目录

神经网络的提升(避免 local optimal)总结为 。如 dropout，batch gradient 增加 noise

词性标注

Number Tag Description

1. CC Coordinating conjunction 连词
2. CD Cardinal number 基数
3. DT Determiner 限定词
4. EX Existential there
5. FW Foreign word
6. IN Preposition or subordinating conjunction 从属连词
7. JJ Adjective 形容词
8. JJR Adjective, comparative 形容词比较级
9. JJS Adjective, superlative 形容词最高级
10. LS List item marker 列表项标记
11. MD Modal 情态动词
12. NN Noun, singular or mass
13. NNS Noun, plural
14. NNP Proper noun, singular 专有名词，单数
15. NNPS Proper noun, plural 专有名词，复数
16. PDT Predeterminer 前置限定词
17. POS Possessive ending 所有格结束词
18. PRP Personal pronoun 人称代词
19. PRP$ Possessive pronoun 物主代词
20. RB Adverb 副词
21. RBR Adverb, comparative 副词，比较级
22. RBS Adverb, superlative 副词，最高级
23. RP Particle 分词
24. SYM Symbol 符号
25. TO to
26. UH Interjection 感叹词
27. VB Verb, base form 动词，原形
28. VBD Verb, past tense 动词，过去式
29. VBG Verb, gerund or present participle 动词，动名词或现在分词
30. VBN Verb, past participle 动词，过去分词
31. VBP Verb, non-3rd person singular present 动词，非第三人称单数
32. VBZ Verb, 3rd person singular present 动词，第三人称单数
33. WDT Wh-determiner Wh-限定词
34. WP Wh-pronoun Wh-代词（疑问代词）
35. WP$ Possessive wh-pronoun 疑问代词所有格
36. WRB Wh-adverb 疑问副词

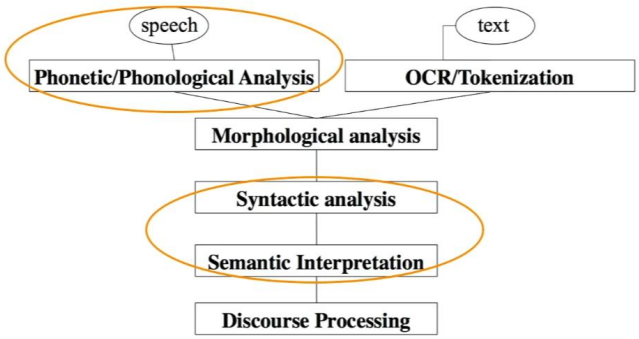
01 Introduction to NLP and Deep Learning

NLP 和 language 特点

Goal: for computers to process or "understand" natural

Fully **understanding and representing** the **meaning** of language (or even defining it) is a difficult goal.

NLP Levels



上面是获取文本

下面是对文本句法以及语义的分析

(A tiny sample of) NLP Applications

Applications range from simple to complex:

- Spell checking, keyword search, finding synonyms
- Extracting information from websites such as
  - product price, dates, location, people or company names
- Classifying: reading level of school texts, positive/negative sentiment of longer documents
- Machine translation
- Spoken dialog systems
- Complex question answering

extract information，阅读理解，从大量的文字中，web, twitter..

NLP in industry … is taking off

- Search (written and spoken)
- Online advertisement matching
- Automated/assisted translation
- Sentiment analysis for marketing or finance/trading
- Speech recognition
- Chatbots / Dialog agents
  - Automating customer support
  - Controlling devices
  - Ordering goods

decision making learn from text

从文本中挖掘有用信息！

What's special about human language?

A human language is a system **specifically constructed to convey the speaker/writer's meaning**

- Not just an environmental signal, it's a deliberate communication
- Using an encoding which little kids can quickly learn (**amazingly!**)

A human language is a **discrete/symbolic/categorical signaling system**

- rocket = 🚀; violin = 🎻
- With very minor exceptions for expressive signaling ("I loooove it." "Whoomppaaa")
- Presumably because of greater signaling reliability
- Symbols are not just an invention of logic / classical AI!

language data 的独特性：

有目的的信号，不是普通的环境信号

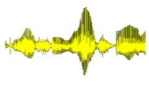
symbolic：象征的　不同的 word 不同的象征

What's special about human language?

The categorical symbols of a language can be encoded as a signal for communication in several ways:

- Sound
- Gesture
- Images (writing)

The symbol is invariant across different encodings!



CC BY 2.0 David Fulmer 2008

National Library of NZ, no known restrictions

language 可以不同的 encoding(表示方法)，**但 symbol 相同**

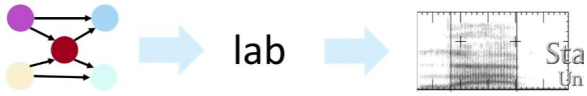
What's special about human language?

A human language is a **symbolic/categorical signaling system**

However, a brain encoding appears to be a **continuous pattern of activation**, and the symbols are transmitted via **continuous signals** of sound/vision

We will explore a continuous encoding pattern of thought

The large vocabulary, symbolic encoding of words creates a problem for machine learning – sparsity!

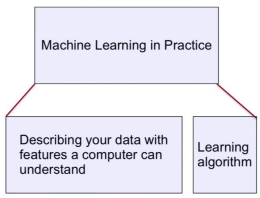


traditional ML 算法：针对一个特定问题，**人工 hands-on 提取重要的特征**
实际是**人先学习，然后再告诉 ML**，不是 machine learning，90%用来提取特征

machine learning 和 DL 的区别

Machine learning becomes just optimizing weights to best make a final prediction

learn 的是 weight



而 deep learning
直接将 raw signal 给算法，**自动提取特征 learn feature**，根据不同问题

What's Deep Learning (DL)?

- Representation learning attempts to automatically learn good features or representations

并且是多层的 representation

- Deep learning algorithms attempt to learn (multiple levels of) representation and an output

- From "raw" inputs **x** (e.g., sound, characters, or words)

On the history of and term "Deep Learning"

- We will focus on different kinds of **neural networks**
- The dominant model family inside deep learning

- Only clever terminology for stacked logistic regression units?
  - Maybe, but interesting modeling principles (end-to-end) and actual connections to neuroscience in some cases

- We will not take a historical approach but instead focus on methods which work well on NLP problems now
- For a long (!) history of deep learning models (starting ~1960s), see: Deep Learning in Neural Networks: An Overview by Jürgen Schmidhuber

St

DL =逻辑回归的 stack？

Reasons for Exploring Deep Learning

- Manually designed features are often over-specified, incomplete and take a long time to design and validate

- **Learned Features** are easy to adapt, fast to learn

- Deep learning provides a very flexible, (almost?) universal, learnable framework for **representing** world, visual and linguistic information.

- Deep learning can learn **unsupervised** (from raw text) and **supervised** (with specific labels like positive/negative)

DL 可以 learn 很多**领域的 representation**
DL 因素：硬件+big data
Improved performance (first in speech and vision, then NLP)

NLP hard

4. Why is NLP hard?

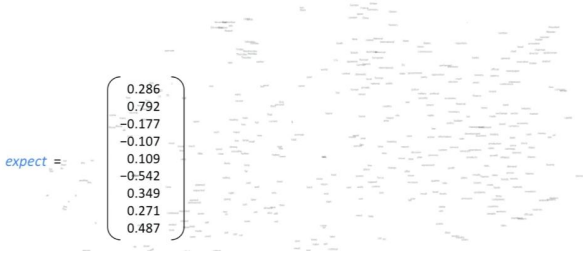
- Complexity in representing, learning and using linguistic/situational/world/visual knowledge

- Human languages are ambiguous (unlike programming and other formal languages)

- Human language interpretation depends on real world, common sense, and contextual knowledge

human language ： 1 ambiguous，2 需要根据特定的场景解释，断句，相同的词多重含义

Word meaning as a neural word vector – visualization



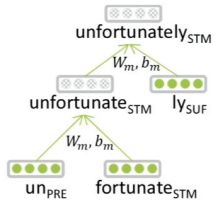
represent a word as high dimension vector，每个 element axis 意义很难理解，除非降维后再解释(但会失去很多有用的信息)有时候 misleading

Representations of NLP Levels: Morphology

- Traditional: Words are made of morphemes

prefix	stem	suffix
un	interest	ed

- DL:
  - every morpheme is a vector
  - a neural network combines two vectors into one vector
  - Luong et al. 2013



word 的表示：将 word 分解
将 parts of word(morpheme)作为 vector，多个 vector 组成一个 vector

02 Word Vector Representations word2vec

1. How do we represent the meaning of a word?

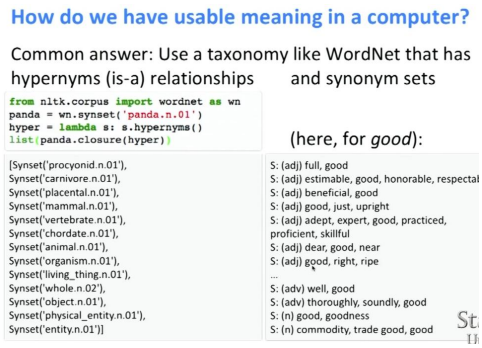
Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

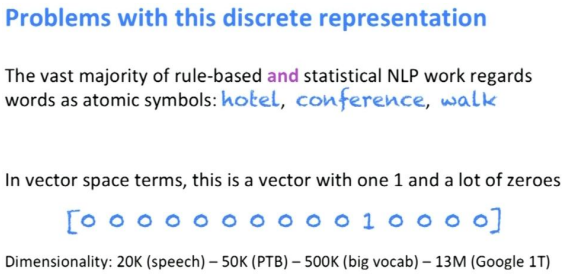
- signifier ⟺ signified (idea or thing) = denotation

denotation 表示真实世界的事物



taxonomy 分类,类别，每个 word 一个类别 同一次
上位词(hypernym)”花”是”鲜花”的上位词,”植物”是”花”的上位词
用**上位词和同义词**来表示一个**词**
会忽略 word 之间细微的差异 ； 不知道 word 直接的 similarity

discrete representation

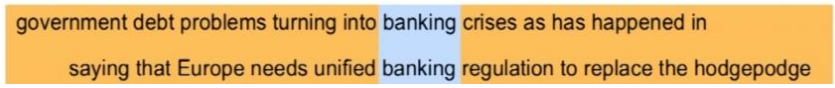


atomic symbols，word 之间是孤立的没有关系，

任意两个向量内积为 0，也就是没有关系，向量互相垂直

word distributed representation

通过**上下文**，查看 word 的不同的 neighbors，不同的 context，来 **represent word**
通过上下文来理解 word 的 meaning



不同的上下文，banking 的意义不同，用 neighbor words 来 **represent banking**

Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

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

找到 word representation 可以更好的预测 context word，或者 context word 可以更好的预测 center word

distributional similarity representation

choose the vector to predict neighbor words better！！！

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(context|w_t) = \ldots$$

which has a loss function, e.g.,

$$J = 1 - p(w_{-t}|w_t)$$

We look at many positions t in a big language corpus

We keep adjusting the vector representations of words to minimize this loss

center word 预测 context word

loss function 预测准的话概率就接近 1，loss 就小！

change vector representation of word 来降低 loss function！！最终得到 word 的 represent！！！！

Directly learning low-dimensional word vectors

Old idea. Relevant for this lecture & deep learning:

- Learning representations by back-propagating errors (Rumelhart et al., 1986)
- **A neural probabilistic language model** (Bengio et al., 2003)
- NLP (almost) from Scratch (Collobert & Weston, 2008)
- A recent, even simpler and faster model: word2vec (Mikolov et al. 2013) → intro now

那时候大家都不关注 bengio 的文章

Mikolov

1 word2vec

2. Main idea of word2vec

Predict between every word and its context words!

Two algorithms

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

Two (moderately efficient) training methods

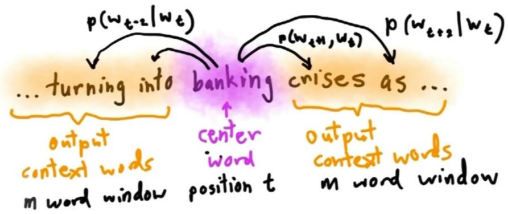
1. Hierarchical softmax
2. Negative sampling

Naïve softmax

2 种 predict 方法　predict target word or context word
2 种 train 方法　直接 softmax 计算量太大

1.1 skip-gram(待整理再提炼)

Skip-gram prediction



1 choose a center word
2 然后**前后**取 window context，**用 center word 来预测 context**，得到每个 context word 的条件概率 p(context i | target)
3 使得全部的 context word 的**联合概率**最大(假设条件独立)　**MLE**
然后上面只是一个 target word，要算上一个文本全部的 target word，全部 target word 的全部 context word 的联合概率！！！全部相乘，针对一个文本，需要同时出现。

Details of word2vec

For each word $t = 1 \dots T$, predict surrounding words in a window of "radius" m of every word.

Objective function: Maximize the probability of any context word given the current center word:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w_{t+j} | w_t ; \theta)$$

theta 其实就是 vector representation of every words！！
m 也是个 parameters！先假设是 constant
变成 minimize negative log likelihood sum：

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

就可以 gradient descent

如何 model 条件概率？

Details of Word2Vec

Predict surrounding words in a window of radius *m* of every word

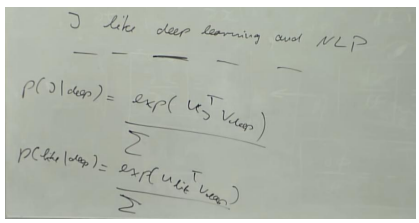
For $p(w_{t+j}|w_t)$ the simplest first formulation is

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

where *o* is the outside (or output) word index, *c* is the center word index, v_c and u_o are "center" and "outside" vectors of indices *c* and *o*

Softmax using word *c* to obtain probability of word *o*

center c word predict output word
p(o|c)概率 如果 model？softmax 直接两个 vector 相连
Vc * matix U 得到每个 output 的 score
matrix U 就是对应每个 output 的 vector: Ui

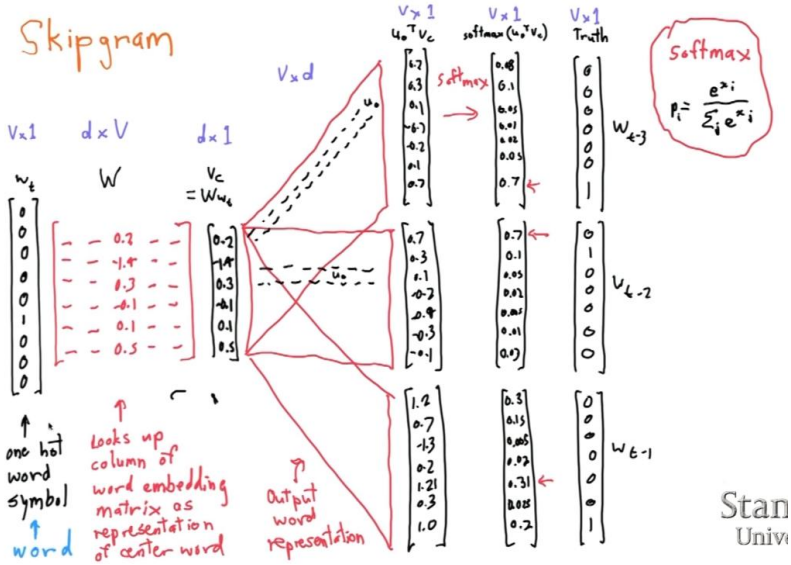


条件概率 softmax 本质：

用向量 similarity 相似度来表示概率！！！两个向量相似，内积越大，代表条件概率大！！
注意：similarity 会受向量元素大小本身影响，因此一般会除以两个向量的膜，只得到 cos 夹角(但这边不重要，因为最终还要计算 softmax，会抵消！！！)
o 和 c 都是 **index** of vocabulary c 是 center；o 是 out 或 context
V 和 U 是 matrix，用 index 来选择对于的 word 向量
v 是 target vector ; u 是 context word vector softmax 的参数矩阵 U 就是 context word vector

v 和 u 越**相似度**就代表**预测的概率，v 越能预测 u**
其实是不断拿一个 center word 和 多个 context word 计算相似度！！
然后将**相似度向量**转成条件**概率**

softmax function 的 **loss** 也是一种 **max** 方法，将 predict 条件概率，和真实概率一致：



忽略上面右边右下角 6 列向量

1center word one hot

2W matrix 是 **center word embedding matrix ，每个 vector 是 word vector**

3Ww 是 select W 一列

4 matrix U **context word embedding matrix**

5 Uv 得到与所有 context word 的内积 向量 v*1

6 softmax vocabulary 没个 word 的概率

和真实 distribution 计算 loss

因此可以看出，每个 word 有两个 vector：center word 和 context word

embedding 最初是 random 初始化为小的数值！！！

one window deep 是 center word

最上面是 embedding 矩阵，左边是每个 word v 向量的矩阵，右边是每个 word u 向量的矩阵

带入公式得到 p，然后更新 embedding 来使得 P 最大！！

Train Model

To train the model: Compute all vector gradients!

- We often define the set of all parameters in a model in terms of one long vector θ
- In our case with *d*-dimensional vector and *V* many words:

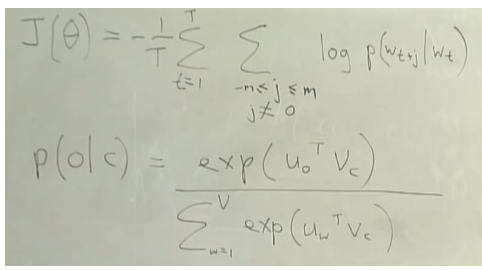
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We then optimize these parameters

Note: Every word has two vectors! Makes it simpler!

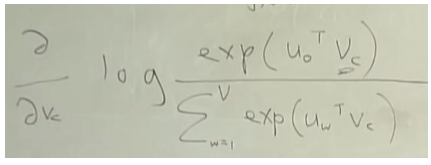
每个 word 的两个 vector 都是参数，可以表示成 big vector 长度是 2*d*V，作为一个参数！

1.2 skip-gram model 具体推导

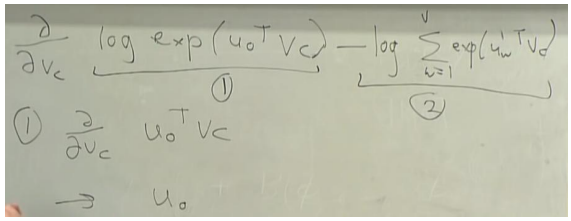


我们用 gradient descent，**参数是每个 word 的 Vc 向量和 Uo 向量**
我们想 change u 和 v vector，使得 J 最小

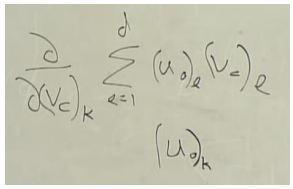
上面式子是很多 log p 的 **sum(sum)**，对每个 word 的全部 window word 的 log p 求 sum
所以我们要先对 **log p 求导**
loss 是 sum 形式，只需先对一个 term 求导
例：
123ABCDEFGH p(O|C)也就有 10 个 term 相加，也就是**含有 vC 的 term 有 10 个**，导数也就是这 10 个导数相加 ，也就是如果计算 gradient，需要加上 10 个 vC gradient term。



先对 Vc 求导，变成减式子，然后再分开求导！

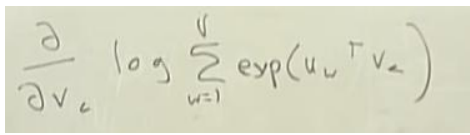


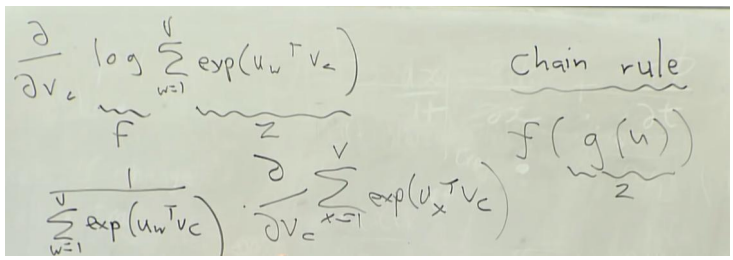
如果向量求导比较难懂，可以转换成 scalar 得到 Uo



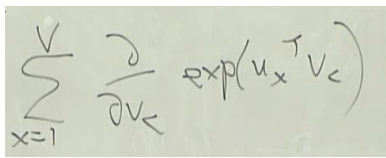
其实是一样的

右边项：函数嵌套函数 用 chain rule(就是 bp 算法)
log sum 没办法消除 exp，log(积或商)才可以变化

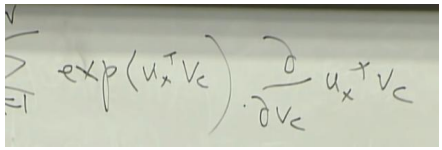


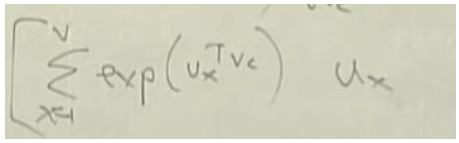


右边将下标 w 改成了 x，防止 indice 冲突

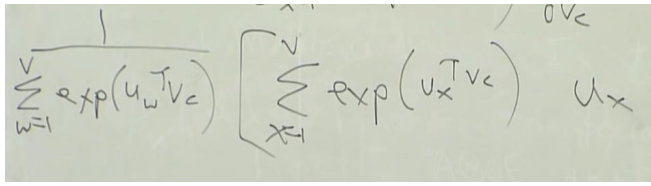


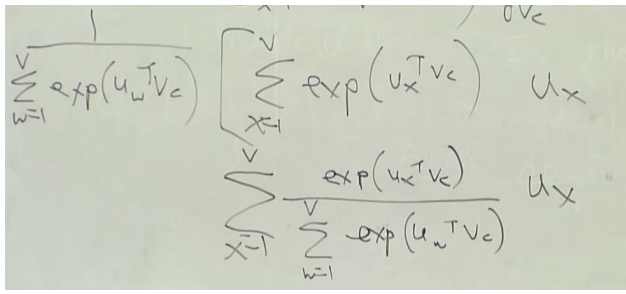
然后再 chain rule
exp 导数不变





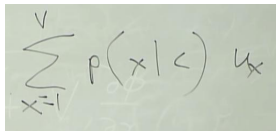
乘以前面的大项



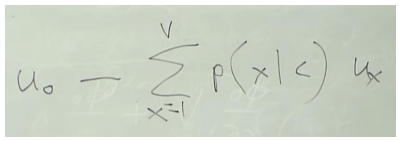


乘法分配： 注意此时就需要 change w to x 了

里面是 softmax 的形式，根据定义，其实就是 p(x|c) c word x word



最终 Vc 的梯度：



左边是实际的 output，右边是**期望**：给定 center word 下所有可能 context 的 word 的期望

expectation of vector

Calculating all gradients!

- We went through gradient for each center vector *v* in a window
- We also need gradients for outside vectors *u*
- Derive at home!

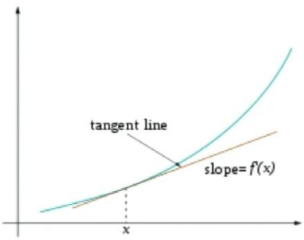
- Generally in each window we will compute updates for all parameters that are being used in that window.
- For example, window size *m* = 1, sentence:
  "We like learning a lot"
- First window computes gradients for:
  - internal vector v_{like} and external vectors u_{We} and $u_{learning}$

