Nature Language Processing - Aug 3'rd, 2018

Ways of representing words

- 需求：The less your algorithms automatically understand analogies like that, *man is to woman, as king is to queen*, and many other examples.
- One-hot representation
  - it treats each word as a thing unto itself, and it doesn't allow an algorithm to easily generalize the cross words.
  - any product between any two different one-hot vector is zero.
  - 并未考虑词与词之间的关系或者相似度 (cosine-similarity).(eg. *Man* vs. *Woman*, *King* vs. *Queue*)
  - Example
    - I want a glass of orange _____.
    - I want a glass of apple _____.

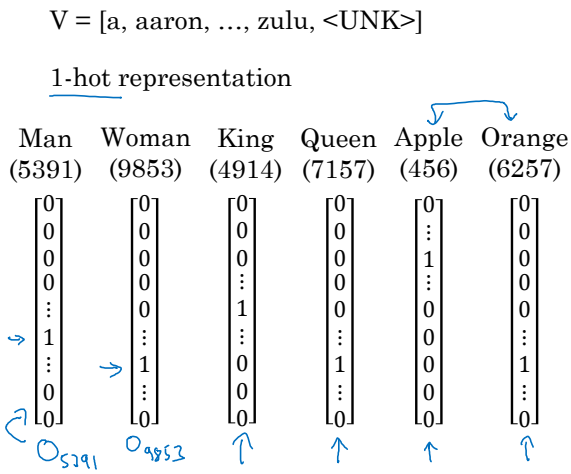


图 1：One-hot 表示法 ⟶if woman is word number 9853, then you represent it with O_{9853} which just has a one in position 9853 and zeros elsewhere.

- Featurized representation ⟶ word embedding(嵌入词向量)

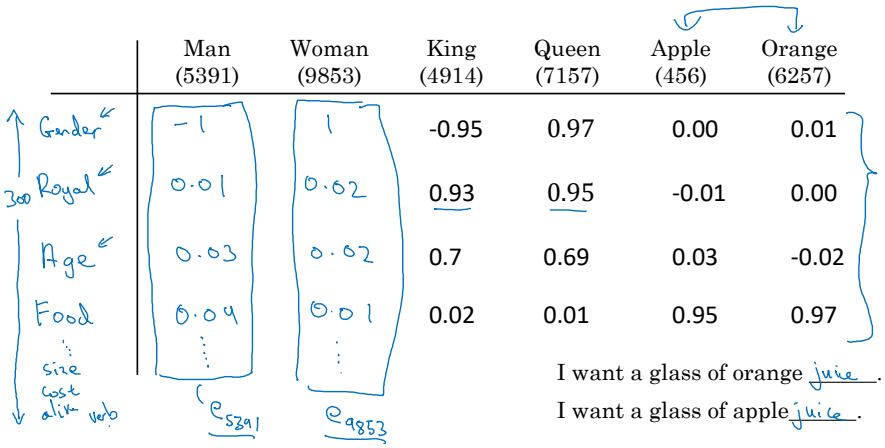


图 2：word embedding ⟶ e9853 to denote a 300 dimensional vector, used to represent the word woman

- it allow an algorithm to quickly figure out that apple and orange are more similar than say, king and orange or queen and orange.

t-SNE algorithm

- Paper: Laurens van der Maaten and Geoff Hinton, 2008. Visualizing data using t-SNE
- visualization

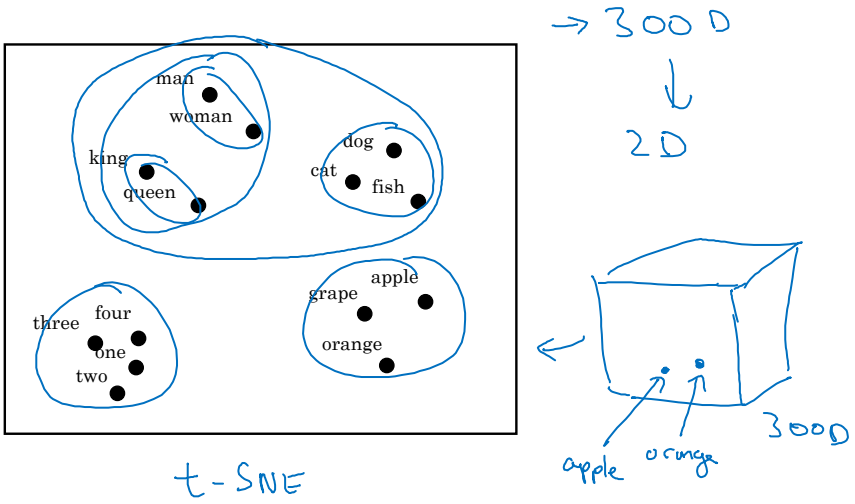


图 3：Visualizing word embeddings

Named entity recognition example

- 从一个例子开始继续以人名识别为例, 假设你想检测出人的名字. 给出一个句子，如“*Sally Johnson is an orange farmer.* 萨莉约翰逊是一个橙色的农夫.”，希望你能指出, *Sally Johnson* 是人的名字, 因此, 输出 1. 确保 *Sally Johnson* 必须是个人，而不是公司名字的一种方法就是你知道 *orange farmer* 是个人. 之前, 我们谈到了 one-hot 的表示法, 来表示这些单词, $x^{<1>}$, $x^{<2>}$, 等等. 但如果你现在使用特征化表示法, 即在上个视频中所谈到的嵌入向量. 模型在训练使用单词嵌入作为输入之后, 如果现在看到一个新的输入, “*Robert Lin is an apple farmer.* 罗伯特·林是一个种苹果的农夫”. 已知橙子和苹果非常相似，这将使学习算法更好地推广, 以找出 *Robert Lin* 罗伯特·林是人, 也是人的名字. 最有趣的情况之一是, 如果在测试集中, 你看到的不是“*Robert Lin is an apple farmer.* 罗伯特·林是个苹果农夫”, 而是不那么常见的词语呢？如果是“*Robert Lin is a durian cultivator* 罗伯特·林是榴莲的耕种者”呢？榴莲是一种罕见的水果, 在新加坡和其他少数节国家很流行. 但如果你为人名识别任务设置了一个小的标签训练集, 你可能根本没有在你的训练集中看到单词：“*durian* 榴莲”或“*cultivator* 耕种者”，严格来说, 应该是 *a durian cultivator* 一位种榴莲的耕种者. 但如果你学了一个单词嵌入, 它告诉你榴莲是一种水果, 它就像橙子一样, 并且告诉你耕种者和农民类似, 然后, 你可能在训练集中见过 *orange farmer* 橙子果民从而得知 *durian cultivator* 榴莲耕种者可能也是个人. 因此，单词嵌入能够做到这一点的原因之一是学习单词嵌入的算法可以检查大量的文本主体, 也许是在网上找到的这些文本所以你可以检查非常大的数据集, 也许会达到 10 亿字, 甚至多达 1000 亿字，这也是相当合理的. 大量的只含未标记文本的训练集通过检查大量的未标记的文本, 这些文本是可以免费下载的你可以弄清楚, 橙子和榴莲是类似的. 农夫和耕种者是类似的, 因此, 学习嵌入向量, 把他们组合在一起.
- 现在通过阅读大量的互联网文本发现, 橙子和榴莲都是水果, 你可以做的是, 采取这个词嵌入, 并将其应用到你的命名识别任务中, 你的训练集可能会小一些, 也许只有 10 万字, 甚至更小. 并且这可以让你进行学习迁移, 你可以利用来自文本的信息，这些大量未标记的文本可从网络上免费得到去得出橘子、苹果和榴莲都是水果. 然后将该知识迁移到任务上, 如命名实体识别, 对于这个任务，你可能有较小的

已标记的训练集. 当然, 简单起见, 我只画出单向 RNN. 如果你确实要执行命名识别任务, 则应该使用双向 RNN, 而不是我画的简单 RNN.

