Introduction to Natural Language Processing

Language Modelling

- Jurafsky, D. and Martin, J. H. (2019): Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Third Edition. Chapters 8, 9, 11
- Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C. (2013): A Neural Probabilistic Language Model. Journal of Machine Learning Research 3 (2003):1137–1155
- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S. (2010): Recurrent neural network based language model. Proceedings of Interspeech 2010, Makuhari, Chiba, Japan, pp. 1045-1048

PLAN OF THE LECTURE

- Language Modelling (LM) Task
- Statistical Language Models
- Neural Language Models: Feedforward, Recurrent, Transformer

Language Model (LM): Guessing the next word

Given a partial sentence, how hard is it to guess the next word?

```
She said ____
She said that ____
I go every week to a local swimming ____
Vacation on Sri ____
```

Language Model (LM): Guessing the next word

Given a partial sentence, how hard is it to guess the next word?

```
She said ____
She said that ____
I go every week to a local swimming ____
Vacation on Sri ____
```

Tasks for a LM:

- Modeling the probability of a next word, given its context (usually: next word based on predecessors).
- Modeling the probability of sequences of words.

Language Model (LM): Guessing the next word

- If the preceding context is “Thanks for all the” and we want to know how likely the next word is “fish”:

$$P(fish|Thanks\ for\ all\ the)$$

- We can also assign probabilities to entire sequences by using these conditional probabilities in combination with the chain rule

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$$

Language Model (LM) Evaluation: Perplexity

- We evaluate language models by examining how well they predict unseen text.

- Perplexity is the inverse probability that model assigns to the test set, normalized by the test set length:

$$\begin{aligned} \mathrm{PP}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}} \end{aligned}$$

- Using the chain rule:

$$\mathrm{PP}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i|w_{1:n-1})}}$$

Perplexity as a branching factor of a language

- Intuitively, perplexity measures the amount of surprise as **average number of choices**.

- If in the Shannon game, perplexity of a model predicting the next word is 10, this means that it chooses on average between 10 equiprobable words (has an average branching factor of 10).

- Let's suppose a sentence consisting of random digits. What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Corpus: source of text data

- Corpus (pl. corpora) = a computer-readable collection of text and/or speech, often with annotations.
- We can use corpora to gather probabilities and other information about language use.
- We can say that a corpus used to gather prior information, or to train a model, is **training data**.
- **Testing data**, by contrast, is the data one uses to test the accuracy of a method.
- We can distinguish types and tokens in a corpus
  - **type** = distinct word (e.g., "elephant")
  - **token** = distinct occurrence of a word (e.g., the type "elephant" might have 150 token occurrences in a corpus)
- Corpora can be raw, i.e. text only, or can have annotations

Power laws in the language

- **Zipf** (1936;1949): The *r*-th most frequent word has a frequency *f*(*r*) that scales according to

$$f(r) \propto \frac{1}{r^{\alpha}}$$

- *r* – frequency rank of a word, *f(r)* – frequence in a corpus, $\alpha \approx 1$
- The most frequent word (r = 1) has a frequency proportional to 1, the second most frequent word (r = 2) has a frequency proportional to 1/2, the third most frequent word has a frequency proportional to 1/3, and so forth.

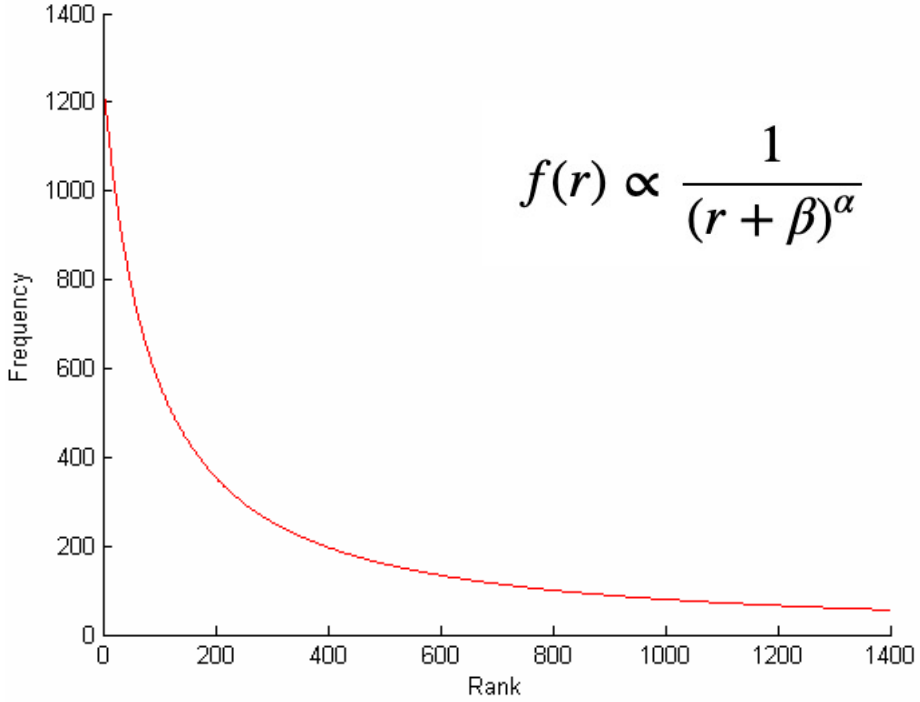
- **Mandelbrot** (1953;1962): "shifting" the rank by an amount β
- for $\alpha \approx 1$ and $\beta \approx 2.7$

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}}$$

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4176592/

Power laws in the language

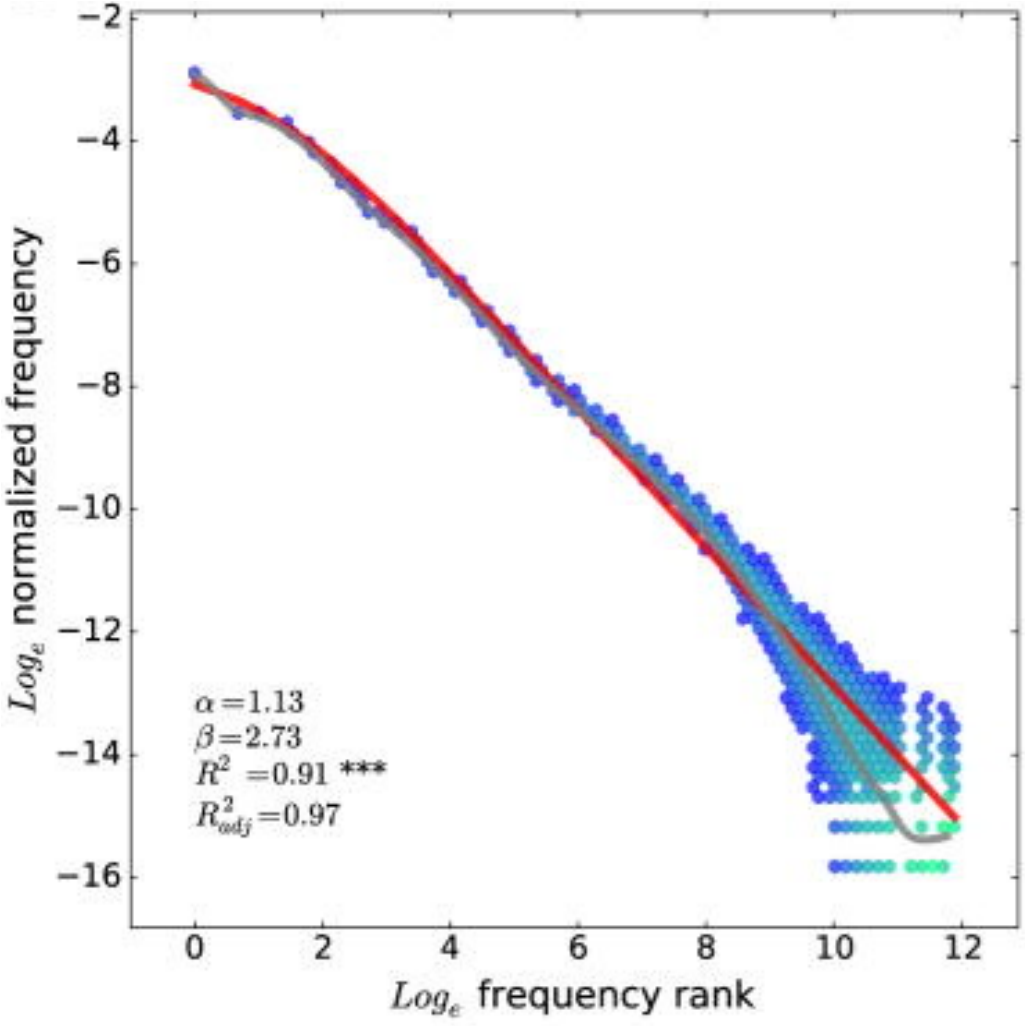
- The most frequent word (r = 1) has a frequency proportional to 1, the second most frequent word (r = 2) has a frequency proportional to 1/2, the third most frequent word has a frequency proportional to 1/3, and so forth.



https://www.researchgate.net/figure/1-Ideal-plot-of-the-Zipf-Mandelbrot-law-To-be-sure-we-can-examine-how-the-training_fig5_268296381

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4176592/

Mandelbrot's law on the American National Corpus (ANC)



https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4176592/

PLAN OF THE LECTURE

- Language Modelling (LM) Task
- Statistical Language Models
- Neural Language Models: Feedforward, Recurrent, Transformer

n-gram Language Models

- Let us assume we want to predict the next word, based on the previous contexts of

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
Every	week	a	go	to	a	swimming	_____

- We want to find the likelihood of w_7 being the next word, given that we have observed $w_1,\ldots w_6$.: $P(w_7|w_1,\ldots w_6)$.
- For the general case, to predict w_n, we need statistics to estimate $P(w_n|w_1,\ldots w_{n-1})$.

Problems:

- sparsity: the longer the contexts, the fewer of them we will see instantiated in a corpus
- storage: the longer the context, the more memory we need to store it
- solution: limit the context length to a fixed n

n-gram Language Models

Predicting a word given its (n-1) predecessors.

The **n** in n-gram models:

- n is the length of the observations a model is trained on
- e.g. a **bigram** model predicts the next word on the basis of **one** predecessor, a **trigram** model on the basis of **two** etc.
- a **unigram LM** is also called bag-of-words model: no sequences are taken into account

Unigram models: n=1

- Unigram models are initialized from word frequencies.
- They do not take context into account: $P(w_n|w_1,\ldots w_{n-1}) \approx P(w_n)$
- The probability of a sentence is the product of the probability of the words:

P(`Every week I go to a swimming`) =
P(`Every`)*P(`week`)*P(`I`)*P(`go`)*P(`to`)*P(`a`)*P(`swimming`)

Bag-of-words model: order of words is irrelevant.

Applications: Language identification, Information Retrieval, ..

Bigram models: n=2

- Bigram models are initialized from bigram frequencies taking one preceding token into account:

$$P(w_n|w_1,\ldots w_{n-1}) \approx P(w_n|w_{n-1})$$

- The probability of a sentence is the product of the probability of the words, given the preceding word:

P(`Every week I go`) =
P(`Every`|`<BOS>`)*P(`week`|`Every`)*P(`I`|`week`)*P(`go`|`I`) =
exp(log(P(`Every`|`<BOS>`))) * exp(log(P(`week`|`Every`))) *
exp(log(P(`I`|`week`))) * exp(log(P(`go`|`I`))).

