第七章：Neural Networks and Neural Language Models

神经网络：基本计算工具，小型计算单元

前馈神经网络：计算从一层单元迭代到下一层

现代神经网络的使用通常被叫做深度学习，因为层数较多（deep）

Units

神经网络由无数计算单元组成

输入x，权重w，偏置b，输出加权和：

$$z = b + \sum_i w_i x_i$$

向量（a list or array of numbers）简化表示：

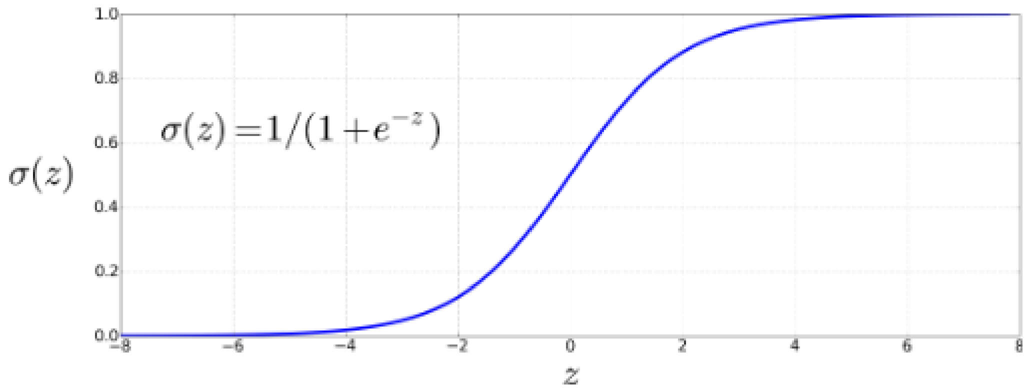
$$z = \mathbf{w} \cdot \mathbf{x} + b$$

神经网络单元避免使用x的线性函数作为输出，非线性拟合：（a表示激活值activation value）

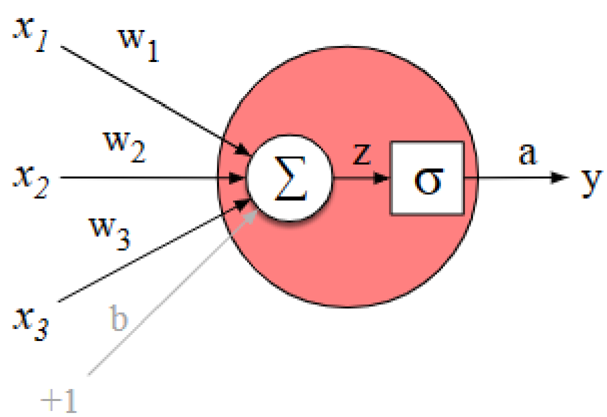
$$y = a = f(z)$$

常见非线性函数：sigmoid、tanh、reactified linear unit（ReLU）

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



计算图示：



sigmoid很少作为激活函数，tanh更好

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

更常用的：

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

各激活函数特点：

- tanh平滑可调，能够将异常值映射到平均值上
- relu：接近线性
- tanh和sigmoid：对于特别大的值，结果为1，发生饱和，求导会很接近0->梯度消失

