



自然语言处理

Natural Language Processing

Chapter 2

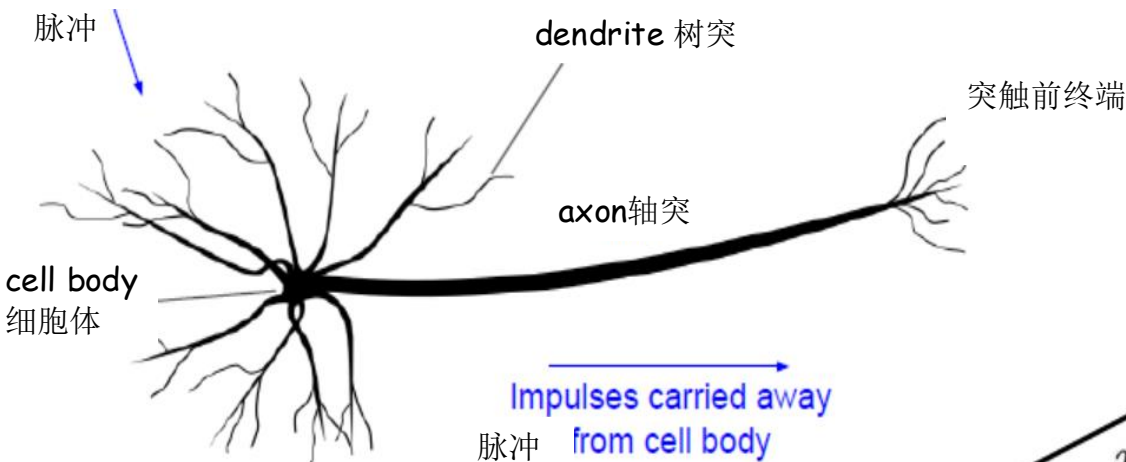
NLP中的前馈神经网络

Outline

- 前馈神经模型
- 反向传播算法
- NLP案例分析

非线性神经元(Non-linear Neurons)

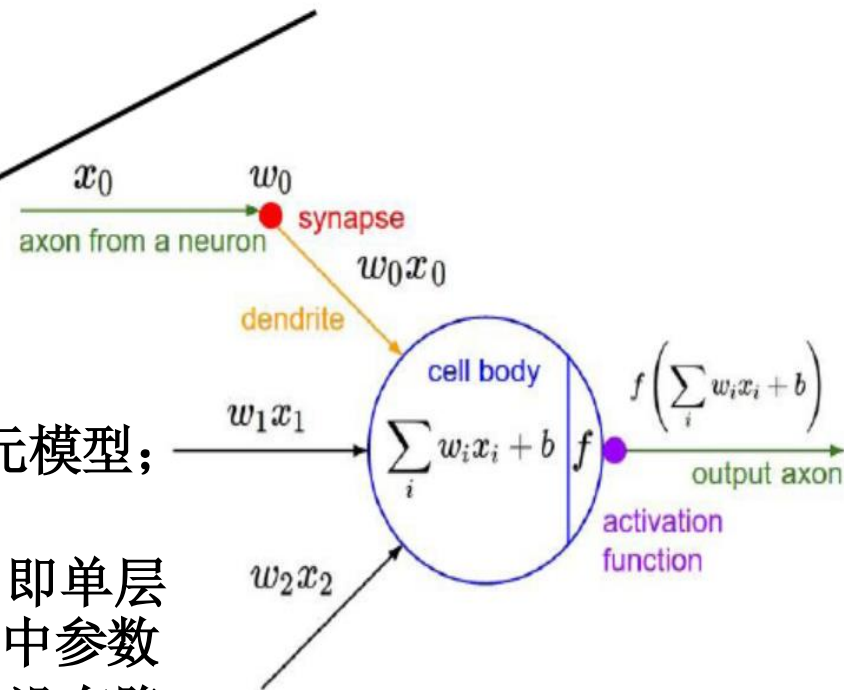
Impulses carried toward cell body



This image by Felipe Peruchio is licensed under [CC-BY 3.0](#)

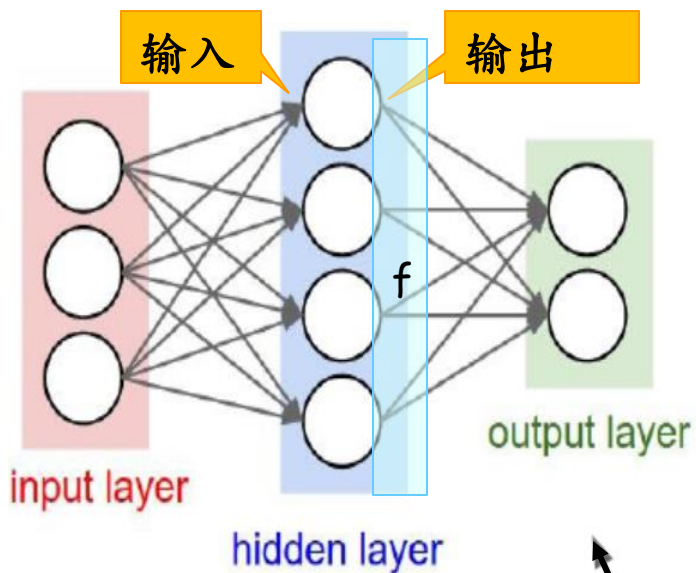
树突用于接受其他神经元传递的信号，轴突用于向其他神经元传递信号，信号在多个神经元之间传导，构成了神经网络。

- 人工神经元模型(1943)。aka MP神经元模型；参数 w , b 等预先设定
- 在此基础上提出最早的神经网络模型，即单层感知机(Single Layer Perceptron)，其中参数需要学习得到。只有输入层和输出层，没有隐藏层



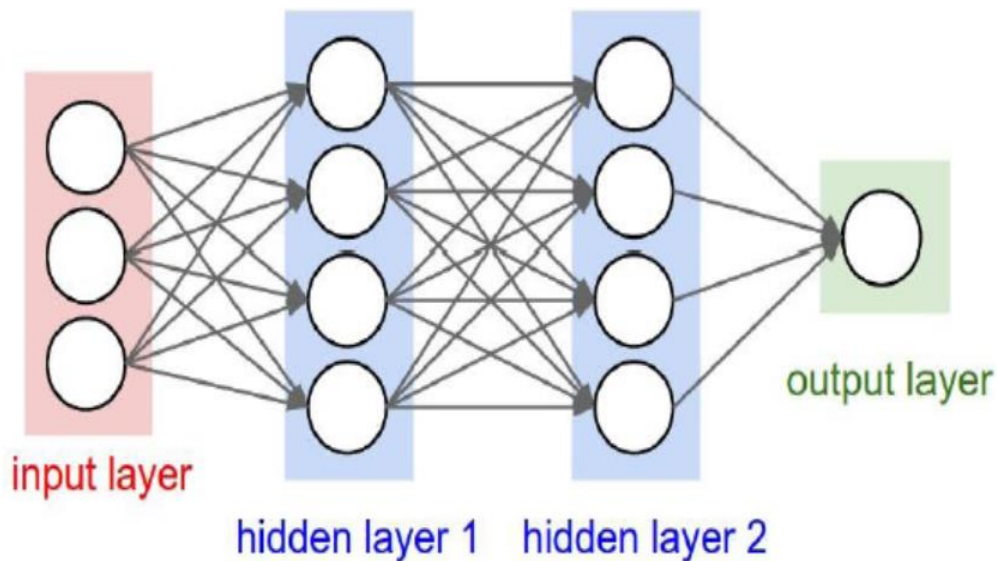
前馈神经网络

Feedforward Neural Network, FNN



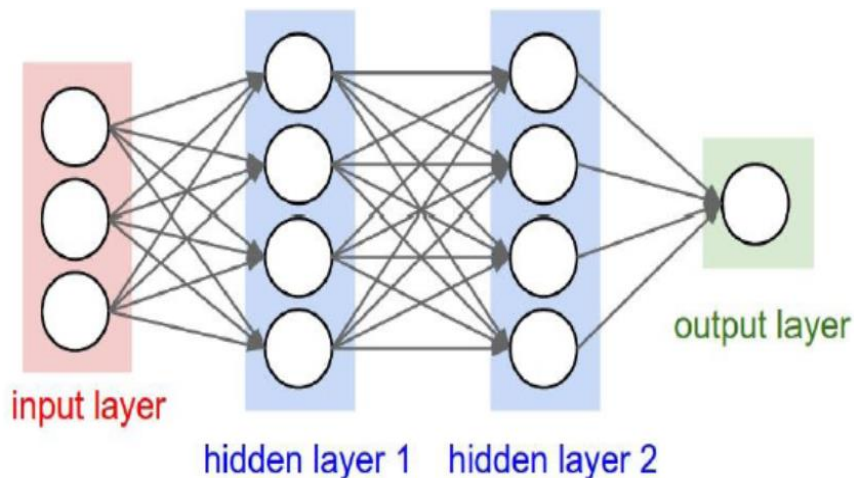
2层的神经网络 or
1个隐藏层的神经网络

全连接层：每一个结点都
与上一层的所有结点相连



3层的神经网络 or
2个隐藏层的神经网络

前馈神经网络



没有隐藏层：仅能够表示线性可分函数或决策

隐藏层数=1：可以拟合任何“包含从一个有限空间到另一个有限空间的连续映射”的函数

隐藏层数=2：搭配适当的激活函数可以表示任意精度的任意决策边界，且可以拟合任何精度的任何平滑映射

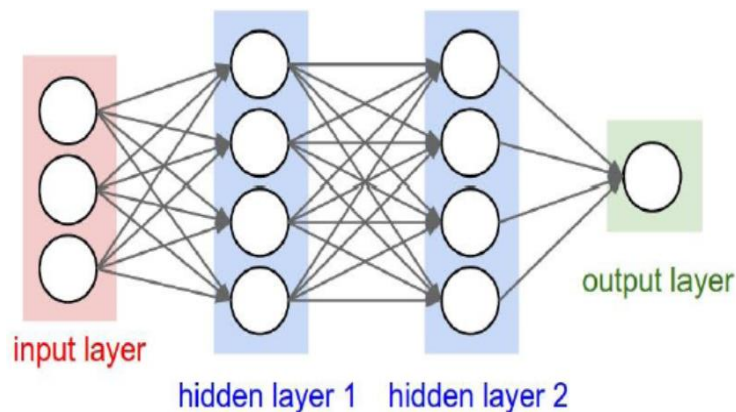
隐藏层数>2：多出来的隐藏层可以学习复杂的描述，类似于某种自动特征工程

通用近似定理：对于具有**线性输出层**和至少一个使用“挤压”性质的激活函数的**隐藏层**组成的前馈神经网络，只要其隐藏层神经元的数量足够，它可以以**任意精度**来近似任何从一个定义在**实数空间**中的**有界闭集函数**。

一个两层的神神经网络可以模拟任何函数。

模型介绍

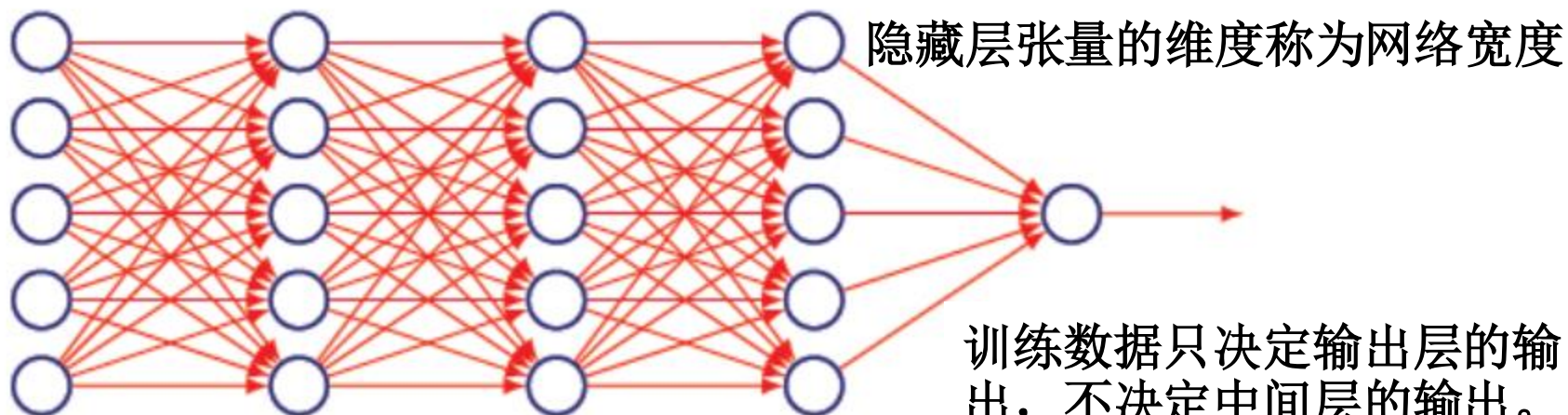
- **目标**: 近似某个未知的理想化函数 $f^*: X \rightarrow Y$
- **E.g. 分类器**: $y = f^*(x)$ with x and category y
- **前馈神经网络**: 定义参数化映射 $y = f(x; \theta)$ ，通过经验风险最小化和正则化在训练样本上学习参数 θ 以让函数 f 近似于 f^*
- **即**: 函数中的信息 x 流动开始于输入端，流经中间每一个计算层，最终产生输出 y
- 神经元没有反馈连接 (所以叫前馈)
- 受神经科学启发(所以叫神经网络)
- f 可以是多个函数的组合(所以是网络)
- 又叫**多层感知机MLP**，多层的连续的非线性函数



模型介绍

- 函数 f 是多个不同函数的组合（所以是网络）

e.g. $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ 链的长度称为网络深度 \rightarrow 深度学习



$f^{(1)}$: 第一层 $f^{(2)}$: 第二层 ...

训练数据只决定输出层的输出，不决定中间层的输出。

↓
隐藏层：学习算法不指明这些层应该怎么样

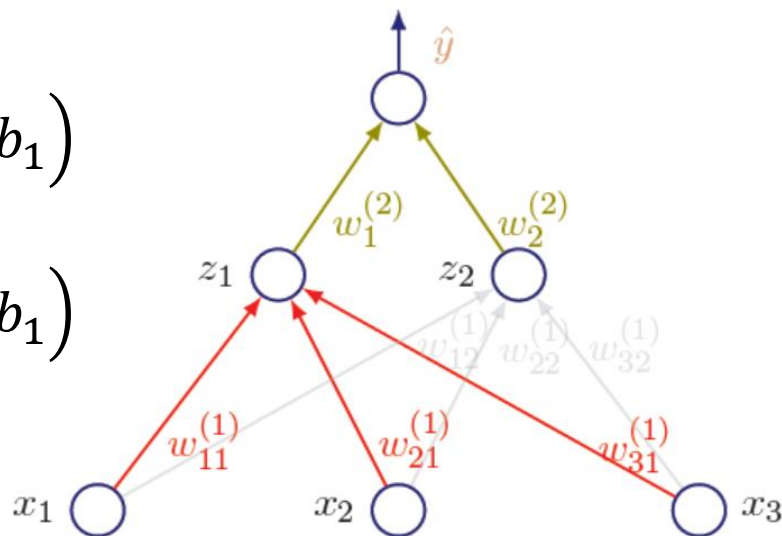
函数组合可以用有向无环图来描述，体现了前馈网络的思想！

简单的前馈神经网络-前向传播

$$z_1 = f(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1)$$

$$z_2 = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_1)$$

矩阵形式: $z = f(W^{(1)}x + b)$



假设 $z^{(0)} = x$:

$$a^{(l)} = W^{(l)}z^{(l-1)} + b^{(l)} \rightarrow \text{神经元的净输入/净活性值}$$

$$z^{(l)} = f(a^{(l)}) \rightarrow \text{神经元的输出/活性值}$$

Wx : 线性变换

$Wx+b$: 仿射变换

- Output $\hat{y} = w_1^{(2)}z_1 + w_2^{(2)}z_2$; Loss $L = (\hat{y} - y)^2$

隐藏层：激活函数

- 为什么要有激活函数：如果不使用激活函数，那么从输入层到隐藏层就是线性计算，无论叠加多少个隐藏层，都是线性的
- $z = W^T x + b \rightarrow$ 逐元素(elementwise)非线性函数 $f \rightarrow f(z)$
- 被激活： f 取值接近1
- 对于 f 的激活函数选择？
- 连续并可导的非线性函数，方便利用数值优化的方法来学习网络参数
函数及其导函数要尽可能的简单，有利于提高网络计算效率
导函数的值域要在一个合适的区间内，否则会影响训练的效率和稳定性

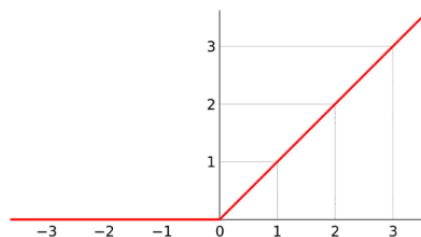
隐藏层：激活函数

■ ReLU (Linear rectification function 线性整流器)

$$f(x) = \max(0, x)$$

Dying ReLU Problem

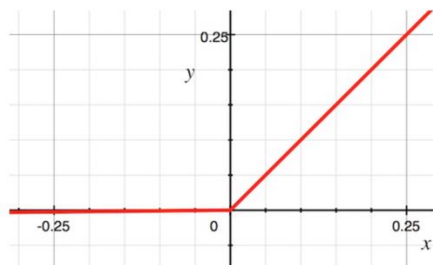
死神经元: 不恰当的更新后, 神经元不能被激活, 梯度为0



$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

■ Leaky ReLU

$$f(x) = \max(\alpha x, x)$$

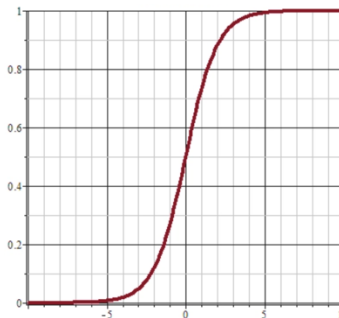


$$f'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

■ Sigmoid: 一类S型曲线

默认logistic函数:

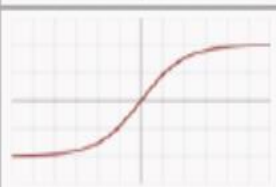
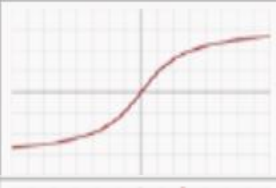

$$f(x) = 1 / (1 + \exp(-x))$$



$$f'(x) = f(x)(1 - f(x))$$

其它激活函数

- Sigmoid: 一类S型曲线。
- 两端饱和。左饱和: x 接近负无穷, 导数接近0; 右饱和: x 接近正无穷, 导数接近于0.

TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Parameteric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

可学习的参数

输出层：输出单元

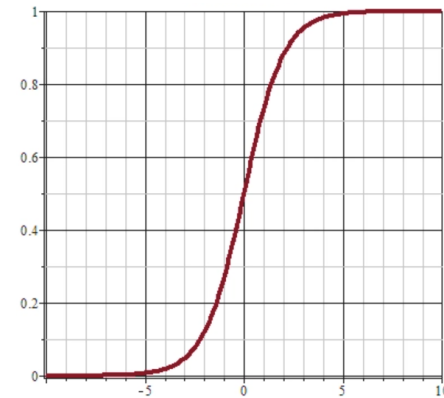
- 线性单元：给出特征 h , 线性输出层产生输出向量:

$$\hat{y} = w^T h + b$$

- sigmoid单元: e.g. logistic

$$\hat{y} = \sigma(w^T h + b)$$

$$\sigma(x) = 1/(1 + \exp(-x))$$



- softmax单元:

$$\hat{y} = \text{softmax}(w^T h + b)$$

分类问题: 产生预测的类

别向量 $\hat{y}_i = p(y = i|x)$

$$\text{softmax}(x)_i = \exp(x_i) / \sum_j \exp(x_j)$$

$$\log \text{softmax}(x)_i = x_i - \log \sum_j \exp(x_j)$$

Loss Functions

- 如何衡量模型的好坏? 计算损失 **loss**/代价 **cost**

e.g. 分类 $y = (1, 0, 0)$, $\hat{y}_A = (0.9, 0.05, 0.05)$, $\hat{y}_B = (0.4, 0.3, 0.3)$

- 使用负对数概率作为损失 **negative log probabilities**:

$$-\log 0.9 = 0.05, -\log 0.4 = 0.92$$

$$L = -\log \prod \hat{y}^y$$

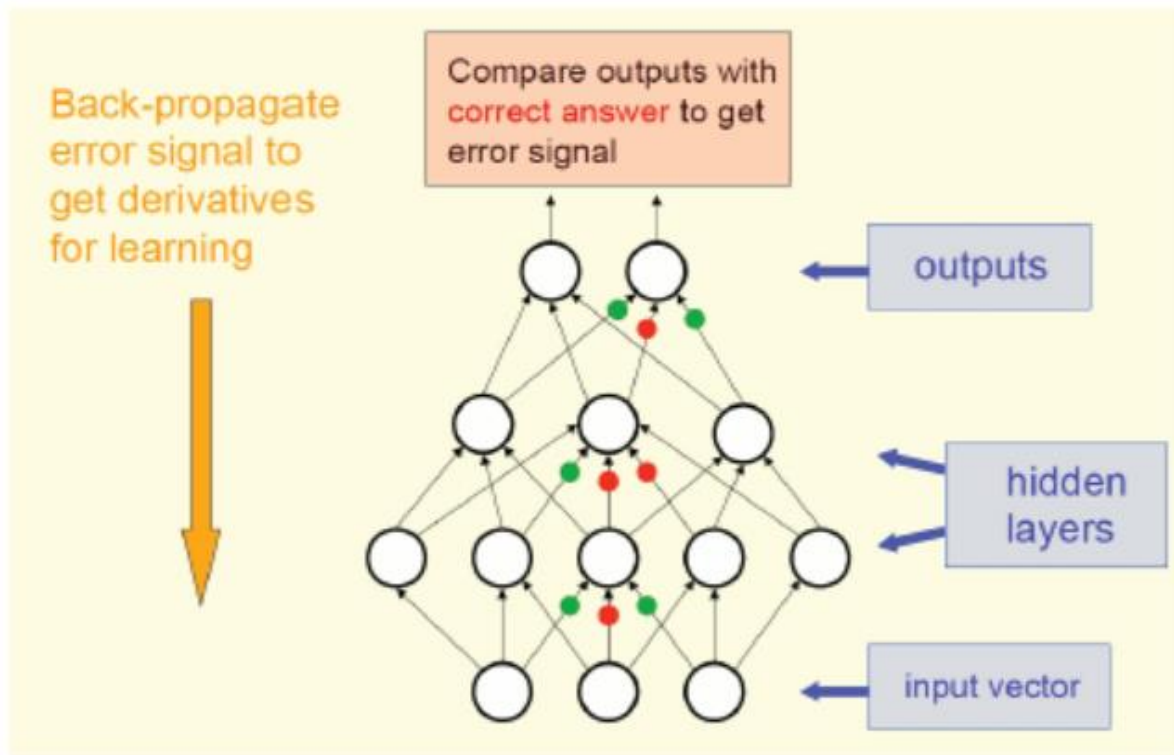
- 等同于交叉熵损失 **cross entropy**, 交叉熵损失通过模型真实输出和预测输出计算得到:

$$L = -\sum y \log(\hat{y})$$

Outline

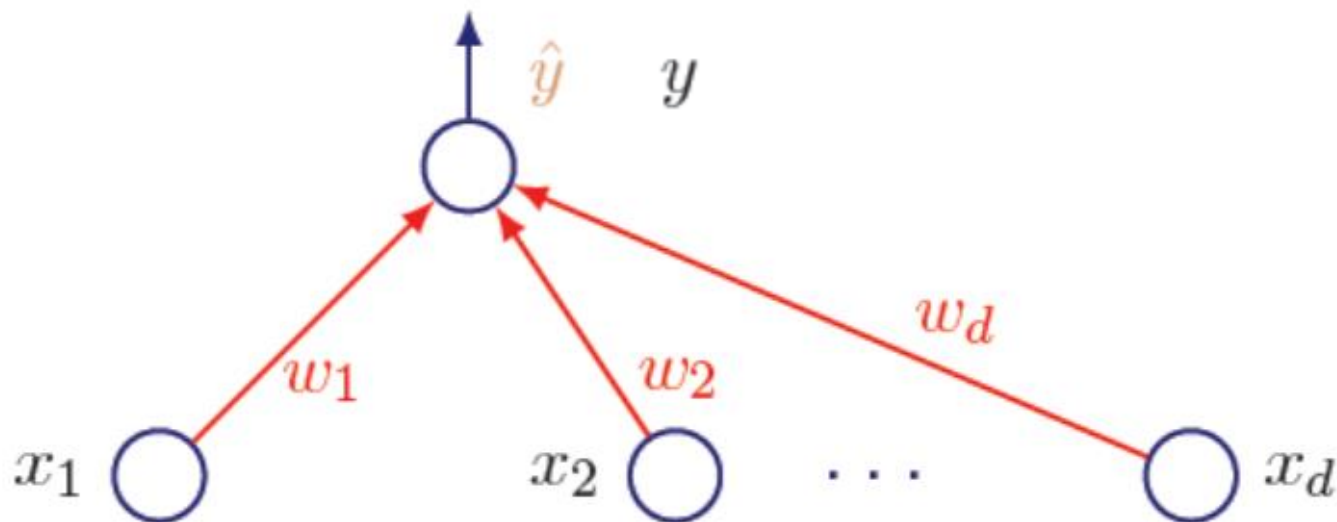
- 前馈神经模型
- 反向传播算法
- 案例分析

反向传播算法



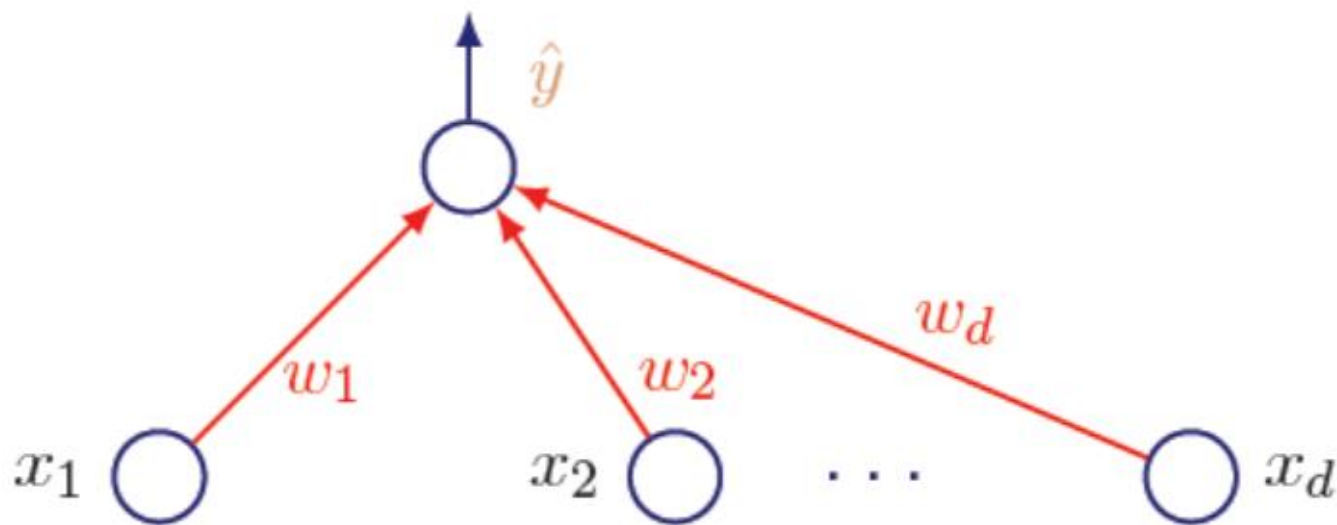
- 前向传播: 接受输入 \mathbf{x} , 通过中间层传输并获得输出 $\hat{\mathbf{y}}$
- 训练过程中: 使用 $\hat{\mathbf{y}}$ 和 \mathbf{y} 计算标量损失 $J(\theta)$
- 反向传播: 信息从损失开始, 向输入端反向地流动, 以计算梯度

例子：一个神经元的线性输出层



- 假设输入样本 \mathbf{x} ; 输出为 \hat{y} ; \mathbf{x} 的正确结果为 y
- $\hat{y} = \mathbf{x}^T \mathbf{w} = x_1 w_1 + x_2 w_2 + \dots + x_d w_d$
- 取平方损失 $L = (y - \hat{y})^2$

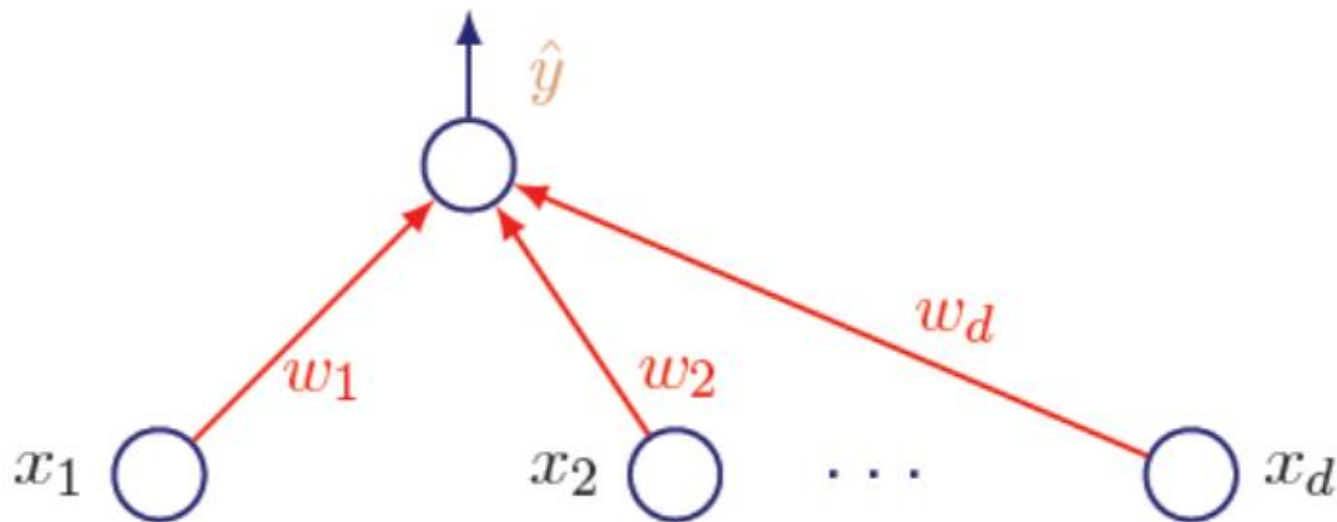
例子：线性输出层



- 模型优化目标：更新 w_i
- 更新规则： $w_i := w_i - \eta \frac{\partial L}{\partial w_i}$ 重点：求导
- 得： $\frac{\partial L}{\partial w_i} = \frac{\partial (\hat{y} - y)^2}{\partial w_i} = 2(\hat{y} - y) \frac{\partial (x_1 w_1 + x_2 w_2 + \dots x_d w_d)}{\partial w_i}$

链式法则

例子：线性输出层



- 根据：
$$\frac{\partial L}{\partial w_i} = \frac{\partial (\hat{y} - y)^2}{\partial w_i} = 2(\hat{y} - y) \frac{\partial (x_1 w_1 + x_2 w_2 + \dots x_d w_d)}{\partial w_i}$$
$$= 2(\hat{y} - y) x_i$$

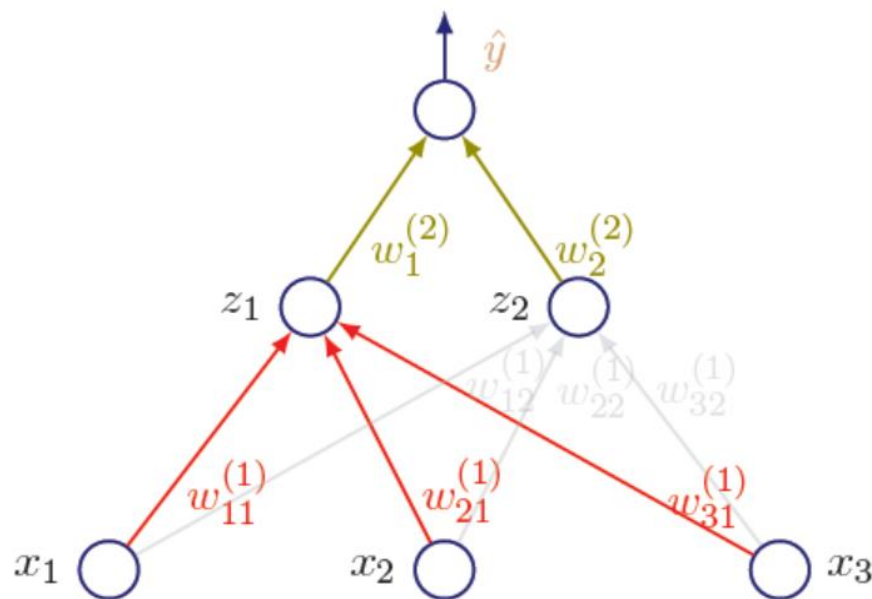
- 按照更新规则：

$$w_i := w_i - \eta(\hat{y} - y)x_i = w_i - \eta\delta x_i \text{ where } \delta = (\hat{y} - y)$$