- For implementation, log-probabilities are used, since these probabilities are generally small: problems with floating-point machine precision.

Markov Assumptions

Probability of symbol w_k at point in time t:

$P(X_t = w_k \mid X_1 X_2 \ldots X_{t-1}) =$

- **Limited horizon (Markov property)**
  value of X_t is only dependent on previous state X_{t-1} :
  $= P(X_t = w_k \mid X_{t-1}) =$

- **Time invariance (stationary)**
  value of the next symbol does not depend on t:
  $= P(X_2 = w_k \mid X_1)$

Markov Model and Markov Chain

A **Markov Model** is a stochastic model that assumes the Markov property.

A **Markov Chain** is a random process that undergoes state transitions, at this obeying the Markov property: the following state is only dependent on the current state, not on earlier or future states.

n-gram models are a special case of Markov chains that can be modeled with weighted finite state automata.

WFSA as Markov Chain

A **weighted finite state automaton** WFSA=(Φ,δ,S) or WFSA = (Φ,δ,Π) consists of:

- finite set of states Φ corresponding to symbols or sequences of symbols
- transition function δ: $\Phi \rightarrow [0,1] \times \Phi$ with weights $w \in [0,1]$ and the sum of weights exiting one state must equal 1
- one start state $S \in \Phi$ OR an initial probability distribution

  Π: $\pi_i = P(X_1 = s_i)$
- all states are final states

Acceptance: determines probability of a sequence

Generation: generates a sequence according to transition weights

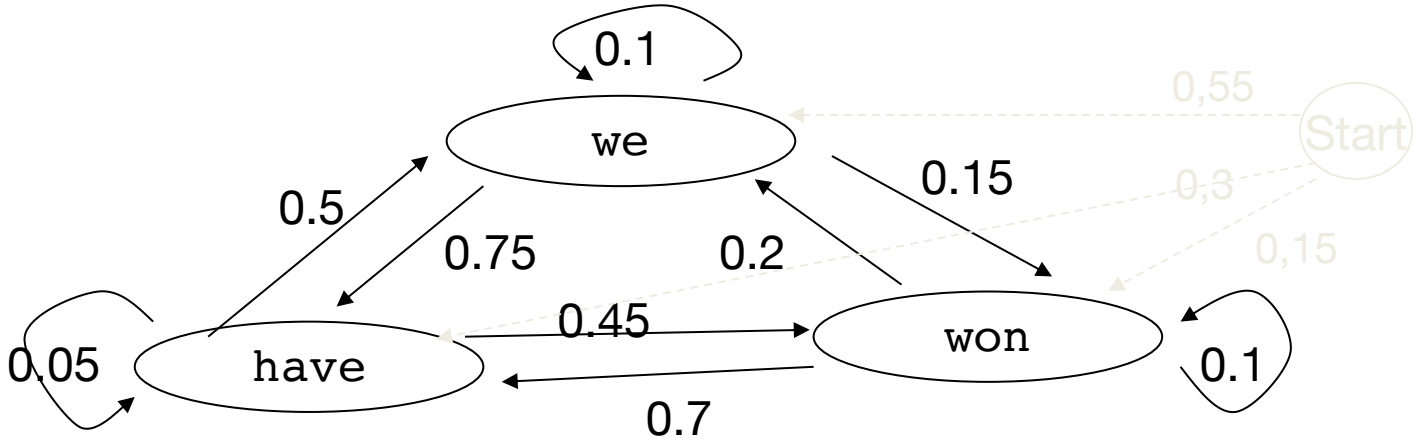
Example of a Markov Chain with horizon 1

$a_{ij}=P(X_t=w_i|\ X_{t-1}=w_j)$

δ	we	have	won
we	0.1	0.75	0.15
have	0.5	0.05	0.45
won	0.2	0.7	0.1

$\pi_i=P(X_1=w_i)$

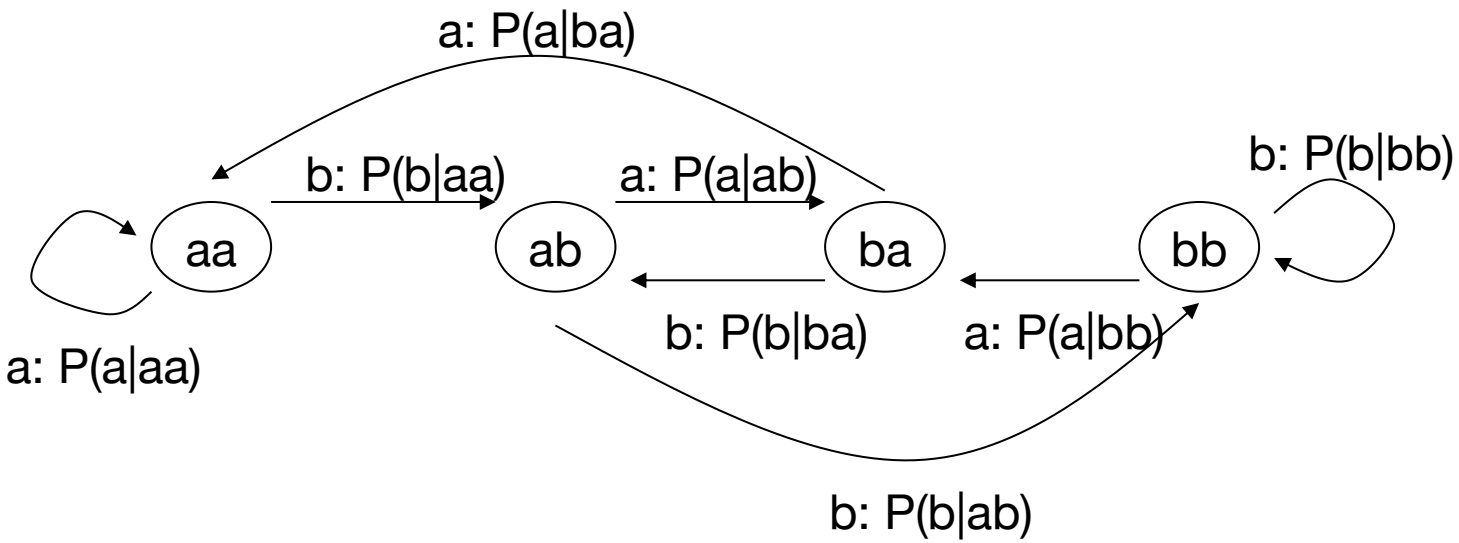
<BOS>	Π
we	0.55
have	0.3
won	0.15



this is equivalent to a bigram model

Higher order Markov Chains

Example for horizon=2, language=(ab)*.



By representing the horizon as a single state, n-gram models of arbitrary n can be formulated as Markov chains.

Text generation with a Markov Process

Generation a sequence of symbols from a Markov Chain:

```
t=1;
Start in state z_i ∈ Φ with probability π_i
While TRUE:
  Choose z^{t+1} = z_j randomly according to
  transition probs
  emit symbol s^t
  t++;
```

Examples: Shakespeare with n-gram models

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have // Every enter now severally so, let //

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain. //

Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. //

Trigram

Sweet prince, Falstaff shall die. Harry of Mommouth's grave. // This shall forbid I should be branded, if renown made it empty. //

4-gram

King Henry. What! I will go seek the traitor Glouchester. Exeunt some of the watch. A great banquet serv'd in; // Will you not tell me who I am? //

Growth in the number of parameters for n-gram models

Assuming, a speaker of a language has 20,000 words of active vocabulary and produces language according to an n-gram model. How many model parameters (probabilities of transitions) need to be stored?

Order of MC	n-gram	calculation	parameters
-	unigram	20,000	2E4
1	bigram	$20{,}000^2$	4E8
2	trigram	$20{,}000^3$	8E12
3	4-gram	$20{,}000^4$	1.6E17
…	…	…	…

- How large needs the corpus to be in order to reliably estimate the parameters for a 4-gram model?
- The main influence is the number of symbols. Can we group words into classes in order to reduce this number?

Maximum likelihood estimation (MLE)

- MLE maximizes the probability of the training corpus.
- Loss (log-likelihood):

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

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

- We train our n-gram model from **corpus counts.** Let $C(w_1,\ldots w_n)$ be the number of times we see the sequence $w_1,\ldots,w_n$ in our corpus. Then, the empirical probability of seeing w_n after $w_1,\ldots w_{n-1}$ is:

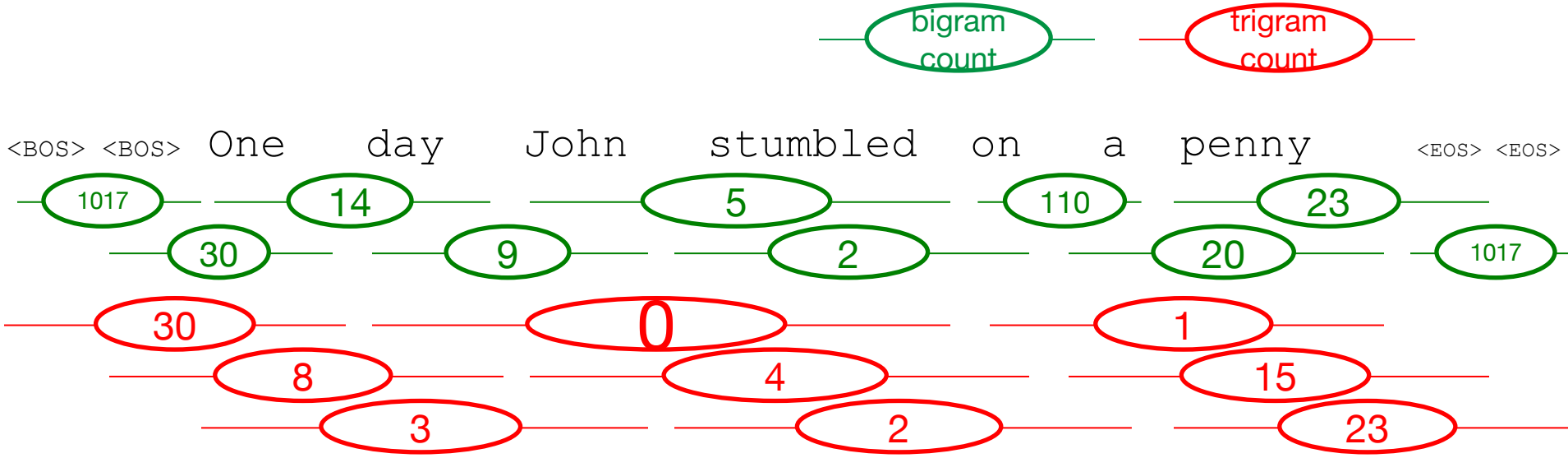
$$P(w_n \mid w_1,\ldots,w_{n-1}) = \frac{C(w_1,\ldots,w_n)}{C(w_1,\ldots,w_{n-1})}$$

- The empirical probability corresponds here to the relative frequency of observing w_n after $w_1,\ldots w_{n-1}$ has been observed already.

https://www.youtube.com/watch?v=oaYsCVtHveQ

Accepting with MLE-models

What happens when a n-gram model trained with MLE encounters an unseen word, or unseen n-gram in a sentence *S*?



$P(\text{stumbled}|\text{day John}) = \frac{C(\text{day John stumbled})}{C(\text{day John})} = \frac{0}{9} = 0$

➔ P(S) = 0 !!

