Natural Language Processing with Deep Learning

Lecture 5 — Text generation 1: Language models and word embeddings

Prof. Dr. Ivan Habernal

November 10, 2023

PADERBORN UNIVERSITY

Natural Language Processing Group
Paderborn University
We focus on Trustworthy Human Language Technologies

www.trusthlt.org

Finishing previous lecture: Stacking layers, non-linearity

Finishing previous lecture: Stacking layers, non-linearity

Language models

- Probability refresher
- ‘Classical’ language models
- Neural language models

Word embeddings

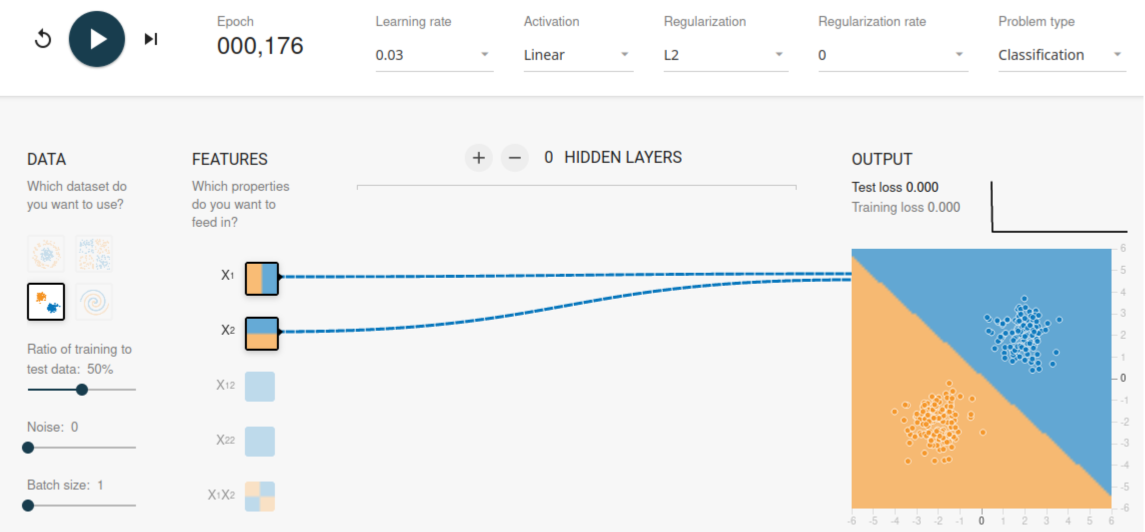


Figure 1: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

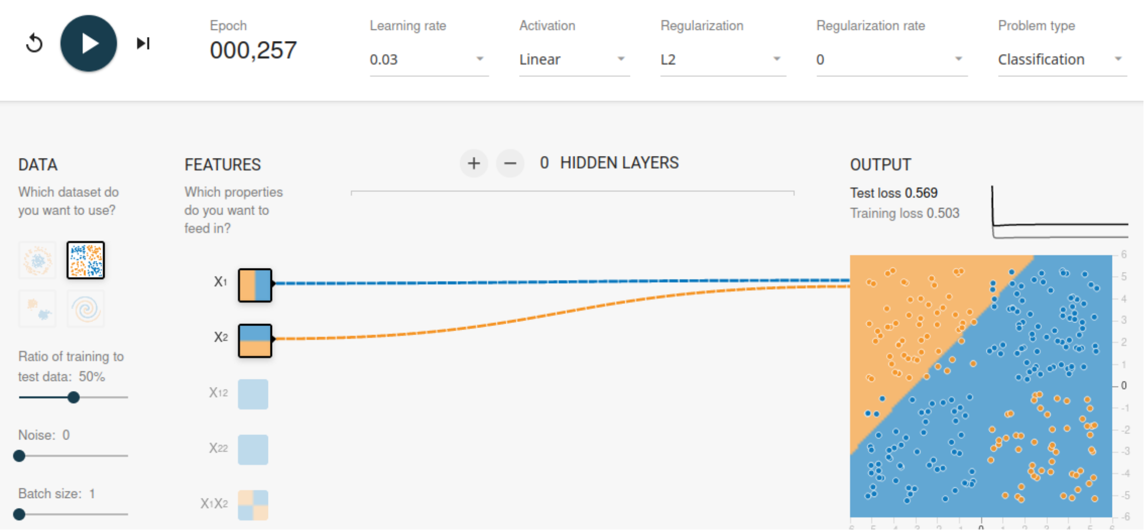


Figure 2: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

Stacking linear layers on top of each other — still linear!

