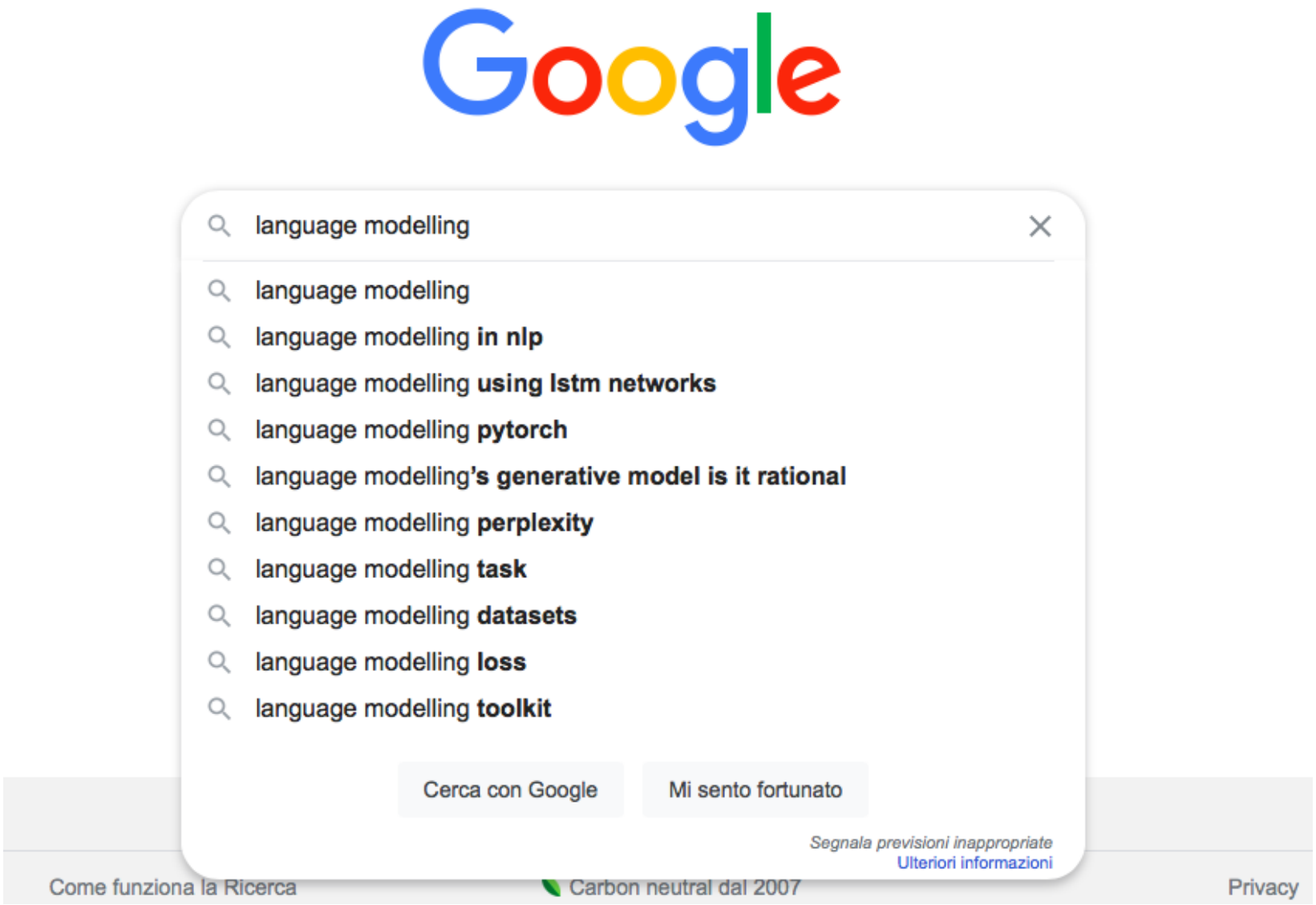
# Natural Language Processing

## Lecture 5 : Language Models

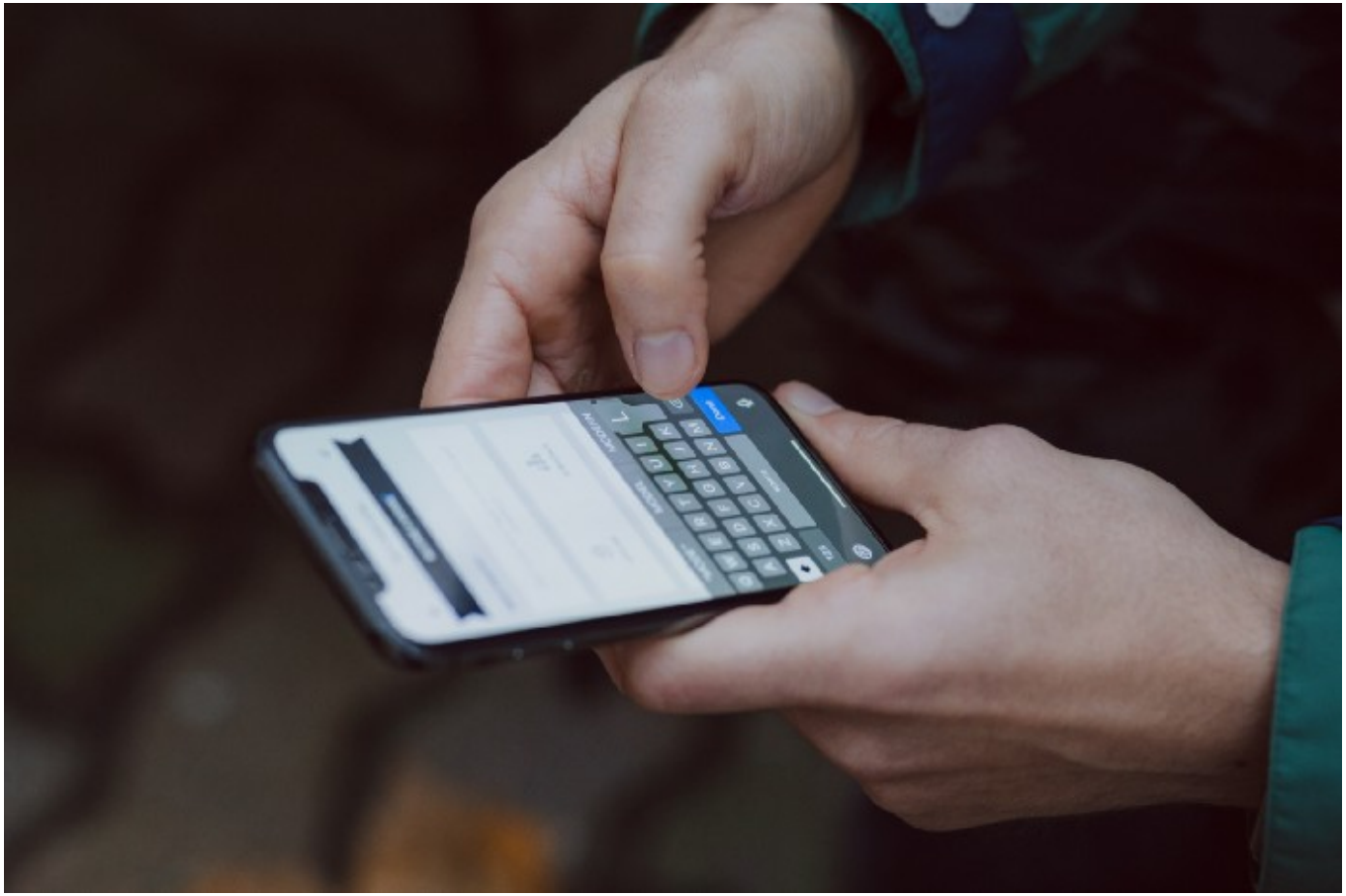
Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Marco Kuhlman, Linköping University
Cristopher Manning, Stanford University
Mark-Jan Nederhof, University of St. Andrews
Elena Voita, University of Edinburgh

# Language modeling



# Language modeling



# Language modeling

**Language modeling** is the task of predicting which word comes next in a sequence of words. More formally, given a sequence of words $w_1 w_2 \cdots w_t$ we want to know the probability of the next word $w_{t+1}$:

$$P(w_{t+1} \mid w_1 w_2 \cdots w_t)$$

This allows language modeling to be treated as a classification task.