5. Cost/Objective functions

We will optimize (maximize or minimize) our objective/cost functions

For now: minimize → gradient descent

Trivial example: (from Wikipedia)
Find a local minimum of the function
$f(x) = x^4-3x^3+2$, with derivative $f'(x) = 4x^3-9x^2$




```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```

Subtracting a fraction of the gradient moves you towards the minimum!

gradient descent 用于 min loss

因此之前将我们的 MLE 求最大值转化为 loss，求最小值的优化

减去一小部分 gradient，来到达 minimum

首先计算 x 的 gradient 值(倒数，然后带入 x 点)，然后对它取负数，然后乘以 lr，来更新参数！！！

In matrix notation for all parameters:

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

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

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

Stochastic Gradient Descent

- But Corpus may have 40B tokens and windows
- You would wait a very long time before making a single update

- Very bad idea for pretty much all neural nets!
- Instead: We will update parameters after each window t
  → Stochastic gradient descent (SGD)

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

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

一篇文章的 center word 和 windows 很多！！不能一篇文章更新一次

而且 gradient descent 需要全部参数的同时更新！！！需要计算全部的 gradient 后再更新，太慢了

每一次 window 里更新一次 vecror

这样更新更快！！！

论文： a simple but tough-to-beat baseline for sentence embeddings

CS224N Research Highlight

A Simple but Tough-to-beat Baseline for **Sentence Embeddings**

Sanjeev Arora, Yingyu Liang, Tengyu Ma
Princeton University
In submission to ICLR 2017

如何 encode sentence to vector

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \text{Natural language processing is fun.} = \begin{pmatrix} -0.132 \\ 1.129 \\ 0.827 \\ 0.110 \\ -0.527 \\ 0.156 \\ 0.349 \\ -0.286 \end{pmatrix}$$

Sentence embedding

- Compute **sentence similarity** using the inner product:

S1: Mexico wishes to guarantee citizen's safety.
S2: Mexico wishes to avoid more violence.
Score: 4 (/5)

S1: Iranians Vote in Presidential Election.
S2: Keita Wins Mali Presidential Election.
Score: 0.4 (/5)

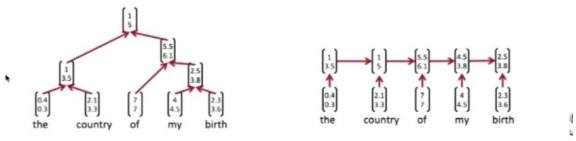
sentence embedding 可以衡量**句子的 similarity**
可以用来做情感分析

From Bag-of-words to Complex Models…

- Bag-of-words (BoW)

v("natural language processing") =
1/3 (v("natural") + v("language") + v("processing"))

- Recurrent neural networks, recursive neural networks, convolutional neural networks..



传统方法：sentence embedding 可以通过词袋模型，**average 每个 word 的 embedding 或更复杂的 RNN**

This paper

- A VERY SIMPLE unsupervised method
  - **weighted Bag-of-words + remove some special direction**
- **Step 1**:

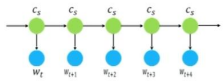
for all sentence s in $\mathcal{S}$ **do**
$v_s \leftarrow \frac{1}{|s|}\sum_{w\in s}\frac{a}{a+p(w)}v_w$
end for

- **Step 2**:

Compute the first principal component u of $\{v_s : s \in \mathcal{S}\}$
for all sentence s in $\mathcal{S}$ **do**
$v_s \leftarrow v_s - uu^\top v_s$
end for

1 给每个 word 乘以一个 **weight**：a constants p(w)是词频
加权平均 word vector
2 减去一些 bias

A Probabilistic Interpretation



$$\Pr[w_t \mid c_s] = \alpha p(w_t) + (1-\alpha)\frac{\exp(v_{w_t}, b_s)}{Z_{\tilde{c}_s}}$$

Smoothing term:
w_t is emitted from background probability (irrelevant to the vector c_s)

w_t is emitted according to correlation with the shifted context vector

$b_s = \beta c_0 + (1-\beta)c_s$

common discourse, often related to syntax

跟 word 出现的频率有关和

Results

sentence similarity

Supervised or not	Results collected from (Wieting et al., 2016) except tfidf-GloVe											Our approach	
	Su.							Un.			Se.	Un.	Se.
Tasks	PP	PP -proj.	DAN	RNN	iRNN	LSTM (no)	LSTM (o.g.)	ST	avg. GloVe	tfidf. GloVe	avg. PSL	GloVe +WR	PSL +WR
STS'12	58.7	**60.0**	56.0	48.1	58.4	51.0	46.4	30.8	52.5	58.7	52.8	56.2	59.5
STS'13	55.8	56.8	54.2	44.7	56.7	45.2	41.5	24.8	42.3	52.1	46.4	56.6	**61.8**
STS'14	70.9	71.3	69.5	57.7	70.9	59.8	51.5	31.4	54.2	63.8	59.5	68.5	**73.5**
STS'15	75.8	74.8	72.7	57.2	75.6	63.9	56.0	31.0	52.7	60.6	60.0	71.7	**76.3**
SICK'14	71.6	71.6	70.7	61.2	71.2	63.9	59.0	49.8	65.9	69.4	66.4	72.2	**72.9**
Twitter'15	52.9	52.8	**53.7**	45.1	52.9	47.6	36.1	24.7	30.3	33.8	36.3	48.0	49.0

sentence classification

	PP	DAN	RNN	LSTM (no)	LSTM (o.g.)	skip-thought	Ours
similarity (SICK)	84.9	85.96	73.13	85.45	83.41	85.8	**86.03**
entailment (SICK)	83.1	84.5	76.4	83.2	82.0	-	**84.6**
sentiment (SST)	79.4	83.4	86.5	86.6	**89.2**	-	82.2

03 Advanced Word Vector Representations

A Deeper Look at Word Vectors

- Finish word2vec
- What does word2vec capture?
- How could we capture this essence more effectively?
- How can we analyze word vectors?

Stochastic gradients with word vectors!

- But in each window, we only have at most $2m + 1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

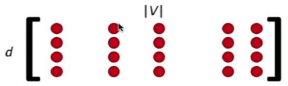
将全部 vector，变成一列参数

一个 window 更新一次，只更新 window 出现的 center 和 output 的向量，因此其他 word 的向量都是 0，**是个 sparce gradient vector，容易内存泄漏。**

如果 batch gradient，是需要将整个 text 扫一遍，保证计算了该 text 每个 word 的 gradient(gradient 不为 0)，**才所有 word 的 embedding 同步更新！**

Stochastic gradients with word vectors!

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



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

只更新出现的 word vector

或者 hash map

Skip-gram Model based on Negative-sampling

- The normalization factor is too computationally expensive.

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

- Hence, in Assignment 1, you will implement the skip-gram model with negative sampling

每次分母的计算量很大，我们可以 negative sampling，但其实不需要每个 target word 都需要对其全部的 negative word 进行训练，只需对除了 context 剩余的全部 word 进行进行 sample

- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus a couple of noise pairs (the center word paired with a random word)

训练一个二分类器，给出 p(o|c)的概率，如果 o 是 c 的 context 概率大，如果不是概率小

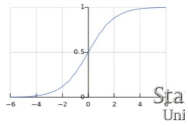
需要正负样本，负样本随机从非 context 里取

Ass 1: The skip-gram model and negative sampling

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function: $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

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

- The sigmoid function! $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)
- So we maximize the probability of two words co-occurring in first log →



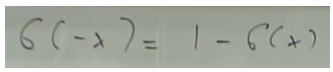
T 代表每个 window　　**J 是 window loss**

左边是**二分类的概率的 log(我们想 maxize)**，右边是 random sample，想 minimize 概率(所以取负就变成 maximize)　两个 term 一起 maximize

概率是一个随机取的权重，因为有些 word 的频率本来就大！！！使得频率低的词也能被选取

每一次：

左边每次只需要一对正样本，右边需要多个负样本(sum)



Ass 1: The skip-gram model and negative sampling

- Slightly clearer notation:

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

- We take k negative samples
- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

- $P(w)=U(w)^{3/4}/Z$, the unigram distribution U(w) raised to the 3/4 power (We provide this function in the starter code).
- The power makes less frequent words be sampled more often

U(w) 代表 每个单词被赋予的一个权重，即 词频，分母 Z 代表所有单词的权重和。

3/4 是保证词频小的 word，sample 也更多(不能根据频率来，the a　of 出现太多了)

Ass 1: The continuous bag of words model

- Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model

- To make assignment slightly easier:

  Implementation of the CBOW model is not required (you can do it for a couple of bonus points!), but you do have to do the written problem on CBOW.

用 vector of **surrounding** word to predict target word

最终会将 similar word 的空间距离变得更近！！！

每个窗口 capture concurrence of woord

Glove

- Why not capture cooccurrence **counts** directly?

I love deep learning
每次窗口遇到 deep learning，都会更新 context vector，可能会出现很多次，**效率不高**
可以 go through text，根据出现的**频率**来更新参数向量(例如 I love 无论出现多少次，都是一个 positive example，不需要学习很多次)

Glove 是 Co-occurence matrix + skip-gram

Co-occurence matrix(**一次性统计全文的 word 的统计特征)**

With a co-occurrence matrix X

- 2 options: windows vs full document
- Window: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information

- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

如果基于 window，concurrence matrix 可以捕捉**句法 POS 和语义信息**
如果基于 document ，concurrence matrix 会给出整个文本的 **topic**

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.

window length=1 center word 前后各一个 word
加上 txt 有三句话 如上
对每个 center word **统计 co-occurrence matrix**
会有重复，因为 word 是 center word 也是 context word，因此对角线对称

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

每个值代表两个 **word 的相关性！！**

此时 word 变成了**向量**，但有问题

Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

txt 词汇越多，表越大 high dimension，难存储，而且向量 **sparce**，难以 train

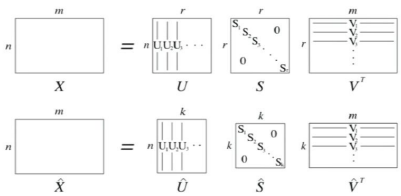
Solution: Low dimensional vectors

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25 – 1000 dimensions, similar to word2vec
- How to reduce the dimensionality?

只留最重要的信息，SVD(PCA)

Method 1: Dimensionality Reduction on X

Singular Value Decomposition of co-occurrence matrix X.



$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

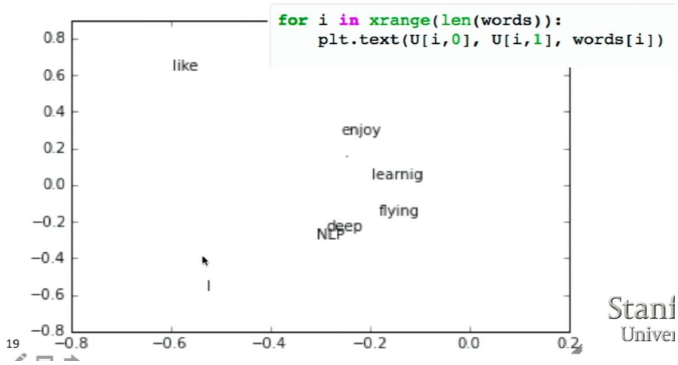
Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values



19

选取 U 最大的两列(**前两列**) plot，就代表每个 word 的 embedding

Hacks to X

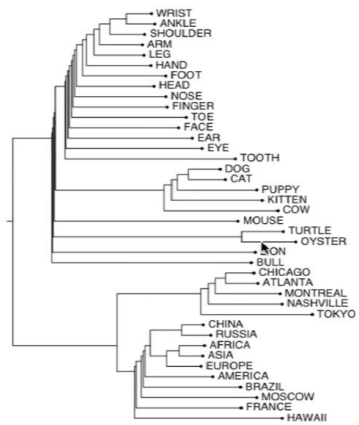
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t~100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- +++

1 **对于 function word 出现频率本身就很高**，会影响最终的 vector(**值会偏大**)(**但本身没什么语义相似性**)，而且**会影响真正语义相似的词**，需要削弱这个频率，**限定上线如 100**

2 离 center word **越近的 word count 值(权重)越大**

3 相关系数 question

Interesting semantic patters emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Problems with SVD

Computational cost scales quadratically for n x m matrix:

$O(mn^2)$ flops (when n<m)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models

1 计算量大

2 有了新 word，需要 rerun 比较麻烦！

count based vs embedding word vector

Count based vs direct prediction

• LSA, HAL (Lund & Burgess), • COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)	• Skip-gram/CBOW (Mikolov et al) • NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)
• Fast training • Efficient usage of statistics • Primarily used to capture word similarity • Disproportionate importance given to large counts	• Scale with corpus size • Inefficient usage of statistics • Generate improved performance on other tasks • Can capture complex patterns beyond word similarity

左边 concurrence based PCA 训练很快，直接统计词频(**用到了 count 统计特征**)

1 只能 capture word similarity，其他的 pattern 没有 catch

2 对于词频高的词，生成的向量值也大，但对于 function 词没有 similarity 意义

右边：训练，每个 window 都要更新参数，慢

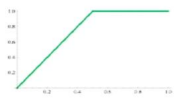
可以 capture complex patterns

GloVe 结合了两点：

Combining the best of both worlds: GloVe

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

- Fast training
- Scalable to huge corpora

$f \sim$ 

- Good performance even with small corpus, and small vectors
- By Pennington, Socher, Manning (2014)

结合在一起：Global Vector Model

P 是 co-occurrence matrix of a text，对每个**矩阵元素**进行遍历
每个元素其实就是 每两个 word 进行的配对！！矩阵的的列名和 index 名就是 **vocabulary**
P12 和 P21 其实就是同一对，当其中一个是 context，另一个是 target 的两种情况。

loss 函数目标：minimize distance of **内积和 log count**
原因: 因为 count 是两个 word 的相关性！！而 word 的向量内积也是相关性，**因此我们希望 count 和内积增长趋势一样，差值是稳定的。 不相似差值就大(就惩罚)，相似差值小(loss 小)。**
f(count)是权重，将频率高的词也能降低权重，**如图 有个上界**

这个模型结合了 co-occurrence matrix 和 skip gram model word vector 模型：
word vector 还是先初始化
然后此时不需要每个 window 就更新一次 vector 参数，而是整个 corperal 为单元一次更新

1 更适合大数据 train 快 2 对于小数据效果也好(因为利用了统计特征 count)

What to do with the two sets of vectors?

- We end up with U and V from all the vectors u and v (in columns)
- Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

$$X_{final} = U + V$$

- One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation*, Pennington et al. (2014)

最终我们得到每个 word 的 2 个 vector，都 capture 了相似性和同时发生的信息
U 和 V 可以相加，的得到最终的 vector
另一个原因：优化时，比起只优化一个参数，不如将一个参数拆分两个，最后合并！！因此需要两个参数

evaluate word vector and adjust hyper-parameter

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!

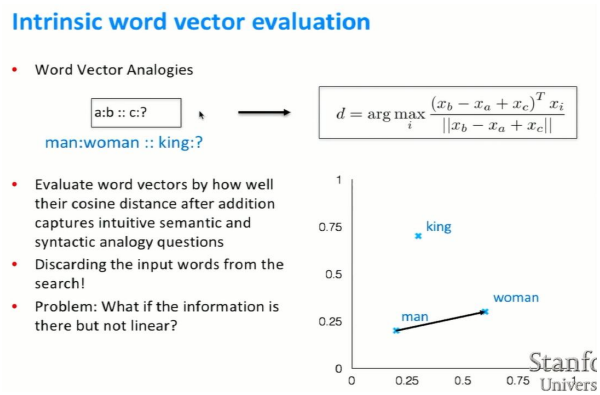
intrinsic: sub task extrinsic:real task

评估 vector 的效果，如果调超参数，需要根据**实际需求！！**
两个方面来评估

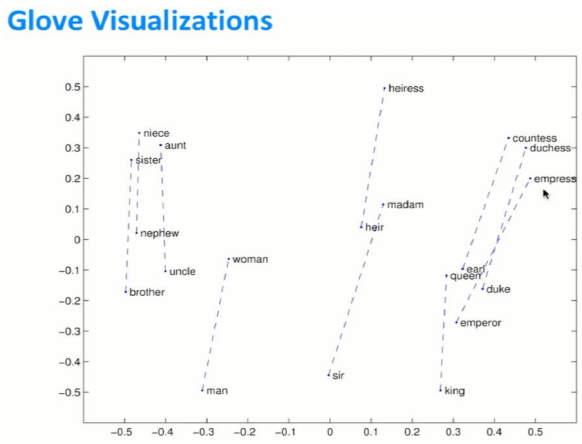
intrinsic

1 analogy

其实就是**玩文字游戏**



用 word 间的**距离来表示 analogy**
xb-xa=x？- xc
x?= xb-xa+xc
看哪个 word 和 x?有**最大的相似度，比上膜是 cos similarity，不受 vector 大小影响**

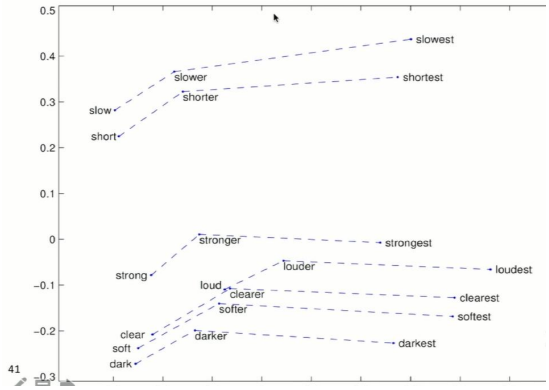


Other fun word2vec analogies

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

nearest token

Glove Visualizations: Superlatives



比较级也适用

Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

problem: different cities may have same name

用于比较的 analogy 数据库，来检验我们的 embedding 的效果，选择效果最好的！！！
通过看 analogy 的准确率，**跟真实的 analogy 比**

Analogy evaluation and hyperparameters

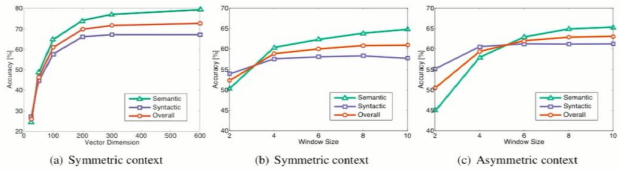
- Very careful analysis: Glove word vectors

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

GloVe 最好
more dimension 并不意味着好
text size 越多，越好对于 DL！！
注意每次只 change 一项来检查模型的是否改进
效果也会受文本内容 topic 的影响！！维基百科比较好

Analogy evaluation and hyperparameters

- Asymmetric context (only words to the left) are not as good



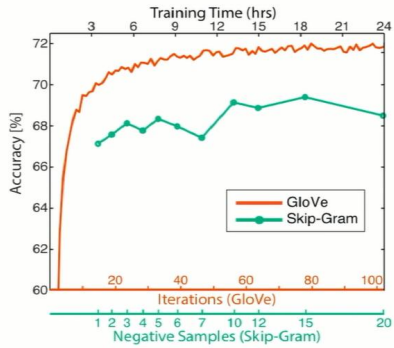
(a) Symmetric context (b) Symmetric context (c) Asymmetric context

- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!
- Window size of 8 around each center word is good for Glove vectors

symmetric context 最好，也就是左右的 window
dimension 有极值 不是越大越好
window size 越大计算时间越长

Analogy evaluation and hyperparameters

- More training time helps



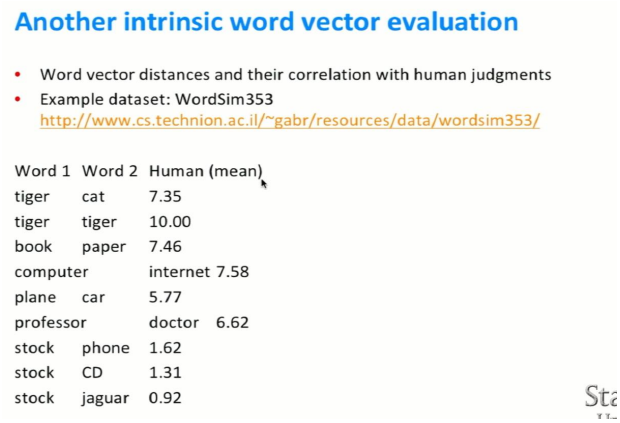
Glove 会更好

Closest words to "Sweden" (cosine similarity)

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

例如 我们得到向量以后，计算 cosine 距离

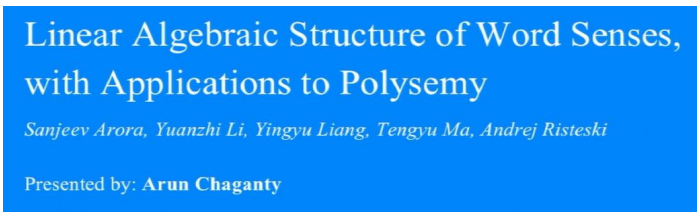
2 correlation with human



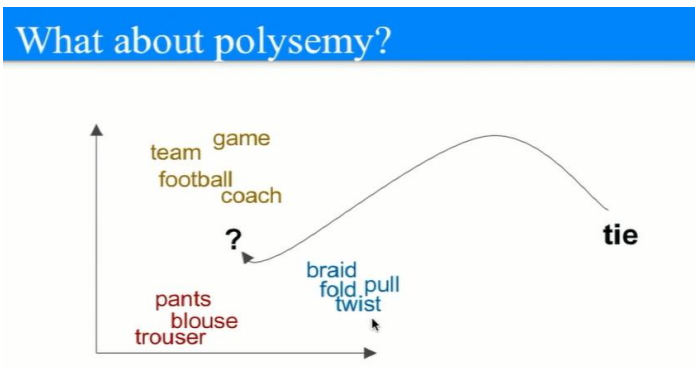
与 human level 进行对比，理想是具有很强的相关性

还有一种是 down stream test 检验 vector 的有效性，也就是应用在模型中，看最终模型的效果，但对于情感分析不适用，因为句子可能**会双重否定**

论 文 Linear algebraic structure of word sense with application to polysemy(一词多意)

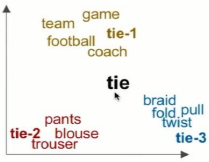


Polysemy： 一词多意



tie 多意词，如何处理？应该靠向那边？

1. Polysemous vectors are superpositioned.



superposition **重叠，叠加**

将 tie 拆成 3 个向量**位置向量（起点原点的向量）**

tie 是 3 个 word 的 combination

2. Senses can be recovered by sparse coding

$$v = \sum_{i=0}^{D} \alpha_i A_i + \eta$$

Context vectors (~2000)

Word vector

Selectors (< 5)

noise

word 的 sense 意义可以被解释，每个意思都有不同的 context 组合

context vector 是所有 word 的 common context 2000

选择 5 个线性组合，得到我们的 target word vector

也就是**每个 word，可以由少量的 sence word vector(sparse code)线性组合得到 tie 的 context 组合：**

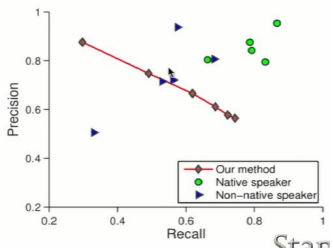
2. Senses can be recovered by sparse coding

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

3. Senses recovered are non-native English level

tie

1. Trousers, blouse, pants
2. Season, teams, winning
3. Computer, mouse, keyboard
4. Bulb, light, flash



接下来需要检验这个效果怎么样，recover mean 得到 sparse coding words

绿色是模型的结果，红色是负样本，随机选取的一组，让人来选择

Summary

Word vectors can capture polysemy!

Word vectors are linear superposition of each sense vector.

Sense/context vectors can be recovered by sparse coding.

The senses recovered are about as good as a non-native English speakers!

04 Word Window Classification and Neural Networks

embedding vector

的好处在于：

Simple single word classification

- What is the major benefit of deep learned word vectors?
  - Ability to also classify words and phrases accurately
    - Countries cluster together → classifying location words should be possible with word vectors
  - Later: Fine tune or learn from scratch for any task and incorporate **more** information. For example:
    - Project sentiment into words to find most positive/negative words in corpus

有了 embedding，我们可以做特定文字的分类，如对城市分类，训练一个 classfication model，**就算我们的 test 不参与 train，也能取得很多好的效果！！！**

word classification sofmax and loss function

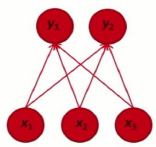
很多情况下 **vector** 是我们 DL 的第一步！！！

The softmax

Logistic regression = Softmax classification on word vector x to obtain probability for class y:

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

$$W \in \mathbb{R}^{C \times d}$$



softmax 和 lr 一样，本身是一个神经网络

计算就是拿到每个 output 的 score，然后归一化！！！

其实就是把向量带入 softmax 函数　sotmax(向量) 得到归一化的向量

softmax 可以作为 word x 的分类，最后得到类别 y 的概率

loss function 交叉熵 learn real distribution

Training with softmax and cross-entropy error

- For each training example {x,y}, our objective is to maximize the probability of the correct class y

- Hence, we minimize the negative log probability of that class:

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

对于 1 个 example，我们希望他正确类别概率最大，等同于 **minimize –log p**

Background: Why "Cross entropy" error

- Assuming a ground truth (or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: p = [0,…,0,1,0,…0] and our computed probability is q, then the cross entropy is:

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

- **Because of one-hot p, the only term left is the negative log probability of the true class**

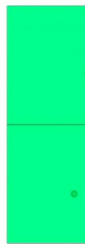
减少交叉熵，减少 KL diverge
以上是对于一个 example loss 的定义

多个 exmaple：
cost function，所有 example 相加

Classification over a full dataset

- Cross entropy loss function over full dataset $\{x_i,y_i\}_{i=1}^{N}$

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



- Instead of

$$f_y = f_y(x) = W_{y\cdot}x = \sum_{j=1}^{d} W_{yj}x_j$$

- We will write *f* in matrix notation: $f = Wx$
- We can still index elements of it based on class

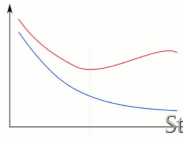
fy 就是求第 y 个输出的 softmax 值=Wyx
f=Wx

Classification: Regularization!

- Really full loss function over any dataset includes **regularization** over all parameters θ:

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

- Regularization will prevent overfitting when we have a lot of features (or later a very powerful/deep model)
  - x-axis: more powerful model or more training iterations
  - Blue: training error, red: test error



x 轴可以是时间，iteration 或者是模型的复杂度

Classification difference with word vectors

- Common in deep learning:
  - Learn both W and word vectors x

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

Very large!

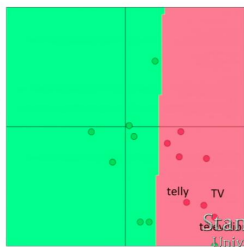
deep learning 不仅要 learn 网络**结构参数 w** 还需要 learn **word vector V*d，参数会很大**，容易过拟合

retain word vector loose generation

会使得这些 vector 偏离原来的 vector

Losing generalization by re-training word vectors

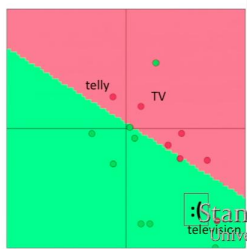
- Setting: Training logistic regression for movie review sentiment single words and in the training data we have
  - "TV" and "telly"
- In the testing data we have
  - "television"
- Originally they were all similar (from pre-training word vectors)
- What happens when we train the word vectors?



最初得到的三个 word glove 是很接近的
然后用 TV telly train 情感分析分类 正负，word 作为参数，会改变 word 向量

Losing generalization by re-training word vectors

- What happens when we train the word vectors?
  - Those that are in the training data move around
  - Words from pre-training that do NOT appear in training stay
- Example:
- In training data: "TV" and "telly"
- Only in testing data: "television"



telly TV 会变，不再和离 test 近了

如果对于某个 Task，我们 **train set 太小，不要 train word vector！！** DL 容易过拟合
就直接用 glove，固定 vector，不 train！！！
如果 train set 多，最好去 train(甚至直接不用 glove vector，直接 random 初始化，从头去 train，来适应 TASK)

window classification

Window classification

- Classifying single words is rarely done.
- Interesting problems like ambiguity arise in context!
- Example: auto-antonyms:
  - "To sanction" can mean "to permit" or "to punish."
  - "To seed" can mean "to place seeds" or "to remove seeds."
- Example: ambiguous named entities:
  - Paris → Paris, France vs Paris Hilton
  - Hathaway → Berkshire Hathaway vs Anne Hathaway

对于单个 word 的分类很少，因为 word 可能有 ambiguity
我们更关心**在 context 中 classify！**
因为不同的 context，word 含义不同！！需要考虑 context

Window classification

- Idea: classify a word in its context window of neighboring words.
- For example named entity recognition into 4 classes:
  - Person, location, organization, none
- Many possibilities exist for classifying one word in context, e.g. averaging all the words in a window but that looses position information

对 word 分类转换为对其从一个 context window 分类
可以 average word vector 但是丢失了 position 信息

Window classification

- Train softmax classifier by assigning a label to a center word and concatenating all word vectors surrounding it
- Example: Classify Paris in the context of this sentence with window length 2:

... museums in Paris are amazing ... .

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

- Resulting vector $x_{window} = x \in R^{5d}$, a column vector!

对 center window 分类，需要将 window 的 vector **拼接成一个大的 vector**
然后 train softmax 输出 label

Simplest window classifier: Softmax

- With $x = x_{window}$ we can use the same softmax classifier as before

predicted model output probability

$$\hat{y}_y = p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C}\exp(W_c.x)}$$

same

- With cross entropy error as before:

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

- But how do you update the word vectors?

如何更新 **concatenated word vector x**？也是更新，更新完再分开！word vector 也是改变了还是求 derivative d

Define:

- $\hat{y}$: softmax probability output vector (see previous slide)
- t: target probability distribution (all 0's except at ground truth index of class y, where it's 1)
- $f = f(x) = Wx \in \mathbb{R}^C$ and f_c = c'th element of the f vector

Updating concatenated word vectors

- Tip 1: Carefully define your variables and keep track of their dimensionality!

$f = f(x) = Wx \in \mathbb{R}^C$
$\hat{y}$ t $W \in \mathbb{R}^{C\times 5d}$

- Tip 2: Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y = f(g(x))$, then:
- Simple example: $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$

1 需要定义好维度：

2 chain rule

$$\frac{dy}{dx} = \frac{d}{dx}5(x^3+7)^4$$

$$y = f(u) = 5u^4 \qquad u = g(x) = x^3 + 7$$

$$\frac{dy}{du} = 20u^3 \qquad \frac{du}{dx} = 3x^2$$

$$\frac{dy}{dx} = 20(x^3+7)^3 \cdot 3x^2$$

Updating concatenated word vectors

$f = f(x) = Wx \in \mathbb{R}^C$
$\hat{y}$ t $W \in \mathbb{R}^{C\times 5d}$

- Tip 2 continued: **Know thy chain rule**
- Don't forget which variables depend on what and that x appears inside all elements of f's

$$\frac{\partial}{\partial x} - \log softmax(f_y(x)) = \sum_{c=1}^{C} -\frac{\partial \log softmax(f_y(x))}{\partial f_c} \cdot \frac{\partial f_c(x)}{\partial x}$$

- Tip 3: For the softmax part of the derivative: First take the derivative wrt f_c when c=y (the correct class), then take derivative wrt f_c when c≠ y (all the incorrect classes)

softmax 看做一个函数，忽略其具体形式

Updating concatenated word vectors

- Tip 4: When you take derivative wrt one element of f, try to see if you can create a gradient in the end that includes all partial derivatives:

$\hat{y} \quad t$

$f = f(x) = Wx \in \mathbb{R}^C$

$$\frac{\partial}{\partial f} - \log softmax(f_y) = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_y - 1 \\ \vdots \\ \hat{y}_C \end{bmatrix}$$

- Tip 5: To later not go insane & implementation! → results in terms of vector operations and define single index-able vectors:

$$\frac{\partial}{\partial f} - \log softmax(f_y) = [\hat{y} - t] = \delta$$

Stan

Updating concatenated word vectors

- Tip 6: When you start with the chain rule, first use explicit sums and look at partial derivatives of e.g. x_i or W_{ij}

$\hat{y} \quad t$

$f = f(x) = Wx \in \mathbb{R}^C$

$$\sum_{c=1}^{C} - \frac{\partial \log softmax(f_y(x))}{\partial f_c} \cdot \frac{\partial f_c(x)}{\partial x} = \sum_{c=1}^{C} \delta_c W_{c\cdot}^T$$

- Tip 7: To clean it up for even more complex functions later: Know dimensionality of variables &simplify into matrix notation

$$\frac{\partial}{\partial x} - \log p(y|x) = \sum_{c=1}^{C} \delta_c W_{c\cdot}^T = W^T \delta$$

- Tip 8: Write this out in full sums if it's not clear!

Stan

以上只是求一个 example 的 x 的 gradient，用 gradient 来**更新整个 window x vector！！**

For example, the model can learn that seeing x_{in} as the word just before the center word is indicative for the center word to be a location

context 里如果含有 in，就会更可能预测为 location(位置的类别)

Similar steps, write down partial wrt W_{ij} first!

Then we have full

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

W 和 x 需要一起更新！！！！

能用 matrix 操作就不用 loop

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

DL 比 sofmax 的优势

1 深度学习，学习更复杂的 function 和 boundary

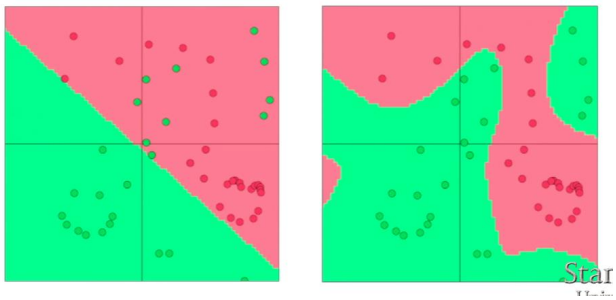
Softmax (= logistic regression) alone not very powerful

- Softmax only gives linear decision boundaries in the original space.
- With little data that can be a good regularizer
- With more data it is very limiting!

softmax 和 lr 在原 space 里都只是 **linear boundary！！**

Neural Nets for the Win!

- Neural networks can learn much more complex functions and nonlinear decision boundaries!



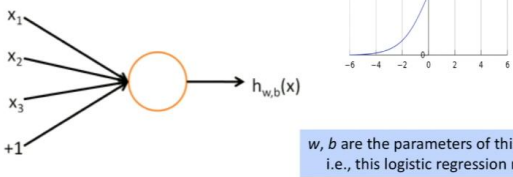
所以要用深度学习，**更复杂的 function 和 boundary**

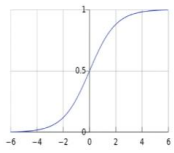
A neuron is essentially a binary logistic regression unit

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

b: We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term

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





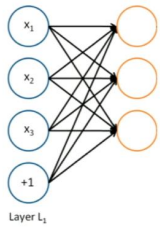
w, b are the parameters of this neuron i.e., this logistic regression model

b 是 class prior 也是参数

2 自动学习 feature

A neural network
= running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …

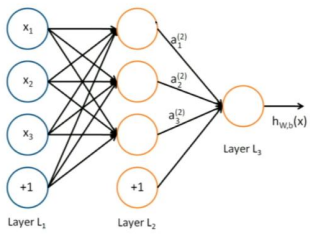


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

many lr at the same time(同一个 input)
如果多层的话，但不需要提前指定每一层 out 的任务，让自己决定(自己学习特征 feature)，并且作为下一层 lr 的 input

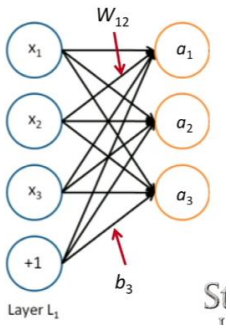
A neural network
= running several logistic regressions at the same time

… which we can feed into another logistic regression function



It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

loss 控制着这些中间变量的值，决定了会学到什么 feature



bias 就是 input 始终为 1

A more powerful, neural net window classifier

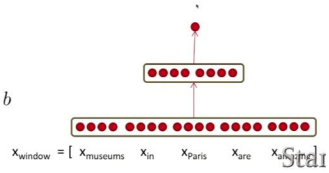
- Revisiting
- $X_{window} = [\ x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing}\ ]$
- Assume we want to classify whether the center word is a location or not

Summary: Feed-forward Computation

Computing a window's score with a 3-layer neural net: *s = score*(museums in Paris are amazing)

$$s = U^T f(Wx + b) \qquad x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$$

$$\begin{aligned} s &= U^T a \\ a &= f(z) \\ z &= Wx + b \end{aligned}$$



我们想给每个 window 一个 score，如果 center word 是 location，希望输出 score high

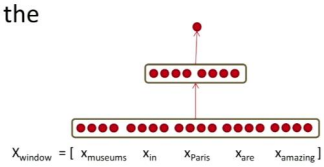
concate vector---> hidden layer---->score

3 学习到了 input 的 interaction 交互

Main intuition for extra layer

The layer learns non-linear interactions between the input word vectors.

Example:
only if "*museums*" is first vector should it matter that "*in*" is in the second position



hidden layer 学到了 window word 的交互，只有 in 前面有博物馆，in 才有作用

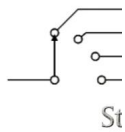
max-margin loss (word 顺序信息也包含了)

max margin loss

The max-margin loss

- *s* = score(museums in Paris are amazing)
- s_c = score(Not all museums in Paris)
- Idea for training objective: make score of true window larger and corrupt window's score lower (until they're good enough): minimize

$$J = \max(0, 1 - s + s_c)$$



- This is continuous --> we can use SGD

两句话的 center word 分别是 Pairs，museum. Pair 是 position，museum 不是

因此希望正确的 sore 大，不正确的 score 小

target: 不仅要求 correct score - wrong score>0. 而且让两者**差距大**

target **转换成 loss:**

wrong score- correct score<0 就不惩罚 0,

wrong score-correct score>0 惩罚 wrong score-correct score

设计为：max(0, wrong score-correct score)

Max-margin Objective function

- Objective for a single window:

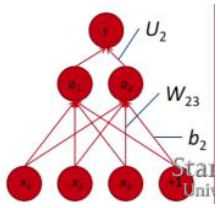
$$J = \max(0, 1 - s + s_c)$$

- Each window with a location at its center should have a score +1 higher than any window without a location at its center
- xxx |← 1 →| ooo
- For full objective function: Sample several corrupt windows per true one. Sum over all training windows.

对于每个 true window，需要 sample 几个 false windows，然后 sum J

就是该 window 的 loss function，每个 ture example 对应多个 negative examples(比如 1:5)，然后多个 window 加总 loss

Train with backpropagation



先对 U 求导：注意 U 是向量！！

Training with Backpropagation

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

Assuming cost *J* is > 0,
compute the derivatives of *s* and s_c wrt all the involved variables: *U, W, b, x*

$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a \qquad \frac{\partial s}{\partial U} = a$$

a=f(x)　ds/dU=a　其实就是 activation 值

又：

J 要么是 0，要么是 1-s+sc 合并在一起得到总的 J

J 导数要么是 0，要么是 1-s+sc 的导数

Putting all gradients together:

- Remember: Full objective function for each window was:

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

- For example: gradient for U:

$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\left(-f(Wx + b) + f(Wx_c + b)\right)$$
$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\left(-a + a_c\right)$$

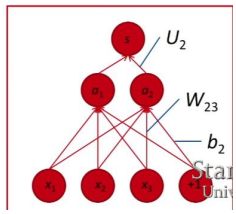
2 对 W 求导：

Training with Backpropagation

- Let's consider the derivative of a single weight W_{ij}

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W}U^T a = \frac{\partial}{\partial W}U^T f(z) = \frac{\partial}{\partial W}U^T f(Wx + b)$$

- This only appears inside a_i
- For example: W_{23} is only used to compute a_2



50

约定 Wij　i 和 j 是层倒过来的　如上图 **2*4 矩阵　i=2　j=4**

每个 Wij 值只用来计算一个 ai

而每个 ai 需要 Wi 向量 来计算

每个 ai 一个 bi

ai=f(zi)　zi=Wi x+bi

Training with Backpropagation

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W}U^T a = \frac{\partial}{\partial W}U^T f(z) = \frac{\partial}{\partial W}U^T f(Wx + b)$$

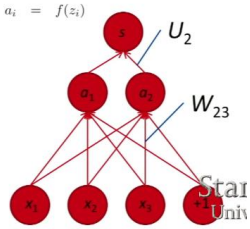
Derivative of weight W_{ij}:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

$$\frac{\partial}{\partial W_{ij}}U^T a \rightarrow \frac{\partial}{\partial W_{ij}}U_i a_i$$

$$\begin{aligned} U_i\frac{\partial}{\partial W_{ij}}a_i &= U_i\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} \\ &= U_i\frac{\partial f(z_i)}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i)\frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i)\frac{\partial W_i x + b_i}{\partial W_{ij}} \end{aligned}$$

$$\begin{aligned} z_i &= W_i x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



51

右边 Wi *x 是向量内积，而 Wij 只是其中一部分(x 向量的一个元素 xj)

Training with Backpropagation

Derivative of single weight W_{ij}:

$$z_i = W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i \qquad a_i = f(z_i)$$

$$U_i \frac{\partial}{\partial W_{ij}} a_i = U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial W_{ij}}$$

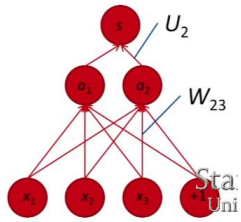
$$= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k$$

$$= \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

$$= \delta_i \; x_j$$

Local error signal / Local input signal

where $f'(z) = f(z)(1 - f(z))$ for logistic f



因此 ds/dWij 的 derivative 分两部分：

1 左边 error 只与 i 有关 也就是从 ai 传来的 error(error:从高层 activation 局部 derivative 乘以高层的权重)

2 右边 x 只与 j 有关 与低层 input 有关 (其实就是把 ai error 的按照 x 值的分配给 Wij)

我们希望激活函数的计算效率越高越好！！！sigmoid 刚好满足！！

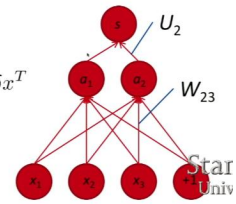
也就是对上层 s 求导，等于这层 a 和下一层 x 的形式！！！

- From single weight W_{ij} to full W:

$$\frac{\partial s}{\partial W_{ij}} = \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

$$z_i = W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i \qquad a_i = f(z_i)$$

- We want all combinations of $i = 1, 2$ and $j = 1, 2, 3$ → ?
- Solution: Outer product: $\frac{\partial s}{\partial W} = \delta x^T$ where $\delta \in \mathbb{R}^{2\times 1}$ is the "responsibility" or error signal coming from each activation a



对全部 W 进行求导，矩阵形式

其实是将 a1 a2 **error** 向量(**每个 a 一个 error**)传递给 x 向量 2*4

两个向量的外积

也就是将 error 通过 x 值比例分配给了 w

- For biases b, we get:

$$U_i \frac{\partial}{\partial b_i} a_i$$

$$= U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial b_i}$$

$$= \delta_i$$

右边 x 始终是 1，因此也是 1

That's almost backpropagation

It's taking derivatives and using the chain rule

Remaining trick: we can **re-use** derivatives computed for higher layers in computing derivatives for lower layers!

3 最后对 X，window **word vector 求导：单个元素的导数**

Training with Backpropagation

- Take derivative of score with respect to single element of word vector
- Now, we cannot just take into consideration one a_i because each x_j is connected to all the neurons above and hence x_j influences the overall score through all of these, hence:

$$\begin{aligned}\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i}\frac{\partial a_i}{\partial x_j} \\ &= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i}\frac{\partial a_i}{\partial x_j} \\ &= \sum_{i=1}^{2} U_i \frac{\partial f(W_{i\cdot}x+b)}{\partial x_j} \\ &= \sum_{i=1}^{2} \underbrace{U_i f'(W_{i\cdot}x+b)}\frac{\partial W_{i\cdot}x}{\partial x_j} \\ &= \sum_{i=1}^{2} \delta_i W_{ij} \\ &= W_{\cdot j}^T \delta\end{aligned}$$

Re-used part of previous derivative

56

每一个 word Xi 都 involved all the 中间变量！不同于 w 和 b(只参与一个 activation)

此时每个 a 都使用了全部的 x 向量，因此需要相加 s1 和 s2 的 gradient

其实 x 和 w 是同等地位，计算 gradient 一样，x 的 gradient 是 a 的 error 通过 w 的分配！

- With $\frac{\partial s}{\partial x_j} = W_{\cdot j}^T \delta$, what is the full gradient? →

$$\frac{\partial s}{\partial x} = W^T \delta$$

整个 window word vector 向量的 gradient

- Observations: The error message δ that arrives at a hidden layer has the same dimensionality as that hidden layer

Putting all gradients together:

- Remember: Full objective function for each window was:

$$J = \max(0, 1 - s + s_c)$$

$$\begin{aligned}s &= U^T f(Wx+b) \\ s_c &= U^T f(Wx_c+b)\end{aligned}$$

- For example: gradient for U:

$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\left(-f(Wx+b) + f(Wx_c+b)\right)$$

$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\left(-a + a_c\right)$$

此时只对 U 求导 example，对其他参数求导也是这样

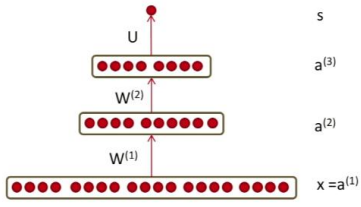
05 Backpropagation and Project Advice

可以自己设计 implement new model！！

Two layer neural nets and full backprop

- Let's look at a 2 layer neural network
- Same window definition for x
- Same scoring function
- 2 hidden layers (carefully define superscripts now!)

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$



Sta

上括号代表层

每个层激活函数都**可以不同类型** sigmoid relu

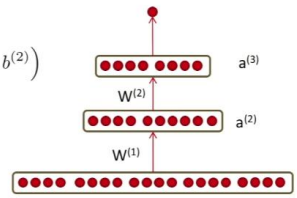
如果我们给每个 x 已经每一层的 a 都增加一个 1 值，就可以把 b 融入在 W

首先对 W(2)求导：

Two layer neural nets and full backprop

- Fully written out as one function:

$$
\begin{aligned}
s &= U^T f\left(W^{(2)} f\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right) \\
&= U^T f\left(W^{(2)}a^{(2)} + b^{(2)}\right) \\
&= U^T a^{(3)}
\end{aligned}
$$



- Same derivation as before for W(2) (now sitting on a(2))

$$
\frac{\partial s}{\partial W_{ij}} = \underbrace{U_i f'(z_i)}_{\delta_i} x_j \qquad \frac{\partial s}{\partial W_{ij}^{(2)}} = \underbrace{U_i f'\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} a_j^{(2)}
$$

Star
Univ

此时 error 是通过 activation(2)分配(等同于 input)

Two layer neural nets and full backprop

- Same derivation as last lecture for top W(2):

$$
\frac{\partial s}{\partial W_{ij}^{(2)}} = \underbrace{U_i f'\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} a_j^{(2)}
$$

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

- In matrix notation: $\frac{\partial s}{\partial W^{(2)}} = \delta^{(3)} a^{(2)T}$

where $\delta^{(3)} = U \circ f'\left(z^{(3)}\right)$ and $\circ$ is the element-wise product also called Hadamard product ($\otimes, \odot$)

- Last missing piece for understanding general backprop: $\frac{\partial s}{\partial W^{(}}$

Stan
Univ

U 是向量！！ error 是 U 向量和 f'向量的 elementwise 相乘

对于 W(1)导数，可以当做 word vector x

Two layer neural nets and full backprop

- Last missing piece: $\frac{\partial s}{\partial W^{(1)}}$
- What's the bottom layer's error message $\delta^{(2)}$?
- Similar derivation to single layer model
- We already derived ${W^{(2)}}^T\delta^{(3)}$ as the next lower update in a model where the next lower layer were just the word vectors
- But now, we'll need to apply the chain rule again: $f'(z^{(2)})$

$$\begin{aligned} x &= z^{(1)} = a^{(1)} \\ z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(3)} &= f\left(z^{(3)}\right) \\ s &= U^T a^{(3)} \end{aligned}$$

Star

error 也被成为上一层的 local derivative

Two layer neural nets and full backprop

- Chain rule for: $s = U^T f\left(W^{(2)} f\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)$
- Get intuition by deriving $\frac{\partial s}{\partial W^{(1)}}$ **as if it was a scalar**
- Intuitively, we have to sum over all the nodes coming into layer
- Putting it all together: $\delta^{(2)} = \left({W^{(2)}}^T\delta^{(3)}\right) \circ f'\left(z^{(2)}\right)$

计算 W 的导数，发现一部分是上一层计算的 error

Two layer neural nets and full backprop

- Final derivative: $\frac{\partial s}{\partial W^{(1)}} = \delta^{(2)}x^T$
- In general for any matrix $W^{(l)}$ at internal layer *l* and any error with regularization E_R all backprop in standard multilayer neural networks boils down to 2 equations:

$$\begin{aligned} x &= z^{(1)} = a^{(1)} \\ z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(3)} &= f\left(z^{(3)}\right) \\ s &= U^T a^{(3)} \end{aligned}$$

$$\delta^{(l)} = \left((W^{(l)})^T\delta^{(l+1)}\right) \circ f'(z^{(l)}),$$

$$\frac{\partial}{\partial W^{(l)}}E_R = \delta^{(l+1)}(a^{(l)})^T + \lambda W^{(l)}$$

- Top and bottom layers have simpler δ

左边是通用的 error 计算方法
右边是正则化的部分，就可以更新 w 了

另一种解释：andrew ng
求变量的 gradient，首先要有 input 值和 output 值，根据值来计算变量的 gradient 更新

Taking a step back

Explanation #2 for backprop:
Circuits

These examples are from CS231n:
http://cs231n.github.io/optimization-2/

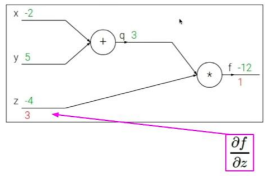
$f(x,y,z) = (x+y)z$

e.g. x = -2, y = 5, z = -4

$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



从计算图的角度，先把过程拆分成每个中间函数(gate)　loss= f=q*z
，然后运用 chain rule 求出全部中间变量的全局 gradient，最后得到 input 的全局 gradient

最右边 df/df=1
因为计算出的 score f 没有用于计算其他 loss，而是一直是本身，或者理解为 loss=f
因此 dl/df=1
f=(x+y)z　df/dz=(x+y)
dl/dz=dl/df *df/dz=(x+y)

df/dq=z　dl/dq=dl/df*df/dq=z
dl/dx=dl/dq*dq/dx=z*1=z

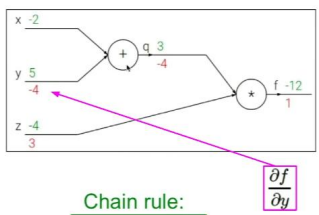
$f(x,y,z) = (x+y)z$

e.g. x = -2, y = 5, z = -4

$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

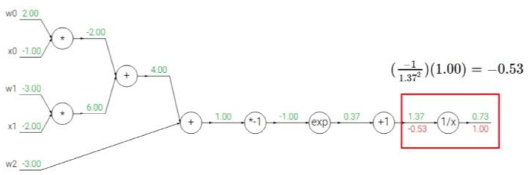
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

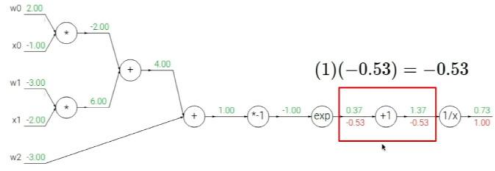
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$	$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

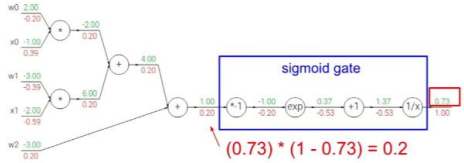
$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

以上每一步计算都分解有点繁琐：我们可以适当的合并成一个函数一个 gate

Combine nodes in the circuit when convenient

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$



将多个步骤合并成 sigmoid 函数(一个 gate)，可以计算这个 gate 的局部 gradient

分解：input-->z-->sigmoid

d sigmoid/dz 可以计算

dz/dw 可以计算

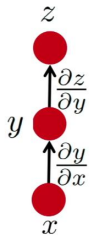
d sigmoid/dw=d sigmoid/dz*dz/dw

Explanation #3 for backprop

The high-level flowgraph

flowgraph

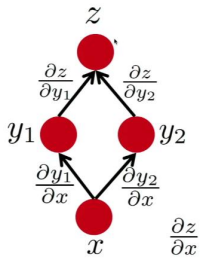
Simple Chain Rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

一个中间变量

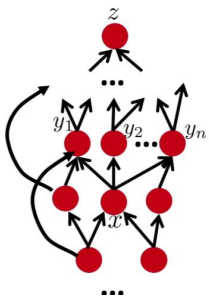
Multiple Paths Chain Rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

可以证明无论 y1 y2 是加还是乘，都满足这个 path 法则
一个变量的 gradient 等于所有前面 path 的和

Chain Rule in Flow Graph



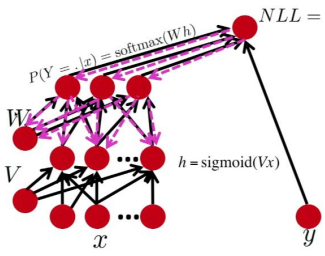
Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

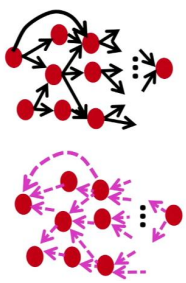
假如很多层，我们只关心前一层的全局*前一层的局部 sum

Back-Prop in Multi-Layer Net



真实场景 V 是参数，计算出的 score 和 y 来计算 loss

Automatic Differentiation



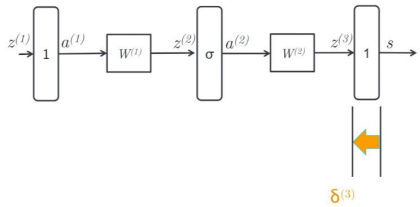
- The gradient computation can be **automatically inferred** from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping

Explanation #4 for backprop

The delta error signals in real neural nets

Visualization of intuition

- Let's say we want $\frac{\partial s}{\partial W^{(1)}} = \delta^{(2)} a^{(1)T}$ with previous layer and f = σ

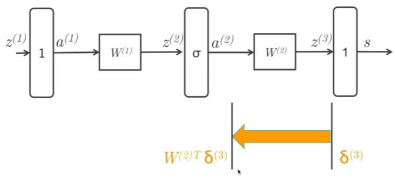


St
u

Gradient w.r.t $W^{(2)} = \delta^{(3)} a^{(2)T}$

先计算 W(2)gradient

Visualization of intuition

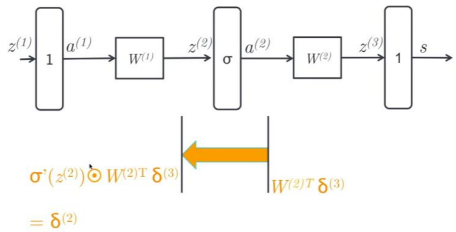


--Reusing the $\delta^{(3)}$ for downstream updates.
--Moving error vector across affine transformation simply requires multiplication with the transpose of forward matrix
--Notice that the dimensions will line up perfectly too!

Sta

通过 d W(2)计算 da(2)

Visualization of intuition

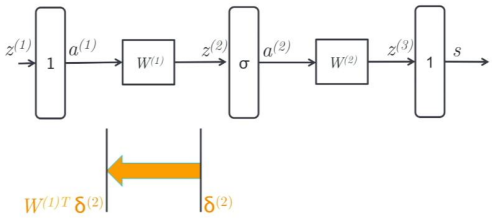


--Moving error vector across point-wise non-linearity requires point-wise multiplication with local gradient of the non-linearity

S

由 da(2)计算 question 非线性

Visualization of intuition



Gradient w.r.t $W^{(1)} = \delta^{(2)} a^{(1)T}$

S

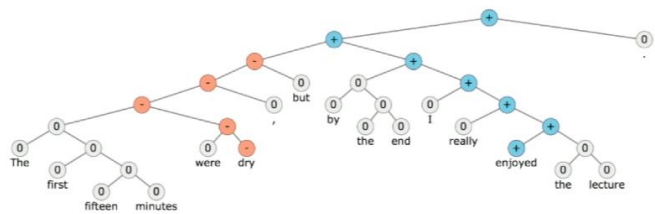
You survived Backprop!

- Congrats!
- You now understand the inner workings of most deep learning models out there
- This was the hardest part of the class
- Everything else from now on is mostly just more matrix multiplications and backprop :)

Bag of Tricks for Efficient Text Classification

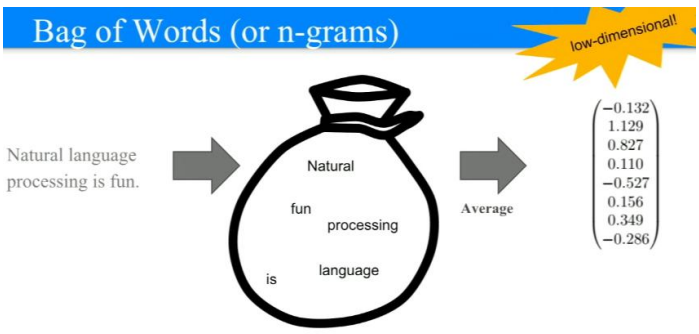
Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov
Facebook AI Research

Text classification



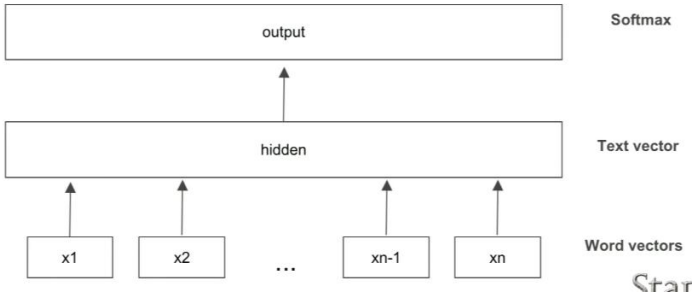
很多种类的问题，sentimatic 只是其中一个问题
可以问各种问题

Bag of Words (or n-grams)

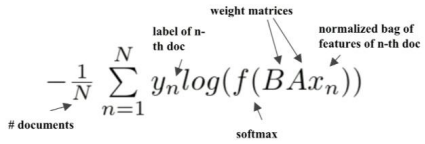


句子向量由全部 word 向量 average　loss order information
n-gram 可以恢复一些 order 信息 question

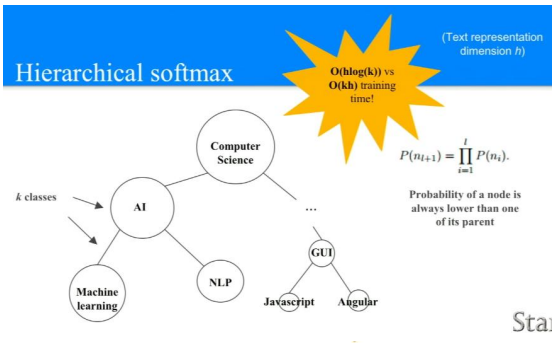
Simple linear model



Learning



分层 softmax：



Sta

	Yahoo		Amazon full		Amazon polarity	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
char-CNN	71.2	1 day	59.5	5 days	94.5	5 days
VDCNN	73.4	2h	63	7h	95.7	7h
fastText	72.3	5s	60.2	9s	94.6	10s

06 Dependency Parsing

Lecture Plan

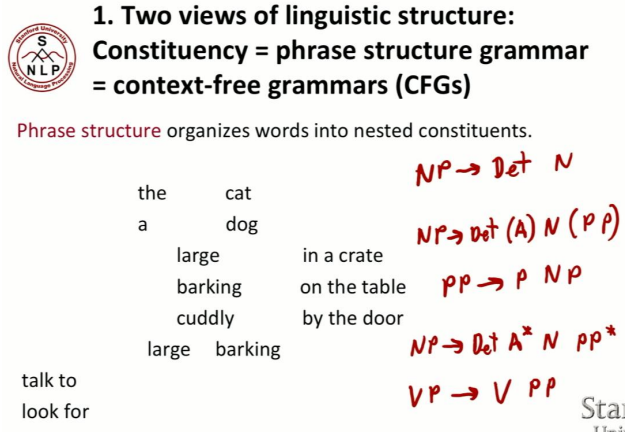
1. Syntactic Structure: Consistency and Dependency
2. Dependency Grammar
3. *Research highlight*
4. Transition-based dependency parsing
5. Neural dependency parsing

Reminders/comments:

Assignment 1 due tonight ☺
Assignment 2 out today ☹
Final project discussion – come meet with us
Sorry that we've been kind of overloaded for office hours

S

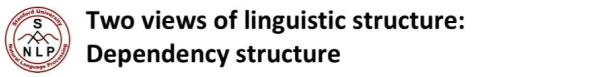
constituency vs dependency



利用固定的语法来解析：

constitute 是 context-free grammar

语法就是为了让 words 变成 nested constituents



- Dependency structure shows which words depend on (modify or are arguments of) which other words.

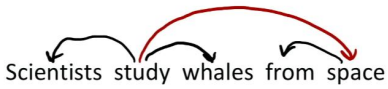
dependency structure

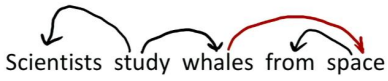
word 之间互相 modify

箭头指向谁，谁是孩子，孩子依赖爸爸

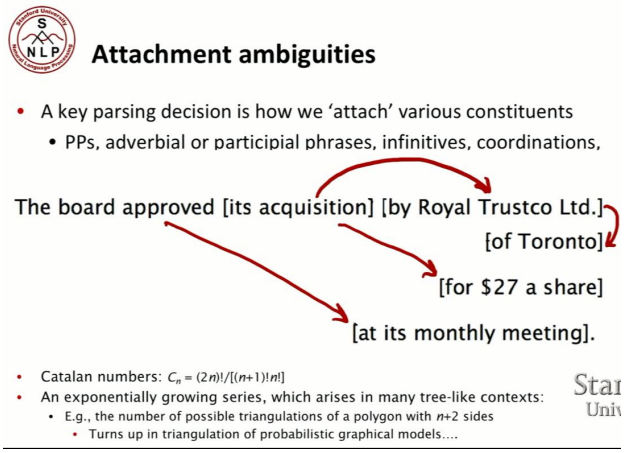
修饰谁，谁是爸爸

Ambiguity: PP attachments



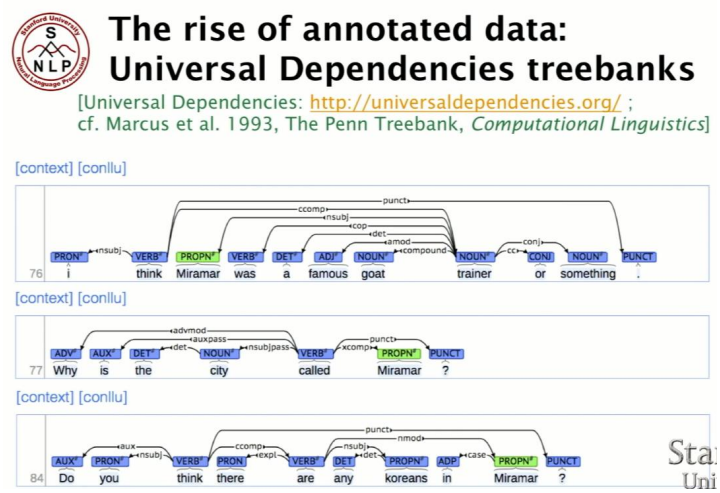


因为句子有歧义，所以会有很多 dependency 结果

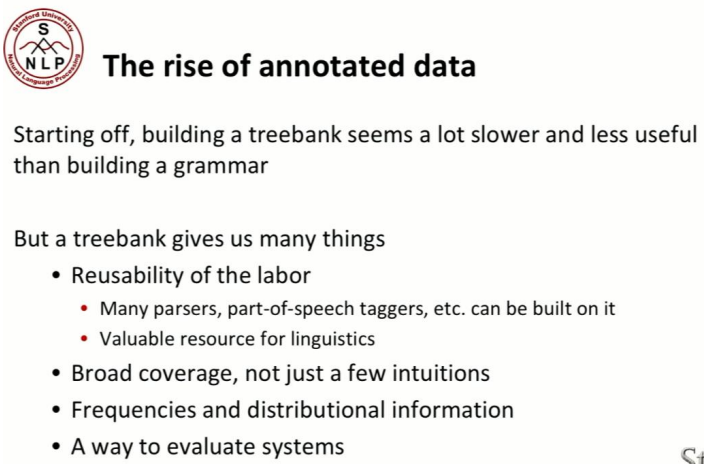


prepositional phrase 介词短语，修饰谁？
修饰名词，动词
可能每句话有很多意思:C 复杂度，会很多解析方式~~~

annotated data Treebank



人工标记的 **dependency structure**，看似效率低，效果会更好



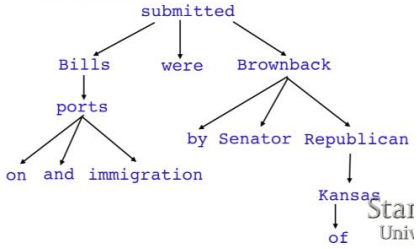
在深度学习的背景下 Treebank 更有效，可以 reuse，也可以作为标准来**评估 parser**

dependency Grammar and structure

Christopher Manning

2. Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies



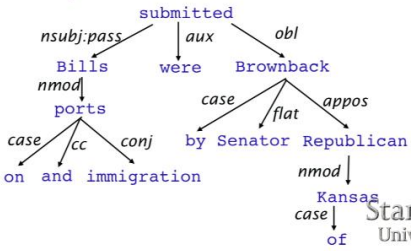
dependency ：句法结构只包含 **词汇的 二分类不对称关系**

Christopher Manning

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)



每个 relation 一个名字，**不同的类型的 dependency**

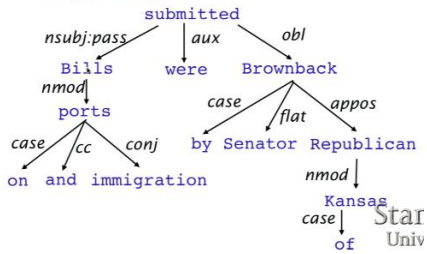
Christopher Manning

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



arrow 由 **head word 以及 dependent word 组成**

single head

Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammars is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947)
- Modern dependency work often linked to work of L. Tesnière (1959)
  - Was dominant approach in "East" (Russia, China, …)
    - Good for free-er word order languages
- Among the earliest kinds of parsers in NLP, even in the US:
  - David Hays, one of the founders of U.S. computational linguistics, built early (first?) dependency parser (Hays 1962)

Dependency Grammar and Dependency Structure

- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent…
- Usually add a fake ROOT so every word is a dependent of precisely 1 other node

添加一个虚拟的 ROOT

Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [discussion → issues] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material
   Dependencies rarely span intervening verbs or punctuation
4. Valency of heads
   How many dependents on which side are usual for a head?

如何进行的？ 根据什么决定的？

1 word 的相似性 2 依赖的距离 dependency 距离 有些很远，插入语

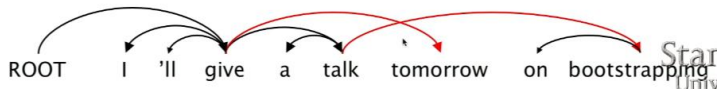
3 很多 dependency 不会跨越 verb，punctuation 标点符号

4 一个 head 多个 denpendent valency(价)多少孩子

Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) is it 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 (non-projective) or not



1 ROOT 只有一个 dependent　2 no cycle

denpendency tree

on bootstrap 修饰 talk，因此 cross tomorrow

有些约定是 on bootstrap 必须紧跟在 talk 后面，也就是不 cross

但是从 dependency tree 不能恢复 word 在原句的**顺序 order**

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
   McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)
3. Constraint Satisfaction
   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
4. "Transition-based parsing" or "deterministic dependency parsing"
   Greedy choice of attachments guided by good machine learning classifiers
   MaltParser (Nivre et al. 2008). Has proven highly effective.

transition-based parsing 是 dominate way

Christopher Manning

4. Greedy transition-based parsing [Nivre 2003]

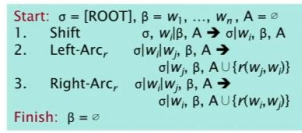
- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser, but the "reduce" actions are specialized to create dependencies with head on left or right

parser greedily do sequence of actions

Arc-standard transition-based parser

(there are other transition schemes ...)

Analysis of "I ate fish"



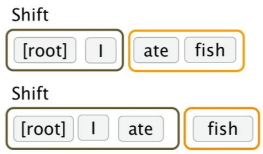
中间是 top

Stack 是灰色框 初始包含 **ROOT symbol** **右边是 top**

buffer 黄色框 是我们要 deal with 的句子 左边是 top

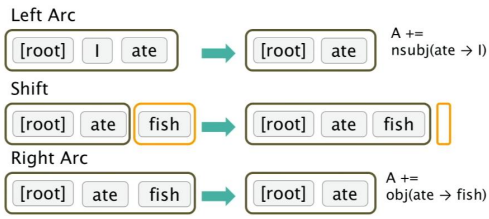
3 种 operation

1shift: **buffer top word I 移动到 Stack top 位置**



每移动一个 word 就需要判断和 stack top word 的 dependency

如果有 dependency **有两种 action：** left-arc <--- right-arc ---> 左右箭头代表 dependency 关系

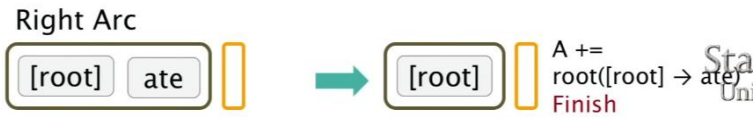


箭头指向的 word 移除(移除 dependent)

注意：每个 arc 都有一个 **dependency 类型 label：** obj subj

一直继续将 buffer 的 word 移至 Stack

当 buffer 为空就结束！



parser 具备的元素：

- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

不同类型的 arc A 集合

3 种不同的 action:

Basic transition-based dependency parser

Start: σ = [ROOT], β = $w_1, \ldots, w_n$, A = ∅

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$

Finish: σ = [w], β = ∅

MaltParser

[Nivre and Hall 2005]

- We have left to explain how we choose the next action
- Each action is predicted by a discriminative classifier (often SVM, can be perceptron, maxent classifier) over each legal move
  - Max of 3 untyped choices; max of |R| × 2 + 1 when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.

得到一个句子，如何选择每一步操作？交给 machine learning classifier

每一个 action 都是一个分类 task

因为 arc 有多种 dependency label

需要多分类器

feature：就是 top of stack　top of buffur

- There is NO search (in the simplest form)
  - But you can profitably do a beam search if you wish (slower but better)
- It provides VERY fast linear time parsing
- The model's accuracy is *slightly* below the best dependency parsers, but
- It provides fast, close to state of the art parsing performance

有点就是快！！！

Feature Representation

binary, sparse dim =10^6 ~ 10^7

0 0 0 1 0 0 1 0 … 0 0 1 0

Feature templates: usually a combination of 1 ~ 3 elements from the configuration.

question

传统做法不是很重要

也就是用不同的组合来代表不同的 feature

假如 stack top word 是 good　00000100000

假如 stack top word 是 bad　00000000100

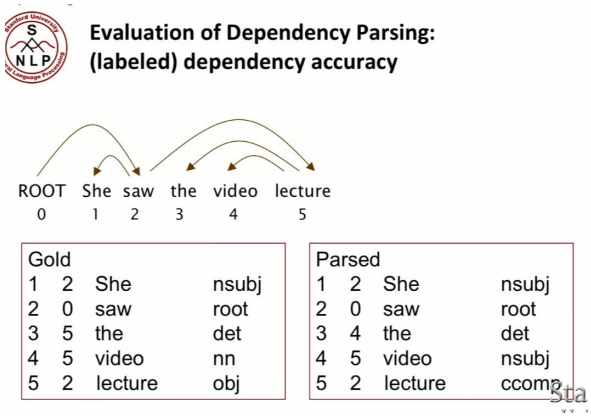
每个向量都是 dict 长度
也可以是 stack 的两个 word 的组合 0000100001000
sparse feature

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

也可以加一些 joint feature：
s2.w stack 的 second word 是 has
s2.t has 的 tag 是动词
s1.w=good 第一个 word 是 good
以上取并作为 feature

评估解析效果：



我们选择几个句子，先手动标记 label 再跟预测的对比！！
label 和 prediction
Gold 是正确的 label 解析(只写出**关联的两个 word**+dependent+ dependency label)
Parsed 是 predict

可以不考虑 label，只考虑两个 word 有没有预测对

Acc = # correct deps / # of deps

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

- Unlabeled attachment score (UAS) = head
- Labeled attachment score (LAS) = head and label

5. Why train a neural dependency parser? Indicator Features Revisited

Problem #1: sparse
Problem #2: incomplete
Problem #3: expensive computation

1 feature sparce
2 incomplete，也就是可能有些情况 coprus 不会出现，feature 不完整
不是全部的 data set

3 feature 计算很慢

Our Approach:
learn a dense and compact feature representation

A neural dependency parser
[Chen and Manning 2014]

English parsing to Stanford Dependencies:
- Unlabeled attachment score (UAS) = head
- Labeled attachment score (LAS) = head and label

Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3***	89.6*	8
C & M 2014	92.0	**89.7**	**654**

更快，准去率也不低

Distributed Representations

- We represent each word as a *d*-dimensional dense vector (i.e., word embedding)
  - Similar words are expected to have close vectors.
- Meanwhile, part-of-speech tags (POS) and dependency labels are also represented as *d*-dimensional vectors.
  - The smaller discrete sets also exhibit many semantical similarities.

question

待读论文看细节

不仅有 word embedding，还增加了语义的信息 POS dep.

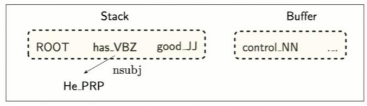
POS 是 word 的词性 如下：

NNS (plural noun) should be close to NN (singular noun).
num (numerical modifier) should be close to amod (adjective modifier).

POS 也是向量，也有相似性

Extracting Tokens and then vector representations from configuration

We extract a set of tokens based on the stack / buffer positions:



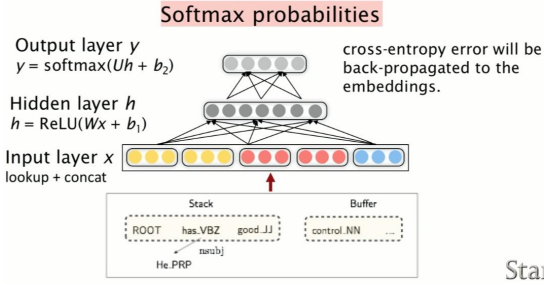
	word	POS	dep.
s_1	good	JJ	∅
s_2	has	VBZ	∅
b_1	control	NN	∅
$lc(s_1)$	∅	∅	∅
$rc(s_1)$	∅	∅	∅
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	∅	∅	∅

We convert them to vector embeddings and concatenate them

根据每个 state，提取特征

将向量 concatenate！！

Model Architecture

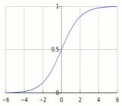


最终得到每个 action

Non-linearities: What's used

logistic ("sigmoid")

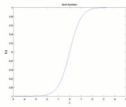
$$f(z) = \frac{1}{1+\exp(-z)}.$$



$$f'(z) = f(z)(1-f(z))$$

tanh

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



$$f'(z) = 1 - f(z)^2$$

tanh is just a rescaled and shifted sigmoid $\tanh(z) = 2\text{logistic}(2z) - 1$

tanh is often used and often performs better for deep nets

- It's output is symmetric around 0

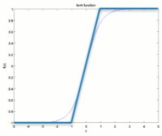
49

-1 到 1

Non-linearities: What's used

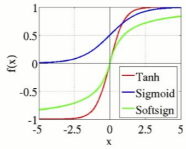
hard tanh

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



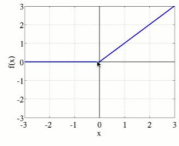
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



linear rectifier (ReLU)

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



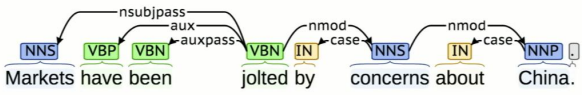
- hard tanh similar but computationally cheaper than tanh and saturates hard.
- [Glorot and Bengio *AISTATS* 2010, 2011] discuss softsign and rectifier
- **Rectified linear unit** is now mega common – transfers linear signal if active

50

first layer 用 linear rectifier(relu) 效果最好

Dependency parsing for sentence structure

Neural networks can accurately determine the structure of sentences, supporting interpretation



Chen and Manning (2014) was the first simple, successful neural dependency parser

The dense representations let it outperform other greedy parsers in both accuracy and speed

St

Improving Distributional Similarity with Lessons Learned from Word Embeddings

Omer Levy, Yoav Goldberg, Ido Dagan

Presented by: **Ajay Sohmshetty**

Word Representation Methods

Count-based distributional models

- SVD (Singular Value Decomposition)
- PPMI (Positive Pointwise Mutual Information)

Neural network-based models

- SGNS (Skip-Gram Negative Sampling) / CBOW
- GloVe

左边是词汇统计 con-currencet matrix

Conventional wisdom:
Neural-network based models > Count-based models

Levy et. al.:
Hyperparameters and system design choices more important, not the embedding algorithms themselves.

Hyperparameters in Skip-Gram

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

# of negative samples

$$P(w) = U(w)^{3/4} / Z$$

Unigram distribution smoothing exponent

→ These can be transferred over to the count-based variants.

#negative sample　　采样权重

Context Distribution Smoothing

$$PMI_\alpha(w,c) = \log \frac{\hat{P}(w,c)}{\hat{P}(w)\hat{P}_\alpha(c)}$$

$$\hat{P}_\alpha(c) = \frac{\#(c)^\alpha}{\sum_c \#(c)^\alpha}$$

也是为了 smooth 词频，最终得到最佳的 alpha 就是对应上面的 3/4

Results

		Word Similarity Tasks						Analogy Tasks	
win	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
2	PPMI	.732	**.699**	.744	.654	.457	.382	.552 / .677	.306 / .535
	SVD	.772	.671	**.777**	.647	**.508**	.425	.554 / .591	.408 / .468
	SGNS	**.789**	.675	.773	**.661**	.449	**.433**	.676 / **.689**	.617 / **.644**
	GloVe	.720	.605	.728	.606	.389	.388	.649 / .666	.540 / .591
5	PPMI	.732	**.706**	.738	**.668**	.442	.360	.518 / .649	.277 / .467
	SVD	.764	.679	**.776**	.639	**.499**	**.416**	.532 / .569	.369 / .424
	SGNS	**.772**	.690	.772	.663	.454	.403	.692 / **.714**	.605 / **.645**
	GloVe	.745	.617	.746	.631	.416	.389	.700 / .712	.541 / .599
10	PPMI	.735	**.701**	.741	.663	.235	.336	.532 / .605	.249 / .353
	SVD	.766	.681	.770	.628	**.312**	.419	.526 / .562	.356 / .406
	SGNS	**.794**	.700	**.775**	**.678**	.281	**.422**	.694 / .710	.520 / **.557**
	GloVe	.746	.643	.754	.616	.266	.375	.702 / **.712**	.463 / .519

检验 vector 效果 similarity task 和 analogy task

Key Takeaways

- This paper challenges the conventional wisdom that neural network-based models are superior to count-based models.
- While model design is important, hyperparameters are also KEY for achieving reasonable results. Don't discount their importance!
- Challenge the status quo!

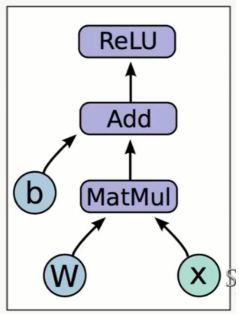
hyper parameter 很重要！！！

7 tensorflow

Big idea: express a numeric computation as a **graph**.

- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes

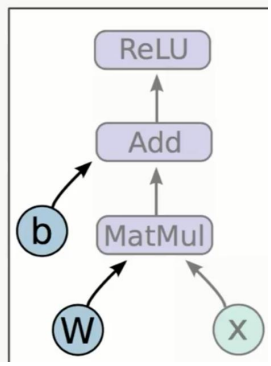
$h = ReLU(Wx + b)$



$h = ReLU(Wx + b)$

Variables are stateful nodes which output their current value. State is retained across multiple executions of a graph

(mostly parameters)



w 和 b 是变量，stateful node 随着计算一直在变

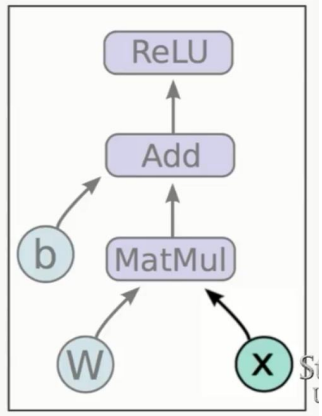
valariable 也是一种 operation，可以 run

存放参数

$$h = ReLU(Wx + b)$$

Placeholders are nodes whose value is fed in at execution time

(inputs, labels, …)



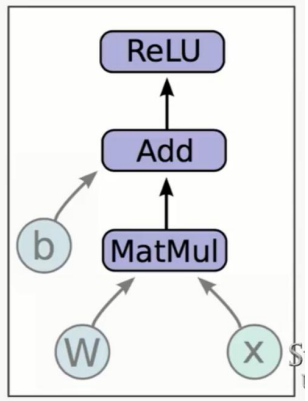
在执行 model 时才给 value，只定义一个类型(内存) symbol 也是 operation

存放 data

$$h = ReLU(Wx + b)$$

Mathematical operations:

MatMul: Multiply two matrix values.
Add: Add elementwise (with broadcasting).
ReLU: Activate with elementwise rectified linear function.




```
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (100, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```

But where is the graph?