The XOR problem

多层网络的必要性：单个unit无法计算一些输入很简单的函数。感知器是简单的unit，能够解决AND和OR，但是对于XOR无法用单个unit表示

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

感知器perceptron：只有一个binary输出的unit，同时没有非线性的激活函数；公式如下：

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

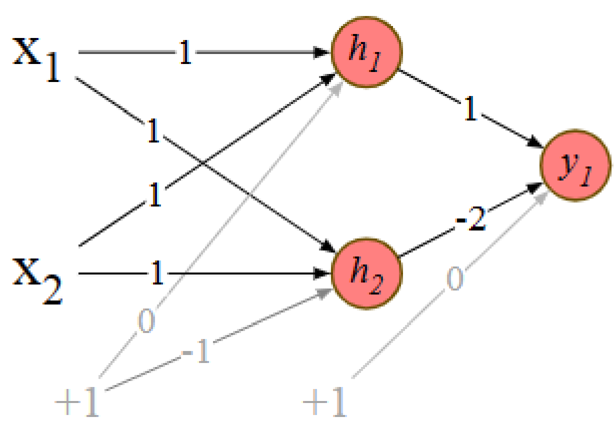
AND：x1+x2-1，OR：x1+x2（小于等于0都输出false）

决策边界：**decision boundary**，对于二维是line，对于三维是超平面

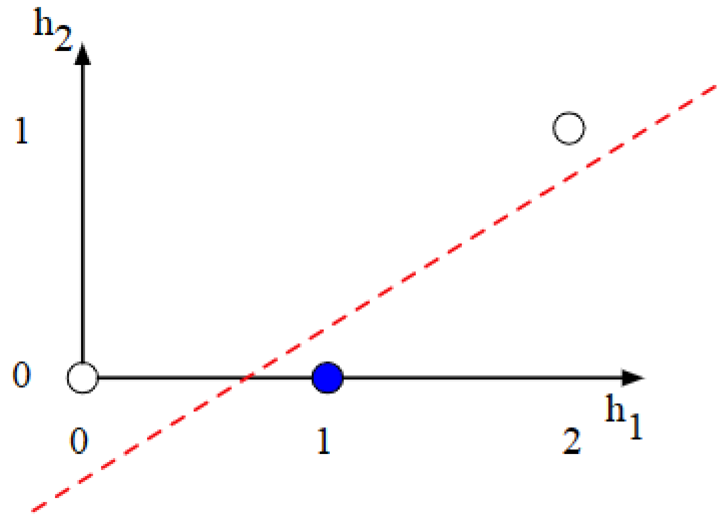
XOR并不是一个线性可分（linearly separable）的函数

The solution: neural networks

使用分层网络解决XOR，两层神经网络：（隐藏层h、输出层y）



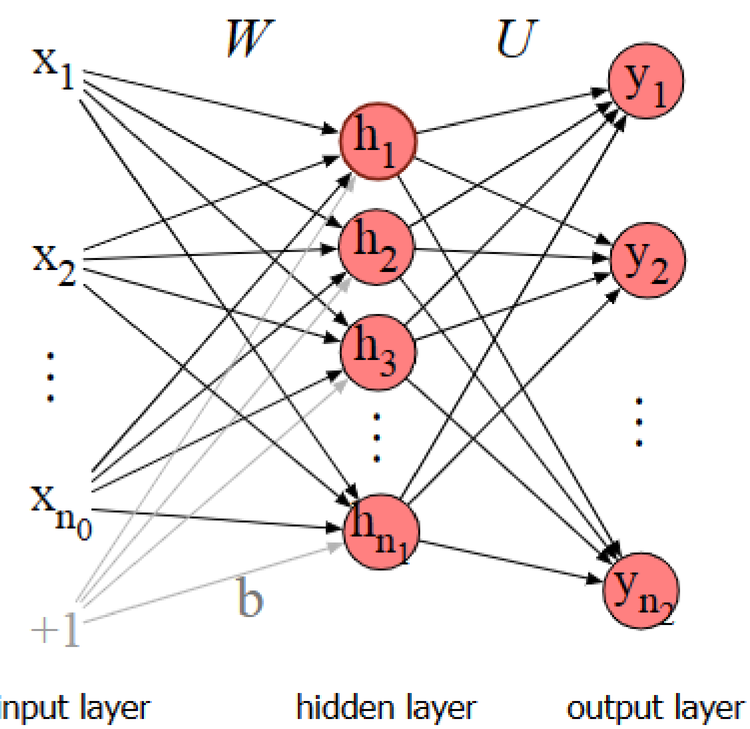
两层网络如何解决XOR，对于输入x的4种情况，在h中表示，离散点是一个线性可分的



b) The new (linearly separable) h space

Feedforward Neural Networks

前馈神经网络：多层网络，units无环连接，outputs不回到lower layer。（成环：RNN）



标准结构中，全连接full-connected

对每一层，向量合并成矩阵表示：

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

对于output层，其权重矩阵为U：

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

softmax(z)对z进行归一化，看做是一种概率表示用于分类任务：

$$\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{d}\exp(\mathbf{z}_j)} \quad 1 \le i \le d$$

神经网络像多项式逻辑回归，但有所不同：

- 有很多层
- 中间层有需要可能的激活函数，不止有sigmoid
- 网络的前几层不是通过特征模板形成特征，而是诱导特征表示其本身

$$\begin{aligned}
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x}+\mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\mathbf{y} &= \text{softmax}(\mathbf{z})
\end{aligned} \tag{7.12}$$

And just to remember the shapes of all our variables, $\mathbf{x} \in \mathbb{R}^{n_0}$, $\mathbf{h} \in \mathbb{R}^{n_1}$, $\mathbf{b} \in \mathbb{R}^{n_1}$, $\mathbf{W} \in \mathbb{R}^{n_1 \times n_0}$, $\mathbf{U} \in \mathbb{R}^{n_2 \times n_1}$, and the output vector $\mathbf{y} \in \mathbb{R}^{n_2}$. We'll call this network a 2-

神经网络层数：input作为第0层，统计层数时一般值算hidden layer和output layer

More details on feedforward networks

神经网络的符号表示

- 上标[i]表示第i层
- $a^{[i]}$：i层输出；y：最终输出
- g(·)：激活函数

简化表示多层网络：

$$\begin{aligned}
&\textbf{for } i \textbf{ in } 1,\dots,n \\
&\quad \mathbf{z}^{[i]} = \mathbf{W}^{[i]}\,\mathbf{a}^{[i-1]} + \mathbf{b}^{[i]} \\
&\quad \mathbf{a}^{[i]} = g^{[i]}(\mathbf{z}^{[i]}) \\
&\hat{\mathbf{y}} = \mathbf{a}^{[n]}
\end{aligned}$$

非线性激活函数的必要性

多层线性网络等价于单层线性网络：

$$\begin{aligned}
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{z}^{[1]}+\mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}(\mathbf{W}^{[1]}\mathbf{x}+\mathbf{b}^{[1]})+\mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathbf{x}+\mathbf{W}^{[2]}\mathbf{b}^{[1]}+\mathbf{b}^{[2]} \\
&= \mathbf{W}'\mathbf{x}+\mathbf{b}'
\end{aligned}$$