**Notation** : When string $w_1 w_2 \cdots w_t$ is understood from the context, we write $w_{1:t}$.

# Language modeling

Rather than as **predictive** models, language models can also be viewed as **generative** models that assign probability to a piece of text:

$$P(w_1 \cdots w_t)$$

Not necessarily a complete sentence.

In this case, the model provides an answer to the question:

*How likely is it that the given text belongs to the modeled language?*

# Language modeling

These two views are **equivalent**, as the probability of a sequence can be expressed as a product of conditional probabilities:

$$P(w_{1:n}) = \prod_{t=1}^{n} P(w_t \mid w_{1:t-1})$$

This is the chain rule. For $t = 1$, we assume $P(w_t \mid w_{1:t-1}) = P(w_1)$.

Conversely, a conditional probability can be expressed as a ratio of two sequence probabilities:

$$P(w_{t+1} \mid w_{1:t}) = \frac{P(w_{1:t+1})}{P(w_{1:t})}$$

We have applied here the definition of conditional probaility.

# Applications

In a machine translation system, we are given the following Chinese sentence

| 他 | 向 | 记者 | 介绍了 | 主要 | 内容 | |
|---|---|---|---|---|---|---|
| *he* | *to* | *reporters* | *introduced* | *main* | *content* | (lit.) |

We get three candidate translations:

- he introduced reporters to the main contents of the statement
- he briefed to reporters the main contents of the statement
- he briefed reporters on the main contents of the statement

We use a language model to select the most likely candidate.

Same idea for speech recognition and spelling correction.

# Probability estimation

©indiamart

# Probability estimation

Assume the text '*its water is so transparent that*'. We want to know the probability that the next word is '*the*'.

Let $C(\textit{its water is so transparent that})$ be the number of times the string is seen in a large corpus.

One way to estimate the above probability is to set

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{C(\textit{its water is so transparent that the})}{C(\textit{its water is so transparent that})}$$

To make these probabilities sum to one, we need to add to the vocabulary $V$ a sentence end marker. In this way, every token in the corpus is *always* followed by some symbol.

# Probability estimation

In general, given a large corpus, we can set:

$$P(w_t \mid w_{1:t-1}) = \frac{C(w_{1:t})}{C(w_{1:t-1})}$$

Under previous assumptions, we have $C(w_{1:t-1}) = \sum_{u \in V} C(w_{1:t-1}u)$.

Quantities $C(w_{1:t})$ are called **frequencies**.

The ratio $C(w_{1:t})/C(w_{1:t-1})$ is called **relative frequency**.

The estimator above is therefore called the **relative frequency estimator**.

# Probability estimation

The relative frequency estimator will be correct in the limit, that is, with infinite data.

In practice, consider an aggressive upper bound of $N = 20$ words in our sequences, and an English vocabulary $V$ of size $|V| = 10^5$. Then the number of possible sequences is $|V|^{20} = 10^{100}$.

This estimator is extremely data-hungry, and suffers from **high variance**: depending on what data happens to be in the corpus, we could get very different probability estimations.

We need to introduce some bias into the model.

# $N$-gram model

©Sunny Srinidhi

# $N$-gram model

A string $w_{t-N+1:t}$ of $N$ words is called **$N$-gram**.

The **$N$-gram model** approximates the probability of a word given the entire sentence history by conditioning only on the past $N-1$ words.

The assumption that the probability of a word depends only on a few previous words is called a Markov assumption.

The 2-gram model, for example, makes the approximation

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-1})$$

1-gram model is just the single word probability $P(w_t)$, unconditioned.

# $N$-gram model

The general equation for the $N$-gram model is

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-N+1:t-1})$$

The relative frequency estimator for the $N$-gram model is then

$$P(w_t \mid w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t})}{C(w_{t-N+1:t-1})}$$

# $N$-gram model

For $N = 2$ we have

$$\begin{aligned} P(w_t \mid w_{t-1}) &= \frac{C(w_{t-1}w_t)}{\sum_u C(w_{t-1}u)} \\ &= \frac{C(w_{t-1}w_t)}{C(w_{t-1})} \end{aligned}$$

For $N = 1$ we have $P(w_t) = \frac{C(w_t)}{n}$ where $n$ is the length of the training set.

Alternatively, we can view $n = \sum_u C(u)$ as the number of 1-gram observations, where $u$ ranges over all tokens excluding the sentence end marker.

The model requires estimating and storing the probability of only $|V|^N$ events, which is exponential in $N$ (a constant), not in the length of the sentence.

# Example

Consider a mini-corpus of three sentences. We augment each sentence with start and end markers <s> and </s>

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

We get the following 2-gram probabilities, among others:

$$P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = .33$$
$$P(\texttt{am}|\texttt{I}) = \frac{2}{3} = .67 \qquad P(\texttt{</s>}|\texttt{Sam}) = \frac{1}{2} = .50$$
$$P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} = .50 \qquad P(\texttt{do}|\texttt{I}) = \frac{1}{3} = .33$$

$N$ is a hyperparameter. When setting its value, we face the **bias-variance tradeoff**.

When $N$ is too small (high bias), it fails to recover long-distance word relations, as for instance in:

> *The* **computer** *that is on the 3rd floor of our office building* **crashed**.

When $N$ is too large, we get data sparsity (high variance).

# Learning

The relative frequency estimator can be mathematically derived by maximizing the **likelihood** of the dataset.

This can be done by solving a **constrained optimization problem**, using Lagrange multipliers.

More precisely, we maximize the log of the product of all sentence probabilities, under the constrain that probabilities sum up to one.

Therefore the relative frequency estimator is also called the **maximum likelihood estimator** (MLE).

# Practical issues

To compute 3-gram probabilities for words at the start of a sentence, we use two **markers**, e.g. $P(\texttt{I}|\texttt{<s><s>})$. Similarly for higher order $N$-grams.

Multiplying many small probabilities results in **underflow**. It is much safer and more efficient to use **negative log probabilities**, $-\log(p)$, and add them.

Note the minus sign: the smaller the probability, the higher the $-$log probability.

For web-scale datasets (large vocabulary), the model has **huge space** requirements, even for small values of $N$.

# Evaluation

# Evaluation

**Extrinsic evaluation** : Use the model in some application (e.g. speech recognition) and measure performance of that application.

Difficult to do reliably, time consuming.

**Intrinsic evaluation** : Look at performance of model in isolation, with respect to a given evaluation measure.

Perplexity, see next slide.

# Perplexity

Intrinsic evaluation of language models is based on the inverse probability of the test set, normalized by the number of words.

For a test set $W = w_1 w_2 \cdots w_n$ we define **perplexity** as:

$$\begin{aligned} \mathrm{PP}(W) &= P(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\prod_{j=1}^{n} \frac{1}{P(w_j \mid w_{1:j-1})}} \end{aligned}$$

We need to discuss inverse probabilities and $n$-th root of product.

# Perplexity

The multiplicative inverse probability $1/P(w_j \mid w_{1:j-1})$ can be seen as a measure of how surprising the next word is.

Equivalently, how much information is carried out by the next word.

The degree of the root averages over all words of the test set, providing **average surprise per word**.

The perplexity of two language models is only comparable if they use identical vocabularies.

For large enough test data obtained in a uniform way, perplexity is more or less constant, i.e. independent of $n$.

The lower the perplexity, the better the model.

# Example

Evaluation of *N*-gram models on the *Wall Street Journal*.

Training set: 38 million tokens;
Vocabulary: 19,979 types;
Test set: 1.5 million words.

| | Unigram | Bigram | Trigram |
|---|---|---|---|
| **Perplexity** | 962 | 170 | 109 |

# Practical issues

Make sure to use a training corpus that has a similar **genre** to whatever task you are trying to accomplish.

An (intrinsic) improvement in perplexity **does not** guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation.

# Sampling sentences

Random sentences generated from 1-gram, 2-gram, 3-gram, and 4-gram models trained on Shakespeare's works.