- 向量形式： $\mathbf{w} := \mathbf{w} - \eta\delta\mathbf{x}$

例子：单隐藏层的前馈网络

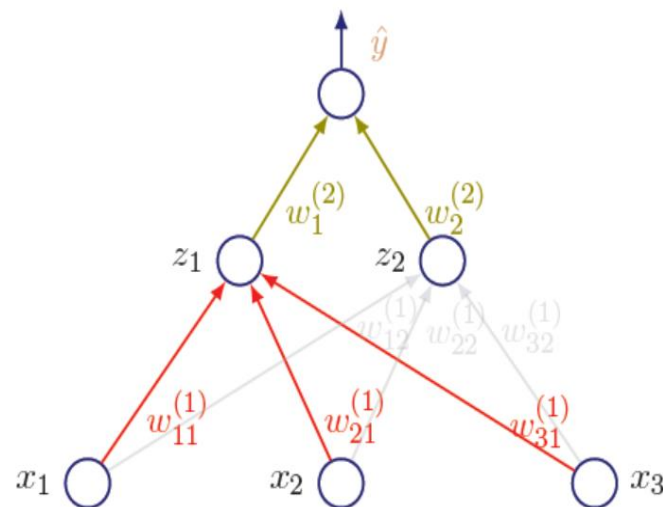
- $z_1 = \tanh(a_1)$ where $a_1 = w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3$
- $z_2 = \tanh(a_2)$ where $a_2 = w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3$



- Output $\hat{y} = w_1^{(2)}z_1 + w_2^{(2)}z_2$; Loss $L = (\hat{y} - y)^2$
- 需要计算损失 L 对于每一层每一个权重参数的偏导数

顶层

- 目标：计算 $\frac{\partial L}{\partial w_1^{(2)}}$ 以及 $\frac{\partial L}{\partial w_2^{(2)}}$
- 首先考虑 $w_1^{(2)}$



$$\frac{\partial L}{\partial w_1^{(2)}} = \frac{\partial (\hat{y} - y)^2}{\partial w_1^{(2)}} = 2(\hat{y} - y) \frac{\partial (w_1^{(2)} z_1 + w_2^{(2)} z_2)}{\partial w_1^{(2)}} = 2(\hat{y} - y) z_1$$

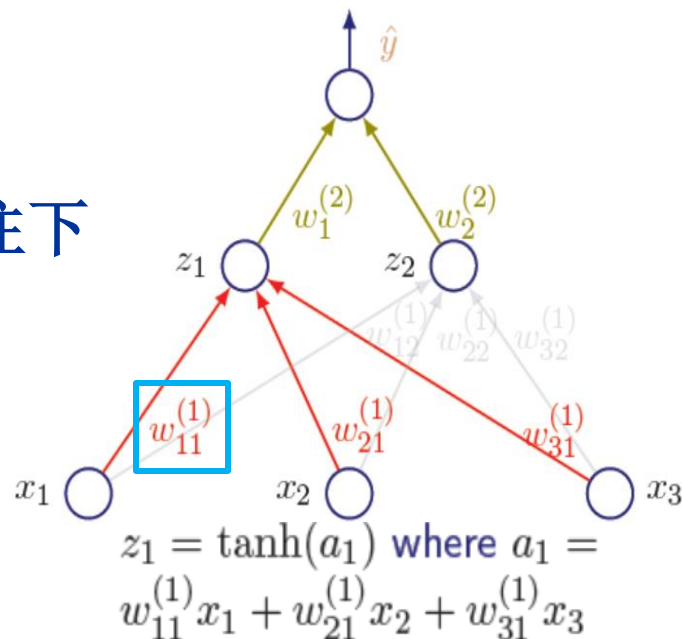
$$w_1^{(2)} := w_1^{(2)} - \eta \frac{\partial L}{\partial w_1^{(2)}} = w_1^{(2)} - \eta \delta z_1 \text{ with } \delta = (\hat{y} - y)$$

- 同样，对于 $w_2^{(2)}$:

$$w_2^{(2)} := w_2^{(2)} - \eta \delta z_2$$

往下的层

- 图中反向传播的第二层（对上层而言往下的一层）有6个权重参数
- 以 $w_{11}^{(1)}$ 为例说明，其余类似
- 首先，根据上页内容计算出：



- $$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial (\hat{y} - y)^2}{\partial w_{11}^{(1)}} = 2(\hat{y} - y) \frac{\partial (w_1^{(2)} z_1 + w_2^{(2)} z_2)}{\partial w_{11}^{(1)}}$$
- 代入 z : $\frac{\partial (w_1^{(2)} z_1 + w_2^{(2)} z_2)}{\partial w_{11}^{(1)}} = w_1^{(2)} \frac{\partial (\tanh(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3))}{\partial w_{11}^{(1)}} + 0$
 - 即: $w_1^{(2)} (1 - \tanh^2(a_1)) x_1$ $\tanh'(x) = 1 - \tanh(x)^2$
 - 因此: $\frac{\partial L}{\partial w_{11}^{(1)}} = 2(\hat{y} - y) w_1^{(2)} (1 - \tanh^2(a_1)) x_1$

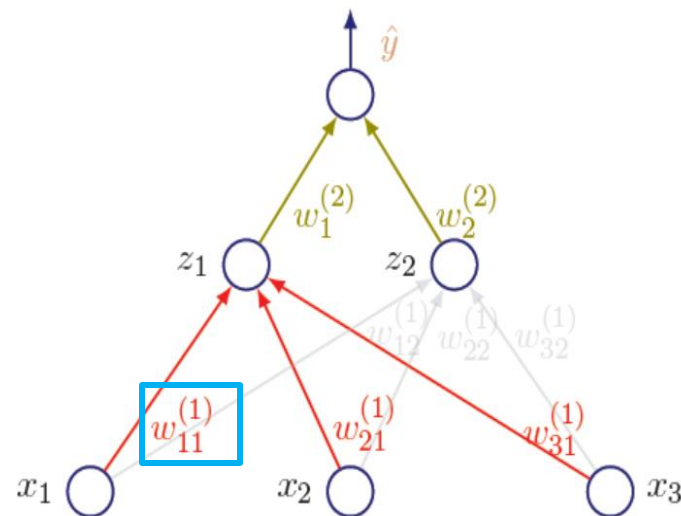
往下的层

$$\frac{\partial L}{\partial w_{11}^{(1)}} = 2(\hat{y} - y)w_1^{(2)}(1 - \tanh^2(a_1))x_1$$

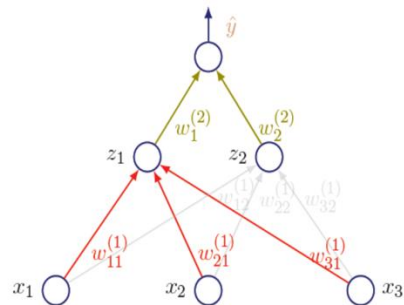
- 权重更新:

$$w_{11}^{(1)} := w_{11}^{(1)} - \eta \frac{\partial L}{\partial w_{11}^{(1)}}$$

- 同样: $\frac{\partial L}{\partial w_{ij}^{(1)}} = 2(\hat{y} - y)w_j^{(2)}(1 - \tanh^2(a_j))x_i$



反向传播过程梳理



- 对于反向传播的第一层: $\frac{\partial L}{\partial w_i^{(2)}} = (\hat{y} - y)z_i = \delta z_i$ (忽略 2)

- 改写:
$$\frac{\partial L}{\partial w_i^{(2)}} = \underbrace{\delta}_{\text{local error}} \underbrace{z_i}_{\text{local input}}$$

局部误差项

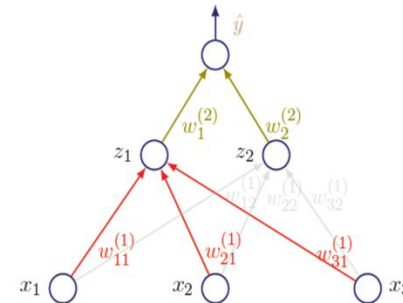
- 假设 $L=g(\hat{y}), \hat{y}=f(w_i^{(2)})$

$$\text{则 } \frac{\partial L}{\partial w_i^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i^{(2)}}, \frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b^{(2)}}$$

$\delta = \frac{\partial L}{\partial \hat{y}}$: 从loss传来的误差信号, 是loss关于本层净输入($wz+b$)的偏导数。

一次计算可用于多个参数的梯度计算。反映了不同神经元对网络能力的贡献程度, 解决了贡献度分配问题。

反向传播过程梳理



- 往下一层:

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = (\hat{y} - y) w_j^{(2)} (1 - \tanh^2(a_j)) x_i$$

- 令 $\delta_j = (\hat{y} - y) w_j^{(2)} (1 - \tanh^2(a_j)) = \delta w_j^{(2)} (1 - \tanh^2(a_j))$

动态规划

- 则: $\frac{\partial L}{\partial w_{ij}^{(1)}} = \delta_j x_i$

local error

local input

本层净输入: $a = wx$. $\frac{\partial L}{\partial a_i} = (\hat{y} - y) \frac{\partial (w_1^{(2)} z_1 + w_2^{(2)} z_2)}{\partial a_i}$

$$\frac{\partial (w_1^{(2)} z_1 + w_2^{(2)} z_2)}{\partial a_i} = \frac{\partial (w_i^{(2)} z_i)}{\partial a_i} = w_i^{(2)} \frac{\partial (\tanh(a_i))}{\partial a_i}$$

$$\frac{\partial (\tanh(a_i))}{\partial a_i} = 1 - \tanh^2(a_i)$$

$$\frac{\partial L}{\partial a_i} = (\hat{y} - y) w_i^{(2)} (1 - \tanh^2(a_i)) = \delta_j$$

即 δ_j 是 loss 关于本层净输入的偏导数

局部误差信号 δ

- $\delta^{(l)}$: 第 l 层的局部误差。 $\delta^{(l)}$ 和 $\delta^{(l+1)}$ 的关系?

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}}$$

$$\begin{aligned} \mathbf{z}^l &= W^l \mathbf{a}^{l-1} + \mathbf{b}^l \\ \mathbf{a}^l &= f_l(\mathbf{z}^l) \end{aligned}$$

$$\begin{aligned} &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l+1)}} \\ &= \text{diag}(f'_l(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^\top \cdot \delta^{(l+1)} \\ &= f'_l(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^\top \delta^{(l+1)}), \end{aligned}$$

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \begin{bmatrix} \frac{\partial f_l(z_1^{(l)})}{\partial z_1^{(l)}} & \cdots & \frac{\partial f_l(z_m^{(l)})}{\partial z_m^{(l)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_l(z_m^{(l)})}{\partial z_1^{(l)}} & \cdots & \frac{\partial f_l(z_m^{(l)})}{\partial z_m^{(l)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_l(z_1^{(l)})}{\partial z_1^{(l)}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial f_l(z_m^{(l)})}{\partial z_m^{(l)}} \end{bmatrix} = \text{diag}(f'_l(\mathbf{z}^{(l)}))$$

局部误差信号 δ

- $\delta^{(l)}$: 第 l 层的局部误差。 $\delta^{(l)}$ 和 $\delta^{(l+1)}$ 的关系?

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}} \quad \begin{array}{l} z^l = W^l a^{l-1} + b^l \\ a^l = f_l(z^l) \end{array} \quad \boxed{z^{l+1} = W^{l+1} a^l + b^{l+1}}$$

$$\begin{aligned} &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l+1)}} \\ &= \text{diag}(f'_l(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^\top \cdot \delta^{(l+1)} \\ &= f'_l(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^\top \delta^{(l+1)}), \end{aligned}$$

第 l 层的误差项可以通过第 $l+1$ 层的误差项计算得到。

Hadamard product哈达玛积
矩阵对应位置相乘，形状不变

$$\frac{\partial \mathcal{L}(W, \mathbf{b}; \mathbf{x}, y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^\top$$

local error local input

反向传播算法的含义是：第 l 层的一个神经元的误差项是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后，再乘上该神经元激活函数的梯度

参数学习：优化器

- (Batch) Gradient Descent 梯度下降

$$\theta \leftarrow \theta + \epsilon \nabla_{\theta} \sum_t L(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}; \theta)$$

每次进行参数更新需要
计算整个数据集的样本

- Stochastic Gradient Descent (SGD) 随机梯度下降
 - Online GD (取1个样本) Minibatch SGD (打乱 取m个)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

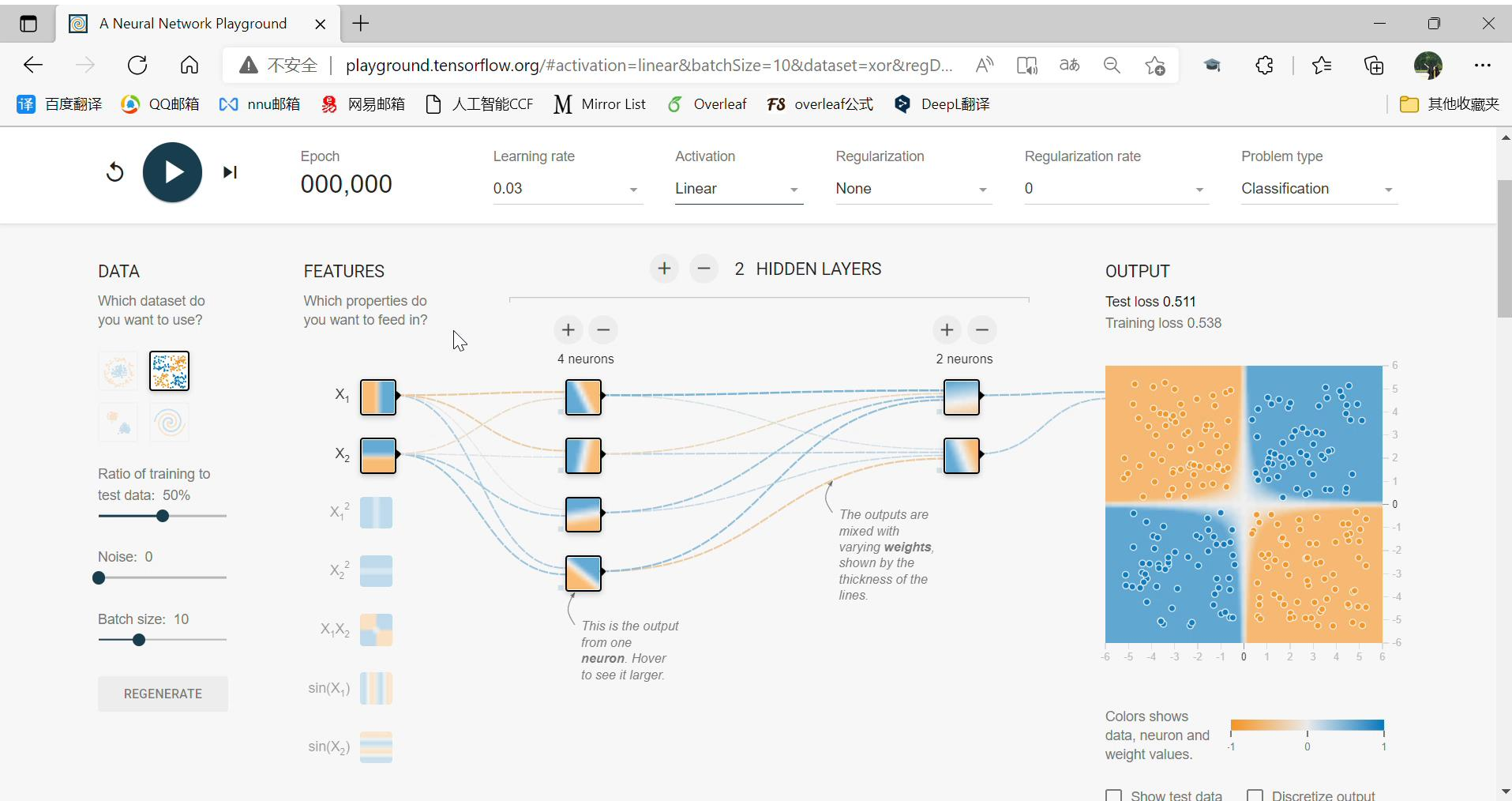
参数学习

- 其他优化器: RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, Ftrl...

