Lecture 10-12: Language Models

USC VSoE CSCI 544: Applied Natural Language Processing

Jonathan May -- 梅約納

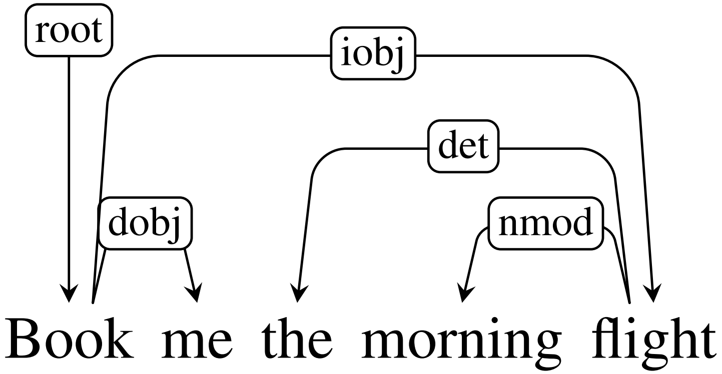
September 27-October 4, 2017

based on slides of Nathan Schneider / Sharon Goldwater

Quiz 1

- Which of the following CFG rules are in CNF (assume capital letters signify nonterminals, lowercase letters signify terminals)?
  - S -> NP VP
  - NP -> NP PP
  - NP -> DT JJ NN
  - NP -> the boy
  - NP -> PP
  - JJ -> red
  - NP -> the NP

Quiz 2



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RArc-dobj	book $\xrightarrow{\text{dobj}}$ me
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]		

- What is the next step?
  - shift
  - reduce
  - LArc
  - RArc
  - LArc-det
  - RArc-det
  - RArc-nmod
  - LArc-nmod

Please turn your homework ...

- What word comes next?
  - in
  - over
  - into
  - the
  - refrigerator
- What are the probabilities of each of these?
- And why should we care?

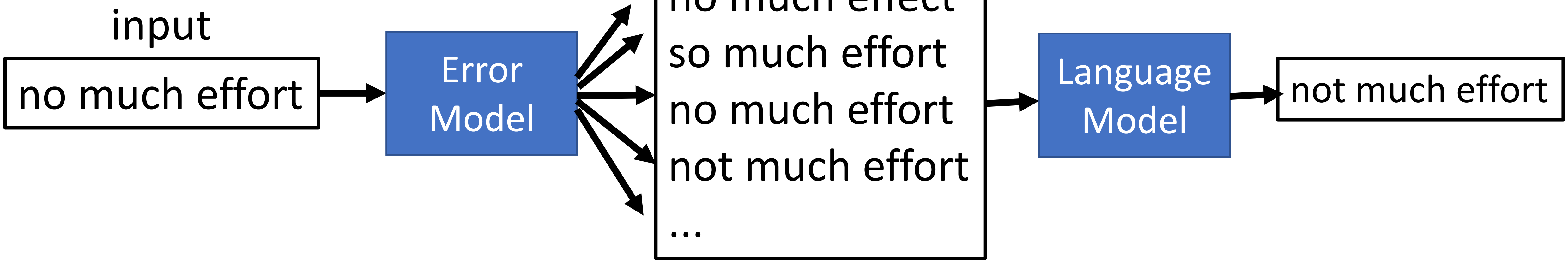
Probability of a sentence; P(s)

- How likely is it to occur?
- Colloquially, how likely is any speaker of language X to utter s?
  - P(the cat slept peacefully) > P(slept the peacefully cat)
  - P(she studies morphosyntax) > P(she studies more faux syntax)

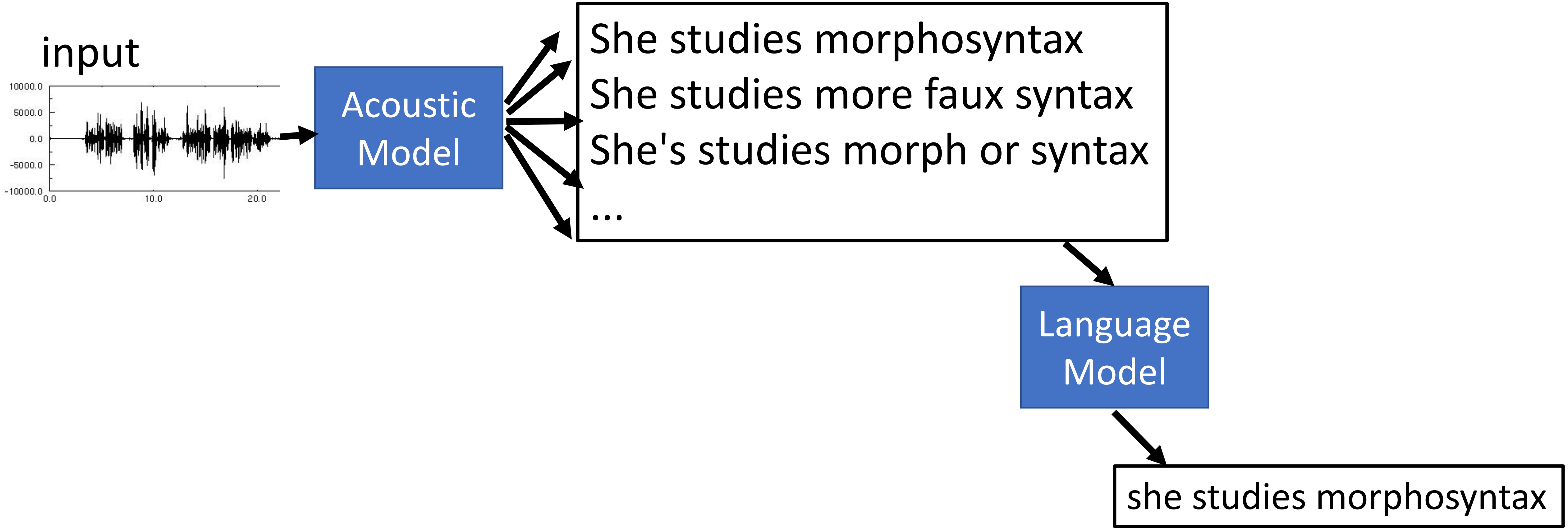
Language Models in NLP

- It's very difficult to know the true P(s) for an arbitrary sequence of words
- But we can define a language model that will give us good approximations
- Language models (LMs) are very useful whenever we are generating output
  - Machine Translation
  - Spelling Correction
  - Summarization
  - Speech Recognition
- There are some very good, easy-to-use toolkits for building and using LMs
  - SRILM: around since the 90s. not advised with >300m tokens
  - KenLM: preferred choice; great scalability in memory and time with even billions of tokens
  - NLTK has some LM training/using support (ok for prototyping)

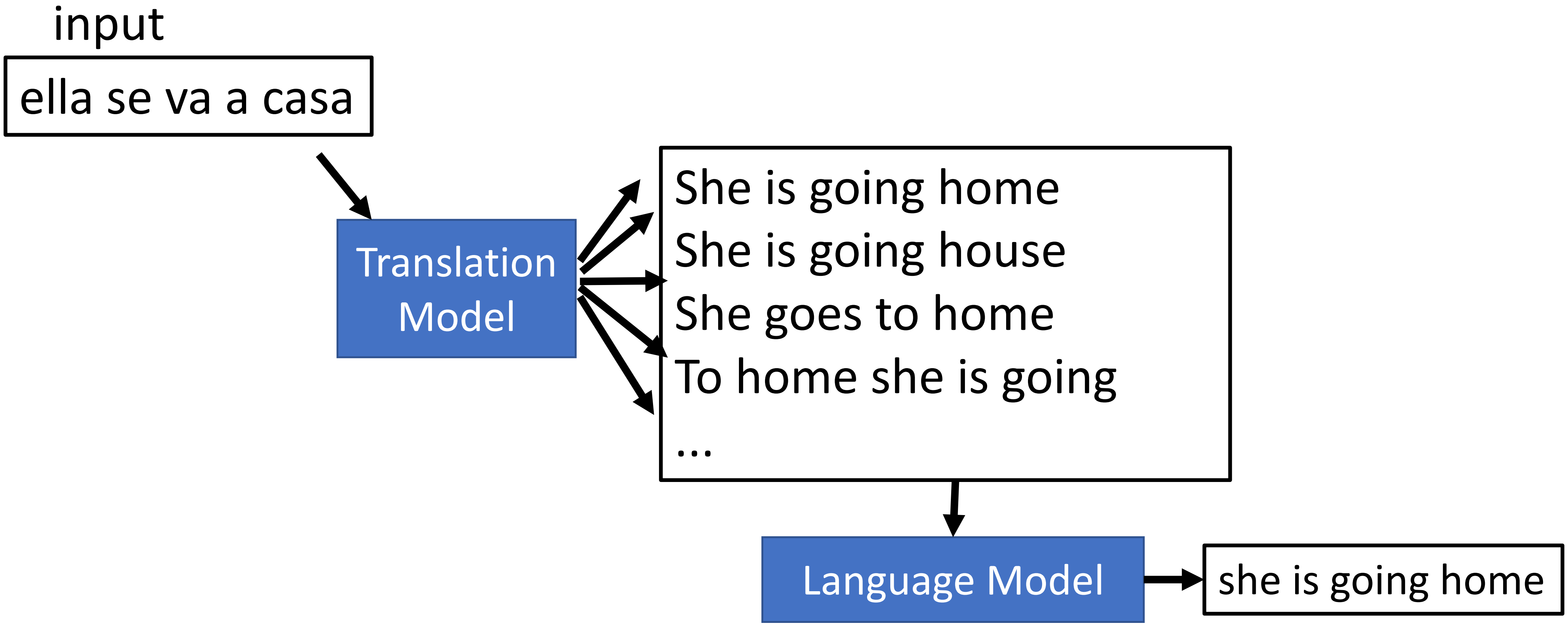
Use of Language Models: Spelling Correction