Corpus: 884,647 tokens, 29,066 types. Sentence generation samples the probabilities of the model.

| | |
|---|---|
| 1 gram | –To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have<br>–Hill he late speaks; or! a more to leg less first you enter |
| 2 gram | –Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.<br>–What means, sir. I confess she? then all sorts, he is trim, captain. |
| 3 gram | –Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.<br>–This shall forbid it should be branded, if renown made it empty. |
| 4 gram | –King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;<br>–It cannot be but so. |

The 3-gram and 4-gram sentences look a lot like Shakespeare. Indeed, for many 4-grams, there is only one continuation.

# Sparse data

$$P(w_t \mid w_{1:t-1}) = \frac{C(w_{1:t})}{C(w_{1:t-1})}$$

If there isn't enough data in the training set, counts will be zero for some grammatical sequences.

Then some of the $N$-gram probabilities will be **zero** or **undefined**.

Perplexity on the test set will also be undefined.

There are three scenarios we need to consider:

- zero numerator: smoothing
- zero denominator: backoff, interpolation
- out-of-vocabulary words in test set: estimation of unknown words

# Smoothing

Susan Wilkinson on Unsplash

# Smoothing

**Smoothing** techniques (also called **discounting**) deal with words that are in our vocabulary $V$ but were never seen before in the given context (zero numerator).

Smoothing prevents LM from assigning **zero probability** to these events.

**Idea** :

- shave off a bit of probability mass from more frequent events
- give it to the events we have never seen in the training set

# Laplace smoothing

**Laplace smoothing** does not perform well enough, but provides a useful baseline.

**Idea** : Pretend that everything occurs once more than the actual count.

In this way, all the counts that used to be 0 will now have a count of 1, the counts of 1 will be 2, and so on.

Also known as add-one smoothing.

# Laplace smoothing

In order to apply smoothing to our $N$-gram model, let us rewrite the relative frequency estimator in a more convenient form:

$$\begin{aligned} P(w_t \mid w_{1:t-1}) &= \frac{C(w_{1:t})}{C(w_{1:t-1})} \\ &= \frac{C(w_{1:t})}{\sum_u C(w_{1:t-1}u)} \end{aligned}$$

In the summation $u$ is a single word ranging over the entire vocabulary $V$ and the sentence end marker.

Recall that every word token is followed by some symbol, either a word or the end marker.

# Laplace for 1-grams

Let $n$ be the number of tokens, that is, the length of the training set, and recall that $|V|$ is the number of word types.

We assume the vocabulary $V$ is fixed.

Recall that the 1-gram model relative frequency estimator is

$$P(w_t) = \frac{C(w_t)}{n}$$

The **adjusted estimate** of the probability of word $w_t \in V$ is then:

$$P_{\mathrm{L}}(w_t) = \frac{C(w_t) + 1}{n + |V|}$$

The extra $|V|$ comes from pretending there are $|V|$ more observations, one for each word type.

# Laplace for 1-grams

Alternatively, we can think of $P_L$ as applying an **adjusted count** $C^*$ to the $n$ actual observations:

$$\begin{aligned} C^*(w_t) &= (C(w_t)+1)\frac{n}{n+|V|} \\ P_L(w_t) &= \frac{C^*(w_t)}{n} \\ &= \frac{C(w_t)+1}{n+|V|} \end{aligned}$$

Under this view, the smoothing algorithm amounts to **discounting** (lowering) counts for high frequency words and redistributing

$$\sum_{u \in V} C(u) = \sum_{u \in V} C^*(u) = n$$

# Laplace for 1-grams

We can consider the **relative discount** $d(w_t)$, defined as the ratio of the discounted counts to the original counts:

$$d(w_t) = \frac{C^*(w_t)}{C(w_t)} = \left(1 + \frac{1}{C(w_t)}\right) \frac{n}{n + |V|}$$

By solving $d(w_t) < 1$, we find that discounting happens for high frequency types $u$ such that $C(u) > \frac{n}{|V|}$.

# Laplace for 2-grams

The 2-gram model relative frequency estimator is

$$\begin{aligned} P(w_t \mid w_{t-1}) &= \frac{C(w_{t-1}w_t)}{\sum_u C(w_{t-1}u)} \\ &= \frac{C(w_{t-1}w_t)}{C(w_{t-1})} \end{aligned}$$

The adjusted estimate of the probability of 2-gram $w_{t-1}w_t$ is then:

$$\begin{aligned} P_{\mathrm{L}}(w_t \mid w_{t-1}) &= \frac{C(w_{t-1}w_t)+1}{\sum_u [C(w_{t-1}u)+1]} \\ &= \frac{C(w_{t-1}w_t)+1}{C(w_{t-1})+|V|} \end{aligned}$$

# Laplace for 2-grams

The adjusted count is:

$$C^*(w_t \mid w_{t-1}) = \frac{[C(w_{t-1}w_t) + 1]C(w_{t-1})}{C(w_{t-1}) + |V|}$$

$P_L(w_t \mid w_{t-1})$ is larger than $P(w_t \mid w_{t-1})$ for 2-gram sequences that occur zero or few times in the training set.

However, $P_L(w_t \mid w_{t-1})$ will be much lower (too low) for 2-gram sequences that occur often. So Laplace smoothing is **too crude** in practice.

# Add-$k$ smoothing

**Add-$k$ smoothing** is a generalization of add-one smoothing.

Also known as Lidstone smoothing

For some $0 \leqslant k < 1$:

$$P_{\text{Add-k}}(w_t \mid w_{t-1}) = \frac{C(w_{t-1}w_t) + k}{C(w_{t-1}) + k|V|}$$

**Jeffreys-Perks law** corresponds to the case $k = 0.5$, which works well in practice and benefits from some theoretical justification.

See for instance (Manning and Schutze, 1999).

# Smoothing and perplexity

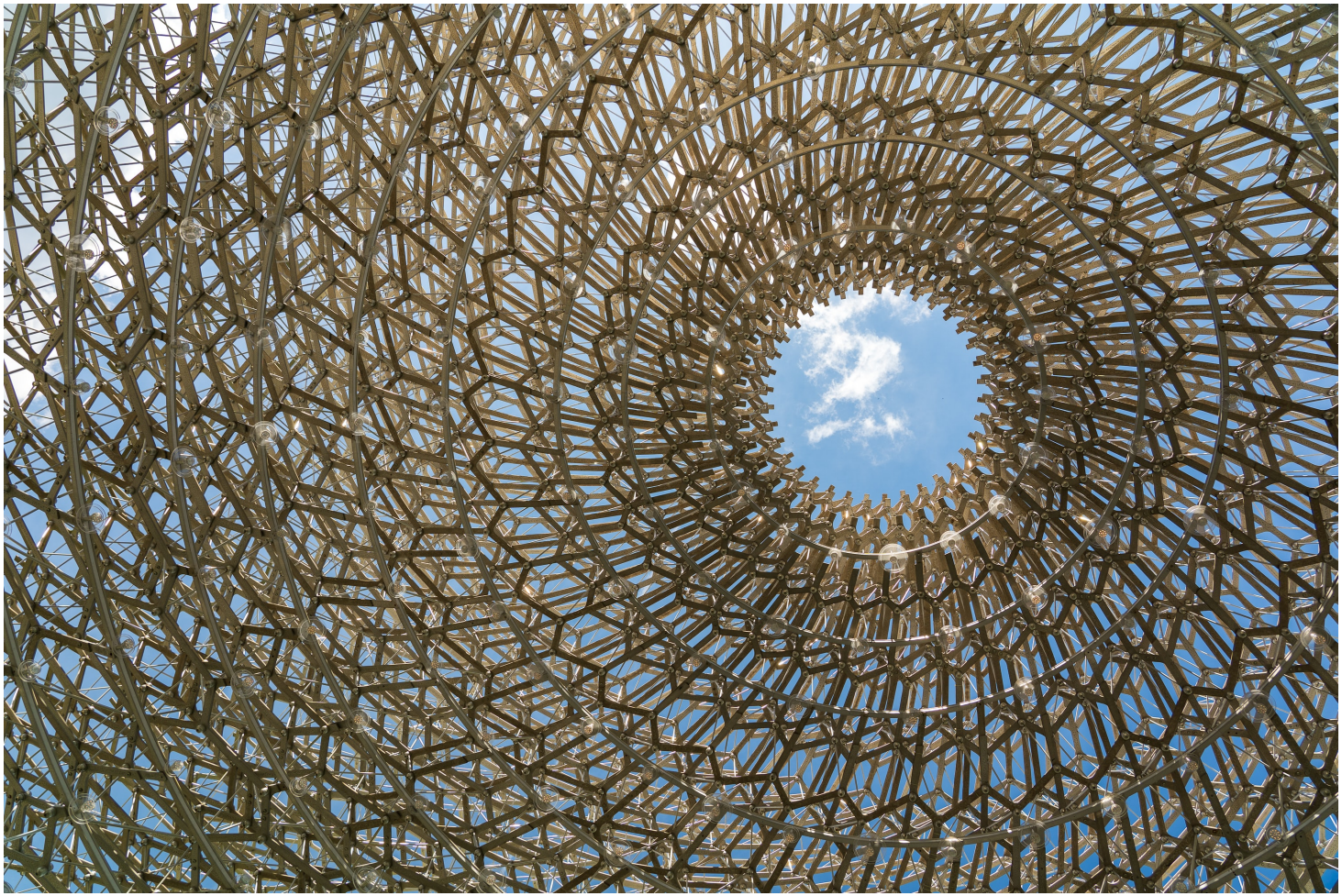
When smoothing a language model, we are redistributing probability mass to outcomes we have never observed.

