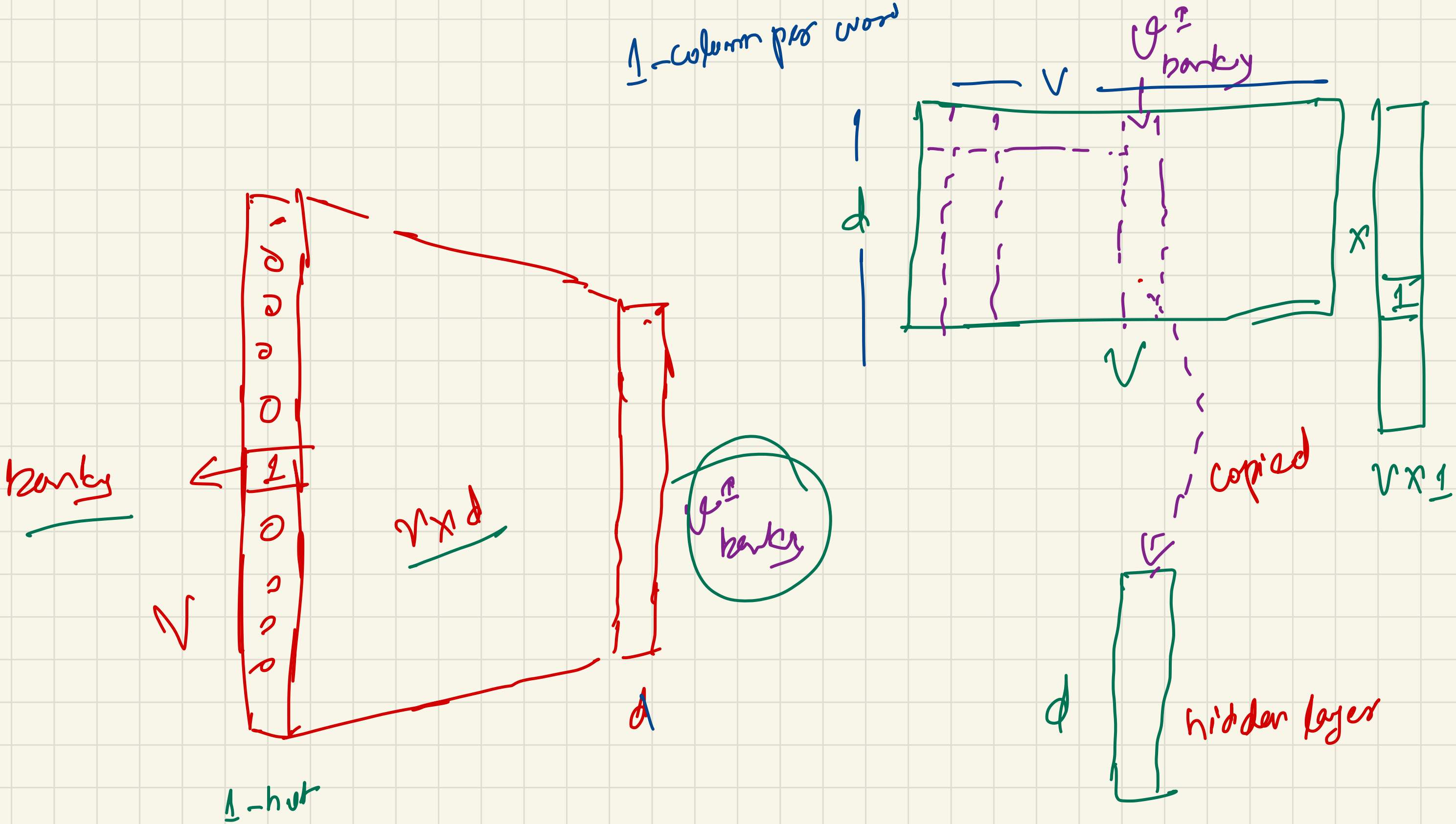
