北京航空航天大學
BEIHANG UNIVERSITY

计算语言学理论与实践

人工智能研究院

主讲教师 沙磊

主讲教师介绍

沙磊

主要研究方向: **可解释自然语言模型、可控自然语言生成、大规模信息抽取、AI4Science。**

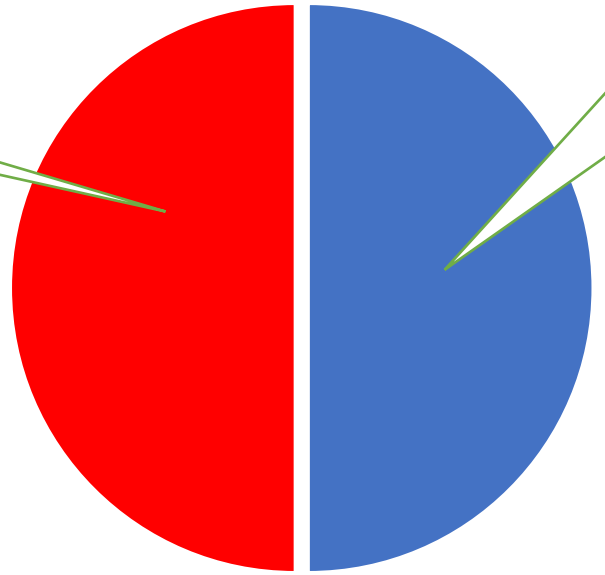
- 履历
  - 2023.2–今 北京航空航天大学 人工智能研究院 教授
  - 2022.4–2023. 2 北京航空航天大学 人工智能研究院 副教授
  - 2020.3-2022.3 牛津大学 智能系统组 Research Associate
  - 2018.8-2020.3 Apple Inc. Siri Understanding Senior NLP Research Scientist
  - 2013.9-2018.7 北京大学 计算语言所 博士

课程简介

- 32学时。助教：王昊 wanghao_ai@buaa.edu.cn
- 目的：（1）具备从头到尾实现NLP模型的能力
- （2）具备在工程上开发并部署NLP模型的知识

16学时：介绍NLP的基础知识以及实践方法

1. 词向量
2. Tree&CNN
3. Transformers
4. LLM
5. 增强记忆
6. 编辑网络
7. 大模型工具
8. AI4science
9. 可解释AI



16学时：由企业专家来介绍NLP在工业界的应用

1. Tencent -- Tianyu Liu
2. Bytedance – Shuangzhi Wu
3. Baidu Search– Xiaodong Zhang
4. MSRA news– Fangzhao Wu
5. Alibaba – Chen Shi
6. Fund – XX Yan

考核方式

- 平时成绩 50%
  - 一般情况满分，出现严重态度不端等情况会扣分
- 大作业 30%
- 期末笔试 20%

- 课程网站：
  - https://shalei120.github.io/course/NLP/CLTP.html

群聊：计算语言学理论与实践 2023秋

该二维码7天内(9月19日前)有效，重新进入将更新

相关学术期刊和会议

- 1. Computational Linguistics (ACL)
- 2. Transactions of the Association for Computational Linguistics (TACL)
- 3. 中文信息学报 (中文信息学会)
- 4. Annual Meeting of the Association for Computational Linguistics (ACL年会)
- 5. Conference on Empirical methods in natural language processing (EMNLP)
- 6, North American Chapter of the Association for Computational Linguistics (NAACL)
- 6. International Conference on Computational Linguistics(COLING)
- 7. 全国自然语言处理联合学术会议(CCL)
- 8, Natural Language Processing and Chinese Computing (NLPCC)
- 9, 机器学习会议：ICML，NeurIPS，ICLR

第一课

自然语言处理概述

Contents

- 词向量
- Machine Translation & Seq2seq
- Decoding
- Decoding: Advanced Topics

Contents

- 词向量
- Machine Translation & Seq2seq
- Decoding
- Decoding: Advanced Topics

How do we represent the meaning of a word?

定义：含义（meaning）

- 用一个词、词组等表示的概念
- 一个人想用语言、符号等来表达的想法
- 表达在作品、艺术等方面的思想

从**语言方式(linguistic way)**来看含义(meaning)：语言符号与语言意义（想法、事情）的相互对应

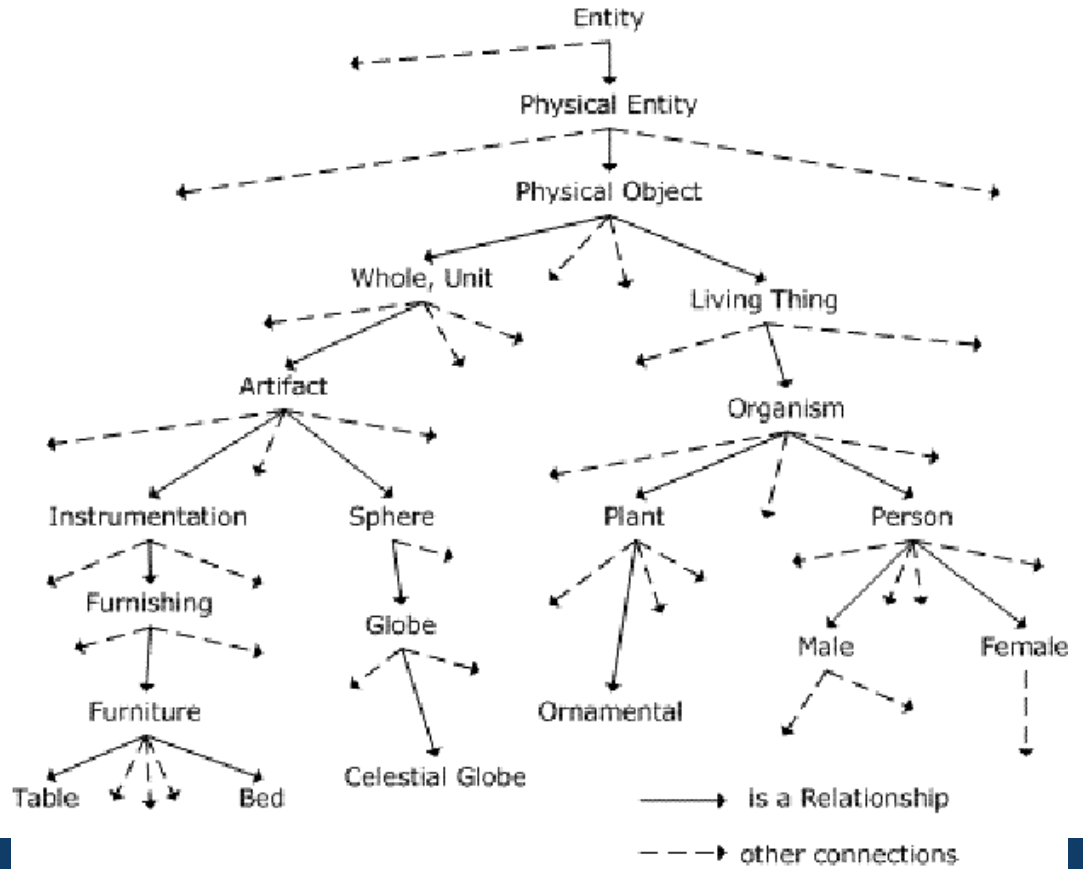
signifier (symbol) ⇔ signified (idea or thing)

= denotational semantics

tree ⇔ {🌳, 🌲, 🌴, …}

How do we have usable meaning in a computer?

- 要使用计算机处理文本词汇，一种处理方式是**WordNet**：即构建一个包含同义词(synonym)集和上位词(hypernym)(“is a”关系)的列表的辞典。



https://wordnet.princeton.edu/

WordNet?

e.g., synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems with resources like WordNet

- **忽略了词汇的细微差别**
  - 例如“proficient”被列为“good”的同义词。这只在某些上下文中是正确的。
- **缺少单词的新含义**
  - 难以持续更新！
  - 例如：wicked、badass、nifty、wizard、genius、ninja、bombast
- 因为是小部分专家构建的，有一定的**主观性**
- 构建与调整都需要很多的**人力成本**
- **无法定量计算出单词相似度**

Representing words as discrete symbols

- 在传统的自然语言处理中，我们会把词语看作离散的符号：例如hotel、conference、motel等。
- 一种文本的离散表示形式是把单词表征为one-hot 向量的形式
  - One-hot：只有一个1，其余均为0的稀疏向量
- 在one-hot向量表示中，向量维度 = 词汇量(如500,000)，以下为一些one-hot向量编码过后的单词向量示例：

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Problem with words as discrete symbols

- 例子：用户搜索“Seattle motel”, 我们想要找到包含“Seattle hotel”的文档,
- 然而：

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- 所有词向量是正交的。对于one-hot向量，没有关于相似性概念。
- **解决思路：**
  - ① 使用类似WordNet的工具中的同义词表？
    - 会因不够完整而失败
  - ② 通过大量数据学习词向量本身相似性，获得更精确的稠密词向量编码

Representing words by their conte

英国语言学家 J.R. Firth

- **分布式语义**：一个单词的意思是由经常出现在它附近的单词给出的
  - "You shall know a word by the company it keeps" (J. R. Firth 1957.11)
  - 这是现代统计NLP最成功的理念之一，总体思路有点物以类聚，人以群分的感觉
- 当一个单词w出现在文本中时，它的上下文是出现在其附近的一组单词(在一个固定大小的窗口中)
- 基于海量数据，使用w的许多上下文来构建w的表示
- 如图所示，banking的含义可以根据上下文的内容来表示

…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

These context words will represent ***banking***

词向量

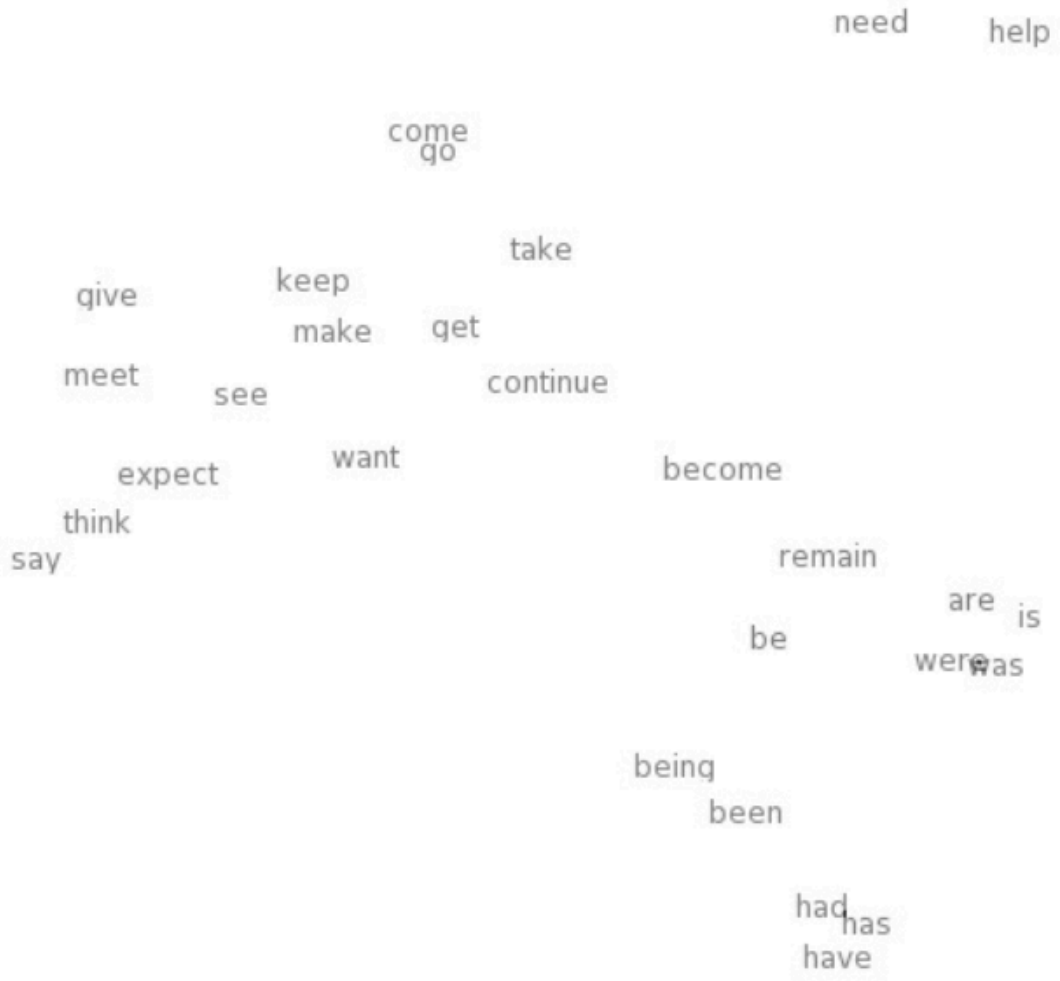
- 下面我们要介绍词向量的构建方法与思想，我们希望为每个单词构建一个稠密表示的向量，使其与出现在相似上下文中的单词向量相似。

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

- 词向量(word vectors)有时被称为词嵌入(word embeddings)或词表示(word representations)，是一种分布式表示(distributed representation)。

