Yandex

UNIVERSITEIT VAN AMSTERDAM

Language models

SEPTEMBER 27, 2018

Elena Voita
Yandex Research,
University of Amsterdam
lena-voita@yandex-team.ru

Plan

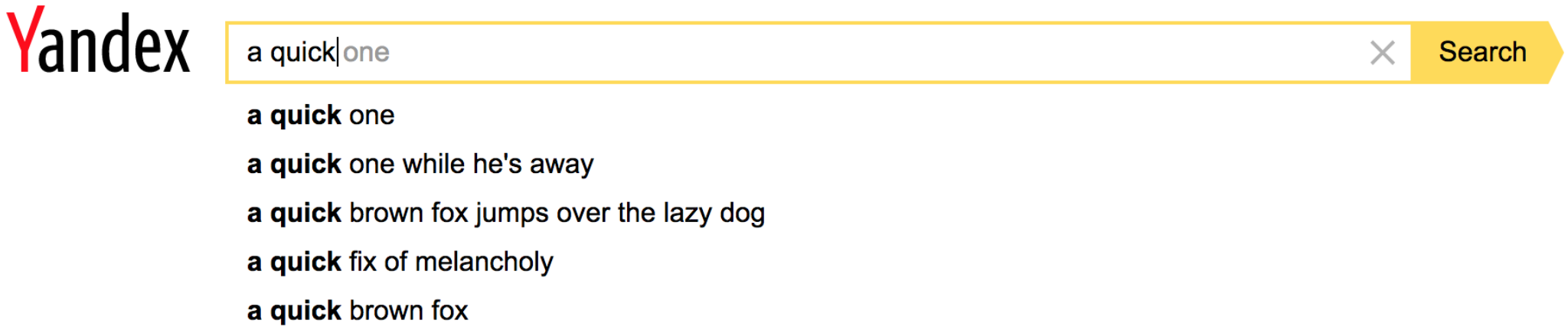
- What is language modeling?
- N-gram Language Models
- Evaluation of LMs
- Neural LMs
- Other versions of LMs
- Training LMs to get embeddings
- Hack of the day!

Language modeling

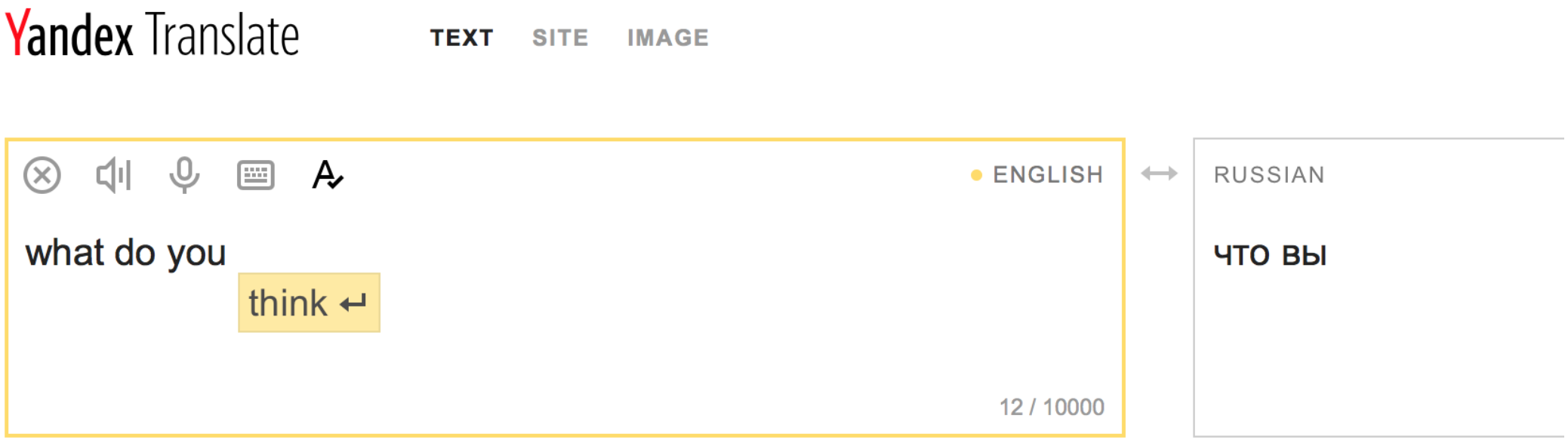
The goal of language modelling is to **estimate the probability** distribution of various linguistic units:

- symbols
- words
- any token sequences

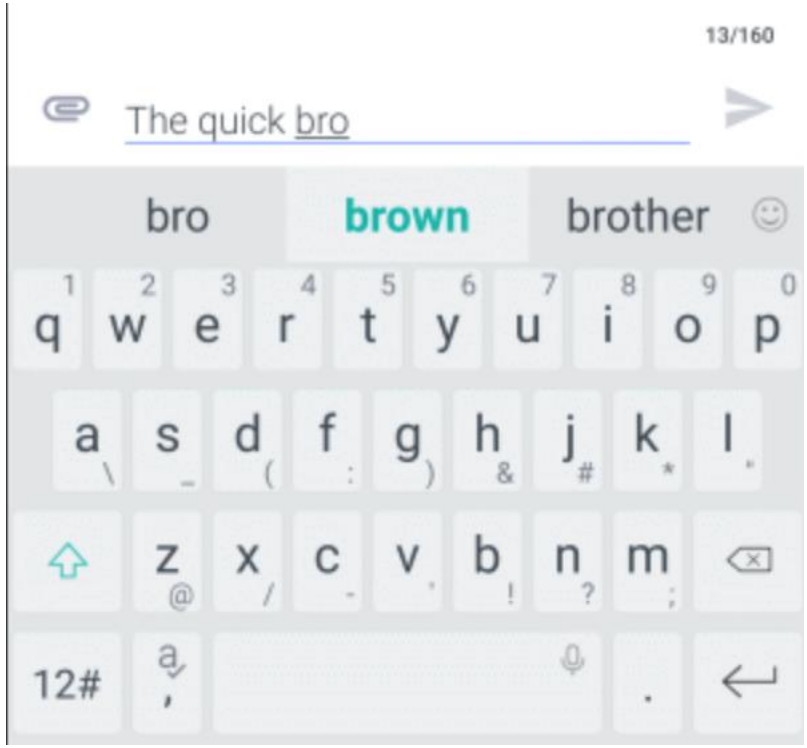
Where could we see LMs?



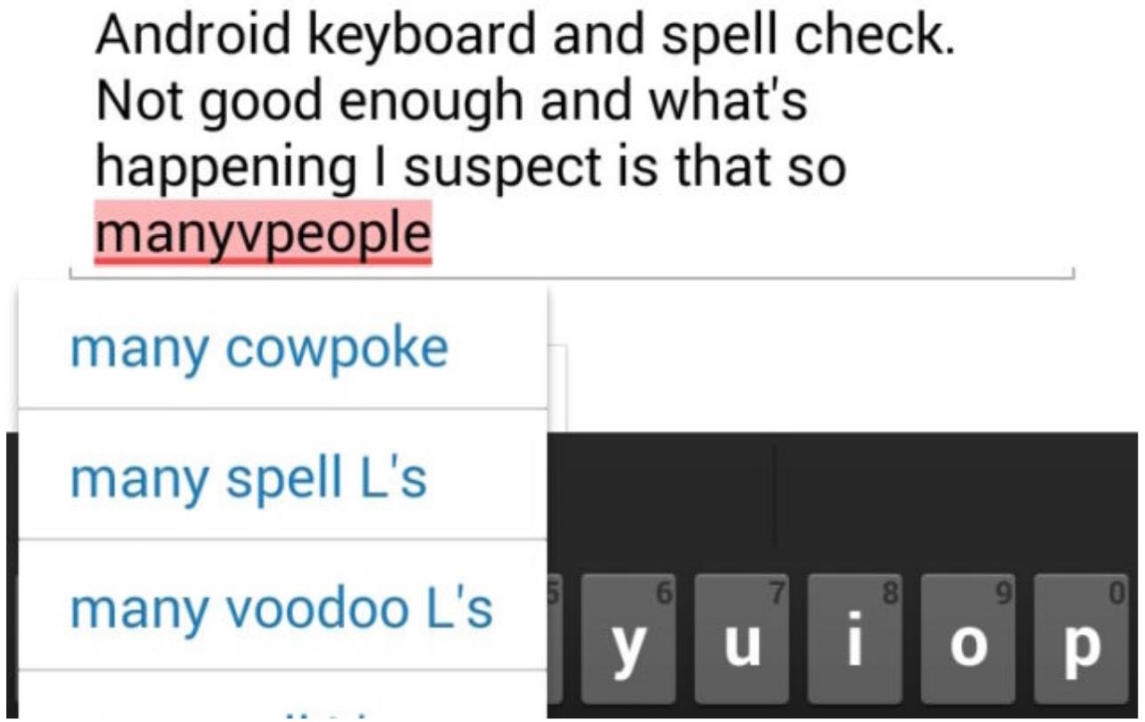
Where could we see LMs?



Where could we see LMs?



Where could we see LMs?



Actually, we use our own (human) language model every day

We all are able to decide between similar-sounding options:

She studies morphosyntax
She studies more faux syntax
She's studies morph or syntax
...

She studies morphosyntax

But for automatic models, **sentence probabilities** help to decide.

Sentence probabilities: intuitive interpretation

"**Probability of a sentence**" = how likely is it to occur in natural language

- Consider only a specific language (English)

$$\mathrm{P}(\text{the cat slept peacefully}) > \mathrm{P}(\text{slept the peacefully cat})$$

$$\mathrm{P}(\text{she studies morphosyntax}) > \mathrm{P}(\text{she studies more faux syntax})$$

Language models in NLP

- It's very difficult to know the true probability of an arbitrary sequence of words.
- But we can define a **language model** that will give us good approximations.
- Like all models, language models will be good at capturing some things and less good for others.

Language Models

Count-based (statistical)

Neural Language Models

N-gram Language Models

(AKA COUNT-BASED LANGUAGE MODELS)

How to estimate probabilities? Task 0: M&M colors

What is the proportion of each color of M&M?

- In 48 packages, I find[1] 2620 M&Ms, as follows:

Red	Orange	Yellow	Green	Blue	Brown
372	544	369	483	481	371

- How to estimate probability of each color from this data?

$$P(color) = \frac{C(color)}{N}, \qquad N = \sum_{color} C(color)$$

[1]Actually, data from: https://joshmadison.com/2007/12/02/mms-color-distribution-analysis/

How to estimate probabilities? Task: sentences

- We want to know the probability of word sequence $s = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$ occurring in English.

- Assume we have some training data: large corpus of general English text.

- We want to use this data to estimate the probability of s (even if we never see it in the corpus!)

Sentences that have never occurred

the Archaeopteryx soared jaggedly amidst foliage

vs

jaggedly trees the on flew

Sentences that have never occurred

the Archaeopteryx soared jaggedly amidst foliage

vs

jaggedly trees the on flew

- First is meaningful, second – not
- C(s) = 0 for both sentences

Sentences that have never occurred

the Archaeopteryx soared jaggedly amidst foliage

VS

jaggedly trees the on flew

- First is meaningful, second – not
- C(s) = 0 for both sentences

Previous (MLE) approach does not work on full sentences

Sparse data problem

- Problem: not enough observations to estimate probabilities well. (Unlike the M&Ms, where we had large counts for all colors!)

- For sentences, many (most!) will occur rarely if ever in our training data. So we need to do something smarter.

How to fix the problem?

Idea:

estimate P(s) by combining the probabilities of smaller parts of the sentence, which will occur more frequently.

What we will get:

N-gram Language Model!

Deriving an N-gram Model

- We want to estimate $P(s = w_1 \dots w_n)$.
  Example: $P(s = the\ cat\ slept\ quietly)$

- This is really a joint probability over the words in s:

$$P(w_1 = the, w_2 = cat, w_3 = slept, w_4 = quietly)$$

- Recall that for a joint probability, $P(X, Y) = P(Y|X)P(X)$. So,

$$P(the\ cat\ slept\ quietly) = P(quietly|the\ cat\ slept) \cdot P(the\ cat\ slept) = P(quietly|the\ cat\ slept) \cdot P(slept \mid the\ cat) \cdot P(cat|the) \cdot P(the)$$

Deriving an N-gram Model

- The chain rule gives us:

$$\boldsymbol{P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1 \ldots w_{i-1})}$$

- But many of these conditional probs are just as sparse!

If we want $P(I\ spent\ three\ years\ before\ the\ mast\ \ldots)$

we still need $P(mast \mid I\ spent\ three\ yers\ before\ the)$

Deriving an N-gram Model

- We make an **independence assumption**: the probability of a word only depends on a fixed number of previous words (**history**).
Namely, we assume that **Markov property** holds.

Markov property:

$$P(w_i|w_1, w_2, \dots, w_{i-1}) = P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

– trigram model: $P(w_i|w_0, w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$
– bigram model: $P(w_i|w_0, w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-1})$
– unigram model: $P(w_i|w_0, w_1, \dots, w_{i-1}) \approx P(w_i)$

N-gram Language Models

(n-1) words

$$P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-n+1}, \dots, w_{i-1}) =$$

(our assumption)

prob of an n-gram

prob of an (n-1)-gram

$$= \frac{P(w_i, w_{i-1}, \dots, w_{i-n+1})}{P(w_{i-1}, \dots, w_{i-n+1})}$$

(definition of a conditional probability)

Question: How do we get these *n*-gram and (*n*-1)-gram probabilities?
Answer: By counting them in some large corpus of text!

$$\approx \frac{count(w_i, w_{i-1}, \dots, w_{i-n+1})}{count(w_{i-1}, \dots, w_{i-n+1})}$$

(statistical approximation)

Example: 4-gram model

The cat slept quietly on the _____

Example: 4-gram model

~~The cat slept~~ quietly on the _____

discard

condition on this

Example: 4-gram model

~~The cat slept~~ quietly on the _____

discard

condition on this

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Problems with n-gram Language Models

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Problem. What if "*quietly on the*" never occurred in the data? Then we can't calculate probability for any w_i!

Problems with n-gram Language Models

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Problem. What if "*quietly on the*" never occurred in the data? Then we can't calculate probability for any w_i!

(Partial) solution. Backoff smoothing

Backoff (aka ”stupid backoff”)

Sometimes it helps to use **less** context

- Condition on less context for contexts you haven’t learned much about

Backoff:

- use trigram if you have good evidence,
- otherwise bigram,
- otherwise unigram

“*quietly on the*” is not seen → try “*on the*”

$$P(w_i|quietly\ on\ the) \approx P(w_i|on\ the)$$

“*on the*” is not seen → try ”*the*”

$$P(w_i|on\ the) \approx P(w_i|the)$$

“*the*” is not seen → try unigram

$$P(w_i|the) \approx P(w_i)$$

More clever: Linear Interpolation

Interpolation: mix unigram, bigram, trigram, …

$$\begin{aligned}\hat{P}(w_i|w_{i-1},w_{i-2},w_{i-3}) \approx\ & \lambda_3 P(w_i|w_{i-1},w_{i-2},w_{i-3}) + \\ & + \lambda_2 P(w_i|w_{i-1},w_{i-2}) \\ & + \lambda_1 P(w_i|w_{i-1}) \\ & + \lambda_0 P(w_i)\end{aligned}$$

$$\sum_{k=0}^{n-1} \lambda_k = 1$$

Question: How to pick λ_k?

More clever: Linear Interpolation

Interpolation: mix unigram, bigram, trigram, …

$$\begin{aligned}\hat{P}(w_i|w_{i-1},w_{i-2},w_{i-3}) \approx\ & \lambda_3 P(w_i|w_{i-1},w_{i-2},w_{i-3}) + \\ & + \lambda_2 P(w_i|w_{i-1},w_{i-2}) \\ & + \lambda_1 P(w_i|w_{i-1}) \\ & + \lambda_0 P(w_i)\end{aligned}$$

$$\sum_{k=0}^{n-1} \lambda_k = 1$$

Question: How to pick λ_k?

Answer: Use your validation set!

Problems with n-gram Language Models

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Problem. What if "*quietly on the*" never occurred in the data? Then we can't calculate probability for any w_i!

(Partial) solution. Backoff smoothing

Problems with n-gram Language Models

Problem. What if "*quietly on the* w_i" never occurred in the data? Then w_i has probability 0

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Problems with n-gram Language Models

Problem. What if "*quietly on the* w_i" never occurred in the data? Then w_i has probability 0

(Partial) solution. Smoothing (add-one, Kneser-Ney, etc.)

$$P(w_i|the\ cat\ slept\ quietly\ on\ the) \approx \frac{count(quietly\ on\ the\ w_i)}{count(quietly\ on\ the)}$$

Laplace-smoothing (aka add-one smoothing)

- Pretend we saw each word one more time than we did
- Just add one to all the counts!