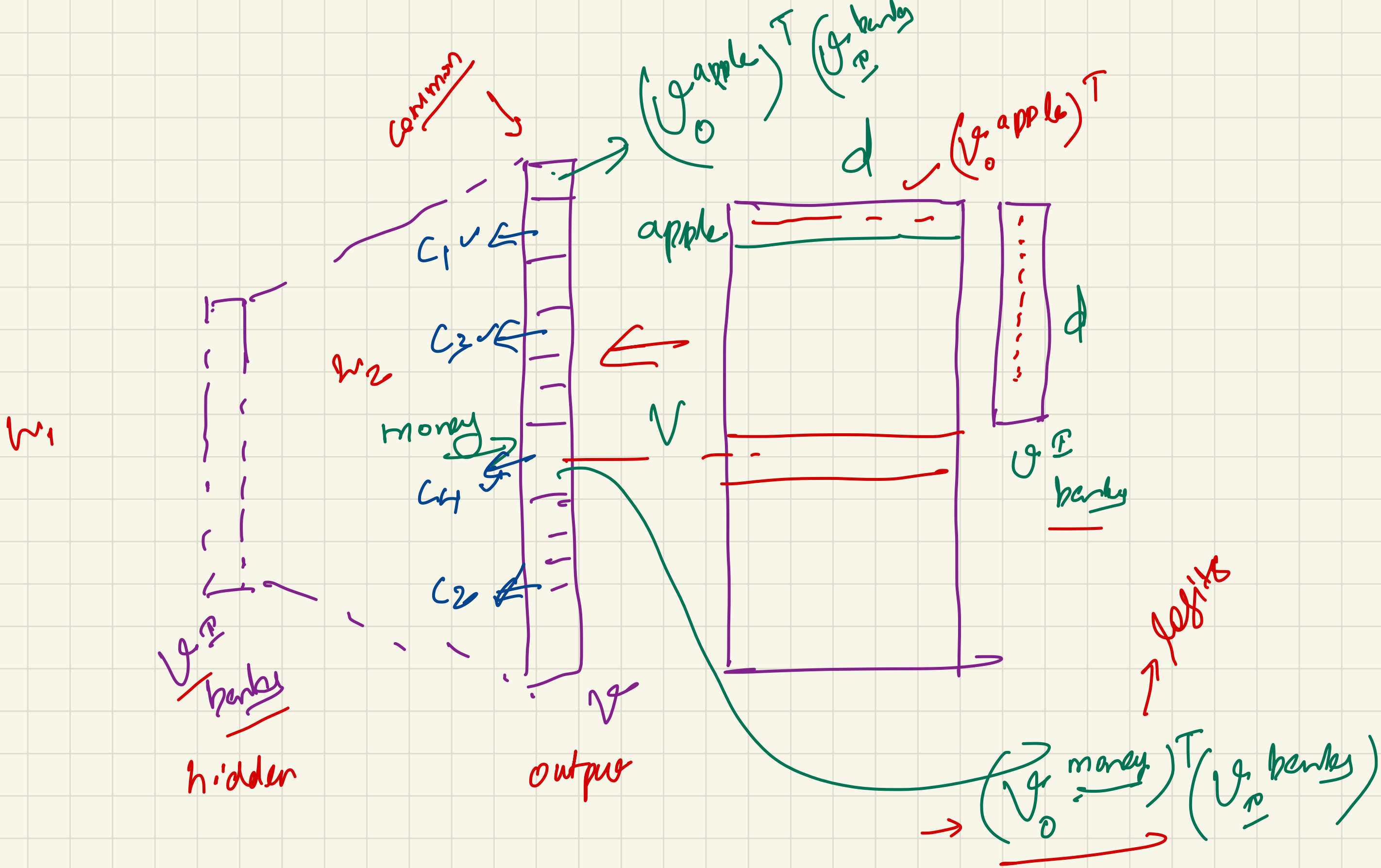

