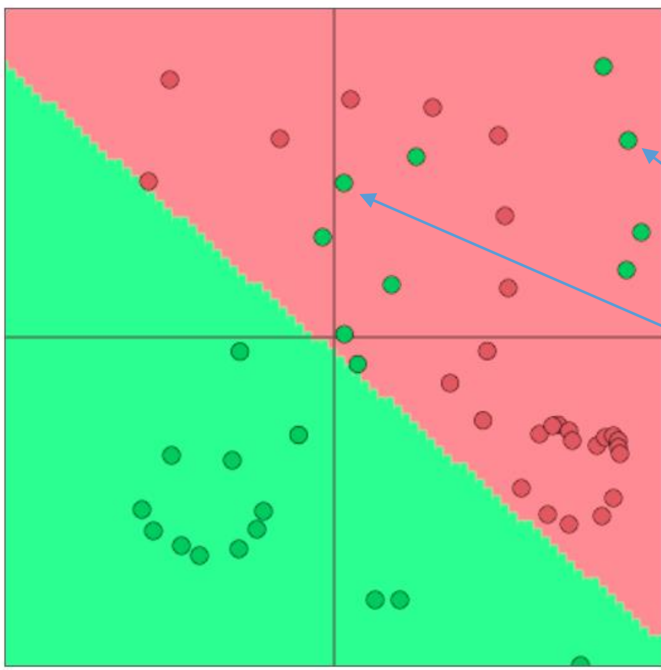
Visualizations with ConvNetJS by Karpathy

- 通过下列计算过程来更新分类面

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \end{bmatrix} \in \mathbb{R}^{Cd}$$

神经网络分类器（Neural Network Classifiers）

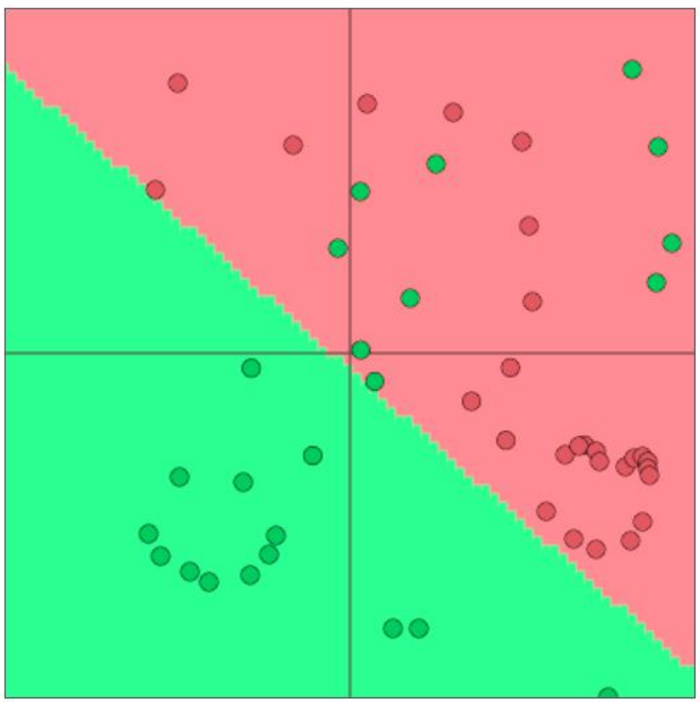
- Softmax只能给出线性分类边界，分类能力受限

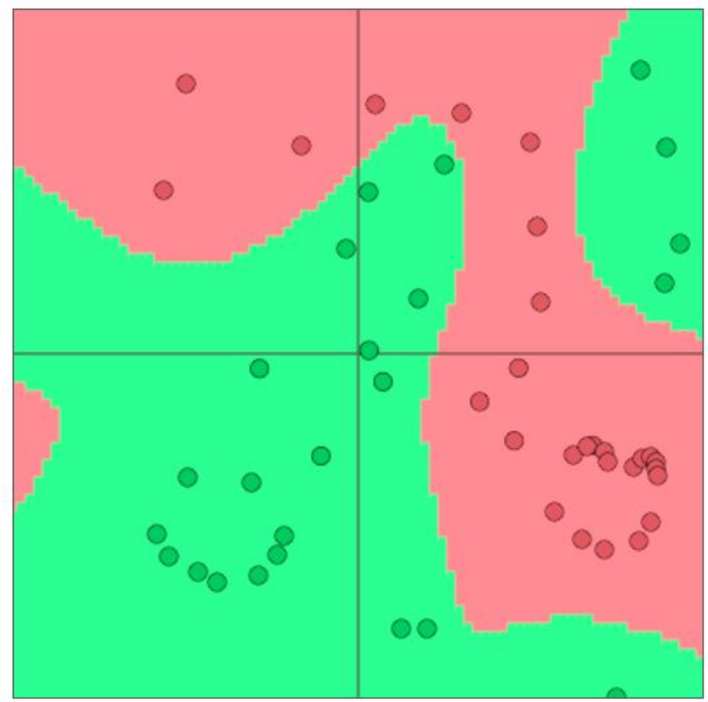


如何把这些样本分对？

神经网络分类器（Neural Network Classifiers）

- Neural networks 可以表示更复杂的函数，得到非线性分类边界

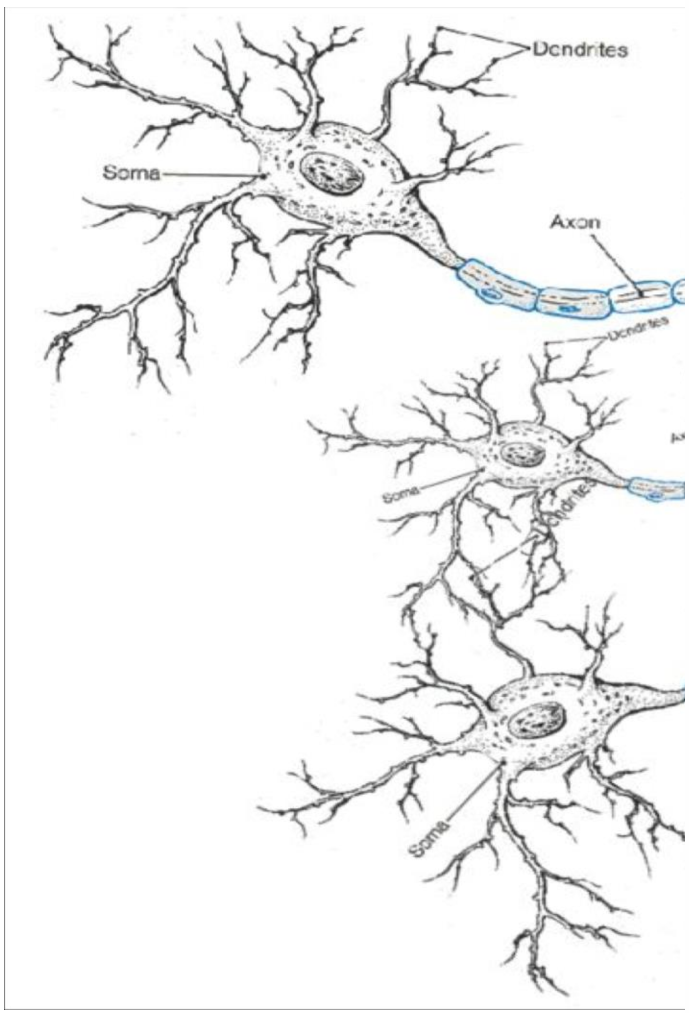


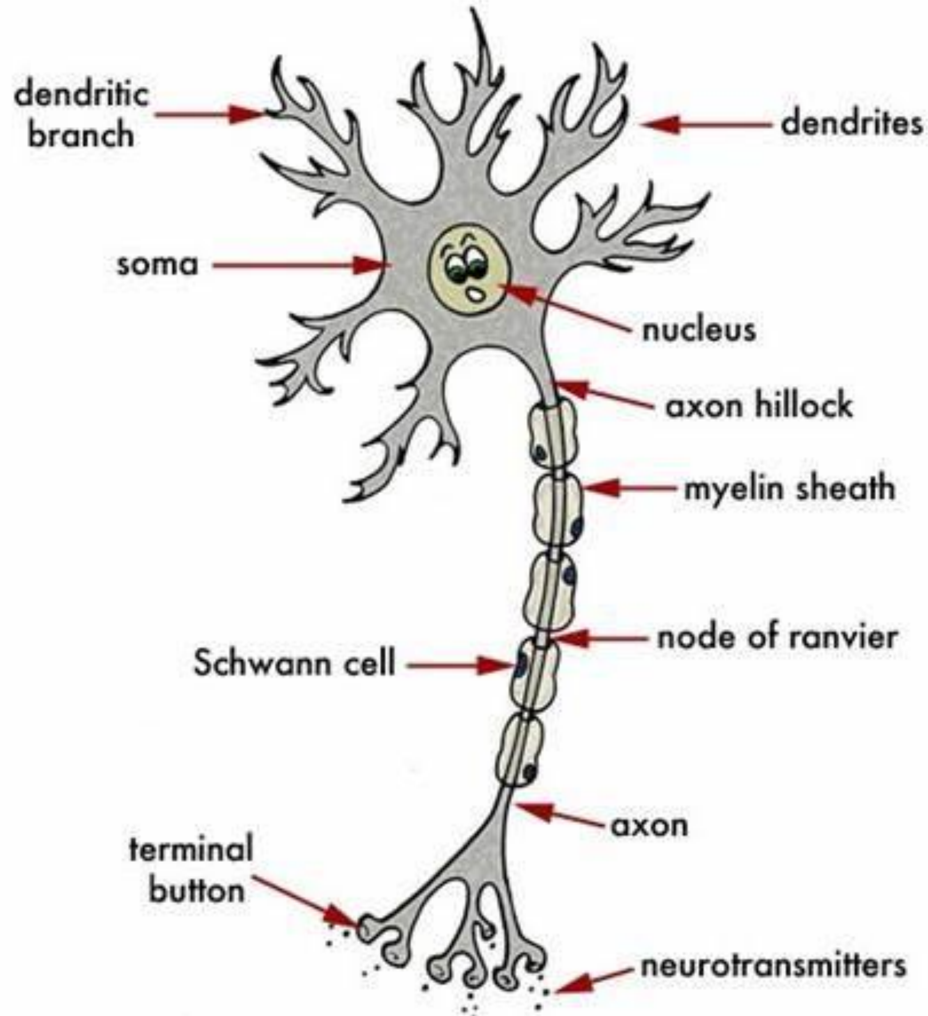


主要内容

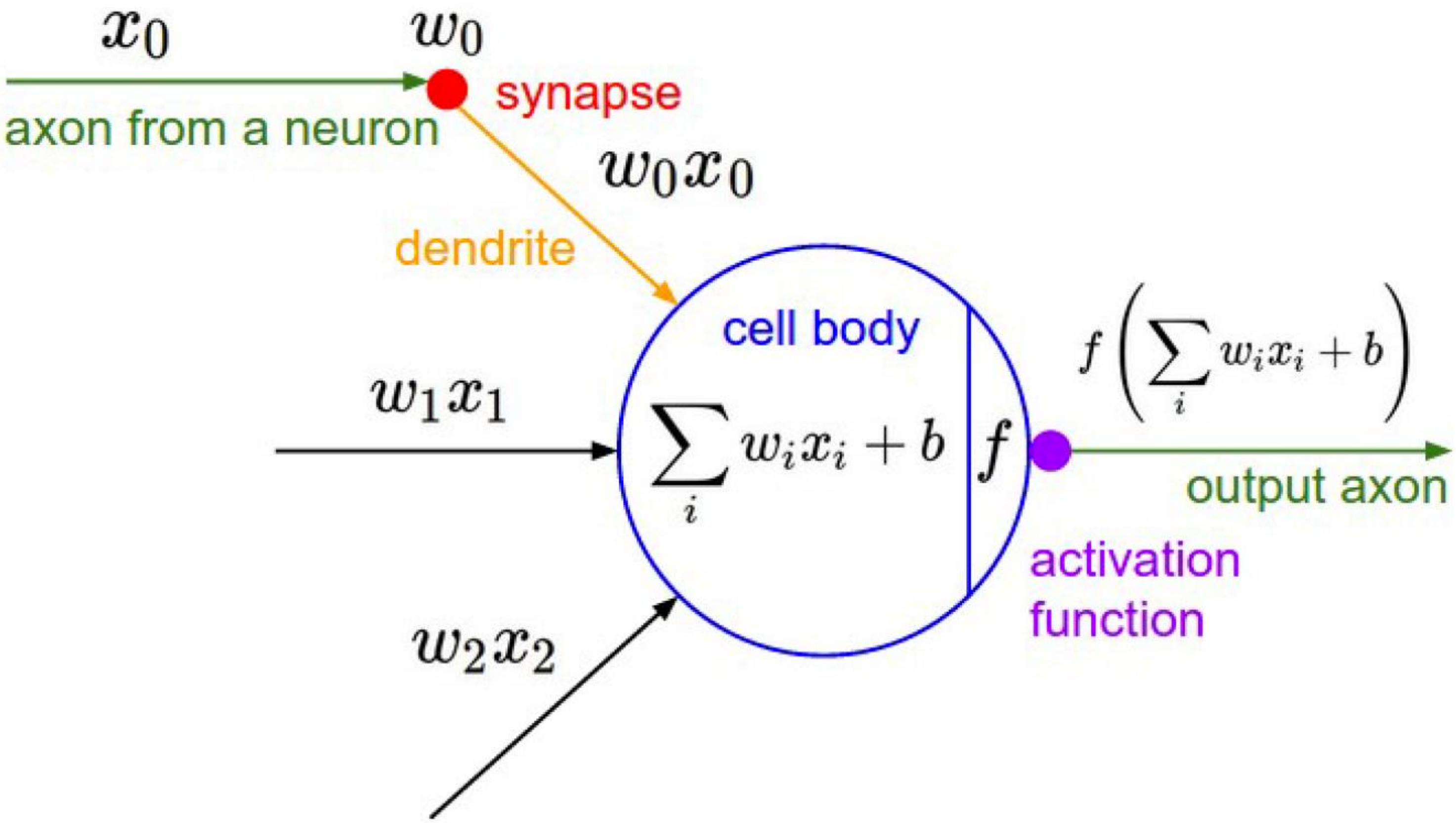
- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