Use of Language Models: Speech Recognition



Use of Language Models: Machine Translation



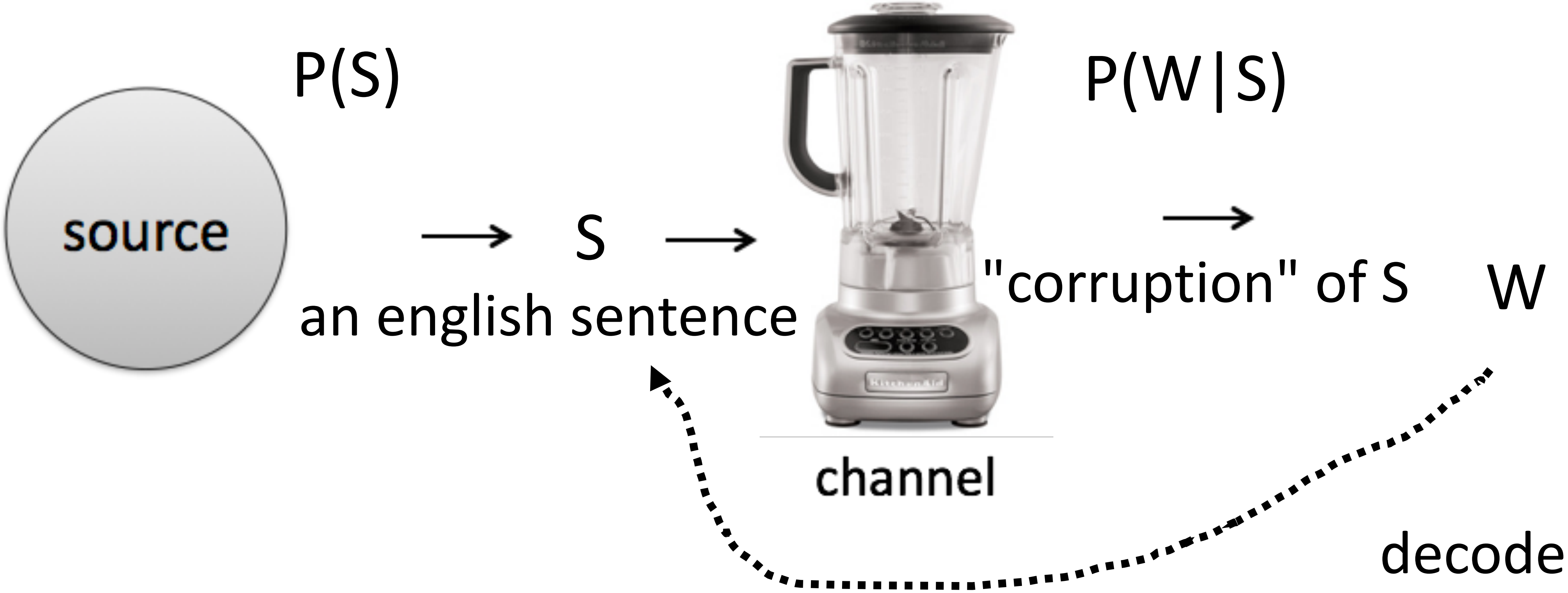
LMs for Prediction

- LMs can be used to <u>predict</u> what a human will do next, rather than <u>correct</u> a possibly faulty model
- Example: predictive text correction/completion on your phone
  - Keyboard is tiny, so it's easy to touch a spot slightly off from the letter you intend
  - Correct these errors as you go and also provide possible completions

i n e f f i cient

- In this case, LM may be defined over sequences of *characters* instead of (or in addition to) sequences of words

Noisy Channel Model



But How To Estimate These Probabilities?

- We want to to know the probability of word sequence **w** = w_1, w_2, ..., w_n occurring in English
- Assume we have some training data: large corpus of general English text
- We use this data to estimate the probability of **w** (even if we never see it in the corpus)

Bit of Notation: Random Variables

- Random Variable = variable that represents all the possible events in some partition of Ω
- So if X is a coin flip I can say P(X=heads) to mean probability the flip comes up heads or just P(X) to mean the probability table for the events (heads, tails)
- We can treat Random Variables like events
  - Flip coin A. Independently, flip coin B
  - P(A=heads | B=tails) = P(A=heads); A=heads and B=tails are independent events
  - for all x in {heads, tails}, for all y in {heads, tails}, P(A=x | B=y) = P(A=x); A and B are <u>independent random variables</u>
- We've been using Random Variables all along, actually, but have been a bit sloppy
- (I probably should have discussed these back in Lecture 4)

Probability of a word sequence

- $P(\mathbf{w}) = P(w_1, w_2, w_3, w_4, ..., w_n)$
- e.g. $P(\mathbf{w}$ = the cat slept quietly) = $P(w_1$=the, w_2=cat, w_3= slept, w_4=quietly)
- We'll often abuse notation when talking about specific events and context is clear, e.g. P(the cat slept quietly)

Maximum Likelihood Estimation?

- Recall Maximum Likelihood Estimations (MLE) for our HMM POS tagger
  - AKA "Count and divide"
- So get a corpus of N sentences
  - $P_{MLE}(\mathbf{w} = \text{the cat slept quietly}) = C(\text{the cat slept quietly})/N$
- But consider these sentences:
  - the long-winded peripatetic beast munched contentedly on mushrooms
  - parsimonius caught the of about for syntax
- Neither is in a corpus (I just made them up), so P_{MLE}=0 for both
  - But one is meaningful and grammatical and the other isn't!

Sparse Data and MLE

- If something doesn't occur, MLE thinks it can't occur
- No matter how much data you get, you won't have enough observations to model all events well with MLE
- We need to make some assumptions so that we can provide a reasonable probability for grammatical sentences, even if we haven't seen them

Independence (Markov) Assumption

- Recall, $P(w_1, w_2, \ldots, w_n) = P(w_n | w_1, w_2, \ldots, w_{n-1}) P(w_{n-1} | w_1, w_2, \ldots, w_{n-2}) \ldots$
  - $\prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$
- Still too sparse (nothing changed; same information)
  - if we want P(I spent three years before the mast)
  - we still need P(mast | I spent three years before the)
- Note: could use chain rule any number of ways
  - P(w_4= years | w_1= I, w_2= spent, w_3= three, w_5= before, w_6= the , w_7= mast)*...
- Remember definition of independence; A and B are independent if P(A) = P(A|B)

Independence (Markov) Assumption

- Make an n-gram independence assumption: probability of a word only depends on a fixed number of previous words (history)
  - **trigram model**: $P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i|\, w_{i-2}, w_{i-1})$
  - **bigram model:** $P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i|\, w_{i-1})$
  - **unigram model:** $P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i)$
- I.e. a trigram model says
  - P(mast | I spent three years before the) ≈ P(mast | before the)
- It also assumes all these are equal:
  - P(mast | I spent three years before the)
  - P(mast | I went home before the)
  - P(mast | I saw the sail before the)
    because all are estimated as P(mast | before the)
- Not always a good assumption! But it does reduce the sparse data problem

Estimating Trigram Conditional Probabilities

- P_{MLE}(mast | before the) = Count(before the mast)/Count(before the)
- In general, for any trigram, we have
  - $P_{MLE}(w_i|w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$
- To be clear, the MLE uses all data, not just data for the particular random variables we're estimating.
  - Count("the sequence *before the mast*"), not Count("the sequence where w_5=*before*, w_6=*the*, w_7=*mast*)

Example from *Moby Dick* corpus

- C(before, the) = 25; C(before, the, mast) = 4
- C(before, the, mast) / C(before, the) = 0.16
- mast is the most common word to come after "before the" (wind is second most common)
- P_{MLE}(mast) = 56/110927 = .0005 and P_{MLE}(mast|the) = .003
- Seeing "before the" vastly increases the probability of seeing "mast" next

Trigram model summary

- To estimate P(**w**), use chain rule and make an independence assumption
  - $\mathrm{P}(\mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n}) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$
  - $\approx P(w_1) P(w_2 | w_1) \prod_{i=3}^{n} P(w_i | w_{i-2}, w_{i-1})$
- Then estimate each trigram prob from data (here, using MLE)
  - $P_{MLE}(w_i | w_{i-2}, w_{i-1}) = \frac{\mathrm{Count}(w_{i-2}, w_{i-1}, w_i)}{\mathrm{Count}(w_{i-2}, w_{i-1})}$

Midterm Info

Some of you will
NOT BE IN SAL 101
YOU WILL BE
IN MHP (Mudd) 101

Who is Where?

- If your last* name is from Aggarwal to Patel, you are in SAL 101
- If it is from Pathapi to Zhu, you are in MHP 101
- * "last" is a very Eurocentric term. I have an alphabetization based on what USC has your surname listed as.
- To avoid confusion, please check the course website, which has a very prominent link to a roster with firstname, last name, partially masked email address, and assignment
- If you are still unsure of where to go, contact us on piazza ASAP

Midterm Logistics

- Length: 1 hour, 40 minutes (you will need time to set up and hand in exam)
- Date: Friday, October 6, 8:00 AM (Please arrive promptly!)
- Please Bring:
  - Pencils/pens/erasers as needed
  - one 8.5x11 inch (or A4) sheet of paper with notes on both sides (optional)
  - NO OTHER NOTES
  - NO ELECTRONIC RESOURCES
  - NO BOOKS
- We will provide extra paper for scratch work
- Sit only at seats with exams on them. Fill up all available space.

What's On The Exam?

- Fair game
  - Anything on the slides
  - Anything in the required reading
  - Anything in the homeworks
- But
  - We're not trying to trick you
  - We're not trying to make this impossible
  - If you understand the lectures well, you should be ok

What's On The Exam

- Major Topics
  - Levels of Linguistic Knowledge (L1)
  - Corpora, Regex, Basic text processing (L2)
  - Morphology, Finite State Automata and Transducers (L3)
  - Probability Theory (L4)
  - Naive Bayes, Features, Perceptron, Logistic Regression (L5)
  - POS Tagging and HMM tagger, Viterbi Decoding (L6)
  - Constituency Syntax Trees, Context-Free Grammars, CKY, CNF, Smoothing, Interpolating, Beam Decoding (L7-8)
  - Dependency Syntax Trees, Arc-Standard and Arc-Eager Dependency Parsing (L9-10)
  - Ngram Language Models, Smoothing, Backoff, Interpolation, alternate language models (L10-11) (Probably no neural; depends on how far we get)
- It won't all be on there because there isn't enough time
  - But there is plenty of room on the final

Practical details (I)

- Trigram model assumes two-word history
- But consider these sentences:

w_1	w_2	w_3	w_4
he	saw	the	yellow
feeds	the	cats	daily

- What's wrong?
  - a sentence shouldn't end with 'yellow'
  - a sentence shouldn't begin with 'feeds'
- Does the model capture these problems?

Beginning / end of sequence

- To capture behavior at beginning/end of sequences, we can augment the input:

w_{-1}	w_0	w_1	w_2	w_3	w_4	w_5
<s>	<s>	he	saw	the	yellow	</s>
<s>	<s>	feeds	the	cats	daily	</s>

- That is, assume $w_{-1}=w_0=$<s> and $w_{n+1}=$</s> so:
  - $\mathrm{P}(\boldsymbol{w}) = \prod_{i=1}^{n+1} P(w_i | w_{i-2}, w_{i-1})$
- Now P(</s>|the, yellow) is low, indicating this is not a good sentence
- P(feeds|<s>, <s>) should also be low

Beginning/end of sequence

- Alternatively, we could model all sentences as one (very long) sequence, including punctuation
  - two cats live in sam 's barn . sam feeds the cats daily . yesterday , he saw the yellow cat catch a mouse . [...]
- Now, trigram probabilities like P(. | cats daily) and P(, | . yesterday) tell us about behavior at sentence edges
- Here, all tokens are lowercased. What are the pros/cons of <u>not</u> doing that?