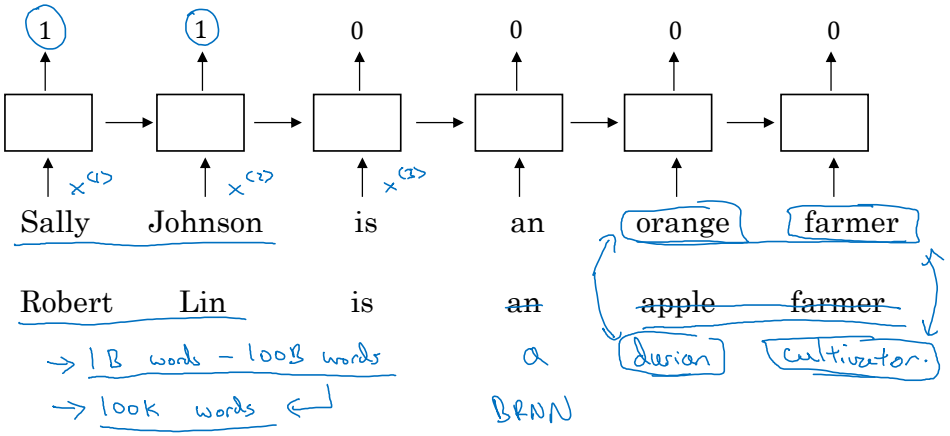


图 4：Visualizing word embeddings

如何使用单词嵌入来进行学习迁移?

1. 从大量的文本语料库学习单词嵌入或者可以从网上下载已经训练好的单词嵌入;
2. 把这些单词嵌入迁移到有着更小的已标记训练集的任务上;
3. 用这个 300 维的词嵌入来代表单词;
4. 最后，当你在新任务上训练你的模型时, 比如，在较小标签数据集的命名识别任务上, 你可以选择性的去继续微调参数, 继续用新数据调整单词嵌入.

- 应用：在命名实体识别, 文本摘要中可以使用, 指代消解，用于语法分析这些都是非常标准的 NLP 任务. 在语言建模、机器翻译上也有用处，尤其是在执行语言建模或机器翻译任务时你的大量数据专用于这些任务. 如同在其他迁移学习设置中看到的, 如果您正在尝试从某个任务 a 转移到某个任务 B, 迁移学习过程在这种情况下最有用的, 就是当你碰巧对 A 有很多数据集，B 有相对更少的数据集.

词嵌入与人脸识别中的编码的关系、

- Paper: Taigman et. al., 2014. DeepFace: Closing the gap to human level performance

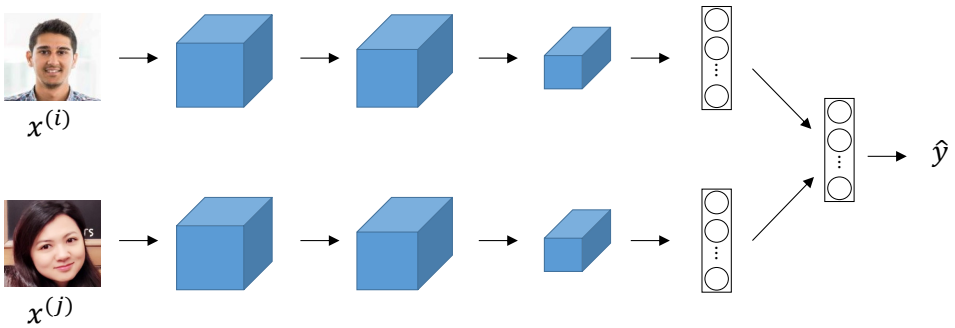


图 5：Encode vs. embedding

Analogy reasoning

- Paper: Mikolov et. al., 2013, Linguistic regularities in continuous space word representations
- Man is to woman as king is to queen

- $e_{man} - e_{women} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}, e_{king} - e_{queue} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Longrightarrow e_{man} - e_{women} \simeq e_{king} - e_{queue}$

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

图 6：Analogy reasoning1

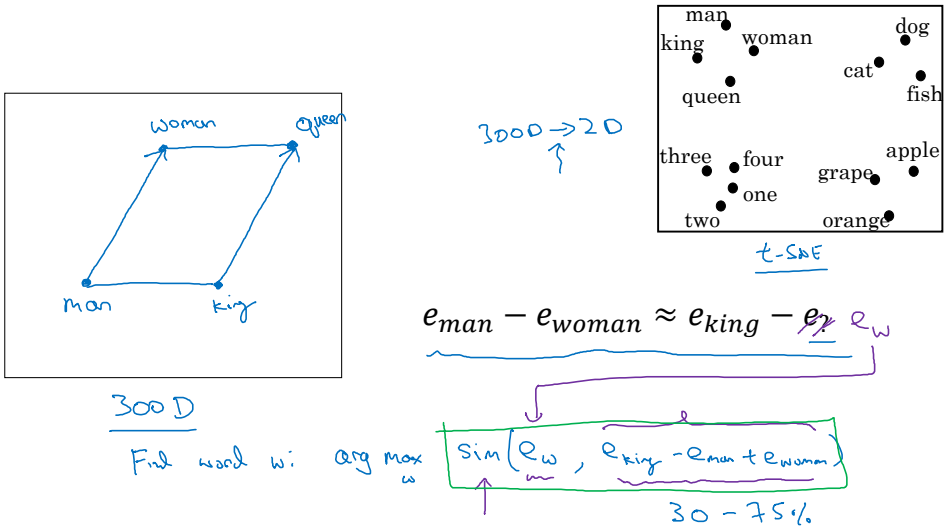
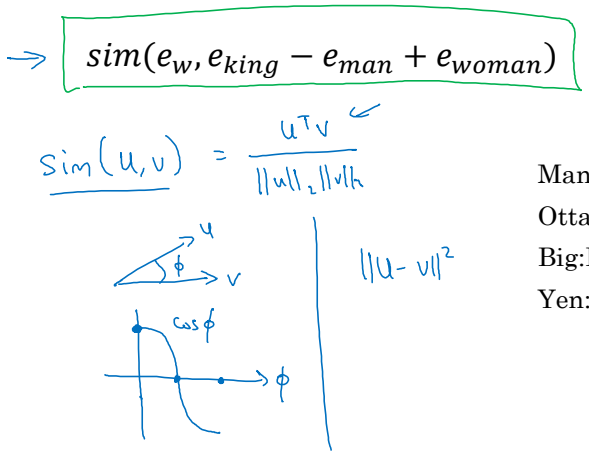


图 7：Analogy reasoning2

Cosine similarity

如何定义图 7 中的 similarity 呢？ ⟶ cosine similarity



Man:Woman as Boy:Girl
Ottawa:Canada as Nairobi:Kenya
Big:Bigger as Tall:Taller
Yen:Japan as Ruble:Russia

图 8：Cosine similarity

嵌入矩阵

当你实现一个算法来学习字嵌入时最终学习到的是一个嵌入矩阵.