神经计算（Neural computation）





人工神经元（artificial neuron）

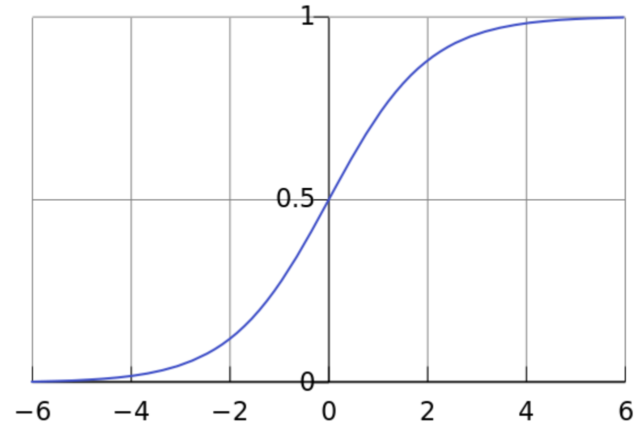


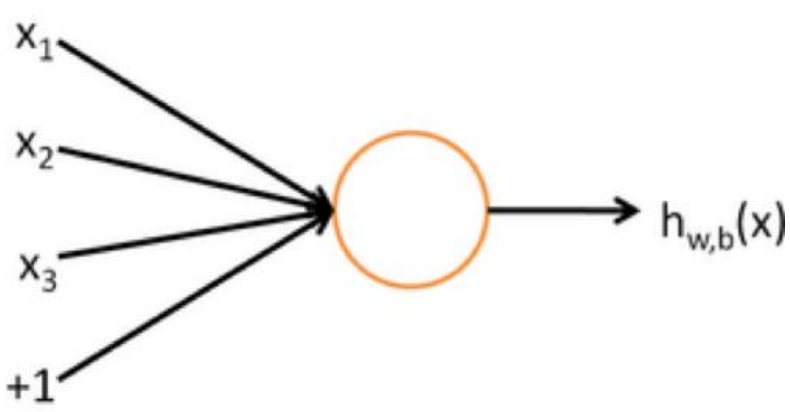
神经元与logistic regression单元

f = nonlinear activation (e.g. sigmoid), w = weights, b = bias, h = hidden, x = inputs