Problems with MLE Models

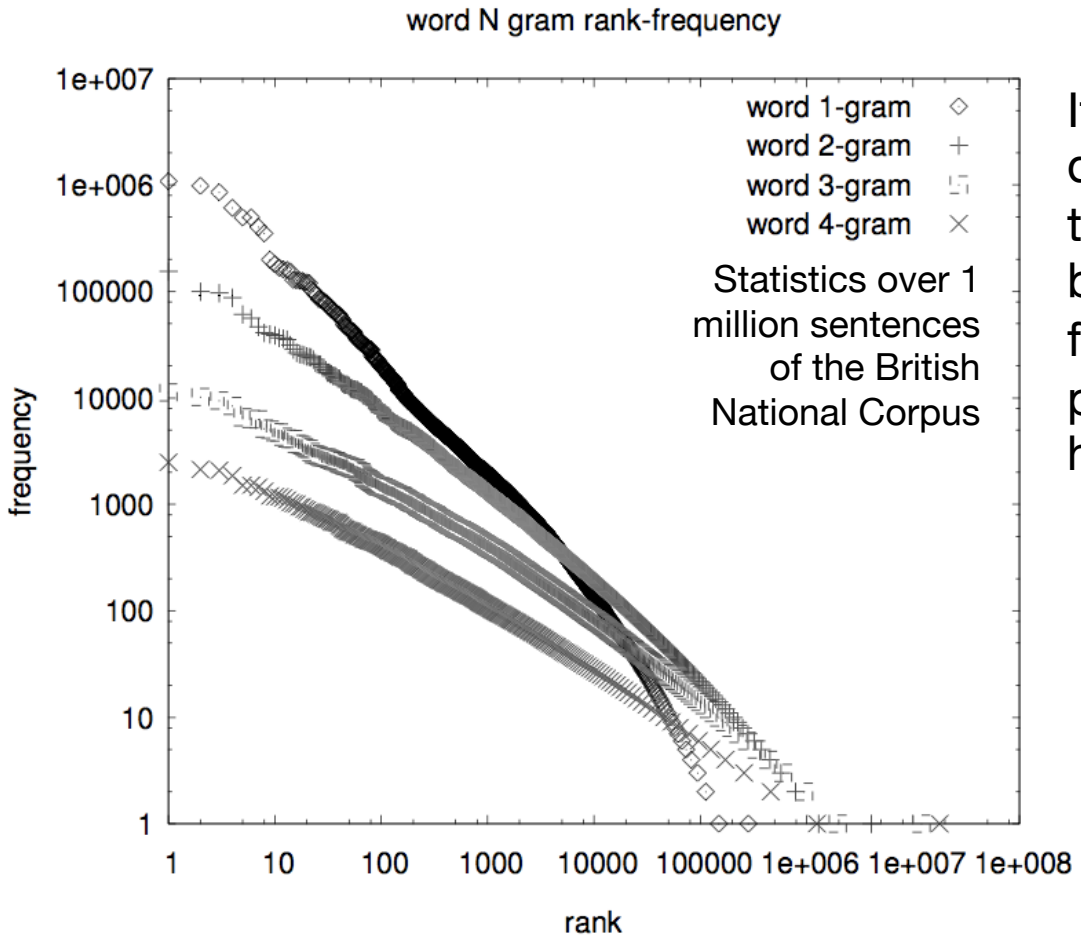
- MLE models maximizes the probability on the observed training data and do not waste any probability mass on unobserved events
- However, we are more interested in applying our model to unseen data
- If no probability mass is assigned to unseen events, then all sentences with unseen events all get a probability of 0, are not comparable

Solving the problem with larger training corpora?

- remember the number of parameters for n-gram models?
- vocabulary size of natural languages is infinite
- Power-law frequency distribution: it is very likely to encounter unseen words in unseen text, even more so unseen n-grams

➔ need method to account for unseen events!

Zipf's law: freq(rank) ~ rank^{-z}



If one orders words by decreasing frequency, then the relation between rank and frequency follows a power-law. This is a heavy tail distribution.

➔ most words are rare. most n-grams are even rarer

Smoothing

- smoothing is a way to deal with unobserved n-grams
- works by taking a little bit of the probability mass from higher counts and shift it to zero counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Smoothing

- smoothing is a way to deal with unobserved n-grams
- works by taking a little bit of the probability mass from higher counts and shift it to zero counts

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Laplace smoothing “add one”

Idea: We add 1 to all possible frequency counts

For vocabulary size V:

unigram $$P_{Lap}(w) = \frac{C(w)+1}{N+V}$$

bigram $$P_{Lap}(w_i \mid w_{i-1}) = \frac{C(w_{i-1},w_i)+1}{C(w_{i-1})+V}$$

n-gram $$P_{Lap}(w_i \mid w_{i-n+1},...,w_{i-1}) = \frac{C(w_{i-n+1},...,w_i)+1}{C(w_{i-n+1},...,w_{i-1})+V}$$

Problem with Laplace smoothing

Not suited for large vocabulary sizes!

Example: C(a b c)=9, C(a b)=10. Vocabulary size: 100K

$$P_{MLE}(\text{c} \mid \text{a b}) = \frac{C(\text{a b c})}{C(\text{a b})} = 0.9$$

$$P_{Lap}(\text{c} \mid \text{a b}) = \frac{C(\text{a b c}) + 1}{C(\text{a b}) + 100000} \approx 0.0001$$

What about adding a smaller value δ as in $P_{Lap}(w) = \frac{C(w) + \delta}{N + \delta \cdot V}$?

- Laplace Smoothing is dependent on vocabulary size.
- “Add δ” still does not work well: for small δ, unseen events are overly punished. For larger δ, the same problem as with “add one” smoothing occurs. Commonly used: δ=0.5

Combining estimators

- Estimators up till now assign the same probability to all unseen events
- idea: Use observed (n-1)-grams in unobserved n-gram for estimating its probability

Linear interpolation (mixture model): Combine probabilities using a linear combination from different n:

$$P_{li}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n \mid w_{n-1}) + \lambda_3 P_3(w_n \mid w_{n-2}w_{n-1}) \text{ where } 0 \leq \lambda_i \leq 1 \text{ and } \sum_i \lambda_i = 1$$

how to set the λs? E.g. with EM training, see next lecture.

This works well, but there are even smarter combination schemes …

Katz's backing off

- Idea: different models are consulted in order depending on their specificity: we use the more detailed model if it seems reliable enough.

$$P_{bo}(w_i \mid w_{i-n+1},\ldots,w_{i-1}) = \begin{cases} \text{if } c(w_{i-n+1},\ldots,w_i) > k: & (1-d_{w_{i-n+1},\ldots,w_i})\dfrac{C(w_{i-n+1},\ldots,w_i)}{C(w_{i-n+1},\ldots,w_{i-1})} \\ \text{otherwise:} & \alpha_{w_{i-n+1},\ldots,w_{i-1}} P_{bo}(w_i \mid w_{\mathbf{i-n+2}},\ldots,w_{i-1}) \end{cases}$$

- if the observed n-gram **has been seen more than k times** in the training, we use an MLE estimate, discounted by some *d* (e.g. using Good-Turing).
- If we back off to a lower order n-gram, the estimate has to be normalized by some *α*, such that only the probability mass left by the discounting is distributed.

This works well in practice, but breaks down in some cases: If e.g. "a b" is a common bigram, "c" is a common word but we never saw "a b c", this true 'grammatical zero' would still get a fairly high estimate.

Issues with n-gram Language Models

Curse of dimensionality:

- increased dimensionality: the volume of the space increases so fast that the available data become sparse.
- Sparsity is problematic for any method that requires statistical significance.
- Smoothing methods for dealing with the sparsity such as Laplace smoothing, linear interpolation and back-off helps to some extend.
- Similarity of the words are not taken into account.

PLAN OF THE LECTURE

- Language Modelling (LM) Task
- Statistical Language Models: n-grams
- Neural Language Models: Feedforward, Recurrent, Transformer

Neural Language Model (Bengio et al., 2003)

Summary of Approach:

1. Associate with each word in the vocabulary V a distributed word feature vector: a real-valued vector in R^m, where $m << |V|$ vocab size.
2. Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. Learn simultaneously the word feature vectors (a.k.a. embeddings) and the parameters of that probability function.

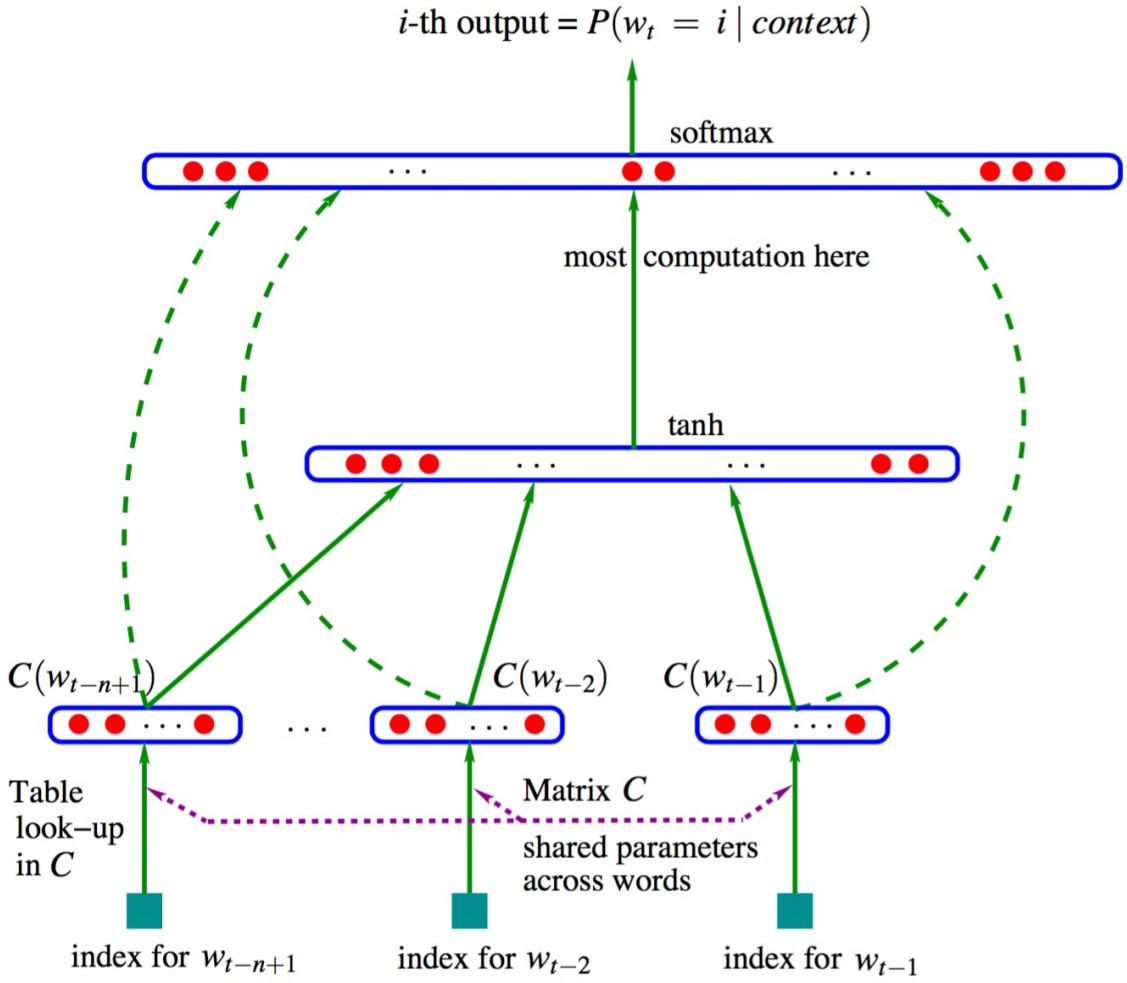
Objective: learn a good model (low perplexity on held-out) for

