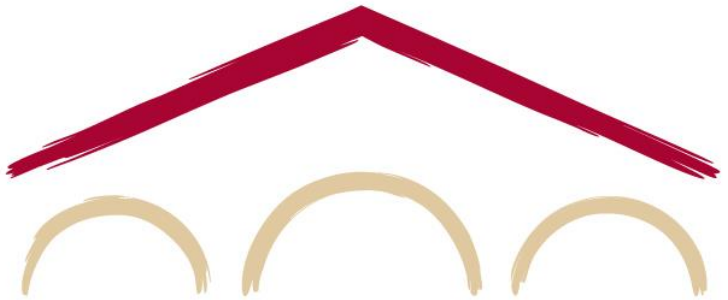
Natural Language P
与 Deep Learning
CS224N/Ling284



Diyi Yang

Lecture 3：神经网络基础

课程计划

Ｌｅｃｔｕｒｅ　２：神经网络基础

1. Course logistics (3 mins) + Word2vec evaluation (7 mins)
2. 神经网络介绍（１０　分钟）
3. 矩阵微积分（２５　分钟）
4. Ｂａｃｋｐｒｏｐａｇａｔｉｏｎ（３５　分钟）

Key Goal: the mathematics and practical implementation of how neural networks are 通过　ｂａｃｋｐｒｏｐａｇａｔｉｏｎ　训练

１．　课程后勤

Assignment 2 is all about making sure you really understand the math of neural networks … then we'll let the software do it! It also teaches us about dependency parsing

This will be *a tough week* for some! → Make sure to get help if you need it:

Visit office hours! Read tutorial materials on the syllabus!

PyTorch tutorial: 本周五下午　１：３０-２：２０，ＮＶＩＤＩＡ　报告厅

一个了解　ＰｙＴｏｒｃｈ　的好机会，它是关键的　Ｄｅｅｐ　Ｌｅａｒｎｉｎｇ　框架！

Poster session: ３　月　１６　日，１２：１５-３：１５ｐｍ，ＡＯＥＲＣ。线下参加要求需到场的学生；其他例外见　Ｅｄ　论坛帖子（第　３　周截止）

Lecture 2 回顾：如何评估 word vectors？

- 与 NLP 中的一般评估相关：内在 vs. 外在
- 内在：
  - 在特定 / 中间子任务上评估
  - 计算速度快
  - 有助于理解该系统
  - Not clear if it's helpful unless correlation to real task is established
- 外在：
  - 在真实任务上评估
  - 计算准确率可能需要很长时间
  - 不清楚是子系统本身有问题还是其交互或其他子系统有问题
  - If replacing exactly one subsystem with another improves accuracy → Winning!

内在 word vector 评估

- Word Vector 类比

a:b :: c:?

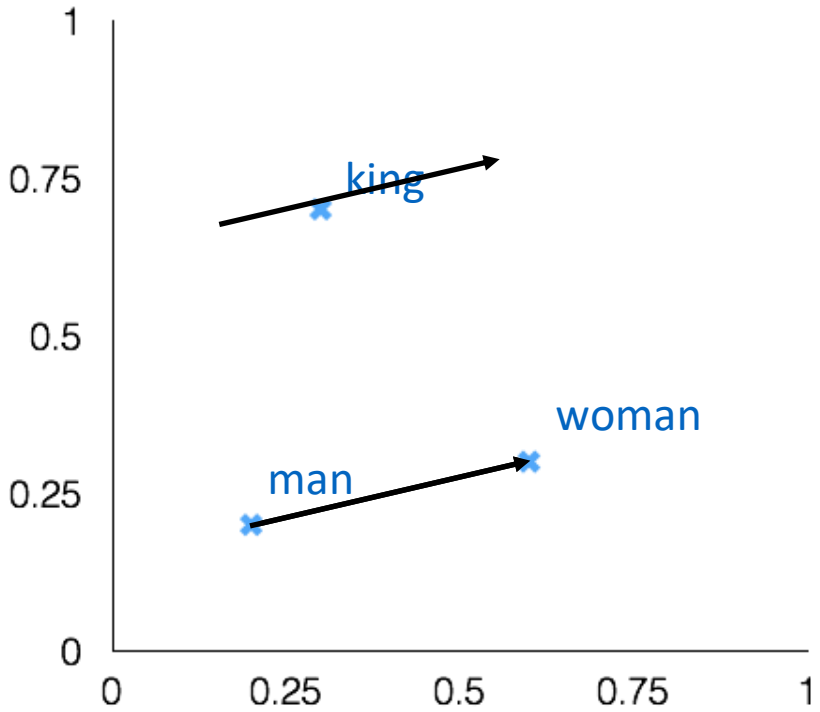
man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

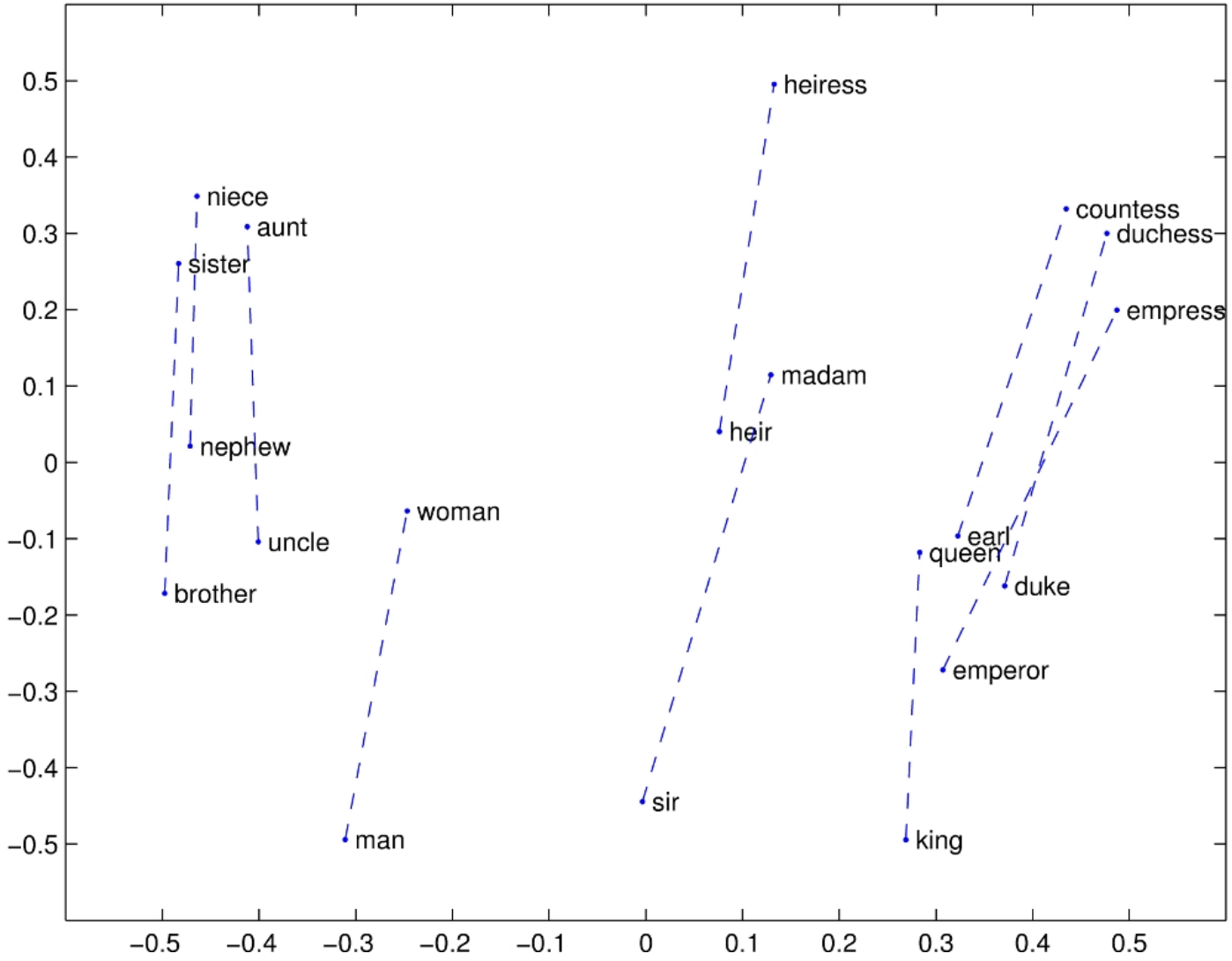
- Evaluate word vectors by how well their cosine distance after addition captures intuitive

  语义和句法类比问题
- 从搜索中丢弃输入词（！）
- Problem: What if the information is there but

  不是线性的？



GloVe 可视化


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king

语义相似度：另一种内在 word vector 评估

- Word vector 距离及其与人类判断的相关性
- Example dataset: WordSim353
  http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	人类（均值）
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

相关性评估

- Word vector 距离及其与人类判断的相关性

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

外在 word vector 评估

- One example where good word vectors should help directly: **named entity recognition**: identifying references to a person, organization or location: Chris Manning lives in Palo Alto.

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

2. Deep Learning 分类：命名实体识别（NER）

- The task: find and classify names in text, by labeling word tokens, 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

- 可能的用途：
  - 在文档中追踪特定实体的提及
  - 对于问答任务，答案通常是命名实体
  - 将情感分析与讨论中的实体关联
- 通常之后是实体链接 / 规范化到如 Wikidata 等知识库

简单 NER：使用二元 logistic 分类器的窗口分类

- 想法 :classify each word in its context window 相邻词的
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors 在一个窗口中
  - Really, we usually use multi-class softmax, but we're trying to keep it simple ☺
- **Example:** Classify "Paris" as +/– location in context of sentence with window length 2:

the museums in Paris are amazing to see .

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

- 结果向量 $x_{window} = \boxed{x \in R^{5d}}$
- To classify all words: run classifier for each class on the vector centered on each word 在句子中

分类回顾和符号

- Supervised learning: we have a training dataset consisting of samples

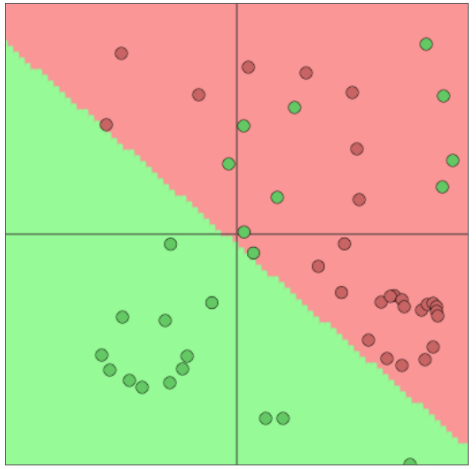
$$\{x_i, y_i\}_{i=1}^{N}$$

- x_i are inputs, e.g., words (indices or vectors!), sentences, documents, etc.
  - Dimension d

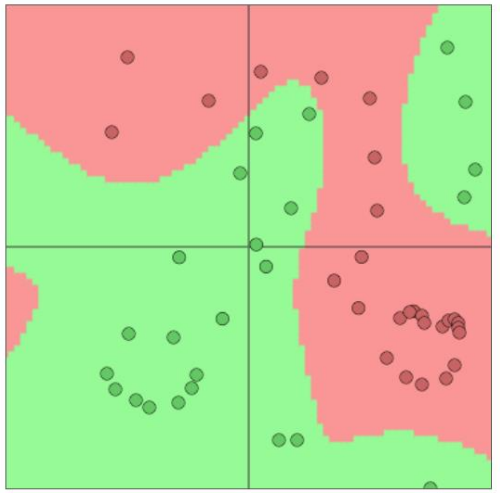
- y_i are labels (one of C classes) 我们尝试预测，例如：
  - 类别：情感（+ / -）、命名实体、买 / 卖决策
  - 其他词
  - 之后：多词序列

