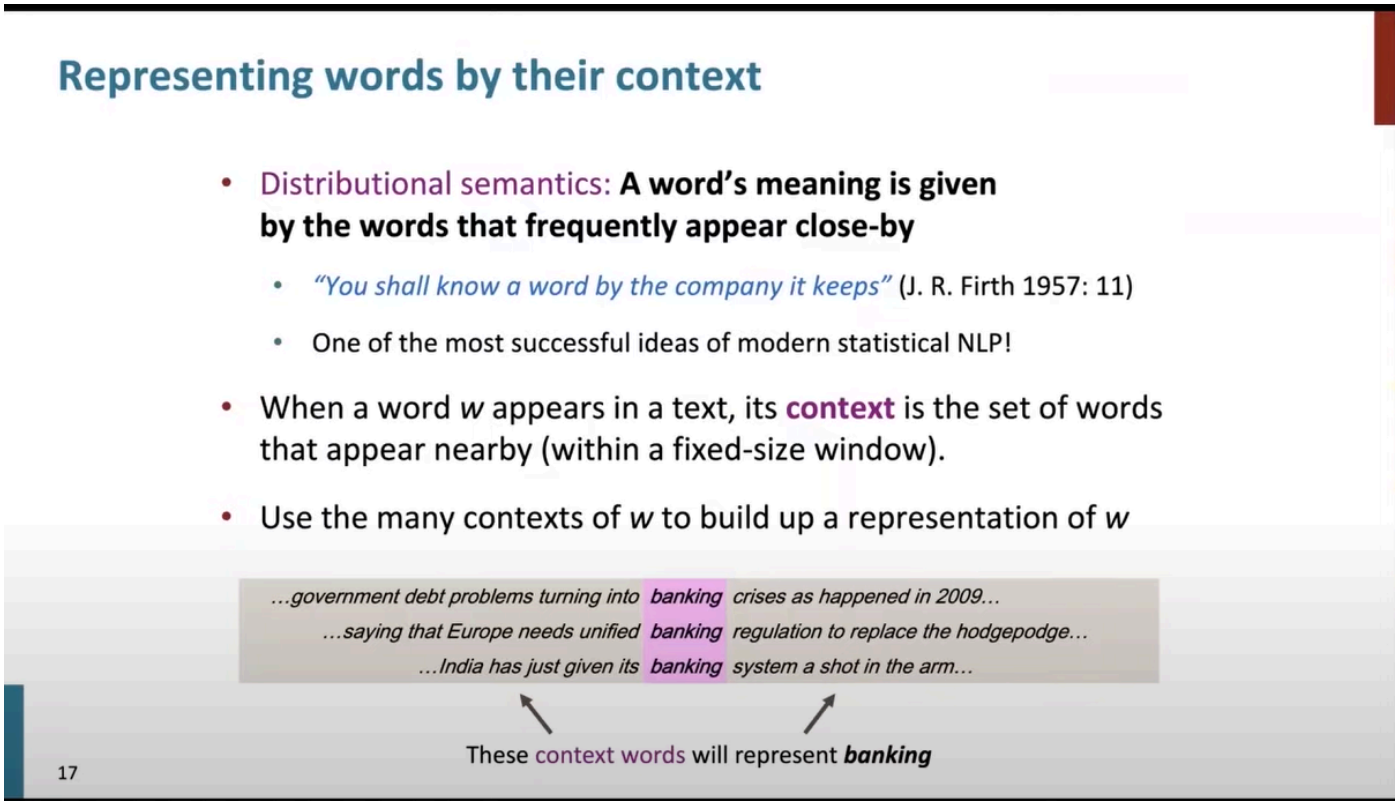
- Intro and Word Vectors
- word vector（2）
- Neural Networks
- Dependency Parsing
- Recurrent Neural networks
- Translation, Seq2Seq, Attention
- Self Attention
- Transformers and Pretraining
- Question Answering
- Natural Language Generation
- Coreference Resolution
- T5 and Large Language Models
- Add Knowledge to Language Models

Intro and Word Vectors



Word vectors

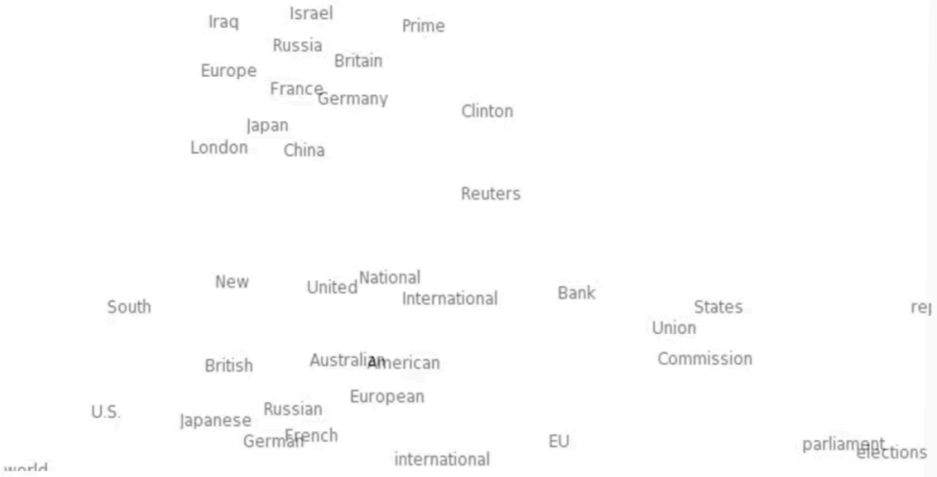
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

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

Note: word vectors are also called word embeddings or (neural) word representations
They are a distributed representation

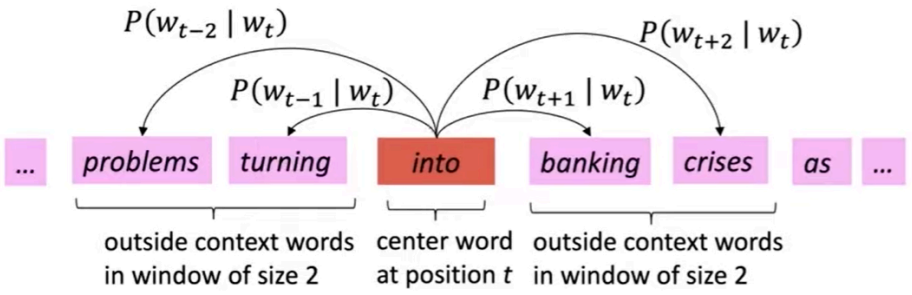
Word meaning as a neural word vector – visualization

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



Word2Vec Overview

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



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

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

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

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

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*:

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

24

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

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ ← Open region

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

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

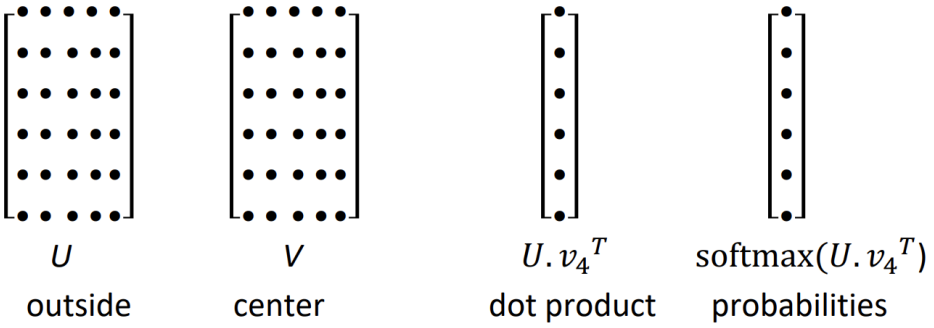
But sort of a weird name because it returns a distribution!

26

word vector（2）

Continuous Bag of Words（CBOW）

Word2vec parameters and computations



"Bag of words" model! → The model makes the same predictions at each position

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (at all often)

SGD

Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of **all** windows in the corpus (often, billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one, or each small batch
- Algorithm:

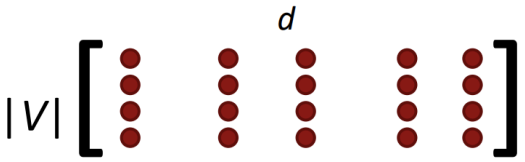
```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

这样的梯度矩阵在算word vector的时候会变得很稀疏， Thus

Stochastic gradients with word vectors!

- We might only update the word vectors that actually appear!
- Solution: either you need sparse matrix update operations to only update certain rows of full embedding matrices *U* and *V*, or you need to keep around a hash for word vectors

Rows not columns in actual DL packages!



- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!

2b. Word2vec algorithm family: More details

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word … and it helps

Two model variants:

1. Skip-grams (SG)
   Predict context ("outside") words (position independent) given center word
2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

We presented: **Skip-gram model**

Additional efficiency in training:

1. Negative sampling

So far: Focus on **naïve softmax** (simpler, but expensive, training method)

negative sampling **(*)**

> **why we do so：**
>
> 我们希望真正的**context word softmax**出来的值尽可能接近**1**，其他则尽可能小

负采样（Negative Sampling）是word2vec算法中的一种训练技巧，用于替代原始的naïve softmax训练方法。它旨在提高训练效率和降低计算复杂度。

在word2vec算法中，原始的训练目标是基于softmax分类器来预测给定上下文中的词汇概率分布。然而，对于大规模的词汇表，计算全局的softmax概率是非常昂贵的，因为它需要对所有词汇进行归一化计算。

为了解决这个问题，负采样技术被引入。它的基本思想是，对于每个训练样本，我们只需要更新一小部分的权重，而不是更新所有的权重。具体来说，对于每个训练样本（中心词和上下文词），负采样算法会从词汇表中随机选择一些负样本（通常是一些未出现在上下文中的词汇），并将它们作为负类标签。

然后，只有这些负样本的权重会被更新，而其他词汇的权重保持不变。

通过负采样，我们可以大大减少计算量，提高训练效率。相比于原始的softmax分类器需要更新整个权重矩阵，负采样只需要更新少量的负样本权重，从而减少了计算和存储开销。

具体的负采样算法会根据词汇的频率分布进行采样，使得高频词汇更有可能被选为负样本，以便更好地训练模型。在实践中，通常会根据经验或者调参来确定负采样的采样数量。

总而言之，负采样是word2vec算法中一种用于改进训练效率的技术，通过只更新少量的负样本权重，减少了计算复杂度，并在大规模词汇表下提供了可伸缩性。

当使用负采样时，我们首先需要定义一个负样本采样函数。这个函数根据词汇表中每个词汇的频率分布，以一定的概率选择负样本。通常，频率较高的词汇被选中的概率也较高。

例如，假设我们有一个包含以下词汇的句子：
"the cat sat on the mat"

现在，我们要训练一个word2vec模型，其中词汇表包含词汇："the"，"cat"，"sat"，"on"，"mat"。

步骤一：构建训练样本
我们以Skip-gram模型为例，选择中心词和上下文词对作为训练样本。假设我们设置上下文窗口大小为1，那么我们可以构建以下的训练样本：
对于中心词 "the"，上下文词为 ["cat"]。
对于中心词 "cat"，上下文词为 ["the"，"sat"]。
对于中心词 "sat"，上下文词为 ["cat"，"on"]。
对于中心词 "on"，上下文词为 ["sat"，"the"，"mat"]。
对于中心词 "mat"，上下文词为 ["on"]。
步骤二：负样本采样
对于每个训练样本，我们需要选择一些负样本。假设我们选择两个负样本。我们使用负采样函数根据词汇表中每个词汇的频率分布，以一定的概率选择负样本。频率较高的词汇被选中的概率也较高。
例如，我们的负样本采样函数根据频率分布选择了负样本词汇："dog"和"house"。

步骤三：更新权重
对于每个训练样本，我们更新权重，以使模型更好地预测上下文词汇。对于中心词和上下文词对，我们将它们视为正样本，更新它们对应的权重。同时，对于负样本词汇，我们将它们视为负样本，仅更新它们对应的权重。
通过这种方式，我们只需要更新少量的权重，而不是整个词汇表的权重。这样可以大大减少计算量和存储开销，提高训练效率。

在上述示例中，我们只显示了一个训练样本和两个负样本，但在实际训练中，会有多个训练样本和负样本。通过不断迭代训练样本，更新权重，模型逐渐学习到词汇之间的语义关系，最终得到每个词汇的词向量表示。

总而言之，负采样是通过随机选择一些负样本，并仅更新这些负样本的权重来提高word2vec模型训练的效率。这个过程涉及样本构建、负样本采样和权重更新等步骤。

假设我们有一个训练样本，中心词是"cat"，上下文词是["the"，"sat"]，并且我们选择了两个负样本词汇："dog"和"house"。

正样本权重的更新：
首先，我们计算中心词"cat"的词向量和上下文词"the"的词向量之间的内积，得到一个预测值。假设这个内积为0.8，并将其输入到sigmoid函数中，得到一个概率值为0.689，表示在给定"cat"的情况下，"the"出现的概率。
然后，我们将这个概率值与实际上下文词"the"的标签进行比较。假设标签为1，表示"the"是正样本。我们计算预测值与实际值之间的差距，即损失。假设损失为0.2。

接下来，我们计算损失函数对中心词"cat"和上下文词"the"的词向量的偏导数。假设得到的偏导数分别为[0.1, 0.2]和[0.05, 0.15]。然后，根据学习率来更新中心词和上下文词的词向量。假设学习率为0.01，我们可以将中心词和上下文词的词向量更新如下：

中心词"cat"的词向量更新为[0.1, 0.2] - 0.01 * [0.1, 0.2] = [0.099, 0.198]
上下文词"the"的词向量更新为[0.05, 0.15] - 0.01 * [0.05, 0.15] = [0.0495, 0.1485]

通过这样的更新过程，我们更新了正样本的词向量，使得模型能够更准确地预测上下文词汇。

负样本权重的更新：
现在，我们来处理负样本词汇"dog"和"house"。
首先，我们计算负样本词汇"dog"的词向量和中心词"cat"的词向量之间的内积，得到一个预测值。假设这个内积为-0.6，并将其输入到sigmoid函数中，得到一个概率值为0.354，表示在给定"cat"的情况下，"dog"出现的概率。

然后，我们将这个概率值与负样本词汇"dog"的标签进行比较。假设标签为0，表示"dog"是负样本。我们计算预测值与实际值之间的差距，即损失。假设损失为0.354。

接下来，我们计算损失函数对负样本词汇"dog"的词向量的偏导数。假设得到的偏导数为[-0.2, -0.1]。然后，根据学习率来更新负样本词汇"dog"的词向量。假设学习率为0.01，我们可以将负样本词汇"dog"的词向量更新如下：

负样本词汇"dog"的词向量更新为[-0.2, -0.1] - 0.01 * [-0.2, -0.1] = [-0.202, -0.101]

通过这样的更新过程，我们降低了负样本词汇"dog"的预测概率，使得模型能够更好地区分正样本和负样本。

对于负样本词汇，我们希望模型能够将它们的预测概率降低，以更好地区分正样本和负样本。这是因为在训练过程中，我们使用了负采样（negative sampling）的方法，从词汇表中选择了一些负样本词汇。这些负样本词汇在训练样本中并不真实地与中心词共现，因此我们希望模型能够将它们与正样本词汇区分开来。

在更新负样本的权重时，我们计算了负样本词汇的预测概率，并将其与实际值进行比较。如果模型在预测负样本时的概率接近于1，说明模型将其错误地预测为正样本，这样就无法区分正样本和负样本了。因此，我们希望通过更新权重，降低负样本的预测概率，使得模型能够更好地区分正样本和负样本。

通过梯度下降的方法更新负样本的权重，我们根据损失函数计算了负样本词汇的梯度，并将其乘以学习率应用于权重更新。这样，负样本词汇的词向量在每次迭代中都会被微调，以使其与中心词更好地区分开来。

总之，在更新权重的过程中，通过降低负样本的预测概率，我们使得模型能够更好地区分正样本和负样本，从而提高了模型的性能和准确性。

但是有的时候一个单词有多个意思，这样的话怎么用一个向量来表示呢

Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$
- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)!

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

Neural Networks

NER（命名实体识别）是自然语言处理（NLP）中的一项任务，旨在从文本中识别和分类具有特定意义的命名实体。命名实体是指在文本中表示具体事物的词汇单元，如人名、地名、组织机构、日期、时间、货币等。

NER的目标是将文本中的命名实体识别并标注其所属的预定义类别，通常包括人名、地名、组织机构名、日期、时间、货币、百分比等。NER任务的输出结果是对文本中每个识别到的命名实体进行标注，指示其所属的类别。

Simple NER: Window classification using binary logistic classifier

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

the museums in Paris are amazing to see .

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

- Resulting vector $x_{window} = x \in R^{5d}$, a column vector!
- To classify all words: run classifier for each class on the vector centered on each word in the sentence

An Example

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

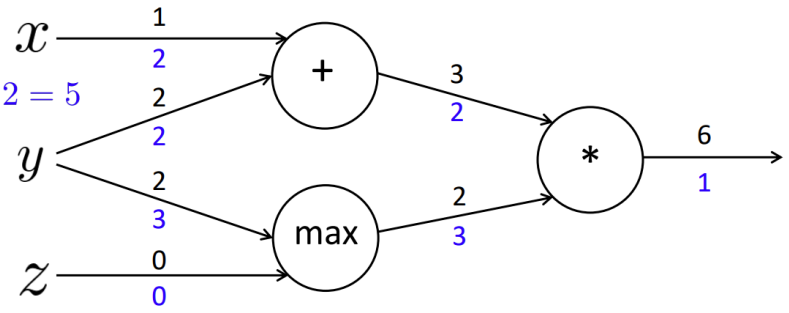
Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$

$\frac{\partial f}{\partial x} = 2$
$\frac{\partial f}{\partial y} = 3 + 2 = 5$
$\frac{\partial f}{\partial z} = 0$



Dependency Parsing

在语言学中，组成性指的是将单词组织成嵌套的成分（constituents），而短语结构语法则是使用上下文无关文法（CFGs）来描述语言结构。

具体地，短语结构是将单词组织成嵌套的成分的方式。起始单位是单词，例如：the、cat、cuddly、by、door。这些单词可以组合成短语，例如：the cuddly cat、by the door。而这些短语又可以组合成更大的短语，例如：the cuddly cat by the door。

上下文无关文法（CFGs）是一种可以用于描述语言结构的形式化工具。它使用产生式规则来定义如何将

非终结符（例如短语）替换为终结符（例如单词）或其他非终结符的序列。这些产生式规则可以用于生成符合语法规则的句子，也可以用于分析一个句子是否符合语法规则。

依存结构显示了每个单词与其他单词之间的依赖关系，即哪些单词依赖（修改、连接或作为论元）于其他单词。

例如，句子 "Look in the large crate in the kitchen by the door" 的依存结构可以表示为以下关系：

"Look" 是整个句子的核心动词，其他单词都依赖于它。
"crate" 依赖于 "Look"，表示 "Look" 的宾语。
"large" 依赖于 "crate"，表示 "crate" 的形容词修饰语。
"in" 依赖于 "crate"，表示 "crate" 的位置关系。
"the" 依赖于 "crate"，表示 "crate" 的限定词。
"kitchen" 依赖于 "in"，表示位置关系。
"by" 依赖于 "Look"，表示 "Look" 的附加关系。
"the" 依赖于 "door"，表示 "door" 的限定词。