$\boldsymbol{x} \in \mathbb{R}^{d_{in}} \qquad \boldsymbol{W^1} \in \mathbb{R}^{d_{in} \times d_1} \qquad \boldsymbol{b^1} \in \mathbb{R}^{d_1} \qquad \boldsymbol{W^2} \in \mathbb{R}^{d_1 \times d_{out}} \qquad \boldsymbol{b^2} \in \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) = \left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right)\, \boldsymbol{W}^2 + \boldsymbol{b}^2$$

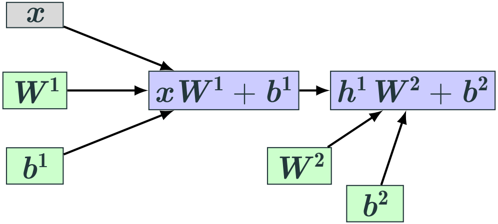


Figure 3: Computational graph; green boxes are trainable parameters, gray are constant inputs

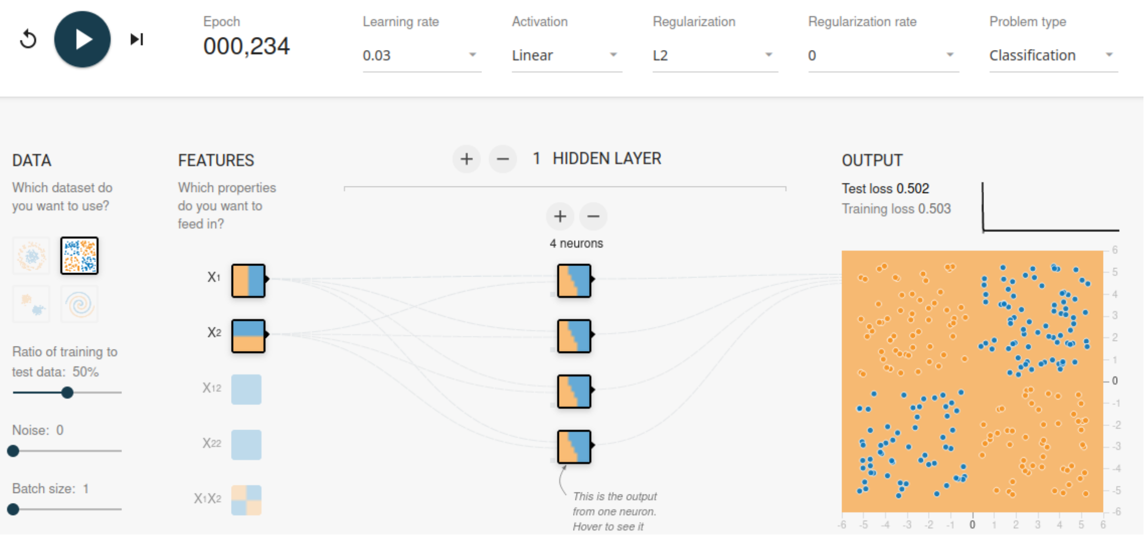


Figure 4: Linear hidden layers do not help (`http://playground.tensorflow.org`)

Adding non-linear function $g : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$

$$f(\boldsymbol{x}) = g\left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right)\ \boldsymbol{W}^2 + \boldsymbol{b}^2$$

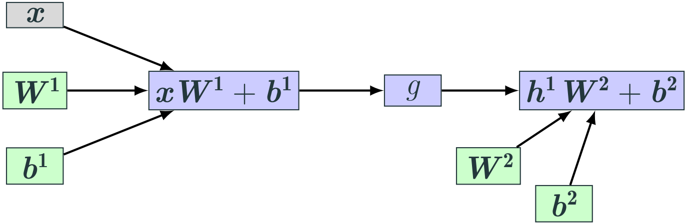


Figure 5: Computational graph; green boxes are trainable parameters, gray are constant inputs

