CS 232: Artificial Intelligence

Fall 2022

Prof. Carolyn Anderson
Wellesley College
language model review

• Goal: compute the probability of a sentence or sequence of words:
  \[ P(W) = P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) \]

• Related task: probability of an upcoming word:
  \[ P(w_5 | w_1, w_2, w_3, w_4) \]

• A model that computes either of these:
  \[ P(W) \text{ or } P(w_n | w_1, w_2 \ldots w_{n-1}) \text{ is called a language model or LM} \]
n-gram models

\[ p(w_j | \text{students opened their}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)} \} \]
Perplexity

The best language model is one that best predicts an unseen test set (gives the highest P(sentence)).

Perplexity is the inverse probability of the test set, normalized by the number of words.

\[
PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}
\]

Chain rule: \( PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}} \)

For bigrams: \( PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \)

Minimizing perplexity is the same as maximizing probability.
Perplexity

The best language model is one that **best predicts an unseen test set** (gives the highest $P(\text{sentence})$).

Perplexity is the **inverse probability of the test set**, normalized by the number of words.

$$PP(W) = \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \log p(w_i | w_{<i})\right)$$

→ *Partial solution smoothing*
Neural Language Models
Problems with n-gram Language Models

Sparsity Problem 1

**Problem:** What if “students opened their $w_j$” never occurred in data? Then $w_j$ has probability 0!

\[
p(w_j | \text{students opened their}) = \frac{\text{count(students opened their $w_j$)}}{\text{count(students opened their)}}
\]
Problems with n-gram Language Models

**Storage**: Need to store count for all possible \( n \)-grams. So model size is \( O(\exp(n)) \).

\[
P(w_j | \text{students opened their}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}
\]

Increasing \( n \) makes model size huge!
another issue:

- We treat all words / prefixes independently of each other!

... Shouldn’t we share information across these semantically-similar prefixes?

students opened their ___
pupils opened their ___
scholars opened their ___
undergraduates opened their ___
students turned the pages of their ___
students attentively perused their ___
...

Slides adapted from Mohit Iyyer
words as basic building blocks

• represent words with low-dimensional vectors called **embeddings** (Mikolov et al., NIPS 2013)

\[
\text{king} = [0.23, 1.3, -0.3, 0.43]
\]
Enter neural networks!

Students opened their

neural language model

books

Slides adapted from Mohit Iyyer
composing embeddings

- neural networks **compose** word embeddings into vectors for phrases, sentences, and documents

neural network (students opened their) = 
Composing Embeddings

\[ \text{NN} \left( \text{students}, \quad \text{read}, \quad \text{their} \right) = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \]

Goal:

Probability distribution over vocabulary

\[ [0, 0.01, \ldots, 0.05] \]

predict "books"
Students opened their books

\[ X = \begin{pmatrix} -2.3 & 0.9 & 5.4 \end{pmatrix} \]

\[ W = \begin{pmatrix} 1.2 & -0.3 & 0.9 \\ 0.2 & 0.4 & -2.2 \\ 0.7 & -1.9 & 6.5 \\ 4.5 & 2.2 & -0.1 \end{pmatrix} \]

\[ Wx = \begin{pmatrix} 1.8, -11.9, 12.9, -8.9 \end{pmatrix} \]

\[ \text{SOFTMAX}(Wx) \]

\[ \sum_{j=0}^{V} \]
so to sum up...

- Given a $d$-dimensional vector representation $\mathbf{x}$ of a prefix, we do the following to predict the next word:

  1. Project it to a $V$-dimensional vector using a matrix-vector product (a.k.a. a “linear layer”, or a “feedforward layer”), where $V$ is the size of the vocabulary

  2. Apply the softmax function to transform the resulting vector into a probability distribution

Slides adapted from Mohit Iyyer
Now that we know how to predict “books”, let’s focus on how to compute the prefix representation $x$ in the first place!

neural network $\text{predict "books" } \left( \begin{array}{c} \text{students} \\ \text{opened} \\ \text{their} \end{array} \right) = \begin{array}{c} \text{something} \end{array}$
Composition functions

*input*: sequence of word embeddings corresponding to the tokens of a given prefix

*output*: single vector

- Element-wise functions
  - e.g., just sum up all of the word embeddings!

- Concatenation

- Feed-forward neural networks

- Convolutional neural networks

- Recurrent neural networks

- Transformers

Slides adapted from Mohit Iyyer
A Fixed-Window Language Model

\[
\hat{y} = \text{softmax} \left( W_2 h \right)
\]

hidden layer \( h = f(W_2 x) \)

\( f \) might be \( \text{ReLU} \) or \( \tanh \) or \( \text{sigmoid} \)

\( i = \text{concatenation} \)

\( X = \left[ c_1; c_2; c_3 \right] \)

Students opened their

\( 1 \quad 479 \quad 10 \)
Let’s look first at *concatenation*, an easy to understand but limited composition function.
A fixed-window neural Language Model

output distribution
\[ \hat{y} = \text{softmax}(W_2h) \]

hidden layer
\[ h = f(W_1x) \]

concatenated word embeddings
\[ x = [c_1; c_2; c_3; c_4] \]

words / one-hot vectors
\[ c_1, c_2, c_3, c_4 \]

Slides adapted from Mohit Iyyer
Neural Language Model

\[ p(\text{aardvark}|...) \quad p(\text{fish}|...) \quad p(\text{for}|...) \quad p(\text{zebra}|...) \]

Output layer
softmax

\[ \hat{y}_1 \quad \ldots \quad \hat{y}_{42} \quad \ldots \quad \hat{y}_{59} \quad \ldots \quad \hat{y}_{35102} \quad \ldots \quad \hat{y}_{|V|} \]

Hidden layer

\[ h_1 \quad h_2 \quad h_3 \quad \ldots \quad h_{d_h} \]

Projection layer
embeddings

\[ E \]

embedding for word 35
embedding for word 9925
embedding for word 45180

... and thanks

for all the

\[ w_{t-3} \quad w_{t-2} \quad w_{t-1} \quad w_t \]

Slides borrowed from Jurafsky & Martin Edition 3
Why Neural LMs work better than N-gram LMs

Training data:

We've seen:  I have to make sure that the cat gets fed.

Never seen:  dog gets fed

Test data:

I forgot to make sure that the dog gets ___

N-gram LM can't predict "fed"

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Slides borrowed from Jurafsky & Martin Edition 3
how does this compare to a normal n-gram model?

**Improvements** over n-gram LM:
- No sparsity problem
- Model size is $O(n)$ not $O(\exp(n))$

Remaining **problems**:
- Fixed window is **too small**
- Enlarging window enlarges $W$
- Window can never be large enough!
- Each $c_i$ uses different rows of $W$. We don’t share weights across the window.

Slides adapted from Mohit Iyyer
Recurrent Neural Networks!
An RNN Language Model

predict: starfish

predict: opened

predict: some

predict: books

\( h^t = f(W_h h^{t-1} + W_c c_t) \)

\( h^y = f(W_h h^3 + W_c c_y) \)
An RNN Language Model
why is this good?

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
- Weights are shared across timesteps → representations are shared

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

\[
\hat{y}^{(4)} = P(x^{(5)}|\text{the students opened their})
\]