This leaves a smaller fraction of the probability mass to the outcomes that we actually did observe during training.

The training data becomes less likely.

Thus, the more probability we are taking away from observed outcomes, the **higher** the perplexity on the training data.

# Backoff and interpolation



Nick Fewings on Unsplash

# Backoff and interpolation

Backoff and interpolation techniques deal with words that are in our vocabulary, but in the test set combine to form previously unseen contexts.

These techniques prevent LM from creating **undefined probabilities** for these events (zero-divide).

# Backoff

**Backoff** combines fine grained models (large $N$) with coarse grained models (low $N$).

**Idea** :

- if you have trigrams, use trigrams
- if you don't have trigrams, use bigrams
- if you don't even have bigrams, use unigrams

**Katz backoff** is a popular but rather complex algorithm for backoff.

Katz backoff is often combined with a smoothing method called Good-Turing.

# Stupid backoff

With very large text collections (web-scale) a rough approximation of Katz backoff is often sufficient, called **stupid backoff**.

For some small $\lambda$:

$$P_S(w_t \mid w_{t-N+1:t-1}) = \begin{cases} P(w_t \mid w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t})}{C(w_{t-N+1:t-1})}, \\ \quad \text{if } C(w_{t-N+1:t}) > 0 \\ \lambda P_S(w_t \mid w_{t-N+2:t-1}), \\ \quad \text{otherwise} \end{cases}$$

It is not difficult to show that $P_S$ is **not a probability distribution**. However, in practical settings stupid backoff turns out to be effective.

# Linear interpolation

In **simple linear interpolation**, we combine different order $N$-grams by linearly interpolating all the models.

Simple linear interpolation for $N = 3$

$$\begin{aligned} P_L(w_t \mid w_{t-2}w_{t-1}) &= \lambda_1 P(w_t \mid w_{t-2}w_{t-1}) + \\ &\quad \lambda_2 P(w_t \mid w_{t-1}) + \\ &\quad \lambda_3 P(w_t) \end{aligned}$$

for some choices of positive $\lambda_1$, $\lambda_2$, $\lambda_3$ such that $\sum_j \lambda_j = 1$

Upper-order models set to zero when undefined. Note that we use lower-order models even in cases we have non-zero counts for the upper-order ones.

# Linear interpolation

What are good choices for the $\lambda_j$'s? Algorithms exist that attempt to optimise likelihood of training data.

We might have different values for the $\lambda_j$'s, depending on sequences $w_{t-2}w_{t-1}$, subject to

$$\sum_j \lambda_j(w_{t-2}w_{t-1}) = 1$$

The more frequent $w_{t-2}w_{t-1}$, the more reliable the trigram probabilities, the higher we can choose $\lambda_1(w_{t-2}w_{t-1})$.

# Unknown Words

Annie Spratt on Unsplash

# Unknown Words

Unknown words, also called **out of vocabulary** (OOV) words, are words we haven't seen before.

Replace by new word toker <UNK> all words that occur fewer than $d$ times in the training set, $d$ some small number.

Proceed to train LM as before, treating <UNK> as a regular word.

At test time, replace all unknown words by <UNK> and run the model.

# Limitations



$N$-gram language models have several **limitations**.

Scaling to larger $N$-gram sizes is problematic, both for computational reasons and because of increased sparsity.

Smoothing techniques are intricate and require careful engineering to retain a well-defined probabilistic interpretation.

Without additional effort, $N$-gram models are unable to share statistical strength across similar words.

Observations of 'red apple' do not affect estimates for 'green apple'.

# Research papers

Janko Ferlic on Unsplash

# Research papers

**Title**: An Empirical Study of Smoothing Techniques for Language Modeling
**Authors**: Stanley Chen, Joshua Goodman
**Conference**: ACL 1996
**Content**: Tutorial introduction to *N*-gram models and survey of the most widely-used smoothing algorithms for such models. Extensive empirical comparison of these techniques.

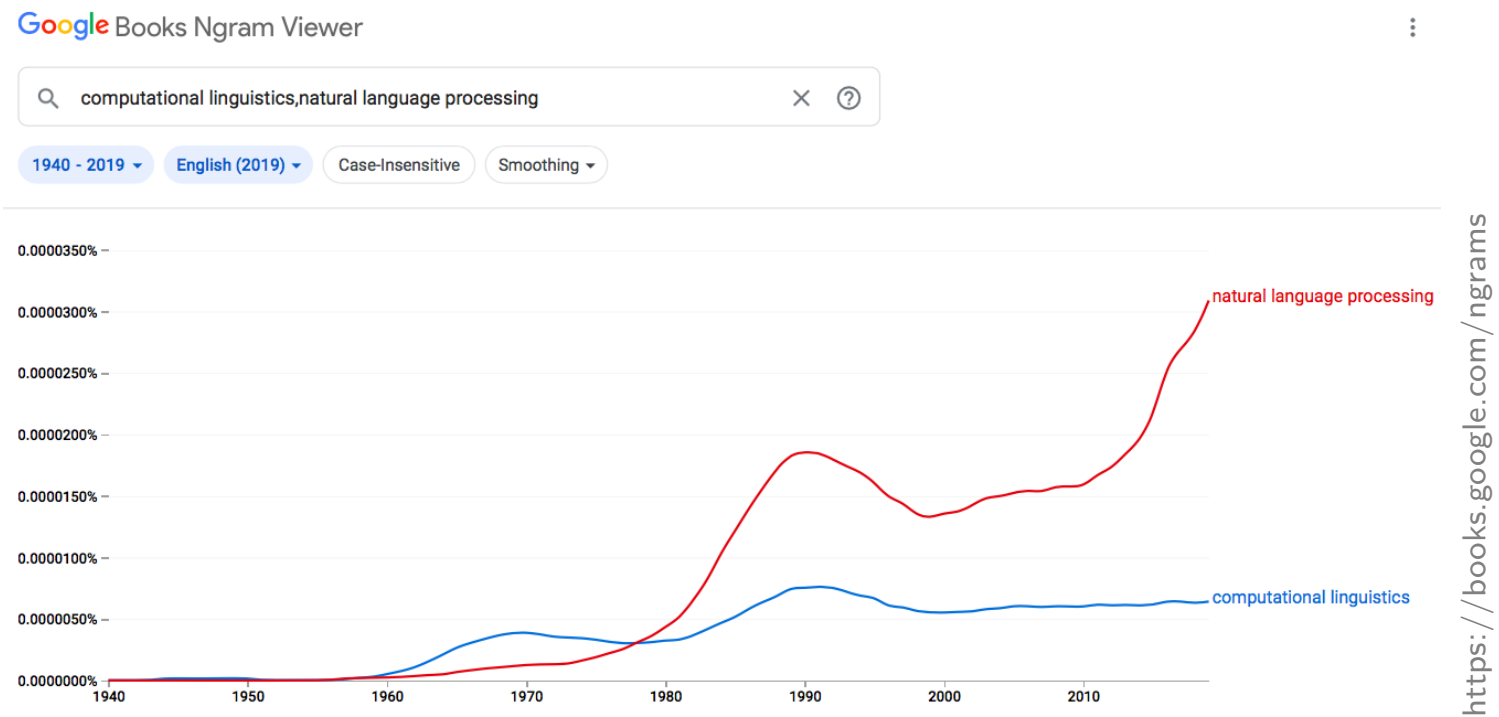
`https://www.aclweb.org/anthology/P96-1041.pdf`

# Research papers

**Title**: Improved backing-off for $M$-gram language modeling
**Authors**: Reinhard Kneser, Hermann Ney
**Conference**: ICASSP 1995
**Content**: Simple back-off smoothing discards context and back off from $n$-grams to $k$-grams with $k < n$. But let's take for example the phrase San Francisco: it is common and Francisco will have a high unigram probability. When backing off, we obtain a large probability of Francisco after any token!