$$f(w_t,..w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$$

subject to:

$$\sum_{i=1}^{|V|} f(i,w_{t-1},...,w_{t-n+1}) = 1$$

Neural Architecture: NN-LM



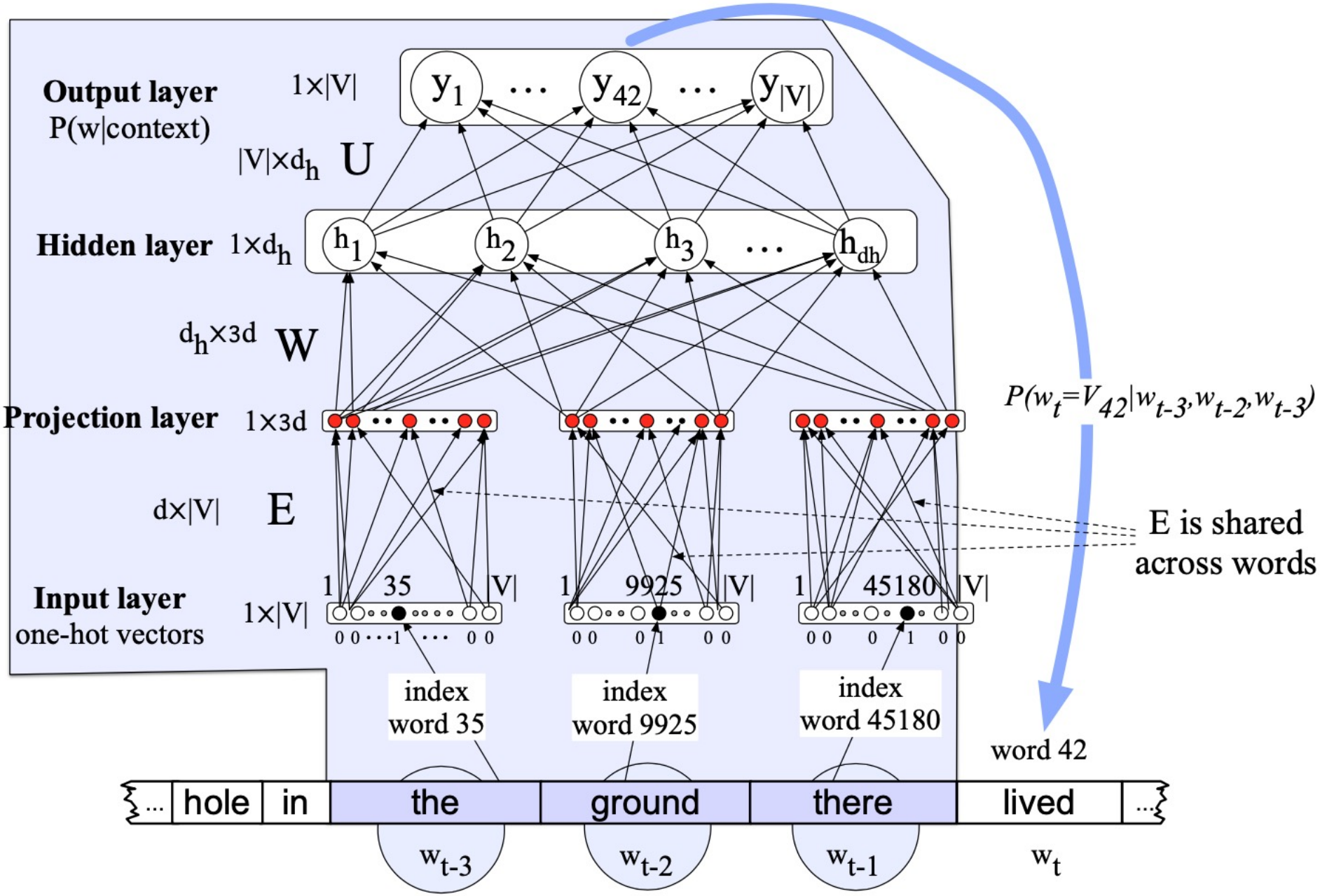
- softmax normalizes P:

$$P(w_t \mid w_{t-1},\ldots,w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

- y: un-normalized log-probs

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

- b,d: biases
- W: words to output weights (direct connections)
- H: hidden layers weights
- U: hidden-to-output weights
- C(i) is i-th word feature vector
- x: concatenation of C(w)'s


Output layer
P(w|context)
1×|V|
y_1 ⋯ y_{42} ⋯ $y_{|V|}$
|V|×d_h U
Hidden layer 1×d_h
h_1 h_2 h_3 ⋯ h_{dh}
d_h×3d W
Projection layer 1×3d
d×|V| E
Input layer
one-hot vectors
1×|V|
1 35 |V|
1 9925 |V|
1 45180 |V|
0 0 ⋯ 1 ⋯ 0 0
0 0 0 1 0 0
0 0 0 1 0 0
index word 35
index word 9925
index word 45180
$P(w_t=V_{42}|w_{t-3},w_{t-2},w_{t-3})$
E is shared across words
word 42
... hole in the ground there lived ...
w_{t-3}
w_{t-2}
w_{t-1}
w_t

Intuition: the use of similarities between representations

Assume these pairs are similar:

- dog – cat
- the – a
- room – bedroom
- is – was
- running – walking

Then, "`The cat is walking in the bedroom`" could transfer probability mass to:

- `The cat is walking in the bedroom`
- `A dog was running in a room`
- `The cat is running in a room`
- `A dog is walking in a bedroom`
- `The dog was walking in the room`
- ...

Training

- Overall parameter set: θ=(C,ω), where ω are the network's parameters and C is the embedding matrix
- Training: Maximize corpus likelihood L:

$$L = \frac{1}{T}\sum_{t} \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta)$$

- R(θ): Regularization term (~smoothing): prevent overfitting, here by weight decay penalty

Stochastic gradient ascent: iterative update with learning rate ε:

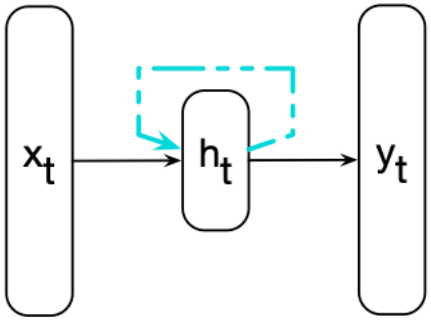
$$\theta \leftarrow \theta + \varepsilon \frac{\partial \log \hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1})}{\partial \theta}$$

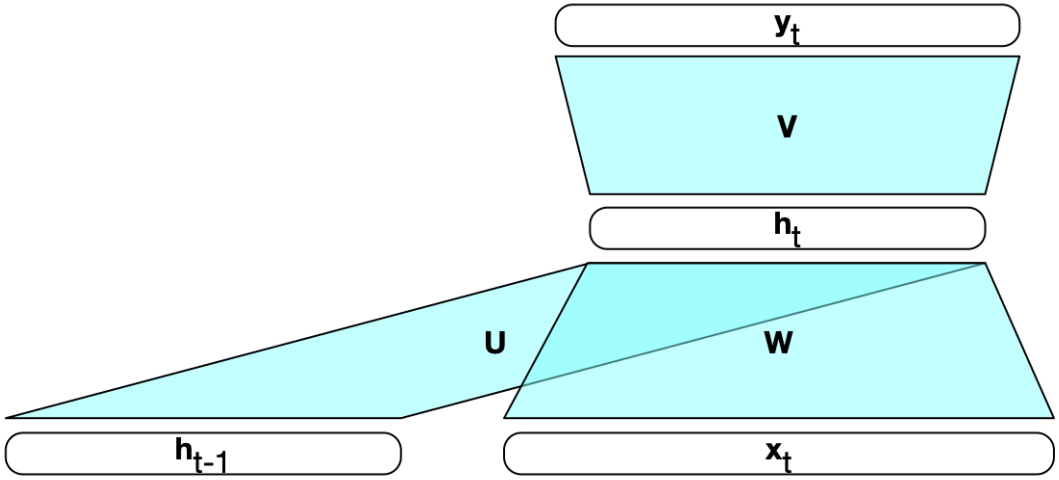
PLAN OF THE LECTURE

- Language Modelling (LM) Task
- Statistical Language Models: n-grams
- Neural Language Models: Feedforward, Recurrent, Transformer

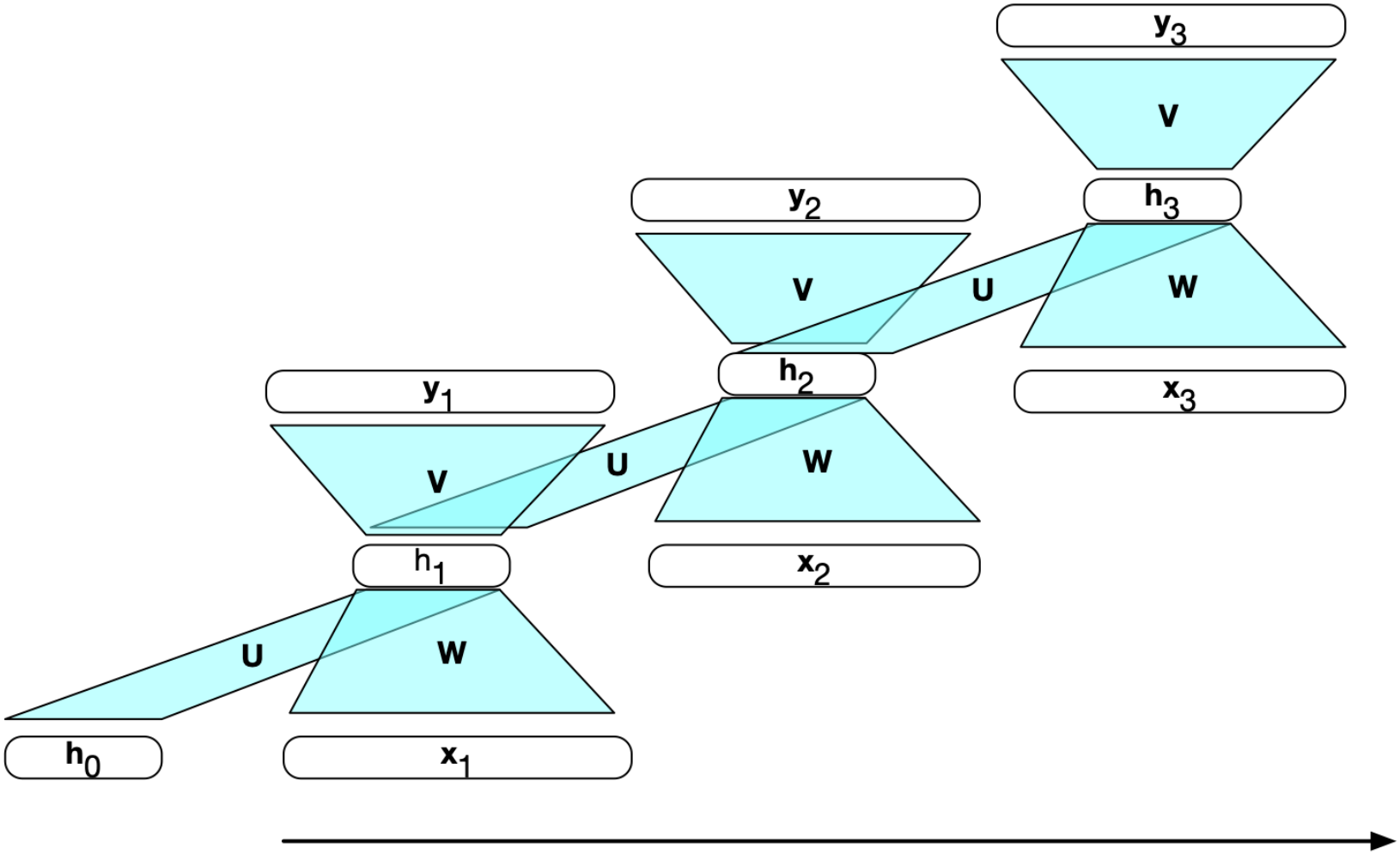
Recurrent Neural Networks

- Simple recurrent neural network (Elman, 1990):