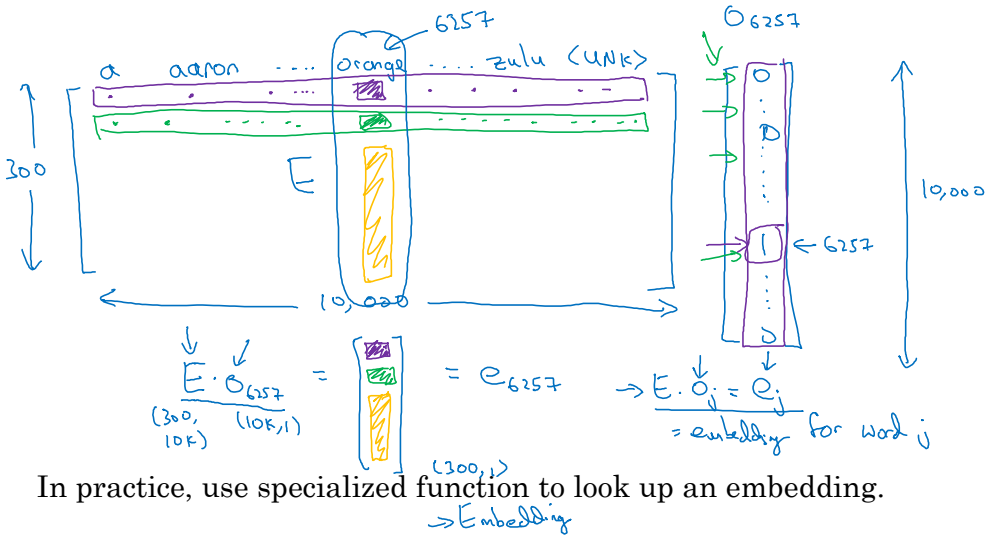


图 9：嵌入矩阵

如何学习嵌入矩阵

- 学习词嵌入的过程：复杂模型 $\longrightarrow$ 简单模型
- 神经网络语言模型
  - Paper: Bengio et. al., 2003, A neural probabilistic language model
  - 让你的神经网络做一些类似输入之类的工作，*I want a glass of orange.* 我想要一杯橙汁，然后预测序列中的下一个单词，在每个单词下面写下这个单词在词汇表上所处的索引.

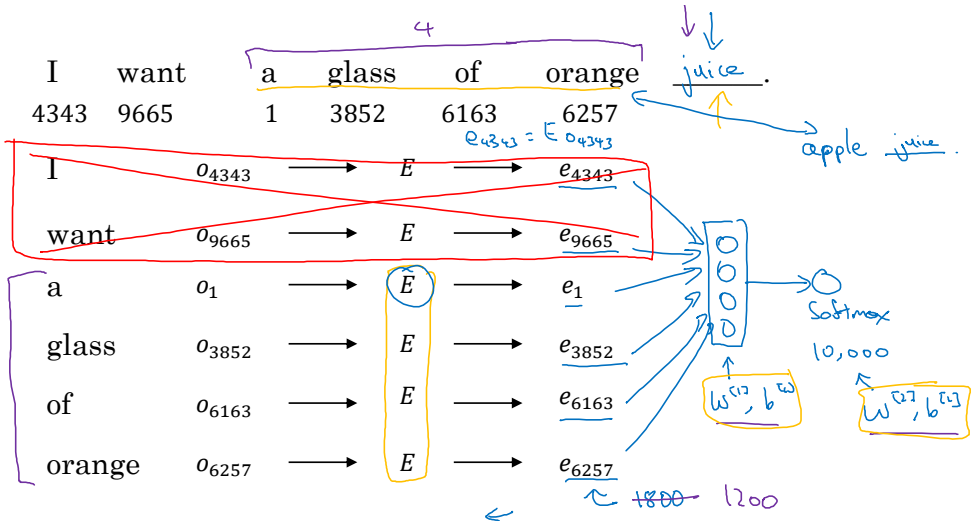


图 10：Neural Language Model

- 缺点：不好确定输入的维度大小，比如输入的句子比较长或者比较短的时候，不好确定维度.
- 常见的做法：确定固定的历史窗口来确定输入数据的大小.
  - Exmaple:I want a glass of orange *juice* to go along with my cereal.
- 构造上下文.

  - Exmaple:I want a glass of orange juice to go along with my cereal.
- 使用相邻的一个词作为上下文
  - Skip-Gram 模型的想法

词嵌入：word2vec 技术

- Credit by: `https://zh.gluon.ai/chapter_natural-language-processing/word2vec.html`

Skip-Gram Model

- 为何不采用 one-hot 向量
  - one-hot 词向量无法表达不同词之间的相似度，例如余弦相似度. 由于任意一对向量 $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ 的余弦相似度为
  $$\frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \in [-1, 1],$$
  任何一对词的 one-hot 向量的余弦相似度都为 0.
- word2vec
  - Paper：Mikolov et. al., 2013. Efficient estimation of word representations in vector space.
  - 背景：2013 年，Google 团队发表了 word2vec 工具. word2vec 工具主要包含两个模型：跳字模型（skip-gram）和连续词袋模型（continuous bag of words，简称 CBOW），以及两种近似训练法：负采样（negative sampling）和层序 softmax（hierarchical softmax）. 值得一提的是，word2vec 的词向量可以较好地表达不同词之间的相似和类比关系.
- 跳字模型
  - 在跳字模型中,我们用一个词来预测它在文本序列周围的词. 举个例子,假设文本序列是“the”、“man”、“loves”、“his”和“son”. 以“loves”作为中心词，设时间窗口大小为 2. 跳字模型所关心的是，给定中心词“loves”生成与它距离不超过 2 个词的背景词“the”、“man”、“his”和“son”的条件概率.
  - 假设词典索引集 $\mathcal{V}$ 的大小为 $|\mathcal{V}|$，且 $\mathcal{V} = 0, 1, ..., |V|-1$. 给定一个长度为 T 的文本序列中，时间步 t 的词为 $w(t)$. 当时间窗口大小为 m 时，跳字模型需要最大化给定任一中心词生成所有背景词的概率
  $$\prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} \mathbb{P}(w^{(t+j)} \mid w^{(t)}).$$
  上式的最大似然估计与最小化以下损失函数等价：
  $$-\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log \mathbb{P}(w^{(t+j)} \mid w^{(t)}).$$
  我们可以用 $\mathbf{v}$ 和 $\mathbf{u}$ 分别表示中心词和背景词的向量. 换言之，对于词典中索引为 i 的词，它在作为中心词和背景词时的向量表示分别是 v_i 和 u_i. 而词典中所有词的这两种向量正是跳字模型所要学习的模型参数. 为了将模型参数植入损失函数，我们需要使用模型参数表达损失函数中的给定中心词生成背景词的条件概率. 给定中心词，假设生成各个背景词是相互独立的. 设中心词 w_c 在词典中索引为 c，背景词 w_o 在词典中索引为 o，损失函数中的给定中心词生成背景词的条件概率可以通过 softmax 函数定义为
  $$\mathbb{P}(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)}.$$

当序列长度 T 较大时，我们通常在每次迭代时随机采样一个较短的子序列来计算有关该子序列的损失. 然后，根据该损失计算词向量的梯度并迭代词向量. 具体算法可以参考“梯度下降和随机梯度下降”一节. 作为一个具体的例子，下面我们看看如何计算随机采样的子序列的损失有关中心词向量的梯度. 和上面提到的长度为 T 的文本序列的损失函数类似，随机采样的子序列的损失实际上是对子序列中给定中心词生成背景词的条件概率的对数求平均. 通过微分，我们可以得到上式中条件概率的对数有关中心词向量 v_c 的梯度

$$\frac{\partial \log \mathbb{P}(w_o \mid w_c)}{\partial \mathbf{v}_c} = \mathbf{u}_o - \sum_{j \in \mathcal{V}} \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \mathbf{u}_j.$$

该式也可写作

$$\frac{\partial \log \mathbb{P}(w_o \mid w_c)}{\partial \mathbf{v}_c} = \mathbf{u}_o - \sum_{j \in \mathcal{V}} \mathbb{P}(w_j \mid w_c) \mathbf{u}_j.$$

随机采样的子序列有关其他词向量的梯度同理可得. 训练模型时，每一次迭代实际上是用这些梯度来迭代子序列中出现过的中心词和背景词的向量. 训练结束后，对于词典中的任一索引为 i 的词，我们均得到该词作为中心词和背景词的两组词向量 $\mathbf{v}_i$ 和 $\mathbf{u}_i$. 在自然语言处理应用中，我们会使用跳字模型的中心词向量.