`https://ieeexplore.ieee.org/document/479394?arnumber=479394`

# Have fun!


Google Books Ngram Viewer
computational linguistics,natural language processing
1940 - 2019
English (2019)
Case-Insensitive
Smoothing
0.0000350%
0.0000300%
0.0000250%
0.0000200%
0.0000150%
0.0000100%
0.0000050%
0.0000000%
1940
1950
1960
1970
1980
1990
2000
2010
natural language processing
computational linguistics
https://books.google.com/ngrams

# Neural language models

Omid Armin on Unsplash

# Neural language models

*N*-gram language models have been largely supplanted by **neural language models** (NLM).

Main advantages of NLM

- can incorporate arbitrarily distant contextual information, while remaining computationally and statistically tractable
- can generalize better over contexts of similar words, and are more accurate at word-prediction

On the other hand, as compared with *N*-gram language models, NLM are much more complex, are slower, need more time to train, and are less interpretable.

For many (especially smaller) tasks *N*-gram language models is still the right tool.

# Neural language models

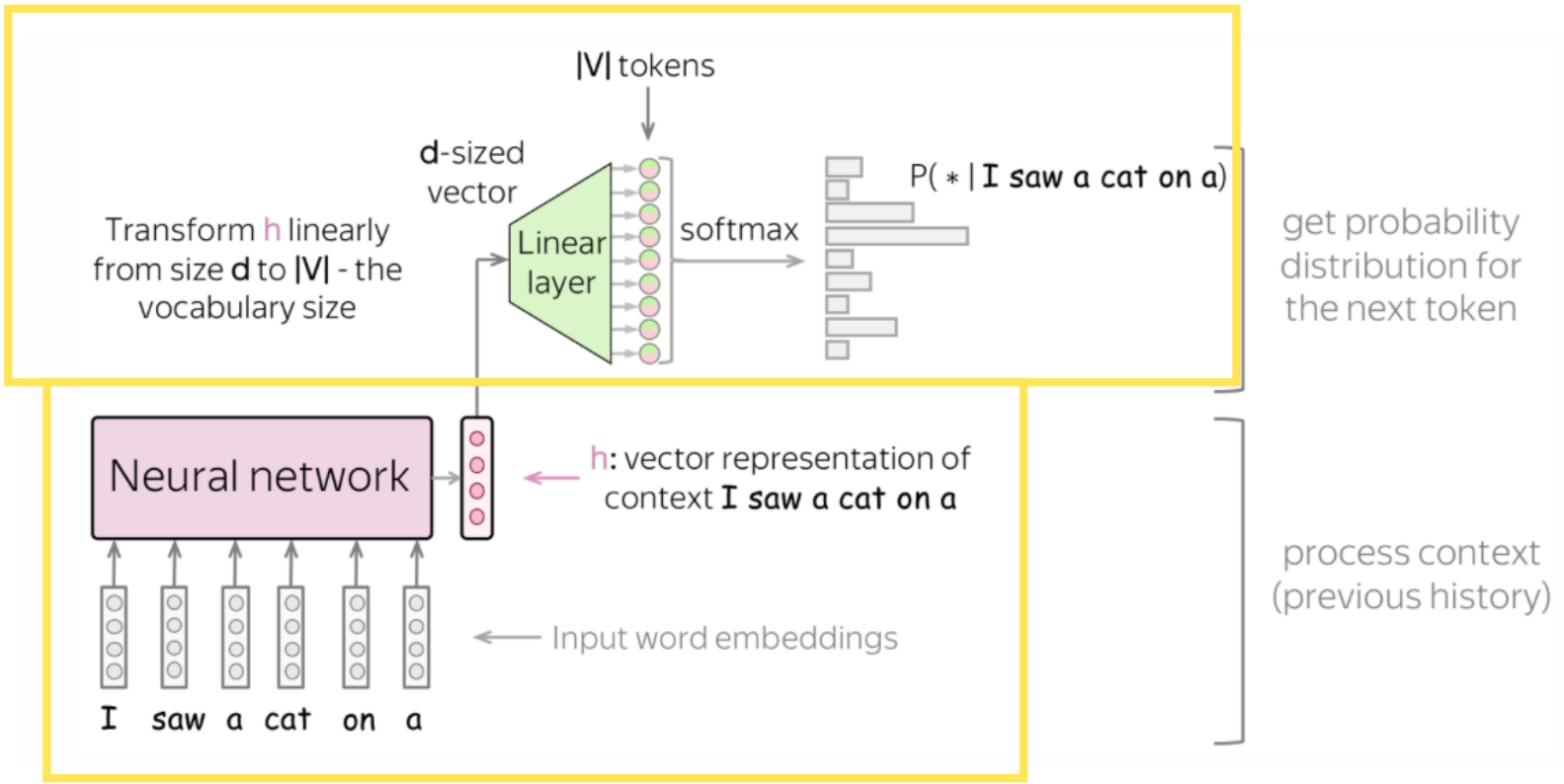
**Idea** :

- get a vector representation for the previous context
- generate a probability distribution for the next token

First bullet depends on NN architecture; second bullet is model-agnostic.

Most natural choice for NN architecture is recurrent neural network (RNN) but feedforward neural network (FNN) and convolutional neural network (CNN) have also been exploited.

## General architecture for NLM




Elena Voita
https://lena-voita.github.io/nlp_course.html

# Feedforward NLM: inference

See Jurafsky & Martin §7.3 and §7.6 for definition and training of feedforward neural networks.

Like the $N$-gram language model, the feedforward NLM uses the following **approximation**

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-N+1:t-1})$$

and a moving window that can see $N-1$ words into the past.

For $w \in V$, let $ind(w) \in [1..|V|]$ be the **index** associated with $w$.

We represent each input word $w_t$ as a **one-hot vector** $\mathbf{x}_t$ of size $|V|$, defined as follows

- element $\mathbf{x}_t[ind(w_t)]$ is set to one
- all the other elements of $\mathbf{x}_t$ are set to zero

# Feedforward NLM: inference

At the first layer

- we convert one-hot vectors for the words in the $N-1$ window into word embeddings of size $d$
- we concatenate the $N-1$ embeddings

The first hidden layer equation is (assuming $N=4$)