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

$$f(z) = \frac{1}{1+e^{-z}}$$

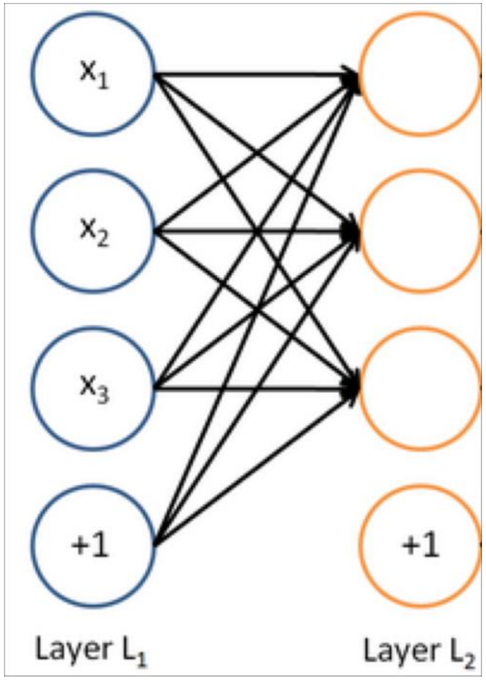




w, b are the parameters of this neuron
i.e., this logistic regression model

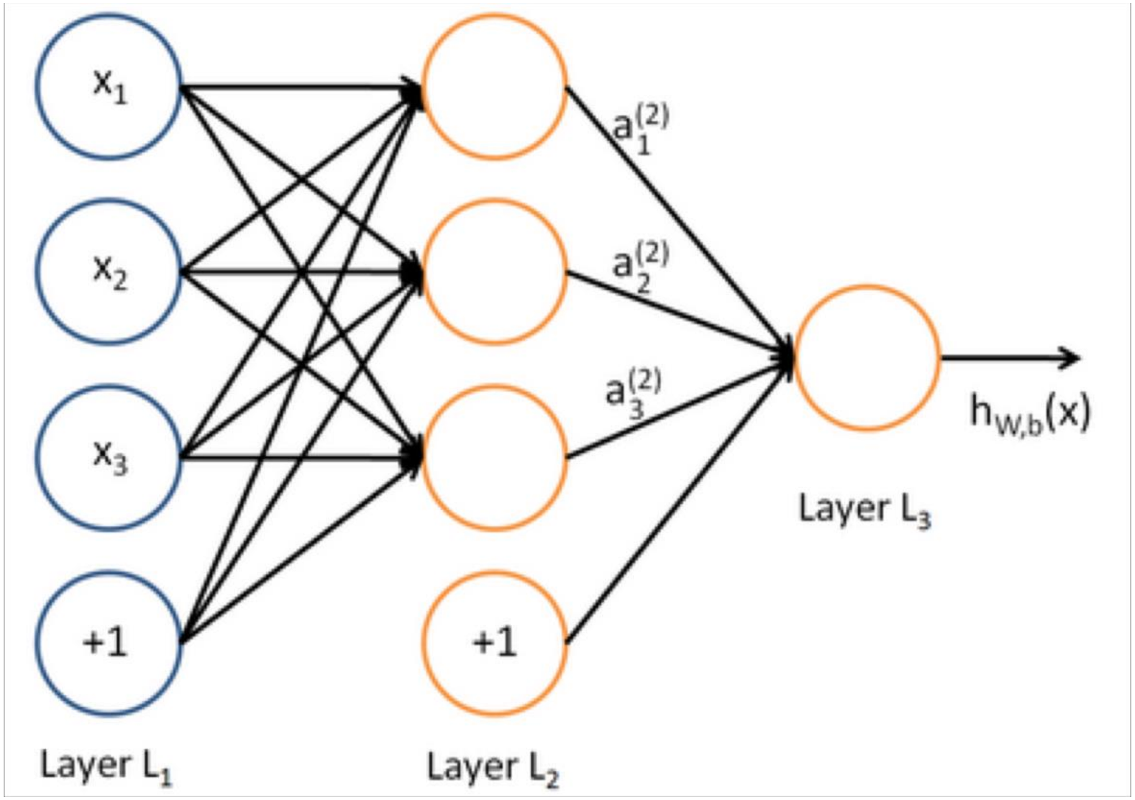
神经网络（neural network）

- 一个神经网络相当于多个logistic regression单元在同时运行



神经网络

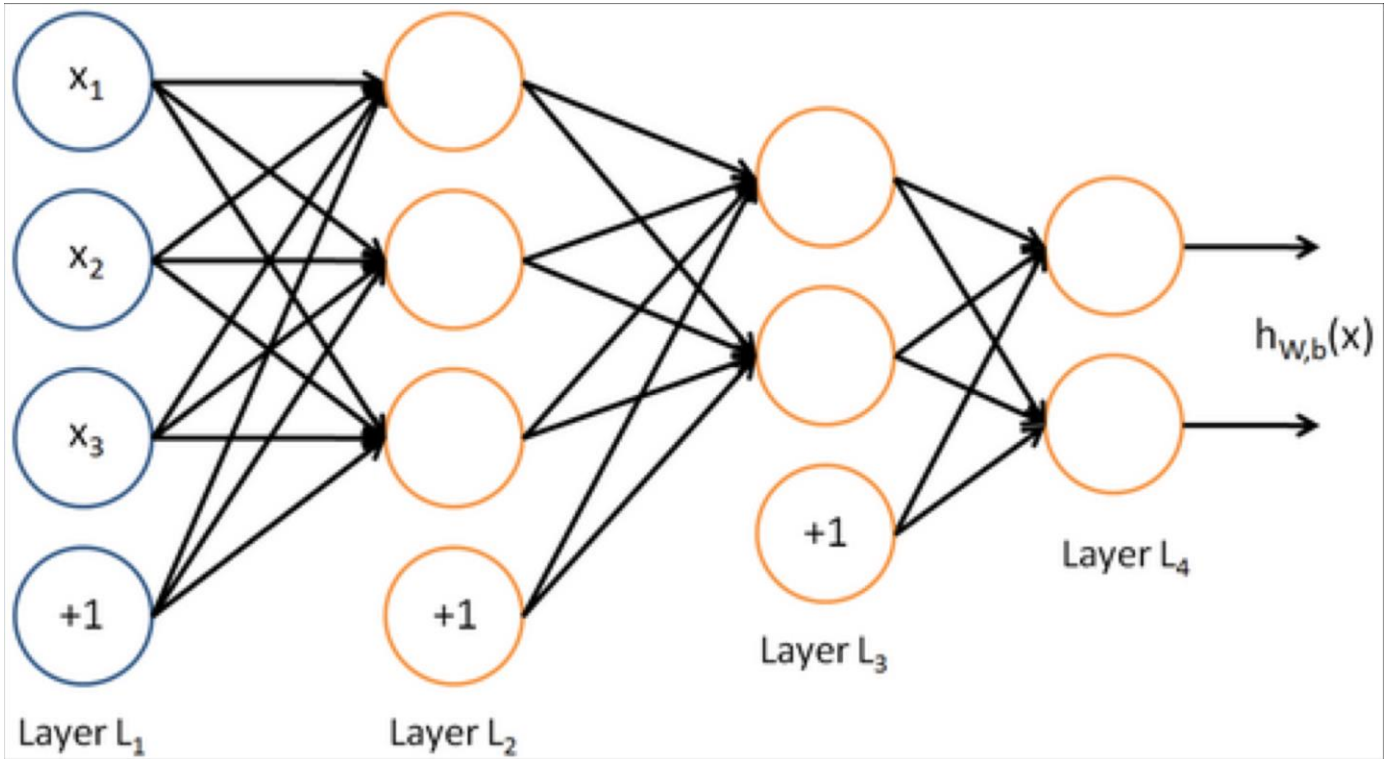
- 多层神经网络



损失函数会指导中间隐藏层变量的取值，以便更好地预测下一层的目标。

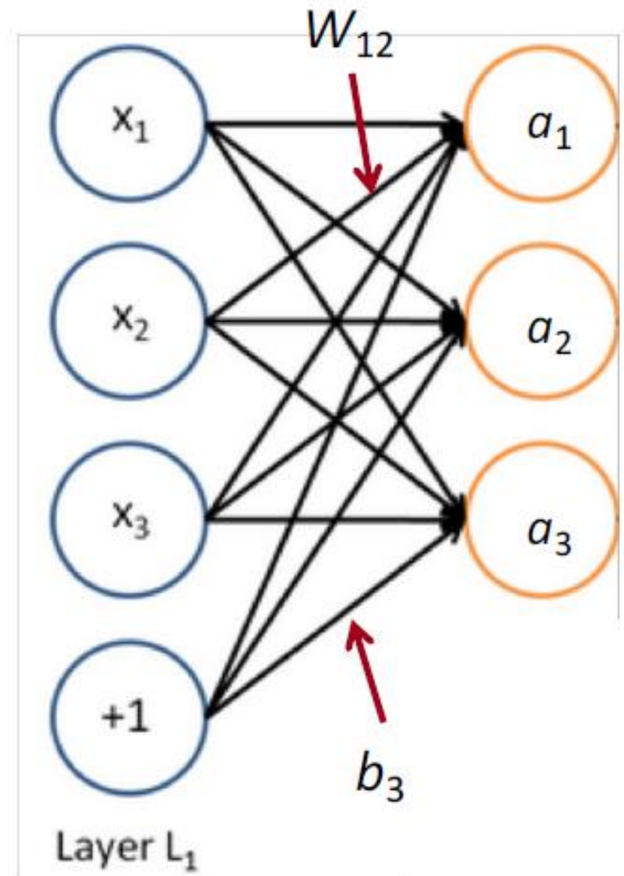
神经网络

- 多层神经网络



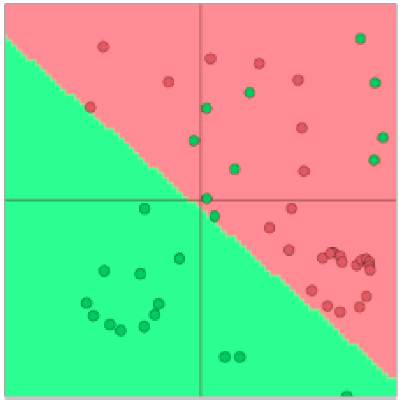
神经网络的矩阵表示

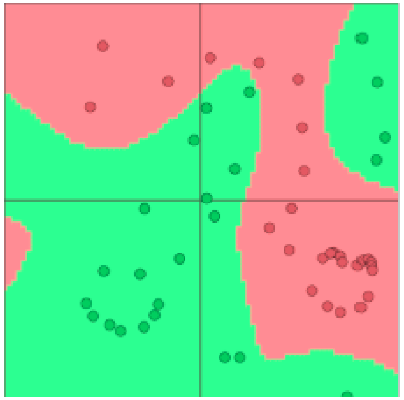
- 对于L$_2$,
  - $a_1 = f(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1)$
  - $a_2 = f(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2)$
  - …
- 使用矩阵表示，可以写成
  - $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
  - $\boldsymbol{a} = f(\boldsymbol{z})$
- 激活函数f是应用于每个元素上的
  - $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$



为什么需要f为非线性函数

- Example: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networks can't do anything more than a linear transform
  - Extra layers could just be compiled down into a single linear transform: $W_1W_2x = Wx$
  - With more layers, they can approximate more complex functions!





主要内容

- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

神经网络应用例子：基于二分类的地名识别

- Example: Not all museums in Paris are amazing .
- Here: one true window, the one with Paris in its center and all other windows are “corrupt” in terms of not having a named entity location in their center.

museums in Paris are amazing

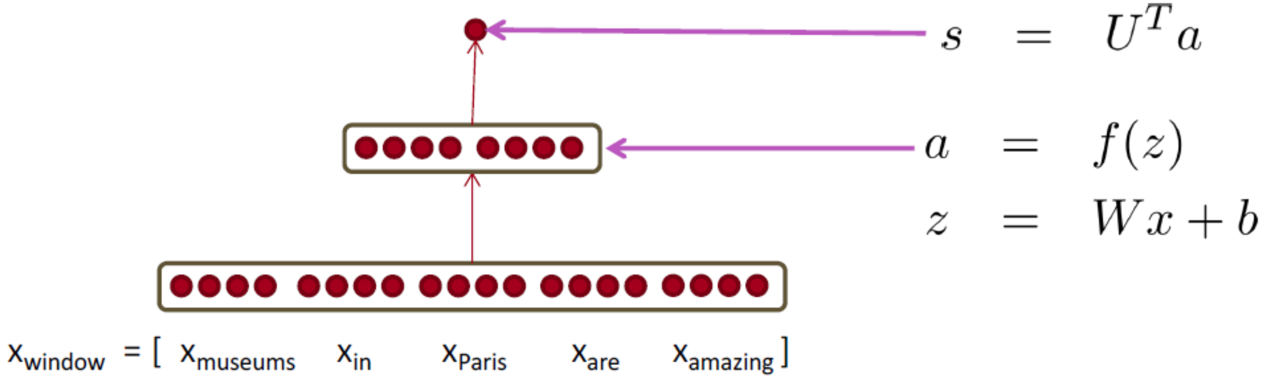
- “Corrupt“ windows are easy to find and there are many: Any window whose center word isn’t specifically labeled as NER location in our corpus

Not all museums in Paris

前向计算（Neural Network Feed-forward Computation）

- 使用一个3层神经网络来给输入的句子片段x打分

$$score(x) = U^T a \in \mathbb{R}$$



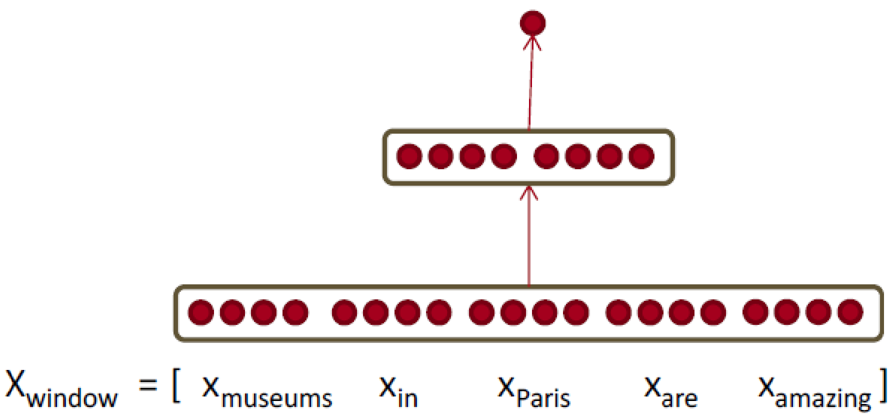
- $s = score$("museums in Paris are amazing”)

$$s = U^T f(Wx + b)$$

$$x \in \mathbb{R}^{20\times1}, W \in \mathbb{R}^{8\times20}, U \in \mathbb{R}^{8\times1}$$

中间层的直觉解释

- 中间层学到的是输入的词向量之间的非线性交互（non-linear interactions between the input word vectors）



- 例如: 当第一个词是 “museums”时， “in”位于第二个词的位置其作用会变得重要

max-margin损失函数

- 训练目标的思想:使正例样本的分数变大，使负利样本的分数变低(直到足够好为止)
- s = score(museums in Paris are amazing)
- s_c = score(Not all museums in Paris)
- 最小化

$$J = \max(0, 1 - s + s_c)$$

- J不是处处可导的，但它是连续的→我们可以使用随机梯度下降（SGD）。

max-margin损失函数

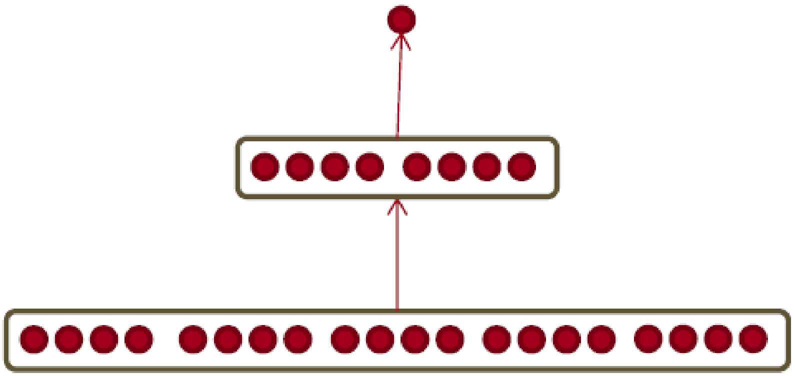
- 对于1个输入窗口（句子片段）

$$J = \max(0, 1 - s + s_c)$$

- 每个地名位于中心的窗口（正例）的score要比没有地名位于中心的窗口（负例）大1
- 完整的优化目标函数：对于每个正例，构造多个负例。然后对所有窗口的 J 求和。

计算score的简单神经网络

- $s = \boldsymbol{u}^T \boldsymbol{h}$
- $\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$
- $\boldsymbol{x}$ (输入)



随机梯度下降

- 参数更新公式

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

α 为步长或者学习率（*step size* or *learning rate*）

- 如何计算$\nabla_{\theta} J(\theta)$
  - 手动
  - 后向传播算法（the backpropagation algorithm）

手动计算梯度

- 多变量求导
- Matrix calculus: Fully vectorized gradients
  - Much faster and more useful than non-vectorized gradients
  - But doing a non-vectorized gradient can be good practice

主要内容

- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

导数计算回顾

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- It’s gradient (slope) is its derivative

$$\frac{df}{dx} = 3x^2$$

导数计算回顾

- Given a function with 1 output and n inputs

$$f(\boldsymbol{x}) = f(x_1, x_2, \dots, x_n)$$

- It’s gradient is a vector of partial derivatives with respect to each input

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

雅可比矩阵（Jacobian Matrix）

- Given a function with m outputs and n inputs

$$\boldsymbol{f}(\boldsymbol{x}) = [f_{1}(x_1, x_2, \dots, x_n), \cdots, f_{m}(x_1, x_2, \dots, x_n)]$$

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

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\boxed{\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}}$$

链式规则（Chain Rule）

- For one-variable functions: 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

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

雅可比矩阵示例：Elementwise activation Function的求导

$$\boldsymbol{h} = f(\boldsymbol{z}) \qquad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = ?$$

$$h_i = f(z_i)$$

- Function has n outputs and n inputs → n by n Jacobian

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) = \begin{cases} f'(z_i) & if\ i = j \\ 0 & if\ i \neq j \end{cases}$$

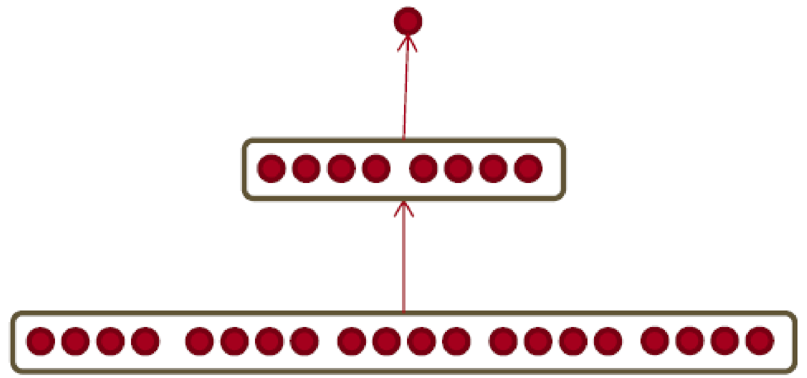
$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = diag(f'(\boldsymbol{z}))$$

其它雅可比矩阵

- $\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})=\boldsymbol{W}$
- $\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})=\boldsymbol{I}$ (单位矩阵)
- $\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h})=\boldsymbol{h}^T$

计算score的简单神经网络

- $s = \boldsymbol{u}^T \boldsymbol{h}$
- $\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$
- $\boldsymbol{x}$ (输入)



- 如何求 $\frac{\partial s}{\partial \boldsymbol{b}}$？

1. 把等式分解

- $s = \boldsymbol{u}^T \boldsymbol{h}$

- $\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ ⇒ $\boldsymbol{h} = f(\boldsymbol{z})$

  $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- $\boldsymbol{x}$ (输入)

2. 应用链式规则

- $s = \boldsymbol{u}^T \boldsymbol{h}$
- $\boldsymbol{h} = f(\boldsymbol{z})$
- $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
- $\boldsymbol{x}$ (输入)
- $\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial z} \frac{\partial z}{\partial \boldsymbol{b}} = \boldsymbol{u}^T \circ diag\big(f'(\boldsymbol{z})\big) I$
  $= \boldsymbol{u}^T \circ diag\big(f'(\boldsymbol{z})\big)$

计算重用

- 如果我们想计算 $\frac{\partial s}{\partial \boldsymbol{W}}$
- 应用链式规则

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

- 令 $\delta = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$，则 $\frac{\partial s}{\partial \boldsymbol{W}} = \delta \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$

对矩阵求导

- $\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{W}}$ 的结果是什么形状？ $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, nm inputs: 1 by nm Jacobian?
- 不方便进行参数更新 $\theta^{new} = \theta^{old} - \propto \nabla_{\theta} J(\theta)$
- 解决办法: 导数的形状(shape)于参数的形状保持一致
- $\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{W}}$ 的结果被表示成$n \times m$的矩阵 $\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$

