Important Announcements and Notes (Dec 3)

- Added video summarizing the different types of errors on Youtube
- Changed activation function to depend on the layer $h^{(b)}$. Different activations are allowed at different layers
- Model $\Leftrightarrow$ Predictor
- Companies release models with various levels of accesss

  Black box access (No architecture or weights):

  Chat GPT, Gemini, Claude

  Open Source (Architecture and weights):

  LLaMA, Gemma

- We studied a NN called a Multilayer Perceptron (MLP)

  Other types of NNs include: CNN, RNN, Transformer

  "GPT": Generative Pre-trained Transformer

- Deep learning refers to using NNs with many layers (B is large)

- "Learning" refers to the process of improving the weights with gradient descent
  - We don't hard code the weights

- Computing gradients of $\hat{L}$ for a NN function class is called "Back Propagation"

Language Models

Task: generating text

Ex: Input: "Once upon a time, in a land far away,"
Output: "there lived a wise old dragon who loved to read books."

Ex: Input: "Why did the chicken cross the road?"
Output: "To get to the other side."

Ex: How ChatGPT works

Input:

"Your are an AI assistant trained to provide accurate, concise, and helpful responses to user inquiries."

+

"Why did the chicken cross the road?"

Output: "To get to the other side."

We will only study auto-regressive models

Ex: ChatGPT, Gemini, Claude

Auto-regressive: generates output sequentially by using its previous outputs and inputs

Ex: Input: "Why did the chicken cross the road?"

Output: "To"

Input: "Why did the chicken cross the road?
To"

Output: "get"

Input: "Why did the chicken cross the road?
To get"

Output: "to"

⋮

Input: "Why did the chicken cross the road?
To get to the other side."

Output: "<EOS>" End Of sequence token

We want a model $f: \mathcal{X} \to \mathcal{Y}$ where

$\vec{x} \in \mathcal{X}$ is a sequence of ~~words~~ tokens

$f(\vec{x}) \in \mathcal{Y}$ is the next ~~word~~ token

$\mathcal{Y} = \{$ all words + punctuation + "<EOS>" + "<PAD>" $\}$

$= \{1, \ldots, K\}$ Vocabulary $|\mathcal{Y}| = K$

token $\in \mathcal{Y}$

Predicting discrete labels causes problems with optimization

Lets predict the probability of each token and then pick the token with the largest probability

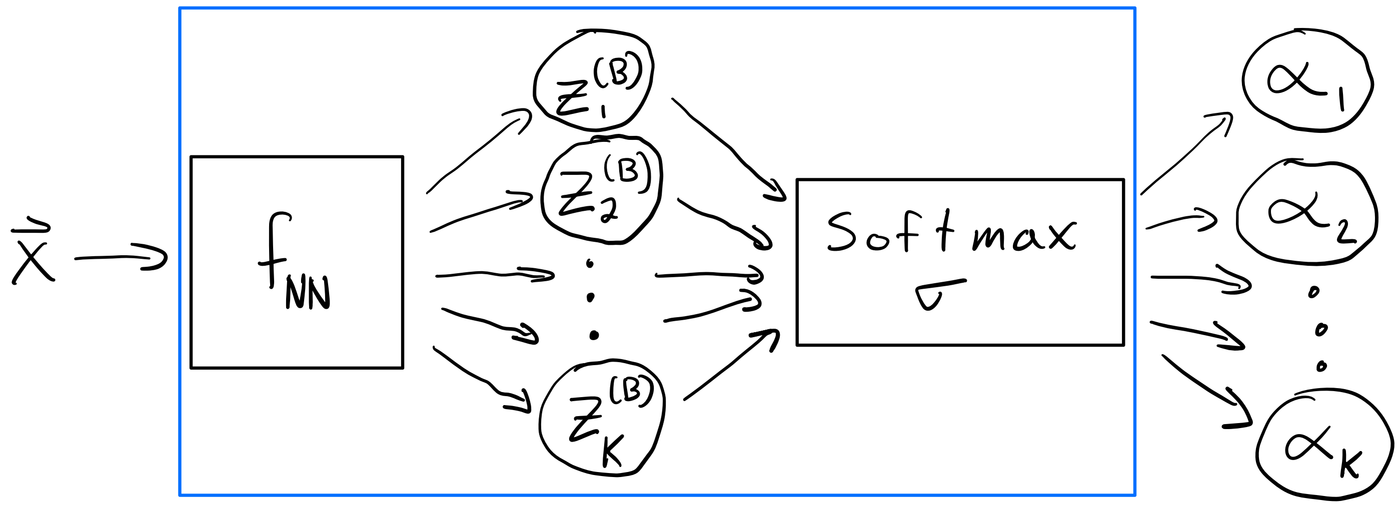
We want a model $f_{prob}: \mathcal{X} \to \mathcal{Y}_{prob}$ where