"door" 依赖于 "by"，表示附加关系。
通过依存结构，可以清晰地表示出每个单词之间的依赖关系和句子中各个成分的作用。依存结构在句法分析、语义分析和机器翻译等任务中起着重要的作用。

Why do we need sentence structure?

Humans communicate complex ideas by composing words together into bigger units to convey complex meanings

Listeners need to work out what modifies [attaches to] what

A model needs to understand sentence structure in order to be able to interpret language correctly

人类语言天生就用多歧义的特点

Recurrent Neural networks

神经网络可以学会非线性划分的原因是：

神经网络由多个层组成，每个层都包含一些神经元，每个神经元都有一个激活函数，激活函数可以是非线性的，如sigmoid、tanh、ReLU等。

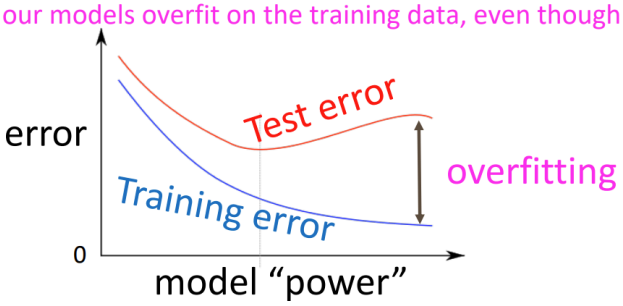
当输入信号通过神经网络的各层传播时，每个神经元都会对输入信号进行非线性变换，从而产生新的特征。
这些新的特征可以在更高维的空间中将原本线性不可分的数据分开，从而实现非线性划分。

We have models with many parameters! Regularization!

- A full loss function includes **regularization** over all parameters θ, e.g., L2 regularization:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) + \lambda \sum_k \theta_k^2$$

- Classic view: Regularization works to prevent overfitting when we have a lot of features (or later a very powerful/deep model, etc.)
- Now: Regularization produces models that generalize well when we have a "big" model
  - We do not care that our models overfit on the training data, even though they are **hugely** overfit



17

但是却如今的大网络中过拟合已经无所谓了，实际上人们总是让其过拟合（our big nurel net these days are always overfit on training fata）

Dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov 2012/JMLR 2014)

Preventing Feature Co-adaptation = Good Regularization Method!

- Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
- Test time: halve the model weights (now twice as many)
- (Except usually only drop first layer inputs a little (~15%) or not at all)
- This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
- In a single layer: A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where weights are set in the context of all others)
- Can be thought of as a form of model bagging (i.e., like an ensemble model)
- Nowadays usually thought of as strong, feature-dependent regularizer [Wager, Wang, & Liang 2013]

18

This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features

i.e. 这防止了功能的共同适应:一个功能不能只在特定的其他功能存在的情况下才有用

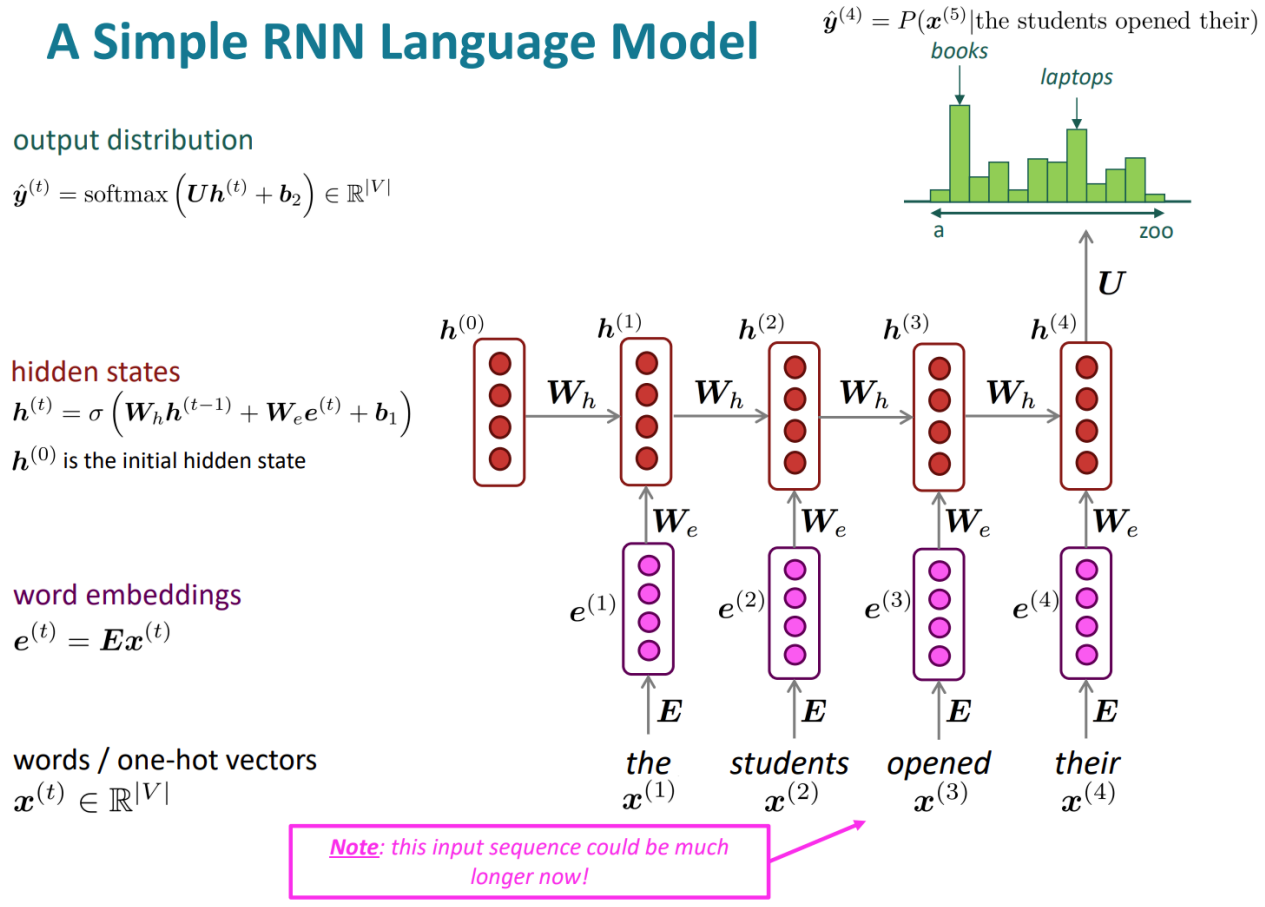
Parameter Initialization

- You normally must initialize weights to small random values (i.e., not zero matrices!)
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights** ~ Uniform(–*r*, *r*), with *r* chosen so numbers get neither too big or too small [later the need for this is removed with use of layer normalization]
- Xavier initialization has variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size):

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

- **Language Modeling** is the task of predicting what word comes next

A Simple RNN Language Model



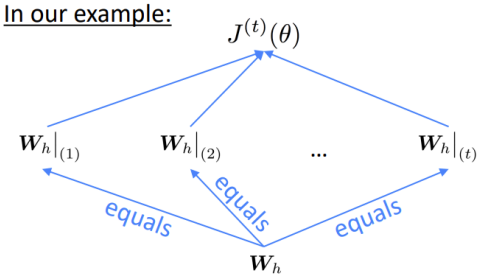
RNN反向传播的梯度

Backpropagation for RNNs: Proof sketch

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

In our example:



Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)} \frac{\partial W_h|_{(i)}}{\partial W_h}$$

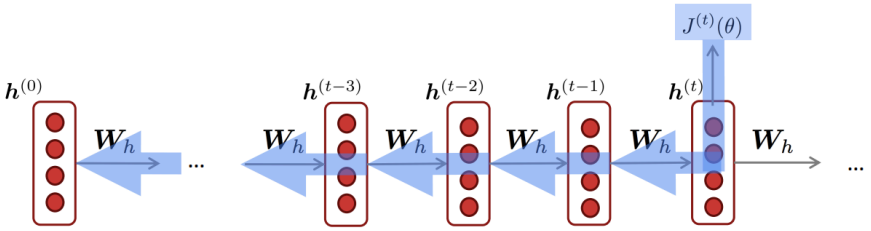
(= 1)

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$$

Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

55

Backpropagation for RNNs



$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i=t,…,0*, summing gradients as you go.
This algorithm is called **"backpropagation through time"**
[Werbos, P.G., 1988, *Neural Networks* **1**, and others]

56

一个评价指标：困惑度

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

62

防止梯度爆炸：CLIP

Gradient clipping: solution for exploding gradient

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- **Intuition**: take a step in the same direction, but a smaller step
- In practice, remembering to clip gradients is important, but exploding gradients are an easy problem to solve

40

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

LSTM

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

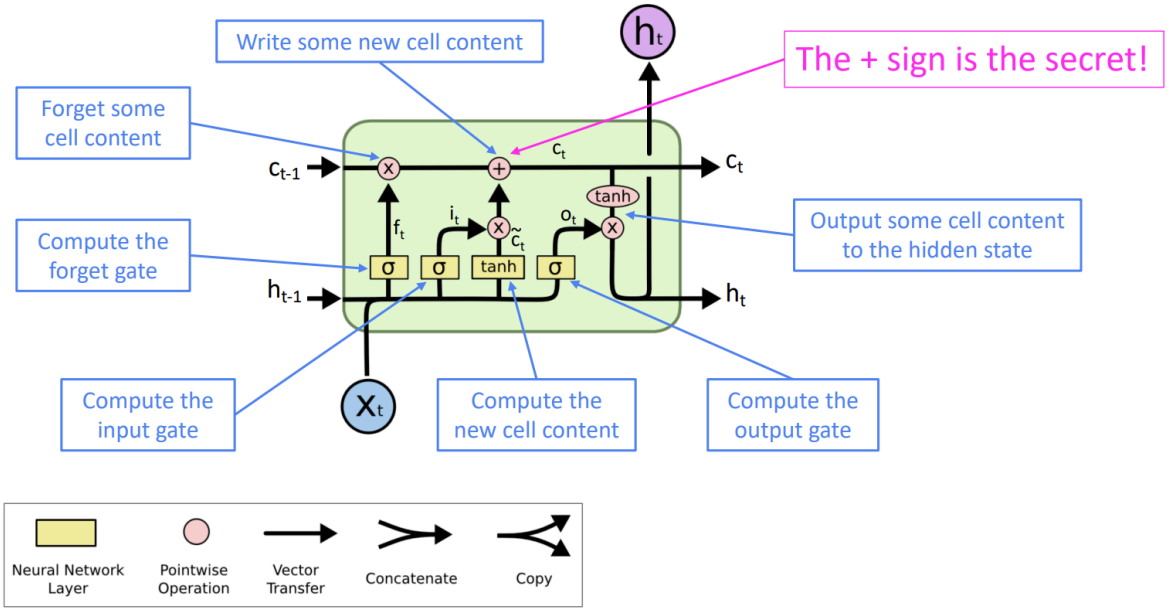
$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



Translation, Seq2Seq, Attention

1990s-2010s: Statistical Machine Translation

- Core idea: Learn a probabilistic model from data
- Suppose we're translating French → English.
- We want to find best English sentence *y*, given French sentence *x*

$$\text{argmax}_y P(y|x)$$

- Use Bayes Rule to break this down into two components to be learned separately:

$$= \text{argmax}_y \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learnt from parallel data.

Language Model

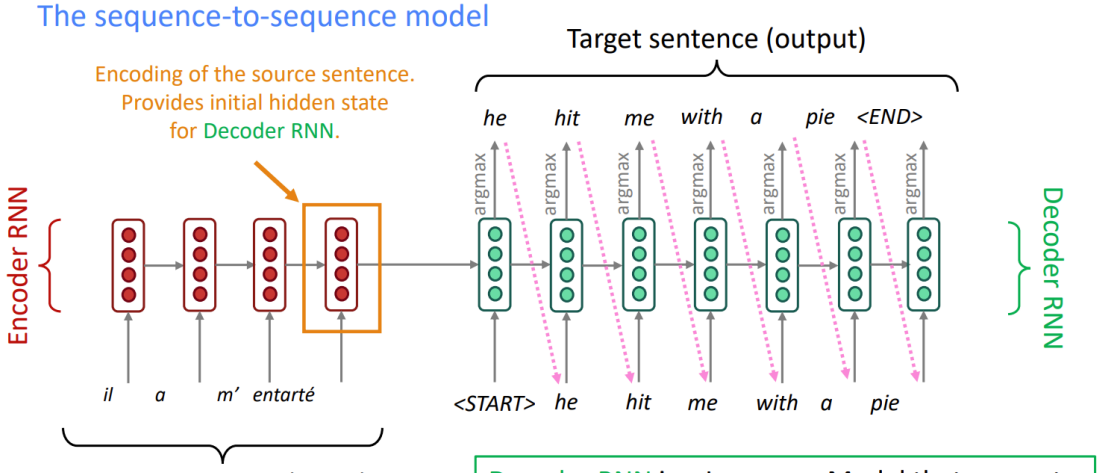
Models how to write good English (*fluency*). Learnt from monolingual data.

1990s-2010s: Statistical Machine Translation

- SMT was a huge research field
- The best systems were extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems had many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

Neural Machine Translation (NMT)

The sequence-to-sequence model



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

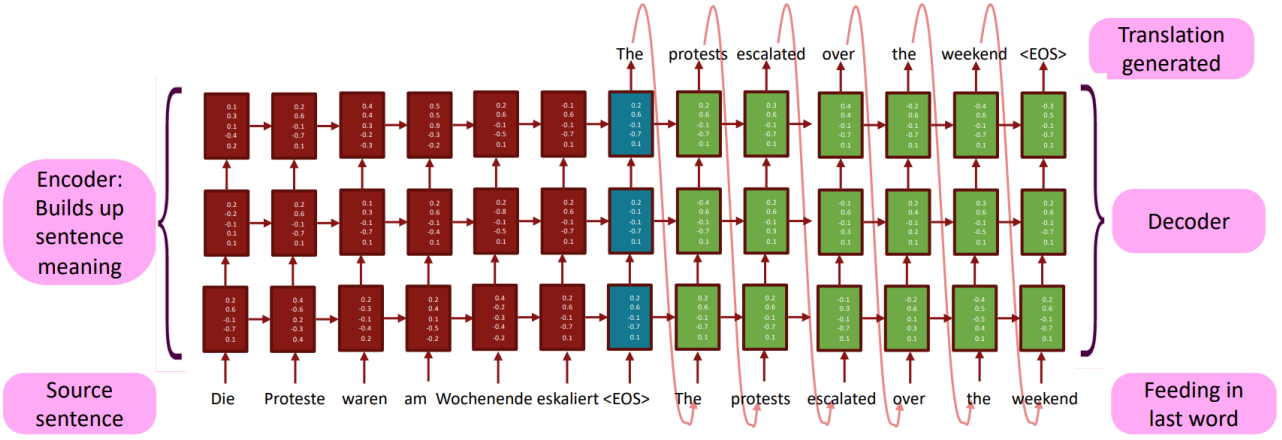
Encoder RNN produces an encoding of the source sentence.

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

Multi-layer deep encoder-decoder machine translation net

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

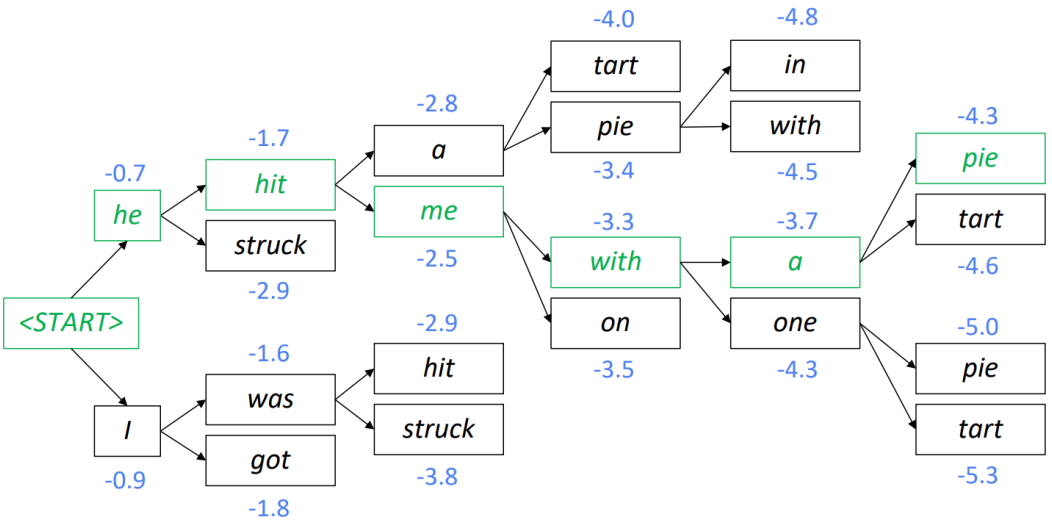
The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Conditioning = Bottleneck

Beam search decoding: example

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



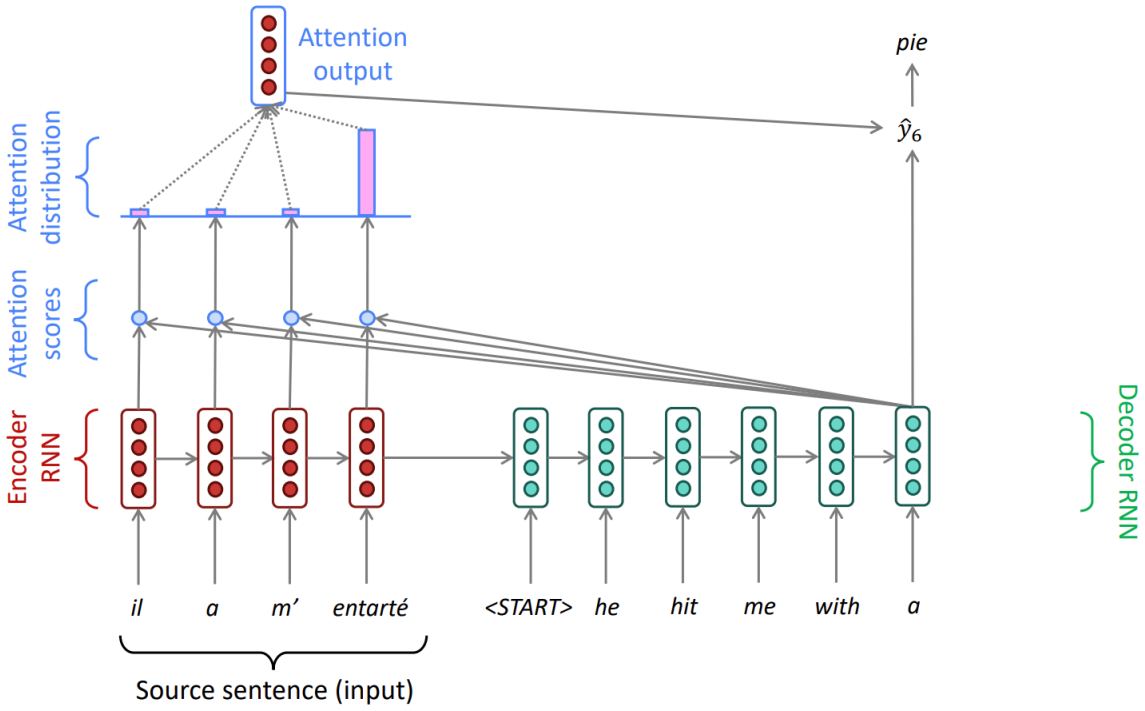
Backtrack to obtain the full hypothesis

Attention

- **Attention** provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

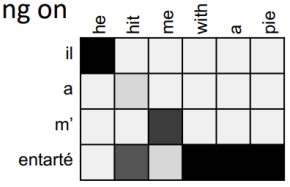
- First, we will show via diagram (no equations), then we will show with equations

1. Sequence-to-sequence with attention



Attention is great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a *general* Deep Learning technique

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Upshot:

- Attention has become the powerful, flexible, general way pointer and memory manipulation in deep learning models. A new idea from after 2010!

Self Attention

Self-Attention

- Recall: Attention operates on **queries**, **keys**, and **values.**
  - We have some **queries** $q_1, q_2, \dots, q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, \dots, k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, \dots, v_T$. Each value is $v_i \in \mathbb{R}^d$

The number of queries can differ from the number of keys and values in practice.

- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, \dots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:

$$e_{ij} = q_i^\top k_j$$

Compute **key-query** affinities

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

Compute attention weights from affinities (softmax)

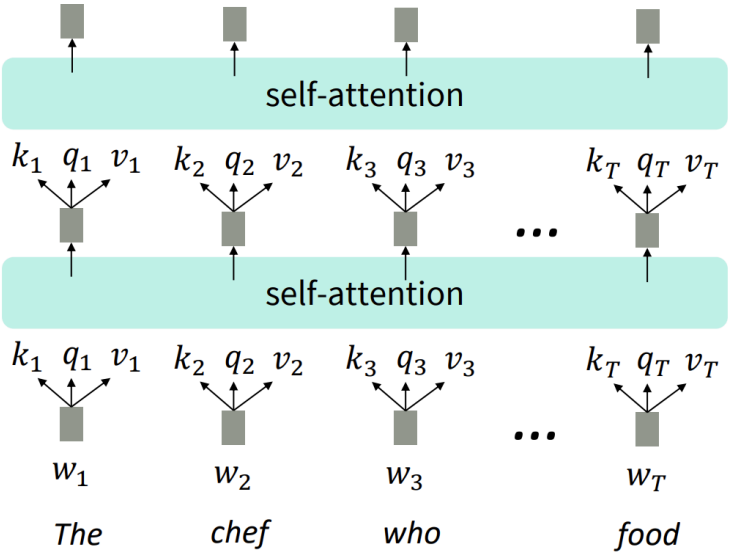
$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

为什么不直接把lstm换成self attention-只是说明关系，没有先后的顺序

Self-attention as an NLP building block

- In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers.
- Can self-attention be a drop-in replacement for recurrence?
- No. It has a few issues, which we'll go through.
- First, self-attention is an operation on **sets**. It has no inherent notion of order.



Self-attention doesn't know the order of its inputs.

so，加上位置编码

Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i\ \tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

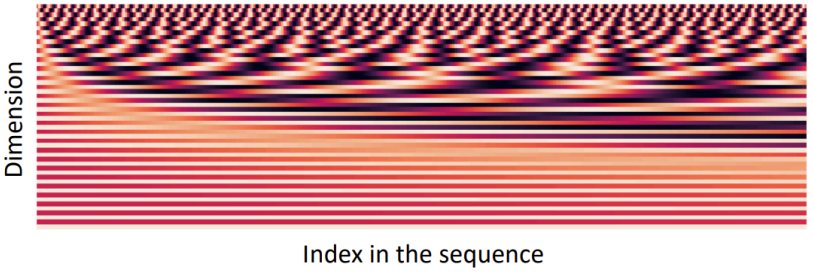
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
  - Not learnable; also the extrapolation doesn't really work!

现在我们通常直接把p里面的元素弄成可学习的参数

Position representation vectors learned from scratch

- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $p \in \mathbb{R}^{d \times T}$, *and let each* p_i *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, T$.
- Most systems use this!

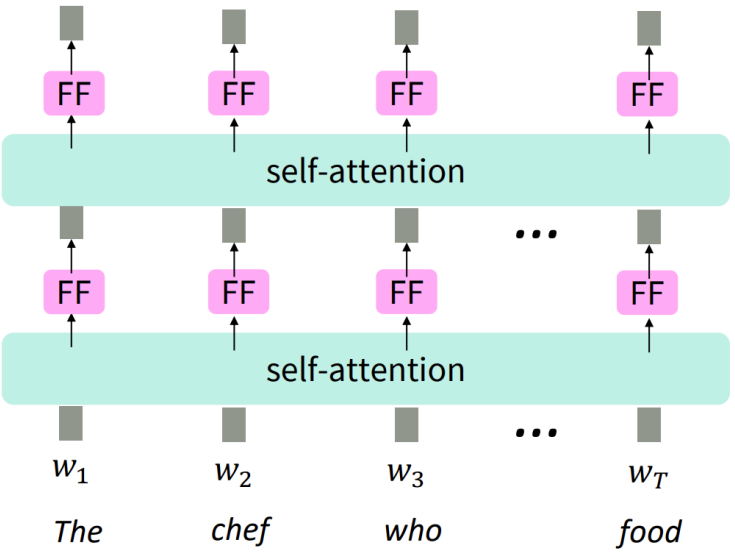
- Sometimes people try more flexible representations of position:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

加上非线性表征

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors
- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



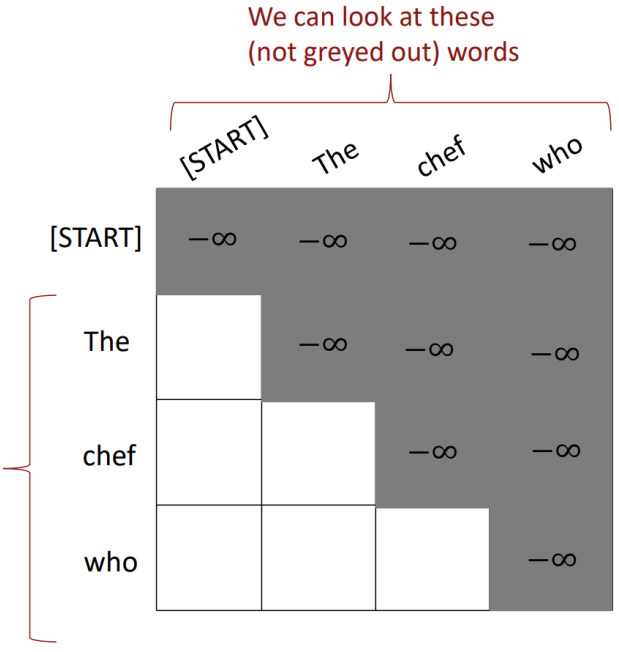
Intuition: the FF network processes the result of attention

防止attention模块预知未来-mask

Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Then here comes Transformer !

The Transformer Encoder: **Key-Query-Value Attention**

- We saw that self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular way:
  - Let $x_1, \ldots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^d$
- Then keys, queries, values are:
  - $k_i = Kx_i$, where $K \in \mathbb{R}^{d \times d}$ is the key matrix.
  - $q_i = Qx_i$, where $\mathrm{Q} \in \mathbb{R}^{d \times d}$ is the query matrix.
  - $v_i = Vx_i$, where $\mathrm{V} \in \mathbb{R}^{d \times d}$ is the value matrix.
- These matrices allow *different aspects* of the x vectors to be used/emphasized in each of the three roles.

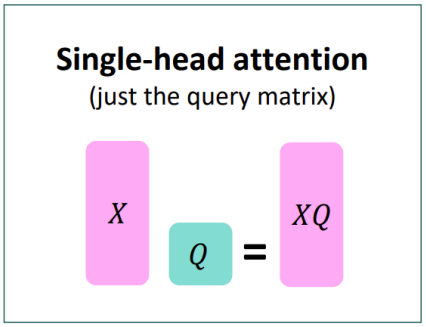
多头注意力🚀

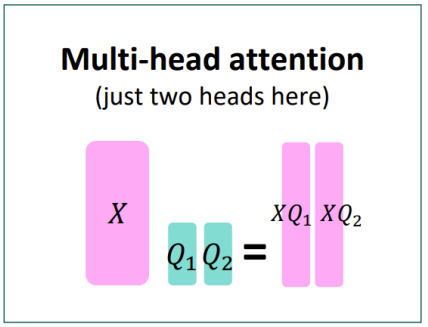
The Transformer Encoder: **Multi-headed attention**

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\mathrm{output}_\ell = \mathrm{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\mathrm{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\mathrm{output} = Y[\mathrm{output}_1; \ldots; \mathrm{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.

The Transformer Encoder: **Multi-headed attention**

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.





Same amount of computation as single-head self-attention!

The Transformer Encoder: **Layer normalization** [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

The Transformer Encoder: **Scaled Dot Product** [Vaswani et al., 2017]

- **"Scaled Dot Product"** attention is a final variation to aid in Transformer training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we've seen:

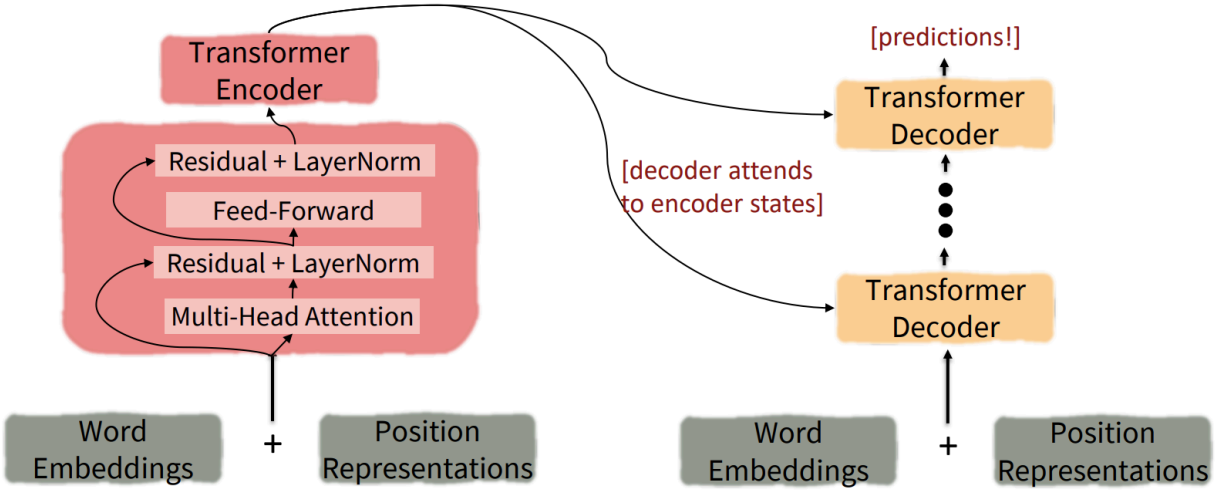
$$\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$$

- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

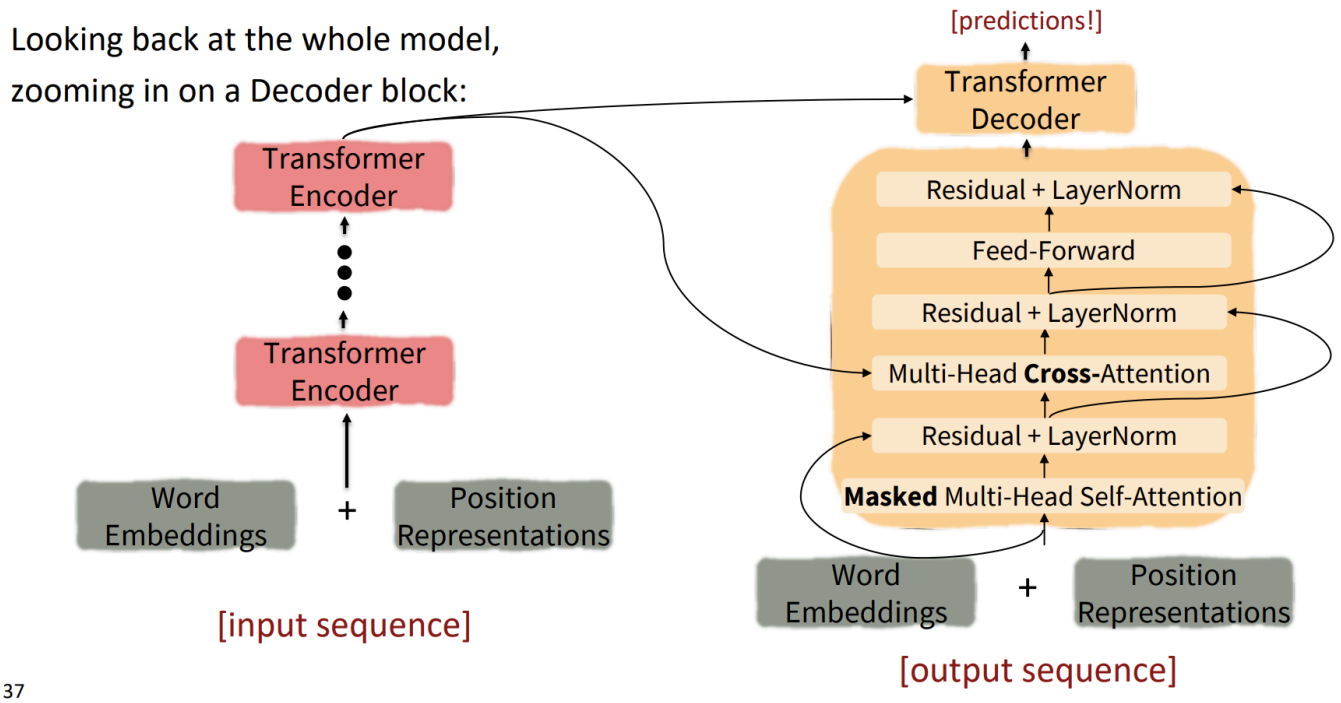
$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



The Transformer Encoder-Decoder [Vaswani et al., 2017]



37

缺点：

What would we like to fix about the Transformer?

- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!
- **Position representations**:
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

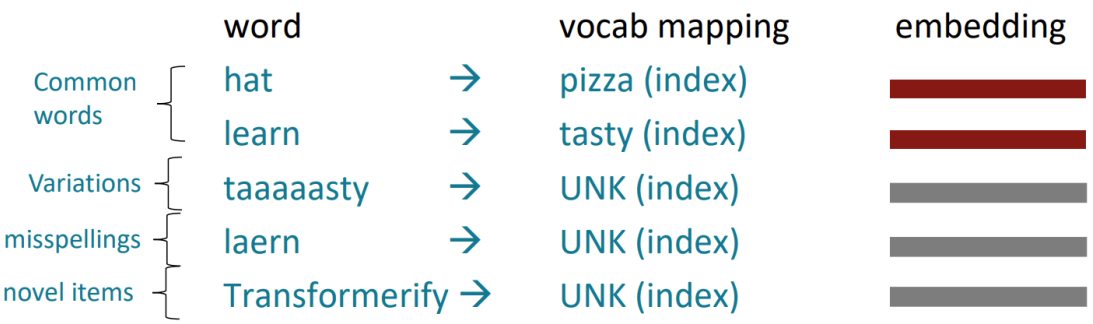
Transformers and Pretraining

why Word structure and subword models [robotic,从字段中学习，大多transformer就是这样分词的]

Word structure and subword models

Let's take a look at the assumptions we've made about a language's vocabulary.

We assume a fixed vocab of tens of thousands of words, built from the training set. All *novel* words seen at test time are mapped to a single UNK.



The byte-pair encoding algorithm

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of **parts of words (subword tokens).**
- At training and testing time, each word is split into a sequence of known subwords.

Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary.

1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
2. Using a corpus of text, find the most common adjacent characters "a,b"; add "ab" as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

Word structure and subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.

	word		vocab mapping	embedding
Common words	hat	→	hat	
	learn	→	learn	
Variations	taaaaasty	→	taa## aaa## sty	
misspellings	laern	→	la## ern##	
novel items	Transformerify	→	Transformer## ify	

Type pretraing

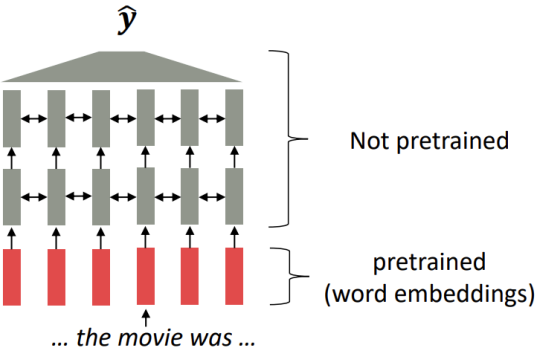
Where we were: **pretrained word embeddings**

Circa 2017:

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.

Some issues to think about:

- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



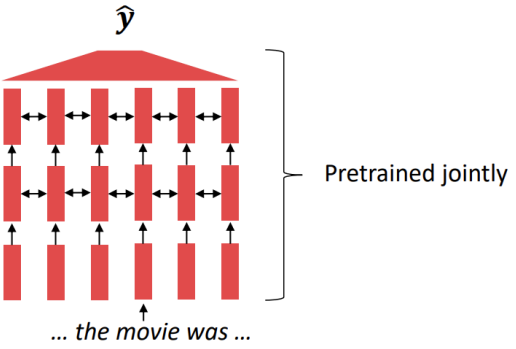
[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we're going: **pretraining whole models**