对矩阵求导

- 前面我们已经得到 $\frac{\partial s}{\partial \boldsymbol{W}} = \delta \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$
  - δ 的计算前面已经完成
  - $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$应该是$\boldsymbol{x}$　$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- $\frac{\partial s}{\partial \boldsymbol{W}}$的结果可以写成$\boldsymbol{\delta}^{\boldsymbol{T}}\boldsymbol{x}^{\boldsymbol{T}}$

$\boldsymbol{\delta}$ is local error signal at $\boldsymbol{z}$

$\boldsymbol{x}$ is local input signal

为什么转置?

$$\frac{\partial s}{\partial \boldsymbol{W}} \quad = \quad \boldsymbol{\delta}^T \quad \boldsymbol{x}^T$$

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

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, ..., x_m] = \begin{bmatrix} \delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m \end{bmatrix}$$

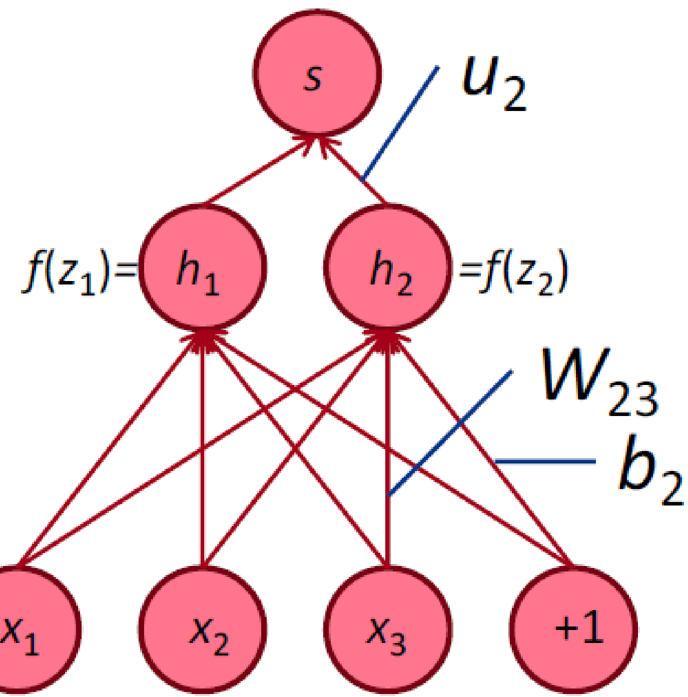
导数应该是什么形状?

- Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementing SGD easy)
  - We expect answers to follow the **shape convention**
  - But Jacobian form is useful for computing the answers
- Two options:
  - 1. Use Jacobian form as much as possible, reshape to follow the convention at the end:
  - 2. Always follow the convention
    - Look at dimensions to figure out when to transpose and/or reorder terms.

面向反向传播的求导

- $\frac{\partial s}{\partial \boldsymbol{W}} = \delta \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \delta \frac{\partial}{\partial \boldsymbol{W}} \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
- 如何对$\boldsymbol{W}$中的每一个权重W_{ij}求导?
- W_{ij} 只用于计算 z_i
  - 例如W_{23} 只用于计算 z_2
- $\frac{\partial z_i}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \boldsymbol{W}_i \boldsymbol{x} + b_i$

$$= \frac{\partial}{\partial W_{ij}} \sum_{k=1}^{m} W_{ik} x_k + b_i = x_j$$



面向反向传播的求导

- 所以得分s对单个W_{ij}求导的结果为
- $\frac{\partial s}{\partial W_{ij}} = \delta_i \; x_j$

Error signal
from above

Local gradient
signal

- 因此，对于整个矩阵$\boldsymbol{W}$的求导结果可以写成

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

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

求导过程中的小提示

- Carefully define your variables and keep track of their dimensionality!
- Chain rule! If y = f(u) and u = g(x), i.e., y = f(g(x)), then:

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

- For the top softmax part of a model: First consider the derivative wrt f_c when $c = y$ (the correct class), then consider derivative wrt f_c when c != y (all the incorrect classes)
- Work out element-wise partial derivatives if you're getting confused by matrix calculus!
- Use Shape Convention. Note: The error message $\boldsymbol{\delta}$ that arrives at a hidden layer has the same dimensionality as that hidden layer

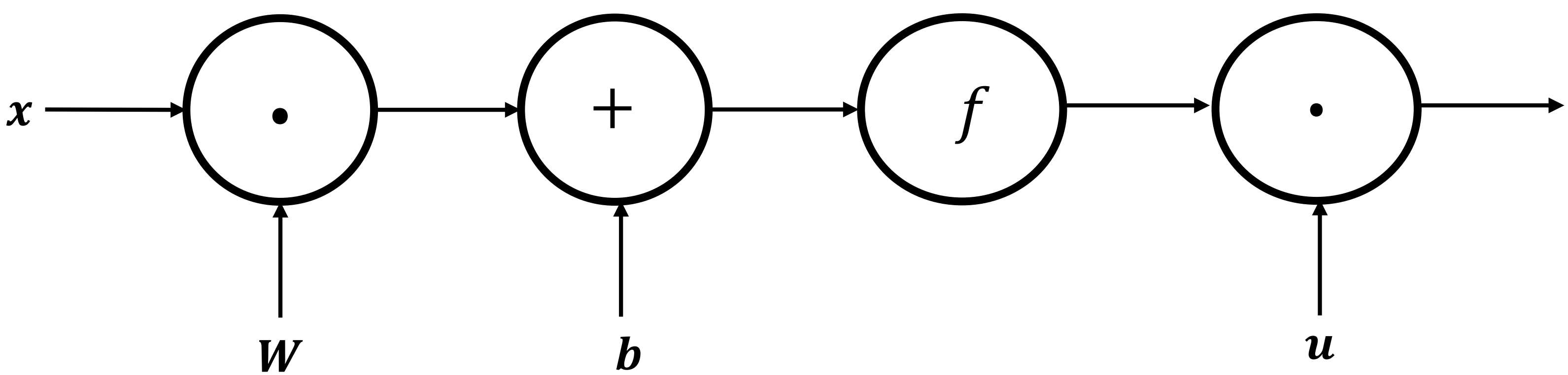
主要内容

- 线性分类方法与神经网络
- 神经元与神经网络的表示
- 神经网络在NLP中的应用示例
- 神经网络中导数的计算
- 计算图与反向传播

计算图与反向传播（Computation Graphs and Backpropagation）

- 我们可以用一个图表示前面提出的神经网络
  - Source nodes: inputs
  - Interior nodes: operations

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \text{ (输入)}$$

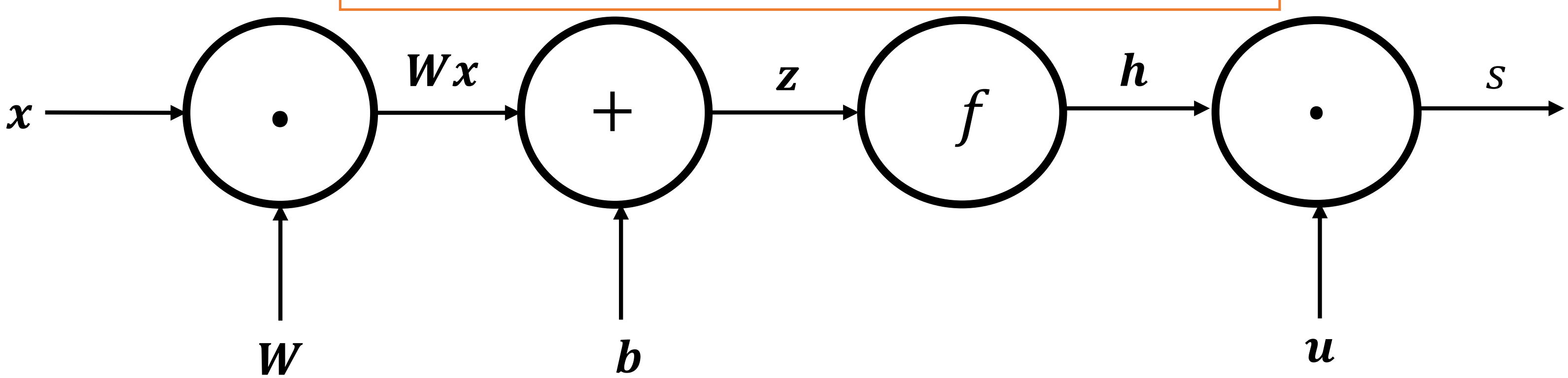


计算图与反向传播（Computation Graphs and Backpropagation）

- 我们可以用一个图表示前面提出的神经网络
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$\boldsymbol{x}$ (输入)

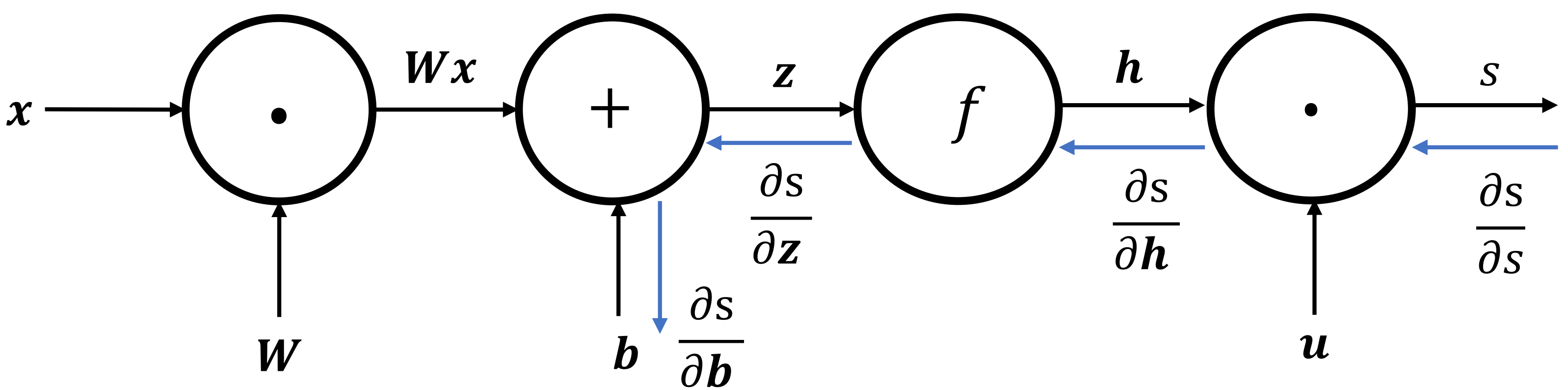
“Forward Propagation”



反向传播（Backpropagation）

- $Go\ backwards\ along\ edges$
  - Pass along **gradients**

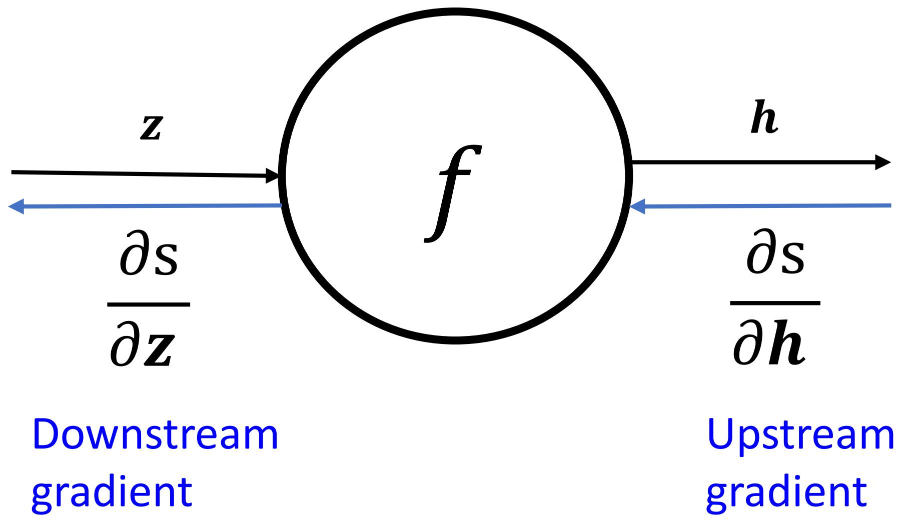
$$s = \boldsymbol{u}^T\boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x}\ (输入)$$



反向传播中的单个节点

- Node receives an “upstream gradient”
- Goal is to pass on the correct “downstream gradient”