Practical details (II)

- Word probabilities are typically very small.
- Multiplying lots of small probabilities quickly gets so tiny we can't represent the numbers accurately, even with double precision floating point.
- So in practice, we typically use log probabilities (usually base-e)
  - Since probabilities range from 0 to 1, log probs range from -∞ to 0
  - Instead of multiplying probabilities, we add log probs
  - Often, negative log probs are used instead; these are often called "costs"; lower cost = higher prob
- Recall: we saw this with bigram HMM for POS tagging

Interim Summary: N-gram probabilities

- "Probability of a sentence": how likely is it to occur in natural language?
- We can never know the true probability, but we may be able to estimate it from corpus data.
- N-gram models are one way to do this:
  - To alleviate sparse data, assume word probs depend only on short history
  - Tradeoff: longer histories may capture more, but are also sparser
  - So far, we estimated N-gram probabilities using MLE

Interim Summary: Language Models

- Language Models tell us $P(\mathbf{w}) = P(w_1, ..., w_n)$: How likely is this sequence of words to occur?
  - Roughly: Is this sequence of words a "good" one in my language?
- LMs are used as a component in applications such as speech recognition, machine translation, and predictive text completion
- To reduce sparse data, N-gram LMs assume words depend only on a fixed-length history, even though we know this isn't true
- Next:
  - How to evaluate a language model
  - Weaknesses of MLE and how to address them (more sparsity)

Quiz 3

- Which of These Statements are True?
  - Naive Bayes is a probabilistic model
  - Perceptron is a generative model
  - Logistic regression has a closed-form solution
  - Logistic regression is a discriminative model

Two Types of Evaluation in NLP

- **Extrinsic**: measure performance on a downstream application
  - For LM, plug it into a machine translation/ASR/etc system
  - The most reliable and useful evaluation: We don't use LMs absent other technology
  - But can be time-consuming
  - And of course we still need an evaluation measure for the downstream system
- **Intrinsic**: design a measure that is inherent to the current task
  - much quicker/easier during development cycle
  - not always easy to figure out what the right measure is. Ideally, it's one that correlates with extrinsic measures
  - Extra-hard for LMs

Intrinsically Evaluating a Language Model

- For parsing, tagging, sentiment, etc. it was fairly clear how to evaluate: Hold a set of labeled data out and see how often your model gets it right
- For LM, it's not quite so clear
  - Given a corpus of sentences and non-sentences, see how often the LM thinks you have a sentence?
  - Not a very realistic evaluation of how an LM is used
  - Often we are deciding between not-that-grammatical outputs
- Ideally we want a regression evaluation
  - Given a sentence, how close is the model probability to the true probability
  - But we don't know the true probability of a sentence!

Idea: Model should give high probability to an unseen corpus

- Assume that you have a proper probability model, i.e. for all sentences S in the language L, $\sum_{S \in L} P(S) = 1$
- Then take a held-out test corpus T consisting of sentences in the language you care about
- $\prod_{t \in T} P(t)$ should be as high as possible; model should think each sentence is a good one
- Let's be explicit about evaluating each word in each sentence
  - $\prod_{t \in T} \prod_{w \in t} P(w)$
- Collapse all these words into one big 'sentence' N:
  - $\prod_{w \in N} P(w)$

Resolving Some Problems

- $\prod_{w \in N} P(w)$ is going to result in underflow. Ok, let's use logs again!
- Also we tend to like positive sums.
  - $-\sum_{w \in N} \log_2 (P(w))$
- This can be tough to compare against corpora of different length (or sentences of different length), so normalize by the number of <u>words</u>:
  - $\frac{-\sum_{w \in N} \log(P(w))}{|N|}$ is called the <u>cross-entropy</u> of the data according to the model
- When comparing models, differences between these numbers tend to be pretty small, so we exponentiate
  - $2^{\frac{-\sum_{w \in N} \log(P(w))}{|N|}}$ is called the <u>perplexity</u> of the data
- Think of this as "how surprised is the model?"

Example

- Three word sentence with probabilities ¼ , ½, ¼
  - ¼ * ½ * ¼ = .03125
  - cross-entropy: -(log(1/4) + log(1/2) + log(1/4))/3 = 5/3; $2^{5/3}$ ≈3.17
- Six word sentence with probabilities ¼, ½, ¼, ¼, ½, ¼
  - ¼ * ½ * ¼ * ¼ * ½ * ¼ = .00097
  - cross-entropy: -(log(1/4) + log(1/2) + log(1/4) + log(1/4) + log(1/2) + log(1/4))/6 = 10/6; $2^{10/6}$ ≈3.17
- If you overfit your training corpus so that P(train) = 1, then Perplexity on train is 0
- But Perplexity on test (which doesn't overlap with train) will be infinite

Intrinsic Evaluation Big Picture

- Lower Perplexity is better
- Roughly = number of bits needed to communicate information about a word
  - The terms 'cross-entropy' and 'perplexity' come out of information theory; it's in the reading if you're interested but we won't dwell on it
- In principle you could compare on different test sets
- In practice, domains shift. To know which of two LMs is better, train on common training sets, test on common test sets

Sparse data, again

- Suppose we build a trigram model from Moby Dick and evaluate the sentence "I spent three years before the mast"
- "I spent three" never occurs in training, so $P_{MLE}(\text{three}|\text{I spent}) = 0$
- so cross-entropy is infinite
- This is basically right; our model says "I spent three" should never occur so when it does our model is infinitely surprised!
- Even with a unigram model we run into words we never saw, so we need better ways to estimate probabilities from sparse data

Add-1 (Laplace) and Add-α (Lidstone) Smoothing Again

- Pretend we saw everything 1 (α) more times than we did before
- $P_{+1}(w_i|w_{i-2}, w_{i-1}) = (C(w_{i-2}, w_{i-1}, w_i)+1)/(C(w_{i-2}, w_{i-1})+|V|)$ where $|V|$ is the size of the vocabulary
- $P_{+\alpha}(w_i|w_{i-2}, w_{i-1}) = (C(w_{i-2}, w_{i-1}, w_i)+\alpha)/(C(w_{i-2}, w_{i-1})+\alpha|V|)$

Dealing with unknown vocabulary

- Can we also add a new 'OOV' token as was done in HMM emission table?
- It gets kind of complicated...
- $P_{+\alpha}(w_i|w_{i-2}, w_{i-1}) = (C(w_{i-2}, w_{i-1}, w_i)+\alpha)/(C(w_{i-2}, w_{i-1})+\alpha|V|+1)$
- $P_{+\alpha}(w_i = OOV|w_{i-2}, w_{i-1}) = \alpha/(\alpha|V|+1)$
- But then we also have to deal with, e.g., $P_{+\alpha}(w_i|w_{i-2}, w_{i-1}=OOV)$
- Better solution: replace low-count words in corpus with "OOV"
- Intuition: 1-count is basically the same as 0-count

Remaining Problem

- In a training corpus, suppose we see *Scottish beer* but neither of
  - *Scottish beer drinkers*
  - *Scottish beer eaters*
- If we build a smoothed trigram model (with any kind of smoothing), which example has higher probability?
  - Both the same! Unknown events are treated equally by smoothing!

Remaining Problem

- Previous smoothing methods assign equal probability to unseen events
- Better: use information from lower-order N-grams (shorter histories)
  - beer drinkers
  - beer eaters
- Two ways: <u>backoff</u> and <u>interpolation</u>

Backoff

- Idea: Trust the highest order language model that contains your N-gram
- P_{BO} $(z|x\ y)$ =
  $(1- \alpha_{xy})P(z|x\ y)$ if count(x y) > 0
  $\alpha_{xy}\ P_{BO}(z \mid y)$ else
- where α_{xy} is an interpolation parameter

Simple Interpolation

- Idea: Trust different amounts of context differently
- $P_{SI}(z|x\ y) =$

  $\lambda_3\ P(z|x\ y) +$

  $\lambda_2\ P(z|y) +$

  $\lambda_1\ P(z) +$

  λ_0
- where $\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 = 0$, all >=0
- We did something similar in lexicalized constituency parsing

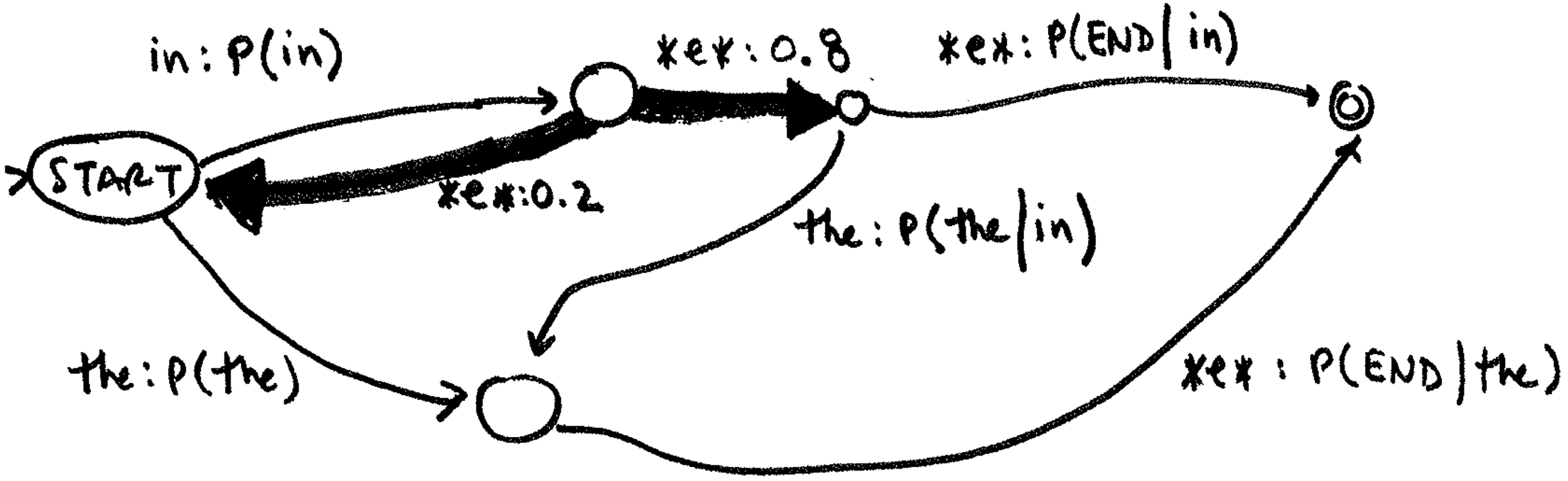
Better Interpolation

- Idea: As in backoff, particular contexts matter
- P_{SI} (z|x y) =
  λ_{xy} P(z|x y) +
  $\lambda_{y/xy}$ P(z|y) +
  $\lambda_{1/xy}$ P(z) +
  $\lambda_{0/xy}$
- where, for each xy, $\lambda_{0/xy} + \lambda_{1/xy} + \lambda_{y/xy} + \lambda_{xy} = 0$, all >=0
- Best not to actually have a different set for each unique context; can group by context <u>count</u>