1-column per word
V
v^T banky
d
x
1
V
copied
V x 1
v
d
hidden layer
banky
V
V x d
d
v^T banky
1-hot
1



loss $= -\log P(GT)$

$$\frac{e^{(v_o^{money})^T (v_I^{banks})}}{\sum^{V} e^{(v_o^{j})^T (v_I^{banks})}}$$

- Language Models

  - n-gram LMs

    issues?

    - Parameters depends on window length

      V^k (for k-prev. words)

Word2Vec: objective function

We want to minimize the loss function:

center

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

context

How to calculate $P(w_{t+j}|w_t;\theta)$?

We will use two vectors per word w:

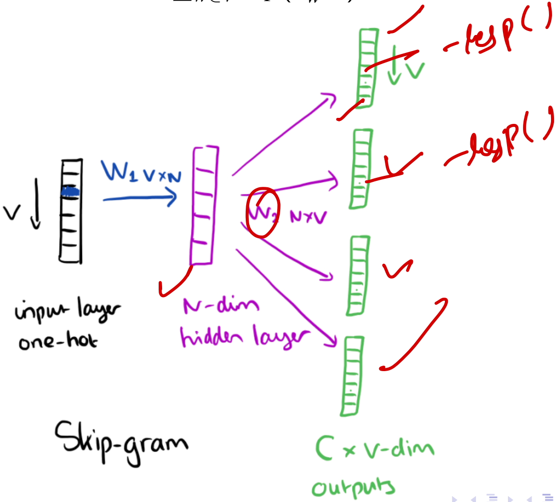
- v_w when w is a center word → Input
- u_w when w is a context word → output

Then, for a center word c and a context word o

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)}$$

Understanding $P(o|c)$ further

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$



center

Suppose you are computing the word vectors using Skip-gram architecture. You have 10 words in your vocabulary, $\{hi, you, they, are, am, how, why, there, who, what\}$ in that order and suppose you have the window, 'hi there how are you' in your corpora. You use this window with 'how' as the center word and two words before and after the center word as your context. Also, suppose that for each word, you have 2-dim in and out vectors, which have the following value at this point given as follows:

window

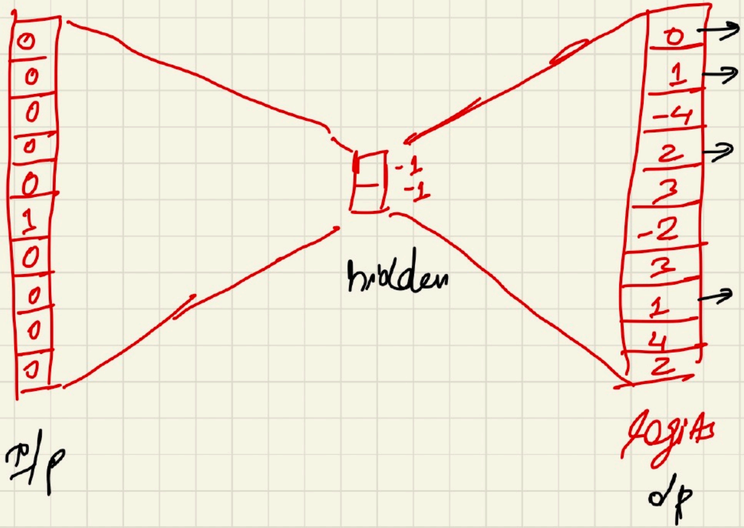
V=10

Word	In-vector	Out-vector	
hi	(1, -1)	(-2, 2)	0
you	(-1, 2)	(1, -2)	3
they	(-2, -2)	(2, 2)	-4
are	(1, 1)	(-1, -1)	2
am	(2, 1)	(-2, -1)	3
how	(-1, -1)	(1, 1)	-2
why	(1, 2)	(-1, -2)	3
there	(2, -1)	(-2, 1)	1
who	(2, 2)	(-2, -2)	4
what	(1, 1)	(-1, -1)	2

Table 1: In and Out representations for words

What will the values at the input, hidden and output layer as per the Skip-gram architecture? What would be the total loss for this window?

$x \rightarrow (e^{x/2})$



50-d
25-d
300-d embeddings

$$Z = e^0 + 2\cdot e^1 + 2\cdot e^2 + 2\cdot e^3 + e^4 + e^{-2} + e^{-4}$$

$$\text{Total loss} = -\left[\log\left(\frac{e^0}{z}\right) + \log\left(\frac{e^1}{z}\right) + \log\left(\frac{e^2}{z}\right) + \log\left(\frac{e}{z}\right)\right]$$

Try this problem

Skip-gram

Suppose you are computing the word vectors using Skip-gram architecture. You have 5 words in your vocabulary, $\{passed, through, relu, activation, function\}$ in that order and suppose you have the window, '*through relu activation*' in your corpora. You use this window with 'relu' as the center word and one word before and after the center word as your context.

