Lec1

3. Word2vec: Overview

Word2vec is a framework for learning word vectors
(Mikolov et al. 2013)

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* 中心词和上下文
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Word2Vec: objective function

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

给定中心词，上下文窗口的概率

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

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

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*:

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

- 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!

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have →
- Remember: every word has two vectors

一个作为中心词，一个作为上下文

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

Stochastic Gradient Descent

SGD随机梯度下降

- **Problem**: $J(\theta)$ is a function of **all** windows in the corpus (potentially 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
- Algorithm:

按批次进行梯度下降

Mini Batch Gradient Descent

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

Lec2

The skip-gram model with negative sampling (Mikolov et al. 2013)

负采样

- We take *k* negative samples (using word probabilities)
- Maximize probability that real outside word appears;
  minimize probability that random words appear around center word
- Using notation consistent with this class, we minimize:

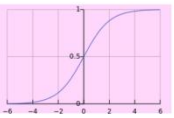
$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K\ sampled\ indices\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

大　　小

sigmoid rather than softmax

- The logistic/sigmoid function:
  (we'll become good friends soon)

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



平衡高频词和低频词，提高低频词的概率

- Sample with $P(w)=U(w)^{3/4}/Z$, the unigram distribution U(w) raised to the 3/4 power
  - The power makes less frequent words be sampled more often

负采样的核心思想是：

- **正样本**：对于给定的中心词，其上下文词被视为正样本。
- **负样本**：从词汇表中随机采样一些非上下文词作为负样本。

通过这种方式，模型的目标变为：

- **最大化正样本出现的概率**：即中心词与其上下文词之间的关联性。
- **最小化负样本出现的概率**：即中心词与随机采样的非上下文词之间的关联性。

这种策略大大减少了计算量，因为只需要对少量的负样本进行计算，而不需要遍历整个词汇表。

---

3. 负采样的优势

(1) 减少计算量

- 在传统的 softmax 损失中，假设词汇表大小为 V，则需要对所有 V 个单词进行计算，时间复杂度为 $O(V)$。
- 而在负采样中，只需要计算正样本和少量负样本（通常为 k 个），时间复杂度降为 $O(k)$，其中 $k \ll V$。
- 这种减少使得模型在大规模词汇表上的训练变得可行。

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)

- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

共现矩阵

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

可以对共现矩阵做奇异值分解以降维

Hacks to X

- Running an SVD on raw counts doesn't work well!!!

- Scaling the counts in the cells can help ***a lot***
  - Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
    - log the frequencies
    - min(*X*, *t*), with *t* ≈ 100
    - Ignore the function words

    降低高频词的影响
    只关注实词
    加权，近的权重大
- Ramped windows that count closer words more than further away words
- Use correlations instead of counts, then set negative values to 0
- Etc.

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Stanfor

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

P(i | j)表示i和j共线概率

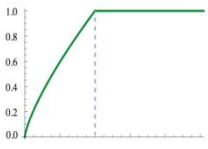
Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Loss: $J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$

Xij表示频率
频率越小权重越小

$f \sim$ 

- Fast training
- Scalable to huge corpora

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
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$ 词有多个意思 "稀疏编码"
- 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

Lec3

Simple NER: Window classification using binary logistic classifier

用二元逻辑斯蒂进行命名实体识别

- **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 手工标注
  - Really, we usually use multi-class softmax, but we're 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 .

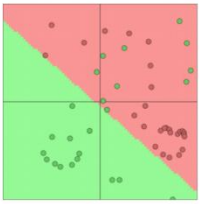
$X_{\text{window}} = [\ x_{\text{museums}}\ \ x_{\text{in}}\ \ x_{\text{Paris}}\ \ x_{\text{are}}\ \ x_{\text{amazing}}\ ]^T$

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

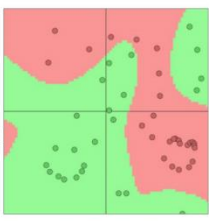
Neural classification

传统ML：只学参数W，往往是线性的

- Typical ML/stats softmax classifier: $p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C}\exp(W_c.x)}$
- Learned parameters θ are just elements of *W* (not input representation *x*, which has sparse symbolic features)
- Classifier gives linear decision boundary, which can be limiting



- A **neural network classifier** differs in that: 神经网络还能学习输入的分布 把输入映射到中间层 多层网络-非线性
  - We learn **both** *W* **and (distributed!)** representations for words
  - The word vectors *x* re-represent one-hot vectors, moving them around in an intermediate layer vector space, for easy classification with a (linear) softmax classifier
    - Conceptually, we have an embedding layer: $x = Le$
  - We use deep networks—more layers—that let us re-represent and compose our data multiple times, giving a non-linear classifier



But typically, it is linear relative to the pre-final layer representation

NER: Binary classification for center word being location

- We do supervised training and want high score if it's a location

$$J_t(\theta) = \sigma(s) = \frac{1}{1+e^{-s}}$$

predicted model probability of class

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

其实是一层神经网络 $\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

$\boldsymbol{x}$ (input) $\in \mathbb{R}^{5d}$

x = [$x_{museums}$ x_{in} x_{Paris} x_{are} $x_{amazing}$]

Embedding of 1-hot words

18

Non-linearities, old and new

当很大或很小时梯度接近于0，计算困难

logistic ("sigmoid")

$$f(z) = \frac{1}{1+\exp(-z)}$$

tanh

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

hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$

(Rectified Linear Unit) ReLU

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

Leaky ReLU / Parametric ReLU

解决x<0时的问题
y=ax斜率可学习

tanh is just a rescaled and shifted sigmoid (2 × as steep, [−1,1]):

$$\tanh(z) = 2\text{logistic}(2z) - 1$$

Logistic and tanh are still used (e.g., logistic to get a probability)

However, now, for deep networks, the first thing to try is ReLU: it trains quickly and performs well due to good gradient backflow.

ReLU has a negative "dead zone" that recent proposals mitigate

GELU is frequently used with Transformers (BERT, RoBERTa, etc.)

Swish arXiv:1710.05941

$$\text{swish}(x) = x \cdot \text{logistic}(x)$$

GELU arXiv:1606.08415

$$\text{GELU}(x) = x \cdot P(X \le x), X \sim N(0,1) \approx x \cdot \text{logistic}(1.702x)$$

Training with "cross entropy loss" – you use this in PyTorch!

- Until now, our objective was stated as to maximize the probability of the correct class *y* or equivalently we can minimize the negative log probability of that class
- Now restated in terms of cross entropy, a concept from information theory
- Let the true probability distribution be *p; l*et our computed model probability be *q*
- The cross entropy is:

$$H(p,q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

交叉熵
p(c)是真实概率，独热编码中只有正确的是1
可以简化为-logq(c)

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else, *p* = [0, …, 0, 1, 0, …, 0], then:
- **Because of one-hot *p*, the only term left is the negative log probability of the true class y_i:** $-\log p(y_i|x_i)$

Cross entropy can be used in other ways with a more interesting *p*, but for now just know that you'll want to use it as the loss in PyTorch

神经网络可以看作多个逻辑回归模型的并行运行，将上一层的输出作为输入

Jacobian Matrix: Generalization of the Gradient

- Given a function with ***m* outputs** and *n* inputs

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)]$$

- It's Jacobian is an ***m* x *n* matrix** of partial derivatives

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^T$$

Fine print: This is the correct Jacobian. Later we discuss the "shape convention"; using it the answer would be ***h***.

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$
$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$
$$= \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

⊙ = Hadamard product = element-wise multiplication of 2 vectors to give vector

对于两个相同维度的矩阵或向量 A 和 B，它们的 Hadamard 乘积记为 $A \odot B$，其结果是一个与 A 和 B 维度相同的矩阵或向量，其中每个元素是对应位置元素的乘积。

具体来说：

Hadamard乘积：逐元素相乘

- 如果 $A = [a_{ij}]$ 和 $B = [b_{ij}]$ 是两个矩阵，则

$$(A \odot B)_{ij} = a_{ij} \cdot b_{ij}$$

- 如果 $\mathbf{u} = [u_1, u_2, \ldots, u_n]$ 和 $\mathbf{v} = [v_1, v_2, \ldots, v_n]$ 是两个向量，则

$$(\mathbf{u} \odot \mathbf{v})_i = u_i \cdot v_i$$

- Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$
$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

在反向传播时
记录上游导数
避免重复计算

The same! Let's avoid duplicated computation …

- What does $\frac{\partial s}{\partial \boldsymbol{W}}$ look like? $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to then do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

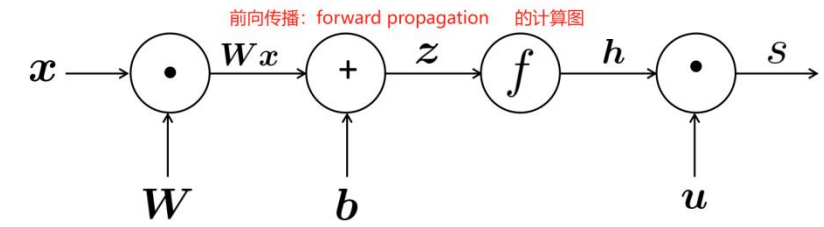
- Instead, we leave pure math and use the **shape convention**: the shape of the gradient is the shape of the parameters!
  - So $\frac{\partial s}{\partial \boldsymbol{W}}$ is *n* by *m*: $\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$

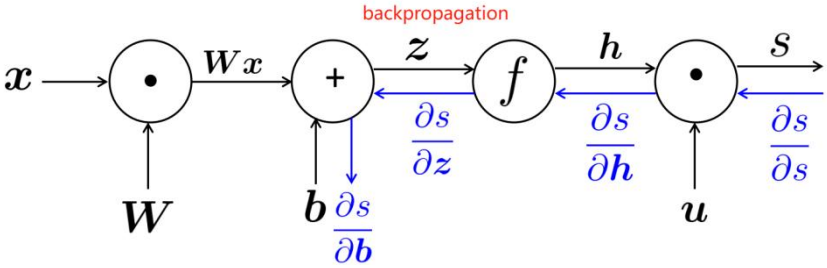
根据矩阵微分的规则，$\frac{\partial s}{\partial W}$ 的形状取决于 s 和 W 的维度：

- s 是一个 n -维向量。
- W 是一个 $n \times m$ 矩阵。

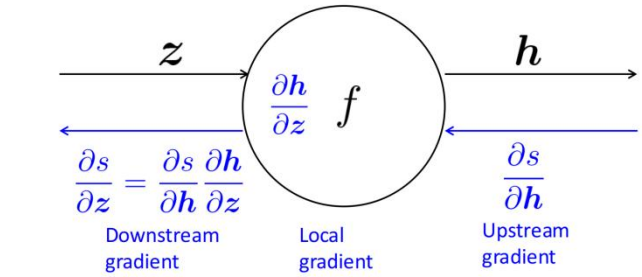
因此，$\frac{\partial s}{\partial W}$ 的形状是 $n \times (n \times m)$ 或者可以理解为一个 $n \times n \times m$ 的三维张量。但在实际应用中，为了方便处理，通常会将 $\frac{\partial s}{\partial W}$ 表示为与 W 同样的形状 $n \times m$，这被称为“形状约定”(shape convention)。

为了方便梯度下降，将结果表示为与参数的形状相同



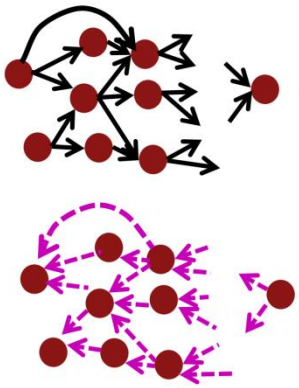


[downstream gradient] = [upstream gradient] x [local gradient]



Automatic Differentiation

自动微分



- The gradient computation can be automatically inferred from the symbolic expression of the fprop
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you but mainly leave layer/node writer to hand-calculate the local derivative

1. 自动微分的核心思想

自动微分的核心思想是：**从函数的前向传播（fprop）的符号表达式中自动推导出梯度**。这意味着，当我们定义了一个神经网络的前向计算过程时，自动微分工具可以自动计算该网络的梯度，而不需要我们手动推导复杂的数学公式。

关键点：

- **前向传播**（fprop）：表示神经网络的正向计算过程，即从输入到输出的计算。
- **反向传播**（backpropagation）：基于链式法则，从输出层逐层向输入层传递误差和梯度的过程。

自动微分通过将前向传播中的每个操作分解为基本的数学运算，并利用这些运算的导数规则，自动构建梯度的计算图。

Lec4

句子中的不同成分可以以不同的方式附着到句子中，导致歧义

PP attachment ambiguities multiply

- A key parsing decision is how we 'attach' various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.

介词短语　adv　不定式

The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto]

[for $27 a share]

[at its monthly meeting].

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with $n+2$ sides
    - Turns up in triangulation of probabilistic graphical models (CS228)….

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

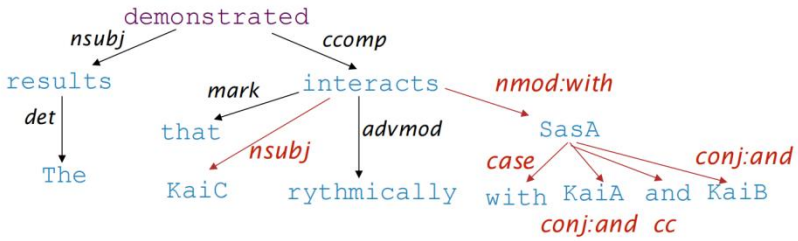
- Catalan 数的性质：
  - 它是一个随 n 指数增长的序列。
  - 在许多树状上下文中出现，例如：
    - 多边形的三角划分数量。
    - 概率图模型（Probabilistic Graphical Models）的三角化过程。

具体解释

- **多边形的三角划分**：
  - 对于一个有 $n+2$ 条边的多边形，其可能的三角划分数量为 C_n 。
  - 这个问题与 PP 附着的歧义性类似，因为两者都涉及到对结构的不同组合方式。
- **概率图模型的三角化**：
  - 在概率图模型中，三角化是一种将图转换为树状结构的方法，以简化计算。
  - Catalan 数也出现在这种场景中，因为三角化的可能性随着节点数量的增加而呈指数增长。

Dependency paths help extract semantic interpretation – simple practical example: extracting protein-protein interaction

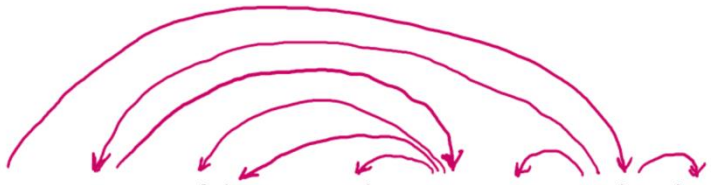
The results demonstrated that KaiC interacts rhythmically with SasA ,KaiA and KaiB



KaiC ←nsubj interacts nmod:with → SasA
KaiC ←nsubj interacts nmod:with → SasA conj:and→ KaiA
KaiC ←nsubj interacts nmod:with → SasA conj:and→ KaiB

[Erkan et al. EMNLP 07, Fundel et al. 2007, etc.]

Dependency Grammar and Dependency Structure



- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

The rise of annotated data 标注数据

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things

- Reusability of the labor 可复用：解析器、词性标注器
  - Many parsers, part-of-speech taggers, etc. can be built on it
  - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate NLP systems

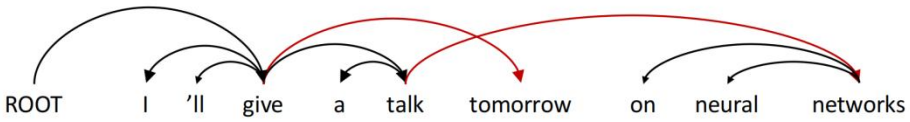
What are the straightforward sources of information for dependency parsing?

1. Bilexical affinities 双词亲和力 The dependency [discussion → issues] is plausible
2. Dependency distance 倾向于找距离近的词 Most dependencies are between nearby words
3. Intervening material 中间材料：支配者和从属者之间的词语或标点符号 Dependencies rarely span intervening verbs or punctuation
4. Valency of heads 支配者能够支配的从属者的类型和数量 How many dependents on which side are usual for a head?

- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of

- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don't want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (be non-projective) or not

投影性，依存是否交叉



Projectivity

- Definition of a projective parse: 依存关系弧在上方 There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies corresponding to a CFG tree must be projective 上下文无关语法树
  - I.e., by forming dependencies by taking 1 child of each category as head
- Most syntactic structure is projective like this, but dependency theory normally does allow non-projective structures to account for displaced constituents 大多数都是投影性的
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies 存在非投影性的，如宾语前置，疑问句等

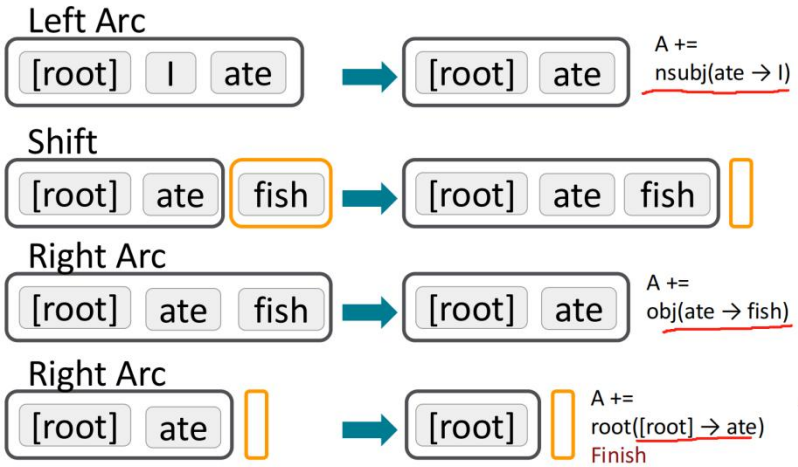
3. Methods of Dependency Parsing

1. Dynamic programming
   Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle
2. Graph algorithms
   You create a Minimum Spanning Tree for a sentence 可能忽略一些上下文关系
   McDonald et al.'s (2005) $O(n^2)$ MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)
   Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!
3. Constraint Satisfaction
   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc. 严格满足规则
4. "Transition-based parsing" or "deterministic dependency parsing"
   Greedy choice of attachments guided by good machine learning classifiers 贪心：局部最优
   E.g., MaltParser (Nivre et al. 2008). Has proven highly effective. And fast.

Start: $\sigma = [ROOT], \beta = w_1, \ldots, w_n, A = \emptyset$ shift：一个单词入栈

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j, w_i)\}$ wi弹出，与wj建立依赖关系
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i, w_j)\}$ wj弹出，与wi建立依赖关系

Finish: $\sigma = [w], \beta = \emptyset$



可以用机器学习的方法，在每个时刻根据堆栈情况、缓冲区情况、已建立的依存弧等特征预测下一步进行的操作，时间复杂度是 O(n)

- But you can profitably do a beam search if you wish (slower but better):
  - You keep *k* good parse prefixes at each time step 束搜索：保留k个最好的

传统的解析器的问题

Categorical features are:

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive to compute

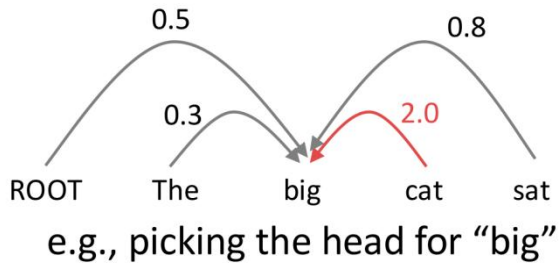
Neural Approach:

learn a dense and compact feature representation

This work was further developed and improved by others, including in particular at Google

- Bigger, deeper networks with better tuned hyperparameters 更大规模的神经网络
- Beam search 束搜索
- Global, conditional random field (CRF)-style inference over the decision sequence 条件随机场

- Repeat the same process for each other word; find the best parse (MST algorithm)



e.g., picking the head for “big”

基于图的依赖解析器
计算每两个词之间的依赖得分
可以借助最小生成树算法

但时间复杂度是 O(n^2)

Lec5

Parameter Initialization

- You normally must initialize weights to small random values (i.e., not zero matrices!) 权重不能初始化为0
  - 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):

使激活值的方差稳定 $$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Optimizers

- Usually, plain SGD will work just fine!
  - However, getting good results will often require hand-tuning the learning rate
    - See next slide
- For more complex nets and situations, or just to avoid worry, you often do better with one of a family of more sophisticated "adaptive" optimizers that scale the parameter adjustment by an accumulated gradient.
  - These models give differential per-parameter learning rates
    - Adagrad
    - RMSprop
    - Adam ← A fairly good, safe place to begin in many cases
    - SparseAdam
    - …

Learning Rates

- You can just use a constant learning rate. Start around *lr* = 0.001?
  - It must be order of magnitude right – try powers of 10
    - Too big: model may diverge or not converge 过大：发散，无法收敛
    - Too small: your model may not have trained by the assignment deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every *k* epochs
    - An epoch = a pass through the data (shuffled or sampled – not in same order each time)
  - By a formula: $lr = lr_0 e^{-kt}$, for epoch t 逐渐减小
  - There are fancier methods like cyclic learning rates (q.v.) 循环
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so you may want to start with a higher learning rate

Definition: A *n*-gram is a chunk of *n* consecutive words.

- unigrams: "the", "students", "opened", "their"
- bigrams: "the students", "students opened", "opened their"
- trigrams: "the students opened", "students opened their"
- 4-grams: "the students opened their"

用 markov 性进行预测

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then *w* has probability 0!

