Natural Language Processing with Deep Learning

Lecture 5 — Language models and word embeddings

Prof. Dr. Ivan Habernal

November 12, 2025

www.trusthlt.org
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security

Language modeling

Language modeling

‘Classical’ language models

Goal of language modeling

Assign a probability to sentences in a language

Example

“What is the probability of seeing the sentence *the lazy dog barked loudly*?”

Assigns a probability for the likelihood of given word (or a sequence of words) to follow a sequence of words

Example

“What is the probability of seeing the word *barked* after the seeing sequence *the lazy dog*?

Example

So I was at this party at Joe's place last night, but man I was so tired, so like at about midnight I said, Joe, I am __

$$\Pr(W_{k+1} = v \mid W_1 = \text{‘So’}, W_2 = \text{‘I’}, \ldots, W_k = \text{‘am’})$$

Higher $\Pr$:

- leaving
- out
- calling *(it a night)*
- gonna *(head home)*
- . . .

Lower $\Pr$:

- hungry
- in *(love)*
- . . .

Language models formally

Sequence of words $w_{1:n} = w_1 w_2 w_3 \dots w_n$ estimate

$$\Pr(w_{1:n}) = \Pr(w_1, w_2, \dots, w_n)$$

Note: We misuse notation and usually omit the RVs

$$\Pr(W_1 = w_1, W_1 = w_2, \dots, W_n = w_n)$$

We *factorize* the joint probability into a product

- One factorization is very useful: left-to-right

$$\Pr(w_{1:n}) = \Pr(w_1|\text{<s>}) \Pr(w_2|\text{<s>}, w_1) \Pr(w_3|\text{<s>}, w_1, w_2) \cdots$$
$$\cdots \Pr(w_n|\text{<s>}, w_1, w_2, \dots, w_{n-1})$$

Simplifications in ‘classical’ language models

Despite factorization, the last term of $\Pr(w_{1:n}) =$
$\Pr(w_1|\text{<s>}) \Pr(w_2|\text{<s>}, w_1) \Pr(w_3|\text{<s>}, w_1, w_2) \cdots \Pr(w_n|\text{<s>}, w_1, w_2, \ldots, w_{n-1})$
still depends on all the previous words of the sequence

k-th order markov-assumption

The next word depends only on the last k words

$$\Pr(w_i|w_{1:i-1}) \approx \Pr(w_i|w_{i-k:i-1}) \qquad \text{(inclusive indexing!)}$$

Estimating probabilities in ‘classical’ language models

Maximum Likelihood Estimation (aka. counting and dividing)

$$\hat{P}_{\mathsf{MLE}}(W_i = w | w_{i-k:i-1}) = \frac{\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1} \quad w)}{\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1})}$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{\Pr(s_i)}\right) =$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{\Pr(s_i)}\right) = \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{\Pr(s_i)}\right) =$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log \left(\frac{1}{\Pr(s_i)} \right) = \frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{1}{\Pr(s_i)} \right) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of 'sentence' s: $\Pr(s)$

Let's have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{\Pr(s_i)}\right) = \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{\Pr(s_i)}\right) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)$$

Perplexity of LM

$$2^{\text{cross-entropy}} = 2^{\left(-\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)\right)}$$

Shortcomings of n-gram language models

Shortcomings of n-gram language models

Long-range dependencies

- To capture a dependency between the next word and the word 10 positions in the past, we need to see a relevant 11-gram in the text

Shortcomings of n-gram language models

Y. Goldberg (2017). **Neural Network Methods for Natural Language Processing.** Morgan & Claypool, p. 108

Long-range dependencies

- To capture a dependency between the next word and the word 10 positions in the past, we need to see a relevant 11-gram in the text

Lack of generalization across contexts

- Having observed *black car* and *blue car* does not influence our estimates of the event *red car* if we haven't see it before

Generating text with language models

We can generate (“sample”) random sentences from the model according to their probability

1. Predict a probability distribution over the vocabulary conditioned on the start symbol <s>
2. Draw the first word from the predicted distribution
3. Predict a probability distribution over the vocabulary conditioned on the start symbol and the first word
4. Draw the second word from the predicted distribution
5. Repeat until generated *end-of-sentence* symbol </s> (or <EOS>)

Motivation for word embeddings

Using word embeddings

I give you a large corpus of plain text data

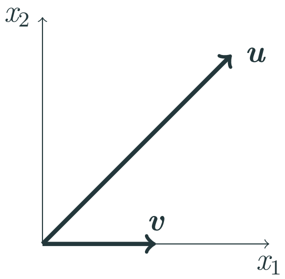
Can you build a model that will answer any of the ‘word analogy’ tasks?

- ‘Germany to Berlin is like France to ?’
- ‘Man to king is like woman to ?’

The dot product revisited

Geometry of dot product

For two n-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$

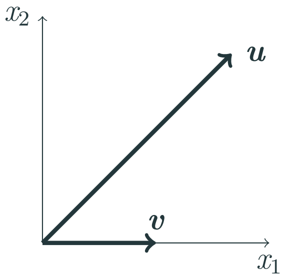


Geometry of dot product

For two n-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$

Algebraic

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}$$



Geometry of dot product

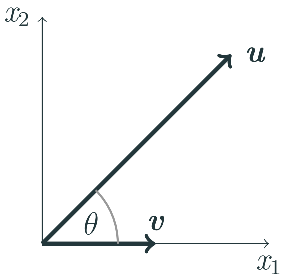
For two n-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$

Algebraic

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}$$

Geometric

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$$



Geometry of dot product

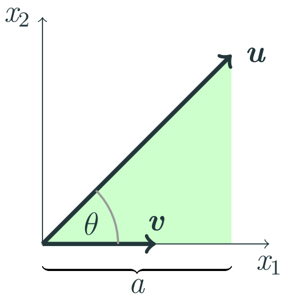
For two n-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$