神经分类

- **Typical** ML/stats softmax classifier: $p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$
- 学习到的参数　θ　只是元素
  of *W* (not input representation *x*, 具有稀疏符号特征)
- 分类器给出线性决策边界，这可能有局限性



- A **neural network classifier** 不同之处在于：
  - We learn **both** *W* **and (distributed!)** representations 对于词
  - The word vectors *x* re-represent one-hot vectors, moving them around in an intermediate layer vector space, for easy classification
    用（线性）ｓｏｆｔｍａｘ　分类器
    - Conceptually, we have an embedding layer: *x* = *Le*
  - We use deep networks—more layers—that let us re-represent and
    多次组合我们的数据，得到一个非线性分类器



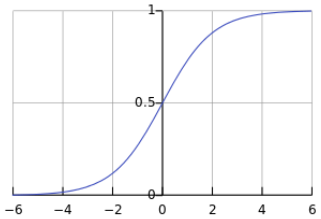
But typically, it is linear relative to the pre-final
层表示

NER：中心词是否为地点的二元分类

- We do supervised training and want high score if it's a location

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

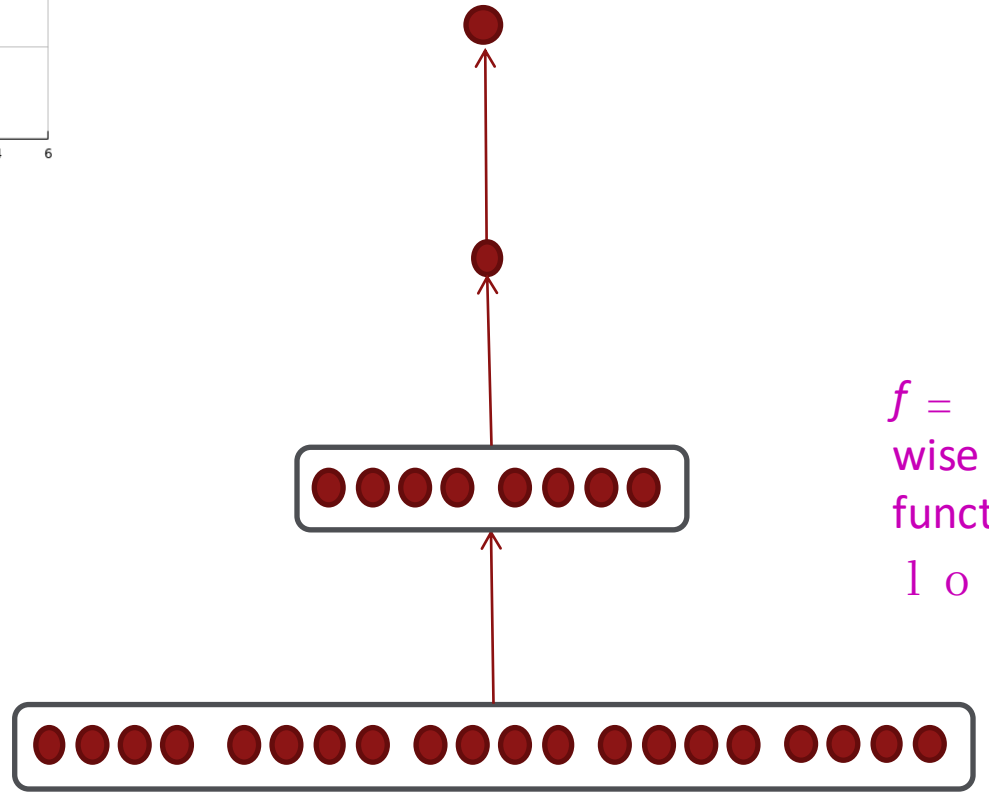
predicted model
类别的概率



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

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

$$\boldsymbol{x} \quad (\text{input}) \in \mathbb{R}^{5d}$$



f = 某些逐元素 wise non-linear function, e.g., logistic、ta

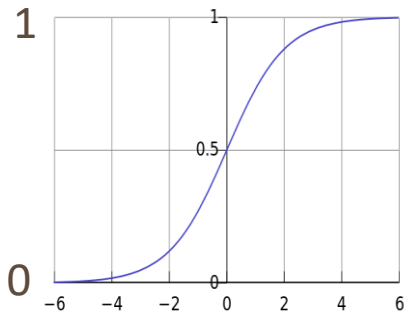
$x = [\, x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing} \,]$

……的 Emb
1-hot 词

非线性函数，新旧对比

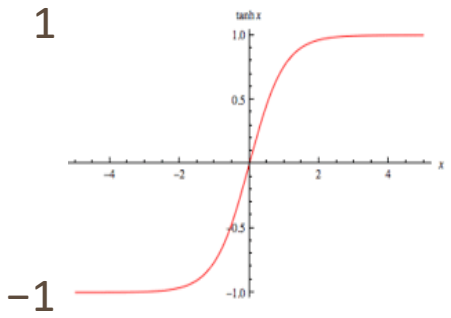
logistic (“sigmoid”)

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



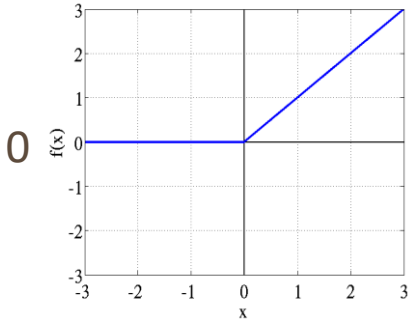
tanh

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

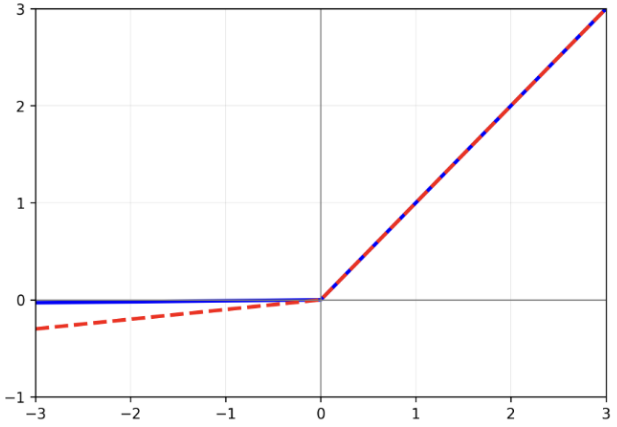


ReLU （修正线性单元）

$$\mathrm{ReLU}(z) = \max(z, 0)$$



Leaky ReLU / Parametric ReLU



t a n h 只是一个缩放和平移的 s i g m o i d（sigmoid）：

$$\tanh(z) = 2\,\mathrm{logistic}(2z) - 1$$

X as steep, [−1,1]):

Logistic and tanh are still used (e.g., logistic to get a probability); however, often, for deep networks, the first

可以尝试 R e L U：它训练快且性能好，因为梯度回流良好。

ReLU has a negative “dead zone” that recent proposals mitigate

非线性函数，新旧对比

GELU (Gaussian error linear unit);
frequently used with Transformers

$$\text{GELU}(x) = x \cdot P(X \le x), X \sim N(0,1)$$
$$\approx x \cdot \text{logistic}(1.702x)$$

SiLU
Sigmoid linear unit

$$\text{SiLU}(x) = x \cdot \sigma(x)$$

Swish

$$\text{Swish}(x) = x \cdot \sigma(\beta x)$$

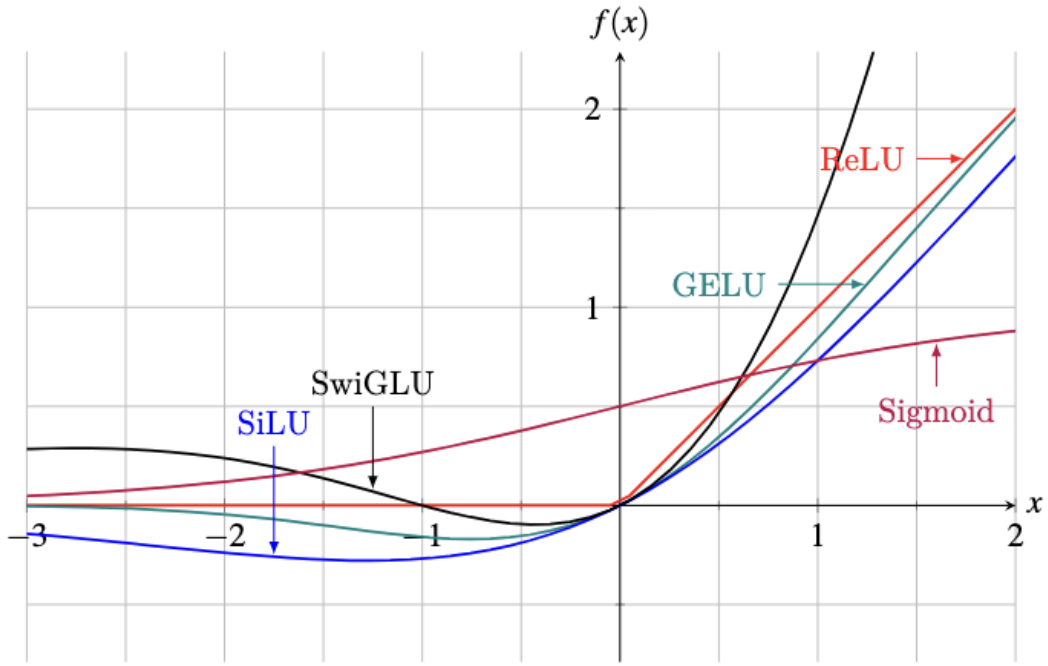
GLU (gated linear unit) uses a gate/switch

$$\text{GLU}(x) = (xV + v) \otimes \sigma(xW + b)$$

SwiGLU (Swish-gated linear unit)

$$\text{SwiGLU}(x) = (xV + c) \otimes Swish_\beta(xW + b)$$

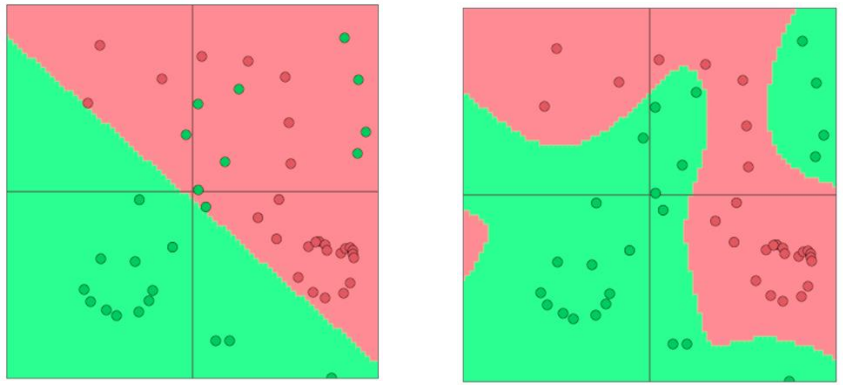
常用，例如在 ＬＬａｍａ３、Ｑｗｅｎ３ 中

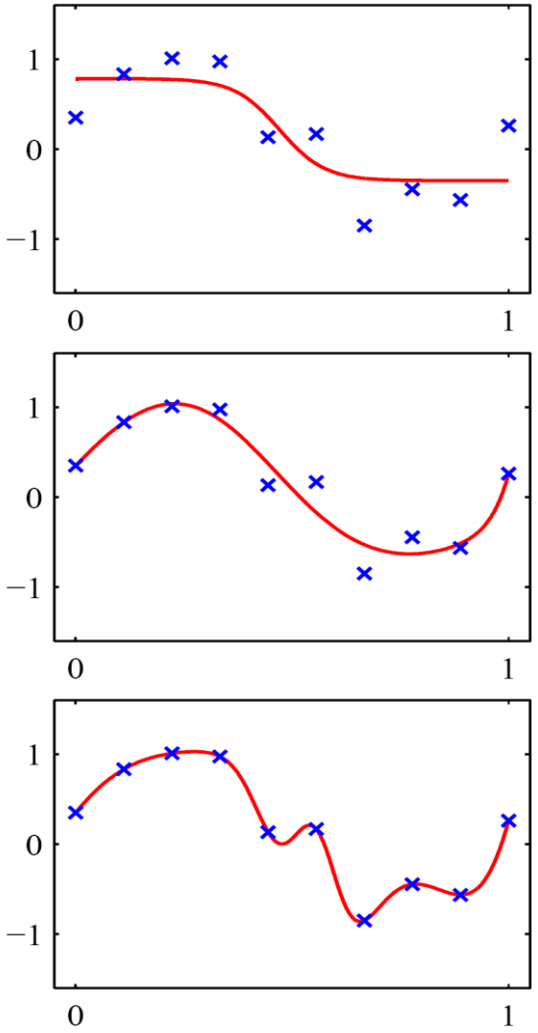


Check out *GLU Variants Improve Transformer*
https://arxiv.org/pdf/2002.05202

Non-linearities (i.e., "*f*" on previous slide): Why they're needed

- Neural networks do function approximation,
  例如回归或分类
  - 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: $W_1 W_2 x = Wx$
  - But, with more layers that include non-linearities,
    它们可以逼近任何复杂函数！





Training with “cross entropy loss” – you use this in PyTorch!