Recurrent Neural Networks



Recurrent NN-LM: ‘infinite’ History (Mikolov et al. 2010)

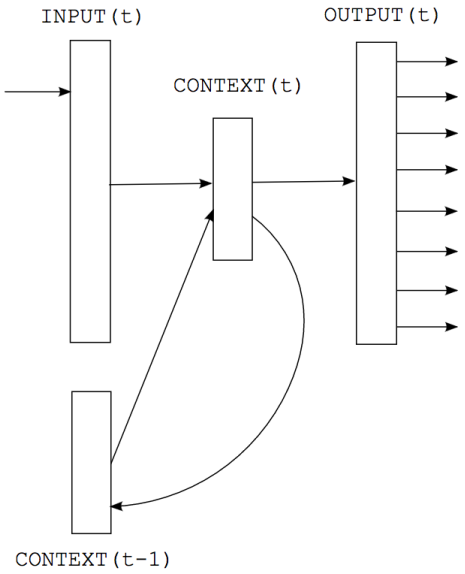


Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

- Key idea: use a recurrent neural network
- A single ‘context’ vector (~300 dimensions) encodes ‘all the history’, computed from the previous context and the input
- input: again, word embedding
- output: again, softmax

Training RNN as language models

- This hidden layer is then used to generate an output layer which is passed through a softmax layer to generate a probability distribution over the entire vocabulary

$$\begin{aligned}\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)\end{aligned}$$

- The probability that a particular word *i* in the vocabulary is the next word is represented by y[*i*], the *i*th component of y

$$P(w_{t+1} = i | w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

- Probability of the entire sequence:

$$\begin{aligned}P(w_{1:n}) &= \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^{n} \mathbf{y}_i[w_i]\end{aligned}$$

Training RNN as language models

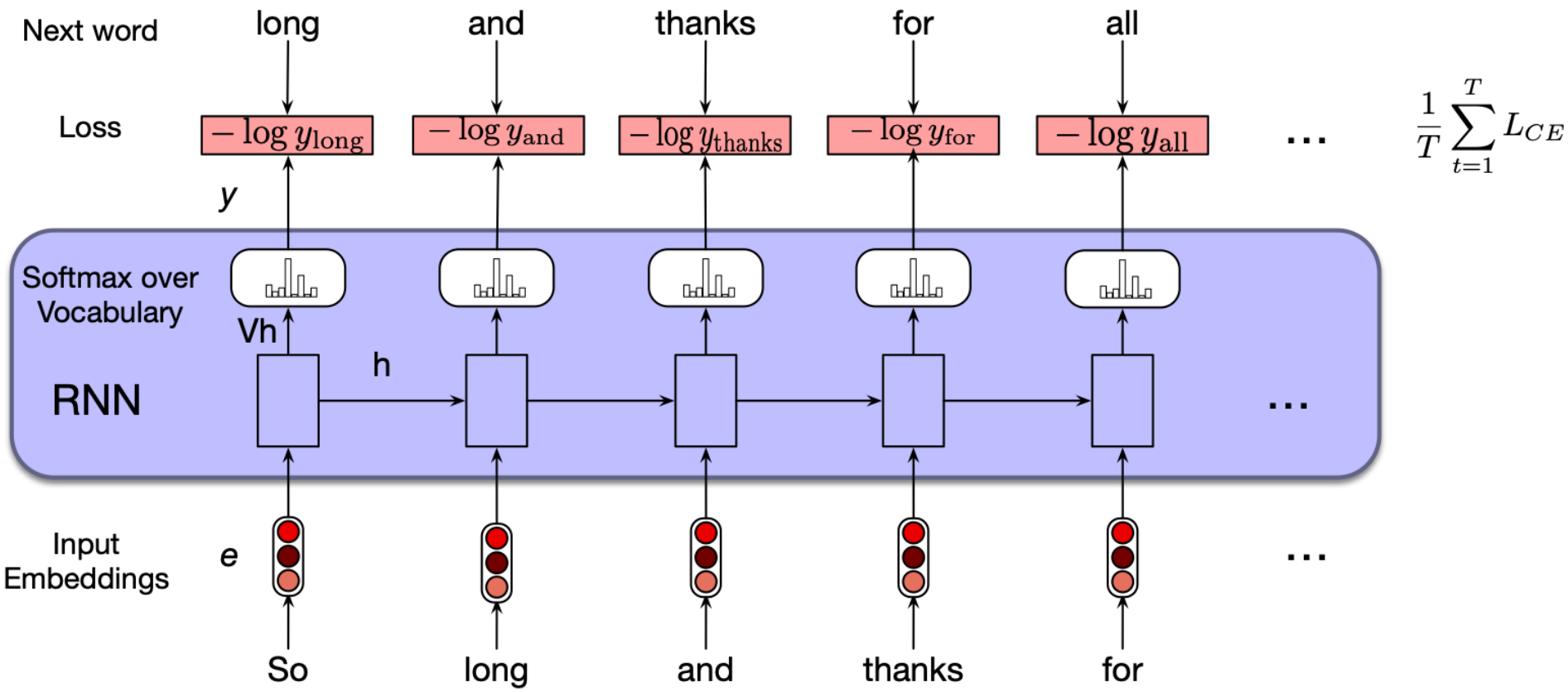
- Using cross-entropy loss, measuring the difference between a predicted probability distribution and the correct distribution:

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

- In the case of language modelling, the cross-entropy is determined by the probability the model assigns to the correct next word.
- At time *t* the CE loss is the negative log probability the model assigns to the next word in the training sequence:

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

Training RNN as language models



RNNs greatly imporved perplexity

n-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

↓ Perplexity improves (lower is better)

https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

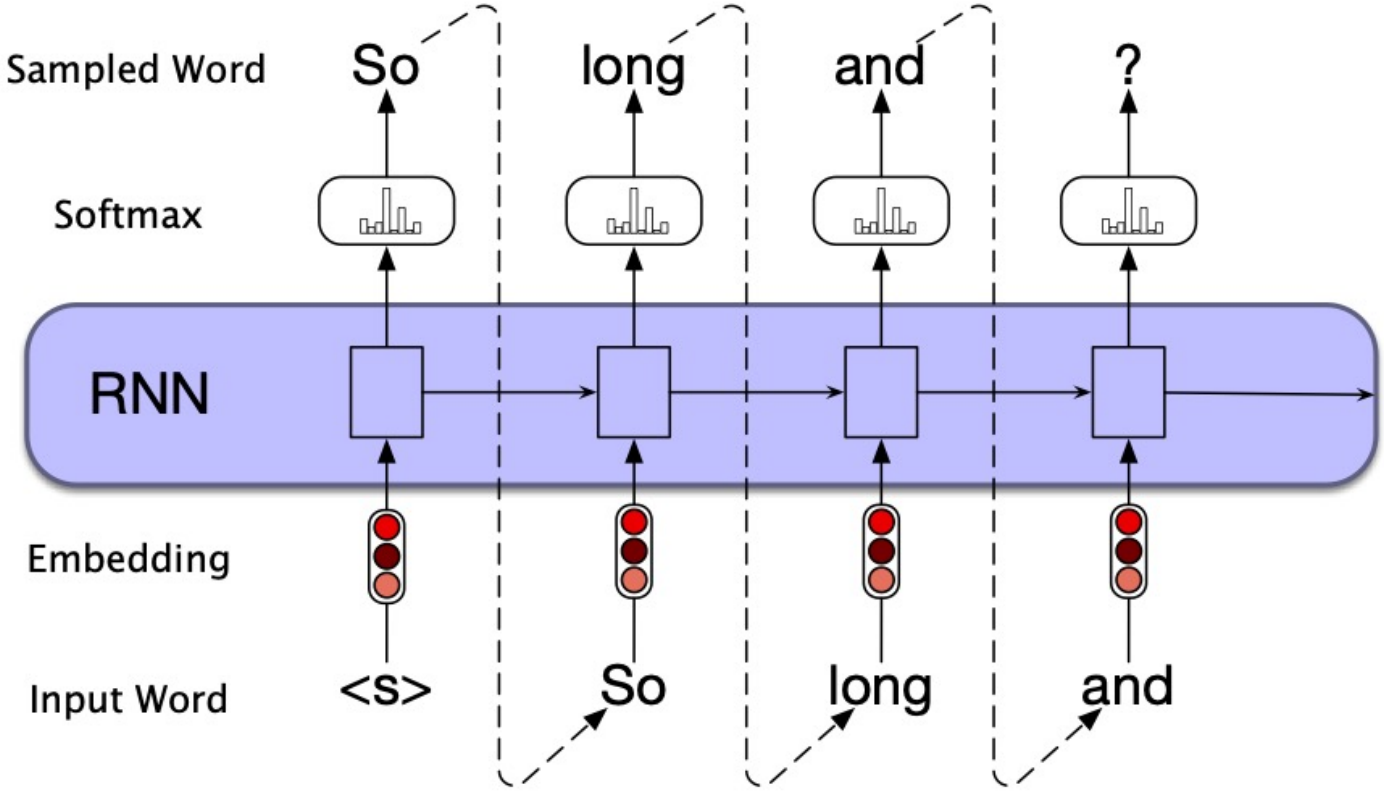
Generation with RNNs

1. Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>, as the first input.

2. Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.