<https://keras.io/api/optimizers/>

- 网络层数
- 每层的神经元个数
- 损失函数设计（正则化项）
- 学习率

A Neural Network Playground



A Neural Network Playground

反向传播梳理

- 神经网络中的反向传播是对于权重(等参数项)的误差的贡献度分配
 - ✓ 从前向传播的输出(**output, y**) 开始, 往回反向计算, 一直到达待求导的参数位置
 - ✓ 通过**链式法则**, 沿用前面步骤的计算结果, 即局部误差 δ , 一直可以追溯到最上层的 δ , 体现了**动态规划**。
- 反向传播仅仅指计算梯度的方法
- 梯度计算完成后, 使用另一种算法: 优化器 (**e.g. SGD**), 来使用梯度进行学习、实现对模型参数的更新

反向传播梳理

- 使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步：

(1) 前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$ ，直到最后一层；

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$$
$$\mathbf{a}^l = f_l(\mathbf{z}^l)$$

(2) 反向传播计算每一层的误差项 $\delta^{(l)}$ ；

(3) 计算每一层参数的偏导数，并更新参数

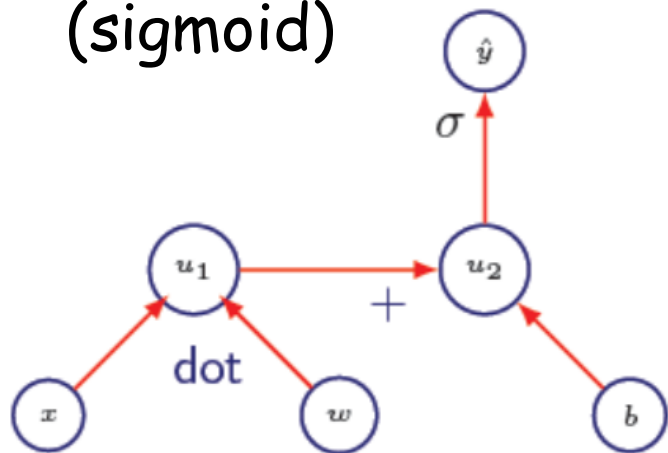
- 反向传播算法的本质：链式法则 + 动态规划

计算图

- 将梯度计算过程通过图(graph)形式化表示, 更加简洁
- **节点** 表示变量或常量 (标量scalar, 向量vector, 张量tensor 等等)
- **边** 表示一个或多个变量的简单函数/操作
- 计算图有一系列可能的操作, e.g. sigmoid, tanh, max...

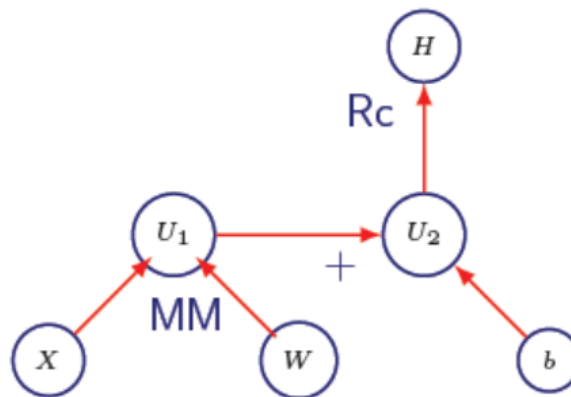
计算图例子

Logistic
(sigmoid)



$$\hat{y} = \sigma(\mathbf{x}^T \mathbf{w} + b)$$

$$H = \max\{0, \mathbf{XW} + b\}$$

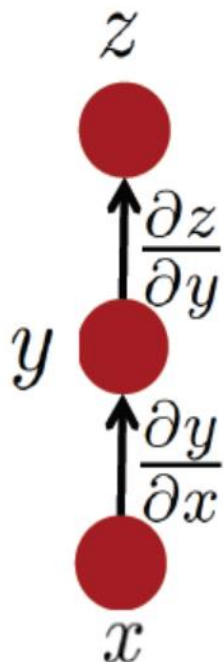


- MM: 矩阵乘法
- Rc: ReLU

链式法则中的计算图

- 反向传播中可以使用链式法则，利用已经计算过的梯度值，帮助简化靠近输入位置的参数的梯度计算
- Let $f, g: \mathbb{R} \rightarrow \mathbb{R}$
- 假设 $y=g(x)$, $z=f(y)=f(g(x))$
- 根据链式法则 $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$

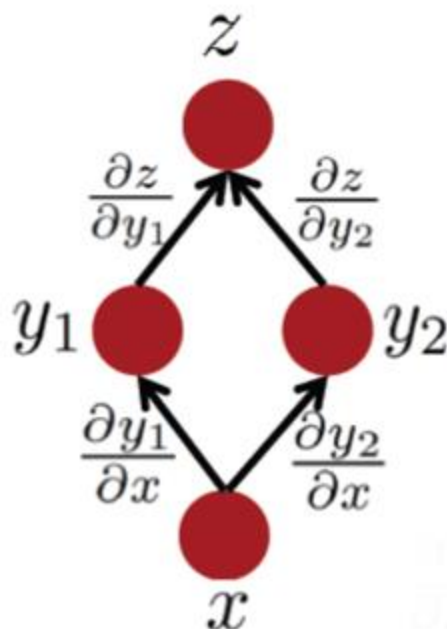
链式法则



$$y = g(x)$$

$$z = f(y) = f(g(x))$$

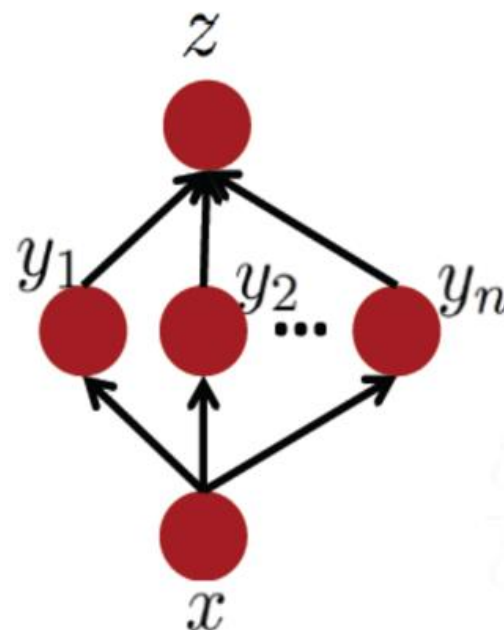
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$



$$y_1 = f_1(x), y_2 = f_2(x)$$

$$z = g(y_1, y_2)$$

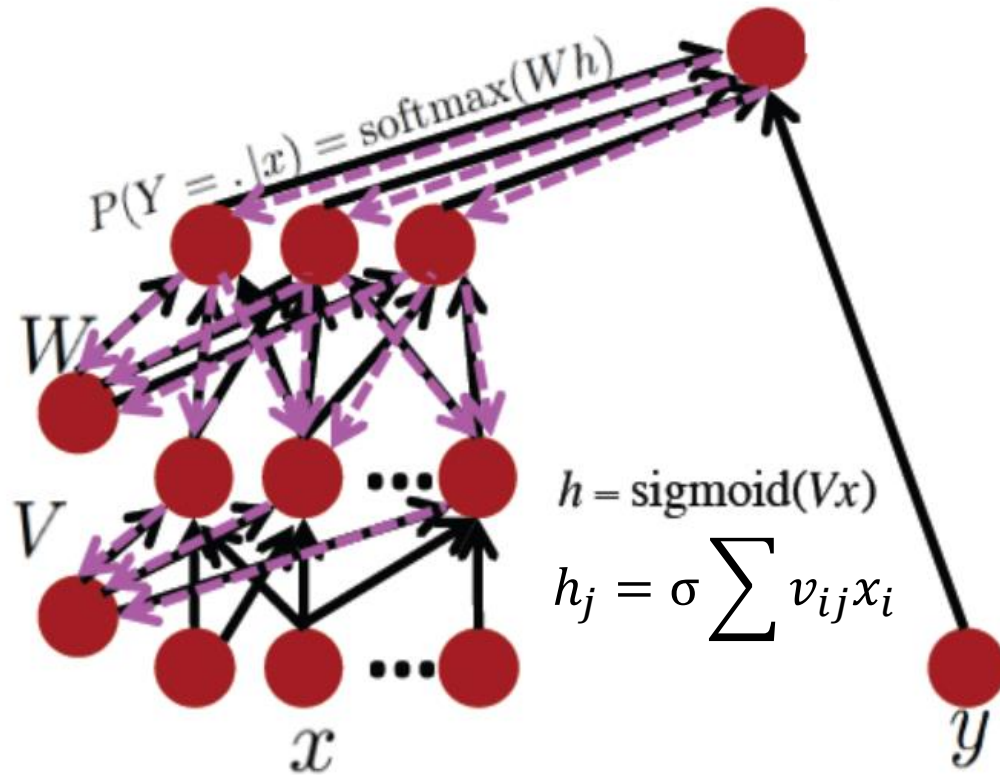
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$



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

多变量链式法则

链式法则



$$\text{Cost} = -\log(Y=y|x)$$

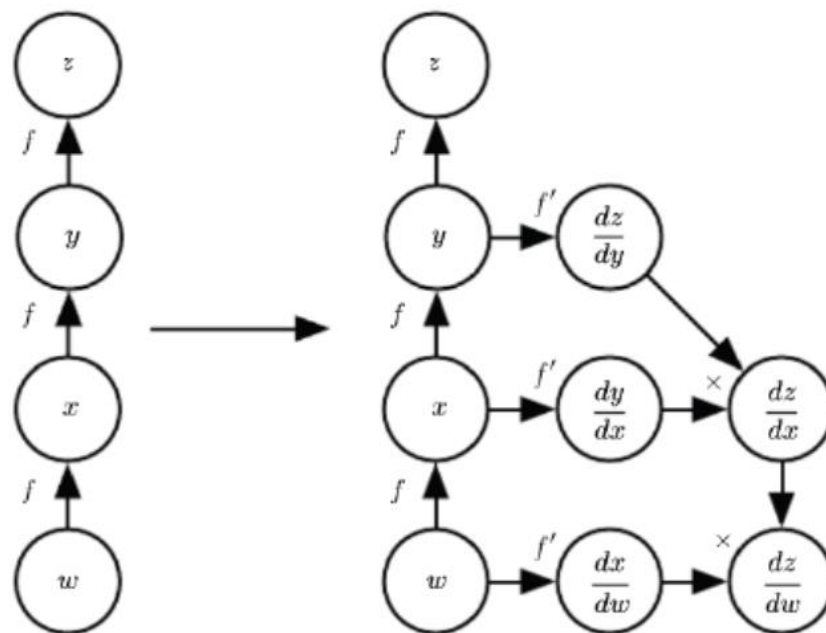
自动梯度计算

- 手动求导过程琐碎并容易出错，导致实现神经网络变得十分低效。
- 目前主流的深度学习框架都包含了自动梯度计算的功能
E.g. 在pytorch中: `loss.backward()`
- 更快更便捷的原型实验结果
- 自动计算梯度的方法可以分为以下三类:
 - 数值微分：使用数值方法计算 $f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$
难以找到合适的扰动 Δx ；计算复杂度
 - 符号微分：基于符号计算/代数计算。变量看成符号，不需要代入具体数值，处理对象是数学表达式。

自动梯度计算

- 自动微分：处理对象是一个函数或一段程序。
- 基本原理：所有的数值计算可以分解为一些基本操作(+, -, ×, / 和一些初等函数 **exp, log, sin, cos** 等)，然后利用链式法则来自动计算一个复合函数的梯度。
- 自动微分可以直接在原始程序代码进行微分，因此自动微分成为目前大多数深度学习框架的首选。
- 两种基本方式：**symbol to symbol; symbol to number**

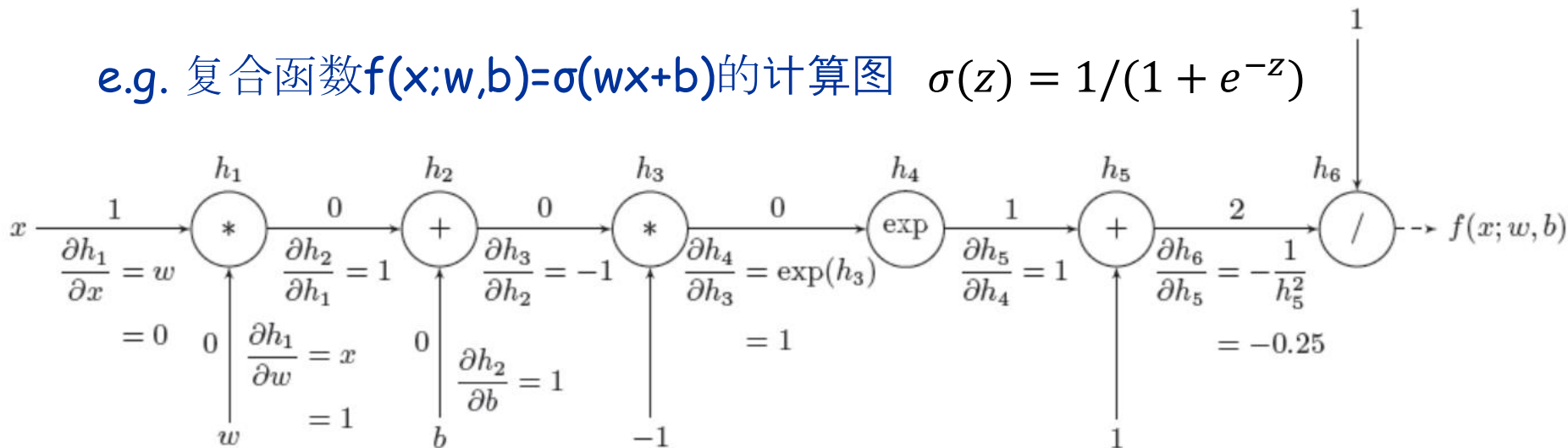
Symbol to Symbol



- 在此方式中反向传播只有符号，不会出现具体数值
- 在此方式中，把描述了导数计算的节点加入到图中
- **A graph evaluation engine** 进行具体的计算
- 典型应用: Theano, TensorFlow 1.0

Symbol to Number

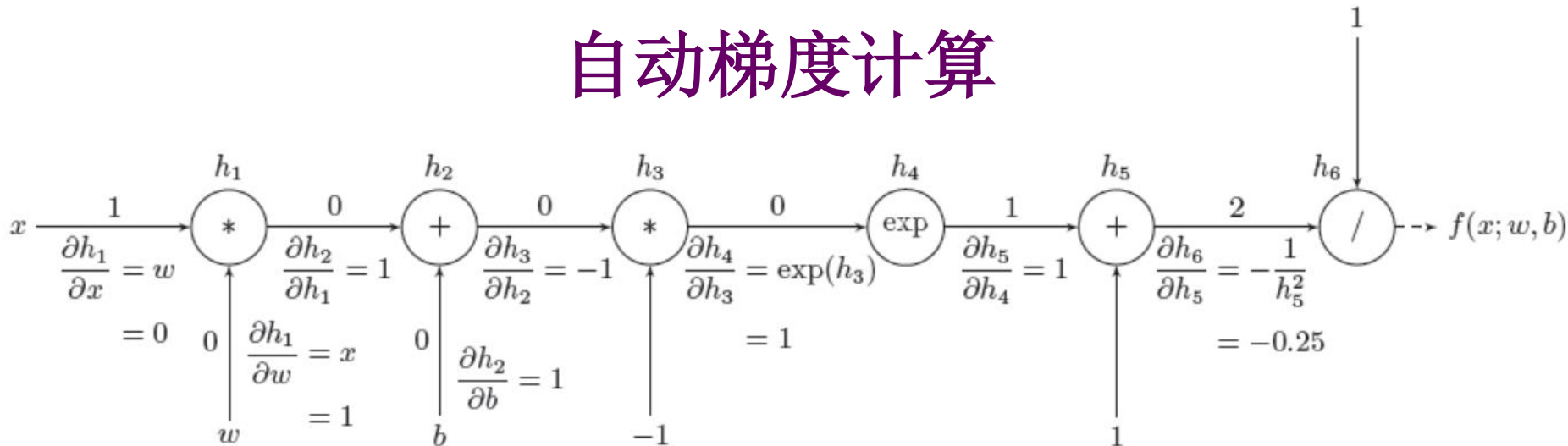
e.g. 复合函数 $f(x;w,b)=\sigma(wx+b)$ 的计算图 $\sigma(z) = 1/(1 + e^{-z})$