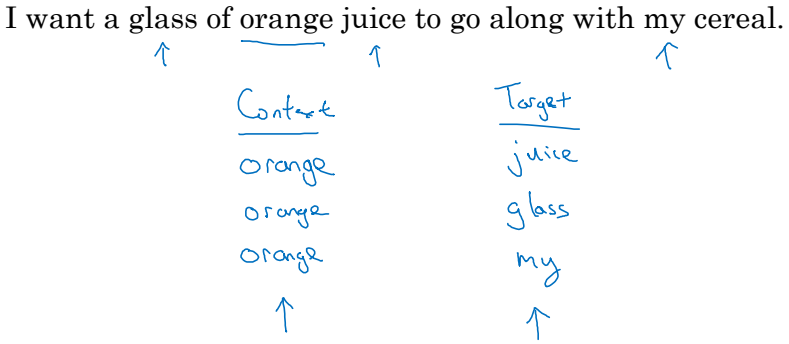


图 11：Skip-Gram Model（通过构造 context 和 target 来设计一个 supervised learning 问题.）

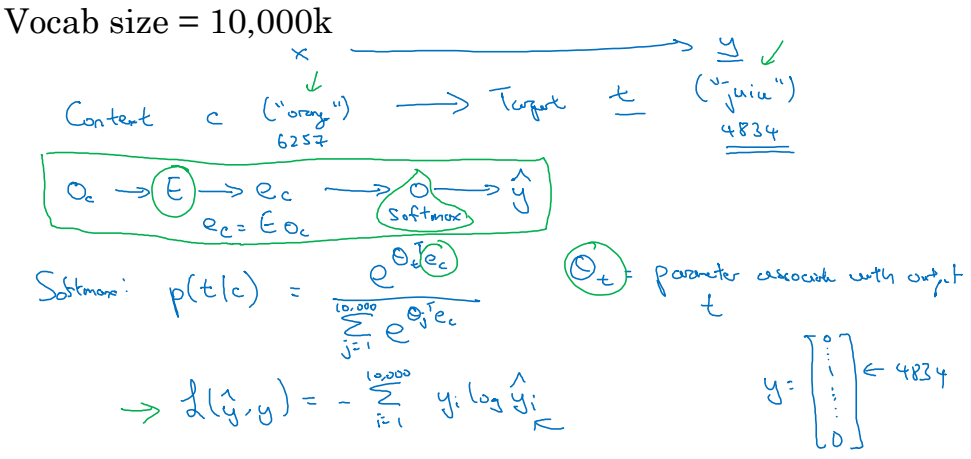


图 12：Skip-Gram Model 2

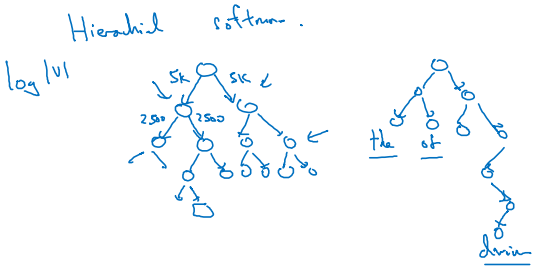
Problems with softmax classification

$$\mathbb{P}(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{i=1}^{10,000} e^{\theta_j^T e_c}}. \tag{1}$$

- 带来的问题：computational speed（看分母有 $\sum$）

- In particular, for the softmax model, every time you want to evaluate this probability, you need to carry out a sum over all 10,000 words in your vocabulary. And maybe 10,000 isn't too bad, but if you're using a vocabulary of size 100,000 or a 1,000,000, it gets really slow to sum up over this denominator every single time.
- And, in fact, 10,000 is actually already that will be quite slow, but it makes even harder to scale to larger vocabularies.
- 解决方法：in the literature, using a **hierarchical softmax classifier**(层序 softmax). And what that means is, instead of trying to categorize something into all 10,000 carries on one go.
  - Imagine if you have one classifier, it tells you is the target word in the first 5,000 words in the vocabulary? Or is in the second 5,000 words in the vocabulary? And lets say this binary cost that it tells you this is in the first 5,000 words, think of second class to tell you that this in the first 2,500 words of vocab or in the second 2,500 words vocab and so on. Until eventually you get down to classify exactly what word it is, so that the leaf of this tree, and so having a tree of classifiers like this, means that each of the retriever nodes of the tree can be just a binding classifier. And so you don't need to sum over all 10,000 words or else it will capsize in order to make a single classification
  - 时间复杂度降为 $\mathcal{O}(\log_2|\mathcal{V}|)$.

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$



- 工作原理：层序 softmax 是另一种常用的近似训练法。它利用了二叉树这一数据结构。树的每个叶子节点代表着词典 $\mathcal{V}$ 中的每个词。我们以图 13 为例来描述层序 softmax 的工作机制。

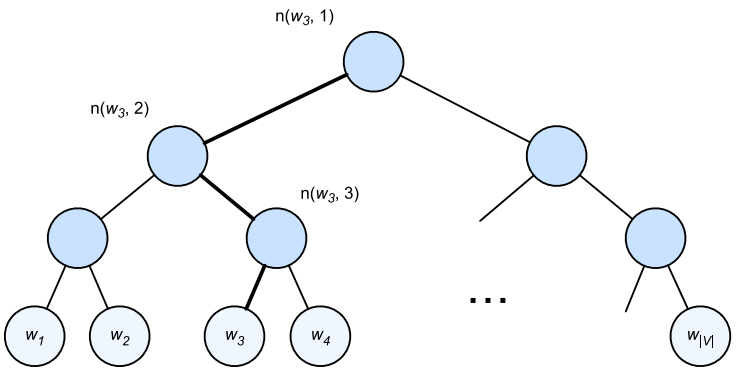


图 13：层序 softmax

假设 $L(w)$ 为从二叉树的根节点到词 w 的叶子节点的路径（包括根和叶子节点）上的节点数。设 $n(w,j)$ 为该路径上第 j 个节点，并设该节点的向量为 $\mathbf{u}_n(w,j)$。以图 13 为例，$L(w_3)$=4。设词典中的词 w_i

的词向量为 $\mathbf{v}_i$。那么，跳字模型和连续词袋模型所需要计算的给定词 w_i 生成词 w 的条件概率为：

$$\mathbb{P}(w \mid w_i) = \prod_{j=1}^{L(w)-1} \sigma\left([\![n(w,j+1) = \text{leftChild}(n(w,j))]\!] \cdot \mathbf{u}_{n(w,j)}^\top \mathbf{v}_i\right),$$