State-of-the-art Smoothing

- There is lots and lots of work done on smoothing and lots of variants
- See Chen and Goodman (optional reading); it's actually quite comprehensive, though mathy
- Best today is Modified Kneser-Ney
  - replace MLE with estimates based on count of unique histories
  - 4 interpolation lambdas based on ngram counts
- For very large data, Google's Stupid Backoff
  - Really fast to calculate; good for very large data
  - Doesn't give proper perplexities!
  - Works well in practice
- These are available in SRILM (K-N) and KenLM (both)

Ngram LM as FSA



from kevin knight

Quiz 4

stack	symbols
[root] a b c e f g	j k l o p

- The above table represents a potential configuration during dependency parsing. If we are doing **arc-eager** parsing, which symbols may be combined in a Left-Arc from this configuration?
  - f and g
  - e and f
  - a and o
  - g and j

Quiz 5

stack	symbols
[root] a b c e f g	j k l o p

- The above table represents a potential configuration during dependency parsing. If we are doing **arc-standard** parsing, which symbols may be combined in a Left-Arc from this configuration?
  - f and g
  - e and f
  - a and o
  - g and j

Other Approaches To Language Modeling

- Maybe we don't always care about the most immediate words
  - "Show Sally a good ____"
  - "Show Dave a good ____"
- P(time | a good) isn't so great
- P(time | Sally a good) isn't really either
- P(time | Show Sally a good) isn't helpful for Dave (and backoff/smoothing doesn't really help)
  - But how much to skip?
  - We could interpolate different skip models; doesn't help that much though

Other Approaches To Language Modeling

- Class-based smoothing
  - Train = ...party on Tuesday ...
  - Test = ...party on Monday ...
    ....celebration on Tuesday...
- Maybe we could predict classes first, and then words...
  - Model $P(w_i\text{=Monday} \mid w_{i-2}\text{=party},\ w_{i-1}\text{=on})$
    - $P(c_i\text{=DAY} \mid w_{i-2}\text{=party},\ w_{i-1}\text{=on}) * P(w_i\text{=Monday} \mid w_{i-2}\text{=party},\ w_{i-1}\text{=on}, c_i\text{=DAY})$
  - Or even drop the words themselves from the conditional
    - $P(c_i\text{=DAY} \mid w_{i-2}\text{=EVENT},\ w_{i-1}\text{=PREP}) * P(w_i\text{=Monday} \mid c_i\text{=DAY})$
- These generally perform worse than trigram on their own but can help when interpolated

Other Approaches To Language Modeling

- Language should be syntactically well formed!
  - Can use law of total probability in reverse:
  - P(s) = $\sum_{t \in T} P(s, t)$ where t is a syntax tree for s
  - Or, instead of going in n-gram order, why not follow dependency links
  - P(I like your shoes) = P(like|root)P(I|like)P(shoes|like)P(your|shoes)
- Difficulties: Requires syntactic analysis which means less data available
- In practice, these methods, when interpolated with 3grams, helped a bit
- once we got beyond 1b words of data, not helpful

Adding Hidden Information

- Maybe there are different types of sentences!
  - Introductory
  - Asides
  - Technical
- And each one behaves differently!
- If you knew the type of the sentences you could evaluate it with a separate LM estimated off of just that type of sentence

Feature-Based LM

- Let's return to perceptron/maxent
- Previously we predicted sentiment, word sense, author given an input text
- Up to now we've been talking about predicting x_i given x_{i-1}, x_{i-2}
- Can we model this as a discriminative feature-based model?

	$\phi(x)$	$w(x_i\text{=bit})$	$w(x_i\text{=bought})$
bias	1	.95	-3.2
x_{i-2}=apple	0	.436	33.6
x_{i-2}=dog	0	-34	…
x_{i-2}=cat	0	…	…
x_{i-2}=the	1	3.4	3.6
…	…	…	…
x_{i-1}=apple	0	…	…
x_{i-1}=dog	1	14.4	-5.6
x_{i-1}=cat	0	6.3	-7.8
x_{i-1}=the	0	-5	-17

P(bit | the cat) vs P(bought | the cat)

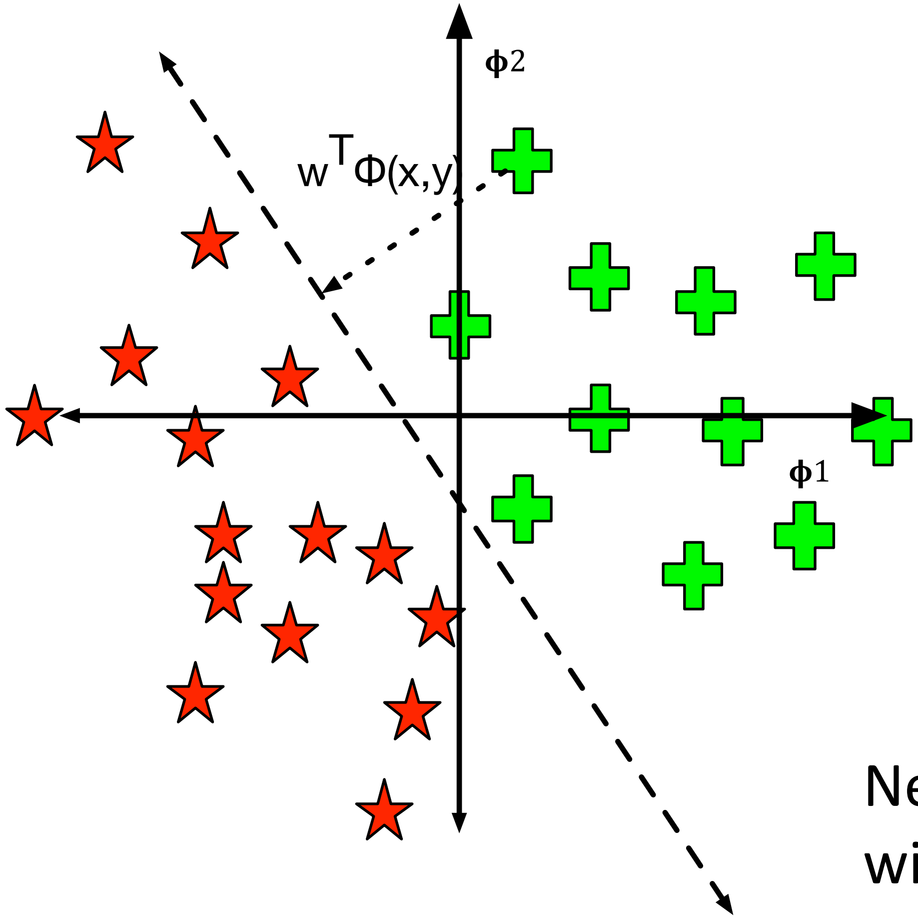
- Kind of like a trigram model!
- Note, separate weight for each output type again (too many to evaluate each)
- Bias term: a priori preference for that word (unigram prob)

	$\phi(x)$	$w(x_i\text{=bit})$	$w(x_i\text{=bought})$
bias	1	.95	-3.2
x_{i-2}=apple	0	.436	33.6
x_{i-2}=dog	0	-34	…
x_{i-2}=cat	0	…	…
x_{i-2}=the	1	3.4	3.6
…	…	…	…
x_{i-1}=apple	0	…	…
x_{i-1}=dog	1	14.4	-5.6
x_{i-1}=cat	0	6.3	-7.8
x_{i-1}=the	0	-5	-17
…		…	…
x_{i-2}=NOUN	0	…	…
x_{i-2}=DET	1	…	…
x_{i-1}=NOUN	1	…	…
x_{i-1}=DET	0	…	…
x_{i-2}=apple ^ X_{i-1}=the	1	…	…
		…	…

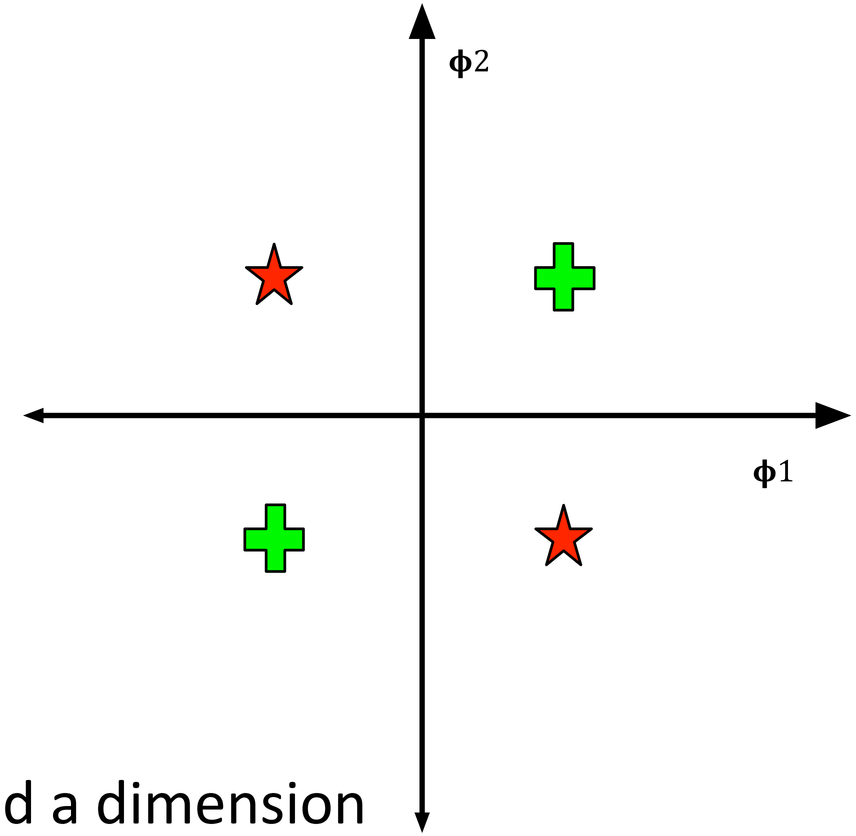
P(bit | the cat) vs P(bought | the cat)

- As before, we can add arbitrary features
  - POS tags
  - word length
  - initial letter
- What about conjuncts of features (true trigram)?
  - Yes, but this will get very lengthy
  - Shouldn't the individual units be enough?

Linear models can only separate linearly



Good Case



Need to add a dimension
with the conjunction of these features!

Bad Case

This May Seem Like A Weird Aside!