- Until now, our objective was stated as to 最大化正确类别的概率 *y*
  or equivalently we can 最小化该类别的负对数概率
- Now restated in terms of 交叉熵 , a concept from 信息论
- Let the true probability distribution be *p; l*et our computed model probability be *q*
- The cross entropy is:

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

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else, *p* = [0, …, 0, 1, 0, …, 0], then:
- **Because of one-hot *p*, the only term left is the negative log probability of the true class y_i:** $-\log p(y_i|x_i)$

回顾：随机梯度下降

更新公式：

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

α = *step size* or 学习率

i.e., for each parameter: $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j^{old}}$

In deep learning, θ includes the data representation (e.g., word vectors) too!

How can we compute $\nabla_\theta J(\theta)$?

1. 手动
2. 算法层面：b a c k p r o p a g a t i o n　算法

3． 手动计算 Gradient

- 矩阵微积分： Fully vectorized gradients
  - “Multivariable calculus is just like single-variable calculus if you use matrices”
  - 比非向量化梯度快得多且更有用
  - But doing a non-vectorized gradient can be good for intuition; recall the second 讲座中的一个例子
  - 讲义和矩阵微积分笔记更详细地介绍了这些内容
  - **You might also review Math 51, which has an online textbook: http://web.stanford.edu/class/math51/textbook.html**

梯度

- 给定一个有 1 个输出和 1 个输入的函数
  $f(x) = x^3$
- 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: $1.01^3 = 1.03$

  At x = 4 it changes about 48 times as much: $4.01^3 = 64.48$

梯度

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

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

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

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

Jacobian 矩阵：梯度的推广

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

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

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

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

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

链式法则

- For composition of one-variable functions: 乘以导数

$$z = 3y$$

$$y = x^2$$

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

- For multiple variables functions: 乘以 Jacobian

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

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

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \dots$$

示例　Ｊａｃｏｂｉａｎ：逐元素激活函数

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

$$h_i = f(z_i)$$

示例　Ｊａｃｏｂｉａｎ：逐元素激活函数

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

$$h_i = f(z_i)$$

Function has *n* outputs and *n* inputs → *n* by *n* Ｊａｃｏｂｉａｎ　矩阵

示例　Ｊａｃｏｂｉａｎ：逐元素激活函数

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

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

示例　Ｊａｃｏｂｉａｎ：逐元素激活函数

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

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

示例　Ｊａｃｏｂｉａｎ：逐元素激活函数

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

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

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

其他 Jacobian

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

在家练习计算这些！

用讲义检查你的答案

其他 Jacobian

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

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \ \ (\text{Identity matrix})$$

在家练习计算这些！

用讲义检查你的答案

其他 Jacobian

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

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^T$$

Fine print: This is the correct Jacobian. Later we discuss the “shape convention”; using it the answer would be ***h***.

在家练习计算这些！

用讲义检查你的答案

其他 Jacobian

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

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

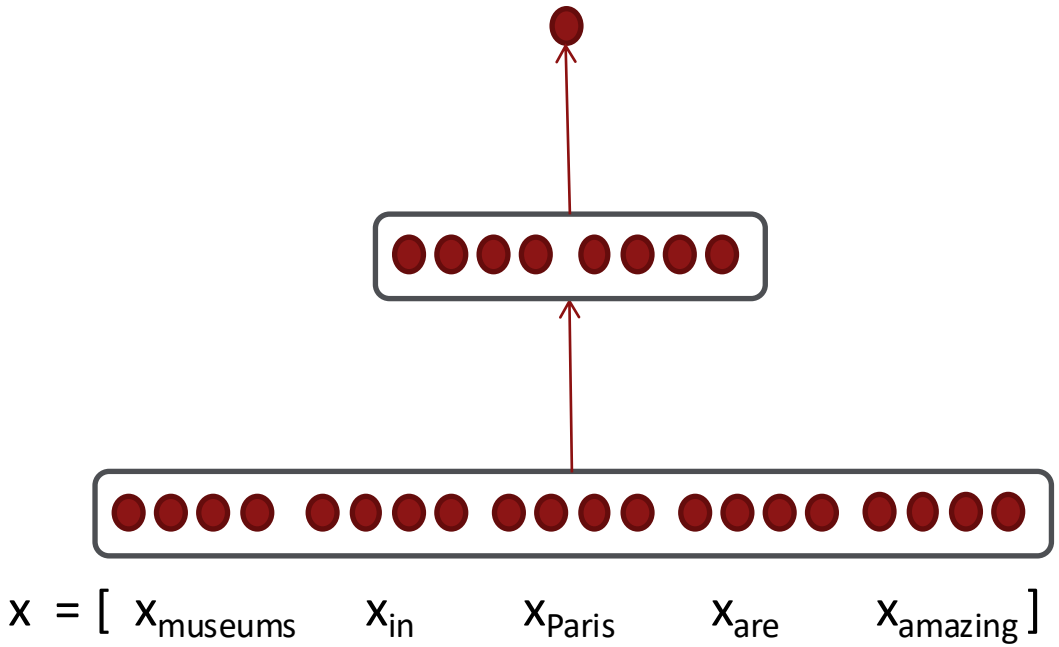
- 在家练习计算这些！
  - 用讲义检查你的答案

回到我们的神经网络！

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

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

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



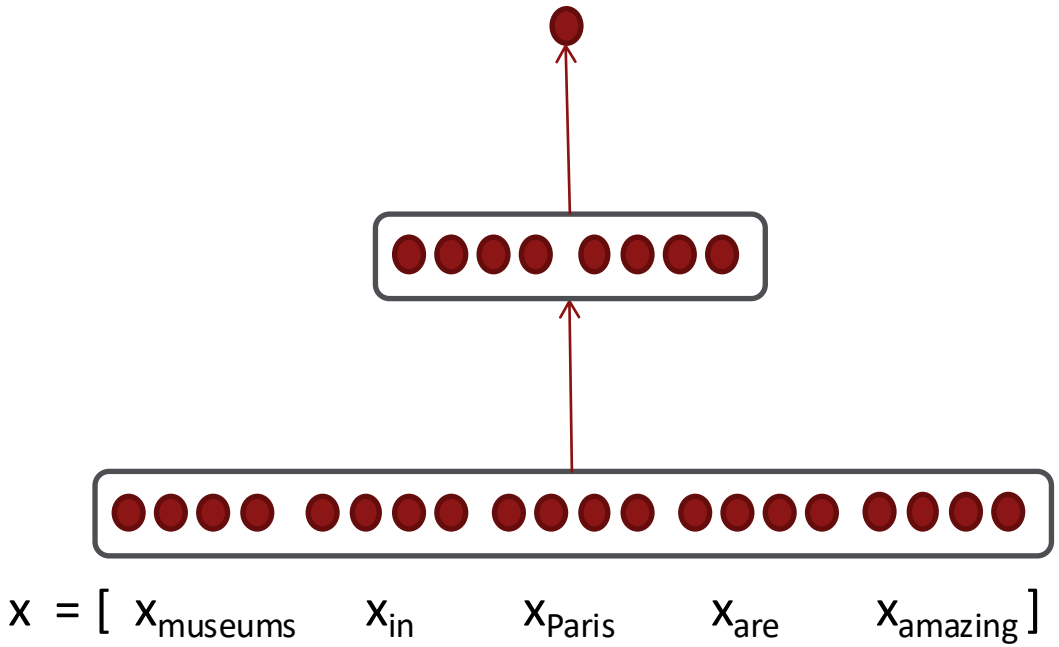
回到我们的神经网络！

- Let's find $\frac{\partial s}{\partial \boldsymbol{b}}$
  - Really, we care about the gradient of the loss *J* but we 为简单起见将计算分数的梯度

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

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

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



a . 将方程分解为简单的部分

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

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

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

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

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

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

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

Carefully define your variables and 跟踪它们的维度 !

b . 应用链式法则

$$
\begin{aligned}
s &= \boldsymbol{u}^T \boldsymbol{h} \\
\boldsymbol{h} &= f(\boldsymbol{z}) \\
\boldsymbol{z} &= \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \\
\boldsymbol{x} &\quad (\text{input})
\end{aligned}
$$

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

b . 应用链式法则

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

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

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

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

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

b . 应用链式法则

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

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

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

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

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

b . 应用链式法则

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

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

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

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

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

c． 写出 Jacobian

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

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

上一页的有用 Jacobian

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

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

c． 写出 Jacobian

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

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

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

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

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

$$\downarrow$$

$$\boldsymbol{u}^T$$

上一页的有用 Jacobian

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

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

c． 写出 Jacobian

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

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

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

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

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

$$\boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))$$

上一页的有用 Jacobian

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$

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

c． 写出 Jacobian

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

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

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

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

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

$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$

上一页的有用 Jacobian

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$

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

c． 写出 Jacobian

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

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$
$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$
$$= \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

上一页的有用 Jacobian

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

⊙ = Hadamard product = 逐元素乘法 of 2 vectors to give vector

复用计算

- 假设我们现在想计算 $\frac{\partial s}{\partial \boldsymbol{W}}$
  - 再次使用链式法则：

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

复用计算

- 假设我们现在想计算 $\frac{\partial s}{\partial \boldsymbol{W}}$
  - 再次使用链式法则：

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

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

The same! Let's avoid duplicated computation …

复用计算

- 假设我们现在想计算 $\frac{\partial s}{\partial \boldsymbol{W}}$
  - 再次使用链式法则：

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

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

$$\boldsymbol{\delta} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

δ is the upstream gradient ("error signal")

对矩阵求导：输出形状

- What does $\frac{\partial s}{\partial \boldsymbol{W}}$ look like? $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to then do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

对矩阵求导：输出形状

- What does $\frac{\partial s}{\partial \boldsymbol{W}}$ look like? $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to then do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

- Instead, we leave pure math and use the shape conven
  梯度的形状就是参数的形状！
  - So $\frac{\partial s}{\partial \boldsymbol{W}}$ is *n* by *m*: $\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$

对矩阵求导

- What is $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$
  - $\boldsymbol{\delta}$ is going to be in our answer
  - 另一项应该是 因为 $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- Answer is: $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$