其中 $\sigma(x) = 1/(1+\exp(-x))$ 是节点 n 的左孩子节点，如果判断 x 为真，$[\![x]\!] = 1$；反之 $[\![x]\!] = -1$。由于 $\sigma(x) + \sigma(-x) = 1$，给定词 w_i 生成词典 $\mathcal{V}$ 中任一词的条件概率之和为 1 这一条件也将满足：

$$\sum_{w \in \mathcal{V}} \mathbb{P}(w \mid w_i) = 1.$$

让我们计算图 13 中给定词 w_i 生成词 w_3 的条件概率。我们需要将 w_i 的词向量 v_i 和根节点到 w_3 路径上的非叶子节点向量一一求内积。由于在二叉树中由根节点到叶子节点 w_3 的路径上需要向左、向右、再向左地遍历（图 13 中加粗的路径），我们得到

$$\mathbb{P}(w_3 \mid w_i) = \sigma(\mathbf{u}_{n(w_3,1)}^\top \mathbf{v}_i) \cdot \sigma(-\mathbf{u}_{n(w_3,2)}^\top \mathbf{v}_i) \cdot \sigma(\mathbf{u}_{n(w_3,3)}^\top \mathbf{v}_i).$$

在使用 softmax 的跳字模型和连续词袋模型中，词向量和二叉树中非叶子节点向量是需要学习的模型参数。假设词典 $\mathcal{V}$ 很大，每次迭代的计算开销由 $\mathcal{O}(|\mathcal{V}|)$ 下降至 $\mathcal{O}(\log_2|\mathcal{V}|)$。

连续词袋模型

- 连续词袋模型与跳字模型类似. 与跳字模型最大的不同是，连续词袋模型用一个中心词在文本序列前后的背景词来预测该中心词. 举个例子，假设文本序列为“the”、“man”、“loves”、“his”和“son”. 以“loves”作为中心词，设时间窗口大小为 2. 连续词袋模型所关心的是，给定与中心词距离不超过 2 个词的背景词“the”、“man”、“his”和“son”生成中心词“loves”的条件概率.
  假设词典索引集 $\mathcal{V}$ 的大小为 $|\mathcal{V}|$，且 $\mathcal{V} = 0, 1, ..., |\mathcal{V}|-1$. 给定一个长度为 T 的文本序列中，时间步 t 的词为 $w(t)$. 当时间窗口大小为 m 时，连续词袋模型需要最大化由背景词生成任一中心词的概率

$$\prod_{t=1}^{T} \mathbb{P}(w^{(t)} \mid w^{(t-m)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+m)}).$$

  上式的最大似然估计与最小化以下损失函数等价：

$$-\sum_{t=1}^{T} \log\mathbb{P}(w^{(t)} \mid w^{(t-m)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+m)}).$$

  我们可以用 $\mathbf{v}$ 和 $\mathbf{u}$ 分别表示背景词和中心词的向量（注意符号和跳字模型中的不同）. 换言之，对于词典中索引为 i 的词，它在作为背景词和中心词时的向量表示分别是 $\mathbf{v}_i$ 和 $\mathbf{u}_i$. 而词典中所有词的这两种向量正是连续词袋模型所要学习的模型参数. 为了将模型参数植入损失函数，我们需要使用模型参数表达损失函数中的给定背景词生成中心词的概率. 设中心词 w_c 在词典中索引为 c，背景词 w_{o1},⋯,w_{o2m} 在词典中索引为 o_1,⋯,o_2m，损失函数中的给定背景词生成中心词的概率可以通过 softmax 函数定义为

$$\mathbb{P}(w_c \mid w_{o_1}, \ldots, w_{o_{2m}}) = \frac{\exp\left(\mathbf{u}_c^\top (\mathbf{v}_{o_1} + \ldots + \mathbf{v}_{o_{2m}})/(2m)\right)}{\sum_{i \in \mathcal{V}} \exp\left(\mathbf{u}_i^\top (\mathbf{v}_{o_1} + \ldots + \mathbf{v}_{o_{2m}})/(2m)\right)}.$$

  和跳字模型一样，当序列长度 T 较大时，我们通常在每次迭代时随机采样一个较短的子序列来计算有关该子序列的损失. 然后，根据该损失计算词向量的梯度并迭代词向量. 通过微分，我们可以计算出上式中条件概率的对数有关任一背景词向量 $v_{oi}(i = 1, ..., 2m)$ 的梯度为：

$$\frac{\partial \log\mathbb{P}(w_c \mid w_{o_1}, \ldots, w_{o_{2m}})}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m}\left(\mathbf{u}_c - \sum_{j \in \mathcal{V}} \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \mathbf{u}_j\right).$$

该式也可写作

$$\frac{\partial \log \mathbb{P}(w_c \mid w_{o_1}, \ldots, w_{o_{2m}})}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m}\left(\mathbf{u}_c - \sum_{j \in \mathcal{V}} \mathbb{P}(w_j \mid w_c)\mathbf{u}_j\right).$$

随机采样的子序列有关其他词向量的梯度同理可得. 和跳字模型一样，训练结束后，对于词典中的任一索引为 i 的词，我们均得到该词作为背景词和中心词的两组词向量 $\mathbf{v}_i$ 和 $\mathbf{u}_i$. 在自然语言处理应用中，我们会使用连续词袋模型的背景词向量.

负采样

- 利用 Skip-Gram 模型构建一个监督学习任务, 使得我们建立从上下文到目标词语的映射, 并基于此训练出一个有用的词嵌入, 但是这种方法的弊端在于 Softmax 函数**训练起来比较慢**.
- Paper: Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality
- 工作原理:建立一个新的监督学习问题,给定一对词语,比如“orange”和“juice”,我们要预测这是否是一对 (context,target),在这个例子中“orange”和“juice”是一个正样本,即可以表示为(“orange”,“juice”, 1)，那么“orange”和“king”就是一组负样本，我们在 label 这一栏标记为 0，那么我们要做的就是采样一个上下文和目标词语，在这个例子中，我们采样的是“orange”和“juice”，并且把这对词语标记为 1，我们把中间的这一栏称为“word”，那么，和之前的方法一样，生成一个正样本之后，及采样一个上下文词语（context），然后在附近的一个窗口，例如前后 10 个词语，选取一个目标词语（word，且 target 为 1），那么这就是你生成这个表格第一行的方法；然后从字典中随机抽取一个词语，在这里随机选取了“king”这个词语，并且将这对词语标记为 0，然后仍然选择“orange”作为 context，并且从字典中再随机选择一个词语，假设这次选择了“book”，同样的标记为目标为 0，..., 只要是从字典中随机抽取的就行，把这些全部标记为 0，这样就得到了我们的负样本.
- 至于如何选取 k，Mikolov 等人建议对于小的数据集，或许 k 介于 5 到 20 如果你有一个很大的数据集，那么选取较小的 k 也就是，对于大的数据集，k 介于 2 到 5 对于小的数据集选取更大的 k 值.