If 1 if too rude, add small δ to count for every $w_i \in V$.

$$\hat{P}(w_i|w_{i-1}, \ldots, w_{i-n+1}) = \frac{\delta + P(w_i, w_{i-1}, \ldots, w_{i-n+1})}{\delta \cdot |V| + P(w_{i-1}, \ldots, w_{i-n+1})}$$

Kneser-Ney Smoothing

- Introduce a clever way of constructing the lower-order (backoff) model.
- Idea: the lower-order model is significant only when count is small or zero in the higher-order model, and so should be optimized for that purpose.
- Example: suppose “San Francisco” is common, but “Francisco” occurs only after “San”.
- “Francisco” will get a high unigram probability, and so absolute discounting will give a high probability to “Francisco” appearing after novel bigram histories.
- Better to give “Francisco” a low unigram probability, because the only time it occurs is after “San”, in which case the bigram model fits well.

Kneser-Ney Smoothing

- Let the count assigned to each unigram be the number of different words that it follows. Define:

$$N_{1+}(\bullet\ w_i) = |\{w_{i-1} : c(w_{i-1}w_i) > 0\}|$$

$$N_{1+}(\bullet\ \bullet) = \sum_{w_i} N_{1+}(\bullet\ w_i)$$

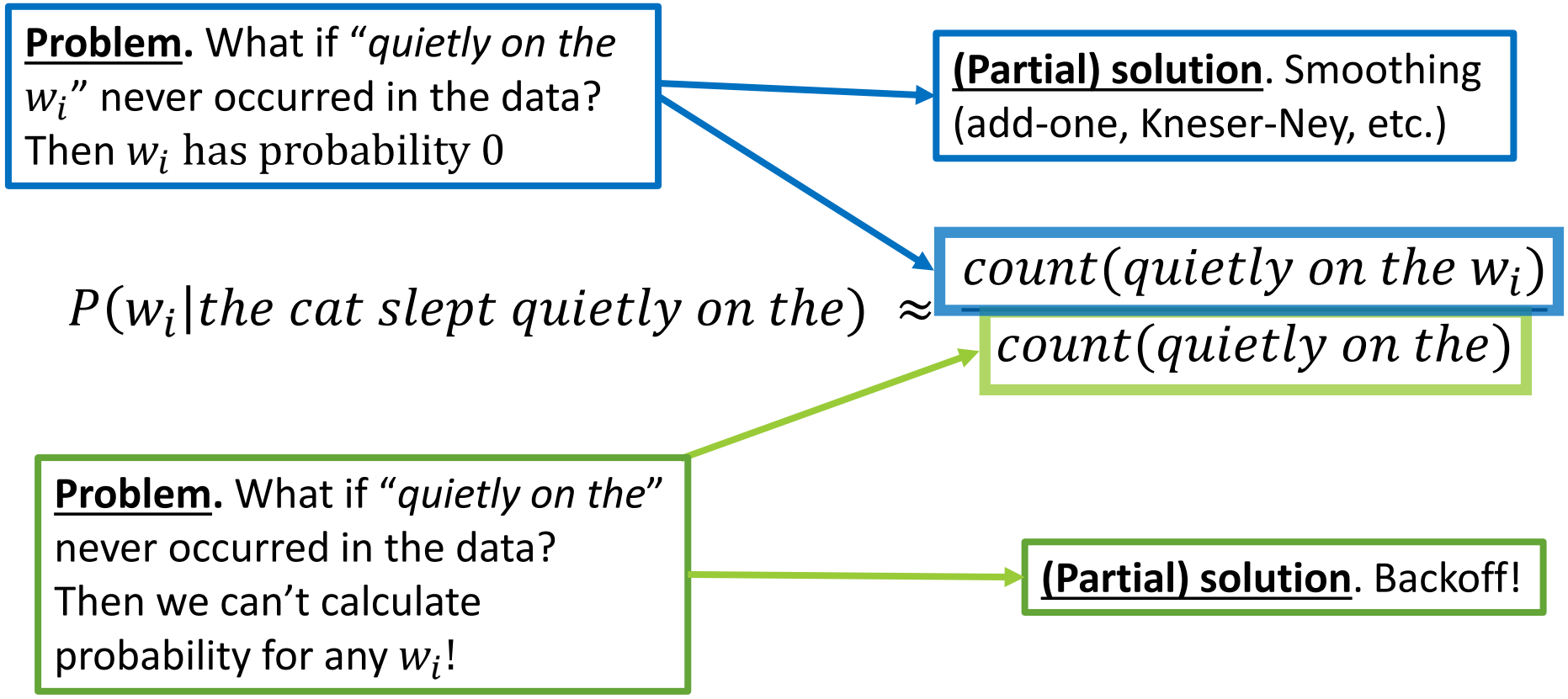
- Let lower-order distribution be:

$$p_{KN}(w_i) = \frac{N_{1+}(\bullet\ w_i)}{N_{1+}(\bullet\ \bullet)}$$

- Put it all together:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - \delta, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + \frac{\delta}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\ \bullet) p_{KN}(w_i|w_{i-n+2}^{i-1})$$

Problems with n-gram Language Models



How to evaluate Language Models

Intrinsic: evaluation on a specific/intermediate subtask

- perplexity/cross-entropy

Extrinsic: evaluation on a real task

- plug your LM it into a SMT/ASR/etc system.
- train the system with different LMs
- If the result with your LM is better -> win!

Cross-entropy and Perplexity

- For $(w_1 w_2 \ldots w_n)$ with large *n*, per-word cross-entropy is well approximated by:

$$H_M(w_1 w_2 \ldots w_n) = -\frac{1}{n} \cdot \log P_M(w_1 w_2 \ldots w_n)$$

- Lower cross-entropy ⇒ model is better at predicting next word.

- Perplexity (reported in papers):

$$\text{perplexity} = 2^{cross-entropy}$$

Language Models

Count-based (statistical)

Neural Language Models

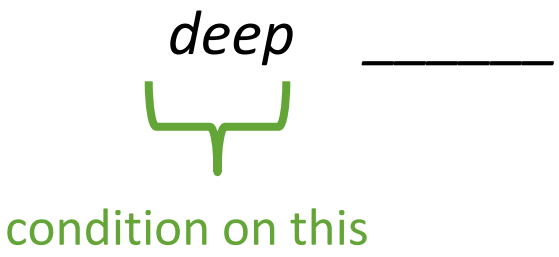
Language Models

Count-based (statistical)

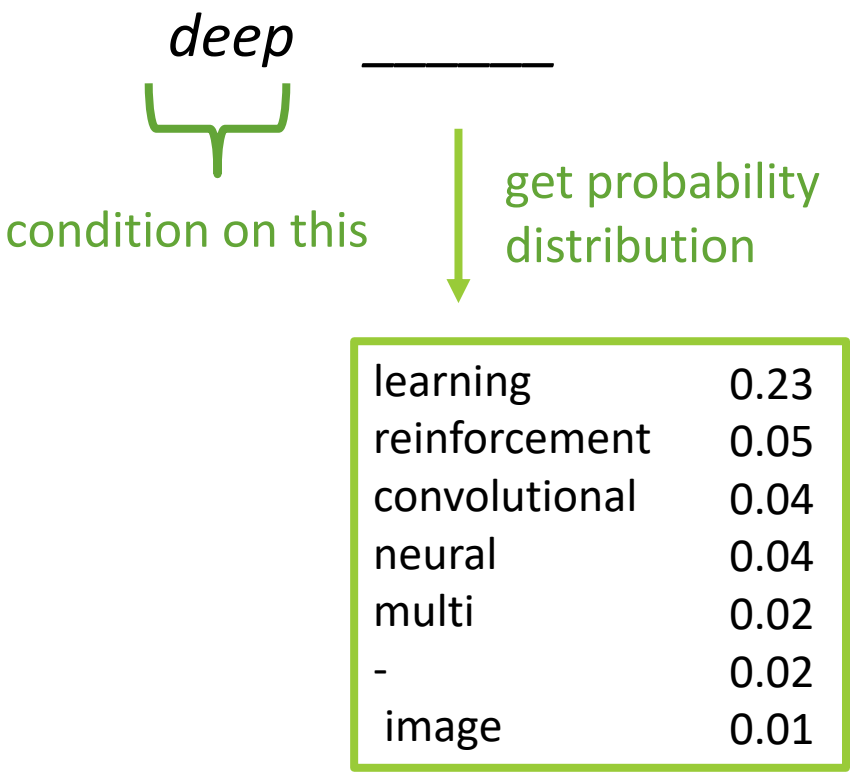
- make an n-th order Markov assumption
- estimate n-gram probabilities (counting and smoothing)

Neural Language Models

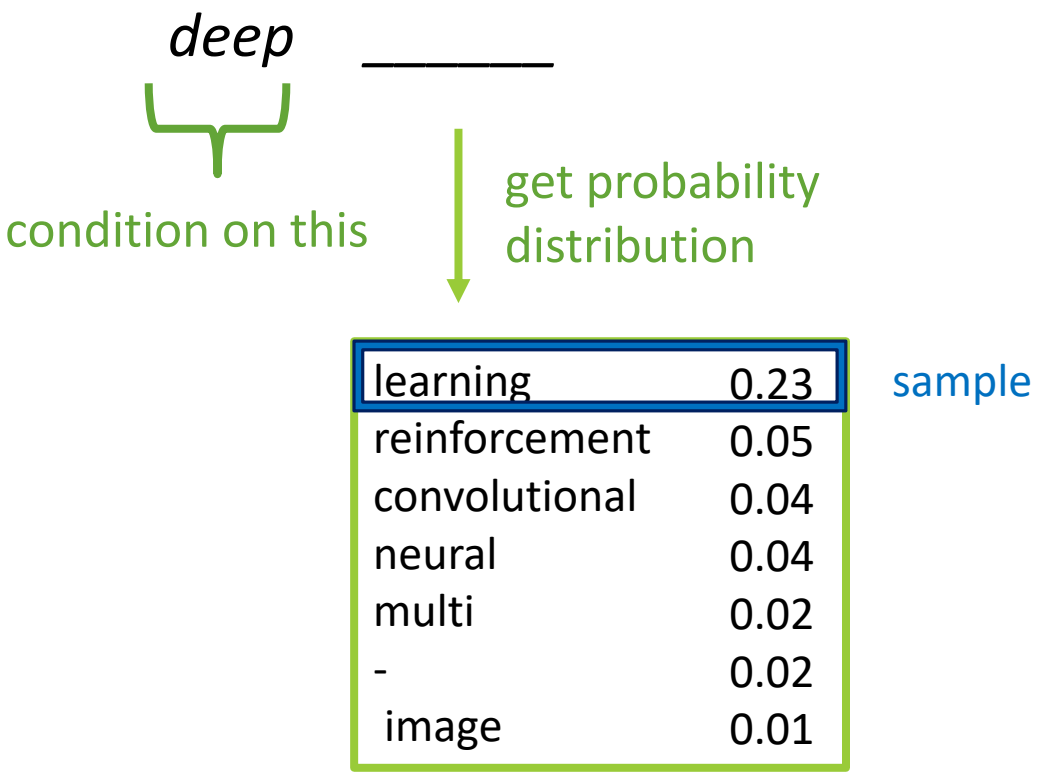
How to generate text using N-gram LM?



How to generate text using N-gram LM?



How to generate text using N-gram LM?

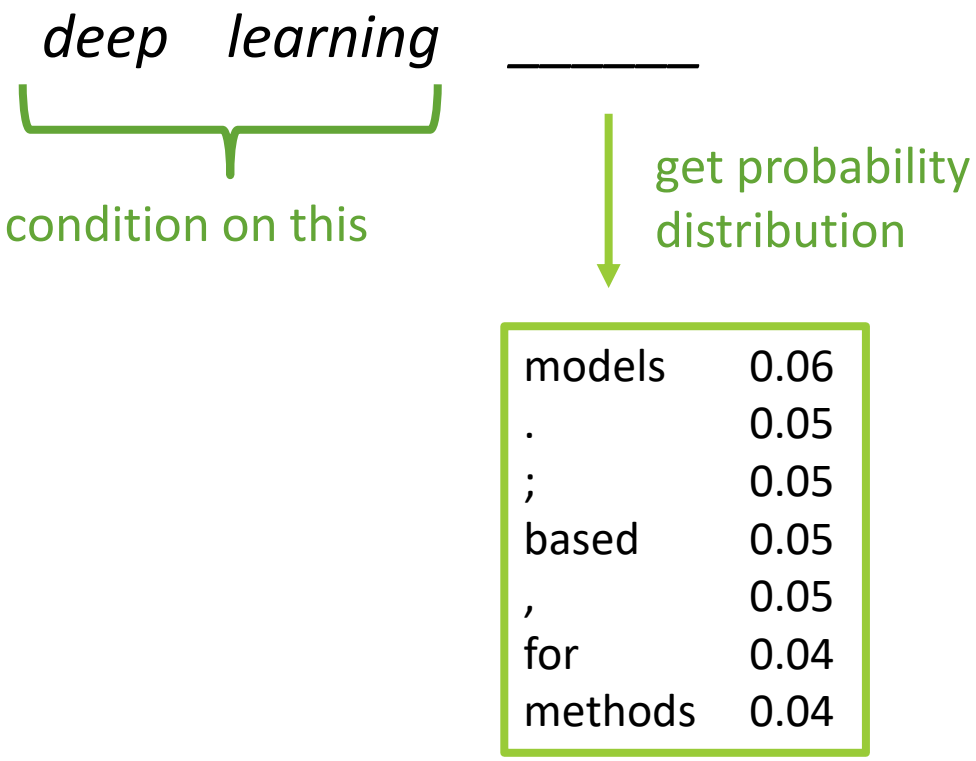


How to generate text using N-gram LM?

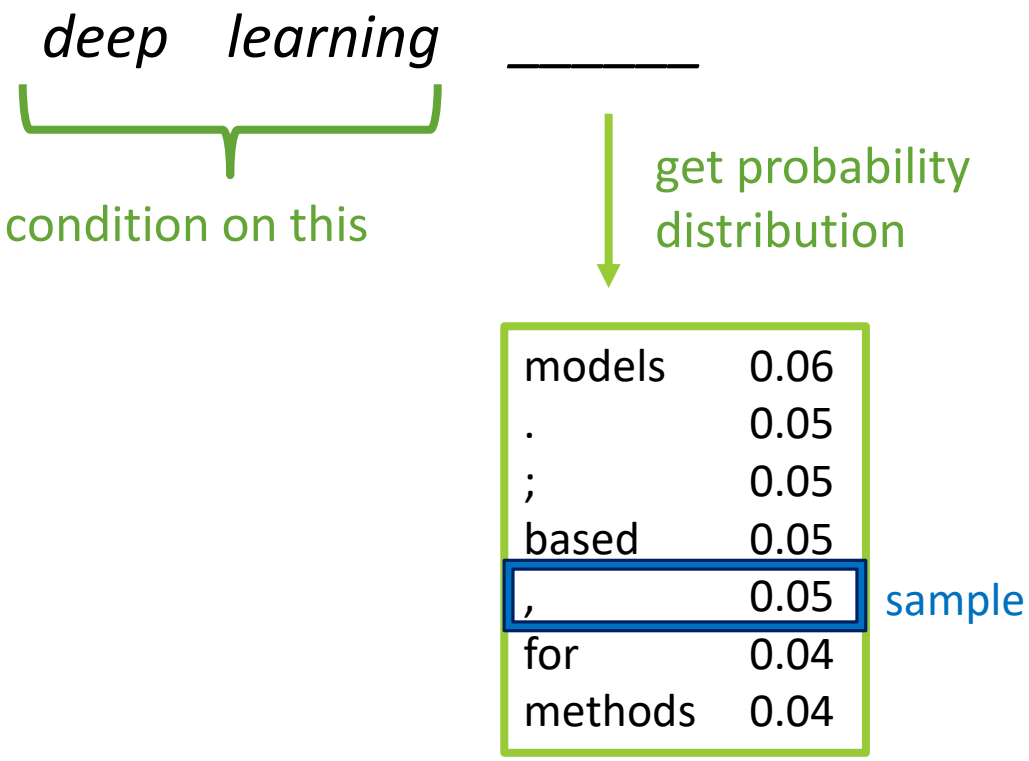
deep learning _____

condition on this

How to generate text using N-gram LM?



How to generate text using N-gram LM?

