in an n-gram model:

the students opened their ____

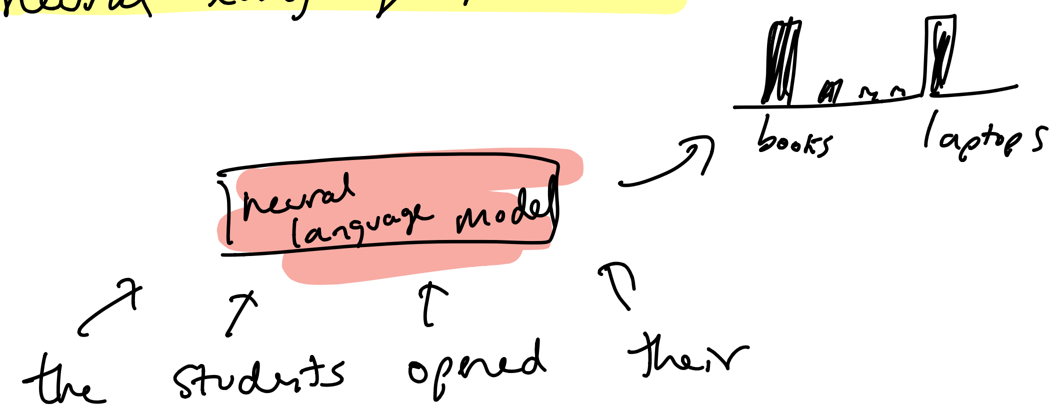
the student opened her ____

the iPad expoded because ____

↳ how do we share info between semantically-similar prefixes?

---

neural language models

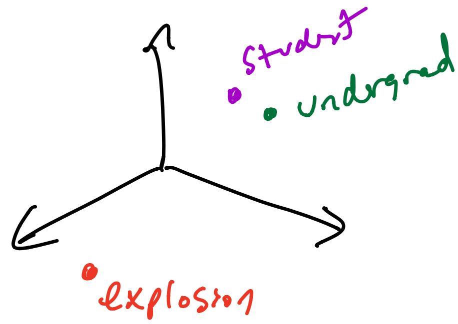


↳ NLMs need to be trained

↳ NLMs represent words, phrases, sentences, documents w/ dense real-valued vectors

embeddings

student : [-0.43, 0.72, 1.1]

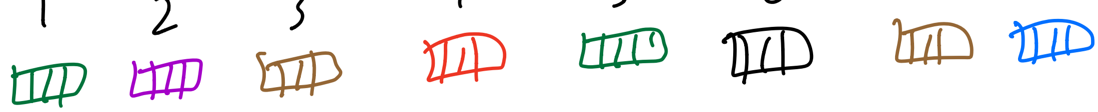


Word embeddings

↳ type vs. tokens

- type ↓ unique entry in vocabulary
- tokens ↓ occurrence of a type in a text

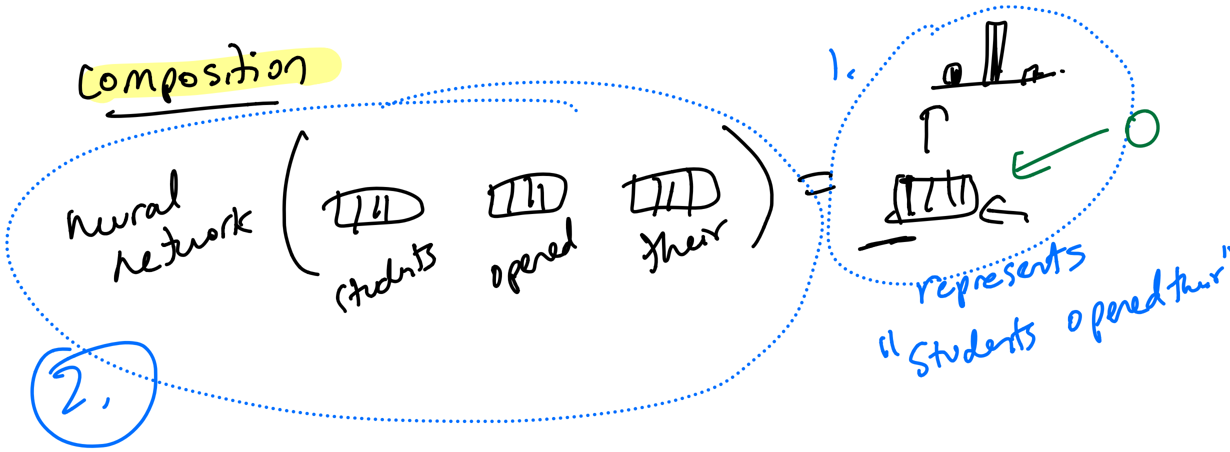
the	blue	ants	hated	the	red	ants	.
1	2	3	4	5	6	7	8