Algebraic

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}$$

Geometric

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$$

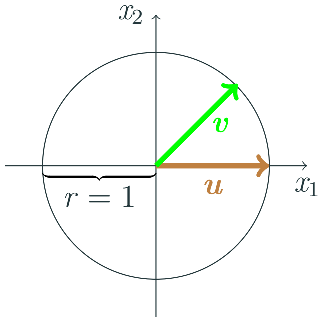


Scalar projection:

$$\implies a = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|}$$

Dot product of unit vectors (aka. cosine similarity)

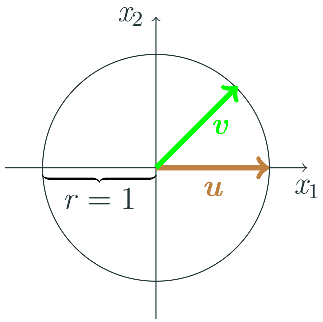
For unit vectors $\boldsymbol{u}, \boldsymbol{v}$:



Dot product of unit vectors (aka. cosine similarity)

For unit vectors $\boldsymbol{u}, \boldsymbol{v}$:

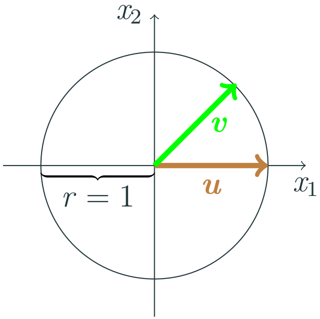
$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta) \quad \rightarrow \quad \boldsymbol{u} \cdot \boldsymbol{v} = \cos(\theta)$$

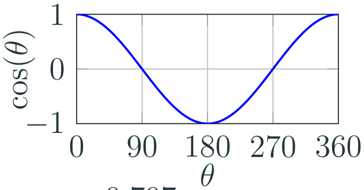


Dot product of unit vectors (aka. cosine similarity)

For unit vectors $\boldsymbol{u}, \boldsymbol{v}$:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta) \quad \rightarrow \quad \boldsymbol{u} \cdot \boldsymbol{v} = \cos(\theta)$$



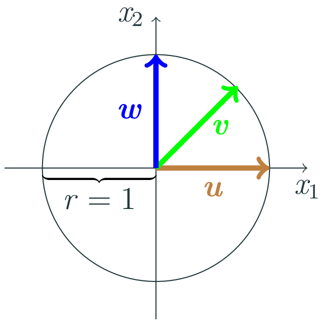


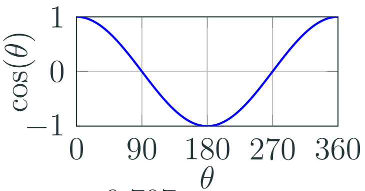
$\boldsymbol{u} \cdot \boldsymbol{v} \approx 0.707$

Dot product of unit vectors (aka. cosine similarity)

For unit vectors $\boldsymbol{u}, \boldsymbol{v}$:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta) \quad \rightarrow \quad \boldsymbol{u} \cdot \boldsymbol{v} = \cos(\theta)$$





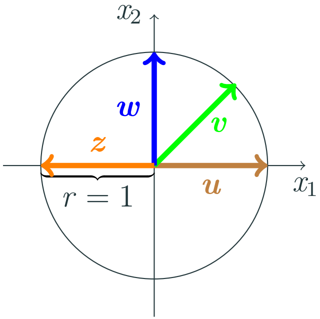
$\boldsymbol{u} \cdot \boldsymbol{v} \approx 0.707$

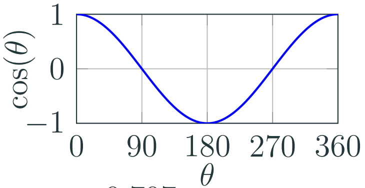
$\boldsymbol{u} \cdot \boldsymbol{w} = 0$ (orthogonal)

Dot product of unit vectors (aka. cosine similarity)

For unit vectors $\boldsymbol{u}$, $\boldsymbol{v}$:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta) \quad \rightarrow \quad \boldsymbol{u} \cdot \boldsymbol{v} = \cos(\theta)$$



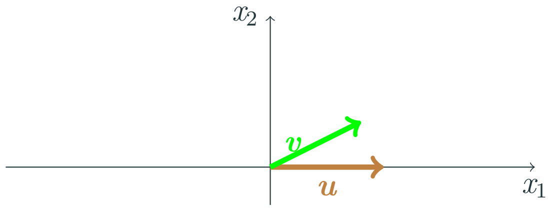


$\boldsymbol{u} \cdot \boldsymbol{v} \approx 0.707$

$\boldsymbol{u} \cdot \boldsymbol{w} = 0$ (orthogonal)

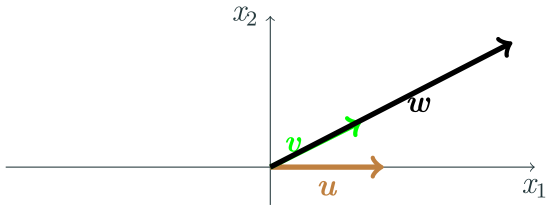
$\boldsymbol{u} \cdot \boldsymbol{z} = -1$ (least 'similar')

Dot product ($\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$) is unbounded in $\mathbb{R}$



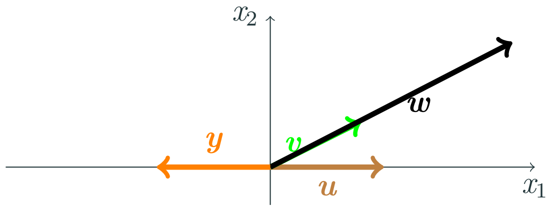
$$\boldsymbol{u} \cdot \boldsymbol{v} > 0$$