Non-linear function g: Rectified linear unit (ReLU) activation

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

or $\quad \text{ReLU}(z) = \max(0, x)$

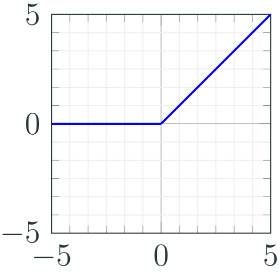


Figure 6: ReLU function

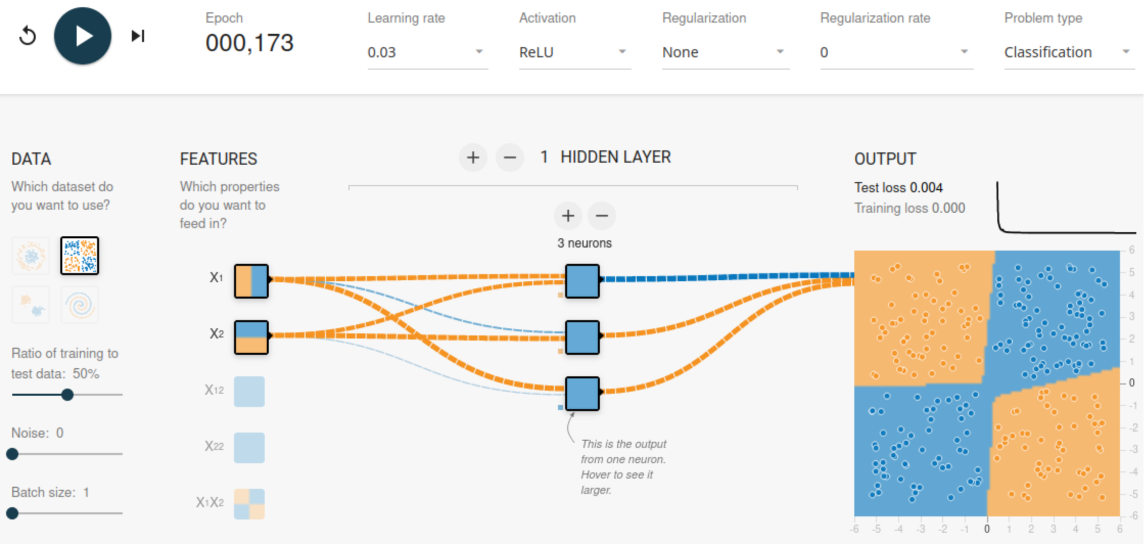


Figure 7: XOR solvable with, e.g., ReLU (`http://playground.tensorflow.org`)

XOR example in super-simplified sentiment classification

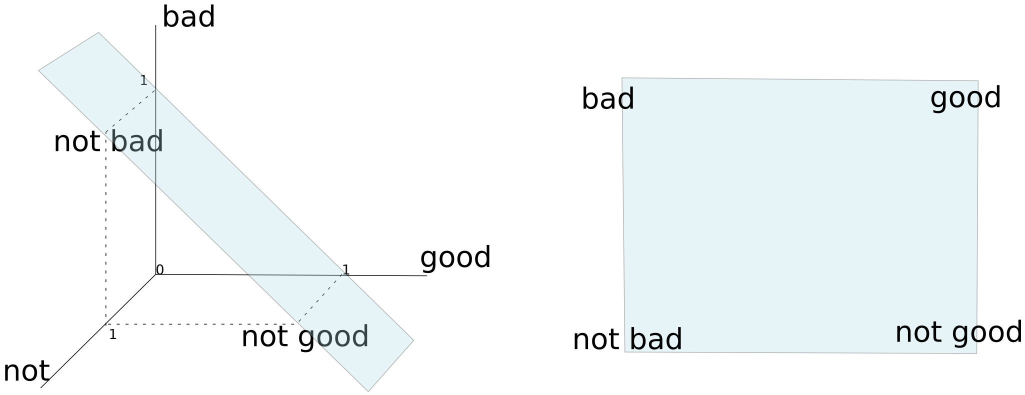


Figure 8: $V = \{\text{not}, \text{bad}, \text{good}\}$, binary features $\in \{0, 1\}$

Multi-layer perceptron (MLP)

$$f(\boldsymbol{x}) = \sigma\left(g\left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right)\,\boldsymbol{W}^2 + \boldsymbol{b}^2\right)$$

