$P(w \mid \langle s \rangle)$

|V|

|V|-dim prob-dist.

Neural Language Model

0.01 0.51 0.71 0.91 1.0

Pawan Goyal

LM to generate

CSE, IIT Kharagpur

sample a word from this

NLP - Interaction Hour

0.01 0.5 0.2

0.2 0.09

<s> I want to eat Chinese food

0.4

0.01
0.51
0.71

0.91
1.0

random pick a number betn 0 & 1

denied the

$$\overbrace{[\,0 \mid 0 \mid 0 \mid 0 \mid 0.1 \mid 0.2 \mid 0.3 \mid 0.4\,]}^{}\,\mid 0 \mid 0 \mid 0\,]$$

V

$\downarrow$ smoothing

$P(w \mid \text{denied the})$

$$[\,1/|V| \mid \ldots\,]$$

Unigram-prior smoothing

Voc $\rightarrow$ 1 million $\rightarrow$ 500,000

words with [illegible].

Quiz on Transl(iter)ation

Sanskrit SLP1

There are names of some countries in South America, written in the Georgian language, together with their translations to English:

WX notation

Brazilia

ბრაზილია	Brazil
პერუ	Peru
ურუგვაი	Uruguay
არგენტინა	Argentina
კოლუმბია	Columbia

जापान Japan

चीन China

रूस Russia

What are the names, in English, of the two untranslated countries?

कमला → Kamala

Sanskrit

Hindi → Kamal

कमला → Kamala

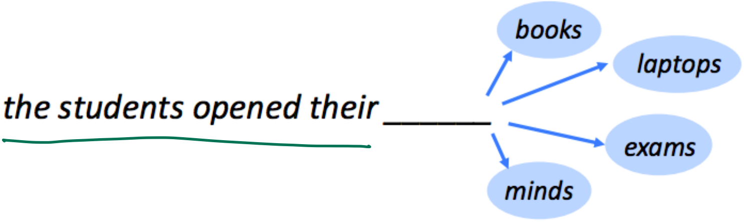
Reference

Speech and Language Processing, 3rd Edition (draft): Chapter 9. Sequence Processing with Recurrent Networks
https://web.stanford.edu/~jurafsky/slp3/9.pdf

Many of these slides have been adapted from CS 224n. Some material is from the Deep Learning book.

Language Modeling

Language Modeling is the task of predicting what word comes next.

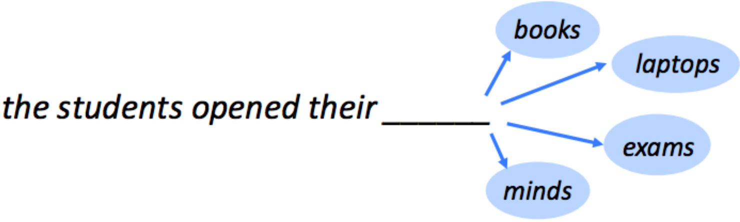


- **Goal:** Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

Language Modeling

Language Modeling is the task of predicting what word comes next.



- **Goal:** Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

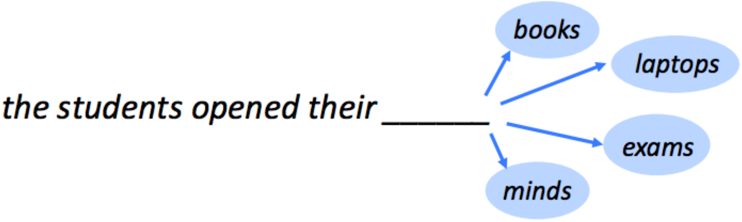
auto-completion

- **Related Task:** probability of an upcoming word:

$$\boxed{P(w_4 | w_1, w_2, w_3)}$$

Language Modeling

Language Modeling is the task of predicting what word comes next.