New nodes are automatically built into the underlying graph!
tf.get_default_graph().get_operations():

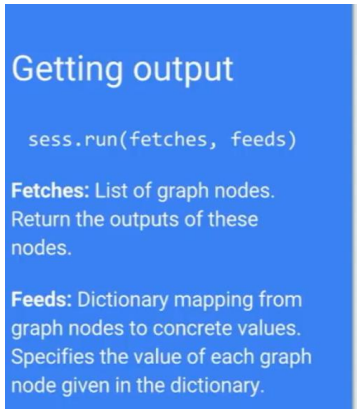
zeros/shape
zeros/Const
zeros
Variable
Variable/Assign
Variable/read
random_uniform/shape
random_uniform/min
random_uniform/max
random_uniform/RandomUniform
random_uniform/sub
random_uniform/mul
random_uniform
Variable_1
Variable_1/Assign
Variable_1/read
Placeholder
MatMul
add
Relu == h

h refers to an op!

built on 默认的 garph

用 Session 来 run graph

Session 决定不同的硬件环境，来执行 graph




```
import numpy as np
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100),
                -1, 1))

x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h, {x: np.random.random(100, 784)})
```

run uinitialize_all_variables()会 run 所有的 variable，进行初始化。可以看出变量也是一个 operation，需要 run，会对全部变量初始化

run(output,dict)　dict 对于 placeholder

就可以执行 graph，然后得到结果

注意：此时 feed_dict 的 key 是 placeholder 引用名！！！然后构建 map

正如：

a=3

{a:[1,2,3]}

如果 train 和自动 propagation

需要将 model 的 output+ label 定义成 loss function

Use **placeholder** for **labels**

Build loss node using labels and **prediction**

```
prediction = tf.nn.softmax(...)  #Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```

tensorflow 会自动将 python 数学运算转换成 tensorflow 运算

如何自动计算 **gradient，完成后向传播**，需要定义**优化算法**

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

tf.train.GradientDescentOptimizer is an **Optimizer** object

tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy)

adds optimization **operation** to computation graph

Training the Model

```
sess.run(train_step, feeds)
```

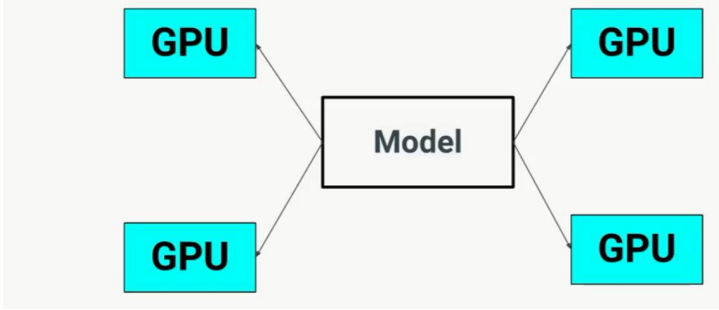
1. Create Session
2. Build training schedule
3. Run train_step

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())

for i in range(1000):
    batch_x, batch_label = data.next_batch()
    sess.run(train_step, feed_dict={x: batch_x,
                                    label: batch_label}
```

batch 是 numpy array 自动转换成 tensorflow tensor

每 run 一次优化算法就更新一次参数！！！需要迭代多次！！！

Variable sharing



分布式

多台机器上跑 batch 来更新参数，需要每台机器都共享参数

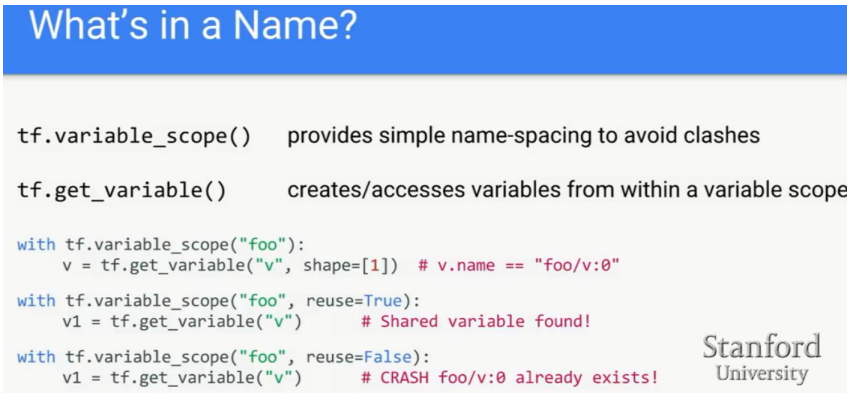
也就是多台机器可以获得同一变量

Variable sharing: naive way

```
variables_dict = {
    "weights": tf.Variable(tf.random_normal([782, 100]),
                           name="weights")
    "biases": tf.Variable(tf.zeros([100]), name="biases")
    }
```

Not good for encapsulation!

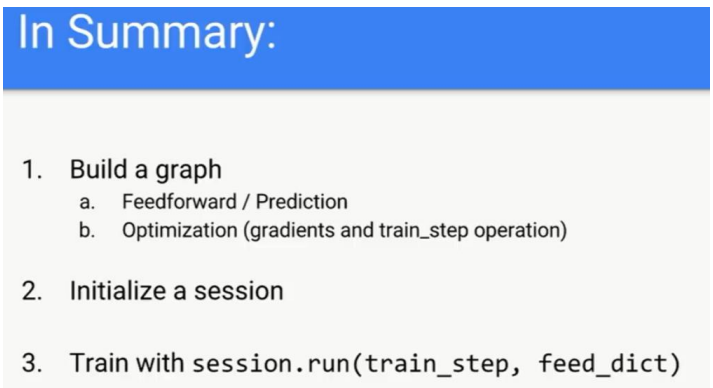
如果只命名的话破坏了封装性，不再是对象的引用，对象引用包含很多信息

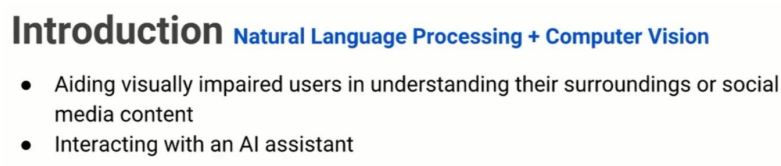


name space

获取或创建一个变量在一个 name space 里

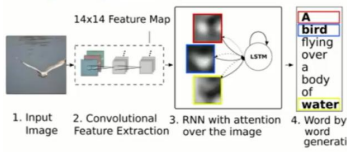
第一行是创建，第二行 reuse 第三行如果想创建新的 variable(reuse=False)，变量已经存在，会报错，变量名冲突





Related Work Image/Video Captioning

Image Captioning




a man is throwing a frisbee in a park


a man riding a wave on top of a surfboard

Visual-Semantic Alignments

Our output: *A cat is playing with a toy.*
Humans: *A Ferret and cat fighting with each other. / A cat and a ferret are playing. / A kitten is playing with a ferret. / A kitten and a ferret are playfully wrestling.*

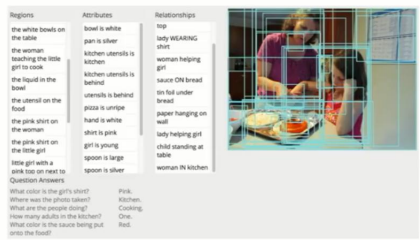
Stanford

左边是图片　　右边是 video

Related Work Visual-Semantic Alignments

Visual-Semantic Alignments

Datasets



Related Work Visual Q&A

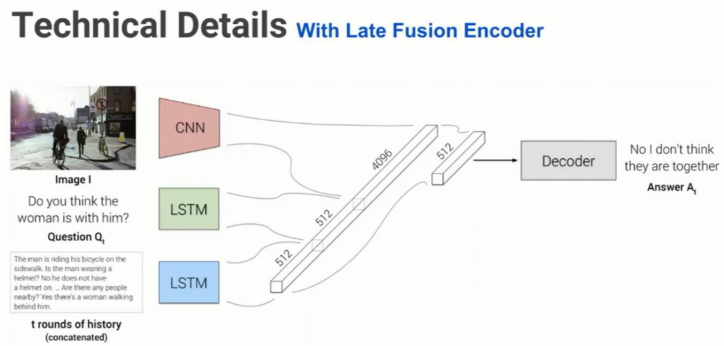
How many pickles are on the plate?	1 1 1	1 1 1
What is the shape of the plate?	circle round round	circle round round

What does the sign say?	stop stop stop	stop stop yield
What shape is this sign?	octagon octagon octagon	diamond octagon round

Contributions

1. Propose a new AI task: Visual Dialog
2. Develop a novel two-person chat data-collection protocol and introduce a new dataset
3. Introduce a family of neural encoder-decoder models for Visual Dialog

Stanford



文字 图片 一起编码


```python
def generate_dataset():
    # data is generated by y = 2x + e
    # where 'e' is sampled from a normal distribution
    x_batch = np.linspace(-1, 1, 101)
    y_batch = 2 * x_batch + np.random.randn(*x_batch.shape) * 0.3
    return x_batch, y_batch

def linear_regression():
    x = tf.placeholder(tf.float32, shape=(None,), name='x')
    y = tf.placeholder(tf.float32, shape=(None,), name='y')

    with tf.variable_scope('lreg') as scope:
        w = tf.Variable(np.random.normal(), name='W')
        y_pred = tf.mul(w, x)

        loss = tf.reduce_mean(tf.square(y_pred - y))
    return x, y, y_pred, loss
```

产生 batch(X,y)的函数

构建计算图得到 score

placeholder 定义多个 exmaple 的维度(None,)是向量,Keras 只需定义一个 example

因为是线性函数因此 w 只有一个变量：y=xw

返回 score 和 loss

而且要返回 placeholder,为了构建 feed_dict

```python
def run():
    x_batch, y_batch = generate_dataset()

    x, y, y_pred, loss = linear_regression()
    optimizer = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

    init = tf.global_variables_initializer()
    with tf.Session() as session:
        session.run(init)

        feed_dict = {x: x_batch, y: y_batch}
        for _ in range(30):
            loss_val, _ = session.run([loss, optimizer], feed_dict)
            print('loss:', loss_val.mean())

        y_pred_batch = session.run(y_pred, {x: x_batch})
```

run 构建优化器 run

Word2Vec

```
Consider the following sentence:
"the first cs224n homework was a lot of fun"

With a window size of 1, we have the dataset:
([the, cs224n], first), ([lot, fun], of) ...

Remember that Skipgram tries to predict each context word from
its target word, and so the task becomes to predict 'the' and
'cs224n' from first, 'lot' and 'fun' from 'of' and so on.

Our dataset now becomes:
(first, the), (first, cs224n), (of, lot), (of, fun) ...
```

用 center word 来 predict context word

dataset 变成如上！！！**第一个 word 是 feature，第二个 word 是 label**

```python
# Let's define some constants first
batch_size = 128
vocabulary_size = 50000
embedding_size = 128  # Dimension of the embedding vector.
num_sampled = 64    # Number of negative examples to sample.

'''
load_data loads the already preprocessed training and val data.

train data is a list of (batch_input, batch_labels) pairs.
val data is a list of all validation inputs.
reverse_dictionary is a python dict from word index to word
'''
train_data, val_data, reverse_dictionary = load_data()
print("Number of training examples:", len(train_data)*batch_size)
print("Number of validation examples:", len(val_data))

def skipgram():
    batch_inputs = tf.placeholder(tf.int32, shape=[batch_size,])
    batch_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
    val_dataset = tf.constant(val_data, dtype=tf.int32)

    with tf.variable_scope("word2vec") as scope:
        embeddings = tf.Variable(tf.random_uniform([vocabulary_size,
                                                    embedding_size],
                                                   -1.0, 1.0))
        batch_embeddings = tf.nn.embedding_lookup(embeddings, batch_inputs)
```

全部的 vocabulary size 50000
embedding size=128 word vector 大小
train data:[[[1,3,6] , [3,6,9]] , [[1,3,6] , [3,6,9]]]
包括 input 和 label，都是对应 word 在字典的编号！！

```
a=[]
a.append(['a','b','c'])
a.append([[1],[2],[3]])
a
```

```
[['a', 'b', 'c'], [[1], [2], [3]]]
```

label 是向量，input 是行

创建 embedding 变量矩阵
得到 batch 的 embedding，通过 word 的字典编号获得 embedding,是 batch of embedding
tf.nn.nec_loss,这个函数直接帮我们完成了负采样及相应的计算

```
weights = tf.Variable(tf.truncated_normal([vocabulary_size,
                                           embedding_size]),
                                           stddev = 1.0/math.sqrt(embedding_size))
biases = tf.Variable(tf.zeros([vocabulary_size])

loss = tf.nn.nce_loss(weights=weights,
                      biases=biases,
                      labels=batch_labels,
                      inputs=batch_inputs,
                      num_sampled=num_sampled,
                      num_classes=vocabulary_size)
```

其实就是 softmax input 是 word embedding
M=(vocab_size , embedding_size) W=(vocab_size , embedding_size)
input 是(embedding_size,1) 乘以 W 得到 vocab_size,也就是预测的 word
question。这里需要再看下 gram model 细节 NCE 如何采样 neg 训练

```
norm = tf.sqrt(tf.reduce_mean(tf.square(embeddings), 1, keep_dims=True)
normalized_embeddings = embeddings/norm

return batch_inputs, batch_labels, normalized_embeddings, loss
```

需要 norm the embedding M

```
val_embeddings = tf.nn.embedding_lookup(normalized_embeddings, val_dataset)
similarity = tf.matmul(val_embeddings, normalized_embeddings, transpose_b=True)
return batch_inputs, batch_labels, normalized_embeddings, loss, similarity
```

还要计算 valid embedding，校验相似度，question 如何矩阵相乘？？
valid 怎么选？

```
def run():
    batch_inputs, batch_labels, normalized_embeddings, loss, similarity = skipgram()
    optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)

    init = tf.global_variables_initializer()
    with tf.Session() as sess:
        sess.run(init)

        average_loss = 0.0
        for step, batch_data in enumerate(train_data):
            inputs, labels = batch_data
            feed_dict = {batch_inputs: inputs, batch_labels: labels}

            _, loss_val = session.run([optimizer, loss], feed_dict)
            average_loss += loss_val

            if step % 1000 == 0:
                if step > 0:
                    average_loss /= 1000
                print('loss at iter', step, ":", average_loss)
                average_loss = 0

            if step % 5000 == 0:
                sim = similarity.eval()
                for i in xrange(len(val_data)):
                    top_k = 8
                    nearest = (-sim[i, :]).argsort()[1:top_k+1]
                    print_closest_words(val_data[i], nearest, reverse_dictionary)

        final_embeddings = normalized_embedding.eval()
INSERT --
```

input,label=batch_data

将 list 拆解

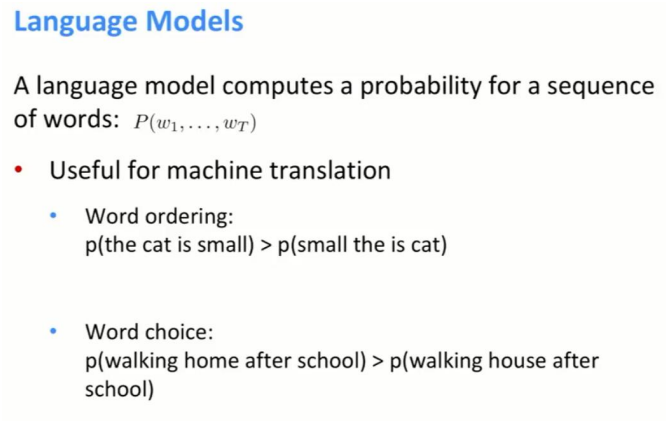
a=['a','b','c']

k,j,e=a

需要计算前 k 个 similarit 的 word

08 - Recurrent Neural Networks and Language Models

作用



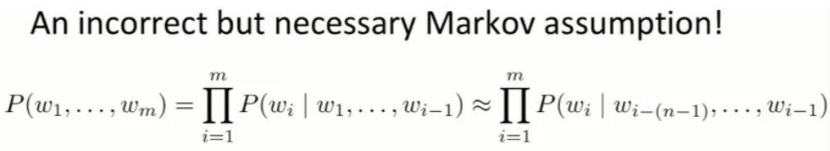
句子的概率
用于翻译，
word order
word choice

traditional Language model：

如何求一句话的概率 p(sentence)
P(ABC)也就是 ABC 三个 word 这样顺序出现的概率，需要统计！！
用 word 前一个 window 来 predict 这个 word
假设 word 只依赖前面的几个 word(假设明显不对)
本来是应该依赖前面所有 word(联合分布的分解)

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1})$$

但为了减少计算量：**假设**只与前面 n 个 word 有关 用来近似！！！
也是 markov 假设，只依赖前面 n 个 state



param estimate 就是 count，联合概率，在一个大的 corpus 里

P(sentence)=p(ABC)=p(A)P(B|A)P(C|B) 通过分解来计算 条件概率！！
此时 n=1

To estimate probabilities, compute for unigrams and bigrams (conditioning on one/two previous word(s):

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

计算**参数**的计算量非常大，这些**参数**内存占用也大！！！

Traditional Language Models

- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)
- There are A LOT of n-grams!
  → Gigantic RAM requirements!
- Recent state of the art: *Scalable Modified Kneser-Ney Language Model Estimation* by Heafield et al.:
  "Using one machine with 140 GB RAM for 2.8 days, we built an unpruned model on 126 billion tokens"

RNN language model

RNN 的每个 hidden state 的 input 是 word embedding，hidden state 也是 word 的 representation，但包含了 word 在**不同 contxt 下的多意义(包含了前面的 context word)，比起单纯的 word embedding，具备 context 的信息，因此可能更 powerful，因此 attention 只对 hidden sate 做，不直接对 embedding 做**

1 首先只有一个网络，一套参数　参数 W 下标都是(高层,低层)
2 hidden layer 圆圈代表**几个 unit**
用来 model: 给定 previous　前面全部 word　来预测下一个 word 的概率
3 递归，**其实就是上一层 hidden state 会传递给下一个 hidden state**
并且是全连接的传递(类似 softmax)，如下图 Whh(4,4) * h t-1 (4,1) =(4,1) ,**然后加在下一层**

Recurrent Neural Network language model

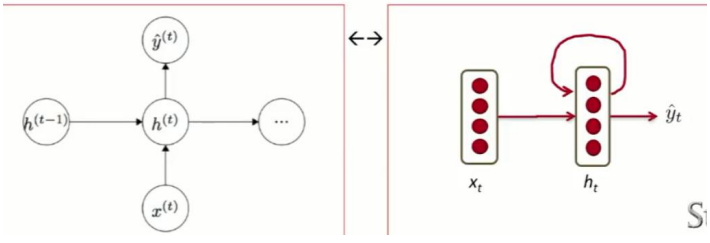
Given list of word **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)}h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$



hidden layer **是需要最后激活的**(元素 elementwise) 就像普通的 hidden layer

计算 **hidden layer z** 时，可以将 x 和 previous hidden state **向量拼在一起做矩阵乘法**，增加计算效率

1 先拼成大向量

$\begin{pmatrix} h \\ x \end{pmatrix}$ 维度是(d+4,1)

纵向拼

2 拼矩阵，Whh：(4,4)　Whx：(4,d)

【Whh，Whx】(4,d+4)　　(横向拼)

两部分相乘等于（4,1）,刚好就是**隐层的维度**

Recurrent Neural Network language model

Main idea: we use the same set of W weights at all time steps!

Everything else is the same:

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)}h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of L at index [t] at time step t

$W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$　$W^{(hx)} \in \mathbb{R}^{D_h \times d}$　$W^{(S)} \in \mathbb{R}^{|V| \times D_h}$

2/2/17

最初的 hidden layer h0 初始化为 0 vector

Ws 是 softmax 参数

Recurrent Neural Network language model

$\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

y 的是给定前面 context word 条件下的每个 **word 的概率值**

Recurrent Neural Network language model

Evaluation could just be negative of average log probability over dataset of size (number of words) T:

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

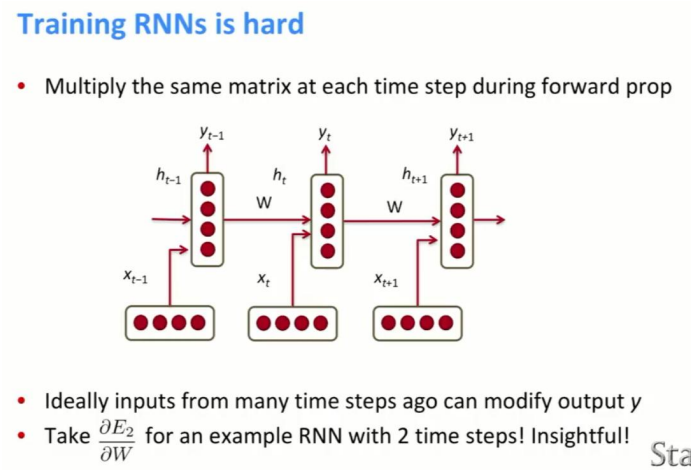
But more common: Perplexity: 2^J

Lower is better!

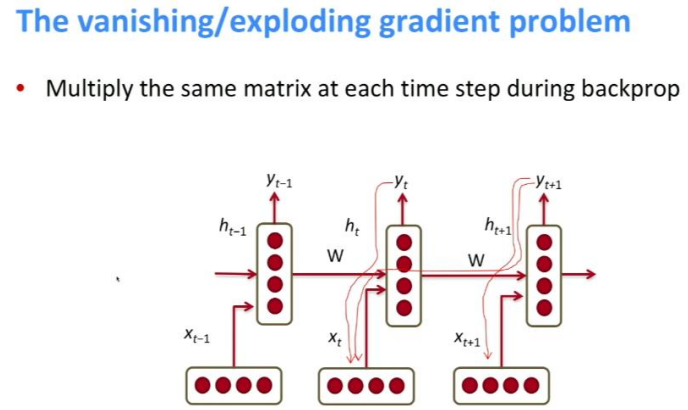
全部 dataset 的 cost function
信息论中，perplexity（复杂度）是用来评价一个语言模型预测一个**样本是否很好的标准**。它可以用来对比语言模型的性能。**复杂度越低，代表模型的预测性能越好。**

Train RNN is hard(gradient varnishing)

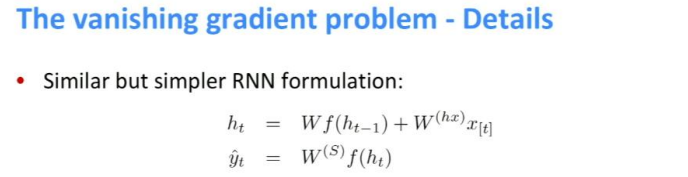
如果 gradient vanishing 后面的 step 的 error 没办法更新到前面的参数，也就是前面的 input 对后面 time step 没有影响！



理想情况下，前面很久前的 input 能影响很长时间以后的 output



但隐层间的 backpropagation ，每次都要乘以 W，**signal 会 vanishing 或 exploding！**



ht 是 ht-1 的函数，自变量和因变量多个值　多对多，求导是 **Jacobian 矩阵**
因为每个因变量值**都受全部自变量影响**

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

total error E 就是整句话的 error，是每个 step 的 error 相加
是来源于**每个 time step y 和真实 label 的计算的 error**

对于每一个 time step 的导数，用 chain rule

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

并且 t 时刻 hidden state 会依赖前一时刻的 hidden state，。。。进而依赖前面所有时刻的

question

dht/dhk 是 **jacobian 矩阵，函数是向量，自变量也是向量**

The vanishing gradient problem - Details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \quad \cdots \quad \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

dht/dhk 式子 可以 chain rule 计算===是多个 **jacobian 矩阵相乘**
每一个因子，都是 jacobian 矩阵！！！
多个矩阵相乘，会越来越大或越来越小！！！

The vanishing gradient problem - Details

- Analyzing the norms of the Jacobians, yields:

$$\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\| \leq \|W^T\| \|\text{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

利用下面公式，来计算 dhj/dhj-1 的 norm，可以得到 upper bound

$$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$

然后每个连乘，就是 beta 的指数形式，可能非常大或非常小

根据 W 的 norm，如果 norm 大于 1,指数就非常大

The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word
- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

不能记住很远的信息

各种导数矩阵

gradient 梯度向量：

定义：

目标函数 f 为单变量，是关于**向量自变量** x=(x1,x2,⋯,xn)T 的函数，

单变量函数 f 对向量 x 求梯度，结果为一个与向量 x 同维度的向量，称之为梯度向量；

$$g(x) = \nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)^T$$

Jacobian 矩阵：

定义：

目标函数 f 为一个**函数向量**，f=(f1(x),f2(x),⋯fm(x))T;

其中，自变量 x=(x1,x2,⋯,xn)T；

函数向量 f 对 x 求梯度，结果为一个矩阵；行数为 f 的维数；列数位 x 的维度，称之为 Jacobian 矩阵；其**每一行**都是由**一个函数的梯度向量转置构成的；**

$$\nabla f(x)=\begin{bmatrix}\frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n}\\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n}\\ \cdots & \cdots & \cdots & \cdots\\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \cdots & \frac{\partial f_m(x)}{\partial x_n}\end{bmatrix}_{m\times n}=\begin{bmatrix}g_1(x)^T\\ g_2(x)^T\\ \cdots\\ g_m(x)^T\end{bmatrix}$$

Hessian 矩阵：二阶导数

实际上，Hessian 矩阵是**梯度向量 g(x)再求导，因为自变量是向量，再求导就是矩阵导数函数向量**对**自变量 x 的 Jacobian 矩阵：**

$$G(x)=\nabla_x\left[g(x)\right]=\nabla_x\left[\left(\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial x_2},\ldots,\frac{\partial f}{\partial x_n}\right)^T\right]$$

$$=\begin{bmatrix}\frac{\partial^2 f}{\partial x_1\partial x_1} & \frac{\partial^2 f}{\partial x_1\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\partial x_n}\\ \frac{\partial^2 f}{\partial x_2\partial x_1} & \frac{\partial^2 f}{\partial x_2\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2\partial x_n}\\ \cdots & \cdots & \cdots & \cdots\\ \frac{\partial^2 f}{\partial x_n\partial x_1} & \frac{\partial^2 f}{\partial x_n\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n\partial x_n}\end{bmatrix}$$

Trick for exploding gradient: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a maximum value.

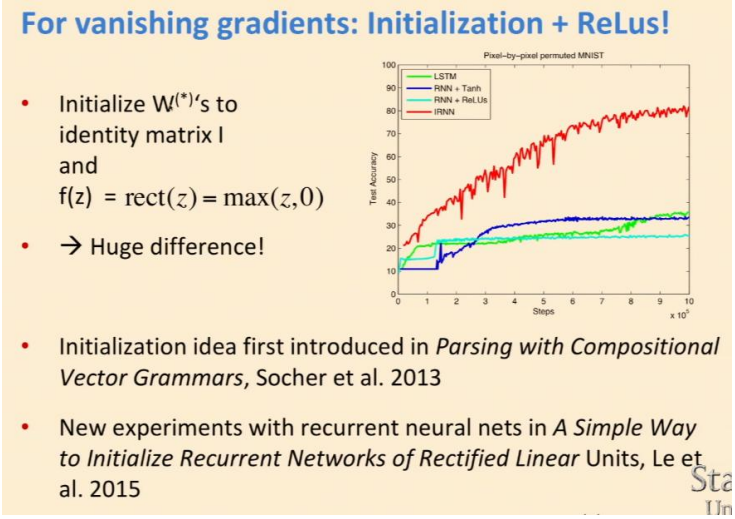
Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

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

- Makes a big difference in RNNs.

explode 处理起来比较简单，**只需 cut**

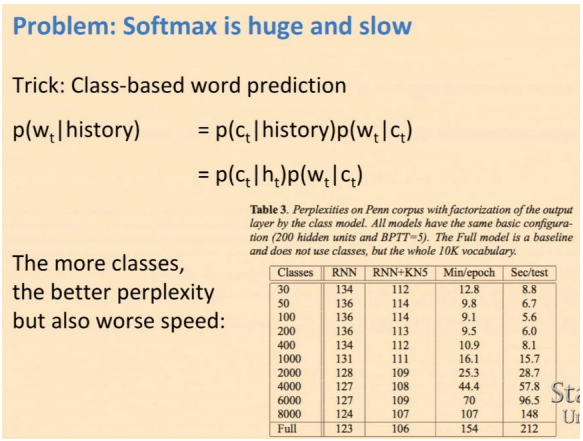
vanishing 处理比较难



解决 vanishing;

1 用 relus 2initial W I 矩阵

Softmax is slow



softmax 每次会对每个 word 进行预测，计算量很大

class-based word prediction

可以先按**词频进行分类 word，先预测分类 C，锁定区域后**再预测该 word

分类越多，越快

也可以对 **word 进行分类**，而不是预测下一个 word，修改 softmax 层

Opinion Mining with Deep Recurrent Nets

Goal: Classify each word as

direct subjective expressions (DSEs) and
expressive subjective expressions (ESEs).

DSE: Explicit mentions of private states or speech events expressing private states

ESE: Expressions that indicate sentiment, emotion, etc. without explicitly conveying them.

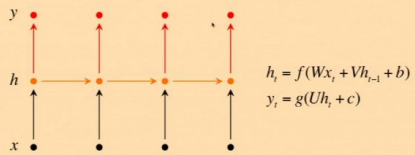
The	committee	,	as	usual	,	has
O	O	O	B_ESE	I_ESE	O	B_DSE

refused	to	make	any	statements	.
I_DSE	I_DSE	I_DSE	I_DSE	I_DSE	O

RNN&Deep biRNN

Approach: Recurrent Neural Network

- Notation from paper (so you get used to different ones)

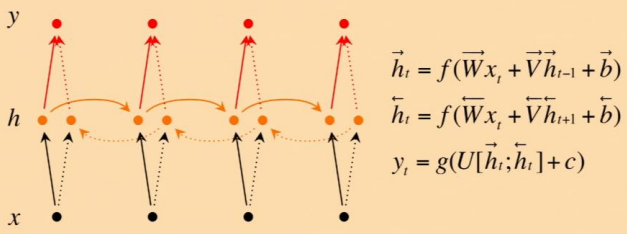


- *x* represents a token (word) as a vector.
- *y* represents the output label (B, I or O) – g = softmax !
- *h* is the memory, computed from the past memory and current word. It summarizes the sentence up to that time.

中间是 memory

Bidirectional RNNs

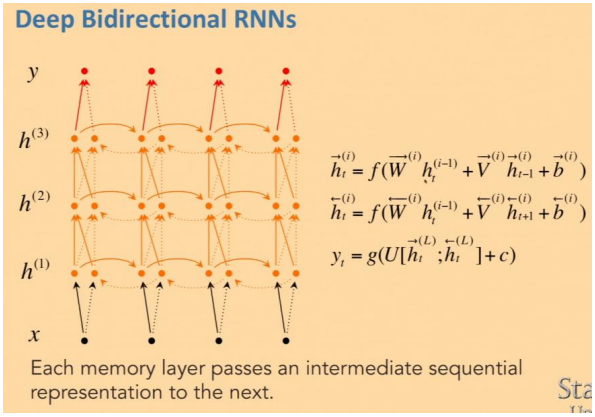
Problem: For classification you want to incorporate information from words both preceding and following



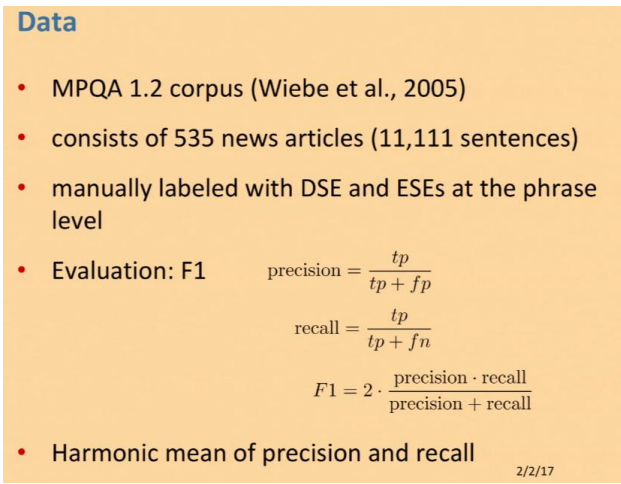
$h = [\vec{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

h 变成 2 个 hidden state 拼接

但需要预先知道**全部的 x，才能计算 hidden state**



其实就是多个 **hidden layer** **deepnet**

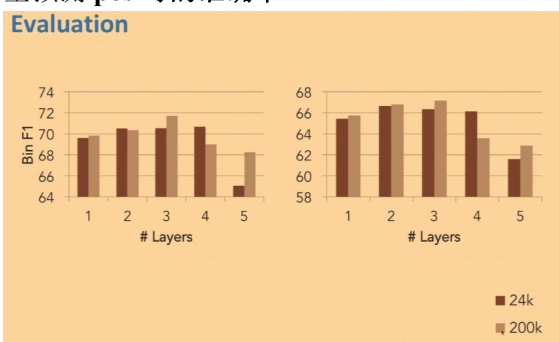


unbalanced data 是指在测试阶段，test data 不平衡

precision 和 recall 相反

precision 是考虑了 negative 情况和 positive 情况

全预测 **pos 时的准确率**



Structured Training for Neural Network Transition-Based Parsing

David Weiss, Chris Alberti, Michael Collins, Slav Petrov

Presented by: Shayne Longpre

What is SyntaxNet?

- ❖2016/5: Google announces the "World's Most Accurate Parser Goes Open Source"
- ❖SyntaxNet (2016): New, fast, performant Tensorflow framework for syntactic parsing.
- ❖Now supports 40 languages -- Parse McParseface's 40 'cousins'

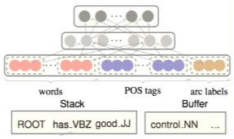
用于解析句子

baseline model: chen&manning model

What is SyntaxNet?

- ❖2016/5: Google announces the "World's Most Accurate Parser Goes Open Source"
- ❖SyntaxNet (2016): New, fast, performant Tensorflow framework for syntactic parsing.
- ❖Now supports 40 languages -- Parse McParseface's 40 'cousins'

Chen & Manning (2014)



Weiss et al. (2015)

+ Unlabelled Data
+ Tune Model
+ Structured Perceptron & Beam Search

Andor et al. (2016)

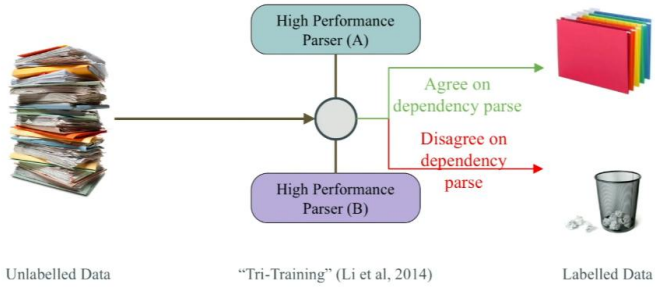
+ Global Normalization

= ***SyntaxNet***

3 New Contributions (... since Q2 in Assignment 2)

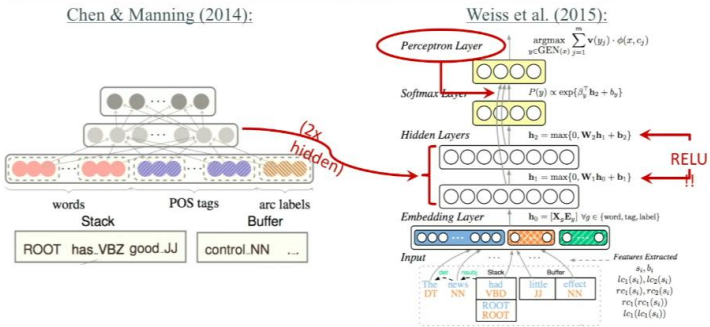
1. Leverage Unlabelled Data -- "Tri-Training"
2. Tuned Neural Network Model
3. Final Layer: Structured Perceptron w/ Beam Search

1. Tri-Training: *Leverage Unlabelled Data*



只取 agree 的 parse tree，变成 label dataset

2. Model Changes



3. Structured Perceptron Training + Beam Search

Problem: Greedy algorithms are unable to look beyond one step ahead, or recover from incorrect decisions.

Solution: Look forward -- search the tree of possible transition sequences.

- Keep track of K top partial transition sequences up to depth m
- Score transition using perceptron:

$$\underset{y \in \mathrm{GEN}(x)}{\operatorname{argmax}} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \phi(x, y_1 \ldots y_{j-1}).$$

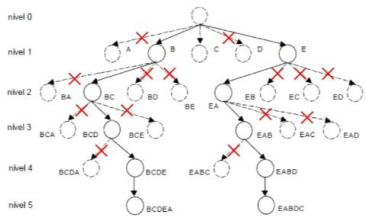


Figura 1 – Árvore de busca utilizando o *beam search*

不是 greedy 遍历，而是用 bean search parse tree

Conclusions

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

- Identify specific flaws in existing models (greedy algorithms) and solve them. In this case, with:
  - More data
  - Better tuning
  - Structured perceptron and beam search
- Final step to **SyntaxNet**: Andor et al. (2016) solve the "Label Bias Problem" using Global Normalization

09 Machine translation and advanced recurrent LSTMs and GRUs

Recap of most important concepts

Word2Vec $J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$

Glove $J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$

Nnet & Max-margin

$$J = \max(0, 1 - s + s_c)$$

Recap of most important concepts

Multilayer Nnet & Backprop

$$\begin{aligned} x &= z^{(1)} = a^{(1)} \\ z^{(2)} &= W^{(1)} x + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\ a^{(3)} &= f\left(z^{(3)}\right) \\ s &= U^T a^{(3)} \end{aligned}$$

$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)}),$

$\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}$

传统 MTS

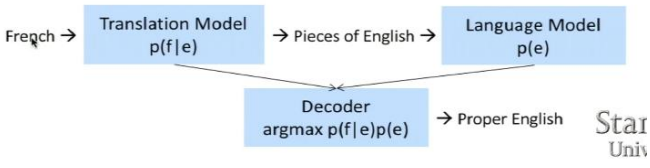
需要很多 **human feature extraction**

Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English
- Probabilistic formulation (using Bayes rule)

$$\hat{e} = \text{argmax}_e p(e|f) = \text{argmax}_e p(f|e)p(e)$$

- Translation model p(f|e) trained on parallel corpus
- Language model p(e) trained on English only corpus (lots, free!)



统计机器翻译：

bayes rule 用 evidence **p(f | e)**来预测后验

p(f | e) 是 translation model 将 f 翻译成 e

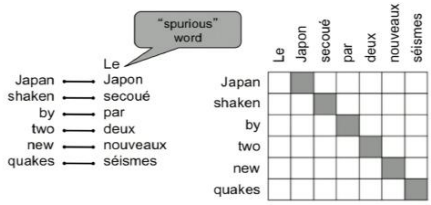
其实就是两种语言的 alignment，1 对 1 配对 **给定 e 就对应 f, 反过来也对!**

question

右边 language model 得出 e 的概率

Step 1 for training translation model: Alignment

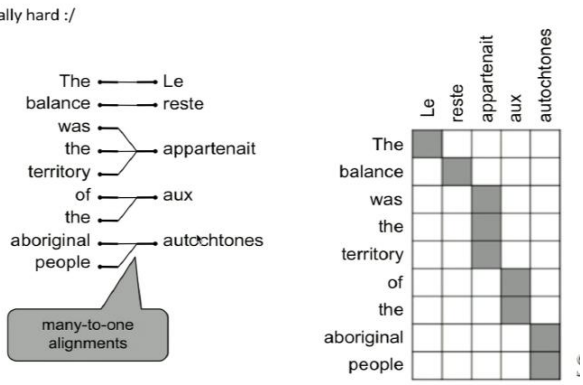
Goal: know which word or phrases in source language would translate to what words or phrases in target language? → Hard already!



translation model 构建 alignment

Step 1: Alignment

Really hard :/



很多对应**多个翻译**或者**没有对应的翻译**，很难！！

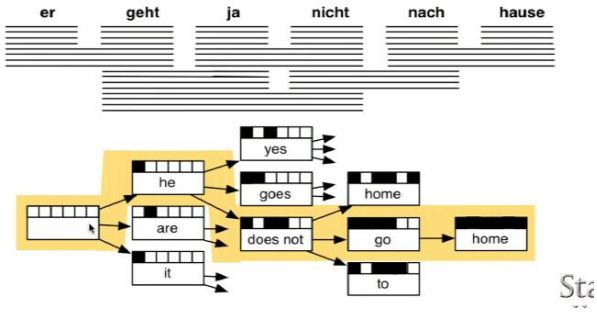
After many steps

Each phrase in source language has many possible translations resulting in large search space:

给定 f 有很多 e 很多 candidate，很多 search space，to 得到最终的 english

Decode: Search for best of many hypotheses

Hard search problem that also includes language model



Traditional MT

- Skipped hundreds of important details
- **A lot** of human feature engineering
- Very complex systems

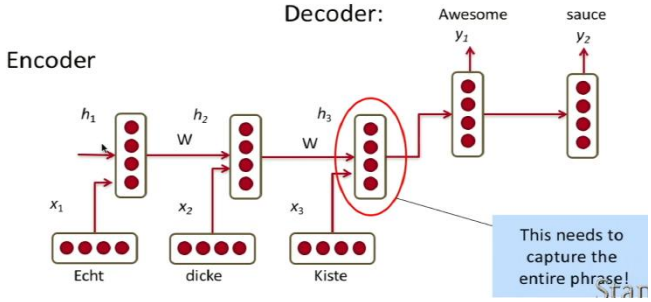
- Many different, independently trained machine learning problems

需要很多的**手工特征**，但我需要

end to end model

Deep learning to the rescue! … ?

Maybe, we could translate directly with an RNN?



包含 encoder 和 decoder 结构不同

encoder 没有 output **decoder 没有 input** 两个独立的网络结构

encoder 最后一个 hidden state **代表了整个句子**

但只能记住前面 **4-5 个 phrase，只能翻译短的句子**

decoder 应该会选择长**度大一点**，如果翻译结束，就输出结束符

最终的输出和正取翻译计算 loss

使得 **p(y1,y2,y3...|x1,x2,x3..)的概率最大**

MT with RNNs – Simplest Model

Encoder: $h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$

Decoder: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)}h_{t-1}\right)$

$y_t = softmax\left(W^{(S)}h_t\right)$

Minimize cross entropy error for all target words conditioned on source words

$$\max_{\theta} \frac{1}{N}\sum_{n=1}^{N} \log p_\theta(y^{(n)}|x^{(n)})$$

decode 没有 x input 只有 hidden state

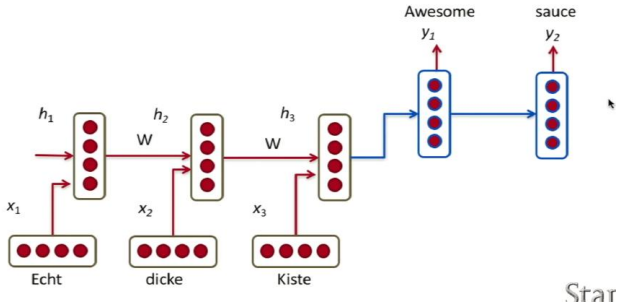
Notation: Each input of ϕ has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)}h_{t-1}\right)$

phi 函数，代表**每个自变量**：ht-1 和 xt 都有属于自己的**线性变换矩阵 Whh Whx**

还需要一些扩展！！！！

RNN Translation Model Extensions

1. Train different RNN weights for encoding and decoding

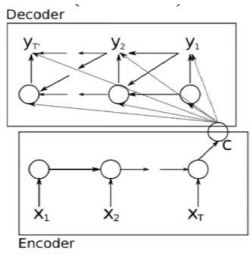


Star

encoding 和 decoding 不同的网络，其实就是两个网络结构，并且 W 参数不同

2. Compute every hidden state in decoder from
   - Previous hidden state (standard)
   - Last hidden vector of encoder $c=h_T$
   - Previous predicted output word y_{t-1}

$$h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1})$$

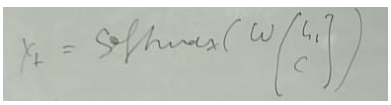


Cho et al. 2014

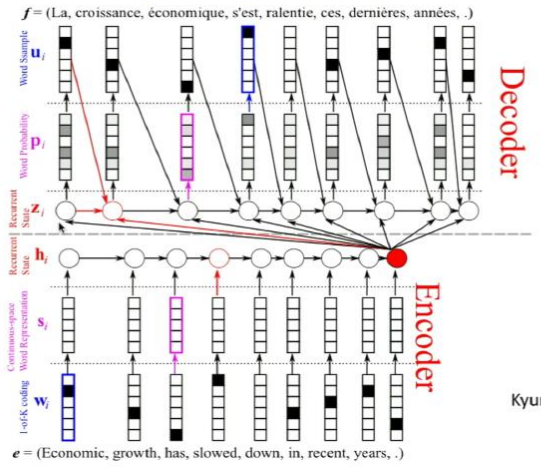
Stanfc

decoder 部分：计算没个 hidden layer 不仅需要前一个 **hidden layer**，还需要 **encoder 的 C 还需要前一次 output y** 三个参数

计算 y 时，不仅需要 hidden layer 还需要 C



Different picture, same idea



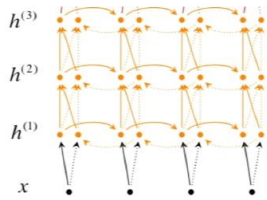
Kyunghyun Cho et al. 2014

Star
Univ

w 是 onehot s 是 embedding 然后 C 传递给 decoder

RNN Translation Model Extensions

3. Train stacked/deep RNNs with multiple layers
4. Potentially train bidirectional encoder

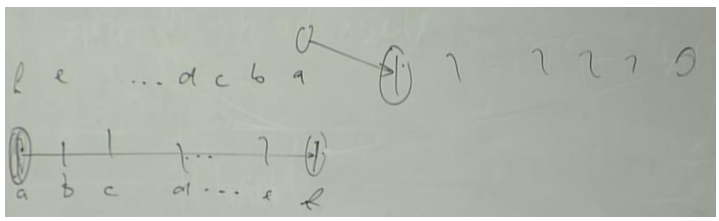


5. Train input sequence in reverse order for simpler optimization problem: Instead of A B C → X Y, train with C B A → X Y

Star

1stacked RNN

2reversed order 的原因 参看讲义



GRU 为了解决 short memory

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

- Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

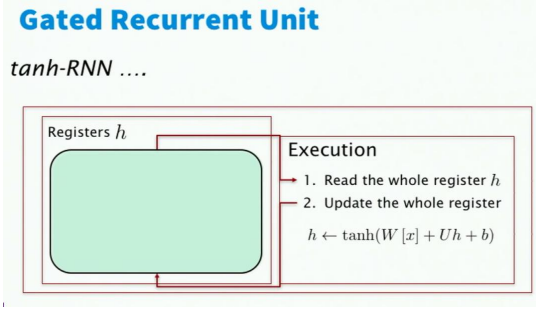
其实就是在**计算当前隐层**的时候加一些步骤：

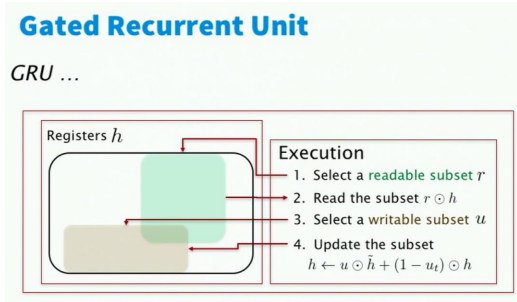
两个 gate：update gate　reset (forget) gate

gate 的输入相同，都是**隐层 t-1 和 xt**，只是 weight 不同

两个 gate 都经过 **sigmoid(每个输出 gate 元素都是 0-1)**

因此 gate 就是 0-1 的向量





GRU 步骤：

1 先计算 h tilde(新值)　**此时用 forget gate 对激活后的 previous hidden state 筛选(先遗忘) h tilde 作为新的预选值**

2 再计算 ht　再 ht-1 和 h tilde　(新的值和旧的值)综合一下　只用一个 update gate

只用 update gate，一个 gate 当两个 gate 用(update 和 forget)

- New memory content: $\tilde{h}_t = \tanh(Wx_t + r_t \circ Uh_{t-1})$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

圆圈是 element wise 相乘

遗忘门的好处：剔除过去不重要的记忆

情感分析：虽然很好很好很好很好.....但很差　需要忽略忘记前面的部分，重点是后面！！

更新门的好处：

如果 z=1　ht=ht-1 此时没有 vanishing gradient 问题 不需要乘以矩阵 W(可以记住很长的过去)

情感分析：我喜欢这个 movie，一个浪漫的故事　我们更注重前面一句话，不遗忘

希望留住记忆

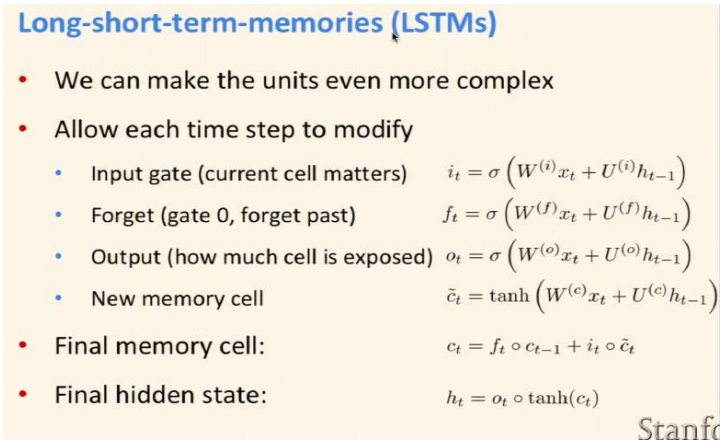
model 会 learn，何时 forget 何时 remember！！！

r=1 z=1 此时就是 standard RNN

RNN 是特殊的 GRU!! GRU 更灵活！！！

缩写 ht=GRU(Xt, ht-1)

LSTM



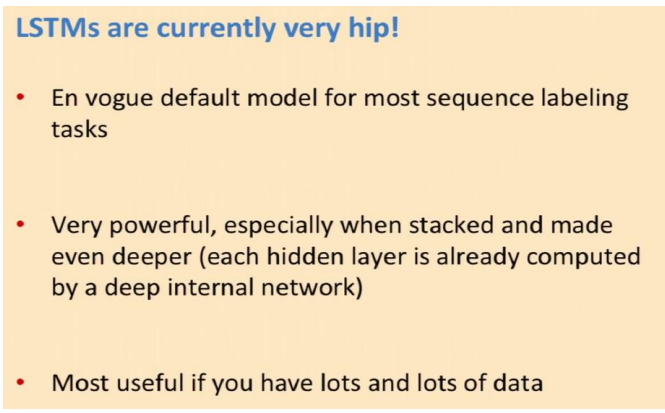
3 个 gate: input gate (对新值 tilde 的筛选)　forget gate　output gate

1 c tilde(新值)**记忆单元**　**就是普通 RNN 计算** hidden state 值 (**先不遗忘**，跟 GRU 不同)
2 再 ct-1 和 c tilde　(新的值和旧的值)综合一下
c tilde 由 input gate;　c t-1 由 forget gate 独立控制

3 output gate 最终筛选　先激活 再筛选
对 prediction 的重要的筛选

gate 的 input 都一样！！

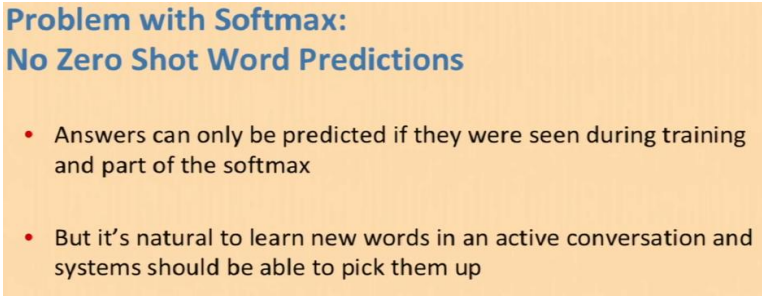
weight initialize 很重要，如果很大，对于 sigmoid，此时导数为 0，grediant vanishing



big data+ stack deep　powerful

point-sentinel model--solve zero shot

recent improve to RNN



训练好的 RNN，参数已经确定

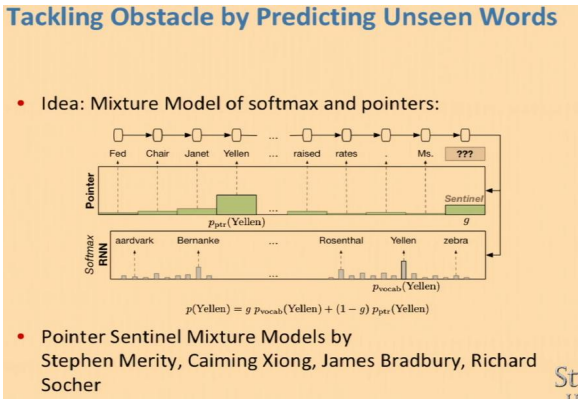
softmax 的问题：只能输出在 train 里见过的 word，没见过的 word 没法输出

train **很 rare 的 word 也不能很好 predict**

解决办法：**character model**，不会出现不存在的问题，但如果想 remenber 很多 word，需要 stack 很多 LSTM.**计算量很大！！！**

解决：其实是一个 **mixed model of softmax and pointer**

question



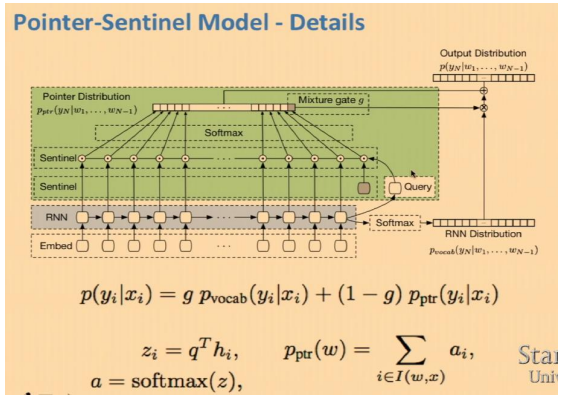
pointer 提供两种选择：

1 是从 corporal 里的前 100 个 input words 里 copy 一个 word，作为 predict　不经过 softmax

因为用 RNN 的时候，不一定是 train 的 words，但我们可以从 input 里 copy

2 还是用 softmax

如果不 copy，就还是用 softmax



混合概率模型；将 pointer 和 softmax 混合

g 来决定混合程度 相当于是 gate

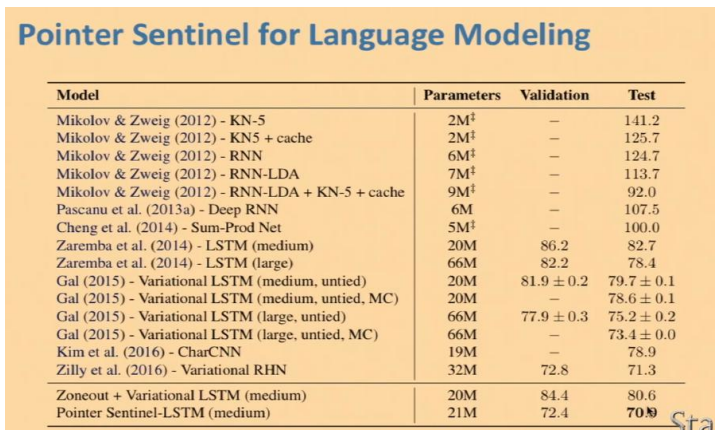
Query 将最后一个 hidden layer 输出传给一个 single layer network **Query 得到权重 q**

然后用 q 和所有 time step 的 h 进行点乘，每个点乘得到一个数 z sentinel 哨兵

其实就是对过去 word 权重的分配(类似于注意力!)**也就是指向前面 time step 的 words**

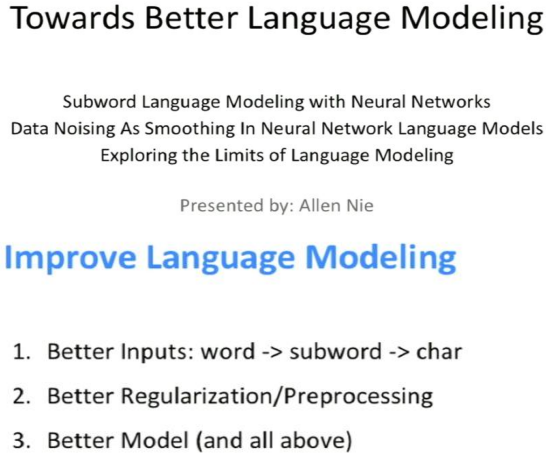
z 再经过 softmax，

如果多个 word 出现多次，sum a 得到概率

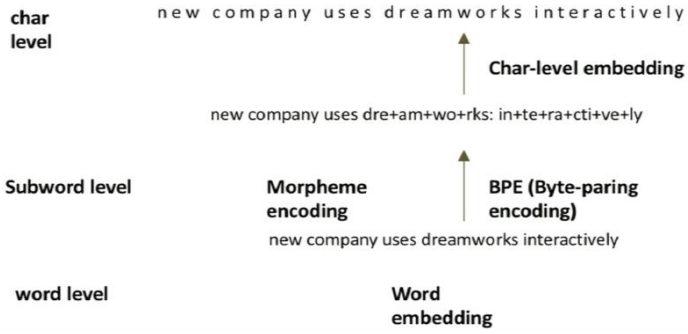


Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M‡	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	–	125.7
Mikolov & Zweig (2012) - RNN	6M‡	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	–	92.0
Pascanu et al. (2013a) - Deep RNN	6M	–	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	–	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	81.9 ± 0.2	79.7 ± 0.1
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	–	78.6 ± 0.1
Gal (2015) - Variational LSTM (large, untied)	66M	77.9 ± 0.3	75.2 ± 0.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	–	73.4 ± 0.0
Kim et al. (2016) - CharCNN	19M	–	78.9
Zilly et al. (2016) - Variational RHN	32M	72.8	71.3
Zoneout + Variational LSTM (medium)	20M	84.4	80.6
Pointer Sentinel-LSTM (medium)	21M	72.4	**70.9**

复杂度的下降



1. Better Inputs



用字符集 embedding 会更好，英文 vocabulary 更小

1. Better Inputs

Table 5. *Re-scoring experiments on NIST RT05 Meeting recognition setup with subword-level RNN models.*

Model	WER [%]	size (MB)	size (MB) compressed
Word-based bigram	27.0	93	11
Word-based 4-gram	25.1	464	43
Word-based 4-gram (pruned)	25.5	142	15
Subword RNN-160 (text format)	26.5	6.7	-
Subword RNN-160 (quantized)	26.5	-	0.6
Subword RNN-480 (text format)	25.0	21.3	-
Subword RNN-480 (quantized)	24.9	-	1.7

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m

Subword Language Modeling with Neural Networks - Tomas Mikolov, Ilya Sutskever, et al. 2012
Character-Aware Neural Language Models - Yoon Kim, et al. 2015

2. Better Regularization/Preprocessing

1. Regularization:
   a. Use dropout (Wojciech Zaremba, et al., 2014)
   b. Use stochastic feedforward depth (Gao Huang, et al., 2016)
   c. Use norm stabilization (David Krueger and Roland Memisevic, 2015)
   d. (and many others...)
2. Preprocessing:
   a. Randomly replacing words in a sentence with other words, or use bigram statistics to generate Kneser-Ney inspired replacement (Ziang Xie, et al., 2016)

preprocessing　因为我们文本有限不一定代表全部的语言如英语(**true distribution**)如何处理：

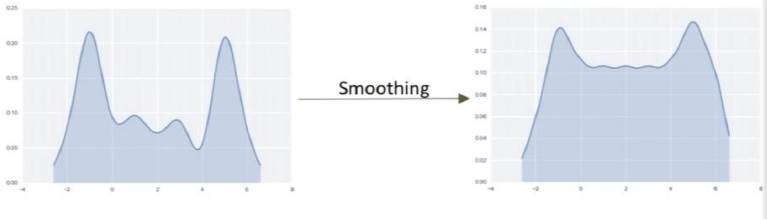
2. Better Preprocessing

1. Replace a word with a **fixed** drop rate
2. Replace a word with **adaptive** drop rate by how rare two words appear together (like "Humpty Dumpty"), and replace by a unigram draw over vocab.
3. Replace a word with **adaptive** drop rate, and draw word from a **proposal distribution** ("New York").