δ is upstream gradient ("error signal") at z

x is local input signal

为什么要转置？

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

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

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

- 技巧性回答：这使得维度对得上！
  - 检查你工作的有用技巧！
- 讲义中有完整解释
  - 每个输入到每个输出 — 你需要得到外积

在 backprop 中推导局部输入梯度

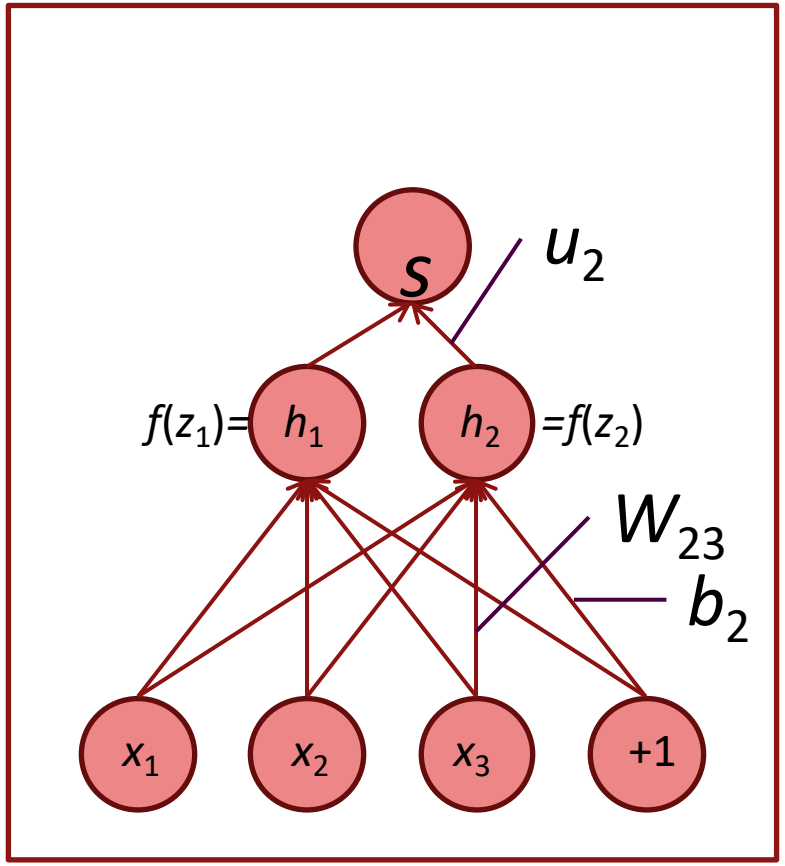
- For $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$ in our equation:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta}\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

- Let's consider the derivative of a single weight W_{ij}
- W_{ij} only contributes to z_i
  - For example: W_{23} is only used to compute z_2 not z_1

$$\frac{\partial z_i}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}}\boldsymbol{W}_{i.}\boldsymbol{x} + b_i$$

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



导数应该是什么形状？

- Similarly, $\frac{\partial s}{\partial \boldsymbol{b}} = \boldsymbol{h}^T \circ f'(\boldsymbol{z})$ is a row vector
  - But shape convention says our gradient should be a column vector because ***b*** is 一个列向量……
- Disagreement between Jacobian form (which makes the chain rule 简单）和 ｓｈａｐｅ ｃｏｎｖｅｎｔｉｏｎ（使实现 ＳＧＤ 变简单）
  - We expect answers in the assignment to follow the **shape convention**
  - 但 Ｊａｃｏｂｉａｎ 形式对计算答案有用

导数应该是什么形状？

解决具体问题的两种方式：

1. Use Jacobian form as much as possible, reshape to 最后遵循 shape convention：
   - What we just did. But at the end transpose $\frac{\partial s}{\partial \boldsymbol{b}}$ to make the 导数为列向量，结果为 $\boldsymbol{\delta}^T$
2. 始终遵循 shape convention
   - Look at dimensions to figure out when to transpose and/or 重排各项
   - The error message $\boldsymbol{\delta}$ that arrives at a hidden layer has the 与该隐藏层相同的维度

Use Jacobian to **compute**; Use shape convention to 格式 .

4. Backpropagation（反向传播）

We've almost shown you backpropagation

It's taking derivatives and using the (generalized, multivariate, or matrix) 链式法则

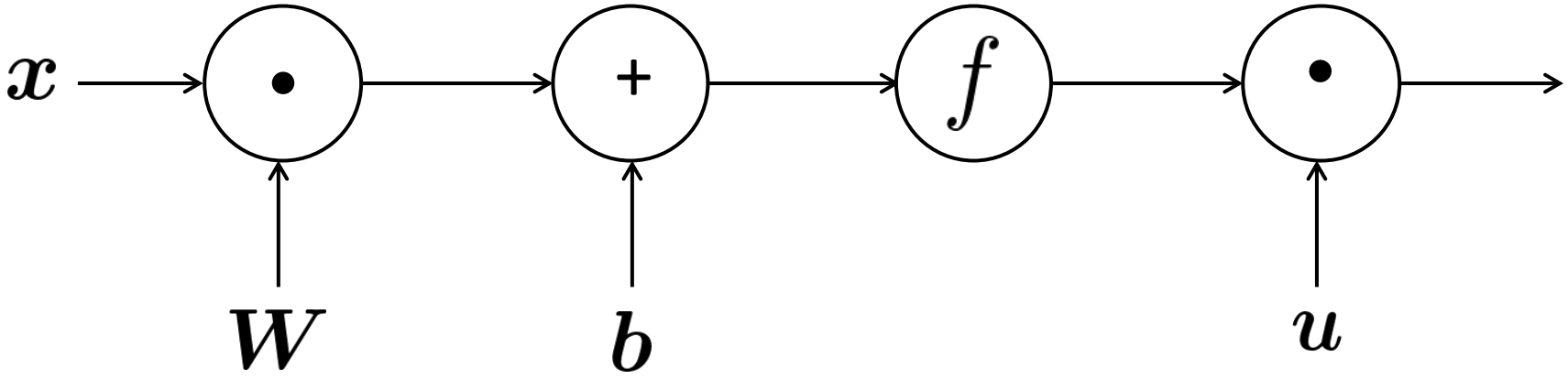
Other trick:

We 复用 derivatives computed for higher layers in computing 更低层的导数以最小化计算

计算图与 Backpropagation

- Software represents our neural net equations as a graph
  - 源节点：输入
  - 内部节点：运算

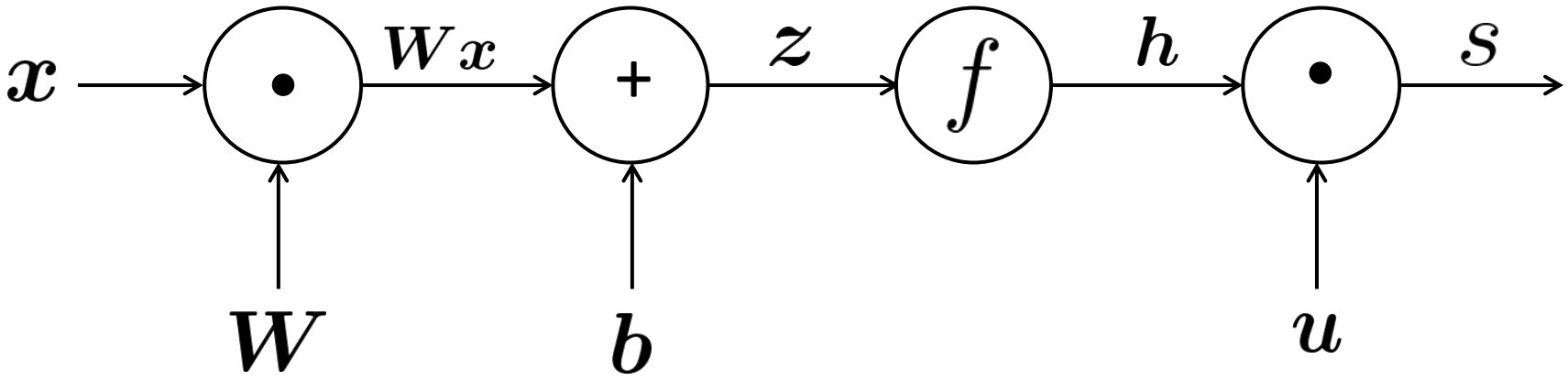
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$



计算图与 Backpropagation

- Software represents our neural net equations as a graph
  - 源节点：输入
  - 内部节点：运算
  - Edges pass along result of the operation

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



计算图与 Backpropagation

- Software represents our neural net equations as a graph

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

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

$+ \boldsymbol{b}$

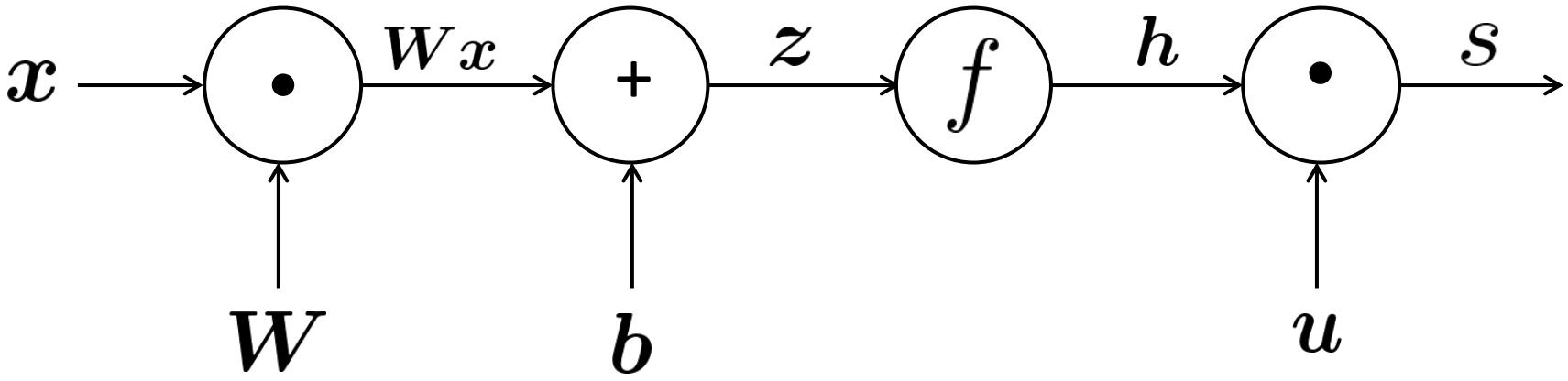
ut)

源节点：输入

内部节点：运算

pagation”