替换bias单元

将bias加入到W和x中去，$x_0=1$，W_{j0}替换第j层的biasb_j，简化表示：

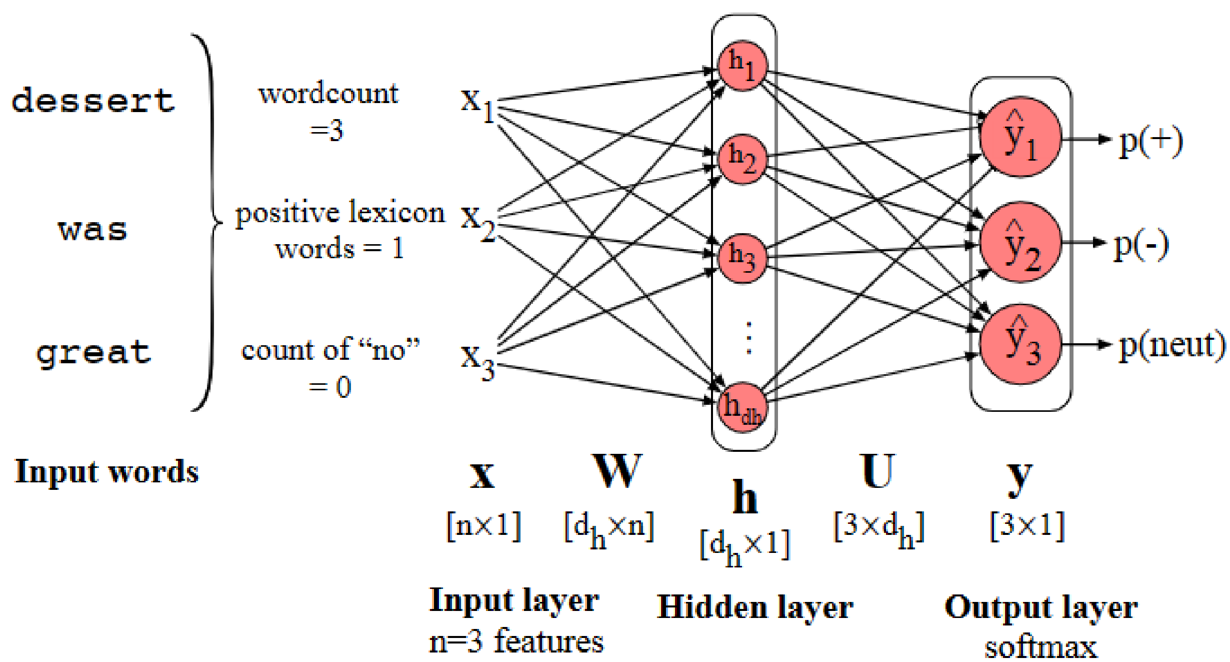
$$\mathbf{h}_j=\sigma\left(\sum_{i=1}^{n_0}\mathbf{W}_{ji}\mathbf{x}_i+\mathbf{b}_j\right),$$