↳ embs are usually 50 - 1000d

↳ embs are stored in a $V \times d$ matrix

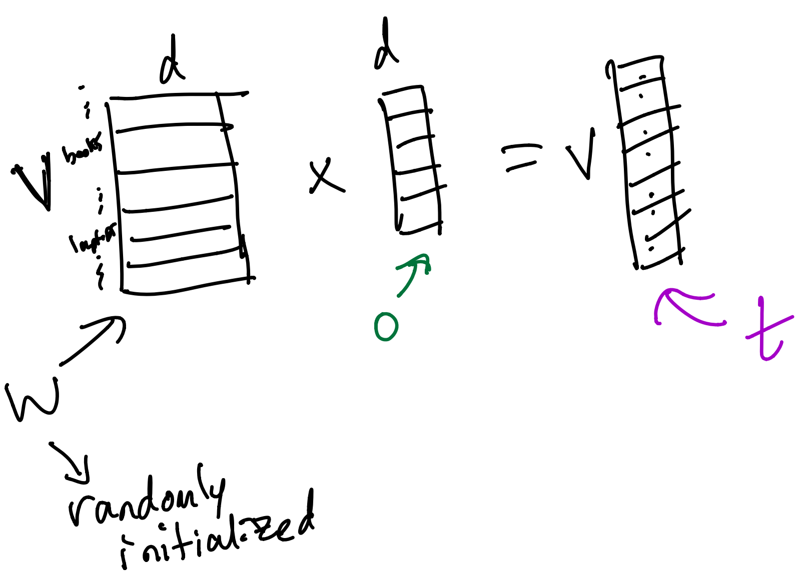
composition



↳ vocab has V word types
embs are d dimensional

↳ if we do a matrix vector product, we can project our emb. into a V-dim vector

→ W is a $V \times d$ matrix



$t = [1.8, -1.9, 2.9, \ldots]$
$t_0 \uparrow \quad t_1 \uparrow$

↳ we use the softmax function to squash any input vector into a probability distribution

$$\mathrm{softmax}(t) = \frac{e^{t}}{\sum_j e^{t_j}}$$

→ making everything positive

→ make everything sum to 1

t_j is the value at the j^{th} dim of t

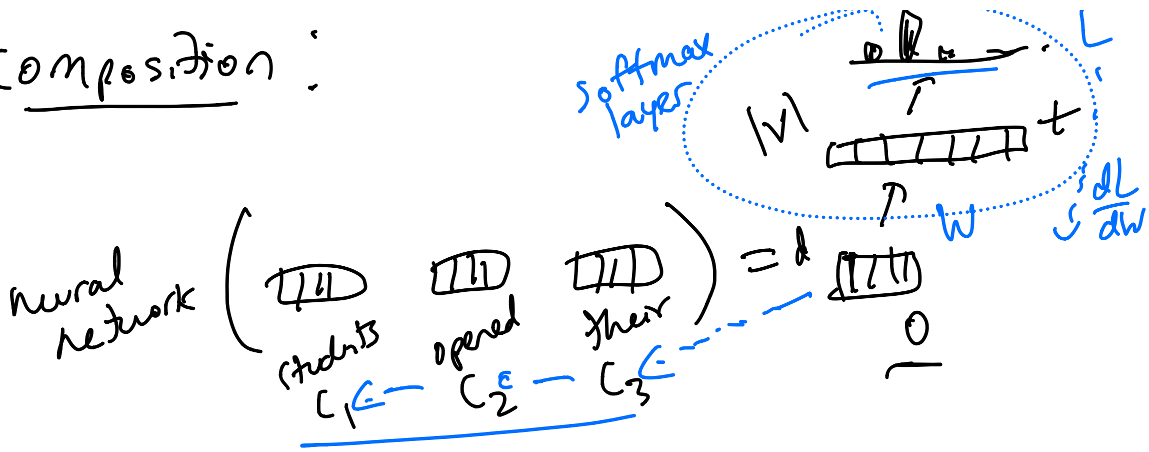
let's say $V = \{\text{books}, \text{cans}, \text{laptops}, \text{zoo}\}$