$\vec{x} \in \mathcal{X}$ is a sequence of tokens

$f_{prob}(\vec{x}) \in \mathcal{Y}_{prob}$ is a vector of probabilities of all possible next tokens

$$\mathcal{Y}_{prob} = [0,1]^K$$

We can use a NN $f_{NN}(\vec{x})$ with $h^{(B)}(z) = z$ and then apply softmax to get a probability



$f_{prob}(\vec{x}) = \sigma(f_{NN}(\vec{x})) \in \mathcal{Y}_{prob}$ $\qquad \sum_{q=1}^{K} \alpha_q = 1, \ \alpha_q \in [0,1]$

Ex of f_{NN}: Transformer, Recurrent NN (RNN)

How do we represent a sequence of tokens s as a vector $\vec{x} \in \mathbb{R}^{d+1}$?

$s = (v_1, \dots, v_c)$

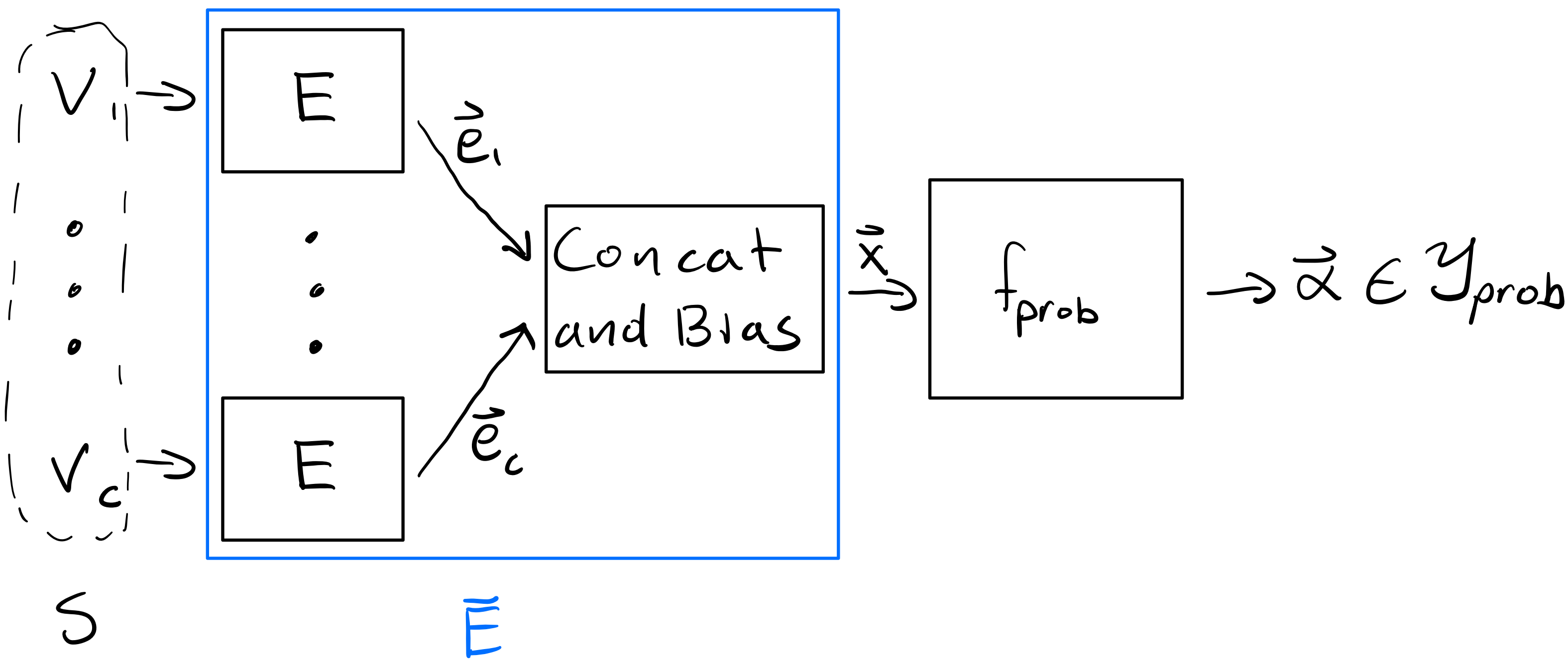
Ex: "Why did the chicken"