Don't worry, here's the road map:

N-gram language models ->

Considering other properties ->

Arbitrary feature language models ->

Handling conjuncts of features ->

Stacks of nonlinear perceptron = neural network language models

Solving the XOR problem

- Let our features be two dimensional input and the class label one-dimensional output
- Can we find W (2x1) and b (2x1) such that the function y = Wx + b solves the problem for this data?
- No
- But we can use a <u>nonlinear</u> function and map this data into a new, separable feature space!

x1	x2	y
1	1	1
-1	1	-1
-1	-1	1
1	-1	-1

Mapping into a new space

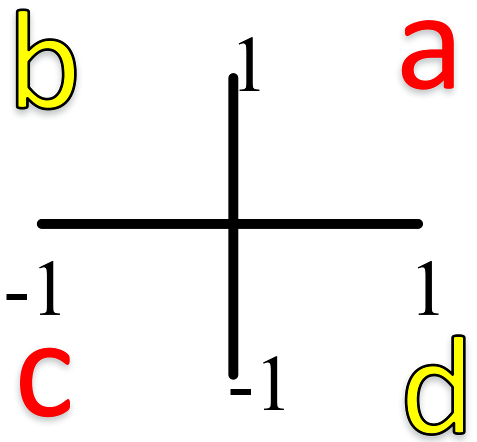
$step(x1*h_{x1}+x2*h_{x2}+b)$

h1 mapper

x1	1
x2	1
b	-1

a: step(1+1-1) = 1
b: step(-1+1-1) = -1
c: step(-1-1-1) = -1
d: step(1-1-1) = -1

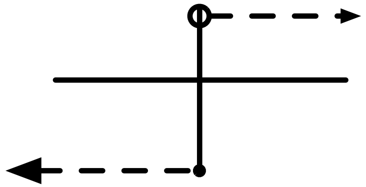
x space



x-space data

	x1	x2	y
a	1	1	1
b	-1	1	-1
c	-1	-1	1
d	1	-1	-1

step function:1 if > 0, else -1

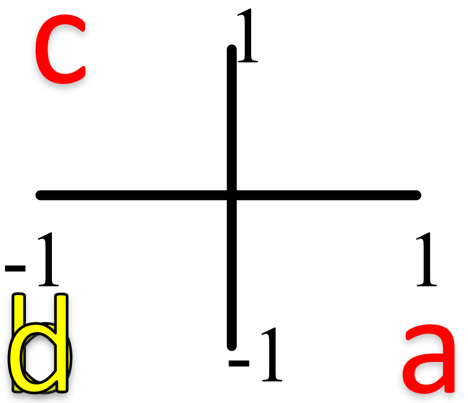


h2 mapper

x1	-1
x2	-1
b	-1

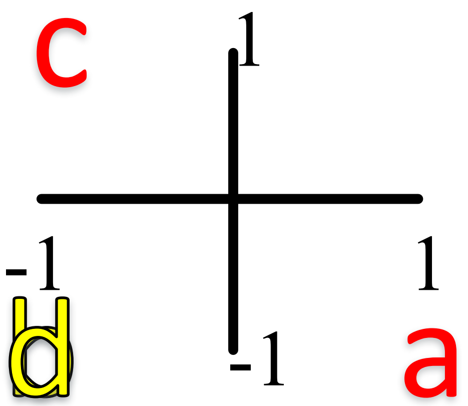
a: step(-1-1-1) = -1
b: step(1-1-1) = -1
c: step(1+1-1) = 1
d: step(-1+1-1) = -1

h space



New ('hidden') space to output space

h space



h-space data

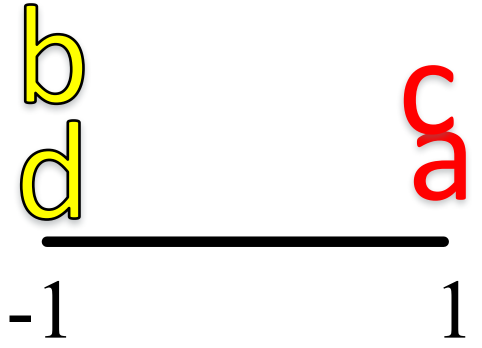
	h1	h2	y
a	1	-1	1
b	-1	-1	-1
c	-1	1	1
d	-1	-1	-1

o mapper

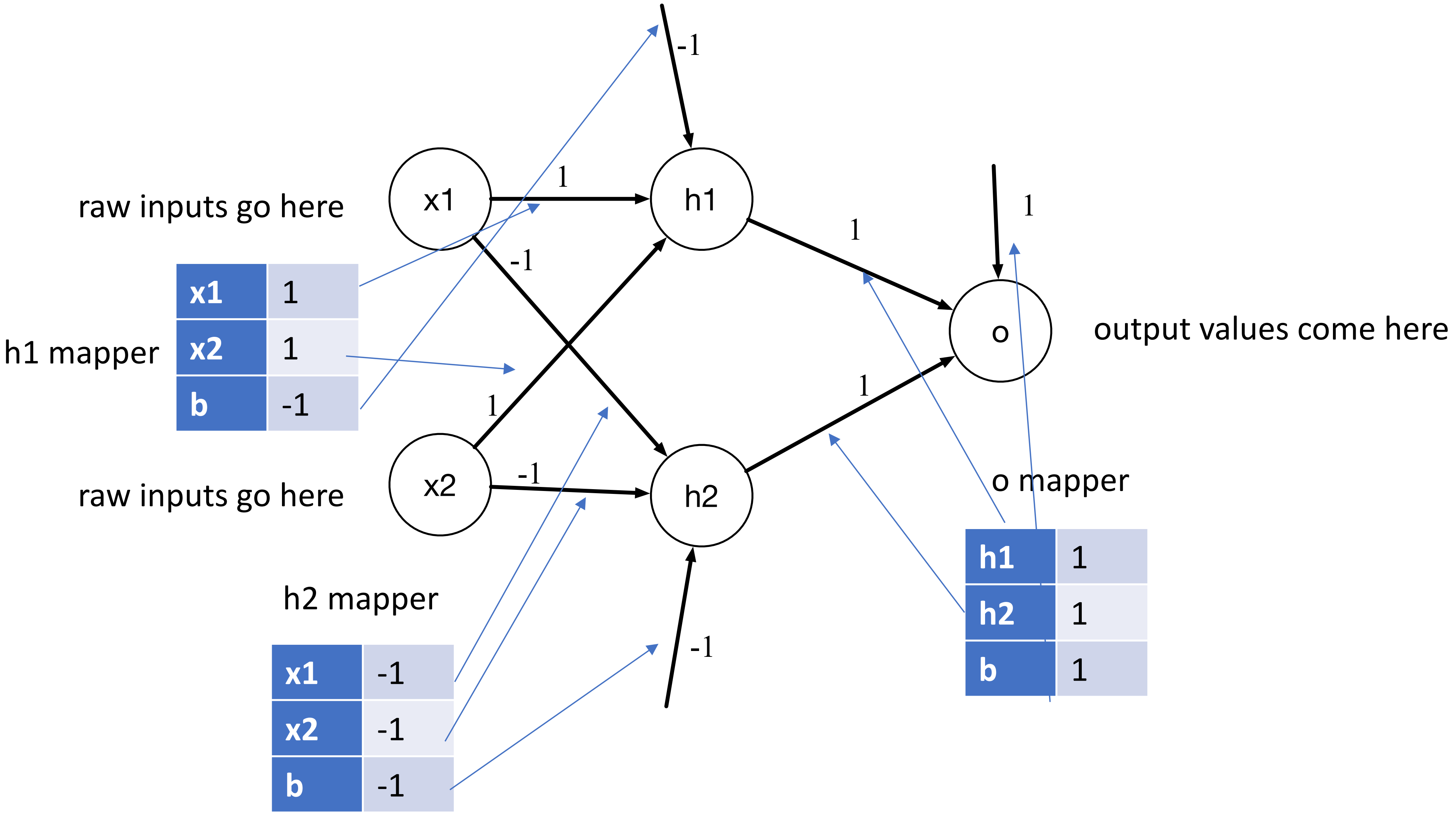
h1	1
h2	1
b	1

step($h1*o_{h1}+h2*o_{h2}+b$)

a: step(1-1+1) = 1
b: step(-1-1+1) = -1
c: step(-1+1+1) = 1
d: step(-1-1+1) = -1



Another way of looking at these matrices



Quiz 6

stack	symbols
[root] a b c e f g	j k l o p

- What is the resulting configuration after an arc-standard Left-Arc?

stack	symbols
[root] a b c e f	g j k l o p

stack	symbols
[root] a b c e f	j k l o p

stack	symbols
[root] a b c e f g j	k l o p

stack	symbols
[root] a b c e g	j k l o p

stack	symbols
[root] a b c f g	j l o p

Quiz 7

stack	symbols
[root] a b c e f g	j k l o p

- What is the resulting configuration after an arc-eager Right-Arc?