$$\mathbf{h}_j=\sigma\left(\sum_{i=0}^{n_0}\mathbf{W}_{ji}\mathbf{x}_i\right),$$

Feedforward networks for NLP: Classification

本节内容：应用于情感分析分类任务

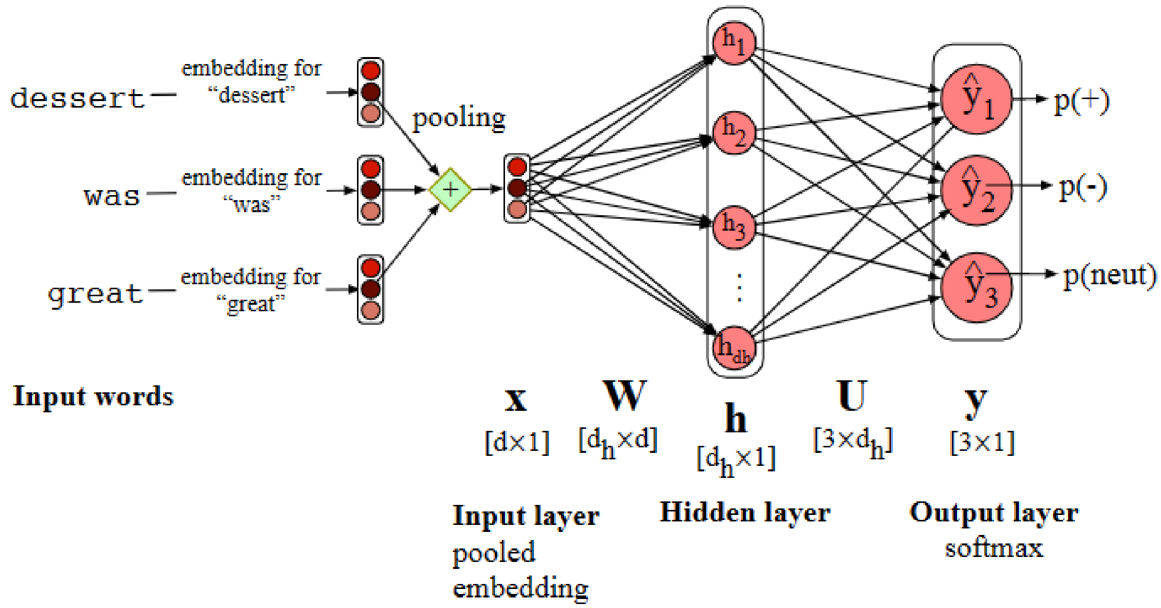
1. 手动构造特征作为输入的结构图：



2. 将words变为embeddings，学习特征

- 池化函数pooling function：

$$\mathbf{x}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{e}(w_i)$$



- 其他方式，如：element-wise max：$n * k -> n * max(k)$

一次性分类具有m个样本的测试集：（拼接为矩阵）

$$\begin{aligned}\mathbf{H} &= \sigma(\mathbf{X}\mathbf{W}^{\mathsf{T}} + \mathbf{b}) \\ \mathbf{Z} &= \mathbf{H}\mathbf{U}^{\mathsf{T}} \\ \hat{\mathbf{Y}} &= \mathrm{softmax}(\mathbf{Z})\end{aligned}$$

预训练**pretraining**：依赖其他已经学习好的模型对input words编码，作为现有模型的输入表征

Feedforward Neural Language Modeling

本节内容：应用于upcoming words prediction

神经网络优势（相比于n-gram models）

- 更长的histories
- 更好地概括相似单词的上下文

- 更准确的单词预测

劣势：复杂、慢、耗电、难以解释

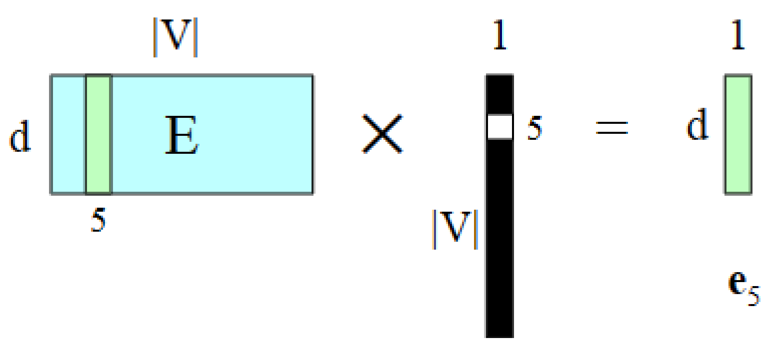
$$P(w_t|w_1,\ldots,w_{t-1}) \approx P(w_t|w_{t-N+1},\ldots,w_{t-1})$$

