

# Language modelling

# Language Modeling

- We want to compute  $P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$ , the probability of a sequence
- Alternatively we want to compute  $P(w_5 | w_1, w_2, w_3, w_4)$ : the probability of a word given some previous words. E.g. continue the sentence ..
- This is the .. House? Did?
- The model that computes  $P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$  is called the **language model**.

# application

- Text generation
  - Suggestions in messengers (automatic email reply)
  - Spell correction
  - MT (generation of destination text)
  - Speech recognition
  - Hand written recognition...

## Very Easy Estimate

- How to estimate?
  - $P(\text{the} \mid \text{the water is so transparent that})$

$P(\text{the} \mid \text{the water is so transparent that})$

=

$\text{Count}(\text{the water is so transparent that the})$

---

$\text{Count}(\text{the water is so transparent that})$

## Very Easy Estimate

- According to Google those counts are 5/9.
  - Unfortunately... 2 of those are to these slides... So its really 3/7
- What if the phrase turns to:

‘The water of Walden pond is so transparent that’

# Computing $P(W)$

- How to compute this joint probability:
  - $P(\text{"the", "other", "day", "I", "was", "walking", "along", "and", "saw", "a", "lizard"})$
- Intuition: let's rely on the Chain Rule of Probability

# The Chain Rule

- Recall the definition of conditional probabilities

$$P(A | B) = \frac{P(A \wedge B)}{P(B)}$$

- Rewriting:

$$P(A \wedge B) = P(A | B)P(B)$$

- More generally
- $P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$
- In general
- $P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1 \dots x_{n-1})$

# The Chain Rule

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned}$$

- It is normalized separately for each seq length!
- $P(\text{“the big red dog was”}) =$
- $P(\text{the}) * P(\text{big}|\text{the}) * P(\text{red}|\text{the big}) * P(\text{dog}|\text{the big red}) * P(\text{was}|\text{the big red dog})$



# Unfortunately

- There are a lot of possible sentences
- Language is creative
- In general, we'll never be able to get enough data to compute the statistics for those long prefixes
- $P(\text{lizard}|\text{the,other,day,I,was,walking,along,and, saw,a})$

# Markov Assumption

- Make the simplifying assumption
  - $P(\text{lizard}|\text{the,other,day,I,was,walking,along,and,saw,a}) = P(\text{lizard}|\text{a})$
- Or maybe
  - $P(\text{lizard}|\text{the,other,day,I,was,walking,along,and,saw,a}) = P(\text{lizard}|\text{saw,a})$
- Or maybe... You get the idea.

# N-gram LM- Markov Assumption

So for each component in the product replace with the approximation (assuming a prefix of N)

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

# Estimating bigram probabilities

- The Maximum Likelihood Estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{\textit{count}(w_{i-1}, w_i)}{\sum_w \textit{count}(w_{i-1}, w)}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

## An example

- $\langle s \rangle$  I am Sam  $\langle /s \rangle$
- $\langle s \rangle$  Sam I am  $\langle /s \rangle$
- $\langle s \rangle$  I do not like green eggs and ham  $\langle /s \rangle$

$$\begin{array}{lll}
 P(I | \langle s \rangle) = \frac{2}{3} = .67 & P(\text{Sam} | \langle s \rangle) = \frac{1}{3} = .33 & P(\text{am} | I) = \frac{2}{3} = .67 \\
 P(\langle /s \rangle | \text{Sam}) = \frac{1}{2} = 0.5 & P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 & P(\text{do} | I) = \frac{1}{3} = .33
 \end{array}$$

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

# Maximum Likelihood Estimates

- Computing **relative frequency**: dividing observed frequency of a particular sequence by the observed frequency of a prefix.
- The maximum likelihood estimate of some parameter of a model  $M$  from a training set  $T$ 
  - maximizes the likelihood of the training set  $T$  given the model  $M$

- Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be “Chinese”
- MLE estimate is  $400/1000000 = .004$ 
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that “Chinese” will occur 400 times in a million word corpus.

# Berkeley Restaurant Project Sentences

- *can you tell me about any good cantonese restaurants close by*
- *mid priced thai food is what i'm looking for*
- *tell me about chez panisse*
- *can you give me a listing of the kinds of food that are available*
- *i'm looking for a good place to eat breakfast*
- *when is caffe venezia open during the day*



# Raw Bigram Counts

- Out of 9332 sentences: Count(col | row)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

# Raw Bigram Probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

# Bigram Estimates of Sentence Probabilities

- $P(< s > \text{ I want english food } < / s >) =$   
 $p(i | < s >) \times p(\text{want} | \text{I}) \times p(\text{english} | \text{want}) \times$   
 $p(\text{food} | \text{english}) \times p(< / s > | \text{food})$   
 $= .000031$

# Kinds of knowledge?

- $P(\text{english}|\text{want}) = .0011$
- $P(\text{chinese}|\text{want}) = .0065$
- $P(\text{to}|\text{want}) = .66$
- $P(\text{eat} | \text{to}) = .28$
- $P(\text{food} | \text{to}) = 0$
- $P(\text{want} | \text{spend}) = 0$
- $P(i | \langle s \rangle) = .25$

• World  
knowledge

• Syntax

• Discourse

$N=?$

- Trigrams are usually the most efficient.