Word meaning as a neural word vector – visualization

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



· Word2vec: Overview

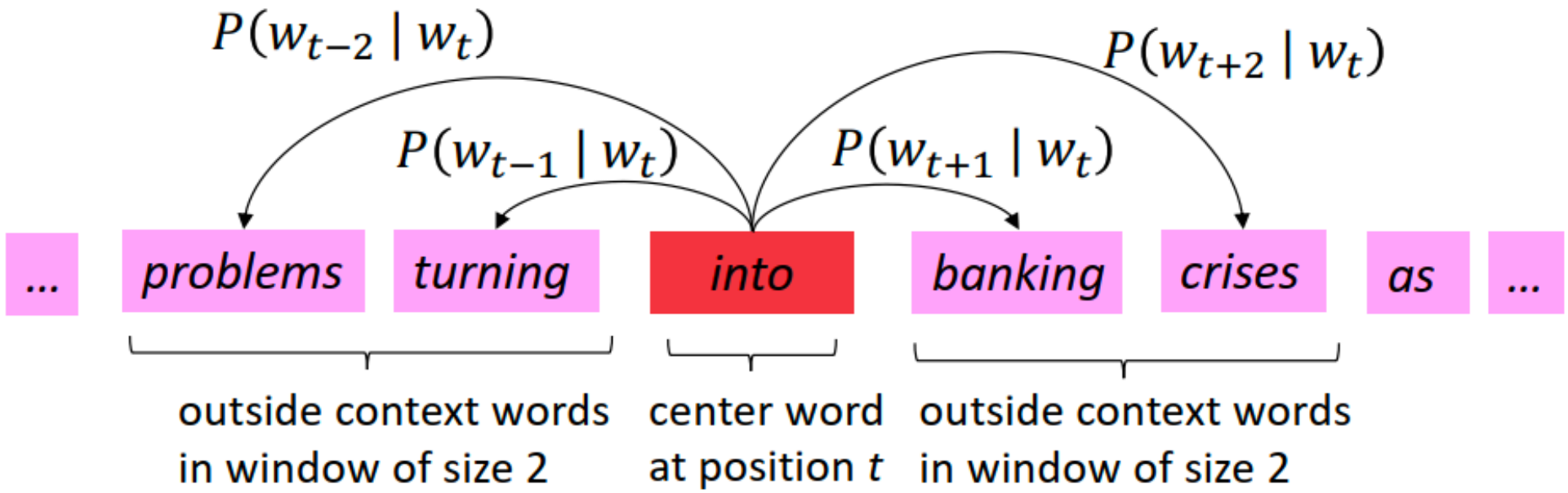
Word2vec (Mikolov et al. 2013)是一个学习词向量表征的框架。

Idea：

- We have a large corpus (“body”) of text： a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context (“outside”) words **o**
- Use the similarity of the word vectors for c and o to calculate the probability of o given c $p(o|c)$ (or vice versa $p(c|o)$)
- Keep adjusting the word vectors to maximize this probability

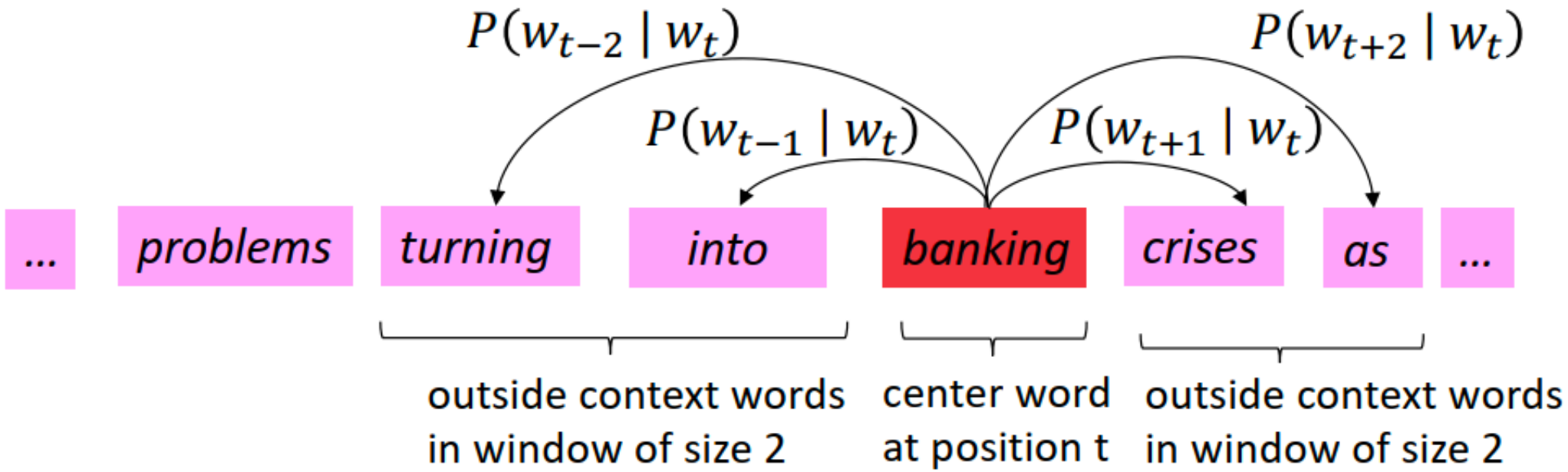
Word2vec: Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

- 对于每个位置 $t = 1, \ldots, T$，在大小为 m 的固定窗口内预测上下文单词，给定中心词 w_t，极大似然函数可以表示为：

$$Likelihoood = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ代表所有要优化的变量

- 对应上述似然函数的目标函数(objective/cost/loss function) $J(\theta)$可以取作(平均)负对数似然

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec: objective function

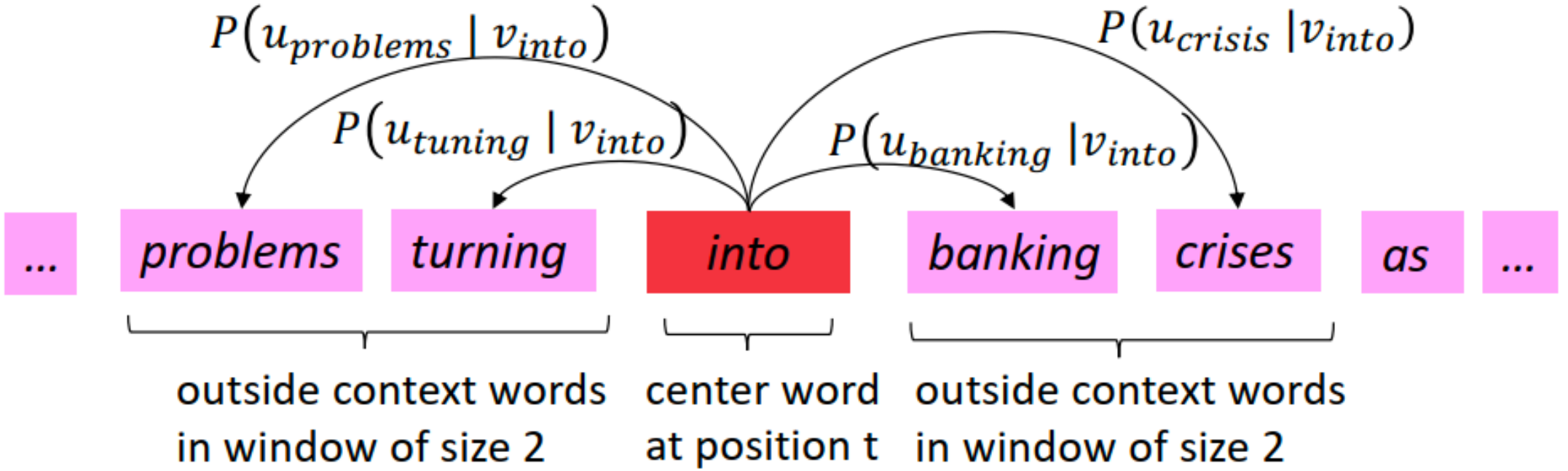
- We want to minimize the objective function

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- Question： How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- Answer： We will use two vectors per word w:
  - v_w when w is a center word
  - u_w when w is a context word
- Then for a center word c and a context word o： $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$

Word2Vec with Vectors

- 下图为计算 $P(w_{t+j} \mid w_t)$ 的示例，这里把 $P(problems \mid into; u_{problems}, v_{into}, \theta)$ 简写为 $P(u_{problems} \mid v_{into})$，例子中的上下文窗口大小2，即“左右2个单词+一个中心词”。



Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

③ Normalize over entire vocabulary to give probability distribution

- Softmax 函数举例： $\mathbb{R}^n \to (0,1)^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

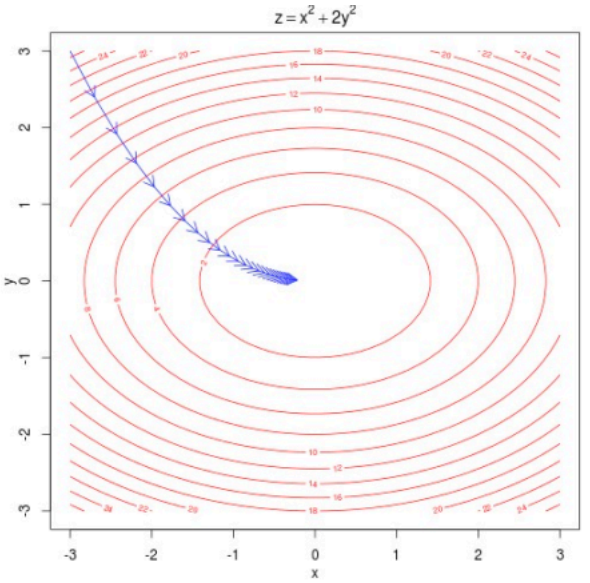
- Softmax函数将任意实数向量x_i (i=1,...,n)映射为一个概率分布p_i (i=1,...,n)
  - Max：最大元素拥有最大的概率值
  - Soft：小的元素依然拥有概率值，不会是0
  - 在深度学习中普遍应用

To train the model: Optimize value of parameters to minimize loss

- 模型训练过程中，我们需要逐步调整参数来最小化loss

- θ代表所有模型的参数
- 此处，我们每个词向量长度为d，一共V个词，那么我们有→
- 每个词有两个向量

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- 整个优化过程顺着梯度的方向下降，直到找到最低点
- 所有向量的梯度都要计算！

具体的loss function求梯度的流程

$$\min J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0}\log p(w'_{t+j}|w_t)$$

负对数似然

文本长度

窗口长度

此处：

$$p(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w=1}^{V}\exp(u_w^\top v_c)}$$

接下来，看看梯度怎么求。

具体的loss function求梯度的流程

$$\frac{\partial}{\partial v_c}\log P(o|c) = \frac{\partial}{\partial v_c}\log\frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

此处是对向量求偏导

$$= \frac{\partial}{\partial v_c}\left(\log\exp(u_o^T v_c) - \log\sum_{w\in V}\exp(u_w^T v_c)\right)$$

$$= \frac{\partial}{\partial v_c}\left(u_o^T v_c - \log\sum_{w\in V}\exp(u_w^T v_c)\right)$$

$$= u_o - \frac{\sum_{w\in V}\exp(u_w^T v_c)u_w}{\sum_{w\in V}\exp(u_w^T v_c)}$$

$$= u_o - \sum_{w\in V}\frac{\exp(u_w^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}u_w$$

Observed - expected

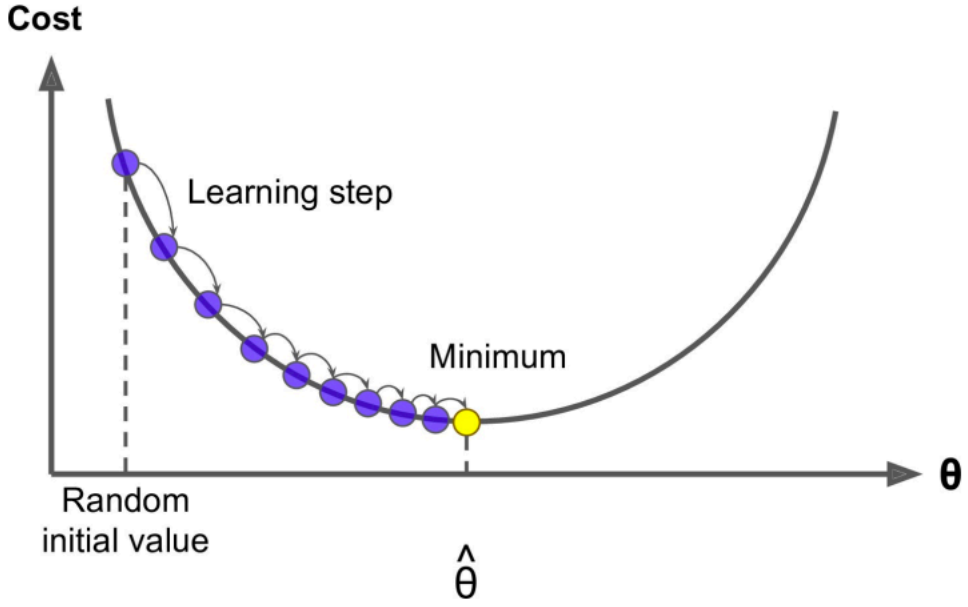
$$= u_o - \sum_{w\in V}P(w|c)u_w$$

把所有上下文向量用其概率加权求和

- 这一段是对中心向量v_c的梯度求法
- 输出向量u_o的求法类似

Optimization: Gradient Descent

- 我们现在需要最小化
- 利用梯度下降法
- 基本思路：对于当前的参数θ，计算 $J(\theta)$ 的梯度$J(\theta)$,然后在负梯度的方向不断地小步迭代