stack	symbols
[root] a b c e f	g j k l o p

stack	symbols
[root] a b c e f	j k l o p

stack	symbols
[root] a b c e f g j	k l o p

stack	symbols
[root] a b c e g	j k l o p

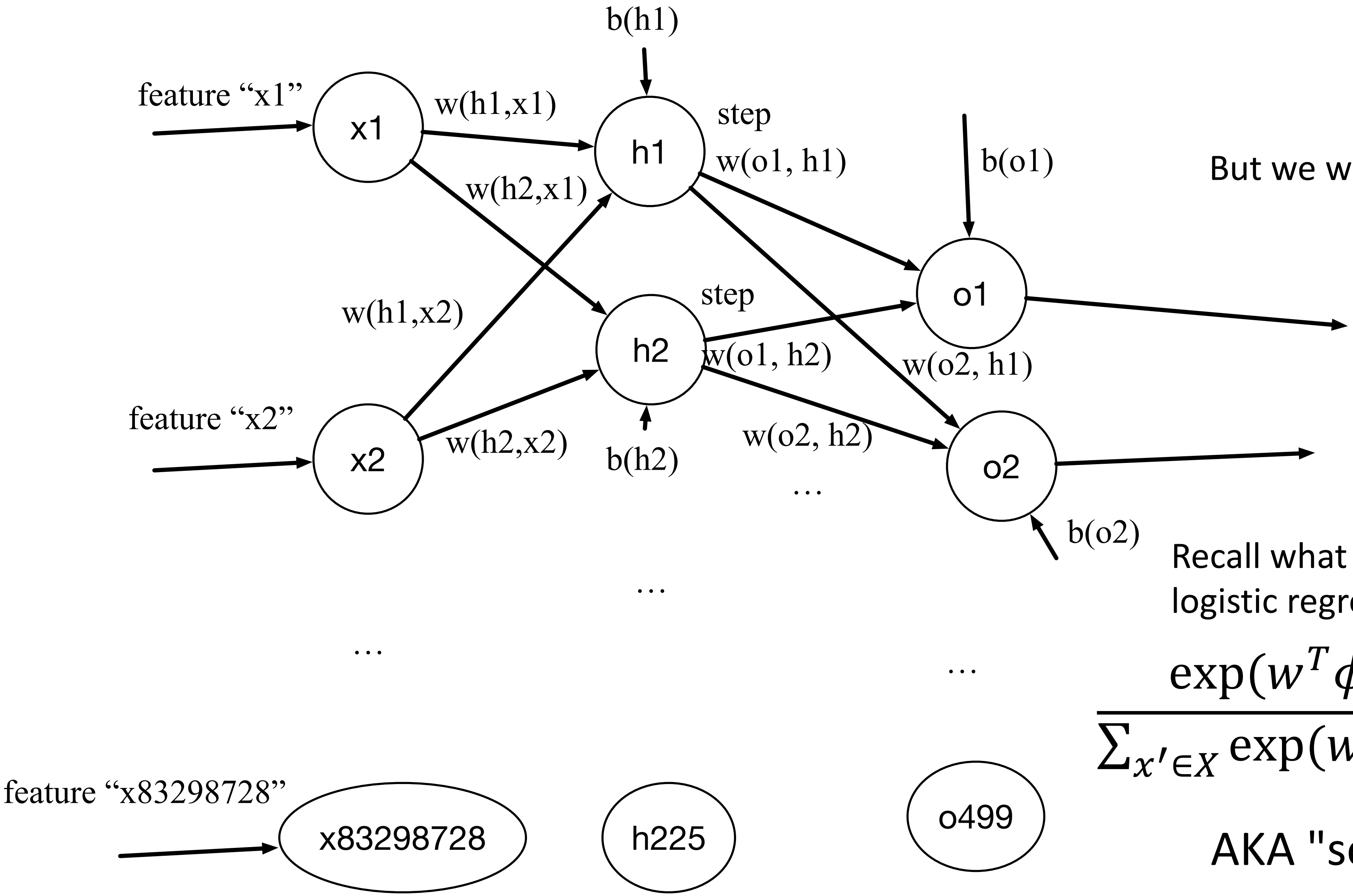
stack	symbols
[root] a b c f g	j l o p

What Has This To Do With Language Models?

- Toy example:
  - x1 (think "feature 1") can have value 1 or -1
  - x2 can have value 1 or -1
  - outcome can be 1 or -1
  - given training data (a, b, c, d), can we use perceptron to learn o from features?
  - no. If we added x3 ("x1 xor x2") we could. But we saw we could add <u>layers</u> and a nonlinear function and now use conjuncts of these features

What Has This To Do With Language Models?

- LM example:
  - x1 can be 'wn-1=dog' (1 or 0)
  - x2 can be 'wn-1 = the' (1 or 0)
  - ...
  - x24657 can be 'wn-4 is adjective'
  - we can have x2947502393 be "wn-1=dog ^ wn-2=the" but we can also use multi-layer structure to get that for 'free'
  - for o, we'd like to know what we think about each word. So have an o for each word.

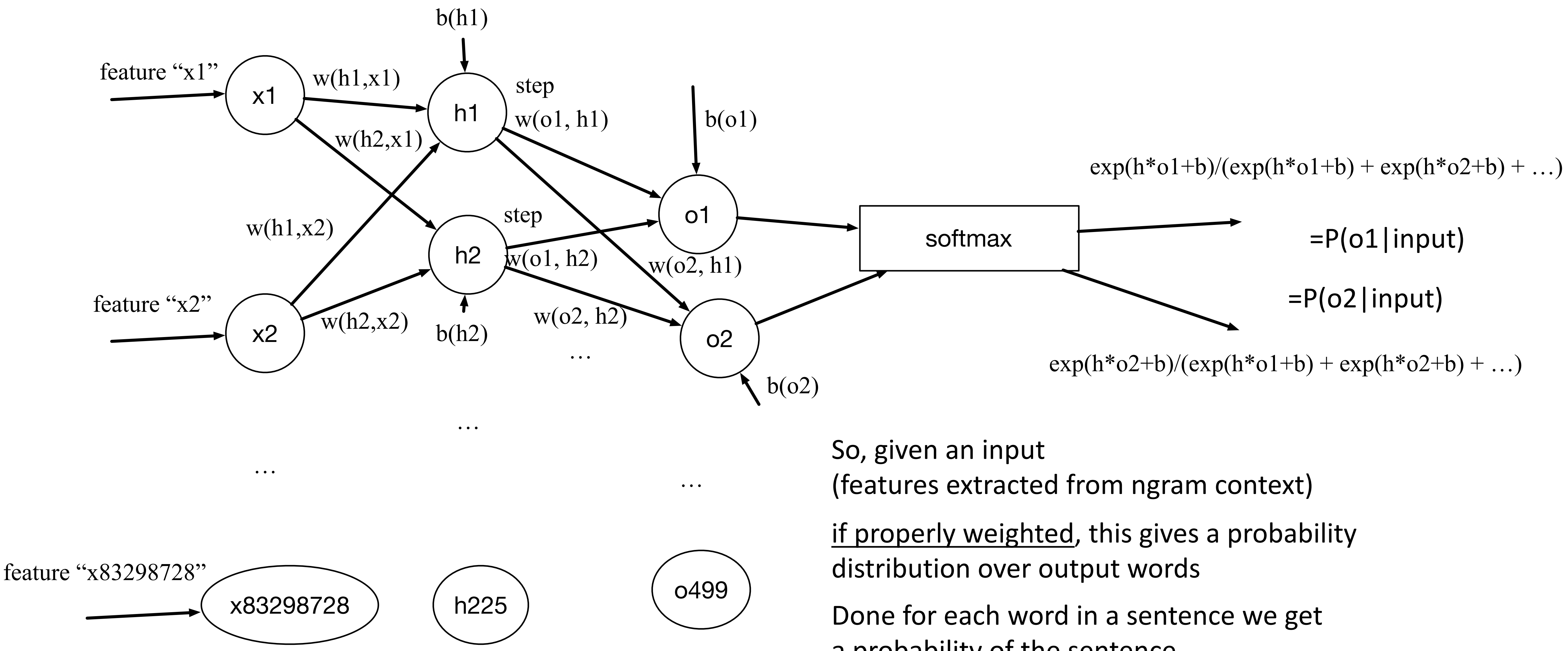


But we want probabilities here

Recall what we did for logistic regression

$$\frac{\exp(w^T\phi(x))}{\sum_{x'\in X}\exp(w^T\phi(x'))}$$

AKA "softmax"



So, given an input
(features extracted from ngram context)

<u>if properly weighted</u>, this gives a probability distribution over output words

Done for each word in a sentence we get a probability of the sentence

This is a language model!

Aside: Softmax

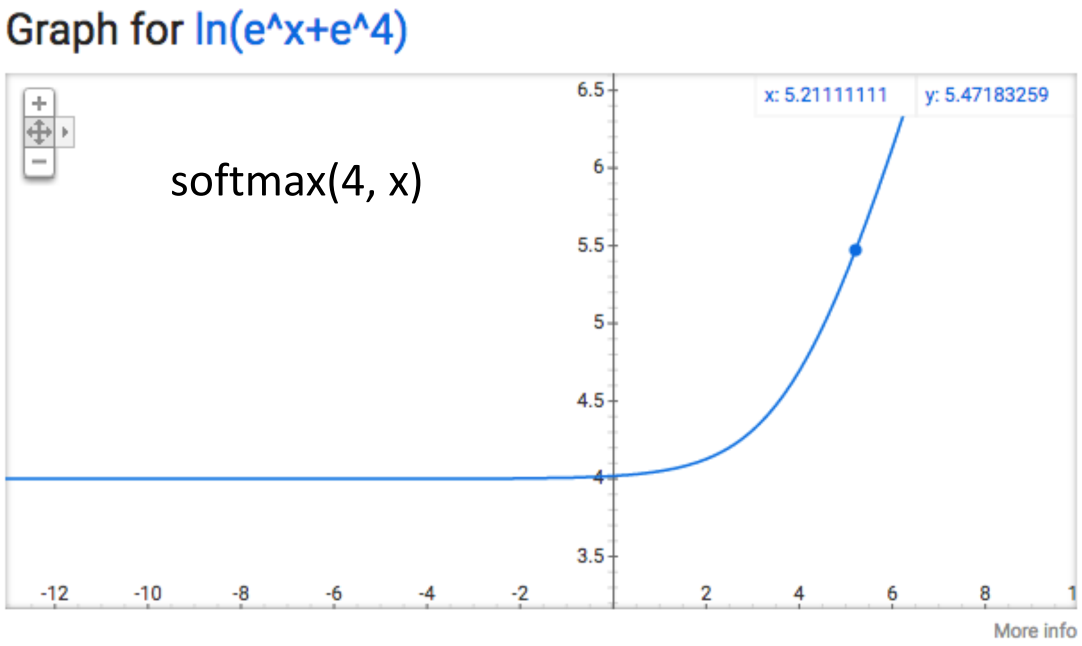
- If you've been exposed to logistic regression and/or neural networks before you've probably heard of softmax

$$\frac{\exp(w^T \phi(x))}{\sum_{x' \in X} \exp(w^T \phi(x'))}$$

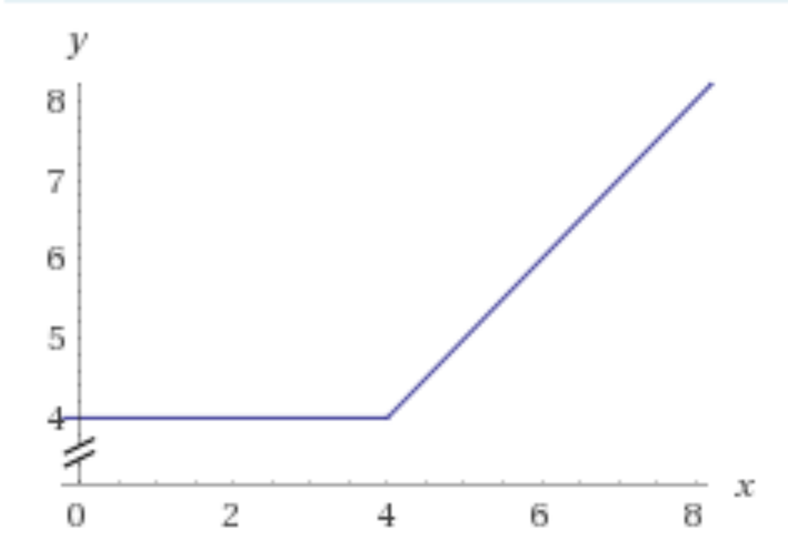
- I always wondered where the name comes from
- Nobody ever told me!
- Do they tell you?