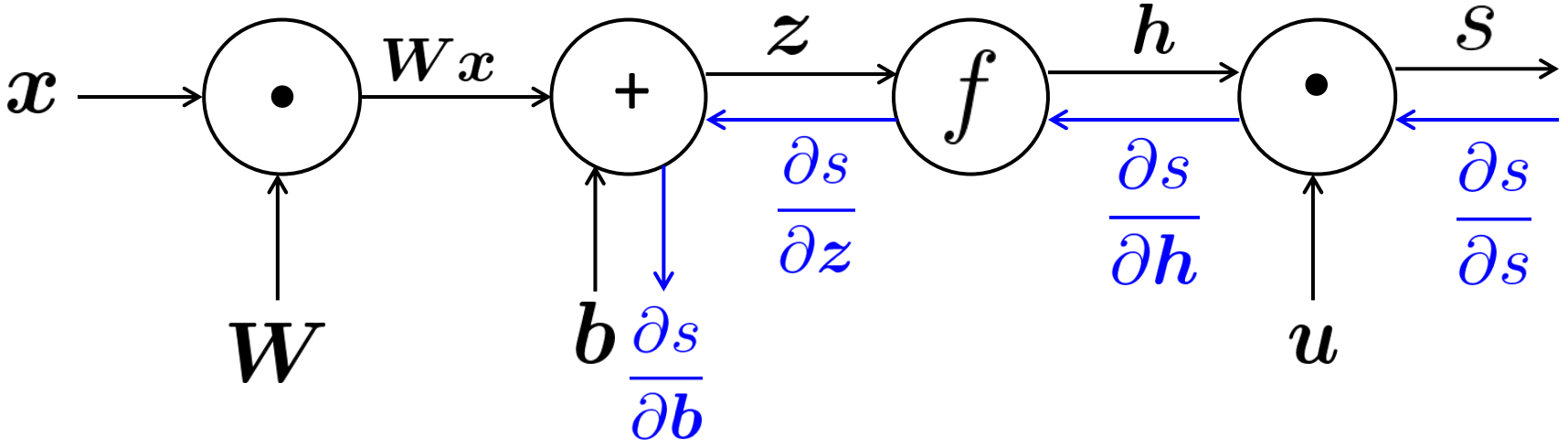
operation



Backpropagation（反向传播）

- 然后沿边反向传播
  - Pass along **gradients**

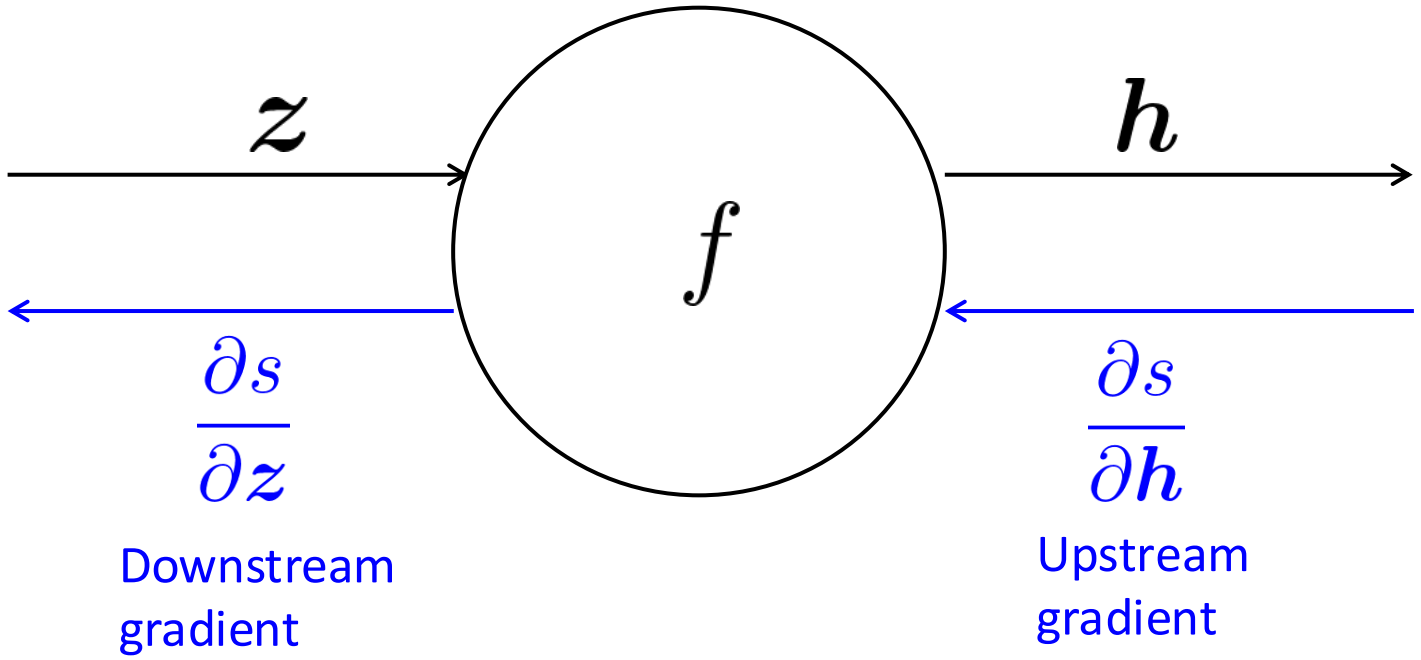
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$



