Final project topics

↳ finetuning / RL on a small LM

↳ "post-training"

↳ Colab Pro / Kaggle

↳ agents

Neural language models

neg. log likelihood (NLL):

$$NLL = \underbrace{\frac{-1}{n}}_{\text{normalize by length}} \sum \log \underbrace{P(w_i \mid w_{<i})}_{\text{prob. of next word}}$$

↳ high P = low NLL
low P = high NLL

↳ training: minimize NLL on training set

↳ evaluation:

↳ measure NLL on test data

↳ PPL = exp(NLL)

↳ effective choices for next word on average

↳ want is PPL(train) and PPL(test) to both be low ⇒ generalization

↳ n-gram models minimize train NLL by counting and dividing

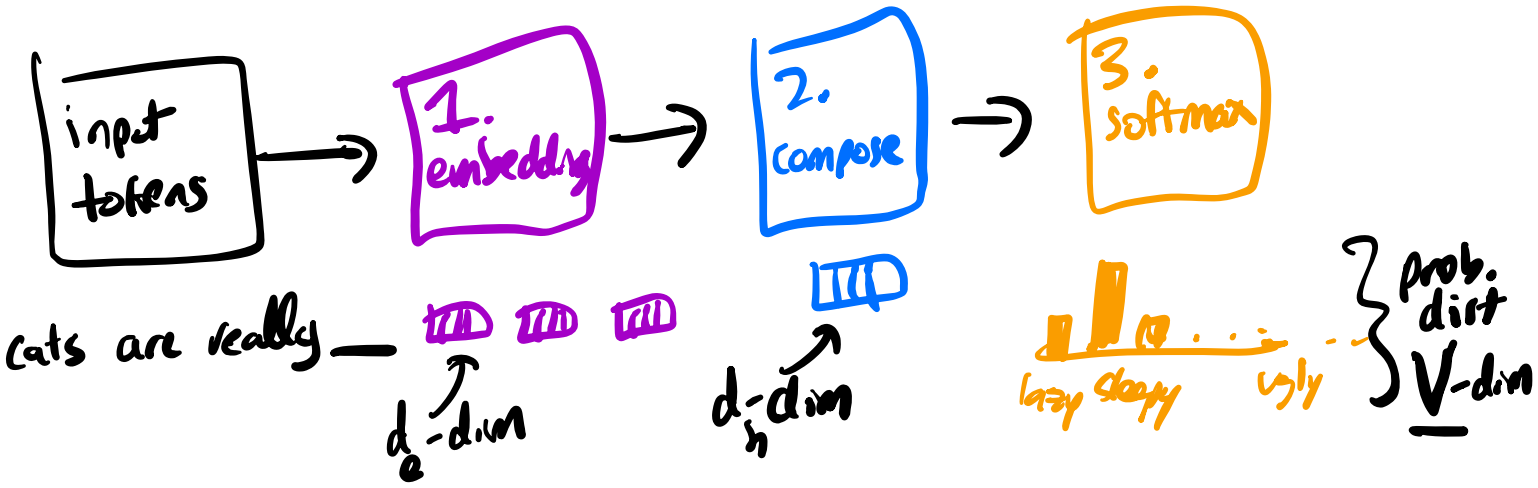
↳ unseen ngrams during training have zero prob.

↳ goal: neural LMs:

minimize training NLL, but no closed form solution

↳ gradient descent

neural LMs:



↳ d_e, d_h usually hundreds / low thousands

512 d, 4096 d

↳ V size usually 100-200k types

---

embeddings

↳ in n-gram models, all word types are equally similar / dissimilar to each other

↳ "movie" and "film" don't share info

↳ "one-hot" word representation

↳ V-dim vector, binary

$\overrightarrow{movie} = \langle 0, 0, 0, \ldots, 1, 0, 0 \rangle$

$\overrightarrow{film} = \langle 0, 0, 1, \ldots, 0, 0, 0 \rangle$

$\overrightarrow{movie} \cdot \overrightarrow{film} = 0$

$\overrightarrow{movie} \cdot \overrightarrow{zebra} = 0$

want: vector space where words/phrases w/ similar meanings have similar representations