Dot product ($\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$) is unbounded in $\mathbb{R}$



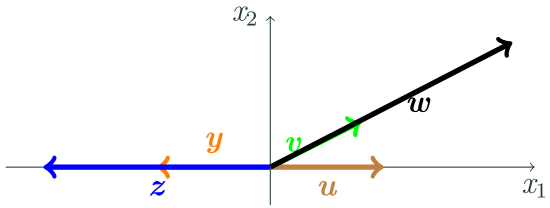
$$\boldsymbol{u} \cdot \boldsymbol{w} > \boldsymbol{u} \cdot \boldsymbol{v} > 0$$

Dot product ($\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$) is unbounded in $\mathbb{R}$



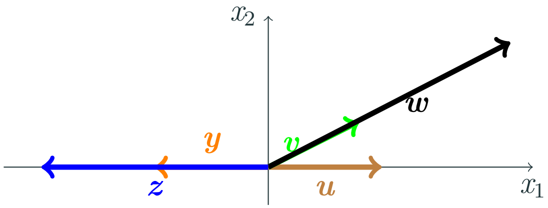
$$\boldsymbol{u} \cdot \boldsymbol{w} > \boldsymbol{u} \cdot \boldsymbol{v} > 0 > \boldsymbol{u} \cdot \boldsymbol{y}$$

Dot product ($\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$) is unbounded in $\mathbb{R}$



$$\boldsymbol{u} \cdot \boldsymbol{w} > \boldsymbol{u} \cdot \boldsymbol{v} > 0 > \boldsymbol{u} \cdot \boldsymbol{y} > \boldsymbol{u} \cdot \boldsymbol{z}$$

Dot product ($\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos(\theta)$) is unbounded in $\mathbb{R}$



$$\boldsymbol{u} \cdot \boldsymbol{w} > \boldsymbol{u} \cdot \boldsymbol{v} > 0 > \boldsymbol{u} \cdot \boldsymbol{y} > \boldsymbol{u} \cdot \boldsymbol{z}$$

Isn't it somehow related to Euclidean distance?

Dot product vs. Euclidean distance $\|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2}$

Let's take the *square* of the Euclidean distance

$$(\|\boldsymbol{u} - \boldsymbol{v}\|_2)^2 = \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2 = \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right) \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)$$

Dot product vs. Euclidean distance $\|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2}$

Let’s take the *square* of the Euclidean distance

$$
\begin{aligned}
\left(\|\boldsymbol{u} - \boldsymbol{v}\|_2\right)^2 &= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2 = \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right) \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right) \\
&= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} \boldsymbol{u}_{[i]} + \boldsymbol{v}_{[i]} \boldsymbol{v}_{[i]} - 2\boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}\right)
\end{aligned}
$$

Dot product vs. Euclidean distance $\|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2}$

Let's take the *square* of the Euclidean distance

$$
\begin{aligned}
\left(\|\boldsymbol{u} - \boldsymbol{v}\|_2\right)^2 &= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2 = \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)\left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right) \\
&= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]}\boldsymbol{u}_{[i]} + \boldsymbol{v}_{[i]}\boldsymbol{v}_{[i]} - 2\boldsymbol{u}_{[i]}\boldsymbol{v}_{[i]}\right) \\
&= \sum_{i=1}^{n} \boldsymbol{u}_{[i]}\boldsymbol{u}_{[i]} + \sum_{i=1}^{n} \boldsymbol{v}_{[i]}\boldsymbol{v}_{[i]} - 2\sum_{i=1}^{n} \boldsymbol{u}_{[i]}\boldsymbol{v}_{[i]} \\
&= \boldsymbol{u} \cdot \boldsymbol{u} + \boldsymbol{v} \cdot \boldsymbol{v} - 2\boldsymbol{u} \cdot \boldsymbol{v} \qquad \left(= 2 - 2\boldsymbol{u} \cdot \boldsymbol{v} \text{ for unit vectors}\right)
\end{aligned}
$$

Dot product vs. Euclidean distance $\|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2}$

Let's take the *square* of the Euclidean distance

Conceptual difference: if the origin shifts, the dot product changes, but the distances remains the same

$$
\begin{aligned}
\left(\|\boldsymbol{u} - \boldsymbol{v}\|_2\right)^2 &= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)^2 = \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right)\left(\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]}\right) \\
&= \sum_{i=1}^{n} \left(\boldsymbol{u}_{[i]}\boldsymbol{u}_{[i]} + \boldsymbol{v}_{[i]}\boldsymbol{v}_{[i]} - 2\boldsymbol{u}_{[i]}\boldsymbol{v}_{[i]}\right) \\
&= \sum_{i=1}^{n} \boldsymbol{u}_{[i]}\boldsymbol{u}_{[i]} + \sum_{i=1}^{n} \boldsymbol{v}_{[i]}\boldsymbol{v}_{[i]} - 2\sum_{i=1}^{n} \boldsymbol{u}_{[i]}\boldsymbol{v}_{[i]} \\
&= \boldsymbol{u} \cdot \boldsymbol{u} + \boldsymbol{v} \cdot \boldsymbol{v} - 2\boldsymbol{u} \cdot \boldsymbol{v} \qquad \left(= 2 - 2\boldsymbol{u} \cdot \boldsymbol{v} \text{ for unit vectors}\right)
\end{aligned}
$$

$\rightarrow$ Minimizing (square) euclidean distance is *proportional* to maximizing cosine similarity (equivalent for unit vectors)

Recap: What was the task of a language model again?

LM as a conditional probability predictor

Given a sequence of k words, return a probability distribution over V (the vocabulary) for the next possible word

In detail, compute

$$\Pr(W_{k+1} = v \mid W_1, W_2, \ldots, W_k)$$

for all possible words v from vocabulary V

LM looks like... a text classification task?!