平滑：加δ使分子不为0

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any w*!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

当这个组合从未出现时，后退

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

困惑度

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

Normalized by number of words

Inverse probability of corpus, according to Language Model

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

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

等价于交叉熵

Lower perplexity is better!

n-gram 预测基本符合语法，但 dont make sense

A fixed-window neural Language Model

output distribution

$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden layer

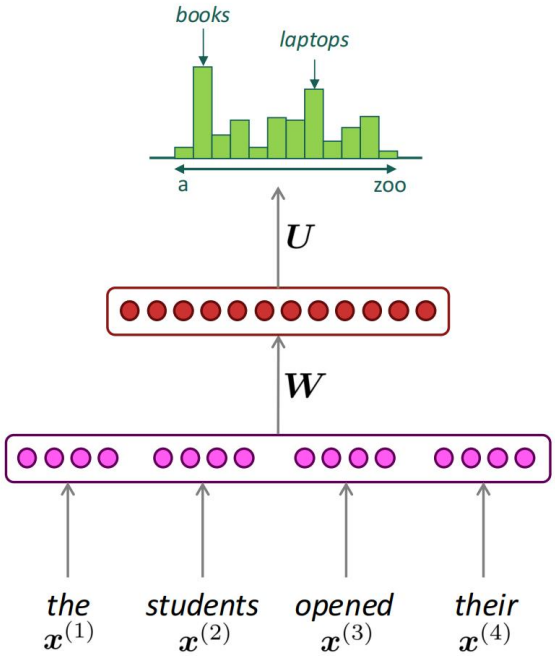
$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$

concatenated word embeddings

$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$

words / one-hot vectors

$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$



Improvements over *n*-gram LM:

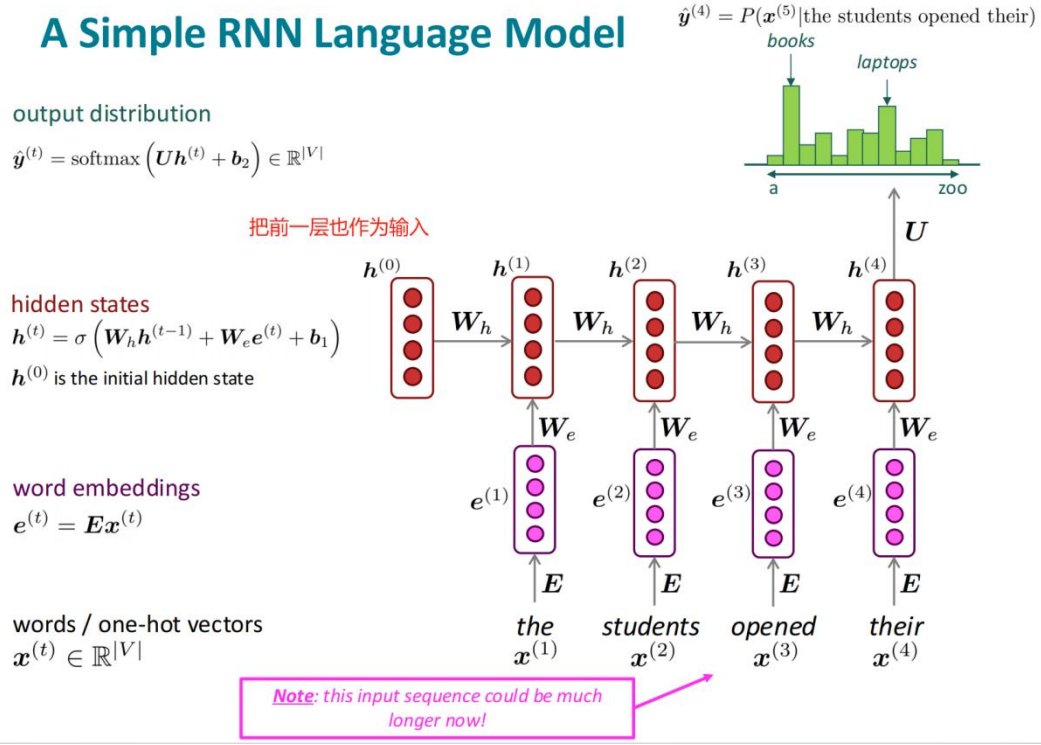
- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed. 损失了一些位置和上下文信息

We need a neural architecture that can process *any length input*

A Simple RNN Language Model



RNN **Advantages**:

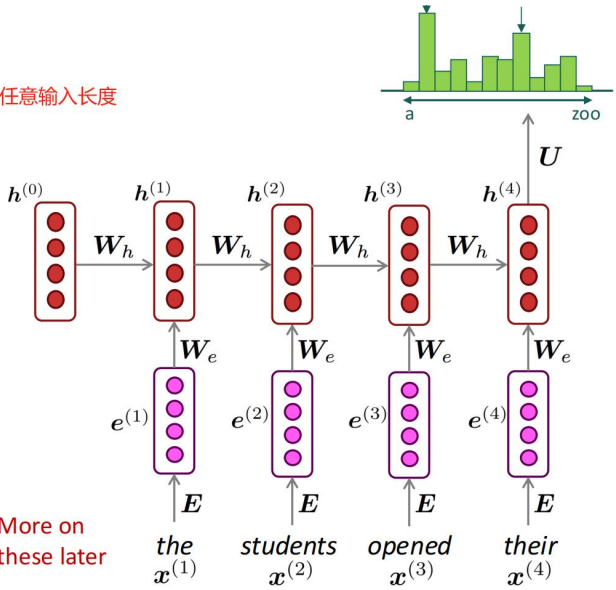
- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

支持任意输入长度

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later



28

Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

交叉熵

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

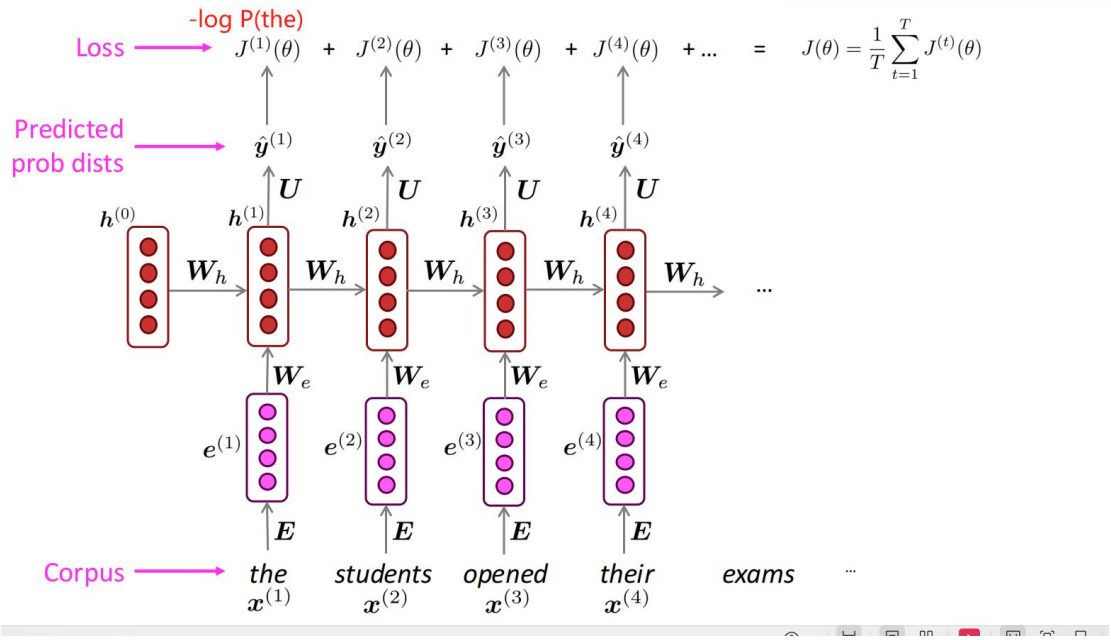
预测真实单词的概率

Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}$$

Training an RNN Language Model

"Teacher forcing"



Teacher Forcing 是指在训练 RNN 时，在每一步输入真实的前一步目标值，而不是使用模型自己在前一步预测出来的输出，从而减小误差传播。

🛠️ **如何缓解 Teacher Forcing 的问题?**

为了缓解训练和推理之间的差异，可以使用一些改进策略，例如：

1. Scheduled Sampling（**计划采样**）：在训练过程中，逐渐从使用真实标签过渡到使用模型预测的输出。
2. Professor Forcing：通过对抗训练的方式，让模型在训练时也能适应使用自己预测的输出。
3. Beam Search 或 Sampling：在推理时使用更合理的解码策略，缓解模型对错误预测的敏感。

However: Computing loss and gradients across entire corpus $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ at once is too expensive (memory-wise)!

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

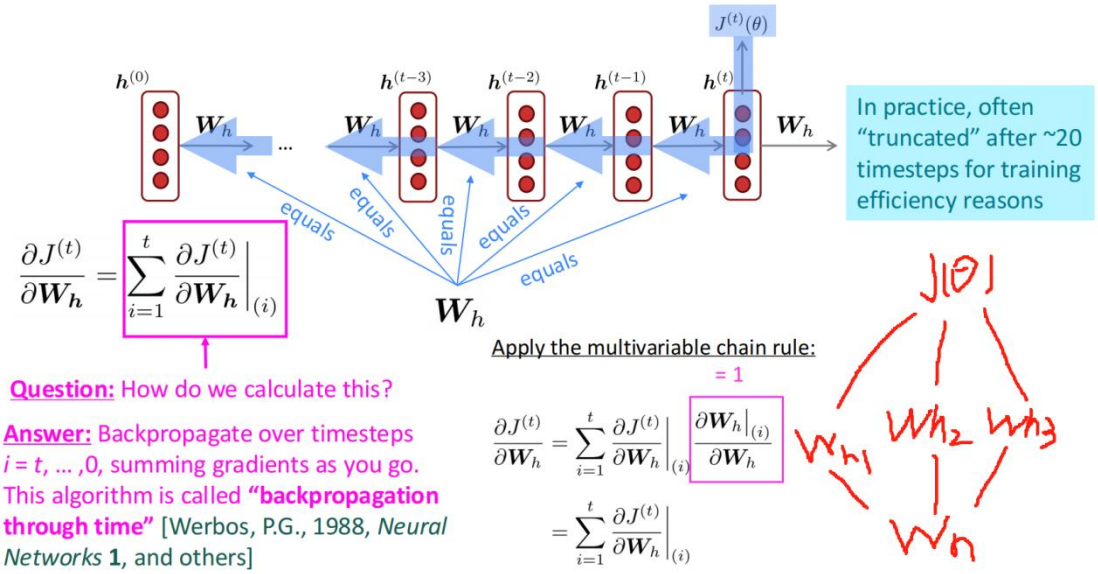
In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ as a sentence (or a document)

Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

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

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"



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

Question: How do we calculate this?

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

Apply the multivariable chain rule:

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

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

正向传播时，Wh 是常数

反向传播时，一步步更新 Wh 并求和

"**时间截断反向传播**"（Truncated Backpropagation Through Time，简称 TBPTT）是训练 **循环神经网络（RNN）** 时常用的一种优化策略。

🧠 什么是时间截断反向传播（TBPTT）？

定义：

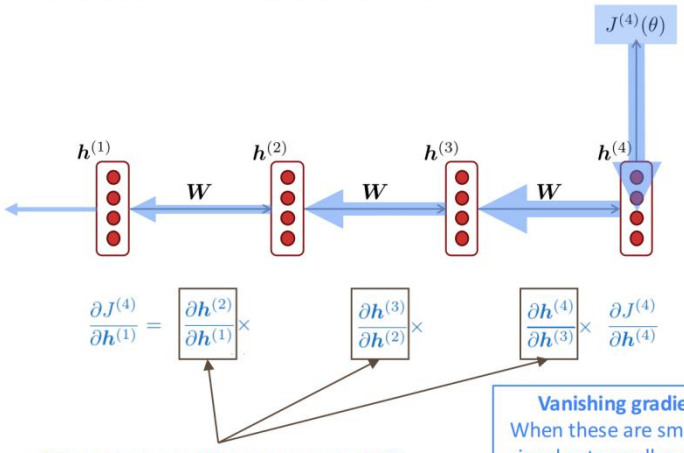
时间截断反向传播 是指在训练 RNN 时，**不是在整个序列上进行完整的反向传播计算梯度**，而是将长序列分成若干段，在每一段上进行反向传播。

每次只反向传播若干步

这样做的目的是为了：

- 减少内存消耗
- 加快训练速度
- 避免梯度消失/爆炸问题

Vanishing gradient intuition



What happens if these are small?

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Vanishing gradient proof sketch (linear case)

- Recall: $$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)$$
- What if σ were the identity function, $\sigma(x) = x$?

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}\left(\sigma'\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)\right) W_h \quad \text{(chain rule)}$$
$$= I\, W_h = W_h$$

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $h^{(j)}$ on some previous step j. *Let* $\ell = i - j$

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \quad \text{(chain rule)}$$

一个很小的数的指数

$$= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} W_h = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{\ell} \quad \text{(value of } \frac{\partial h^{(t)}}{\partial h^{(t-1)}}\text{)}$$

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

Vanishing gradient proof sketch (linear case)

- What's wrong with W_h^{ℓ} ?
- Consider if the eigenvalues of W_h are all less than 1: (sufficient but not necessary)

特征值 $\lambda_1, \lambda_2, \ldots, \lambda_n < 1$

$q_1, q_2, \ldots, q_n$ (eigenvectors)

- We can write $\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{\ell}$ using the eigenvectors of W_h as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{\ell} = \sum_{i=1}^{n} c_i \lambda_i^{\ell} q_i \approx 0 \text{ (for large } \ell\text{)}$$

Approaches 0 as ℓ grows, so gradient vanishes

- What about nonlinear activations σ (i.e., what we use?)
  - Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ

Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

导致远处的梯度信息丢失

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

梯度爆炸的解决方法

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

Lec6

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:

σ：sigmoid函数

遗忘门：哪些信息被保留

输入门：哪些新的信息被写入

输出门：哪些信息被输出到隐藏层h(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

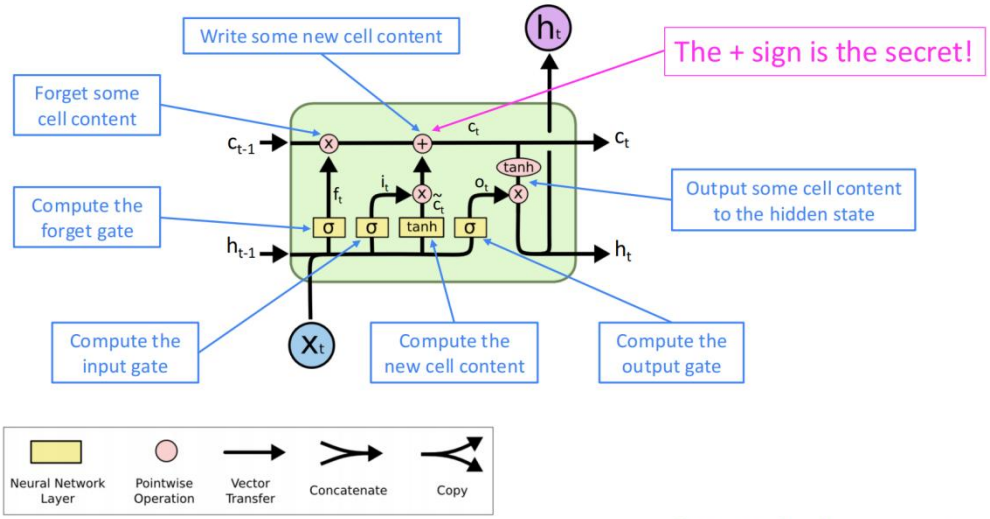
$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)}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

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: ⊙

7

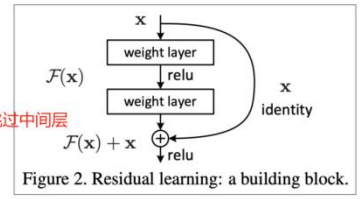
You can think of the LSTM equations visually like this:



梯度消失并非只在 RNN 中出现，但 RNN 特别不稳定

For example:

- Residual connections aka “ResNet”
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train



输入x直接跳过中间层

Figure 2. Residual learning: a building block.

Other methods:

- Dense connections aka “DenseNet”
- Directly connect each layer to all future layers!

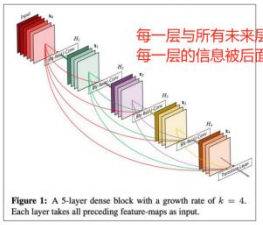
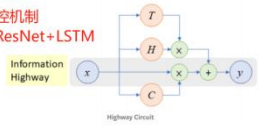

每一层与所有未来层相连
每一层的信息被后面所有层使用

Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- Highway connections aka “HighwayNet”
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks

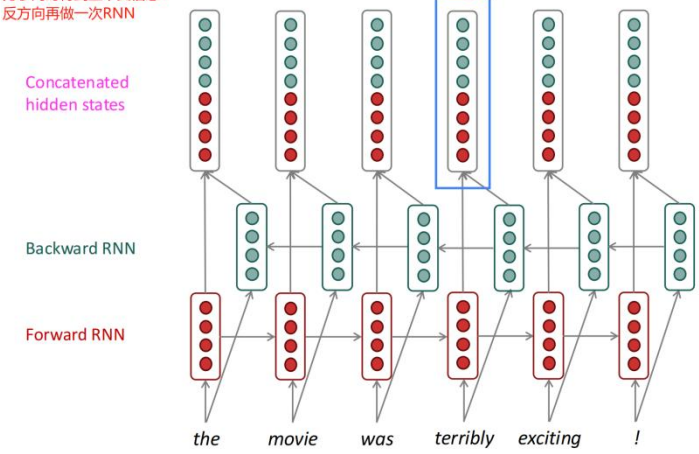
动态门控机制
类似于ResNet+LSTM



Bidirectional RNNs

为了同时得到上下文信息：
反方向再做一次RNN

This contextual representation of “terribly” has both left and right context!



On timestep t:

This is a general notation to mean “compute one forward step of the RNN” – it could be a simple RNN or LSTM computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

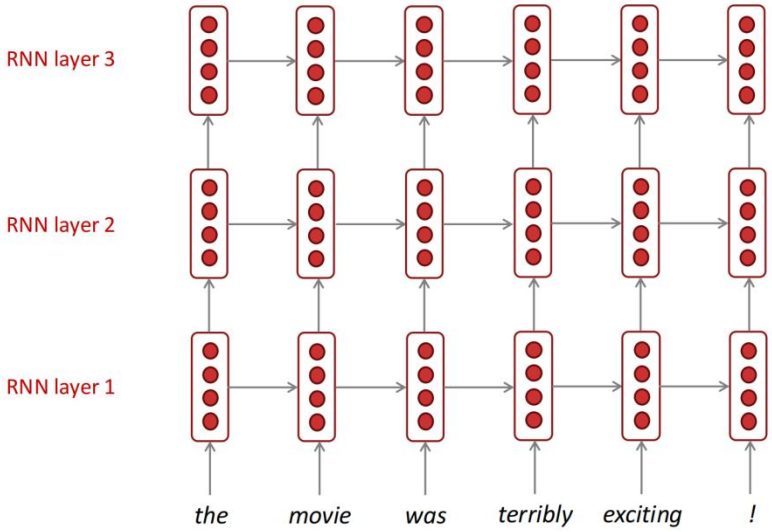
We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

16

两个方向的 RNN 必须有完整的句子

Multi-layer RNNs

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



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:

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

Translation Model

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

Language Model

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

Learning alignment for SMT

- We learn $P(x, a|y)$ as a combination of many factors, including:
  - Probability of particular words aligning (also depends on position in sent)
  - Probability of particular words having a particular fertility (number of corresponding words)
  - etc.