Note: Our objectives may not be convex like this ☹

But life turns out to be okay ☺

Gradient Descent

- 更新方程

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

α = *step size* or *learning rate*

- Python代码

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Stochastic Gradient Descent

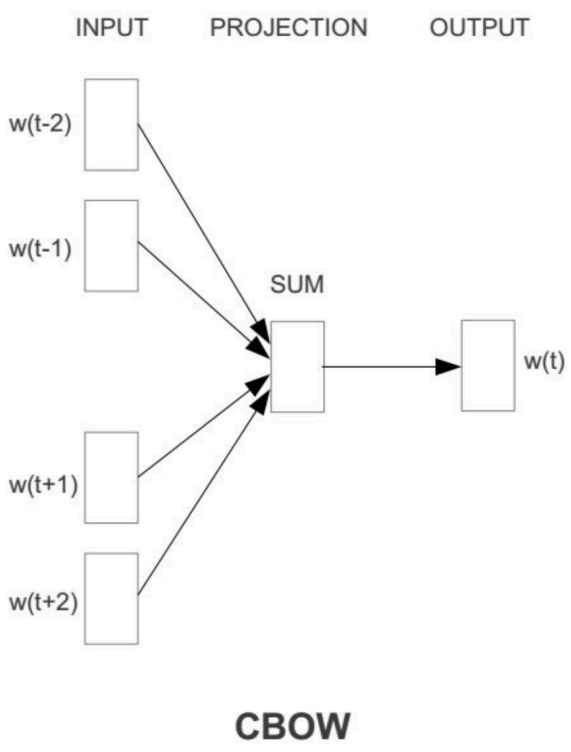
- 不足之处 ： $J(\theta)$ 是语料库中所有窗口的函数（上亿）
  - $\nabla J(\theta)$ 的计算需要花费大量资源

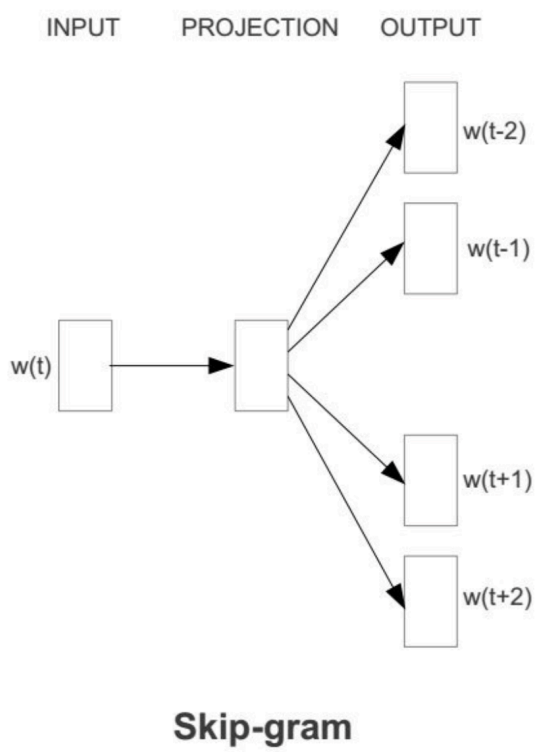
- 对于任意神经网络都不是好主意
- 解决方案： Stochastic gradient descent (SGD) 随机梯度下降
  - 不断的采样窗口，每采样一次做一次更新
- 代码示例

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Word2vec algorithm family: More details

- 为什么每个单词一定要对应两个向量？
  - 最终两个向量的平均值代表词向量
  - 实际操作中，每个词只对应一个向量。。。而且效果更好一些。
- 其他可能的模型结构
  - Skip-gram：利用中心词预测上下文词 $p(o|c)$
  - Continuous Bag of Words(CBOW)：利用上下文词预测中心词 $p(c|o)$





我们刚刚讲过的是skip-gram model

Training efficiency

- 为了提高训练效率
  - Hierarchical Softmax （在算力足够的条件下不实用，略）
  - Negative sampling
- Why？归一化项的计算太费资源 $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

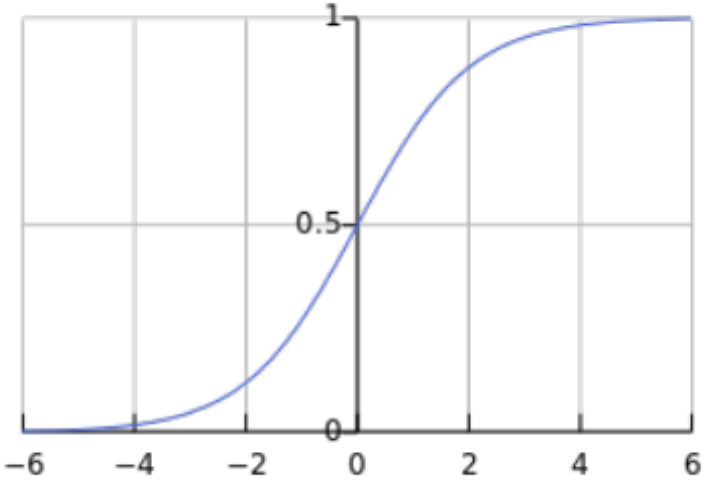
- 所以在Word2vec的标准实现中用的是负采样方法
- 主要思想：使用一个 true pair (中心词及其上下文窗口中的词)与几个 noise pair (中心词与随机词搭配) 形成的样本，训练二元逻辑回归。

The skip-gram model with negative sampling

- From paper： “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- 目标函数（最大化） $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

- Sigmoid函数 $\sigma(x) = \frac{1}{1+e^{-x}}$
- 我们要最大化2个词共现的概率，
  最小化与噪音词的共现概率



The skip-gram model with negative sampling

- 实际操作中：

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log\sigma(\boldsymbol{u}_o^T\boldsymbol{v}_c) - \sum_{k\in\{K\ sampled\ indices\}} \log\sigma(-\boldsymbol{u}_k^T\boldsymbol{v}_c)$$

- 我们取 k 个负例采样
- 最大化出现在窗口中的“中心词”之外的词语出现的概率，而最小化其他没有出现在窗口中的随机词的概率
- 对词进行采样的时候使用概率分布：$P(w) = U(w)^{3/4}/Z$ ，其中U()代表一元的分布（w出现的概率）
- 3/4 次幂减少了单词出现频率之间的差异，从而提高了低频词被抽中的概率

Why not capture co-occurrence counts directly?

- 与其一遍一遍的遍历语料库优化模型，为什么不直接统计一下单词与附近单词的共现情况呢？

- 建立共现矩阵X：两种形式：window和全文档
  - Window：与word2vec类似，在每个单词周围都使用Window，捕捉一些语法和语义信息
  - Word-document：共现矩阵的基本假设是在同一篇文章中出现的单词更有可能相互关联。假设单词 i 出现在文章 j 中，则矩阵元素 X_{ij} 加一，当我们处理完数据库中的所有文章后，就得到了矩阵 X，其大小为 $|V| \times M$，其中 |V| 为词汇量，而 M 为文章数。这一构建单词文章 co-occurrence matrix的方法也是经典的Latent Semantic Analysis（LSA）所采用的

GloVe

Encoding meaning components in vector differences [Pennington, Socher, and Manning, EMNLP 2014]

Contents

- 词向量
- Machine Translation & Seq2seq
- Decoding
- Decoding: Advanced Topics

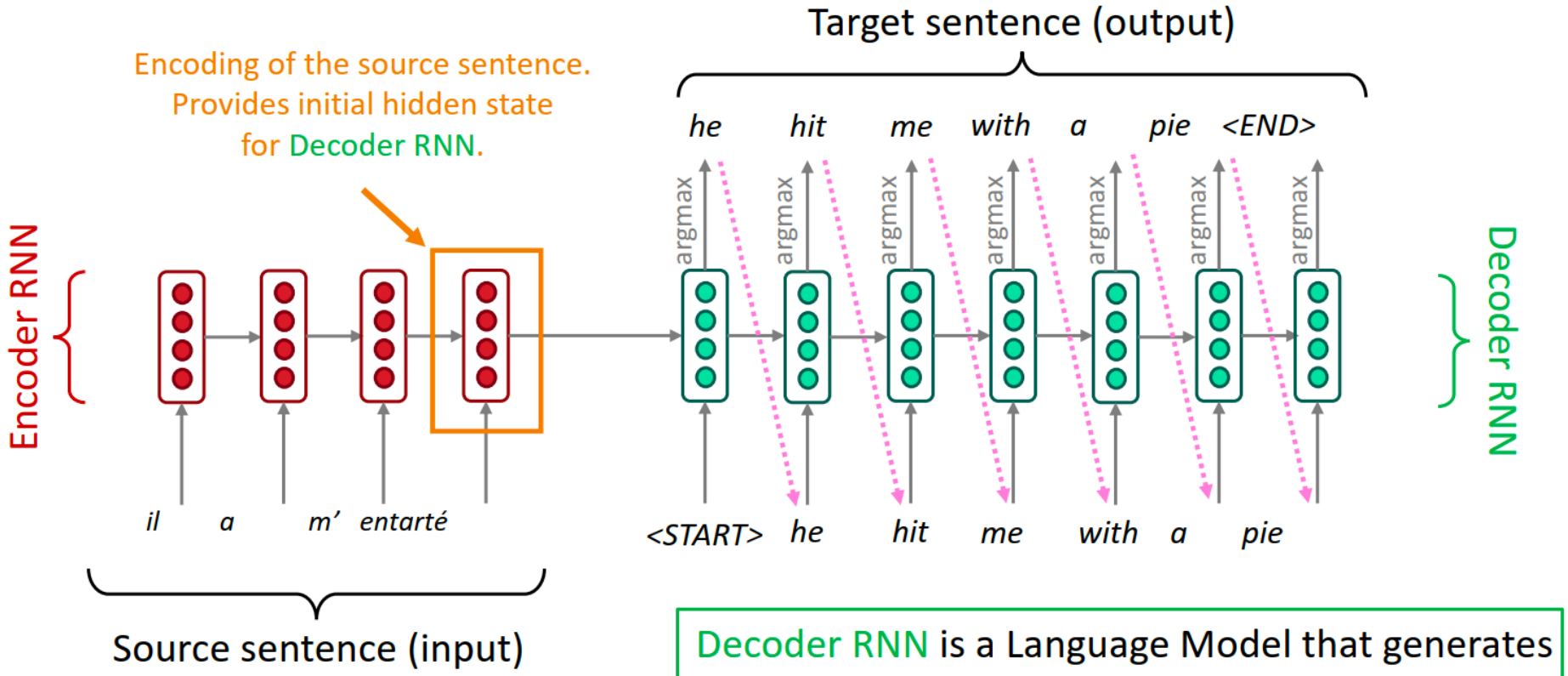
What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a single end-to-end neural network

- The neural network architecture is called a sequence-to-sequence model (aka seq2seq) and it involves two RNNs

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ⋯→ as next step's input

Sequence-to-sequence is versatile!

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq mode
- Sequence-to-sequence is useful for more than just MT
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

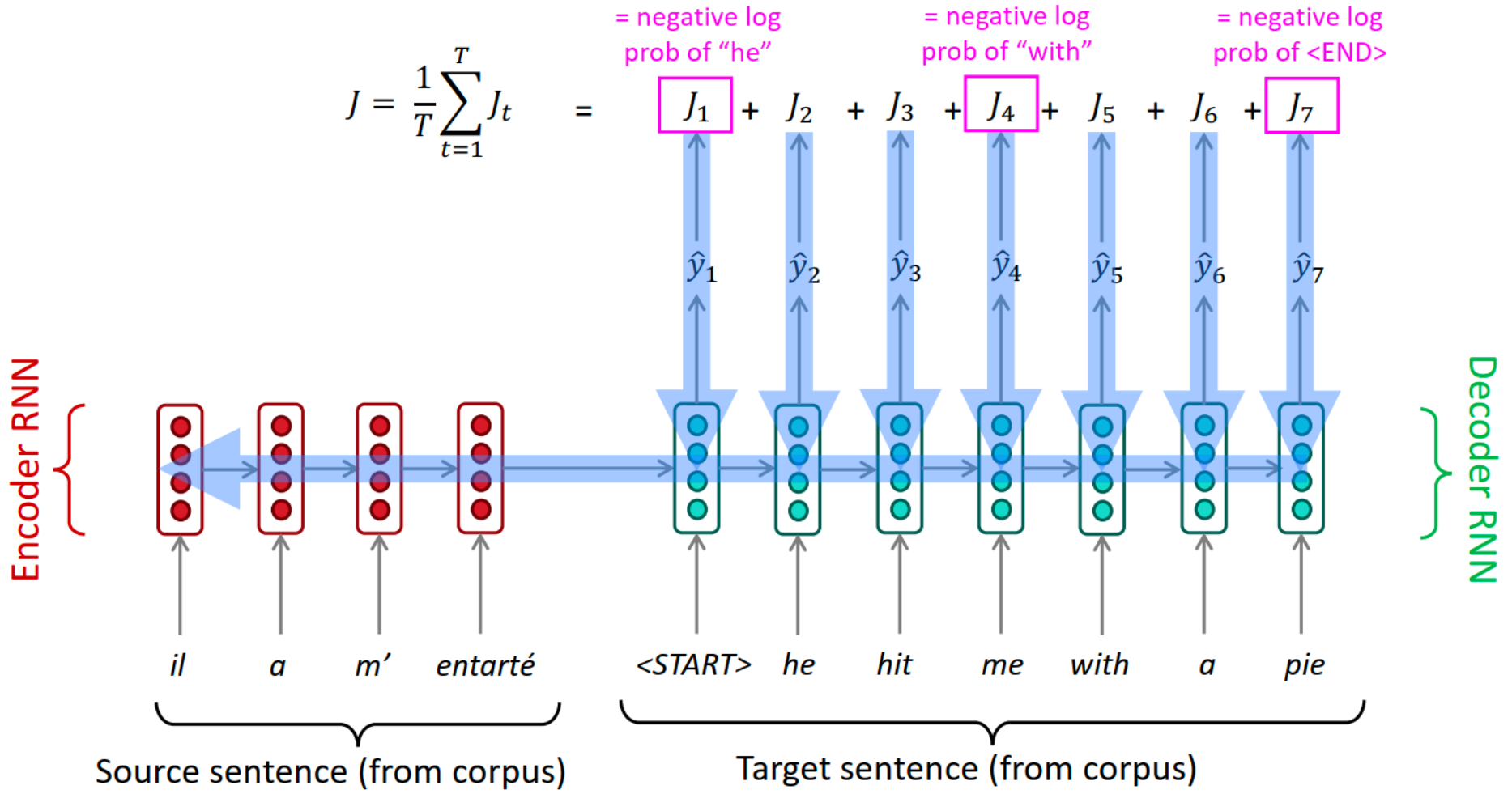
- The sequence-to-sequence model is an example of a Conditional Language Model
  - Language Model because the decoder is predicting the next word of the target sentence y
  - Conditional because its predictions are also conditioned on the source sentence x
- NMT directly calculates $P(y|x)$