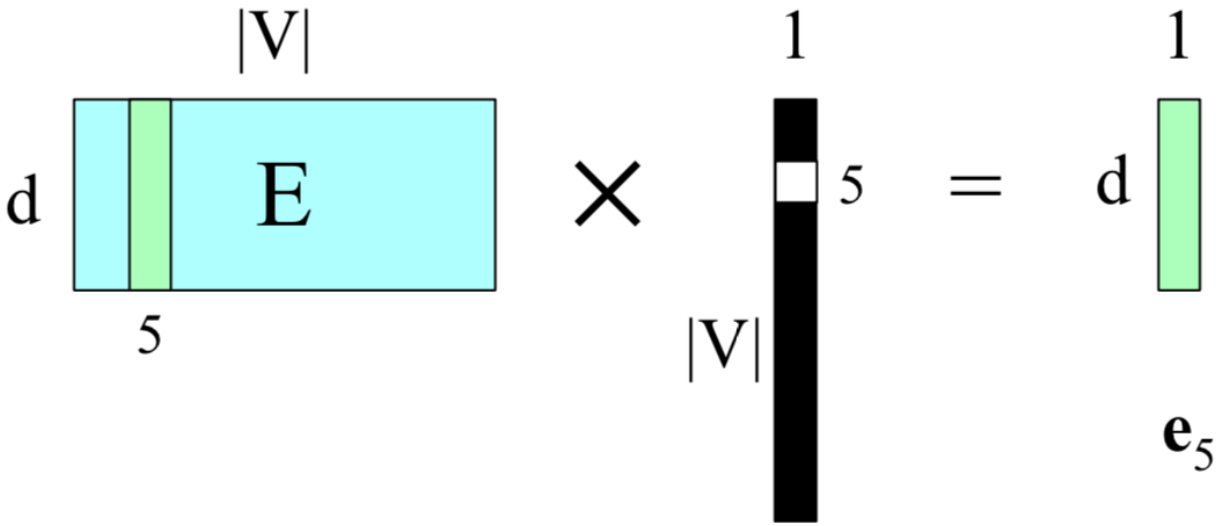
$$\mathbf{e}_t = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$

where

- $\mathbf{E} : d \times |V|$ is a learnable matrix with the word embeddings
- $\mathbf{x}_{t-i} : |V| \times 1$ are 1-hot representation of word $w_{t-i}$
- $\mathbf{e}_t : 3d \times 1$ is the concatenation of the embeddings of the $N-1$ previous words

# Feedforward NLM: inference

Conversion from 1-hot word representation into word embedding.



# Feedforward NLM: inference

The model equations for the remaining layers

$$\begin{aligned} \mathbf{h}_t &= g(\mathbf{W}\mathbf{e}_t + \mathbf{b}) \\ \mathbf{z}_t &= \mathbf{U}\mathbf{h}_t \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{z}_t) \end{aligned}$$

where

- $\mathbf{W} : d_h \times 3d$ is a learnable matrix, $d_h$ the size of the second hidden vector representation
- $\mathbf{b} : d_h \times 1$ is a learnable vector
- $\mathbf{h}_t : d_h \times 1$ is obtained through some activation function $g$
- $\mathbf{U} : |V| \times d_h$ is a learnable matrix
- $\mathbf{z}_t, \hat{\mathbf{y}}_t : |V| \times 1$ scores and distribution (see next slide)

Minor changes to indices wrt the textbook.

# Feedforward NLM: inference

The vector $\mathbf{z}_t$ can be thought of as a set of scores over $V$, also called **logits**: raw (non-normalized) predictions that a classification model generates.

Passing these scores through the **softmax** function normalizes into a probability distribution.

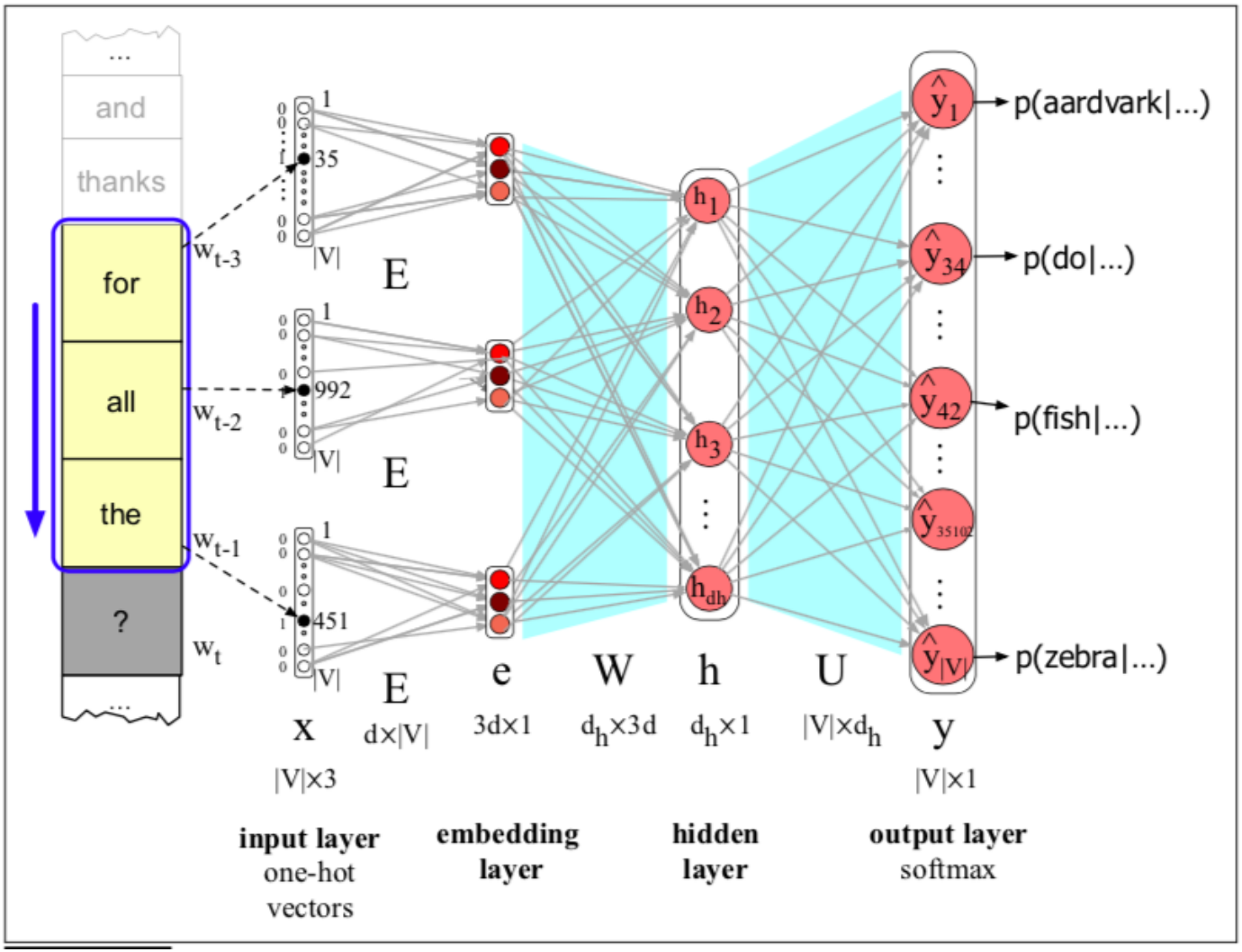
The element of $\hat{\mathbf{y}}_t$ with index $ind(w_t)$ is the probability that the next word is $w_t$:

$$\hat{\mathbf{y}}_t[ind(w_t)] = P(w_t \mid w_{t-3:t-1})$$

Recall we are assuming $N = 4$.

# Feedforward NLM: inference

## Feedforward NLM architecture



# Feedforward NLM: training

The **parameters** of the model are $\theta = \mathbf{E}, \mathbf{W}, \mathbf{U}, \mathbf{b}$.

The number of parameters is $\mathcal{O}(|V|)$, since $d$ is a constant.

Let $w_t$ be the word at position $t$ in the training data. Then the **true distribution** $\mathbf{y}_t$ for the word at position $t$ is a 1-hot vector of size $|V|$ with

- $\mathbf{y}_t[ind(w_t)] = 1$
- $\mathbf{y}_t[k] = 0$ everywhere else

We use the cross-entropy loss for training the model:

$$\begin{aligned} L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) &= -\sum_{k=1}^{|V|} \mathbf{y}_t[k] \log \hat{\mathbf{y}}_t[k] \\ &= -\log \hat{\mathbf{y}}_t[ind(w_t)] \end{aligned}$$

Expected information of $\hat{\mathbf{y}}_t$ computed w.r.t. $\mathbf{y}_t$.