Noised	$\gamma(x_{1:2})$	$q(x)$	Analogue
x_1	γ_0	$q(\text{“\_”}) = 1$	interpolation
x_1	γ_0	unigram	interpolation
x_1	$\gamma_0 N_{1+}(x_1, \bullet)/c(x_1)$	unigram	absolute discounting
x_1, x_2	$\gamma_0 N_{1+}(x_1, \bullet)/c(x_1)$	$q(x) \propto N_{1+}(\bullet, x)$	Kneser-Ney

替换一些 word

2. Better Preprocessing



使得 frequency 高的 word 通过替换，变得词频低(相反...)

2. Better Regularization/Preprocessing

Model	Char-PTB Valid	Char-PTB Test	Word-PTB Valid	Word-PTB Test	Text8 Valid	Text8 Test
Unregularized LSTM	1.466	1.356	120.7	114.5	1.396	1.408
Weight noise	1.507	1.344	–	–	1.356	1.367
Norm stabilizer	1.459	1.352	–	–	1.382	1.398
Stochastic depth	1.432	1.343	–	–	1.337	1.343
Recurrent dropout	1.396	1.286	91.6	87.0	1.386	1.401
Zoneout	1.362	1.252	81.4	77.4	1.331	1.336
NR-dropout (Zaremba et al., 2014)	–	–	82.2	78.4	–	–
V-dropout (Gal, 2015)	–	–	–	**73.4**	–	–
RBN (Cooijmans et al., 2016)	–	1.32	–	–	–	1.36
H-LSTM + LN (Ha et al., 2016)	1.281	1.250	–	–	–	–
3-HM-LSTM + LN (Chung et al., 2016)	–	**1.24**	–	–	–	**1.29**

Noising scheme	Validation	Test
Medium models (512 hidden size)		
none (dropout only)	84.3	80.4
blank	82.7	78.8
unigram	83.1	80.1
bigram Kneser-Ney	**79.9**	**76.9**
Large models (1500 hidden size)		
none (dropout only)	81.6	77.5
blank	79.4	75.5
unigram	79.4	76.1
bigram Kneser-Ney	**76.2**	**73.4**
Zaremba et al. (2014)	82.2	78.4
Gal (2015) variational dropout (tied weights)	77.3	75.0
Gal (2015) (untied weights, Monte Carlo)	—	**73.4**

Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations - Krueger, et al. 2016

Data Noising as Smoothing in Neural Network Language Models - Ziang Xie, et al. 2016

6. Main Improvement: Better Units

- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU) introduced by Cho et al. 2014 (see reading list)
- Main ideas:
  - keep around memories to capture long distance dependencies
  - allow error messages to flow at different strengths depending on the inputs

10 Neural Machine Translation and Models with Attention

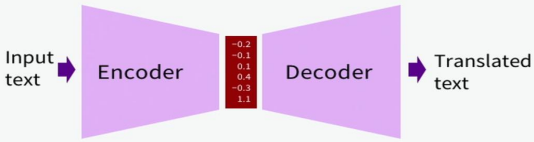
Neural MT 定义

What is Neural MT (NMT)?

Neural Machine Translation is the approach of modeling the **entire** MT process via **one** big artificial neural network*

*But sometimes we compromise this goal a little

Neural encoder-decoder architectures



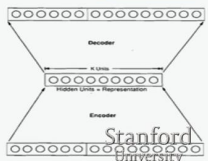
Neural MT: The Bronze Age

[Allen 1987 IEEE 1st ICNN]

3310 En-Es pairs constructed on 31 En, 40 Es words, max 10/11 word sentence; 33 used as test set

The grandfather offered the little girl a book ➔
El abuelo le ofrecio un libro a la nina pequena

Binary encoding of words – 50 inputs, 66 outputs; 1 or 3 hidden 150-unit layers. Ave WER: 1.3 words



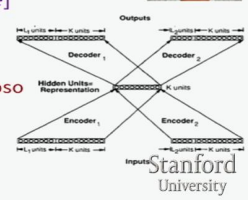
Neural MT: The Bronze Age

[Chrisman 1992 *Connection Science*]

Dual-ported RAAM architecture [Pollack 1990 *Artificial Intelligence*] applied to corpus of 216 parallel pairs of simple En-Es sentences:

You are not angry ↔ Usted no esta furioso

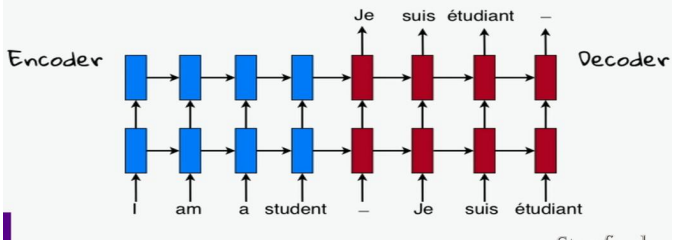
Split 50/50 as train/test, 75% of sentences correctly translated!



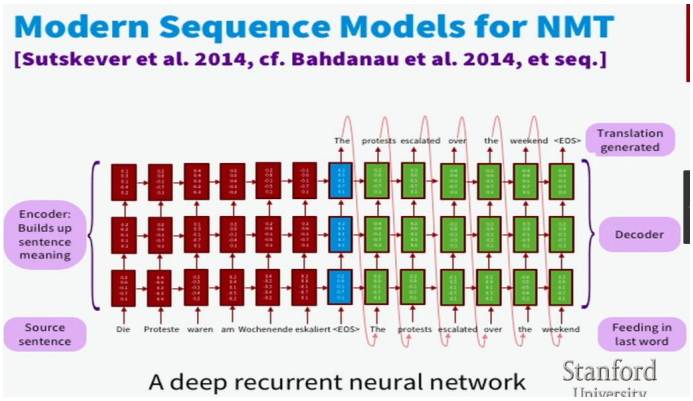
8

Modern Sequence Models for NMT

[Sutskever et al. 2014, cf. Bahdanau et al. 2014, et seq.]

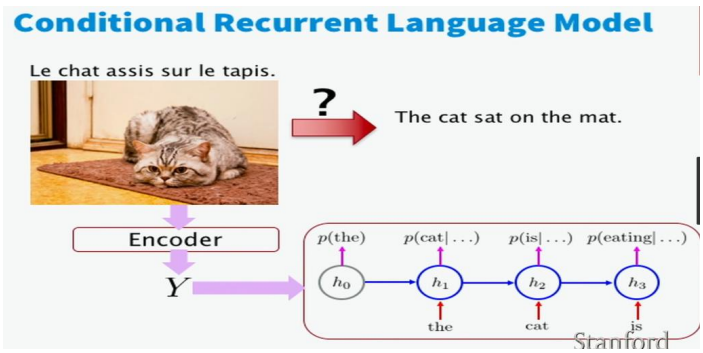


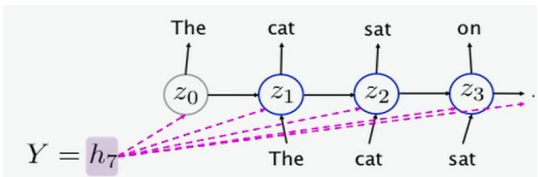
用 sequence model 进行翻译



两个 RNN　一个作为 encoder 一个是 decoder

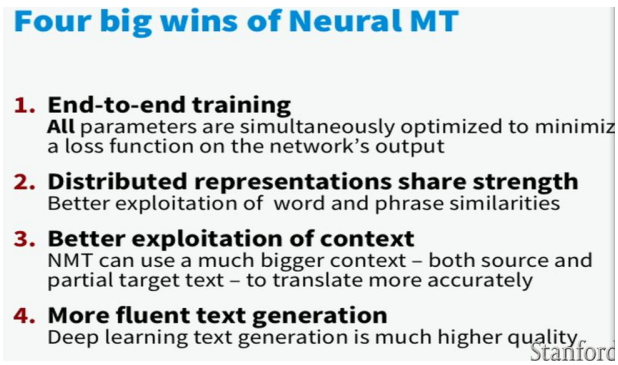
conditional recurrent language model





condition 是 **encoder** hidden state(**被翻译的句子**) ，会作为每一次 decoder output 的输入(条件)

Neural MT 的优势

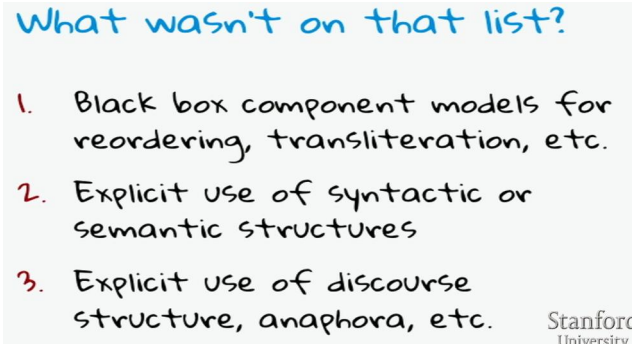


1 end to end 所有参数全部一起更新，一起 train，end to end training **很有效率！！**

2 distribute representation word(word 的分布，其实就是 similarity)

3 更好的探索文本的 context

4 生成更流畅的文本



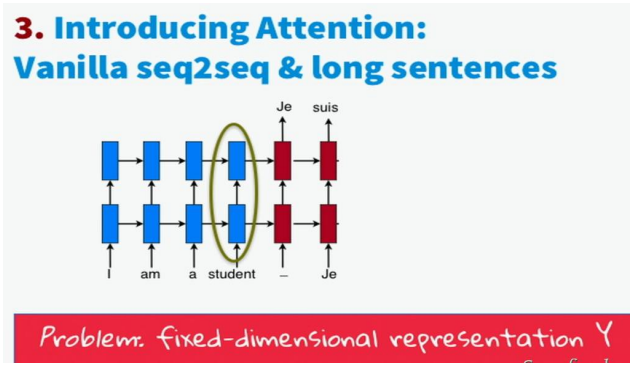
1end2end 是黑匣子

2 都是自己学的，没有用现有的句法和句义信息

3 没有显示的利用 **语言论述的结构**

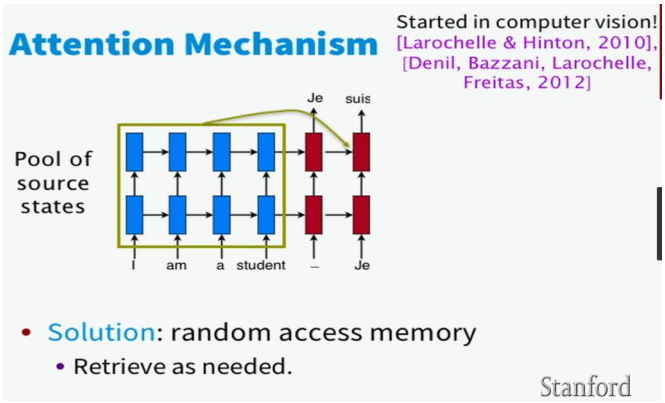
google 翻译的不是很好，不准确

Attention



只有 encoder 的 **last hidden state are available to decoder**，能记住的不是很长，或者这样的话**可能对前面的 word 记忆不是很好**，因此实际的表现：

只对短的句子效果好，长的句子效果不好



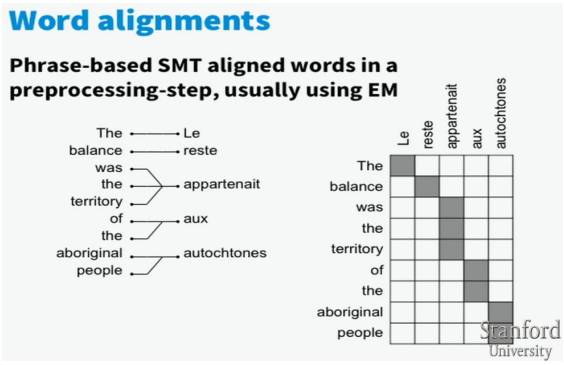
all encoder hidden state are available to **decoder**

两种思路：hard & soft

全部 word 的 encoding 都用来翻译(全部 hidden state)**建立一个 pool**，每次 decoder 输出一个翻译，会从 pool 里选取一个 hidden layer state(**用强化学习来选择**)**hard decision**，如果用权重，可以用 **soft decision**

attention 只是上面的一种思想的实现，可以翻译更长的句子

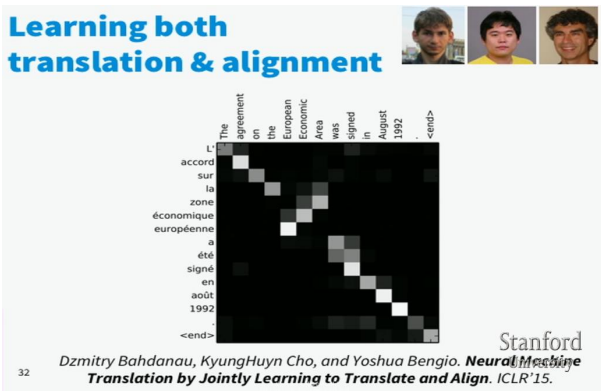
attention 和 word alignment 本质相似，都是找到每个 output word 的 alignment

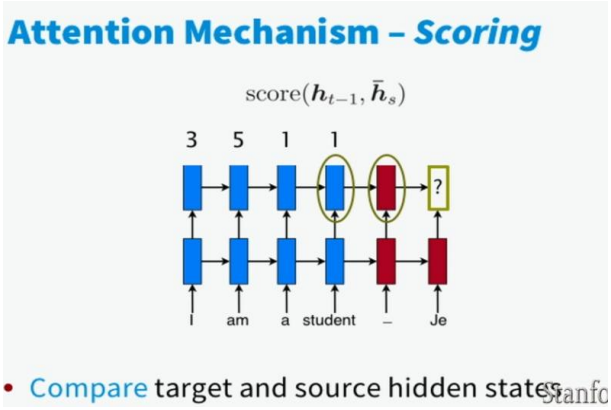


我们想知道 decoder 每次翻译的 output word 需要 pay attention 哪些 word。

target phrase 会 pay 不同的 attention to the **source phrase**

word alignments 是 explicit，而 attention 是 implicit！！！





计算 decoder 某一层的 hidden state 需要下面一层的 1input，2ht-1，3 encoder hidden state

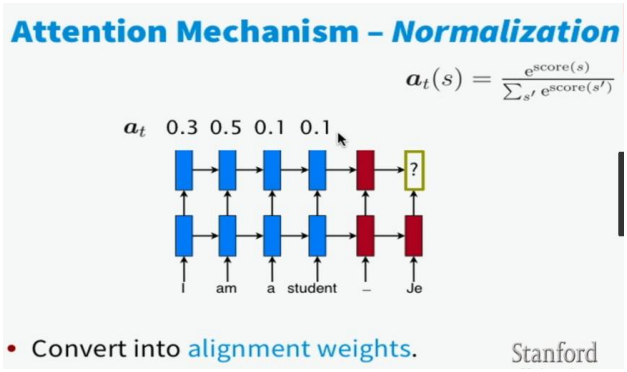
计算 encoder hidden state：先要 score

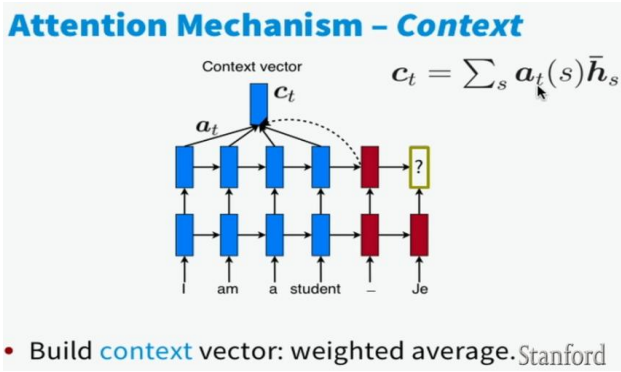
score 也就是 encoding hidden state 的权重

每个 encoding hidden state score 由前一个 **decoding hidden state ht-1 和 前面一个决定**

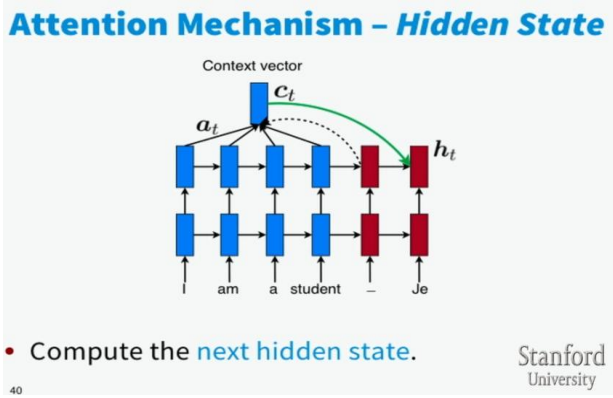
然后 softmax 将 score 转换为**权重，得到多个 encoder state 的 average vector (c)**

然后 c 和 ht-1 再计算该层的 ht

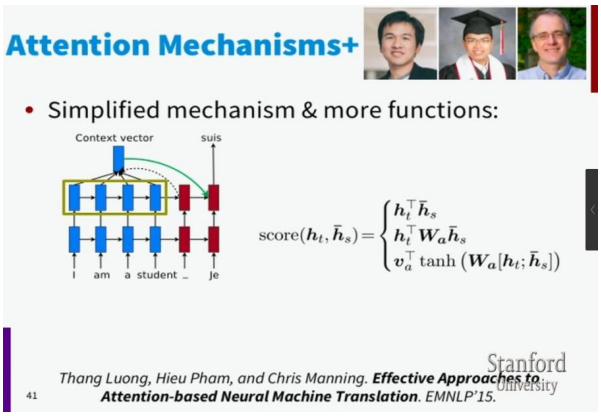




计算得到 **context vector** (weighted source hidden state)



上面我们计算 score 是直接 softmax，但没有交互作用



计算 score **最简单的方法**就是计算当前 decoder state 和 encoder state 的 vector 内积，越相似 score 越大

此时应该是先按照原来的方法计算出 ht，然后再计算 score，再重新计算 ht？？

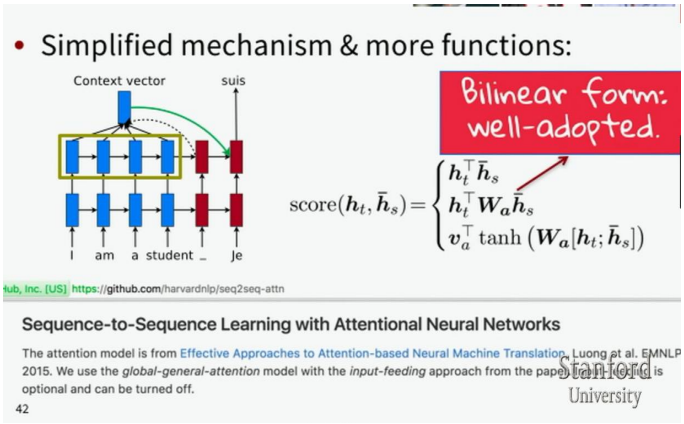
question

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) \end{cases}$$

这里我们用最后一个：

将 decoder ht 和 encoder 的一个 state **两个向量拼在一起**，然后加一层神经网络**计算 weight a，得到该 state 的 weight(上面也可以多个 endoder state 一起做，最终得到向量)**

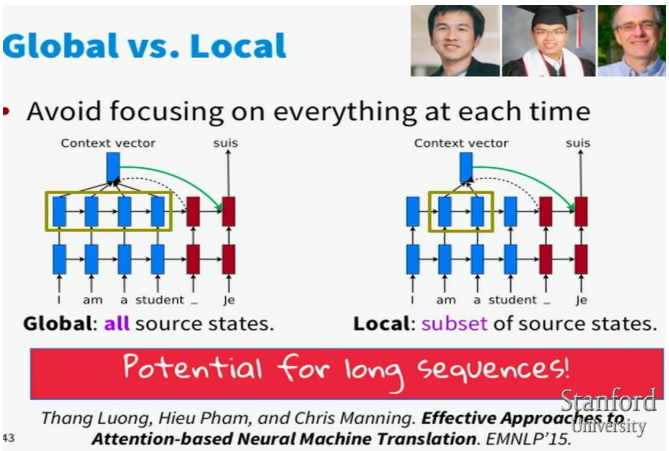
其实就是训练 ht，hs 得到函数 a



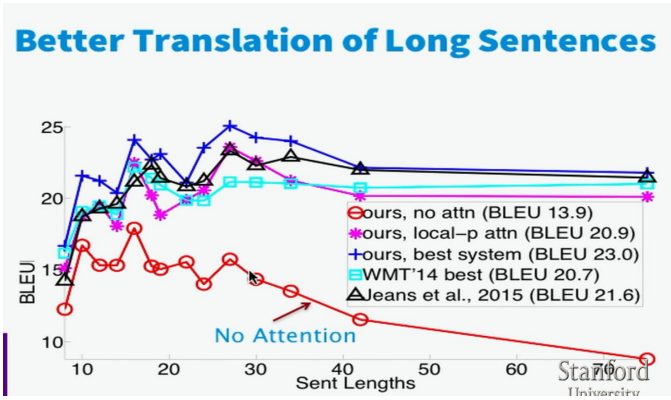
中间这个也挺好，更 inner product 类似(hs 和 ht 的**交互作用**)，中间加个 W，使得**交互作用更加复杂，因此效果更好，第三种没有交互作用，各自乘以一个 W**

因此对于 perceptron 一层的神经网络，input 直接只是线性组合，没有交互关系！！！！

所以也可以多层网络来求 a

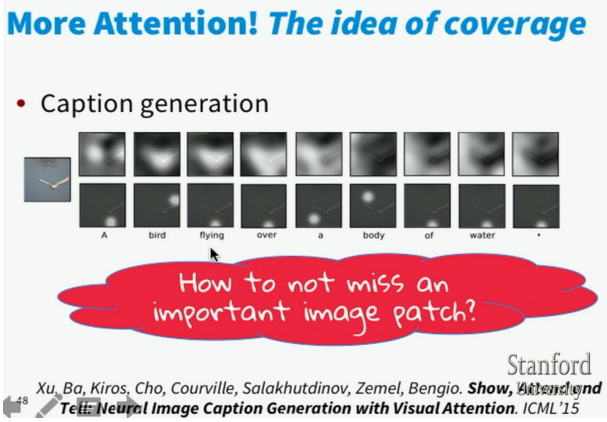


global 计算量很大！！尤其是后向传播的时候，因此只对**局部**的 encoder state pay attention



普通的随着句子越长，score 越低

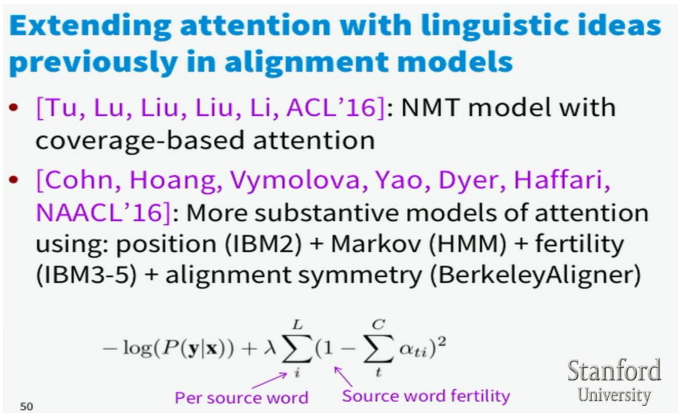
有 attention 的 model，随着句子的增长，score 趋于稳定！！！



图片文字生成

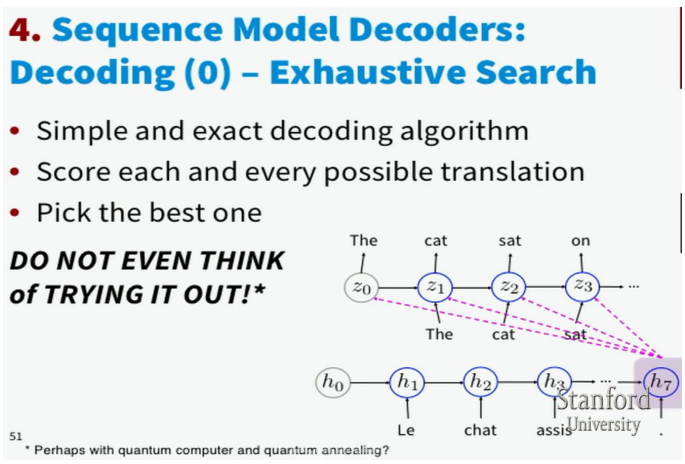
每生成一个文字，**需要 pay 图片不同的部位** patch

courage_attention

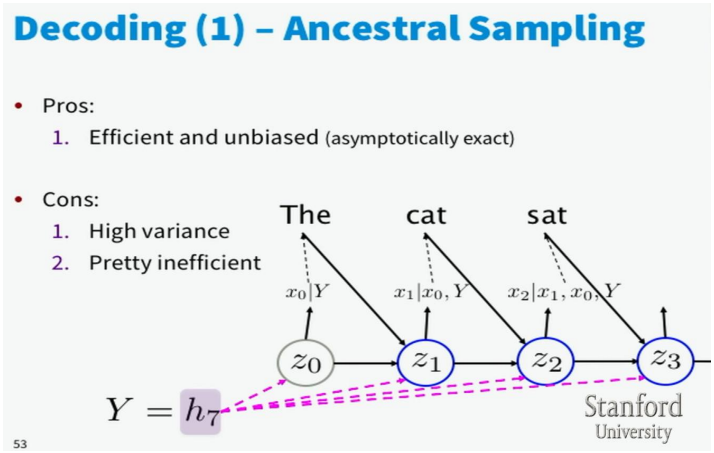


question

Decoder

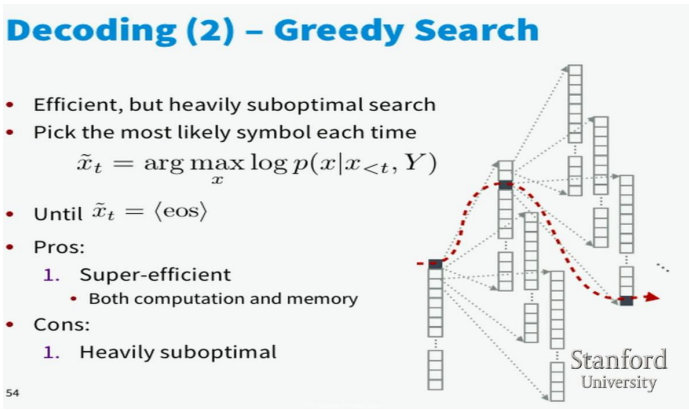


如何 decoding，因为翻译的话会受 **decoder sequence 长度的影响**，而穷尽所有的的 length 选择 score 最好的！错误！！！



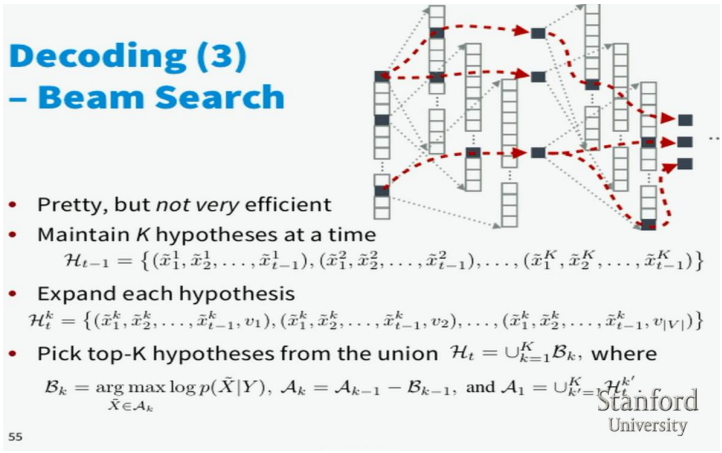
根据每个 output 的概率随机 sample 一个 word，然后也输入给下个 state，然后再随机 sample，直到遇到结束符！！！！(不固定长度)

variance high，每一次翻译都不同！



每次选择，然后再选概率最大的

但**不能保证最佳的概率最大的 word 翻译**，很多时候是 suboptimal ！**但效率高**



maintain top k hypothesis(**是 word 组合联合概率**)

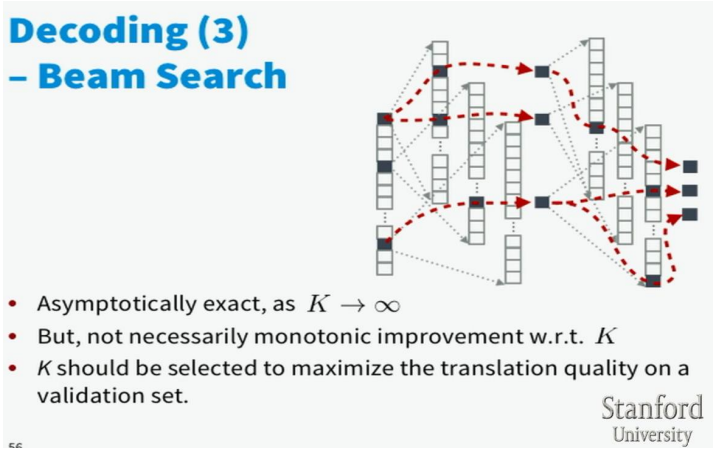
1 第一次次选 top **k=5 word** 然后并行的 **generate 下个 top k=5 word**，共 25 种可能，

2 第二次**然后选 top5 (2 个 word)**

再继续！！！ **maintain constant num hypothesis**

k 无穷大 unbias 就是 gready

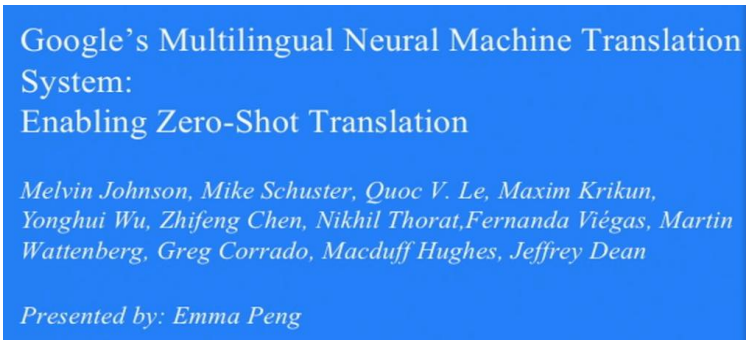
k 小，是 bias

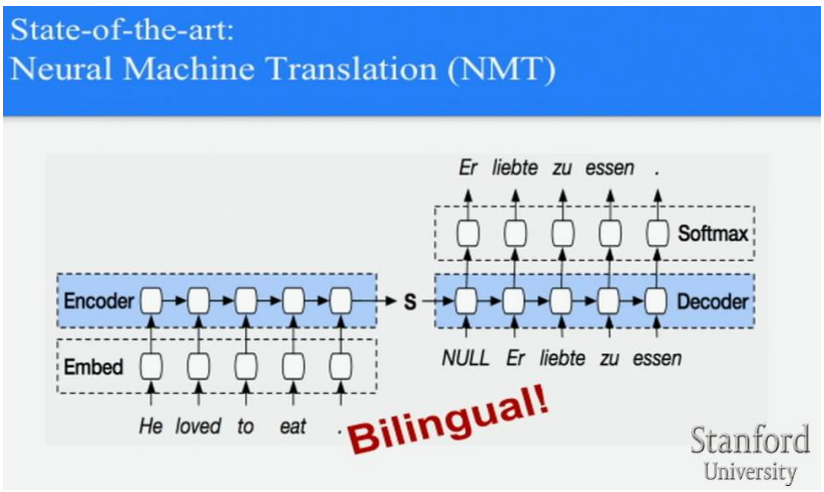


K 是一个超参数，需要在 validation 上选

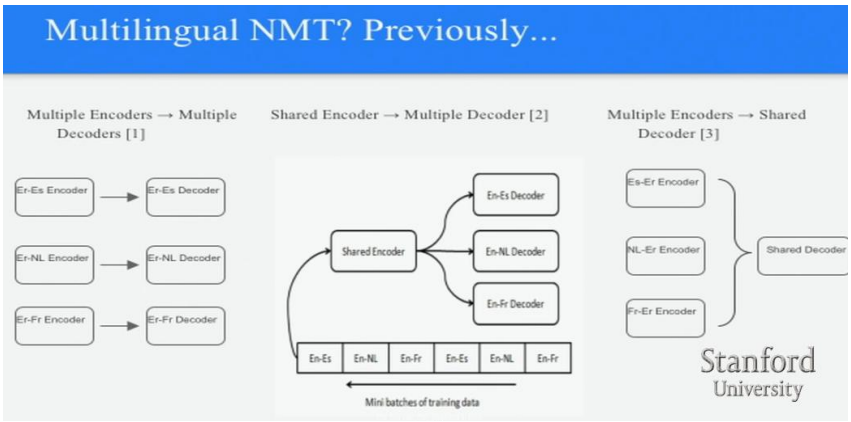
通常 **K 越大效果越好，越有可能找到最佳的 hypothesis**

google 多语言翻译模型论文





这种结构只能翻译双语，不能多种语言一起翻译



以上是之前的翻译模型：

第二是一个 encoder 一种语言，多个 decoder 翻译成多种语言

第三 多个 source language 一种 target language

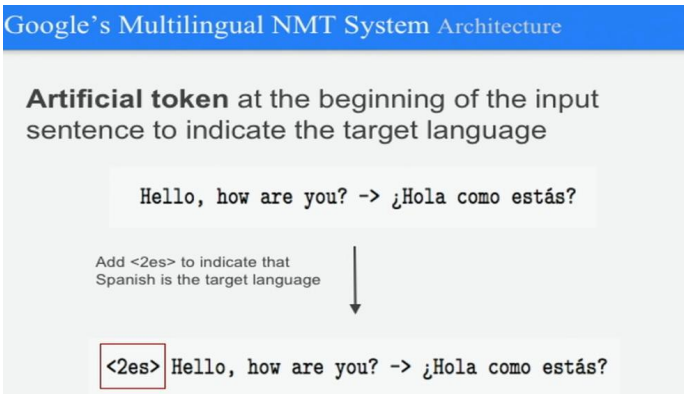
多语言模型需要训练时间很长，需要训练多个 model

多语言翻译模型：

先 train single model

然后对于 data 少的语言进行迁移

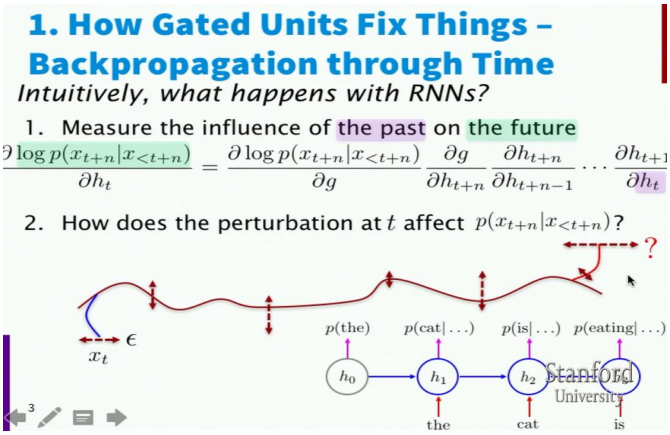
zero-shot **:可以翻译 language pair(没有出现在 train data 里)**



并且在 input 前加一个 token 标记，代表想要翻译成什么语言

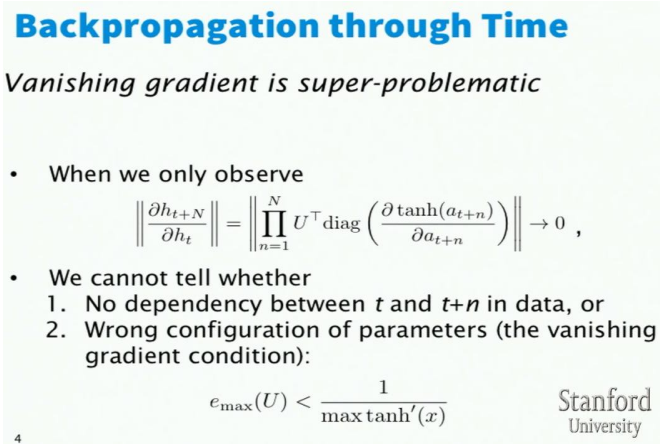
11 Gated Recurrent Units and Further Topics in NMT

Gated Recurrent Units 的本质！！！



我们想要 input t affect n time step 以后的！！

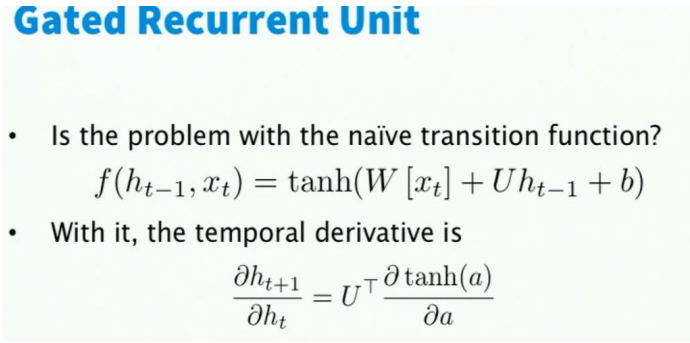
influence of the past on future！

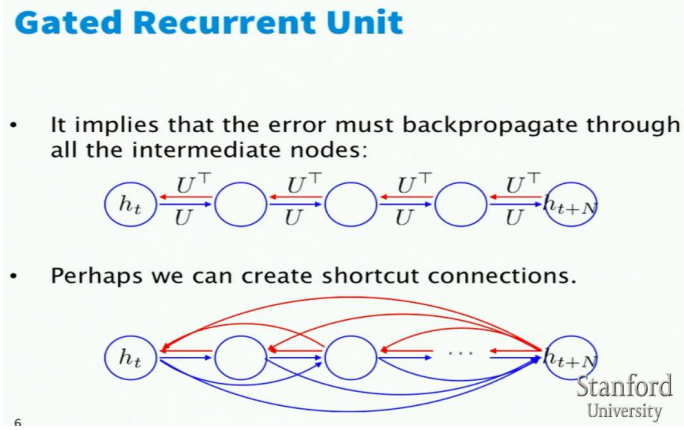


每一次 hidden state 都乘以一个 MAtrix,到下一个 hidden sate….

如果 gradient vanishing 后面的 step 的 error 没办法更新到前面的参数，也就是前面的 input 对后面 time step 没有影响！

gradient=0 是的 past 对 future 没有了影响，因为没办法后向传播更新参数了





Gated Recurrent Units 本质：**shortcut connections**

我们可以增加 early state 到 late state 的连接，**避免了 long sequence matrix 乘积**

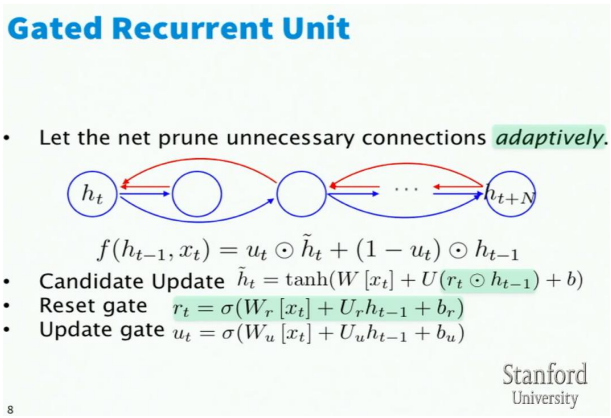
可以缓解解决 vanishing，**back propogation 可以直接传递到 early state，也就是 early state 可以影响 late state！！！！**

GRU：

如果 u gate=0 就不在乘以矩阵，而是直接将 ht-1 传递，是 linear 关系 gradient=1

这就是 GRU 的思想，如果需要的话可以多个 time step 不乘以矩阵！！

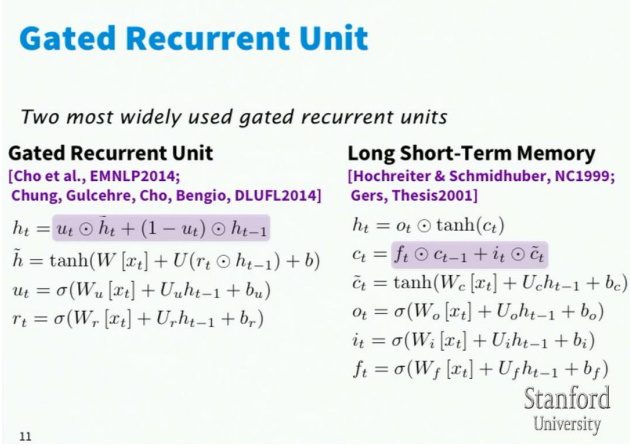
这些 gate 都是让 model 自己学习！！



再加一个 reset gate forget，忘记过去

两次对过去记忆的操作，首先筛选，再和 tilde 融合

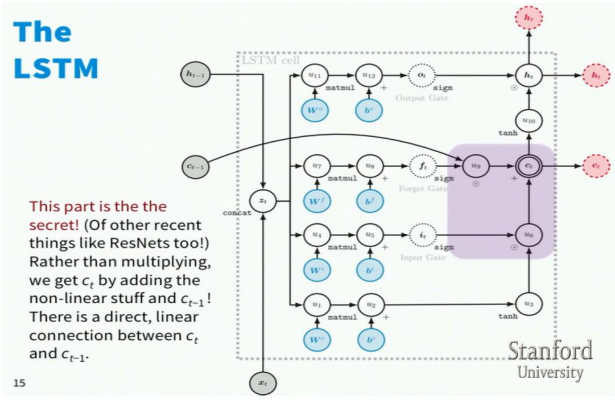
普通 RNN



lstm 只是多了一个独立的 **input gate** i

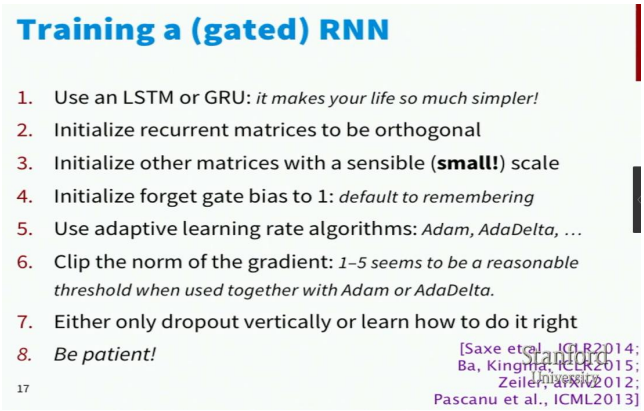
还有就是最后激活值的选择 output gate

为什么最后要再激活下 tanh(c)？因为 ct 的计算是线性的，再增加一个非线性更加 powerful



此时 ct 和 ct-1 可能是**线性的**，也就是 copy c t-1 gradient=1，可以跨越多个 time step

这种思想也在 ResNet



1 weight matrix orthogonal

2 small

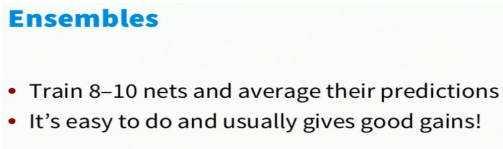
initialization 最重要！！！

3 forget gate set=1

4 clip exploding gradient to 1-5

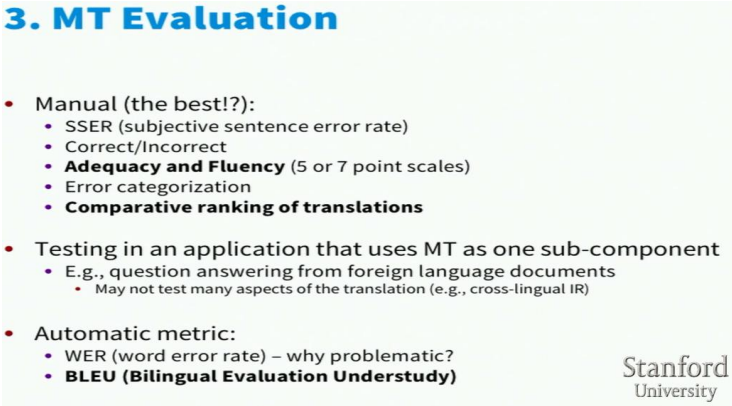
5 drop out 只用在垂直方向，而不是水平方向(容易 forget)！！

Ensemble:



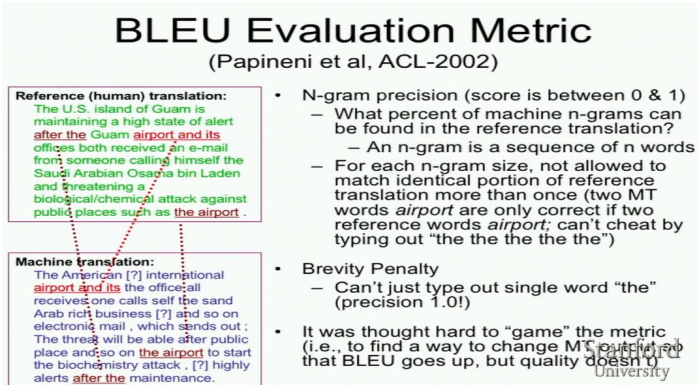
会提高准确率 2%　train 多个 model

MT Evaluation



同一句话给不同人有不同的翻译
而且不同的场合需要不同的翻译
如果评价翻译？
1 manual 让人去判断，有时很难判断量化，很慢
2 WER　BLEU

BLUE



让人先提供一个**标准的翻译 reference**
看机器翻译的 **word** 有多少出现在 **标准翻译里 phrase** **frequency**
不能考虑 order，因为语言太灵活，order 经常不同

n-gram

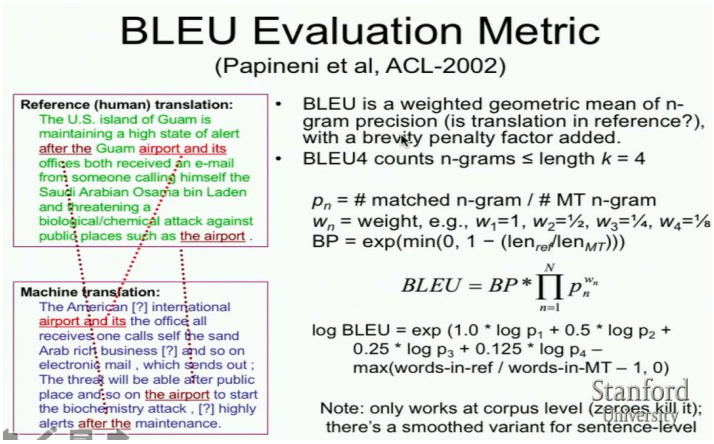
也就是机器翻译 n 个单词的 phrase 的出现在标准翻译里的频率，计算 precision

brevity penalty　　BP

如果只输出 the，precision=1 就不对了

少于标准翻译的字数，就惩罚了

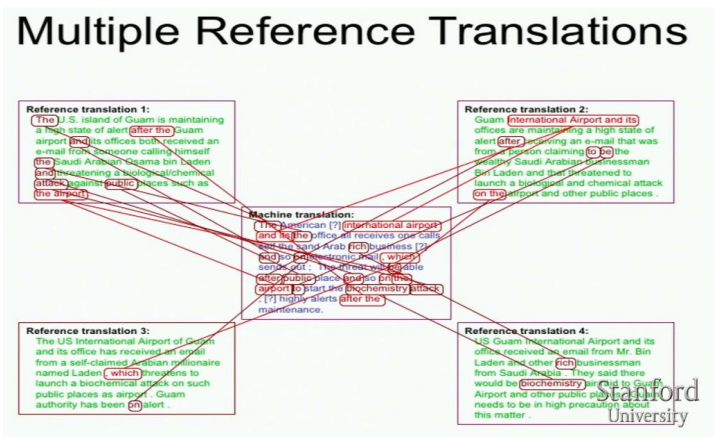
question 具体如何计算：



计算出所有 **weighted precision**

p：matched n-gram 数量/机器翻译的 n-gram 总数量

power 是权重

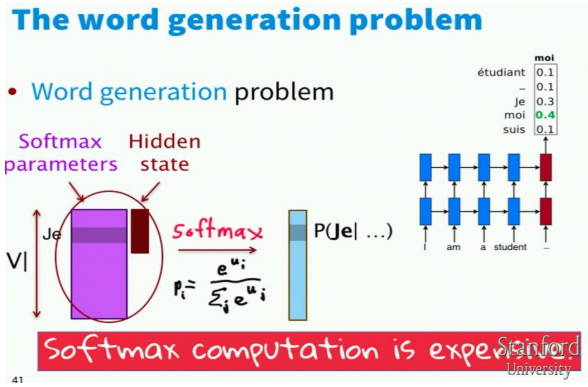


为了更加准确，提供多个标准的翻译

一起计算 precision

但 BLUE 更多用一个 reference，更快

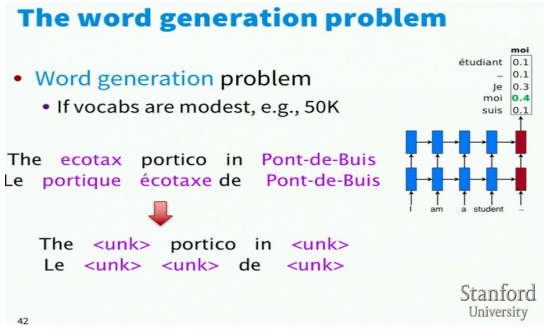
word generation problem(softmax 计算量太大！)



太多的 words

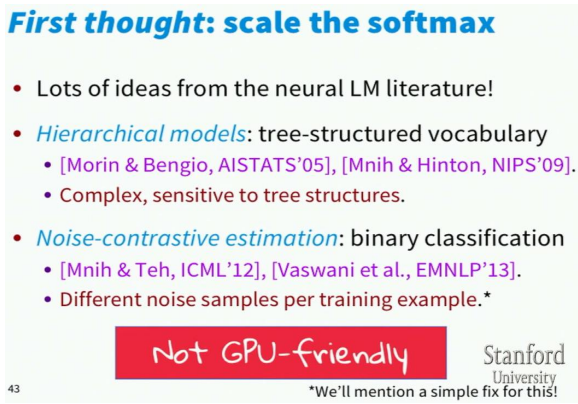
size of vocab |V|

softmax 计算量太大



1 small vocab 会有 word generation 问题

test 时很多不认识的词 不合适

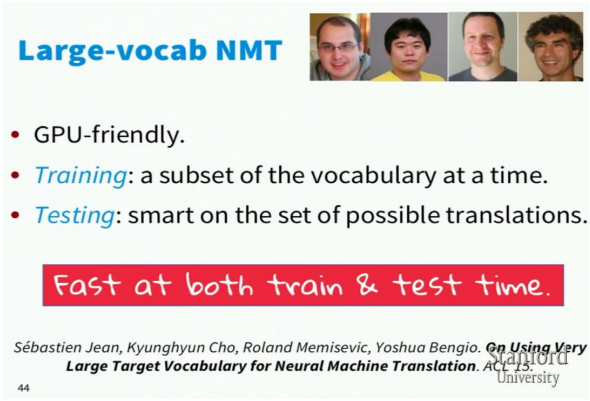


1 分层 softmax model

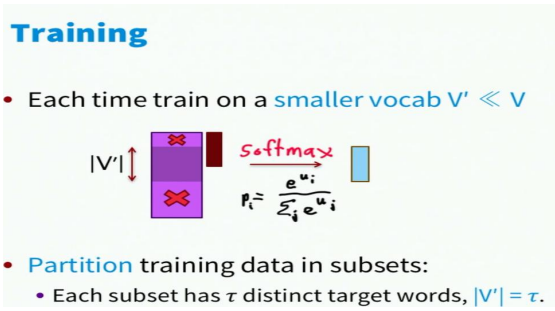
2 noise

但都不很适合在 GPU 上跑

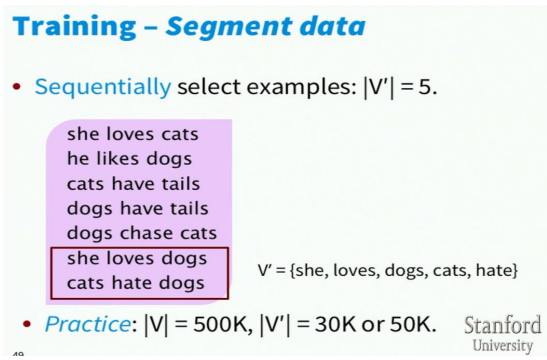
对于 **large vocab 的 MT：**



也就是将 **vocab partition，每次只训练一部分，使得 onehot 维度下降**



Train：

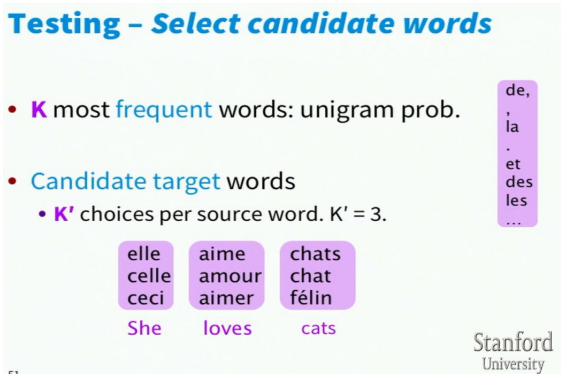


如何得到词汇？

可以将 **article 随机分割**得到不同部分的词汇，会有重复

然后 train，使得 softmax 计算量减少很多

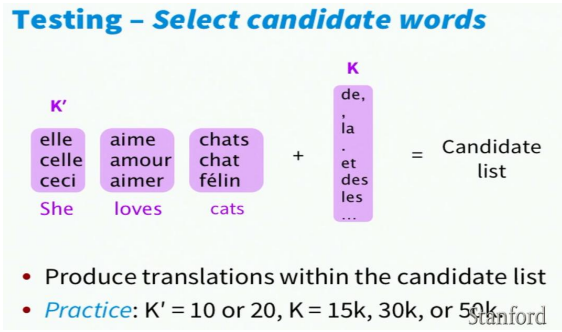
Test：



英语-->法语

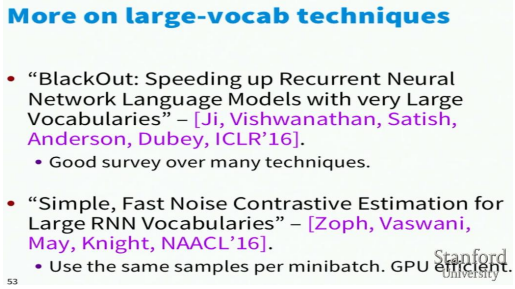
test 先找到**法语最常见的 function word** **k most**

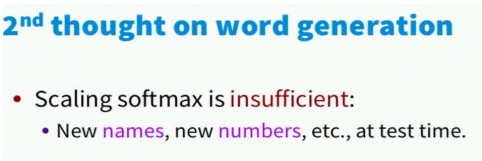
每个 source，有 k'个 candidate word(翻译的备选法语)



从 candidate list 的词汇里，得到翻译

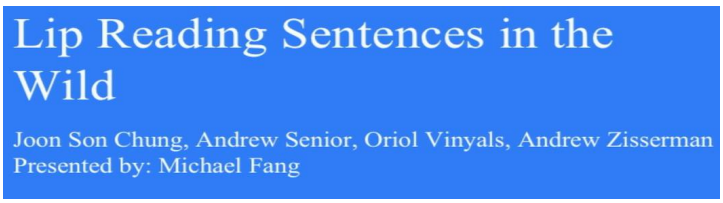
只对这些 word 进行 **softmax 计算，得到最佳的句子**



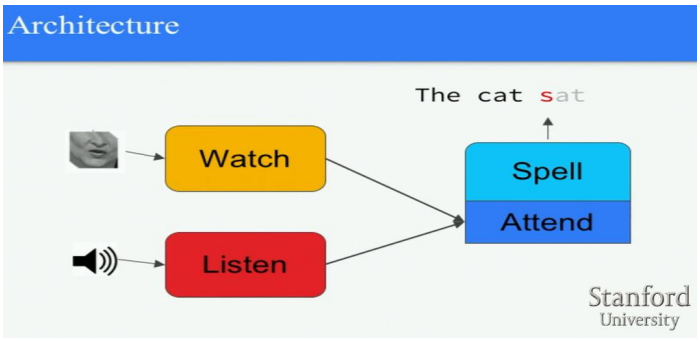


这样还不够，因为在 test 会不断创造新的词汇

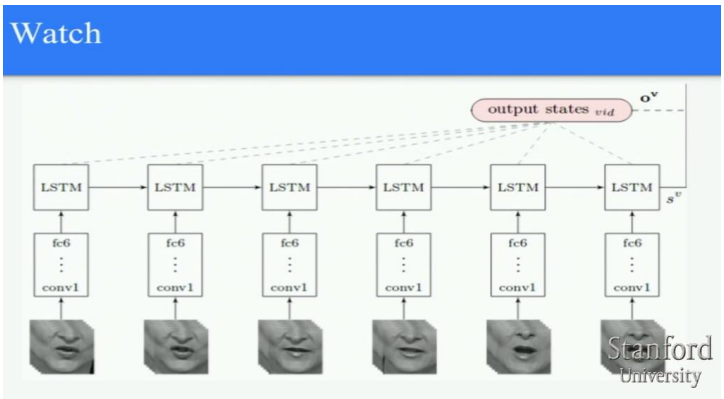
需要不断的跟新,这个也要解决



通过唇语视频+audio 来预测 text 会比单纯的语音识别很准的预测字幕

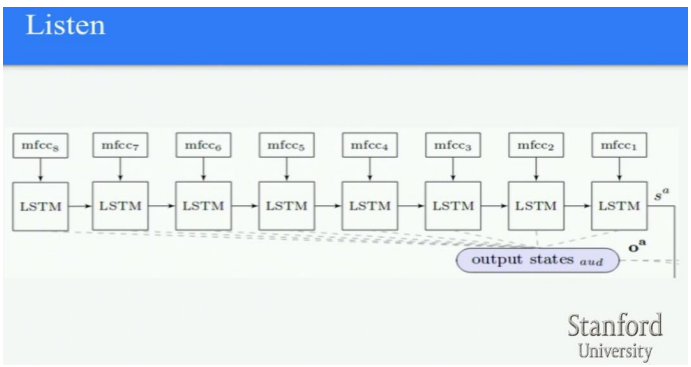


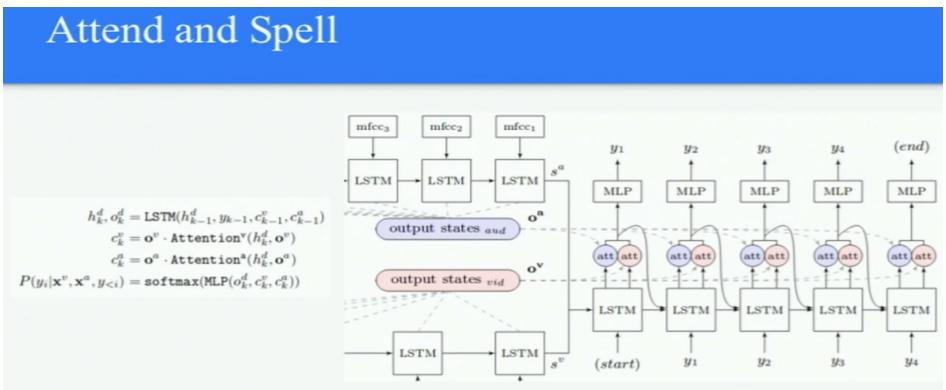
右边的 model 是 character 级别的模型



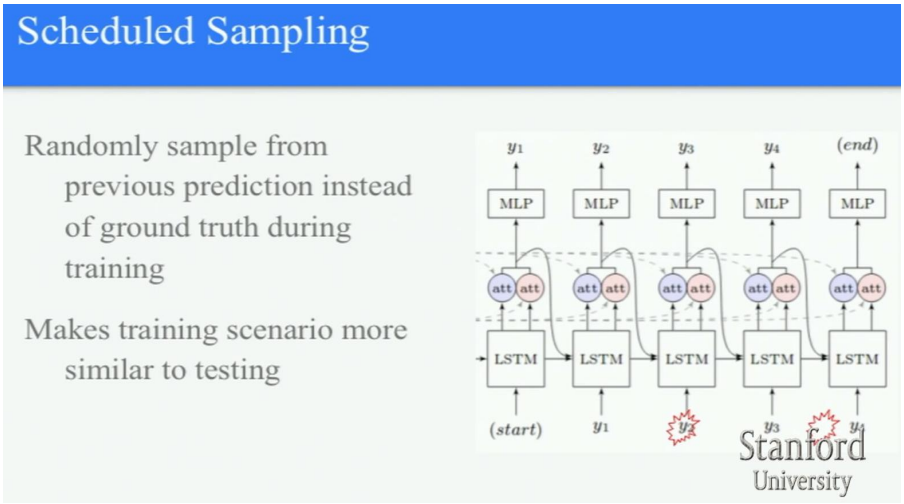
slide window CNN　　fc 是 fully connected layer

每次输出一个 vector，最后一个 state 是 ov



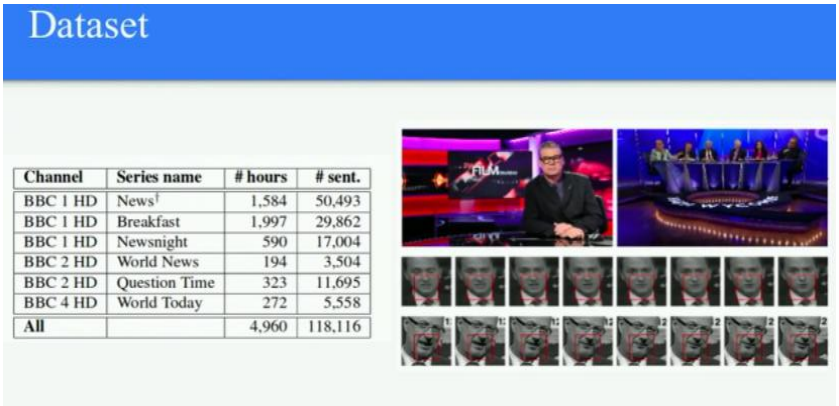


左边就是先 encoding 右边是双重 attention

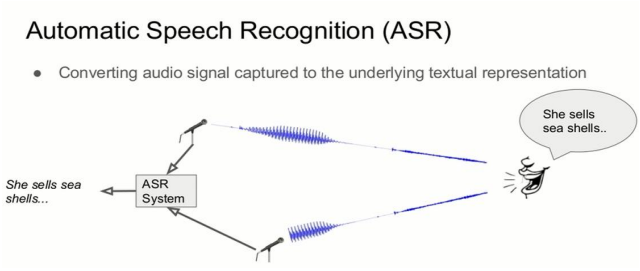


train 和 test 不同

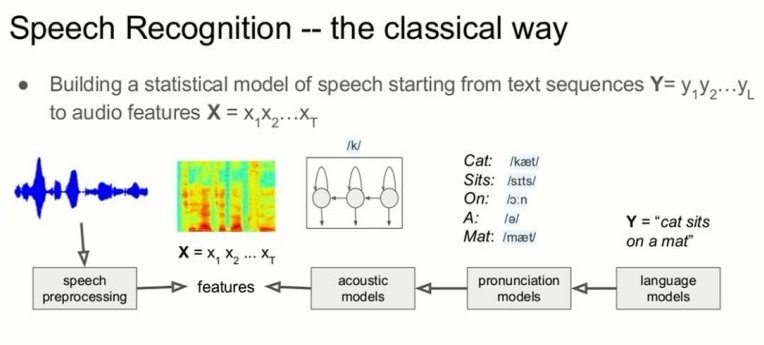
train 是有 label　　test 是自己生成



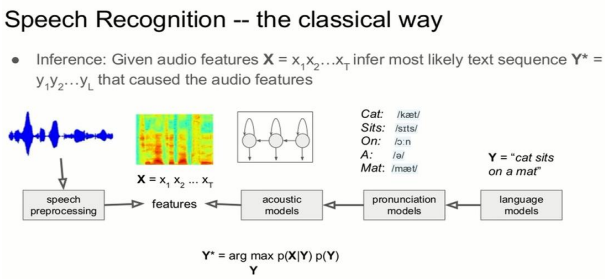
12 End-to-End Models for Speech Processing（待补充）



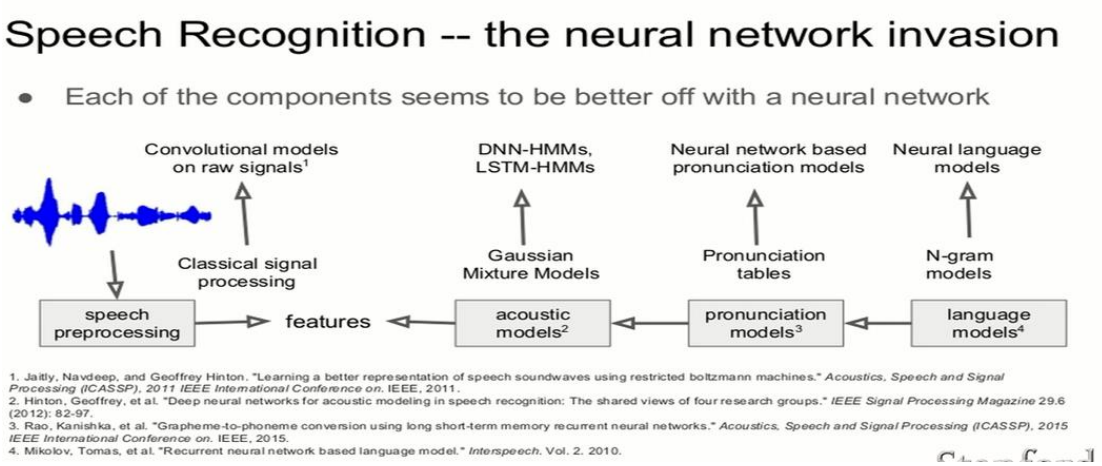
ASR 将语音转成信号，然后推断文字



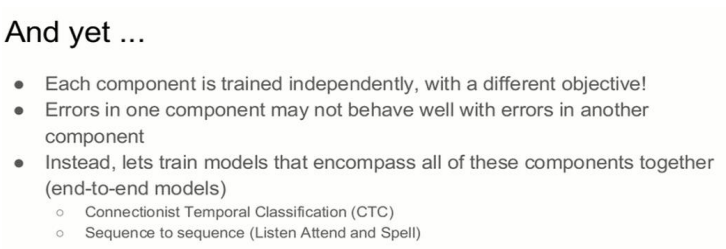
language model 将每个 word 转成**发音 token　发音 model**
声音 model，然后转换为声音 token，每个 word **发音的 element(高斯混合模型)**
得到 feature，然后频域分析



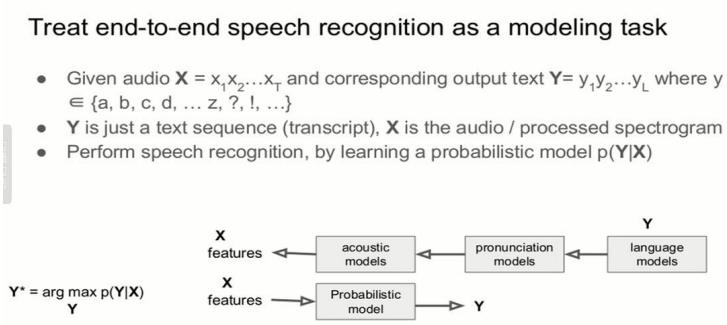
我们想选出使得 P(Y|X)最大的 Y
P(Y|X) 比例 P(X|Y)P(Y)
P(Y)是 language model



每个步骤都可以被替换！！



如果每个网络分开训练，目标函数不同，会导致最终的全部加在一起效果不好
我们希望一起训练，同一个目标函数！！end-to-end



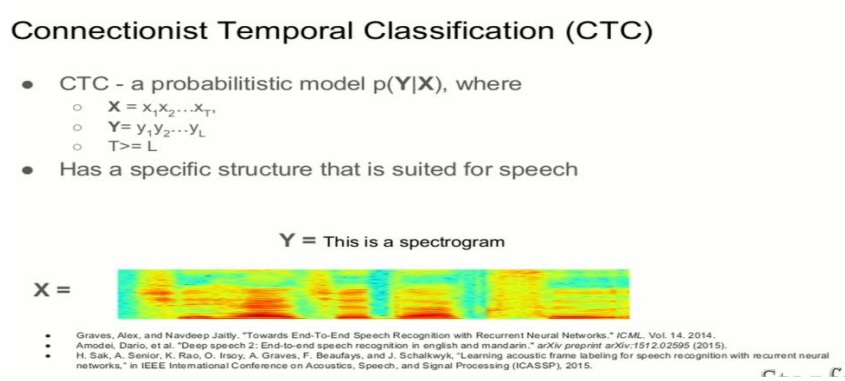
X 是语音，可能已经提过频域特征　Y 是 transcript TOKEN

learn p(Y|X)

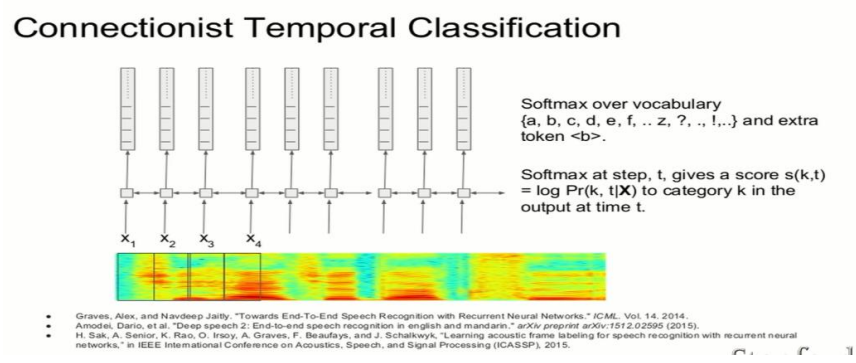
先 language model-发音 model—声音 model　得到 X feature

然后带入 p(Y|X)模型 得到 Y

p(Y|X):



X 是**提取了 频域特征　还要消除话筒的 bias(距离人远近)**

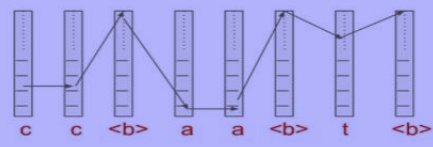


双向的 RNN，每个 hidden state 都需要全部的 input data，每个 softmax 输出的 token(score) P(k,t|X) t 时刻**输出第 k 个类的 token 的概率**，给定全部 data X

<b>是空格

CTC - How frame predictions map to output sequences

- Repeated tokens are deduplicated
  - cc <b> aa <b> t <b>
- Any original transcript, maps to all possible paths in the duplicated space:
  - cc<b>aa<b>t<b> maps to cat
  - cc<b><b>a<b>t<b> maps to cat
  - ccccc<b>aaaaa<b>ttttt<b> maps to cat
  - ccccc<b>aaaaa<b>ttttt<b> maps to cat
- The score (log probability) of any path is the sum of the scores of individual categories at the different time steps
- The probability of any transcript is the sum of probabilities of all paths that correspond to that transcript



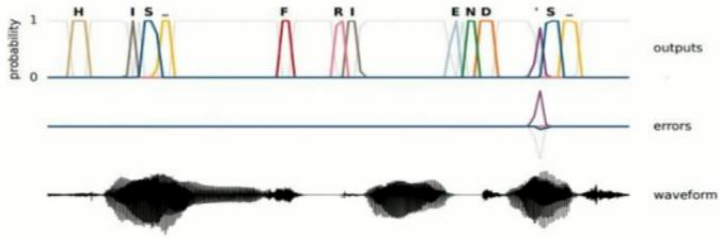
Because of dynamic programming, it is possible to compute both the log probability p(Y|X) and its gradient exactly! This gradient can be propagated to neural network whose parameters can then be adjusted by your favorite optimizer!

每一次**只能输出该字符或者 blank 字符，不会 feed previous output，因此每次 output 独立！！**

最上面一行红色是 **cat**

后面几行也是 cat

Connectionist Temporal Classification



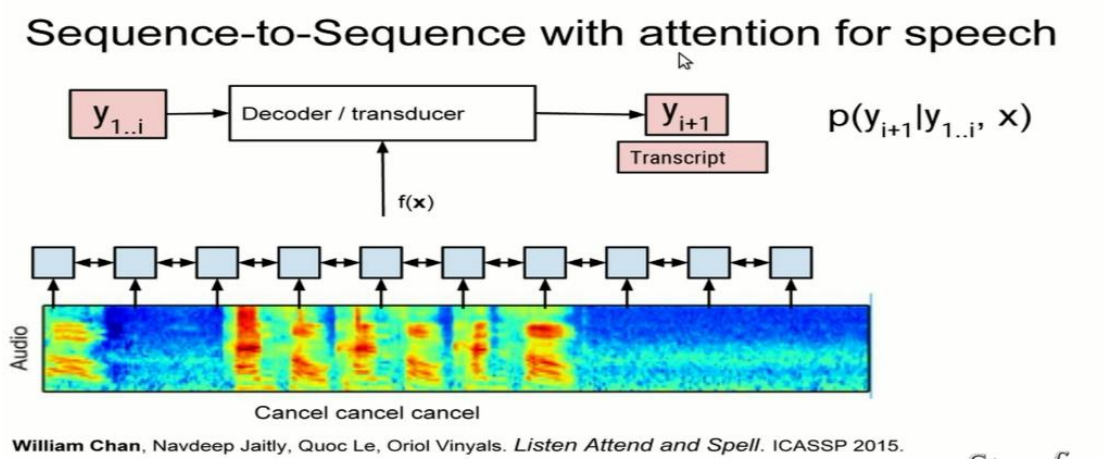
Model learns to make peaky predictions!

CTC - Language Models

- Previous transcripts sounded correct, but clearly lacked the correct spelling and grammar
  - More training data can help, but eventually, a language model is required to fix these problems
  - With a simple language model rescoring, word error rate (WER) goes from 30.1% to 8.7%

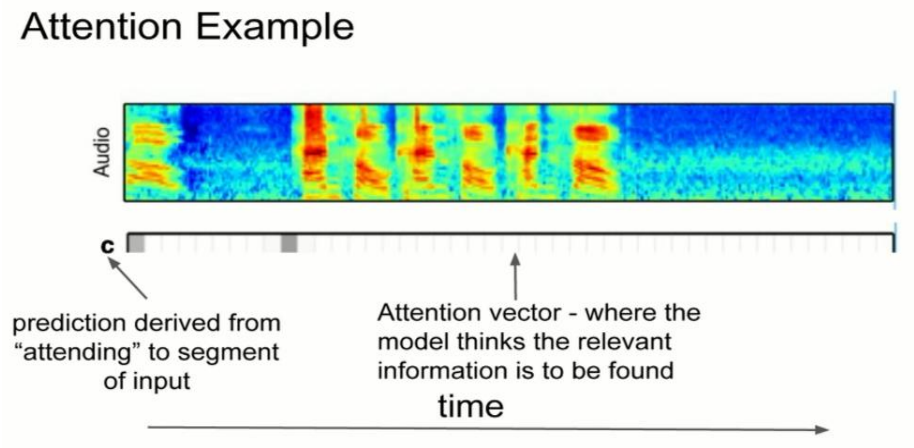
- Google's CTC implementation fixes these problems by integrating a language model into CTC during training

H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2015.

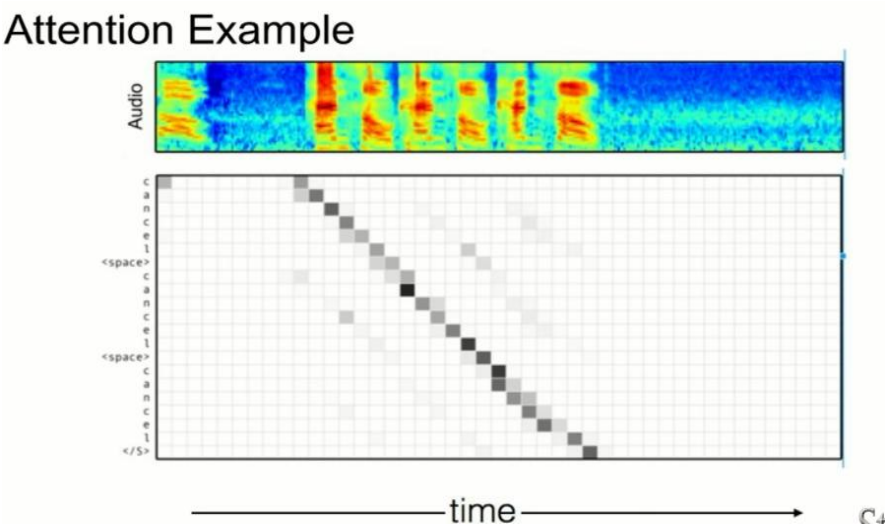


给定 X 和前面 y 来预测 下个 y

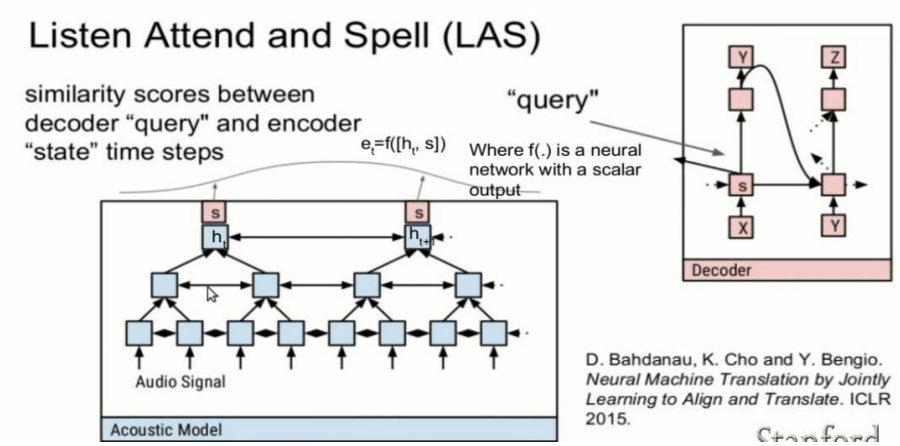
因为是 RNN decoder 所有需要 1 y1..i (上一个的输出 y) 2 所有 input X (encoder hidden state)



audio 越长效果越差，需要 attention



可以看到每个输出 y pay 的 attention

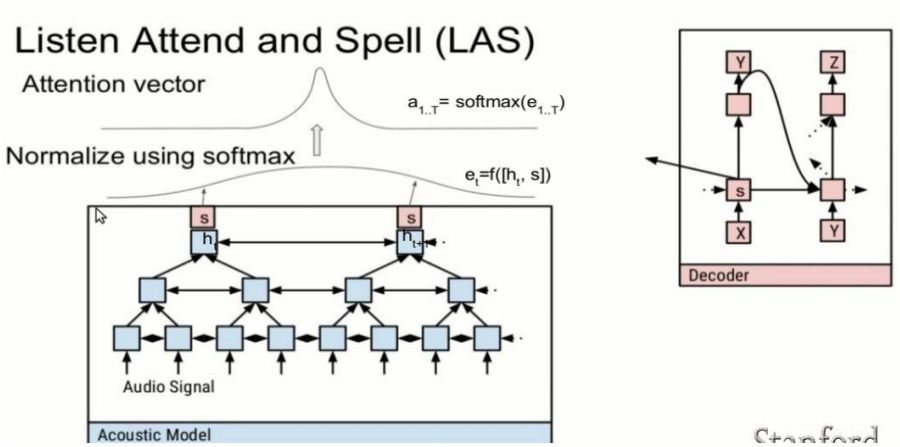


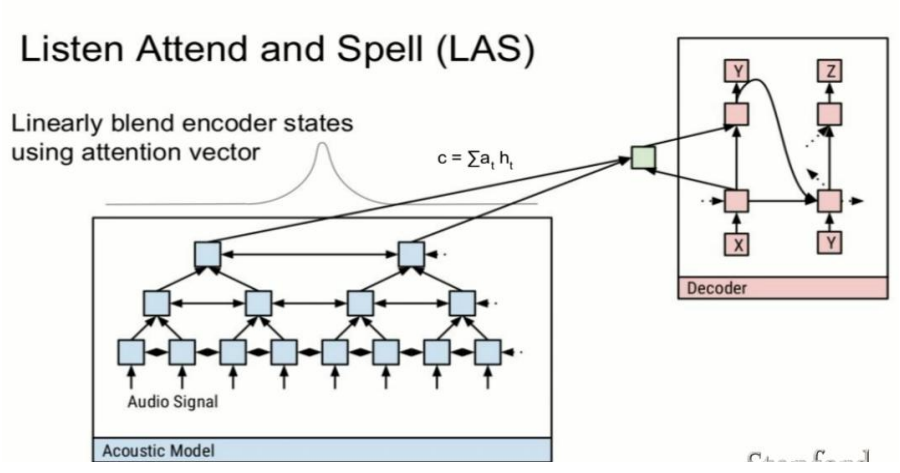
左边是 encoder，得到 hidden state vector　得到 h

decoer 是 hidden state vector 是 s

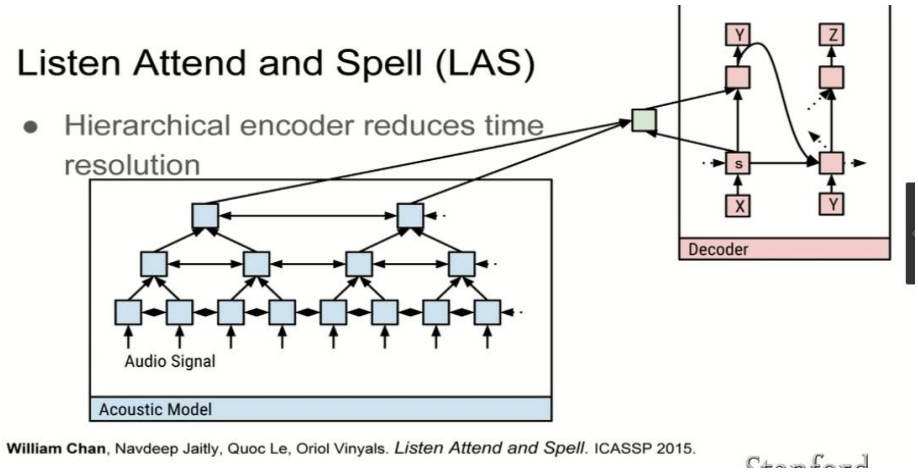
用 s 更前面的 encoder 的全部 h 比较 query，**类似于 attention**

f(ht，s)对每个 ht 得到一个 scalar　权重





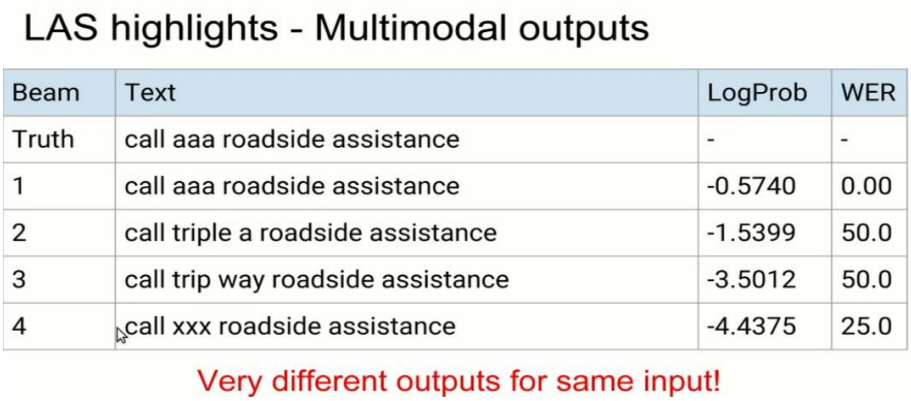
根据权重计算 attention vector c　给 decoder 做 prediction



sequence 很长，效果不好，即使用了 attention，每一次 attention 需要回顾很多的 sequence

层次结构 encoder 特殊结构

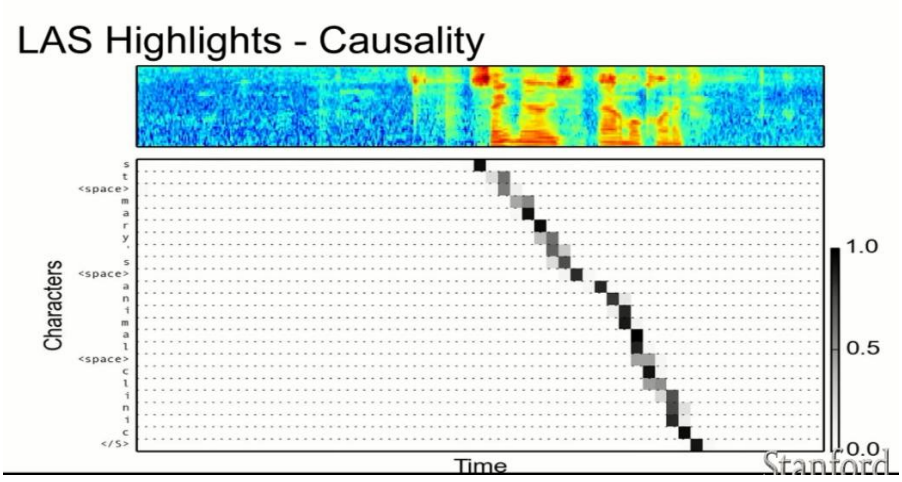
多了几个上层结构，使得 attention 时，回顾的 sequence 更少



Beam	Text	LogProb	WER
Truth	call aaa roadside assistance	-	-
1	call aaa roadside assistance	-0.5740	0.00
2	call triple a roadside assistance	-1.5399	50.0
3	call trip way roadside assistance	-3.5012	50.0
4	call xxx roadside assistance	-4.4375	25.0

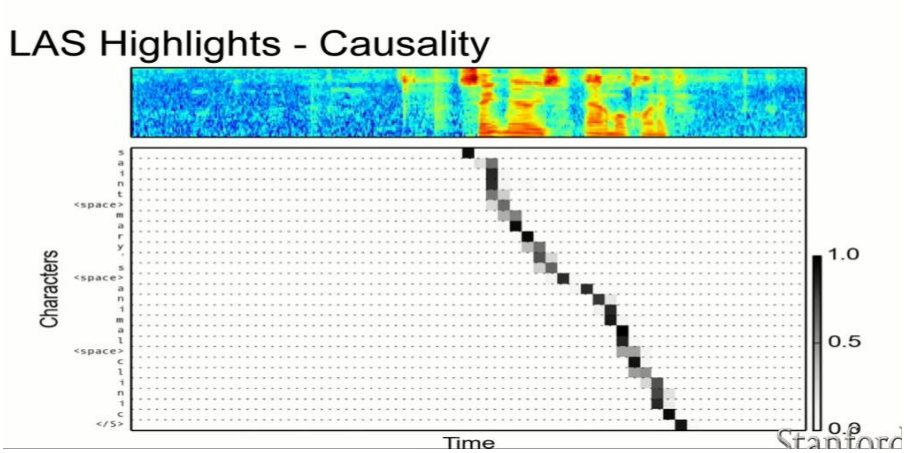
会有很多**输出版本 multimodal**

因为 train data 里出现了，model 听到了



每个 predition 的 attention

st



saint 和 st 都是听一个声音
causality 因果关系 可以从 st 推理出 saint

Limitations of LAS (seq2seq)

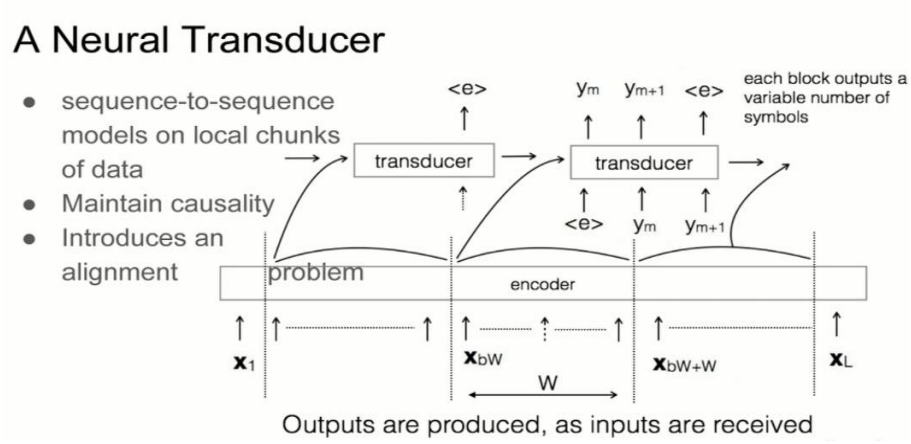
- Not an online model - input must all be received before transcripts can be produced
- Attention is a computational bottleneck since every output token pays attention to every input time step
- Length of input has a big impact on accuracy

需要 all X，不能实时

Online Sequence to Sequence Models

- Overcome limitations of the sequence to sequence models
  - No need to wait for the entire input sequence to arrive
  - Attention over the entire sequence is an overkill (attention is very locally peaked)
- Produce outputs as inputs arrive
  - Has to solve the following problem:
    - When has enough information arrived that the model is confident enough to output symbols

attention 也不用对全部 sequence pay attention

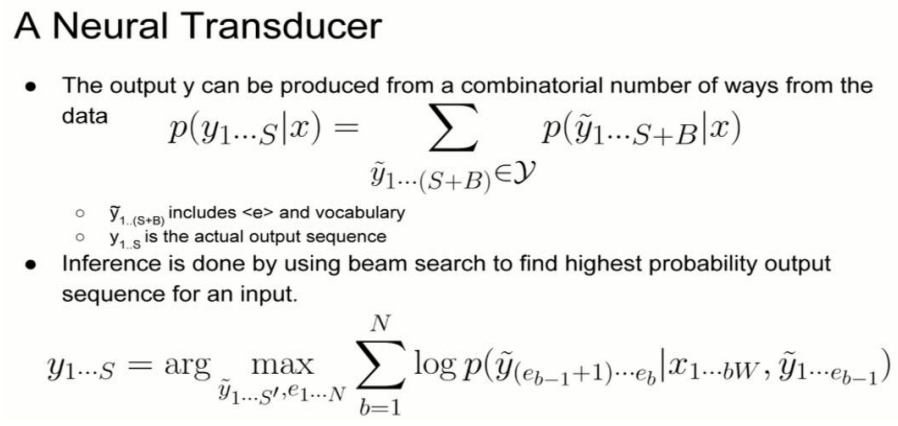


decoder 是 transducer 只关注一段 X

有时候听到很多东西，但是没有说话，没有 word，因此需要 symbol <e> blank

仍然 maintain causality

而且每次 output feed to procduce next output！！每次 output 不再独立！



y 从 1- S+B 是输出的词汇+<e>

y 从 1-S 是实际输出

每个 y 从 1-S 可能含有多个 y 从 1- S+B 组合，所有组合 sum

得到 p(y|x)

为了使得 **max p(y|x)**

每次 output 不再独立！不能分解，不能用动态规划？？question

然后 inference：

用 bean search

A Neural Transducer - Training

- Correct gradient of log likehood:

$$\sum_{\tilde{y}\in\mathcal{Y}} p\left(\tilde{y}_{1\cdots(S+B))}|\mathbf{x}_{1\cdots L}, y_{1\cdots S}\right) \frac{d}{d\theta}\log p\left(\tilde{y}_{1\cdots(S+B)}|\mathbf{x}_{1\cdots L}\right)$$

- Viterbi-like training works well in practice:

$$\frac{d}{d\theta}\log p\left(\tilde{y}_{1\cdots(S+B)}|\mathbf{x}_{1\cdots L}\right)$$

(forced alignment with a DNN-HMM system works well too!)

$$\tilde{y}_{1\cdots(S+B)} = \arg\max_{\hat{y}_{1\cdots(S+B)}} p\left(\hat{y}_{1\cdots(S+B))}|\mathbf{x}_{1\cdots L}, y_{1\cdots S}\right)$$

A Neural Transducer - Training

- Correct gradient of log likehood:

$$\sum_{\tilde{y}\in\mathcal{Y}} p\left(\tilde{y}_{1\cdots(S+B))}|\mathbf{x}_{1\cdots L}, y_{1\cdots S}\right) \frac{d}{d\theta}\log p\left(\tilde{y}_{1\cdots(S+B)}|\mathbf{x}_{1\cdots L}\right)$$

- Viterbi-like training works well in practice:

$$\frac{d}{d\theta}\log p\left(\tilde{y}_{1\cdots(S+B)}|\mathbf{x}_{1\cdots L}\right)$$

(forced alignment with a DNN-HMM system works well too!)

$$\tilde{y}_{1\cdots(S+B)} = \arg\max_{\hat{y}_{1\cdots(S+B)}} p\left(\hat{y}_{1\cdots(S+B))}|\mathbf{x}_{1\cdots L}, y_{1\cdots S}\right)$$

A Neural Transducer - Finding best path

- Finding best path is tricky -- beam search fails easily

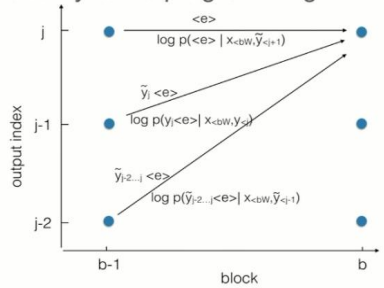
$$\tilde{y}_{1\cdots(S+B)} = \arg\max_{\hat{y}_{1\cdots(S+B)}} p\left(\hat{y}_{1\cdots(S+B))}|\mathbf{x}_{1\cdots L}, y_{1\cdots S}\right)$$

- Approximate Dynamic programming works well

beansearch：每次全部 word 作为 canditate 选 top5 然后再….
容易失败
近似动态规划，就是递归算法

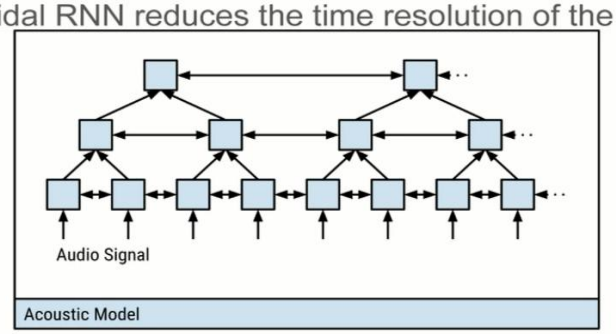
A Neural Transducer - Dynamic programming

- Approximate Dynamic programming -- finding best alignment



Very Deep Convolutional Encoders for LAS

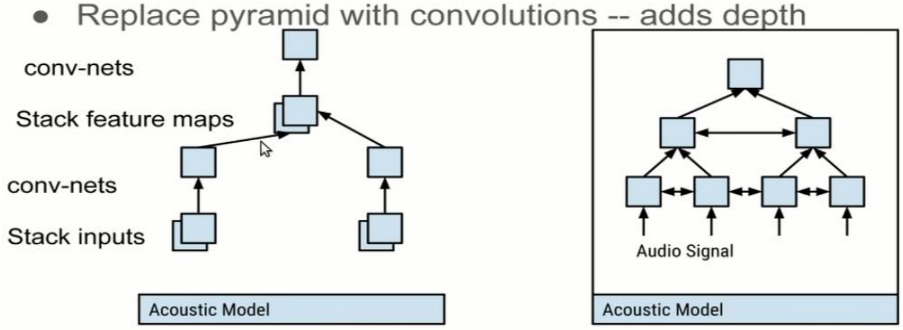
- Pyramidal RNN reduces the time resolution of the inputs



Yu Zhang, William Chan, Navdeep Jaitly. *Very Deep Convolutional Networks for End-to-End Speech Recognition*. https://arxiv.org/abs/1610.03022.

Very Deep Convolutional Encoders for LAS

- Replace pyramid with convolutions -- adds depth



Yu Zhang, William Chan, Navdeep Jaitly. *Very Deep Convolutional Networks for End-to-End Speech Recognition*. https://arxiv.org/abs/1610.03022.

Choosing the correct output targets

- Each output sequence is represented by an unique fixed representation:
  - "HELLO CHARMANDER"
    - CHARMANDER is a rare pokemon
- Word:
  - ["HELLO", "CHARMANDER"]
    - "CHARMANDER" is a OOV/rare word -- large softmax? Overfit?
- Characters:
  - ["H", "E", "L", "L", "O", " ", "C", "H", "A", "R", "M", "A", "N", "D", "E", "R"]
    - Long decoding length
    - Listen, Attend and Spell (Chan et al., 2016; Bahdanau et al., 2016)
- Word + Characters:
  - ["HELLO", " ", "C", "H", "A", "R", "M", "A", "N", "D", "E", "R"]
    - MT: Words vocab, use characters for rare/OOV words (Thang et al., 2016)

但对于 speech，**我们用 n-gram character**

Choosing the correct output targets - Word Pieces

- Word Pieces from N-grams of Characters
  - Count n-grams from text, take top 256/512/1024 word pieces
  - Possibly better units to model speech? Closer to phonemes.
    - “th”, “he”, “in”, “ing”, “tion”, “est”
- Multiple Decompositions per Sequence
  - “HELLO”
    - [“H”, “E”, “L”, “L”, “O”]
    - [“HE”, “L”, “L”, “O”]
    - [“HE”, “LLO”]
    - [“HELL”, “O”]
    - … combinatorial # of choices ...

William Chan, Yu Zhang, Quoc Le, Navdeep Jaitly. *Latent Sequence Decompositions*. https://arxiv.org/abs/1610.03035.