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots\underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}$$

Probability of next target word, given target words so far and source sentence *x*

- **Question**: How to train an NMT system?
- **(Easy) Answer**: Get a big parallel corpus…
  - But there is now exciting work on "unsupervised NMT", data augmentation, etc

Training a Neural Machine Translation system

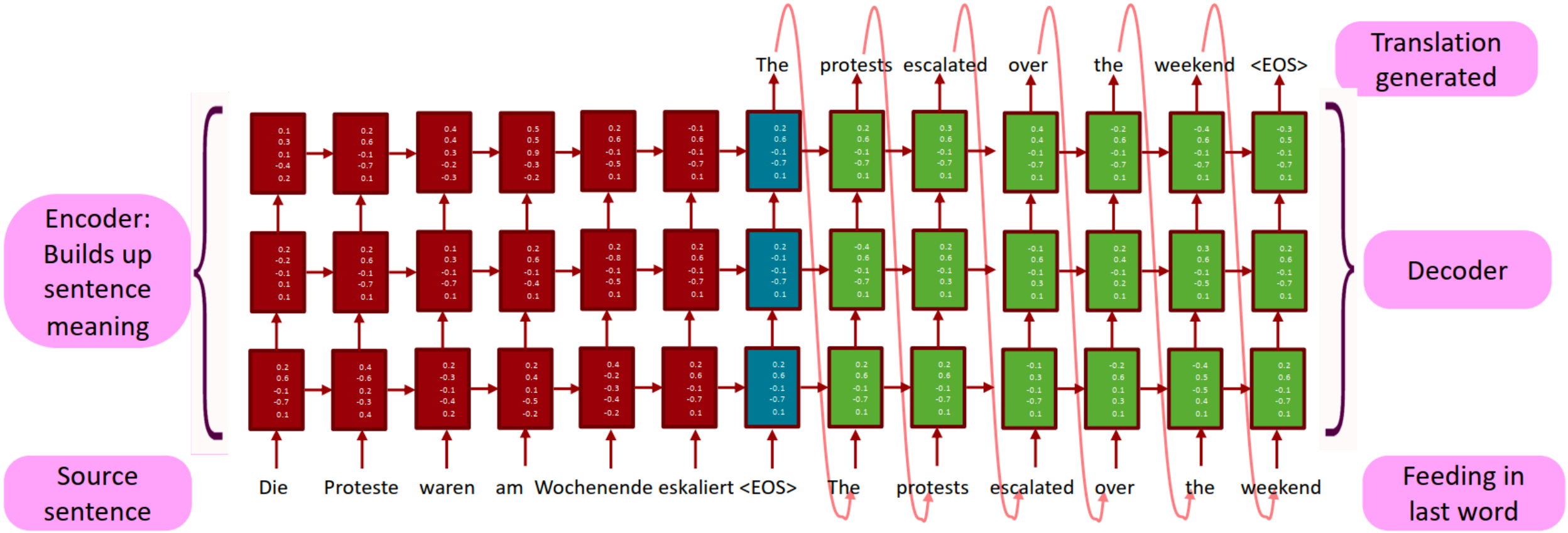


Seq2seq is optimized as a **single system.** Backpropagation operates *"end-to-end"*.

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1

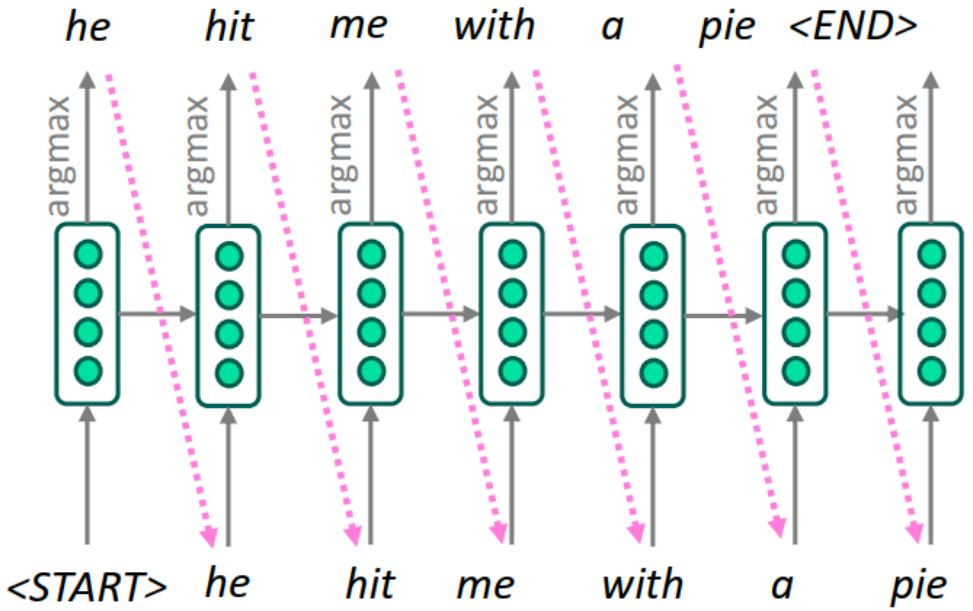


Contents

- 词向量
- Machine Translation & Seq2seq
- Decoding
- Decoding: Advanced Topics

Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)
- Problems with this method?

Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: il a m'entarté (he hit me with a pie)
  - → he ____
  - → he hit ____
  - → he hit a ____ (whoops! no going back now…)

- How to fix this?

Exhaustive search decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$

$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences y
  - This means that on each step t of the decoder, we're tracking V^T possible partial translations, where V is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Beam search decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam search decoding: example

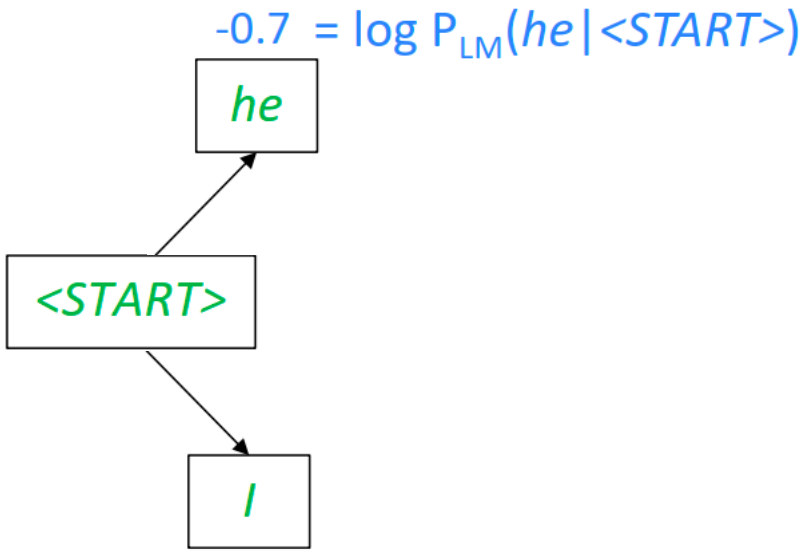
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam search decoding: example

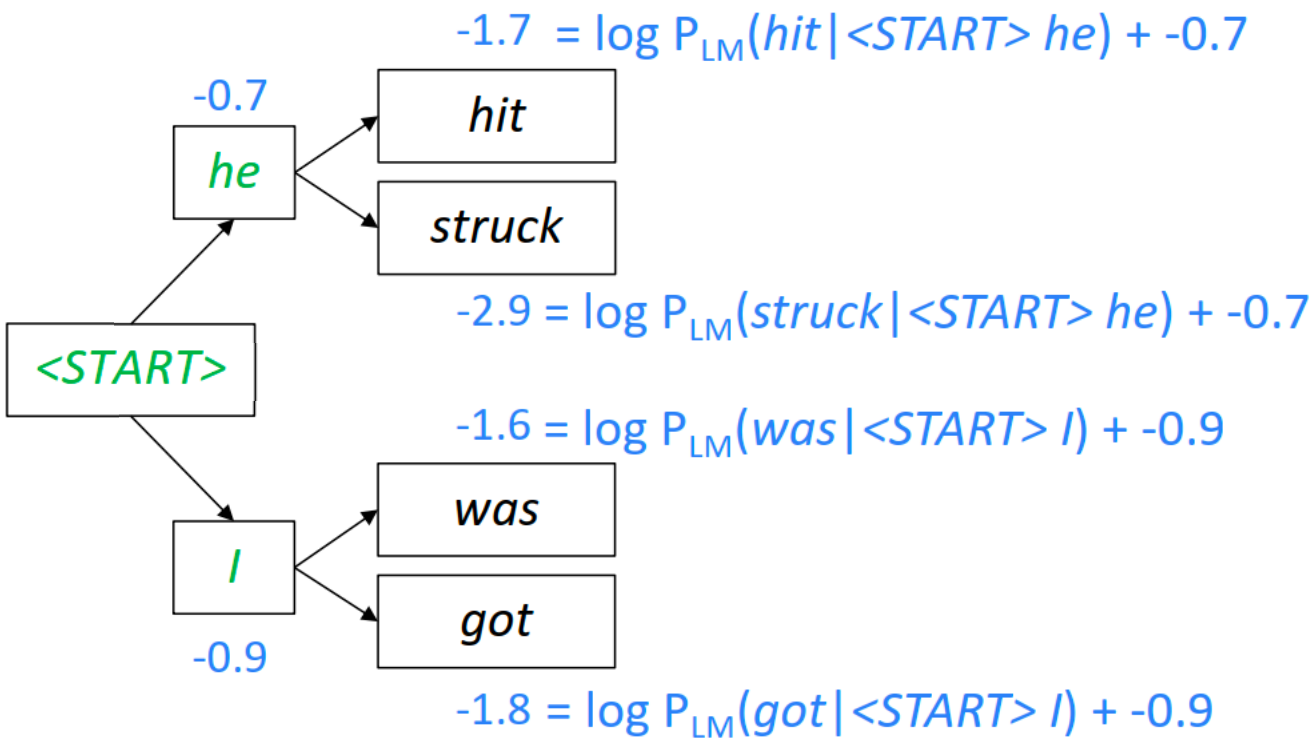
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

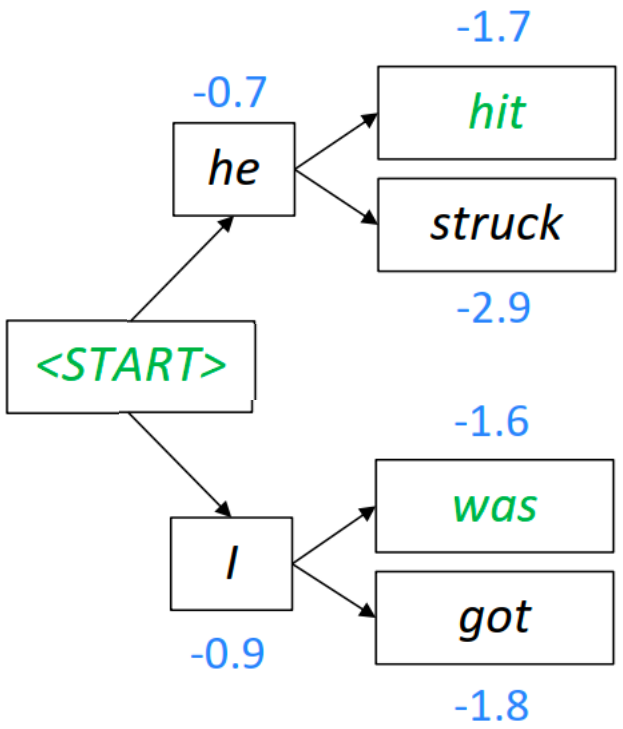
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