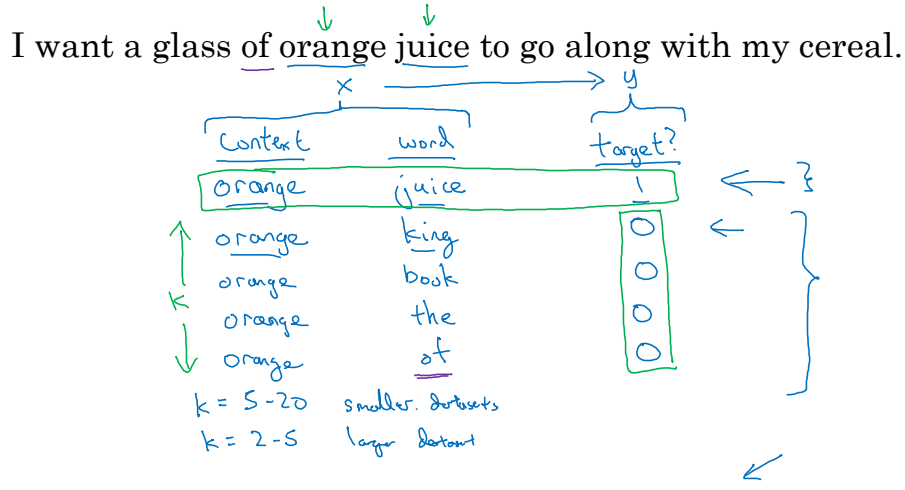


图 15：定义一个新的监督问题

- 定义一个逻辑回归模型例如，在给定 c、t 的条件下，$y = 1$ 的概率，我们要把这个建模为一个逻辑回归模型，但具体我们采用的公式是 Sigmoid 函数，应用在 $\theta_t^\top$，乘以 e_c, 这些参数和以前差不多，对每一个可能的目标词语，你有一个参数向量 θ 以及对于每一个上下文词语，另一个参数向量实际上也就是词嵌入向量, 我们将用这个公式来估计 $y = 1$ 的概率假设你有 k 个样本，那么你可以看成是有一个 $k-1$ 的负样本-正样本比率对每一个正样本，你有 k 个负样本来训练这个类似逻辑回归的模型那么把这个转换成一个神经网络，如果输入词语是“orange”，也就是第 6257 个词语，你要做的是输入这个

one-hot 向量传递给 e，利用向量乘法，得到嵌入向量 e_6275，那么你实际上得到了 10,000 个可能的逻辑回归分类问题，其中一个就是对应于目标词语是否是“juice”的分类器.

- 想象这是有 10,000 个二进制逻辑回归分类器但是，并非要在每次迭代中训练全部 10,000 个分类器，我们只训练其中的 5 个，也就是对应于实际的目标词语的那个以及对应于其他 4 个随机抽取的负样本的分类器，这就是吱吱吱吱 $k = 4$ 时的情形.

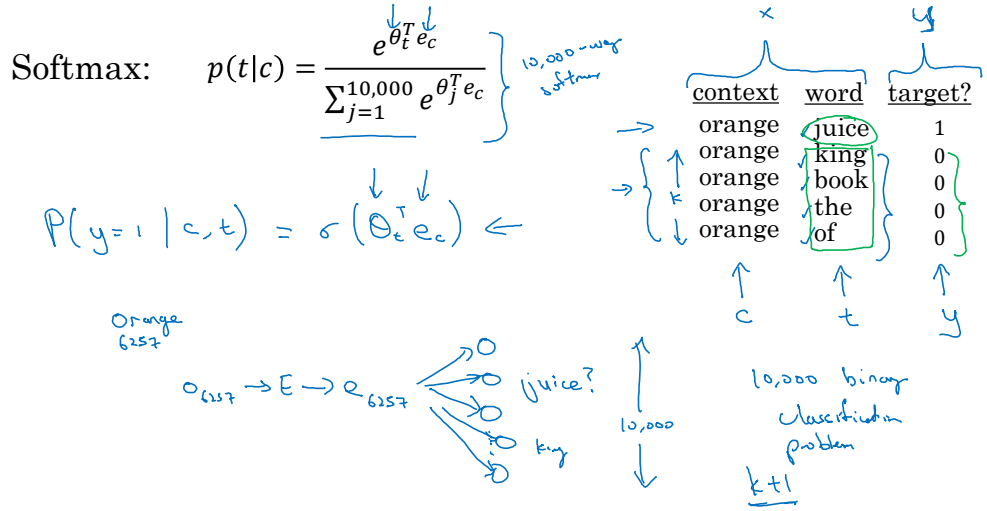


图 16：负采样

- 如何选取这些负样本
  - 抽取中间的词语
  - 根据你的文本中词语的经验频率分布来抽样也就是根据词语出现的频率来抽样.
    - 会得到很多例如“the”、“of”、“and”之类的词语
  - 根据字典词汇量的倒数均匀地抽取负样本 *longrightarrow* 不能够代表英语中单词的分布
  - **启发式的做法：**从经验频率分布中采样，即观察到的英语文本中的分布，到从均匀分布中采样. 依词频的 3/4 次幂来抽样，那么例如 $f(w_i)$ 表示某个观察到的英语中的词频或者说你训练集中的词频，那么取它的 3/4 次幂 $f(w_i)^{3/4}$，这样它就介于取均匀分布和取经验分布的两个极端之间.
    - 目前都在使用这种方法

负采样总结

- 如何用 Softmax 分类器来训练词向量，但这样计算上成本很高
- 如何把它变为一系列二元分类问题, 并且非常高效地训练词向量
  - 如果你运行这种算法，你会得到非常好的词向量
  - 有开源的实现

GloVe 算法

- 有一定的发展潜力的算法，简单，但没有 word2vec 和 Skip-Gram 算法那样广泛使用
- Paper: Pennington et. al., 2014. GloVe: Global vectors for word representation
- GloVe 的意思是用于词汇表征的全局矢量

GloVe

`https://zh.gluon.ai/chapter_natural-language-processing/glove-fasttext.html`

GloVe 使用了词与词之间的共现（co-occurrence）信息。我们定义 X 为共现词频矩阵，其中元素 x_{ij} 为词 j 出现在词 i 的背景的次数。这里的“背景”有多种可能的定义。举个例子，在一段文本序列中，如果词 j 出现在词 i 左边或者右边不超过 10 个词的距离，我们可以认为词 j 出现在词 i 的背景一次。令 $x_i = \sum_k x_{ik}$ 为任意词出现在词 i 的背景的次数。那么，

$$p_{ij} = \mathbb{P}(j \mid i) = \frac{x_{ij}}{x_i}$$

为词 j 在词 i 的背景中出现的条件概率。这一条件概率也称词 i 和词 j 的共现概率。

共现概率比值

GloVe 论文里展示了以下词对的共现概率与比值 [1]：

- $\mathbb{P}(k \mid \text{ice})$：0.00019（$k$= solid），0.000066（$k$= gas），0.003（$k$= water），0.000017（$k$= fashion）
- $P\mathbb{P}(k \mid \text{steam})$：0.000022（$k$= solid），0.00078（$k$= gas），0.0022（$k$= water），0.000018（$k$= fashion）
- $\mathbb{P}(k \mid \text{ice})/\mathbb{P}(k \mid \text{steam})$：8.9（$k$= solid），0.085（$k$= gas），1.36（$k$= water），0.96（$k$= fashion）