HELLO 有多种**分解方式**
很难确定如何分解？learn

Latent Sequence Decompositions

- Marginalize all possible paths:

$$\log p(\mathbf{y}^*|\mathbf{x};\theta) = \log \sum_{\mathbf{z}} p(\mathbf{y}^*, \mathbf{z}|\mathbf{x};\theta)$$
$$= \log \sum_{\mathbf{z}} p(\mathbf{y}^*|\mathbf{z},\mathbf{x}) p(\mathbf{z}|\mathbf{x};\theta)$$
$$= \log \sum_{\mathbf{z}} p(\mathbf{y}^*|\mathbf{z}) p(\mathbf{z}|\mathbf{x};\theta)$$

- MLE gradient estimator:

$$= \mathbb{E}_{z \sim p(z|x;\theta), z \in \{z' : p(y^*|z')=1\}} [\nabla \log p(z|x;\theta)]$$

William Chan, Yu Zhang, Quoc Le, Navdeep Jaitly. *Latent Sequence Decompositions*. https://arxiv.org/

latent 是指 隐含的 token：例如 he ll o
z 是全部的 decoposition 的可能 sum
question

Latent Sequence Decompositions (WSJ test_eval92)*

Model	WER	WER (greedy)
CTC+character (Graves et al., 2014)	27.3	-
seq2seq+character	14.7	-
2-gram word piece	13.2	15.3
3-gram word piece	12.9	15.7
4-gram word piece	13.1	15.3
5-gram word piece	13.2	15.1

* *No language models or dictionaries*

Qualitative Results

shamrock's pretax profit from the sale was one hundred twenty five million dollars a spokeswoman said

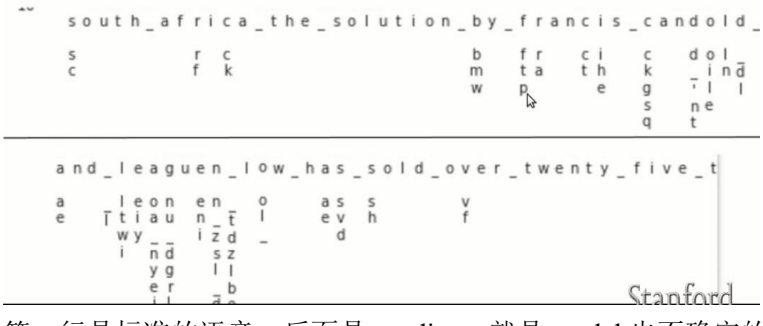
character:
c|h|a|m|r|o|c|k|'|s| |p|r|e|t|a|x| |p|r|o|f|i|t| |f|r|o|m| |t|h|e| |s|a|l|e|
|w|a|s| |o|n|e| |h|u|n|d|r|e|d| |t|w|e|n|t|y| |f|i|v|e| |m|i|l|l|i|o|n|
|d|o|l|l|a|r|s| |a| |s|p|o|k|e|s|w|o|m|a|n| |s|a|i|d -1.373868

4-gram
sh|a|m|ro|c|k|'s| |pre|ta|x| | **pro**|fi|t| |**fro**|m| |t|h|e| |sa|l|e| |was|
|on|e| |hu|n|dr|e|d| |t|we|nt|y| | **fiv**|e| |**mil**|**lio**|n| |**doll**|a|r|s| |a|
|sp|ok|e|s|wo|ma|n| |said -28.111485

- "shamrock" is in-vocab, "shamrock's" is OOV.
- Bolded parts to highlight the 3/4-grams

Model Shortcomings

- More ambiguity at start of words



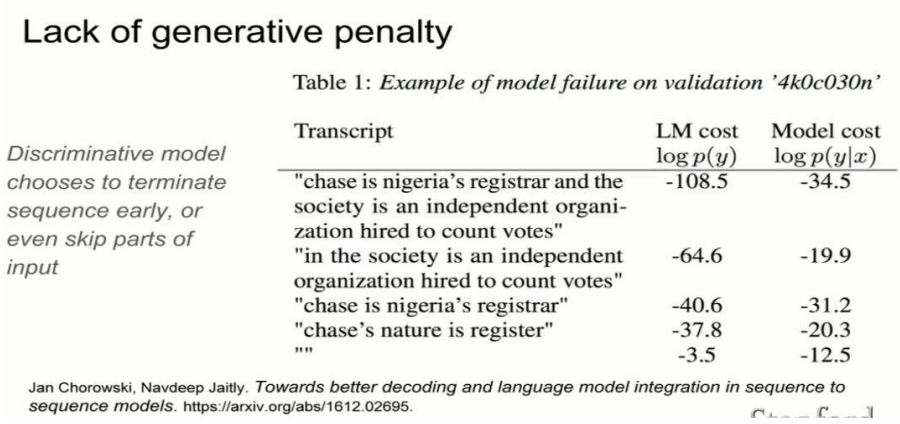
第一行是标准的语音　后面是 canditate 就是 model 也不确定的 character

Overconfidence at word boundaries

- Entropy Regularization of target distribution at each step counters overconfidence

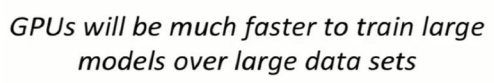
Model	Parameters	dev93	eval92
CTC [3]	26.5M	-	27.3
seq2seq [12]	5.7M	-	18.6
seq2seq [24]	5.9M	-	12.9
seq2seq [22]	-	-	10.5
Baseline	6.6M	17.9	14.2
Unigram LS	6.6M	**13.7**	**10.6**
Temporal LS	6.6M	14.1	10.7

Jan Chorowski, Navdeep Jaitly. *Towards better decoding and language model integration in sequence to sequence models*. https://arxiv.org/abs/1612.02695.

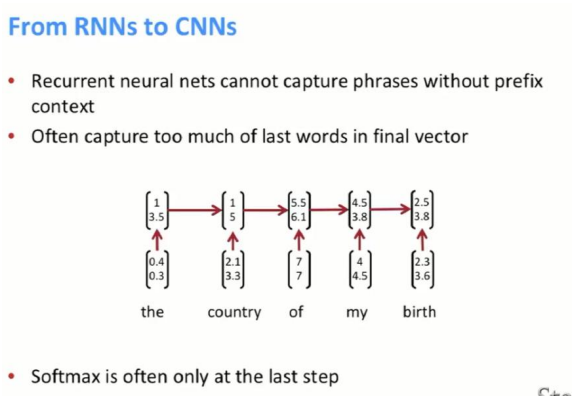


model 会选择输出少的句子
question　这节课没听懂

13 - Convolutional Neural Networks



CNN 的目的 for sentence classification



RNN **最后一个 state vector 包含了前面所有的 phrase**，但会**包含过多离的近的 words**，因为 classifier softmax output 会放在最后一个 state 上面，不容易做**句子分类(情感分类)**

From RNNs to CNNs

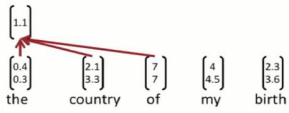
- Main CNN idea:
- What if we compute vectors for every possible phrase?
- Example: "the country of my birth" computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth
- Regardless of whether phrase is grammatical
- Not very linguistically or cognitively plausible
- Then group them afterwards (more soon)

Sta

对一个句子**穷尽全部的可能的 phrase**，可能会语法错误

Single Layer CNN

- A simple variant using one convolutional layer and **pooling**
- Based on Collobert and Weston (2011) and Kim (2014) "Convolutional Neural Networks for Sentence Classification"
- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$ (vectors concatenated)
- Concatenation of words in range: $\mathbf{x}_{i:i+j}$
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (goes over window of h words)
- Could be 2 (as before) higher, e.g. 3:



Star
Univ

CNN for 句子分类

句子是 word vector concatenate　big vector

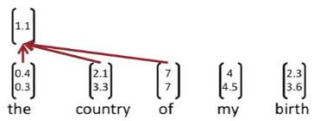
然后 filter w　长度是 h 个 word　也就 **h*k**　　**k 是 vector 长度**

例如上面　filter　长度是 3 个 word，然后 step 是一个 word

Single layer CNN

- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (goes over window of h words)
- Note, filter is vector!
- Window size h could be 2 (as before) or higher, e.g. 3:
- To compute feature for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



Stan

Xi：i+h-1　其实就是将这几个向量 concate　然后和 filter　内积，得到值 1.1

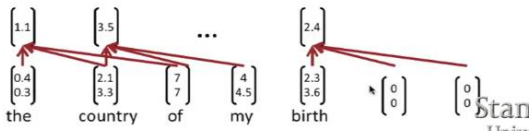
每一次 filter　都增加了 non-linearity

有个缺点是两边的 vector 参与计算的频率小，中间的 vector 计算次数多

可以 0 padding，包括对于不同长度的句子也可以 0 padding

Single layer CNN

- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



得到一系列的 feature map 值
不同句子长度的到的 feature map 维度不同

filter 的本质

Single layer CNN: Pooling layer

- New building block: Pooling
- In particular: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)
- From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$
- But we want more features!

pooling layer capture most important activation(因为对于句子分类，句子本身 word(feature) 少，通过 max pooling 保留一个 feature 对于分类影响不大，但像图片就不行了！)
也就是 ci 如果大，**那个窗口的 phrase 是更符合 filter 所要找的标准(pattern)**
其实就是 filter 向量和 word 向量的内积越大，代表越相似，也就是 phrase 和 filter 相似
也就是每句话只留一个最大的 c，每一次只关心和 filter 最吻合的 patch！！

神奇的 filter 可以不定长

Solution: Multiple filters

- Use multiple filter weights w
- Useful to have different window sizes h
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** irrelevant
  $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- So we can have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

然后有**多个 filter 捕捉不同的 feature**，这里神奇的是，**filter 的长度可以不同，长度为 k，是捕捉 k-gram phrase**
而且就算是相同的 len filter 可能学的也是不同的 phrase
1 多个 filter　2 改变 filter 长度

Muli-channel

Multi-channel idea

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies
- Backprop into only one set, keep other "static"
- Both channels are added to c_i before max-pooling

对于句子分类，可以 update word vector 也可以不 update，也可以折中，保持多个 channel
每一句话是一个 **word concate vector**，我们 copy 2 份
可以直接求两个 vector 的均值，然后再用 filter

word vector 在大量的 corpus 的条件下 backprop 比较合适，能够很好捕捉 semantic！但在这种特定任务下 backprop 会使得 vector 变形(过拟合)，也就是说在 train 的 word 变形以后，对于出现在 train 里的 vector (已改变的)分类效果好，但在 test 的没有出现在 train 的 words(没有改变)，在分类时可能会分错

因此我留 2 分 copy channels，一份用于改变 word vector ，另一份用于更好的 **generalization**
计算 c 的时候**两个 channel 相加**(中和一下效果) 然后再 **maxpooling**

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4

1 **random initial word vector and back propagation(如果 text 很大，效果好，text 小效果不好)**
2 固定的 pretrained word vector
3 不固定的 pretrained word vector
4 muti-channel 好像是最好的

Classification after one CNN layer

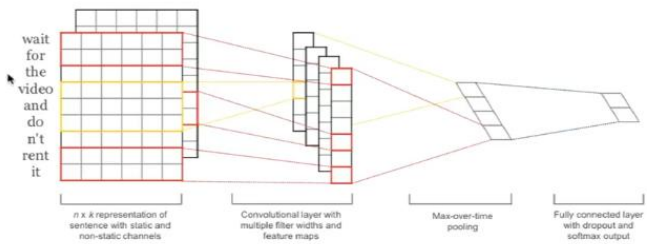
- First one convolution, followed by one max-pooling
- To obtain final feature vector: $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$
  (assuming m filters w)
- Simple final softmax layer $y = softmax\left(W^{(S)}z + b\right)$

几个 filter 就有几个 feature 元素
最后得到 maxpooling 的 c vector，给 softmax，**做分类，也就是对**句子分类

GPU-Based

每个 filter 是独立的，可以 run 在不同的 gpu 上

Figure from Kim (2014)



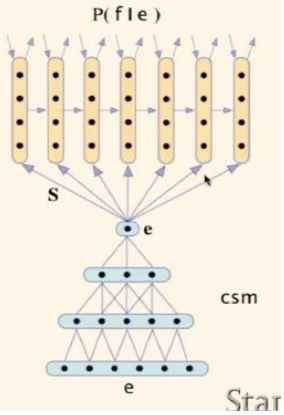
n words (possibly zero padded) and each word vector has k dimensions

上图其实跟将 vector 拼在一起实质一样

CNN application

CNN application: Translation

- One of the first successful neural machine translation efforts
- Uses CNN for encoding and RNN for decoding
- Kalchbrenner and Blunsom (2013)
  "Recurrent Continuous Translation Models"



机器翻译： CNN 作为 encoder RNN decoder

1dropout

Tricks to make it work better: Dropout

- Idea: randomly mask/dropout/set to 0 some of the feature weights z
- Create masking vector r of Bernoulli random variables with probability p (a hyperparameter) of being 1
- Delete features during training:

$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$

- Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations)
- Paper: Hinton et al. 2012: Improving neural networks by preventing co-adaptation of feature detectors

Stan
Unive

z 是我们 CNN max pooling 后的 feature vector dropout
因为 z 可能是 **bigram 或 3-gram**...可以**避免对某一个 phrase 的过拟合**
prevent co-adaptation

Tricks to make it work better: Dropout

$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$

- At training time, gradients are backpropagated only through those elements of z vector for which $r_i = 1$
- At test time, there is no dropout, so feature vectors z are larger.
- Hence, we scale final vector by Bernoulli probability p

$$\hat{W}^{(S)} = pW^{(S)}$$

- Kim (2014) reports **2 – 4% improved accuracy** and ability to use very large networks without overfitting

Star

test 时因为用全部的 z，会使得激活值变大，因此需要 scale 下，假如 drop 了 0.6，需要乘以 0.6

2 L2norm

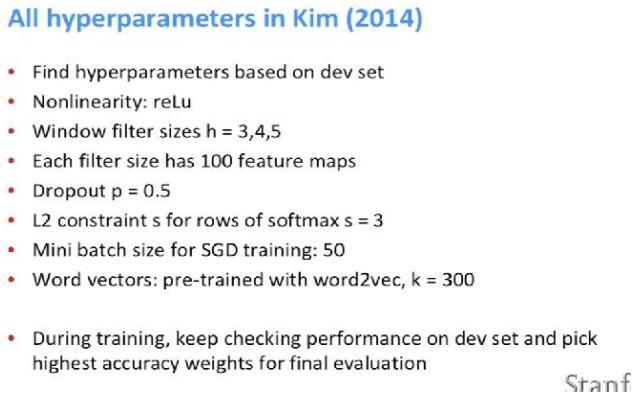
每个 class 的 weight 矩阵 L2 norm 防止过大

Another regularization trick

- Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number s (also a hyperparameter)
- If $\|W_{c\cdot}^{(S)}\| > s$, then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$
- Not very common

不是很常用！限定权重的值

所有的超参数



超参数的影响：

在有限的超参数里，**运用 validation set 来寻找最佳参数**

random parameter search 效果会更好！！！

boost：same model 5 time average result，因为不同 model 的不同的 local optimal

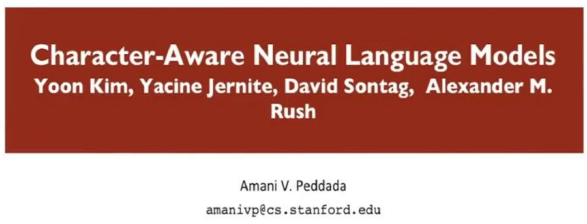
或者是选取 top5 model 进行 assemble！

K 是 embedding 维度

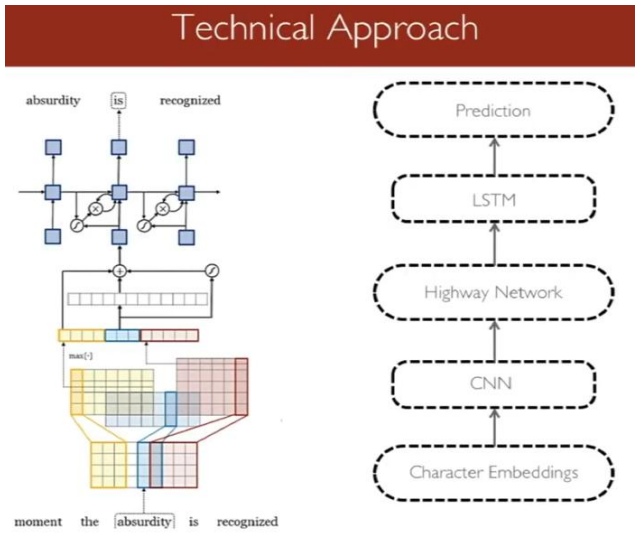
选择最好的 model，就是 keep tracking **model weight**，然后**选择时间点最好的 weight！！**

如果对于 convex 问题(**global optima**)，选择 model 参数后，需要合并 dev set 然后 retrain

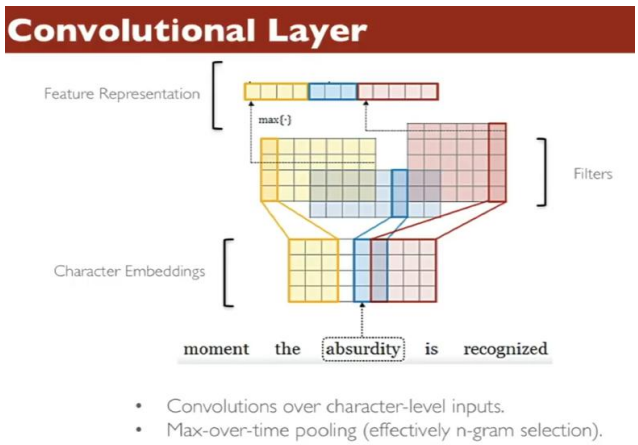
但对于 no- convex 不需要！！！



- Derive a powerful, robust language model effective across a variety of languages.
- Encode subword relatedness: *eventful, eventfully, uneventful...*
- Address rare-word problem of prior models.
- Obtain comparable expressivity with fewer parameters.



character level



没有使用 word embedding

每个 character 一个 embedding，需要被 train

然后一句话就一个 embedding 矩阵

然后用 CNN 得到 feature map

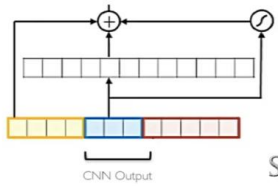
Highway Network (Srivastava et al. 2015)

- Model *n*-gram interactions.
- Apply transformation while carrying over original information.
- Functions akin to an LSTM memory cell.

$$\mathbf{t} = \sigma(\mathbf{W}_T\mathbf{y} + \mathbf{b}_T)$$

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

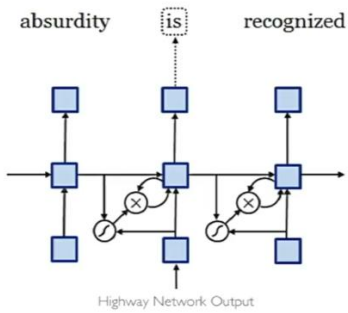
Transform Gate　Input　Carry Gate



Stanf
Univer

将 output 给 highway network 和 LSTM 相似，**carry** 原始信息+**transform 改写**或遗忘一些信息

Long Short-Term Memory Network



- Hierarchical Softmax to handle large output vocabulary.
- Trained with truncated backprop through time.

再给 lstm，**预测下个 word**

Qualitative Insights

	In Vocabulary				
	while	*his*	*you*	*richard*	*trading*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*
	letting	*her*	*we*	*robert*	*advertising*
	though	*my*	*guys*	*neil*	*turnover*
	minute	*their*	*i*	*nancy*	*turnover*
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*
	whole	*hhs*	*young*	*rich*	*training*
	meanwhile	*is*	*four*	*richer*	*reading*
	white	*has*	*youth*	*richter*	*leading*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*
	whole	*this*	*your*	*gerard*	*training*
	though	*their*	*doug*	*edward*	*traded*
	nevertheless	*your*	*i*	*carl*	*trader*

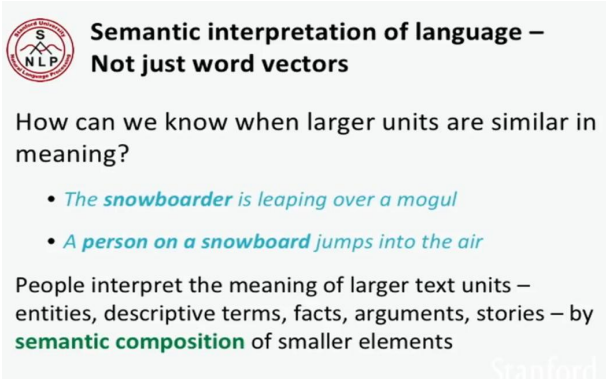
普通 LSTM 基础上通过增加 CNN，效果提升
再增加 highway 效果更好
也就是我们学习到的 **word embedding(怎么得到的)更具有 similarity**

- Questions the necessity of using word embeddings as inputs for neural language modeling.
- CNNs + Highway Network over characters can extract rich semantic and structural information.
- Key takeaway: can compose "building blocks" to obtain more nuanced or powerful models!

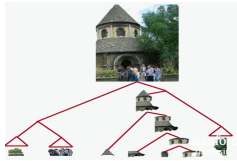
不需要 embedding，而是直接可以 learn
但就是训练太慢了

14 Tree Recursive Neural Networks and Constituency Parsing

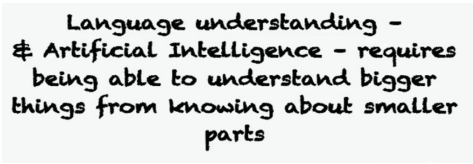
意群



句子或意群之间的距离
意群很重要　**不只是 word**
意群也是 semantic composition，也就是把**小部分**拼起来，**来理解整个部分的内容**



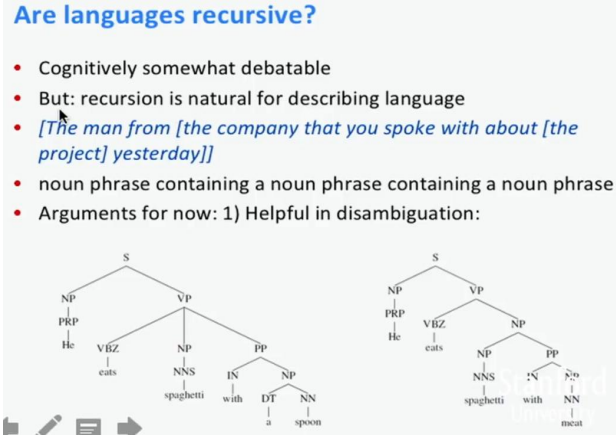
图片也是这种模式



也就阅读理解，仅停留在 word 上很难理解，需要以意群为单位进行理解

Tree RNN(recursive NN) 更好的捕捉句子(意群)的 semantic

将一些 small pieces (word vector)，按照结构 built combined representation



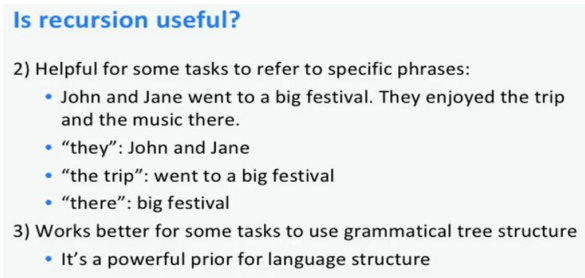
递归的解析句子意思

the project 是个 non-phrase

the company ... the project 也是个 non-phrase

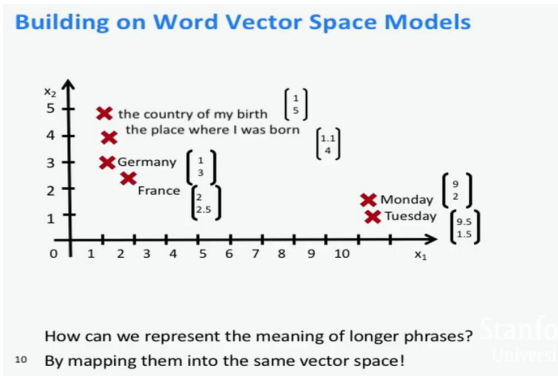
the man from 整句也是个 non-phrase

也就是先从 man 开始，然后 中间，然后 the project，最后返回得到**这句话整个意思**



对于 **refer to 句子有用**，也就是后一句会指示代词，指代第一句内容

tree RNN 对句子的结构很重视！

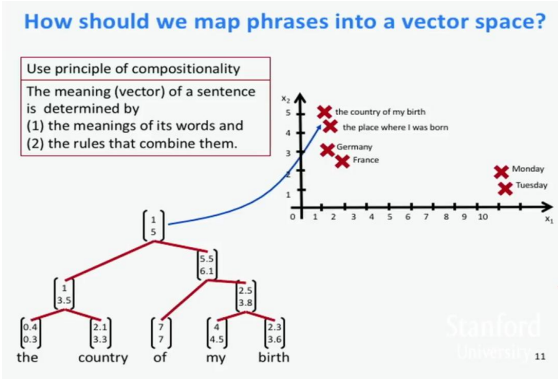


把 long phrase，也加进如我们的 word embedding(same vector space)

same vector space

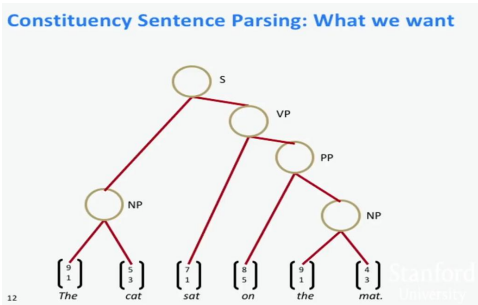
一个 word 有很多 meaning，如果只用一个 vector space，可能不能全部的表示出来，可能需要多个 vector space，所以也可以 word 和 pharase 在不同的 space

但不像 word 一样，**提前存储这些 phrase 的 embedding，太多了**



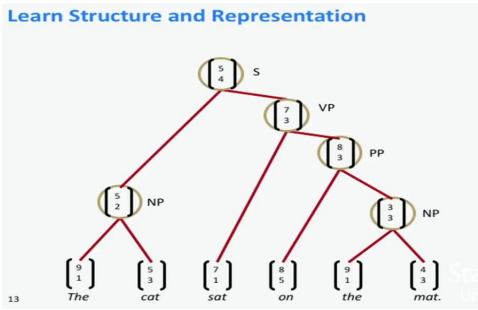
1 word embedding　2 句子结构
然后从下往上进行计算

1 sentence structure

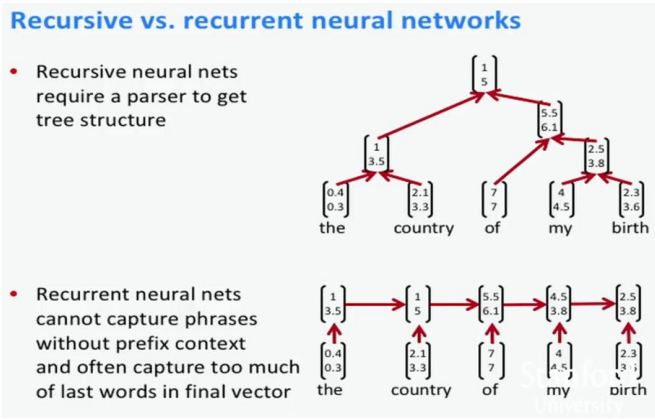


建立句子结构

2 semantic 语义　　meaning of phrase



建立语义



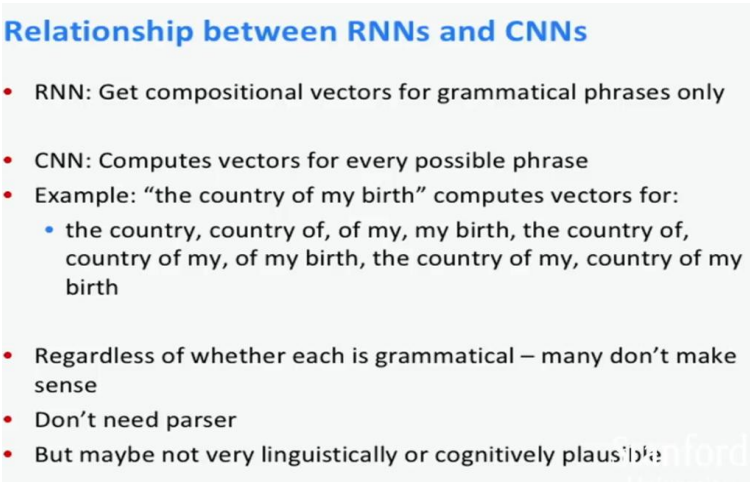
下面 RNN，其实也是个树，只是一边倒，**只有一个分支**
有些句子也是一边倒 只有一种成分，例如 NP

RNN 不能反映句子结构，但 RNN 每个 state 都包含前面**全部的**文字信息，没有小的 phrase unit 的信息，**而且最后一个 word 的信息量太大**

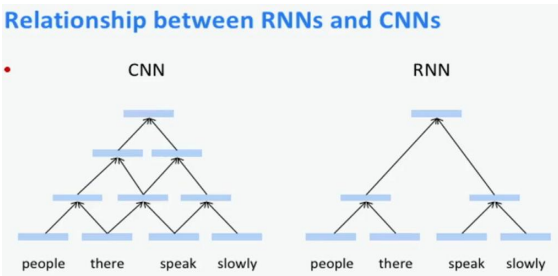
而 TRNN 是在将一些 small pieces built combined representation，也就是有些 hidden sate 包含 phrase unit 的信息

但 TRNN 需要我们得到 Tree structure(需要先解析)，分类问题。比较麻烦

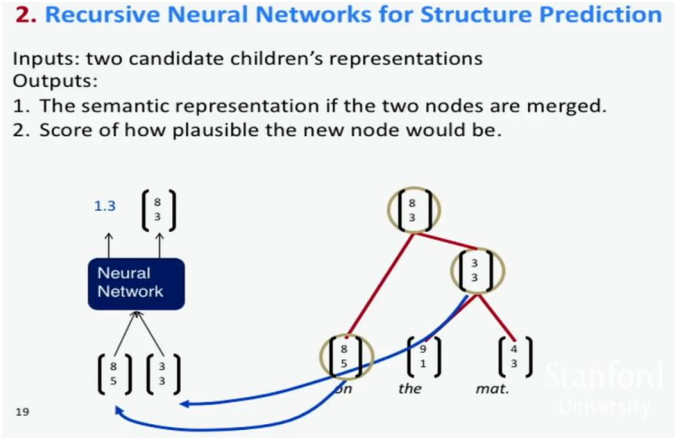
Recursive NN 和 CNN 关系



TRNN 只对有效的有意义的 phrase unit 计算
CNN，是滑动，不一定是有意义的 phrase unit，是对所有可能的 phrase，不用 parse，速度快，但不符合语言学



RecNN 先确定结构： Parsing or (words pair) structure prediction

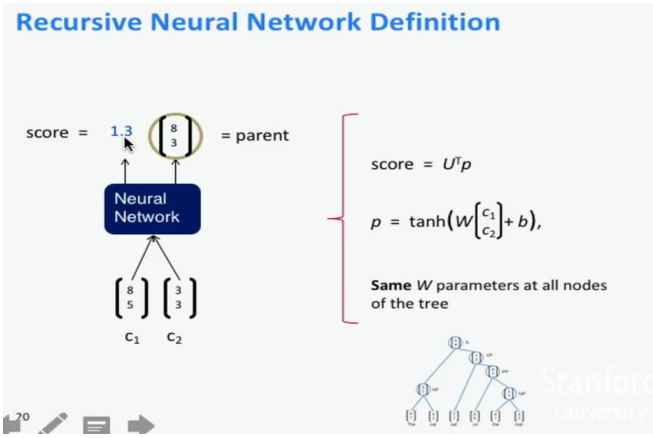


解析 structure，**从最基本底层开始往上做 classifier，predict structure：将两个子部分输入给 NN,**

output:

1 vector

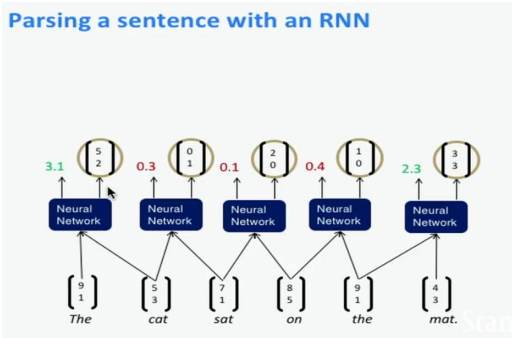
2 score 来评价 **这个结构的有效性**



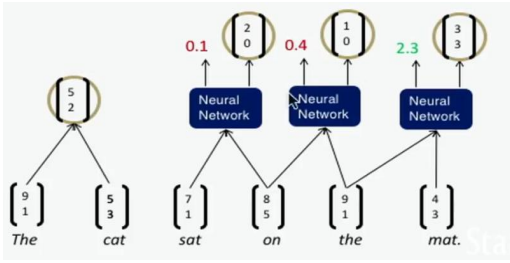
1 将两个 c 拼接然后通过 softmax 得到 **parent vector**

2 然后用 Parent vector 和 U vector 计算 **score**

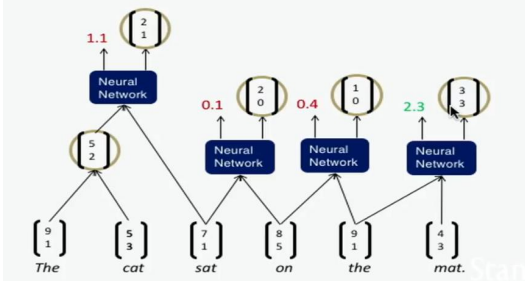
注意是同一个 **NN same weight**，专门用来**判定是否为 phrase** 的分类器



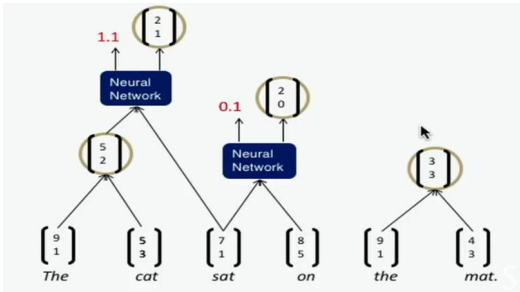
greedy 算法：得到每个相邻 word pairs 组合的分类结果
选择最高分数



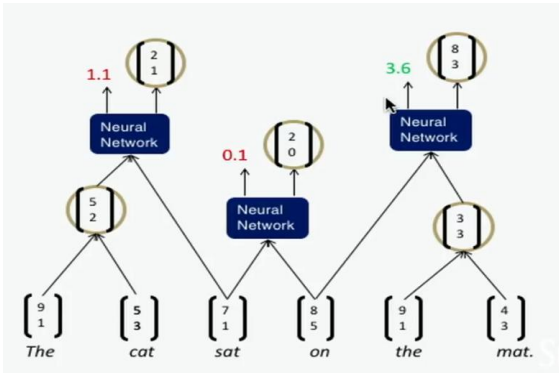
先选取最高的分数，得到 combination vector
这个 vector 当做普通的 vector，再上面的计算

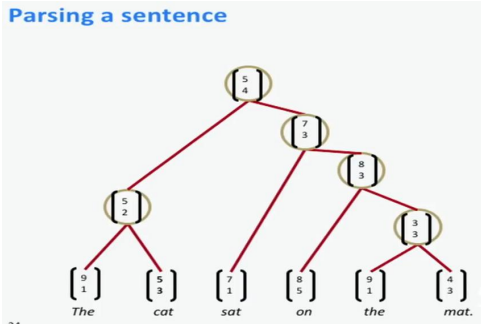


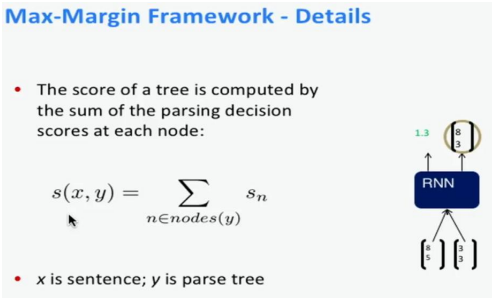
然后再跟临近的 word 组合计算分数
此时又得到每个 word 参与后的分数



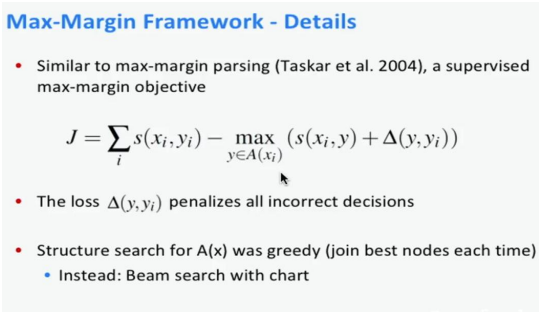
然后再选最高的
再跟临近的计算







优化问题，所有 decision 的 score 的总和作为最终的 score



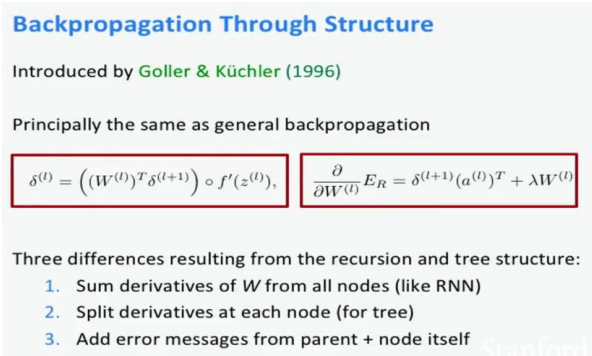
构建最终的 loss

1 score sum　　2 结构是否正确 penalty

但效率太低了，费时间！！！！

替代方法：　bean search　10 best 但不能保证是句子的最好解析

derivative



BTS: 1) Sum derivatives of all nodes

You can actually assume it's a different *W* at each node

Intuition via example:

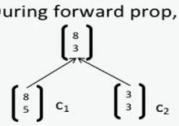
$$\begin{aligned} & \frac{\partial}{\partial W} f(W(f(Wx)) \\ = \; & f'(W(f(Wx)) \left(\left(\frac{\partial}{\partial W} W \right) f(Wx) + W \frac{\partial}{\partial W} f(Wx) \right) \\ = \; & f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right) \end{aligned}$$

If we take separate derivatives of each occurrence, we get same:

$$\begin{aligned} & \frac{\partial}{\partial W_2} f(W_2(f(W_1x)) + \frac{\partial}{\partial W_1} f(W_2(f(W_1x)) \\ = \; & f'(W_2(f(W_1x)) \left(f(W_1x) \right) + f'(W_2(f(W_1x)) \left(W_2 f'(W_1x) x \right) \\ = \; & f'(W_2(f(W_1x)) \left(f(W_1x) + W_2 f'(W_1x) x \right) \\ = \; & f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right) \end{aligned}$$

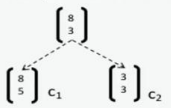
BTS: 2) Split derivatives at each node

During forward prop, the parent is computed using 2 children



$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

Hence, the errors need to be computed wrt each of them:

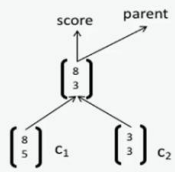


where each child's error is *n*-dimensional

$$\delta_{p \to c_1 c_2} = [\delta_{p \to c_1} \delta_{p \to c_2}]$$

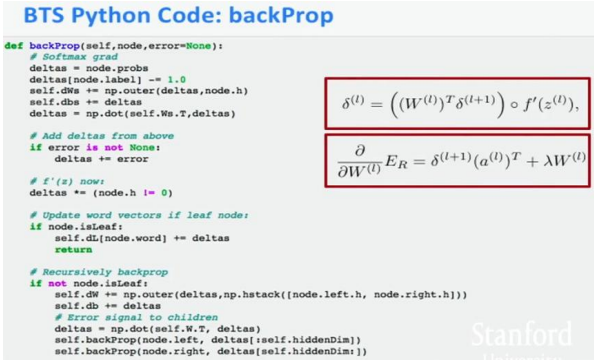
BTS: 3) Add error messages

- At each node:
  - What came up (fprop) must come down (bprop)
  - Total error messages = error messages from parent + error message from own score



BTS Python Code: forwardProp

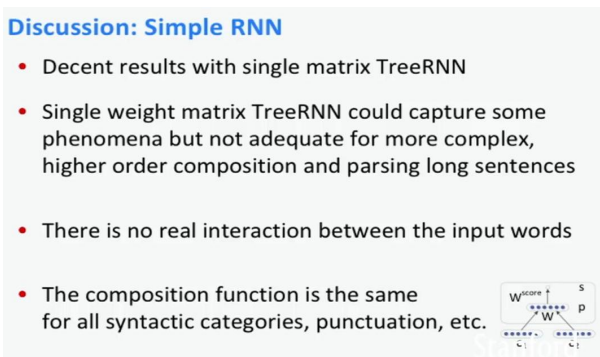
```
def forwardProp(self,node):
    # Recursion
    ...

    # This node's hidden activation
    node.h = np.dot(self.W,np.hstack([node.left.h, node.right.h])) + self.b
    # Relu
    node.h[node.h<0] = 0

    # Softmax
    node.probs = np.dot(self.Ws,node.h) + self.bs
    node.probs -= np.max(node.probs)
    node.probs = np.exp(node.probs)
    node.probs = node.probs/np.sum(node.probs)
```

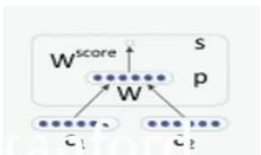


Recursive NN 变种

single matrix TRNN

前面每次 decision 都只是同一个矩阵





single matrix TRNN 缺陷：

1 如果仅仅 concate vector 然后跟矩阵相乘，**没有 input 的 interaction 没有交互** 只有 combination

其实就是两个 matrix 跟 vector 分布相乘然后 concate

2 **只有一个 classifier，针对不同的 syntactic category**

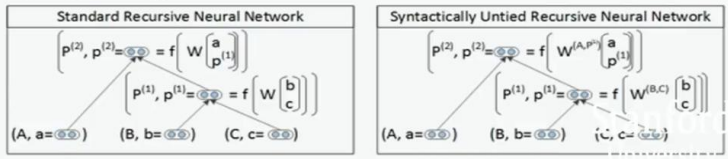
可以每一种一个 syntactic classifier

SU-RNN

结合了句法的 RecNN 句法就是先用语法来判定 syntactic category，不同 category 不同 composition matrix

Version 2: Syntactically-Untied RNN

- A symbolic Context-Free Grammar (CFG) backbone is adequate for basic syntactic structure
- We use the discrete syntactic categories of the children to choose the composition matrix
- A TreeRNN can do better with different composition matrix for different syntactic environments
- The result gives us a better semantics



不同句法类别，**不同的 composition matrix 参数(构成矩阵**，构成两个部分的向量)
需要先用语法来 built syntactic structure，告诉我们两个 word 的**句法类别**
用子节点的 syntactic category(np pp)然后利用 symbolic grammar 来 choose **different 类型的 matrix**

question

Compositional Vector Grammars

- Problem: Speed. Every candidate score in beam search needs a matrix-vector product.
- Solution: Compute score only for a subset of trees coming from a simpler, faster model (PCFG)
  - Prunes very unlikely candidates for speed
  - Provides coarse syntactic categories of the children for each beam candidate
- Compositional Vector Grammar = PCFG + TreeRNN

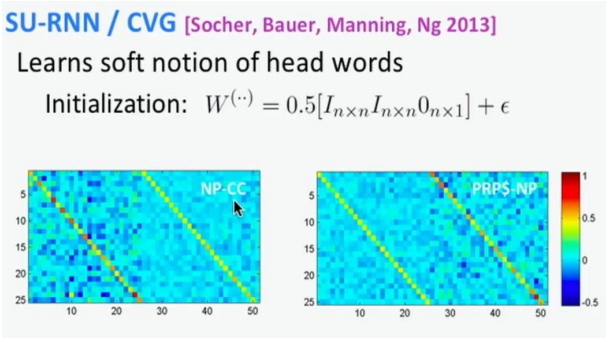
CVG

Related Work for parsing

- Resulting CVG Parser is related to previous work that extends PCFG parsers
- Klein and Manning (2003a) : manual feature engineering
- Petrov et al. (2006) : learning algorithm that splits and merges syntactic categories
- Lexicalized parsers (Collins, 2003; Charniak, 2000): describe each category with a lexical item
- Hall and Klein (2012) combine several such annotation schemes in a factored parser.
- CVGs extend these ideas from discrete representations to richer continuous ones

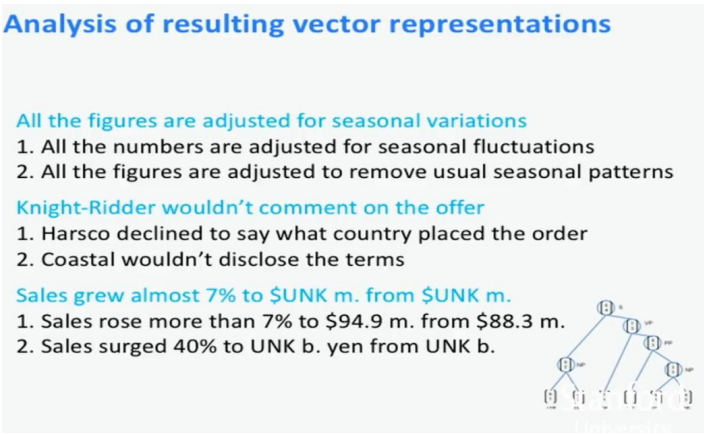
Parser	Test, All Sentenc
Stanford PCFG, (Klein and Manning, 2003a)	85.5
Stanford Factored (Klein and Manning, 2003b)	86.6
Factored PCFGs (Hall and Klein, 2012)	89.4
Collins (Collins, 1997)	87.7
SSN (Henderson, 2004)	89.4
Berkeley Parser (Petrov and Klein, 2007)	90.1
CVG (RNN) (Socher et al., ACL 2013)	85.0
CVG (SU-RNN) (Socher et al., ACL 2013)	90.4
Charniak - Self Trained (McClosky et al. 2006)	91.0
Charniak - Self Trained-ReRanked (McClosky et al. 2006)	92.1

parse 是用来得到 semantic representation of phrase，关键在 semantic 语义提取，不只是句子结构



两个矩阵拼在一起得到 W，每个矩阵都**初始化为对角阵，分别用于乘以左右子节点**
对于 NP CC　　student　and　　更多的信息来自于 student，因此左边的 matrix 会大
右边 his cat

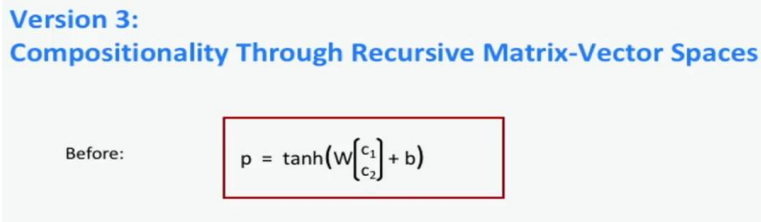
evaluate-sentence vector



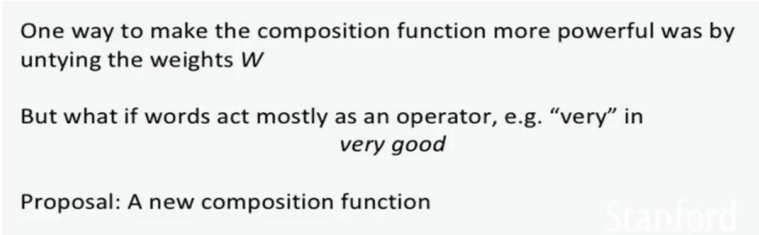
最终得到句子的向量结果，进行评估，**向量是否真正的 catch 了句子的 semantic sentence similarity　句意的相似性，但不绝对是同义句！！**

MV-RNN

Matrix-Vector RNN



更加 powerful！！当 **word 作为 modifier 或者 operator 修饰另一个 word 时**



我们需要一个新的 composition function 方法

MV-RNN

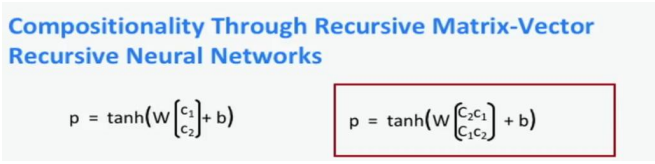
每个 word 表示为一个**向量+矩阵　　矩阵就是 operator**

因为不确定两个 word 那个是 operator，每个都带个 matrix

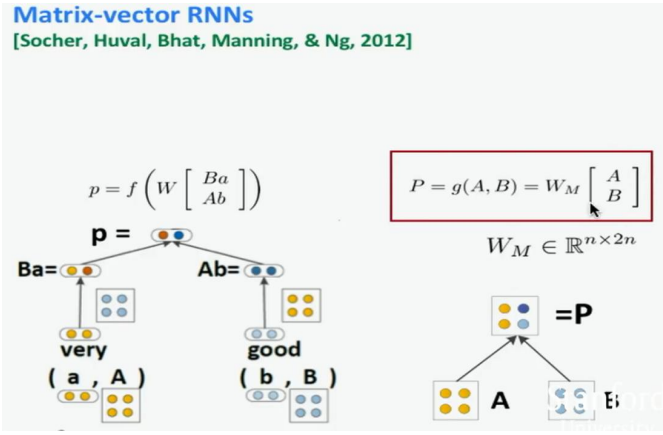
matrix multiply vector still is a vector，然后 concate 两个 vector 如下图：

这样两个 words 之间就有了交互作用

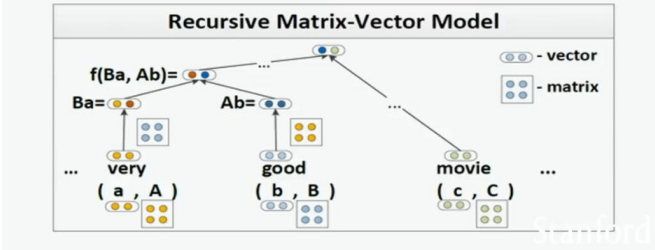
得到了 parent vector p，还差个 matrix P

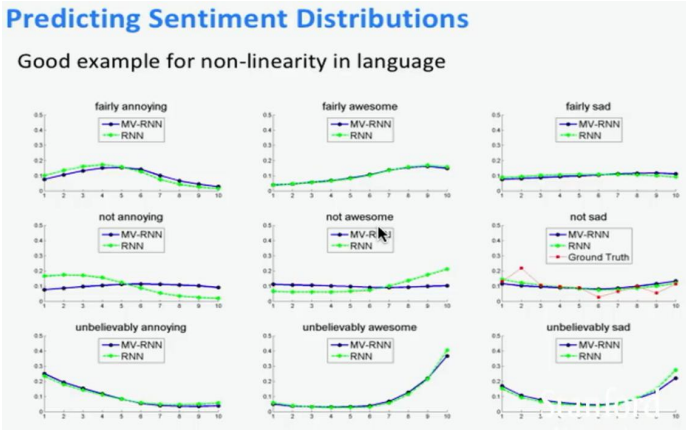


用子节点的 AB matrix 拼接然后乘以矩阵，来计算大 P 矩阵



然后再跟下一个 word 进行同样操作

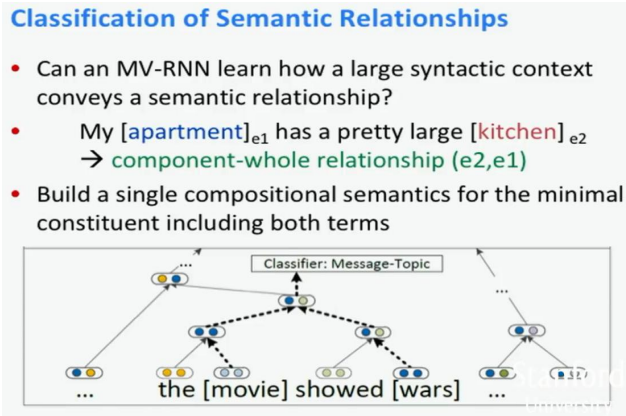




MV RNN 对 phrase 的意义更准确　　情感分析

两个 word 的交互意思体现出来 not 是形容词的 motifier(operator)　not +形容词 RNN 就不好

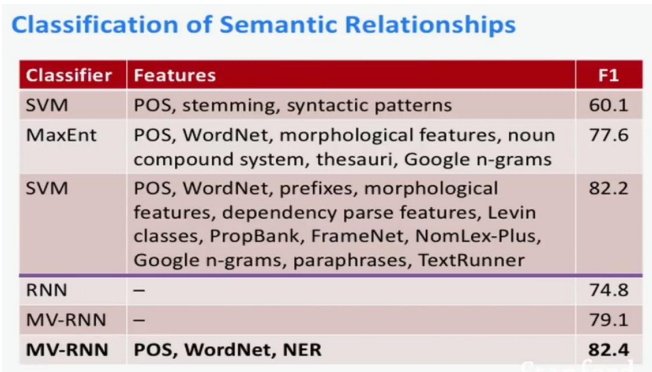
MVRNN 捕捉 word semantic relationship



假设 movie 和 wars 是 **message--topic 关系**

可以用 MV-RNN 捕捉，先合并了 the movie，又合并了 showed wars

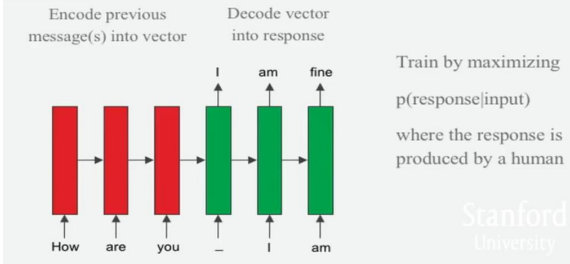
等到包含 movie 和 wars 的词语合并为一个节点时，就可以用分类器分类了



Deep Reinforcement Learning for Dialogue Generation

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao and Dan Jurafsky

Seq2Seq for Dialogue



Problems with Seq2Seq

容易死循环

回答**太范了**

probable response != good response

high proba 但不是好的 response

What is a good response?