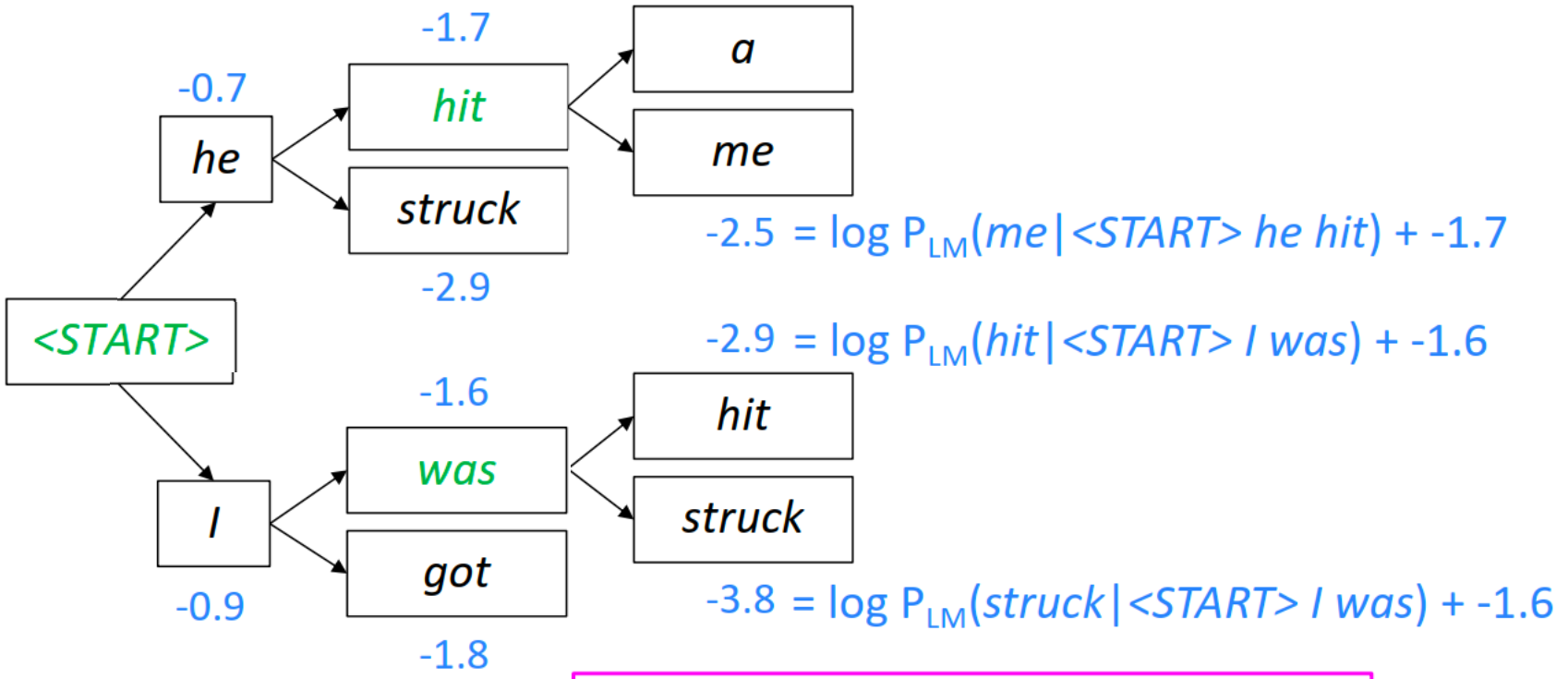
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

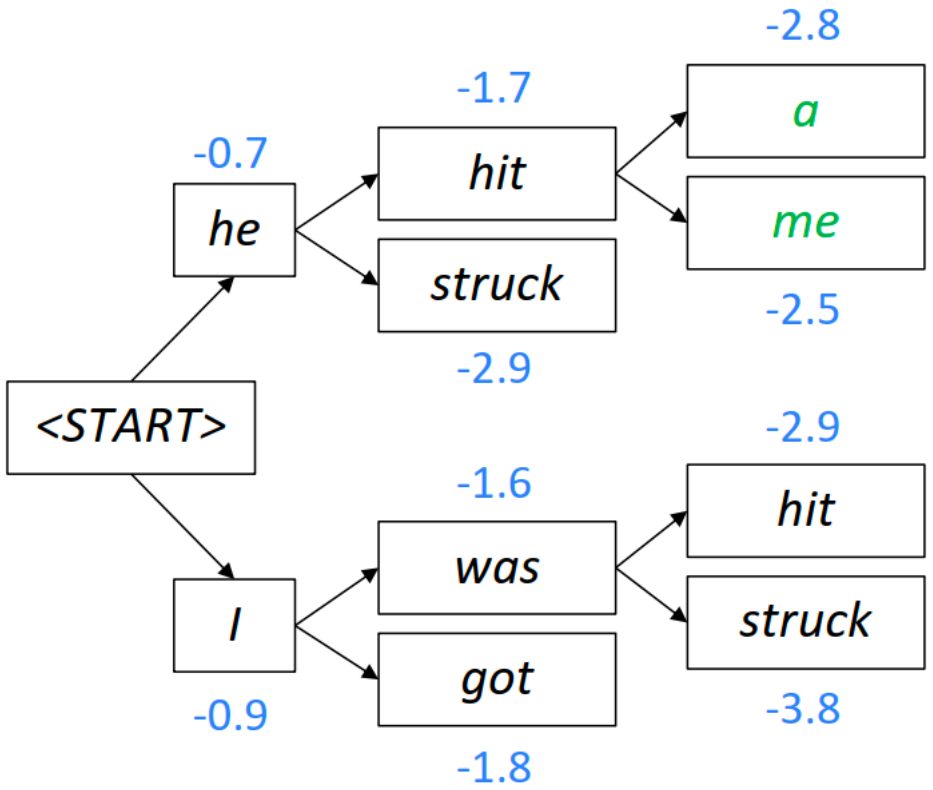
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

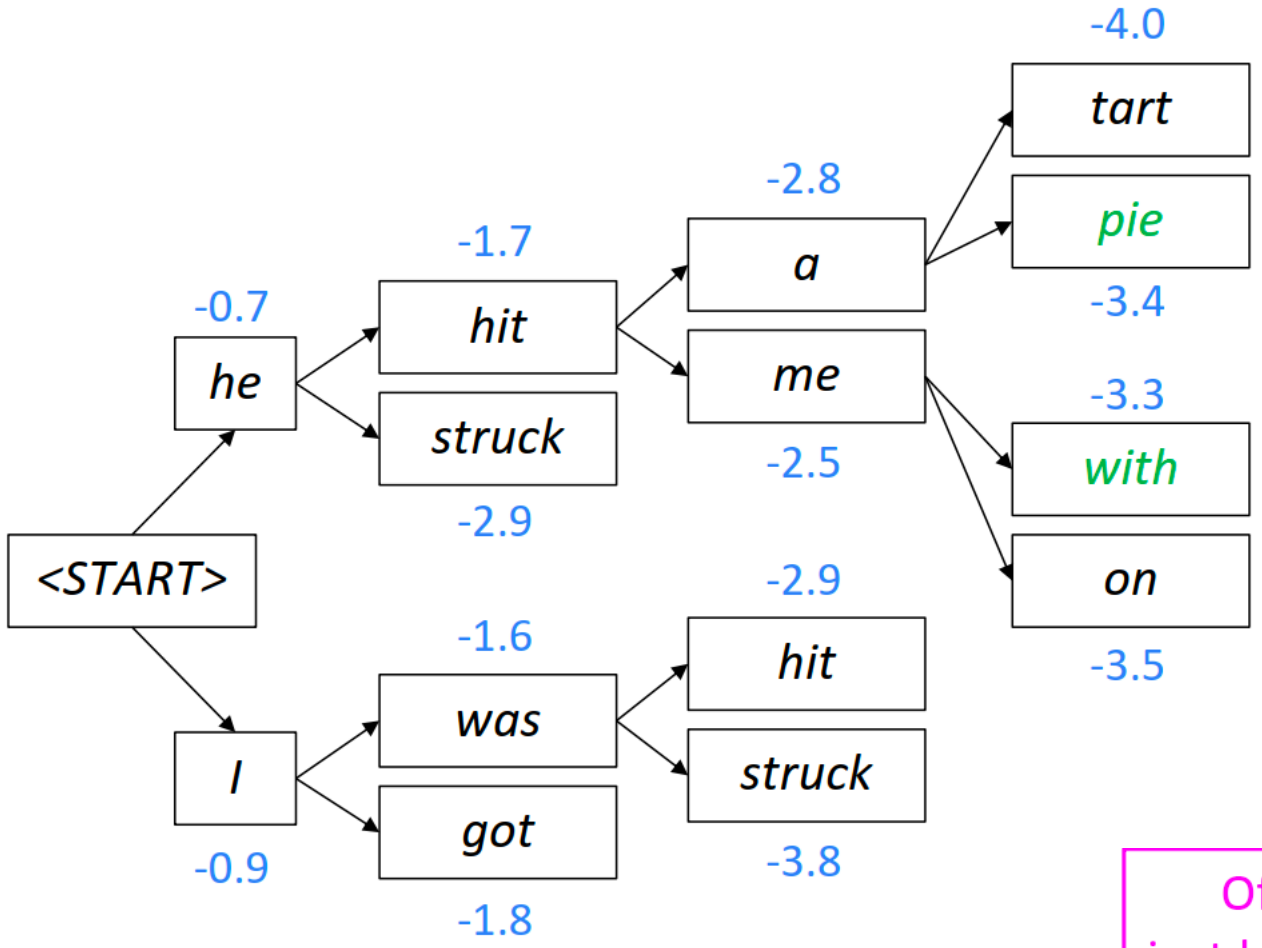
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

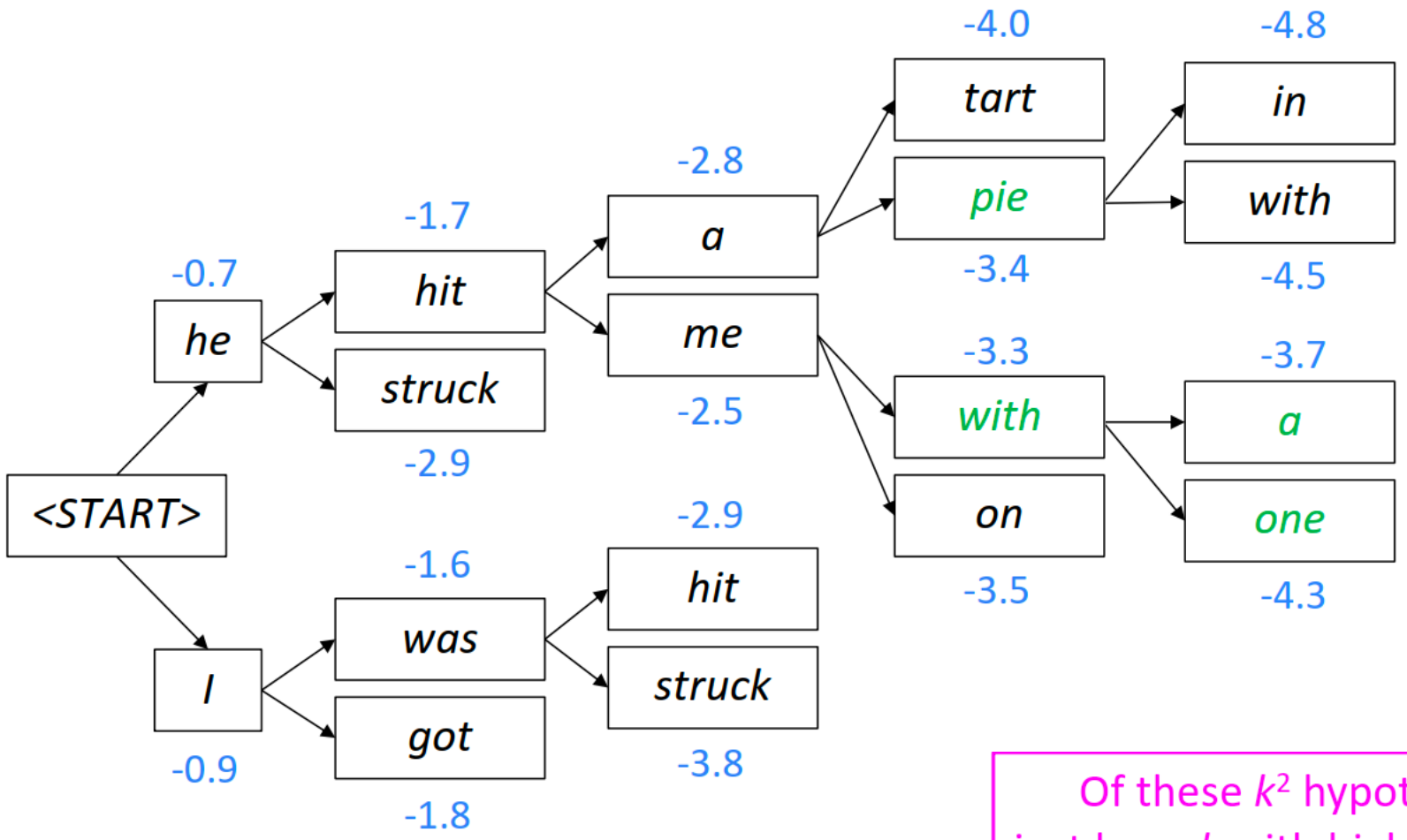
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