Text classification

Given a text,* predict **its label** from a set of arbitrarily ordered items (**classes such as {sports, politics, travel}**). We model cond. probability for each **label** (a probability distribution over **all the classes**), for example by softmax.

LM looks like... a text classification task?!

Text classification

Given a text,* predict **its label** from a set of arbitrarily ordered items (**classes such as {sports, politics, travel}**). We model cond. probability for each **label** (a probability distribution over **all the classes**), for example by softmax.

LM

Given a text,* predict **the next word** from a set of arbitrarily ordered items (**the vocabulary**). We model cond. probability for each **word** (a probability distribution over **the entire vocabulary**), for example by softmax.

* We already know how to turn text into a feature vector, e.g., by bag of words

Neural language models

Neural LMs

LM as 'text classification' again

Given a text,* predict **the next word** from a set of arbitrarily ordered items (**the vocabulary**). We model cond. probability for each **word** (a probability distribution over **the entire vocabulary**), for example by softmax.

Let's build a neural network to solve this task

Neural LMs

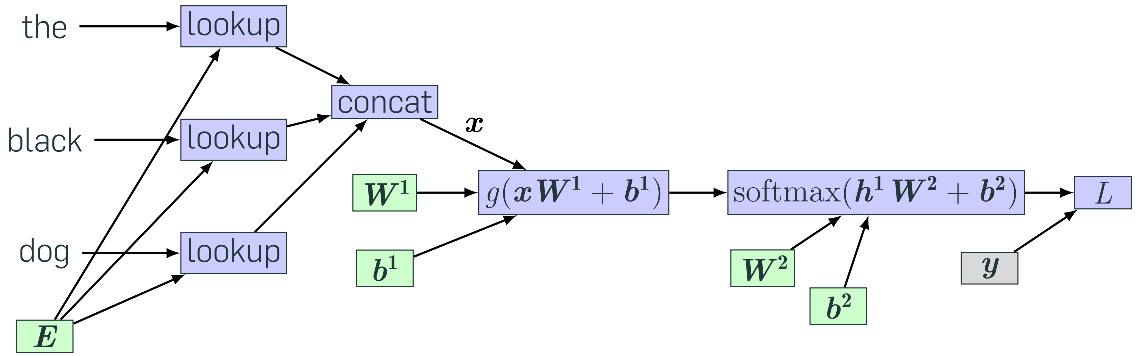
LM as 'text classification' again

Given a text,* predict **the next word** from a set of arbitrarily ordered items (**the vocabulary**). We model cond. probability for each **word** (a probability distribution over **the entire vocabulary**), for example by softmax.

Let's build a neural network to solve this task

- Input: a k-gram of words $w_{1:k}$
- Desired output: a conditional probability distribution over the vocabulary V for the next word w_{k+1}

Network concatenating 3 words as embeddings ($d_w = 50$)



Each word $\in \mathbb{R}^{|V|}$ (one hot), $\boldsymbol{E} \in \mathbb{R}^{|V| \times 50}$, each lookup output $\in \mathbb{R}^{50}$, concat output $\boldsymbol{x} \in \mathbb{R}^{150}$

Embedding layer

If the input are symbolic **categorical features**

- e.g., words from a closed vocabulary

it is common to associate each possible feature value

- i.e., each word in the vocabulary

with a d-dimensional vector for some d

These vectors are also *parameters* of the model, and are trained jointly with the other parameters

Embedding layer: Lookup operation

Mapping from a symbolic feature such as `word-number-48` to d-dimensional vectors is performed by an embedding layer (a lookup layer)

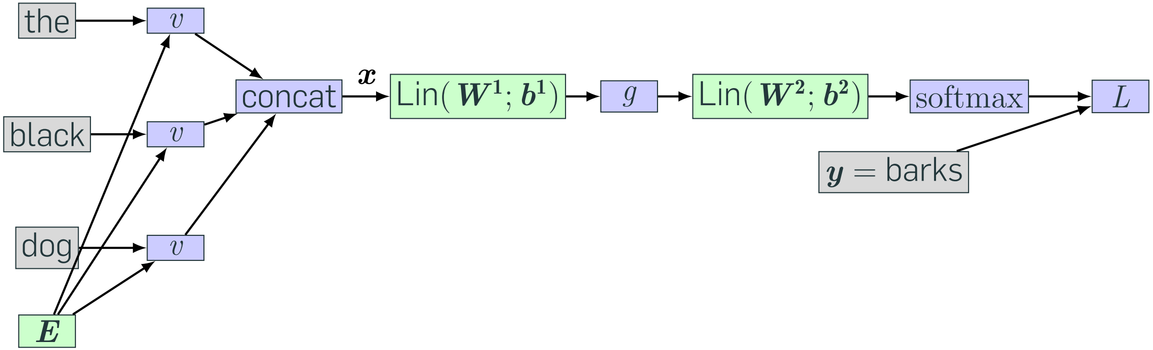
The parameters in an embedding layer is a matrix $\boldsymbol{E}^{|V| \times d}$, each row corresponds to a different word in the vocabulary

The lookup operation is then indexing $v(w)$, e.g.,

$$v(w) = v_{48} = \boldsymbol{E}_{[48,:]}$$

If the symbolic feature is a one-hot vector $\boldsymbol{x}$, the lookup operation can be implemented as the multiplication $\boldsymbol{x}\boldsymbol{E}$

Simplify notation: Lookup as v, linear layer with params



the, black, dog, $\boldsymbol{y}$ = one-hot encoding, $\mathbb{R}^{|V|}$

Loss L and gold label $\boldsymbol{y}$ — only for training

Training neural LMs

Where to get training examples?