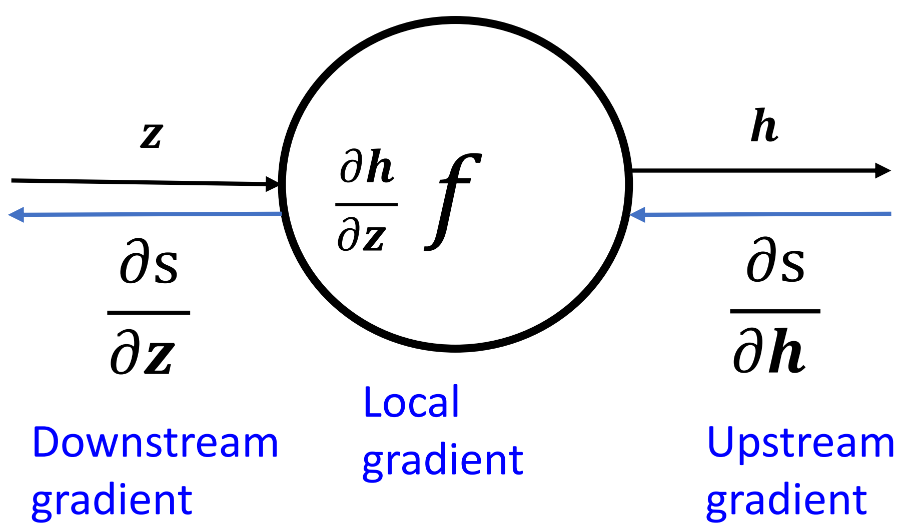
$$\boxed{\boldsymbol{h} = f(\boldsymbol{z})}$$



反向传播中的单个节点

- Each node has a **local gradient**
  - The gradient of it’s output with respect to it’s input

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

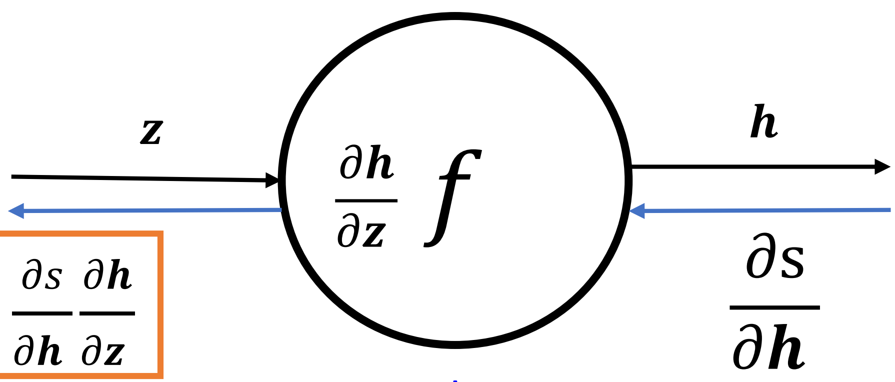


反向传播中的单个节点

- Each node has a **local gradient**
  - The gradient of it’s output with respect to it’s input

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

- [downstream gradient] = [upstream gradient] x [local gradient]



Chain rule

$$\boxed{\frac{\partial s}{\partial \boldsymbol{z}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}}$$

Downstream gradient

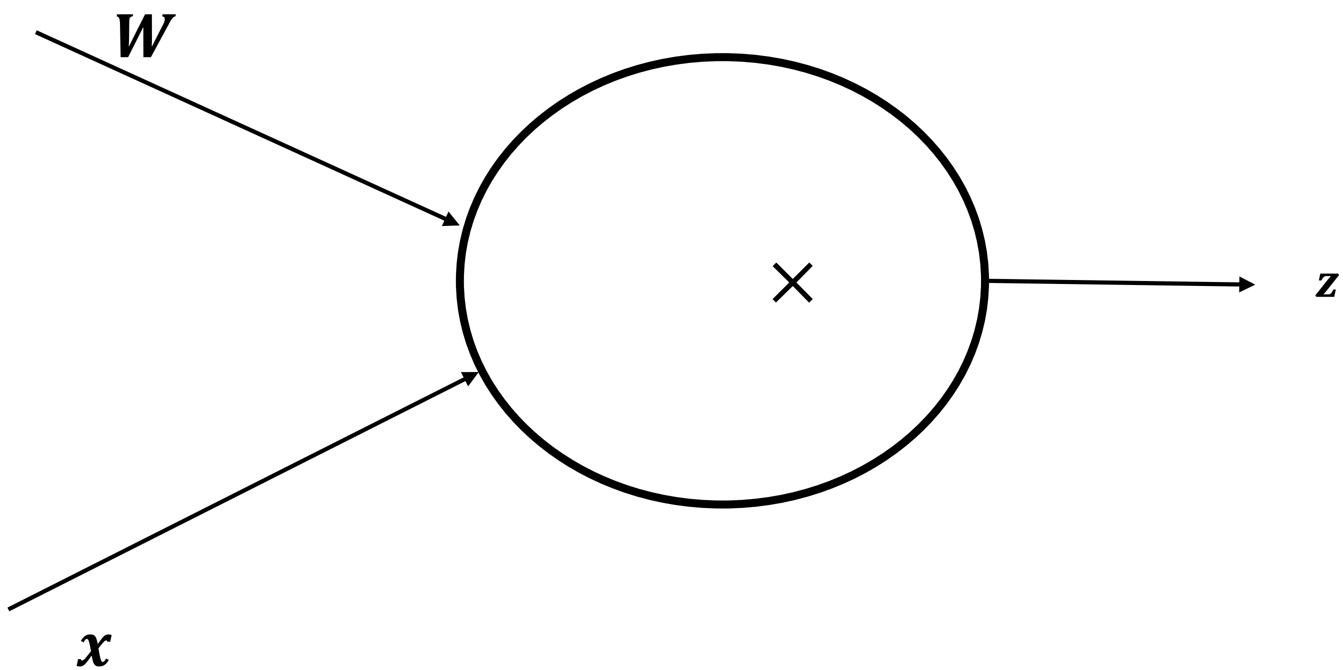
Local gradient

Upstream gradient

反向传播中的单个节点

- 具有多个输入的节点该如何处理?

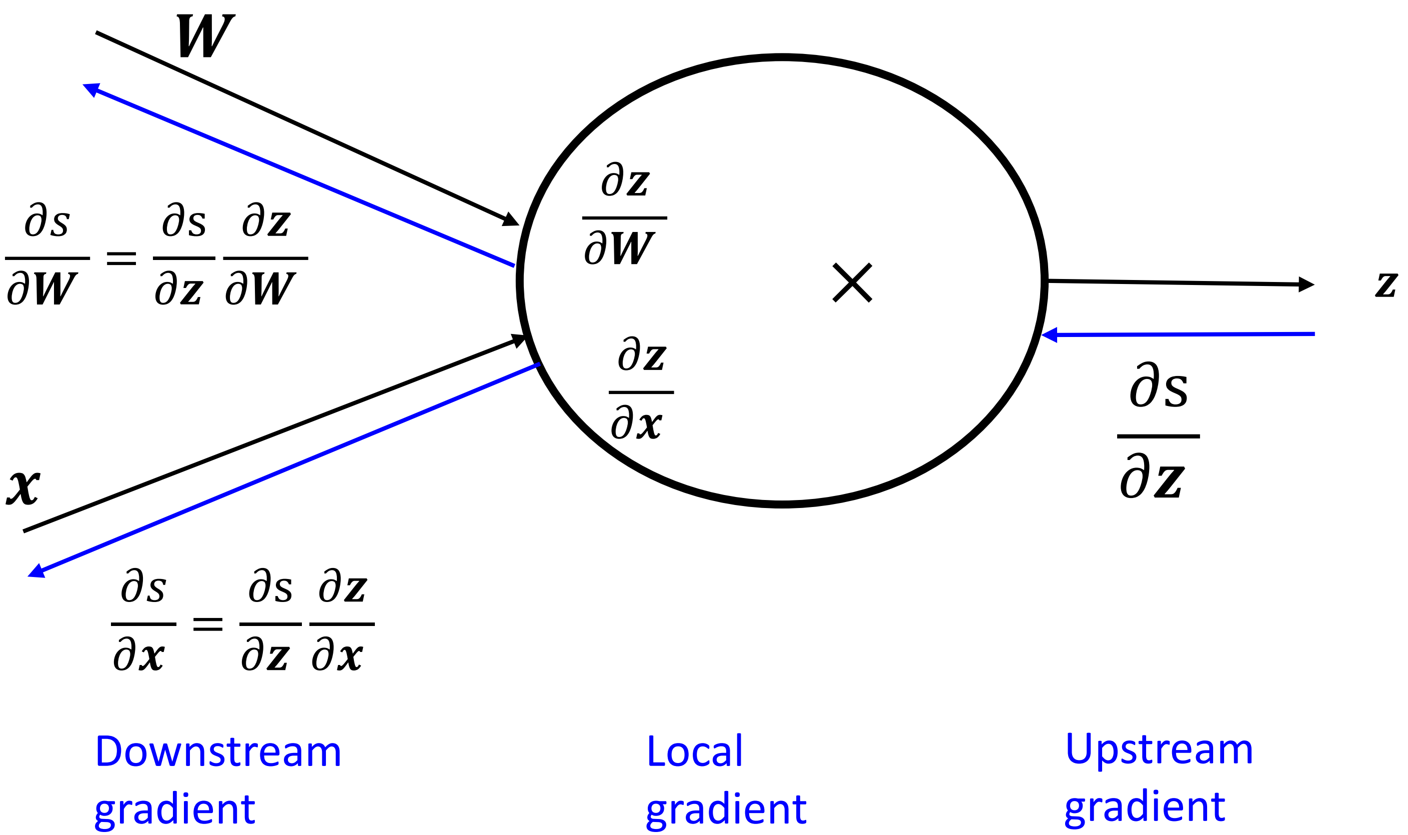
$$\boxed{\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}}$$



反向传播中的单个节点

- 具有多个输入的节点该如何处理?