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from

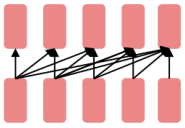


[This model has learned how to represent entire sentences through pretraining]

Pretraining

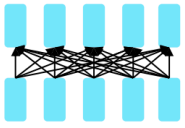
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



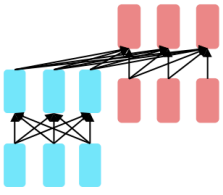
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

怎么训练encorder（他们是双向的，能够看到整个句子）

- 把一些单词mask掉，去学这些mask掉的单词

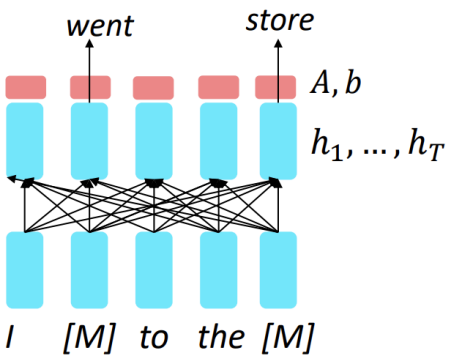
Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.



[Devlin et al., 2018]

35

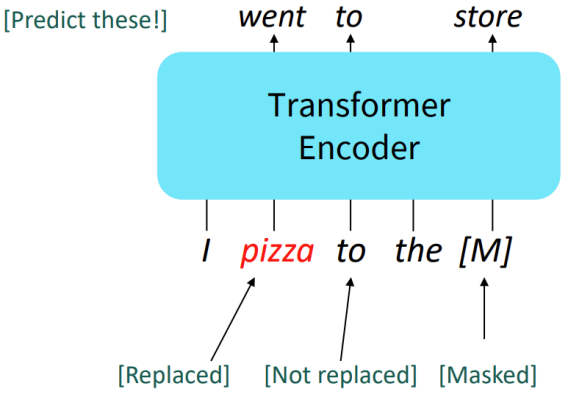
- 更进一步，为了学习更好的表示，而不仅仅是在看到mask以后才去预测（通过随机替换词，模型被迫依赖于上下文中的其他词来预测被替换的词）

BERT: Bidirectional Encoder Representations from Tranformers

Devlin et al., 2018 proposed the "Masked LM" objective and **released the weights of a pretrained Transformer**, a model they labeled BERT.

Some more details about Masked LM for BERT:

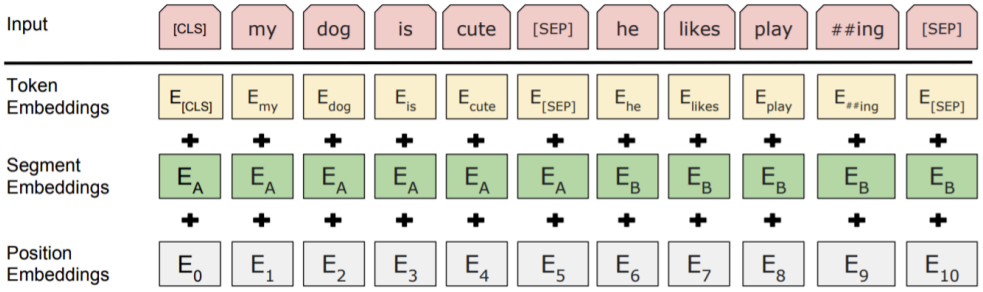
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



- 也会随机替换一般句子，目的同上

BERT: Bidirectional Encoder Representations from Tranformers

- The pretraining input to BERT was two separate contiguous chunks of text:



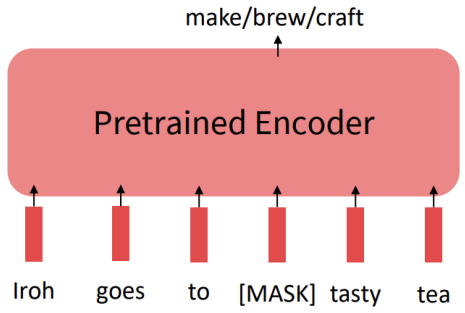
- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - Later work has argued this “next sentence prediction” is not necessary.

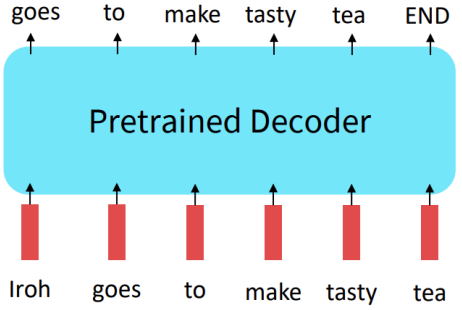
但是实际上我们希望它能生成句子而不是只是去预测mask掉的单词or被替换的东西

Limitations of pretrained encoders

Those results looked great! Why not used pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don’t naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





Encorder-Decorder架构 [T5]

Question Answering

Why do we care about this problem?

- Useful for many practical applications
- Reading comprehension is an important testbed for evaluating how well computer systems understand human language
  - Wendy Lehnert 1977: "Since questions can be devised to query **any aspect** of text comprehension, the ability to answer questions is the **strongest possible demonstration of understanding**."
- Many other NLP tasks can be reduced to a reading comprehension problem:

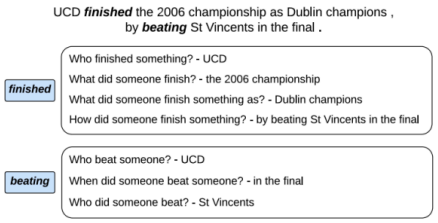
Information extraction
(Barack Obama, educated_at, ?)

Question: Where did Barack Obama graduate from?

Passage: Obama was born in Honolulu, Hawaii. After graduating from Columbia University in 1983, he worked as a community organizer in Chicago.

(Levy et al., 2017)

Semantic role labeling



(He et al., 2015)

Neural models for reading comprehension

How can we build a model to solve SQuAD?

(We are going to use **passage, paragraph and context,** as well as **question** and **query** interchangeably)

- Problem formulation
  - Input: $C = (c_1, c_2, \ldots, c_N), Q = (q_1, q_2, \ldots, q_M), c_i, q_i \in V$
  - Output: $1 \leq$ start $\leq$ end $\leq N$

N~100, M ~15

answer is a span in the passage

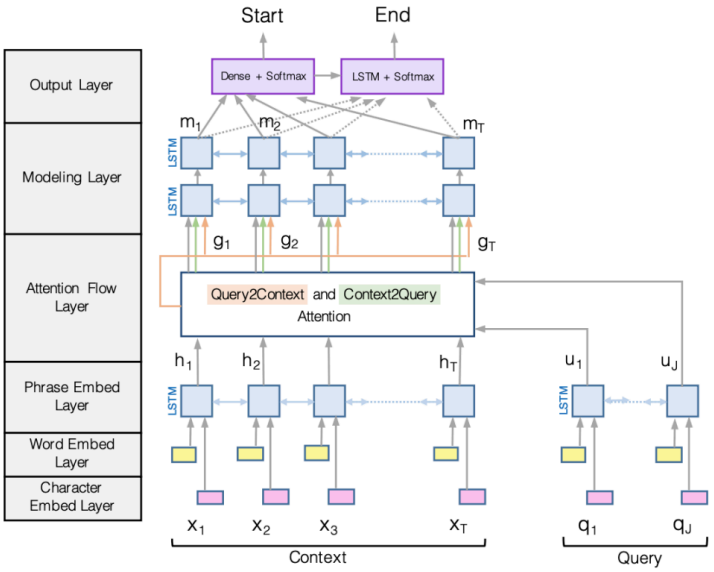
- A family of LSTM-based models with attention (2016-2018)

  Attentive Reader (Hermann et al., 2015), Stanford Attentive Reader (Chen et al., 2016), Match-LSTM (Wang et al., 2017), BiDFA (Seo et al., 2017), Dynamic coattention network (Xiong et al., 2017), DrQA (Chen et al., 2017), R-Net (Wang et al., 2017), ReasoNet (Shen et al., 2017)..

- Fine-tuning BERT-like models for reading comprehension (2019+)

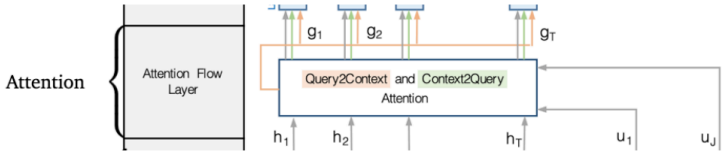
- 以前：用BiDAF

BiDAF: the Bidirectional Attention Flow model



(Seo et al., 2017): Bidirectional Attention Flow for Machine Comprehension

BiDAF: Attention



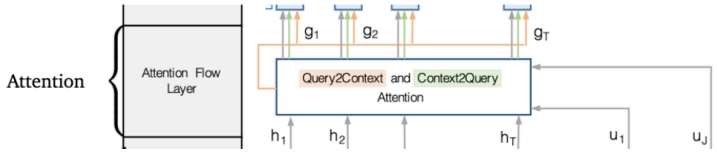
- Context-to-query attention: For each context word, choose the most relevant words from the query words.

Q: *Who leads the United States?*

C: *Barak Obama is the president of the USA.*

For each context word, find the most relevant query word.

BiDAF: Attention

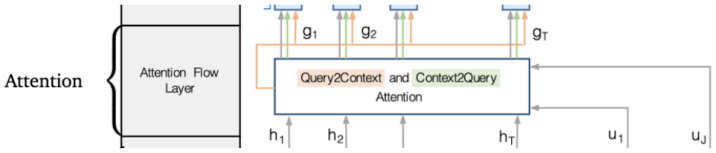


- Query-to-context attention: choose the context words that are most relevant to one of query words.

While Seattle's weather is very nice in summer, its weather is very rainy in winter, making it one of the most gloomy cities in the U.S. LA is ...

Q: Which city is gloomy in winter?

BiDAF: Attention



The final output is
$\mathbf{g}_i = [\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{b}] \in \mathbb{R}^{8H}$

- First, compute a similarity score for every pair of $(\mathbf{c}_i, \mathbf{q}_j)$:

$$S_{i,j} = \mathbf{w}_{\text{sim}}^{\intercal}[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \odot \mathbf{q}_j] \in \mathbb{R} \qquad \mathbf{w}_{\text{sim}} \in \mathbb{R}^{6H}$$

- Context-to-query attention (which question words are more relevant to c_i):

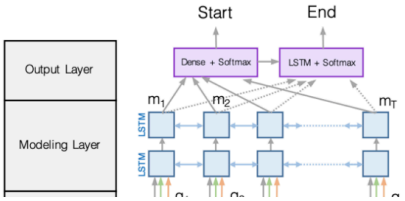
$$\alpha_{i,j} = \text{softmax}_j(S_{i,j}) \in \mathbb{R} \qquad \mathbf{a}_i = \sum_{j=1}^{M} \alpha_{i,j}\mathbf{q}_j \in \mathbb{R}^{2H}$$

- Query-to-context attention (which context words are relevant to some question words):

$$\beta_i = \text{softmax}_i(\max_{j=1}^{M}(S_{i,j})) \in \mathbb{R}^N \qquad \mathbf{b} = \sum_{i=1}^{N} \beta_i \mathbf{c}_i \in \mathbb{R}^{2H}$$

训练的是两个定位器，找到答案在文中开始和结束的位置

BiDAF: Modeling and output layers



The final training loss is

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

Modeling layer: pass $\mathbf{g}_i$ to another two layers of **bi-directional** LSTMs.

- Attention layer is modeling interactions between query and context
- Modeling layer is modeling interactions within context words

$$\mathbf{m}_i = \text{BiLSTM}(\mathbf{g}_i) \in \mathbb{R}^{2H}$$

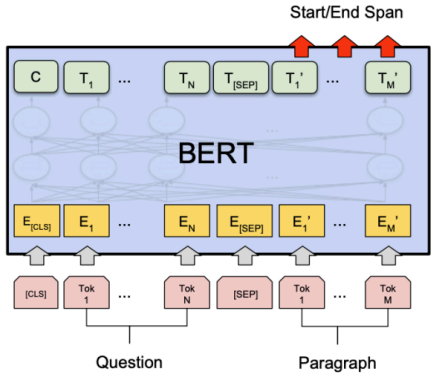
Output layer: two classifiers predicting the start and end positions:

$$p_{\text{start}} = \text{softmax}(\mathbf{w}_{\text{start}}^\intercal [\mathbf{g}_i; \mathbf{m}_i]) \qquad p_{\text{end}} = \text{softmax}(\mathbf{w}_{\text{end}}^\intercal [\mathbf{g}_i; \mathbf{m}_i'])$$

$$\mathbf{m}_i' = \text{BiLSTM}(\mathbf{m}_i) \in \mathbb{R}^{2H} \quad \mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}} \in \mathbb{R}^{10H}$$