分解成6个基本函数，每个函数的导数都很简单，可以通过规则实现按照计算导数的顺序，自动微分可以分为前向和反向两种模式。

- **(1) 前向模式：**按计算图中计算方向来递归地计算梯度。需要对每个输入变量遍历。
- 计算 $\frac{\partial f}{\partial w}$ 为例，计算顺序：先计算 $\frac{\partial h_1}{\partial w} = x = 1$ ，再计算 $\frac{\partial h_2}{\partial w} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial w} = 1 * 1 = 1$ ， $\frac{\partial h_3}{\partial w} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial w} = \dots$ 。直到 $\frac{\partial f}{\partial w} = \frac{\partial f}{\partial h_6} \frac{\partial h_6}{\partial w} = \dots$

自动梯度计算



▪ (2) 反向模式：和反向传播的计算梯度的方式相同。对输出遍历。

▪ 顺序： $\frac{\partial f}{\partial h_6} = 1$ ，再计算 $\frac{\partial f}{\partial h_5} = \frac{\partial f}{\partial h_6} \frac{\partial h_6}{\partial h_5} = 1 * -0.25 = -0.25 \dots$

直到 $\frac{\partial f}{\partial w} = \frac{\partial f}{\partial h_1} \frac{\partial h_1}{\partial w} = \dots$ 一般神经网络的输出层神经元远少于输入层，

所以反向模式用得更多

$$\begin{aligned} \frac{\partial f(x; w, b)}{\partial w} \Big|_{x=1, w=0, b=0} &= \frac{\partial f(x; w, b)}{\partial h_6} \frac{\partial h_6}{\partial h_5} \frac{\partial h_5}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial w} \\ &= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1 \\ &= 0.25. \end{aligned}$$

自动梯度计算

Pytorch

```
optimizer.zero_grad()
```

清空上一个batch
的参数梯度

```
x, y = batch
```

```
outputs = model(x)
```

```
loss = criterion(outputs, y)
```

计算损失

```
loss = loss.mean()
```

损失的平均

```
loss.backward()
```

反向传播

```
optimizer.step()
```

更新参数

tensorflow

```
with tf.GradientTape() as tape: # 使用with打开
```

```
    z = g(x1, x2) # 具体操作
```

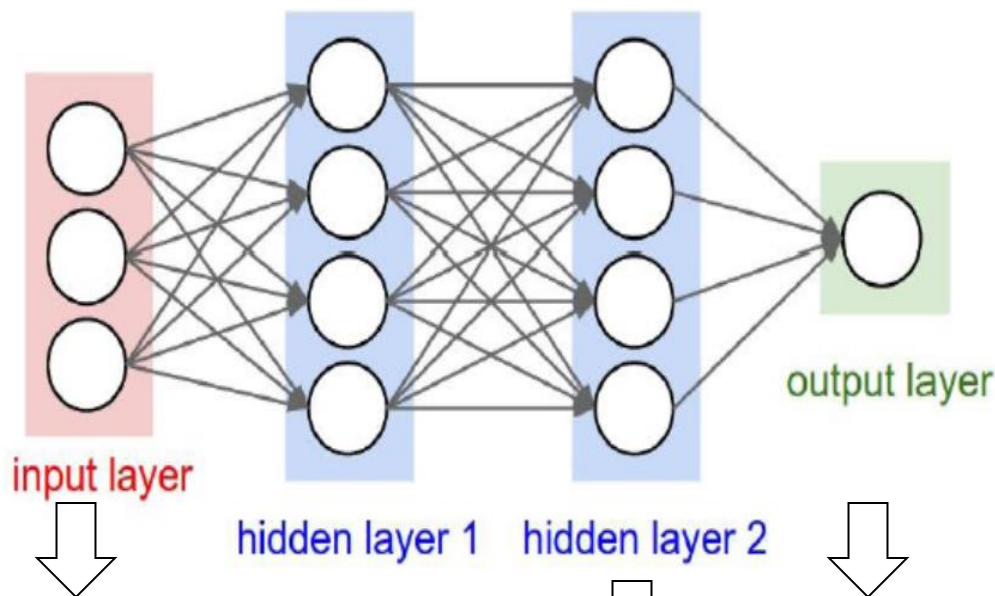
```
dz_x1 = tape.gradient(z, x1) # 求得z对x1的偏导
```

Outline

- 前馈神经模型
- 反向传播算法
- 案例分析

通用流程

- 前馈神经网络可以作为一个编码器(encoder)
- 编码(encoding): 把输入文本序列用一个固定向量来进行表示



文本 → 文本特征(向量)
e.g. 文本每一个词的向量
e.g. 文本长度、文本标点
等特征组成的向量
...

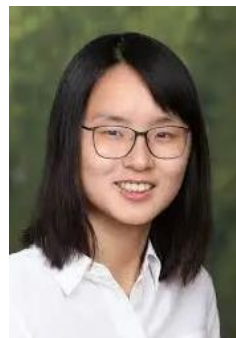
可以看成文本
最终的向量化
表示

一般是sigmoid/ softmax单元
e.g. 二元分类, $\text{sigmoid} < 0.5$ 代
表第一类, 否则第二类;
 $\text{softmax} = [0.6, 0.4]$ 代表第一类
...

案例一：FNN for 依存分析

- 基于文章：A Fast and Accurate Dependency Parser using Neural Networks
- 发表于EMNLP 2014. [PDF](#)
- 第一项将神经网络用于依存句法分析的工作
- 传统的依存句法分析使用手工特征，人工总结特征难以覆盖全面，而且特征向量非常稀疏，计算速度慢
- 用神经网络分类器做基于转移(transition-based)的贪心(greedy)模型来缓解上述问题.

单位：斯坦福NLP组



陈丹琦
普林斯顿NLP组
创始人



Christopher
Manning

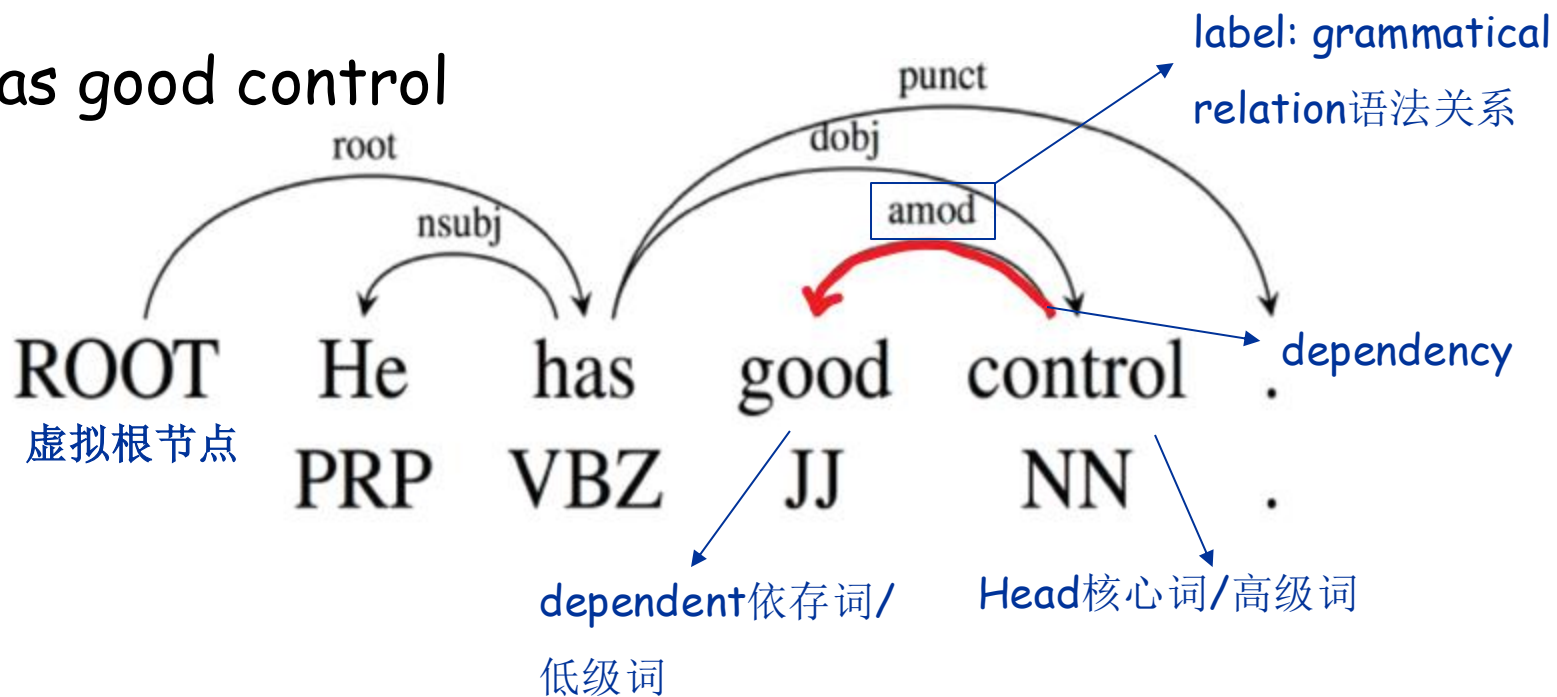
依存分析任务：回顾

Dependency Parsing

词之间的二元非对称依赖关系

形成一棵**连通**，**非循环**，**单根**，**无环**，**无交叉**的树结构

e.g. He has good control

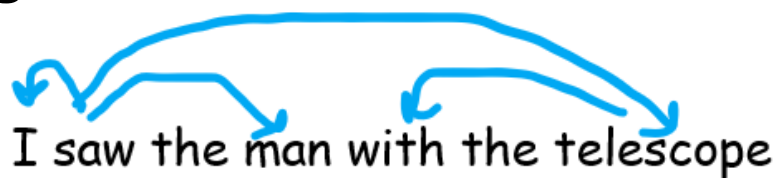


依存分析任务：回顾

Dependency Parsing

帮助减少句法歧义, 更好地理解句子语义, 服务于后续应用

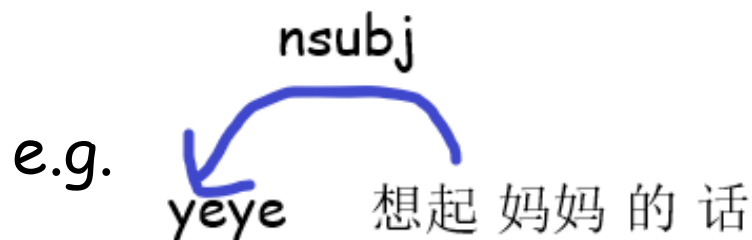
e.g. I saw the man with the telescope



e.g.

英语 ▾ ⇌ 中文(简体) ▾ 翻译 人工翻译 通用 ▾

Mutilated body washes up on Rio beach to be used for Olympics beach volleyball × 里约海滩上的尸体残骸将用于奥运会沙滩排球

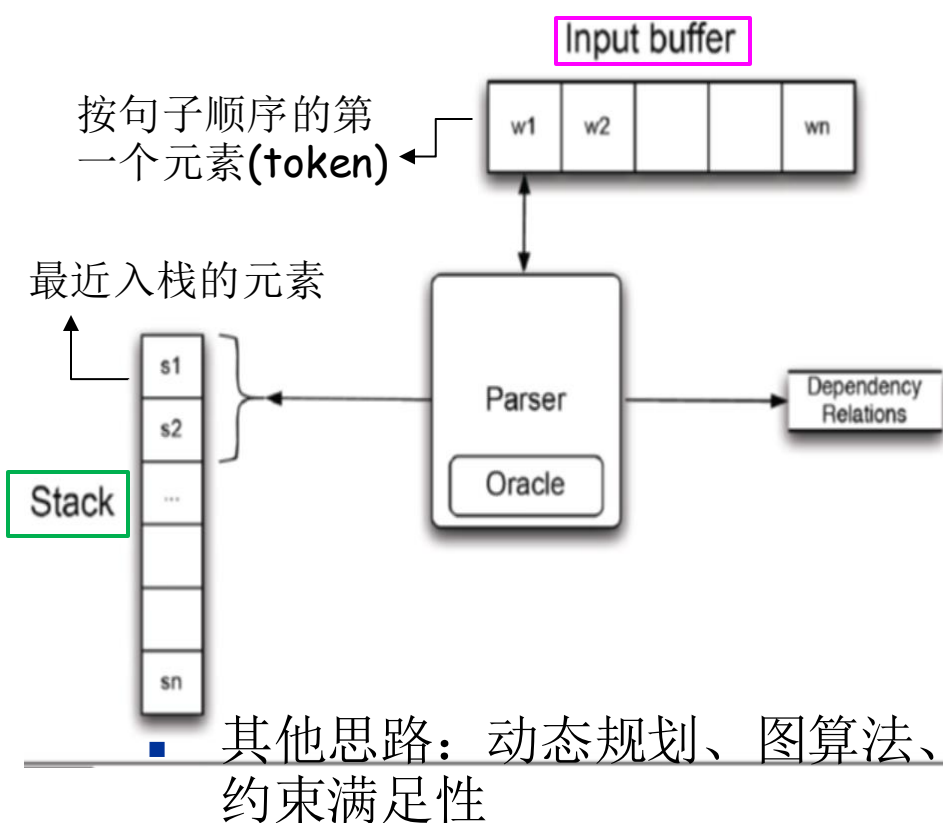


???



经典思路: Transition-based (基于转移)

- 基于移进归约(shift-reduce): 包含三个组成部分: 一个上下文无关文法, 一个堆栈, 以及一个将要被分析的 token 列表。首先将待分析的 token 依次输入到堆栈中。栈顶两个元素去与语法规则的右值比较, 如果匹配成功, 则元素被语法规则的左值替换 (规约)。被用来分析程序语言



- 状态 **Configuration**: 记录不完整的预测结果
 - **Stack** 栈 (先进后出)
 - **Input buffer of words** 缓存队列
 - 依存关系集合 (存放依存边)
- 转移 **Transition**: 控制每一步状态的变化
- 依存分析目标: 找到一个最终的状态, 其中所有涉及关系的单词都会形成依存树

转移操作

- 每一步Transition做什么：根据当前状态（栈**stack**，缓冲区**buffer**，依存关系**dependency**），产生一个新的状态
- 开始状态
 - **Stack**使用根节点**root**初始化
 - **Buffer**使用句子中的词序列初始化
 - 依存关系集合为空
- 结束状态
 - **Stack**有一个根节点
 - **Buffer**为空此时，依存关系集合为最终的依存分析结果
- 此时依存分析也就是：找到一个转移序列，该转移序列实现了从开始状态到理想结束状态的过程



转移操作

- 转移操作：改变状态中的 **stack**, **buffer**, **dependency**。
- **arc-standard transition-based parser** 包含3类动作：

(1) LEFT-ARC:

- 添加一个依存边为 $s1 \rightarrow s2$, $s1$ 是栈顶的词（最后入栈）， $s2$ 是第二个词（要求 **stack** 中元素个数大于等于2）
- 将 $s2$ 从 **stack** 中移除

(2) RIGHT-ARC:

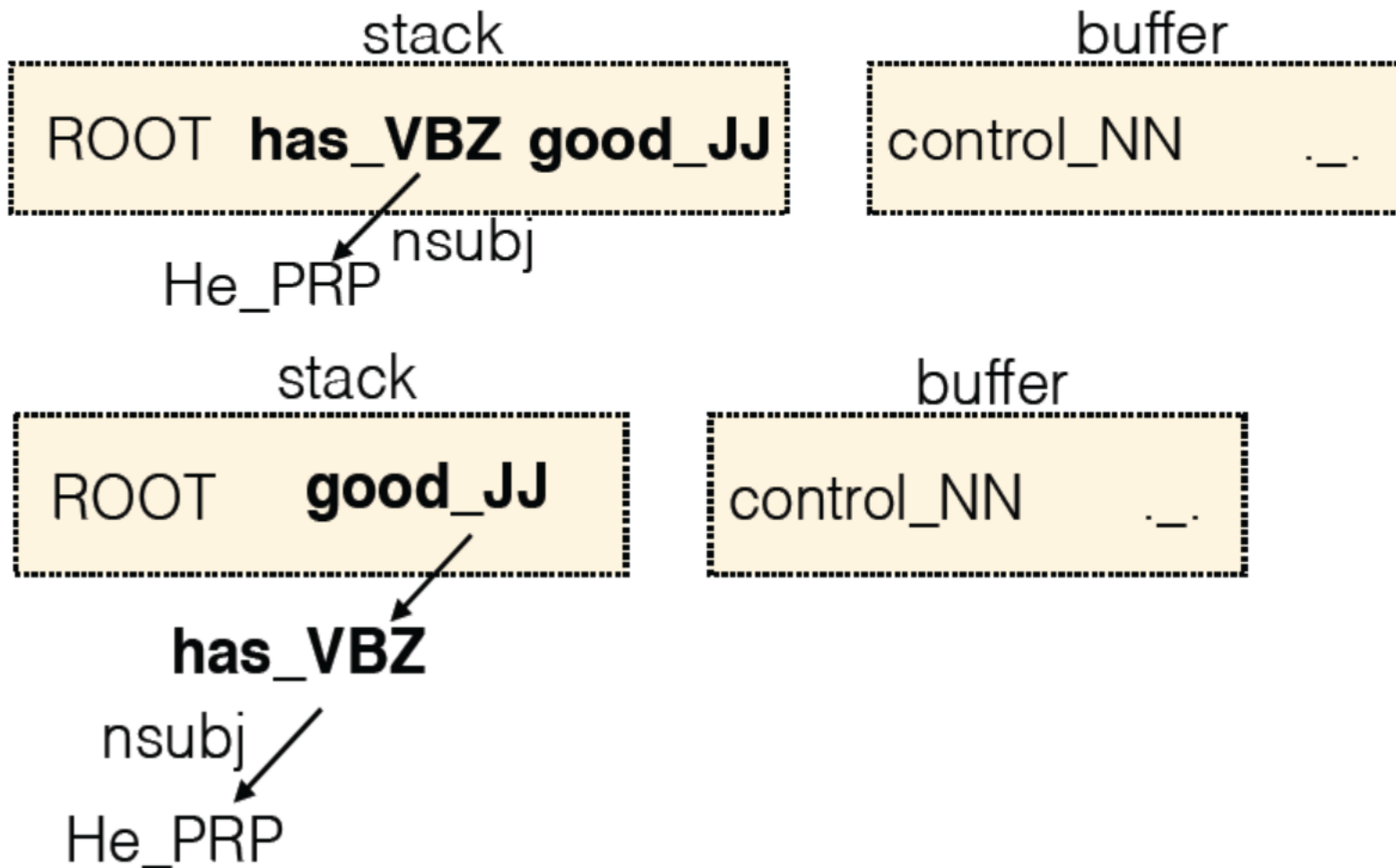
- 添加一个依存边为 $s2 \rightarrow s1$ （要求 **stack** 中元素个数大于等于2）
- 将 $s1$ 从 **stack** 中移除

(3) SHIFT

- 从 **buffer** 中移除第一个词 $b1$ （要求 **buffer** 中元素个数大于等于1）
- $b1$ 压入栈

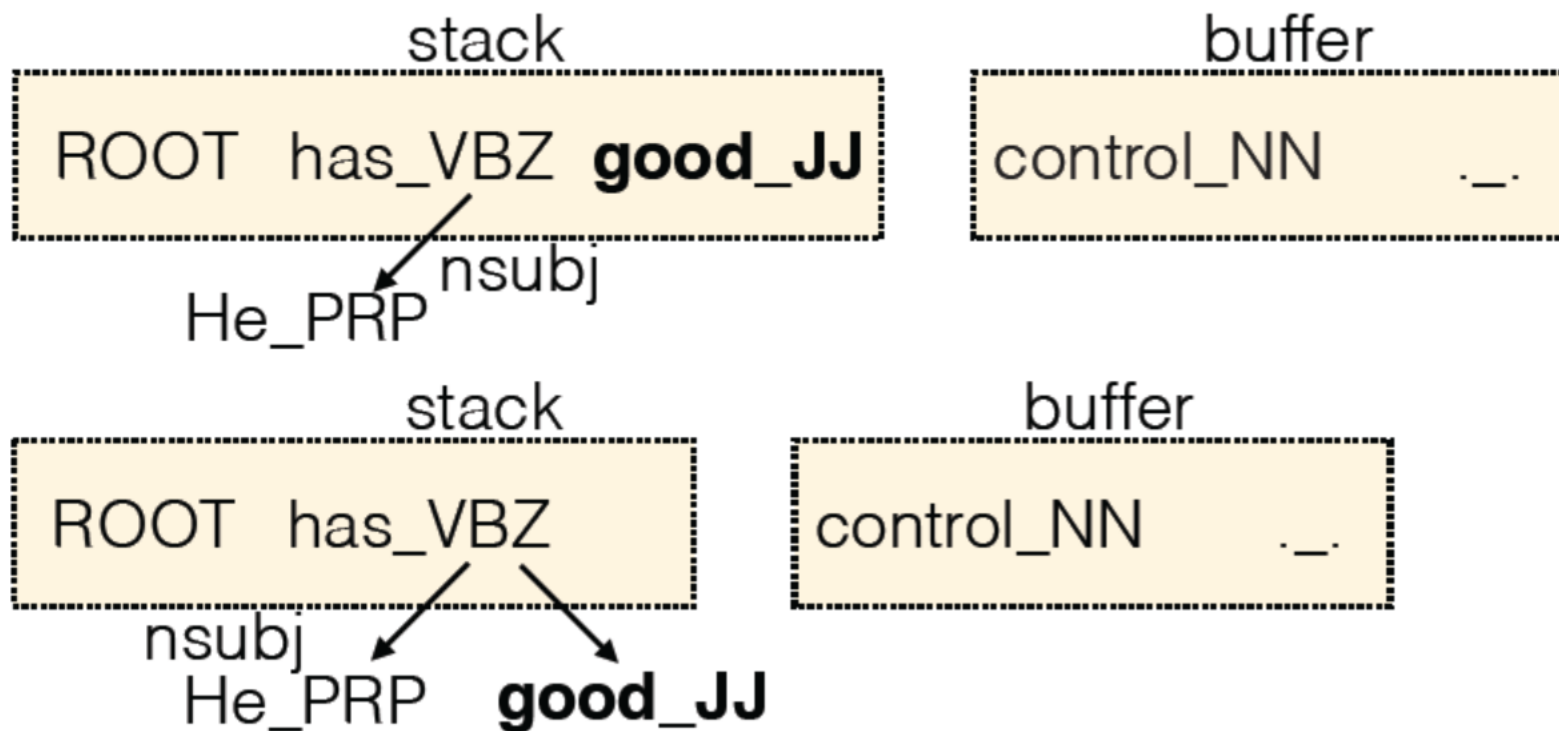
LEFT-ARC

添加依存边 $s1 \rightarrow s2$ ，stack中移除 $s2$



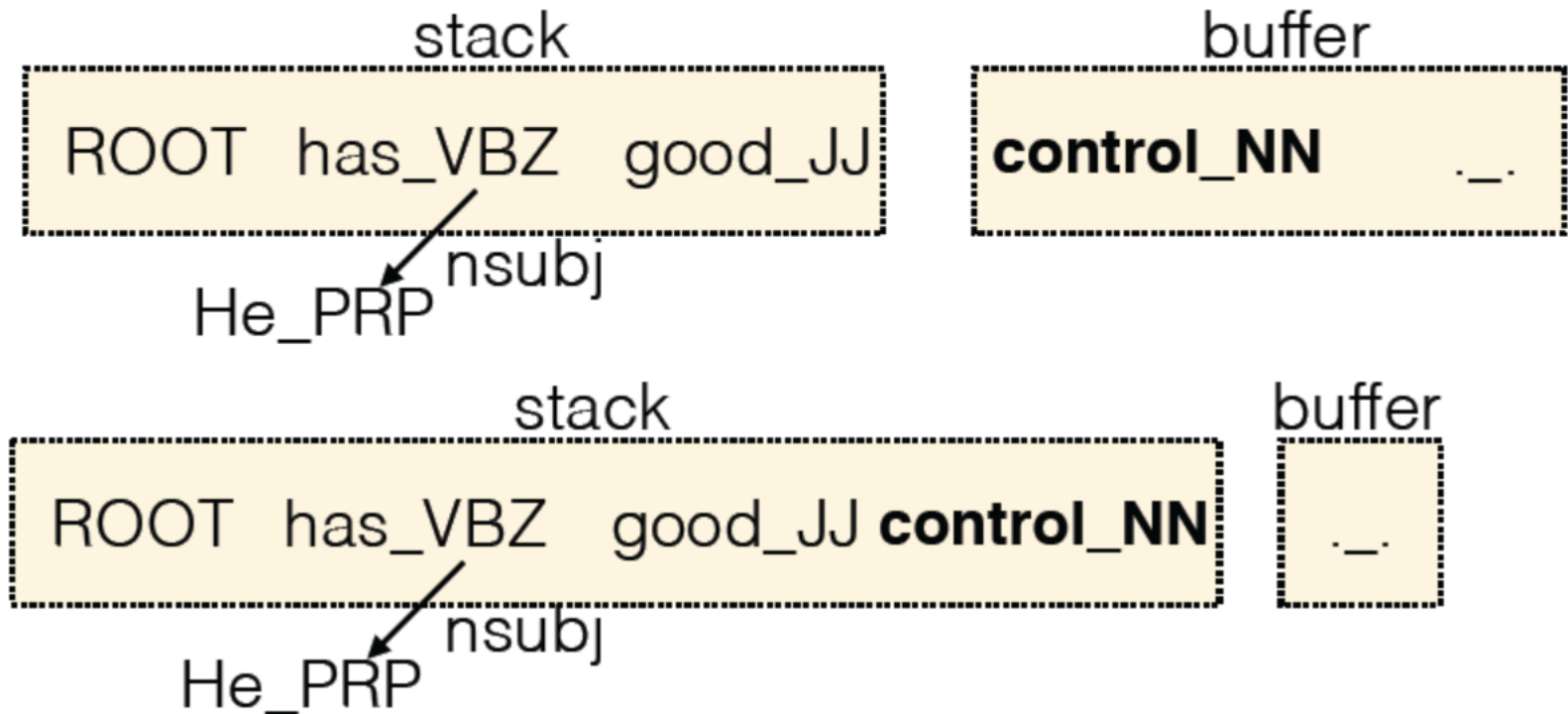
RIGHT-ARC

添加依存边 $s2 \rightarrow s1$, $stack$ 中移除 $s1$

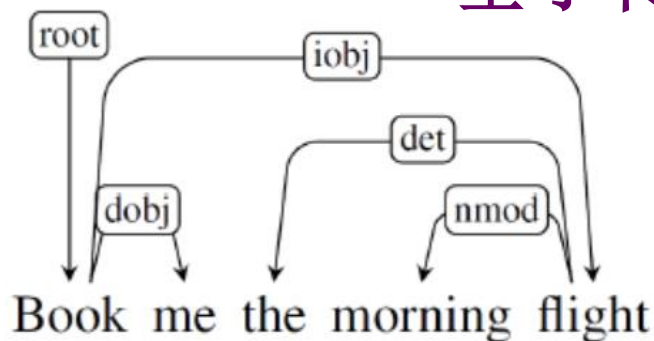


SHIFT

从buffer中移除第一个词**b1**，压入栈



基于转移的依存分析步骤演示



- **LEFT-ARC**: 创建 $s1 \rightarrow s2$; 删除 $s2$
- **RIGHT-ARC**: 创建 $s2 \rightarrow s1$; 删除 $s1$
- **SHIFT**: 删除 $b1$; $b1$ 入栈
- 结束状态: **stack**只有 $root$, **buffer**为空

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book \rightarrow me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning \leftarrow flight)
7	[root, book, the, flight]	[]	LEFTARC	(the \leftarrow flight)
8	[root, book, flight]	[]	RIGHTARC	(book \rightarrow flight)
9	[root, book]	[]	RIGHTARC	(root \rightarrow book)
10	[root]	[]	Done	

基于转移的依存分析：贪心策略

- 每一个步骤贪心地预测下一个要采取的动作，只考虑当前状态下概率最大的动作，完成转移

■ E.g. $C_i =$

Stack	Buffer
<div>ROOT has_VBZ good_JJ</div> <div>↙ nsubj</div> <div>He_PRP</div>	<div>control_NN ...</div>

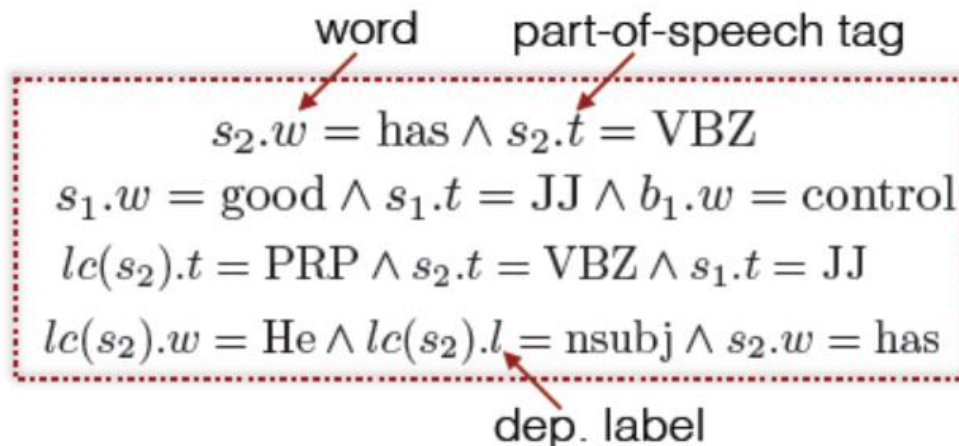
动作是什么?
 $C_{i+1} = ?$

- 两种设置：**难点：如何分类？** 预言机(oracle): 提供操作符
 - (1) unlabeled: 只预测哪一种动作(left-arc, right-arc, shift)。
 - (2) labeled: 预测哪一种动作，以及left-arc或right-arc时两个词之间的依存关系。假设一共有n种依存关系，则进行(2n+1)类分类。

MaltParser(2005): 判别式分类器；s1的词和词性, b1的词和词性...

传统分类特征

Indicator
features

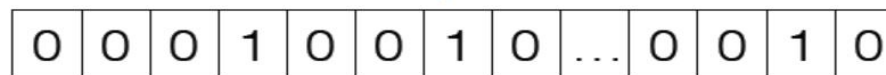


查看是否满足?