$\boxed{\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}}$



一个简单示例

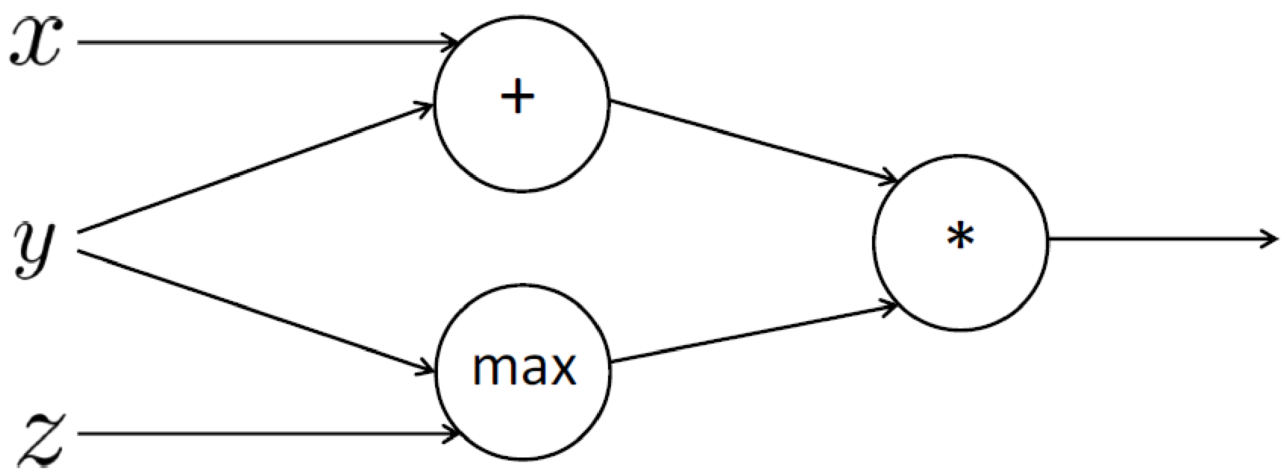
- 前向传播步骤

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$

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

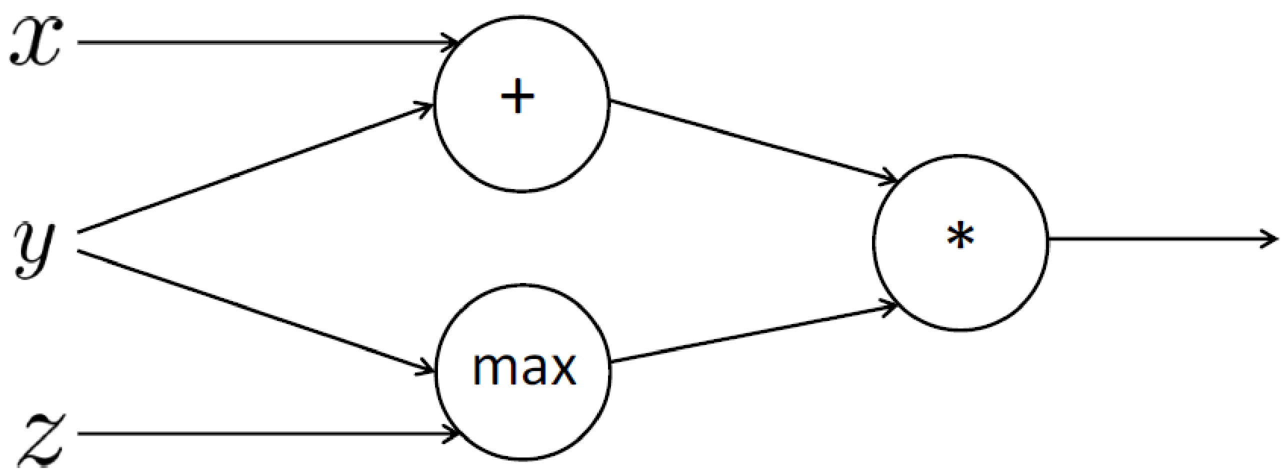


一个简单示例

- 前向传播步骤

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

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



一个简单示例

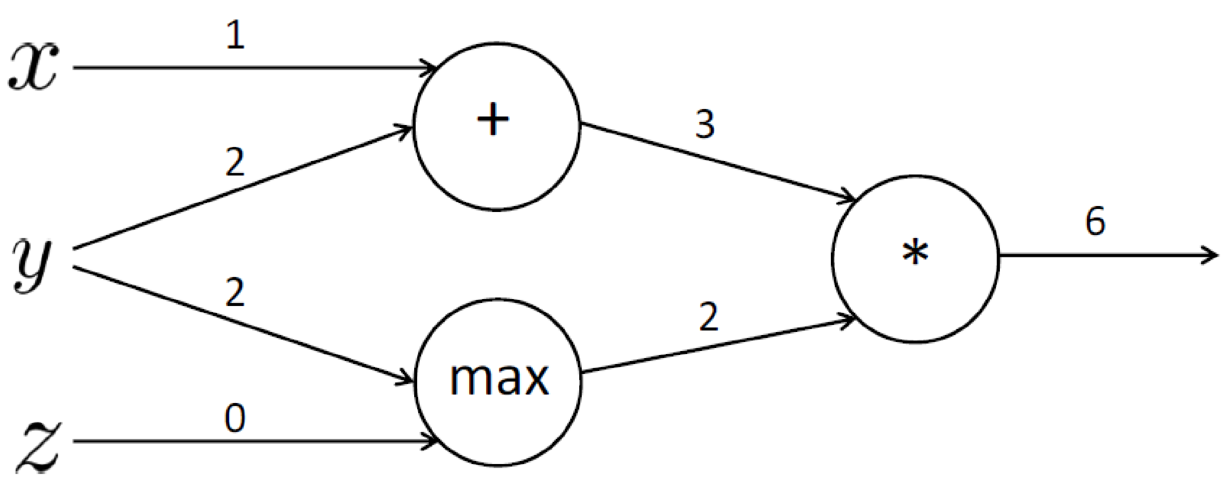
- 前向传播步骤

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$

$$\boxed{\begin{array}{l} f(x,y,z) = (x+y)\max(y,z) \\ x = 1, y = 2, z = 0 \end{array}}$$



一个简单示例

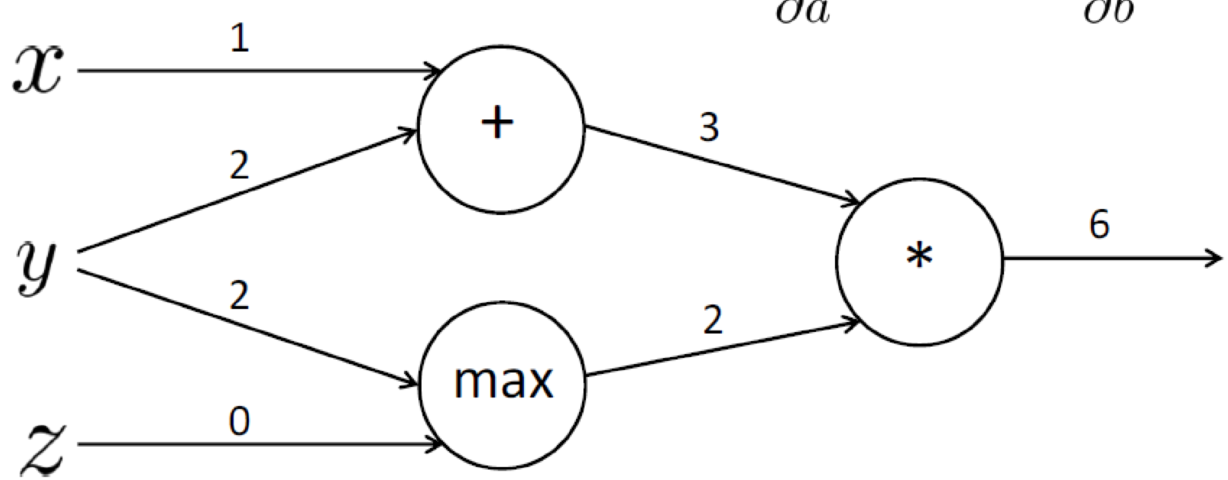
$$f(x,y,z) = (x+y)\max(y,z)$$
$$x=1, y=2, z=0$$

- 前向传播步骤

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

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



一个简单示例

$$\boxed{\begin{aligned}&f(x, y, z) = (x + y)\max(y, z)\\&x = 1, y = 2, z = 0\end{aligned}}$$

- 前向传播步骤

$$a = x + y$$

$$b = \max(y, z)$$

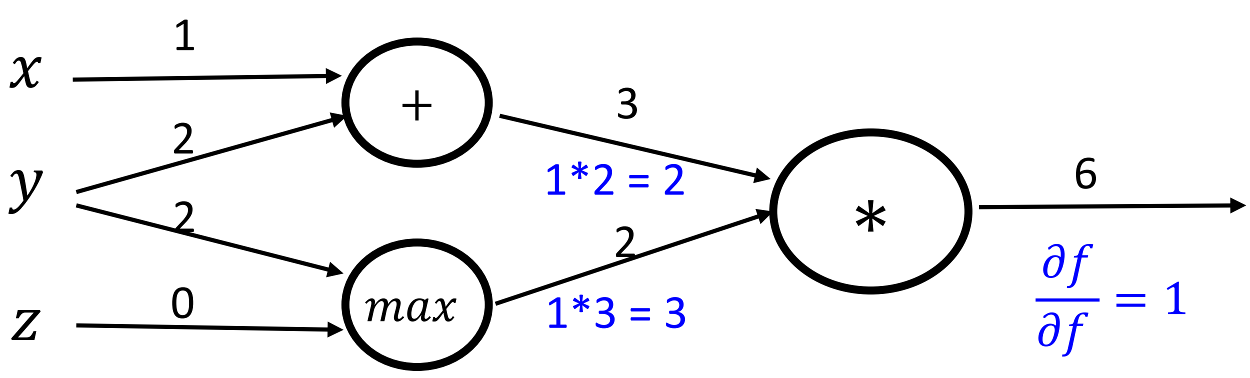
$$f = ab$$

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

一个简单示例

$$\boxed{\begin{aligned}&f(x,y,z)=(x+y)\max(y,z)\\&x=1,y=2,z=0\end{aligned}}$$

- 前向传播步骤

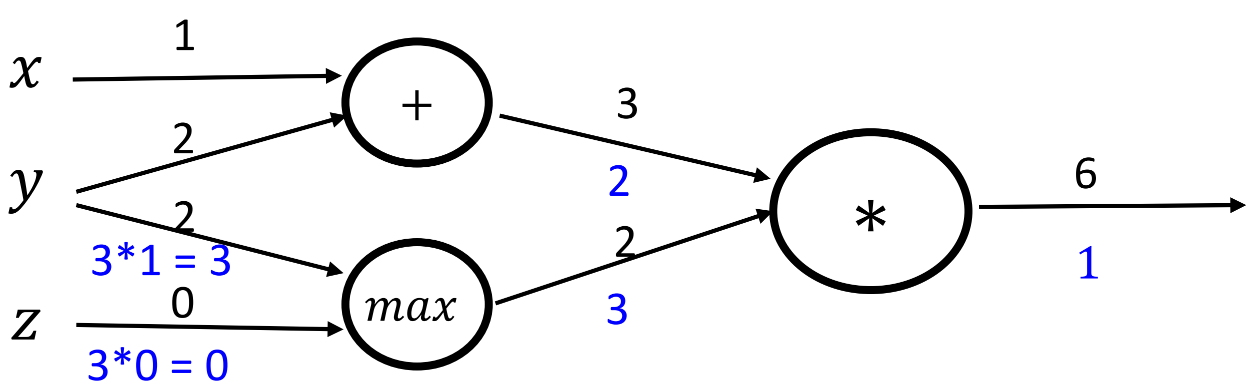
$$a=x+y$$
$$b=\max(y,z)$$
$$f=ab$$

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

一个简单示例

$$\boxed{\begin{array}{l} f(x,y,z) = (x+y)\max(y,z) \\ x=1, y=2, z=0 \end{array}}$$

- 前向传播步骤

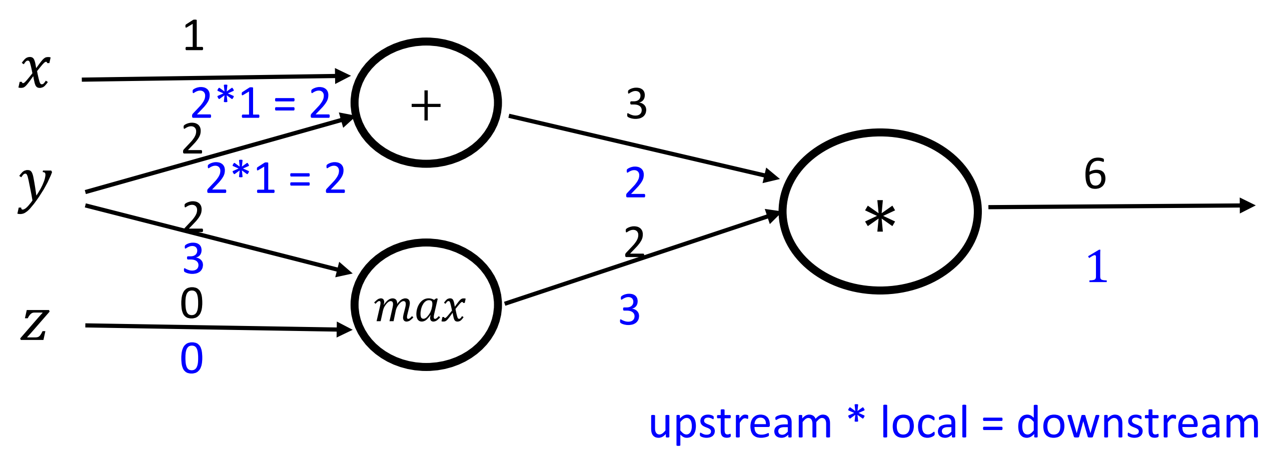
$$a = x + y$$
$$b = \max(y,z)$$
$$f = ab$$

Local gradients

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

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

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



一个简单示例

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

- 前向传播步骤

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

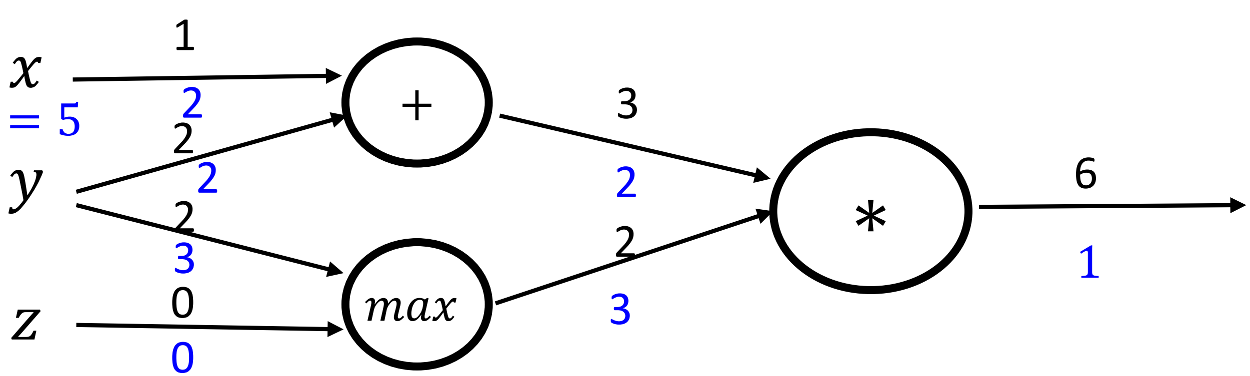
Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y>z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z>y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$