Forward inference in the neural language model

正向推理：forward inference or decoding，给定输入，forward产生probability distribution

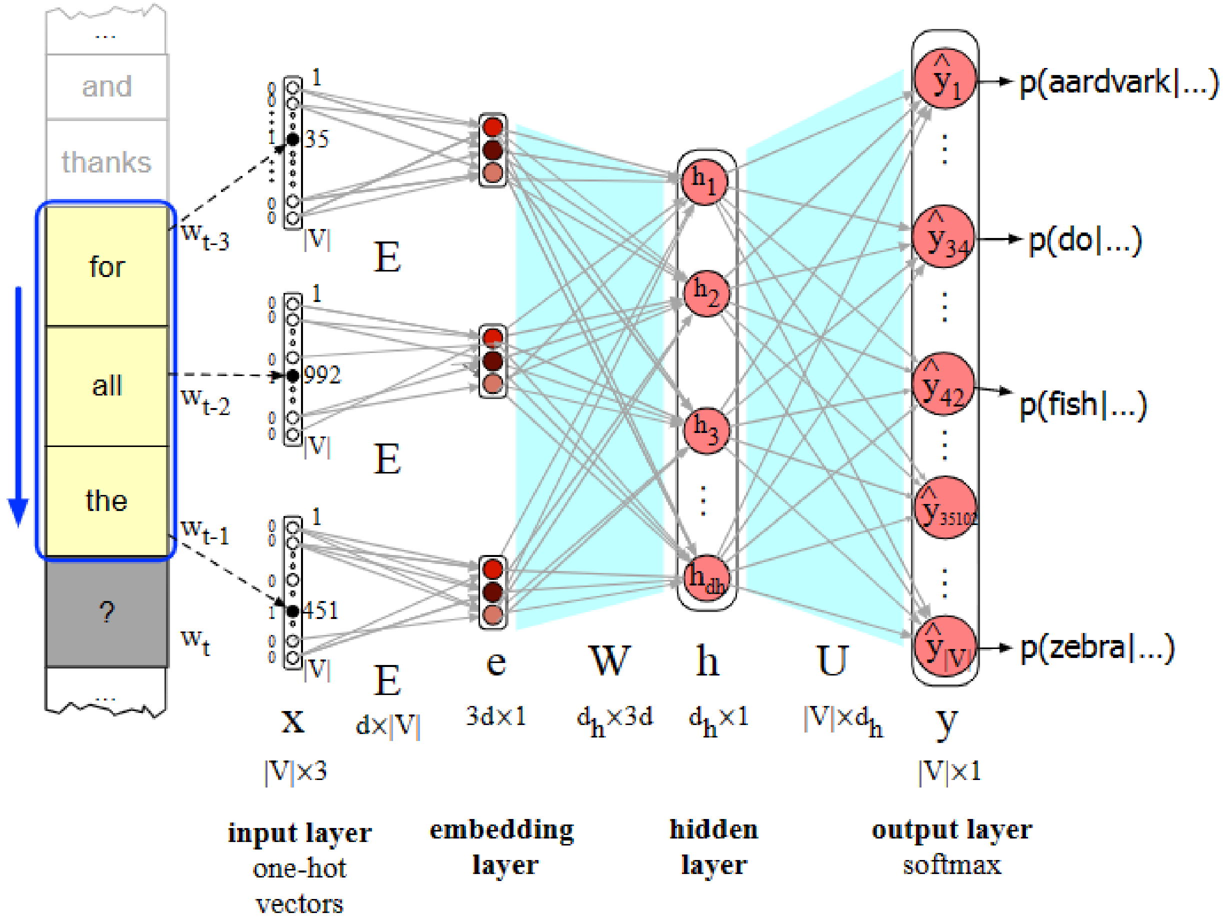
one-hot下对单词编码

- 单词用one-hot向量表示
- embedding 矩阵中，每个单词对应一列，通过乘法可以取出该列，得到该词的embedding



对于n-grams，前n-1个词的embedding连接在一起，作为embedding层的输入

如4-grams：



$$
\begin{aligned}
\mathbf{e} &= [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}] \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{e} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

Training Neural Nets

训练目标：每一层的可学习参数W和b，使得y的系统估计尽可能靠近真实的y

loss function：建模output与y的距离，使用梯度下降最下滑loss 函数、

误差反向传播error backpropagation或backward differentiation反向微分算法：计算中间层的loss

Loss function

二分类交叉熵loss

$$L_{CE}(\hat{y},y) = -\log p(y|x) \quad = \quad -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

多分类的交叉熵loss

hard classification：只有一个类别为true；K分类，$y_c = 1$表示c类别为true，其他为false

简化后的交叉熵loss：

$$L_{CE}(\hat{\mathbf{y}},\mathbf{y}) = -\sum_{k=1}^{K}\mathbf{y}_k\log\hat{\mathbf{y}}_k$$

符号表示：

$$L_{CE}(\hat{\mathbf{y}},\mathbf{y}) \quad = \quad -\sum_{k=1}^{K}\mathbb{1}\{\mathbf{y}_k=1\}\log\hat{\mathbf{y}}_k$$

进一步简化：negative og likelihood loss

$$L_{CE}(\hat{\mathbf{y}},\mathbf{y}) \quad = \quad -\log\hat{\mathbf{y}}_c \qquad (\text{where } c \text{ is the correct class})$$

插入softmax公式：

$$L_{CE}(\hat{\mathbf{y}},\mathbf{y}) \quad = \quad -\log\frac{\exp(\mathbf{z}_c)}{\sum_{j=1}^{K}\exp(\mathbf{z}_j)} \qquad (\text{where } c \text{ is the correct class})$$

Computing the Gradient

sigmoid求导：

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

对于一个只有一个参数层和sigmoid output的网络：

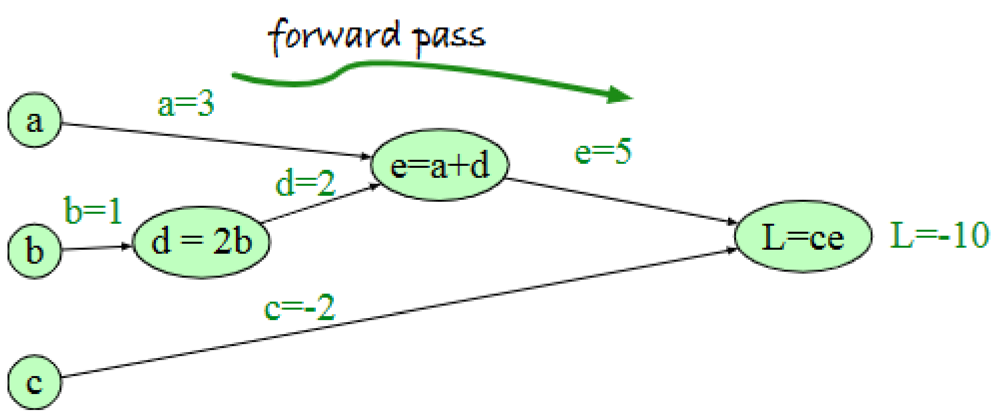
$$\begin{aligned}\frac{\partial L_{CE}(\hat{\mathbf{y}},\mathbf{y})}{\partial w_j} &= (\hat{y}-y)\,\mathbf{x}_j \\ &= (\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y)\,\mathbf{x}_j\end{aligned}$$