# The Shannon Visualization Method

- Generate random sentences:
- Choose a random bigram  $\langle s \rangle, w$  according to its probability
- Now choose a random bigram  $(w, x)$  according to its probability
- And so on until we choose  $\langle /s \rangle$
- Then string the words together
- $\langle s \rangle$  I  
I want  
want to  
to eat  
eat Chinese  
Chinese food  
food  $\langle /s \rangle$

# Shakespeare

Unigram	<ul style="list-style-type: none"> <li>• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have</li> <li>• Every enter now severally so, let</li> <li>• Hill he late speaks; or! a more to leg less first you enter</li> <li>• Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like</li> </ul>
Bigram	<ul style="list-style-type: none"> <li>• What means, sir. I confess she? then all sorts, he is trim, captain.</li> <li>• Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.</li> <li>• What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?</li> <li>• Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt</li> </ul>
Trigram	<ul style="list-style-type: none"> <li>• Sweet prince, Falstaff shall die. Harry of Monmouth's grave.</li> <li>• This shall forbid it should be branded, if renown made it empty.</li> <li>• Indeed the duke; and had a very good friend.</li> <li>• Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.</li> </ul>
Quadrigram	<ul style="list-style-type: none"> <li>• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;</li> <li>• Will you not tell me who I am?</li> <li>• It cannot be but so.</li> <li>• Indeed the short and the long. Marry, 'tis a noble Lepidus.</li> </ul>

# Shakespeare as corpus

- $N=884,647$  tokens,  $V=29,066$
- Shakespeare produced 300,000 bigram types out of  $V^2= 844$  million possible bigrams: so, 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare



# The Wall Street Journal is Not Shakespeare

*unigram:* Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

*bigram:* Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

*trigram:* They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

# Unknown words: Open versus closed vocabulary tasks

- **If we know all the words in advanced**
  - Vocabulary  $V$  is fixed
  - Closed vocabulary task
- **Often we don't know this**
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- **Instead: create an unknown word token <UNK>**
  - Training of <UNK> probabilities
    - Create a fixed lexicon  $L$  of size  $V$
    - At text normalization phase, any training word not in  $L$  changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

# Train model

- Log-likelihood maximization (easier to compute log of sum instead of product of probabilities)

$$\log p(\mathbf{w}_{\text{train}}) = \sum_{i=1}^{N+1} \log p(w_i | w_{i-n+1}^{i-1}) \rightarrow \max$$

- estimate for param

$$p(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i)}{\sum_{w_i} c(w_{i-n+1}^i)} = \frac{c(w_{i-n+1}^i)}{c(w_{i-n+1}^{i-1})}$$

- (N is length of corpus)

# Evaluation

- We train parameters of our model on a **training set**.
- How do we evaluate how well our model works?
- We look at the models performance on some new data
- This is what happens in the real world; we want to know how our model performs on data we haven't seen
- So a **test set**. A dataset which is different than our training set

# Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in an application (e.g. speech recognizer, MT, ...)
  - Run recognition, get word error rate (WER) for A
  - Put model B in speech recognition, get word error rate for B
  - Compare WER for A and B
  - **Extrinsic evaluation**

# Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **just** like the training data
  - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**
  - **But is helpful to think about.**

# Perplexity-is our model surprised with a real text

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

$$\begin{aligned}\text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}\end{aligned}$$

- Chain rule:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

- For bigrams:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

- Minimizing perplexity is the same as maximizing probability
  - ◆ **The best language model is one that best predicts an unseen test set**

# Smoothing te

- 1- laplacian smoothing
- Add 1 to the counts (add-one smoothing)

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i) + 1}{c(w_{i-n+1}^{i-1}) + V}$$

- Or tune k (add-k smoothing)

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i) + k}{c(w_{i-n+1}^{i-1}) + V k}$$



- 2- katz backoff
- Larger n-grams are better, but data is not always enough. => try longer n-grams and back off to shorter if needed

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \begin{cases} \tilde{p}(w_i | w_{i-n+1}^{i-1}), & \text{if } c(w_{i-n+1}^i) > 0 \\ \alpha(w_{i-n+1}^{i-1}) \hat{p}(w_i | w_{i-n+2}^{i-1}), & \text{otherwise} \end{cases}$$

