[Lec 06]

Language Modeling

- Task of predicting **what word comes next**

$$= \underbrace{\prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}$$

n-gram Language Models (pre Deep Learning)

n-gram: chunk of n consecutive words ex) unigrams, bigrams, trigrams ... etc

- `count probabilities` to get probabilities

q. Context matters? → throwing away too much context leads to problems

Sparsity Problems

q. sparse data? → never appear on the training data (sparsity problem)

sol) add small delta value to every vocabulary. → smoothing method

q. denominator zero?

sol) back off to just conditioning lesser words (n words → n-1 words)

q. as n increases → sparsity problem gets even more common

usually use up to 5-grams

Storage Problems

q. increase n → storage space gets larger

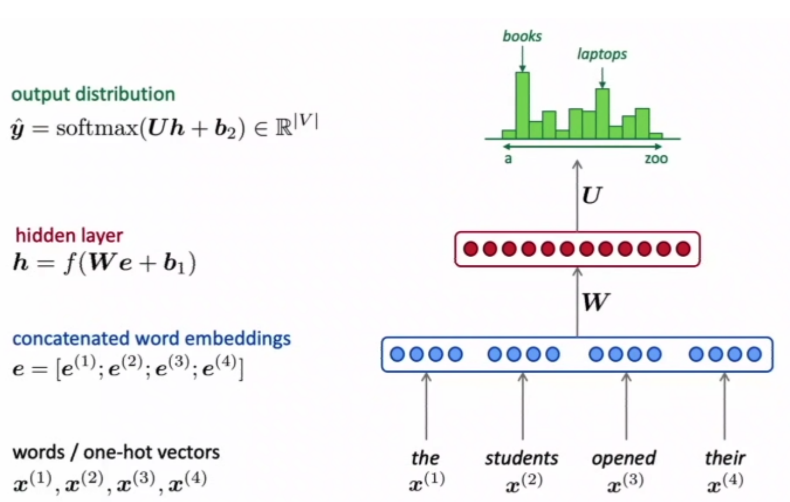
Practice

- sparsity problem occur
- can use language model to `generate text` (conditioning → sampling → conditioning → sampling ,,,)
  - Surprisingly grammatical but incoherent (only consider 2 words prior)

Neural Language Model?

Window-based neural Model

- Lec 3 (NER)



+

- No sparsity problem
- don't need to store all observed n-grams → rather store all vocab

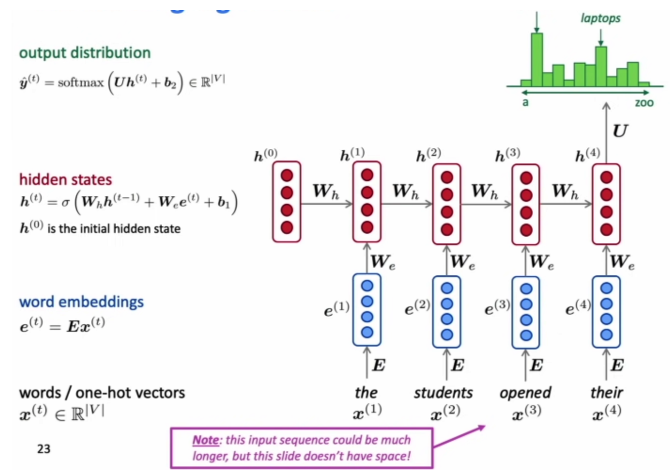
-

- fixed window too small
- enlarging window `enlarges W`
- commonalities between word embeddings are learned separately (not efficient)

RNN

Recurrent Neural Network

- sequence of hidden states (single state that is mutating over time) : time-steps
- apply the same weights W repeatedly
- can be any length (sequence of text)



We and Wh learned !