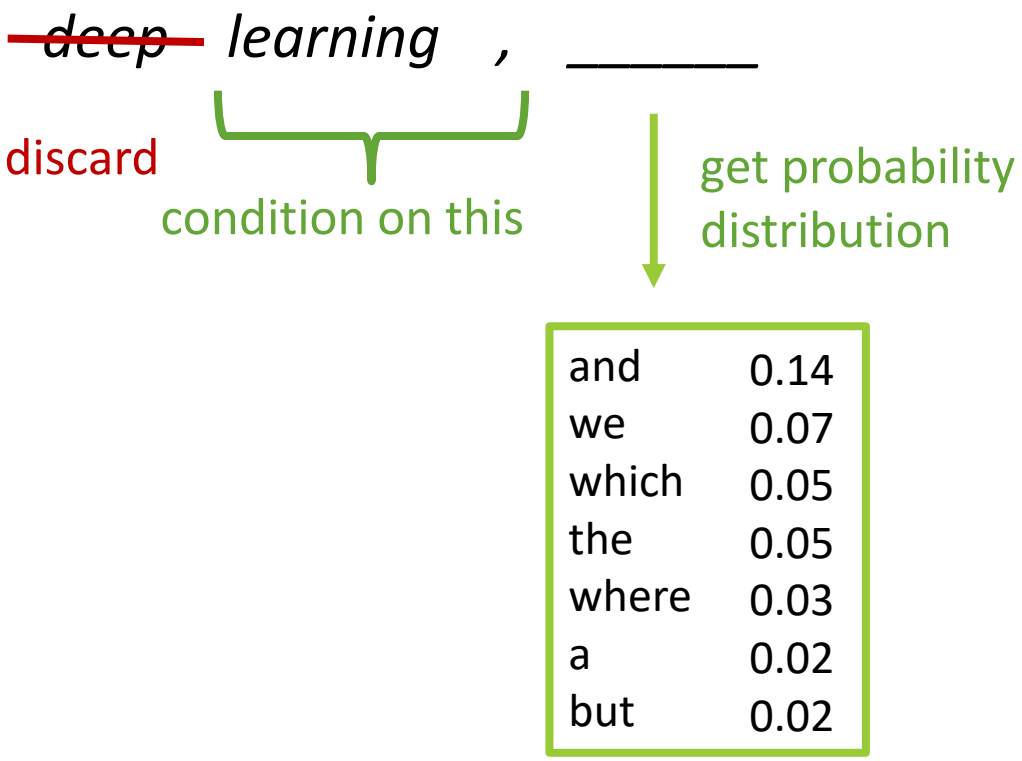


How to generate text using N-gram LM?

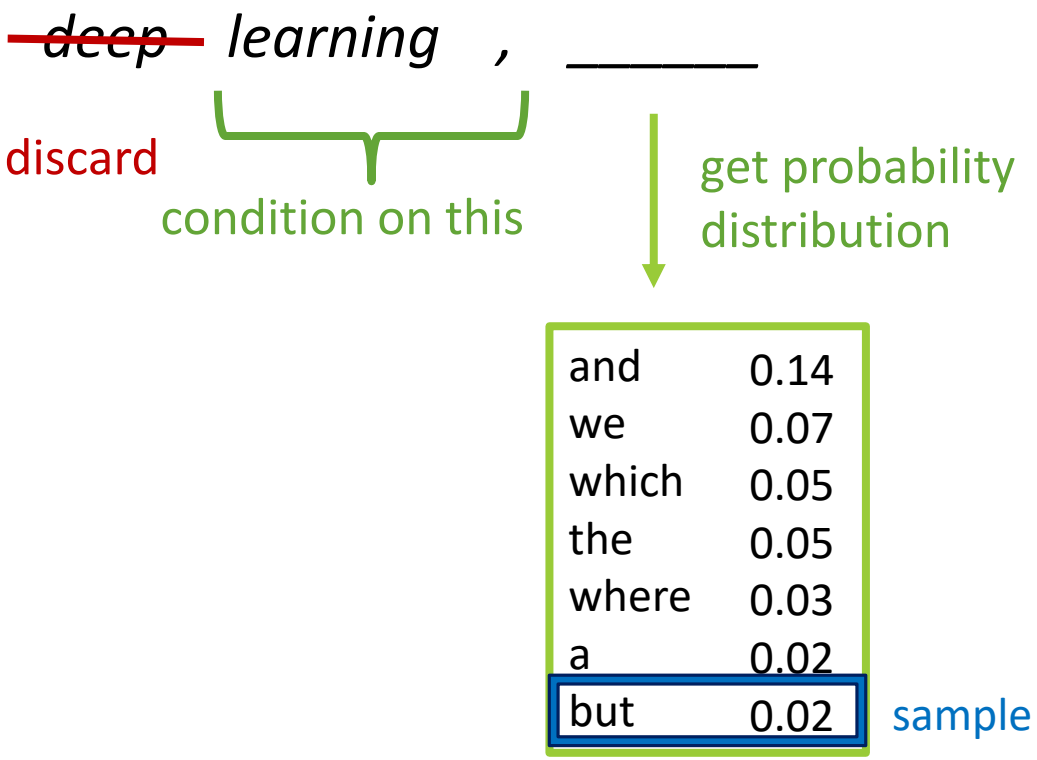
deep learning , _____

How to generate text using N-gram LM?

How to generate text using N-gram LM?



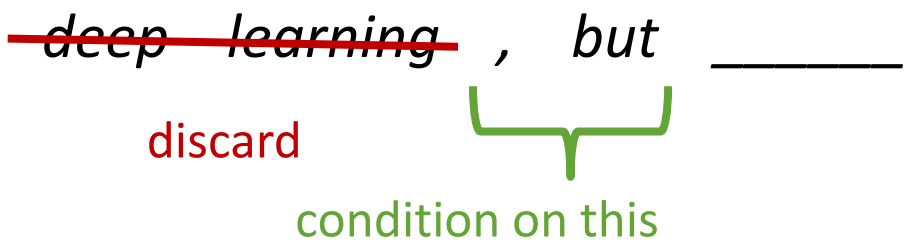
How to generate text using N-gram LM?



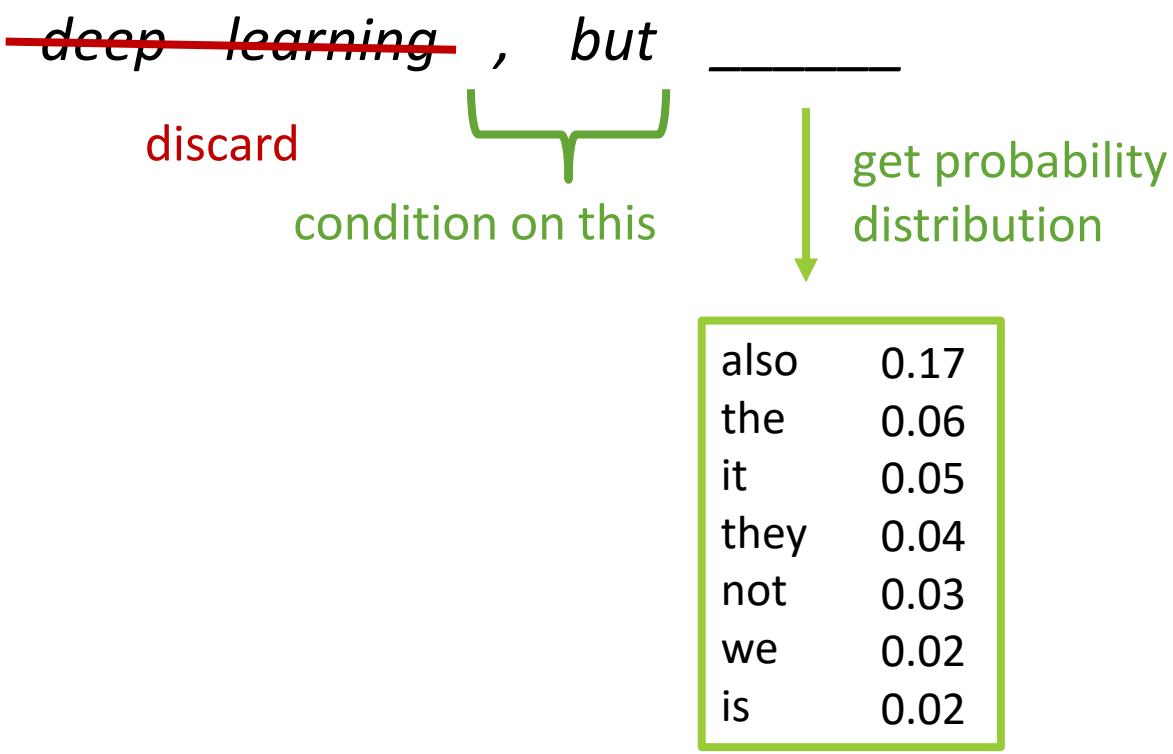
How to generate text using N-gram LM?

deep learning , but _____

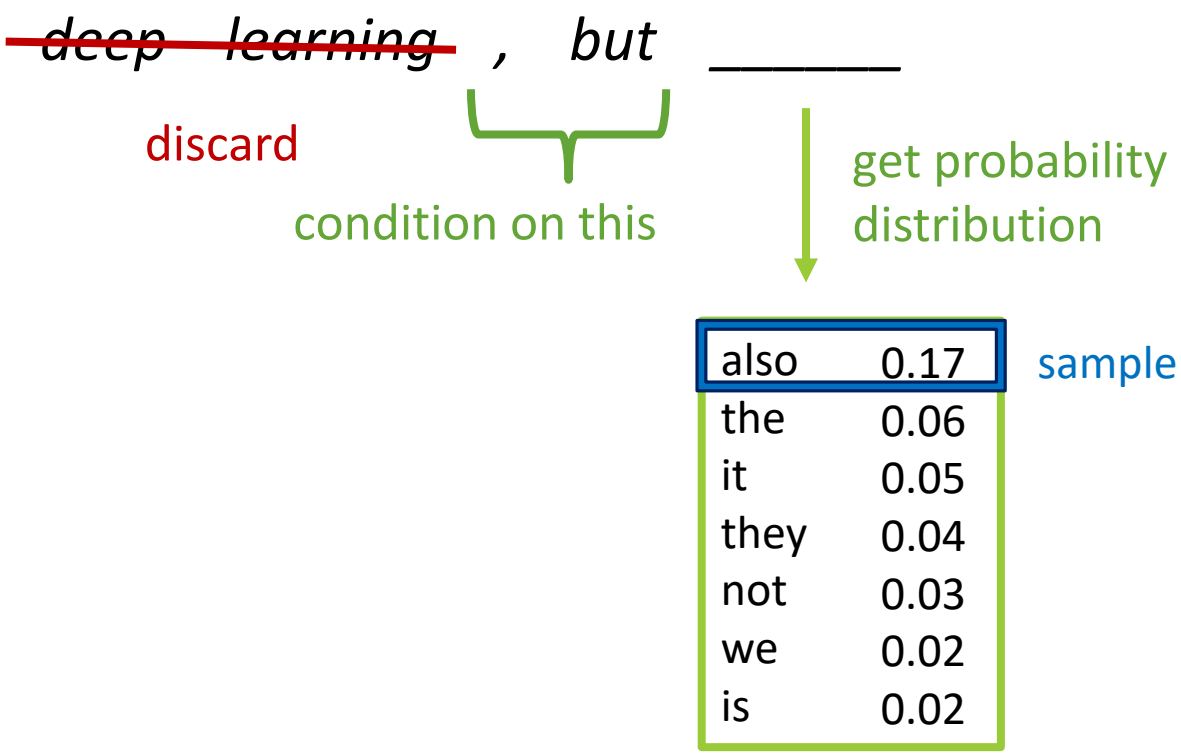
How to generate text using N-gram LM?



How to generate text using N-gram LM?



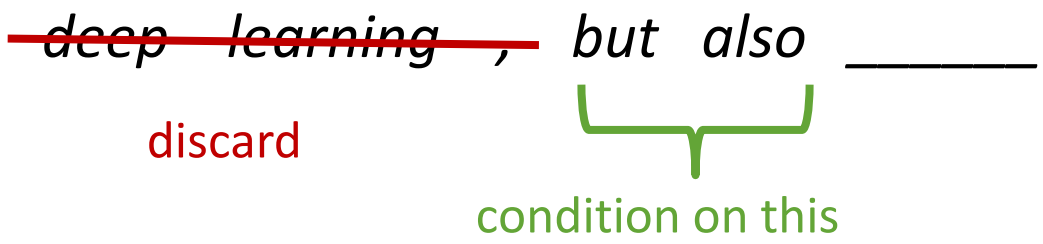
How to generate text using N-gram LM?



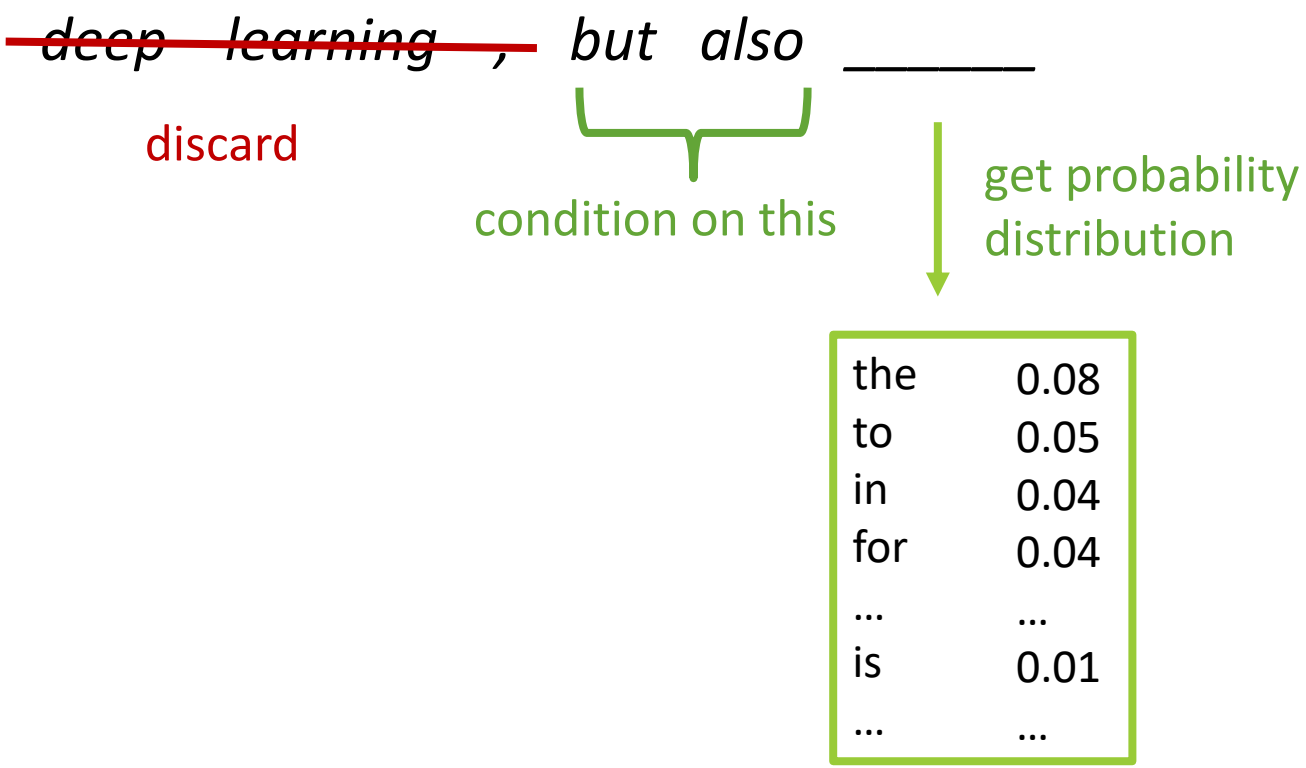
How to generate text using N-gram LM?

deep learning , but also _____

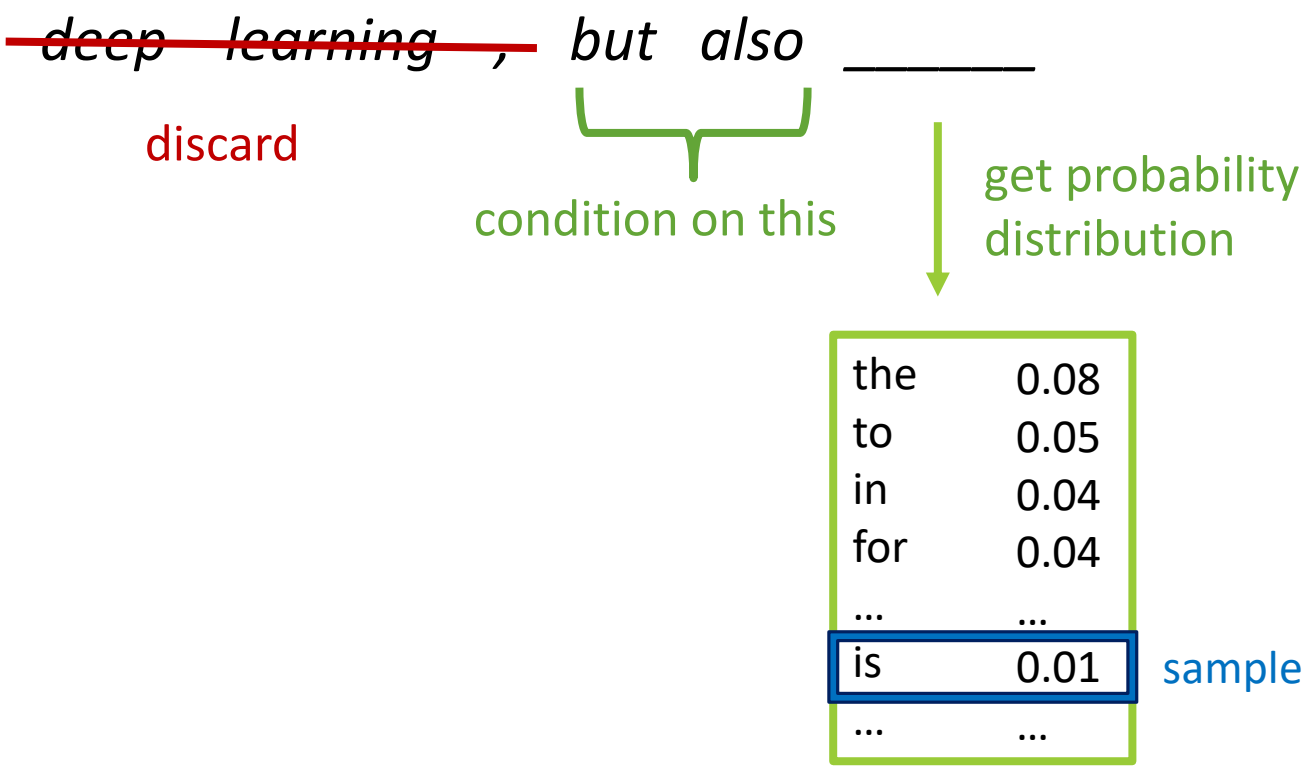
How to generate text using N-gram LM?



