CS 1671 / CS 2071 / ISSP 2071
Human Language Technologies

Session 18: Transformers part 2

Michael Miller Yoder

March 23, 2026

School of Computing and Information

Quiz

- Go to **Quizzes > Quiz 03-23** on Canvas.
  - Session 11: J+M 4.7-4.10, 4.12
  - Session 12: J+M 5-5.2, 5.5-5.8, 5.10
  - Session 13: J+M 6-6.1, 6.3-6.4
  - Session 14: J+M 6.5-6.6
  - Session 16: J+M 7-7.5, 7.7-7.8
- You have until **1:10pm** to complete it

- Allowed resources
  - Textbook
  - Your notes (on a computer or physical)
  - Course slides and website
- Resources not allowed
  - Generative AI
  - Internet searches

Assessments: project

- Project progress report is due **this Thu Mar 26**
- Part 1: Basic data analysis (if any updates are required from the proposal)
- Part 2: Result from baseline approach
  - Ideally performance metric result from the baseline system you proposed
- Part 3: LLM proposal
  - How might you use an LLM programmatically to attempt your task?
  - Zero-shot and more advanced approaches

SCI open-source LLMs available for the project

- Exact models available (feel free to use any)
  - gemma3 (gemma 3:27b)
  - llama3.1 (llama3.1:70b)
  - deepseek-r1 (deepseek-r1:70b)
- There are no rate limits. Just keep in mind it is a shared resource

Overview: Transformers part 2

- Transformer blocks
- Transformer input and output details
  - Position embeddings
  - Language modeling head
- Activity: draw a diagram of a transformer-based LM
- Coding activity: Prompt SCI open-source LLMs through an API

DataSci + AI Forum 2026 from Pitt's Hub for AI Leadership (HAIL)

- Info and registration: https://www.aihub.pitt.edu/network/datasciai-forum
- Network with professionals, faculty, and industry leaders
- Engage in conversations on responsible AI and emerging technologies
- Apply to be a HAIL Ambassador
  - https://pitt.co1.qualtrics.com/jfe/form/SV_24519cEVX0CZDCe
  - Free transportation to and from the Heinz History Center
- **2 points extra credit** for attending Day 2 (Mar 27) at the Heinz History Center
  - Take a picture of yourself at the event and upload to a Canvas assignment