对于一个只有一个参数层和softmax output的网络：

$$\begin{aligned}\frac{\partial L_{\text{CE}}(\hat{\mathbf{y}},\mathbf{y})}{\partial \mathbf{w}_{k,i}} &= -(\mathbf{y}_k-\hat{\mathbf{y}}_k)\mathbf{x}_i \\ &= -(\mathbf{y}_k-p(\mathbf{y}_k=1|\mathbf{x}))\mathbf{x}_i \\ &= -\left(\mathbf{y}_k-\frac{\exp(\mathbf{w_k}\cdot\mathbf{x}+b_k)}{\sum_{j=1}^{K}\exp(\mathbf{w_j}\cdot\mathbf{x}+b_j)}\right)\mathbf{x}_i\end{aligned}$$

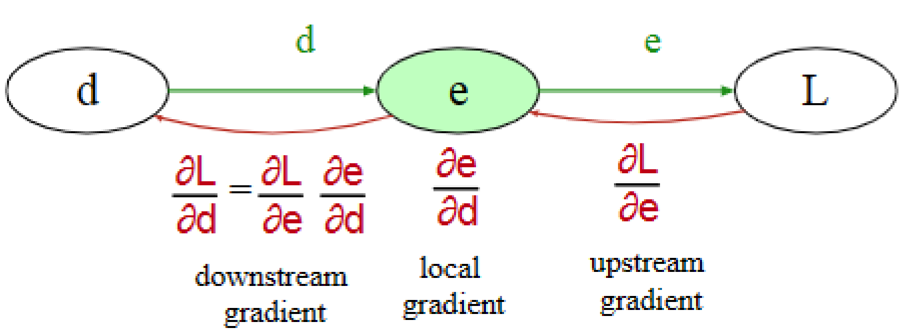
上述只能计算最后一层的梯度，计算多层梯度的解决方式：error backpropagation or backward differentiation

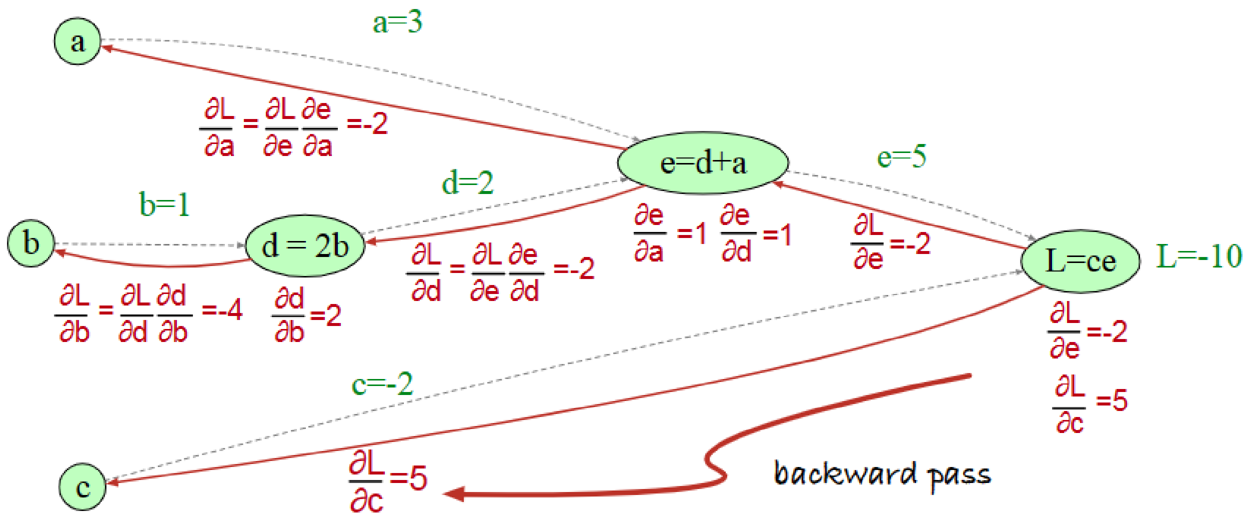
Computation Graphs



Backward differentiation on computation graphs

链式求偏导





Backward differentiation for a neural network

设网络：

$$
\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{x}+\mathbf{b}^{[1]} \\
\mathbf{a}^{[1]} &= \mathrm{ReLU}(\mathbf{z}^{[1]}) \\
z^{[2]} &= \mathbf{W}^{[2]}\mathbf{a}^{[1]}+b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

Loss：

$$
\begin{aligned}
L_{CE}(\hat{y},y) &= -[y\log\hat{y}+(1-y)\log(1-\hat{y})] \\
L_{CE}(a^{[2]},y) &= -\left[y\log a^{[2]}+(1-y)\log(1-a^{[2]})\right]
\end{aligned}
$$