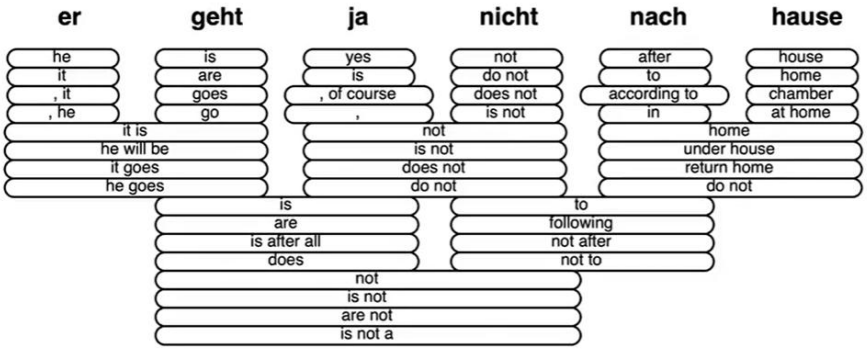
生育率：一个源单词可能对应多个目标单词

学习隐变量a表示x和y之间的对应关系：用EM算法

- Alignments *a* are **latent variables**: They aren't explicitly specified in the data!
  - Require the use of special learning algorithms (like Expectation-Maximization) for learning the parameters of distributions with latent variables

Decoding for SMT

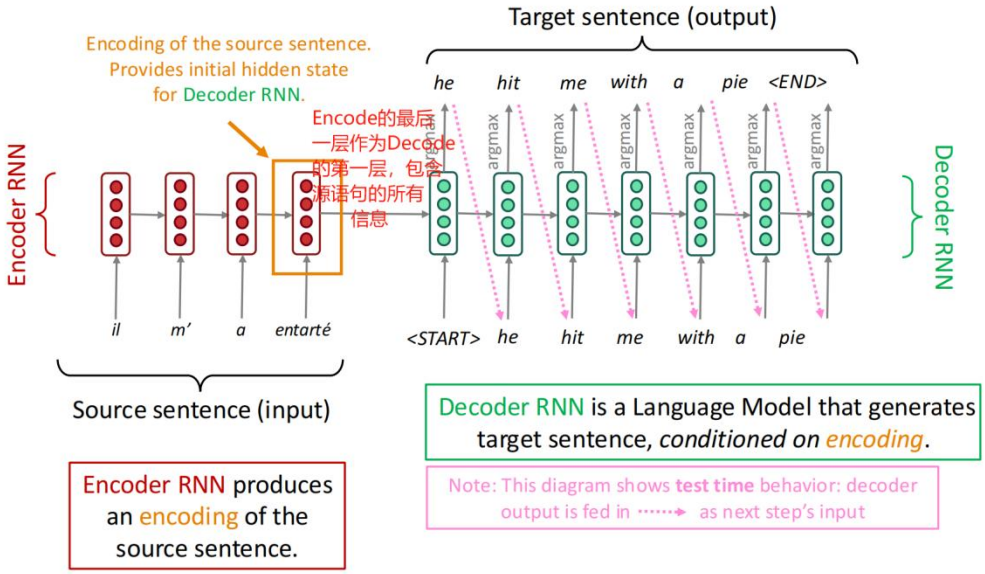
用动态规划的思想

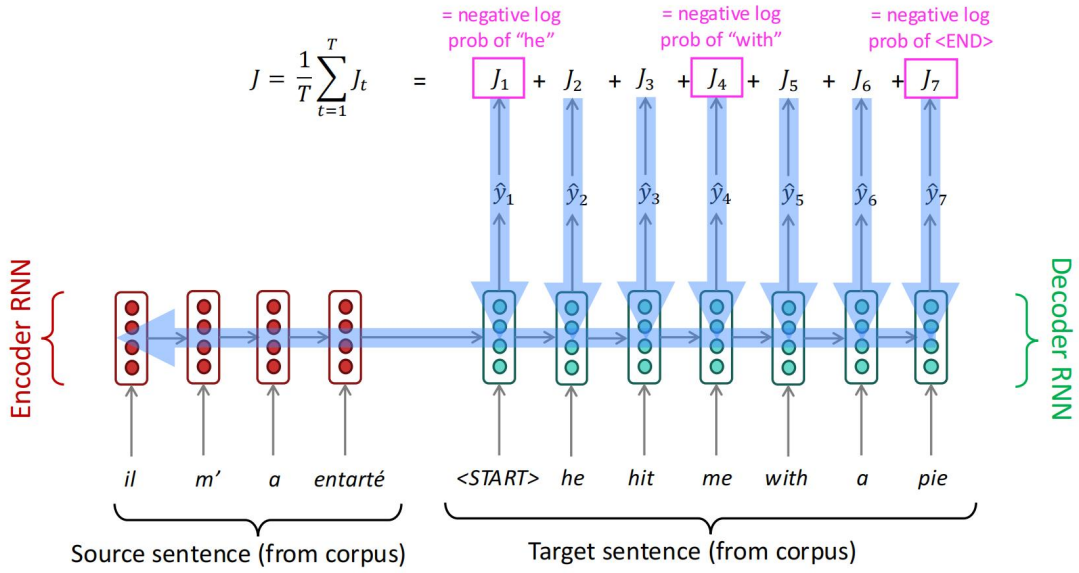


Neural Machine Translation (NMT)

The sequence-to-sequence model

seq2seq：端到端，直接学习源语句到目标语句的映射，无需对齐



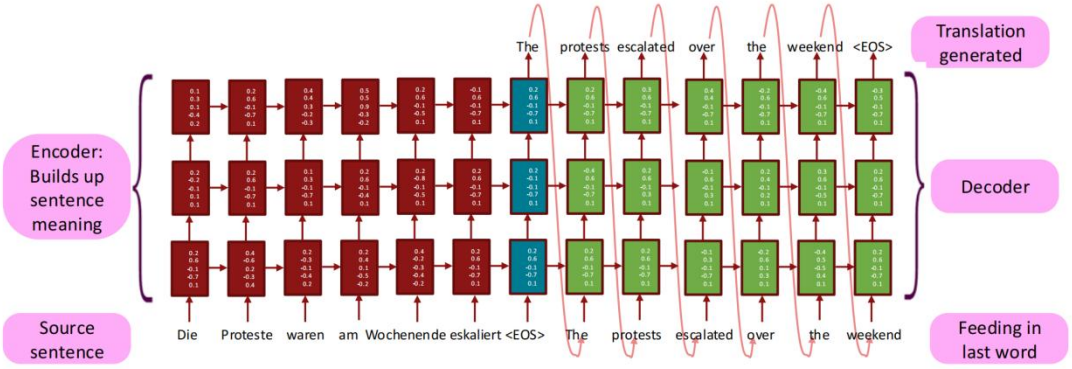


Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

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



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) 对话系统
- Parsing (input text → output parse as sequence) 文本解析（词性标注、依存关系
- Code generation (natural language → Python code)

Greedy decoding 贪心解码:无法回撤

Exhaustive search decoding:计算量指数级

Beam search decoding

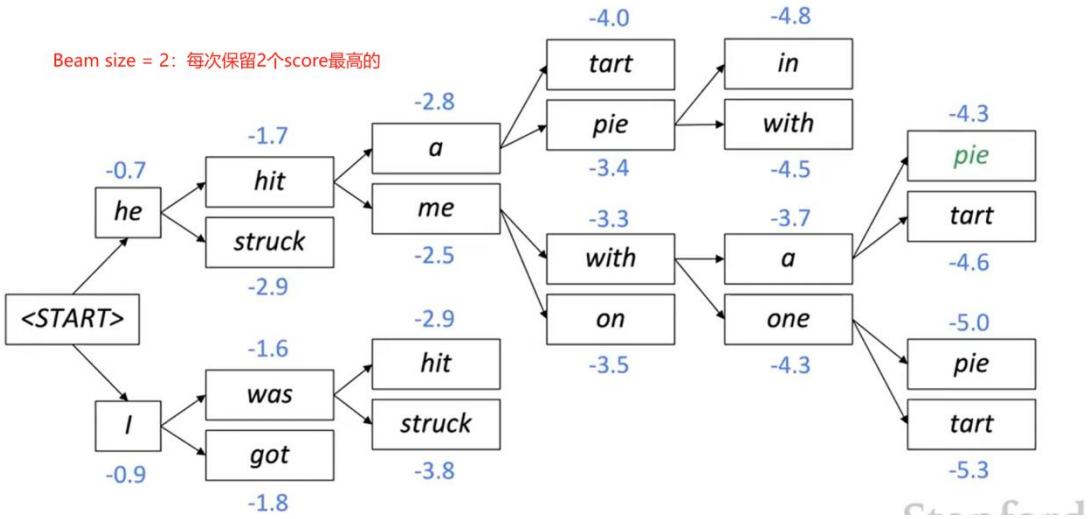
- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10) 找K个最有可能的

- 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 都是负的，越大越好
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is not guaranteed to find optimal solution



In beam search decoding, different hypotheses may produce <END> tokens on different timesteps 一个停止时，其他可能还未停止

- 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)

越长的句子 score 越低：归一化，除以句子的长度

How do we evaluate Machine Translation?

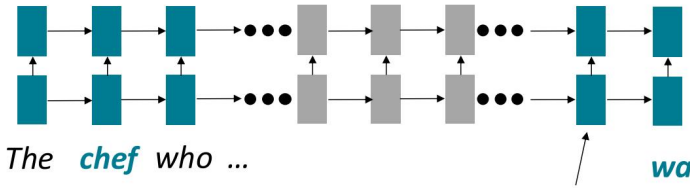
BLEU (**Bil**ingual **E**valuation **U**nderstudy)

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams) 和人工翻译进行对比
  - Plus a penalty for too-short system translations 惩罚过短的翻译
- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹ 一个句子的翻译有很多种

Issues with recurrent models: **Linear interaction distance**

难以学到长距离依赖关系：梯度消失/爆炸

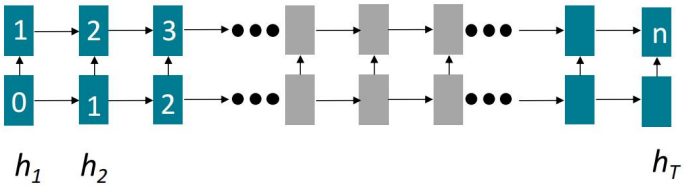
- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences... 句子不一定是线性的



Info of ***chef*** has gone through O(sequence length) many layers!

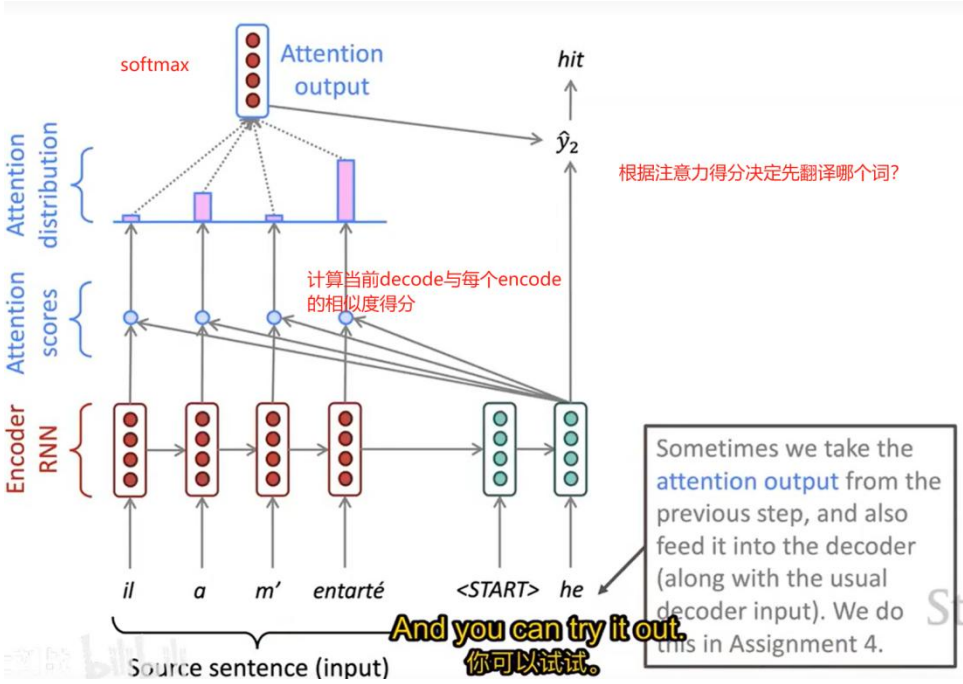
Issues with recurrent models: **Lack of parallelizability**

- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

Lec7



We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$

On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$

We get the attention scores e^t for this step: decoder的隐藏状态和encoder的隐藏状态做点积得到注意力分数

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$ αt实际上是个权重

We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

一起作为输入传给decoder

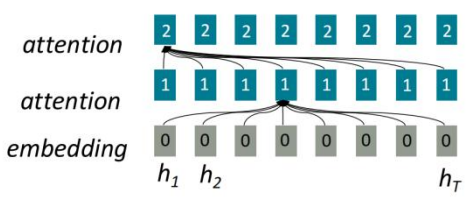
Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Attention is parallelizable, and solves bottleneck issues.

可并行化：GPU

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!



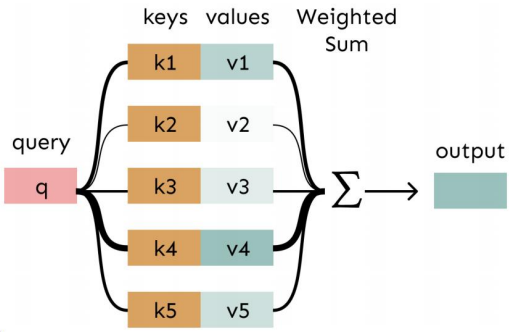
All words attend to all words in previous layer; most arrows here are omitted

Attention is *weighted* averaging, which lets you do lookups!

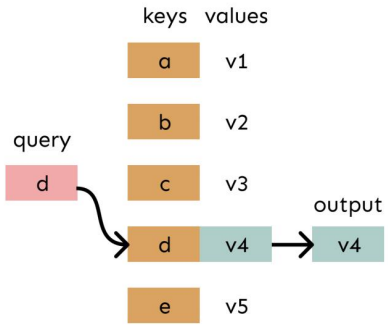
Attention is just a **weighted** average – this is very powerful if the weights are learned!

类似于查找表，q和k计算出一个权重，值加权求和

In **attention**, the query matches all keys *softly*, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



In a **lookup table**, we have a table of keys that map to values. The query matches one of the keys, returning its value.



39

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices $\mathrm{Q}, \mathrm{K}, \mathrm{V}$, each in $\mathbb{R}^{d \times d}$

$$\boldsymbol{q}_i = Q\boldsymbol{x}_i \text{ (queries)} \qquad \boldsymbol{k}_i = K\boldsymbol{x}_i \text{ (keys)} \qquad \boldsymbol{v}_i = V\boldsymbol{x}_i \text{ (values)}$$

2. Compute pairwise similarities between keys and queries; normalize with softmax

点积：相似度

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

加权求和

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**

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1, 2, \ldots, n\} \text{ are position vectors}$$

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 $\boldsymbol{p}_i$ to our inputs!

Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i. The positioned embedding is:

把位置编码和单词的嵌入连接在一起作为输入

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

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

Learned absolute position representations: Let all p_i be learnable parameters!

Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{d \times n}$, *and let each* $\boldsymbol{p}_i$ *be a column of that matrix!*

让位置信息成为一种可学习的参数

Pros:

- Flexibility: each position gets to be learned to fit the data

Cons:

- Definitely can't extrapolate to indices outside $1, \ldots, n$.

Most systems use this! 句子序列可能超过n

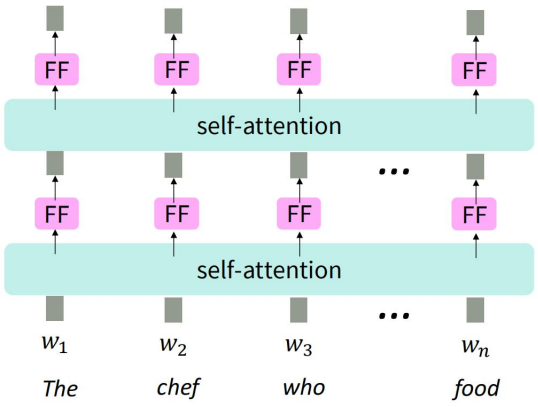
Sometimes people try more flexible representations of position:

- Relative linear position attention [Shaw et al., 2018] 相对位置
- Dependency syntax-based position [Wang et al., 2019] 句子结构

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 (Why? Look at the notes!)
- Easy fix: add a **feed-forward network** to post-process each output vector.

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



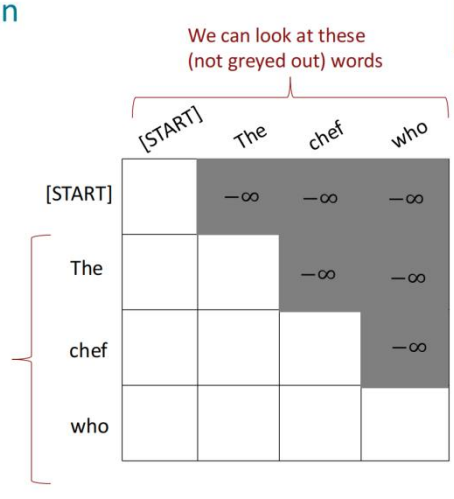
Intuition: the FF network processes the result of attention

15

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 $-\infty$. 未来不可见

$$e_{ij} = \begin{cases} q_i^T k_j, j \leq i \\ -\infty, j > i \end{cases}$$



17

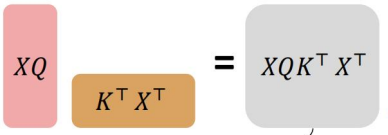
这样在 softmax 中，未来的权重会被置为 0

Sequence-Stacked form of Attention

适合GPU加速

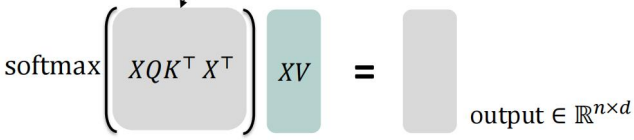
- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{n \times d}$, $XQ \in \mathbb{R}^{n \times d}$, $XV \in \mathbb{R}^{n \times d}$.
  - The output is defined as $\text{output} = \text{softmax}(XQ(XK)^\top)XV \in\in \mathbb{R}^{n \times d}$.

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$



All pairs of attention scores!

Next, softmax, and compute the weighted average with another matrix multiplication.



24

Multi-headed attention

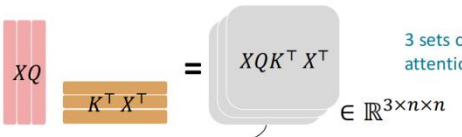
多头注意力机制：关注句子中的不同部分

- 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:
  - $\text{output}_\ell = \text{softmax}(XQ_\ell K_\ell^\top X^\top) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \ldots; \text{output}_h]Y$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differentlv.

Multi-head self-attention is computationally efficient

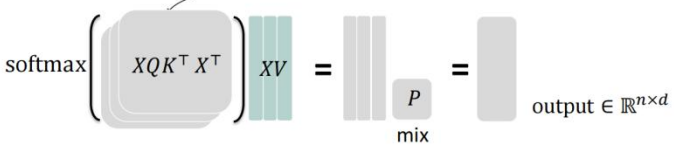
- Even though we compute h many attention heads, it's not really more costly.
  - We compute $XQ \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times h \times d/h}$. (Likewise for XK, XV.)
  - Then we transpose to $\mathbb{R}^{h \times n \times d/h}$; now the head axis is like a batch axis.
  - Almost everything else is identical, and the **matrices are the same sizes.**

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$



3 sets of all pairs of attention scores!

Next, softmax, and compute the weighted average with another matrix multiplication.



26

Scaled Dot Product [Vaswani et al., 2017]

缩放点积

- **"Scaled Dot Product"** attention aids in 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:

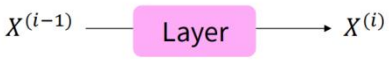
$$\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$$

- 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.)

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

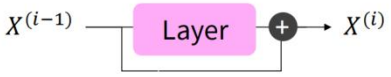
The Transformer Encoder: **Residual connections** [He et al., 2016]

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

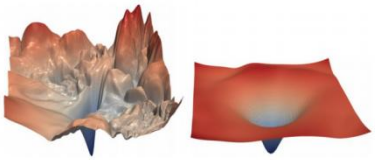


避免梯度消失/爆炸

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

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

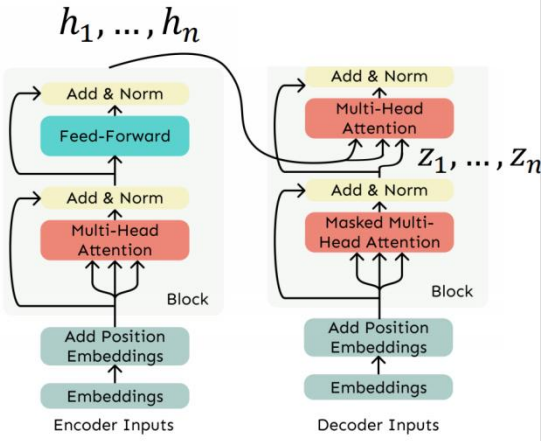
Mask 是一种掩码矩阵，用于在注意力计算过程中屏蔽某些位置的注意力权重。它的作用是限制模型对输入序列中某些部分的关注，从而实现特定的行为模式（如单向或双向注意力）。

- Padding Mask： 用于忽略输入序列中的填充（padding）标记，避免模型浪费计算资源。 encoder中
- Sequence Mask （也称为因果掩码，Causal Mask）：用于确保解码器在生成输出时只能看到当前及之前的 token，而不能看到未来的 token，以模拟自回归生成过程。 decoder中

双向（类似双向 RNN）：decoder 中取消 mask

Cross-attention (details)

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_n$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i.$
- And the queries are drawn from the **decoder**, $q_i = Qz_i.$



34

However, its total number of operations grows as $O(n^2 d)$, where n is the sequence length, and d is the dimensionality.

Lec 8

The byte-pair encoding algorithm

可能会出现训练集中未出现的单词（比如拼写错误），全部置为<unk>损失了太多信息

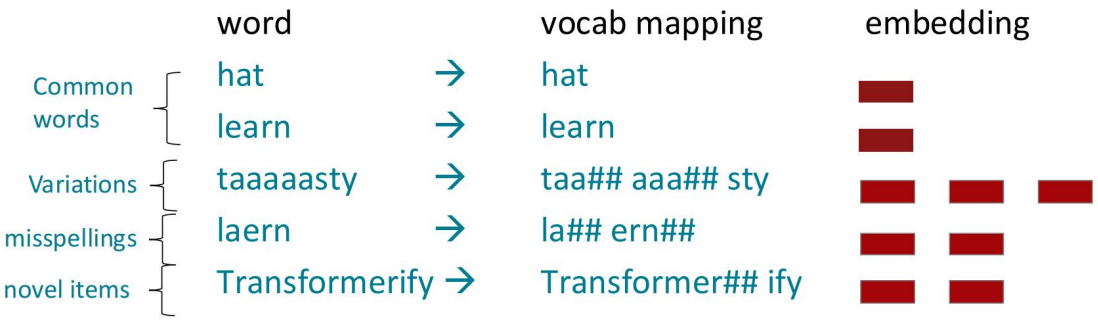
Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- 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.

从单个字符开始合并
每时出现频率最高的合并成子串

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.



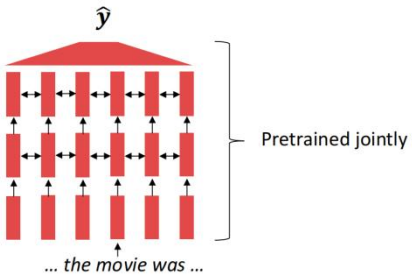
Where we're going: **pretraining whole models**

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.

预训练：学会如何表达整个句子，更具一般性
训练方法：将句子中的某几个词遮蔽，让模型进行学习

- 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]

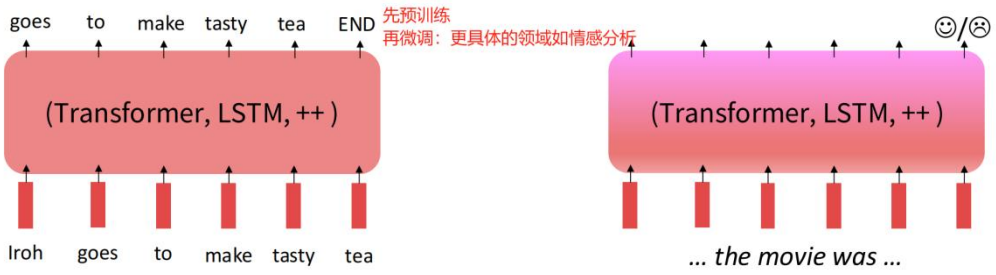
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!