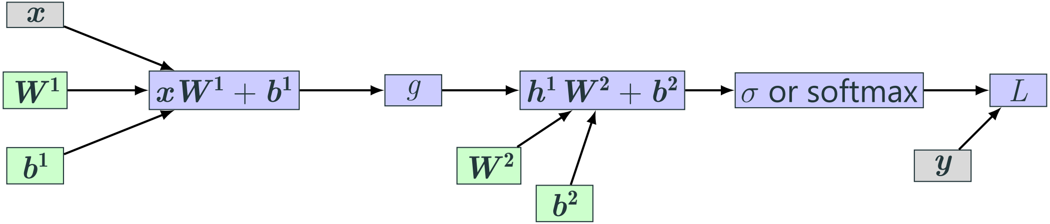


Figure 9: Computational graph; green boxes are trainable parameters, gray are constant inputs

Today: Language models and word embeddings

Finishing previous lecture: Stacking layers, non-linearity

Language models

- Probability refresher
- ‘Classical’ language models
- Neural language models

Word embeddings

Language models

Finishing previous lecture: Stacking layers, non-linearity

Language models

- Probability refresher
- 'Classical' language models
- Neural language models

Word embeddings

Language models

Probability refresher

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 refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the, be, to, of, and, } \ldots\}$, 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

$W_1 \in \{\text{the, be, to, of, and, } \ldots\}$, 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: $\Pr(W_1 = w_1) \rightarrow P(W_1), P(\text{the})$, 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

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

Language models

‘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*?

Language models formally

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

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

Note: We misuse notation and usually omit the RVs

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

We *factorize* the joint probability into a product

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

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

Simplifications in ‘classical’ language models

Despite factorization, the last term of $\Pr(w_{1:n}) =$
$\Pr(w_1 | \text{<s>}) \Pr(w_2 | \text{<s>}, w_1) \Pr(w_3 | \text{<s>}, w_1, w_2) \cdots \Pr(w_k | \text{<s>}, w_1, w_2, \ldots, w_{n-1})$
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

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

Simplifications in ‘classical’ language models

Despite factorization, the last term of $\Pr(w_{1:n}) =$
$\Pr(w_1|\text{<s>})\Pr(w_2|\text{<s>}, w_1)\Pr(w_3|\text{<s>}, w_1, w_2)\cdots\Pr(w_k|\text{<s>}, w_1, w_2, \ldots, w_{n-1})$
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

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

<s>_0 The_1 cat_2 sat_3 on_4 the_5 w_6

$i = 6, k = 2 \rightarrow \Pr(w_i|w_{1:i-1}) \approx \Pr(w_6|W_4 = \text{on}, W_5 = \text{the})$

Estimating probabilities in 'classical' language models

Maximum Likelihood Estimation (aka. counting and dividing)

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

What if $\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1}) = 0$**?**

Estimating probabilities in ‘classical’ language models

Maximum Likelihood Estimation (aka. counting and dividing)

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

What if $\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1}) = 0$**?**

- Add-alpha smoothing ($0 \leq \alpha \leq 1$)

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

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** (last lecture!) 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** (last lecture!) 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** (last lecture!) 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** (last lecture!) 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

Language models

Neural language models

Neural LMs

Let's build a neural network

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

Embedding layer once again (recall last lecture)

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

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

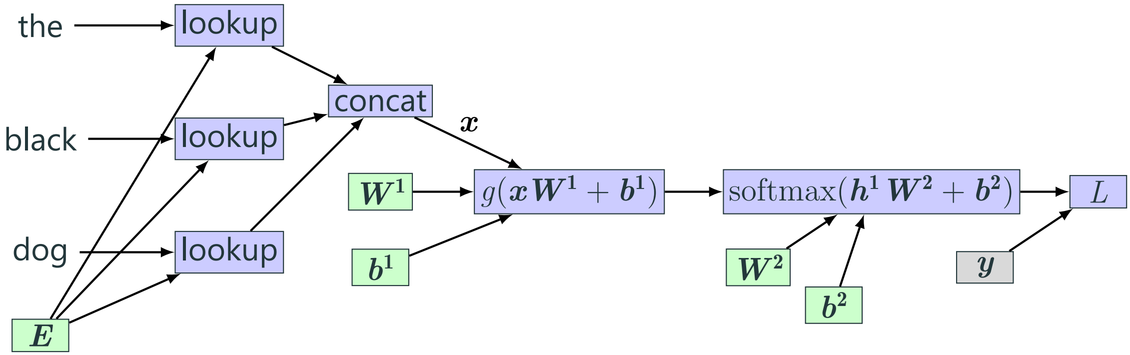
The parameters in an embedding layer are a matrix $\boldsymbol{W}^{|V| \times d}$, each row corresponds to a different word in the vocabulary