Max and softmax functions

- the softmax *function* is defined as is a "soft" approximation of max
- softmax(x, y, z) = $\ln(e^x+e^y+e^z)$



max(4, x)



from Abishek Patnia, via Quora

Softmax vs softmax activation

- So in general, softmax(X) = $\ln \sum_{x \in X} e^x$
- But that's not what we're usually talking about in logreg/neural network land
- We talk about $\frac{e^{x_i}}{\sum_{x \in X} e^x}$ (I simplified from the dot-product-of-features notation)
- This is useful because it squashes a collection of numbers into a probability distribution, yet preserves order
  - Remember, e^x to make everything positive, then normalize

Softmax vs softmax activation

- Consider how you might really use this, though:
  - $\frac{e^{x_i}}{\sum_{x \in X} e^x}$ is going to run into underflow issues. Best to take log
  - $\ln(\frac{e^{x_i}}{\sum_{x \in X} e^x})$ = $\ln e^{x_i} - \ln \sum_{x \in X} e^x$
  - $x_i - \text{ softmax}(X)$
- So when we say "apply the softmax activation function" we really mean "subtract softmax from each element"

Quiz 8

stack	symbols
[root] a b c e f g	j k l o p

- What is the resulting configuration after an arc-standard Right-Arc?

stack	symbols
[root] a b c e f	g j k l o p

stack	symbols
[root] a b c e f	j k l o p

stack	symbols
[root] a b c e f g j	k l o p

stack	symbols
[root] a b c e g	j k l o p

stack	symbols
[root] a b c f g	j l o p

Quiz 9

stack	symbols
[root] a b c e f g	j k l o p

- What is the resulting configuration after an arc-eager Left-Arc?

stack	symbols
[root] a b c e f	g j k l o p

stack	symbols
[root] a b c e f	j k l o p

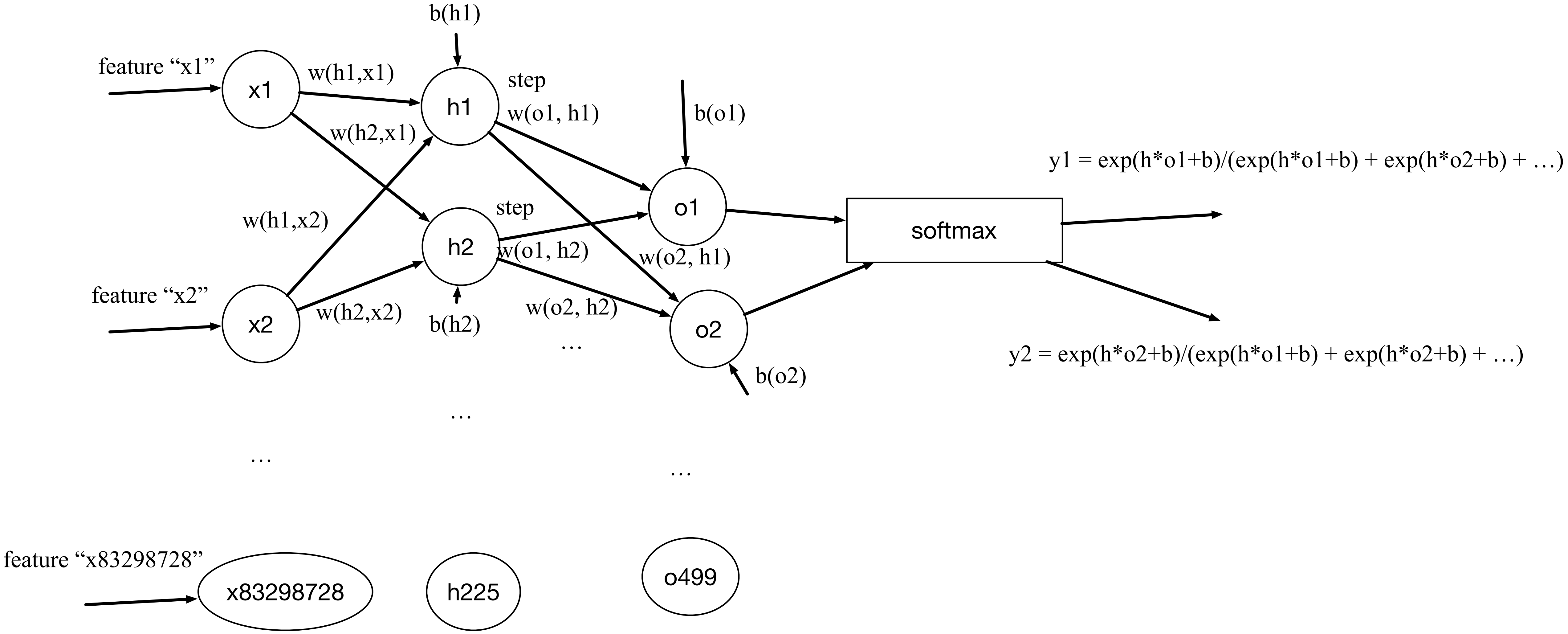
stack	symbols
[root] a b c e f g j	k l o p

stack	symbols
[root] a b c e g	j k l o p

stack	symbols
[root] a b c f g	j l o p

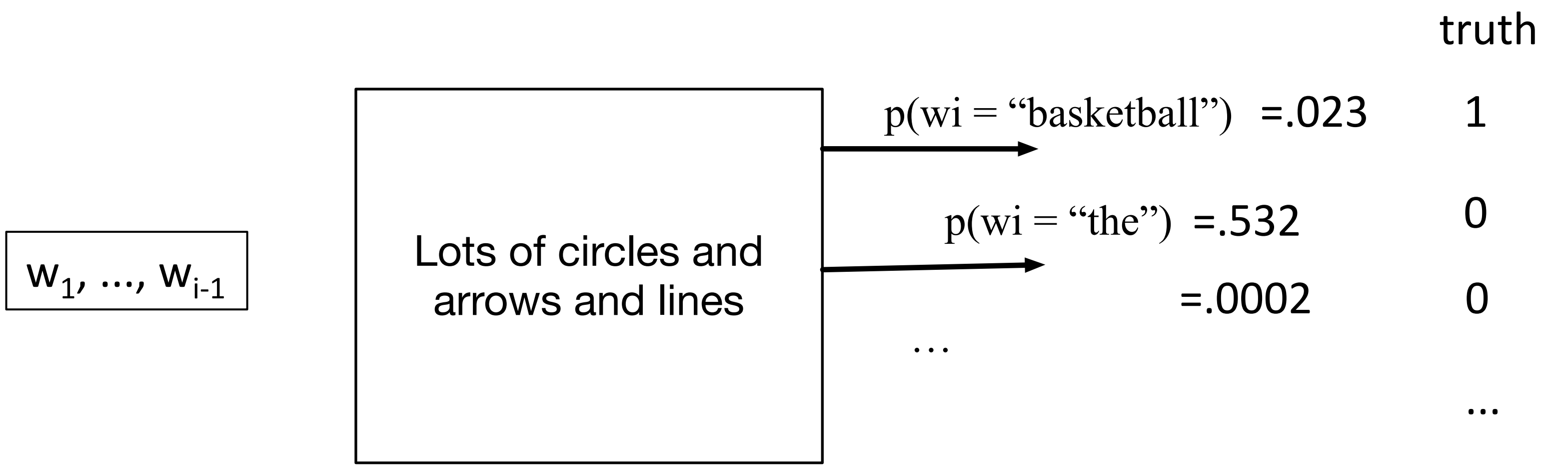
How Do We Set The Weights?

- For the very simple xor case we set the 9 weights by hand
- But the general problem has lots of parameters!
- Fear not, we can use the same approach we used before:
  - Define a <u>loss</u> (how bad was our decision vs reality?)
  - Calculate the <u>gradient</u> (derivative w/r/t our parameters)
  - Adjust parameters, to move away from the gradient
  - Try again with more data, until we find something good



b(h1)
feature "x1"
x1
w(h1,x1)
h1
step
w(o1, h1)
b(o1)
w(h2,x1)
y1 = exp(h*o1+b)/(exp(h*o1+b) + exp(h*o2+b) + …)
step
o1
softmax
w(h1,x2)
h2
w(o1, h2)
w(o2, h1)
feature "x2"
x2
w(h2,x2)
b(h2)
w(o2, h2)
o2
y2 = exp(h*o2+b)/(exp(h*o1+b) + exp(h*o2+b) + …)
b(o2)
…
…
…
…
feature "x83298728"
x83298728
h225
o499

Training Setup

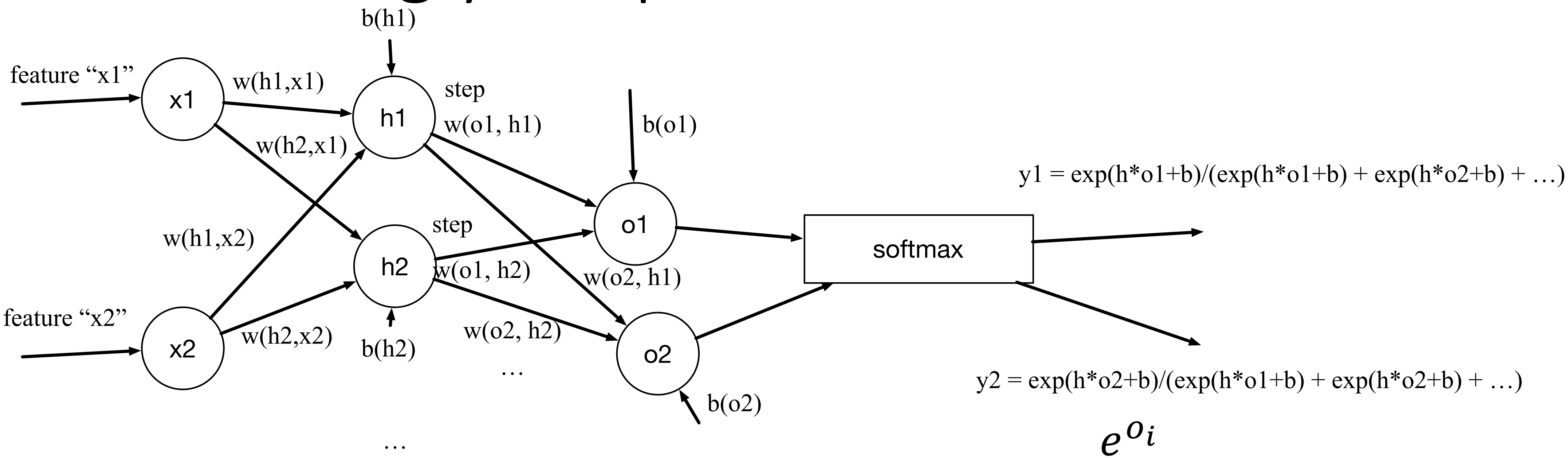


Loss

- A common one is <u>squared error</u>: $(y_{truth}-y_{hyp})^2$
- gradient = $2(y_{truth}-y_{hyp})\ y_{hyp}'$
- y is a vector (one entry per output vocab member)
- Note: $2(y_{truth}-y_{hyp}) > 0$ for truth, <0 else
  - = move toward the good thing, away from the bad
- Ok, so what's y_{hyp}?

hyp	truth	
.023	1	
.532	0	
.0002	0	(scalar, given)
(result of equation; function of parameters)		