Step 2: Finetune (on your task)

Not many labels; adapt to the task!

先预训练
再微调：更具体的领域如情感分析



Stochastic gradient descent and pretrain/finetune

Why should pretraining and finetuning help, from a "training neural nets" perspective?

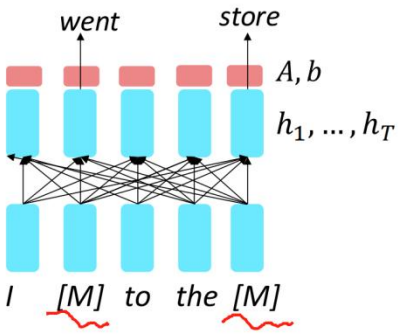
- Consider, provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{\text{pretrain}}(\theta)$.
  - (The pretraining loss.)
- Then, finetuning approximates $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$. 微调在预训练得到的θ基础上
  - (The finetuning loss)
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to $\hat{\theta}$ during finetuning.
  - So, maybe the finetuning local minima near $\hat{\theta}$ tend to generalize well! 这样微调得到的局部最优解可能具备更好的泛化能力，梯度传播也会更好
  - And/or, maybe the gradients of finetuning loss near $\hat{\theta}$ propagate nicely!

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.

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



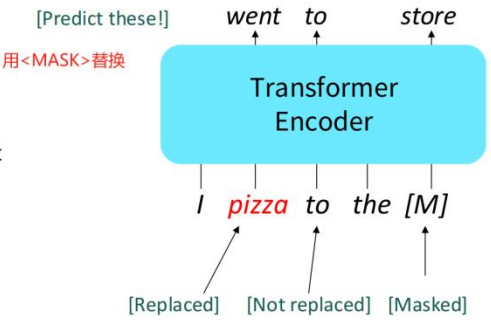
Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.

BERT: Bidirectional Encoder Representations from Transformers

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

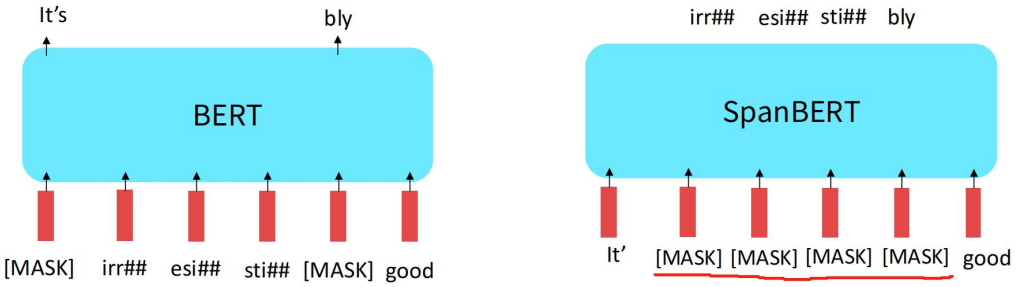
Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time 用<MASK>替换
  - Replace input word with a random token 10% of the time 随机选另一个词进行替换
  - Leave input word unchanged 10% of the time (but still predict it!) 保持词语不变，但仍要求预测
- 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!)



Some generally accepted improvements to the BERT pretraining formula:

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task

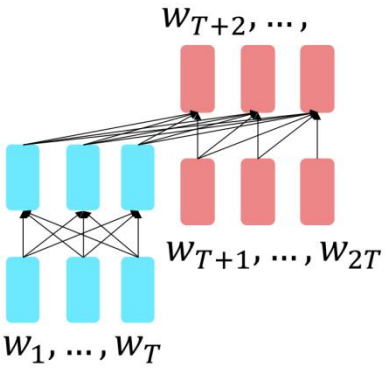


Pretraining encoder-decoders: what pretraining objective to use?

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

前缀输入编码器，解码器预测后缀

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$

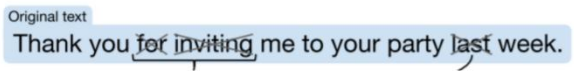


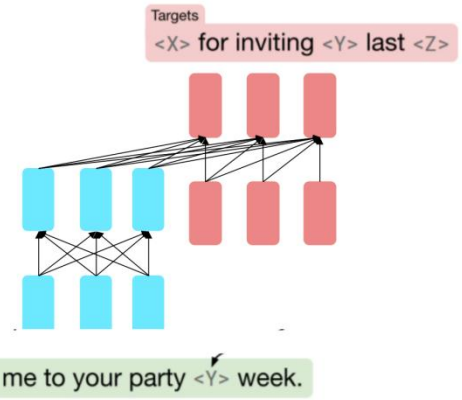
The **encoder** portion benefits from bidirectional context; the **decoder** portion is used to train the whole model through language modeling.

What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.

随机选取不同长度用唯一占位符替代，解码器要预测这些被替代的词

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!





This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.

Pretraining decoders

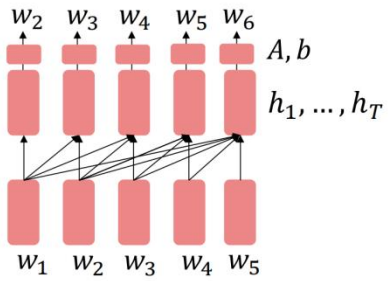
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

Lec 9

Zero-Shot Learning（ZSL）：

你只学过“狗”和“猫”，但没学过“狐狸”。
你被告知：“狐狸是毛茸茸的、尖耳朵、长尾巴的动物，和狗很像。”
你看到一张图片后，说：“这可能是狐狸。”

> “✅ 没见过“狐狸”的图片，但通过语义描述识别。 ”

Few-Shot Learning（FSL）：

你只学过“狗”和“猫”，但有人给你看了**一张“狐狸”的图片** ，并告诉你：“这是狐狸。”
之后你看到另一张狐狸的图片，也能认出来。

> “✅ 只看一次（One-Shot），就能识别。 ”

Zero-Shot Learning（ZSL）

- **核心思想**： 建立一个“语义空间”，将视觉特征和语义描述对齐。
- **依赖**： 每个类别都有一个语义描述（如属性、词向量）。
- **典型方法**：
  - 属性学习（Attribute-based）
  - 词向量嵌入（Word2Vec、BERT）
  - 生成模型（GAN、VAE）

```python
image_features = CNN(image)
text_features = BERT("a red fox with sharp ears")
similarity = cosine_similarity(image_features, text_features)
```

Few-Shot Learning（FSL）

- **核心思想**： 利用**少量样本** 快速适应新类别。
- **常用方法**：
  - 元学习（Meta-Learning）：如 MAML、Reptile
  - 基于度量的方法（Metric-based）：如 Siamese 网络、Prototypical Networks
  - 基于优化的方法（Optimization-based）

```python
# 示例：Prototypical Network
support_set = [few_samples_of_new_class]
query_image = new_image
prototype = mean(support_set_features)
distance = distance(query_features, prototype)
```

不进行梯度更新？

✅ 2. 上下文学习（In-context Learning）

现代大模型具有强大的**上下文学习能力**，即：

> “给模型几个带思维链的示例（few-shot prompting），它就能模仿并生成自己的思维链。”

示例：

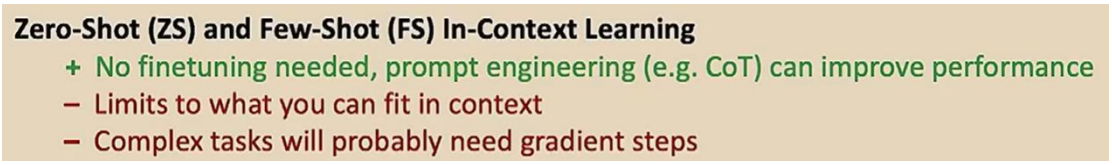
```text
Q: 小明每天跑步3公里，一周跑几天？
A: 小明每天跑3公里，一周有7天，所以总共跑了 3 × 7 = 21 公里。

Q: 一辆车每小时行驶60公里，3小时能行驶多远？
A: 车速是每小时60公里，行驶3小时，总路程是 60 × 3 = 180 公里。

Q: 一本书有200页，小红每天看20页，几天能看完？
A:
```

模型看到前两个例子后，会模仿生成：

> “小红每天看20页，200 ÷ 20 = 10 天能看完。”



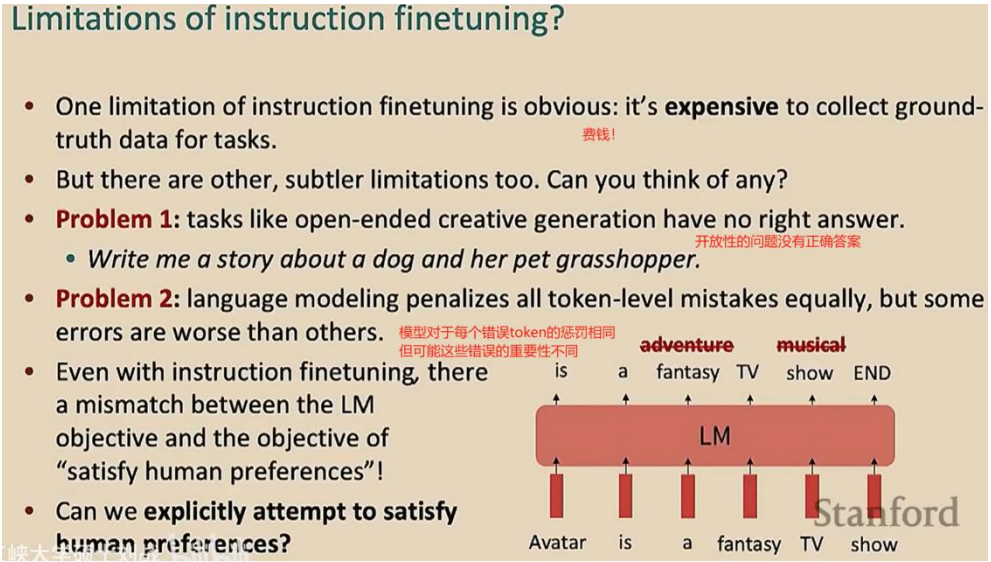
指令微调（Instruction Tuning）是一种**通过有监督学习方式，让大语言模型（LLM）更好地理解和遵循人类指令** 的技术。它通过在包含“指令-输入-输出”三元组的数据集上微调预训练模型，使其能够更准确地响应各种自然语言指令。

---

1. 什么是指令微调？

指令微调的核心思想是：**让模型学会“听懂人话”**。
在预训练阶段，大语言模型（如GPT、LLaMA）通过海量文本学习语言模式，但并未专门训练如何遵循指令。指令微调则通过提供结构化数据（指令+输入+期望输出），让模型学会：

- 理解任务要求（如“翻译”“总结”“生成代码”）
- 根据上下文生成符合指令的响应
- 处理多轮对话和复杂任务



How do we actually change our LM parameters θ to maximize this?

强化学习：使期望奖励最大化 $\mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})]$

Let's try doing gradient ascent!

梯度上升 $\theta_{t+1} \coloneqq \theta_t + \alpha \nabla_{\theta_t} \mathbb{E}_{\hat{s}\sim p_{\theta_t}(s)}[R(\hat{s})]$

How do we estimate this expectation??

What if our reward function is non-differentiable??

策略梯度方法

Policy gradient methods in RL (e.g., REINFORCE; [Williams, 1992]) give us tools for estimating and optimizing this objective.

REINFORCE 的核心公式可以写为：

$$\nabla_{\theta_t} \mathbb{E}_{\hat{s}\sim p_{\theta_t}(s)}[R(\hat{s})] \approx \frac{1}{N}\sum_{i=1}^{N} R(\hat{s}_i)\nabla_{\theta_t} \log p_{\theta_t}(\hat{s}_i)$$

其中：

- N 是采样的样本数量。
- $\hat{s}_i$ 是第 i 个采样的状态。
- $R(\hat{s}_i)$ 是对应状态的奖励。
- $\nabla_{\theta_t} \log p_{\theta_t}(\hat{s}_i)$ 是对数概率的梯度，表示状态 $\hat{s}_i$ 的生成概率对参数 θ_t 的敏感度。

We want to obtain

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \nabla_\theta \sum_s R(s)p_\theta(s) = \sum_s R(s)\nabla_\theta p_\theta(s)$$

(defn. of expectation) (linearity of gradient)

Here we'll use a very handy trick known as the **log-derivative trick**. Let's try taking the gradient of $\log p_\theta(s)$

$$\nabla_\theta \log p_\theta(s) = \frac{1}{p_\theta(s)}\nabla_\theta p_\theta(s) \quad \Rightarrow \quad \nabla_\theta p_\theta(s) = p_\theta(s)\ \nabla_\theta \log p_\theta(s)$$

(chain rule)

对p(s)求梯度 = p(s) * logp(s) 的梯度

Plug back in:

$$\sum_s R(s)\nabla_\theta p_\theta(s) = \sum_s p_\theta(s) R(s)\nabla_\theta \log p_\theta(s)$$

This is an expectation of this

Now we have put the gradient "inside" the expectation, we can approximate this objective with Monte Carlo samples:

然后用蒙特卡洛法采样

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})\ \nabla_\theta \log p_\theta(\hat{s})] \approx \frac{1}{m}\sum_{i=1}^{m} R(s_i)\ \nabla_\theta \log p_\theta(s_i)$$

This is why it's called **"reinforcement learning"**: we **reinforce** good actions, increasing the chance they happen again.

- Giving us the update rule: $\theta_{t+1} \coloneqq \theta_t + \alpha \frac{1}{m}\sum_{i=1}^{m} R(s_i)\ \nabla_{\theta_t} \log p_{\theta_t}(s_i)$

If R is +++ Take gradient steps to maximize $p_\theta(s_i)$

If R is --- Take steps to minimize $p_\theta(s_i)$

This is **heavily simplified**! There is a *lot* more needed to do RL w/ LMs. **Can you see any problems with this objective?**

- **Problem 1:** human-in-the-loop is expensive!
  - **Solution:** instead of directly asking humans for preferences, **model their preferences** as a separate (NLP) problem! [Knox and Stone, 2009] 训练一个模型预测人类的偏好 用这个模型的输出作为奖励函数

An earthquake hit San Francisco. There was minor property damage, but no injuries.

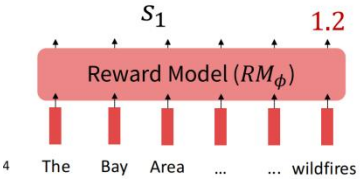
The Bay Area has good weather but is prone to earthquakes and wildfires.

s_1 $R(s_1) = 8.0$ s_2 $R(s_2) = 1.2$

Train an LM $RM_\phi(s)$ to predict human preferences from an annotated dataset, then optimize for RM_ϕ instead.

- **Problem 2:** human judgments are noisy and miscalibrated! 人类打分存在噪声，因此只做比较
- **Solution:** instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

An earthquake hit San Francisco. There was minor property damage, but no injuries. > A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage. > The Bay Area has good weather but is prone to earthquakes and wildfires.



s_3 训练一个比较模型，根据人类比较的s_2结果打分

Bradley-Terry [1952] paired comparison model

$$J_{RM}(\phi) = -\mathbb{E}_{(s^w, s^l)\sim D}\left[\log \sigma(RM_\phi(s^w) - RM_\phi(s^l))\right]$$

"winning" sample "losing" sample s^w should score higher than s^l

RLHF: Putting it all together [Christiano et al., 2017; Stiennon et al., 2020]

Reinforcement Learning from Human Feedback

- Finally, we have everything we need:
  - A pretrained (possibly instruction-finetuned) LM $p^{PT}(s)$ 预训练+微调的模型
  - A reward model $RM_\phi(s)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons 奖励模型
  - A method for optimizing LM parameters towards an arbitrary reward function.
- Now to do RLHF:
  - Initialize a copy of the model $p_\theta^{RL}(s)$, with parameters θ we would like to optimize
  - Optimize the following reward with RL:

有个惩罚项，防止偏离原模型太远

$$R(s) = RM_\phi(s) - \beta \log \left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)}\right)$$

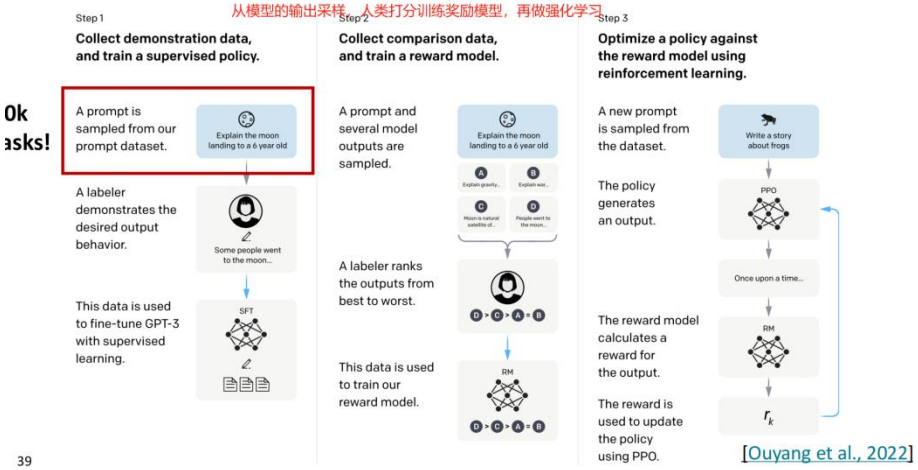
Pay a price when $p_\theta^{RL}(s) > p^{PT}(s)$

实际上是KL散度

This is a penalty which prevents us from diverging too far from the pretrained model. In expectation, it is known as the **Kullback-Leibler** (**KL**) divergence between $p_\theta^{RL}(s)$ and $p^{PT}(s)$.

InstructGPT: scaling up RLHF to tens of thousands of tasks

从模型的输出采样，人类打分训练奖励模型，再做强化学习



[Ouyang et al., 2022]

Limitations of RL + Reward Modeling

- Human preferences are unreliable!
  - ”Reward hacking” is a common problem in RL 奖励欺骗：用不符合预期的方式获得高分
    奖励函数设计的缺陷
  - Chatbots are rewarded to produce responses that *seem* authoritative and helpful, *regardless of truth*
  - This can result in making up facts + hallucinations 编造事实 幻觉
- **Models** of human preferences are *even more* unreliable!

1. **Zero-Shot (ZS) and Few-Shot (FS) In-Context Learning**
   - \+ No finetuning needed, prompt engineering (e.g. CoT) can improve performance
   - – Limits to what you can fit in context
   - – Complex tasks will probably need gradient steps
2. **Instruction finetuning**
   - \+ Simple and straightforward, generalize to unseen tasks
   - – Collecting demonstrations for so many tasks is expensive
   - – Mismatch between LM objective and human preferences
3. **Reinforcement Learning from Human Feedback (RLHF)**
   - \+ Directly model preferences (cf. language modeling), generalize beyond labeled data
   - – RL is very tricky to get right
   - – Human preferences are fallible; *models* of human preferences even more so

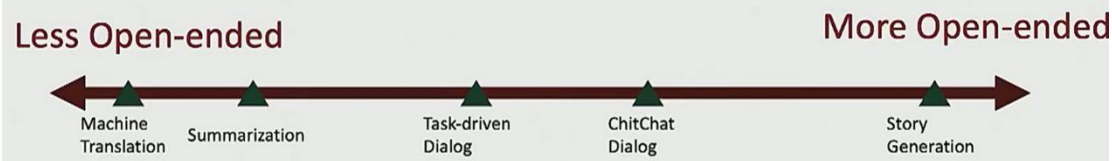
RLHF is still a very underexplored and fast-moving area: by the next lecture these slides may look completely different!

RLHF gets you further than instruction finetuning, but is (still!) data expensive.

Recent work aims to alleviate such data requirements:

- RL from **AI feedback** [Bai et al., 2022] 用AI生成反馈
- Finetuning LMs on their own outputs [Huang et al., 2022; Zelikman et al., 2022] LLM可以自我改进

Lec NLG



Open-ended generation: the output distribution still has high freedom

Non-open-ended generation: the input mostly determines the output generation.

NLG的开放程度可以用熵来衡量：熵大意味着随机性大，比较开放

Remark: One way of formalizing categorization this is by **entropy**.

- For non-open-ended tasks (e.g., MT), we typically use a encoder-decoder system, where this autoregressive model serves as the decoder, and we'd have another bidirectional encoder for encoding the inputs.
- For open-ended tasks (e.g., story generation), this autoregressive generation model is often the only component.

机器翻译等不那么开放的一般用encoder-decoder
开放问题一般只用decoder，因为encoder-decoder性能和decoder差不多

- **Greedy Decoding**
  - Selects the highest probability token in $P(y_t|y_{<t})$

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

softmax

- **Beam Search**
  - Discussed in Lecture 7 on Machine Translation
  - Also aims to find strings that maximize the log-prob, but with wider exploration of candidates

存在自放大效应：重复次数越多，模型对重复越有信心

How can we reduce repetition?

Simple option:

- Heuristic: Don't repeat *n*-grams 禁止出现相同的短语（或降低概率）

More complex:

- Use a different training objective: 在最大化目标token时增加损失项，惩罚生成重复的token
  - Unlikelihood objective (Welleck et al., 2020) penalize generation of already-seen tokens
  - Coverage loss (See et al., 2017) Prevents attention mechanism from attending to the same words 防止注意力机制关注相同的单词，同样增加损失项（覆盖向量记录出现次数？）
- Use a different decoding objective:
  - Contrastive decoding (Li et al, 2022) searches for strings x that maximize logprob_largeLM (x) – logprob_smallLM (x). 最大化大模型和小模型的差异

Solution: Top-*k* sampling

- Only sample from the top *k* tokens in the probability distribution

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

可以用整个分布的熵重新分配单词得分的权重

Typical Sampling (Meister et al. 2022)

- Reweights the score based on the entropy of the distribution.