- **Goal:** Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

- **Related Task:** probability of an upcoming word:

$$P(w_4 | w_1, w_2, w_3)$$

- A model that computes either of these is called a **language model**

Why should we care about language modeling?

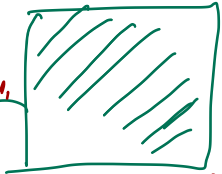
- Language Modeling is a benchmark task that helps us measure our progress on understanding language
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition } LM
  - Spelling/grammar correction
  - Authorship identification

S.P. — LM_1 ①

A.C. — LM_2 ②

Complete works available

P_{LM_1}



P_{LM_2} $\arg\min\{P_{LM_1}, P_{LM_2}\}$

n-gram language models

the students opened their ______

Question: How to learn a Language Model?

Answer (pre- Deep Learning): learn a *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.

n-gram language models

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

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \qquad \text{(assumption)}$$

n-1

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}} \qquad \text{(definition of conditional prob)}$$

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

Bi $P(x_2|x_1)$

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

$C(x_1, x_2) / C(x_1)$

n-gram language models: Example

$|V|^4$ $|V| \approx 500{,}000$

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their ______ books

discard

$\underbrace{\text{students opened their}}_{\text{condition on this}}$

$|V|^9$ T 9-gram rare

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

How many parameters

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

unigram LM : $|V|$

bi-gram: $|V|^2$

Storage Problems with n-gram Language Model

Storage: Need to store count for all *n*-grams you saw in the corpus.

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

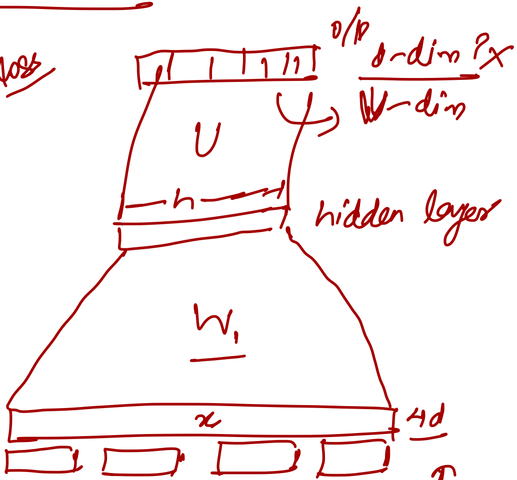
Increasing *n* or increasing corpus increases model size!

A fixed-window neural language model

2003 Bengio

CE loss

$$\boxed{U^T\left(\sigma\left(Wx+b\right)\right)+c}$$



~~as the proctor started the clock~~ the students opened their ______

discard

fixed window

A fixed-window neural language model

Adv. over n-gram LM?

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

O(N)

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

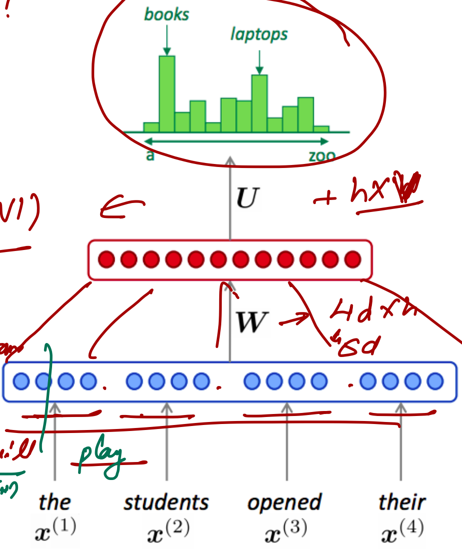
$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