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

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

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

Example 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 $\boldsymbol{x} \in \mathbb{R}^{150}$

Neural LMs

Let's build a neural network

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

Each input word w_k is associated with an embedding vector $v(w) \in \mathbb{R}^{d_w}$ (d_w — word embedding dimensionality)

Input vector $\boldsymbol{x}$ is a concatenation of k words

$$\boldsymbol{x} = [v(w_1); v(w_2); \ldots; v(w_k)]$$

Neural LMs

MLP with one (or more) hidden layers

$$
\begin{aligned}
v(w) &= \boldsymbol{E}_{w,:} \\
\boldsymbol{x} &= [v(w_1); v(w_2); \ldots; v(w_k)] \\
\boldsymbol{h} &= g(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}) \\
\hat{\boldsymbol{y}} = \Pr(W_i | w_{1:k}) &= \operatorname{softmax}(\boldsymbol{h}\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}})
\end{aligned}
$$

Output dimension: $\hat{\boldsymbol{y}} \in \mathbb{R}^{|V|}$

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 and limitations of neural LMs

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

- The size of the output vocabulary affects the computation time
- The softmax at the output layer requires an expensive matrix-vector multiplication with the matrix $\boldsymbol{W^2} \in \mathbb{R}^{d_{\text{hid}} \times |V|}$, followed by $|V|$ exponentiations

Solutions: Hierarchical softmax, noise-contrastive estimation

Generating text with language models

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

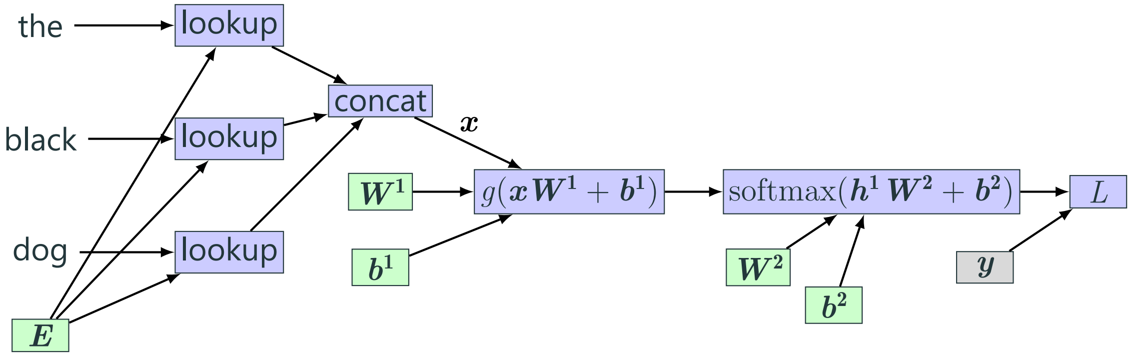
1. Predict a probability distribution over the vocabulary conditioned on the start symbol <s>
2. Draw a random word (the first word) according to the predicted distribution
3. Predict a probability distribution over the vocabulary conditioned on the start symbol and the first word
4. Draw a random word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

Sampling words — alternatives

Sampling (generating) the most probable word at each step might not be optimal globally

- Beam search — generate top k candidates at each step

Learned word representations as a by-product



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

Each column of $\boldsymbol{W}^2$ learns a word representation

Word embeddings

Finishing previous lecture: Stacking layers, non-linearity

Language models

- Probability refresher
- ‘Classical’ language models
- Neural language models

Word embeddings

Word embeddings as pre-trained word representation

Option A: We can initialize the embeddings matrix $\boldsymbol{E}$ randomly and learn during our supervised task

Option B: Use pre-trained word embeddings from task for which we have a lot of data

- Use self-supervised learning (create labeled data ‘for free’ using the next word prediction objective)
- Learned word embedding matrix plugged into our supervised task

Desired word embeddings properties: ‘Similar’ words have similar embeddings vectors

Recap

- Finishing previous lecture: Stacking layers, non-linearity
- Language models
  - Probability refresher
  - ‘Classical’ language models
  - Neural language models
- Word embeddings

Take aways

- Language modeling is an essential part of contemporary NLP
- Self-supervised models, unlabeled data, next word prediction
- Neural language models learn embedding of words

License and credits

Credits

Ivan Habernal