Backpropagation：单节点

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

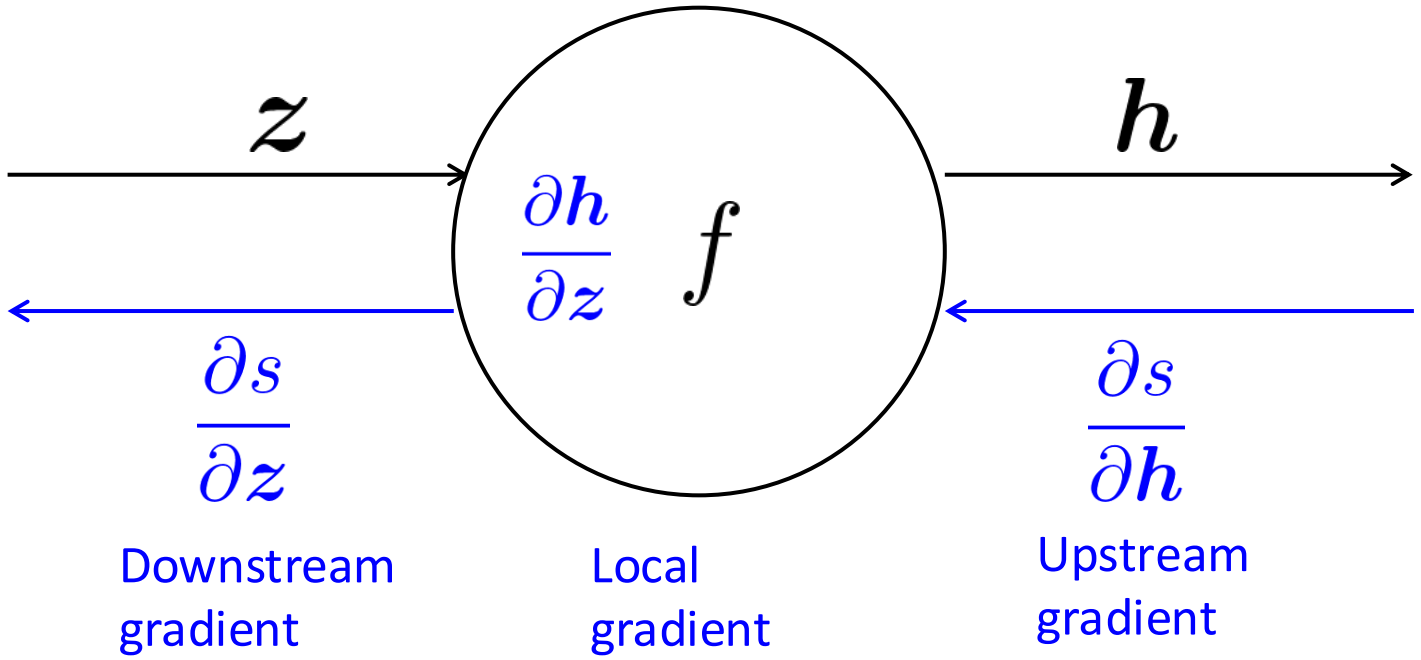
$$\boldsymbol{h} = f(\boldsymbol{z})$$



Backpropagation：单节点

- Each node has a 局部梯度
  - The gradient of its output with 对其输入

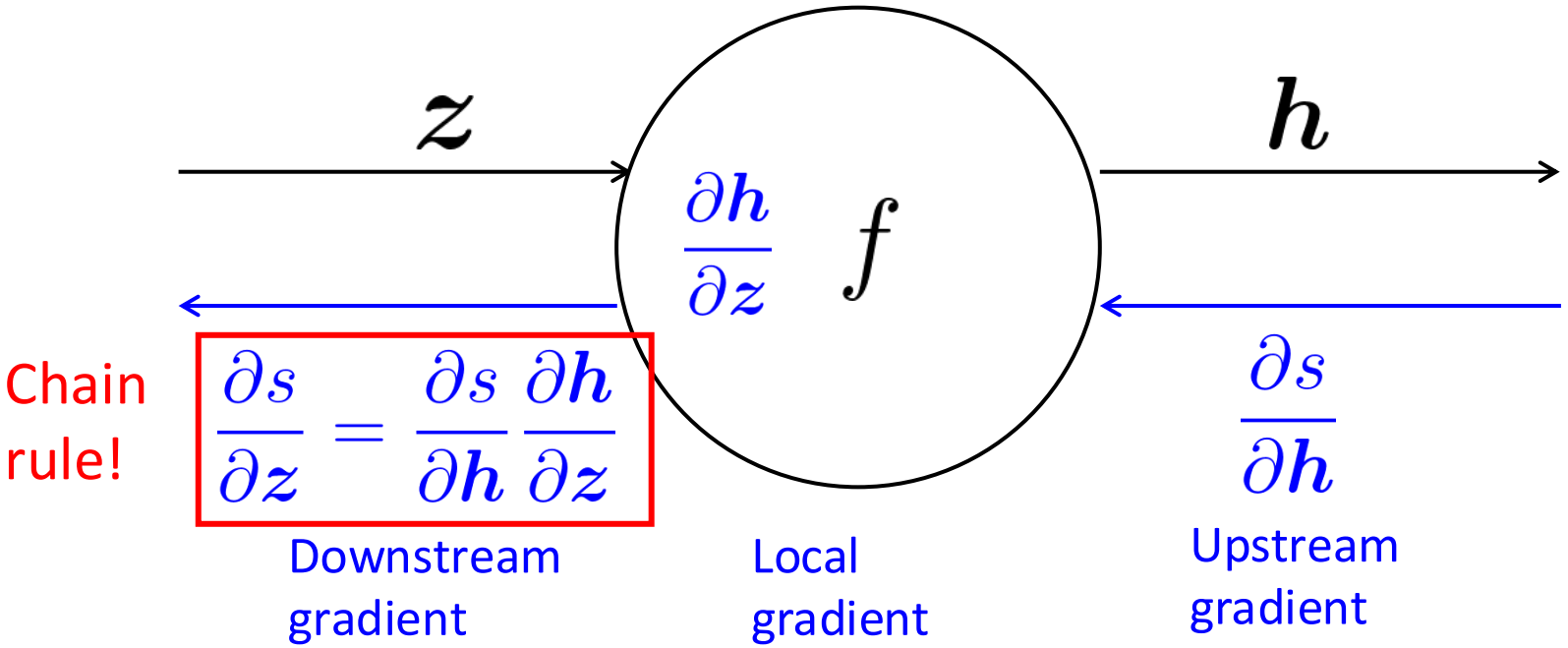
$$\boldsymbol{h} = f(\boldsymbol{z})$$



Backpropagation：单节点

- Each node has a 局部梯度
  - The gradient of its output with 对其输入

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

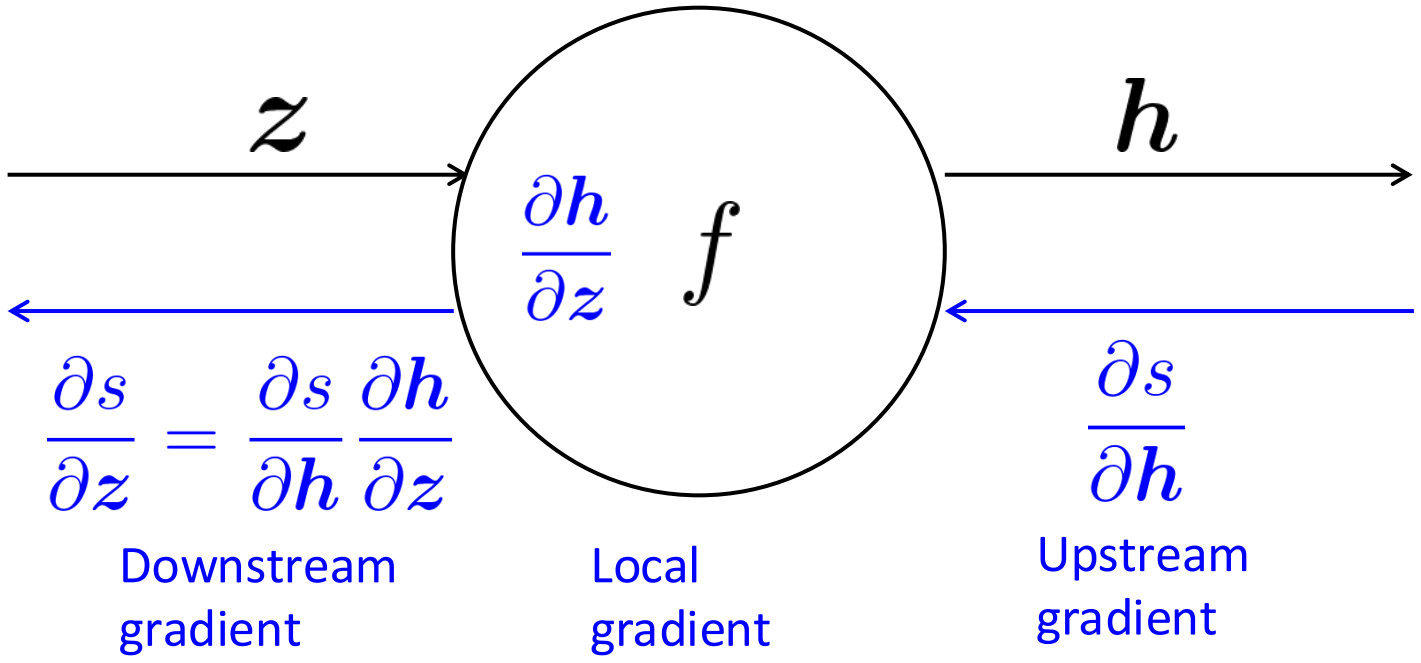


Backpropagation：单节点

- Each node has a 局部梯度
  - The gradient of its output with 对其输入

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

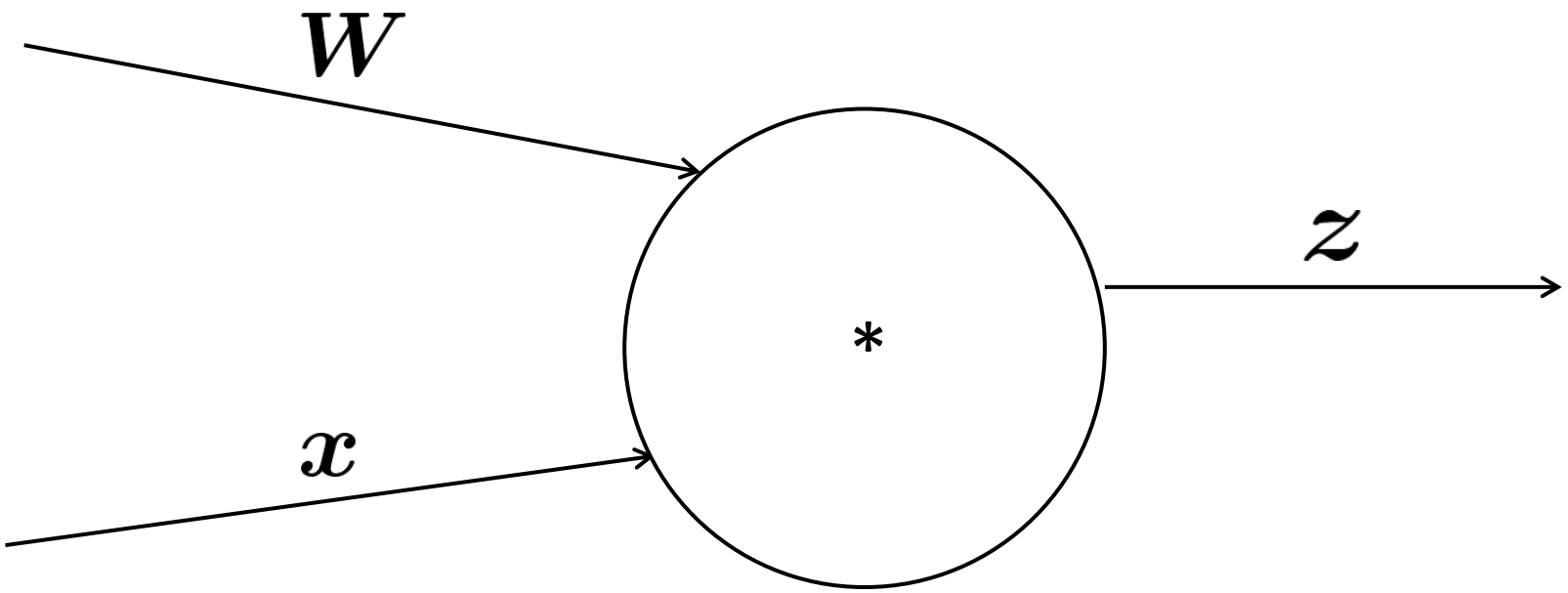
[下游梯度] = [上游梯度] x [局部梯度]