- $\tilde{p}$ ,  $\alpha$  are used for normalization

- 3- interpolation smoothing: (tune params on a dev set)

$$\hat{p}(w_i|w_{i-2}w_{i-1}) = \lambda_1 p(w_i|w_{i-2}w_{i-1}) + \lambda_2 p(w_i|w_{i-1}) + \lambda_3 p(w_i)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

- 3-1- absolute discounting
- Subtract 0.75 and get a good estimate of test count (empirical)

$$\hat{p}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i) - d}{\sum_x c(w_{i-1}x)} + \lambda(w_{i-1})p(w_i)$$

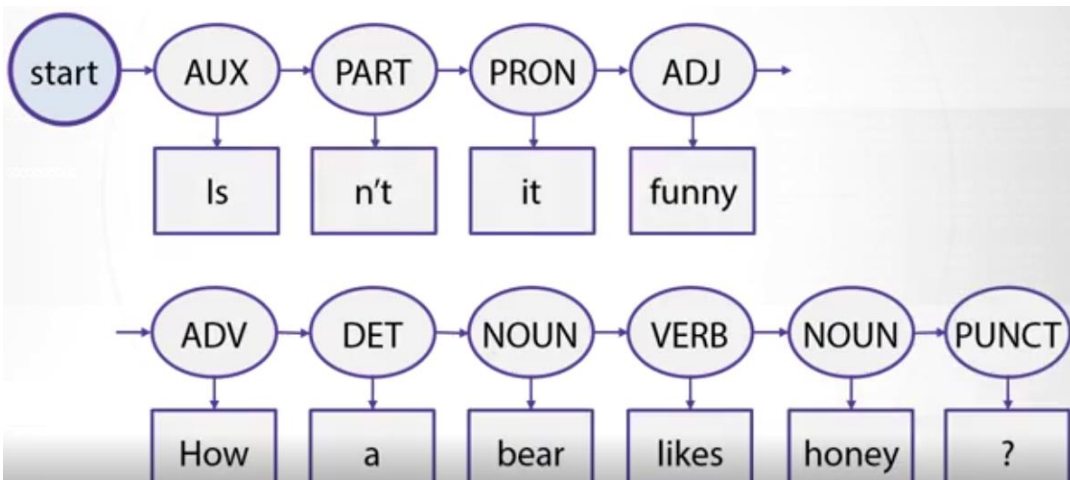
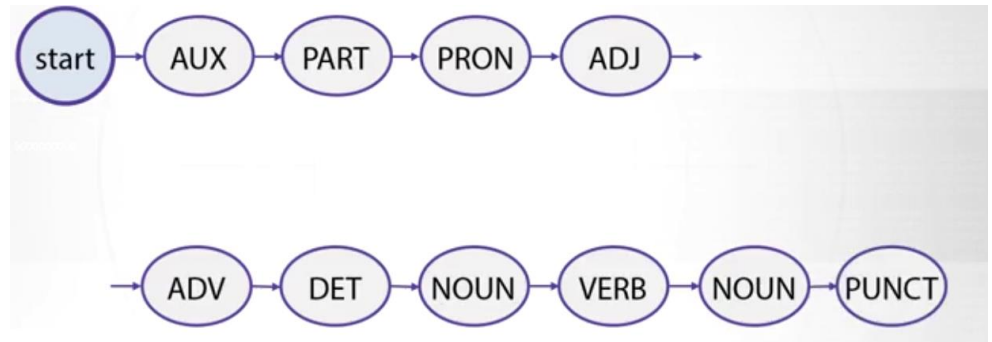
- 3-2- kneser

- 3-2-kneser-ney smoothing (the most popular technique)
- Idea: unigram distributions captures the word frequency
- We need to capture the diversity of contexts for the word

$$\hat{p}(w) \propto |\{x : c(x w) > 0\}|$$

# HMM language model

- Choose the next POS tag given the previous tag
- Given the current tag, generate another word
- => neighbouring words don't depend on each other and depend on the tag.



# Train- supervised: MLE

$$a_{ij} = p(s_j | s_i) = \frac{\sum_{t=1}^T [y_{t-1} = s_i, y_t = s_j]}{\sum_{t=1}^T [y_t = s_i]}$$