How to generate text using N-gram LM?



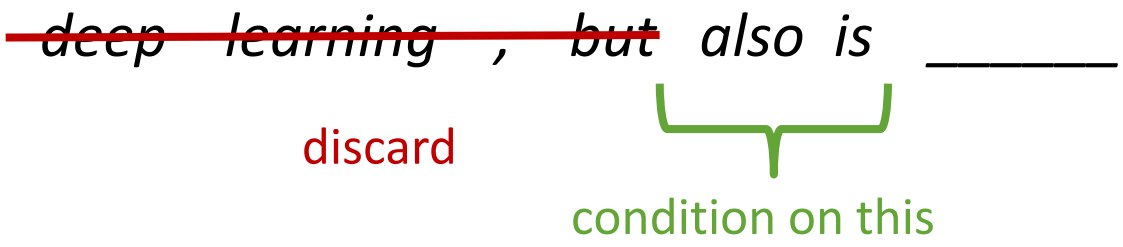
How to generate text using N-gram LM?



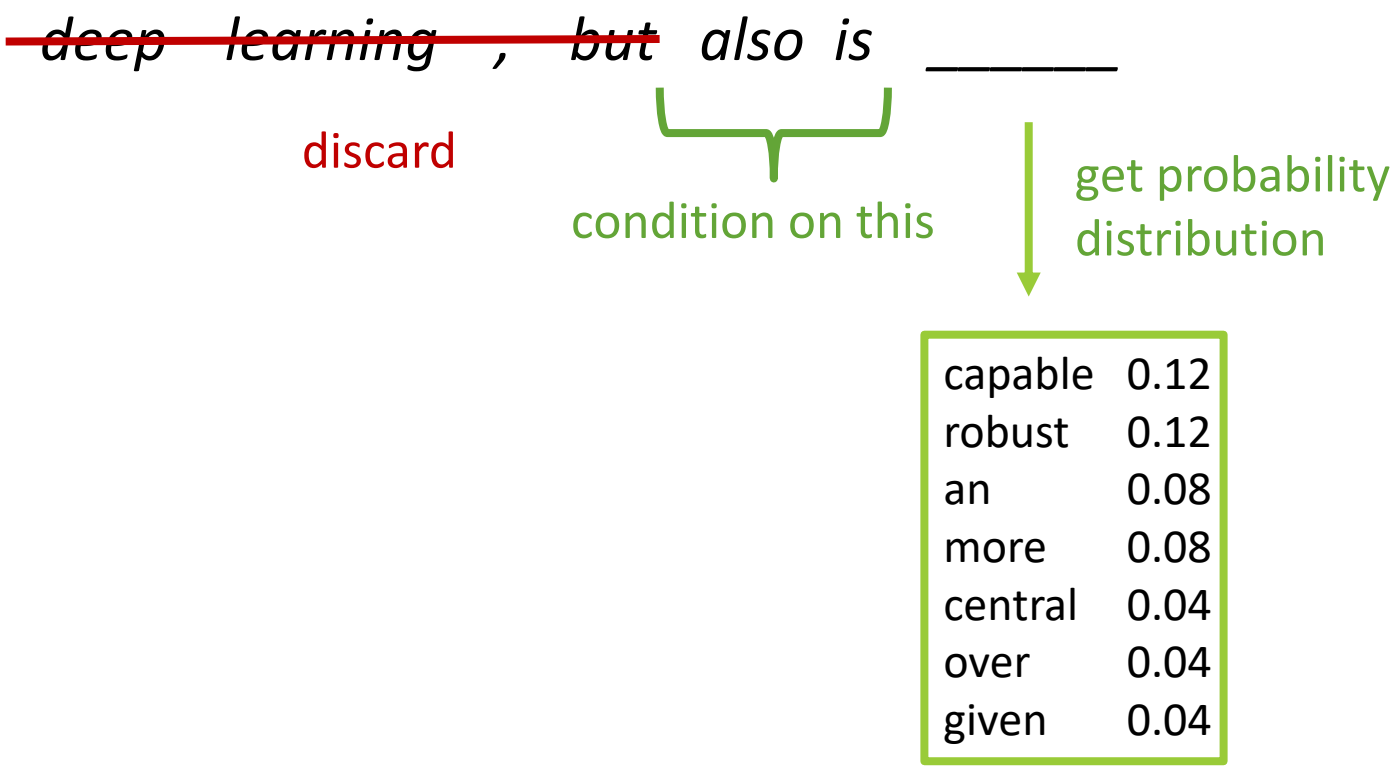
How to generate text using N-gram LM?

deep learning , but also is _____

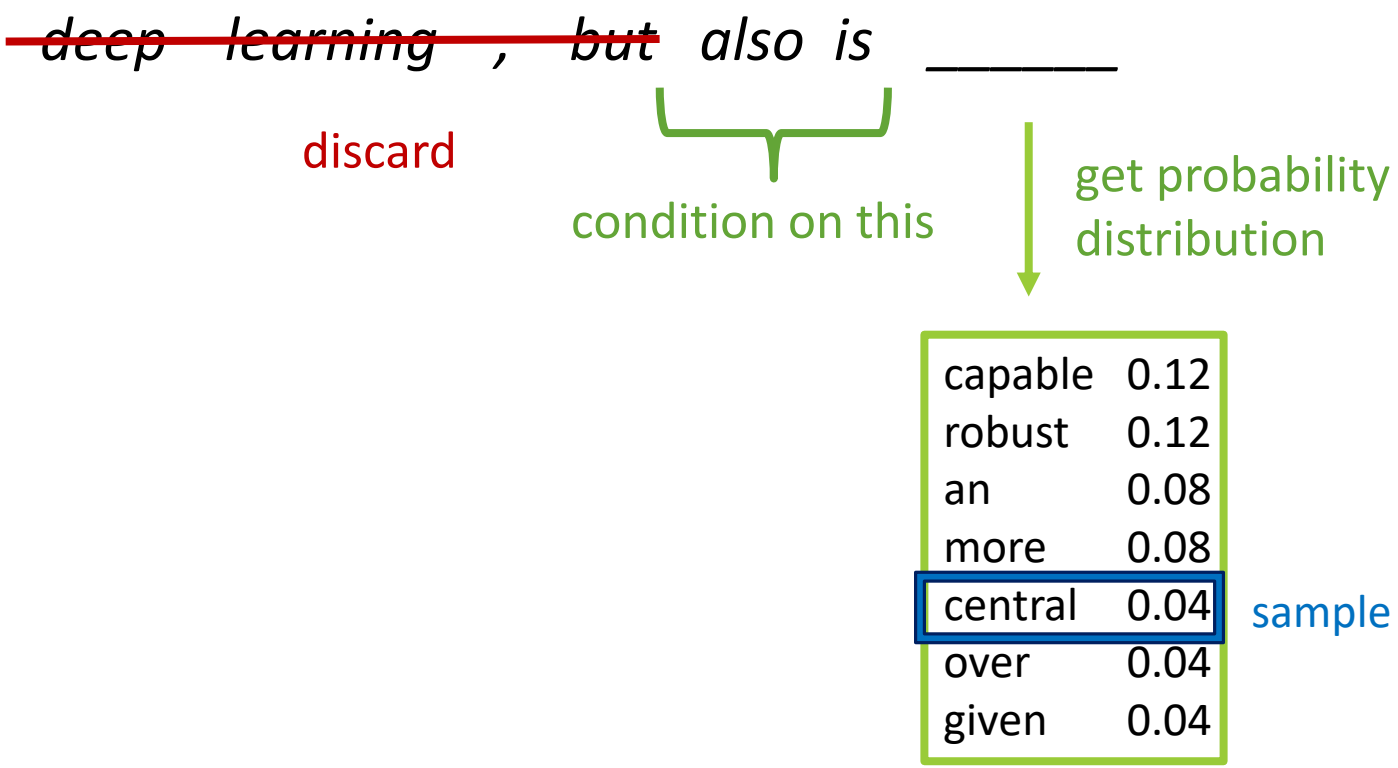
How to generate text using N-gram LM?



How to generate text using N-gram LM?



How to generate text using N-gram LM?



How to generate text using N-gram LM?

deep learning , but also is central _____

How to generate text using N-gram LM?

deep learning , but also is central to _____

How to generate text using N-gram LM?

deep learning , but also is central to human _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance . _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance . however _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance . however , _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance .
however , using _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance .
however , using structural _____

How to generate text using N-gram LM?

deep learning , but also is central to human performance .
however , using structural similarity _____

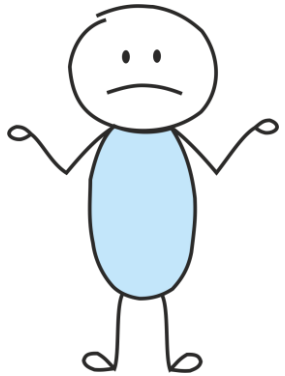
How to generate text using N-gram LM?

deep learning , but also is central to human performance . however , using structural similarity index measure than other partitioned sampling schemes , while making the approach with empirical data has the effect of phonetics has received little attention within the context of information on …

How to generate text using N-gram LM?

deep learning , but also is central to human performance . however , using structural similarity index measure than other partitioned sampling schemes , while making the approach with empirical data has the effect of phonetics has received little attention within the context of information on …

What's wrong with this text?

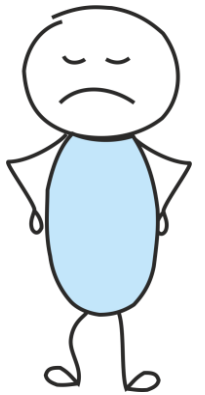


How to generate text using N-gram LM?

deep learning , but also is central to human performance . however , using structural similarity index measure than other partitioned sampling schemes , while making the approach with empirical data has the effect of phonetics has received little attention within the context of information on …

What's wrong with this text?

It's completely incoherent!



Neural Language Models

AKA NLM

Fixed-window Neural Language Models

Output distribution

$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$

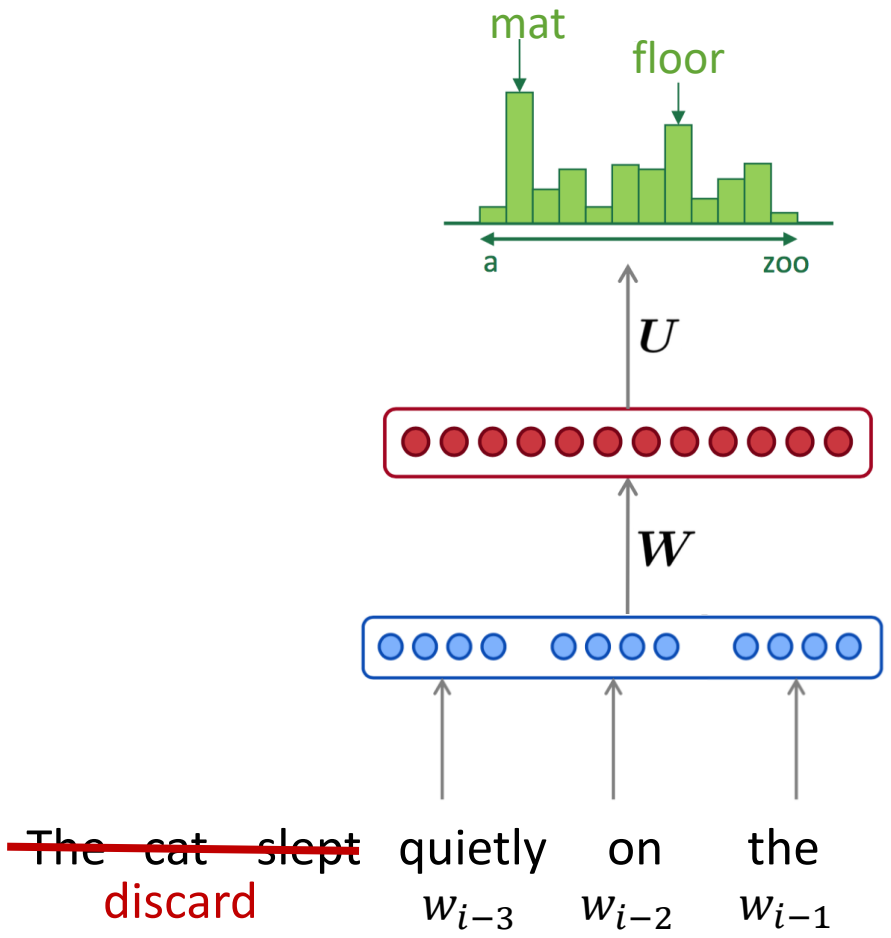
Hidden layer (or any feed-forward NN)