Back to the Ugly Graph



$$y_i = \frac{e^{o_i}}{e^{o_1} + e^{o_2}}$$

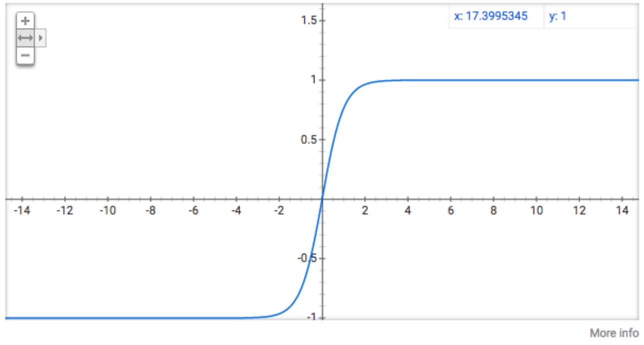
$$h_i = \text{step}(w_{h_i x_1} x_1 + w_{h_i x_2} x_2 + b_{h_i})$$

$$\text{o}_i = \text{step}(w_{o_i h_1} h_1 + w_{o_i h_2} h_2 + b_{o_i})$$

- y is pretty complicated! Need to differentiate w/r/t each parameter
- y contains step function: not differentiable at 0 and =0 elsewhere!
- tanh is a nicer approximation

Making differentiable

tanh

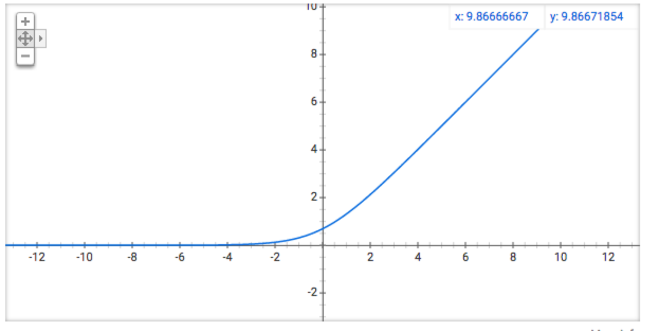


$$h_i = \tanh(w_{h_i x_1} x_1 + w_{h_i x_2} x_2 + b_{h_i})$$

$$y_i = \frac{e^{o_i}}{e^{o_1} + e^{o_2}} \qquad o_i = \tanh(w_{o_i h_1} h_1 + w_{o_i h_2} h_2 + b_{o_i})$$

- y is pretty complicated! Need to differentiate w/r/t each parameter
- y contains step function: not differentiable at 0 and =0 elsewhere!
- tanh is a nicer approximation
- Can also use ReLU (or softplus = softmax(x, 0))

softplus



Differentiating

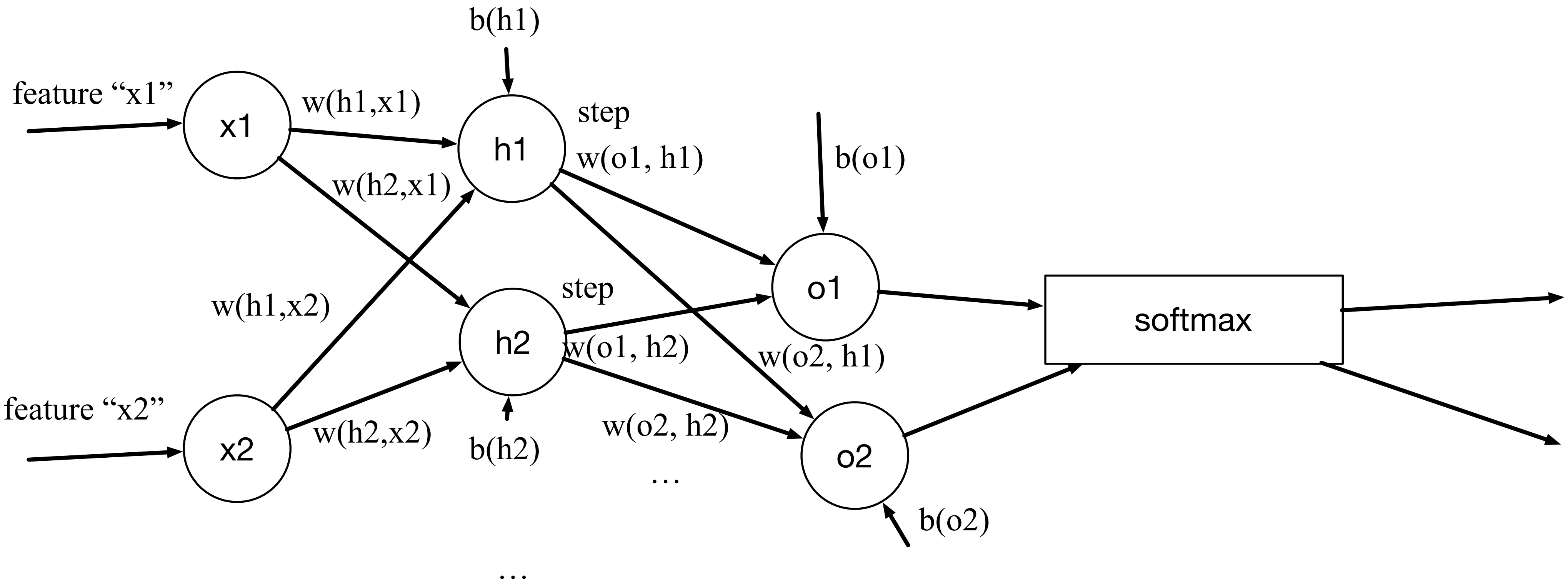
- gradient = $2(y_{truth}-y_{hyp})\ y_{hyp}'$
- y_{truth} given; y_{hyp} found by propagating data through the messy function
- y_{hyp}'? lots of partials
- $dy/dw_{oh} = dy/do\ do/dw_{oh}$
- $dy/db_{o} = dy/do\ do/db_{o}$
- $dy/d_{hx} = dy/do\ do/dh\ dh/dw_{hx}$
- $dy/db_{h} = dy/do\ do/dh\ dh/db_{h}$

$$h_i = \mathrm{step}(w_{h_i x_1} x_1 + w_{h_i x_2} x_2 + b_{h_i})$$

$$y_i = \frac{e^{o_i}}{e^{o_1} + e^{o_2}}$$

$$\mathrm{o}_i = \mathrm{step}(w_{o_i h_1} h_1 + w_{o_i h_2} h_2 + b_{o_i})$$

Good News: You don't really have to worry about it!



Auto-differentiation: topologically calculate values forward, derivatives backward

Partial values stored at each cell; dynamic programming makes it all efficient

Implemented in e.g. tensorflow, theano, Dynet

What Should We Connect? What Features Should We Use?

- Motivation was bigram features via structured perceptron
- So just connect the unigrams for adjacent words together?
  - i.e. all w_1=... to all w_2=...
  - seems like a lot of careful planning
- What about similar word-class behavior?
  - Maybe all days should function similarly
  - Or all animals
- Maybe we can characterize a <u>single word</u> by a set of features
  - But which features?
  - Letter it starts with?
  - Part of speech?
  - Class?
- Idea: let the learning figure out how to assign features; we just choose the number of features

Fully connected

"Embeddings" shared

Embedding cell 14:
animate noun(?)

hidden cell 44:
topic is business (?)

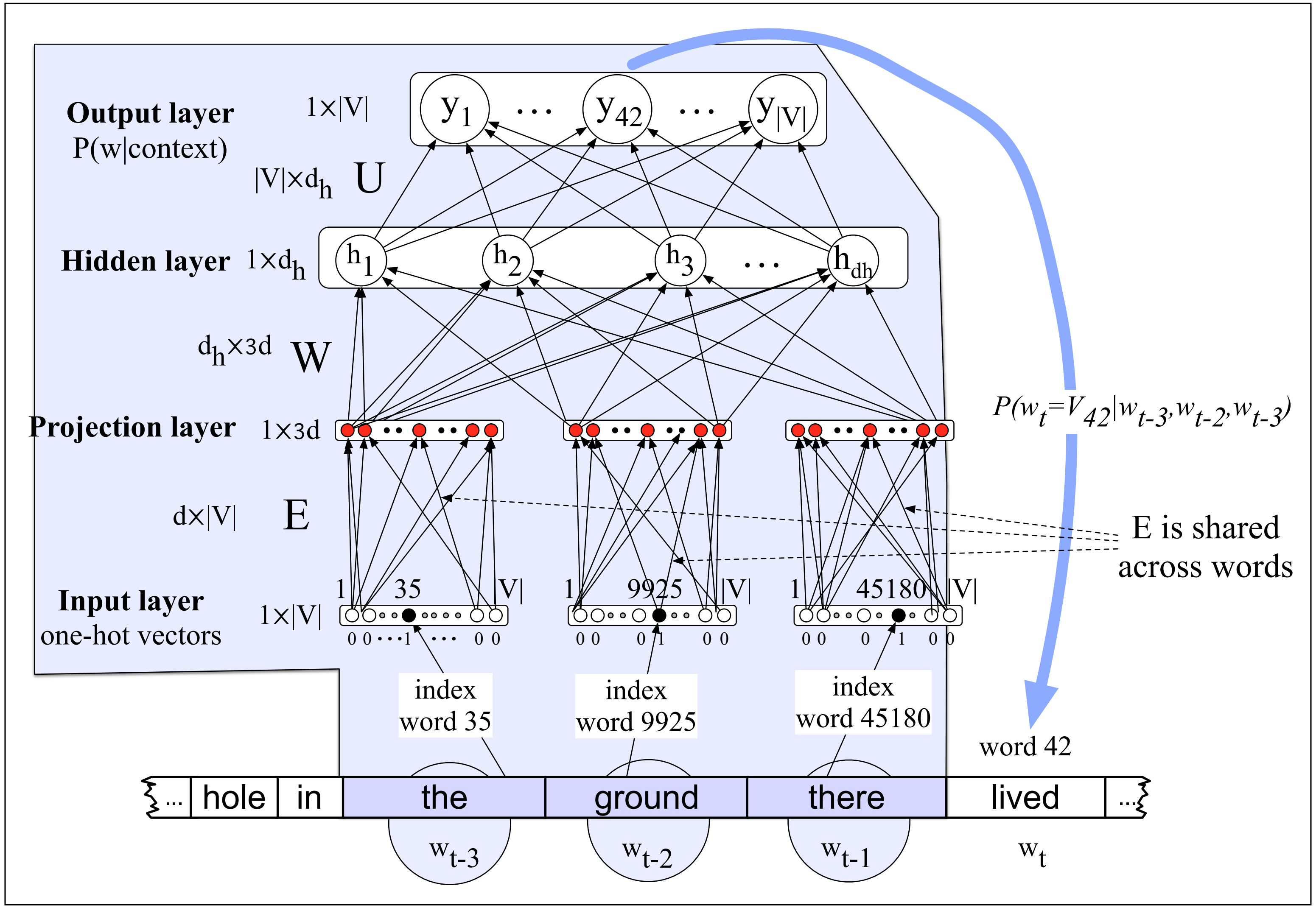


Figure 8.13 learning all the way back to embeddings. notice that the embedding matrix E is shared among the 3 context words.

from J&M v3