

Notes

02 March 2024 07:19

Lecture Outline:

Lecture Plan

1. Neural dependency parsing (20 mins)
2. A bit more about neural networks (15 mins)
3. Language modeling + RNNs (45 mins)
 - A new NLP task: **Language Modeling**

↓ motivates

- A new family of neural networks: **Recurrent Neural Networks (RNNs)**

These are two of the most important concepts for the rest of the class!

Reminders:

- You should have handed in Assignment 2 by today
- In Assignment 3, out today, you build a neural dependency parser using PyTorch

Stanford

Neural Dependency Parser:

Limitations of Symbolic Dependency parsing (cont. From Lecture – 4)

- Features are generated in a rule based manner, hence they are sparse, Incomplete (difficult to set rules for each sentence in the training data) and involve extensive computations

1. How do we gain from a neural dependency parser? Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive computation

Neural Approach:
learn a dense and compact feature representation

Stanford

Symbolic to Neural Classifier:

- Sparse symbolic features are replaced by dense vectors of each word
- The Stack – buffer structure is still retained. However the classification Whether to form a dependency or not is now done by a Softmax classifier
- Two step approach:
 - Distributed representations of words + Part of Speech words + dependency labels are represented as vectors

Extracting Tokens & vector representations from configuration

- We extract a set of tokens based on the stack / buffer positions:

	word	POS	dep.
s1	good	JJ	∅
s2	has	VBZ	∅
b1	control	NN	∅
lc(s1)	∅	+	∅
rc(s1)	∅	∅	+
lc(s2)	He	PRP	∅
rc(s2)	∅	∅	nsbj

A concatenation of the vector representation of all these is the neural representation of a configuration

Stanford

- Using Softmax classifier

Graph-based dependency Parser:

- Moving on from transition based dependency parsing: Until now we are parsing the sentence from left to right to find the dependencies.
- How about computing every possible dependency in between all the words of the sentence? --- Graph based approaches

Graph-based dependency parsers

- Compute a score for every possible dependency (choice of head) for each word
 - Doing this well requires more than just knowing the two words
 - We need good "contextual" representations of each word token, which we will develop in the coming lectures
- Repeat the same process for each other word; find the best parse (MST algorithm)

e.g., picking the head for "big"

14

Concepts behind Neural Networks:

This part discussed following concepts: (approx. 15:00 to 45:00 in the video)

- L2 Regularization
- Dropout
- Activation Functions
- Weight Initialization
- Learning rate

Language Models and RNN:

What is language modelling?

- Task of predicting the next word.
- Mathematically, predict the probability of a word occurring next given the context of N previous words

Language Modeling

- **Language Modeling** is the task of predicting what word comes next

the students opened their _____

- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$
- A system that does this is called a **Language Model**

28

N-gram language Models:

- Models of pre-deep learning era.
- n-grams means collection of N consecutive words-
- Based on how frequent different combinations of word appear

n-gram Language Models

the students opened their _____

- **Question:** How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an **n-gram Language Model!**
- **Definition:** A **n-gram** is a chunk of n consecutive words.
 - **unigrams:** "the", "students", "opened", "their"
 - **bigrams:** "the students", "students opened", "opened their"
 - **trigrams:** "the students opened", "students opened their"
 - **4-grams:** "the students opened their"
- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

29

Use Markov Models:

- Markov assumption – a next sample depends on only last (n-1) samples
Turns this problem into conditional probability problem.
- Order of the Markov model – the "n" in the n-gram

n-gram Language Models

- First we make a **Markov assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | x^{(t)}, \dots, x^{(t-n+2)}) \quad (\text{assumption})$$

prob of a n-gram $\rightarrow P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})$
 prob of a (n-1)-gram $\rightarrow P(x^{(t)}, \dots, x^{(t-n+2)})$ (definition of conditional prob)

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

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \quad (\text{statistical approximation})$$

- An example of 4-gram model:
Disadvantage of n-gram:
 - As we discarded everything before "n" Words, we lost the context of this sentence.
 - The output then depends on stats of the dataset and not context of this sentence.

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their _____
 discard condition on this

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
 $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
- "students opened their exams" occurred 100 times
 $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$

Should we have discarded the "proctor" context?

Sparsity Problem in n-gram models:

- Sparsity – absence of context words from the dataset.
e.g. Students opened their w – not occurring in data – numerator = 0
e.g.2 students opened their – not occurring in data – denominator = 0
- Two solutions:
 - Smoothing when numerator is 0.
 - Backoff when denominator is 0.
- Increasing N makes the problem worse: I think we need graph based NN

Sparsity Problems with n-gram Language Models

Sparsity Problem 1
Problem: What if "students opened their w" never occurred in data? Then w has probability 0!
(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2
Problem: What if "students opened their" never occurred in data? Then we can't calculate probability for any w!
(Partial) Solution: Just condition on "opened their" instead. This is called *backoff*.

Note: Increasing n makes sparsity problems worse. Typically, we can't have n bigger than 5.

Storage Problems:

- Need of storing all stats of n-gram models in the corpus.

How to build a Neural Language Model?

- Recap: Language modelling is predicting next word given previous N words.
- Idea of Neural language model:
 - Input: sequence of N words
 - Output: Probabilities of next word over whole dataset
- Pros and Cons:

Pros	Cons
No sparsity problems. As system is not count based but considers whole dataset To calculate probabilities	Context is still limited to window of N words
Storage problem is solved as no need to store stats	Can't increase Window as it increase W matrix – computation time increases
	Sequence of word is not strictly considered as x_1 and x_2 gets multiplied by w_1 and w_2 . so features of x_1 and x_2 might end of far away from each other

A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over n -gram LM:

- No sparsity problem
- Don't need to store all observed n -grams

Remaining problems:

- Fixed window is **too small**
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W .
No symmetry in how the inputs are processed.

42

Why RNNs?

- Need or processing variable/larger input lengths
- Need to strictly consider the sequence of words

Simple RNN architecture:

A Simple RNN Language Model

output distribution
 $\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_u) \in \mathbb{R}^{|V|}$

hidden states
 $h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_h)$
 $h^{(0)}$ is the initial hidden state

word embeddings
 $e^{(t)} = E x^{(t)}$

words / one-hot vectors
 $x^{(t)} \in \mathbb{R}^{|V|}$

Note: this input sequence could be much longer than!

44

What problems are solved by using RNNs?

- Input length is decoupled from model size, so can use any input length
- Information of N previous steps is available in current hidden states – helps to provide large context for predicting next word
- Each input word vector gets multiplied by same Weight matrix W_e – order of words is preserved