我们可以观察到以下现象：

- 对于与 ice（冰）相关而与 steam（蒸汽）不相关的词 k，例如 k=solid（固体），我们期望共现概率比值 p_{ik}/p_{jk} 较大，例如上面最后一行结果中的值 8.9。
- 对于与 ice 不相关而与 steam 相关的词 k，例如 k=gas（气体），我们期望共现概率比值 p_{ik}/p_{jk} 较小，例如上面最后一行结果中的值 0.085。
- 对于与 ice 和 steam 都相关的词 k，例如 k=water（水），我们期望共现概率比值 p_{ik}/p_{jk} 接近 1，例如上面最后一行结果中的值 1.36。
- 对于与 ice 和 steam 都不相关的词 k，例如 k=fashion（时尚），我们期望共现概率比值 p_{ik}/p_{jk} 接近 1，例如上面最后一行结果中的值 0.96。

由此可见，共现概率比值能比较直观地表达词与词之间的关系。GloVe 试图用有关词向量的函数来表达共现概率比值。

用词向量表达共现概率比值

GloVe 的核心思想在于使用词向量表达共现概率比值。而任意一个这样的比值需要三个词 i、j 和 k 的词向量。对于共现概率 $p_{ij} = \mathbb{P}(j \mid i)$，我们称词 i 和词 j 分别为中心词和背景词。我们使用 $\boldsymbol{v}_i$ 和 $\tilde{\boldsymbol{v}}_i$ 分别表示词 i 作为中心词和背景词的词向量。

我们可以用有关词向量的函数 f 来表达共现概率比值：

$$f(\boldsymbol{v}_i, \boldsymbol{v}_j, \tilde{\boldsymbol{v}}_k) = \frac{p_{ik}}{p_{jk}}.$$

需要注意的是，函数 f 可能的设计并不唯一。下面我们考虑一种较为合理的可能性。

首先，用向量之差来表达共现概率的比值，并将上式改写成

$$f(\boldsymbol{v}_i - \boldsymbol{v}_j, \tilde{\boldsymbol{v}}_k) = \frac{p_{ik}}{p_{jk}}.$$

由于共现概率比值是一个标量，我们可以使用向量之间的内积把函数 f 的自变量进一步改写，得到

$$f((\boldsymbol{v}_i - \boldsymbol{v}_j)^\top \tilde{\boldsymbol{v}}_k) = \frac{p_{ik}}{p_{jk}}.$$

由于任意一对词共现的对称性，我们希望以下两个性质可以同时被满足：

- 任意词作为中心词和背景词的词向量应该相等：对任意词 i，$\boldsymbol{v}_i = \tilde{\boldsymbol{v}}_i$。
- 词与词之间共现词频矩阵 X 应该对称：对任意词 i 和 j，$x_{ij} = x_{ji}$。

为了满足以上两个性质，一方面，我们令

$$f((\boldsymbol{v}_i - \boldsymbol{v}_j)^\top \tilde{\boldsymbol{v}}_k) = \frac{f(\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_k)}{f(\boldsymbol{v}_j^\top \tilde{\boldsymbol{v}}_k)},$$

并得到 $f(x) = exp(x)$。以上两式的右边联立，

$$f(\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_k) = \exp(\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_k) = p_{ik} = \frac{x_{ik}}{x_i}.$$

由上式可得

$$\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_k = \log(p_{ik}) = \log(x_{ik}) - \log(x_i).$$

另一方面，我们可以把上式中 $\log(xi)$ 替换成两个偏差项之和 $b_i + \tilde{b}_k$，得到

$$\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_k = \log(x_{ik}) - b_i - \tilde{b}_k.$$

因此，对于任意一对词 i 和 j，我们可以用它们的词向量表达共现词频的对数：

$$\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_j + b_i + \tilde{b}_j = \log(x_{ij}).$$

<u>损失函数</u>

GloVe 中的共现词频是直接在训练数据上统计得到的。为了学习词向量和相应的偏差项，我们希望上式中的左边与右边尽可能接近。给定词典 $\mathcal{V}$ 和权重函数 $h(x_{ij})$，GloVe 的损失函数为

$$\sum_{i \in \mathcal{V}, j \in \mathcal{V}} h(x_{ij}) \left(\boldsymbol{v}_i^\top \tilde{\boldsymbol{v}}_j + b_i + \tilde{b}_j - \log(x_{ij})\right)^2,$$

其中权重函数 $h(x)$ 的一个建议选择是，当 $x < c$（例如 $c = 100$），令 $h(x) = (x/c)^\alpha$（例如 $\alpha = 0.75$），反之令 $h(x) = 1$。由于权重函数的存在，损失函数的计算复杂度与共现词频矩阵 X 中非零元素的数目呈正相关。我们可以从 X 中随机采样小批量非零元素，并使用优化算法迭代共现词频相关词的向量和偏差项。

我们提到过，任意词的中心词向量和背景词向量是等价的。但由于初始化值的不同，同一个词最终学习到的两组词向量可能不同。当学习得到所有词向量以后，GloVe 使用一个词的中心词向量与背景词向量之和作为该词的最终词向量。

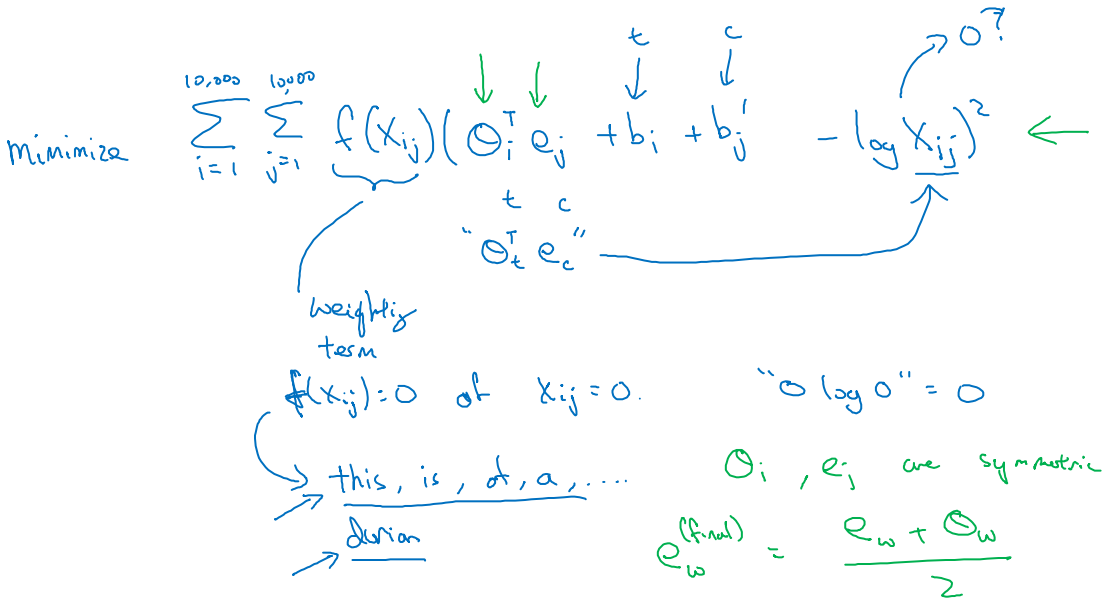


图 17：GloVe Model

实例研究：情绪分类

- 情感分类的任务是分析一段文本告诉人们某个人是不是喜欢他们正在谈论的东西
  - 它是自然语言处理最重要的组成部分
- 挑战
  - 可能缺乏一个特别大的标签训练集
    - 解决方案：使用词嵌入（word embedding）后，依靠一个中等大小的标签训练集也可以构建出一个很好的情感分类器.