$h = f(Wx + b)$

Concatenate word embeddings

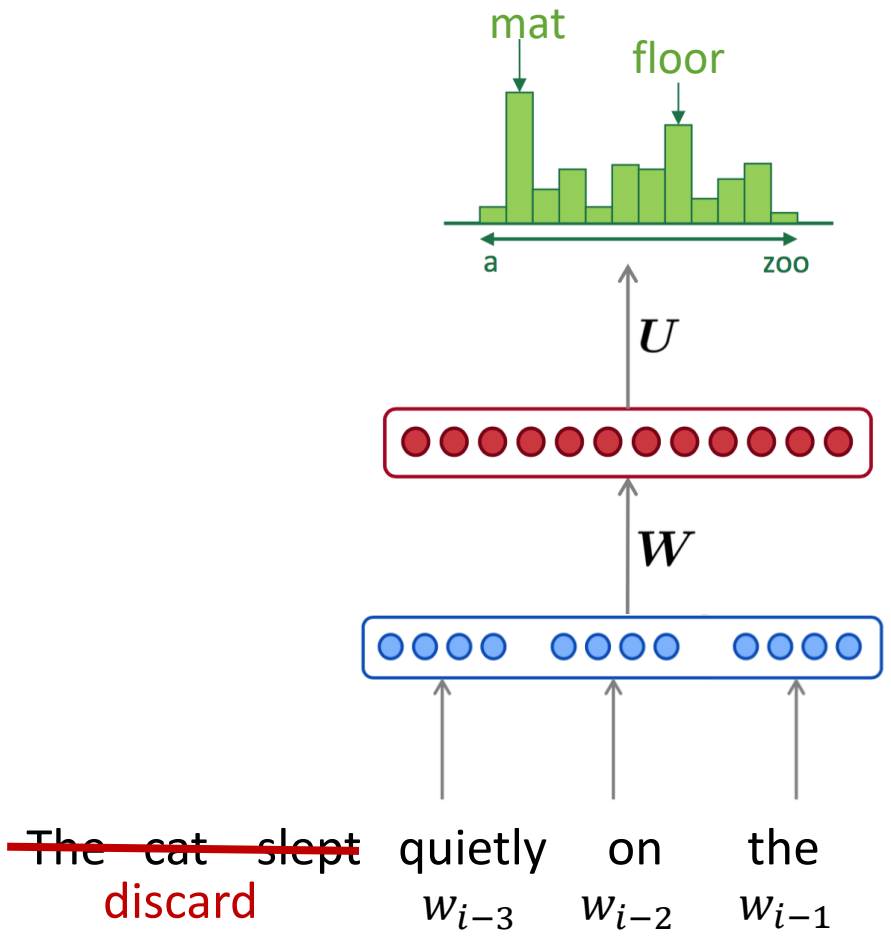
$x = (x_{i-3}, x_{i-2}, x_{i-1})$

Word embeddings



Fixed-window Neural Language Models

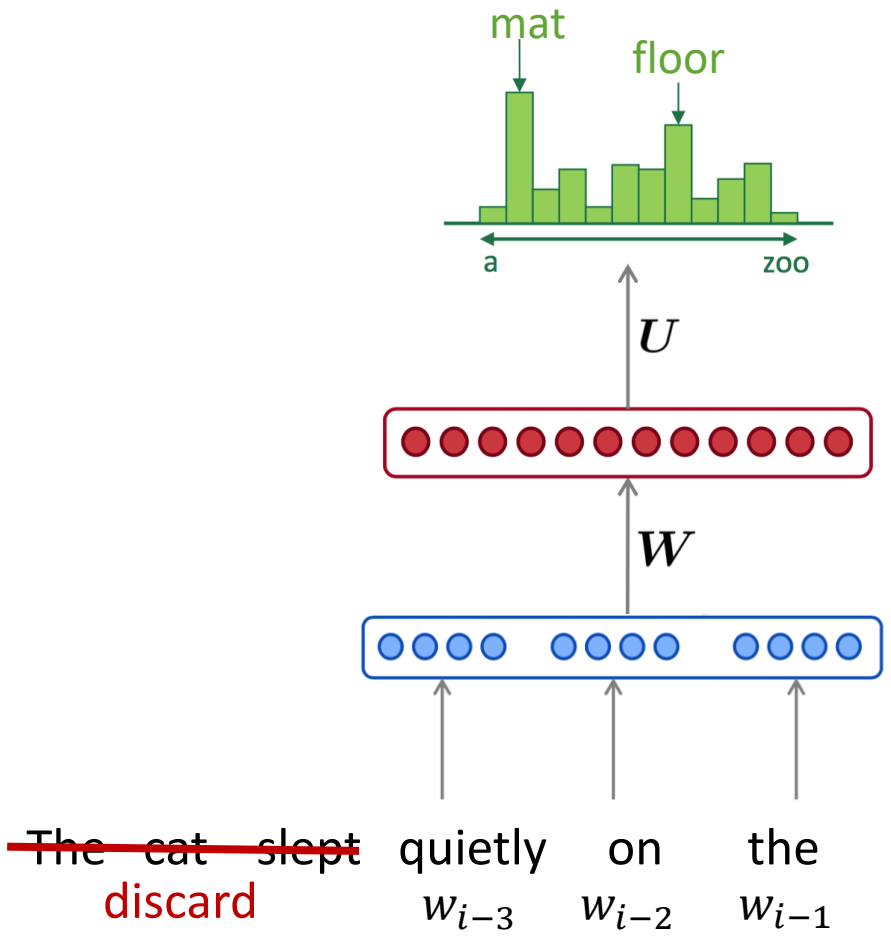
Improvements over N-gram LMs:



Fixed-window Neural Language Models

Improvements over N-gram LMs:

- no sparsity problem
- model size O(n) (not O(exp(n)))

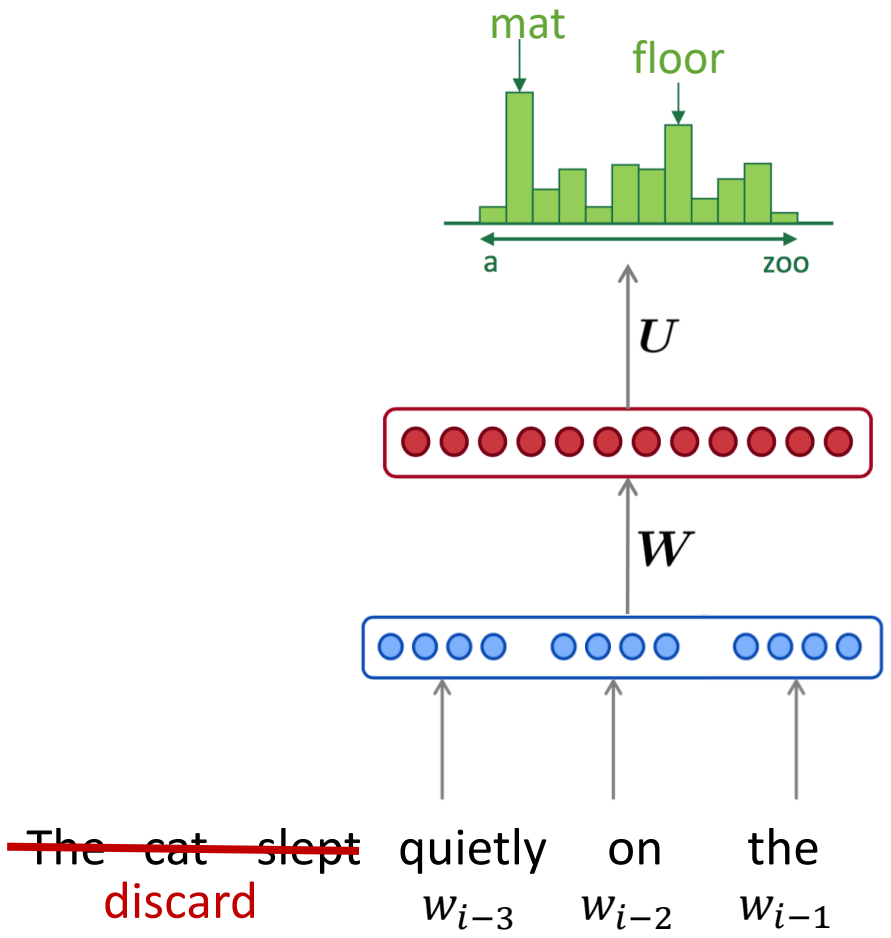


Fixed-window Neural Language Models

Improvements over N-gram LMs:

- no sparsity problem
- model size O(n) (not O(exp(n)))

Remaining **problems**:



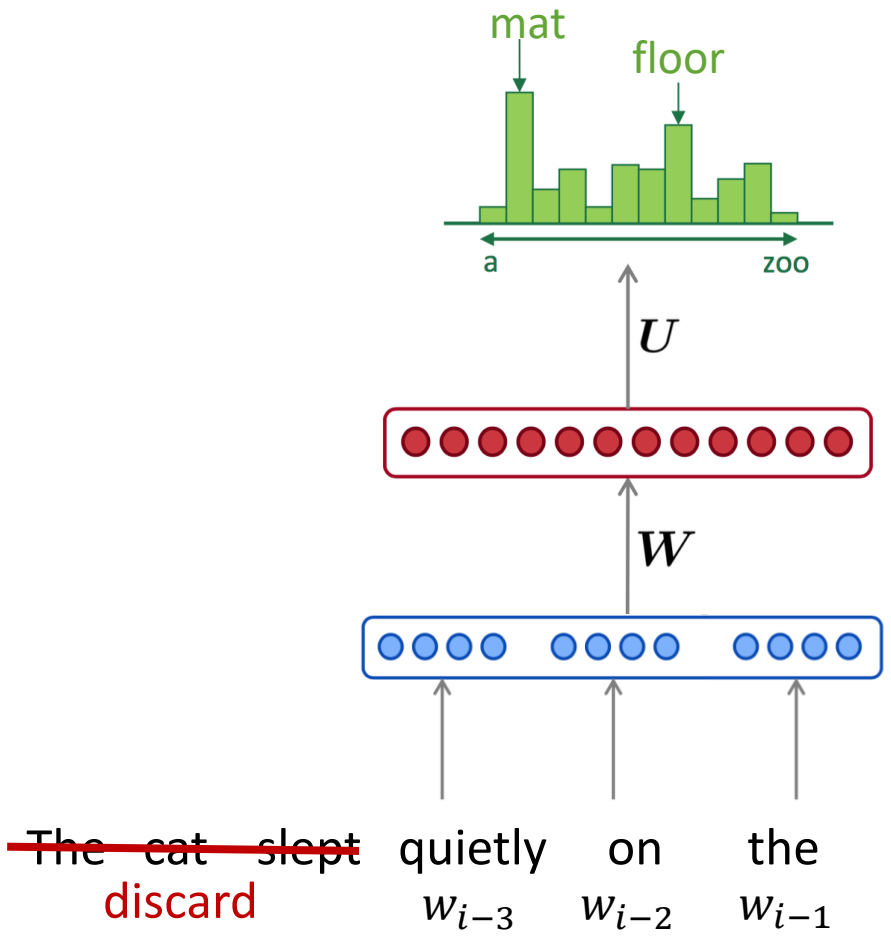
Fixed-window Neural Language Models

Improvements over N-gram LMs:

- no sparsity problem
- model size O(n) (not O(exp(n)))

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- Each uses different rows of . We don't share weights across the window.



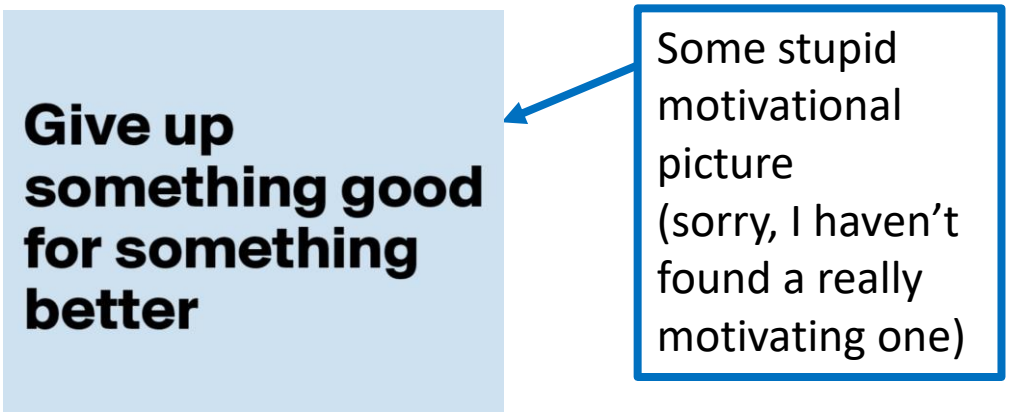
Fixed-window Neural Language Models

Improvements over N-gram LMs:

- no sparsity problem
- model size O(n) (not O(exp(n)))

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- Each uses different rows of . We don't share weights across the window.



We need a neural architecture that can process input of *any length!*

RNN Language Models

Output distribution

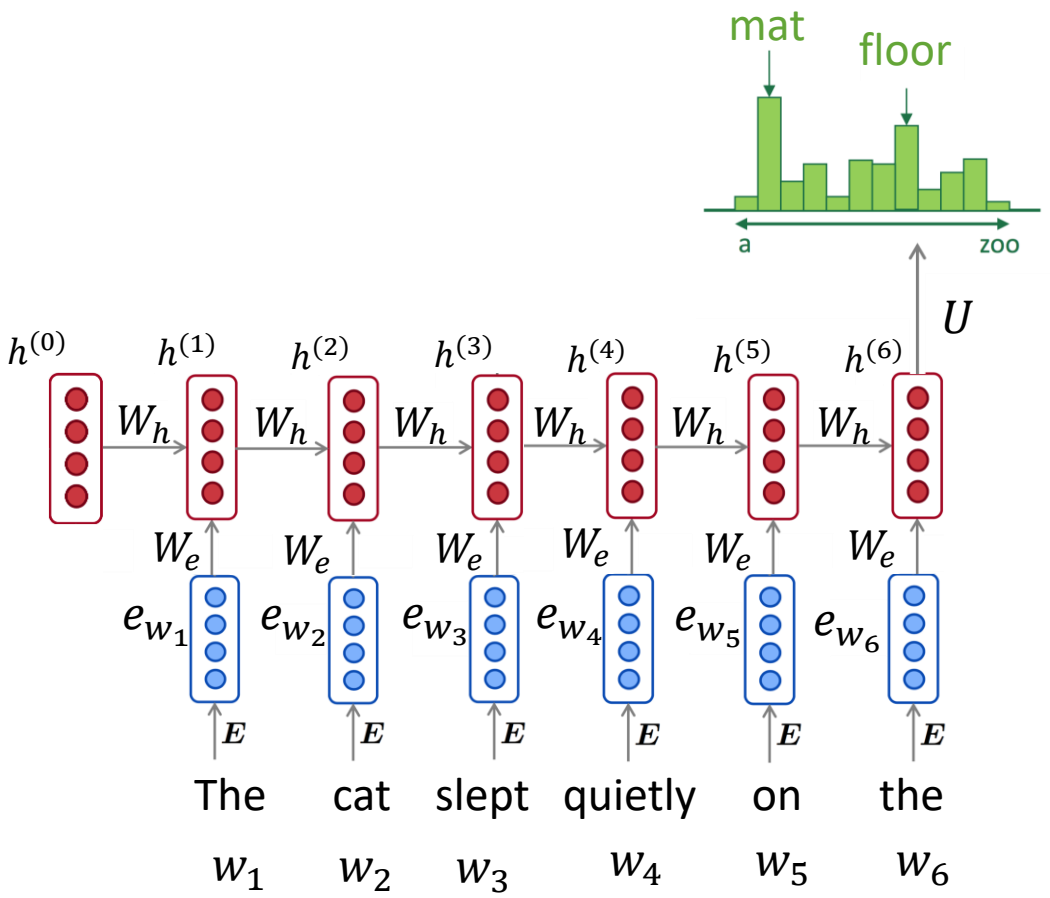
$$\hat{y}_t = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

Hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e_{w_{t-1}} + b_1\right)$$

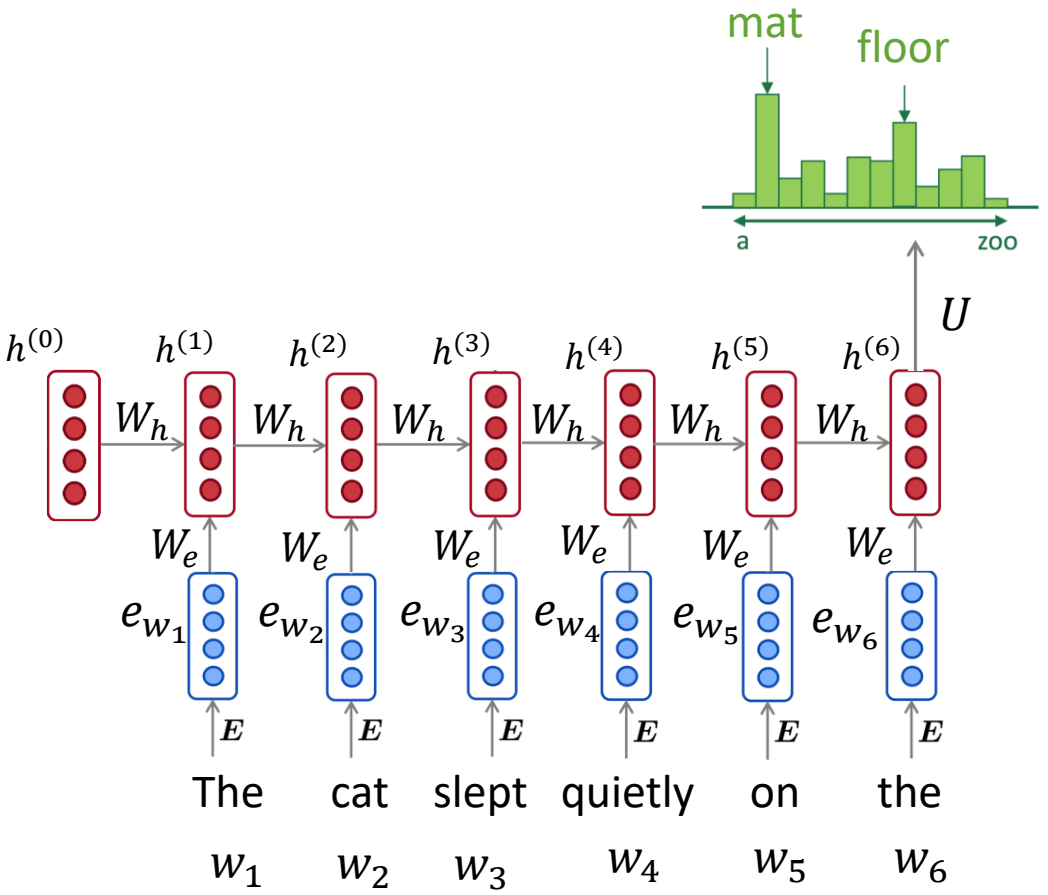
Word embeddings

Words/tokens



RNN Language Models

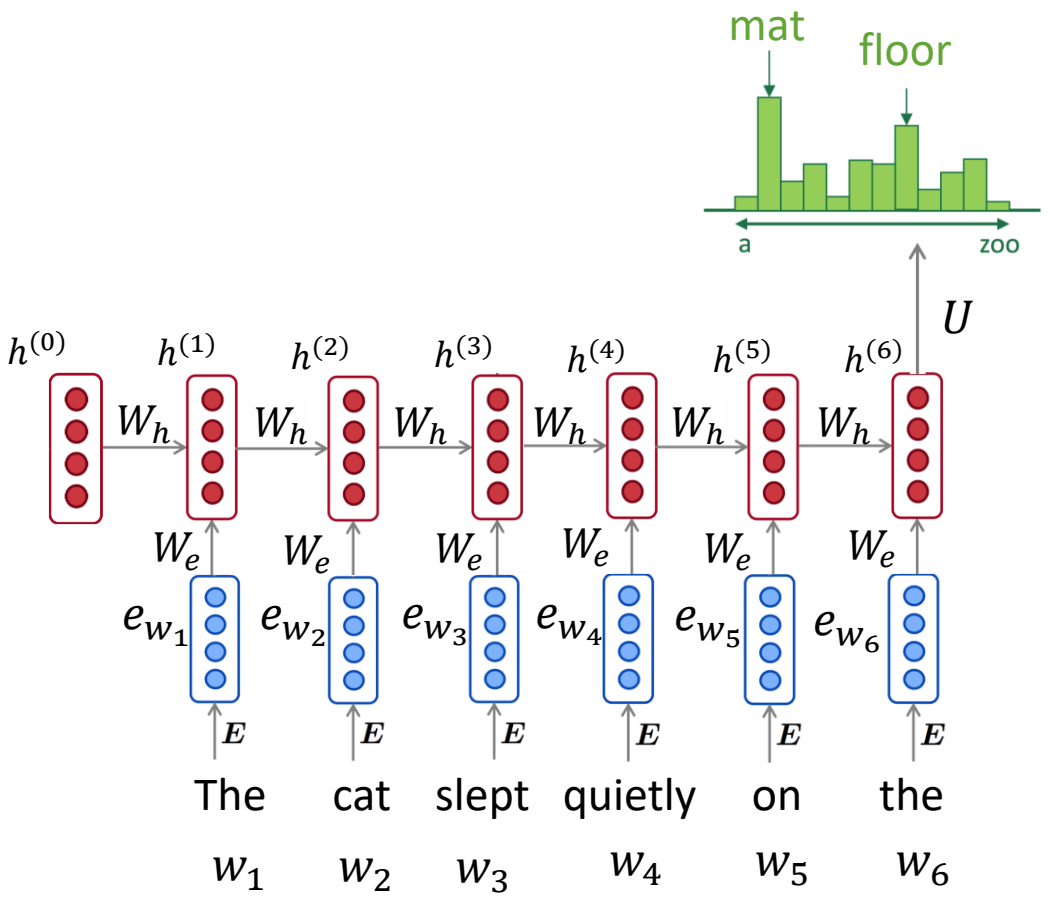
RNN **Advantages**:



RNN Language Models

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- representations shared across timesteps

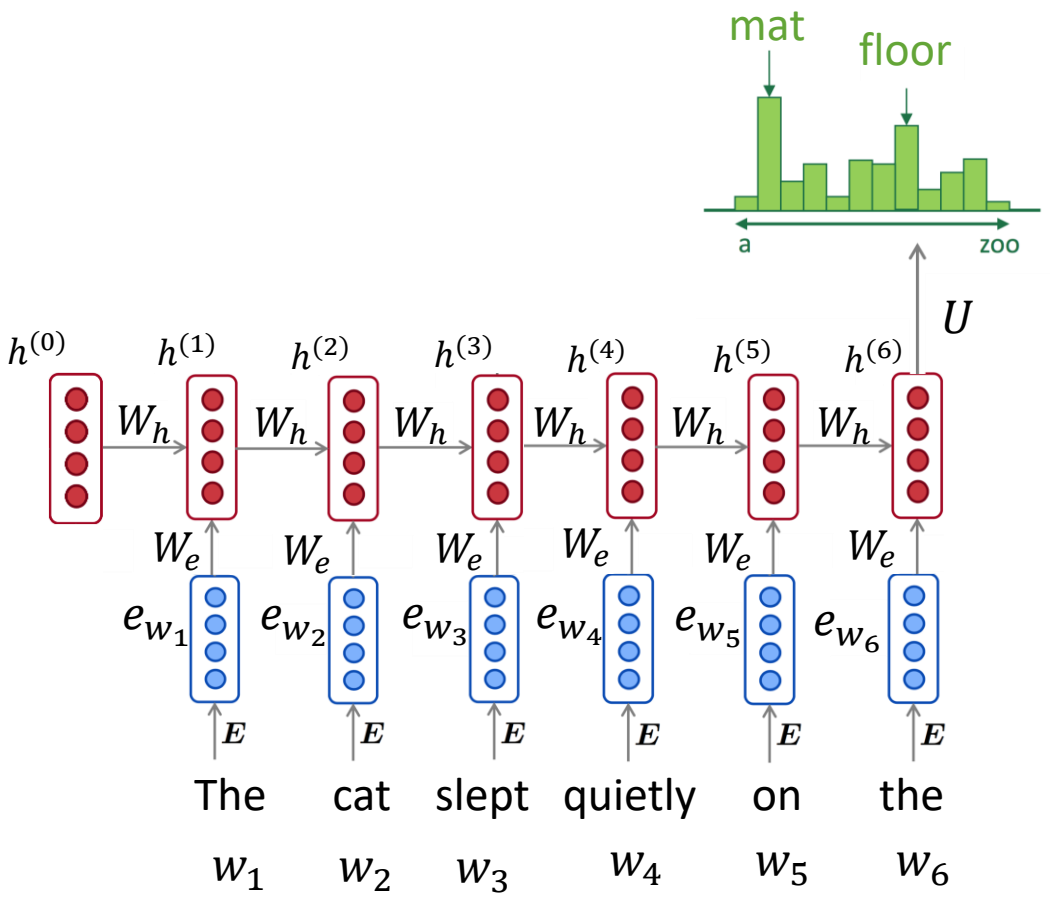


RNN Language Models

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- representations shared across timesteps

Remaining **problems**:



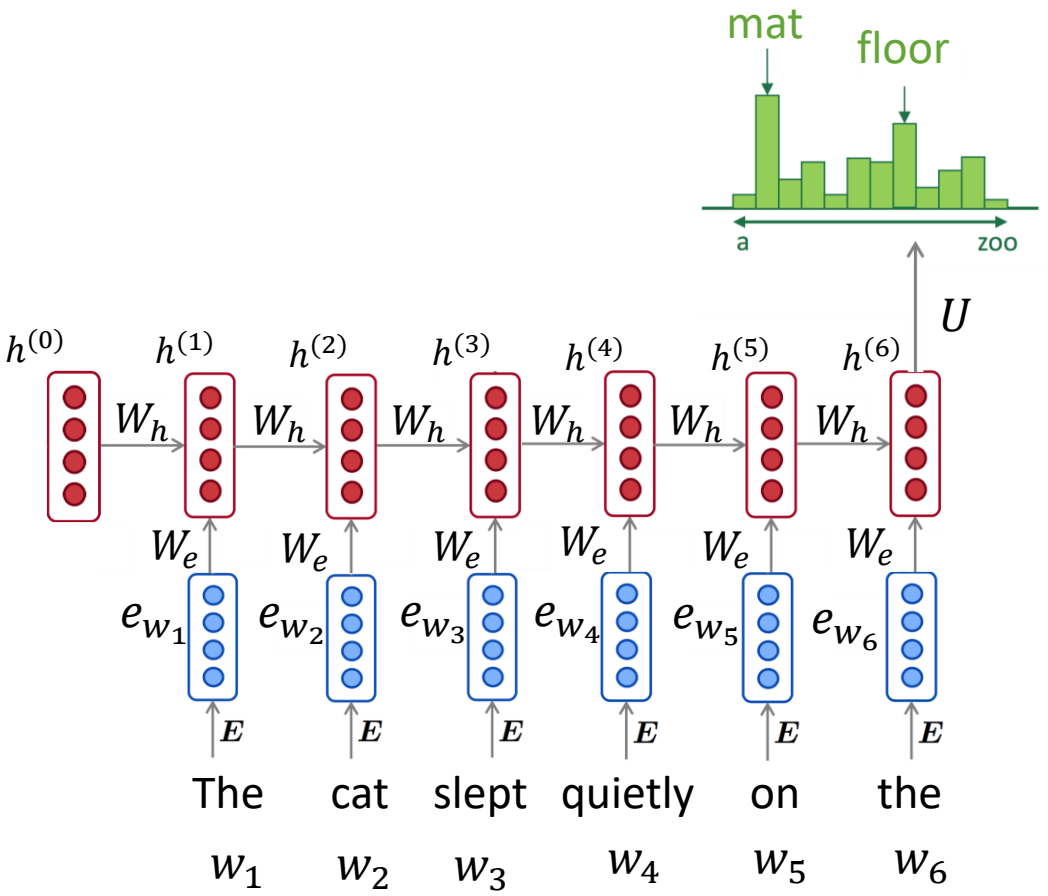
RNN Language Models

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- representations shared across timesteps

Remaining **problems**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

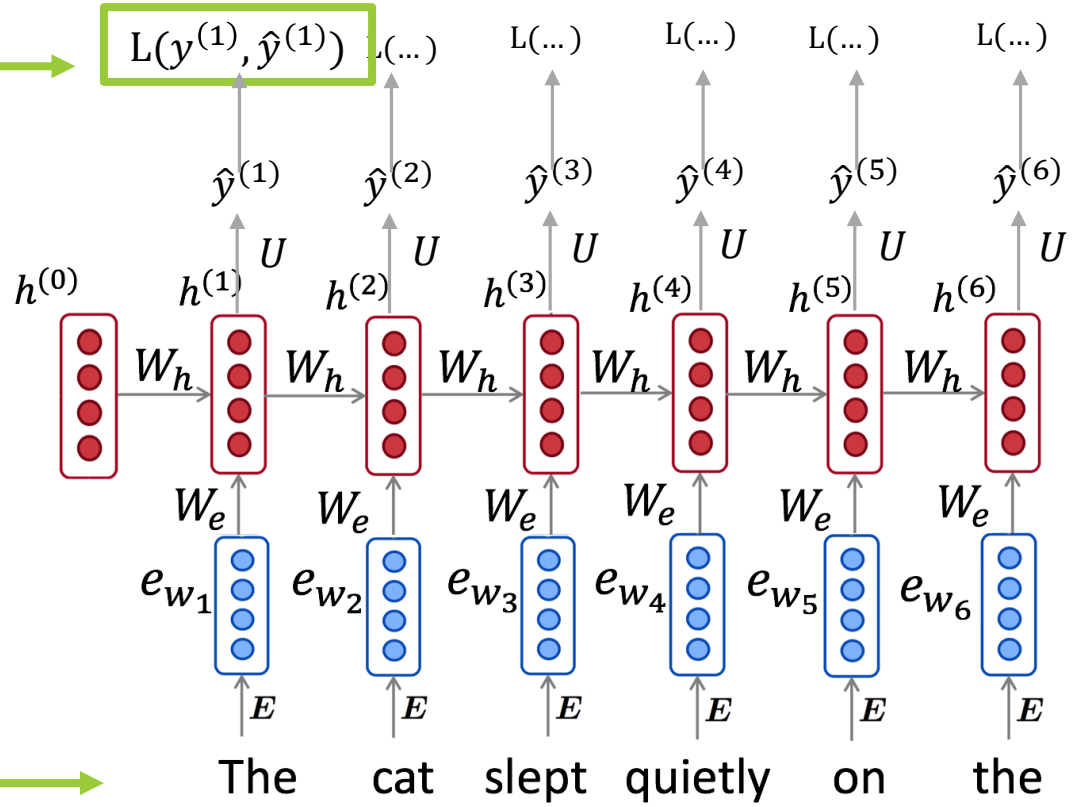


Training RNN-LM

negative log prob of "cat"

Loss

$$\mathrm{L}\left(y^{(t)}, \hat{y}^{(t)}\right) = \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$



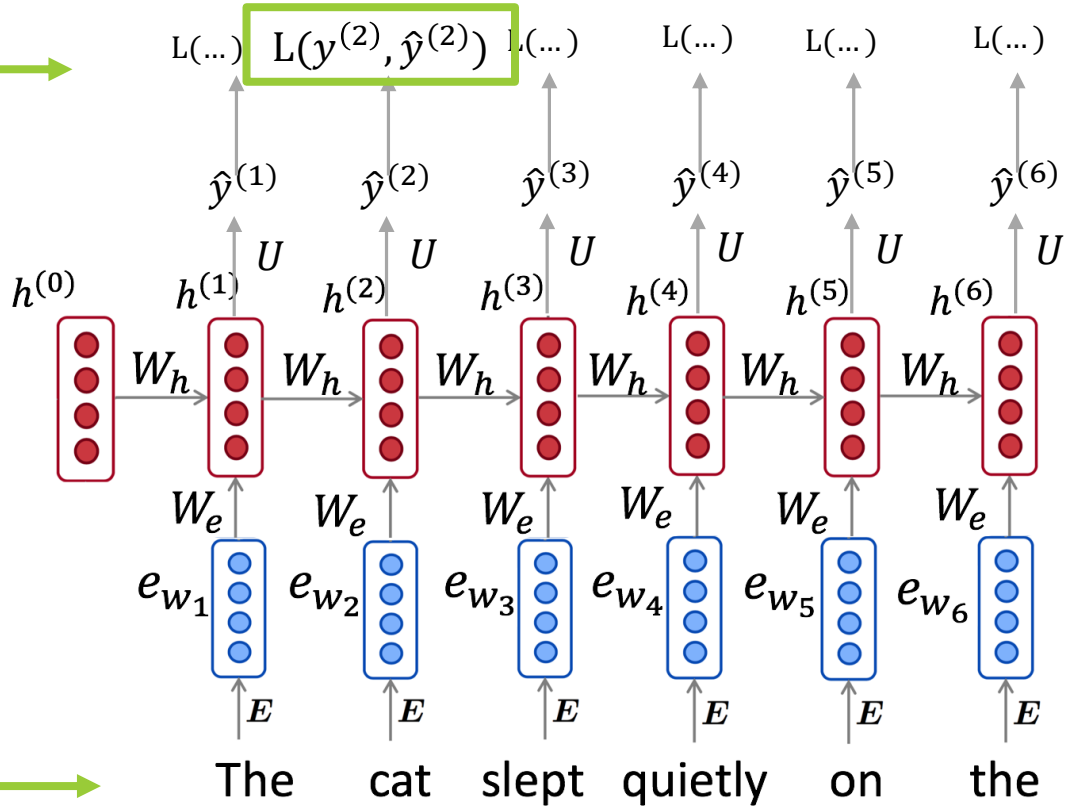
corpus

Training RNN-LM

negative log prob of "slept"

Loss

$$\mathrm{L}\left(y^{(t)}, \hat{y}^{(t)}\right) = \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