- 现在用Bert

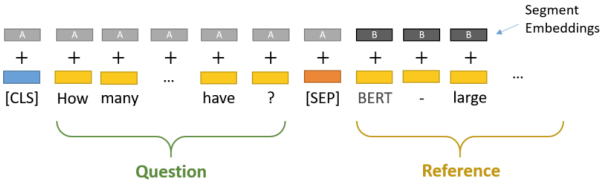
BERT for reading comprehension



Question = Segment A

Passage = Segment B

Answer = predicting two endpoints in segment B



Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

Image credit: https://mccormickml.com/

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

$$p_{\text{start}}(i) = \text{softmax}_i(\mathbf{w}_{\text{start}}^\intercal \mathbf{H})$$

$$p_{\text{end}}(i) = \text{softmax}_i(\mathbf{w}_{\text{end}}^\intercal \mathbf{H})$$

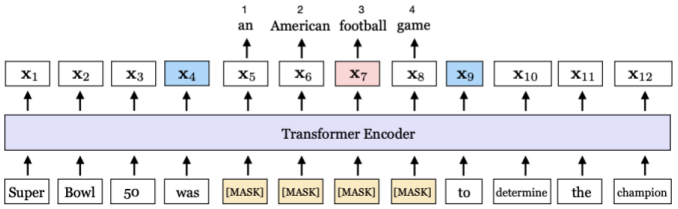
where $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N]$ are the hidden vectors of the paragraph, returned by BERT

span bert（预测两个指针之间的内容）

Can we design better pre-training objectives?

The answer is yes!

$$\mathcal{L}(\text{football}) = \mathcal{L}_{\text{MLM}}(\text{football}) + \mathcal{L}_{\text{SBO}}(\text{football})$$
$$= -\log P(\text{football} \mid \mathbf{x}_7) - \log P(\text{football} \mid \mathbf{x}_4, \mathbf{x}_9, \mathbf{p}_3)$$



Two ideas:

1) masking contiguous spans of words instead of 15% random words

2) using the two end points of span to predict all the masked words in between = compressing the information of a span into its two endpoints

$$\mathbf{y}_i = f(\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_{i-s+1})$$

(Joshi & Chen et al., 2020): SpanBERT: Improving Pre-training by Representing and Predicting Spans

但是，The current systems still perform poorly on adversarial examples or examples from out-of domain distributions

Is reading comprehension solved?

- We have already surpassed human performance on SQuAD. Does it mean that reading comprehension is already solved? **Of course not!**
- The current systems still perform poorly on adversarial examples or examples from out-of-domain distributions

Article: Super Bowl 50
Paragraph: "*Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver's Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV.*"
Question: "*What is the name of the quarterback who was 38 in Super Bowl XXXIII?*"
Original Prediction: John Elway
Prediction under adversary: Jeff Dean

	Match Single	Match Ens.	BiDAF Single	BiDAF Ens.
Original	71.4	75.4	75.5	80.0
ADDSENT	27.3	29.4	34.3	34.2
ADDONESENT	39.0	41.8	45.7	46.9
ADDANY	7.6	11.7	4.8	2.7
ADDCOMMON	38.9	51.0	41.7	52.6

Natural Language Generation

Decorder 用贪心算法

Greedy methods

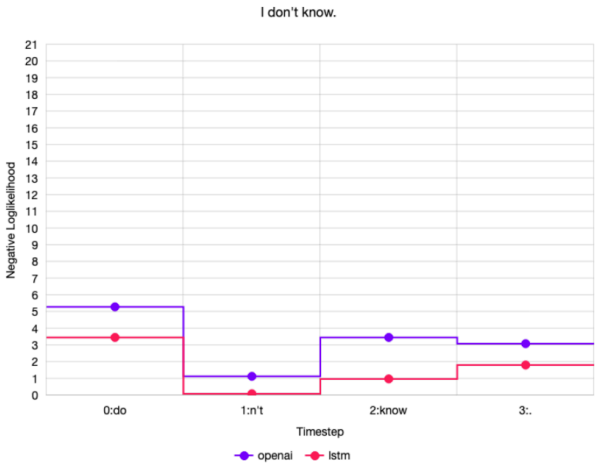
- **Recall**: Lecture 7 on Neural Machine Translation…
- Argmax Decoding
  - Selects the highest probability token in $P(y_t|y_{<t})$

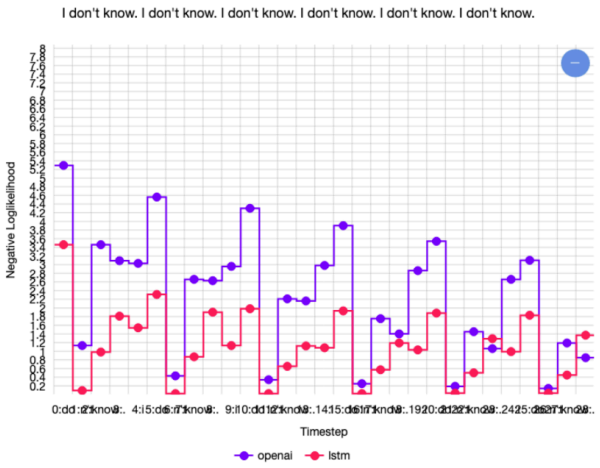
$$\hat{y}_t = \underset{w \in V}{\text{argmax}}\, P(y_t = w|y_{<t})$$

- Beam Search
  - Discussed in Lecture 7 on Machine Translation
  - Also a greedy algorithm, but with wider search over candidates

为什么model总是喜欢在结尾一直重复一句话

Why does repetition happen?





How can we reduce repetition?

Simple option:

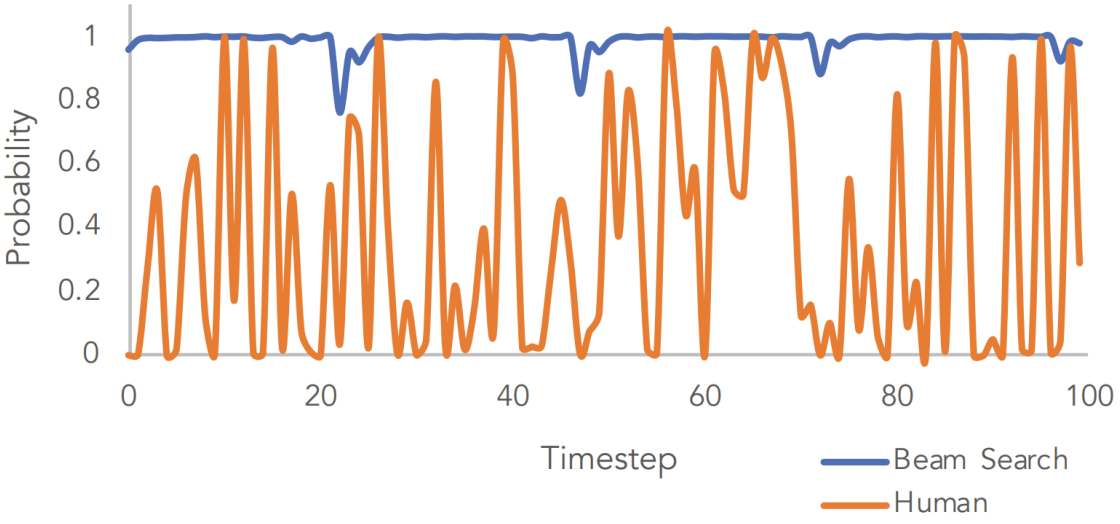
- Heuristic: Don't repeat *n*-grams

More complex:

- Minimize embedding distance between consecutive sentences (Celikyilmaz et al., 2018)
  - Doesn't help with intra-sentence repetition
- Coverage loss (See et al., 2017)
  - Prevents attention mechanism from attending to the same words
- Unlikelihood objective (Welleck et al., 2020)
  - Penalize generation of already-seen tokens

所以，用贪心算法的话其实并不合理

Are greedy methods reasonable?

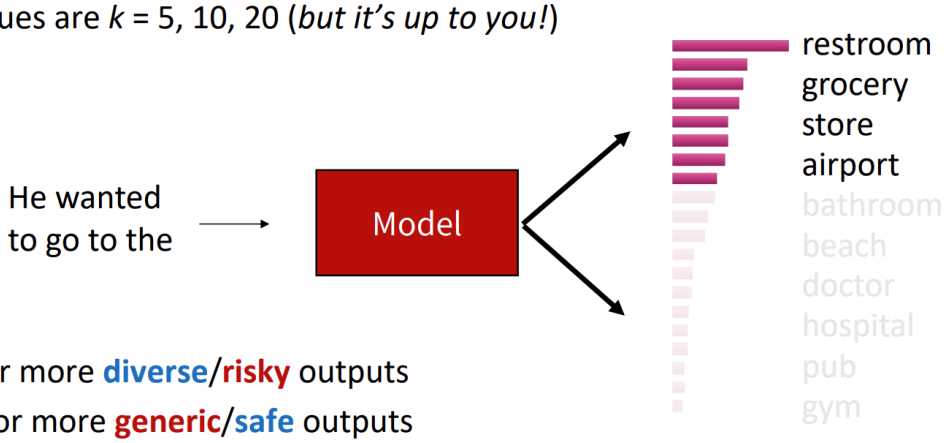


如果人类遵循这个，那日常交流就没有必要了，因为我们都知道说话的人下一个说出的单词😁

解决方法：sampling

Decoding: Top-*k* sampling

- Solution: Top-*k* sampling
  - Only sample from the top *k* tokens in the probability distribution
  - Common values are *k* = 5, 10, 20 (*but it's up to you!*)



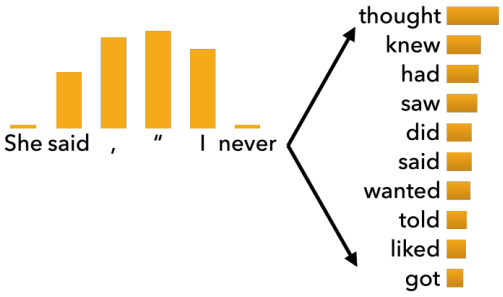
  - Increase *k* for more diverse/risky outputs
  - Decrease *k* for more generic/safe outputs

34

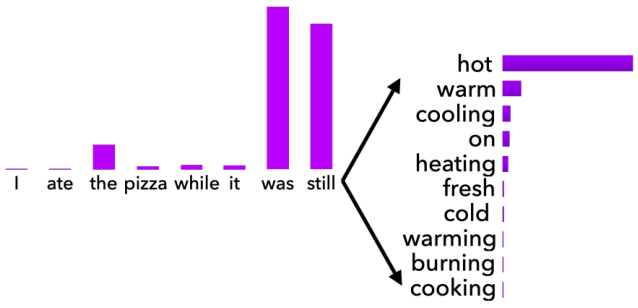
但是还是存在两个问题，过早截断和过慢截断

- 过早截断指的是在生成过程中，Top-k抽样过早地将概率较低的候选项排除在外，导致可能存在更有意义或更合适的候选项被过早地剔除。这可能会导致生成的文本缺乏多样性或创造力，过于保守或重复。
- 相反，过慢截断指的是Top-k抽样过于宽松，将概率较低的候选项保留在选择范围内的时间较长。这可能导致生成的文本存在不相关或无意义的内容，或者延长生成过程，导致生成速度较慢。

Issues with Top-*k* sampling



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



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

解决方法

Decoding: Top-*p* (nucleus) sampling

- Solution: Top-*p* sampling
  - Sample from all tokens in the top *p* cumulative probability mass (i.e., where mass is concentrated)
  - Varies *k* depending on the uniformity of P_t

$P_t^1(y_t = w \mid \{y\}_{<t})$ $P_t^2(y_t = w \mid \{y\}_{<t})$ $P_t^3(y_t = w \mid \{y\}_{<t})$

Scaling randomness: Softmax temperature

- Recall: On timestep *t*, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

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

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

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

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - **More** diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - **Less** diverse output (probability is concentrated on top words)

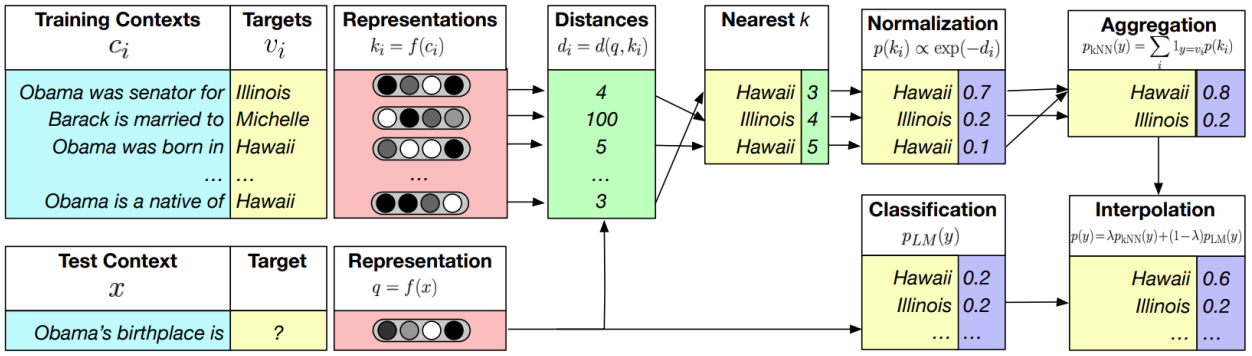
Note: softmax temperature is not a decoding algorithm!

It's a technique you can apply at test time, in conjunction with a decoding algorithm (such as beam search or sampling)

动态调整，使用来自n-gram短语统计的检索来重新平衡概率分布：

Improving decoding: re-balancing distributions

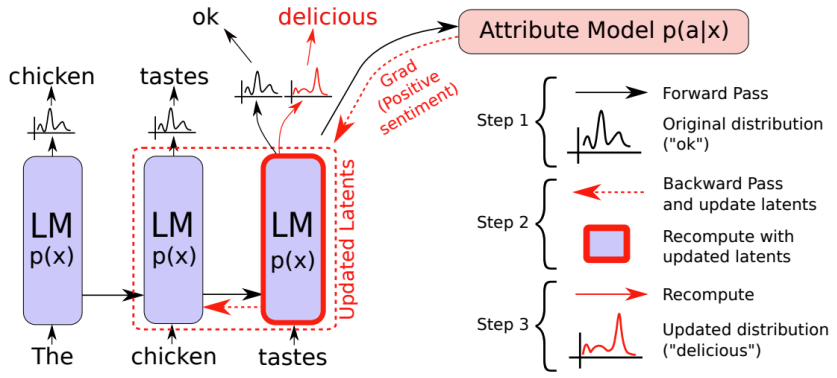
- Problem: What if I don't trust how well my model's distributions are calibrated?
  - Don't rely on **ONLY** your model's distribution over tokens
- Solution #1: Re-balance P_t using retrieval from n-gram phrase statistics!



反向传播调整

Backpropagation-based distribution re-balancing

- Can I re-balance my language model's distribution in to encourage other behaviors?
  - Yes! Just define a model that evaluates that behavior (e.g., sentiment, perplexity)
  - Use soft token distributions (e.g., Gumbel Softmax -- P_t with tiny temperature τ) as inputs to the evaluator