Epsilon Sampling (Hewitt et al. 2022)

- Set a threshold for lower bounding valid probabilities.

- 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)

- 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 (low) perplexity!
    - Careful! Remember that repetitive utterances generally get low perplexity.

非常重复的句子也会有非常低的困惑度

  - 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

Exposure Bias Solutions

decode时模型只能依赖前面自己生成的token导致偏差

- Scheduled sampling (Bengio et al., 2015)
  - With some probability *p*, decode a token and feed that as the next input, rather than the gold token.

训练时用p的概率使用模型生成的token，1-p的概率使用真实的token

  - Increase *p* over the course of training

不断增加p的概率，一开始真实的概率大

  - Leads to improvements in practice, but can lead to strange training objectives

- Dataset Aggregation (DAgger; Ross et al., 2011)

把训练生成的序列加入训练集

  - At various intervals during training, generate sequences from your current model
  - Add these sequences to your training set as additional examples

Retrieval Augmentation (Guu*, Hashimoto*, et al., 2018)

- Learn to retrieve a sequence from an existing corpus of human-written prototypes (e.g., dialogue responses) 从已有的句子中替换/增加/删除等 这时不是从左到右的了
- Learn to edit the retrieved sequence by adding, removing, and modifying tokens in the prototype – this will still result in a more "human-like" generation

Reinforcement Learning: cast your text generation model as a Markov decision process

- **State** s is the model's representation of the preceding context
- **Actions** a are the words that can be generated
- **Policy** π is the decoder
- **Rewards** r are provided by an external score

不一定奖励的分数越高模型越好！

Content overlap metrics

评估NLG模型：计算相似度

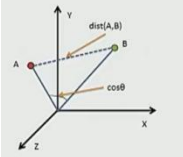
Ref: They walked to the grocery store .

Gen: The woman went to the hardware store .

- Compute a score that indicates the lexical similarity between *generated* and *gold-standard (human-written) text*
- Fast and efficient and widely used
- *N*-gram overlap metrics (e.g., **BLEU**, ROUGE, METEOR, CIDEr, etc.)

缺点：短语之间可能语义相似但没有重叠，导致分数较低；或语义相反，但分数较高

Model-based metrics: Word distance functions

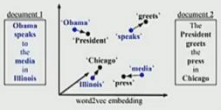


词相似度

Vector Similarity

Embedding based similarity for semantic distance between text.

- **Embedding Average** (Liu et al., 2016)
- **Vector Extrema** (Liu et al., 2016)
- **MEANT** (Lo, 2017)
- **YISI** (Lo, 2019)



匹配

Word Mover's Distance

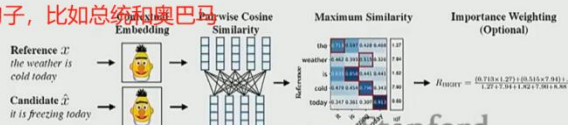
Measures the distance between two sequences (e.g., sentences, paragraphs, etc.), using word embedding similarity matching.

(Kusner et.al., 2015; Zhao et al., 2019)

用词相似度匹配生成的句子和原句子，比如总统和奥巴马

BERTSCORE

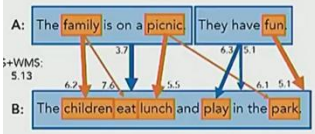
Uses pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity.



Sentence Movers Similarity :

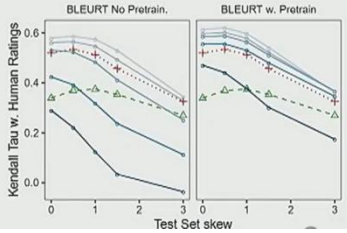
Based on Word Movers Distance to evaluate text in a continuous space using sentence embeddings from recurrent neural network representations.

(Clark et.al., 2019)



BLEURT:

A regression model based on BERT returns a score that indicates to what extent the candidate text is grammatical and conveys the meaning of the reference text.



适用于开放问题：

MAUVE (details)

把生成的句子嵌入到一个高维空间，然后用k-means降维，分成离散的空间，方便计算距离

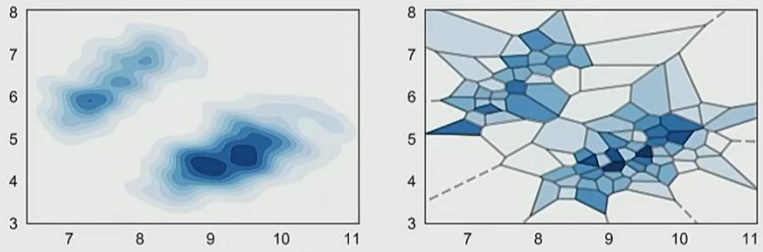


Figure 3: Illustration of the quantization. **Left**: A continuous two-dimensional distribution P. **Right**: A partitioning of the Euclidean plane $\mathbb{R}^2$ and the corresponding quantized distribution $\tilde{P}$.

人类评估是最好的标准，但仍有缺点：慢，贵，不一致...

Lec Question-Answering(问答系统)

许多 NLP 问题可以看作是阅读理解问题

- Problem formulation
  - Input: $C = (c_1, c_2, \ldots, c_N)$ (Context), $Q = (q_1, q_2, \ldots, q_M)$ (Questions), $c_i, q_i \in V$ — N~100, M ~15
  - Output: $1 \leq$ start $\leq$ end $\leq N$ — 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)..

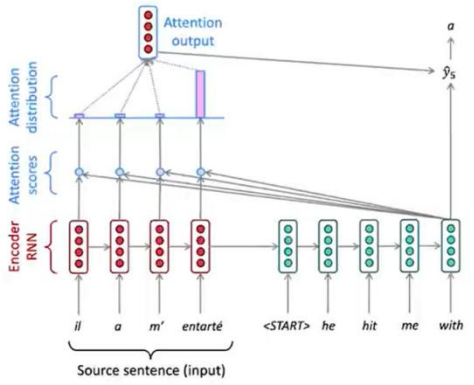
- Fine-tuning BERT-like models for reading comprehension (2019+)

Recap: seq2seq model with attention

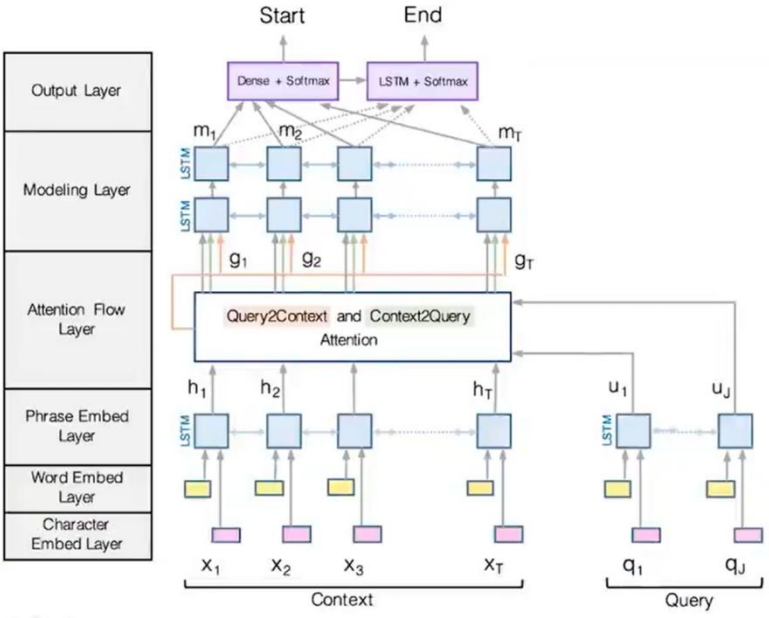
阅读理解和机器翻译类似，都有两个序列

- Instead of source and target sentences, we also have two sequences: passage and question (lengths are imbalanced)
- We need to model which words in the passage are most relevant to the question (and which question words)

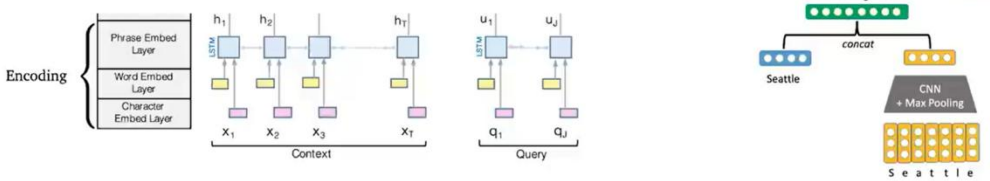
  Attention is the key ingredient here, similar to which words in the source sentence are most relevant to the current target word…
- We don't need an autoregressive decoder to generate the target sentence word-by-word. Instead, we just need to train two classifiers to predict the start and end positions of the answer



BiDAF: the Bidirectional Attention Flow model



BiDAF: Encoding



- Use a concatenation of word embedding (GloVe) and character embedding (CNNs over character embeddings) for each word in context and query.

$$e(c_i) = f([\text{GloVe}(c_i); \text{charEmb}(c_i)]) \qquad e(q_i) = f([\text{GloVe}(q_i); \text{charEmb}(q_i)])$$

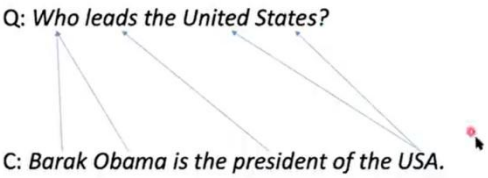
对于每个单词，有一个单词的嵌入和一个每个字符的嵌入，连接在一起　　f: high-way networks omitted here

- Then, use two **bidirectional** LSTMs separately to produce contextual embeddings for both context and query.

$$\overrightarrow{\mathbf{c}}_i = \text{LSTM}(\overrightarrow{\mathbf{c}}_{i-1}, e(c_i)) \in \mathbb{R}^H \qquad \overrightarrow{\mathbf{q}}_i = \text{LSTM}(\overrightarrow{\mathbf{q}}_{i-1}, e(q_i)) \in \mathbb{R}^H$$
$$\overleftarrow{\mathbf{c}}_i = \text{LSTM}(\overleftarrow{\mathbf{c}}_{i+1}, e(c_i)) \in \mathbb{R}^H \qquad \overleftarrow{\mathbf{q}}_i = \text{LSTM}(\overleftarrow{\mathbf{q}}_{i+1}, e(q_i)) \in \mathbb{R}^H$$
$$\mathbf{c}_i = [\overrightarrow{\mathbf{c}}_i; \overleftarrow{\mathbf{c}}_i] \in \mathbb{R}^{2H} \qquad \mathbf{q}_i = [\overrightarrow{\mathbf{q}}_i; \overleftarrow{\mathbf{q}}_i] \in \mathbb{R}^{2H}$$

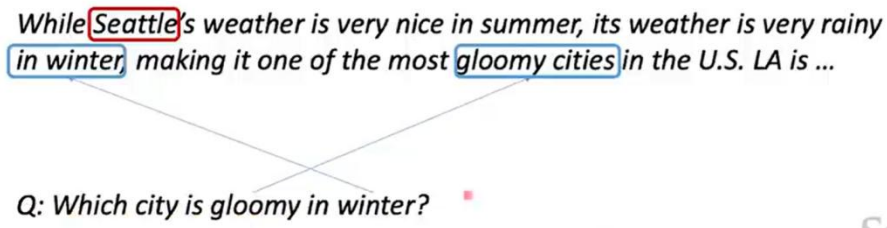
Stanford

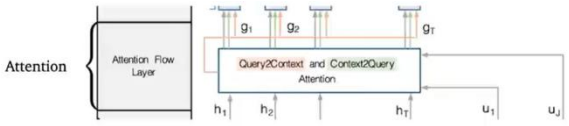
Context-to-query attention: For each context word, choose the most relevant words from the query words.



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

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





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

- 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):

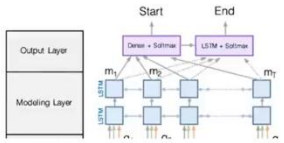
$$\alpha_{i,j} = \text{softmax}_j(S_{i,j}) \in \mathbb{R} \qquad \mathbf{a}_i = \sum_{j=1}^{M} \alpha_{i,j} \mathbf{q}_j \in \mathbb{R}^{2H}$$

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

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

Stanford

第一个注意力是哪些问题词与当前文本有关；第二个注意力是对于某个问题词，哪些文本是有关的，哪些是无关的



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

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

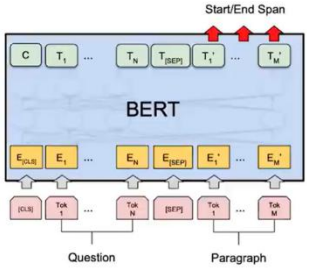
Output layer: two classifiers predicting the start and end positions:

$$p_{\text{start}} = \text{softmax}(\mathbf{w}_{\text{start}}^\intercal [\mathbf{g}_i; \mathbf{m}_i]) \quad p_{\text{end}} = \text{softmax}(\mathbf{w}_{\text{end}}^\intercal [\mathbf{g}_i; \mathbf{m}_i'])$$
$$\mathbf{m}_i' = \text{BiLSTM}(\mathbf{m}_i) \in \mathbb{R}^{2H} \quad \mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}} \in \mathbb{R}^{10H}$$

Stanford

BERT for reading comprehension

BERT是个双向注意力的encoder



Question = Segment A
Passage = Segment B
Answer = predicting two endpoints in segment B

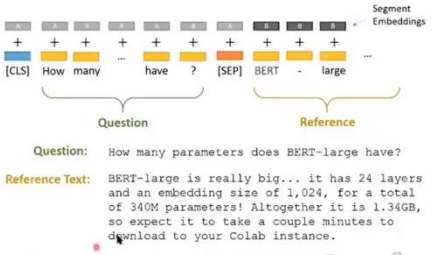


Image credit: https://mccormickml.com/

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$
$$p_{\text{start}}(i) = \text{softmax}_i(\mathbf{w}_{\text{start}}^\intercal \mathbf{h}_i)$$
$$p_{\text{end}}(i) = \text{softmax}_i(\mathbf{w}_{\text{end}}^\intercal \mathbf{h}_i)$$

where $\mathbf{h}_i$ is the hidden vector of c_i, returned by BERT 通过注意力机制计算出hi

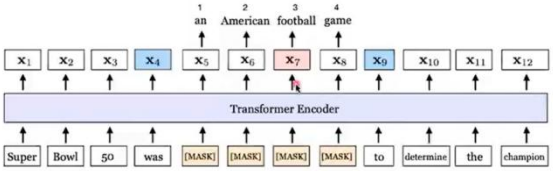
- BERT model has many many more parameters (110M or 330M) and BiDAF has ~2.5M parameters.
- BiDAF is built on top of several bidirectional LSTMs while BERT is built on top of Transformers (no recurrence architecture and easier to parallelize).
- BERT is **pre-trained** while BiDAF is only built on top of GloVe (and all the remaining parameters need to be learned from the supervision datasets).

Pre-training is clearly a game changer but it is expensive..

Can we design better pre-training objectives?

The answer is yes!

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



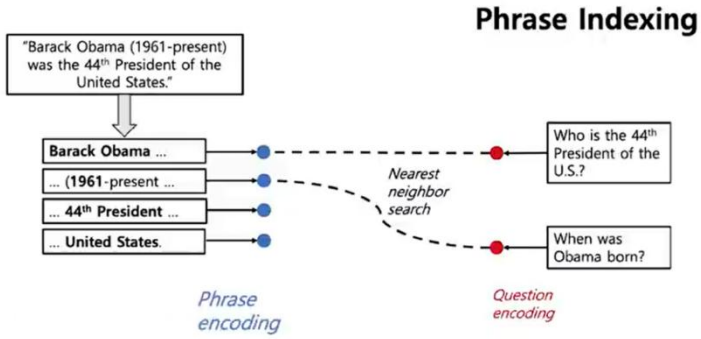
Two ideas: 在预训练时，不是随机遮蔽单词，而是遮蔽一个连续跨度的单词

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 原先的目标是找两个端点，那能否用这两个端点中间的信息预测其余的单词(即用x4和x9之间的单词进行预测)

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

It is possible to encode all the phrases (60 billion phrases in Wikipedia) using **dense** vectors and only do nearest neighbor search without a BERT model at inference time!



Lec11 Adaption

One key emergent ability in GPT-2 [Radford et al., 2019] is **zero-shot learning**: the ability to do many tasks with **no examples,** and **no gradient updates,** by simply:

- Specifying the right sequence prediction problem (e.g. question answering):

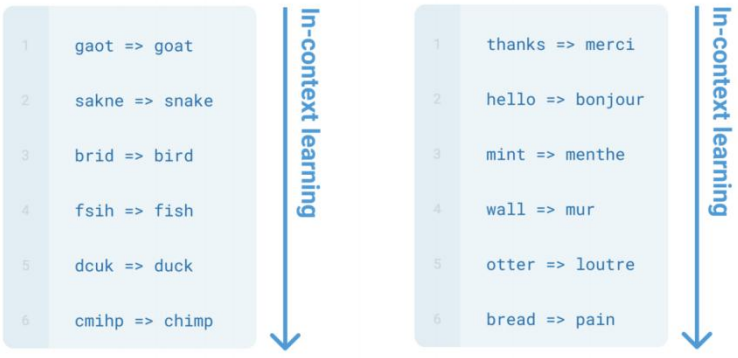
```
Passage: Tom Brady... Q: Where was Tom Brady born? A: ...
```

- Comparing probabilities of sequences (e.g. Winograd Schema Challenge [Levesque, 2011]):

```
The cat couldn't fit into the hat because it was too big.
Does it = the cat or the hat?
≡ Is P(...because the cat was too big) >=
     P(...because the hat was too big)?
```

Emergent few-shot learning [Brown et al., 2020]

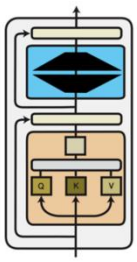
- Specify a task by simply **prepending examples of the task before your example**
- Also called **in-context learning,** to stress that *no gradient updates* are performed when learning a new task (there is a separate literature on few-shot learning with gradient updates)



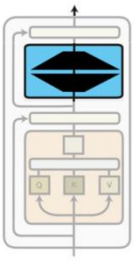
Downside of prompt–based learning

1. **Inefficiency:** The prompt needs to be processed *every time* the model makes a prediction.
2. **Poor performance**: Prompting generally performs worse than fine-tuning [Brown et al., 2020].
3. **Sensitivity** to the wording of the prompt [Webson & Pavlick, 2022], order of examples [Zhao et al., 2021; Lu et al., 2022], etc.
4. **Lack of clarity** regarding what the model learns from the prompt. Even random labels work [Zhang et al., 2022; Min et al., 2022]!

2. From fine-tuning to parameter efficient fine-tuning (PEFT)



Full Fine-tuning
Update **all model parameters**



Parameter-efficient Fine-tuning
Update a **small subset** of model parameters

Why fine-tuning *only some* parameters?