激活函数求导：

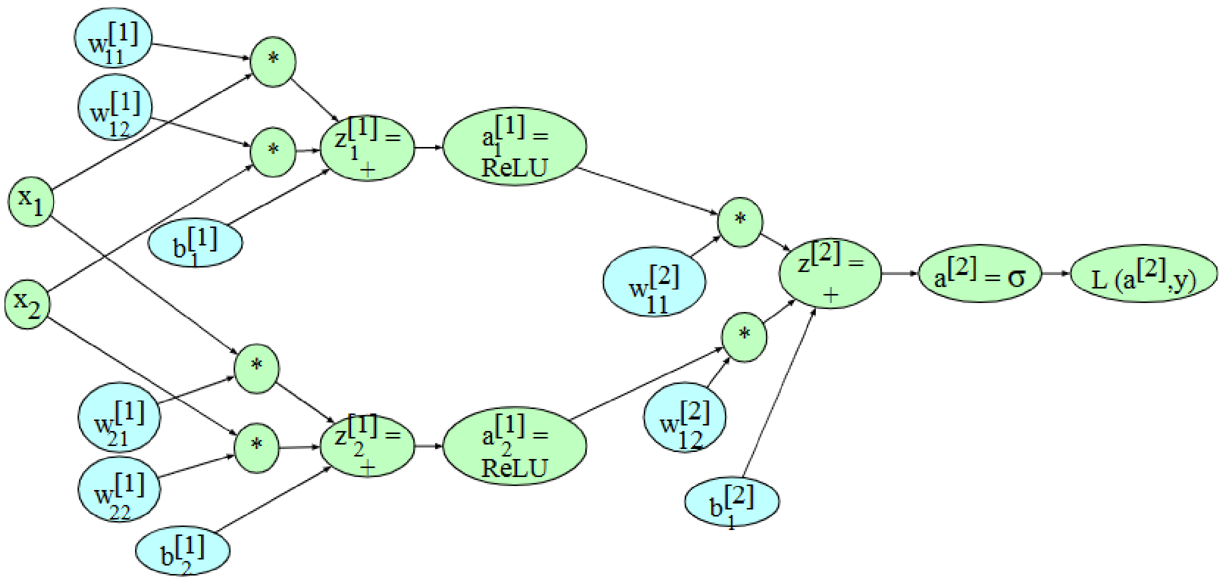
$$
\frac{d\sigma(z)}{dz}=\sigma(z)(1-\sigma(z))
$$

$$
\frac{d\tanh(z)}{dz}=1-\tanh^2(z)
$$

$$
\frac{d\,\mathrm{ReLU}(z)}{dz}=\begin{cases}0 & for \quad z<0\\ 1 & for \quad z\geq 0\end{cases}
$$

二分类softmax得到的交叉熵的求导过程：

计算树：



求导过程：

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z}$$

$$\begin{aligned}
L_{CE}(a^{[2]}, y) &= -\left[y \log a^{[2]} + (1-y)\log(1-a^{[2]})\right] \\
\frac{\partial L}{\partial a^{[2]}} &= -\left(\left(y\frac{\partial \log(a^{[2]})}{\partial a^{[2]}}\right) + (1-y)\frac{\partial \log(1-a^{[2]})}{\partial a^{[2]}}\right) \\
&= -\left(\left(y\frac{1}{a^{[2]}}\right) + (1-y)\frac{1}{1-a^{[2]}}(-1)\right) \\
&= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right)
\end{aligned}$$

$$\frac{\partial a^{[2]}}{\partial z} = a^{[2]}(1-a^{[2]})$$

初步求导结果：

$$
\begin{aligned}
\frac{\partial L}{\partial z} &= \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z} \\
&= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right) a^{[2]}(1-a^{[2]}) \\
&= a^{[2]} - y
\end{aligned}
$$

多分类softmax得到的交叉熵的求导过程：

$$\frac{\partial \mathrm{C}}{\partial \mathrm{z_i}} = \sum_{\mathrm{j}} \left(\frac{\partial \mathrm{C_j}}{\partial \mathrm{a_j}} \frac{\partial \mathrm{a_j}}{\partial \mathrm{z_i}}\right)$$

$$\frac{\partial \mathrm{C_j}}{\partial \mathrm{a_j}} = \frac{\partial(-\mathrm{y_j} \ln \mathrm{a_j})}{\partial \mathrm{a_j}} = -\mathrm{y_j} \frac{1}{\mathrm{a_j}}$$

①如果$\mathrm{i} = \mathrm{j}$：

$$\frac{\partial \mathrm{a_i}}{\partial \mathrm{z_i}} = \frac{\partial\left(\frac{e^{z_i}}{\sum_k e^{z_k}}\right)}{\partial \mathrm{z_i}} = \frac{\sum_k e^{z_k} e^{z_i} - (e^{z_i})^2}{(\sum_k e^{z_k})^2} = \left(\frac{e^{z_i}}{\sum_k e^{z_k}}\right)\left(1 - \frac{e^{z_i}}{\sum_k e^{z_k}}\right) = \mathrm{a_i}(1-\mathrm{a_i})$$

②如果$\mathrm{i} \neq \mathrm{j}$：

$$\frac{\partial \mathrm{a_j}}{\partial \mathrm{z_i}} = \frac{\partial\left(\frac{e^{z_j}}{\sum_k e^{z_k}}\right)}{\partial \mathrm{z_i}} = -e^{z_j}\left(\frac{1}{\sum_k e^{z_k}}\right)^2 e^{z_i} = -\mathrm{a_i}\mathrm{a_j}$$

$$\frac{\partial C}{\partial z_i} = \sum_j \left(\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}\right) = \sum_{j \neq i}\left(\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}\right) + \sum_{i=j}\left(\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}\right)$$

$$= \sum_{j \neq i} -y_j \frac{1}{a_j}(-a_i a_j) + (-y_i \frac{1}{a_i})(a_i(1 - a_i))$$