  - Backpropagate gradients directly to your language model and update P_t



(Dathathri et. al., ICLR 2020; Qin et al., EMNLP 2

or定义一个评分函数来近似衡量序列的质量，并根据该评分对候选序列进行重新排名

Improving Decoding: Re-ranking

- Problem: What if I decode a bad sequence from my model?

- Decode a bunch of sequences
  - 10 candidates is a common number, but it's up to you
- Define a score to approximate quality of sequences and re-rank by this score
  - Simplest is to use perplexity!
    - Careful! Remember that repetitive methods can generally get high perplexity.
  - Re-rankers can score a variety of properties:
    - 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…
    - Beware poorly-calibrated re-rankers
  - Can use multiple re-rankers in parallel

Coreference Resolution

Coreference Resolution（指代消解）是自然语言处理中的一个任务，旨在确定文本中的代词（pronouns）或名词短语（noun phrases）所指代的具体实体或其他短语。代词和名词短语可能在文本中多次出现，而核心指代消解任务的目标是将它们与其对应的先行词（antecedents）进行关联。

4. On to Coreference! First, some linguistics

- **Coreference** is when two mentions refer to the same entity in the world
  - *Barack Obama traveled to … Obama …*

- A different-but-related linguistic concept is **anaphora:** when a term (anaphor) refers to another term (antecedent)
  - the interpretation of the anaphor is in some way determined by the interpretation of the antecedent
  - *Barack Obama said he would sign the bill.*
    antecedent anaphor

这一页PPT是关于核心指代消解（Coreference Resolution）的介绍，首先解释了一些与语言学相关的概念。

1. 核心指代消解是指当文本中的两个提及（mentions）指代世界上的同一实体时发生的情况。例如，“Barack Obama traveled to … Obama ...”中的两个提及都指代同一个实体，即巴拉克·奥巴马。

2. 另一个与核心指代消解相关的语言学概念是“回指”（anaphora），它指的是一个术语（回指词）指代另一个术语（先行词）的情况。回指词的解释在某种程度上由先行词的解释决定。

3. 例如，“Barack Obama said he would sign the bill.”中的“he”是一个回指词，它指代先行词“Barack Obama”。回指词“he”的解释与先行词“Barack Obama”的解释有关。

这一页PPT主要介绍了核心指代消解的概念和与之相关的语言学概念。核心指代消解是在自然语言处理中重要的任务之一，旨在识别文本中不同提及之间的指代关系，以提高文本理解和生成的准确性。

Not all anaphoric relations are coreferential

- Not all noun phrases have reference

- *Every dancer* twisted *her knee*.
- *No dancer* twisted *her knee.*

- There are three NPs in each of these sentences; because the first one is non-referential, the other two aren’t either.

并非所有的回指关系都是核心指代关系。

在一些句子中，不是所有的名词短语都具有参照关系。例如：

1. "Every dancer twisted her knee."（每个舞者都扭伤了她的膝盖。）
   在这个句子中，有三个名词短语（dancer、her、knee），但第一个名词短语（dancer）并没有特定的参照对象，它是泛指的，表示所有的舞者。因此，其他两个名词短语（her、knee）也不具有参照关系。

2. "No dancer twisted her knee."（没有一个舞者扭伤了她的膝盖。）
   在这个句子中，同样有三个名词短语（dancer、her、knee），但由于第一个名词短语（dancer）表示否定，即没有任何舞者，因此其他两个名词短语（her、knee）也没有特定的参照对象。

因此，这些例子中的回指关系并非核心指代关系。并非所有的回指关系都涉及到具体的实体或短语，有时候回指仅仅是在语言表达中的一种修辞手法或语法结构。在理解和处理文本时，需要注意区分核心指代关系和其他类型的回指关系。

Hobbs' naive algorithm(1976) [现在没怎么用了]

5. Traditional pronominal anaphora resolution: Hobbs' naive algorithm

1. Begin at the NP immediately dominating the pronoun
2. Go up tree to first NP or S. Call this X, and the path p.
3. Traverse all branches below X to the left of p, left-to-right, breadth-first. Propose as antecedent any NP that has a NP or S between it and X
4. If X is the highest S in the sentence, traverse the parse trees of the previous sentences in the order of recency. Traverse each tree left-to-right, breadth first. When an NP is encountered, propose as antecedent. If X not the highest node, go to step 5.

Hobbs' naive algorithm (1976)

5. From node X, go up the tree to the first NP or S. Call it X, and the path p.
6. If X is an NP and the path p to X came from a non-head phrase of X (a specifier or adjunct, such as a possessive, PP, apposition, or relative clause), propose X as antecedent

   (The original said "did not pass through the N' that X immediately dominates", but the Penn Treebank grammar lacks N' nodes….)
7. Traverse all branches below X to the left of the path, in a left-to-right, breadth first manner. Propose any NP encountered as the antecedent
8. If X is an S node, traverse all branches of X to the right of the path but do not go below any NP or S encountered. Propose any NP as the antecedent.
9. Go to step 4

Knowledge-based Pronominal Coreference[必须建立在理解语义的基础上去分析]

Knowledge-based Pronominal Coreference

- She poured water from the pitcher into the cup until it was full.
- She poured water from the pitcher into the cup until it was empty.

- The city council refused the women a permit because they feared violence.
- The city council refused the women a permit because they advocated violence.
  - Winograd (1972)

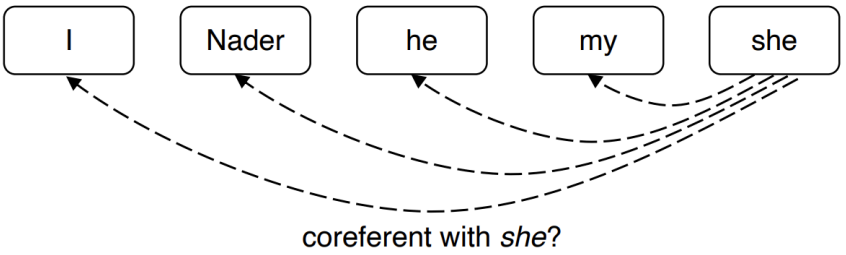
- These are called **Winograd Schema**
  - Recently proposed as an alternative to the Turing test
    - See: Hector J. Levesque "On our best behaviour" IJCAI 2013 http://www.cs.toronto.edu/~hector/Papers/ijcai-13-paper.pdf
    - http://commonsensereasoning.org/winograd.html
  - If you've fully solved coreference, arguably you've solved AI !!!

一个一个往前找

Coreference Models: Mention Pair

- Train a binary classifier that assigns every pair of mentions a probability of being coreferent: $p(m_i, m_j)$
  - e.g., for "she" look at all **candidate antecedents** (previously occurring mentions) and decide which are coreferent with it

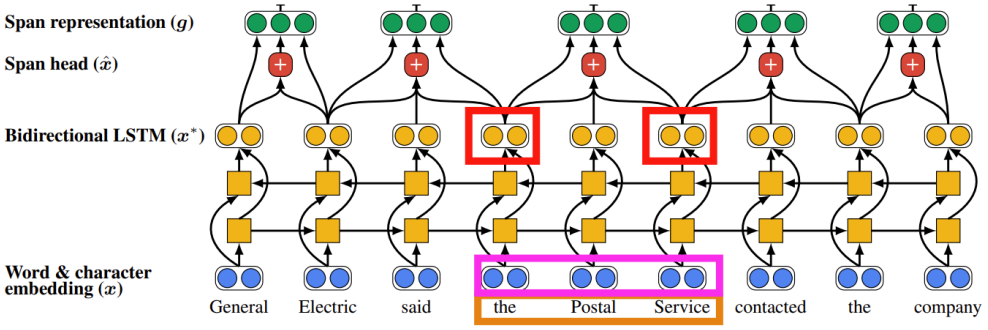
*"**I** voted for **Nader** because **he** was most aligned with **my** values," **she** said.*



上网络

End-to-end Model

- Next, represent each span of text *i* going from START(*i*) to END(*i*) as a vector
- For example, for "the postal service"



Span representation: $\boldsymbol{g}_i = [\boldsymbol{x}^*_{\text{START}(i)}, \boldsymbol{x}^*_{\text{END}(i)}, \hat{\boldsymbol{x}}_i, \phi(i)]$

BILSTM hidden states for span's start and end

Attention-based representation (details next slide) of the words in the span

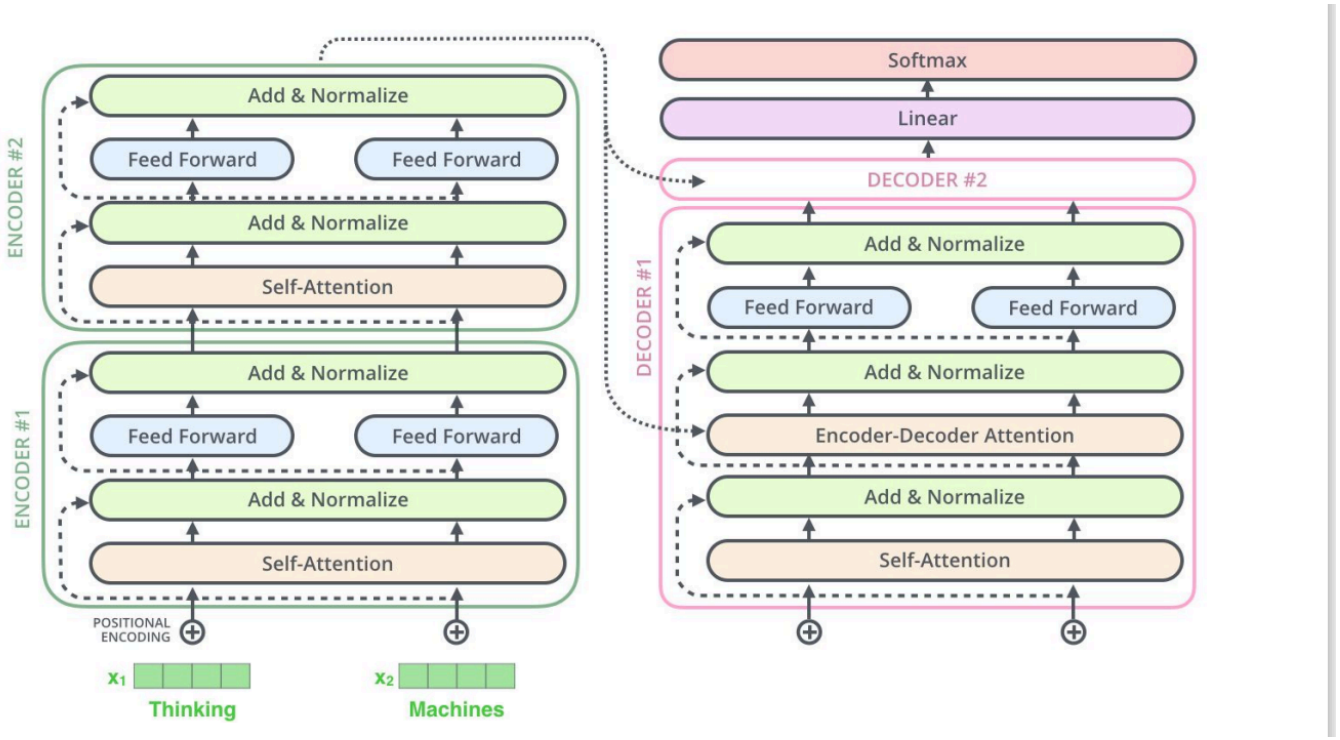
Additional features

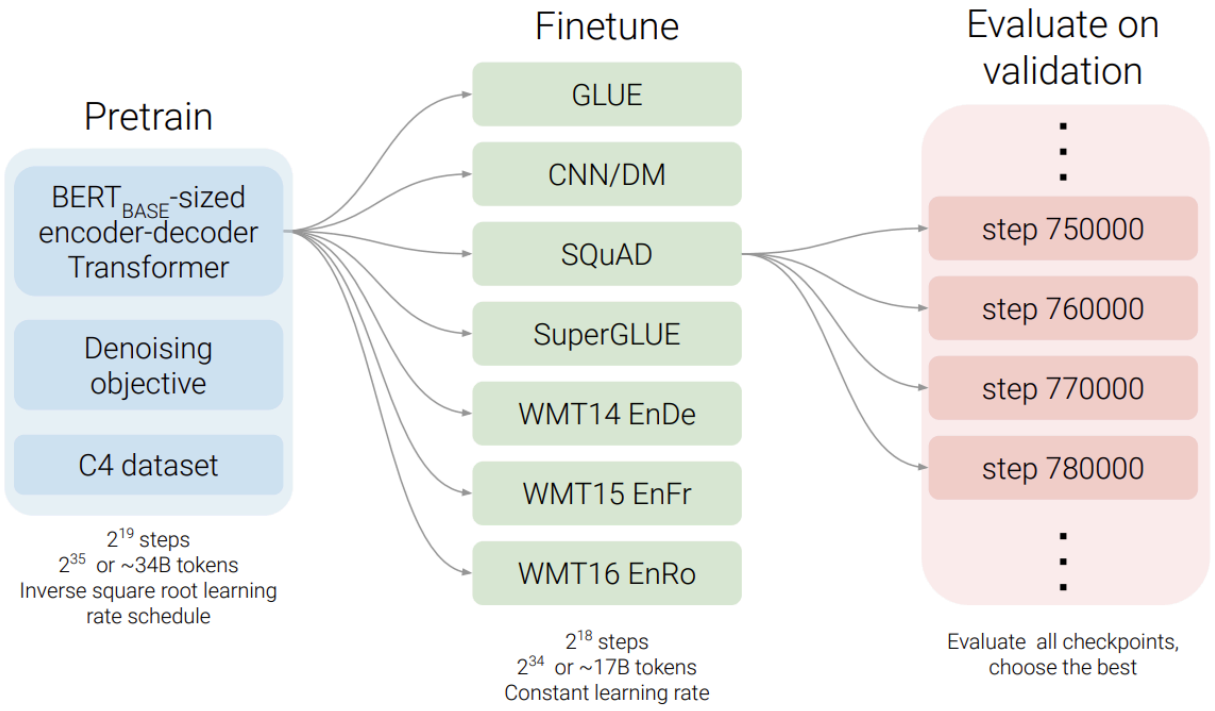
SOTA: base on bert

BERT-based coref: Now has the best results!

- Pretrained transformers can learn long-distance semantic dependencies in text.
- Idea 1, SpanBERT: Pretrains BERT models to be better at span-based prediction tasks like coref and QA