5-gram

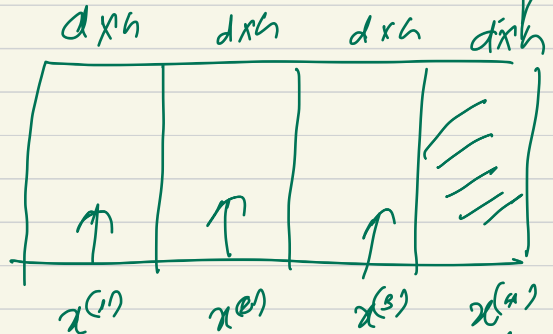
Sachin Tendulkar will play

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



play played



$4d \times h$

Sachin Tendulkar will

Sachin Tendulkar will surely

A fixed-window neural language model

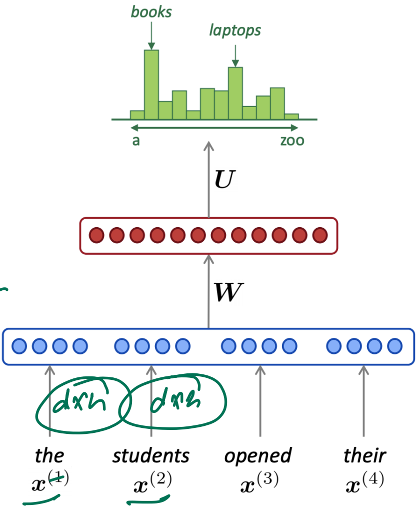
Improvements over *n*-gram LM:

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

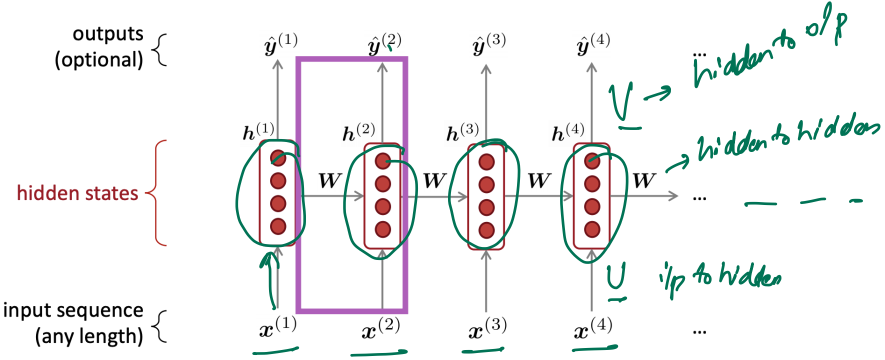
Remaining **problems**:

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

We need a neural architecture that can process *any length input*



Recurrent Neural Networks



Core Idea

Apply the same weights repeatedly!

We can process a sequence of vectors x by applying a recurrence formula at each step:

Recurrent Neural Networks

We can process a sequence of vectors x by applying a recurrence formula at each step:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

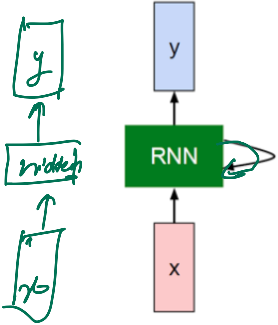
new state

old state

input vector at some time step

some function with parameters W

Notice: the same function and the same set of parameters are used at every time step.