传统的分类特征由人工总结。根据状态：词汇、词性、依存关系标
由这些**indicator features**通过**拼接**构成了一个很大的特征向量，该向量的
值是0或1，且0占据非常大的比例，是一个稀疏向量，维度达 10^6-10^7



binary, sparse
dim = $10^6 \sim 10^7$



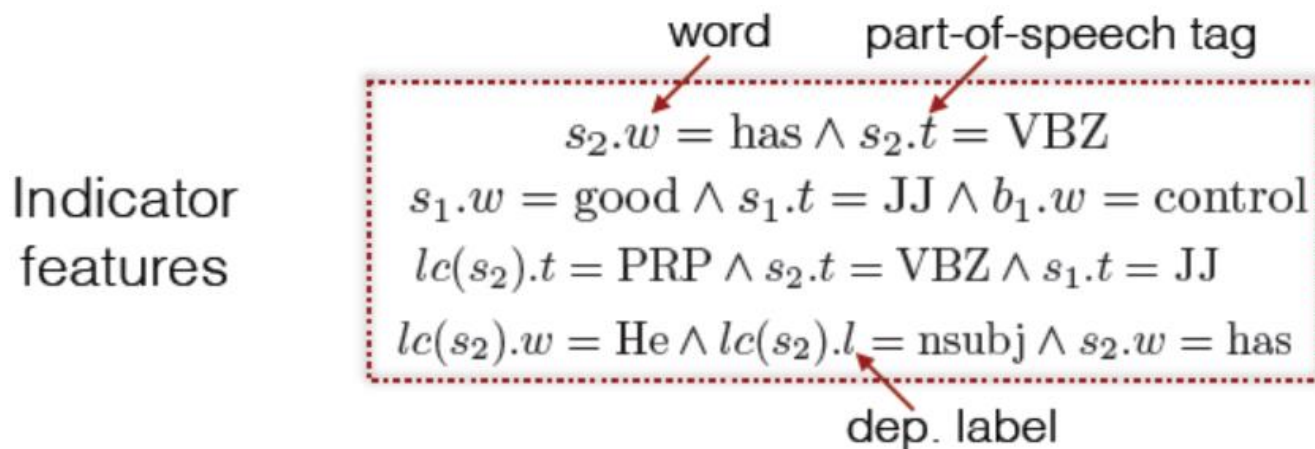
Logistic,
SVM... 类别

传统分类特征

问题1: 向量稀疏。在indicator特征中匹配，本身很稀疏；难以表示向量的相互作用(乘法)

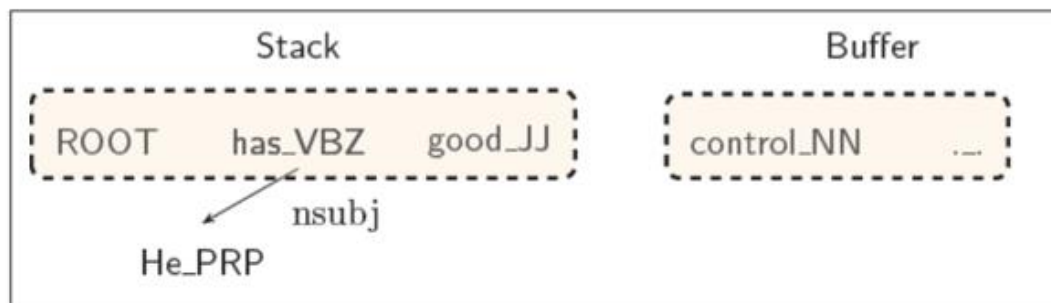
问题2: 特征不完整。难以总结所有特征模板

问题3: 计算代价昂贵。对词语、词性标签或语法关系标签进行拼接来生成特征字符串，并在包含数百万特征的巨大表格中进行查找。**95%**以上的解析时间用于特征计算。



使用神经网络的分类特征

- 3大问题的解决思路：使用神经网络
- 学习稠密（不会出现很多0）、紧凑（维数远小）的特征表示向量



dense
dim = 200



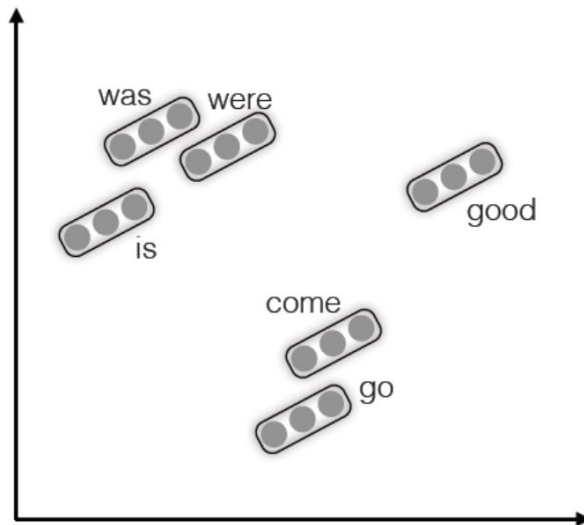
0.1	0.9	-0.2	0.3	...	-0.1	-0.5
-----	-----	------	-----	-----	------	------

- 问题：
 - ✓ 如何对所有可用信息进行编码？
 - ✓ 如何对高阶特征建模？

使用神经网络的分类特征

- 首先考虑到这是一个针对文本数据的处理任务，采用词语的分布式表示：把每一个词语表示为一个 d 维的稠密向量 (即词向量 **Word embeddings**).

- 理想：相似的词，则词向量也相近
- 映射到2维平面：



- 考虑到该任务是一个依存分析任务，词性和依存关系也是紧密相关的信息：使用向量表示词性和依存关系标签。

NNS (plural noun) should be close to NN (singular noun).

num (numerical modifier) should be close to amod (adjective modifier).

神经依存分析模型结构

在转移中每一个步骤都进行一次预测，每一次预测使用这样一个单隐藏层的神经网络：输入层、隐藏层、以**softmax**作为输出单元的
输出层，实现对状态的更新。

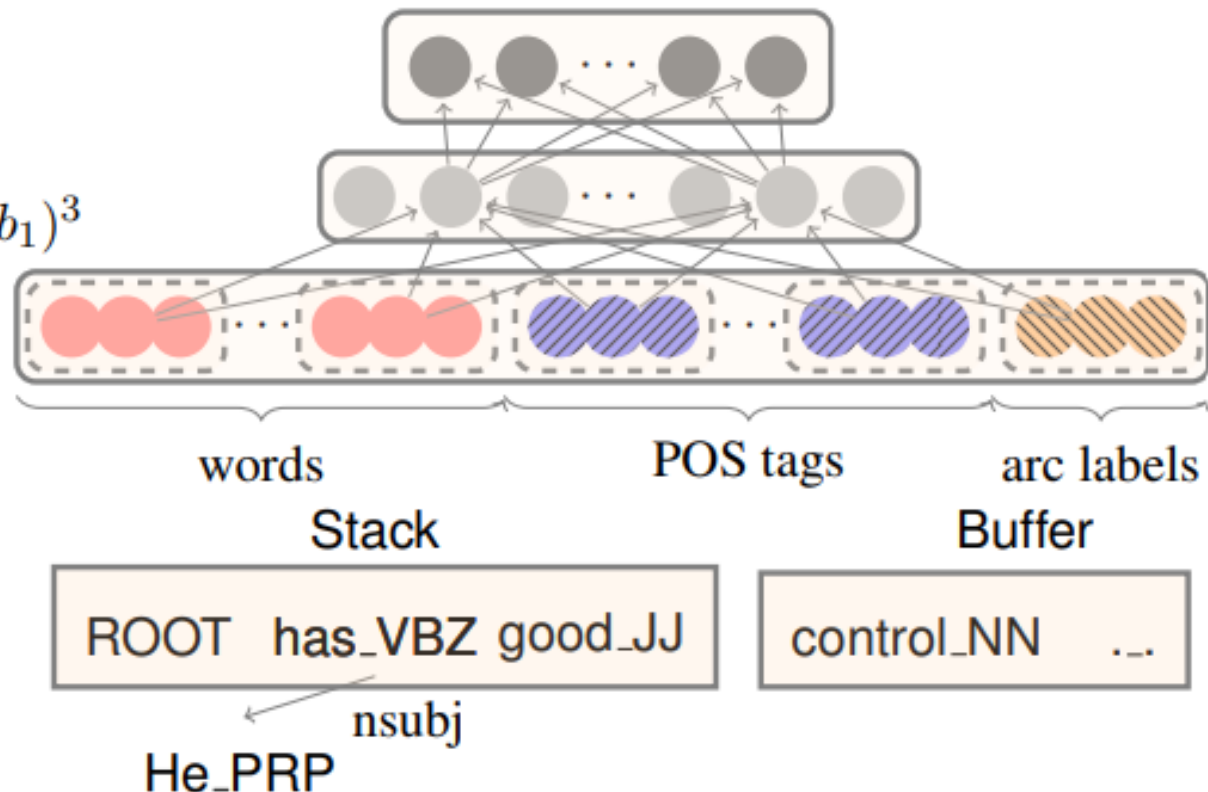
Softmax layer:

$$p = \text{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$



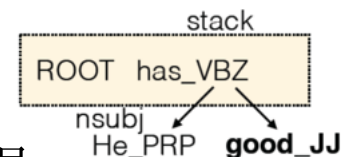
神经网络实际上作为一个分类器组件，完成了文本分类

输入层特征

- 词语特征：取以下词的词向量
- **stack**和**buffer**前3个单词: $s_1, s_2, s_3, b_1, b_2, b_3$ (不够补null_token)
- **stack**前两个单词的左、右孩子中距离最近的两个孩子:
 $lc_1(s_1), rc_1(s_1), lc_2(s_1), rc_2(s_1), lc_1(s_2), rc_1(s_2), lc_2(s_2), rc_2(s_2)$
- **stack**前两个单词距离最近左孩子的最近左孩子, 最近右孩子的最近右孩子: $lc_1(lc_1(s_1)), rc_1(rc_1(s_1)), lc_1(lc_1(s_2)), rc_1(rc_1(s_2))$

- 词性特征:

- 以上18个词的词性标记: Stanford POS tagger获得

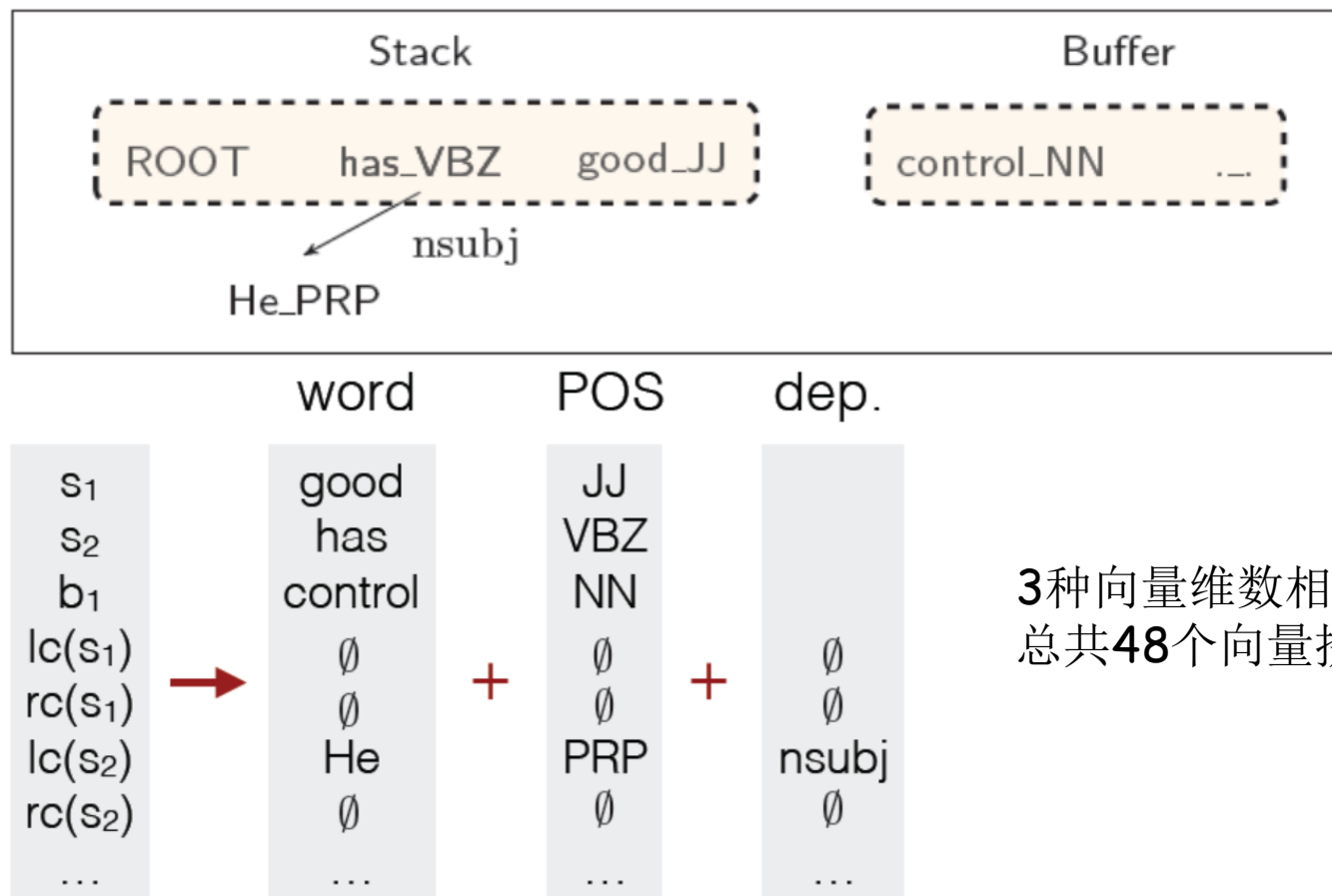


- 依存边特征

- 以上 后12个词的依存边的标签
- 英文工具: CoNLL Syntactic Dependencies, Stanford Basic Dependencies; 中文工具: Penn2Malt
- 比传统的手工特征工作量小

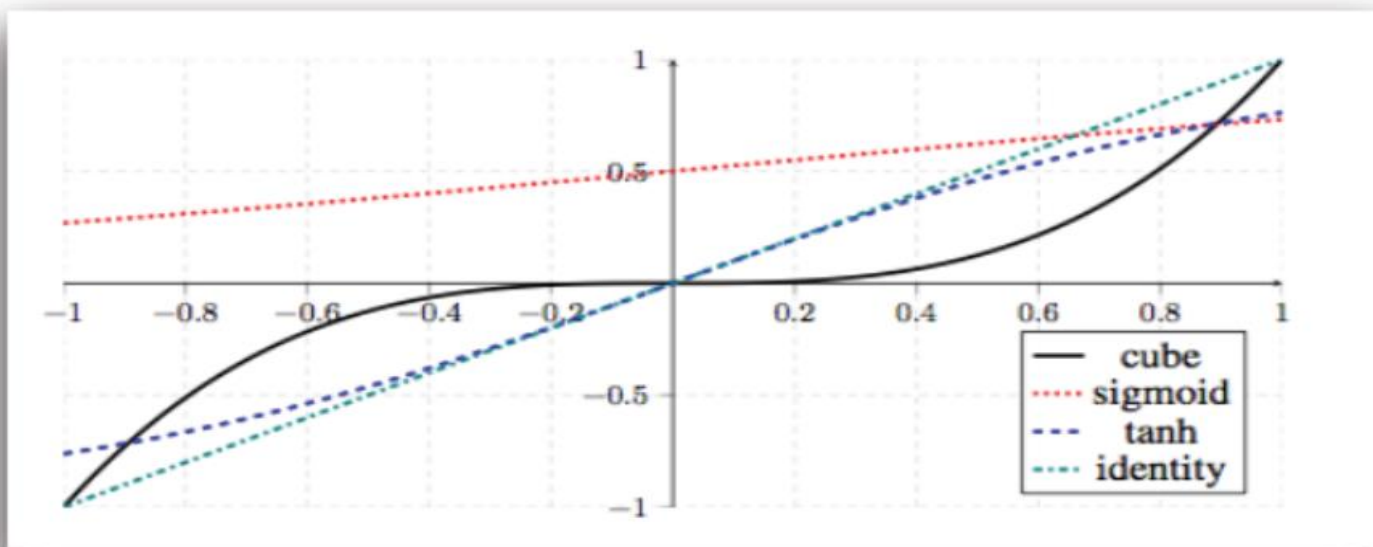
输入层特征

- 根据当前状态确定输入特征



3种向量维数相等
总共**48**个向量拼接

隐藏层激活函数：立方激活函数



$$g(w_1x_1 + \dots + w_mx_m + b) = \sum_{i,j,k} (w_iw_jw_k)x_ix_jx_k + \sum_{i,j} b(w_iw_j)x_ix_j \dots$$

x_i, x_j, x_k 可以来自三类特征中的任意一种，使用立方激活函数可以对它们进行建模，更好地捕捉不同种特征之间的相互作用

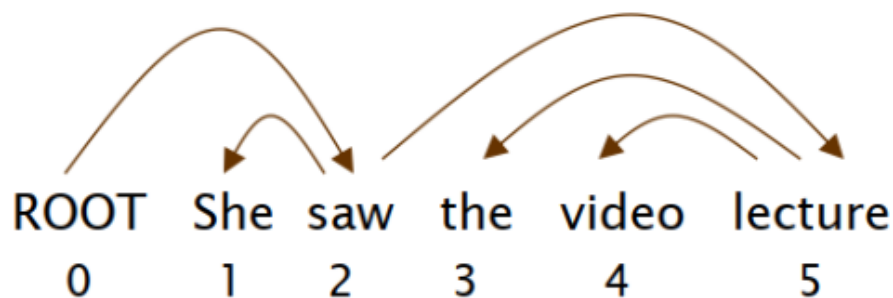
神经模型训练

- 训练样本选择：本项工作从训练文本和真实(gold)语法解析树中生成训练样本 $\{(c, t)\}$, 其中 c 是 configuration, t 是 transition. 并且遵循一种规则, 要让 stack 尽量短.
- 训练目标/损失函数: 交叉熵 **Cross entropy loss**
- 所有向量均使用反向传播计算梯度
- 优化器: 小批量(mini-batch)的 AdaGrad
- 初始化:
 - 词语向量采用预训练的 w2v 向量. 50 维
 - 其他向量(词性、依存关系标签) 随机初始化 (首次使用)

依存分析的评估

UAS: unlabeled 无标记依存准确率

LAS: labeled 有标记依存准确率



$$\text{Acc} = \frac{\# \text{ correct deps}}{\# \text{ of deps}}$$

$$\text{UAS} = 4 / 5 = 80\%$$

$$\text{LAS} = 2 / 5 = 40\%$$

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

其他:

- (1) 依存准确率: 中心词预测正确的非根节点词语个数 / 总非根节点词数
- (2) 根准确率: 正确根节点的个数 / 句子个数
- (3) 完全匹配率: 无标记依存结构完全正确的句子 / 句子总数

传统特征与稠密特征的比较

✓ 问题 #1: 向量稀疏性

indicator特征是稀疏向量，而本工作使用的是稠密的分布式向量，能更好地表达词语的语义相似度

✓ 问题 #2: 特征完整性

indicator特征是手工整理的，特征之间可能需要进行组合，而人工枚举特征非常可能不完整。神经网络方法使用一个立方激活函数，可以自动地对不同类别、类别内的特征进行组合。

✓ 问题 #3: 计算代价

人工整理特征费时费力，必须对词语、词性标签或语法关系标签进行拼接来生成特征字符串，并在包含数百万特征的巨大表格中查找它们。神经网络方法，只需要做一些矩阵操作。

实验1要求

✓ 基本要求

完成依存分析模型的构建，包括整体架构、每一层、特别是**48**个特征的获取

✓ 进阶要求

完成训练数据的构造、损失函数、模型训练过程、测试过程

可以参考<http://fancyerii.github.io/books/nndepparser/>

后续工作

- A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing
- ACL 2015 (NLP创新的一个方式就是对已有工作的不足进行改进)
- 为什么需要改进? 贪心策略的缺点: 每步选择得分最高的操作, 得到一个局部最优, 不能保证全局最优; 无法修正, 错误传递。



提出结构化的神经概率依存分析框架, 最大化**整个动作序列**的概率



修改 解码算法 和 训练目标

还有其他优化思路吗?