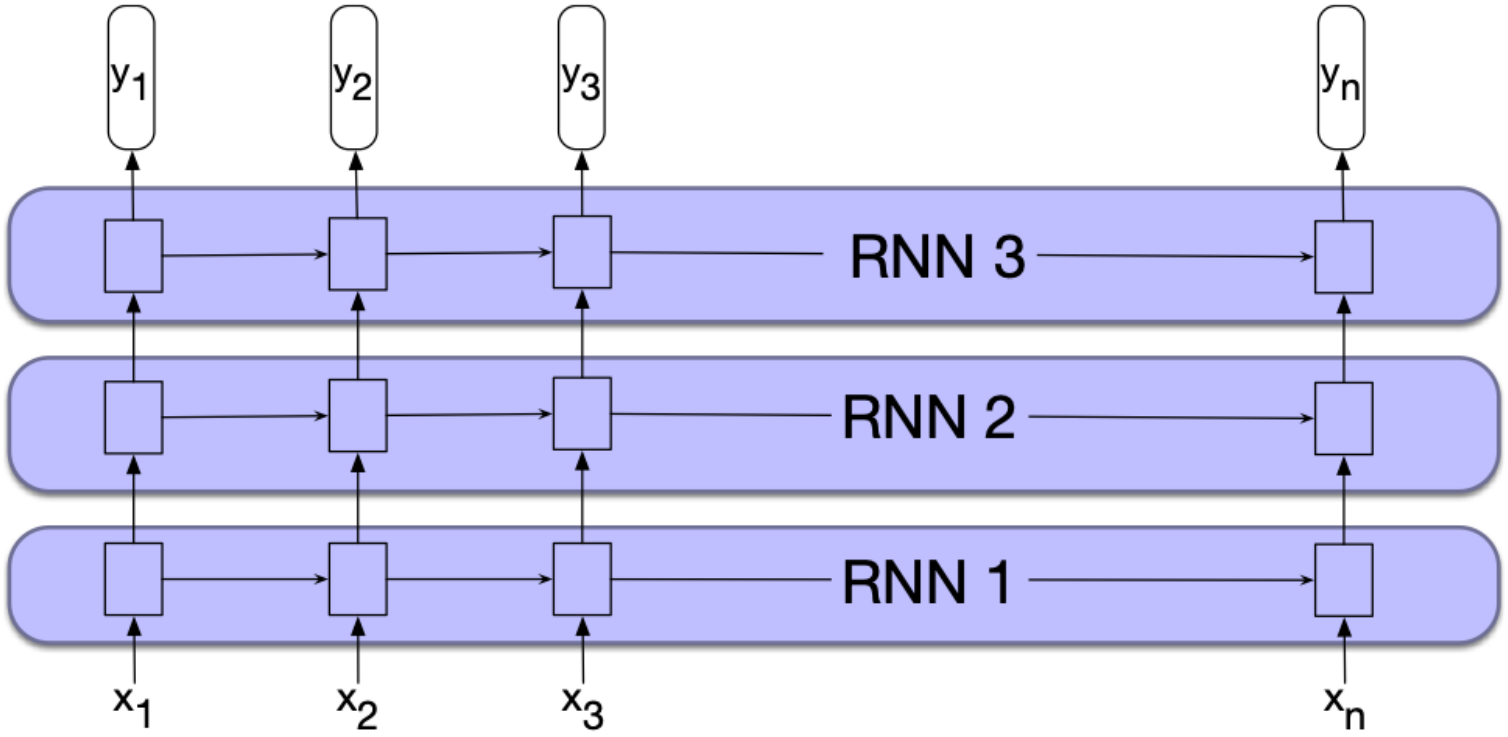
3. Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.

Generation with RNNs



Stacked RNNs

- The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

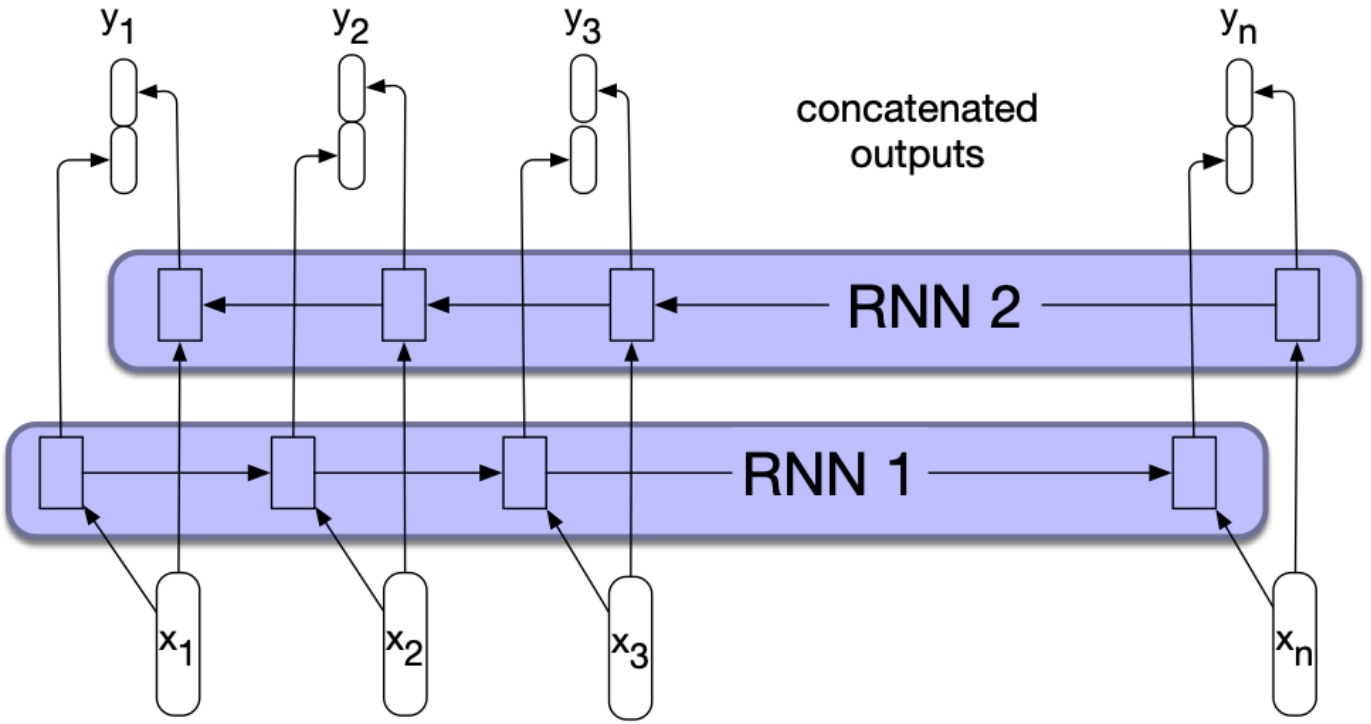


Bidirectional RNNs

- Combines two independent RNNs:

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t) \qquad \mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

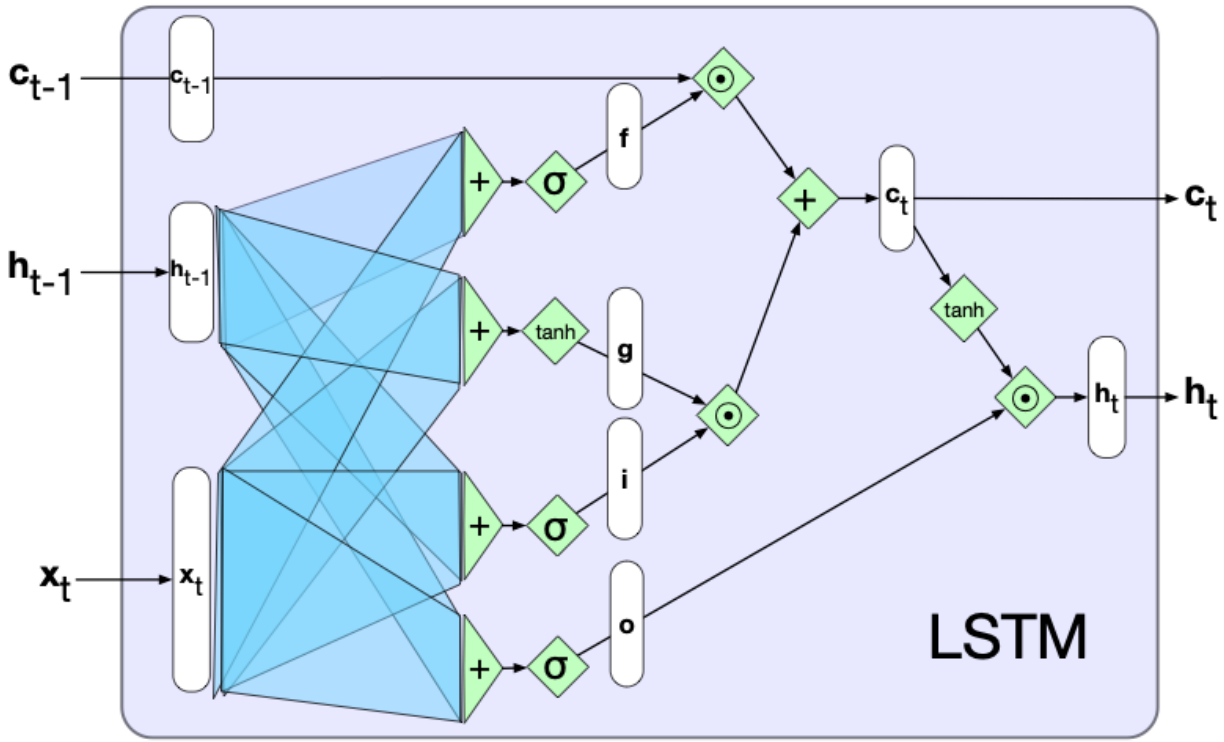
$$\mathbf{h}_t = [\mathbf{h}_t^f ; \mathbf{h}_t^b] = \mathbf{h}_t^f \oplus \mathbf{h}_t^b$$



LSTMs cells can be used in LMs

- Combines two independent RNNs:

$$\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$



LSTMs in language models further improve perplexity

- The use of gated units in the recurrent network allows to deal with longer dependencies.
- AWD-LSTM

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
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal & Ghahramani (2016) - Variational LSTM (medium)	20M	81.9 ± 0.2	79.7 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (medium, MC)	20M	–	78.6 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (large)	66M	77.9 ± 0.3	75.2 ± 0.2
Gal & Ghahramani (2016) - Variational LSTM (large, MC)	66M	–	73.4 ± 0.0
Kim et al. (2016) - CharCNN	19M	–	78.9
Merity et al. (2016) - Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) - LSTM	–	–	82.3
Grave et al. (2016) - LSTM + continuous cache pointer	–	–	72.1
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	24M	75.7	73.2
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. (2016) - Variational RHN (tied)	23M	67.9	65.4
Zoph & Le (2016) - NAS Cell (tied)	25M	–	64.0
Zoph & Le (2016) - NAS Cell (tied)	54M	–	62.4
Melis et al. (2017) - 4-layer skip connection LSTM (tied)	24M	60.9	58.3
AWD-LSTM - 3-layer LSTM (tied)	24M	60.0	57.3
AWD-LSTM - 3-layer LSTM (tied) + continuous cache pointer	24M	53.9	52.8

Merity et al. (2018): Regularizing and optimizing LSTM language models.
Merity et al. (2018): An analysis of neural language modeling at multiple scales.