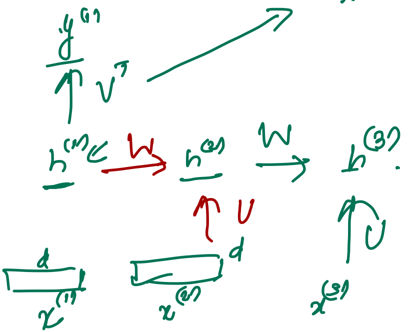


Forward propagation for the RNN: first model

Activation function for the hidden units

Assume the hyperbolic tangent activation function

$\tanh(U x^{(1)} + b) + c$



$y^{(t)}$

$V^T h^{(2)} + c$

$h^{(t)}$

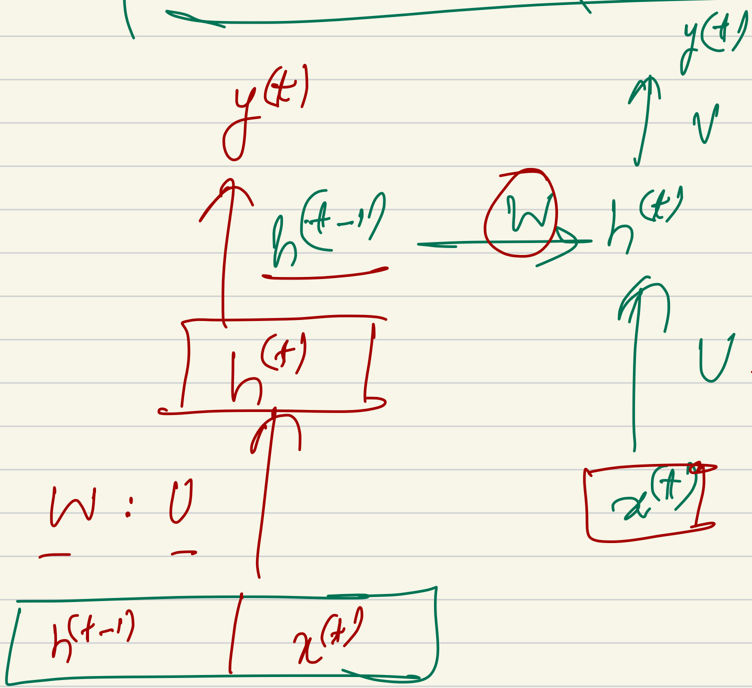
$h^{(2)} = \tanh(U x^{(2)} + W h^{(1)} + b)$

— — —

$x^{(t)}$

$$h^{(t)} = \tanh\left(W h^{(t-1)} + U x^{(t)} + b\right)$$

$$y^{(t)} = \text{softmax}\left(c + V h^{(t)}\right)$$



$\underline{V}$, $\underline{W}$, $\underline{U}$

$h \times |V|$, $h \times h$, $h \times d$

Forward propagation for the RNN: first model

Activation function for the hidden units

Assume the hyperbolic tangent activation function

Form of output and loss function

Assume output is discrete - predicting words

We can obtain a vector normalized probabilities over the output - $\hat{y}$.

Forward propagation for the RNN: first model

Activation function for the hidden units

Assume the hyperbolic tangent activation function

Form of output and loss function

Assume output is discrete - predicting words

We can obtain a vector normalized probabilities over the output - $\hat{y}$.

Update Equations

Initial state - $h^{(0)}$

Forward propagation for the RNN: first model

Activation function for the hidden units

Assume the hyperbolic tangent activation function

Form of output and loss function

Assume output is discrete - predicting words
We can obtain a vector normalized probabilities over the output - $\hat{y}$.

Update Equations

Initial state - $h^{(0)}$
From $t = 1$ to $t = \tau$, the following update equation is applied:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

Forward Propagation

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = tanh(a^{(t)})$$

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = softmax(o^{(t)})$$

pred.

Forward Propagation

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$\hat{y}^{(t)} = softmax(o^{(t)})$$

This maps an input sequence to an output sequence of the same length.

Loss Function

Total loss is sum of the losses over all the time steps.
So, if $L^{(t)}$ is the negative log likelihood of $y^{(t)}$ given $x^{(1)}, \ldots, x^{(\tau)}$, then

Loss Function

Total loss is sum of the losses over all the time steps.
So, if $L^{(t)}$ is the negative log likelihood of $y^{(t)}$ given $x^{(1)}, \ldots, x^{(\tau)}$, then

$$\begin{aligned} & L(\{x^{(1)}, \ldots, x^{(\tau)}\}, \{y^{(1)}, \ldots, y^{(\tau)}\}) \\ = & \sum_t L^{(t)} \\ = & -\sum_t log\, p_{model}(\underbrace{y^{(t)}}_{\text{correct class}} | \{x^{(1)}, \ldots, x^{(\tau)}\}) \end{aligned}$$