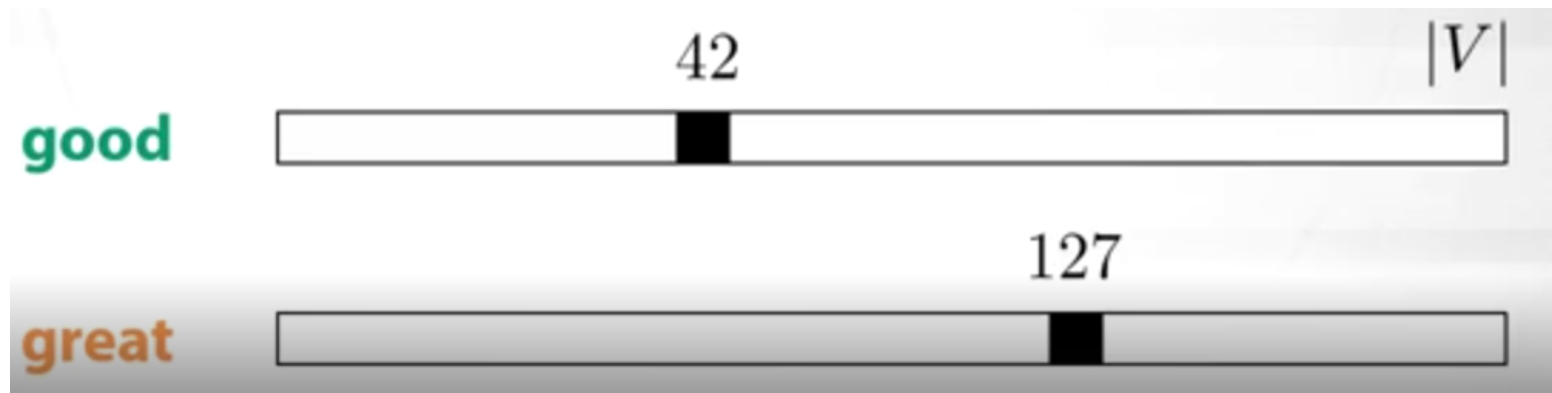
# Baum-welch

- E-step: posterior prob for hidden variables
- $P(y_{t-1}=s_i, y_t=s_j)$
- Effectively done by DP forward-backward algorithm
- M-step: maximum likelihood updates for the parameters

$$a_{ij} = p(s_j | s_i) = \frac{\sum_{t=1}^T [y_{t-1} = s_i, y_t = s_j]}{\sum_{t=1}^T [y_t = s_i]}$$

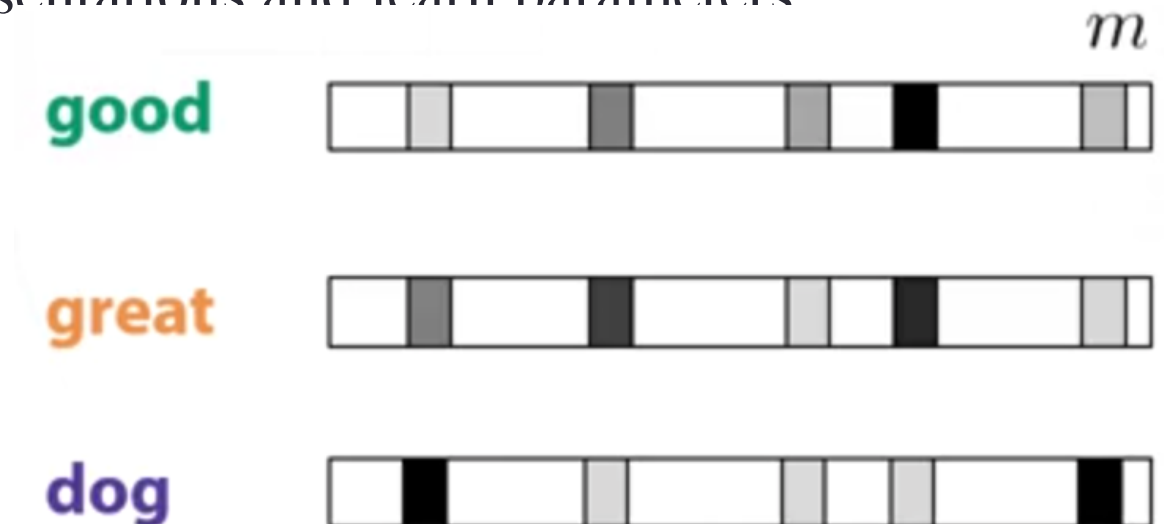
# Curse of dimensionality- Neural Language model

- Imagine you have seen the following many times
  - Have a good day
- However , you have not seen the following
  - Have a great day
- What happens then (even with smoothing)?