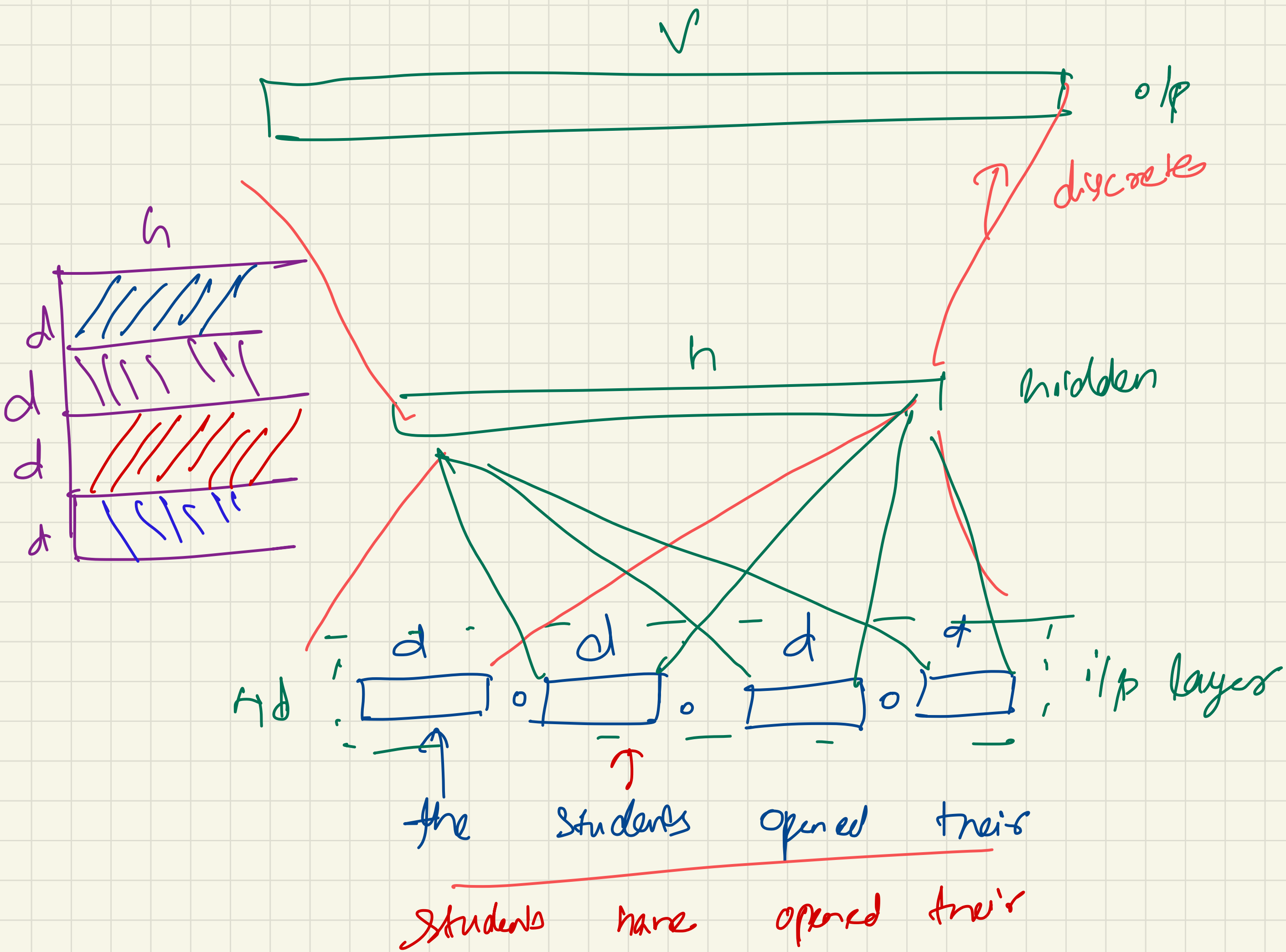
Compute the loss

Also, suppose that for each word, you have 2-dim in and out vectors, which have the same value at this point given by [1,-1],[1,1],[-2,1],[0,1],[1,0] for the 5 words, respectively. As per the Skip-gram architecture, the loss corresponding to the target word "activation" would be $-log(x)$. What is the value of x?