if $t = [1.8, -1.9, 2.9, -0.9]$

$\mathrm{softmax}(t) = [0.24, 0.006, 0.73, 0.02]$

$p(\text{books} \mid \text{prefix})$ $\quad$ $p(\text{can} \mid \text{prefix})$

composition:



architecture:

↳ Transformers

↳ recurrent neural networks

↳ element-wise | feed-forward NNs

$o = c_1 + c_2 + c_3 \Rightarrow$ element-wise addition

↳ issue #1: no position info

let's stick w/ simple addition to define our NLM

↳ NLMs contain parameters θ

$$\theta = \{W, c_1, c_2, c_3 \ldots\}$$

↳ randomly initialized

↳ trained to minimize neg. log likelihood of training dataset

↳ how do we train NLM?

1. define a loss fn

   ↳ tell us how bad we are at predicting the next word

   ↳ $L(\theta) = -\log p(w_n \mid w_1 \ldots n-1)$

   → neg log prob of correct next word in training dataset

2. given $L(\theta)$, we compute the gradient of L WRT θ

   → gradient gives the direction of steepest ascent

   → $\frac{dL}{d\theta} = \left\{ \frac{dL}{dW}, \frac{dL}{dc_1}, \frac{dL}{dc_2}, \frac{dL}{dc_3} \right\}$

→ $\frac{dL}{dW}$ is same dim as W

→ <u>intuition</u>: for each param in θ,

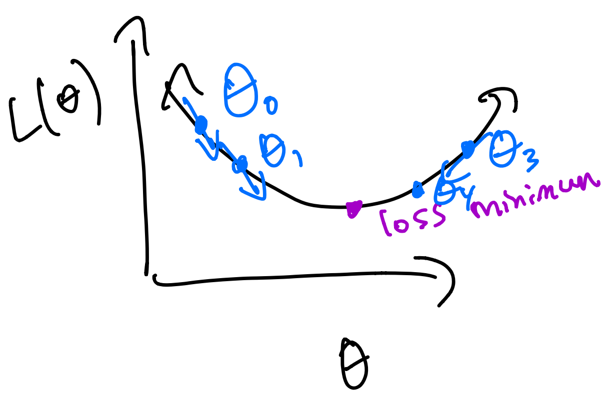
$\frac{dL}{d\theta}$ tells us how much L changes if we increase that param by a tiny amount

3. given gradient $\frac{dL}{d\theta}$, we take a step in the direction of negative gradient

$$\theta_{new} = \theta_{OLD} - \eta \frac{dL}{d\theta}$$

η: learning rate, "step size"

$\frac{dL}{d\theta}$: gradient



<u>optimizers</u>:

↳ Adam

↳ Sophia

↳ SGD

↑ this is called gradient descent

hyperparameters:

- learning rate
- batch size
  - how many training examples do you use to estimate $\frac{dL}{d\theta}$

- algorithm to compute gradient efficiently in neural networks is called backpropagation
  - loss.backward()

- architectures for NLMs
  - recurrent neural network
  - Transformer
    - attention
  - state space models (Mamba)