1. Fine-tuning all parameters is impractical with large models

节省计算资源和时间，性能接近

2. State-of-the-art models are massively over-parameterized
   → Parameter-efficient fine-tuning matches performance of full fine-tuning

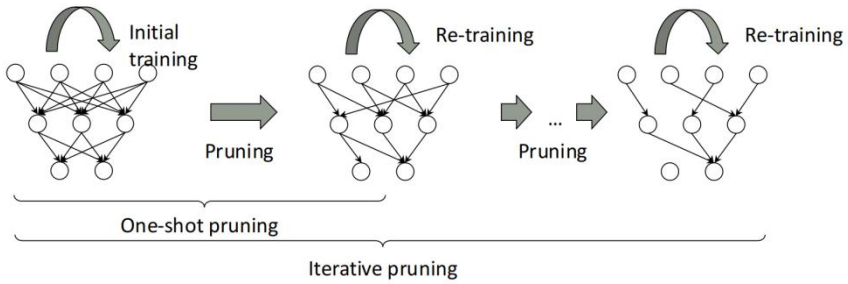
3. Sparse subnetworks

归纳偏置

- A common inductive bias on the module parameters is **sparsity**
- Most common sparsity method: **pruning** 剪枝
- Pruning can be seen as applying a binary mask $\mathbf{b} \in \{0,1\}^{|\theta|}$ that selectively keeps or removes each connection in a model and produces a subnetwork.
- Most common pruning criterion: **weight magnitude** [Han et al., 2017] 权重的绝对值大小：小的权重贡献小

Pruning

- During pruning, a fraction of the lowest-magnitude weights are removed
- The non-pruned weights are re-trained
- Pruning for multiple iterations is more common (Frankle & Carbin, 2019)



The Lottery Ticket Hypothesis

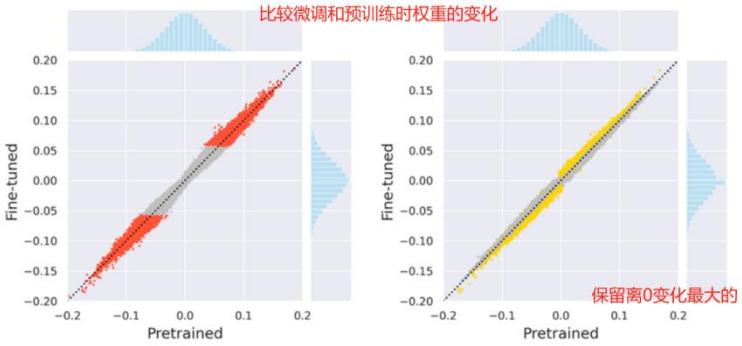
彩票假设：一些子网络在单独训练时能在相似的迭代次数内达到和原神经网络相似的性能

- Dense, randomly-initialized models **contain subnetworks** ("winning tickets") that—when trained in isolation—**reach test accuracy comparable to the original network** in a similar number of iterations [Frankle & Carbin, 2019]
- Has also been verified in RL and NLP [Yu et al., 2020] and for larger models in computer vision [Frankle et al., 2020]
- Prior work [Chen et al., 2020; Prasanna et al., 2020] has found winning tickets in pre-trained models such as BERT
  - Sparsity ratios: from 40% (SQuAD) to 90% (QQP and WNLI)
- Subnetworks trained on a general task like masked language modelling **transfer** best

- Pruning does not consider how weights change during fine-tuning
- **Magnitude pruning**: keep weights farthest from 0
- **Movement pruning** [Sanh et al., 2020]: keep weights that *move the most away* from 0

Fine-tuned weights stay close to their their pre-trained values. Magnitude pruning (left) selects weights that are far from 0

保留离0最远的



Movement pruning (right) selects weights that move away from 0

LoRA: low rank adaptation (Hu et al., 2021)

- For each downstream task, we learn a different set of parameters $\Delta\phi$
  - $|\Delta\phi| = |\phi_o|$
  - GPT-3 has a $|\phi_o|$ of 175 billion
  - Expensive and challenging for storing and deploying many independent instances

- **Key idea**: encode the task-specific parameter increment $\Delta\phi = \Delta\phi(\Theta)$ by **a smaller-sized set of parameters Θ**, $|\Theta| \ll |\phi_o|$

- The task of finding $\Delta\phi$ becomes optimizing over Θ

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\phi_o+\Delta\phi(\Theta)}(y_t|x, y_{<t}))$$

Low-rank-parameterized update matrices

- Updates to the weights have a low "intrinsic rank" during adaptation (Aghajanyan et al. 2020)

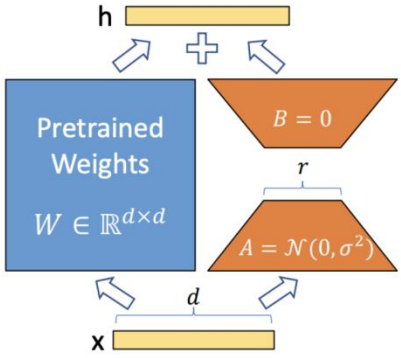
- $W_0 \in \mathbb{R}^{d\times k}$: a pretrained weight matrix
- Constrain its update with a low-rank decomposition:

$$W_0 + \Delta W = W_0 + \alpha BA$$

where $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}, r \ll \min(d, k)$

- α is the tradeoff between pre-trained "knowledge" and task-specific "knowledge"

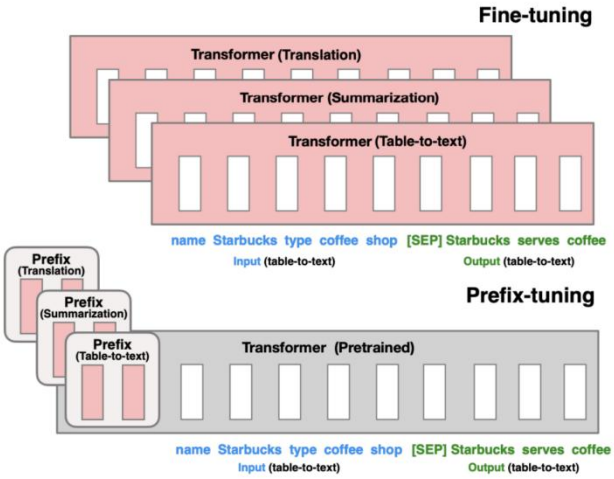
- Only A and B contain **trainable** parameters



Prefix-Tuning (Li and Liang, 2021)

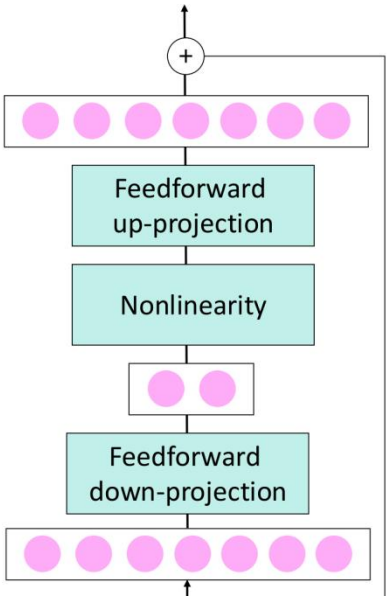
微调时不更新预训练的参数

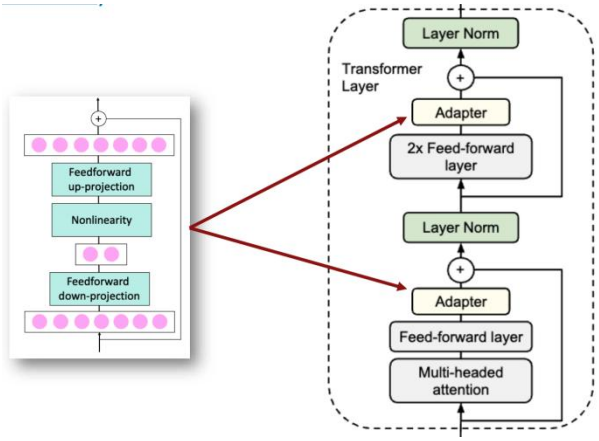
- Prefix-Tuning adds a **prefix** of parameters and **freezes all pretrained parameters.**
- The prefix is a sequence of continuous task-specific vector and is processed by the model just like real words would be, i.e., **"virtual tokens".**
- **Advantage:** each element of a batch at inference could run a different tuned model.



Adapter (Houlsby et al. 2019)

- Insert a new function f_ϕ between layers of a pre-trained model to adapt to a downstream task --- known as "adapters"
- An **adapter** in a Transformer layer consists of:
  - A feed-forward down-projection $W^D \in R^{k \times d}$ 映射到低维空间
  - A feed-forward up-projection $W^U \in R^{d \times k}$
- $f_\phi(\boldsymbol{x}) = W^U(\sigma(W^D\boldsymbol{x}))$





Lec

Integrating knowledge in LM

T5模型（Text-to-Text Transfer Transformer）是由Google在2019年提出的一种基于Transformer架构的预训练语言模型。它的核心思想是将所有自然语言处理（NLP）任务统一为“文本到文本”（text-to-text）的形式，也就是说，无论任务是翻译、摘要、问答、分类还是其他任务，模型的输入和输出都被视为文本序列。

核心特点：

1. **统一的文本到文本框架：**
   - T5将所有任务都视为“给定输入文本，生成输出文本”的问题。
   - 例如：
     - 翻译：输入 `"translate English to German: Hello!"`，输出 `"Hallo!"`
     - 情感分类：输入 `"sentiment: I love this movie"`，输出 `"positive"`
     - 问答：输入 `"question: What is the capital of France?"`，输出 `"Paris"`
2. **基于Transformer的编码器-解码器结构：**
   - 使用完整的Transformer架构，包含编码器（Encoder）和解码器（Decoder），类似于原始的Transformer模型（用于机器翻译）。
   - 这与BERT（仅编码器）或GPT（仅解码器）不同。
3. **大规模预训练 + 微调：**
   - 在大量无标注文本上进行预训练，使用“掩码语言建模”（Span Corruption）任务：随机遮盖一段连续的token，让模型根据上下文重建被遮盖的内容。
   - 预训练后，可在具体任务上进行微调。

- 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”

LM对提示词非常敏感，提示词不同

- 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.

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

人工标注

- 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

难以信任：对于无法查询的问题，KB会返回空，LM可能返回一个看似正确但错误的答案

- 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)

难以修改：即使用新的数据微调，也可能对旧的保留记忆

- 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"

实体连接：把USA、America等连接在一起

- Entity embeddings can be useful to LMs *iff* you can do entity linking well!

Many techniques for training entity embeddings:

知识图谱：实体1：美国 关系：首都 实体2：华盛顿

- Knowledge graph embedding methods (e.g., TransE)

单词和实体共现

- Word-entity co-occurrence methods (e.g., Wikipedia2Vec)

对知识库对实体的描述进行编码

- Transformer encodings of entity descriptions (e.g., BLINK)

Answer: Learn a fusion layer to combine context and entity information.

学习一个融合层，单词嵌入+实体+激活函数

$$\boldsymbol{h}_j = F\left(\boldsymbol{W}_t \boldsymbol{w}_j + \boldsymbol{W}_e \boldsymbol{e}_k + b\right)$$

We assume there's a known alignment between entities and words in the sentence such that $e_k = f(w_j)$

- $\boldsymbol{w}_j$ is the embedding of word j in a sequence of words
- $\boldsymbol{e}_k$ is the corresponding entity embedding

ERNIE: Enhanced Language Representation with Informative Entities [Zhang et al., ACL 2019]

- Text encoder: multi-layer bidirectional Transformer encoder over the words in the sentence
- Knowledge encoder: stacked blocks composed of:
  - Two multi-headed attentions (MHAs) over entity embeddings and token embeddings
  - A fusion layer to combine the output of the MHAs

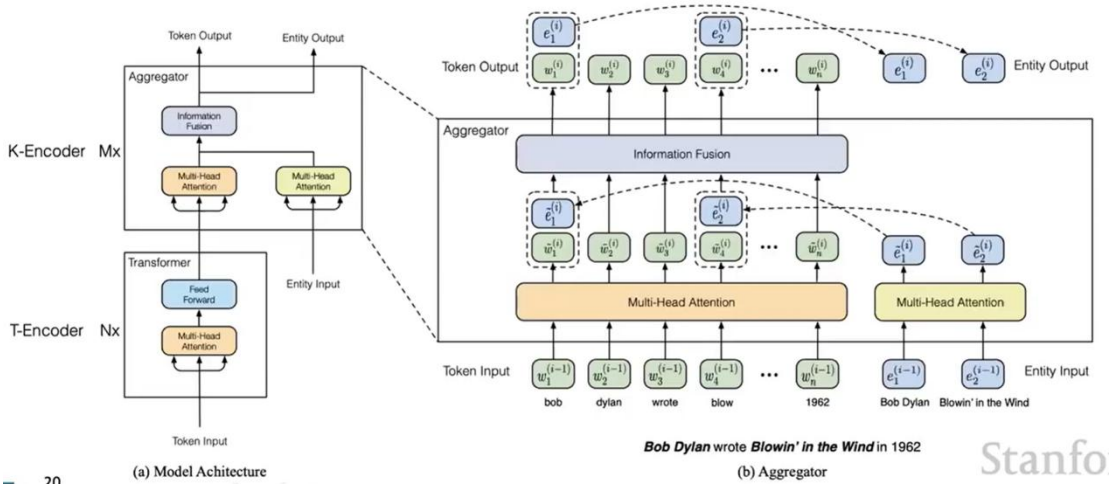
$$\boldsymbol{h}_j = \sigma\left(\widetilde{\boldsymbol{W}}_t^{(i)} \widetilde{\boldsymbol{w}}_j^{(i)} + \widetilde{\boldsymbol{W}}_e^{(i)} \widetilde{\boldsymbol{e}}_k^{(i)} + \widetilde{\boldsymbol{b}}^{(i)}\right)$$

$$\boldsymbol{w}_j^{(i)} = \sigma\left(\boldsymbol{W}_t^{(i)} \boldsymbol{h}_j + \boldsymbol{b}_t^{(i)}\right)$$

$$\boldsymbol{e}_k^{(i)} = \sigma\left(\boldsymbol{W}_e^{(i)} \boldsymbol{h}_j + \boldsymbol{b}_e^{(i)}\right)$$

用 hj 来更新词表示和实体表示

开始的实体表示是外部给出的



Pretrain with three tasks:

- Masked language model and next sentence prediction (i.e., BERT tasks)
- Knowledge pretraining task (dEA[1]): randomly mask token-entity alignments and predict corresponding entity for a token from the entities in the sequence

$$p(e_j \mid w_i) = \frac{\exp(\boldsymbol{W}\boldsymbol{w}_i \cdot \boldsymbol{e}_j)}{\sum_{k=1}^{m} \exp(\boldsymbol{W}\boldsymbol{w}_i \cdot \boldsymbol{e}_k)}$$

$$\mathcal{L}_{ERNIE} = \mathcal{L}_{MLM} + \mathcal{L}_{NSP} + \mathcal{L}_{dEA}$$

MLM：通过随机遮盖部分词并让模型预测被遮盖的词，迫使模型学习上下文信息
NSP：给定两个句子，判断第二个句子是否是第一个句子的下一句
DEA： 根据遮蔽的词预测对应的实体

增强后的词表示和实体表示可以用于下游任务

Jointly learn to link entities with KnowBERT [Peters et al., EMNLP 2019

- Key idea: pretrain an integrated entity linker (EL) as an extension to BERT

$$\mathcal{L}_{KnowBERT} = \mathcal{L}_{NSP} + \mathcal{L}_{MLM} + \mathcal{L}_{EL}$$

Predict over set of hard candidates (not just those in sentence)

- On downstream tasks, EL predicts entities so entity annotations aren't required

  和ERNIE的区别在于KnowBERT中实体连接作为模型的一部分，因此在下游任务中不需要给出实体标记

- Learning EL may better encode knowledge - shows performance gains over ERNIE on downstream tasks

- Like ERNIE, KnowBERT uses a fusion layer to combine entity and context information and adds a knowledge pretraining task

Barack's Wife Hillary: Using Knowledge-Graphs for Fact-Aware Language Modeling (KGLM) [Logan et al., ACL 2019]

- Key idea: condition the language model on a knowledge graph (KG)

一个局部的知识图谱

- Recall that language models predict the next word by computing

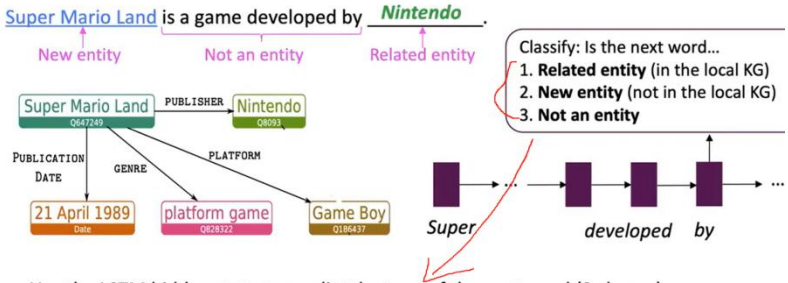
$$P(x^{(t+1)}|x^{(t)}, \dots, x^{(1)}), \text{ where } x^{(1)}, \dots, x^{(t)} \text{ is a sequence of words}$$

- Now, predict the next word using entity information, by computing

$$P(x^{(t+1)}, \mathcal{E}^{(t+1)}|x^{(t)}, \dots, x^{(1)}, \mathcal{E}^{(t)}, \dots, \mathcal{E}^{(1)})$$

where $\mathcal{E}^{(t)}$ is the set of KG entities mentioned at timestep t

KGLM [Logan et al., ACL 2019]



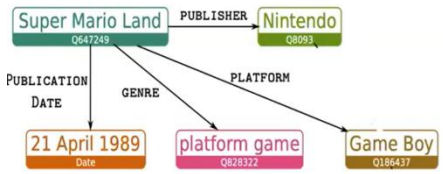
- Use the LSTM hidden state to predict the type of the next word (3 classes)
- How does the LM predict the next entity and word in each case?

Super Mario Land is a game developed by Nintendo.
New entity Not an entity Related entity

Related entity (in the local KG)

父实体，关系，尾部实体

KG triple = (parent entity, relation, tail entity)



Example
Top scoring parent entity: "Super Mario Land"
Top scoring relation: "publisher"
-> Next entity is "Nintendo", due to KG triple
(Super Mario Land, *publisher,* Nintendo).

Related entity (in the local KG)

- Find the top-scoring parent and relation in the local KG using the LSTM hidden state and pretrained entity and relation embeddings
  - $P(p_t) = \text{softmax}(\boldsymbol{v_p} \cdot \boldsymbol{h_t})$, where p_t is the "parent" entity, $\boldsymbol{v_p}$ is the corresponding entity embedding, and $\boldsymbol{h_t}$ is from the LSTM hidden state

这里其实用了实体的嵌入

- **Next entity:** tail entity from KG triple of (top parent entity, top relation, tail entity)
- **Next word:** most likely next token over vocabulary expanded to include entity aliases[1]

一个实体可能有多个别名

New entity (not in the local KG)

- Find the top-scoring entity in the full KG using the LSTM hidden state and pretrained entity embeddings
- **Next entity:** directly predict top-scoring entity
- **Next word:** most likely next token over vocabulary expanded to include entity aliases

Not an entity 正常的LSTM

- **Next entity:** None
- **Next word:** most likely next token over standard vocabulary

通过改变知识图谱可以改变预测

More recent takes: **Nearest Neighbor Language Models (kNN-LM)**

[Khandelwal et al., ICLR 2020]

- Key idea: learning similarities between text sequences is easier than predicting the next word
  - Example: *"Dickens is the author of _____" ≈ "Dickens wrote_____"*
  - Qualitatively, researchers find this is especially true for "long-tail patterns", such as rare facts 学习序列间的相似度比预测下一个单词更容易
- So, store all representations of text sequences in a nearest neighbor datastore!
- At inference:
  1. Find the *k* most similar sequences of text in the datastore
  2. Retrieve the corresponding values (i.e. the next word) for the *k* sequences
  3. Combine the kNN probabilities and LM probabilities for the final prediction

$$P(y|x) = \lambda P_{kNN}(y|x) + (1-\lambda)P_{LM}(y|x)$$

1. 预处理阶段：构建"记忆库"（Key-Value Store）

- 使用预训练语言模型（如 GPT-2）对**整个训练集**进行前向传播。
- 在每个时间步 t，记录：
  - Key：模型某一层（通常是最后一层）的隐藏状态 $\mathbf{h}_t$
  - Value：该位置的真实下一个词 y_t （即 ground truth token）

步骤 1：获取当前隐藏状态

- 给定输入序列，运行 GPT-2 到当前位置 t，得到隐藏状态 $\mathbf{h}_t$。

步骤 2：检索 k 个最相似的历史状态

- 在记忆库中，使用 $\mathbf{h}_t$ 作为查询向量，通过**最近邻搜索**（如 FAISS 加速）找到最相似的 k 个历史隐藏状态 $\mathbf{h}_{t-i}$。
- 对应的 k 个真实输出词就是检索结果。

步骤 3：基于检索结果计算概率

- 计算每个词 w 的 KNN 概率：

$$P_{\text{knn}}(w|x_{<t}) = \sum_{i \in \text{neighbors}} \mathbf{1}(y_i = w) \cdot \text{sim}(\mathbf{h}_t, \mathbf{h}_i)$$

  - 即：所有检索到的邻居中，输出为 w 的那些样本的相似度之和。

步骤 4：与原始模型概率插值

- 将 KNN 概率与原始语言模型概率 $P_{\text{LM}}(w|x_{<t})$ 进行加权平均：

$$P_{\text{combined}}(w) = (1-\lambda)P_{\text{LM}}(w) + \lambda P_{\text{knn}}(w)$$

  - λ 是一个可学习或调优的插值权重。

Pretrained Encyclopedia: Weakly Supervised Knowledge-Pretrained Language Model (WKLM) [Xiong et al., ICLR 2020]

直接修改训练数据，这样不需要额外的存储和计算

- Key idea: train the model to distinguish between true and false knowledge

- Replace mentions in the text with mentions that refer to different entities of the same type to create negative knowledge statements
  - Model predicts if entity as been replaced or not
  - Type-constraint is intended to enforce linguistically correct sentences

模型需要区分实体是否被替换

True knowledge statement: J.K. Rowling is the author of Harry Potter.

Negative knowledge statement: J.R.R. Tolkien is the author of Harry Potter.

- Uses an entity replacement loss to train the model to distinguish between true and false mentions

注意这里的指示函数和对数

$$\mathcal{L}_{entRep} = \mathbb{1}_{e\in\mathcal{E}^+} \log P(e \mid C) + (1 - \mathbb{1}_{e\in\mathcal{E}^+}) \log(1 - P(e \mid C))$$

where e is an entity, C is the context, and $\mathcal{E}^+$ represents a true entity mention

- Total loss is the combination of standard masked language model loss (MLM) and the entity replacement loss.

$$\mathcal{L}_{WKLM} = \mathcal{L}_{MLM} + \mathcal{L}_{entRep}$$

- MLM is defined at the token-level; entRep is defined at the entity-level

  - Can we encourage the LM to learn factual knowledge by being clever about masking?
  - Thread in several recent works:
    - ERNIE[1]: Enhanced Representation through Knowledge Integration, Sun et al., arXiv 2019
      - Shows improvements on downstream Chinese NLP tasks with phrase-level and entity-level masking 短语级别掩码和实体级别掩码
    - How Much Knowledge Can You Pack Into the Parameters of a Language Model?, Roberts et al., EMNLP 2020
      - Uses "salient span masking" (Guu et al., ICML 2020) to mask out salient spans (i.e. named entities and dates) 显著跨度掩码：遮蔽重要内容，如实体和日期
      - Shows that salient span masking helps T5 performance on QA

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

Lec 多模态

Multimodal distributional semantics (Bruni et al., 2014)

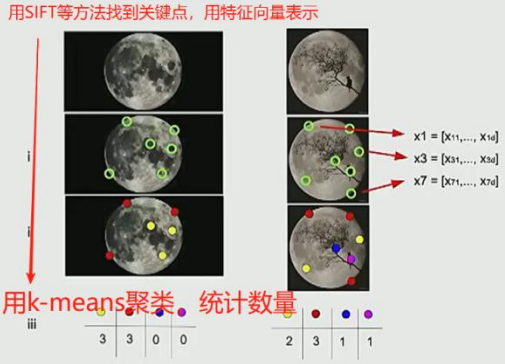
Algorithm:

视觉单词袋

- Obtain visual “word vector” via BOVW:
  - Identify keypoints and get their descriptors
  - Cluster these and map to counts
- Concatenate with textual word vector
- Apply SVD to “fuse” information

奇异值分解

This approach was shown to lead to better word representations on human similarity judgment datasets.