# Feedforward NLM: training

Recall the equation for the estimated distribution (general $N$)

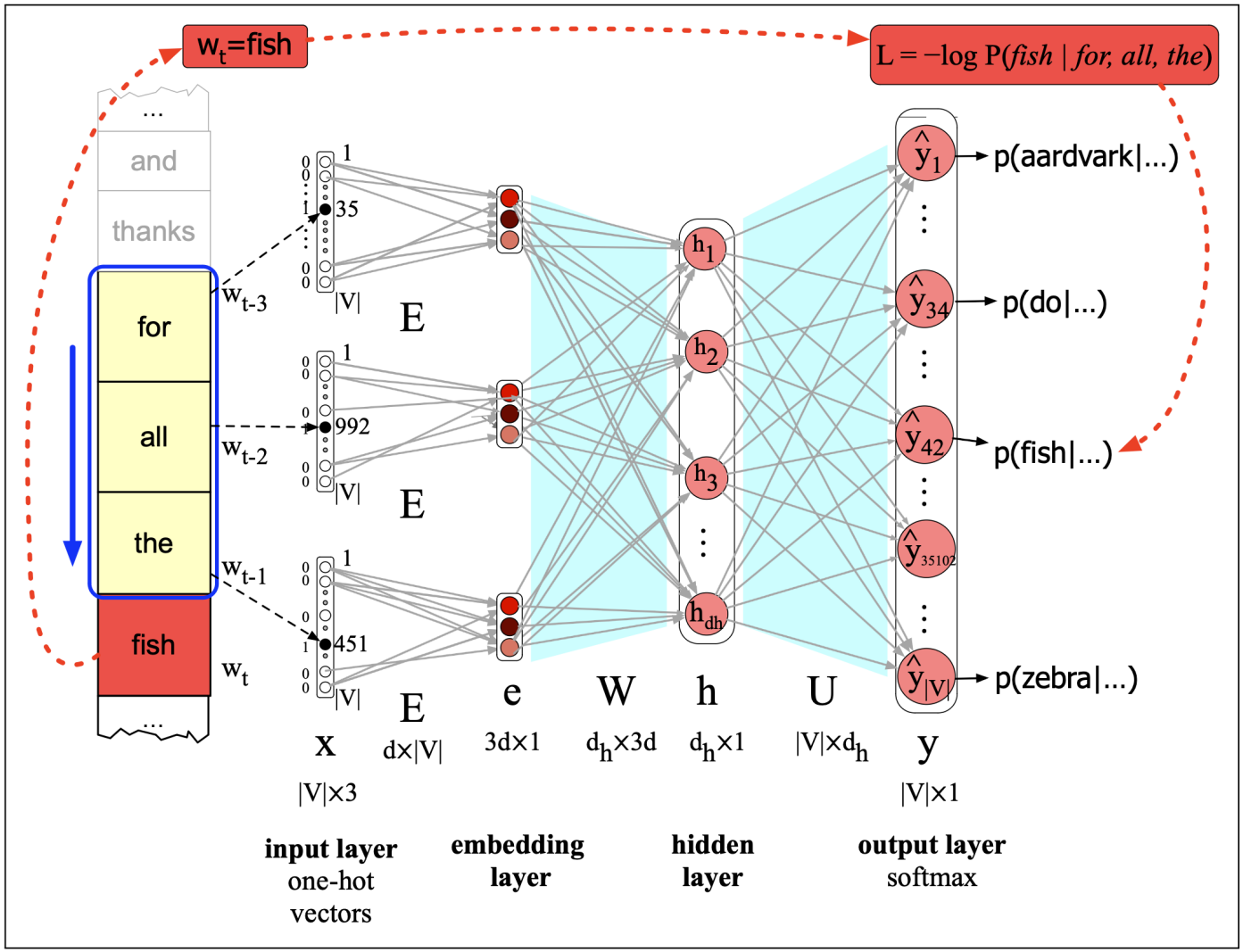
$$\hat{\mathbf{y}}_t[ind(w_t)] \quad = \quad P(w_t \mid w_{t-N+1:t-1})$$

Replacing in the cross-entropy loss equation, we obtain

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) \quad = \quad -\log P(w_t \mid w_{t-N+1:t-1})$$

Observe that the cross-entropy loss equals the **negative log likelihood** of the training data (see later slides).

## Feedforward NLM training



# Recurrent NLM: inference

See Jurafsky & Martin §9.1 for definition and training of recurrent neural networks.

RNN language models process the input one word at a time, predicting the next word from

- the current word
- the previous **hidden state**

RNNs can model probability distribution $P(w_t \mid w_{1:t-1})$ without the $N-1$ window approximation of feedforward NLM.

The hidden state of the RNN model can (in principle) represent information about all of the preceding words.

# Recurrent NLM: inference

The model equations are (for some $\mathbf{h}_0$)

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

where ($d_h$ is the size of the hidden vectors)

- $\mathbf{x}_t : |V| \times 1$ is a 1-hot representation of word $w_t$
- $\mathbf{E} : d_h \times |V|$ is a learnable matrix with the word embeddings
- $\mathbf{U}, \mathbf{W} : d_h \times d_h$ are learnable matrices
- $\mathbf{h}_t : d_h \times 1$ is the hidden vector at step $t$
- $\mathbf{V} : |V| \times d_h$ is a learnable matrix
- $\hat{\mathbf{y}}_t : |V| \times 1$ is a probability distribution

Minor changes to notation wrt the textbook.

# Recurrent NLM: inference

The vector resulting from $\mathbf{V}\mathbf{h}_t$ records the **logits** (unormalized scores) over the vocabulary $V$, given the evidence provided by $\mathbf{h}_t$.

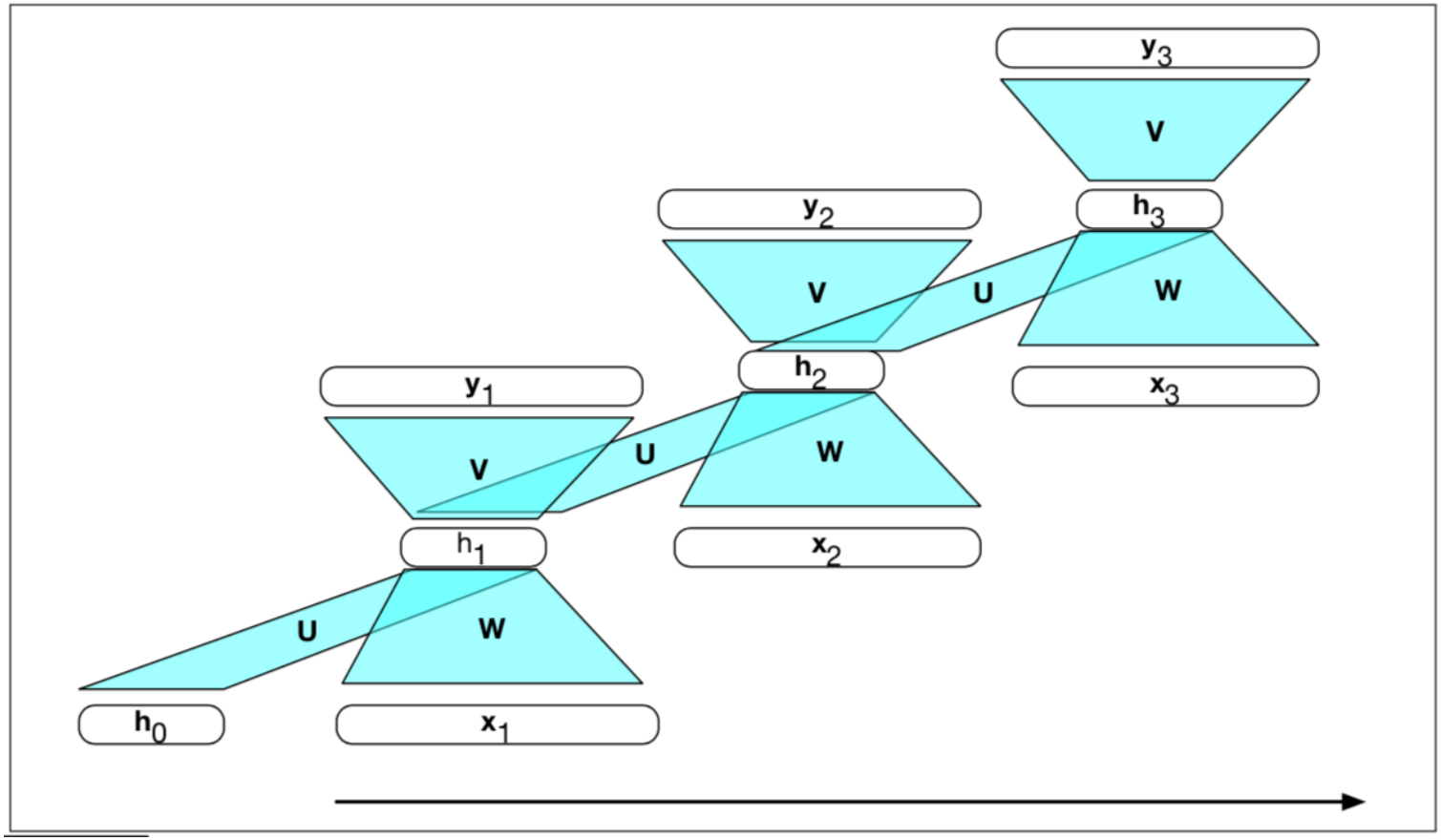
The softmax normalizes the logits, resulting in the **estimated distribution** $\hat{\mathbf{y}}_t$ for the word at step $t$.