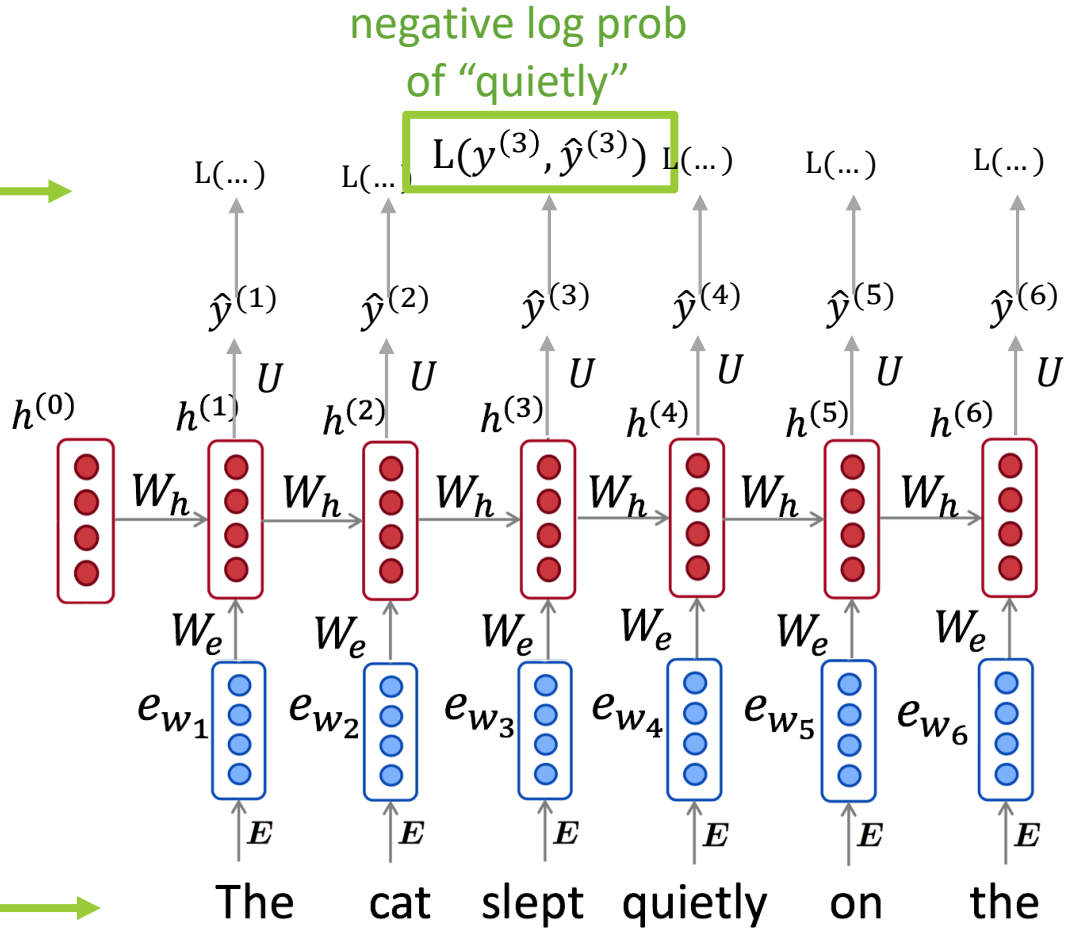


corpus

Training RNN-LM

Loss

$$\mathrm{L}\left(y^{(t)}, \hat{y}^{(t)}\right) = \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

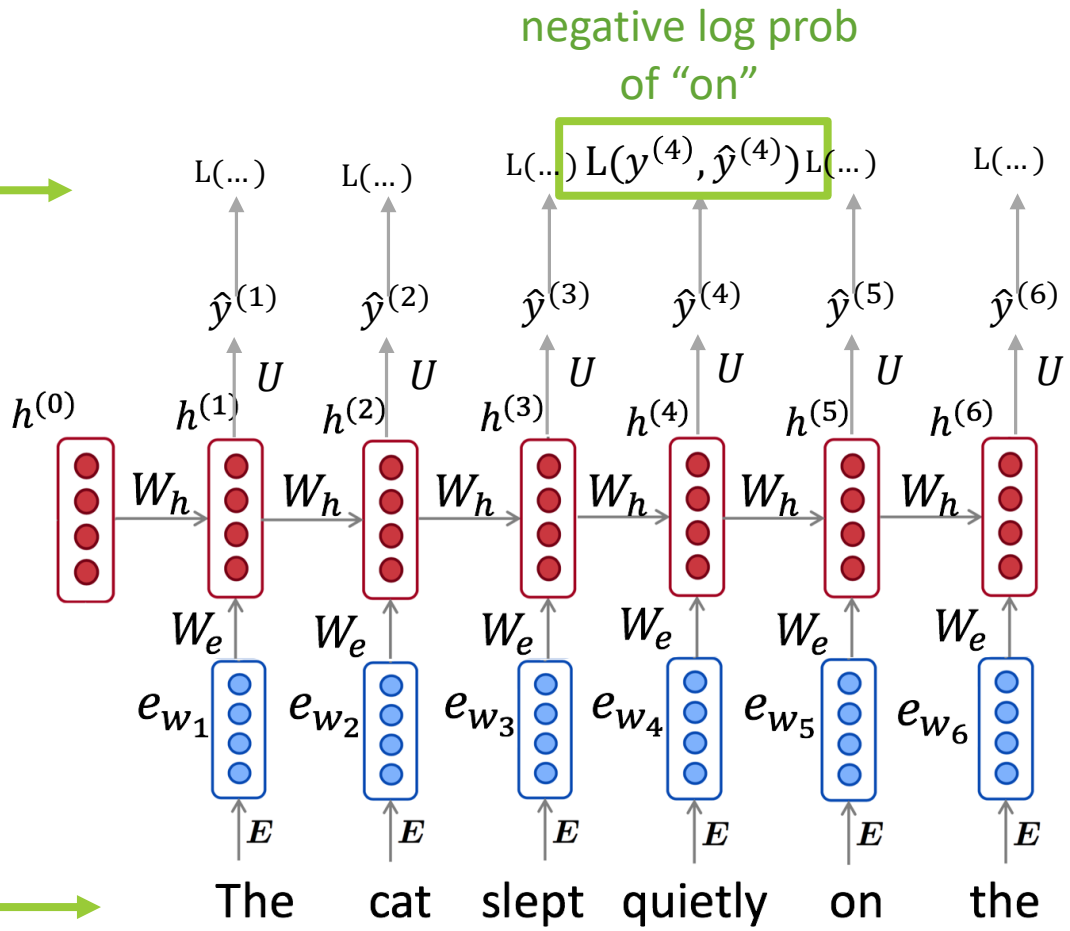


corpus

Training RNN-LM

Loss

$$\mathrm{L}\left(y^{(t)}, \hat{y}^{(t)}\right) = \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

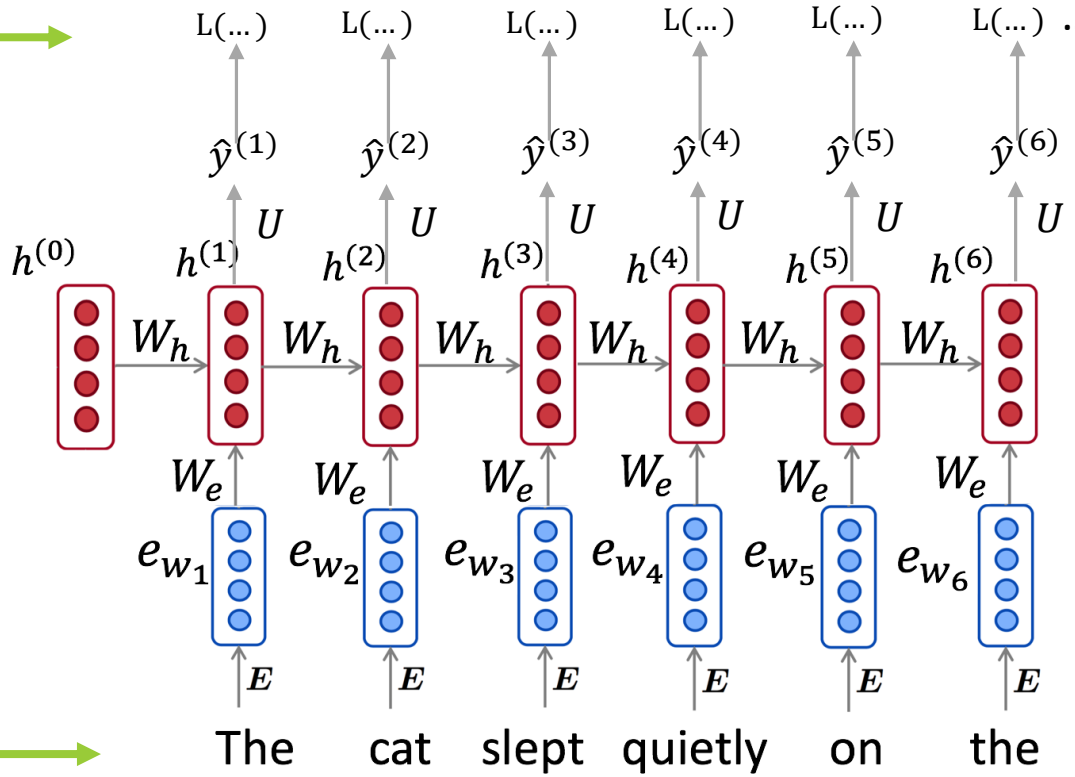


corpus

Training RNN-LM

Loss

$$\mathrm{L}\left(y^{(t)}, \hat{y}^{(t)}\right) = \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$



sum

$$\sum_{t=1}^{T} L(y^{(t)}, \hat{y}^{(t)})$$

corpus

Language Models

Count-based (statistical)

- make an n-th order Markov assumption
- estimate n-gram probabilities (counting and smoothing)

Neural Language Models

Language Models

Count-based (statistical)

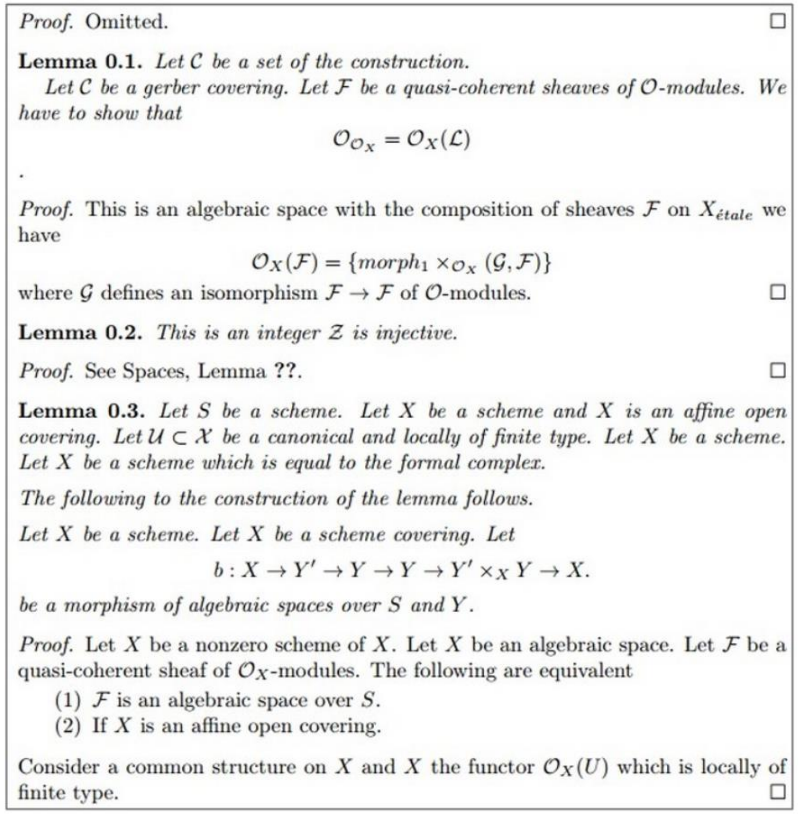
- make an n-th order Markov assumption
- estimate n-gram probabilities (counting and smoothing)

Neural Language Models

- solve the problem of data sparsity of the n-gram model, by representing words as vectors
- the parameters are learned as part of the training process

Visualizing and Understanding Recurrent Networks

Char-based LM trained on Latex of book on algebraic geometry



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

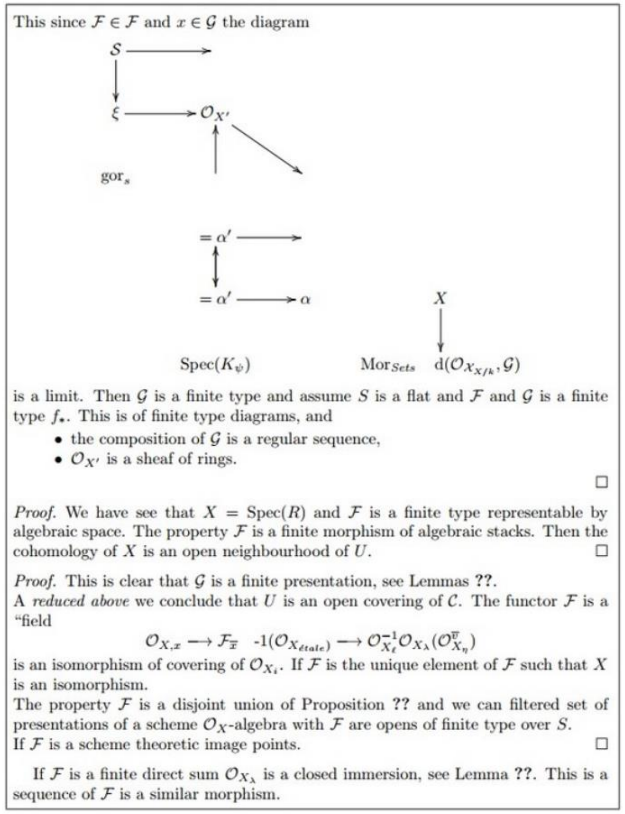
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □



This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \text{gor}_s & & \searrow \\ & & \\ = \alpha' & \longrightarrow & \\ \updownarrow & & \\ = \alpha' & \longrightarrow & \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \\ \text{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G}) \end{array}$$

$\text{Spec}(K_\psi)$ $\quad$ Mor_{Sets}

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{x}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

More hallucinated algebraic geometry. Nice try on the diagram (right).

Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

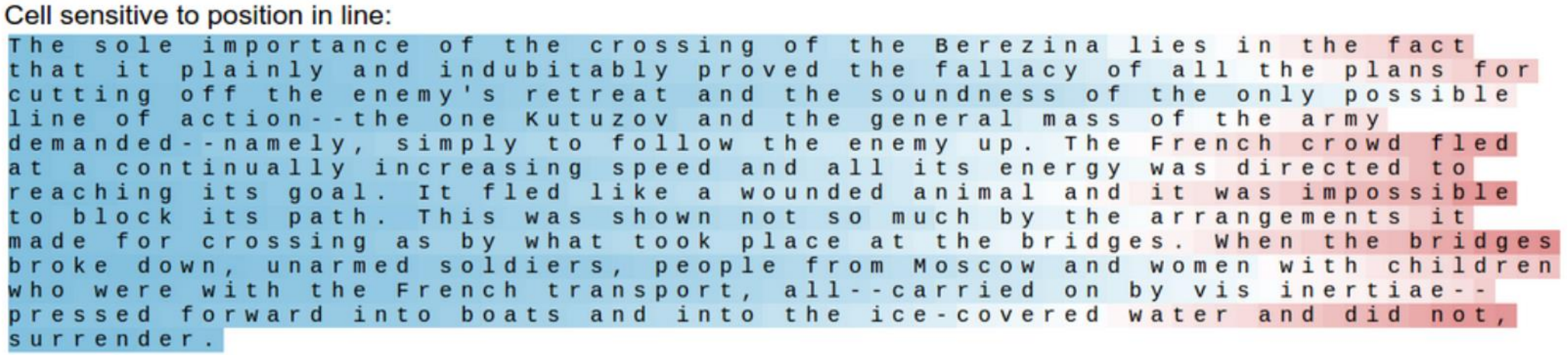
Char-based LM trained on Linux Source Code

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

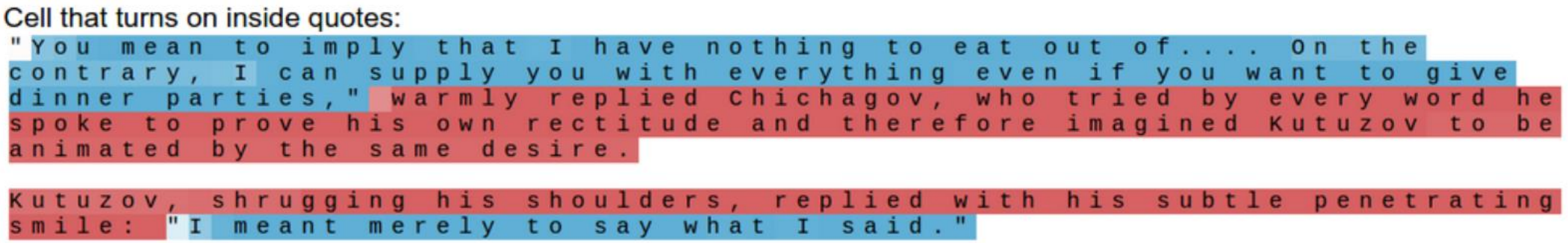
Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.



Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.



Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.