# Distributed representation-better generalization

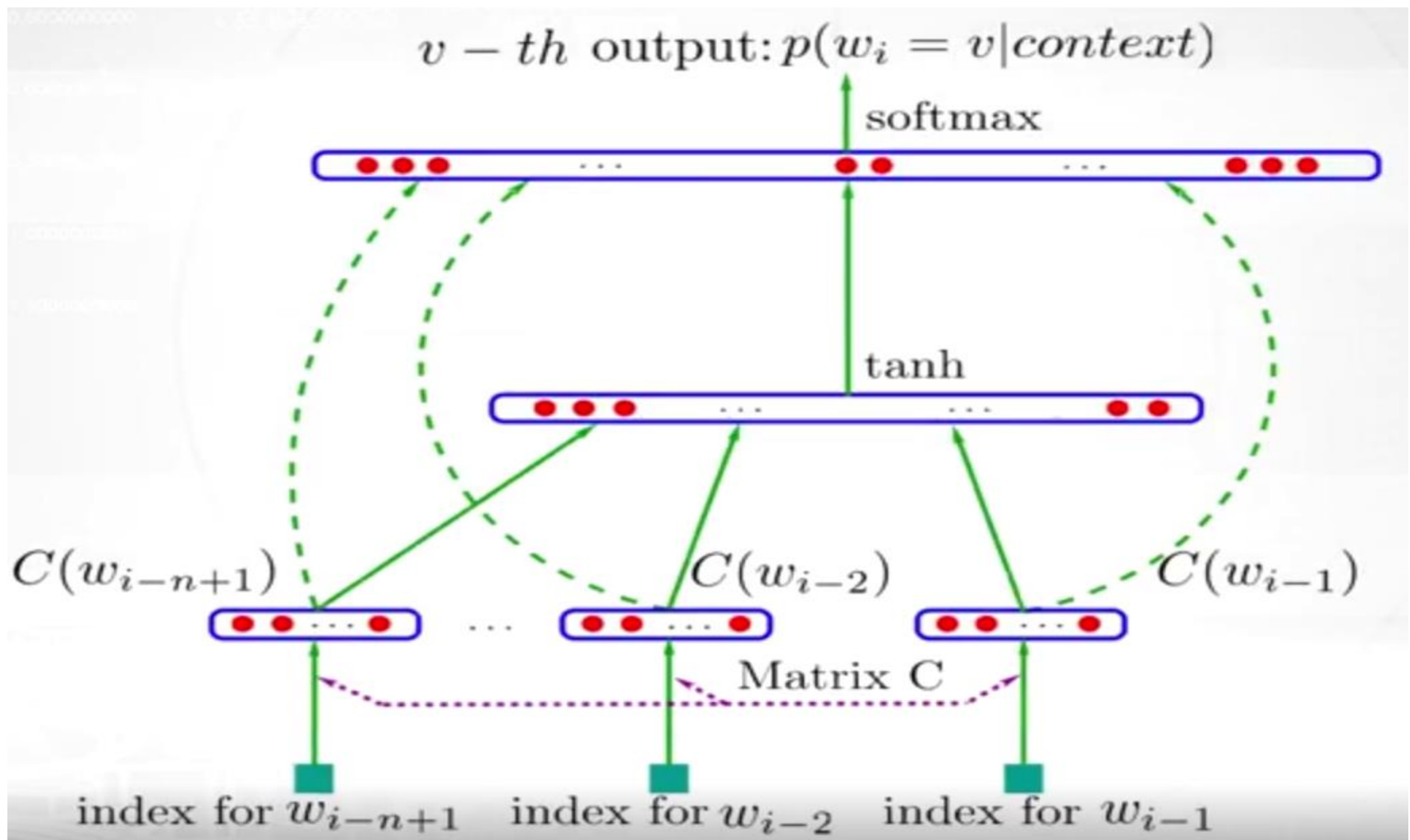
- Learn distributed representations of words. Express probabilities of sequences in terms of these distributed representations and learn parameters



- $C|V|*m$  – matrix of Distributed word representations



# NLM



- A neural probabilistic language model, 2003

$$p(w_i | w_{i-n+1}, \dots, w_{i-1}) = \frac{\exp(y_{w_i})}{\sum_{w \in V} \exp(y_w)}$$