- Idea 2, BERT-QA for coref: Treat Coreference like a deep QA task
  - "Point to" a mention, and ask "what is its antecedent"
  - Answer span is a coreference link

T5 and Large Language Models





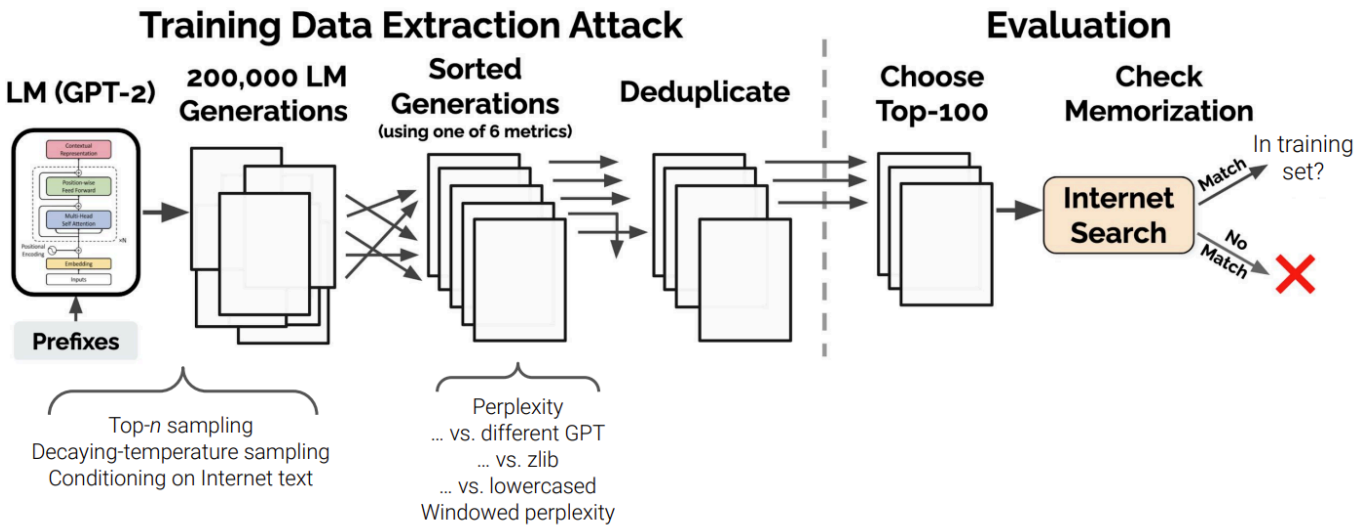
模型会记住并输出训练集的内容吗

"... the extent that a work is produced with a machine learning tool that was trained on a large number of copyrighted works, the degree of copying with respect to any given work is likely to be, at most, de minimis."

– Electronic Frontier Foundation

"Well-constructed AI systems generally do not regenerate, in any nontrivial portion, unaltered data from any particular work in their training corpus."

– OpenAI



测试实验，用网址填空看看GPT-2是否记住

URL (trimmed)	Occurrences		Memorized?		
	Docs	Total	XL	M	S
/r/[redacted]51y/milo_evacua...	1	359	✓	✓	½
/r/[redacted]zin/hi_my_name...	1	113	✓	✓	
/r/[redacted]7ne/for_all_yo...	1	76	✓	½	
/r/[redacted]5mj/fake_news_...	1	72	✓		
/r/[redacted]5wn/reddit_admi...	1	64	✓	✓	
/r/[redacted]lp8/26_evening...	1	56	✓	✓	
/r/[redacted]jla/so_pizzagat...	1	51	✓	½	
/r/[redacted]ubf/late_night...	1	51	✓	½	
/r/[redacted]eta/make_christ...	1	35	✓	½	
/r/[redacted]6ev/its_officia...	1	33	✓		
/r/[redacted]3c7/scott_adams...	1	17			
/r/[redacted]k2o/because_his...	1	17			
/r/[redacted]tu3/armynavy_ga...	1	8			

Add Knowledge to Language Models

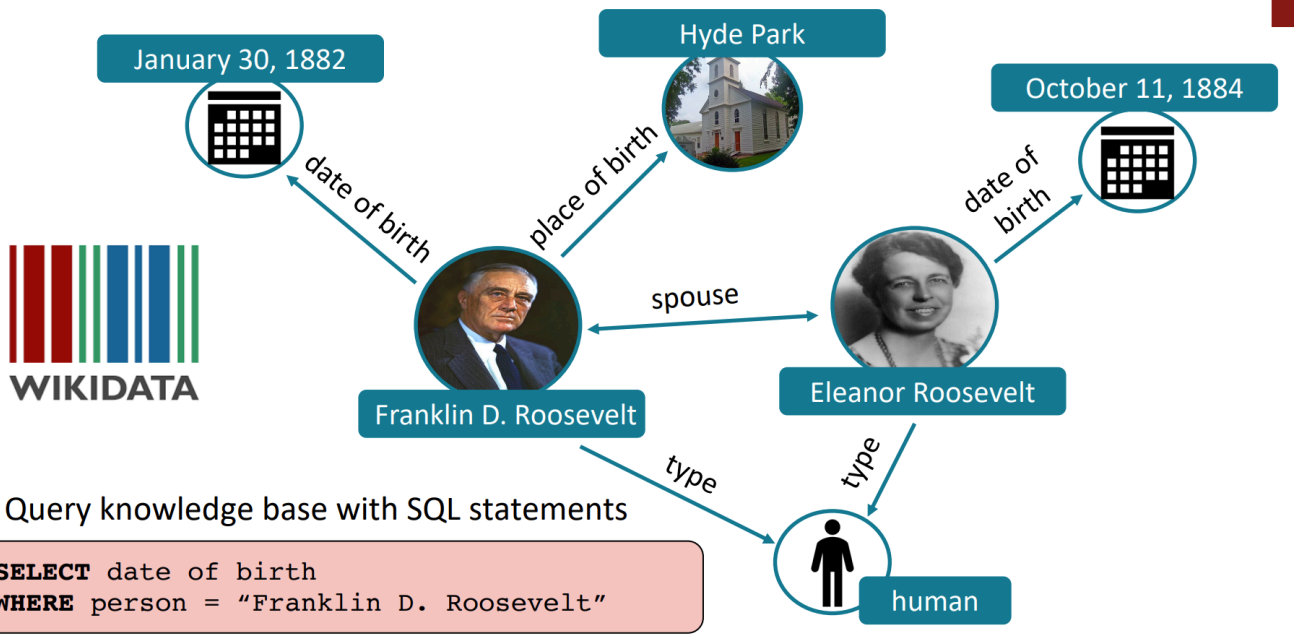
What does a language model know?

- Takeaway: predictions generally make sense (e.g. the correct types), but are not all factually correct.
- Why might this happen?
  - Unseen facts: some facts may not have occurred in the training corpora at all
  - Rare facts: LM hasn't seen enough examples during training to memorize the fact
  - Model sensitivity: LM may have seen the fact during training, but is sensitive to the phrasing of the prompt
    - Correctly answers "x was *made* in y" templates but not "x was *created* in y"
- The inability to *reliably* recall knowledge is a key challenge facing LMs today!
  - Recent works have found LMs can recover *some* knowledge, but have a way to go.

7

Querying traditional knowledge bases

Querying traditional knowledge bases



- Query knowledge base with SQL statements

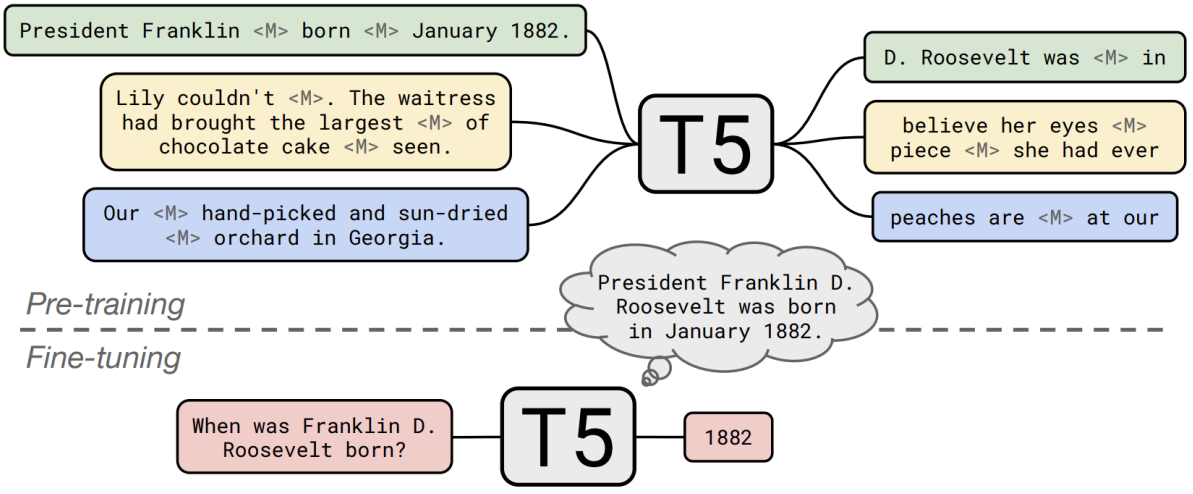
```
SELECT date of birth
WHERE person = "Franklin D. Roosevelt"
```

9

Querying language models as knowledge bases

Querying language models as knowledge bases

- Pretrain LM over unstructured text and then query with natural language.



Roberts et al., EMNLP 2020

10

LMs相对于KB的优点

Advantages of language models over traditional KBs

- LMs are pretrained over large amounts of unstructured and unlabeled text
  - KBs require manual annotation or complex NLP pipelines to populate
- LMs support more flexible natural language queries
  - Example: *What does the final F in the song U.F.O.F. stand for?*
    - Traditional KB wouldn't have a field for "final F"; LM *may* learn this
- However, there are also many open challenges to using LMs as KBs:
  - Hard to interpret (i.e., why does the LM produce an answer)
  - Hard to trust (i.e., the LM may produce a realistic, incorrect answer)
  - Hard to modify (i.e., not easy to remove or update knowledge in the LM)

Petroni et al., EMNLP 2019 & Roberts et al., EMNLP 2020

Techniques to add knowledge to LMs

(1)加上实体嵌入

Method 1: Add pretrained entity embeddings

- Facts about the world are usually in terms of entities
  - Example: Washington was the first president of the United States.
- Pretrained word embeddings do **not** have a notion of entities
  - Different word embeddings for "U.S.A.", "United States of America" and "America" even though these refer to the same entity
- What if we assign an embedding per entity?
  - Single entity embedding for "U.S.A.", "United States of America" and "America"
- Entity embeddings can be useful to LMs *iff* you can do entity linking well!

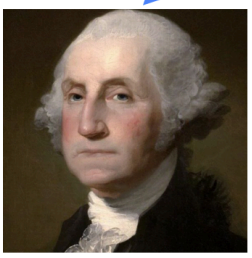
Aside: What is entity linking?

- Link mentions in text to entities in a knowledge base

mention *mention*

Washington was the first president of the United States.

candidate *candidate* *candidate*



Q23 (Wikidata) Q1223 (Wikidata) Q30 (Wikidata)

- Entity linking tells us which entity embeddings are relevant to the text

More resources: Orr et al., CIDR 2021 & Li et al., EMNLP 2020

（2）用外部知识库

Method 2: Use an external memory

- Previous methods rely on the pretrained entity embeddings to encode the factual knowledge from KBs for the language model.

- Question: Are there more direct ways than pretrained entity embeddings to provide the model factual knowledge?
- Answer: Yes! Give the model access to an external memory (a key-value store with access to KG triples or context information)

- Advantages:
  - Can better support injecting and updating factual knowledge
    - Often without more pretraining!
  - More interpretable

（3）

Method 3: Modify the training data

- Previous methods incorporated knowledge explicitly through pretrained embeddings and/or an external memory.
- Question: Can knowledge also be incorporated implicitly through the unstructured text?
- Answer: Yes! Mask or corrupt the data to introduce additional training tasks that require factual knowledge.
- Advantages:
  - No additional memory/computation requirements
  - No modification of the architecture required

总结

Recap: Techniques to add knowledge to LMs

1. Use pretrained entity embeddings
   - Often not too difficult to apply to existing architectures to leverage KG pretraining
   - Indirect way of incorporating knowledge and can be hard to interpret
2. Add an external memory
   - Can support some updating of factual knowledge and easier to interpret
   - Tend to be more complex in implementation and require more memory
3. Modify the training data
   - Requires no model changes or additional computation. May also be easiest to theoretically analyze! Active area of research
   - Still open question if this is always as effective as model changes