$s = (\text{"Why"}, \text{"did"}, \text{"the"}, \text{"chicken"})$

$E : \mathcal{Y} \to \mathbb{R}^{d'}$ Embedding function

tokens vectors

assume it is given



$\bar{E}(s) = \vec{x} \in \mathcal{X} = \mathbb{R}^{d+1}$ where $d = cd'$

context length

s is a sequence of at most c tokens

To handle sequences shorter than c words a padding token "<PAD>" is added

Ex: $c = 3$

$$S = (\text{"Why"}, \text{"did"}) \Rightarrow (\text{"<PAD>"}, \text{"Why"}, \text{"did"})$$

Creating a Dataset

$$S = \Big(\underbrace{\underbrace{v_1}_{s_1}, \underbrace{\overbrace{v_2}^{s_2}}_{y_1}, \overbrace{v_3}^{y_2}, \dots, v_c}_{S_c}, \underbrace{\overbrace{v_{c+1}}^{S_{c+1}}}_{y_c}, \overbrace{v_{c+2}}^{y_{c+1}}, \dots, v_a\Big)$$

__Ex:__ $C=2$, $S = (\text{"Why"}, \text{"did"}, \text{"the"})$

$s_1 = (\text{"<PAD>"}, \text{"Why"})$, $y_1 = \text{"did"}$

$s_2 = (\text{"Why"}, \text{"did"})$, $y_2 = \text{"the"}$

Repeat for all the sequences of tokens that you have

$\vec{x}_1 = \bar{E}(s_1)$, $\vec{x}_2 = \bar{E}(s_2)$, $\dots$, $\vec{x}_n = \bar{E}(s_n) \in \mathbb{R}^{d+1}$

$\vec{y}_1 = \text{onehot}(y_1) \in \{0,1\}^K \subset [0,1]^K$

$\vec{y}_2 = \text{onehot}(y_2) \in \{0,1\}^K$

$\vdots$

$\vec{y}_n = \text{onehot}(y_n) \in \{0,1\}^K$

$$D = ((\vec{x}_1, \vec{y}_1), \dots, (\vec{x}_n, \vec{y}_n))$$

ERM Learner:

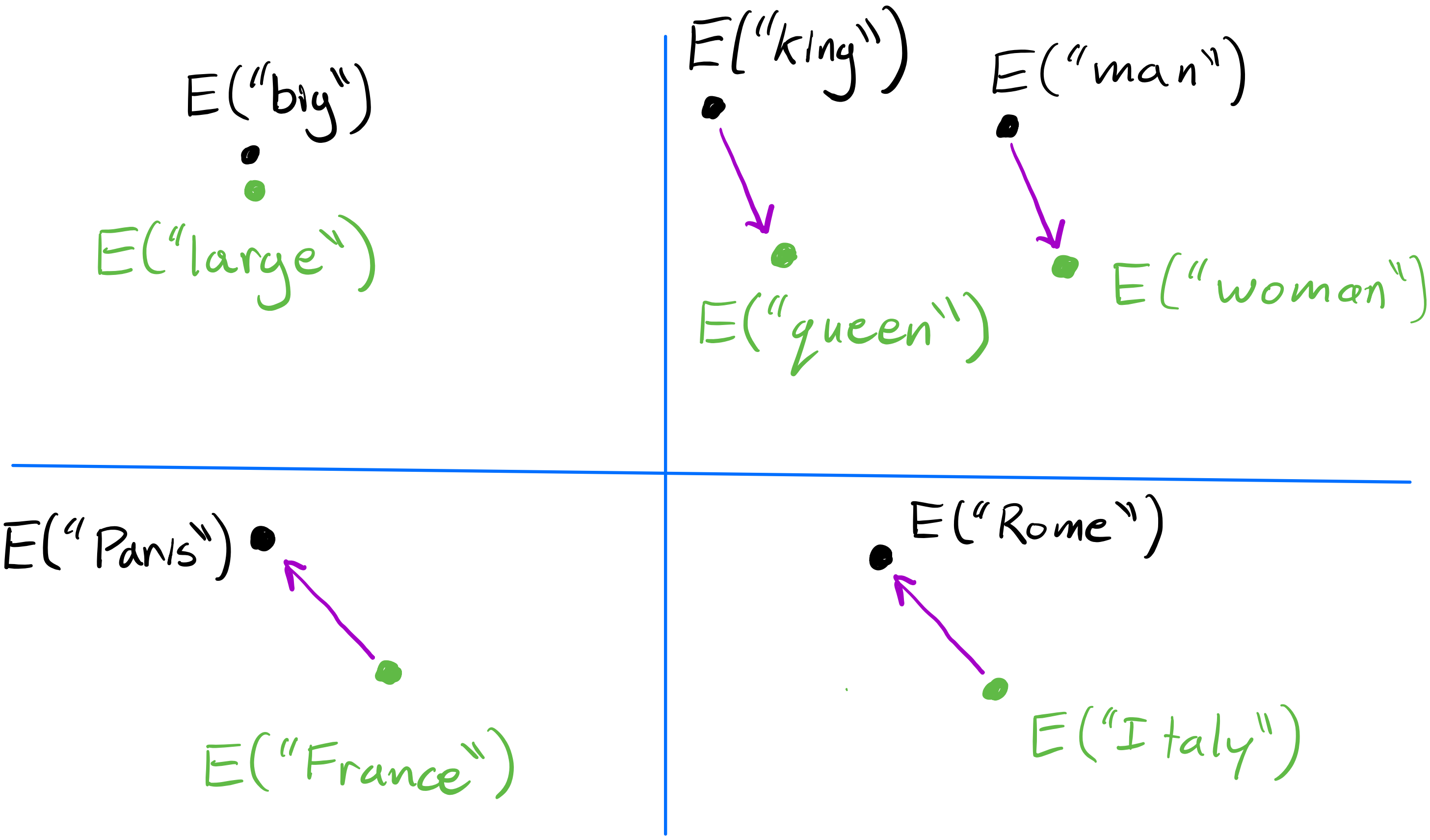
$$\mathcal{A}(D) = \underset{f \in \mathcal{F}}{\arg\min}\ \hat{L}(f)$$

$\mathcal{F} = \{ f \mid f : \mathcal{X} \to \mathcal{Y}_{prob}$ where $f = \sigma(f_{NN})$ and f_{NN} is a NN with a fixed architecture $\}$

ℓ is multiclass cross-entropy loss

Embeddings

Ex: $d' = 2$ $\qquad E : \mathcal{Y} \to \mathbb{R}^{d'}$



Notes

- Embedding E can be learned
- Vocabulary can use characters instead of words or sub-words
- Most probable word is not always chosen, instead can sample based on probabalities
- "Large" language models (LLM) means a NN with a lot of weights

  Ex: GPT-3 has 175 billion weights

  LLaMA 3 405B has 405 billion weights

  The brain has ~ 100 trillion connections between neurons