PLAN OF THE LECTURE

- Language Modelling (LM) Task
- Statistical Language Models: n-grams
- Neural Language Models: Feedforward, Recurrent, Transformer

Issues of RNNs

- Extended series of recurrent connections leads to information loss and difficulties in training
- Inherently sequential nature of recurrent networks makes it hard to do computation in parallel
- **Transformers** – an approach to sequence processing that eliminates recurrent connections and returns to architectures reminiscent of the fully connected networks

Transformers: key ideas

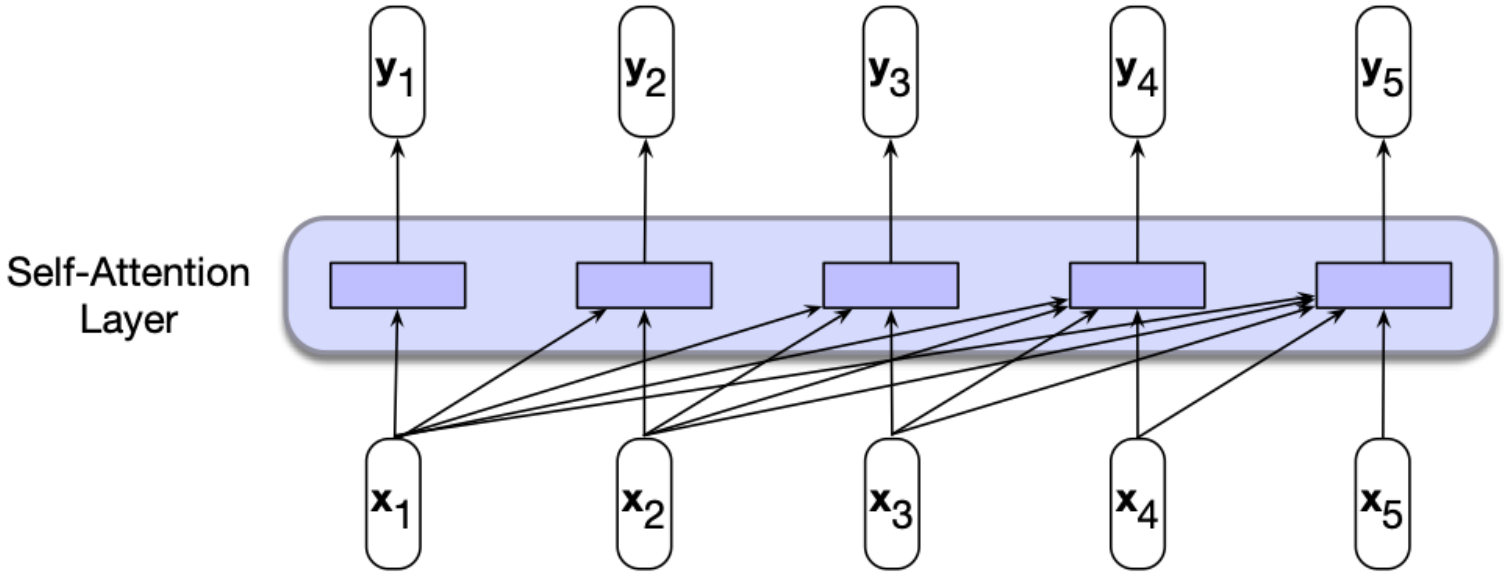
- Transformers **map sequences** of input vectors (x_1,...,x_n) to sequences of output vectors (y_1,...,y_n) of the same length.

- Transformers are **made up of stacks of transformer blocks**, which are multilayer networks made by combining simple linear layers, feedforward networks, and self-attention layers, they key innovation of transformers.

- **Self-attention** allows a network to directly extract and use information from arbitrarily large contexts **without the need to pass it through intermediate recurrent connections** as in RNNs.

Self-attention Networks: Transformers

- Information flow in a **causal** (or **masked** / **backward-looking**) **self-attention** model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.



- Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-attention Networks: Transformers

- Similarity score:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

- Computation of attention score:

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i \end{aligned}$$

- Output value y_i:

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Query, Key, Value: three different roles of embeddings

- **Query** (Q): *the current focus of attention* when being compared to all of the other preceding inputs.

- **Key** (K): In its role as *a preceding input* being compared to the current focus of attention. We'll refer to this role as a key.

- **Value** (V): a value used to compute the output for the current focus of attention.

Computing queries, keys, and values

- To capture these different roles (queries, keys, values) transformers introduce weight matrices used to project each input vector x_i into a representation of its role as a key, query, or value:

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

$$\mathbf{W}^{Q} \in \mathbb{R}^{d \times d}, \mathbf{W}^{K} \in \mathbb{R}^{d \times d}, \text{and } \mathbf{W}^{V} \in \mathbb{R}^{d \times d}$$

- Comparison calculation: $\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$

- Output calculation: $\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$

Parallel computation of queries, keys, and values

- Computations can be parallelized by taking advantage of efficient matrix multiplication routines by packing the input embeddings of the N tokens of the input sequence into a single matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$

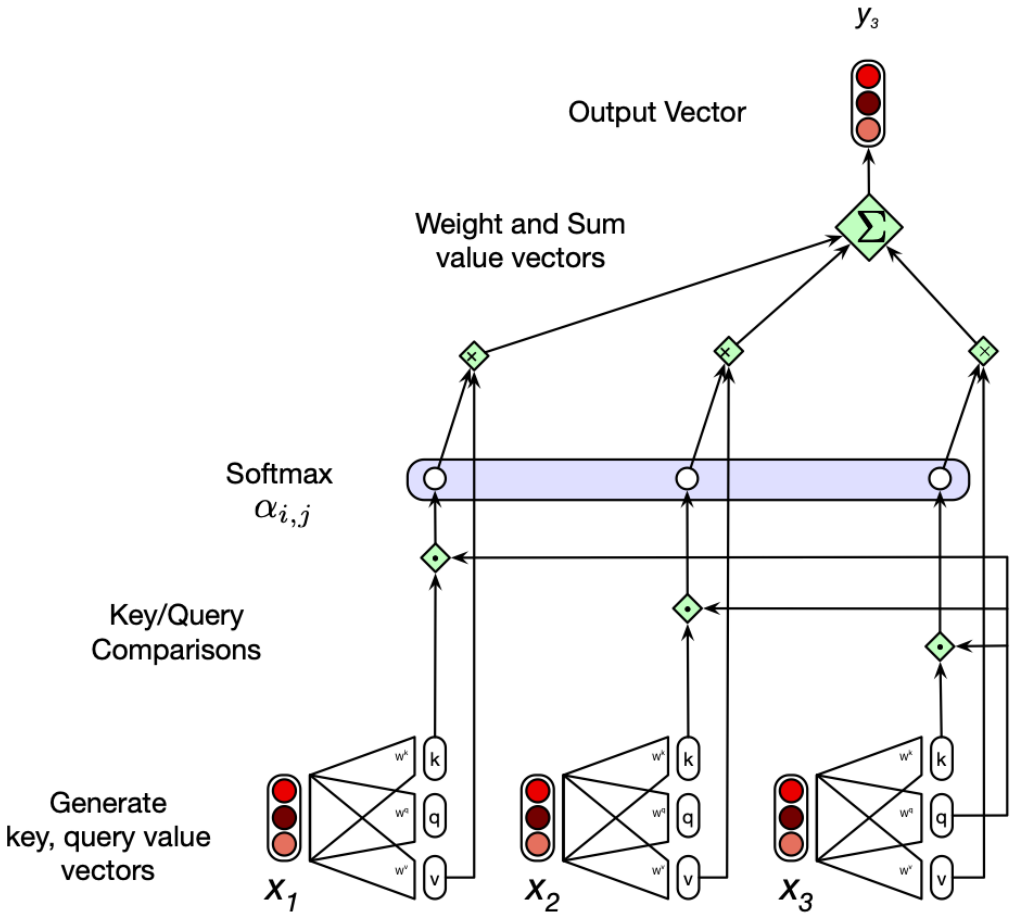
$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \;\; \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \;\; \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

- Self-attention step for an entire sequence of *N* tokens

$$\textit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

Calculating an updated value using self-attention

- Calculating the value y_3, the third element of a sequence using causal (left-to-right) self-attention.



Quadratic complexify of the self-attention

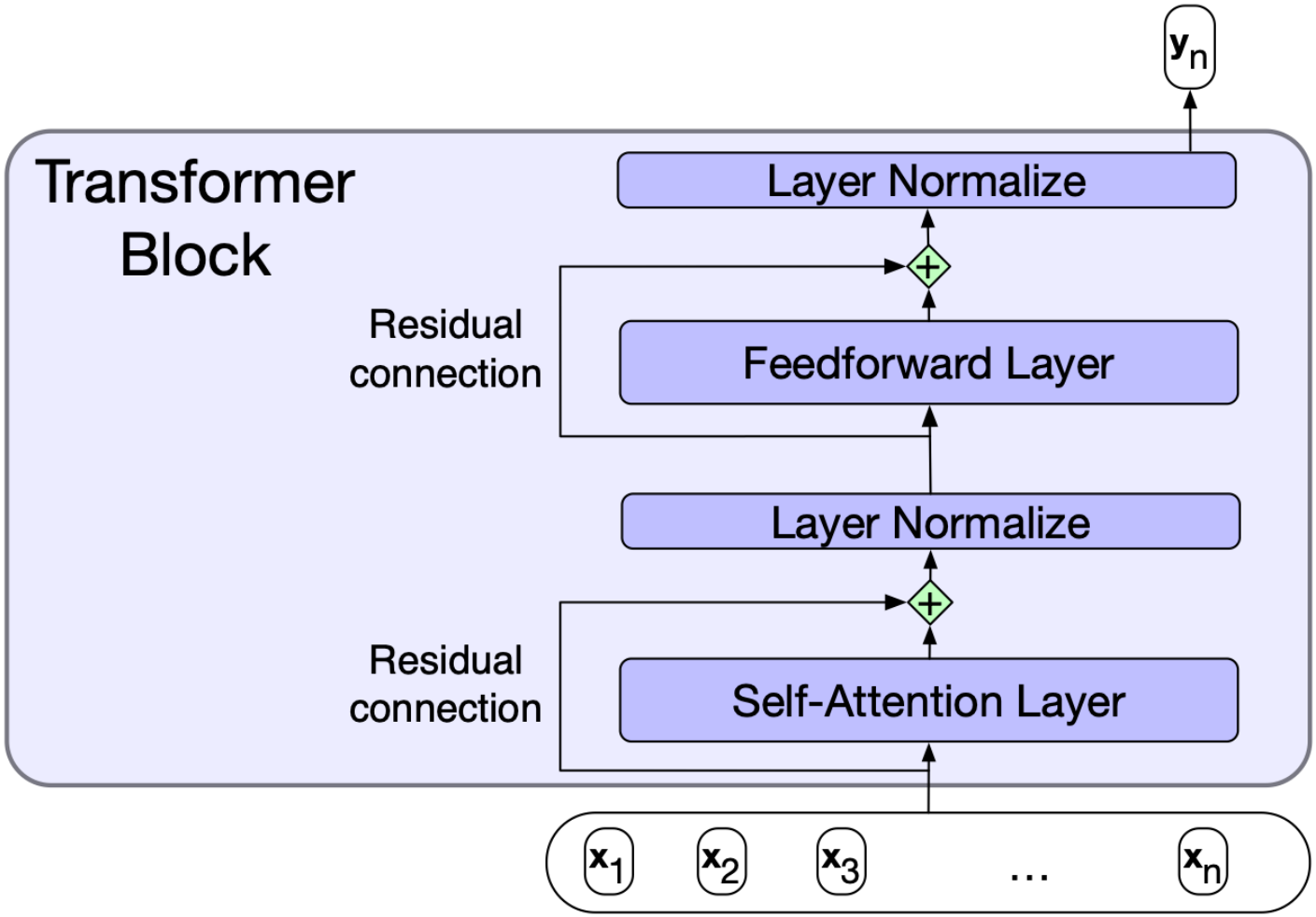
- The N x N matrix $\mathbf{QK}^T$ matrix with the upper-triangle portion of the comparisons matrix zeroed out