Loss Function

Total loss is sum of the losses over all the time steps.
So, if $L^{(t)}$ is the negative log likelihood of $y^{(t)}$ given $x^{(1)}, \ldots, x^{(\tau)}$, then

$$\begin{aligned} & L(\{x^{(1)}, \ldots, x^{(\tau)}\}, \{y^{(1)}, \ldots, y^{(\tau)}\}) \\ = & \sum_t L^{(t)} \\ = & -\sum_t log\, p_{model}(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\}) \end{aligned}$$

CE

where $p_{model}(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\})$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{y}^{(t)}$

Loss Function

Total loss is sum of the losses over all the time steps.
So, if $L^{(t)}$ is the negative log likelihood of $y^{(t)}$ given $x^{(1)}, \ldots, x^{(\tau)}$, then

$$
\begin{aligned}
& L(\{x^{(1)}, \ldots, x^{(\tau)}\}, \{y^{(1)}, \ldots, y^{(\tau)}\}) \\
= & \sum_t L^{(t)} \\
= & -\sum_t log\ p_{model}(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\})
\end{aligned}
$$

where $p_{model}(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\})$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{y}^{(t)}$

Back propagation - right to left - back propagation through time (BPTT)

A RNN Language Model

output distribution

$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\cancel{\boldsymbol{U}} V \boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

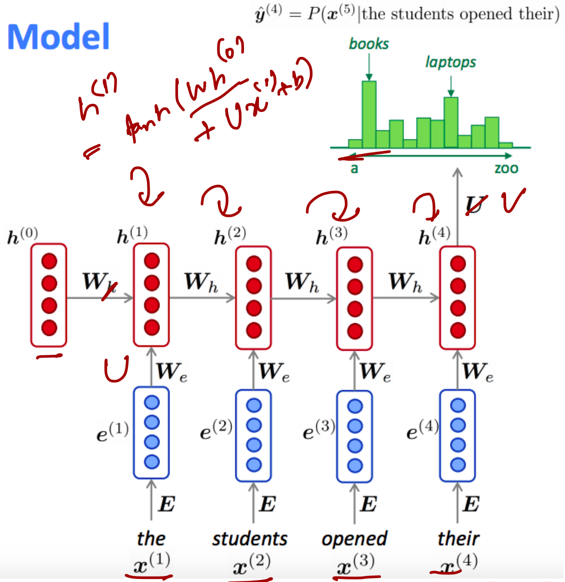
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

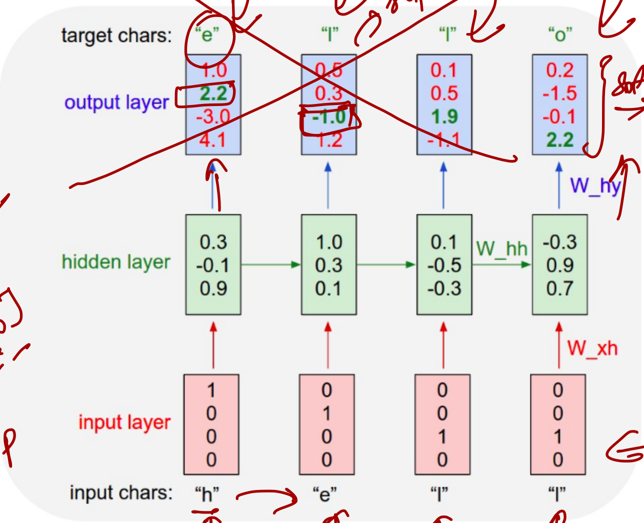
$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$

words / one-hot vectors

$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



Example RNN



predict the next char.

4 chars [h, e, l, o]

hell i/p

softmax

softmax

0.2
0.03
0.07
0.7

highest prob.