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

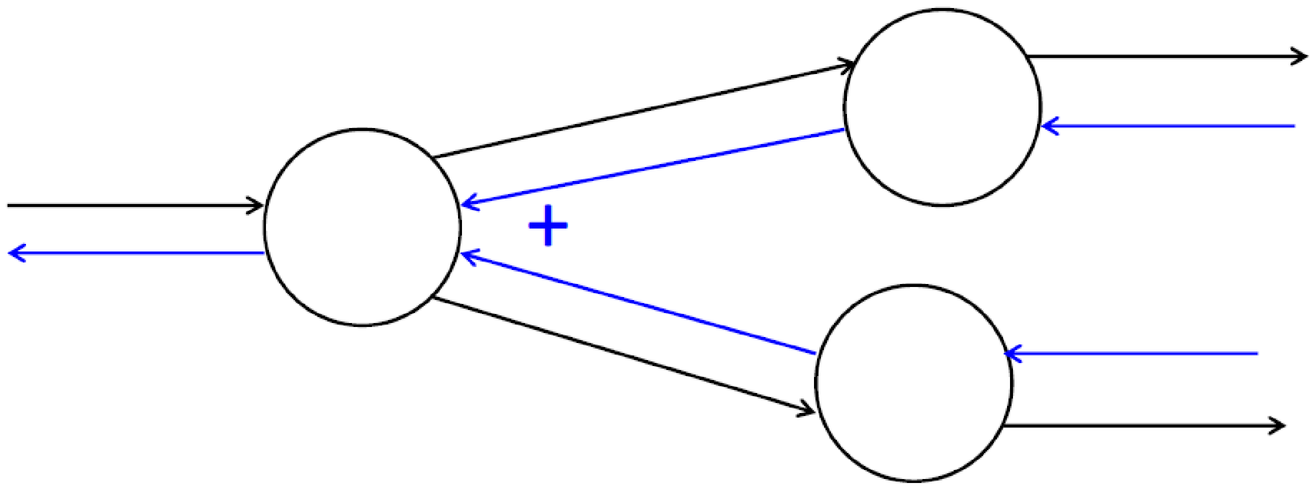
$$\frac{\partial f}{\partial y} = 3 + 2 = 5$$

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



upstream * local = downstream

Gradients sum at outward branches



$$a = x + y$$

$$b = \max(y, z)$$

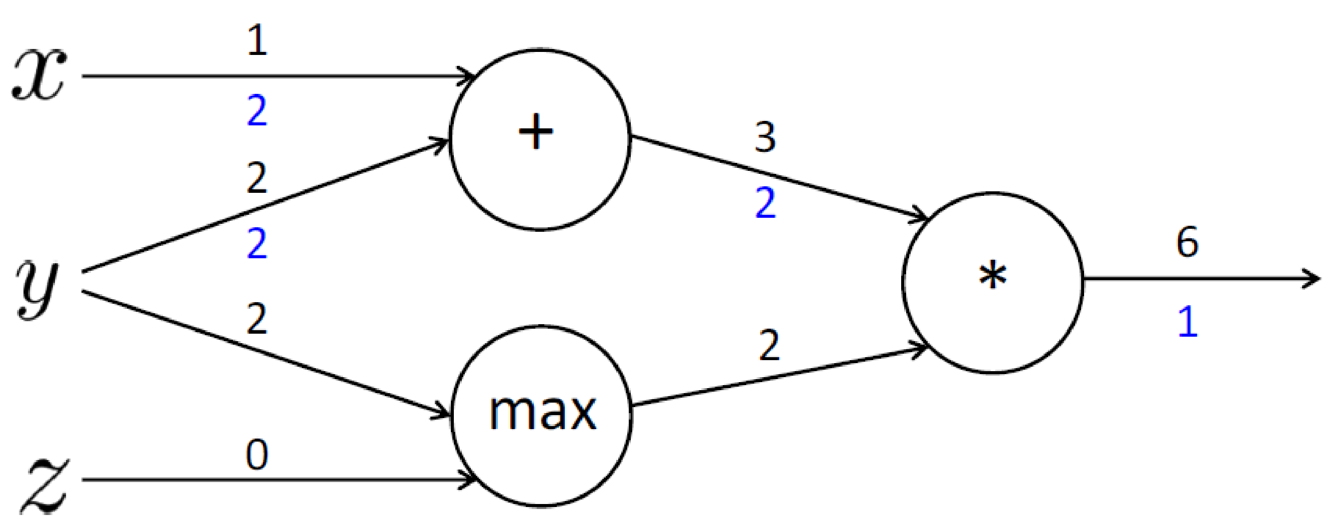
$$f = ab$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

对于节点的直观理解

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

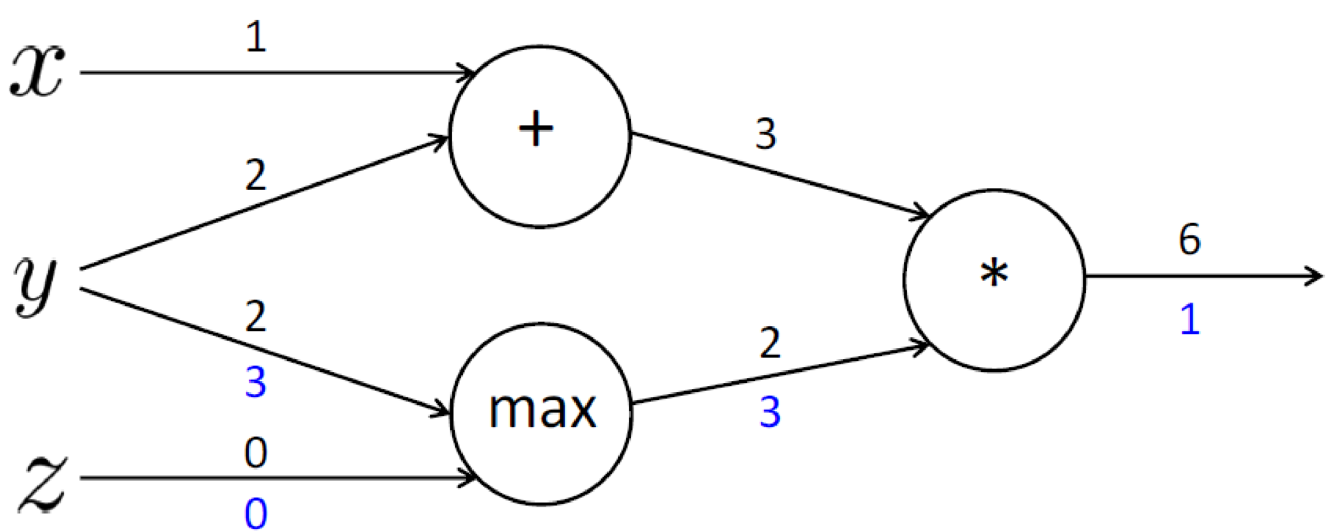
- + “distributes” the upstream gradient



对于节点的直观理解

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

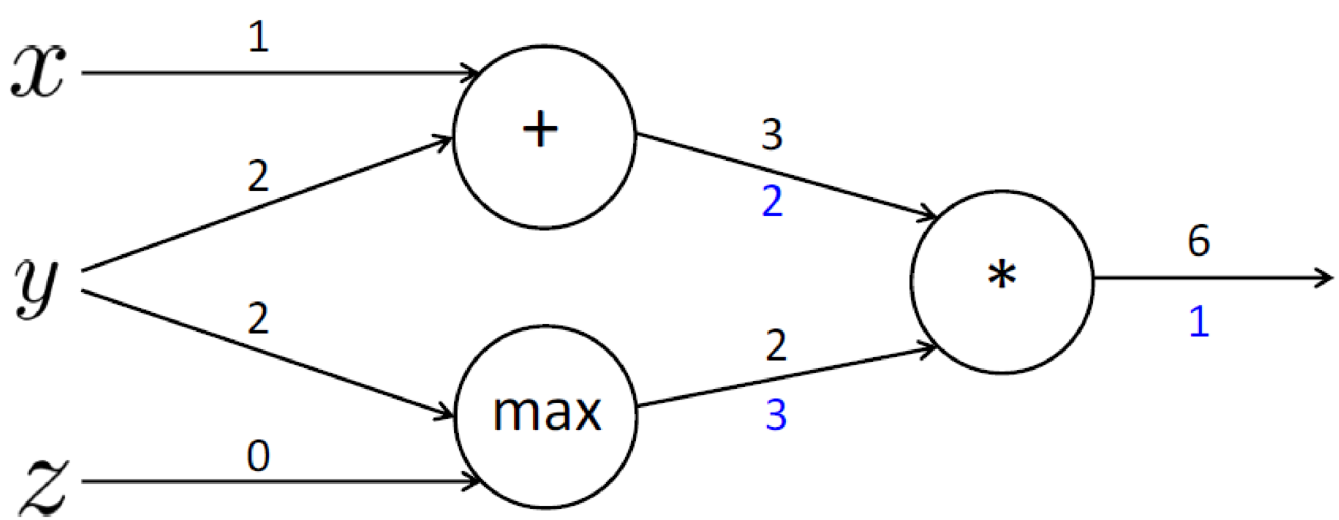
- + “distributes” the upstream gradient
- max “routes” the upstream gradient



对于节点的直观理解

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

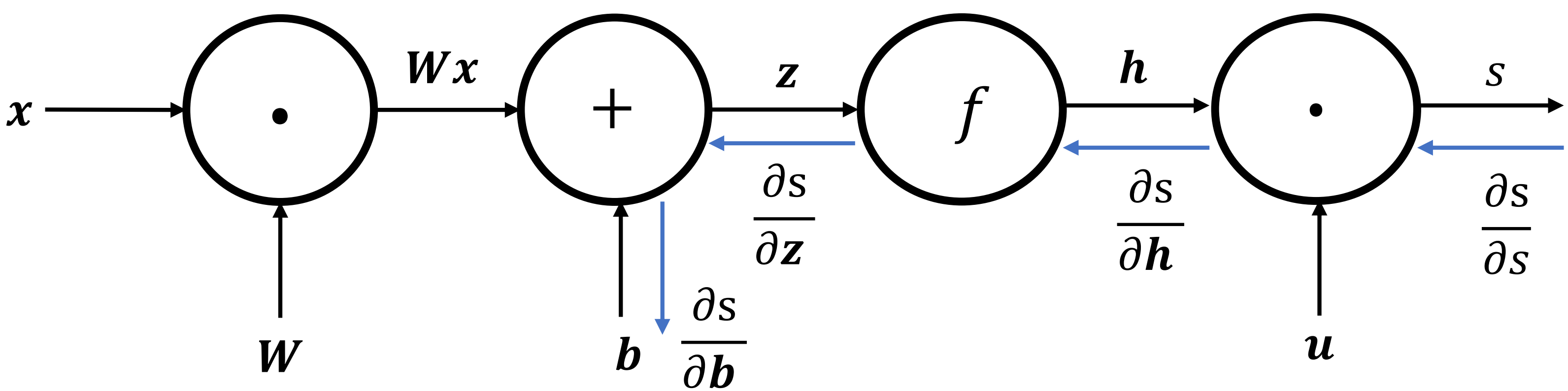
- + “distributes” the upstream gradient
- max “routes” the upstream gradient
- * “switches” the upstream gradient



反向传播算法的效率

- Incorrect way of doing backprop
  - 先计算$\frac{\partial s}{\partial \boldsymbol{b}}$