Bag of visual words (BOVW) illustration. Bruni et al., 2014.

Multimodal fusion

Similarity

- Inner product: **uv**

Linear / sum

- Concat: W[**u**,**v**]
- Sum: W**u**+V**v**
- Max: max(W**u**, V**v**)

Multiplicative

- Multiplicative: W**u**⊙V**v**
- Gating: σ(W**u**)⊙V**v**
- LSTM-style: tanh(W**u**)⊙V**v**

Attention

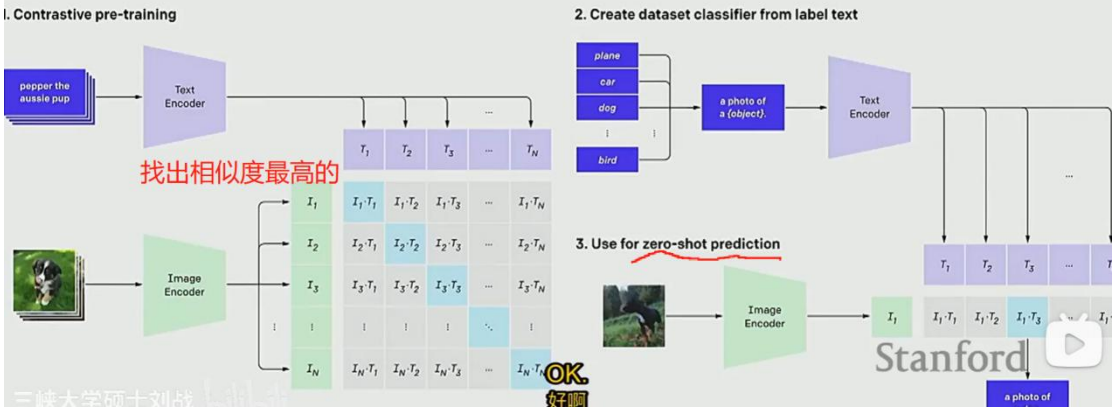
- Attention: αW**u**+βV**v**
- Modulation: [α**u**,(1−α)**v**]

Bilinear

- Bilinear: **u**W**v**
- Bilinear gated: **u**Wσ(**v**)
- Low-rank bilinear: $\mathbf{u}U^{T}V\mathbf{v}=P(U\mathbf{u}\odot V\mathbf{v})$
- Compact bilinear: $FFT^{-1}(FFT(\Psi(\mathbf{x},\mathbf{h}_1,\mathbf{s}_1))\odot FFT(\Psi(\mathbf{x},\mathbf{h}_2,\mathbf{s}_2)))$

CLIP (Radford et al. 2021)

Exact same contrastive loss as earlier, but.. Transformers and *web data*!



1. 统一的 Transformer 架构处理图像与文本

VisualBERT 使用一个共享的 Transformer 编码器，同时处理图像和文本输入。它将图像和文本都转换为序列形式，然后拼接在一起输入到 Transformer 中，让模型在统一的语义空间中学习跨模态的对齐和交互。

- **文本部分**：使用标准的 WordPiece 分词，得到词向量序列。
- **图像部分**：使用 Faster R-CNN 等目标检测模型提取图像中的感兴趣区域（Regions of Interest, RoIs），每个区域对应一个特征向量（如 2048 维），形成图像特征序列。用卷积+残差连接的方法

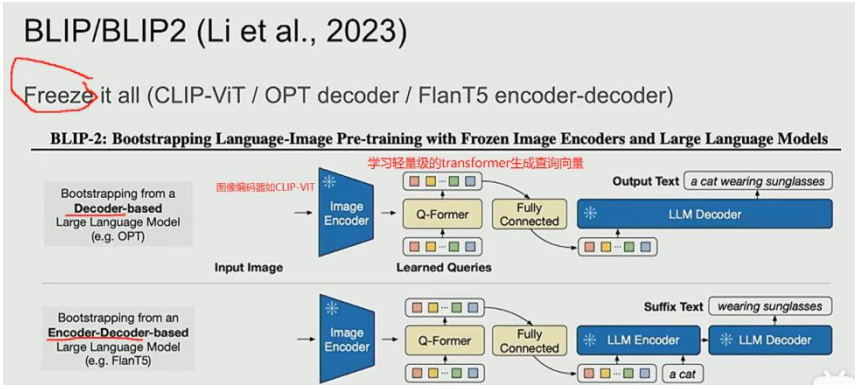
然后将图像特征序列和文本词向量序列 **拼接成一个长序列**，送入 Transformer 编码器。

---

2. 图像区域作为"视觉词"

文本和图像的注意力可以交互

VisualBERT 将图像中的每个检测区域（region）视为一个"视觉词"（visual word），与文本中的每个词（token）地位对等。这样，图像和文本都被表示为序列，可以被同一个自注意力机制处理。



✅ **情况一：使用预训练大模型 + 提示工程（Prompting）—— 不需要额外训练**

这是最常见的用法，比如使用 GPT-3、GPT-3.5、GPT-4、PaLM、LLaMA 等大模型。

- **做法**：在输入中加入"思维链示例"作为提示（few-shot prompting），引导模型自己生成推理步骤。
- **例子**：

```
Q: 小明有5个苹果……（前面的例子）
A: 小明一开始有5个苹果……答案是6。

Q: 小红有10元，买笔花了4元，买本子花了3元，还剩多少？
A:
```

模型会模仿上面的格式，自动写出推理过程。

- **优点**：无需训练，只需设计好提示（prompt）。
- **适用条件**：模型参数量足够大（通常 > 100B），才能"理解"并模仿这种推理模式。

有些研究或应用希望模型**更稳定、更擅长推理**，于是会进行专门训练：

- **监督微调**（Supervised Fine-tuning）：
  - 使用大量带有"思维链"标注的数据（即问题 + 推理过程 + 答案）来微调模型。
  - 例如：用 GSM8K 数据集（小学数学题，带人工写的推理步骤）训练模型。
- **强化学习**（RL）：
  - 结合奖励机制，鼓励模型生成正确且合理的推理链。

📌 **结论**：如果你想让一个较小的模型也能稳定地进行思维链推理，或者追求更高性能，**就需要训练或微调**。

---

3. 多模态思维链（如你图中的例子）是否需要训练？

是的，**多模态思维链通常需要专门训练或微调**，因为：

- 它要融合图像和文本信息。
- 要学会从图像中提取语义（比如"薯条是咸的"）。
- 要结合视觉和语言进行联合推理。

像你图中提到的 Zhang et al., 2023 的工作，很可能使用了**多模态数据集**（图像 + 问题 + 推理链 + 答案）来训练模型，使其能生成跨模态的推理过程。

Lec Evaluation

Finding model shortcuts via **diagnostic tests**

What if our model is using simple heuristics to get good accuracy?

A **diagnostic test set** is carefully constructed to test for a specific skill or capacity of your neural model.

For example, **HANS**: (Heuristic Analysis for NLI Systems) tests syntactic heuristics in NLI

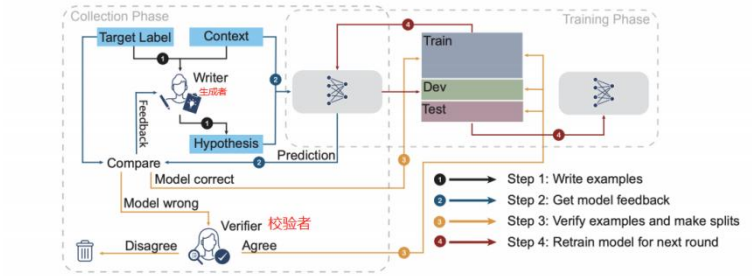
这些是错误的例子

	Heuristic	Definition	Example
词汇重叠	Lexical overlap	Assume that a premise entails all hypotheses constructed from words in the premise	**The doctor** was **paid** by **the actor**. → WRONG The doctor paid the actor.
子序列	Subsequence	Assume that a premise entails all of its contiguous subsequences.	The doctor near **the actor danced**. → WRONG The actor danced.
成分	Constituent	Assume that a premise entails all complete subtrees in its parse tree.	If **the artist slept**, the actor ran. → WRONG The artist slept.

Adversarial (and multi objective) benchmarking

Adversarial NLI (ANLI)

对抗性基准测试



Content overlap metrics

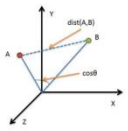
效果不好

Ref: They walked to the grocery store .

Gen: The woman went to the hardware store .

- Compute a score that indicates the lexical similarity between *generated* and *gold-standard (human-written) text*
- Fast and efficient and widely used
- *N*-gram overlap metrics (e.g., **BLEU**, ROUGE, METEOR, CIDEr, etc.)

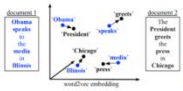
Model-based metrics: Word distance functions



Vector Similarity

Embedding based similarity for semantic distance between text.

- **Embedding Average** (Liu et al., 2016)
- **Vector Extrema** (Liu et al., 2016)
- **MEANT** (Lo, 2017)
- **YISI** (Lo, 2019)



Word Mover's Distance

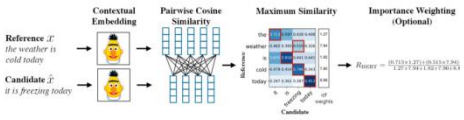
Measures the distance between two sequences (e.g. sentences, paragraphs, etc.), using word embedding similarity matching.

(Kusner et.al., 2015; Zhao et al., 2019)

BERTSCORE

Uses pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity.

(Zhang et.al. 2020)



Human evaluations

人类的评价是最好的，但也存在很多问题

- Ask *humans* to evaluate the quality of generated text

- Overall or along some specific dimension:
  - fluency
  - coherence / consistency
  - factuality and correctness
  - commonsense
  - style / formality
  - grammaticality
  - typicality
  - redundancy

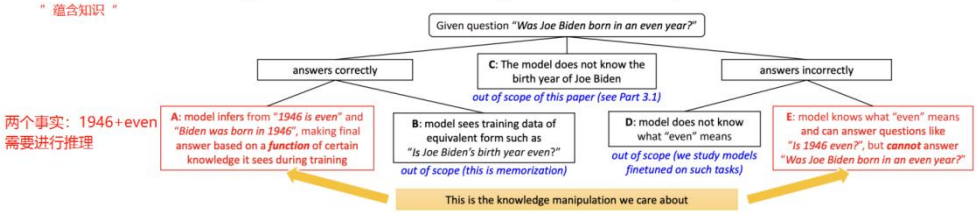
Note: Don't compare human evaluation scores across differently conducted studies

Even if they claim to evaluate the same dimensions!

Lec QA

Entailed knowledge is hard to learn (without CoT)

" 蕴含知识 "



'Reversal curse' constrains knowledge use

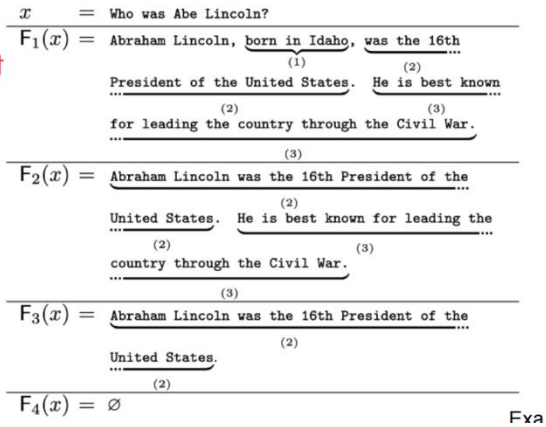
能正向推理却不能反向推理

Methods for abstention

'Backing off'

- Estimate the *confidence* of a statement (by sampling or self-scoring)
- Abstain whenever the confidence is too low (or remove details)

当给出的回答置信度较低时
逐步减少输出



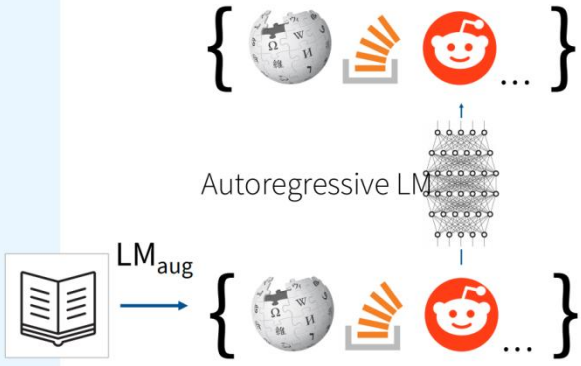
Exa

***Synthetic* continued pretraining:** Train on LLM-transformed data

在经过LLM转换过的数据上进行训练，以增加模型的知识

Goal – replicate the diversity of pretraining

- Vary content (topics)
- Vary style (how it's presented)
- Data diversity for generalization



Black-box LMs are impressive, but parametric memory has its limitations

- (Large) language models store an impressive amount of information in their parameters
  - But LMs can't memorize everything
  - The world changes over time
  - You might want it to use private documents that aren't on the web
- Also, black-box LMs are opaque
  - Given an query, it produces an answer, but it's difficult to verify if the answer is correct

Using retrieval to overcome LMs' shortcomings

检索机制

- Instead of asking the LM to memorize everything, can we provide the LM with relevant and useful content just-in-time?
- Retrieval / search is a common mechanism for identifying such relevant information.
  - **Dynamic**: it's easy to update / add documents to your retrieval system
  - **Interpretable**: LM can generate pointers to retrieved documents that support human verification of its generations (citations)

Retriever-reader framework

- Input: a large collection of documents $\mathcal{D} = D_1, D_2, \dots, D_N$ and Q
- Output: an answer string A

先检索，然后转换成阅读理解问题

- Retriever: $f(\mathcal{D}, Q) \longrightarrow P_1, \dots, P_K$ K is pre-defined (e.g., 100)
- Reader: $g(Q, \{P_1, \dots, P_K\}) \longrightarrow A$ A reading comprehension problem!

- Retriever = A standard TF-IDF information-retrieval sparse model (a fixed module)
- Reader = a neural reading comprehension model that we just learned
  - Could be Trained on SQuAD and other distantly-supervised QA datasets
  - Or a zero-shot LLM (ChatGPT etc)

Dense retrieval + generative models

Recent work shows that it is beneficial to generate answers instead of to extract answers.

Lec 模型分析与解释

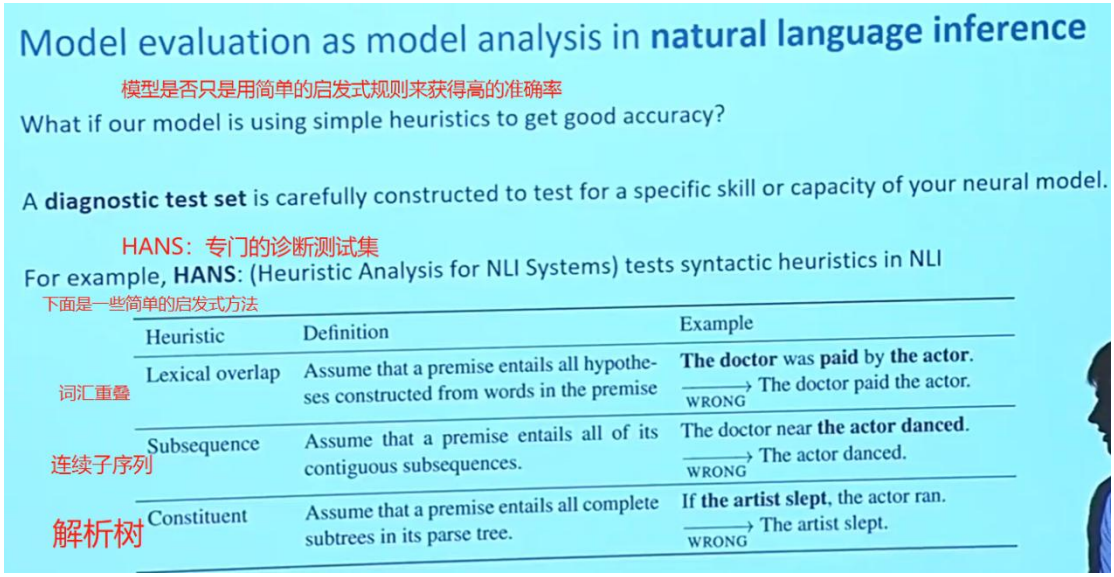
BLEU（Bilingual Evaluation Understudy，双语评估替代指标）分数是一种用于评估机器翻译质量的自动评价指标。它通过比较机器翻译生成的文本与一个或多个参考人工翻译之间的相似度，来衡量翻译的准确性。BLEU分数广泛应用于自然语言处理（NLP）领域，尤其是在机器翻译、文本生成等任务中。

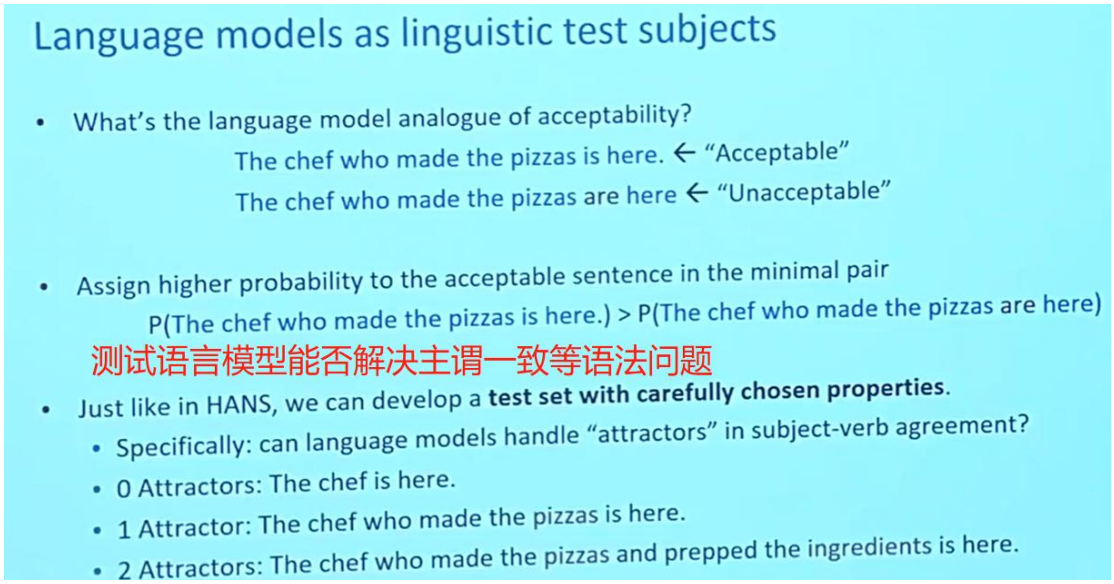
BLEU分数的核心思想：

BLEU分数基于n-gram**精度**（n-gram precision），即统计机器翻译结果中多少n个连续词的组合（n-gram）出现在参考翻译中。同时，它还引入了一个**简洁惩罚**（Brevity Penalty, BP），以防止过短的翻译获得高分。

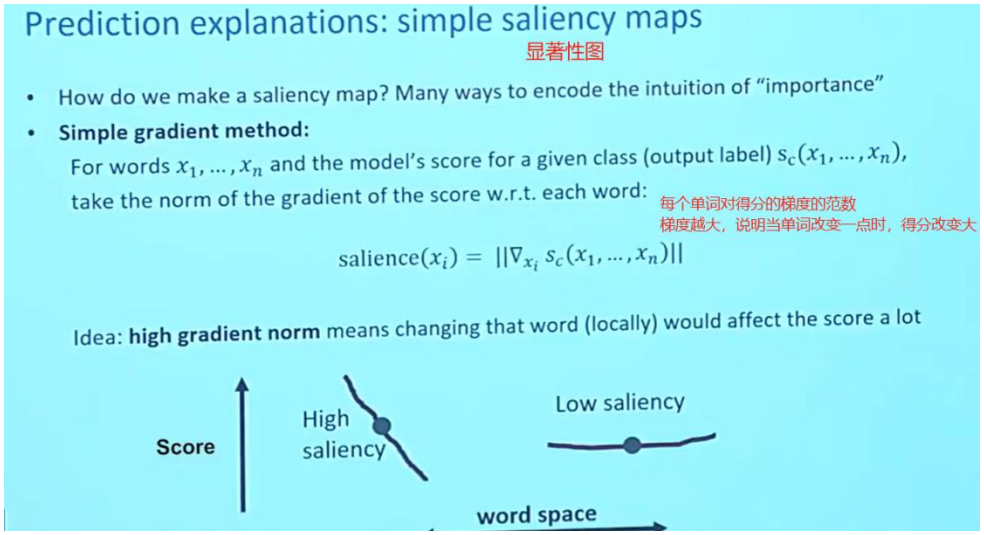
BLEU分数的计算步骤：

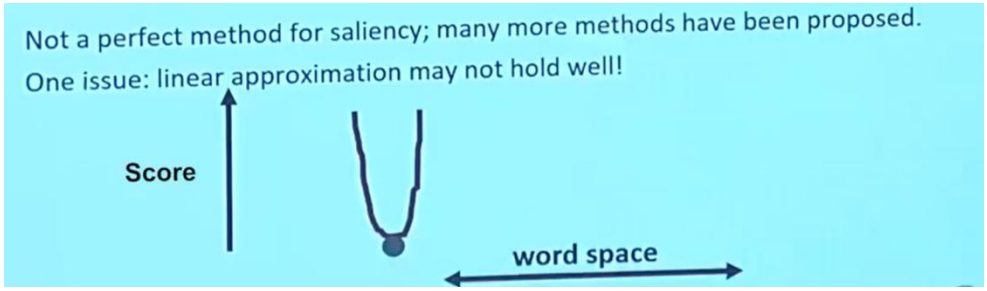
1. n-gram**精度计算**：
   - 计算机器翻译结果中1-gram、2-gram、3-gram、4-gram等与参考翻译匹配的数量。
   - 对每个n-gram，取匹配数与总n-gram数的比例，并进行加权平均。