**Softmax over components of  $y$**

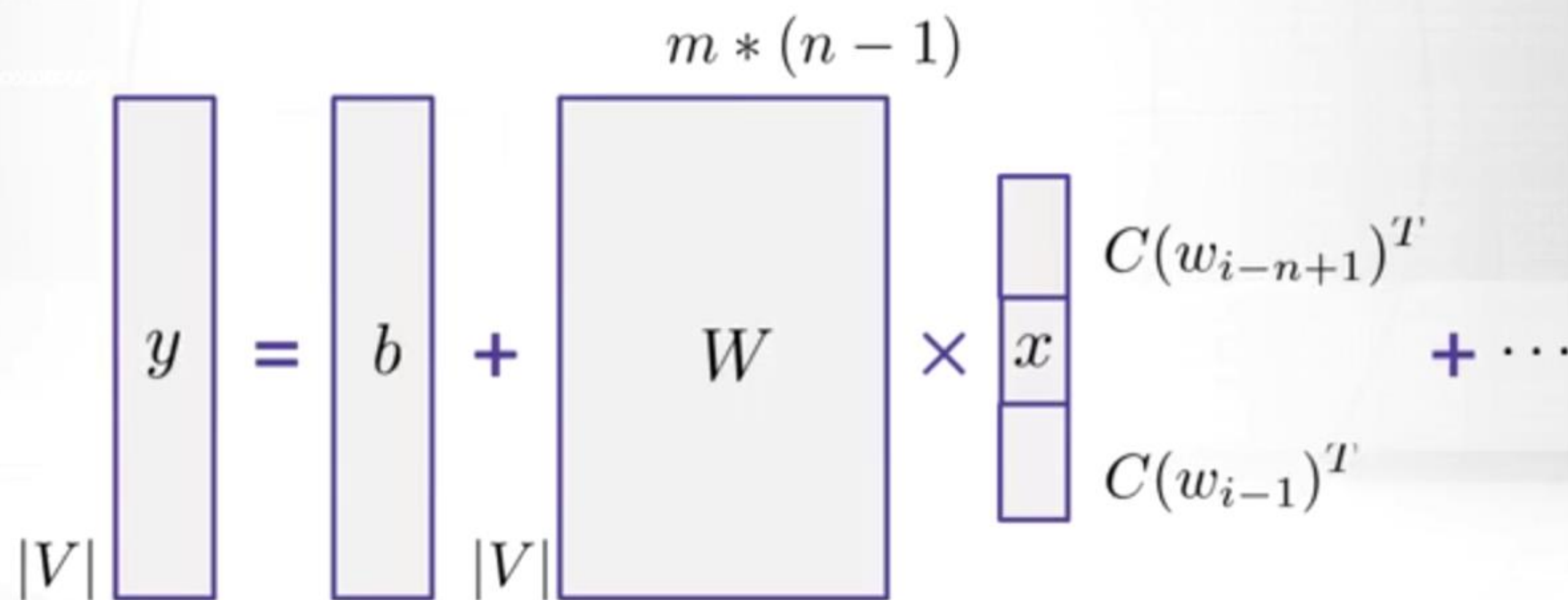
$$y = b + Wx + U \tanh(d + Hx)$$

**Feed-forward NN with tons of parameters**

$$x = [C(w_{i-n+1}), \dots, C(w_{i-1})]^T$$

**Distributed representation of context words**

$$y = b + Wx + U \tanh(d + Hx)$$



# Log-bilinear LM

- Has much less parameters and non-linear activations
- Measures similarity between the word and the context:

$$p(w_i | w_{i-n+1}, \dots, w_{i-1}) = \frac{\exp(\hat{r}^T r_{w_i} + b_{w_i})}{\sum_{w \in V} \exp(\hat{r}^T r_w + b_w)}$$

- Representation of word
- Representation of context

$$r_{w_i} = C(w_i)^T$$

$$\hat{r} = \sum_{k=1}^{n-1} W_k C(w_{i-k})^T$$

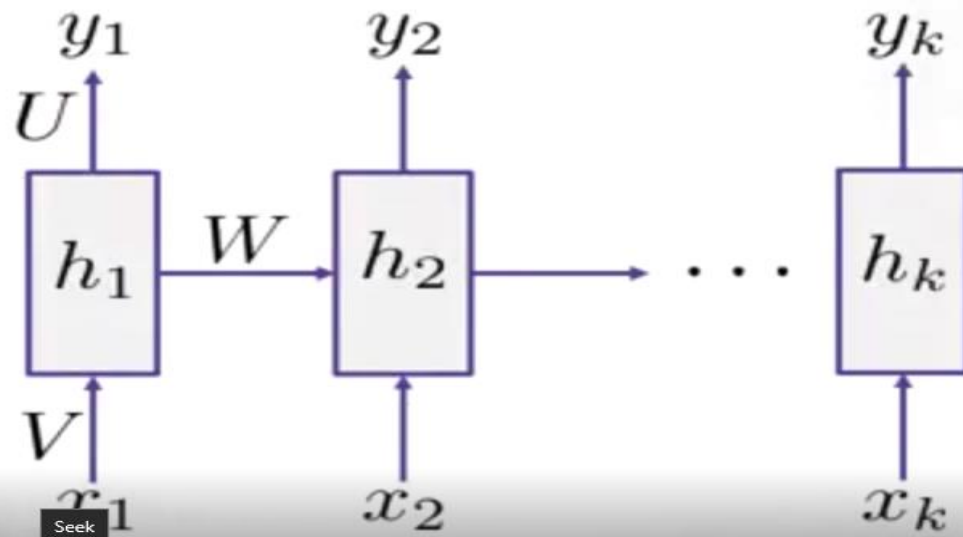
- Three new graphical models for statistical language modelling, 2007