+

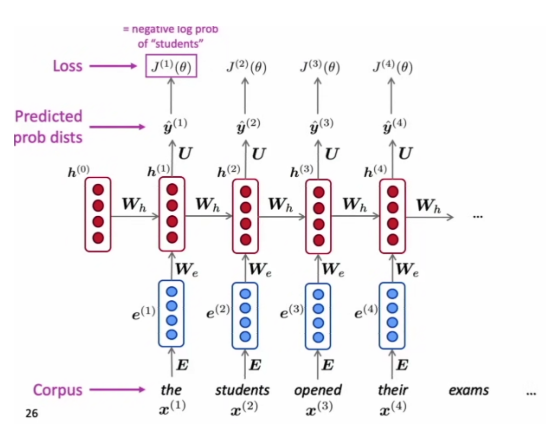
- any length input
- computation for step t can use information from many steps back (in theory)
- Model size doesn't increase for longer input (W_e, W_h, and bias)
- Same weights applied on every time step —> symmetry / efficient

-

- slow → compute them in a sequence (cant be in parallel)
- difficult to use info from many steps back (in practice)

Training a RNN LM

1. Big corpus of text (sequence of words)
2. Feed it to RNN → compute output distribution y for every step t
3. Loss function → cross entropy btw predicted probability and true word t

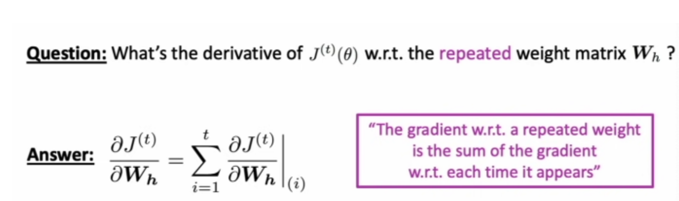


4. Average this to get overall loss

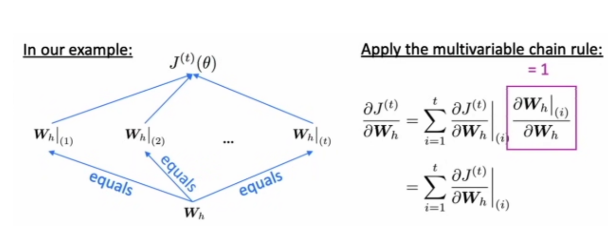
Practice

- Compute by stochastic gradient descent
  - by batch of sentences & compute gradients & update weights

BackProp for RNN



- By using the `chain rule`



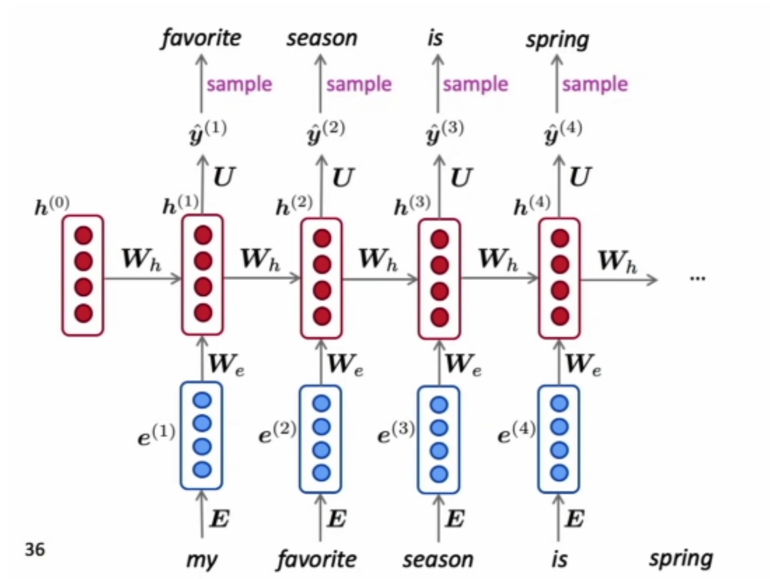
- Back Propagation through time (summing gradients as you go)

Q> Batch casts huge impact?

→ shuffle data

Text Generation with RNN

- Repeated sampling technique



Evaluating LM

- perplexity

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

- Lower perplexity is **Better**

RNN can be used for tagging