Week 3 CS224n : NLP with Deep Learning

Lecture 5 : Dedendency Parsing

Phrase structure organizes words into nested constituents

- Constituency = phrase structure grammar = context-free grammars
- Dedendency = which words depends on (modify or are argument of) which other word

route

Look in the large crate in the kitchen by the door

Machine needs to understand sentence structure to interpret language

[PP attachment ambiguities multiply]

subj obj

The board approved [its acquisition] [by Royal Trust co LTd]

V N N

[of Toronto]

[for $27 a share]

[at its monthly meeting].

[Coordination scope ambiguity]

[Adjectival Modifier ambiguity]

[verb phrase attachment ambiguity]

Dependency Structure : a tree (acyclic, connected, single-head)

the rise of annotated data: Universal Dependencies tree bank

- very reusable
- broad coverage
- Frequencies and distributional information
- A way to evaluate systems.

Sources of information for dependency parsing?

- Bilexical affinities

- distance
- intervening material
- valency of heads

— a sentence is parsed by choosing for each word what other word (including Root) is it a dependent of

evaluation:

$$Acc = \frac{\# \text{ correct deps}}{\# \text{ of deps}}$$

□ a neural dependency parser ?

problem : Indicator feature revisited

- sparse
- incomplete
- expensive computation
- more than 95% of parsing time is by feature computation.

□ Distributed representations

- represent as d-dimensional dense vector
- Part of Speech (POS) and dependency labels as as d-dimensional dense vector

Model architecture:

Output layer y

$y = \text{softmax}(Uh + b_2)$

Hidden layer h

$\text{reLU}(Wx + b)$

Input layer x

lookup + concat

Lecture 6 : Language Models and RNNs

- Language Modeling : predict what words come next.

compute $\mathbb{P}(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)})$, where $x \in V$ vocabulary

$\mathbb{P}(X^{(t)}, X^{(t-1)}, \ldots, X^{(1)}) = \mathbb{P}(X^{(1)}) \cdot \mathbb{P}(X^{(2)} | X^{(1)}) \cdot \ldots \cdot \mathbb{P}(X^{(t)} | X^{(t-1)}, \ldots X^{(1)})$

n-gram Language Models

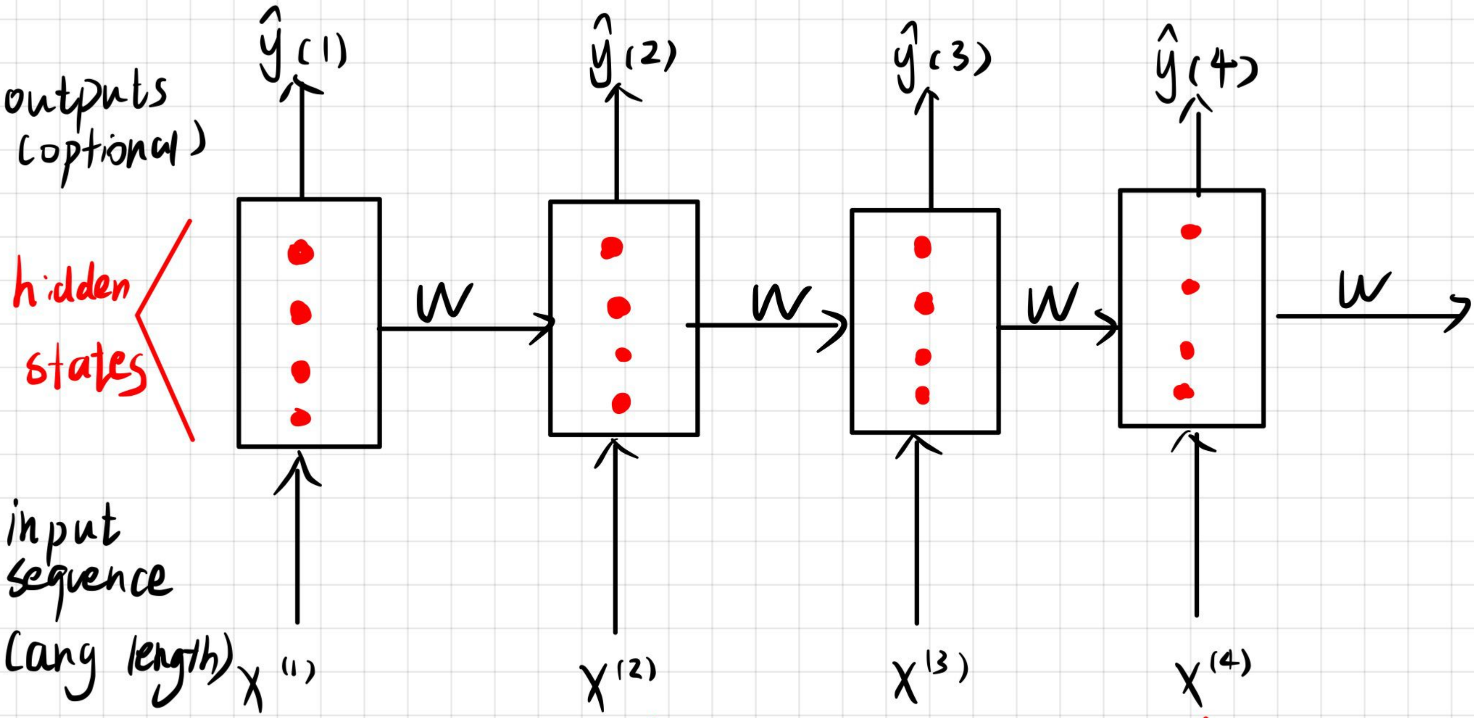
- simplifying assumption: $X^{(t+1)}$ depends only on preceding (n-1) words.
- Sparsity problem 1: add small δ for every $w \in V$ called smoothing
- sparsity problem 2: back-off
- storage problem: increasing n ⟹ increasing size of model

A fixed-window neural Language Model

- Improvement: no sparsity problem
  don't need to store all observed n-grams
- remaining problem: · Fixed window is small
  · No symmetry in vector processing

we need a neural architecture that processing <u>any length</u> text.

Recurrent Neural Network (RNN)



Use the same W matrix again and again

advantage: • can process any length input

• Computation for step t can (in theory) use information from many steps back

• Model size doesn't increase for longer size

disadvantage: · recurrent computation is slow

· diffcult to assess information from many steps back

Training RNN Language Model

1. get a very large corpus of text

2. Feed into RNN-LM; compute distribution $\hat{y}^{(t)}$ for every step t.

3. Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (1-hot vector):

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_m^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

4. Average this to get overall loss for entire training set.

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}$$

However, compute loss and gradient across entire corpus is too expensive! Use [SGD] in real practice training.

$$\frac{\partial J^{(t)}}{\partial W_n} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}$$

"Backpropagation through time"

Generate text with a RNN Language Model

Evaluation Language Model:

- Perplexity: $\underbrace{\prod_{t=1}^{T}\left(\frac{1}{P_{LM}(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)})}\right)}_{\text{Inverse probability of corpus, according to language model}}^{1/T}$ Normalized

Low perplexity is better.