x	y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

图 18：情感分类问题（酒店的星级评定）

Simple sentiment classification model

- 问题：忽视了单词的顺序

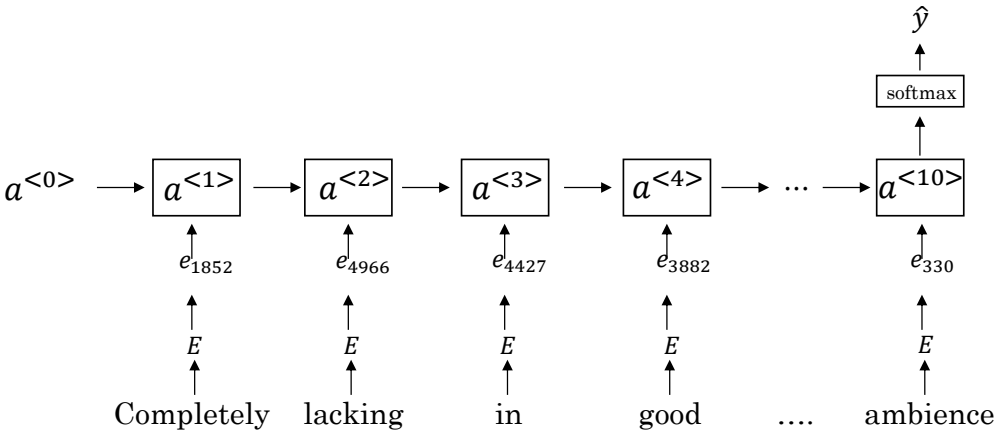


图 19：architecture of sentiment classification model

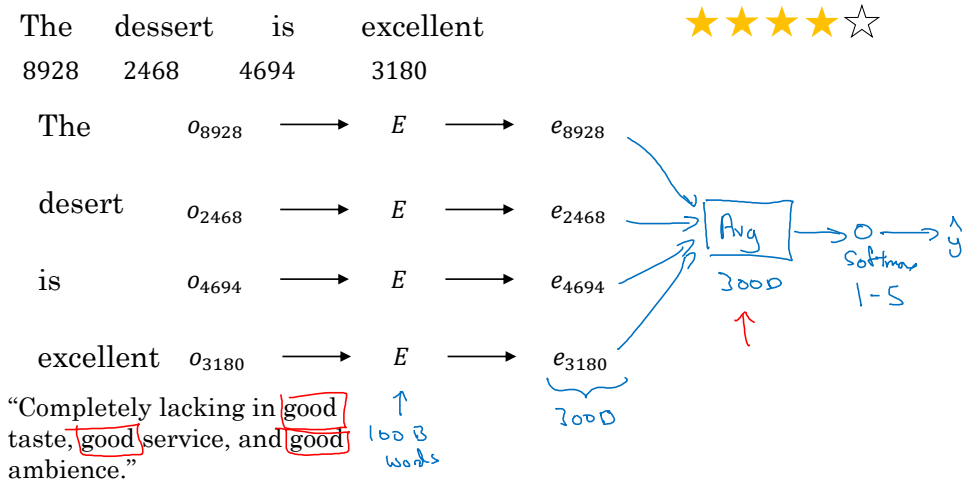


图 20：sentiment classification model

Debiasing word embeddings

Paper: Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings

机器学习和 AI 算法越来越受人信任，可以用来帮人做出非常重要的决策，所以我们需要尽可能确保算法中没有我们不希望看到的偏见，如性别偏见、种族偏见等，这节课我想教你一些方法来减少或消除在词嵌入中的这些形式的偏差，当我用术语偏差时，我指的不是“偏差”这个变量不是说数据偏离，我指的是性别、种族、性取向的偏差，这两者之间的意思是不同的，这里就指的不是我在在机器学习方面技术方面提到的偏差，而是关于词偏差方面的问题，我们要讲的是如何让词嵌入学习相似性，如男人对应女人，国王对应皇后但如果你说男人对应程序猿，那么女人对应什么呢？

所以这篇论文的作者：Tolga Bolukbasi Kai Wei Chang，James Zou，Venkatesh Saligrama 和 Adam Kalai 找到一种很奇特的方法，使得词嵌入可以做到这样的输出：男人对应程序猿，女人对应家庭主妇, 而这看起来还是错的，因为这导致了一个不正常的性别刻板印象，比较好的算法应该是可以得出：男人对应程序猿，而女人也对应程序猿. 同时他们还发现，父亲对应医生，那母亲对应什么呢？不幸的是，一些训练好的词嵌入会产生这样的输出：母亲对应护士，所以词嵌入会反应性别、种族、年龄性取向，以及其他的一些偏差用来训练模型. 其中有一个我特别感兴趣的是社会经济学地位的偏差, 我认为每个人，无论你是来自富有的家庭, 还是低收入的家庭，或者之间, 我认为每个人都应该有很大的机会来发展自己, 并且因为机器学习算法

能够用来做出许多重要的决定, 这些算法能影响每件事，从申请学校到求职再到申请贷款，从你的贷款申请有没有被批准再到司法系统里，来量刑审判罪犯学习算法都在做出非常重要的决策，所以我认为尽量消除算法中的这些偏差是很重要的. 更明确来讲，是消除这些不想看到的偏差结果. 目前的单词嵌入可以找出偏差的文本用来训练模型，而这些找出来的偏差是人为影响的几十年间，几个世纪间, 我认为人类在减少这类偏差方面有了很大的进步, 对 AI 而言幸运的是，我们有更好的方法来更快地减少由 AI 造成的偏差, 而且相比人造成的偏差更快. 尽管我们从未说 AI 已经发展得很好了, 仍然有许多研究要做, 我们需要很多操作来减少这些在我们算法中的这些偏差.

Man:Woman as King:Queen

Man:Computer_Programmer as Woman:Homemaker

Father:Doctor as Mother:Nurse

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

图 21：The problem of bias in word embeddings

解决词偏差问题

Paper: Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings

1. Identify bias direction.
2. Neutralize: For every word that is not definitional, project to get rid of bias.
3. Equalize pairs.

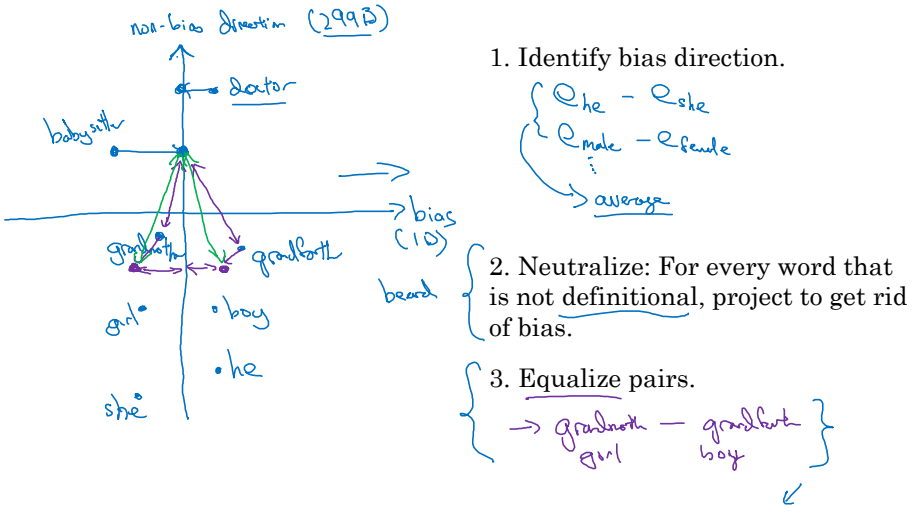


图 22：解决词偏差问题