Backpropagation：单节点

- 多输入节点呢？

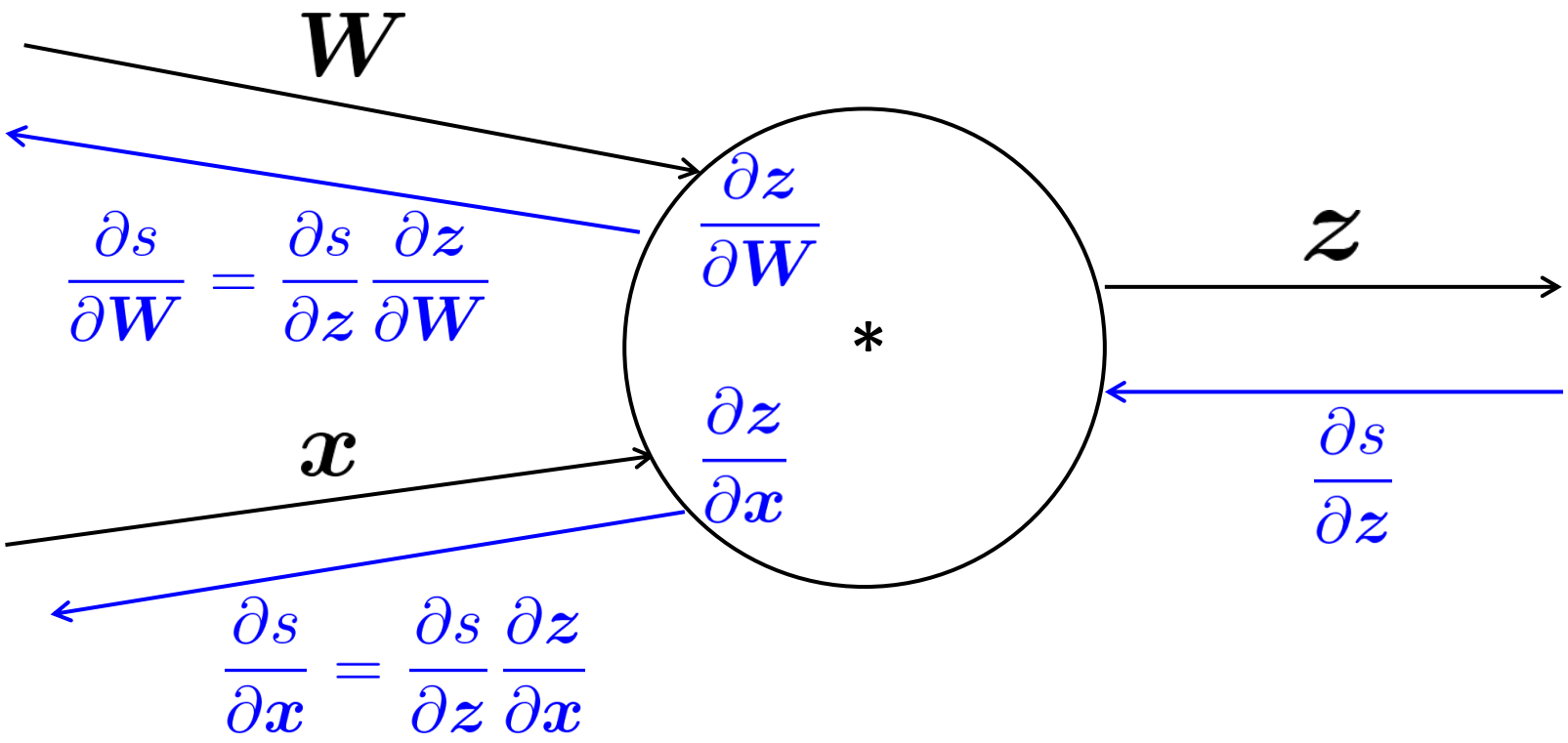
$$z = Wx$$



Backpropagation：单节点

- 多输入 → 多个局部梯度

$$z = Wx$$

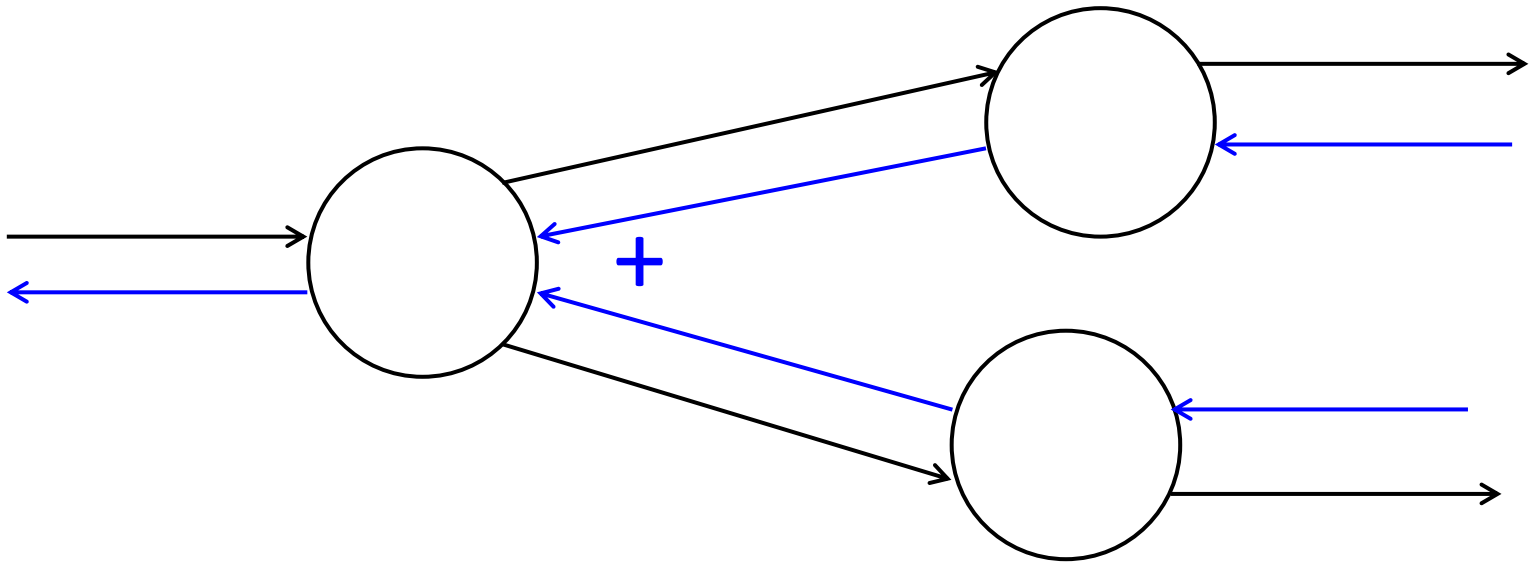


Downstream gradients

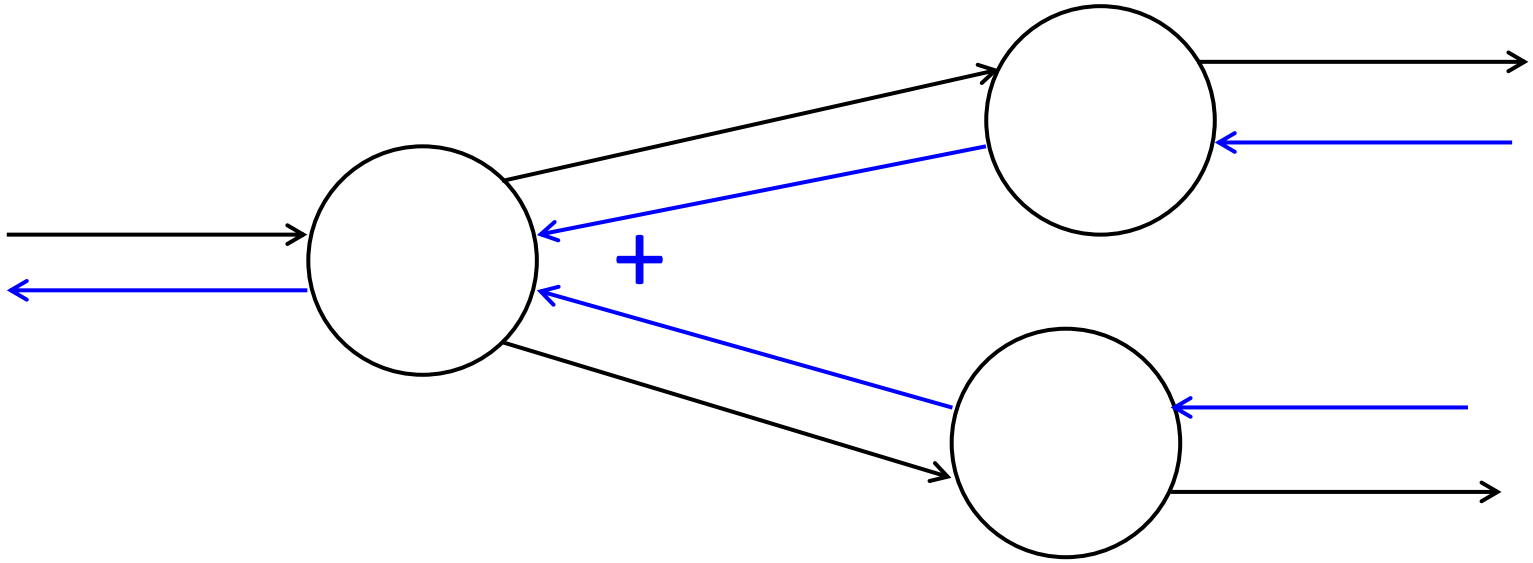
Local gradients

Upstream gradient

梯度在分叉处求和



梯度在分叉处求和



$$a = x + y$$

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

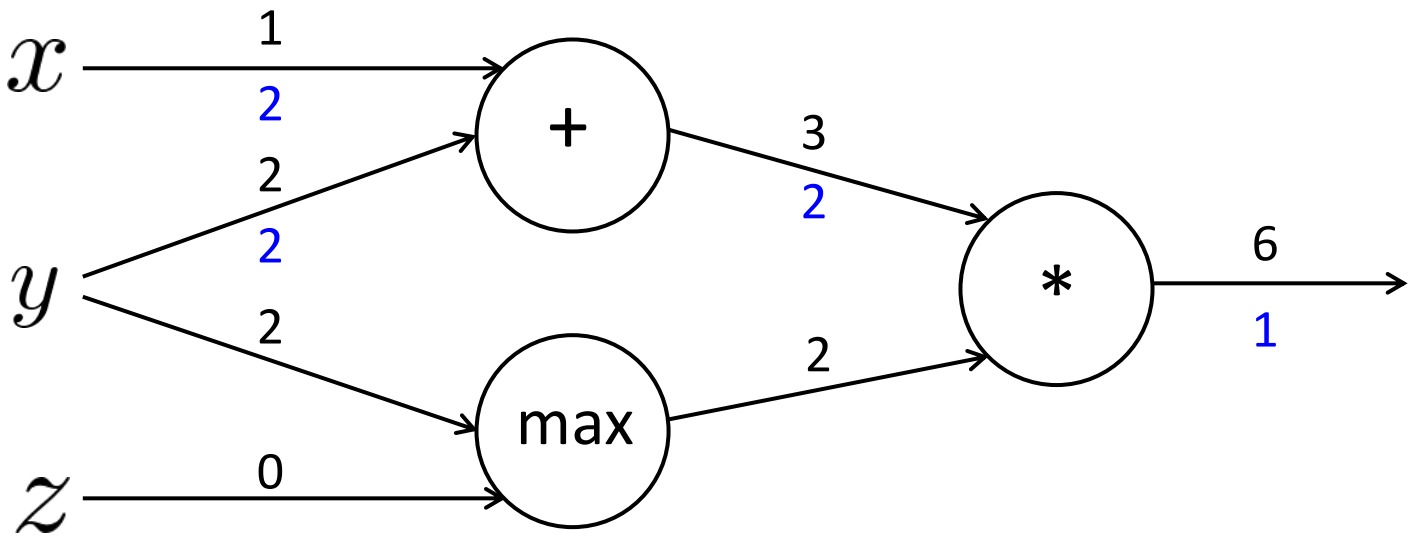
$$f = ab$$

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

节点直觉

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

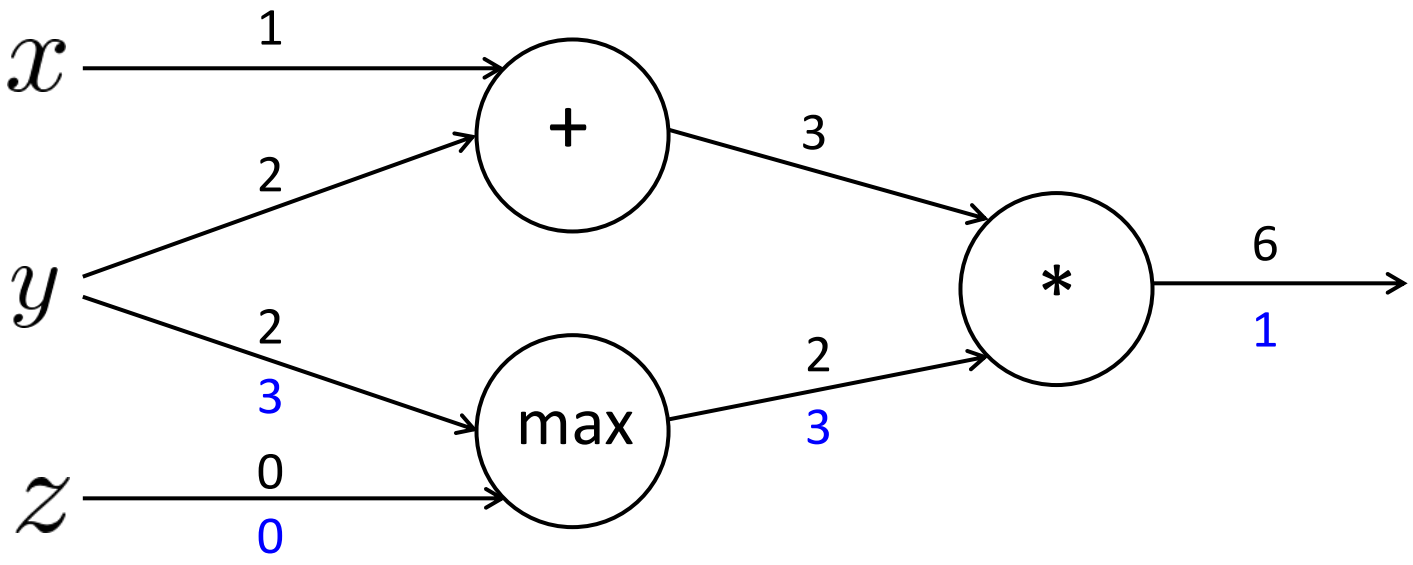
- + “distributes” the upstream gradient to each summand



节点直觉

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

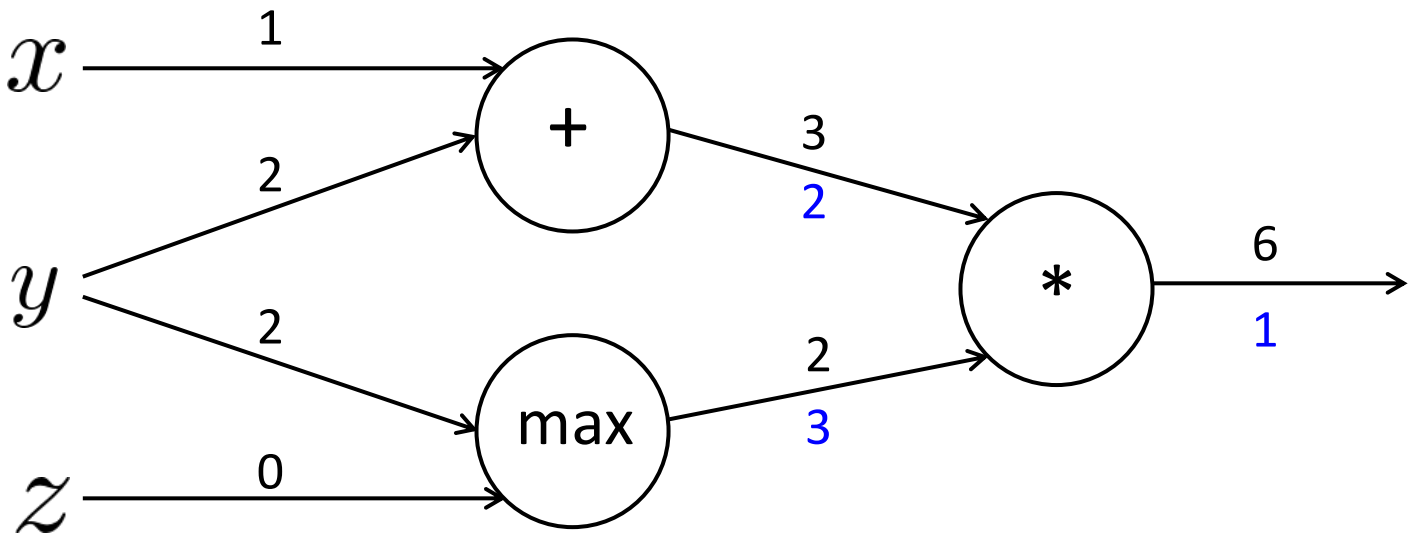
- + “distributes” the upstream gradient to each summand
- max “routes” the upstream gradient



节点直觉

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

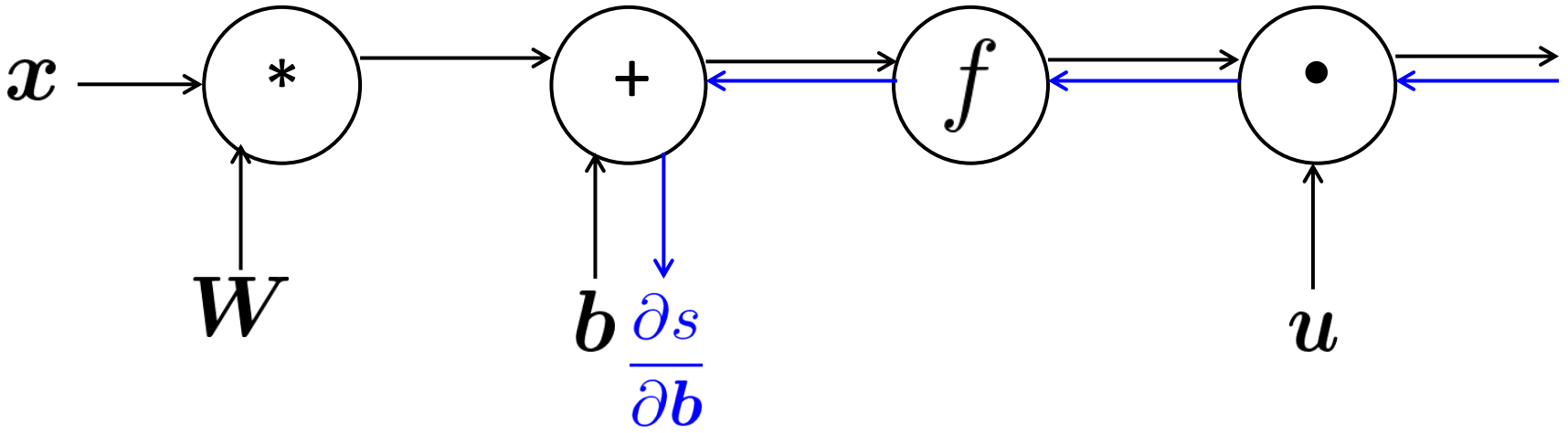
- + “distributes” the upstream gradient
- max “routes” the upstream gradient
- * “switches” the upstream gradient