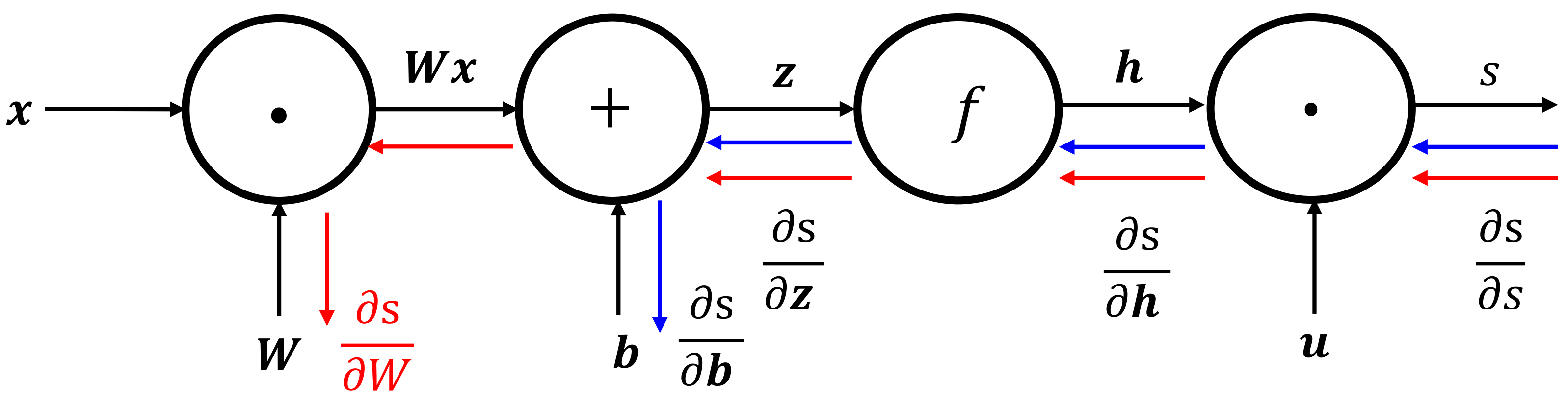
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$\boldsymbol{x}$ (输入)



反向传播算法的效率

- Incorrect way of doing backprop
  - 先计算$\frac{\partial s}{\partial \boldsymbol{b}}$
  - 再独立计算$\frac{\partial s}{\partial W}$
  - *Duplicated computation!*

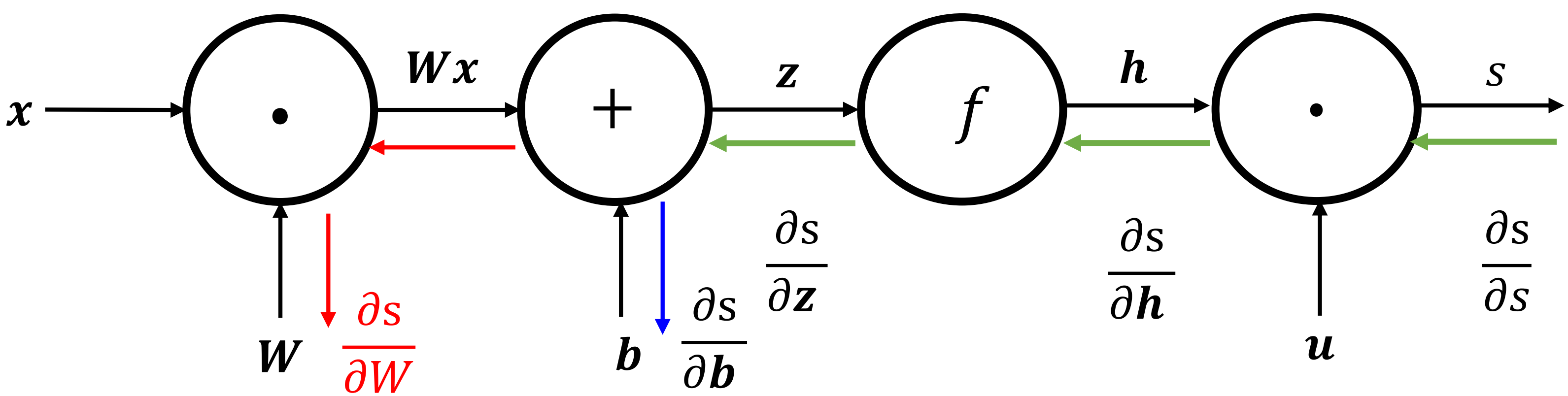
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$\boldsymbol{x}$ (输入)



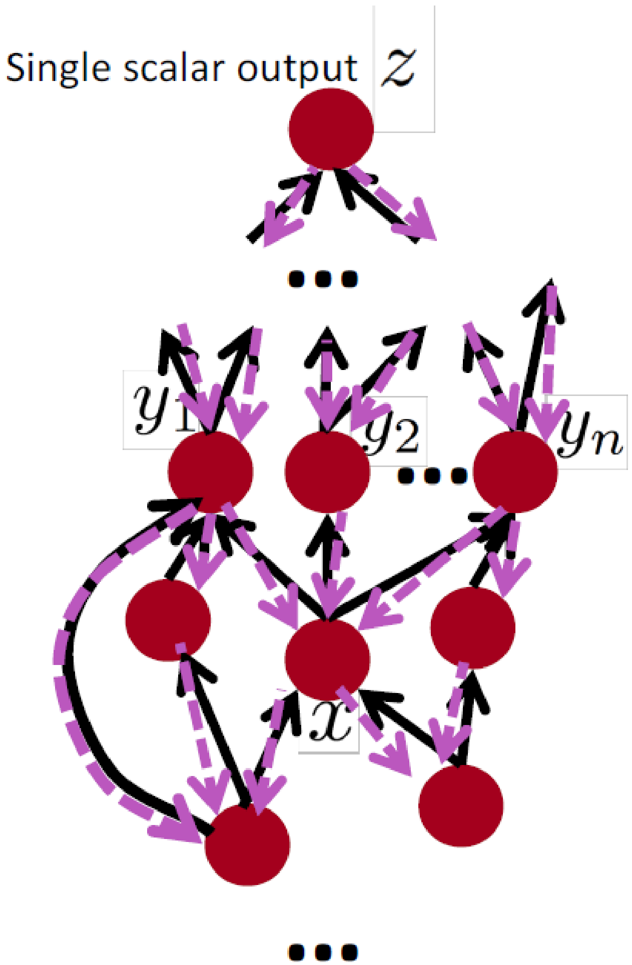
反向传播算法的效率

- Correct way
  - *Compute all the gradients at once*
  - *Analogous to using δ when we computed gradients by hand*

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$\boldsymbol{x}$ (输入)



Back-Prop in General Computation Graph



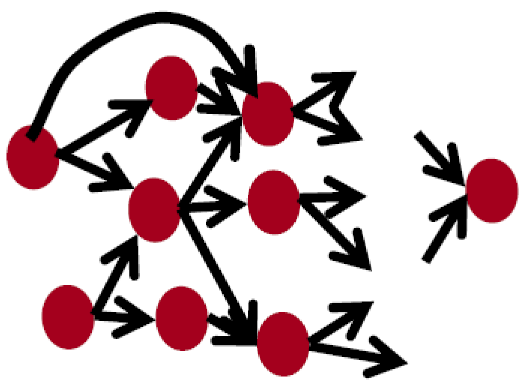
1. Fprop: visit nodes in topological sort order
    - Compute value of node given predecessors
2. Bprop:
    - initialize output gradient = 1
    - visit nodes in reverse order:
        - Compute gradient wrt each node using gradient wrt successors
        - $\{y_1, y_2, \ldots, y_n\}$ = successors of x

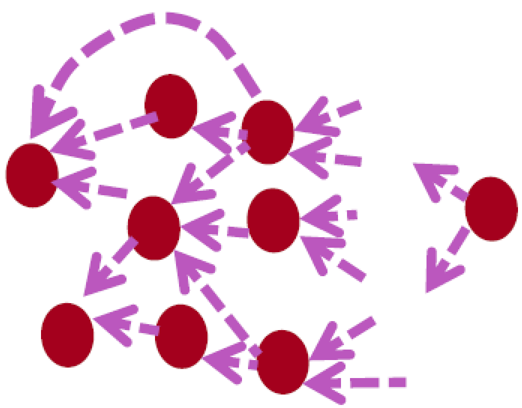
$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

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

In general our nets have regular layer-structure and so we can use matrices and Jacobians…

Automatic Differentiation



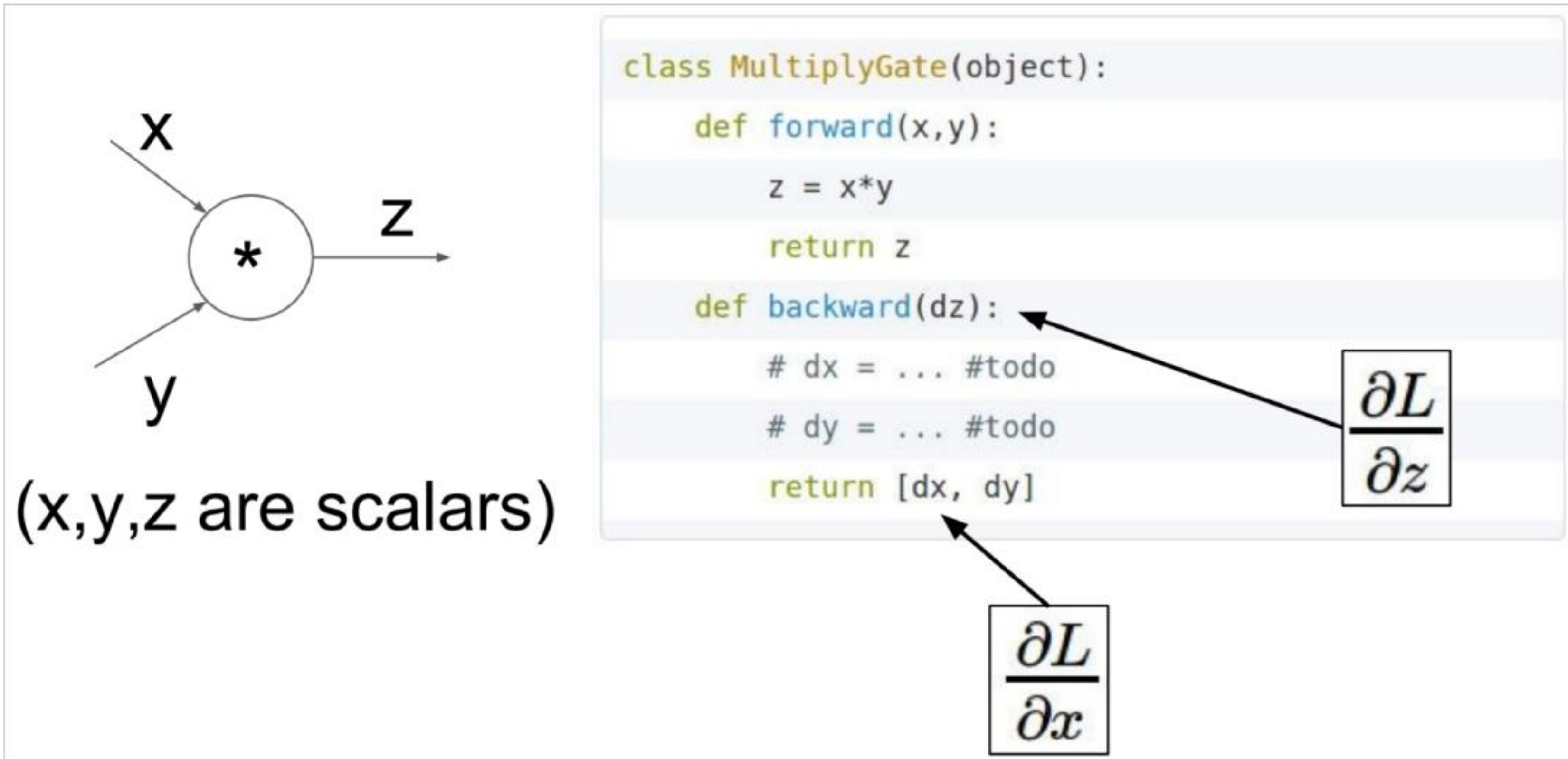


- The gradient computation can be automatically inferred from the symbolic expression of the fprop
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you but mainly leave layer/node writer to hand-calculate the local derivative

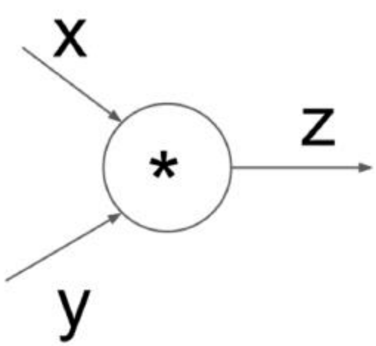
Backprop Implementations

```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Implementation: forward/backward API



Implementation: forward/backward API



(x,y,z are scalars)

```python
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]
```

总结

- 神经网络是一种非线性分类方法
- 神经网络参数的优化可以采用随机梯度下降（SGD）的方法
- SGD中的参数更新需要用到反向传播方法(Backpropagation)
- 反向传播: recursively apply the chain rule along computation graph
  - [downstream gradient] = [upstream gradient] x [local gradient]
  - Forward pass: compute results of operations and save intermediate values
  - Backward pass: apply chain rule to compute gradients