- **Reasonable** p(response|input) is high according to seq2seq model
- **Nonrepetitive** similarity between response and previous messages is low
- **Easy to answer** p("i don't know"|response) is low

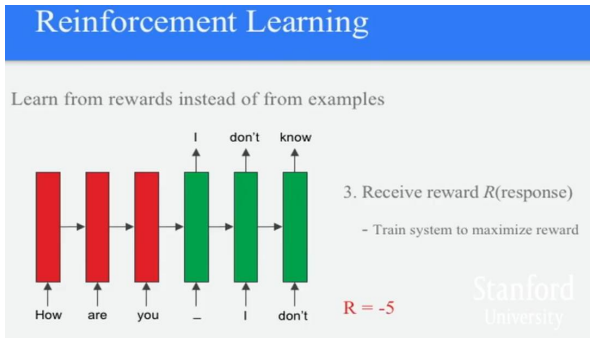
Scoring function: *R*(response) = reasonable_score + nonrepetitive_score + easy_to_answer_score

Stanford University

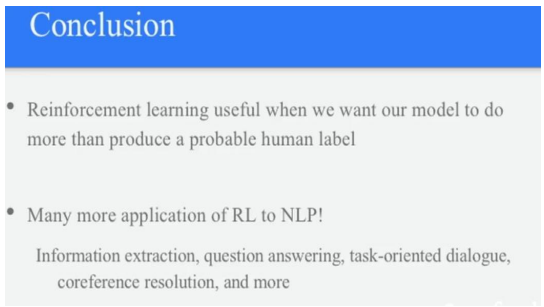
1

2 不重复

3easy to answer 不能 I don't know 不好回答

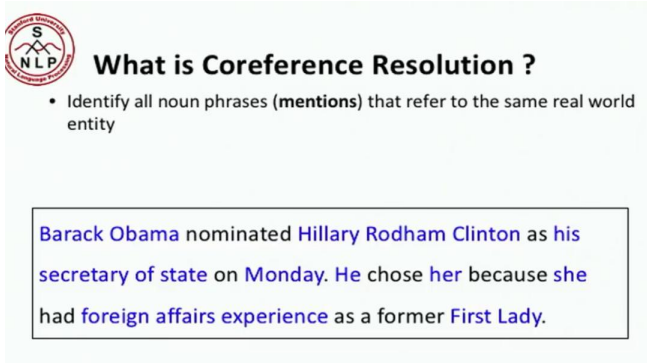


利用 score 作为 RL 的 reward



15 - Coreference Resolution

针对 large text

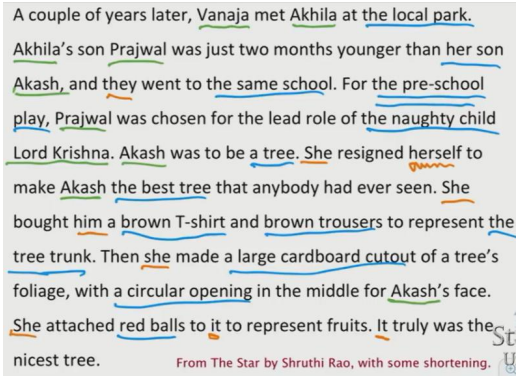


coreference: two mentions refer to same entity in the world

找到文章中的所有 noun phrase，这些 noun phrase 都指向现实中的 entity

共指解析消解(coreference resolution)　将指向同一个 entity 的 noun phrase 消掉

比如 name　he　his　herself



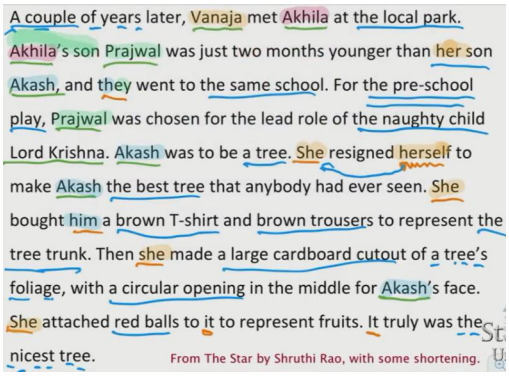
name entity recognition

1 主语

2 pronon 代词

3 名词 entity

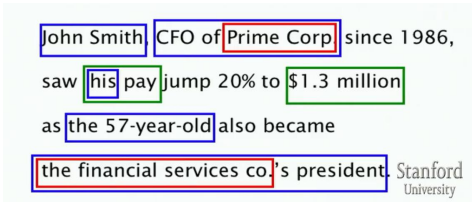
coreference



解析每个词相同的指代(**指代同一个 entity**)(人名)

‘Tree’含义不同，有些是 reference to the real tree of world

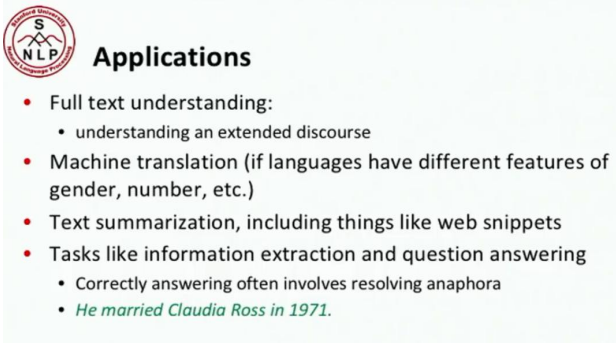
有些不是



- Noun phrases refer to entities in the world, many pairs of noun phrases co-refer, some nested inside others

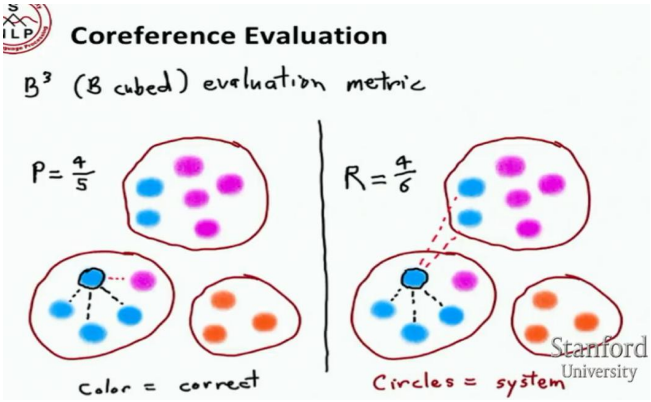
很多 nest 情况

his pay mentioned，his 也 mentioned

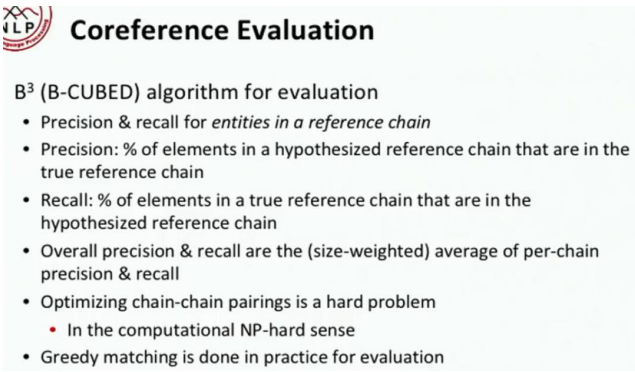


text summarize/information extraction

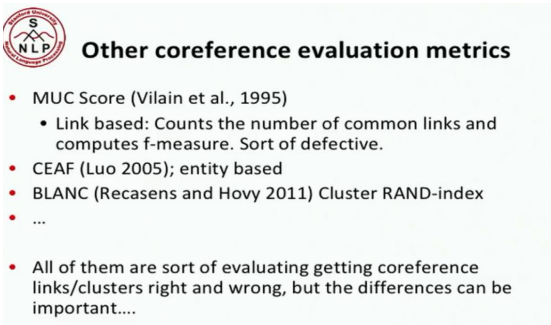
evaluate B cube



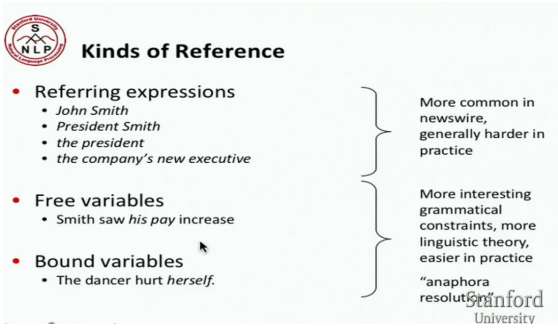
coreference 和分类一样，看同类的词是否分在一起
目标是将三种颜色分开，三个圈代表三种颜色类别
对于蓝色的 recall precision： 也就是左下角分类
precision p 针对一类 蓝色的 precision 是 4/5
recall 该分类对于 蓝色的 recall 是 4/6



一个分类就是一个 **reference chain**
需要根据类别数量大小，给与 recall precision 权重！
downside ： 不容易优化
实际用 greedy



reference 种类

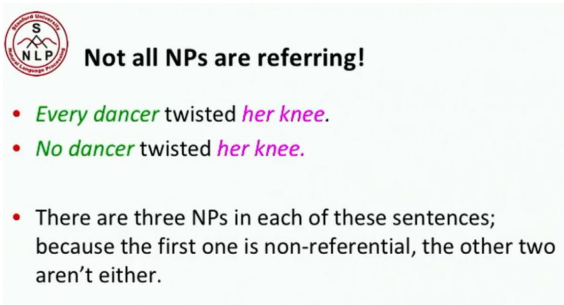


reference 种类

1referencing 表达方式

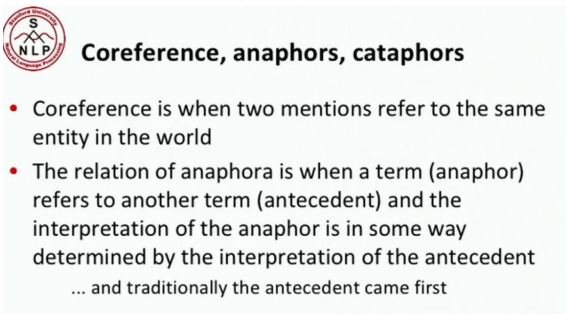
2 **自由变量** his pay 指向 smith simth 指向前面的词

3 固定变量 herself 只跟指向附近的词 dancer **紧跟前面某个词**



每句话都有 3 个 noun phrase

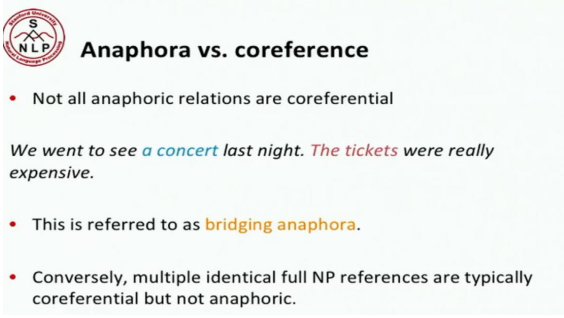
但因为第一个 phrase 不是 reference，剩下的 phrase 都没有 reference



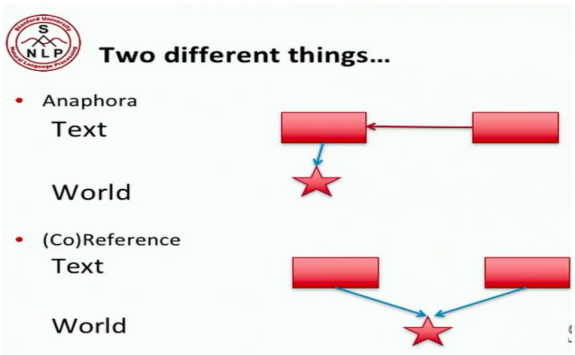
1coreference: two mentions refer to same entity in the world **(identity 关系 等价关系)**

2anaphor 指向前面的词，并且 **anaphor 的解释**也受前面的词解释影响，但不一定**等价**

是 anaphor 不一定是 coreference

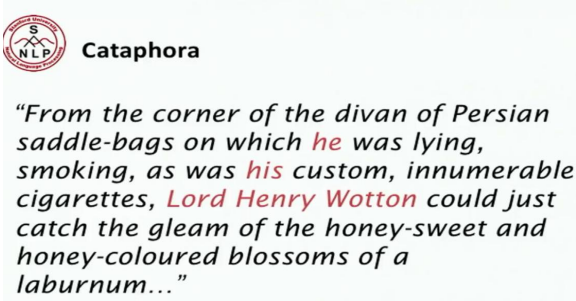


the ticket 受 concert 解释的影响 **不是 identity 关系** 不是 coreference
此时是 **bridging anaphor 将两个 entity 联系起来**
相反 coreference 不一定是 anaphor



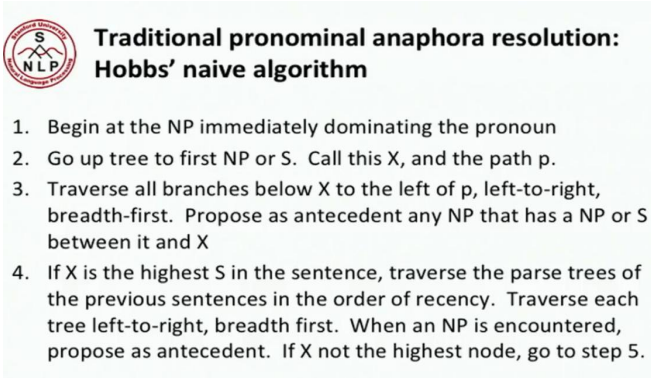
refer same thing in the world

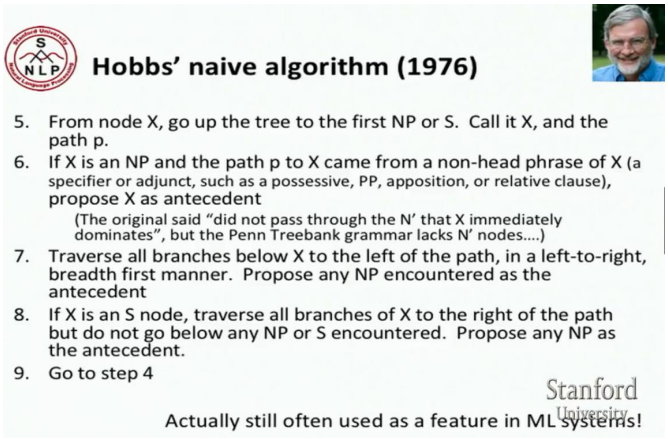
cataphors 相反指向后面



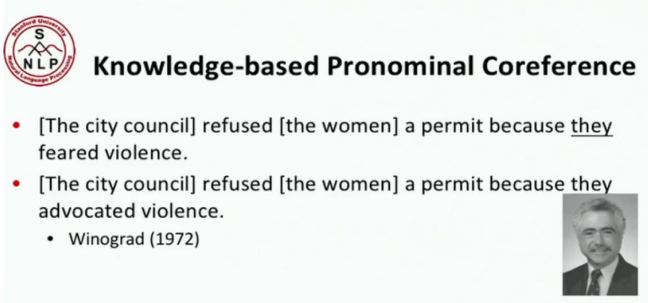
he 和 his 指向后面的 LHW LHW 绝对前面的词的意思
但实际 我们会将 he 作为第一个词出现，后面 refer

如何做 coreference resolution

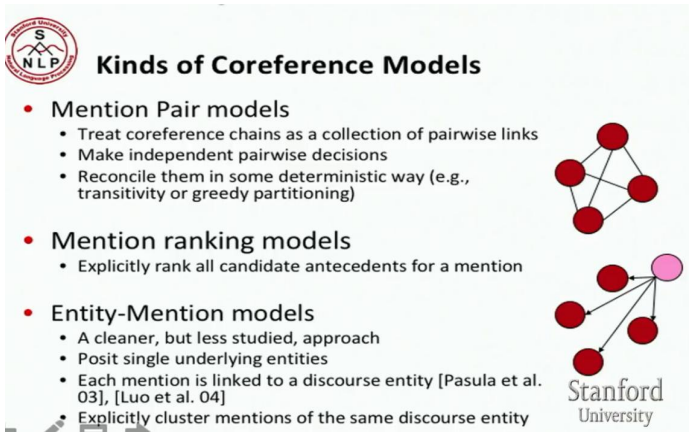




很复杂！！！
但解决不了;



they 指向的不同 **不能仅仅从句子分析**
需要**知识**

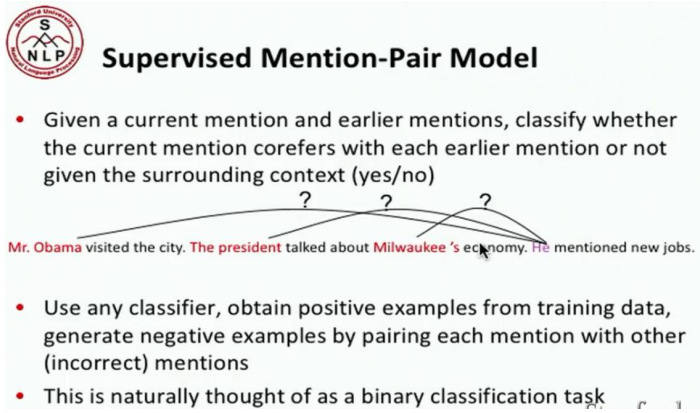


最常用的

Mention pair model

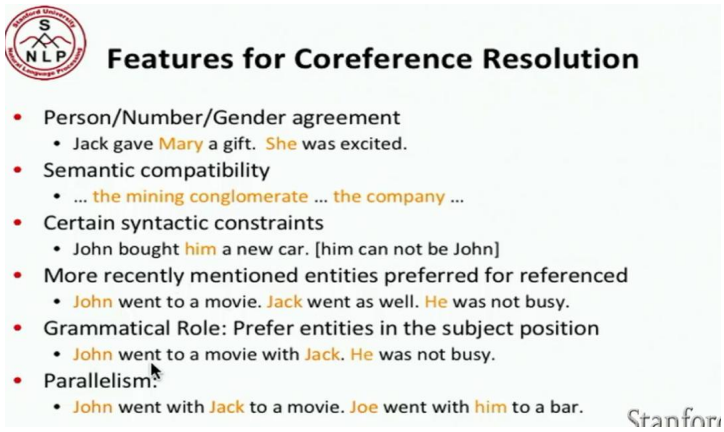
1binary classifer 来独立**判断每个 word 直接的关系**

2 所有的 coreference 变成了 pairwises links(有方向)



其实就是做二分类

上面三个词，有一个 pos 和多个 neg example 做训练 **binary classifier**

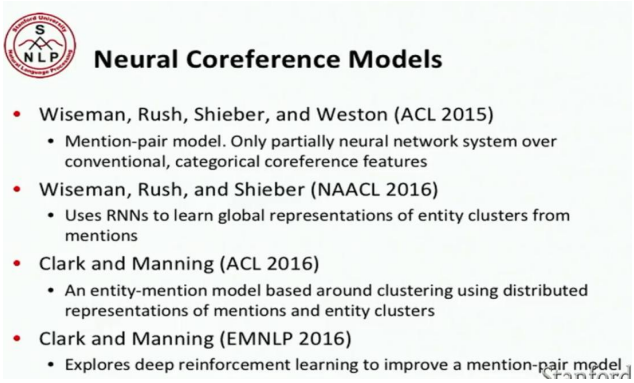


传统方法：hands-on feature

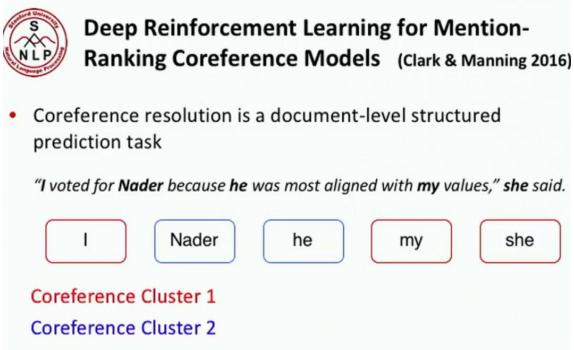
有很多规则(**也就是语言 feature**)判断是否是 refer，

word feature 性别 是否是谓语，是否是平行

来 built binary 分类器

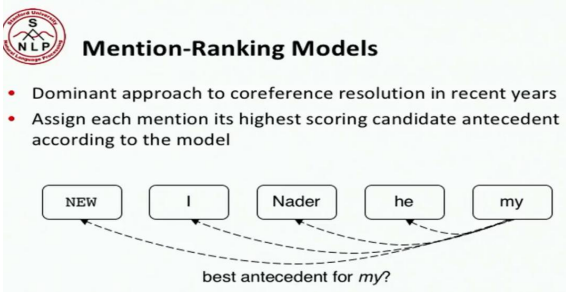


深度学习方法

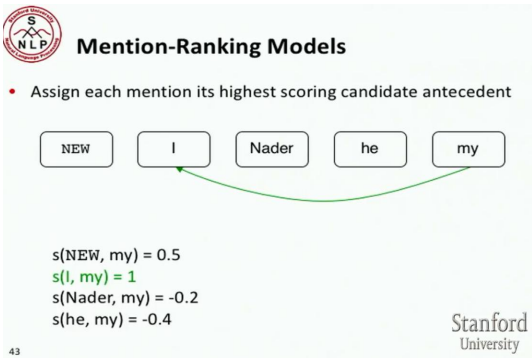


mention ranking model

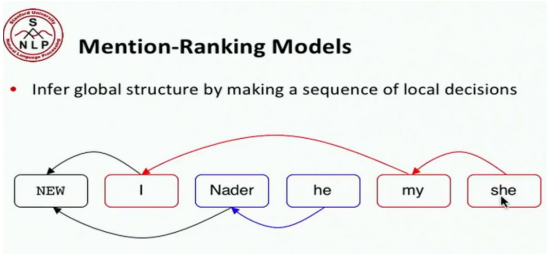
coreference cluster



assign score 给每个 anntecedent,选择最高的

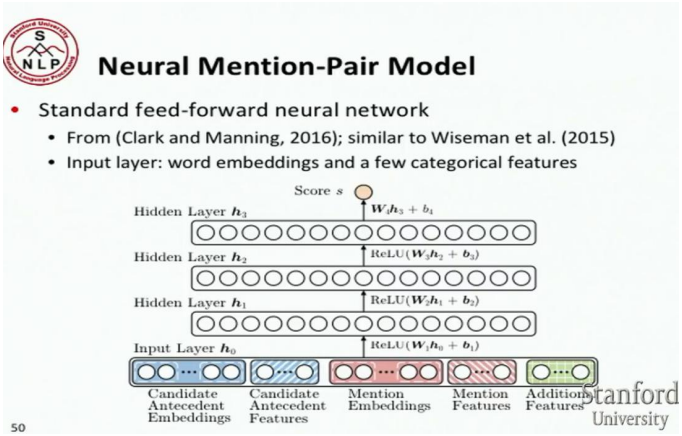


从左到右的每个 mention 进行这样的操作，连接最高分数的 mention

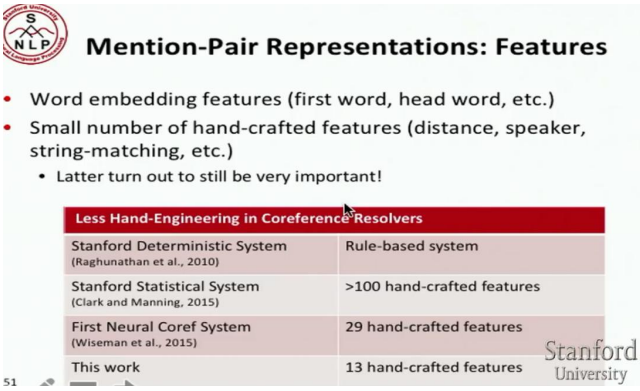


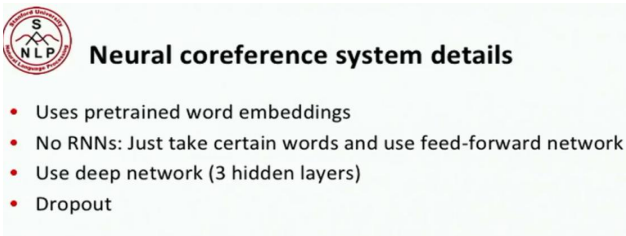
如何得到 classifier？**由局部 decision** 得到如上**全局的句子结构**
给分数，并且要全局最优，强化学习在行

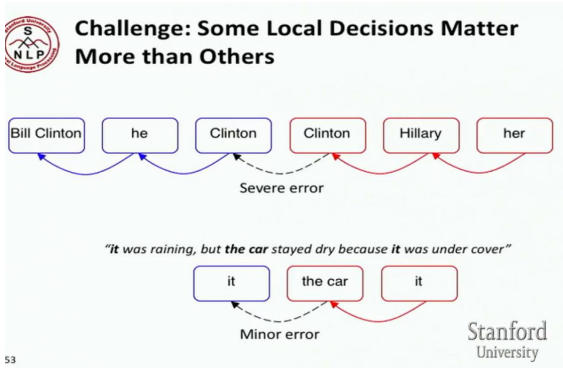
Neural mention-pair model



其实就是 **plain deep learning**
input： **canditate antecedent 和 mention 的** word embedding+**语言 feature** 来弥补 word embdding 的遗漏，如下：







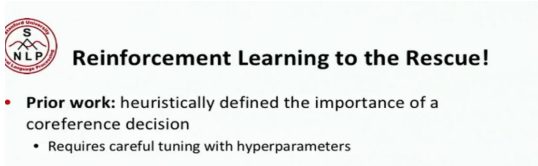
上面是严重的错误(一旦分类错就问题很大)，下面是小的错误(影响不大)

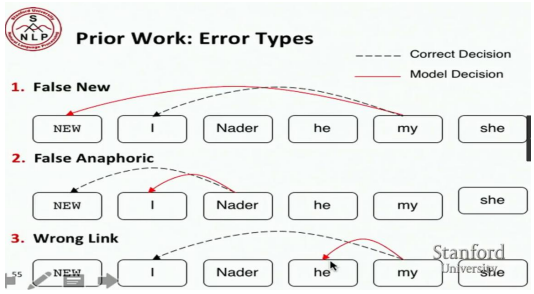
如何避免严重的错误，定义这样的 loss function 不容易(每个分类器相同的 loss，惩罚相同)

只有一个 loss 不能判断某次是严重错误还是小错误， 而且需要等句子结束也就是，在 ending 的时候才能决定问题的严重性

没办法定义 local 局部 loss decision

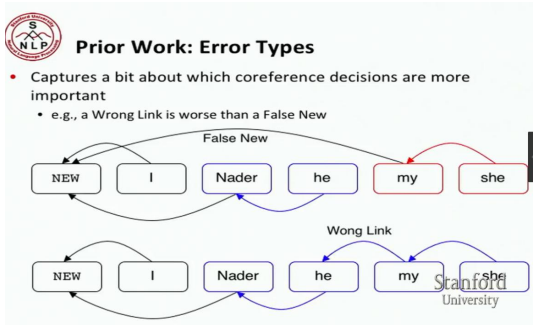
此时需要 Reinforcement learning 定义 local 的 loss





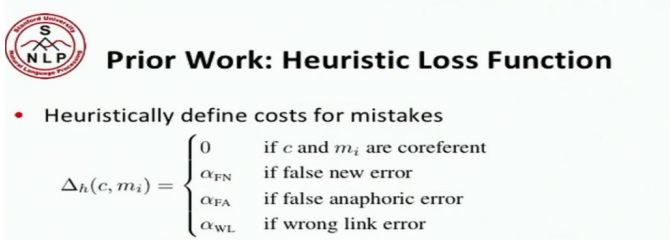
可以定义不同的错误，定义不同 weight 的 loss

不同的 reward，就强化学习



heuristic loss：

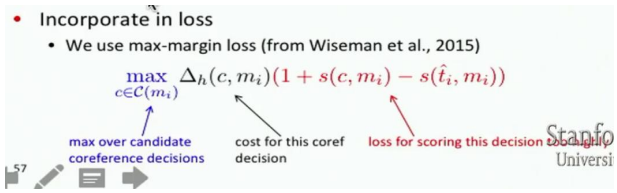
对不同类型的 error 设定不同的惩罚 作为因子



首先不同的类型 error，定义不同的 reward

c 是 canditate precedent

m 是不同的 mention

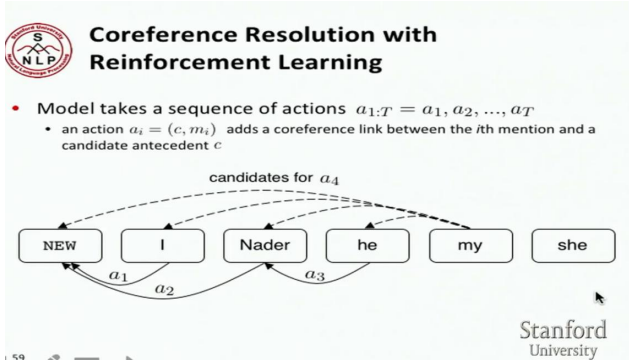


定义 loss

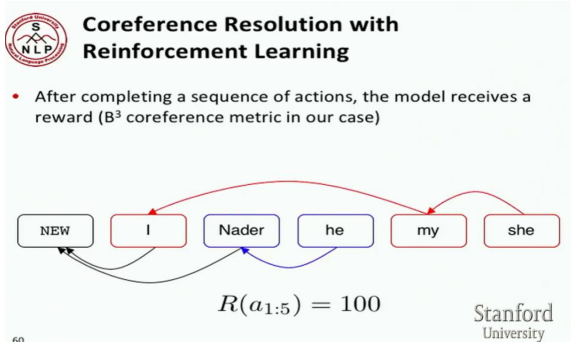
s 是 score 红色是前面见过的 margin loss(就是普通的 loss) **t 是正确的 candidate**

一旦 loss 是正，代表分错，再乘以不同错误的权重

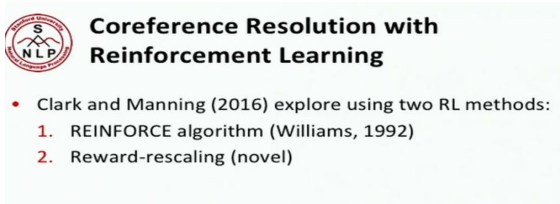
每一次**选择让 loss 最大的 c**

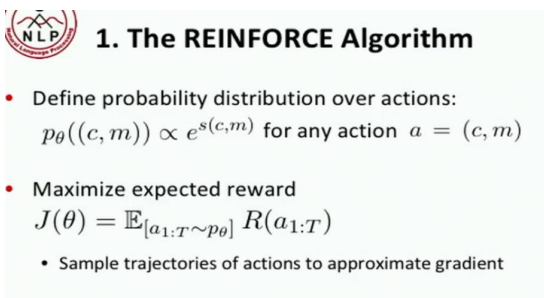


看做**一些列的 action，最终得到最佳的 coreference**



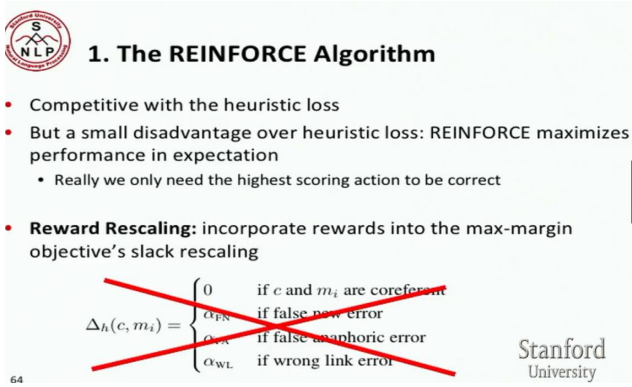
reward 就是 **cluster metric score**



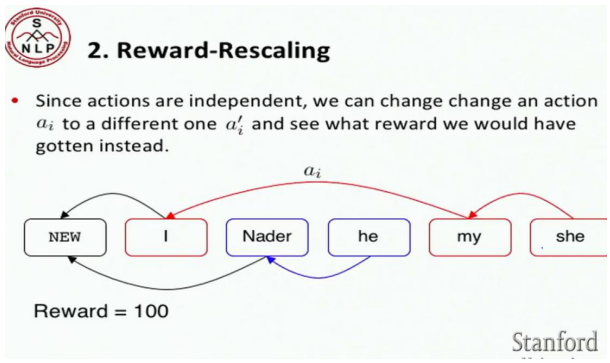


先得到**每个 action 的概率分布**
然后一系列的 action maximize **reward 期望**
有指数级的 action sequence 太多了，我们随机 sample action 来近似 gradient

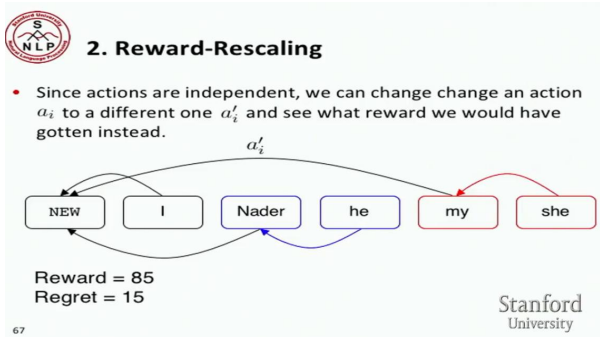
RL 只 maxmize 期望 performance，我们想 maximum action score



不用之前手动制定的的值，而是能够 scaling margin loss



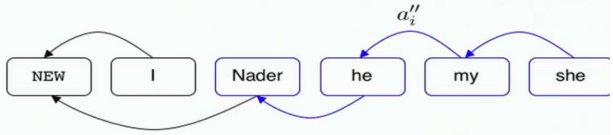
每个 action 独立，可以 change 一个 action，看看 award 的变化
先从正确的 coreference 开始，分数=100 可以是 B-cube 分数



更改 action 后变成 85 regret=15

2. Reward-Rescaling

- Since actions are independent, we can change change an action a_i to a different one a_i' and see what reward we would have gotten instead.



Reward = 66
Regret = 34

Stanford

换其他的，更低了

2. Reward-Rescaling

- Use this idea to do reward-based slack-rescaling

$$\Delta_r(c, m_i) = \max_{a_i' \in \mathcal{A}_i} R(a_1, ..., a_i', ..., a_T) \quad \text{reward for best action}$$
$$- R(a_1, ..., (c, m_i), ..., a_T) \quad \text{reward for current action}$$

- Cost is the regret of taking the action
  - Replaces the heuristic cost, otherwise use the same max-margin loss function

此时 scaling factor 可以由差值决定(和**正确决定的差值 100**)
cost 是 regret 分数

Experimental Setup

- English and Chinese Portions of the CoNLL 2012 Shared Task dataset
- Predicted mentions from the Stanford rule-based system (Lee et al., 2011)
- Scores are CoNLL F_1 scores
  - Average of MUC, B-cubed, and CEAFe metrics

System Performance

Model	English	Chinese
Heuristic Loss	65.36	63.54
REINFORCE	65.41	63.64
Reward Rescaling	**65.73**	**63.88**

- Small but statistically significant ($p < 0.05$) improvement from reward rescaling

Clark and Manning (EMNLP 2016) results

CoNLL average, CoNLL 2012 data, scorer v8.01

Model	English	Chinese
Chen & Ng (2012) [CoNLL 2012 Chinese winner]	54.52	**57.63**
Fernandes (2012) [CoNLL 2012 English winner]	**60.65**	51.46
Björkelund & Kuhn. (2014) Best previous Chinese system]	61.63	**60.06**
Wiseman et al. (2016) [Best previous English system]	**64.21**	—
Clark & Manning (ACL 2016)	65.29	63.66
(Clark & Manning, EMNLP 2016)	**65.73**	**63.88**

72

Where do neural scoring models help?

- Especially with nominals with no head match
  - 18.9 F_1 vs 10.7 F_1 on this type compared to 68.7 vs 66.1 F_1 overall when compared with Clark & Manning (2015).

Example Wins

Anaphor	Antecedent
the country's leftist rebels	the guerillas
the company	the New York firm
216 sailors from the ``USS cole''	the crew
the gun	the rifle

Stanford University

73

在哪里提升多？

nominal 名词性词，

Error Breakdown

- Reward-rescaling model actually makes more errors!
- However, the errors are less severe
  - ≈0.7% lower cost on average

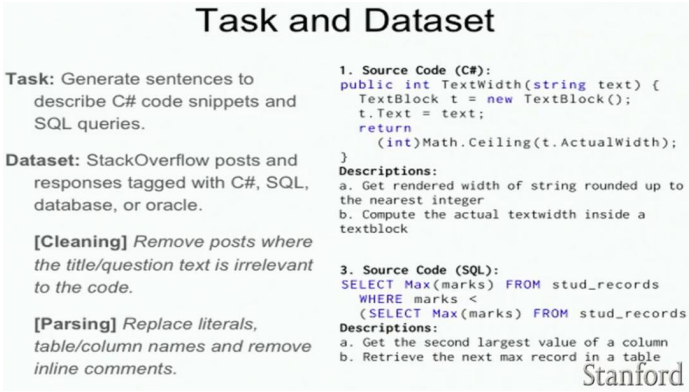
Model	# False Anaphoric	# False New	# Wrong Link
Heuristic Loss	1956	1719	1258
Reward Rescaling	1994	1725	1247

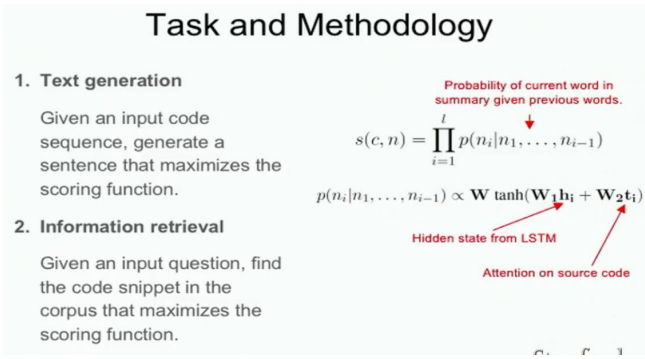
关键是 make minor mistake

Summarizing Source Code using a Neural Attention Model

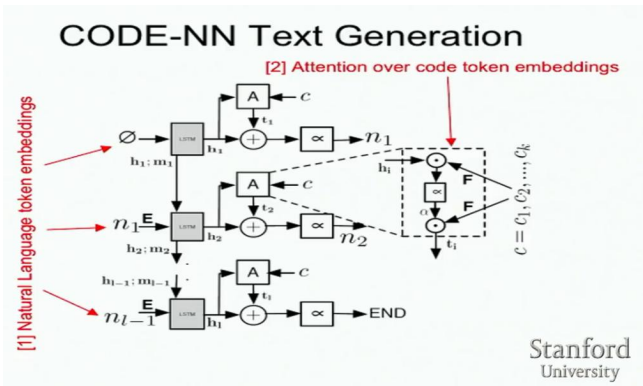
Srinivasan Iyer, Ionnis Konstas, Alvin Cheung, Luke Zettlemoyer
University of Washington CSE
In proceedings of ACL '16

总结 code 的功能





最后 获取 code



16 - Dynamic Neural Networks for Question Answering

NLP 最终的目的是否是 QA?

Can all NLP tasks be seen as question answering problems?

QA Examples

I: Mary walked to the bathroom.
I: Sandra went to the garden.
I: Daniel went back to the garden.
I: Sandra took the milk there.
Q: Where is the milk?
A: garden
I: Everybody is happy.
Q: What's the sentiment?
A: positive

I: Jane has a baby in Dresden.
Q: What are the named entities?
A: Jane - person, Dresden - location
I: Jane has a baby in Dresden.
Q: What are the POS tags?
A: NNP VBZ DT NN IN NNP .
I: I think this model is incredible
Q: In French?
A: Je pense que ce modèle est incroyable

1obstacle no single architecture

First Major Obstacle

- For NLP no single model **architecture** with consistent state of the art results across tasks

Task	State of the art model
Question answering (babI)	Strongly Supervised MemNN (Weston et al 2015)
Sentiment Analysis (SST)	Tree-LSTMs (Tai et al. 2015)
Part of speech tagging (PTB-WSJ)	Bi-directional LSTM-CRF (Huang et al. 2015)

针对不同任务，不同类型结构

2 obstacle multi-task learning

Second Major Obstacle

- Fully joint multitask learning* is hard:
  - Usually restricted to lower layers
  - Usually helps only if tasks are related
  - Often hurts performance if tasks are not related

* meaning: same decoder/classifier and not only transfer learning

一起训练多任务是我们想要的，但很难

我们不只是想训练好一个 task，然后 transfer 另一个 task 上，是需要一起一个任务
需要 share 相同的 decoder 和 classifier
有时候会使得 performance 下降 **比单独的 train**

solve first obstacle ： Dynamic Memory Networks

Tackling First Obstacle

Dynamic Memory Networks

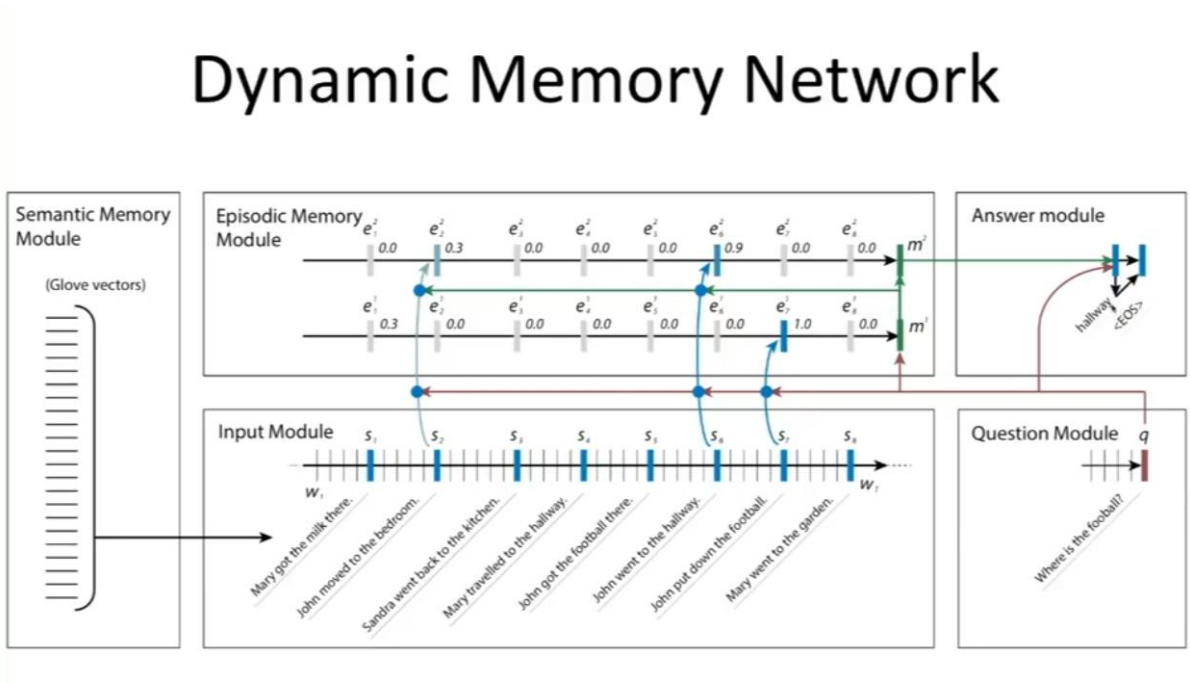
An architecture for any QA task

High level idea for harder questions

- Imagine having to read an article, memorize it, then get asked various questions → Hard!
- You can't store everything in working memory
- **Optimal:** give you the input data, give you the question, allow as many glances as possible

Star

不是把全部内容记在内存里，而是**需要多次重新 go over input**

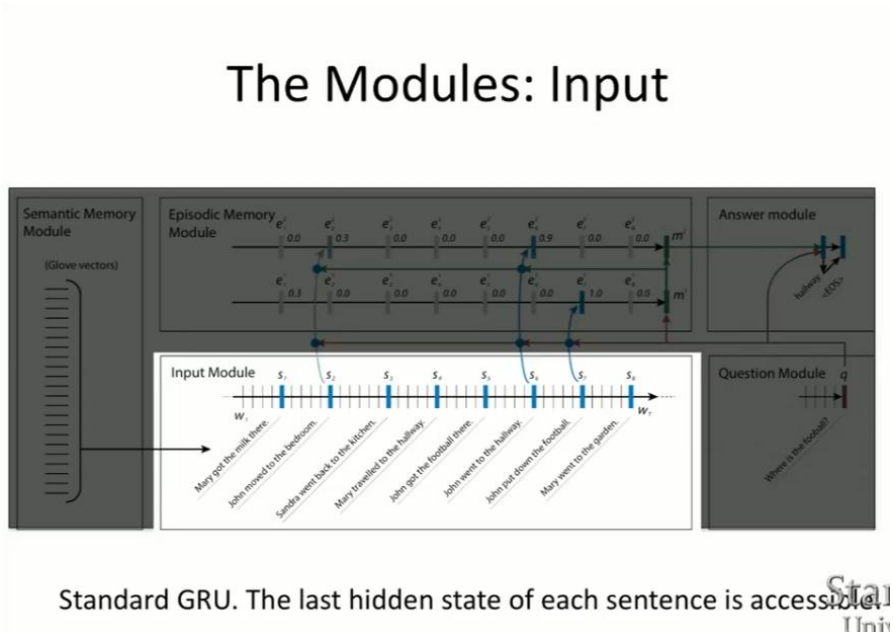


end to end model，一起 train！！从 answer backpropagation
不同的 **modue** 不同的接口
左边是预训练的 word embedding
下面是 GRU compute 每个 sentence 的 represent hidden state 也就是对句子编码 每个 input

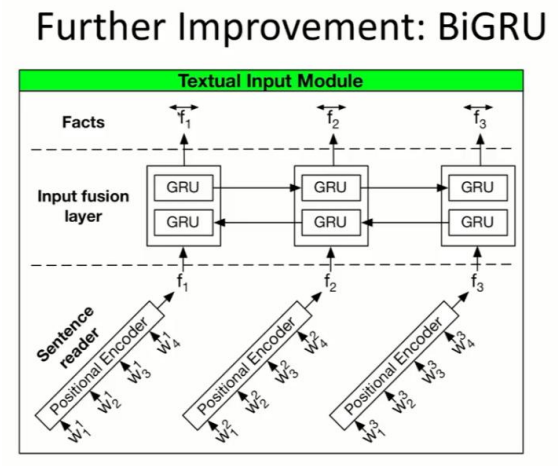
是个句子**(多个句子)，应该就是得到多个句子的多个 hidden state，按时间顺序编号**
应该只是一个 GRU 结构，每个句子执行一次，得到一个 representation

question GRU 结构

右下 也是 GRU 用作 **question 句子编码 得到 question vecor (laste state)** **(只有一个句子)**
然后拿到**问题 q 后**，利用 attention 来回顾 input hidden state(来匹配)，一旦匹配程度高**的 input hidden state** **再**给上面的 RNN，然后进行多次，也就是多次回顾 RNN
最上面的两条线，是两次回顾，**每次回顾会 pay 不同的 attention of input hidden states**
第一次只利用 q 是找到包含 question 的 football 的句子，和问题最相关的句子，然后计算 m1，**第二次利用 m1 和 q pay attention to john 获取新的相关信息得到 m2**
算是一种 reasoning 第一次回顾发现句子有 john，第二次回顾就关注 john
只要 train input data 有这些信息，就可以做 reasoning

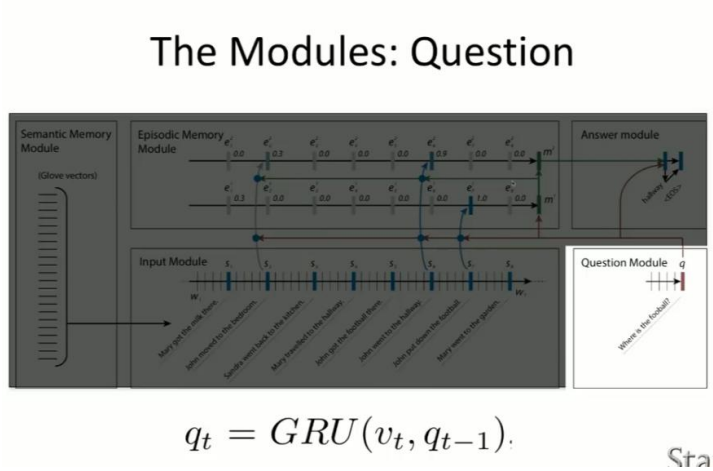


每个句子的最后一个 **hidden state 代表整个句子** ， 每个句子的 representation 有了
方便 attention 的时候回顾整个文章的所有句子 **question 这个结构不是很理解**

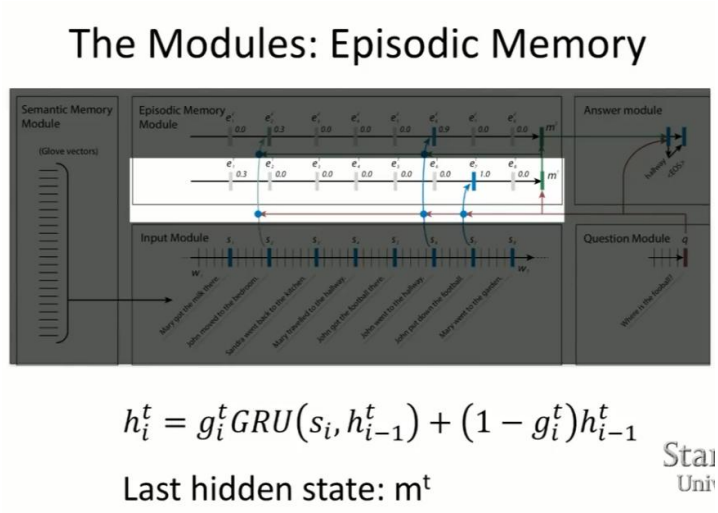


应该是先将 **sentence encoding 此时是 GRU**，然后用 sentence 的 hidden state 给再给**大的 GRU**！**使得句子与句子之间也有依赖关系，每个 hidden state 都代表前面整个文章**
第 t 时刻的 hidden state 包含第 t sentence 的信息最多！！！！

question 如何 encoding



是一个独立的 GRU，v 是 word vector
q 是每个 word 的 hidden state



Episodic memory 是个 GRU，结构和下面一致
t 是 review 第 t 次　　s 是 sentence 的 encoding　　i 是第几个 sentence
g 是 0-1 scalar，代表 attention，也就是对应每个 hidden state 的权重(pay attention)
如果是 0，就还是用 t-1 的 hidden state，如果是 1，就用**当前的 hidden state**(**代表当前的很相关！**)
然后最后的 hidden state 是 m

如何计算 g?