# DL language model (RNN)

- RNN: Extremely popular architecture for any sequential data

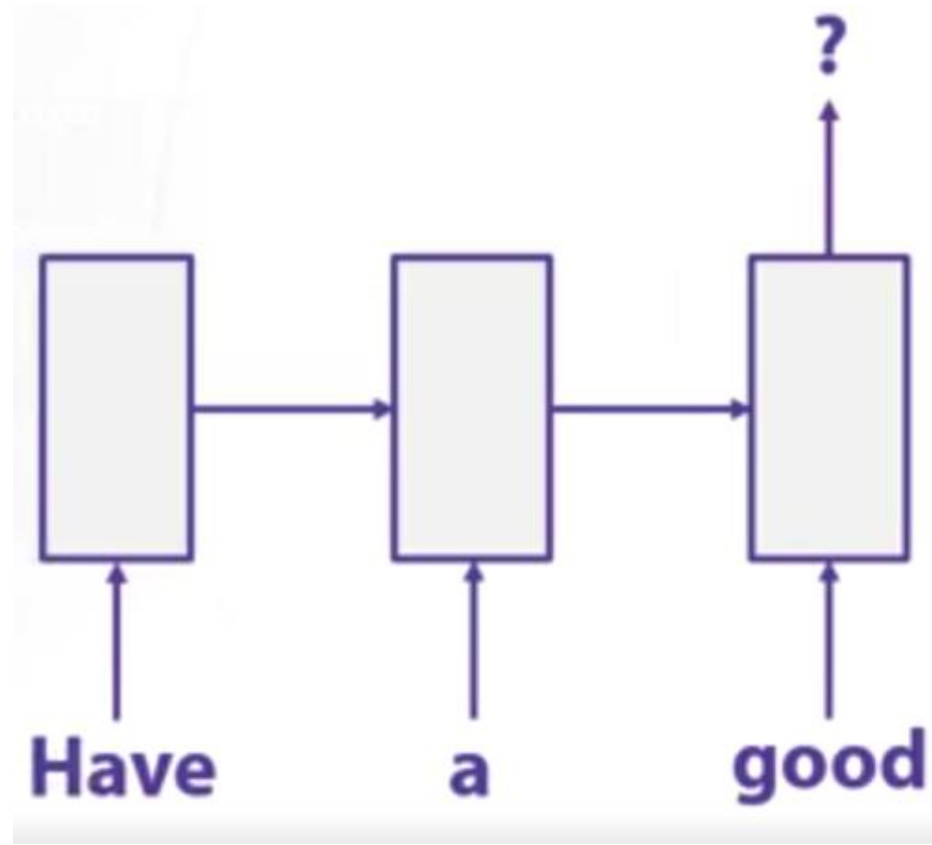
$$h_i = f(W h_{i-1} + V x_i + b)$$

$$y_i = U h_i + \tilde{b}$$



# RNN LM

- Predicts a next word based on a previous context
  - Architecture: use the current state output
  - Apply a linear layer on top
  - Do softmax to get probabilities
- 
- Recurrent NN based LM, 2010