解码算法

- 编码**encoding**: 把输入文本序列用一个固定向量来进行表示
- 解码**decoding**: 把固定向量转化为输出序列。本任务中解码的目标是给定输入 x ，找到全局上分数最高的动作序列
- 贪心策略修改为**beam search**束搜索策略，每一步骤保留 k 个分数最高的预测，取得的总体分数能够更好，效果接近于**exact inference**精确推断

句子级概率计算

- 避免局部最优：直接对整个动作序列的概率分布进行建模。
- 给定一个句子 x 和神经网络参数，第 i 个动作序列 y_i 的概率由 **softmax**函数根据所有动作序列的分数给出：

序列 y_i 的概率：
$$p(y_i | x, \theta) = \frac{e^{f(x, \theta)_i}}{\sum_{y_j \in \text{GEN}(x)} e^{f(x, \theta)_j}}$$
 $\text{GEN}(x)$: 所有可能的序列

序列 y_i 的分数 f ：包含的所有动作 a_k 的分数之和

$$f(x, \theta)_i = \sum_{a_k \in y_i} o(x, y_i, k, a_k)$$

a_k ：第 k 步的转移动作
 o ：神经网络输出

模型结构和前一个工作相同

训练目标/损失

- 句子级的负对数似然损失:

$$\begin{aligned} L(\theta) &= - \sum_{(x_i, y_i) \in (X, Y)} \log p(y_i \mid x_i, \theta) \\ &= - \sum_{(x_i, y_i) \in (X, Y)} \log \frac{e^{f(x_i, \theta)_i}}{Z(x_i, \theta)} \\ &= \sum_{(x_i, y_i) \in (X, Y)} \log Z(x_i, \theta) - f(x_i, \theta)_i \end{aligned}$$

$$Z(x, \theta) = \sum_{y_j \in \text{GEN}(x)} e^{f(x, \theta)_j}$$

Z 包含了一个样本所有可能的
预测序列
减少搜索范围: **beam search**

$Z(x, \theta)$ 的计算

- **对比学习**：给观测到的数据分配一个较大的概率值，给噪声数据分配一个较小的概率值。
- 本任务中，给**gold**动作序列更大的概率值，给束(**beam**)中的错误序列更小的概率值。本任务中的对比学习属于监督对比学习。
- 这样，我们只需要对比**gold**动作序列和**beam**中的错误序列(噪声序列)，而不需要对比全部的序列。**beam**选中的序列都是分数比较高的，所以我们希望模型可以区分正确答案，和得分高的错误答案。

新的训练目标

- 采用了对比学习的训练目标:

$$\begin{aligned} L'(\theta) &= - \sum_{(x_i, y_i) \in (X, Y)} \log p'(y_i | x_i, \theta) \\ &= - \sum_{(x_i, y_i) \in (X, Y)} \log \frac{e^{f(x_i, \theta)_i}}{Z'(x_i, \theta)} \\ &= \sum_{(x_i, y_i) \in (X, Y)} \log Z'(x_i, \theta) - f(x_i, \theta)_i \end{aligned}$$

$$Z'(x, \theta) = \sum_{y_j \in \text{BEAM}(x)} e^{f(x, \theta)_j}$$

Z'选择的范围是**BEAM** 一个范围有限的束

Input: training examples (X, Y)

Output: θ

$\theta \leftarrow$ pretrained embedding

for $i \leftarrow 1$ **to** N **do**

$\mathbf{x}, \mathbf{y} = \text{RANDOMSAMPLE}(\mathbf{X}, \mathbf{Y})$

$\delta = 0$

foreach $x_j, y_j \in \mathbf{x}, \mathbf{y}$ **do**

$\text{beam} = \phi$

$\text{goldState} = \text{null}$

$\text{terminate} = \text{false}$

$\text{beamGold} = \text{true}$

while beamGold **and** **not** terminate

do

$\text{beam} = \text{DECODE}(\text{beam}, x_j, y_j)$

$\text{goldState} =$

$\text{GOLDMOVE}(\text{goldState}, x_j, y_j)$

if not $\text{ISGOLD}(\text{beam})$ **then**

$\text{beamGold} = \text{false}$

if $\text{ITEMSCOMPLETE}(\text{beam})$ **then**

$\text{terminate} = \text{true};$

$\delta = \delta + \text{UPDATE}(\text{goldState}, \text{beam})$

$\theta = \theta + \delta$

搜索和学习集成在一个
统一的框架中

随机采样一些训练集数据

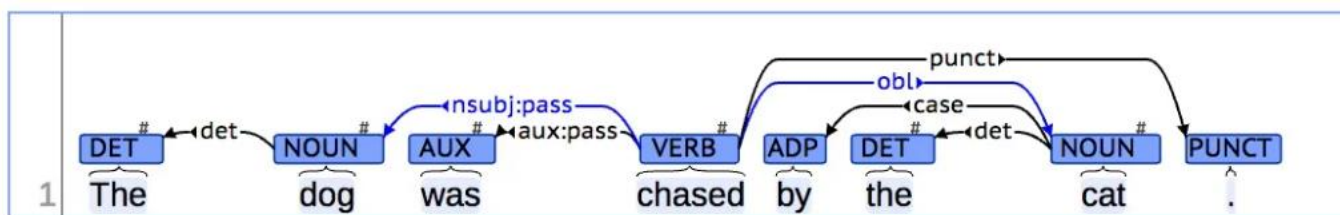
每个样本都有一个gold序列

循环里decode得到k个预测序列。如果beam里的序列有gold，就把其他预测当成负样本，内部循环终止，进行参数更新；否则重新解码，直到满足内部循环的终止条件

Mini-batched
AdaGrad

依存分析数据资源

- 句法分析语料库也称为句法树库，包含大规模句子以及其对应句法树的集合。
- Universal Dependencies treebanks
- 一个在**100**多种不同人类语言中对语法（词性、形态特征和句法依赖性）进行一致注释的框架



依存分析数据资源

- 句法分析语料库也称为句法树库，包含大规模句子以及其对应句法树的集合。

语料库名称	单词数量	语法类型	语言
英语宾州树库 (PTB)	117 万	成分语法	英文
通用依存树库 (UD V2.0 CoNLL 2017)	281 万	依存语法	多语言
通用依存树库 (UD V2.2 CoNLL 2018)	1714 万	依存语法	多语言
组合范畴语法树库 (CCGBank)	116 万	组合范畴语法	英文
中文宾州树库 6.0 (CTB 6.0)	78 万	成分语法	中文
中文宾州树库 7.0 (CTB 7.0)	120 万	成分语法	中文
中文宾州树库 8.0 (CTB 8.0)	162 万	成分语法	中文
中文宾州树库 9.0 (CTB 9.0)	208 万	成分语法	中文
中文语义依存树库 (SDP)	52 万	语义依存	中文

依存分析数据资源

宾州树库的数据组织结构（中英文相同）

```
( ( IP ( NP-SBJ ( DNP ( NP-PN ( NR 北海市 ))
                        ( DEG 的 ))
                    ( NP ( NN 崛起 )))
  ( PU , )
  ( VP ( VC 是 )
    ( NP-PRD ( CP-APP ( IP ( IP-SBJ ( LCP-TMP ( NP ( NT 近年 ))
                                                ( LC 来 ))
                                              ( NP-PN-SBJ ( NR 广西 )
                                                            ( NN 壮族 )
                                                            ( NN 自治区 ))
                                              ( VP ( PP-DIR ( P 对 )
                                                            ( NP ( NN 外 )))
                                              ( VP ( VV 开放 ))))
                                              ( VP ( VV 取得 )
                                                ( NP-OBJ ( ADJP ( JJ 卓著 )
                                                            ( NP ( NN 成就 )))))
                                              ( DEC 的 ))
                                              ( ADJP ( JJ 重要 ))
                                              ( NP ( NN 标志 )
                                                ( NN 之一 ))))
    ( PU 。 )) )
```

依存分析工具

■ Stanford Parser

网页版: <https://corenlp.run>

Python库: `stanfordcorenlp`

我在春天的

我在春天的南京师范大学等待你

parts-of-spe

— Annotations —

parts-of-speech x named entities x dependency parse x

Submit

Part-of-Speech:

1 我 在 春天 的

1 我 在 春天 的 南京 师范 大学 等待 你

Named Entity Recognition:

1 我 在 春天 的

1 我 在 春天 的 南京 师范 大学 等待 你

Basic Dependencies:

1 我 在 春

1 我 在 春天 的 南京 师范 大学 等待 你

有错误!

1 我 在 春天 的 南京 师范 大学 等 你

相关测评任务

- SemEval
- 子任务-中文语义依存评测

数据格式: conll

示例:

dependent

head

1	城建	城建	NN	NN		2	Exp			
2	成为	成为	VV	VV		0	Root			
3	外商	外商	NN	NN		4	Agt			
4	投资	投资	VV	VV		7	dDesc			
5	青海	青海	NR	NR		4	Datv			
6	新 新	JJ JJ			7	Desc				
7	热点	热点	NN	NN		2	Clas			

结构: 中文语义依存树库。从语义角度构建依存关系, 定义了 **45** 个标签用来描述论元 (**Argument**) 之间的语义关系, **19** 个标签用来描述谓词 (**Predicate**) 之间的关系, 以及 **17** 个标签用来提供谓词描述。

构造: 两个词依存的粒度? 句子粒度?

案例二：FNN for 文本分类

■ Bag of Tricks for Efficient Text Classification

■ 2016. PDF

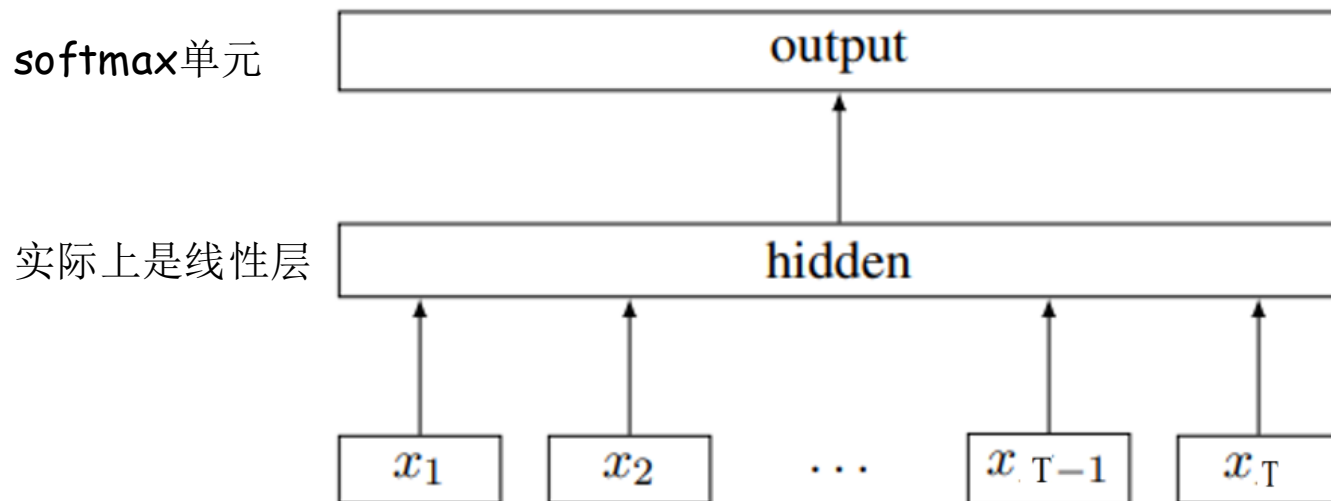
单位：Facebook, 现在的Meta

■ 命名 fastText



Meta

fast: 在使用标准多核CPU的情况下10分钟内处理超过10亿个词汇



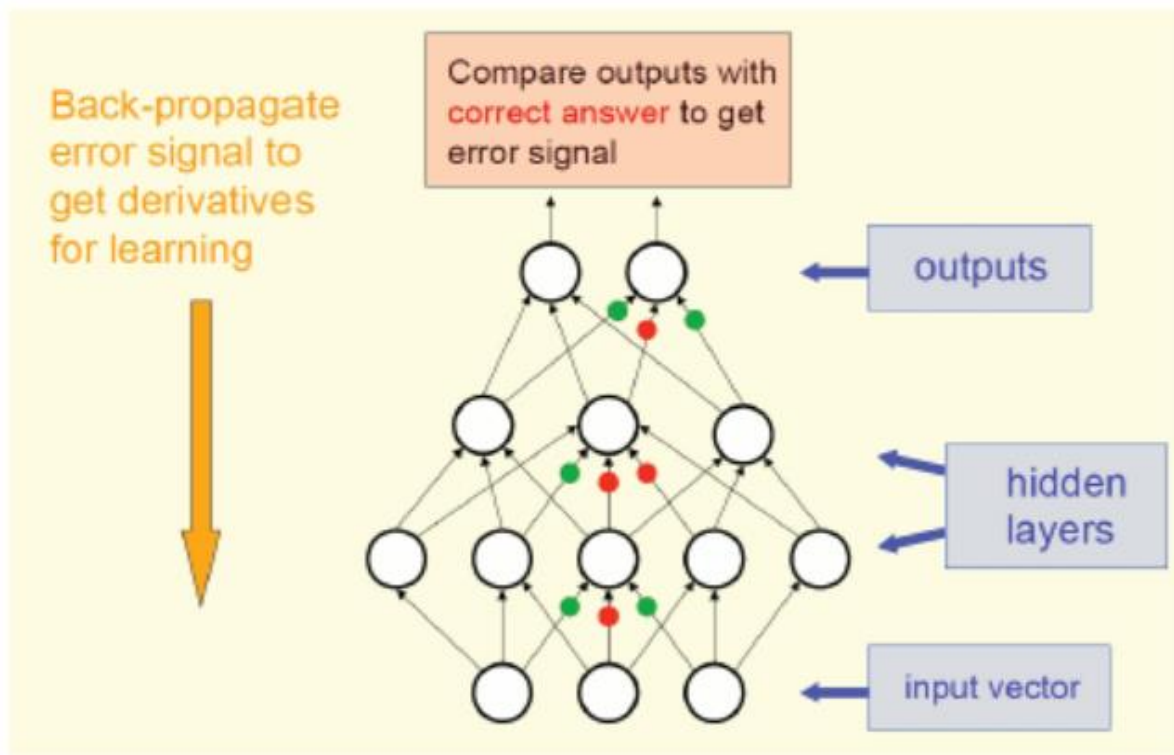
输入层：N元语法特征。

实际上是一个句子所有词的词向量累加
不考虑词序：词袋(bag of words, bow)

$$\text{损失: } -\frac{1}{N} \sum_{i=1}^N y_i \log(f(BAx_i))$$

更好的特征？ 更好的模型设计？

FNN复习



- (1) 前馈计算每一层的净输入和激活值，直到最后一层；
- (2) 反向传播计算每一层的误差项 $\delta^{(l)}$ ，计算每一层参数的偏导数（链式法则+动态规划）；
- (3) 根据优化器算法更新参数