$$= \sum_{j \neq i} a_i y_j + (-y_i(1 - a_i))$$

$$= \sum_{j \neq i} a_i y_j + a_i y_i - y_i$$

$$= a_i \sum_j y_j - y_i$$

More details on learning

神经网络的优化：非凹优化问题

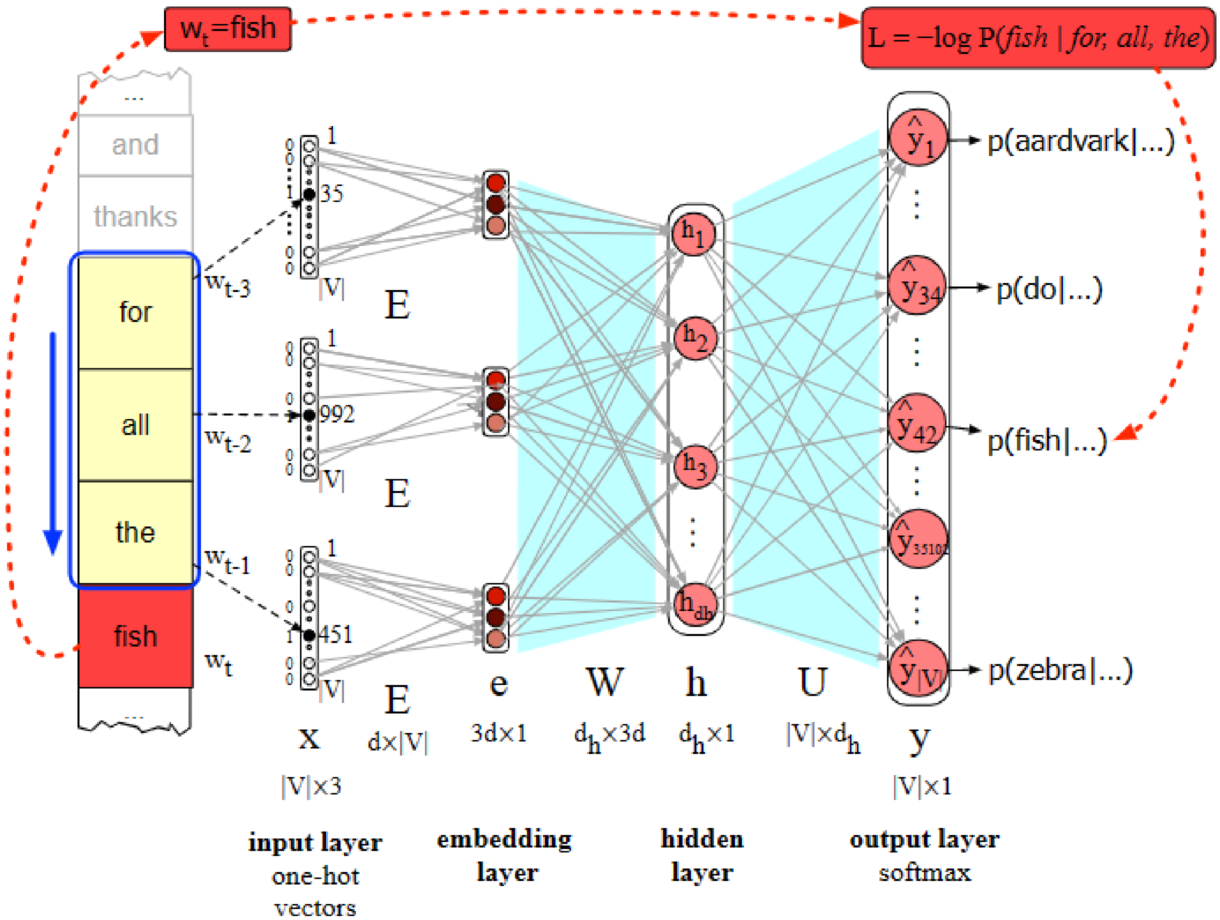
防止过拟合：

- dropout：随机丢弃部分units的结果
- 超参数微调

Training the neural language model

freeze：冻结参数，如冻结E，使得每个单词的编码固定不变，训练后续的网络

训练所有参数（E，W，U，b）的示例：



不想学习单独的权重矩阵，将前面的单词组合投影到投影层，而是使用一个权重矩阵，投影每个单词，而不是每个单词组合

loss：

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad \text{(where } i \text{ is the correct class)}$$

$$L_{CE} = -\log p(w_t | w_{t-1}, ..., w_{t-n+1})$$

每一步的参数更新：

$$\theta^{s+1} = \theta^s - \eta \frac{\partial \left[-\log p(w_t | w_{t-1}, ..., w_{t-n+1})\right]}{\partial \theta}$$

Summary

- 神经网络建立在units的基础上

- 每个units的公式
- 全连接前馈网络
- 神经网络的能力：深层利用浅层学习到的表征
- 神经网络训练使用梯度下降等优化算法
- Error backpropagation和backward differentiation on a computation graph，用于计算损失函数的梯度
- 神经网络预测upcoming word
- 神经网络使用训练好的embeddings或重新语料库中重新学习