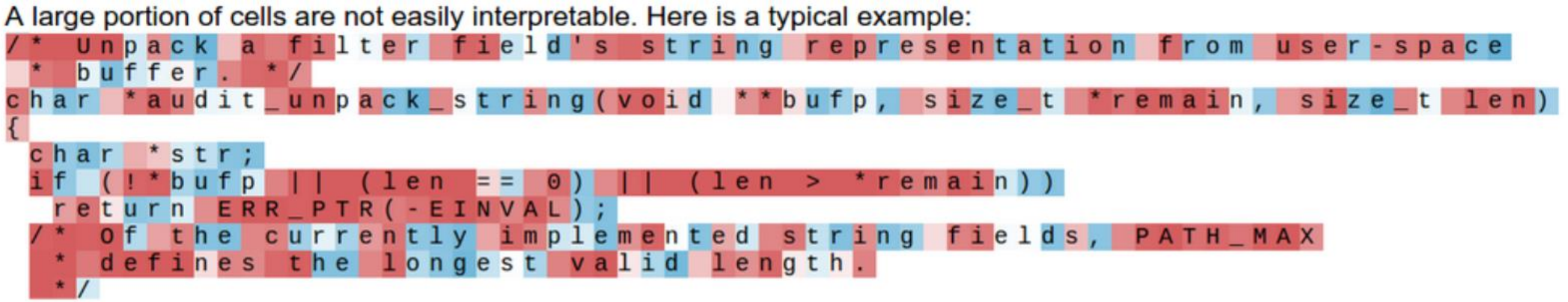
Cell that might be helpful in predicting a new line. Note that it only turns on for some ")":

```
char *audit_unpack_string(void **bufp, size_t *remain, si
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
  if (len > PATH_MAX)
    return ERR_PTR(-ENAMETOOLONG);
  str = kmalloc(len + 1, GFP_KERNEL);
  if (unlikely(!str))
    return ERR_PTR(-ENOMEM);
  memcpy(str, *bufp, len);
  str[len] = 0;
  *bufp += len;
  *remain -= len;
  return str;
}
```

Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Visualizing and Understanding Recurrent Networks

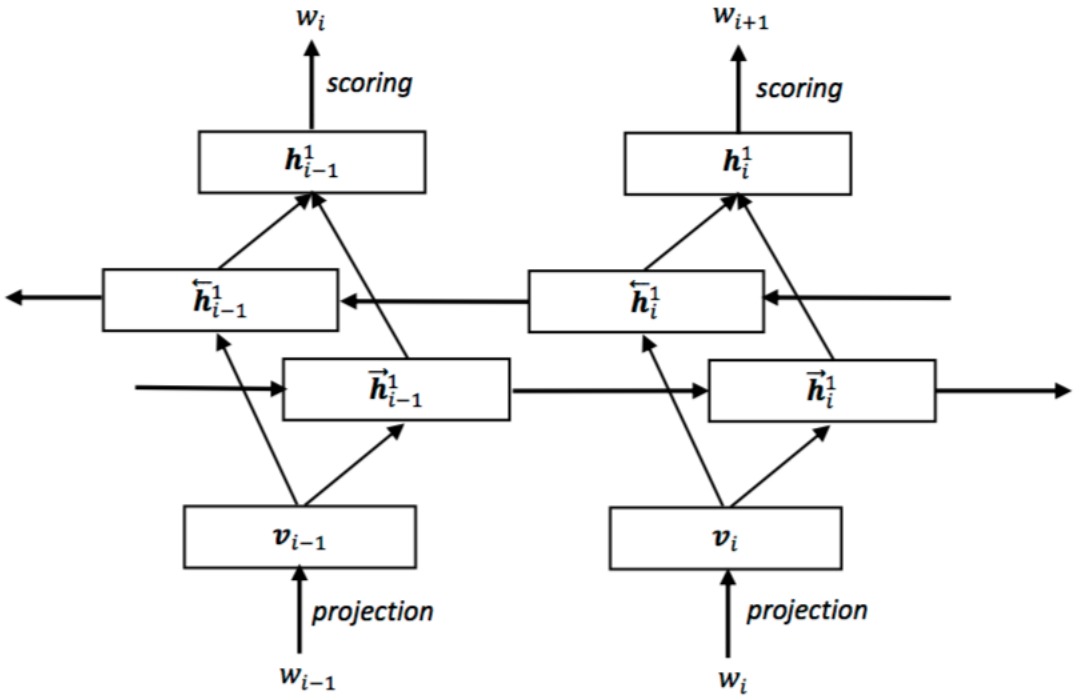
Train char-based RNN (LSTM) language model. The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN.



Karpathy et al, ICLR workshop, 2016 https://arxiv.org/pdf/1506.02078.pdf

Bi-RNN Language Models

- Can take into account both left and right contexts
- Can not be used for generation
- Then, where can it be used?



https://arxiv.org/pdf/1602.06064.pdf

CNN Language Models

Also fixed-window!
But instead of concatenation convolutional layers are used

(Of course, you can build CNN-LM with non-fixed window. Do you remember how to get a representation for a sequence of any length? It was in the previous lecture!

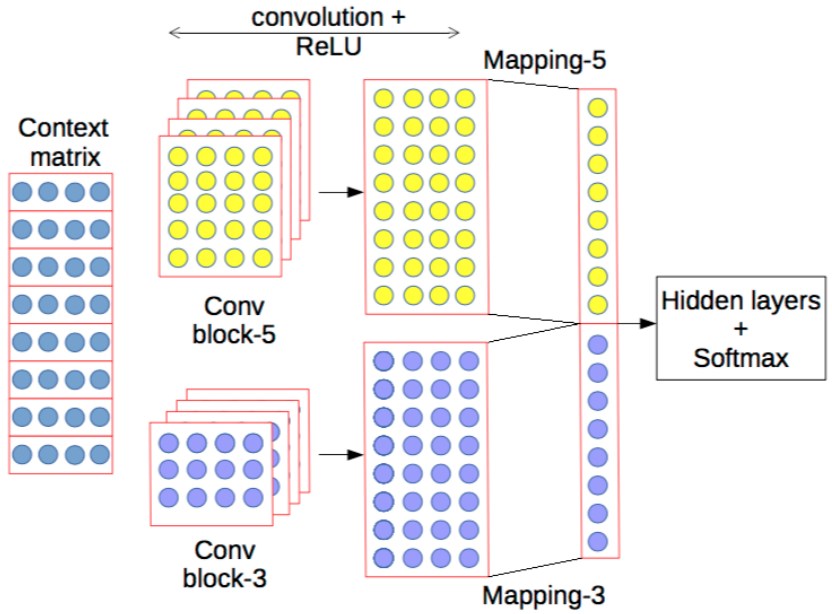


Figure 3: Combining kernels with different sizes. We concatenate the outputs of 2 convolutional blocks with kernel size of 5 and 3 respectively.

Pham et al, EMNLP 2016, https://www.aclweb.org/anthology/D16-1123

CNN Language Models

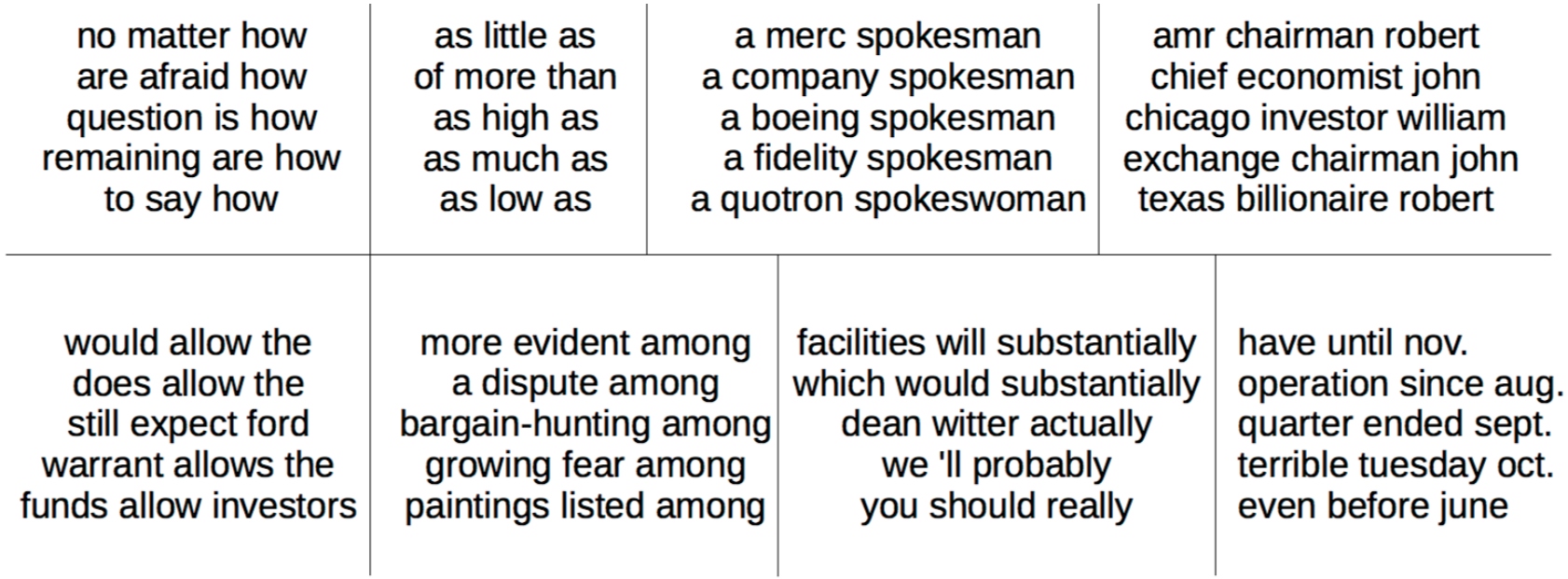


Figure 4: Some example phrases that have highest activations for 8 example kernels (each box), extracted from the validation set of the Penn Treebank. Model trained with 256 kernels for 256-dimensional word vectors.

Pham, et al, EMNLP 2016, https://www.aclweb.org/anthology/D16-1123

Training LMs to get embeddings

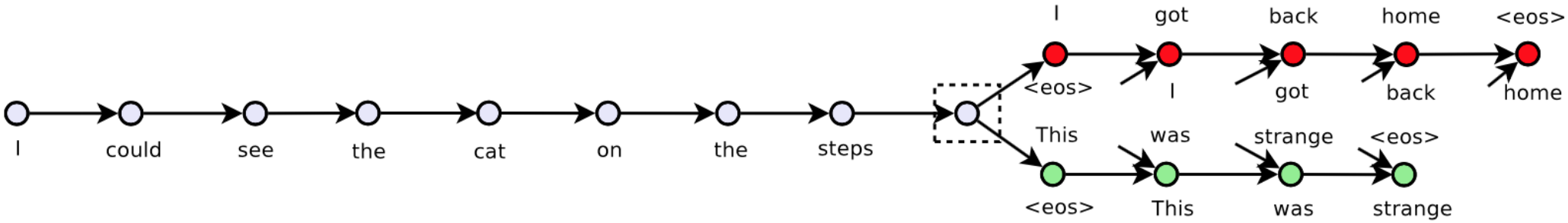
RECAP: Skip-Thought Vectors (NLM inside!)

Before:

- use a word to predict its surrounding context

Now:

- encode a sentence to predict the sentences around it



Kiros et al., NIPS 2015 https://papers.nips.cc/paper/5950-skip-thought-vectors.pdf

Deep contextualized word representations (ELMO)

- Word embedding: concat(embedding of a word, char-cnn)
- Train Bi-LSTM LM on a large corpus
- Use weighted sum of hidden representations from different layers, weights are trained with the task

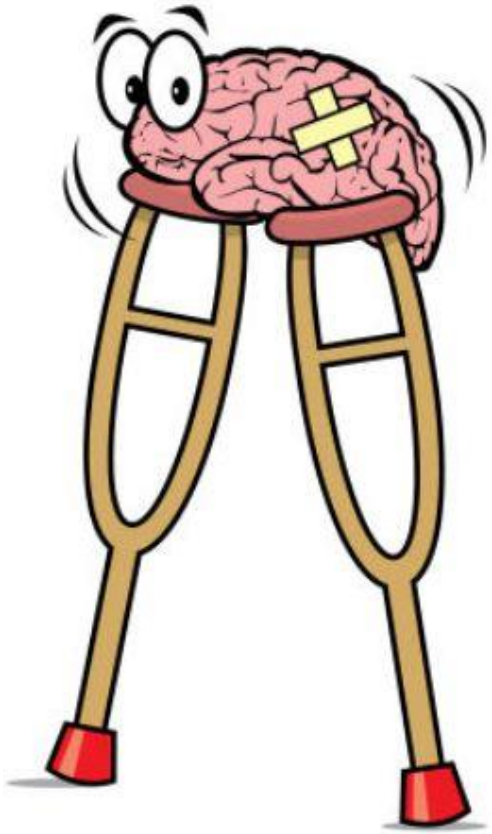
Why profit?

- char-based information used
- context of a sentence is used

(**ELMO** – **E**mbeddings from **L**anguage **MO**del)

Peters et al, NAACL 2018, http://aclweb.org/anthology/N18-1202

Hack of the day



How we trained our NLMs?

- In a sense, it's basically multi-class classification. For each training example x, the model computes probability for each label k:

$$p(k|x) = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)}$$

- Let the ground-truth distribution for for this training example be $q(k|x)$,

$$\sum_{k=1}^{K} q(k|x) = 1$$

- Loss for the example is:

$$L = -\sum_{k=1}^{K} \mathrm{q(k)} \cdot \log p(k)$$

Label smoothing

- In training we have single ground-truth label y, so that

$$q(y) = 1, q(k) = 0 \; \forall \; k \neq y \; (\text{namely}, \text{q(k)} = \delta_{k,y})$$

Szegedy et al, 2015, https://arxiv.org/pdf/1512.00567.pdf

Label smoothing

- In training we have single ground-truth label y, so that

$$q(y) = 1, q(k) = 0 \; \forall \; k \neq y \; (\text{namely}, \text{q(k)} = \delta_{k,y})$$

- **Problem**: this encourages the model to be too confident is the predictions and to have a large margin between the highest probability and the others

Szegedy et al, 2015, https://arxiv.org/pdf/1512.00567.pdf

Label smoothing

- In training we have single ground-truth label y, so that

$$q(y) = 1, q(k) = 0 \; \forall \; k \neq y \; (\text{namely}, \text{q(k)} = \delta_{k,y})$$

- **Problem**: this encourages the model to be too confident is the predictions and to have a large margin between the highest probability and the others

- **Solution:** Label smoothing! Helps encourage a model to be less confident. Consider a distribution over labels u(k), *independent of the training example* x, and a smoothing parameter ε.

$$q'(k|x) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k)$$

Szegedy et al, 2015, https://arxiv.org/pdf/1512.00567.pdf

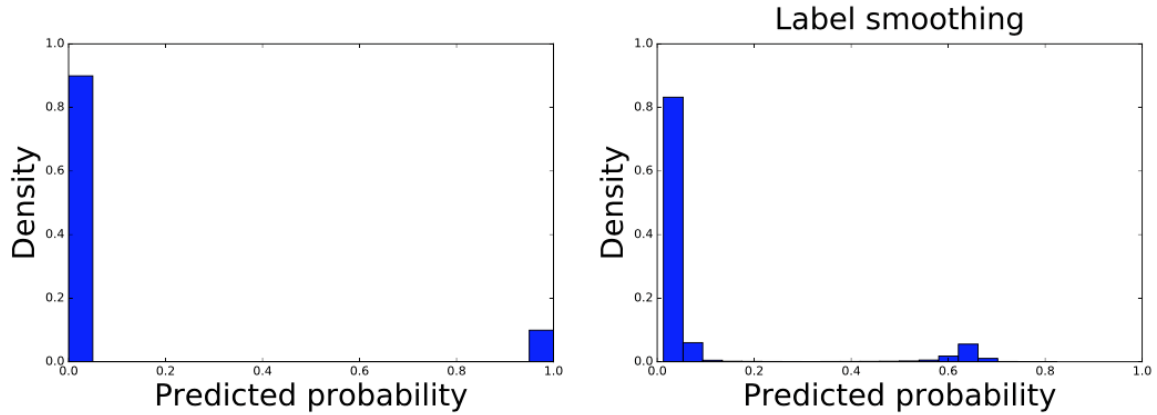
Label smoothing

- Often u(k) is uniform, then:

$$q'(k) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K}.$$

- This helps for a lot of tasks: image classification, LMs, Machine Translation, etc.

- What happens to the predictions?



That’s all for today!

Sincerely yours,
Yandex Research

That’s all for today!

Coming soon:

- Hard to keep something in mind? Use your attention!
- Don’t what you need in this world? We have an answer:
  Attention is all you need!
- And many, many cool things

Sincerely yours,
Yandex Research