The Modules: Episodic Memory

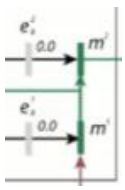
- Gates are activated if sentence relevant to the question or memory

$$z_i^t = [s_i \circ q\ ;\ s_i \circ m^{t-1}; |s_i - q|\ ;\ |s_i - m^{t-1}|]$$

$$Z_i^t = W^{(2)} \tanh\left(W^{(1)} z_i^t + b^{(1)}\right) + b^{(2)}$$

$$g_i^t = \frac{\exp(Z_i^t)}{\sum_{k=1}^{M_i} \exp(Z_k^t)}$$

用**当前** sentence 的 hidden state 和 question state 以及 上一次 review 的 m 来计算距离
找到本次最相关的 input sentences
similarity 很简单：**元素乘&元素距离绝对值**
注意第一次 review 时，m0 会被初始化为 q
通过上面向量的拼接，得到当前 **z 向量**
然后 dense，得到我们的 Z score，然后每个 sentence 的 sore 转换成概率值 g
归一化，有个缺点，只能 pay attention g 值很大的 sentence，会使得整体的值都不大，可以用 sigmoid 代替，能 pay 更多 sentence

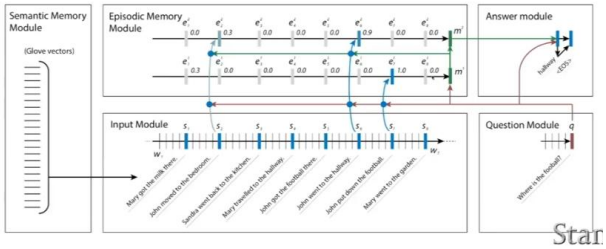


m 也是个 GRU，最终得到一个 memory state 给 answer

- When the end of the input is reached, the relevant facts are summarized in another GRU

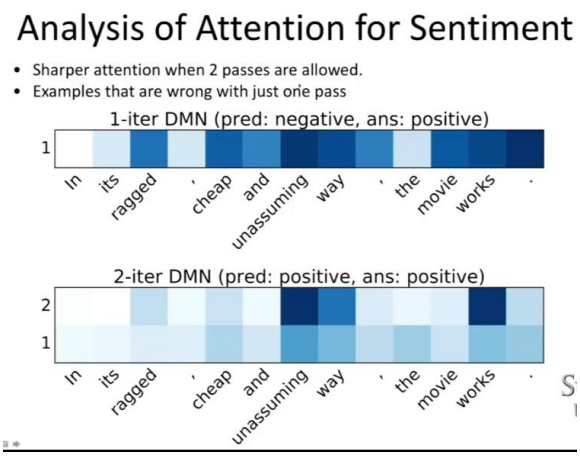
The Modules: Answer

$$a_t = \text{GRU}([y_{t-1}, q], a_{t-1}), \qquad y_t = softmax(W^{(a)} a_t)$$

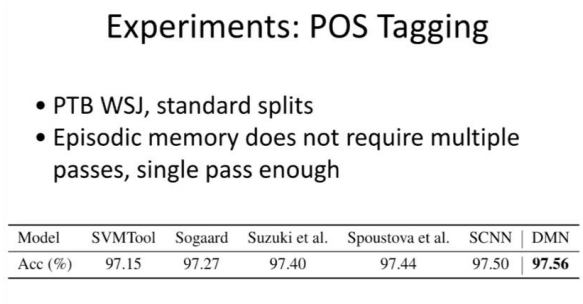


m 作为初始的 a0
yt-1 是该 module 前一个输出+question（每次都加）
输出答案

review 多少，要根据 task！！



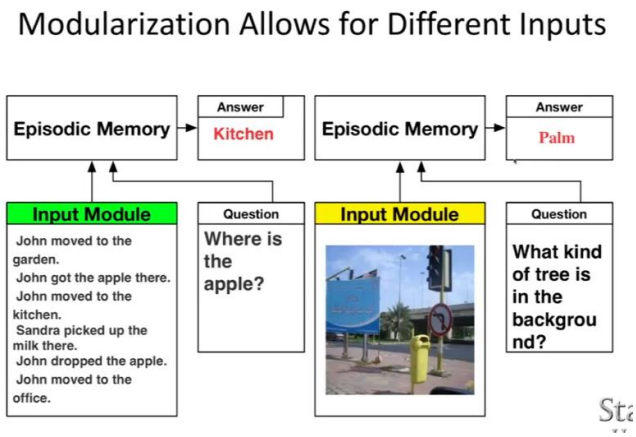
再第 2 次 iteration 才正确找到 word



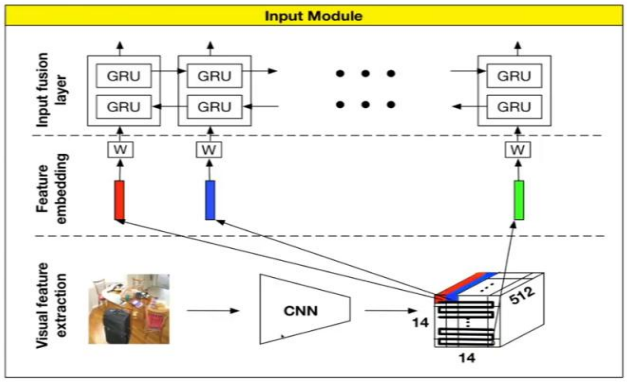
Model	SVMTool	Sogaard	Suzuki et al.	Spoustova et al.	SCNN	DMN
Acc (%)	97.15	97.27	97.40	97.44	97.50	**97.56**

part of speech tagging

加上图片 input



Input Module for Images



CNN 会将图片的所有 block 的特征提取，pretrained on **image net**

最后一张图片变成 16*16 *512 其实就是**图片划分为 16*16 格子**，每个格子有不同的特征(图片不同位置的特征(包含什么东西))　**每个格子当做 word embedding**

将这每个部位给 bi-direction RNN，希望能记住前面的部位

Accuracy: Visual Question Answering

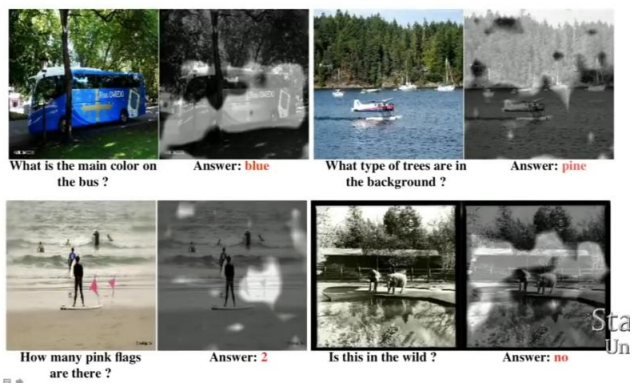
VQA test-dev and test-standard:

- Antol et al. (2015)
- ACK Wu et al. (2015);
- iBOWIMG - Zhou et al. (2015);
- DPPnet - Noh et al. (2015); D-NMN - Andreas et al. (2016);
- SAN - Yang et al. (2015)

	test-dev				test-std
Method	All	Y/N	Other	Num	All
VQA					
Image	28.1	64.0	3.8	0.4	-
Question	48.1	75.7	27.1	36.7	-
Q+I	52.6	75.6	37.4	33.7	-
LSTM Q+I	53.7	78.9	36.4	35.2	54.1
ACK	55.7	79.2	40.1	36.1	56.0
iBOWIMG	55.7	76.5	42.6	35.0	55.9
DPPnet	57.2	80.7	41.7	37.2	57.4
D-NMN	57.9	80.5	43.1	37.4	58.0
SAN	58.7	79.3	46.1	36.6	58.9
DMN+	**60.3**	80.5	48.3	36.8	**60.4**

attention visualization

Attention Visualization

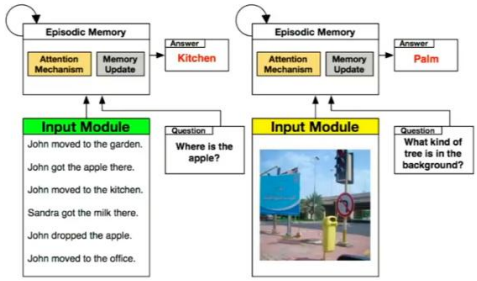


右边是 **pay 的 attention**，是自动完成的~~自动学习的

对于数字 num 的问答效果不好

Summary

- Most NLP tasks can be reduced to QA
- DMN accurately solves variety of QA tasks
- More advanced: Dynamic Coattention Networks



Learning Program Embeddings to Propagate Feedback on Student Code

Authors: Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, Leonidas Guibas.
Published at ICML 2015.
http://www.jmlr.org/proceedings/papers/v37/piech15.pdf

如何自动给 feedback

Representing Computer Programs

```
// Example student solution
function run() {
   // move then loop
   move();
   // the condition is fixed
   while (notFinished()) {
      if (isPathClear()) {
         move();
      } else {
         turnLeft();
      }
      // redundant
      move();
   }
}
```

Want concise representation of computer program, capturing the intended functionality of the code, even if the program would crash.

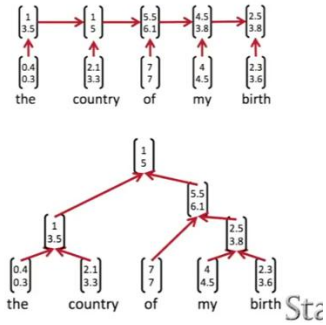
Given a pre-condition state, what would the post-condition be after executing the program?

program embedding vector

What you already know: Encoding Sentences

On natural language sentences:
→ E.g. train RNN/CNN/Recursive NN on a language modeling task
→ use trained network to create embeddings of sentences

Can we do the same for computer programs?



产生 sentence 的 embedding

Encoding and Decoding States

At each node of the tree: small neural network to encode and decode state

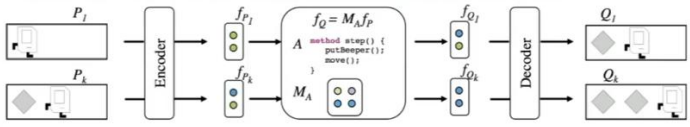


Figure 2. Diagram of the model for a program A implementing a simple "step forward" behavior in a small 1-dimensional gridworld. Two of the k Hoare triples that correspond with A are shown. Typical worlds are larger and programs are more complex.

Encoder: $$f_P = \phi(W^{enc} \cdot P + b^{enc}),$$

Decoder: $$\hat{Q} = \psi(W^{dec} \cdot f_Q + b^{dec}),$$
$$= \psi(W^{dec} \cdot M_A \cdot f_P + b^{dec}).$$

在程序执行前有 precondition state 执行后 postconfition state
P1 和 Q1 对应 执行前后状态改变
中间是程序
需要 encode P 得到 vector fp 然后乘以 M 矩阵 得到 fq
然后解码得到 Q
learn the M ， M 代表程序的功能，也就是 embedding

Objective Loss Function

$$L(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \ell^{pred}(Q_i, \hat{Q}_i(P_i, A_i; \Theta))$$
$$+ \frac{1}{n}\sum_{i=1}^{n} \ell^{auto}(P_i, \hat{P}_i(P_i, \Theta)) + \frac{\lambda}{2}\mathcal{R}(\Theta),$$

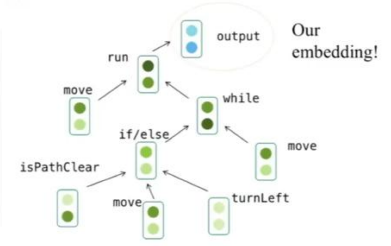
ℓ^{pred} measures how well the model is doing on predicting post-conditions

ℓ^{auto} quantifies quality of encoder / decoder on reconstructing provided pre-conditions

1 对 post state 的预测的准确性 2auto encoding 的衡量，也就是 encoder 和 decoder 的效果

Recursive Neural Network to Generate Program Embeddings

```
// Example student solution
function run() {
   // move then loop
   move();
   // the condition is fixed
   while (notFinished()) {
      if (isPathClear()) {
         move();
      } else {
         turnLeft();
      }
      // redundant
      move();
   }
}
```



Note: Programs already have inherent tree structure, so no additional parsing necessary!

Training task: Starting at the leaves, predict the post-condition given pre-condition and the program at that node

Sta
Uni

程序本身就是树形结构 函数调用函数，最终 root 是最外层函数

Summary

- Paper presented a neural network method to **encode programs** as mapping from precondition space to postcondition space, using **recursive neural nets**
- Learned representations can be used for other tasks, e.g.:
  - Cluster students by program similarity
  - Predict feedback
  - Perform knowledge tracing over multiple code submissions.

Related work

- Sequence to Sequence (Sutskever et al. 2014)
- Neural Turing Machines (Graves et al. 2014)
- Teaching Machines to Read and Comprehend (Hermann et al. 2015)
- Learning to Transduce with Unbounded Memory (Grefenstette 2015)
- Structured Memory for Neural Turing Machines (Wei Zhang 2015)

- Memory Networks (Weston et al. 2015)
- End to end memory networks (Sukhbaatar et al. 2015)
  →

Comparison to MemNets

Similarities:

- MemNets and DMNs have input, scoring, attention and response mechanisms

Differences:

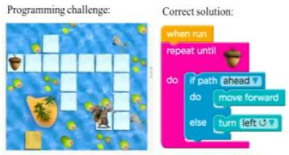
- For input representations MemNets use bag of word, nonlinear or linear embeddings that explicitly encode position
- MemNets iteratively run functions for attention and response

- **DMNs show that neural sequence models can be used for input representation, attention and response mechanisms**
  → naturally captures position and temporality
- Enables broader range of applications

Sta

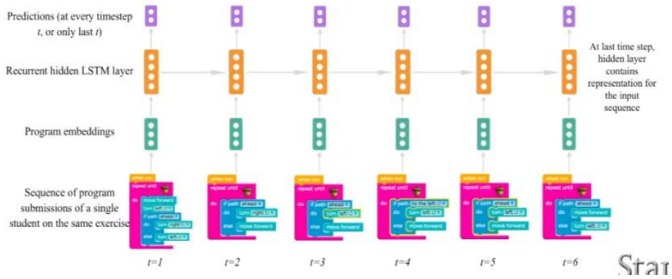
Application (ongoing research): Knowledge Tracing over Program Submissions

- Understand a student's knowledge over time while she is solving a programming challenge (potentially with intermediate submissions)
- Predict/suggest interventions:
  - Hint
  - Instructional video
  - Motivational video
  - Choice of next exercise



https://studio.code.org/hoc/18

Training objective: Given the sequence of program embeddings, predict future student performance.



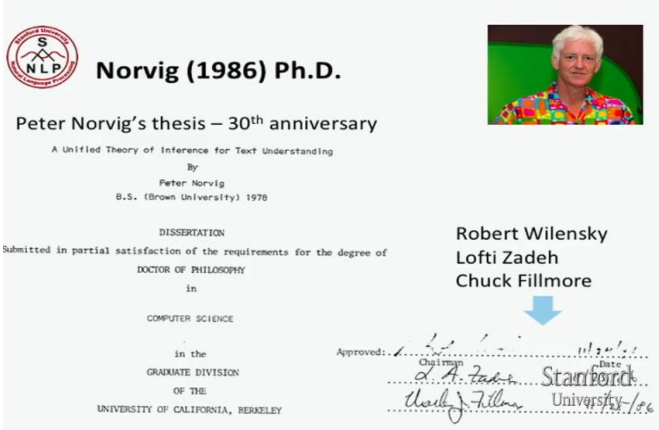
17 - Issues in NLP and Possible Architectures for NLP

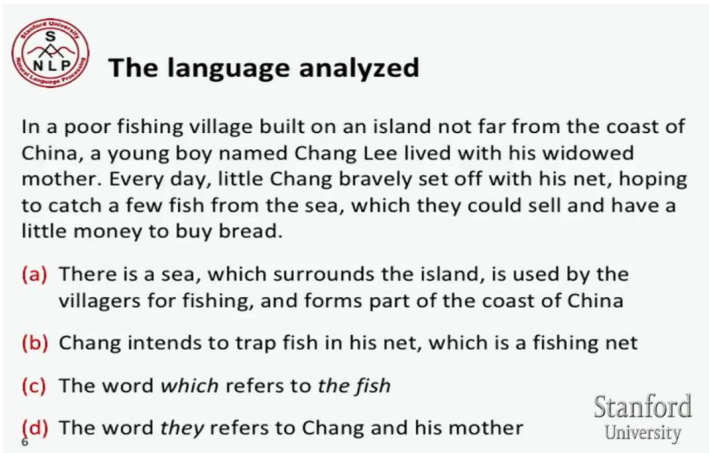
nlp&cv

nlp 跟 cv 不同，有很多细分的 task，没有一个技术能解决全部问题

computer vision, dialogue systems, virtual assistants, speech recognition, natural language understanding, translation, things like that."

part1：unified theory of inference





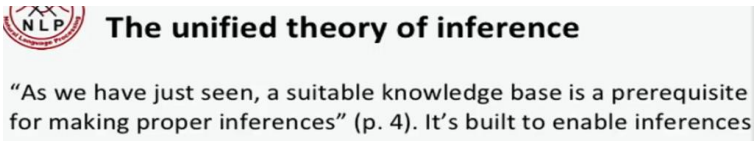
abcd 是我们希望得到的

a 是间隔很长的事物之间的 relation 的 capture

b 对事物的解释

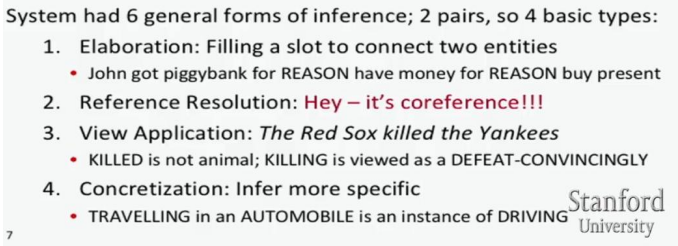
ab 效果不是很好

cd 相对简单



论文提出 knowledge base 是做 **inference** 的必要条件

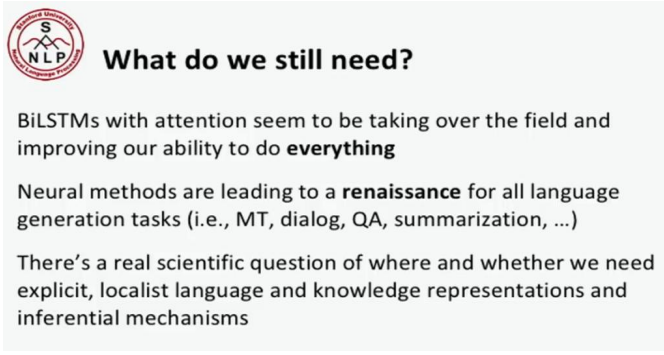
有些推理不需要 knowledge base



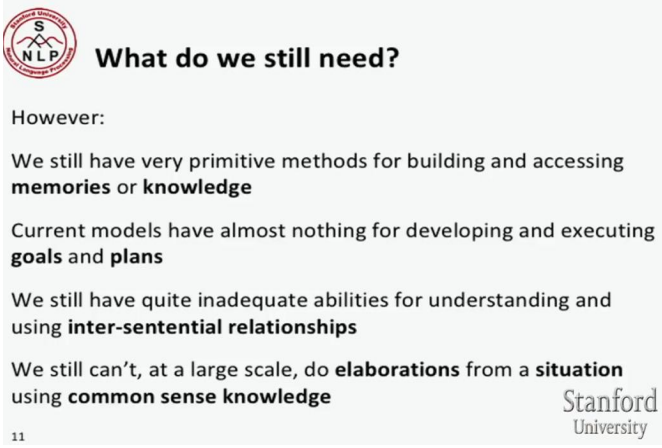
4 种推理

1connection between entities

2reference
3view **比喻** 不是字面意义
4 更具体的推理

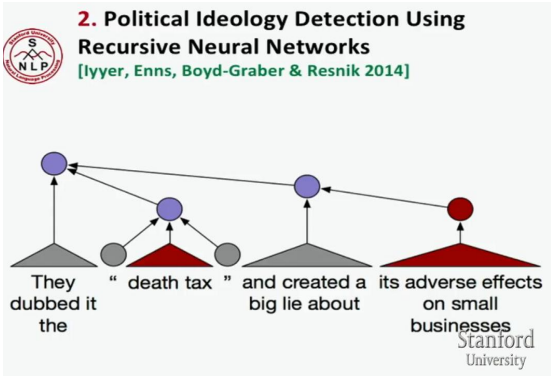


generation task 复兴了
end2end 帮我们完成了这一切，显得不需要这些 explicit 工作

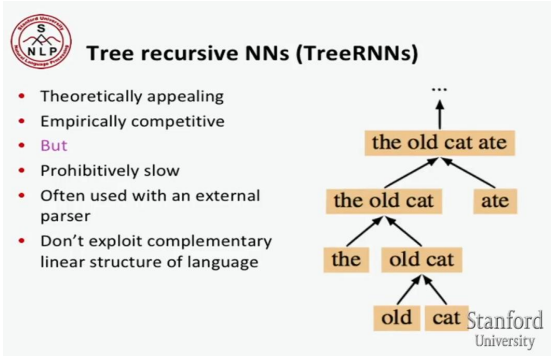


1 memory 还是太小了 short term memory。相对于人类来演，year's memory
2 模型缺少目标，都是比较通用的 loss 函数，缺少针对特殊问题的 goal
3 句间关系推理不太好(从句)
4 entities relationship

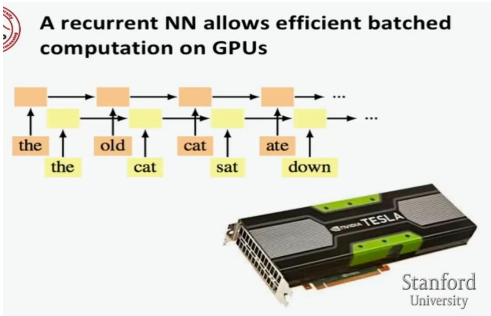
part2 Tree RNN Model



句法结构，consititute parsing



1 太慢了 2 需要很多 parser **3 有些 sentence 是 linear structure**

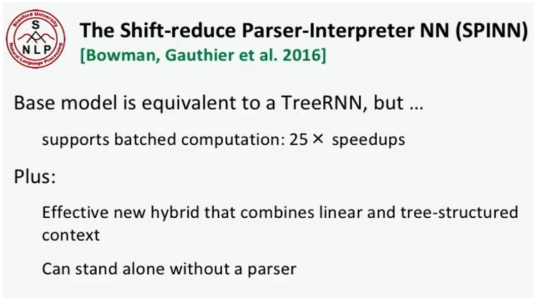


不能用 batch 计算

batch gradient 是不同的 batch 在相同的 structure 下计算，可以 gpu

普通 RNN 所有句子相同的结构,可以并行

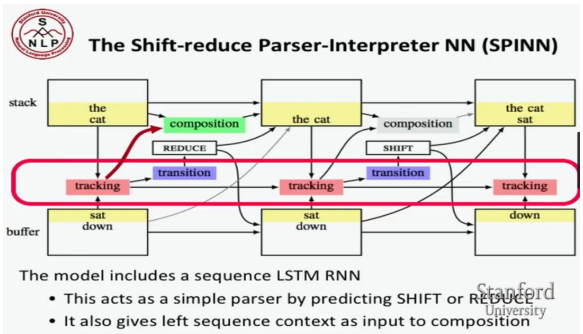
但 TRNN 不同的 sentence 不同的 structure，不能并行



还是 TRNN 提高 25 倍

需要 parse 混合了 tree 以及 linear

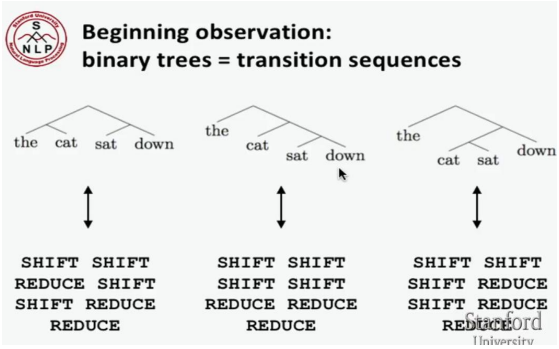
首先需要个 parse 相同的 sentence 不同的结构



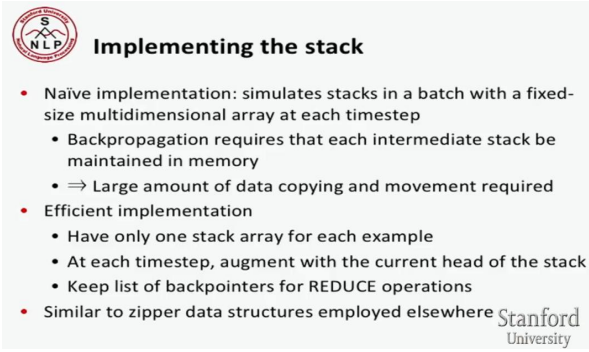
类似于 dependent parsor transition

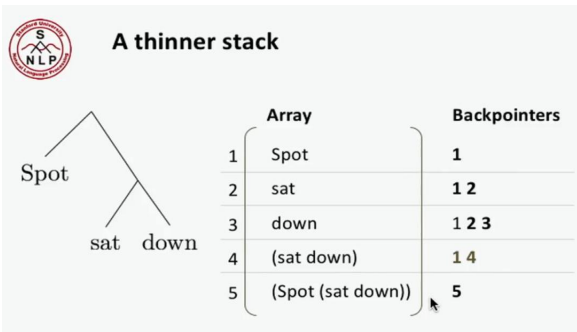
决定每两个 word 的 action(shift 或 reduce)，最终得到 tree

是一个 RNN parsor！！SPINN

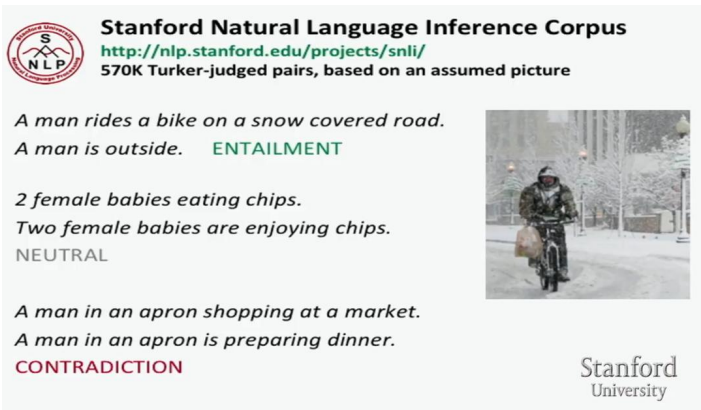


相同的句子不同的解析

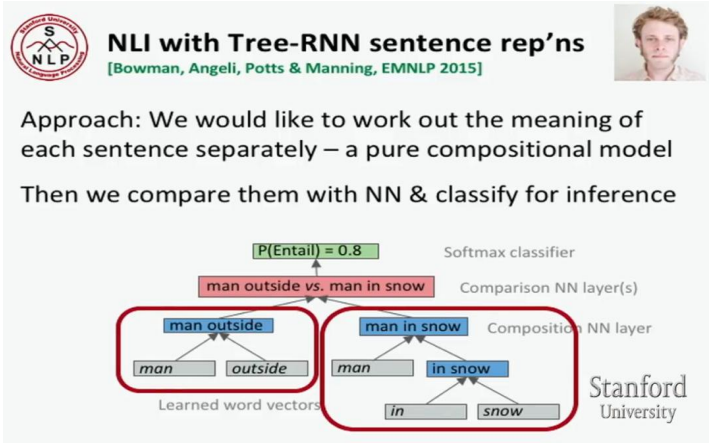




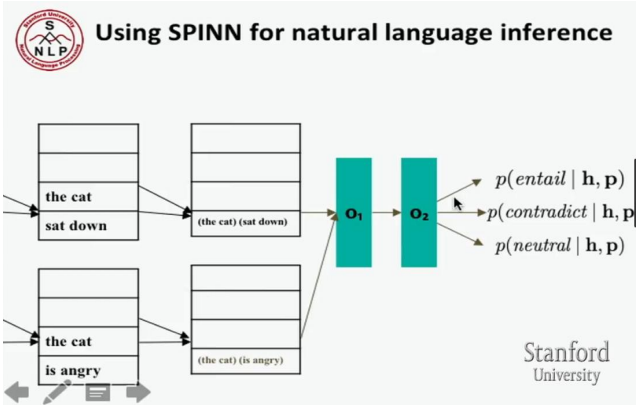
question

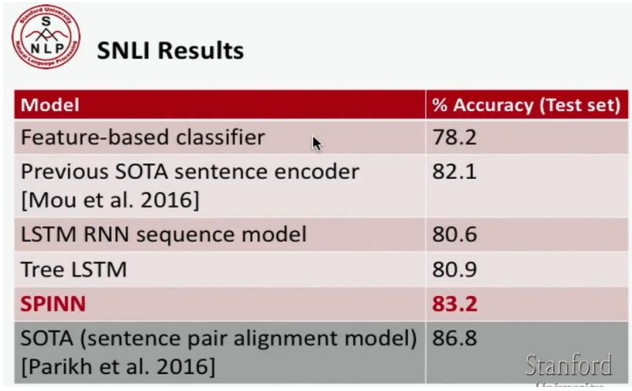


推断：1entaiment "*if A then B,*"　2contradiction　3 neural



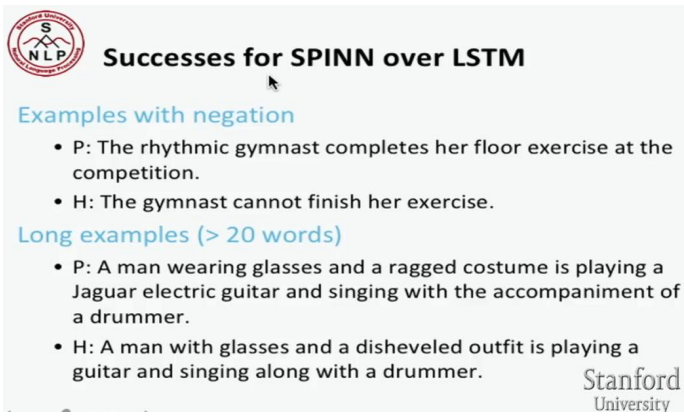
composition model 结构模型 得到每个句子的结构(Tree represent)，然后再分类(推断)



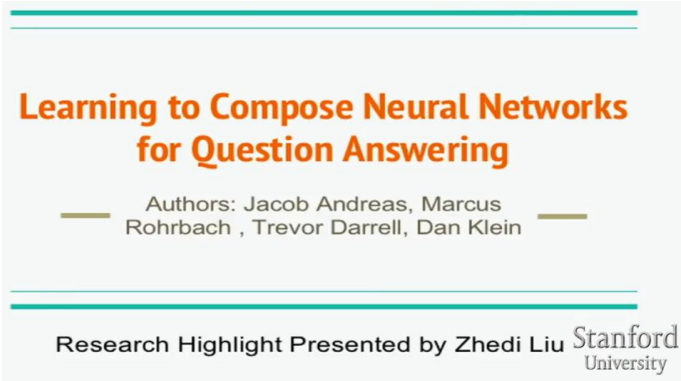


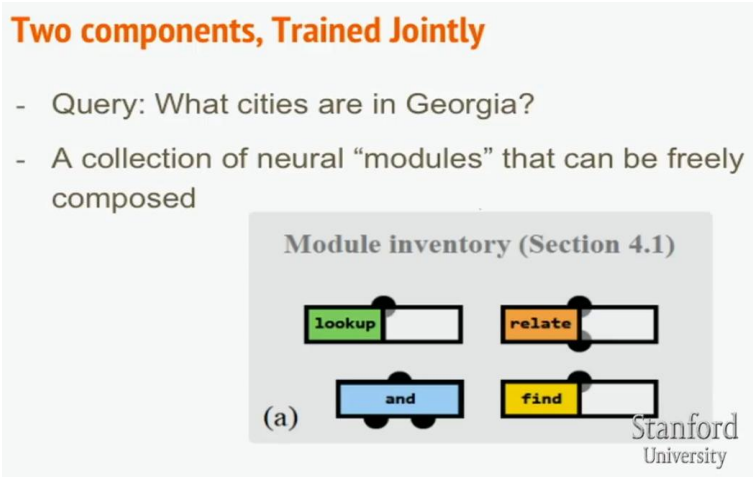
Tree model 表现一般

SPINN 结合了 tree model 和 linear sequence model

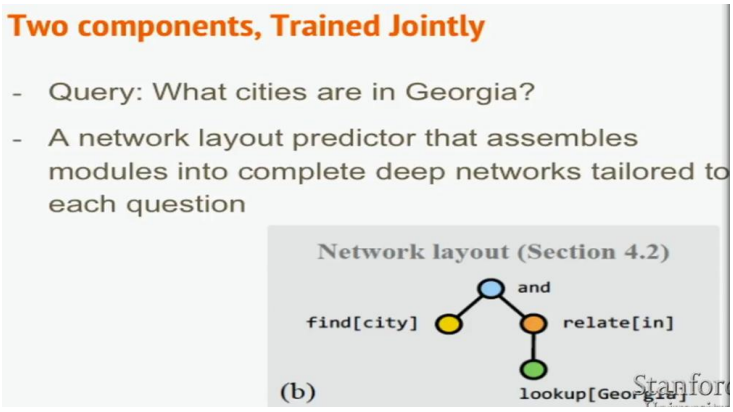


Tree structure 还是比 LSTM 会好

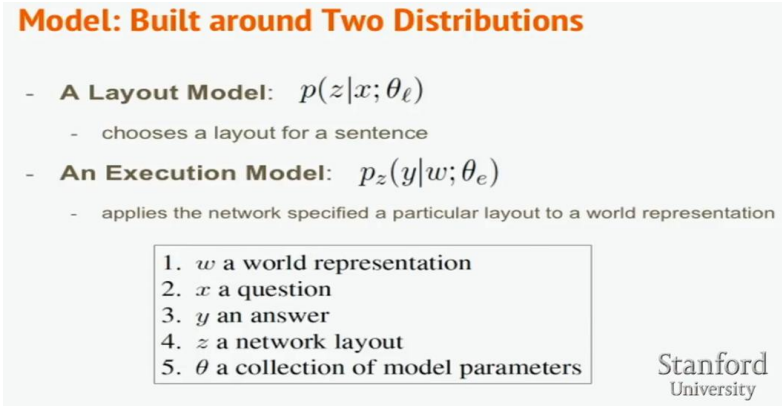




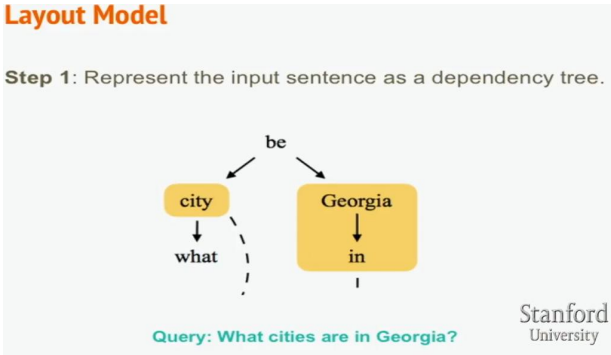
4 种结构单元



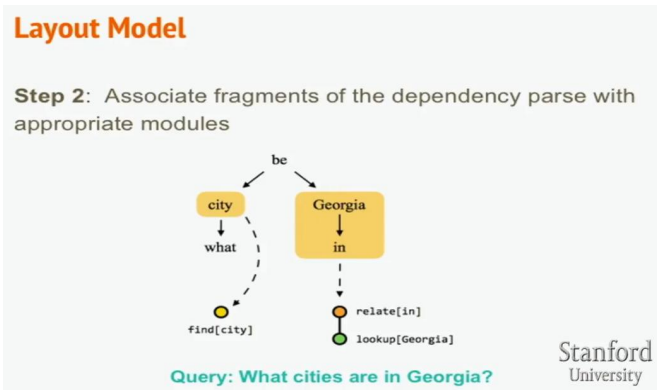
对每个问题**结构化(layout)**



将 layout 给 ANN 然后得到 answer y

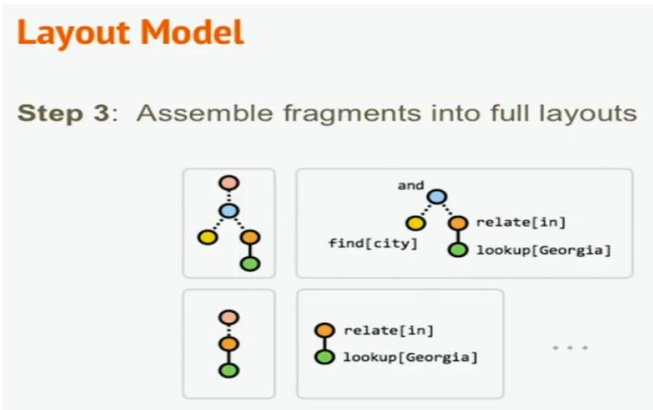


dependency tree



然后跟 module 建立联系

例如：find module associate with city



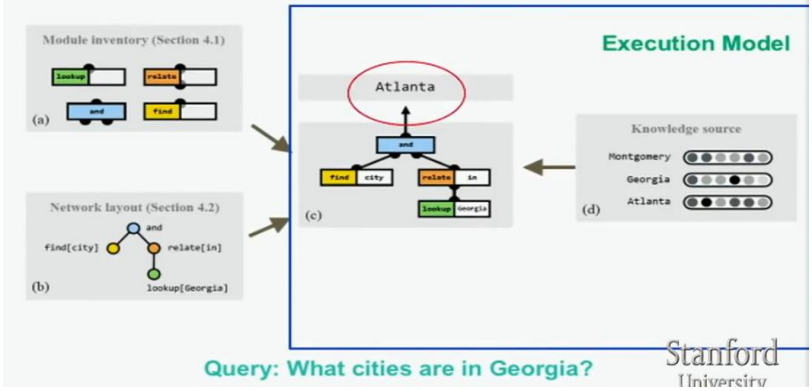
建立 layout，一个 sentence 可能多个 layout

Layout Scoring Model

- Produce an LSTM representation of the question, a feature-based representation of the query, and pass both representations through a multilayer perceptron

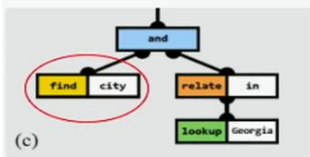
- The update to the layout-scoring model at each timestep is simply **the gradient of the log-probability of the chosen layout, scaled by the accuracy of that layout's predictions**

Execution Model



将 layout 给 ANN + knowledge base 然后得到 answer y

Module: find



Find (→ Attention)

`find[i]` computes a distribution over indices by concatenating the parameter argument with each position of the input feature map, and passing the concatenated vector through a MLP:

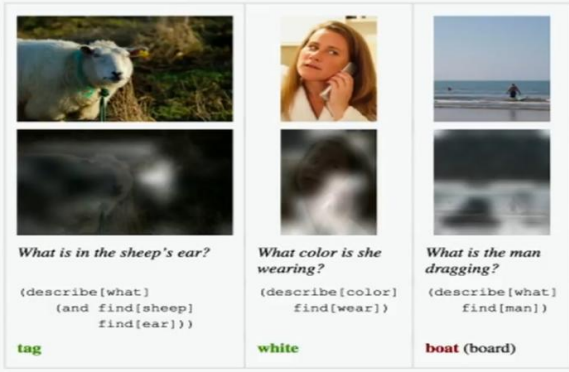
$$[\![\texttt{find}[i]]\!] = \text{softmax}(a \odot \sigma(Bv^i \oplus CW \oplus d)) \quad (3)$$

Train an Execution Model

- Maximize $\sum_{(w,y,z)} \log p_z(y|w;\theta_e)$

1. w a world representation
2. x a question
3. y an answer
4. z a network layout
5. θ a collection of model parameters

State-of-the-art Performance: VQA



State-of-the-art Performance: GeoQA

Is Key Largo an island?
(exists (and lookup[key-largo] find[island]))
yes: correct

What national parks are in Florida?
(and find[park] (relate[in] lookup[florida]))
everglades: correct

What are some beaches in Florida?
(exists (and lookup[beach] (relate[in] lookup[florida])))
yes (daytona-beach): wrong parse

What beach city is there in Florida?
(and lookup[beach] lookup[city] (relate[in] lookup[florida]))
[none] (daytona-beach): wrong module behavior

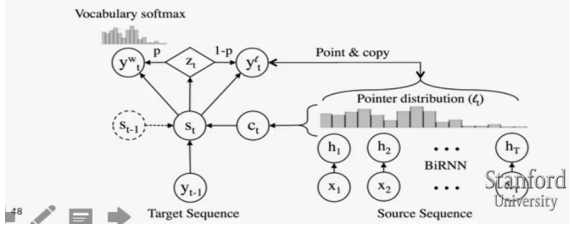
Model	Accuracy GeoQA	GeoQA+Q
LSP-F	48	–
LSP-W	51	–
NMN	51.7	35.7
D-NMN	**54.3**	**42.9**

4. Brief Interlude: Models with a pointer/copying

Recall the Pointer Sentinel Mixture Models (Merity et al. 2017) that Richard mentioned a few weeks ago

Copying/pointer networks

- Gulcehre, Ahn, Nallapati, Zhou, Bengio (2016) Pointing the Unknown Words



右边是 encoding

左边 decoding　生成 yt　yt-1 是 input

1 pointer 是 attention　然后 softmax

2 copy 是 **y 直接输出 source word　从 input sentence 里 copy words　不经过 softmax**

可以 copy 一些生僻字

用 p 和 1-p 来折中

Copying/pointer networks

- MT:

Table 5: Europarl Dataset (EN-FR)

	BLEU-4
NMT	20.19
NMT + PS	**23.76**

- Effective in tasks like summarization as well
- Caution from Google NMT paper: 'In principle can train a "copy model" but this approach is both unreliable at scale – the attention mechanism is unstable when the network is deep – and copying may not always be the best strategy for rare words – sometimes transliteration is more appropriate'

Stanford University

对于 summarize task 效果好

5. Below the word: Writing systems

Most deep learning NLP work begins with language in its written form – it's the easily processed, found data

But human language writing systems aren't one thing!

- Phonemic (maybe digraphs) jiyawu ngabulu
- Fossilized phonemic thorough failure
- Syllabic/moraic つぃりくぇしくゎ
- Ideographic (syllabic) 去年太空船二号坠毁
- Combination of the above インド洋の島

Stanford University

50

各国语言写的模式不同

Writing systems

Writing systems vary in how they represent words – or don't

- No word segmentation 美国关岛国际机场及其办公室均接获
- Words segmented
  - Clitics?
    - Separated **Je vous ai apporté** des bonbons
    - Joined فقلناها = ف+ قال+ نا+ها = so+said+we+it
  - Compounds?
    - Separated life insurance company employee
    - Joined Lebensversicherungsgesellschaftsangestellter

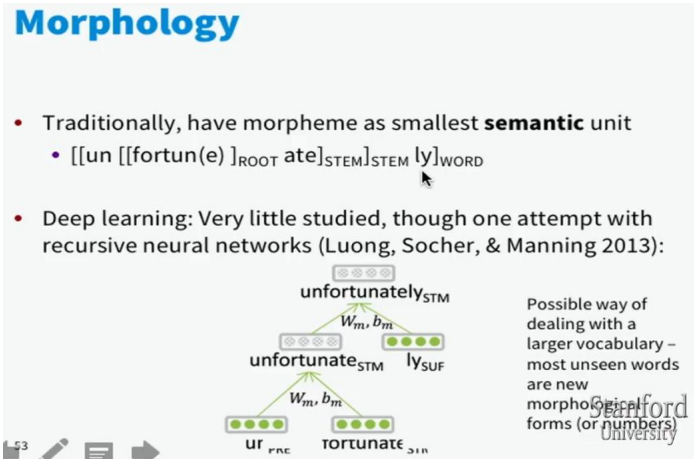
Stanford

Models below the word level

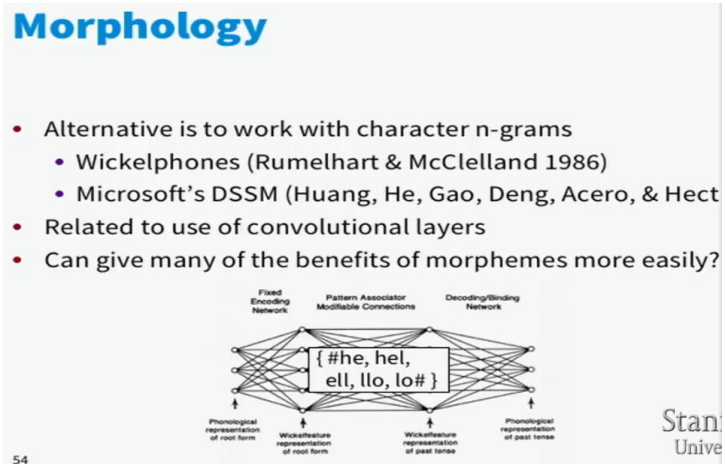
- Need to handle **large, open vocabulary**
  - Rich morphology: nejneobhospodařovávatelnějšímu ("to the worst farmable one")
  - Informal spelling: goooooood morning !!!!!
  - Transliteration: Christopher ↦ Kryštof

morphology 形态不同

拼写不同，含义不同

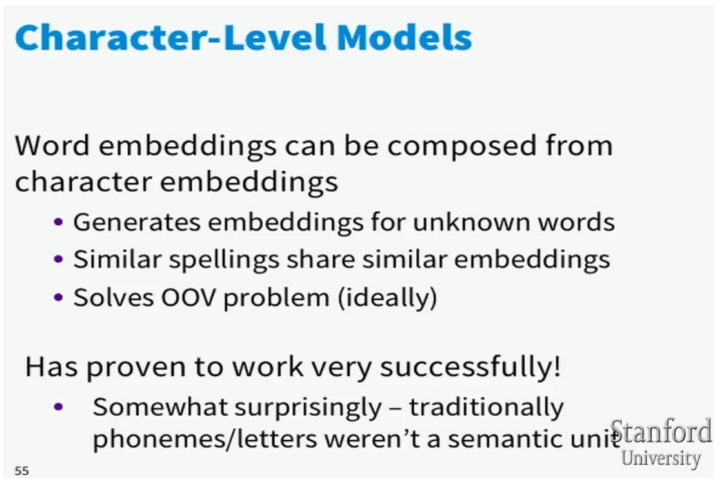


很少研究



或者 **n-gram　　CNN**

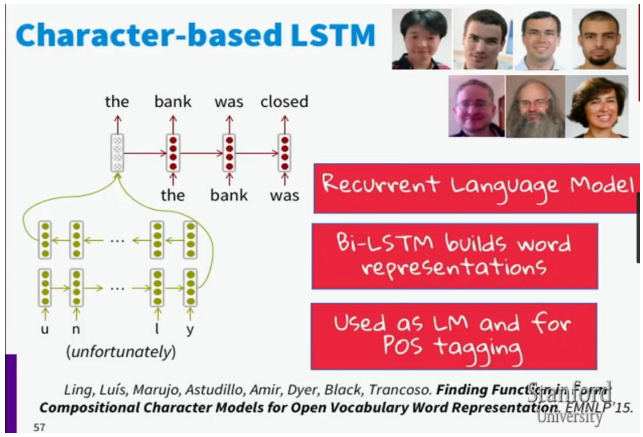
将所有 word 转成 n-gram 的集合，得到每个 gram 的 represent，最终得到 word 的结构



1 如果用 character embedding 可以更好地 generate to unknown words

2 相似拼写的 word 有相似的意义

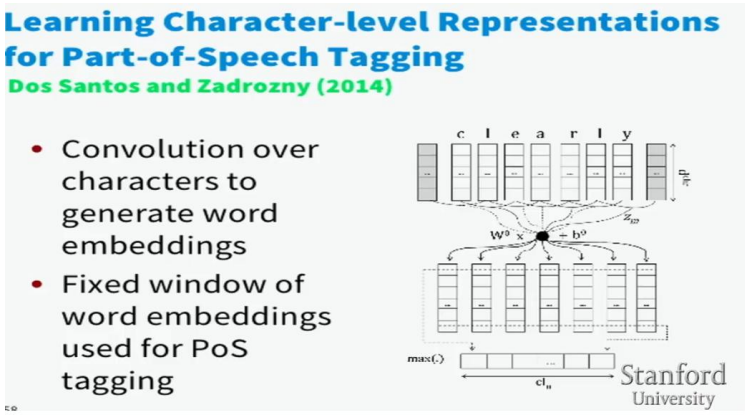
效果非常好！很意外，因为 character 本身没有什么意义 u　　n un 才有意义，到结果确实好

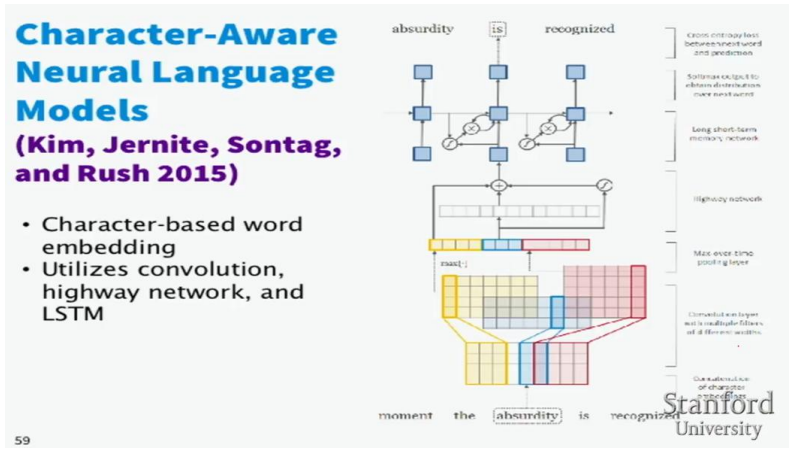


learn **character representation**

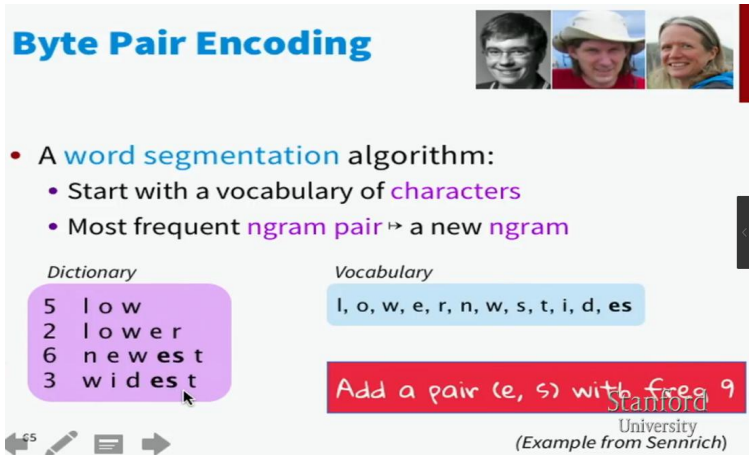
第一层 lstm 是 character level，最后一个隐层得到 **word representation**

再给另一个 LSTM。有具体任务例如： predict sequence of word

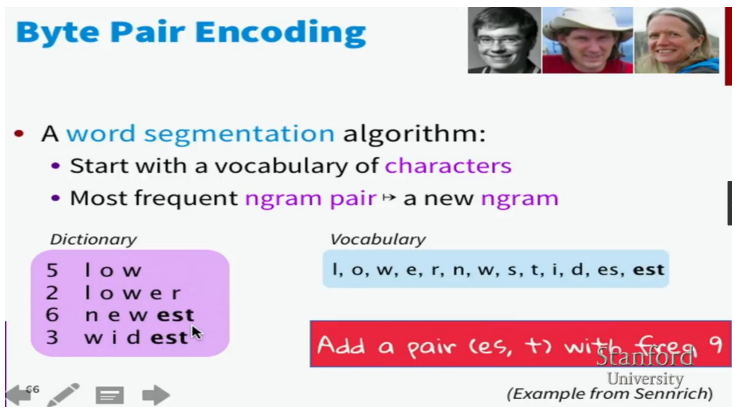




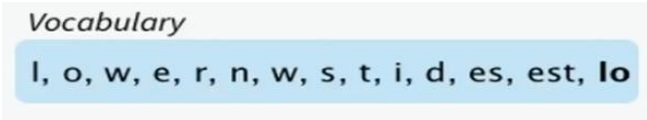
sub-word=character



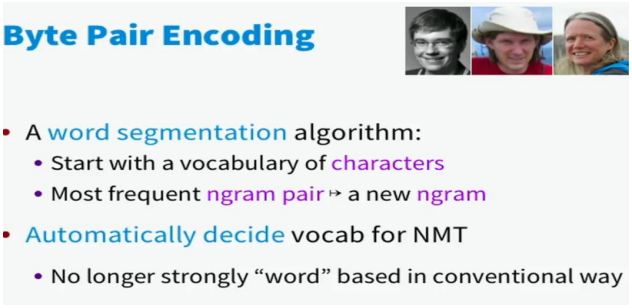
从 n=1 开始　es 是 1-gram pair 频率最高的
加入字典



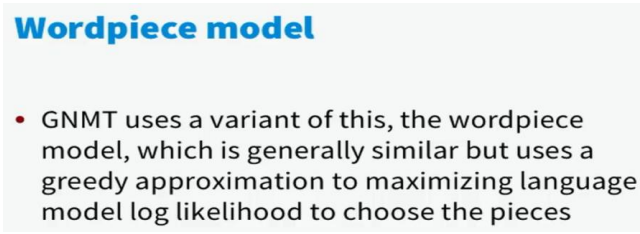
est=es , t 频率最高，加入 est

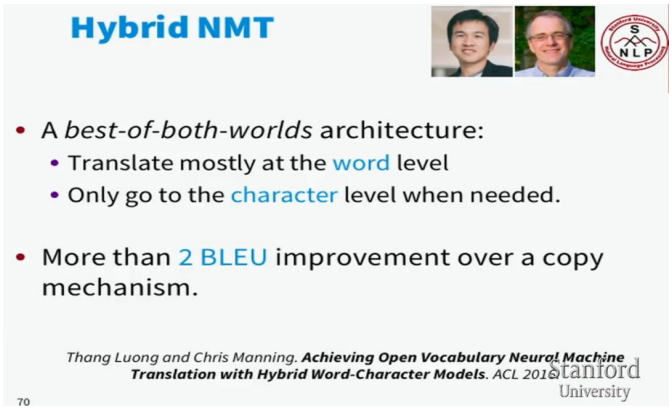


直到 len of 字典达到我们的**长度要求**
自动决定 vocabulary

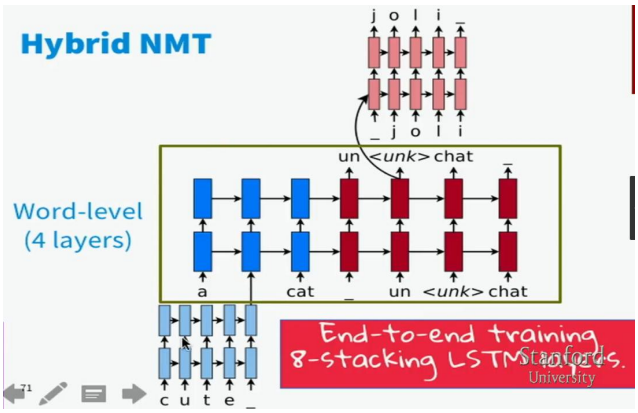


非常有效的方法来 maintain small vocab！！！不需要每个 word
而是 word 的基本词根 word piece 碎片
对于 unknown word 也不怕

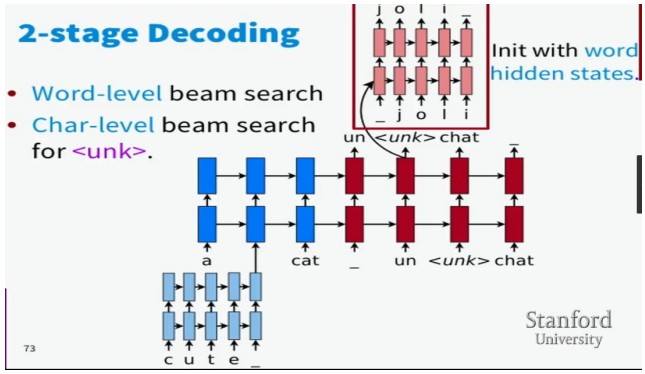




pure character model 还是计算量太大 太慢了
得到的 word represent 还是有 similarity 可能会稍微逊色于直接 word vector



end2end train
中间是 word level 翻译 LSTM
如果词汇里没有 word 就用 character embedding input 来生成 **word represent**
同样输出时如果是 unknow，word embedding 也可以输出 **character output** **bean search**


2-stage Decoding
Word-level beam search
Char-level beam search for <unk>.
Init with word hidden states.
j o l i _
_ j o l i
un <unk> chat
a cat _ un <unk> chat
c u t e _
73
Stanford University

18 待补充