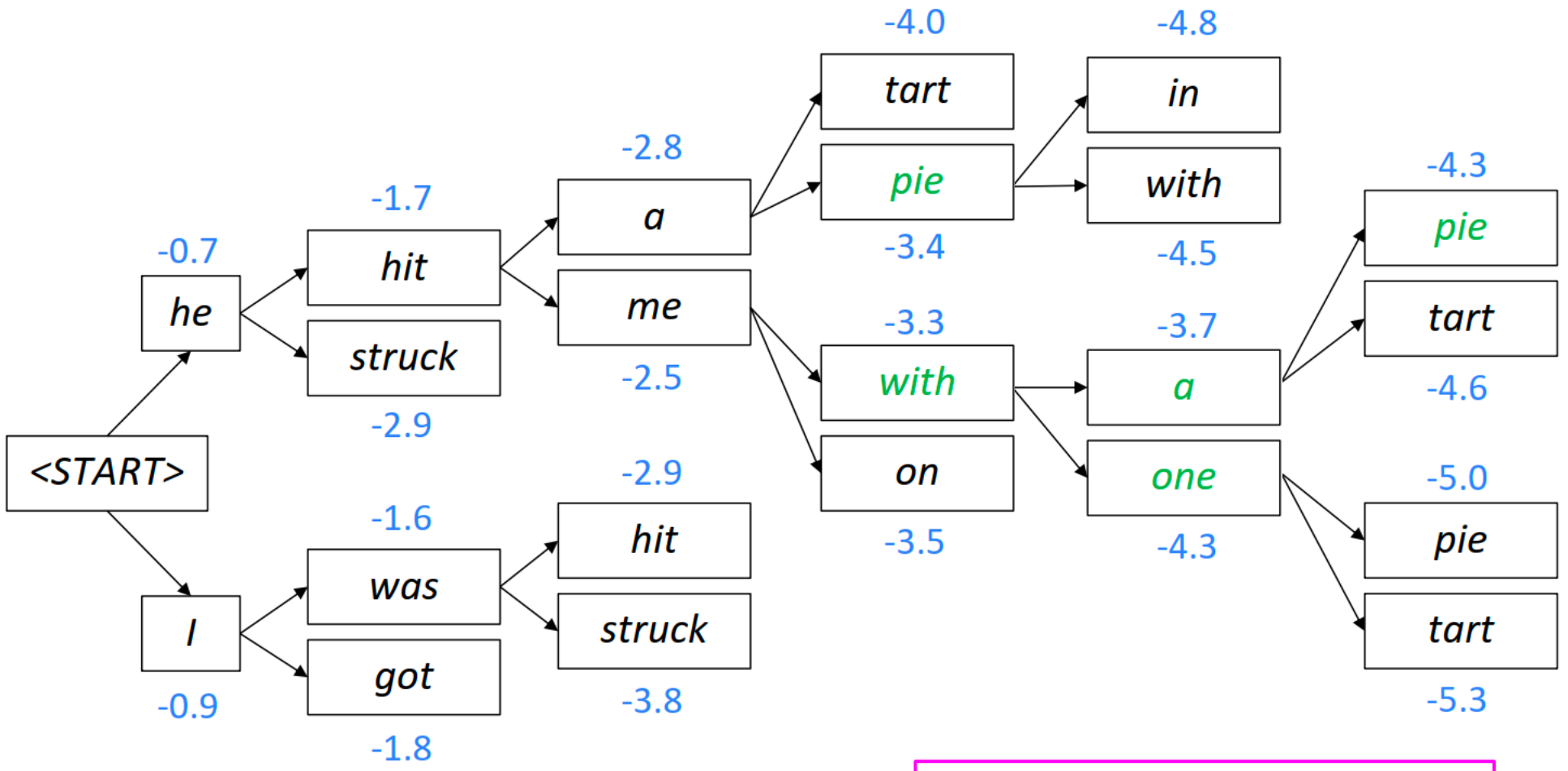
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

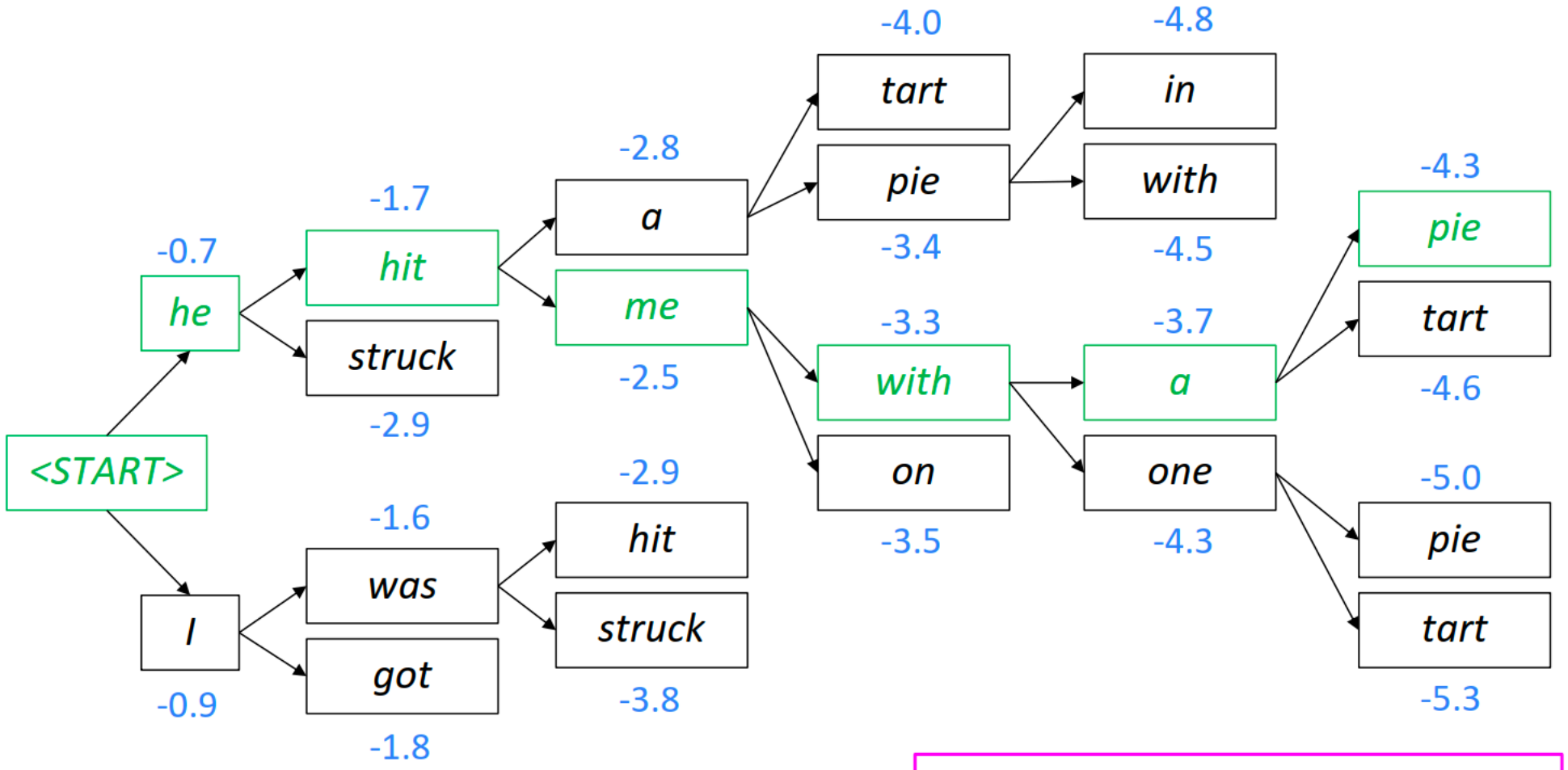
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces an <END> token
  - **For example**: <START> he hit me with a pie <END>

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- **Problem with this**: longer hypotheses have lower scores
- **Fix**: Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

Contents

- 词向量
- Machine Translation & Seq2seq
- Decoding
- Decoding: Advanced Topics

Decoding: what is it all about?

- 在每个时间步t，模型计算一个分数向量 $S \in \mathbb{R}^V$，里面针对此表中的每个单词都计算了一个分数

$$S = f(\{y_{<t}\})$$

$f(.)$ is your model

- 然后，用这些分数计算一个概率分布P（通常使用softmax）

$$P(y_t = w|\{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- 解码模型定义一个函数g，从这个分布中选择一个单词

$$\hat{y}_t = g(P(y_t|\{y_{<t}\}))$$

$g(.)$ is your decoding algorithm

Greedy methods

- Argmax Decoding
  - 选择此分布中 $P(y_t|y_{<t})$ 最高概率的词

$$\hat{y}_t = \underset{w \in V}{\mathbf{argmax}}\, P(y_t = w|y_{<t})$$

- Beam Search
  - 在介绍机器翻译介绍过
  - 核心也是贪心算法，但探索候选的范围更广

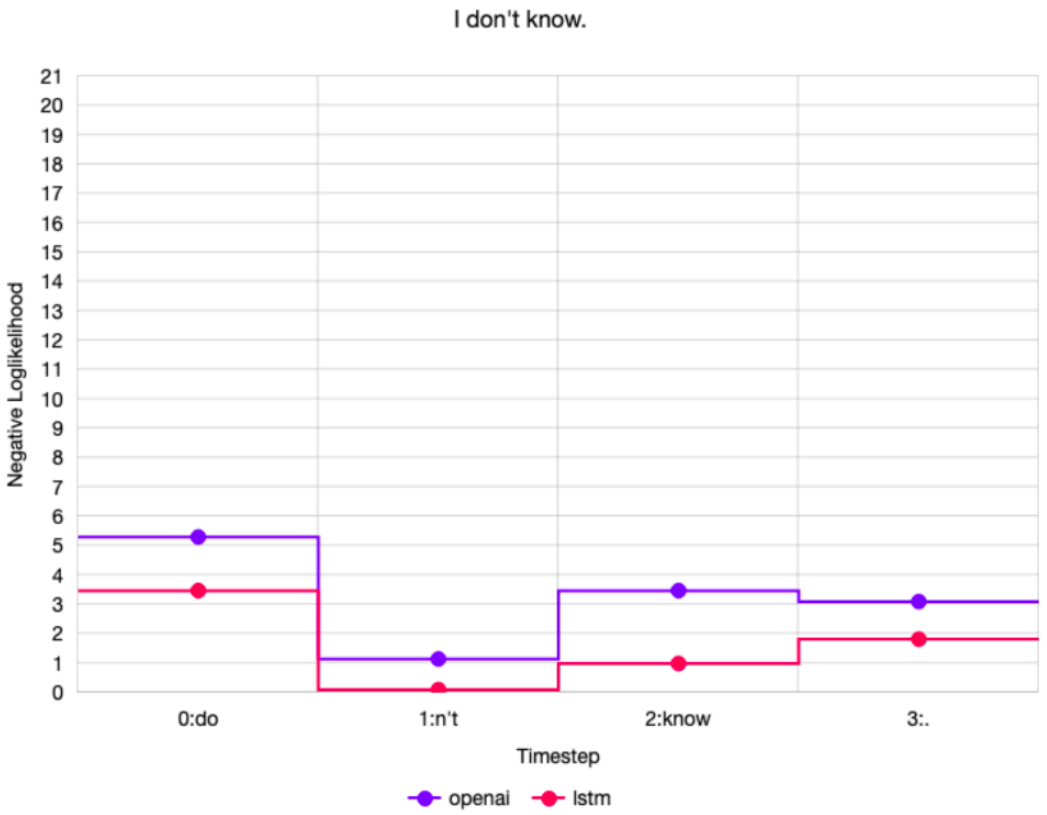
Greedy methods get repetitive

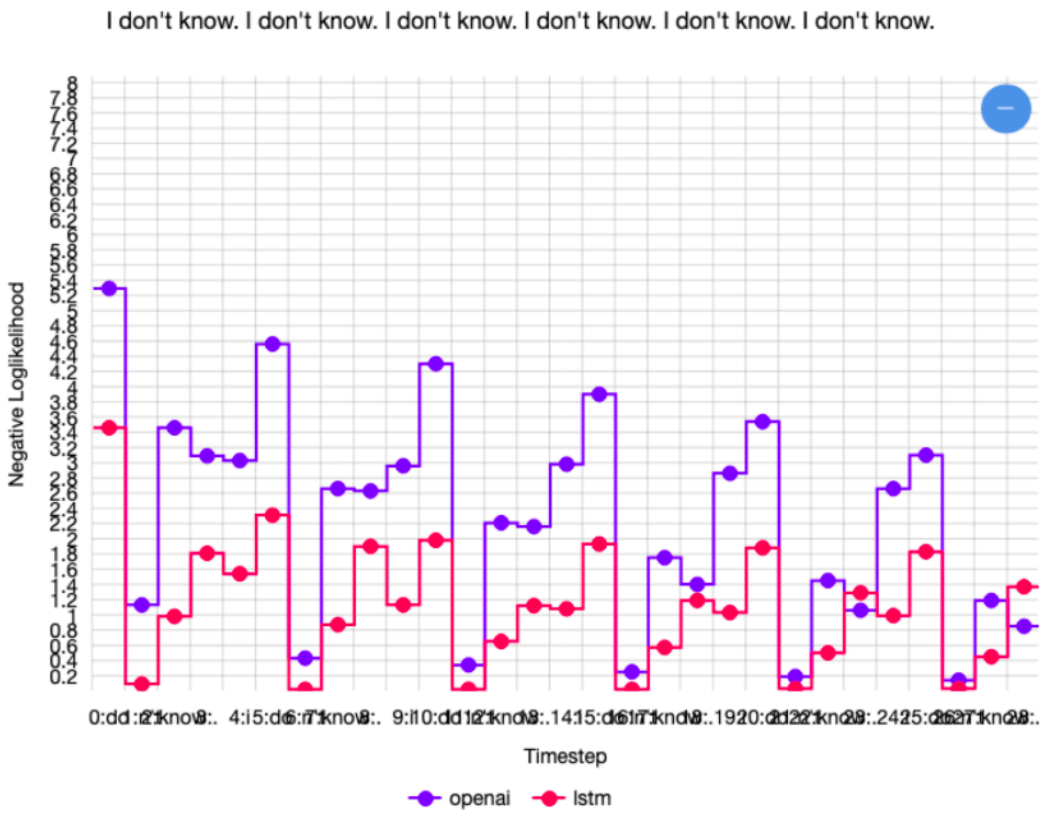
Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México…