Training examples are simply word k-grams from an unlabeled corpus

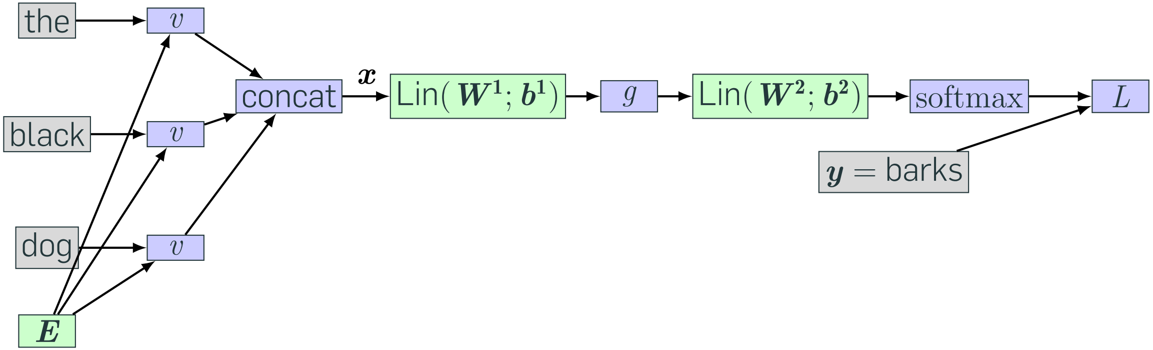
- Identities of the first $k-1$ words are used as features
- The last word is used as the target label for the classification

The model is trained using cross-entropy loss

Some advantages of neural LMs

$\approx$ linear increase in parameters with $k + 1$ (better than 'classical' LMs)

Learned word representations as a by-product



Each row of $\boldsymbol{E}$ learns a word representation

Learning word embeddings

Learning word embeddings

Distributional hypothesis

Recall: One-hot encoding of words

Major drawbacks?

- No ‘semantic’ similarity, all words are equally ‘similar’

Example (see Lecture 3 for more)

$$V = \begin{pmatrix} \text{a}_1 & \text{abandon}_2 & \dots & \text{zone}_{2,999} & \text{zoo}_{3,000} \end{pmatrix}$$

$$\text{nice} = \begin{pmatrix} 0_1 & \dots & 1_{1,852} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$
$$\text{pleasant} = \begin{pmatrix} 0_1 & \dots & 1_{2,012} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$
$$\text{horrible} = \begin{pmatrix} 0_1 & \dots & 1_{696} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

Distributional hypothesis

The distributional hypothesis stating that *words are similar if they appear in similar contexts*

Example

Intuitively, when we encounter a sentence with an unknown word such as the word **wampinuk** in

Distributional hypothesis

The distributional hypothesis stating that *words are similar if they appear in similar contexts*

Example

Intuitively, when we encounter a sentence with an unknown word such as the word **wampinuk** in

Marco saw a hairy little **wampinuk** *crouching behind a tree*

We infer the meaning of the word based on the context in which it occurs

From neural language models to training word embeddings

(Neural) language model's goal: Predict probability distribution over $|V|$ for the next word conditioned on the previous words

- Side product: Can learn useful word embeddings

From neural language models to training word embeddings

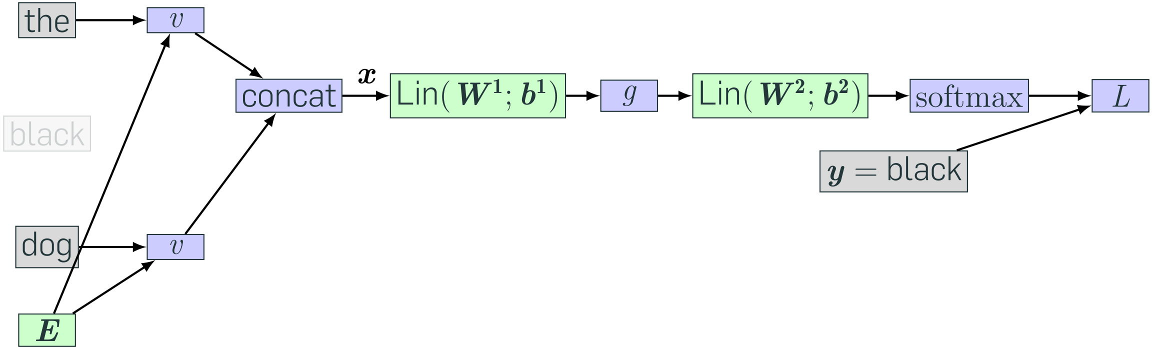
(Neural) language model's goal: Predict probability distribution over $|V|$ for the next word conditioned on the previous words

- Side product: Can learn useful word embeddings

What if we don't need probability distribution but just want to learn word embeddings?

- We can relax our Markov assumption of 'look at k previous words only'
- We can get rid of the costly normalization in softmax

Simplification 1: Ditch the Markov property — look into the future!



For example, instead of modeling $\Pr(w_3 | w_1, w_2, \boxdot)$, we model $\Pr(w_2 | w_1, \boxdot, w_3)$

Simplification 2: Give up the costly probability distribution

Instead of predicting probability distribution over the entire vocabulary, we just want to predict some score of **context** and **target word**

What could such a score be?

Simplification 2: Give up the costly probability distribution

Instead of predicting probability distribution over the entire vocabulary, we just want to predict some score of **context** and **target word**

What could such a score be?

- Prefer words in their true contexts (high score)

Simplification 2: Give up the costly probability distribution

Instead of predicting probability distribution over the entire vocabulary, we just want to predict some score of **context** and **target word**

What could such a score be?