N

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

A transformer block showing all layers (head)



Multi-headness

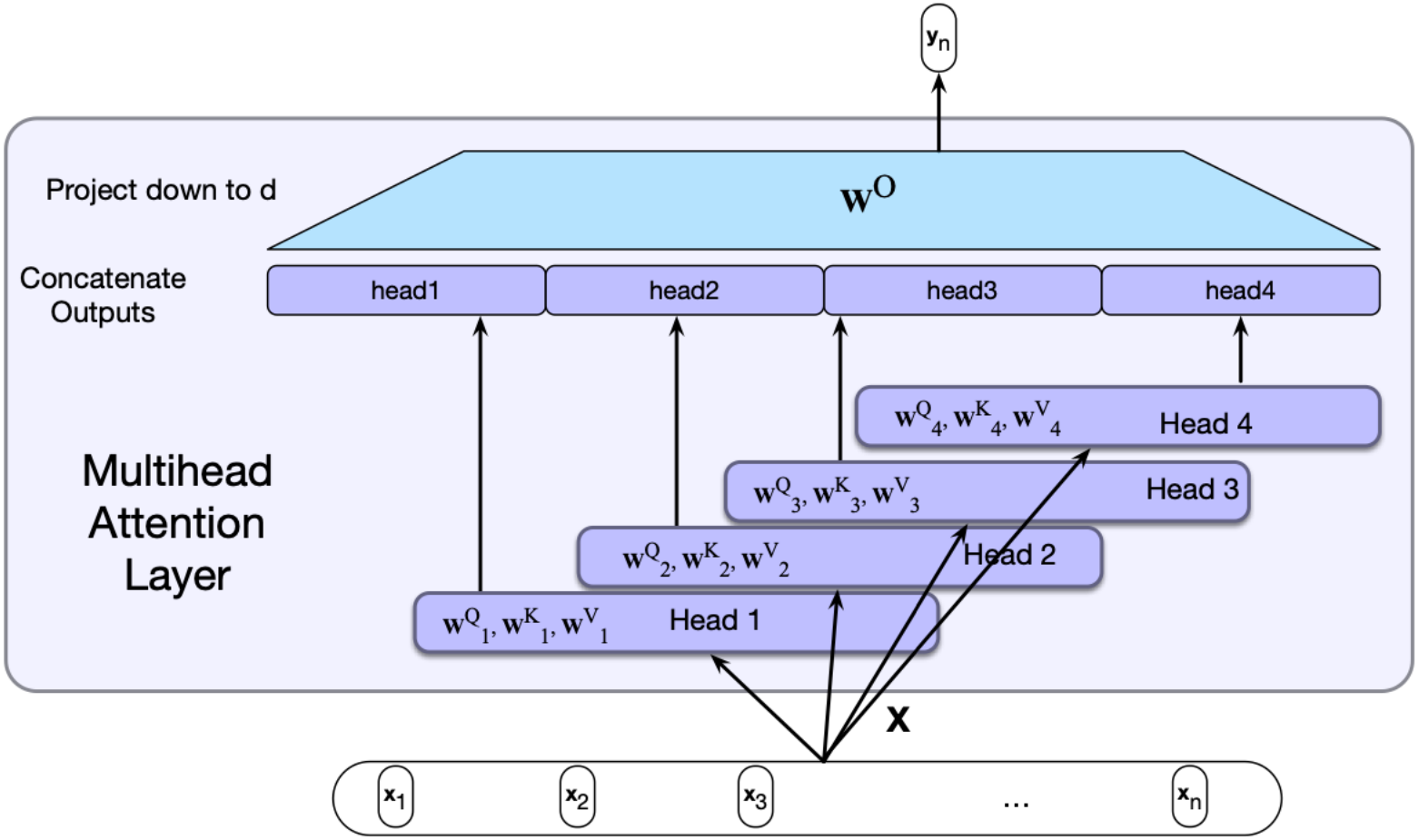
- It will be difficult for a single transformer block to learn to capture **all of the different kinds of parallel relations** among its inputs.
- Transformers address this issue with **multihead self-attention** layers:

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

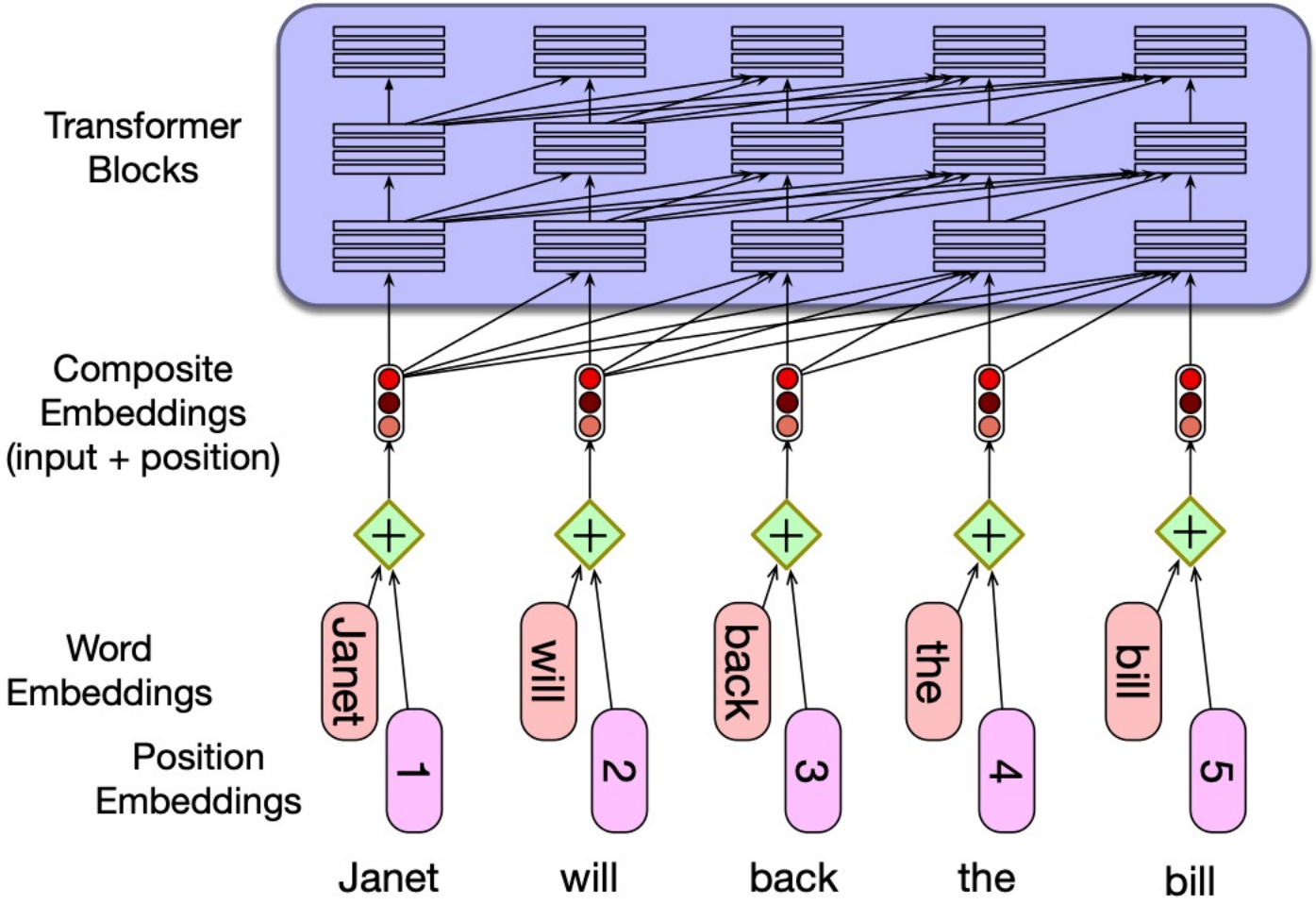
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

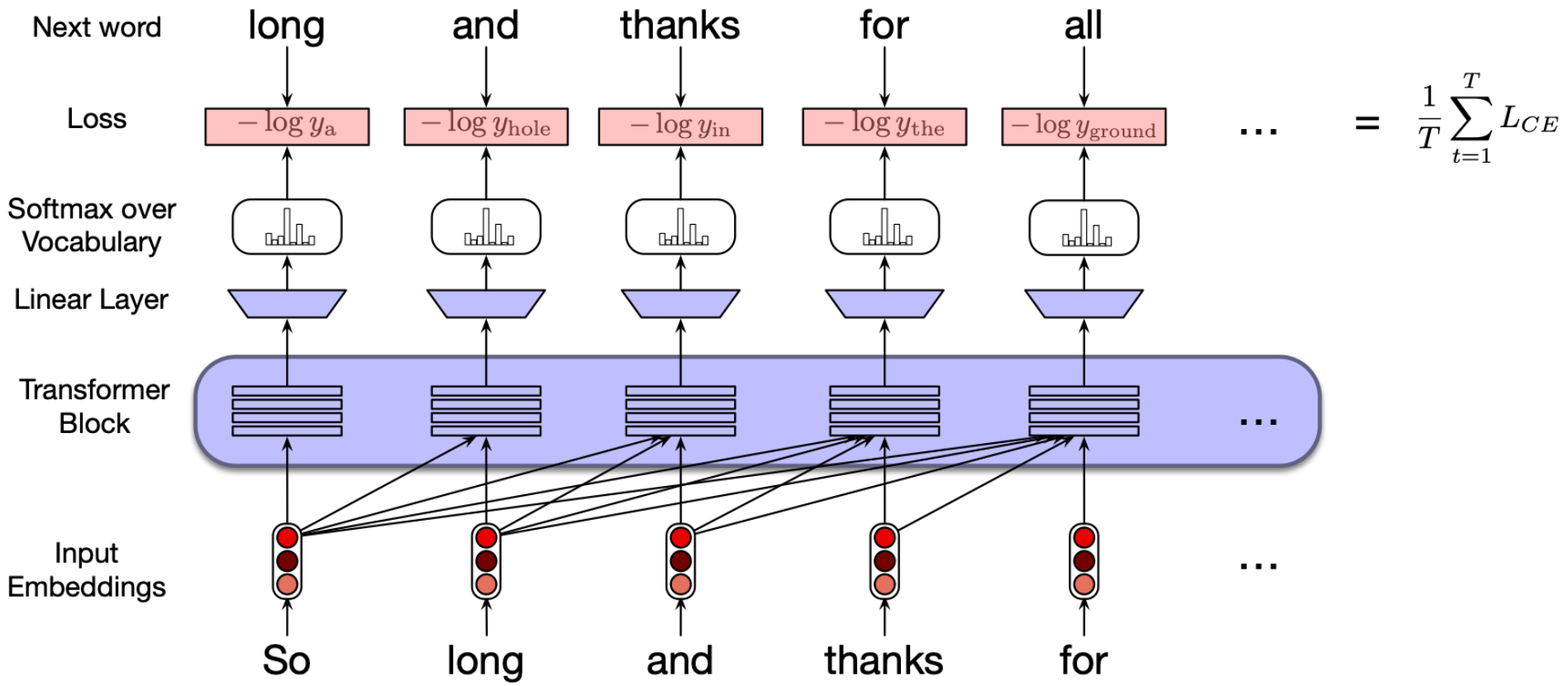
Multi-headness



Adding embedding of the absolute position to word embedding



Transformers as language models



Generative Transformer based LM: GPT2