(Holtzman et. al., ICLR 2020)

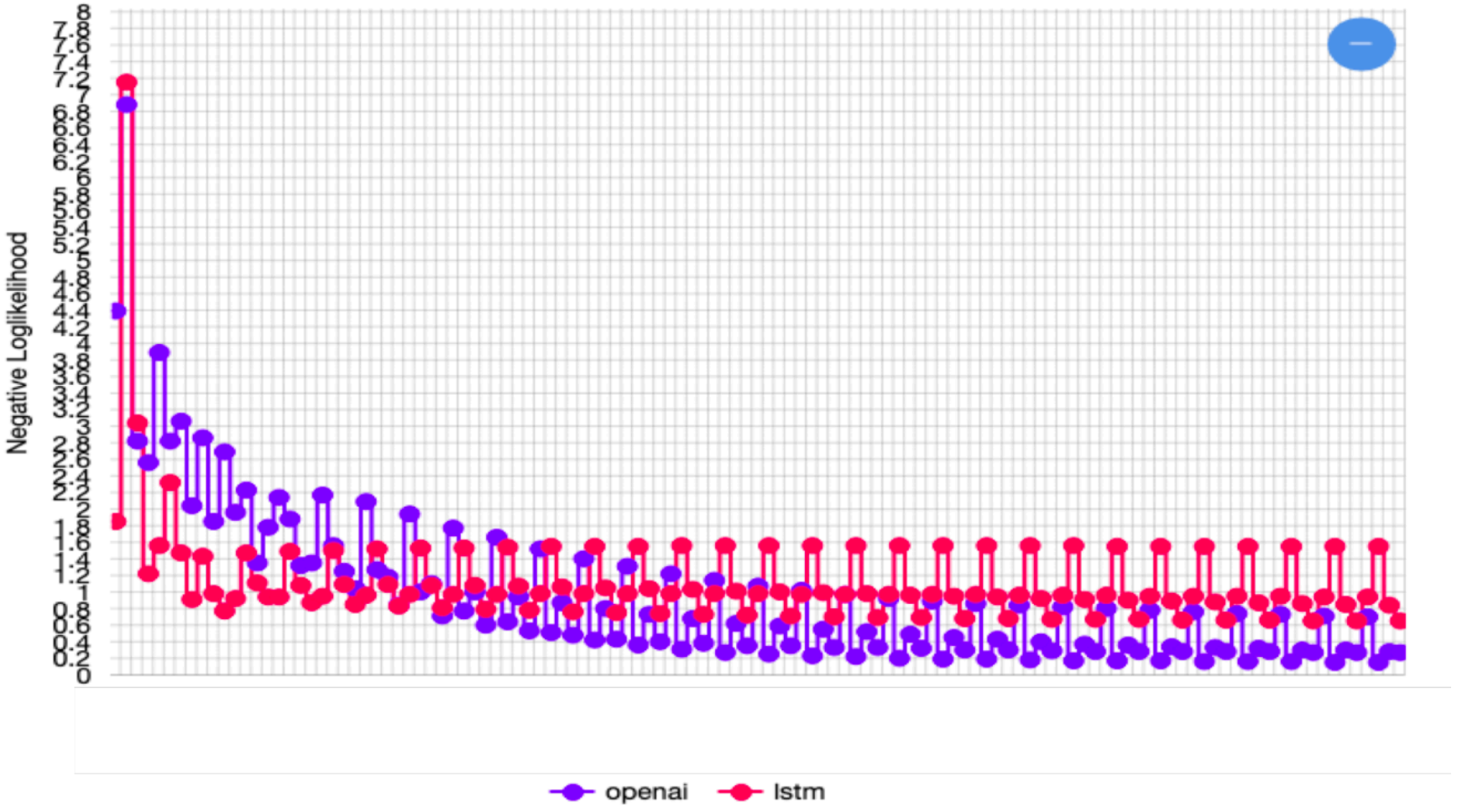
Why does repetition happen?





And it keeps going…

I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired.



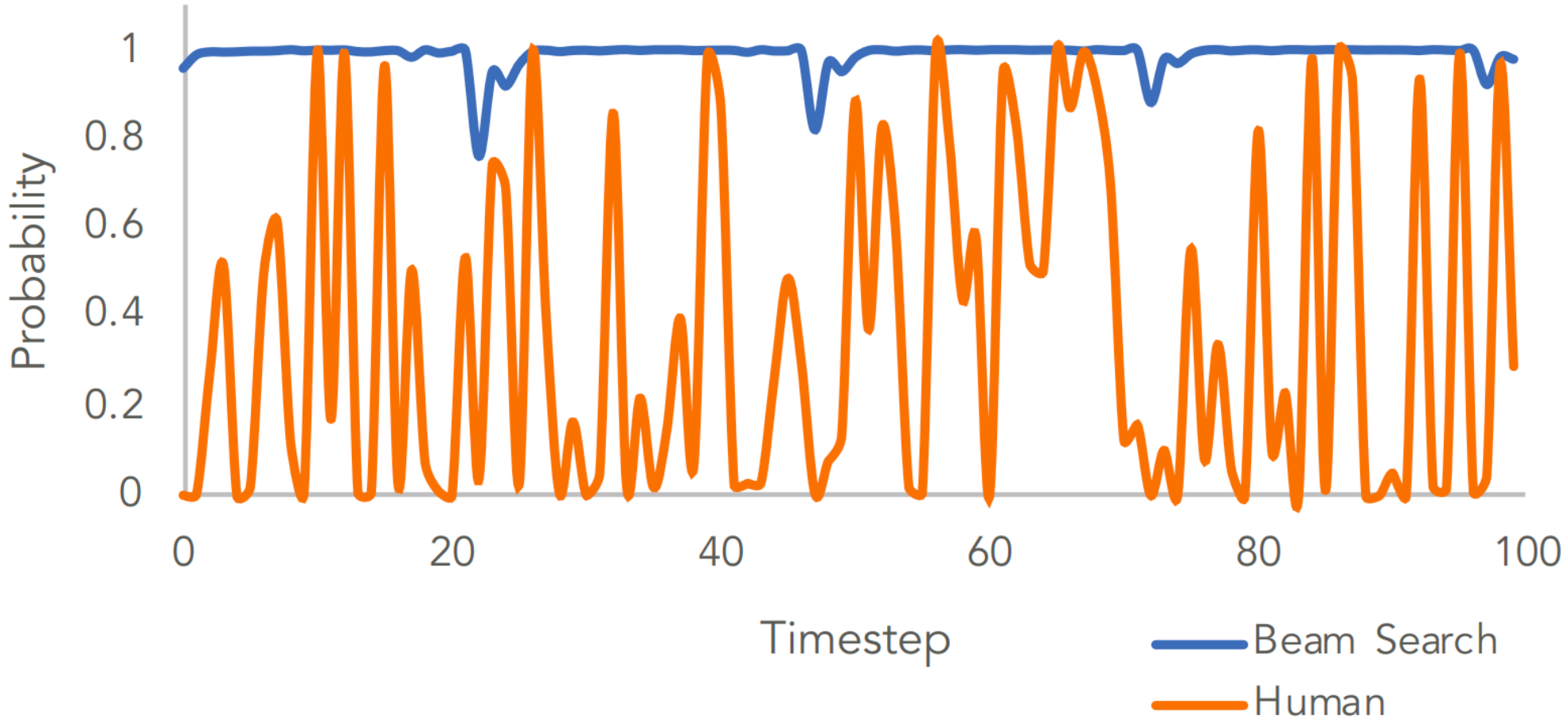
(Holtzman et. al., ICLR 2020)

How can we reduce repetition?

- 简单的选择：
  - 启发式方法：不要重复，直接截断

- 复杂一些：
  - 最大化相邻句子间表示向量的距离(Celikyilmaz et al., 2018)
    - 对句子内的重复无效
  - Coverage loss(See et al, 2017)
    - 避免attention关注到相同的单词
  - Unlikelihood objective (Welleck et al., 2020)
    - 生成已出现过的单词会给与惩罚

Are greedy methods reasonable?



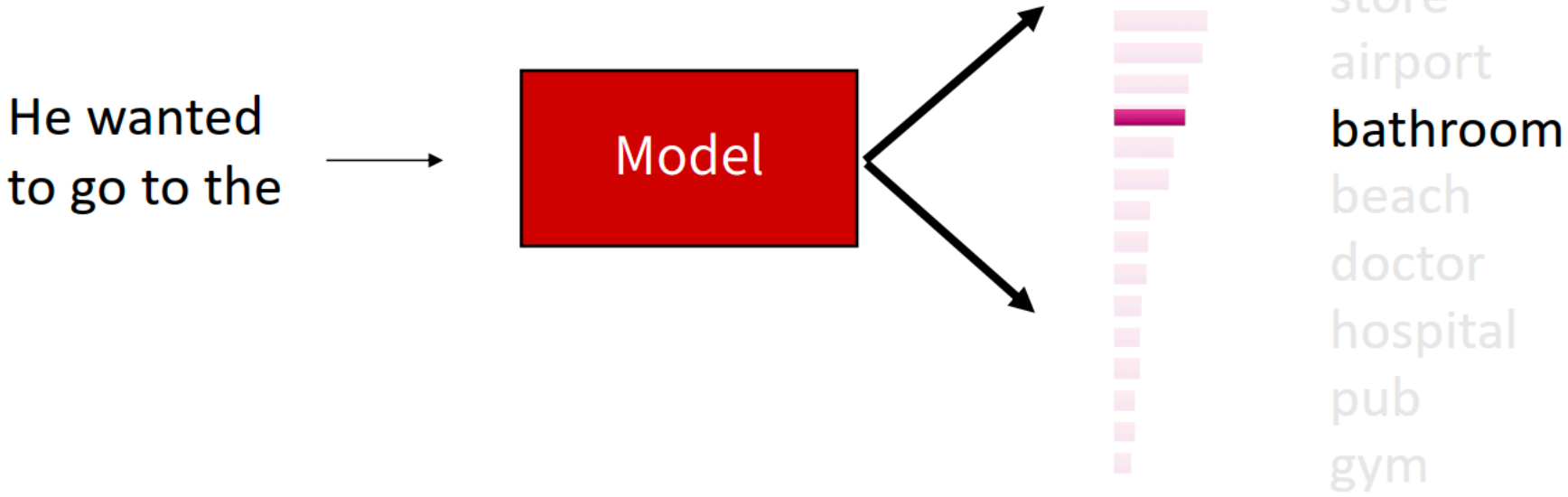
(Holtzman et. al., ICLR 2020)

Time to get *random* : Sampling!

- 从单词的概率分布中采样单词

$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

- 由于是随机的，所以你可以采样到任何单词

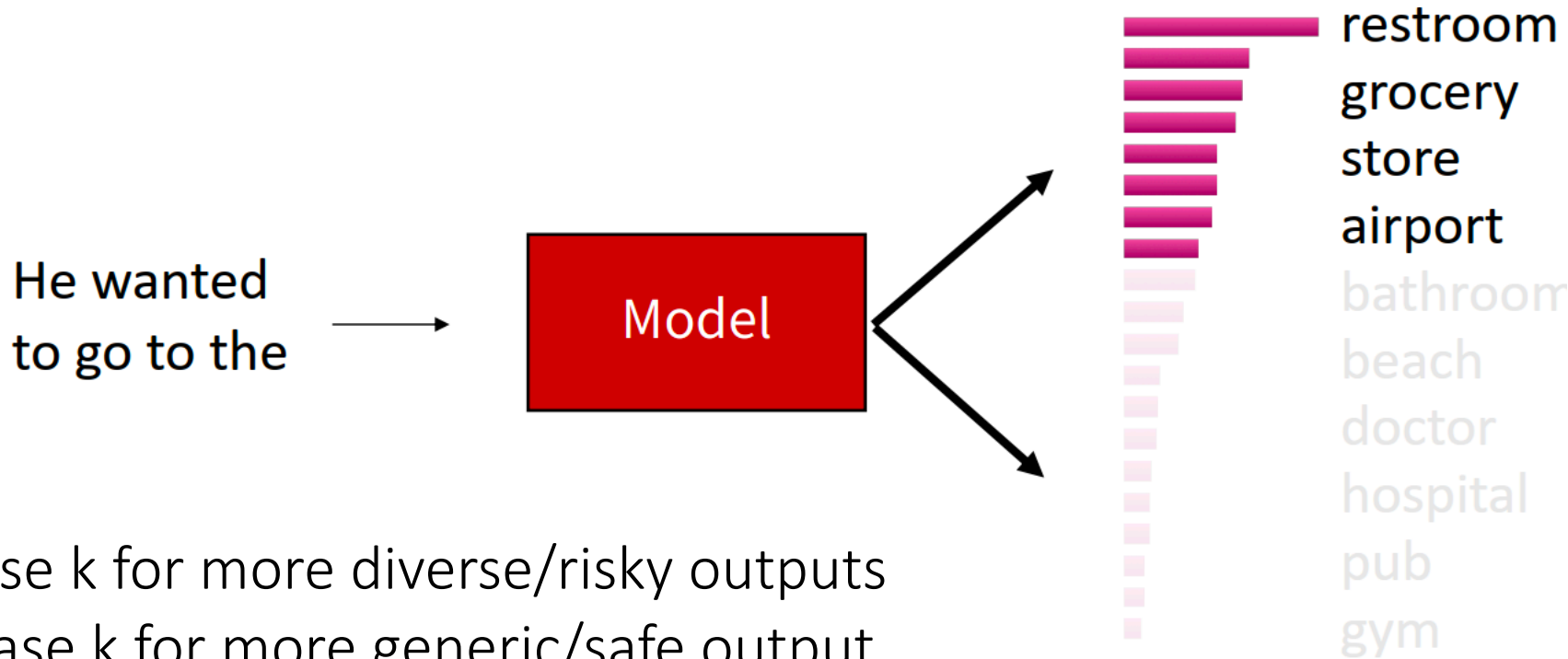


Decoding: Top-k sampling

- 问题：普通采样使得每个单词都有可能被选中
  - 长尾分布
  - 一些明显错误的单词也会占据一些小概率值
  - 多个明显错误的单词的概率总和很大
- 解决方案：Top-k采样
  - 只从概率最高的k个单词中间采样