idea: switch from sparse one-hot vectors to dense real-valued vectors

$\overrightarrow{movie}: \langle 0.3, -1.4, 5.8 \rangle$ $d=3$

$\overrightarrow{film}: \langle 0.2, -1.8, 6.3 \rangle$

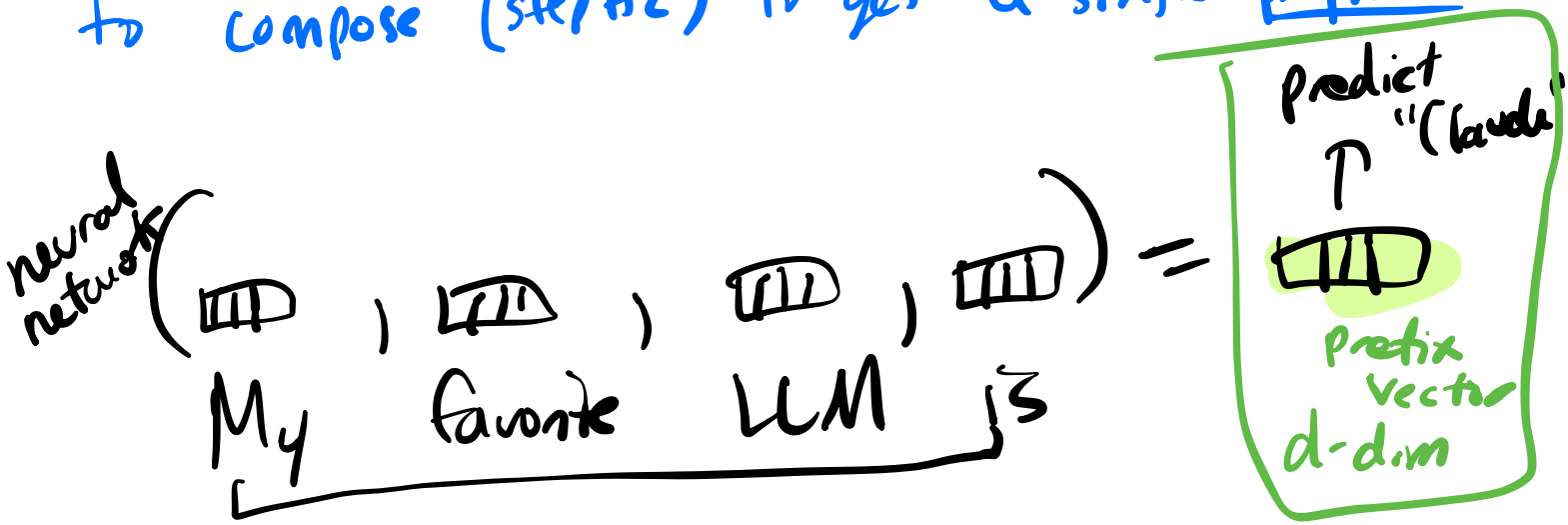
$\overrightarrow{zebra}: \langle 7.9, 3.2, -17.5 \rangle$

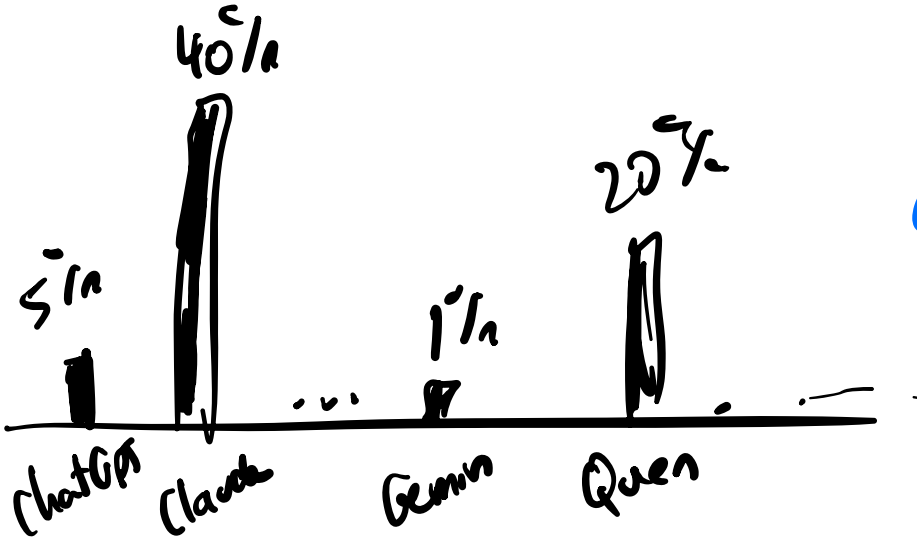
word embeddings, learned during training

$\overrightarrow{movie} \cdot \overrightarrow{film} \approx 39$

$\overrightarrow{movie} \cdot \overrightarrow{zebra} \approx -104$

now, given a sequence of word embs associated w/ an input prefix, we need to compose (step #2) to get a single prefix vector.





output dist
↳ V-dim
↳ vector sums to 1
↳ non-negative

Softmax layer:

↳ goal: convert a d-dim dense vector into a V-dim probability distribution over next word

example: $V = \{$ ChatGPT, Claude, Gemini, Qwen $\}$

given prefix vector  ← x

↳ $x = \langle -2.3, 0.9, -5.4 \rangle$

$d = 3, \quad V = 4$

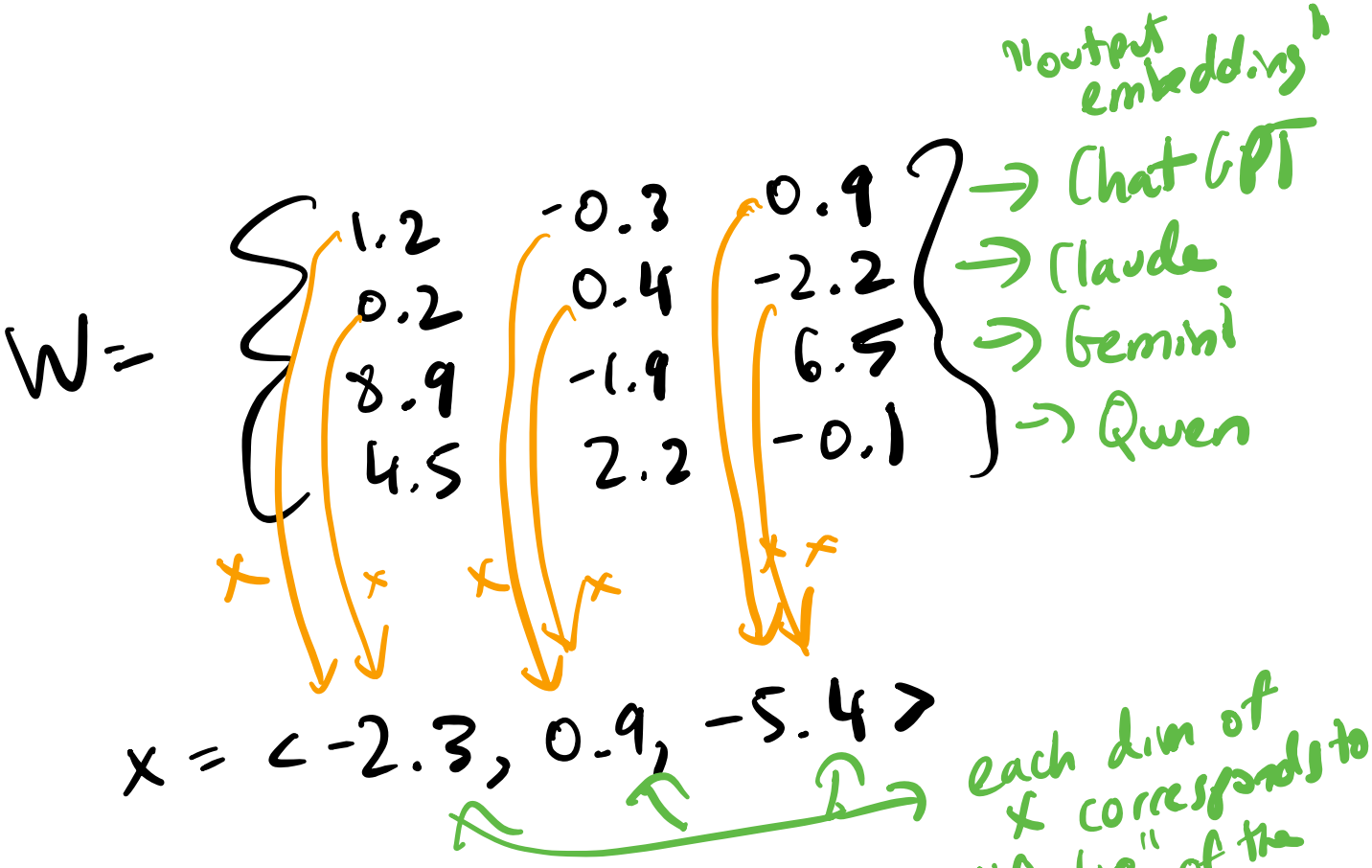
Step 1: project x (3-d) to V (4-dim)

using a **weight matrix** W

↳ weight matrix contains **parameters** that we **update** during training to change the next word distribution

$$W = \begin{Bmatrix} 1.2 & -0.3 & 0.9 \\ 0.2 & 0.4 & -2.2 \\ 8.9 & -1.9 & 6.5 \\ 4.5 & 2.2 & -0.1 \end{Bmatrix}$$ 4x3 matrix

matrix vector product Wx is a 4-d vector



↳ step 1 embeddings

↳ embedding matrix $E : V \times d$ matrix,

each row corresponds to a word embedding

↳ softmax fn to convert logits to probabilities

$$\text{Softmax}(x) = \frac{e^{x}}{\sum_{j} e^{x_j}}$$

$$\text{Softmax}(Wx) =$$

$$\langle 1.5 \cdot 10^{-5}, 1.6 \cdot 10^{-11}, 0.999984, 3.4 \cdot 10^{-10} \rangle$$

$P(\text{ChatGPT prefix})$ $P(\text{Claude prefix})$ $P(\text{Gemini})$ $P(\text{Qwen})$