精心设计的小型测试集，类似于软件开发中的单元测试套件

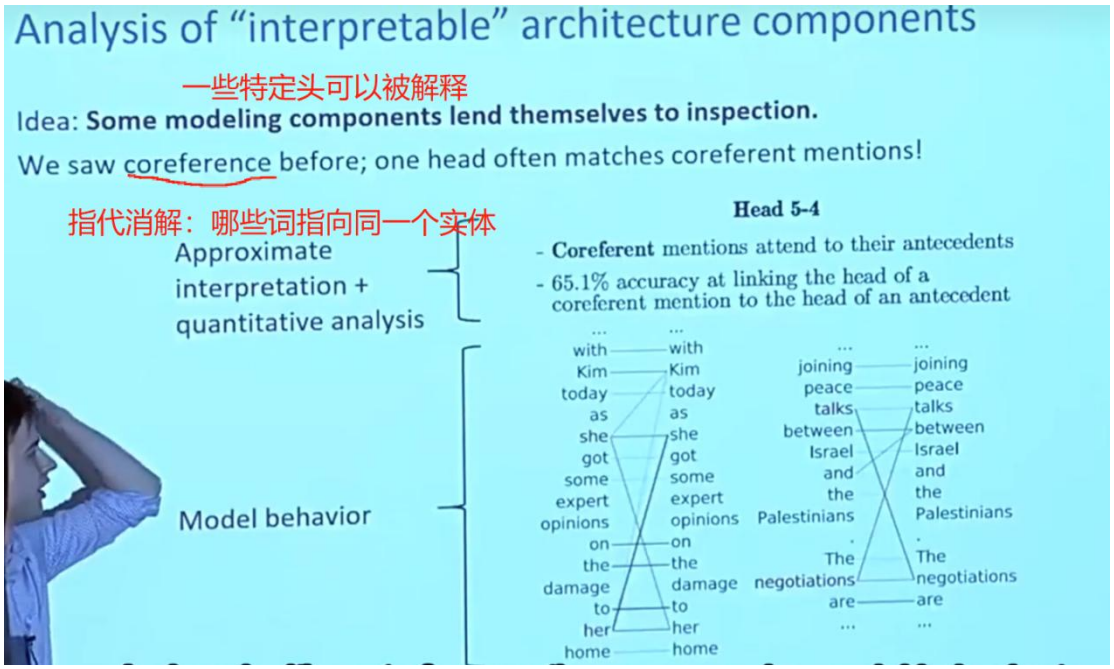


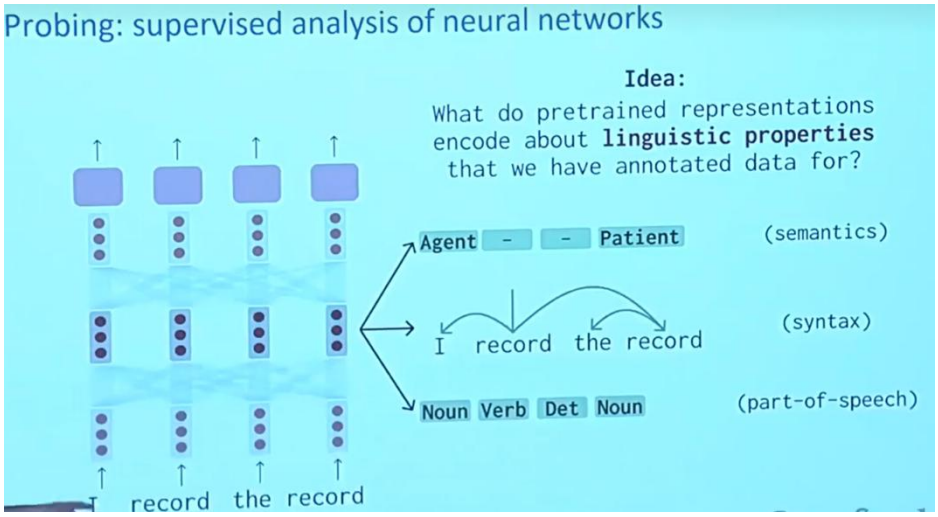


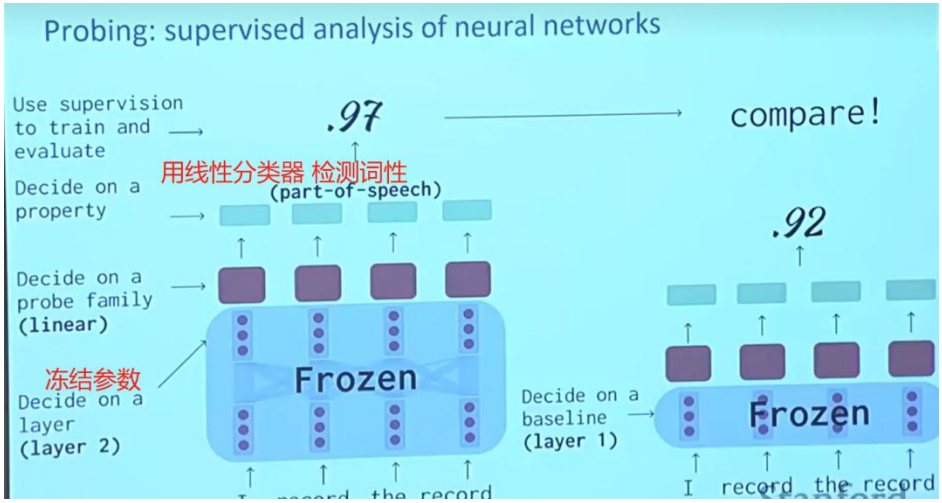
一些研究显示，当减少提问的信息(比如从 what did he fund -> did)，模型的回答没有改变；而改变模型训练时的数据(如把 what 改为 whats，增加干扰的文本等)，模型会回答错误。说明模型可能没有学习到我们希望它学习的模式，对噪声的鲁棒性并不是很好。（捷径学习？）

五、如何改进？—— 提升模型的鲁棒性与真正理解

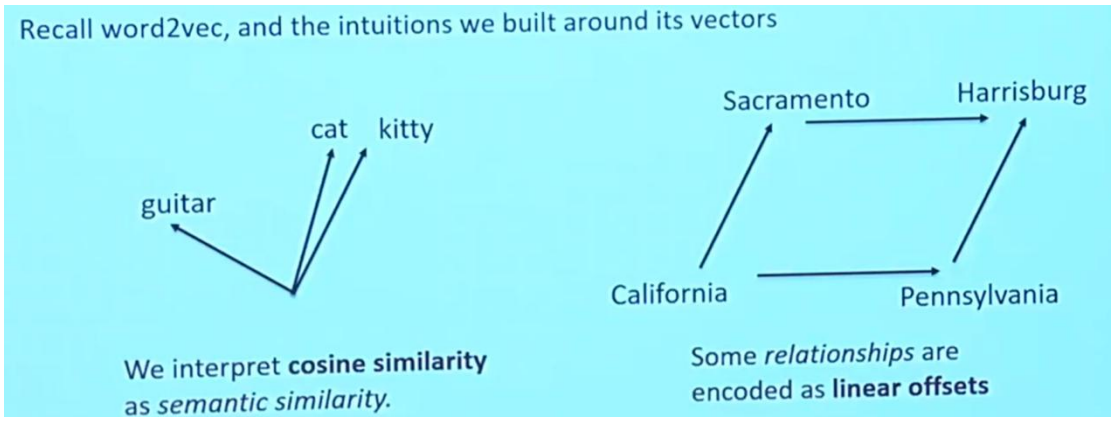
方法	说明
对抗训练（Adversarial Training）	在训练中加入扰动样本（如拼写错误、同义替换），迫使模型关注语义而非表面形式。
数据增强（Data Augmentation）	自动生成变体问题（如"whats", "what's", "which thing"），提升泛化能力。
对比学习（Contrastive Learning）	让模型区分语义相同但形式不同的句子 vs. 形式相似但语义不同的句子。
可解释性分析（如显著性图）	检查模型是否关注正确的词（如"what"而不是"fund"），用于诊断和干预。
结构化监督（Structured Supervision）	引入中间表示（如依存句法、语义角色标注）作为监督信号，引导模型学习深层结构。



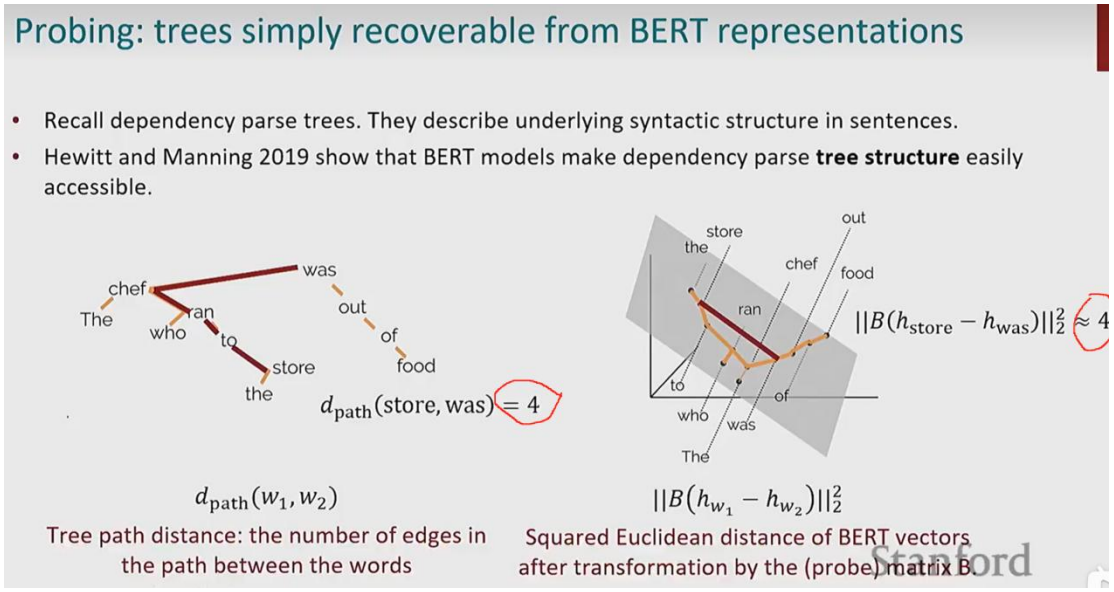




Increasingly abstractlinguistic propertiesare more accessiblelater in the network.



探针矩阵 B 是一个线性变换矩阵，用于将 BERT 的隐藏层表示映射到一个新的空间



Lec NLP 和深度学习的未来

- General Representation Learning Recipe:
  - Convert your data (images, text, videos) into sequence of integers.
  - Define a loss function to maximize data likelihood or create a denoising auto encoder loss.
  - Train on ***lots*** of data

GPT-3: Limitations and Open questions

- Seems to do poorly on more structured problems that involve decomposing into atomic / primitive skills:
  - RTE / arithmetic / word problems / analogy making

文本蕴含 类比推理

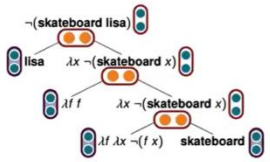
与用户交互得到的知识无法长期记忆

- Performing permanent knowledge updates interactively is not well studied.
- **Doesn't seem to exhibit human like generalization (systematicity).**
- **Language is situated and GPT-3 is merely learning from text without being exposed to other modalities.**

Are neural representations compositional? [Andreas 2019]

最小化神经网络输出与通过组合规则重建的输出之间的差异进行评估

NN(Lisa does not skateboard) ≈
f(v(Lisa), f(v(does), f(v(not), v(skateboard))))



leaf vectors as well as the composition operator are *learnt by TRE*

Tree Reconstruction Error (TRE)

First choose :

- a distance function $\delta : \Theta \times \Theta \to [0, \infty)$ satisfying $\delta(\theta, \theta') = 0 \Leftrightarrow \theta = \theta'$
- a composition function $* : \Theta \times \Theta \to \Theta$

Define $\hat{f}_\eta(d)$, a *compositional approximation to* f with parameters η, as:

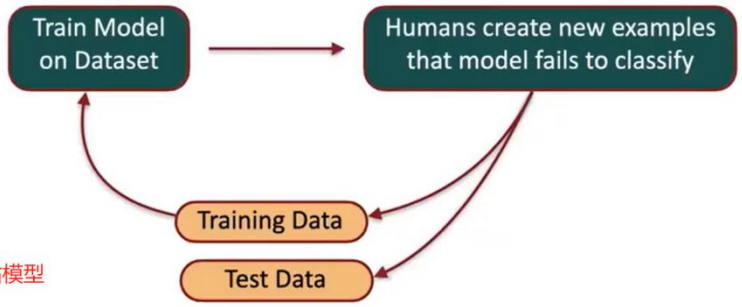
$$\hat{f}_\eta(d_i) = \eta_i \qquad \text{for } d_i \in \mathcal{D}_0$$
$$\hat{f}_\eta((d, d')) = \hat{f}_\eta(d) * \hat{f}_\eta(d') \qquad \text{for all other } d$$

Given a dataset $\mathcal{X}$ of inputs x_i with derivations $d_i = D(x_i)$, compute:

$$\eta^* = \arg\min_\eta \sum_i \delta(f(x_i), \hat{f}_\eta(d_i))$$

Then we can define datum- and dataset-level evaluation metrics:

$$\text{TRE}(x) = \delta(f(x), \hat{f}_{\eta^*}(d))$$
$$\text{TRE}(\mathcal{X}) = \frac{1}{n} \sum_i \text{TRE}(x_i)$$



动态地评估模型

- Main Challenges: Ensuring that humans are able to come up with hard examples and we are not limited by creativity.
- Current approaches use examples from other datasets for the same task as prompts

Grounding Language to other modalities

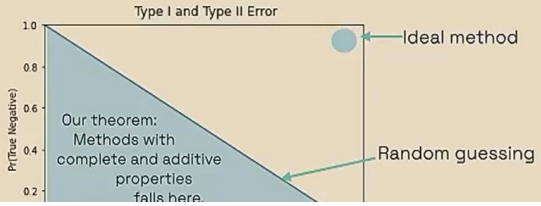
1. Open questions in this space:
    1. Given that we might need to move beyond just text, what is the best way to do this at scale?
    2. Babies cannot learn language from watching TV alone [Snow et al 1976] but how far can models (especially when combined with scale)?
    3. If interactions with the environment is necessary, how do we collect data and design systems that interact minimally or in a cost effective way?
    4. Could pre-training on text still be useful by making any of the above more sample-efficient?

Lec 模型可解释性与编辑

Theoretical performance guarantees for feature attribution

- Main theorem sketch:
    - formulation: Interpreting attribution <> a hypothesis testing on the shape of the function (e.g., recourse, spurious correlation)因果关系，虚假相关性
    - result: popular feature attribution methods (e.g., SHAP, IG) to conduct hypothesis testing about the model's behavior near a single data point (local explanation) implies:

p(true positive) <= 1- p(true negative)



Stanford

1. IG（Integrated Gradients）

定义：

- Integrated Gradients (IG) 是一种基于梯度的特征归因方法，用于解释深度神经网络等可微模型的预测结果。
- 它通过积分从“基线”(baseline）到实际输入的梯度来计算每个特征的重要性。

核心思想：

- 假设模型是可微的（例如神经网络），IG通过计算从一个“参考输入”(通常是零向量或某个中性输入）到当前输入的路径上的梯度累积，来量化每个特征对模型输出的贡献。
- 这种方法基于**微积分中的路径积分**，确保归因满足一些理想的性质，如**完整性**（Completeness）和**敏感性**（Sensitivity）。

公式：

对于一个输入 x 和基线 x'，特征 i 的归因值为：

$$\mathrm{IG}_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

其中：

- f 是模型的预测函数。
- x_i 是第 i 个特征的实际值。
- x_i' 是第 i 个特征的基线值。
- 积分表示从基线到实际输入的路径上梯度的累积。

- SHAP 是一种基于博弈论的特征归因方法，灵感来源于**Shapley值**（Shapley Value），这是合作博弈论中用来公平分配收益的一种方法。
- SHAP 将每个特征视为一个“玩家”，并将模型的预测结果视为“总收益”，然后计算每个特征对预测结果的贡献。

核心思想：

- SHAP 债务于 Shapley 值的数学框架，确保每个特征的归因值满足以下性质：
  1. **效率性**（Efficiency）：所有特征的归因值之和等于模型预测与基准预测的差值。
  2. **对称性**（Symmetry）：如果两个特征在所有情况下对模型的贡献相同，则它们的归因值也相同。
  3. **零贡献性**（Null Player）：如果某个特征对模型没有任何贡献，则其归因值为零。
  4. **可加性**（Additivity）：多个模型的归因值可以线性组合。

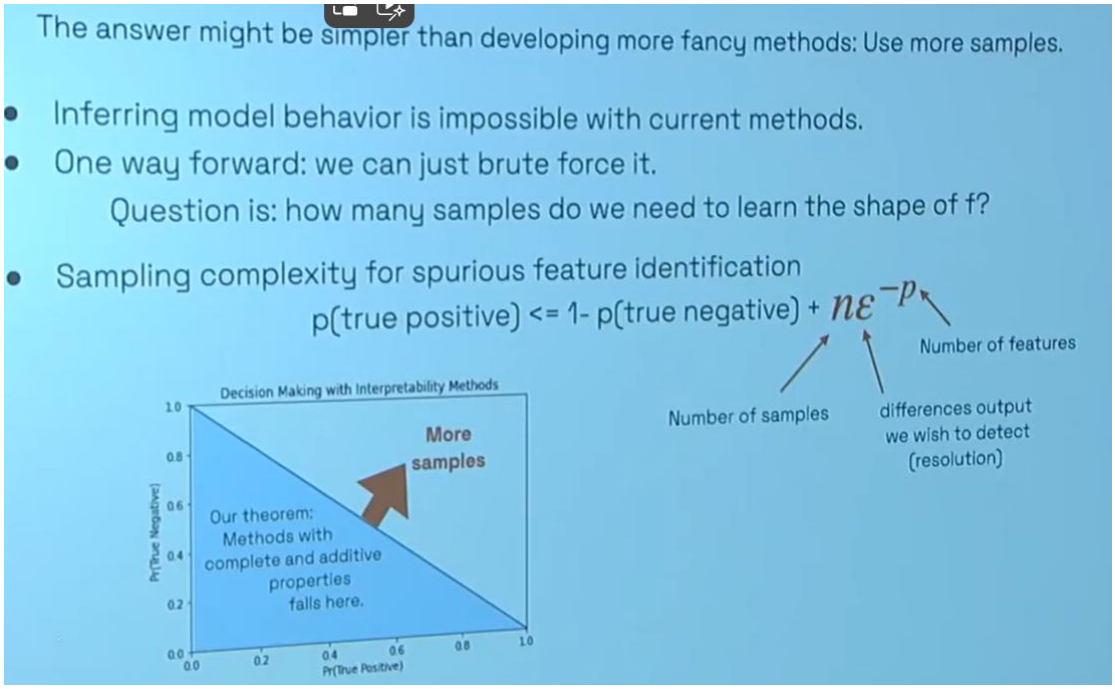
公式：

对于一个输入 x ，特征 i 的 SHAP 值为：

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f(S \cup \{i\}) - f(S)]$$

其中：

- N 是所有特征的集合。
- S 是不包含特征 i 的特征子集。
- $f(S)$ 是仅使用特征子集 S 时模型的预测值。
- M 是特征总数。



TL;DR: model edit success is unrelated to location of factual information in models

- Existing assumption:
  - Model editing is about changing what the model knows about.
  - We find where a fact is stored first then we edit it.

包含大量事实的知识并不存储在我们认为存储知识的层中

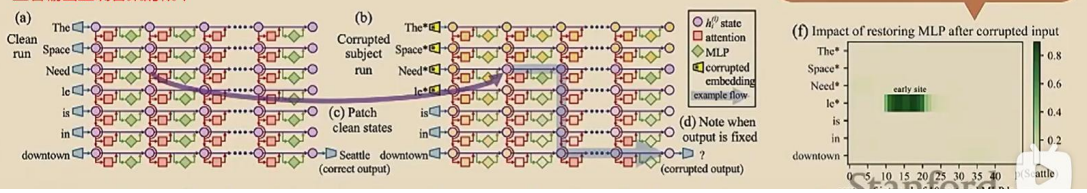
- TL;DR of our work: the assumption is not true.
  1. Substantial fraction of factual knowledge is stored **outside** of layers 'identified' as storing knowledge.
  2. The correlation between location vs editing is **near zero** (for ROME [Meng et al. 2022], MEMIT [Meng et al. 2022] and Adam-based fine-tuning).
  3. Our (unsuccessful) attempts to recover connection between location vs editing.

Method of interest: ROME and MEMIT [Meng et al. 2022 a, b]

- "Causal tracing" algorithm:
  1. Run the model on the input (s,r,o) e.g., (The space needle, located in, Seattle)
  2. Add noise to the embedding of s, then continue.
  3. Intervention: copy embeddings from 1 into 2, then continue.
  4. Calculate p(Seattle|noising, intervention)

Assumption:This tells us which layer to edit
(In reality ROME uses layer 6 for all edits, avg. causal tracing results)

正常运行->在主体的embedding添加噪声运行->将正确的embedding复制到添加噪声的位置，查看输出正确答案的概率



- Q. Localize helps editing?
- A. no. There is no relationship between localization and editing success (for current localization methods, GPT-j + CounterFact data)
- Q. Are there any other editing that correlate better with localization?
- A. no.

因果追踪　　事实信息在前向传播某个位置被传递不意味着是更改信息的最好位置

- Causal tracing = factual information is carried in presentations in Transformer's forward pass, and that only. != where is best to intervene to change the factual information.

Thoughts:

- Causal tracing revealed the role that early-to-mid-range MLP representations at the last subject token index play in factual association. -> useful
- Important NOT to (1) validate the results of localization via editing or (2) motivate the editing method via localization.