Decoding: Top-k sampling

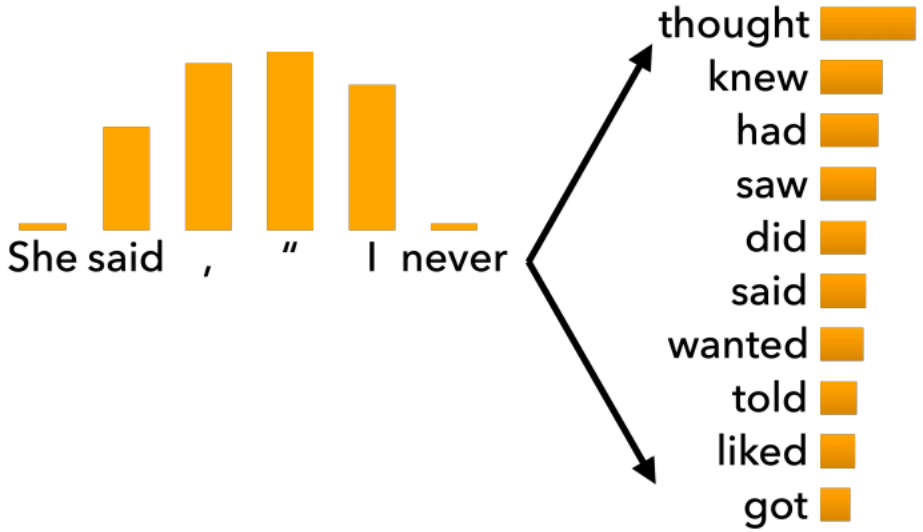
- 一般取值 k = 5, 10, 20



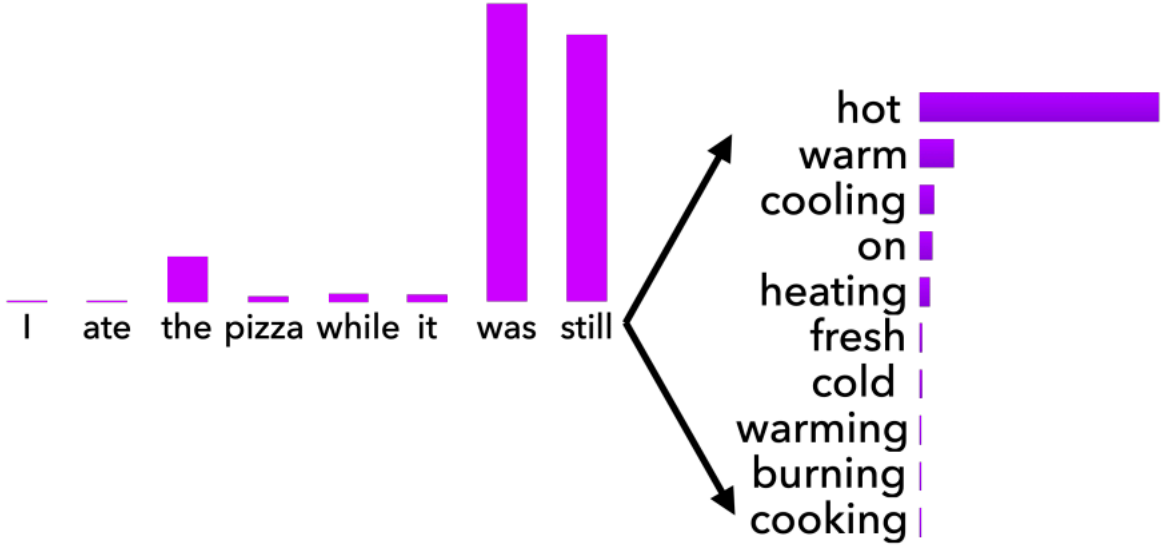
- Increase k for more diverse/risky outputs
- Decrease k for more generic/safe output

Issues with Top-k sampling

(Holtzman et. al., ICLR 2020)



Top-*k* sampling can cut off too ***quickly***!



Top-*k* sampling can also cut off too ***slowly***!

Decoding: Top-p (nucleus) sampling

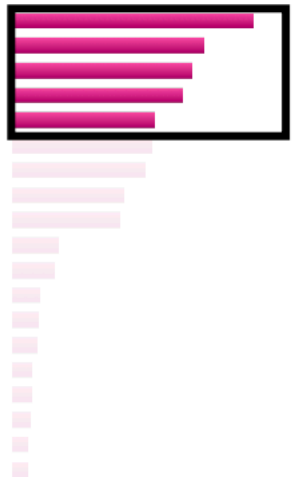
- 问题：用来采样的概率分布是动态的
  - 更平缓的分布：k可能会排除掉一些可能的选项
  - 有尖峰的分布：k可能使一些不好的选项拥有可能性
- 解决方案：Top-p采样
  - 预定义一个阈值p
  - 采样范围从概率最高的单词开始，按顺序向下，直到采样范围内单词的概率总和超过p为止 $\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p.$
  - 计算Top-p之后的概率分布，并进行采样 $p' = \sum_{x \in V^{(p)}} P(x|x_{1:i-1})$

$$P'(x|x_{1:i-1}) = \begin{cases} P(x|x_{1:i-1})/p' & \text{if } x \in V^{(p)} \\ 0 & \text{otherwise.} \end{cases}$$

Decoding: Top-p (nucleus) sampling

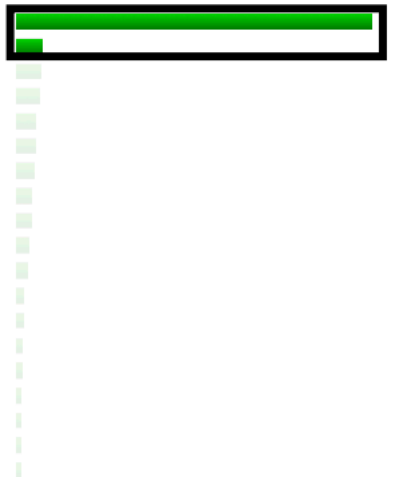
- k根据概率分布的不同而不同

$P_t^1(y_t = w \mid \{y\}_{<t})$



$P_t^2(y_t = w \mid \{y\}_{<t})$

$P_t^3(y_t = w \mid \{y\}_{<t})$



Scaling randomness: Softmax temperature

- 回忆Softmax： $P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$

- 我们可以在softmax中加一个温度变量来调整分布

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

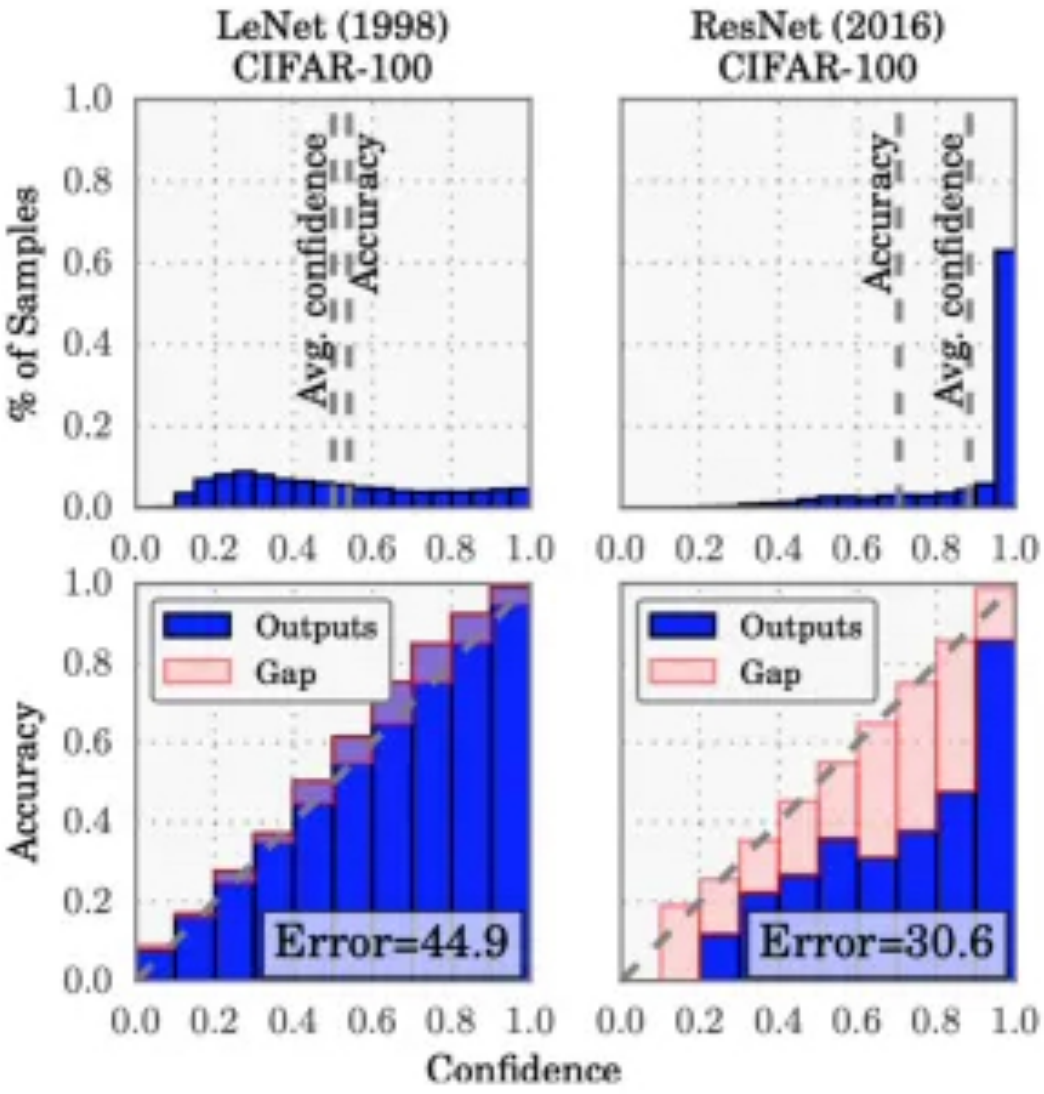
- 提升 $\tau > 1$,分布会变得更平缓
  - 输出会更加多样化
- 降低 $\tau < 1$,分布会变的更尖锐
  - 输出的多样化性降低

Improving decoding: Re-ranking

- 问题：如果解码出很糟糕的序列怎么办？
- 解决方案：解码出很多句子
  - 一般准备10个候选
- 定义一个分数来估计句子的质量，来重新排序
  - 最简单的是用perplexity!
    - 但不要用重复的模型
- Re-rankers 可以为很多属性进行评分:
  - style (Holtzman et al., 2018), discourse (Gabriel et al., 2021), entailment/factuality (Goyal et al., 2020, logical consistency (Lu et al., 2020), and many more ...
  - 小心没有对齐过的（poorly-calibrated ）re-rankers
- 可以并行使用多个reranker

Calibration?

- 模型给出预测的时候，我们希望模型给出预测的概率与真实概率尽可能一致。
  - 例如，我们把某模型预报明天下雨的概率为 80% 的历史上所有的案例都放在一起，然后后验地统计真正下雨的比例，我们希望这个比例就差不多是 80%；如果是这样，我们就认为这个模型是校准的（calibrated）。



Calibration References

- 1. Guo, Chuan, et al. "On calibration of modern neural networks." arXiv preprint arXiv:1706.04599 (2017).
- 2. Kuleshov, Volodymyr, Nathan Fenner, and Stefano Ermon. "Accurate uncertainties for deep learning using calibrated regression." arXiv preprint arXiv:1807.00263 (2018).
- 3. Kull, Meelis, et al. "Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration." Advances in neural information processing systems 32 (2019): 12316-12326.
- 4. Kumar, Ananya, Percy S. Liang, and Tengyu Ma. "Verified uncertainty calibration." Advances in Neural Information Processing Systems. 2019.
- 5. Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." international conference on machine learning. 2016.
- 6. Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." Advances in neural information processing systems 30 (2017): 6402-6413.
- 7. Shift, Evaluating Predictive Uncertainty Under Dataset. "Can you trust your model’s uncertainty?."

Conclusion

- Word vector
- Seq2seq
- Generation

- Next lesson:
  - Tree & CNN

Thank you