- Prefer words in their true contexts (high score)
- Penalize words in their 'untrue' contexts (low score)

Negative sampling

Instead of predicting probability distribution for the target word, we **create an artificial binary task** by choosing either the correct or a random incorrect target word

Negative sampling

Instead of predicting probability distribution for the target word, we **create an artificial binary task** by choosing either the correct or a random incorrect target word

$$y = \begin{cases} 1 & \text{if } (w, c_{1:k}) \text{ is a positive example from the corpus} \\ 0 & \text{if } (w', c_{1:k}) \text{ is a negative example from the corpus} \end{cases}$$

Negative sampling

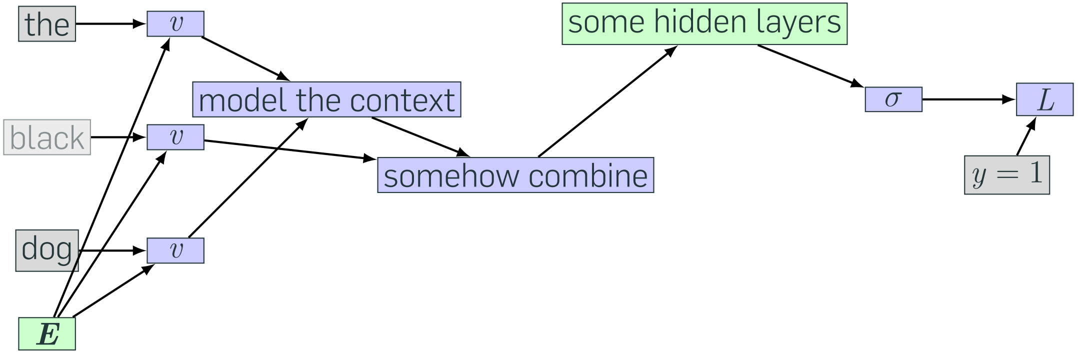
Instead of predicting probability distribution for the target word, we **create an artificial binary task** by choosing either the correct or a random incorrect target word

$$y = \begin{cases} 1 & \text{if } (w, c_{1:k}) \text{ is a positive example from the corpus} \\ 0 & \text{if } (w', c_{1:k}) \text{ is a negative example from the corpus} \end{cases}$$

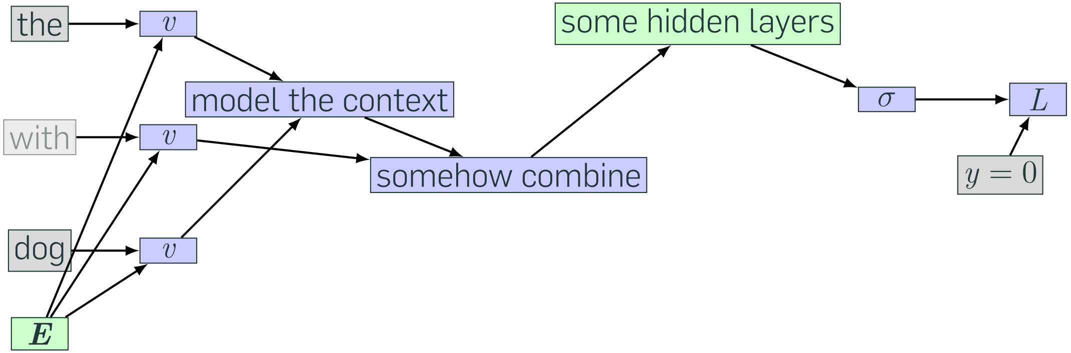
Which distribution for sampling w' from V?

- Corpus-based frequency: $\frac{\#(v)}{\sum_{v' \in V} \#(v')}$ (pr. of each word v)
- Re-weighted: $\frac{\#(v)^{0.75}}{\sum_{v' \in V} \#(v')^{0.75}}$ (more weight on less frequent words done in word2vec)

Turn the problem into binary classification (positive example)



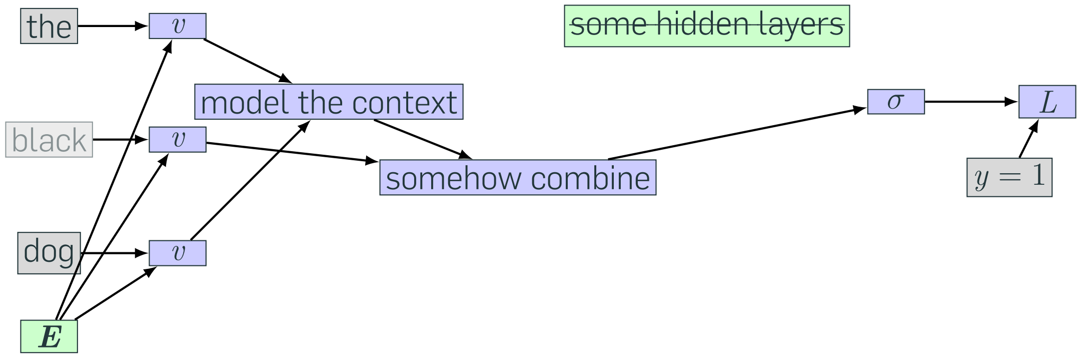
Turn the problem into binary classification (negative example)



word2vec

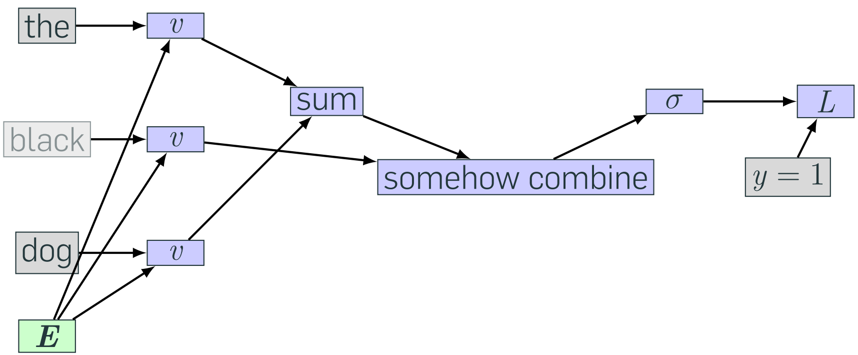
word2vec

word2vec simplifies the neural LM by removing the hidden layer (so turning it into a log-linear model!)



word2vec — how to model the context?