Homework

- Compute partial derivative of the loss with respect to v_c

$$\frac{\partial J}{\partial v_c}$$



V
o/p
discrete
h
d
d
d
d
h
hidden
d
d
d
d
4d
i/p layer
the students opened their
students have opened their

$d \ll V$

A fixed-window neural language model: Pros and Cons

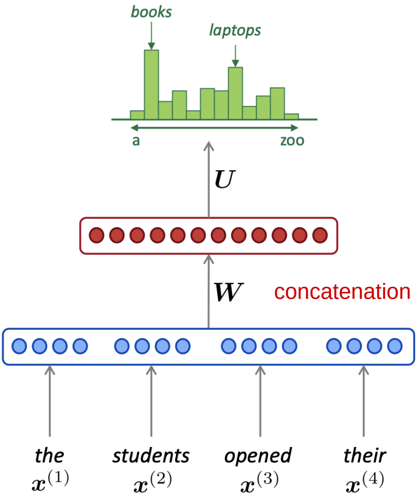
Improvements over *n*-gram LM:

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

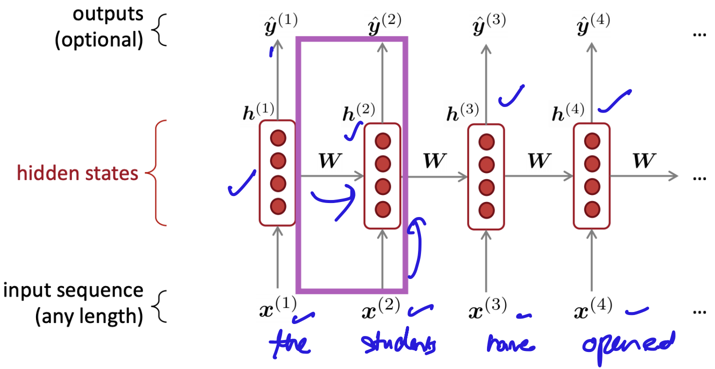
Remaining **problems**:

- 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:

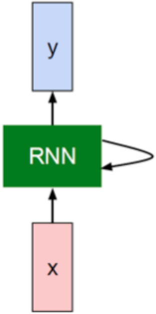
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

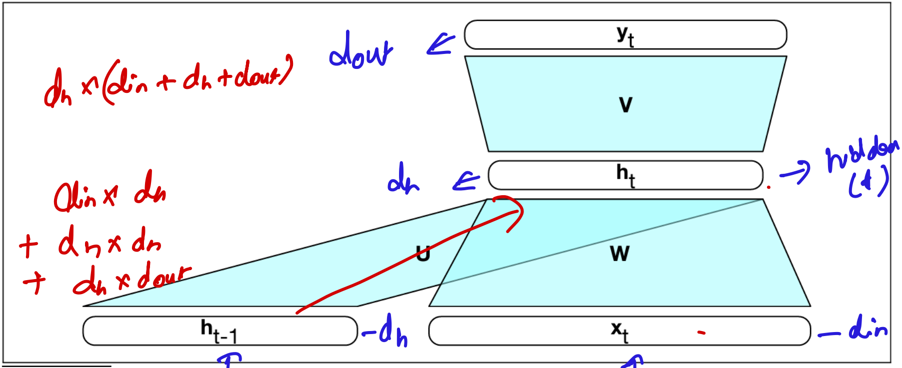
input vector at some time step

some function with parameters W



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

RNN as a feed-forward network



prev. hidden state (t-1)