More precisely, for each word $w \in V$, the element of $\hat{\mathbf{y}}_t$ with index $ind(w)$ estimates the probability that the next word is $w$:

$$\hat{\mathbf{y}}_t[ind(w)] \quad = \quad P(w_{t+1} = w \mid w_{1:t})$$

# Recurrent NLM: inference

The recurrent NLM unrolled in time



# Recurrent NLM: training

The number of parameters of the model is $\mathcal{O}(|V|)$, since $d$ is a constant.

Let $\mathbf{y}_t$ be the **true distribution** at step $t$. This is a 1-hot vector over $V$, obtained from the training set.

We train the model to **minimize** the error in predicting the true next word $w_{t+1}$ in the training sequence, using cross-entropy as the loss function:

$$\begin{aligned} L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) &= -\sum_{w \in V} \mathbf{y}_t[ind(w)] \log \hat{\mathbf{y}}_t[ind(w)] \\ &= -\log \hat{\mathbf{y}}_t[ind(w_{t+1})] \end{aligned}$$

Again, the cross-entropy loss equals the negative log likelihood of training set.

# Recurrent NLM: training

At each step $t$ during training

- prediction is based on the correct sequence of tokens $w_{1:t}$
- we ignore what the model predicted at previous steps

The idea that we always give the model the correct history sequence to predict the next word is called **teacher forcing**.

Teacher forcing has some disadvantages: the model is never exposed to prediction mistakes; therefore at inference time the model is not able to recover from errors.

# Character-level NLM

**Character-level NLM**

- improves modeling of uncommon and unknown words
- reduces training parameters due to the small softmax

Performance usually worse than the word-level NLMs, since longer history is needed to predict the next word correctly.

A variety of solutions that combine character- and word- level information have been proposed, called **character-aware LM**.

# Practical issues

# Practical issues

Both the feedforward NLM and the recurrent NLM learn word embeddings **E** simultaneously with training the network.

This is useful when the task at hand places strong constraints on word representation, e.g. in sentiment analysis.

Alternatively, one can resort to **freezing**:

- use pretrained word embeddings, for instance word2vec
- hold **E** constant while training, and modify the remaining parameters in $\theta$

# Practical issues

In the recurrent NLM model

- the columns of $\mathbf{E}$ provide the learned word embeddings
- the rows of $\mathbf{V}$ provide a second set of learned word embeddings, that capture relevant aspects of word meaning and function

**Weight tying**, also known as **parameter sharing**, means that we impose $\mathbf{E}^\top = \mathbf{V}$.

Weight tying can significantly reduce model size, and has an effect similar to **regularization**, preventing overfitting of the NLM.

# Practical issues

RNNs suffer from the **vanishing gradient** problem: past events have weights that decrease exponentially with the distance from actual word $w_t$.

Gated recurrent units (GRU) and long-short term memory (LSTM) neural networks are **much better** in capturing long distance relations.

# Practical issues

The last step in NLMs, involving softmax over the entire vocabulary, dominates the computation both at training and at test time.

Vocabulary usually contains 10K to 100K words.

An effective alternative is **hierarchical softmax**, based on word clustering.

**Adaptive softmax** is a simple variant of hierarchical softmax, based on Zipf's law, especially tailored for GPUs.

This provides giant speedup with only minimal costs in accuracy.

The text generated by sampling our NLM should be

- **coherent**: text has to make sense
- **diverse**: the model has to be able to produce very different samples

There is a trade-off between the two.

A very popular method for modifying language model behavior is to change the **softmax temperature** $\tau$

$$\frac{\exp(\frac{\mathbf{V}_i \mathbf{h}_t}{\tau})}{\sum_j \exp(\frac{\mathbf{V}_j \mathbf{h}_t}{\tau})}$$

where $\mathbf{V}_i$ denotes the $i$-th row of $\mathbf{V}$.

Low $\tau$ produces peaky distribution (high coherence); large $\tau$ produces flat distribution (high diversity).

**Contrastive evaluation** is used to test specific linguistic constructions in NLM.

**Example** : Subj/Obj agreement, compare $P(\text{is} \mid w_{1:t-1})$ to $P(\text{are} \mid w_{1:t-1})$

- The **roses** ___
- The **roses** in the vase ___
- The **roses** in the vase by the door ___

# Research papers


Henry Be on Unsplash

# Research papers

**Title**: A Neural Probabilistic Language Model
**Authors**: Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin
**Journal**: Journal of Machine Learning Research 3 (2003) 1137-1155
**Content**: The authors attack the language modeling problem by learning a distributed representation for words, which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences.

https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

# Research papers

**Title**: A Survey on Neural Network Language Models
**Authors**: Kun Jing, Jungang Xu
**Repository**: ArXiv
**Content**: Neural Network Language Models (NNLMs) overcome the curse of dimensionality and improve the performance of traditional LMs. A survey on NNLMs is performed in this paper.

https://arxiv.org/abs/1906.03591

# Review: Machine learning

Chris Henry on Unsplash

# Review: Machine learning

Let $W = w_1 w_2 \cdots w_n = w_{1:n}$ be the training set. We now focus on step $t$.

NLMs use probability distribution $P_t(X) = P(X \mid w_{1:t-1})$.

Here $X$ is a stochastic variable assuming values in $V$.

The target distribution we want to learn is $P_t^*(X) = \text{one-hot}(w_t)$.

This distribution assignings probability 1 to $w_t$.

Our loss function is the **cross-entropy** of $P_t^*$ and $P_t$:

$$Loss(P_t^*, P_t) = \mathbb{E}_{P_t^*}[-\log(P_t)] = -\sum_{w \in V} P_t^*(w) \log(P_t(w))$$

Cross-entropy is minimal when the two distributions are equal. In the special case of $P_t^*$ (deterministic), cross-entropy equals the Kullback-Leibler divergence.

# Review: Machine learning

Since $P_t^*$ is non-zero only for the correct token $w_t$, we have

$$\begin{aligned} Loss(P_t^*, P_t) &= -\sum_{w \in V} P_t^*(w) \log(P_t(w)) \\ &= -\log(P_t(w_t)) \\ &= -\log(P(w_t \mid w_{1:t-1})) \end{aligned}$$

For the whole training set $W$ the loss will be

$$\begin{aligned} -\sum_{t=1}^{n} \log(P(w_t \mid w_{1:t-1})) &= -\log(\prod_{t=1}^{n} P(w_t \mid w_{1:t-1})) \\ &= -\log(P(w_{1:t})) \end{aligned}$$

which is the inverse of the **log-likelihood** of the training set $W$.

This connects to the maximum likelihood estimator used for $N$-gram models.

# Review: Machine learning

We can also compare with the **perplexity** measure, used to evaluate the $N$-gram LM

$$\begin{aligned}
\log(\text{PP}(W)) &= \log(P(w_{1:n})^{-\frac{1}{n}}) \\
&= -\frac{1}{n}\log\left(\prod_{t=1}^{n} P(w_t \mid w_{1:t-1})\right) \\
&= -\frac{1}{n}\sum_{t=1}^{n} \log(P(w_t \mid w_{1:t-1}))
\end{aligned}$$

which is the (additive) inverse of the log-likelihood, averaged over the size of $W$.

So in $N$-gram LM we are using same objective function for training and testing.