Efficiency: compute all gradients at once

- Incorrect way 做 backprop 的方式:
  - First compute $\frac{\partial s}{\partial \boldsymbol{b}}$

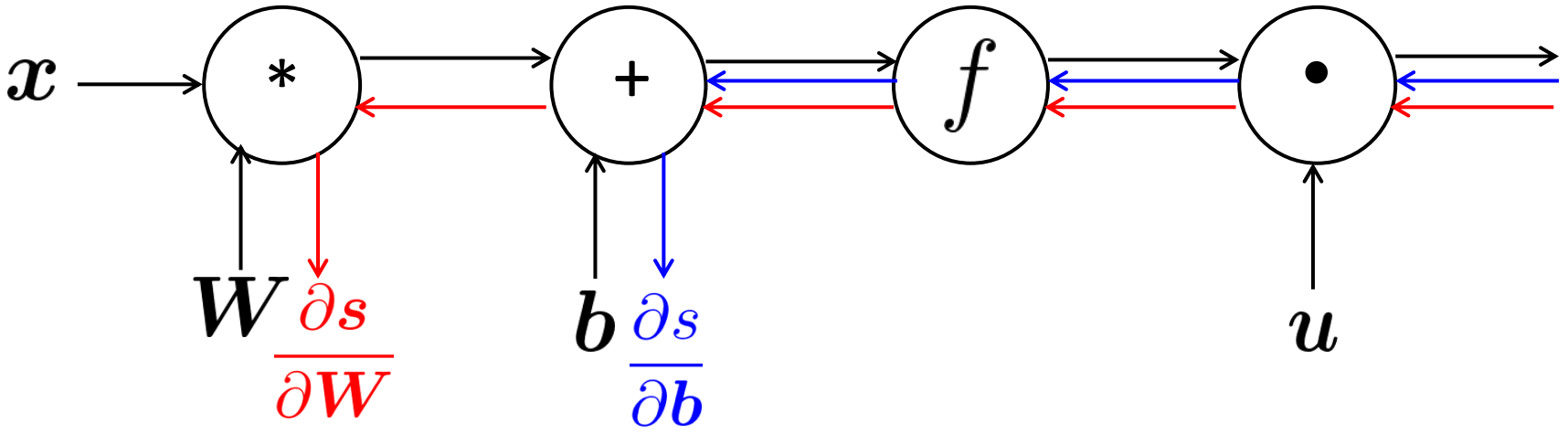
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$



Efficiency: compute all gradients at once

- Incorrect way 做 backprop 的：
  - First compute $\frac{\partial s}{\partial b}$
  - 然后独立计算 $\frac{\partial s}{\partial W}$
  - 重复计算！

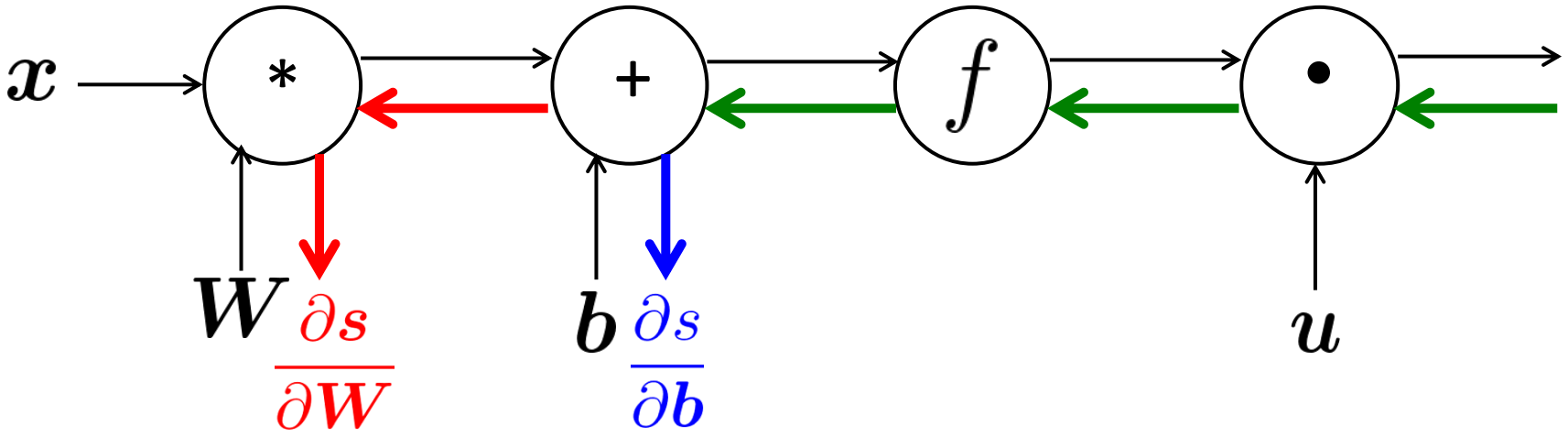
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$



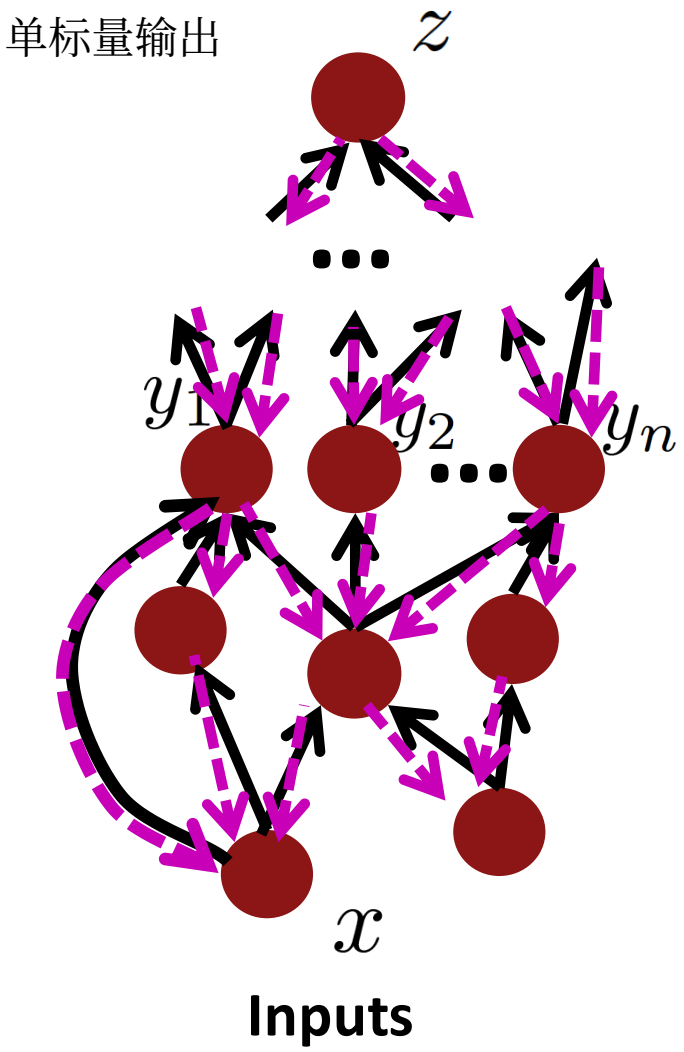
Efficiency: compute all gradients at once

- 正确方式：
  - 一次计算所有梯度
  - Analogous to using $\boldsymbol{\delta}$ when we 手动计算梯度

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



一般计算图中的 Back-Prop



1. Fprop: visit nodes in topological sort order
   - 给定前驱节点计算节点值
2. 反向传播：
   - initialize output gradient = 1
   - 按逆序访问节点：

Compute gradient wrt each node using 对后继的梯度

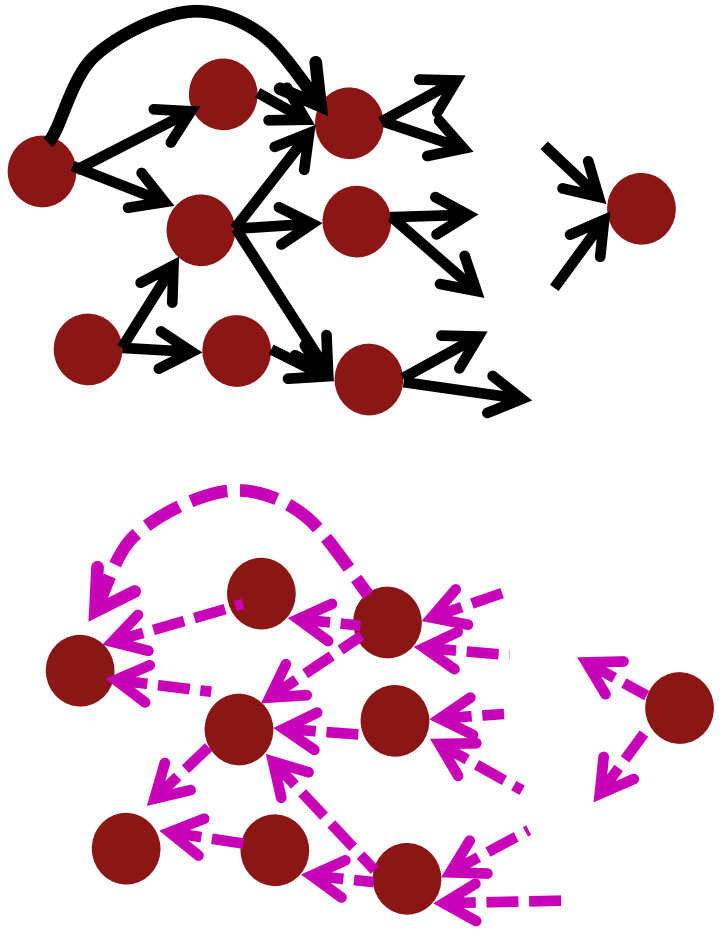
$\{y_1, y_2, \ldots y_n\}$ = successors of x

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

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

In general, our nets have regular layer-structure 所以我们可以使用矩阵和 Jacobian……

自动微分



- The gradient computation can be automatically inferred from the symbolic

  前向传播的表达式
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the

  对其输出的梯度
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you but mainly leave layer/node writer

  手动计算局部导数

Backprop 实现

```
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
```

手动梯度检查：数值梯度

- For small h (≈ 1e-4),

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- 容易正确实现
- But approximate and **very** slow:
  - You have to recompute f for **every parameter** 我们模型的
- 对检查你的实现有用
  - 在过去，我们手写所有代码，到处这样做是关键测试
  - 现在不太需要了；可以用它来检查层是否正确实现

总结

We've mastered the core technology of neural nets! 🎉🎉🎉

- **Backpropagation:** recursively (and hence efficiently) apply the chain rule 沿计算图
  - [下游梯度] = [上游梯度] x [局部梯度]
- **Forward pass:** compute results of operations and save intermediate values
- **Backward pass:** 应用链式法则计算梯度

为什么要学习所有这些梯度的细节？

- 现代 Deep Learning 框架为你计算梯度！
  - 本周五来参加 PyTorch 入门讲座！

- 但为什么还要上编译器或系统的课程，当它们已经为你实现了？
  - 了解底层原理是有用的！

- Backpropagation doesn’t always work perfectly out of the box
  - 理解原因对于调试和改进模型至关重要
  - 参见 Karpathy 的文章（在教学大纲中）：
    - https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
  - 在后续讲座中的示例：梯度爆炸和梯度消失