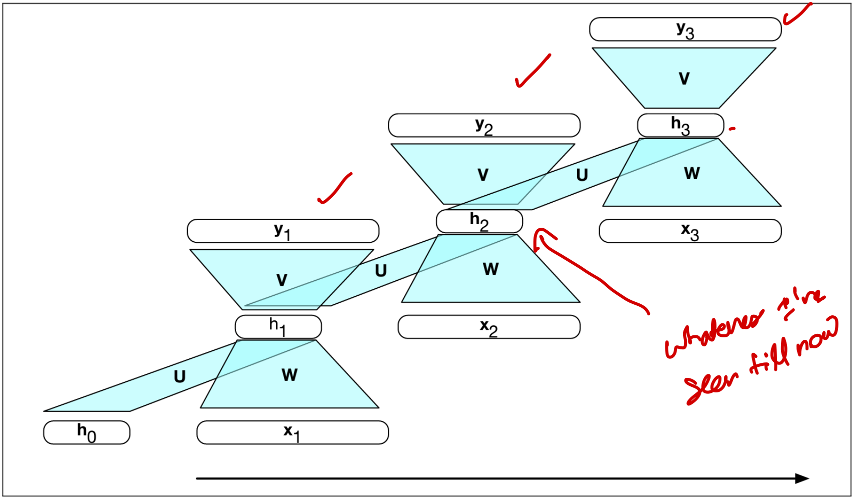
current i/p time (t)

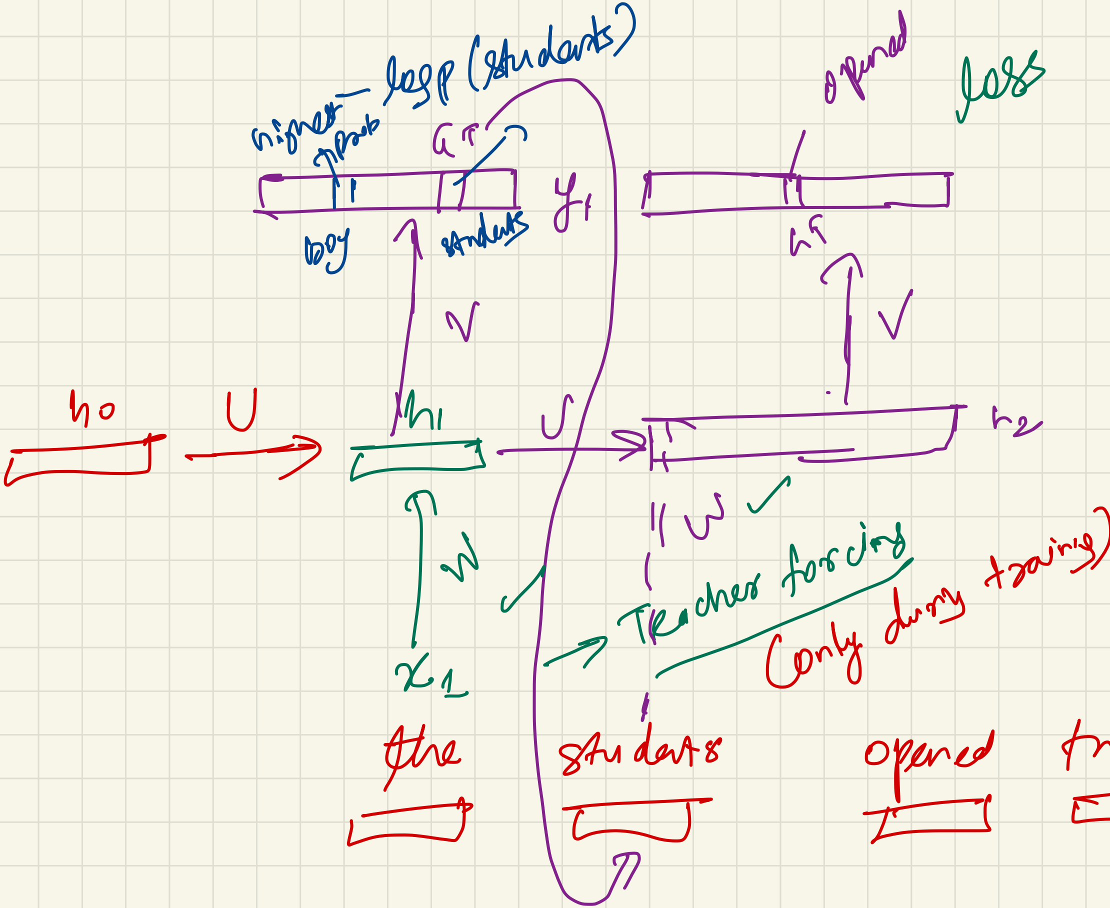
Forward Propagation

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = softmax(Vh_t)$$

- Let the dimensions of the input, hidden and output be d_{in}, d_h and d_{out}, respectively
- The three parameter matrices: $W : d_h \times d_{in}$, $U : d_h \times d_h$, $V : d_{out} \times d_h$