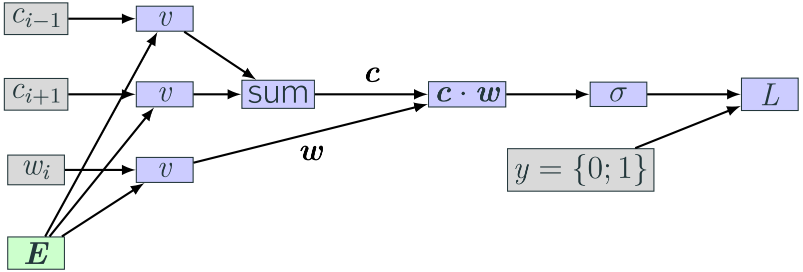
word2vec — how to model the context?



T. Mikolov, K. Chen, G. Corrado, and J. Dean (2013). **"Efficient estimation of word representations in vector space".** In: *1st International Conference on Learning Representations ICLR, Workshop Track Proceedings*. Ed. by Y. Bengio and Y. LeCun, pp. 1–12

Variant 1: Continuous bag of words (CBOW) $\boldsymbol{c} = \sum_{i=1}^{k} v(c_i)$

Final CBOW word2vec — similarity score is the dot product



w_i — target word, c_{i-1}, c_{i+1} — context words (one-hot)

$y = 1$ for correct word-context pairs, $y = 0$ for random w_i

The only learnable parameter is the embedding matrix $\boldsymbol{E}$

What is $\sigma(\boldsymbol{c} \cdot \boldsymbol{w})$ doing?

word2vec: Learning useful word embeddings

Train the network to distinguish 'good' word-context pairs from 'bad' ones

Create a set D of correct word-context pairs and set $\bar{D}$ of incorrect word-context pairs

The goal of the algorithm is to estimate the probability $\Pr(D = 1 \mid w, c)$ that the word-context pair w, c comes from the correct set D

This should be high $(\to 1)$ for pairs from D and low $(\to 0)$ for pairs from $\bar{D}$

Advantages and limitations of words embeddings

Using word embeddings

Pre-trained embeddings: 'Semantic' input to any neural network instead of one-hot word encoding

- Instance of **transfer learning** — pre-trained (self-trained) on an auxiliary task, plugged into a more complex model as pre-trained weights

Example: Represent a document as an average of its words' embeddings (average bag-of-words through embeddings) for text classification

Side note: word2vec and word embeddings $\rightarrow$ part of the deep-learning revolution in NLP around 2015

Semantic similarity, short document similarity, query expansion

S. Kuzi, A. Shtok, and O. Kurland (2016). **“Query Expansion Using Word Embeddings”.** In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, pp. 1929–1932

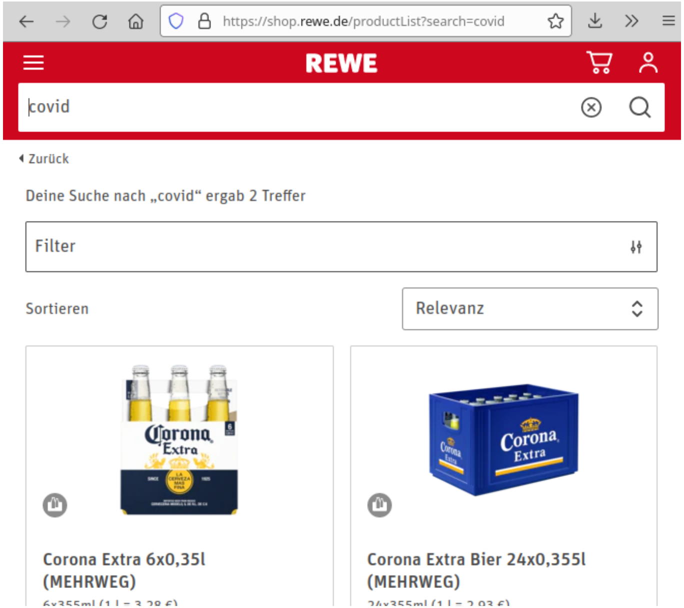
“Using Word2Vec’s CBOW embedding approach, applied over the entire corpus on which search is performed, we select terms that are semantically related to the query.”

Semantic similarity, short document similarity, query expansion

S. Kuzi, A. Shtok, and O. Kurland (2016). **"Query Expansion Using Word Embeddings".** In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, pp. 1929–1932

"Using Word2Vec's CBOW embedding approach, applied over the entire corpus on which search is performed, we select terms that are semantically related to the query."

What can possibly go wrong?


https://shop.rewe.de/productList?search=covid

REWE

covid

Zurück

Deine Suche nach „covid“ ergab 2 Treffer

Filter

Sortieren Relevanz

Corona Extra 6x0,35l (MEHRWEG)

6x355ml (1 l = 3,28 €)

Corona Extra Bier 24x0,355l (MEHRWEG)

24x355ml (1 l = 2,93 €)

Searched for **covid** (test), returned the closest items with **corona** in the title (because their embeddings learned that covid ≈ corona).

Query expansion with word embeddings might be tricky

Mining word analogies with word2vec

'Germany to Berlin is France to ?'