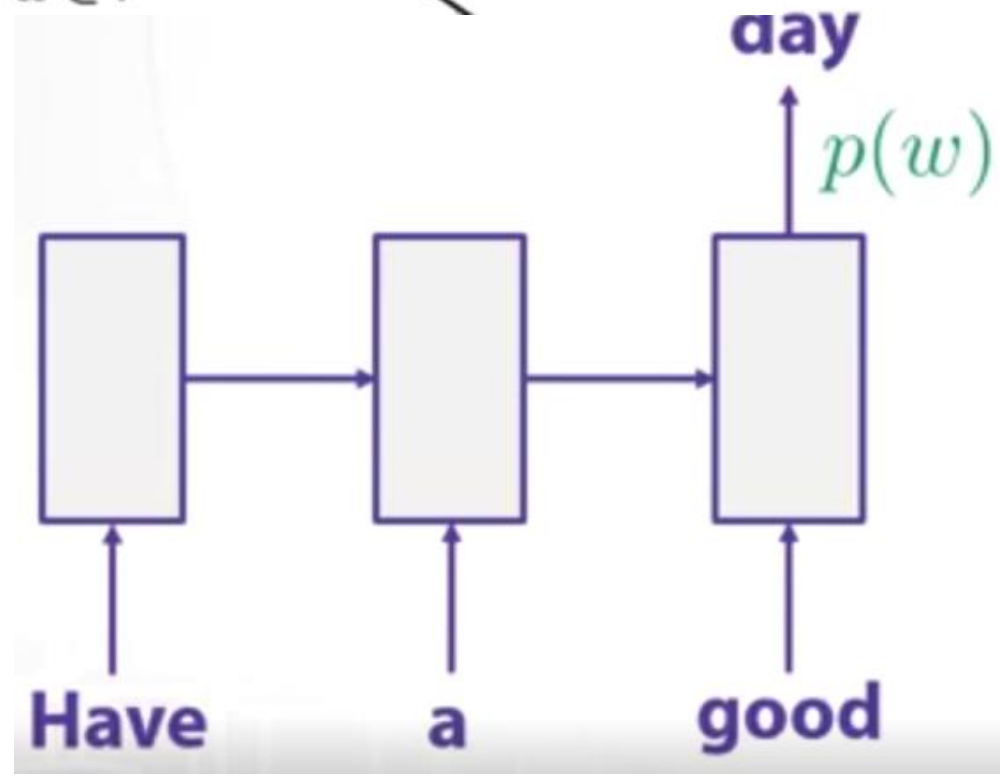


# train

- Cross-entropy loss for one position (W has only one non-zero)

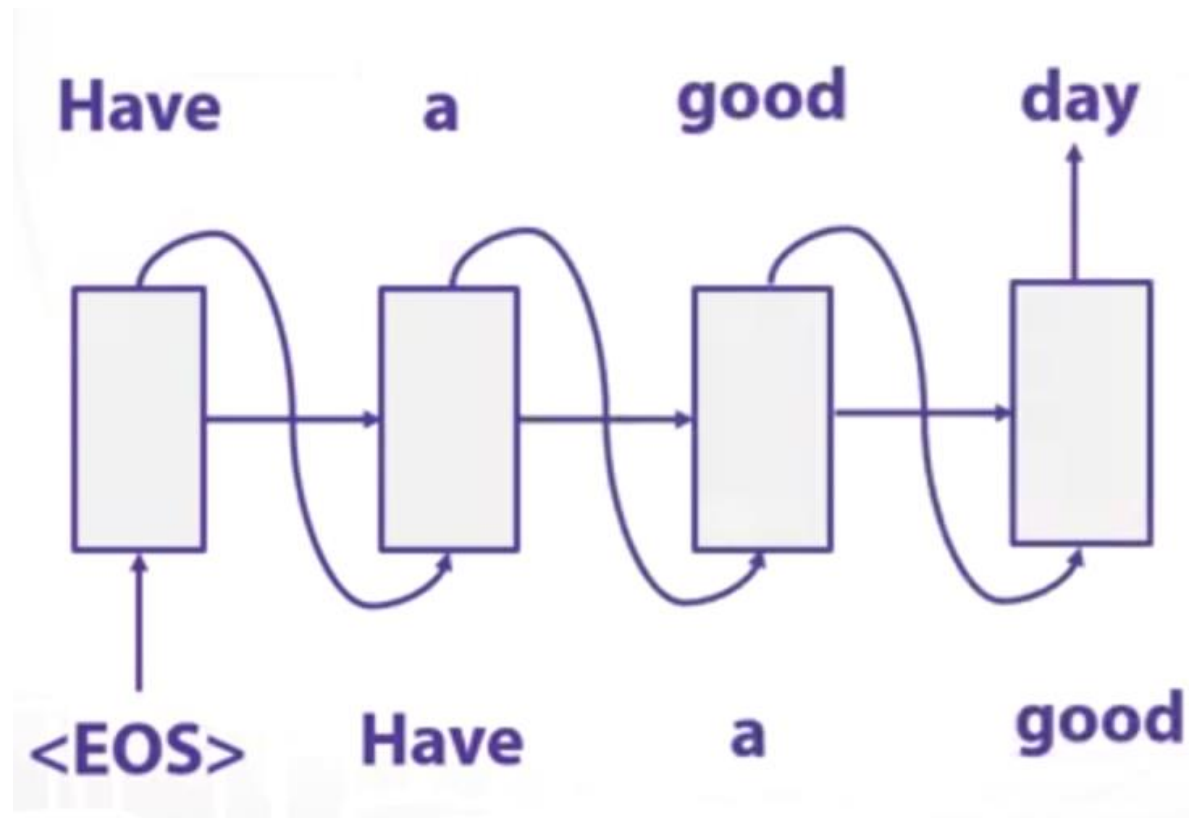
$$-\log p(w_i) = - \sum_{w \in V} [w = w_i] \log p(w)$$

- Target: word  $w_i$ . Output  $p(w)$



# Generate LM

- Idea:
- Feed the previous output as the next input
- Take argmax at each step (greedily) or use beam search





# Char-level RNN

# Cook your own LM

- Use LSTM or GRU and gradient clipping
- Start with one layer, then stack 3-4, use skip connections
- Use dropout for regularization
- Have a look into TF tutorial for a working model
- Tune learning rate schedule in SGF or use Adam
- Explore state-of-the-art improvements
- On the state-of-the-art of evaluation in NLM, 2017
- Regularizing and optimizing LSTM lang.model, 2017