RNN Unrolled in Time





U, W, V

BPTT

back-propagation through time

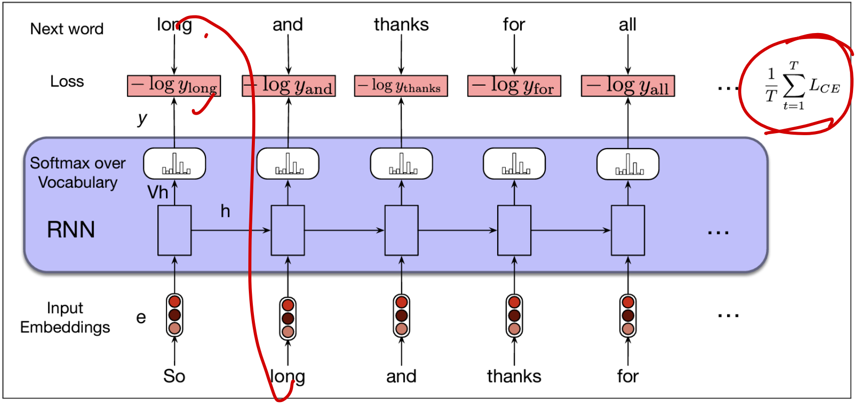
Training an RNN language model

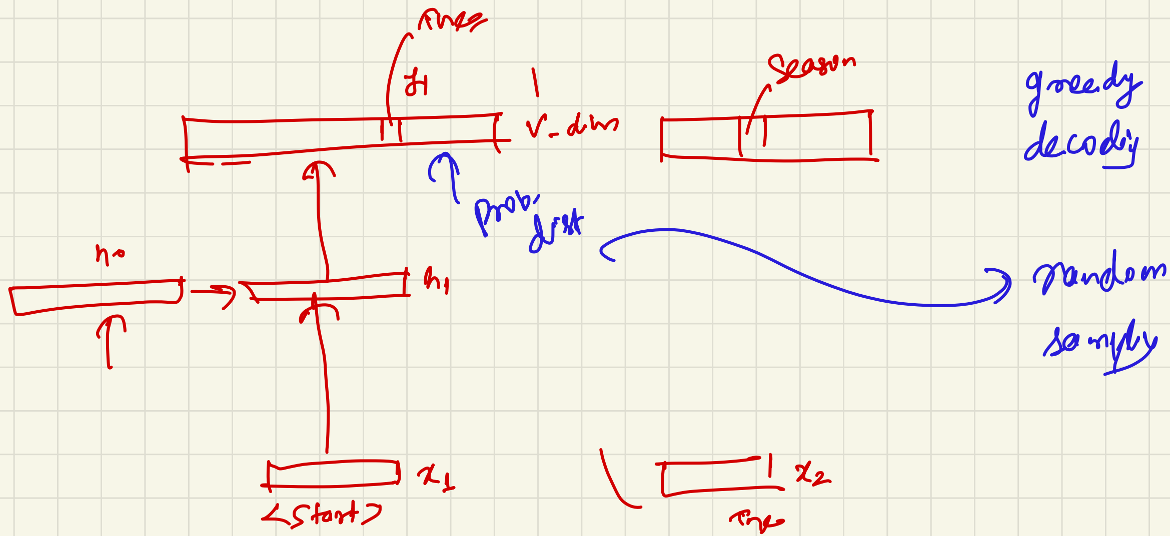
- To train RNN LM, we use self-supervision (or self-training)
- We take a corpus of text as training material
- At each time step t, we ask the model to predict the next word

Why is it called self-supervision?

- We do not add any gold data, the natural sequence of words is its own supervision!
- We simply train the model to minimize the error in predicting the true next word in the training sequence

Training an RNN language model





V_dim
Season
prob. dist
h0
h1
x1
<start>
x2
greedy decoding
random sampling