Solved by $v(\text{Berlin}) - v(\text{Germany}) + v(\text{France})$, outputs vector $\boldsymbol{x}$ which is closest to Paris in the embeddings space (the closest row in $\boldsymbol{E}$)

Find the queen

$v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$

Limitations of word embeddings (1)

Definition of similarity

Completely operational: words are similar if used in similar contexts

Antonyms

Words opposite of each other (buy—sell, hot—cold) tend to appear in similar contexts (things that can be hot can also be cold, things that are bought are often sold)

Models might tend to judge antonyms as very similar to each other

Limitations of word embeddings (2)

Biases

Distributional methods reflect the usage patterns in the corpora on which they are based

The corpora reflect human biases in the real world (cultural or otherwise)

“Word embeddings encode not only stereotyped biases but also other knowledge [..] are problematic as toward race or gender, or even simply veridical, reflecting the status quo distribution of gender with respect to careers or first names.”

A. Caliskan, J. J. Bryson, and A. Narayanan (Apr. 2017). **“Semantics derived automatically from language corpora contain human-like biases”.** In: *Science* 356 (6334), pp. 183–186

Limitations of word embeddings (3)

Polysemy, context independent representation

Some words have obvious multiple senses

A *bank* may refer to a financial institution or to the side of a river, a *star* may be an abstract shape, a celebrity, an astronomical entity

Using a single vector for all forms is problematic

License and credits

Credits

Ivan Habernal

FastText embeddings: Sub-word embeddings

FastText embeddings

Popular word embedding models ignore the morphology of words, by assigning a distinct vector to each word.

- Limitation, especially for languages with large vocabularies and many rare words

FastText embeddings

Popular word embedding models ignore the morphology of words, by assigning a distinct vector to each word.

- Limitation, especially for languages with large vocabularies and many rare words

$\rightarrow$ Model each word as a bag of character n-grams

- Each character n-gram has own embedding
- Word is represented as a sum of n-gram embeddings

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov (2017). **"Enriching Word Vectors with Subword Information".** In: *Transactions of the ACL* 5, pp. 135–146

FastText embeddings example

Extract all the character n-grams for $3 \leq n \leq 6$

Example

eating $\rightarrow G_w = \{$ <ea, eat, ati, tin, ing, ng>, <eat, eati, atin, ting, ing>, <eati, eatin, ating, ting>, <eatin, eating, ating> $\}$

$$v(\text{eating}) = \sum_{g \in G_w} v(G_w)$$

Train with skip-gram and negative sampling (same as word2vec)

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov (2017). **"Enriching Word Vectors with Subword Information".** In: *Transactions of the ACL* 5, pp. 135–146

Appendix: Probability refresher

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\mathrm{the}, \mathrm{be}, \mathrm{to}, \mathrm{of}, \mathrm{and}, \ \ldots\}$, we assume a fixed vocabulary

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the}, \text{be}, \text{to}, \text{of}, \text{and}, \ldots\}$, we assume a fixed vocabulary

Probability distribution over random variables

For example, probability of *'the'* at position 1

$$\Pr(W_1 = w_1) = \Pr(W_1 = \text{the}) = 0.00024$$

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the}, \text{be}, \text{to}, \text{of}, \text{and}, \ldots\}$, we assume a fixed vocabulary

Probability distribution over random variables

For example, probability of *'the'* at position 1

$\Pr(W_1 = w_1) = \Pr(W_1 = \text{the}) = 0.00024$

Notation shortcuts: $\Pr(W_1 = w_1) \to P(W_1)$, $P(\text{the})$, etc.

Probability refresher 2

Joint probability

For example, probability of *'the'* at position 1 and *'cat'* at position 2

$$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$$

Probability refresher 2

Joint probability

For example, probability of *‘the’* at position 1 and *‘cat’* at position 2

$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$

Notation shortcuts: $P(W_1, W_2) = P(W_2, W_1)$

Probability refresher 2

Joint probability

For example, probability of *'the'* at position 1 and *'cat'* at position 2

$$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$$

Notation shortcuts: $P(W_1, W_2) = P(W_2, W_1)$

Conditional probability

For example, probability of *'cat'* at position 2, **given** *'the'* at position 1

$$\Pr(W_2 = \text{cat} | W_1 = \text{the}) = \frac{P(W_1, W_2)}{P(W_1)}$$

Probability refresher 3

Independence

Two random variables X, Y are **independent** if and only if

$$P(X, Y) = P(X) \cdot P(Y)$$

Probability refresher 3

Independence

Two random variables X, Y are **independent** if and only if

$$P(X, Y) = P(X) \cdot P(Y)$$

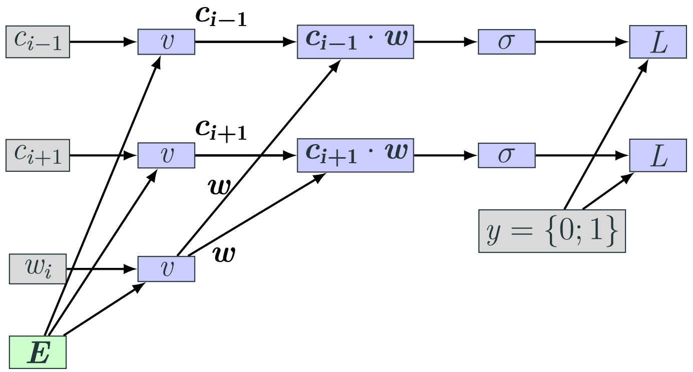
Conditional independence

Two random variables X, Y are **conditionally independent** given Z if and only if

$$P(X, Y|Z) = P(X|Z) \cdot P(Y|Z)$$

Appendix: Skip-Gram in word2vec

Skip-Gram — even more relaxed context notion



For k-element context $c_{1:k}$, treat as k different independent contexts (w_i, c_i)

Choosing the context

Sliding window approach — CBOW

Auxiliary tasks are created by taking a sequence of $2m + 1$ words

- The middle word is the target (focus) word
- The m words to each side is the context

Sliding window approach — Skip-Gram

$2m$ distinct tasks are created, each pairing the focus word with a different context word

Skip-gram-based approaches shown to be robust and efficient to train