Review: Describe self-attention in transformers

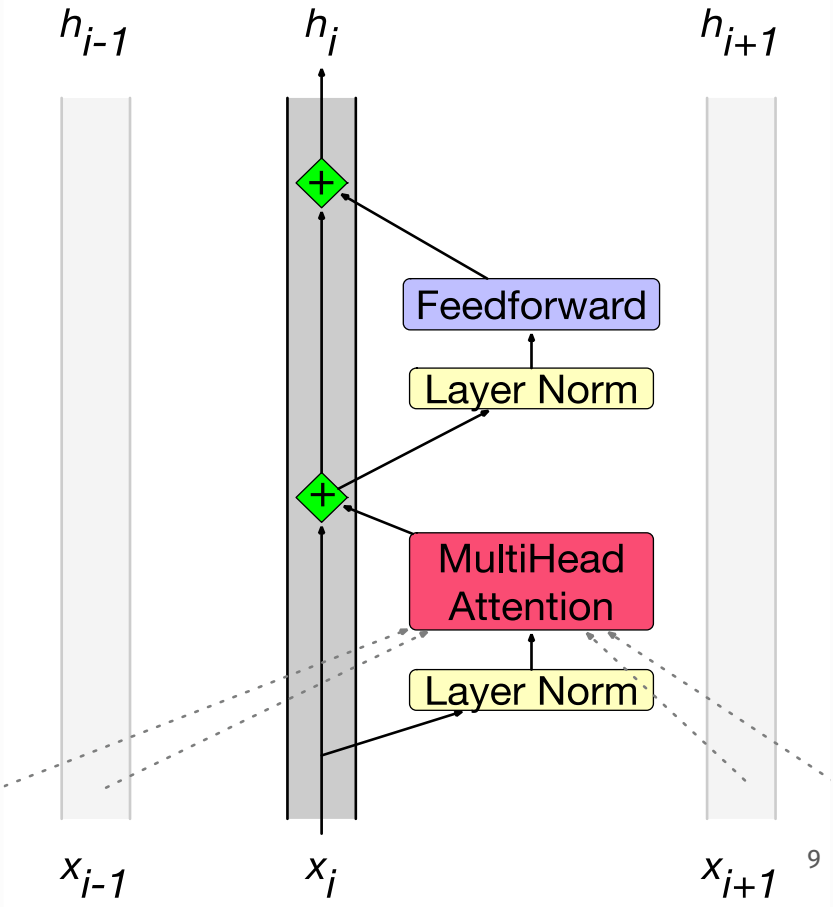
Transformer blocks

Transformer blocks

Each block consists of:

- Self-attention
- Layer normalization and residual connections: tricks to optimize learning
- Feedforward neural network

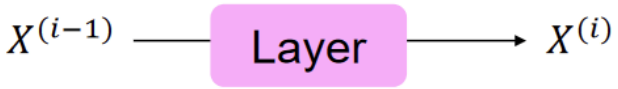
Output: 1 vector for every input token



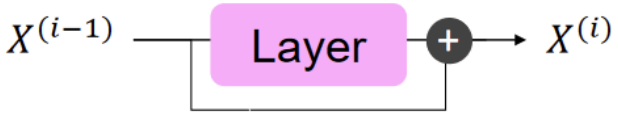
Slide adapted from Jurafsky and Martin, John Hewitt

Residual connections [He et al. 2016]

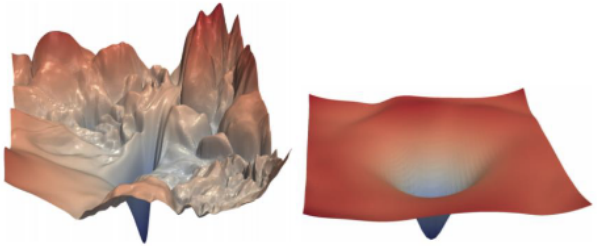
- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Slide credit: John Hewitt

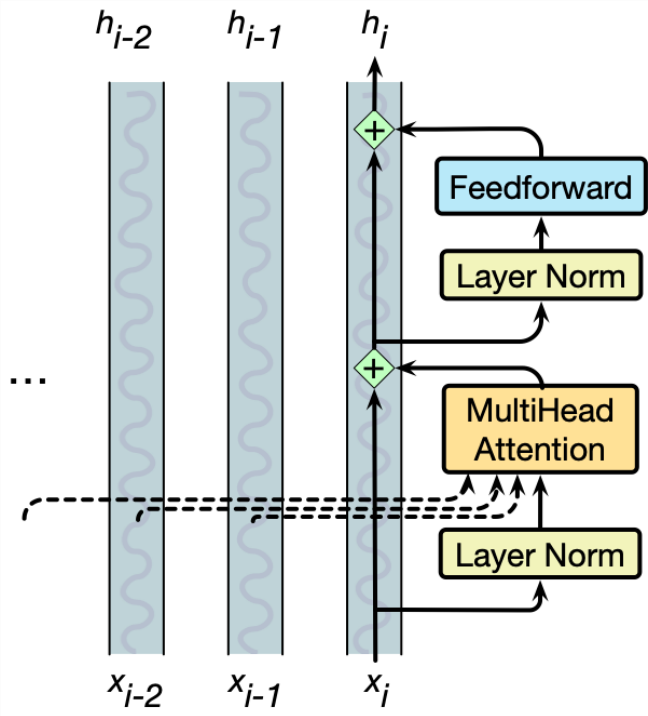
Layer norm: the vector $\mathbf{x}_i$ is normalized twice

- Cuts down on uninformative variation to train faster
- Variation of the z-score from statistics, applied to each token's vector in a hidden layer
- Original implementation had a couple of other learnable parameters, but these can be omitted

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

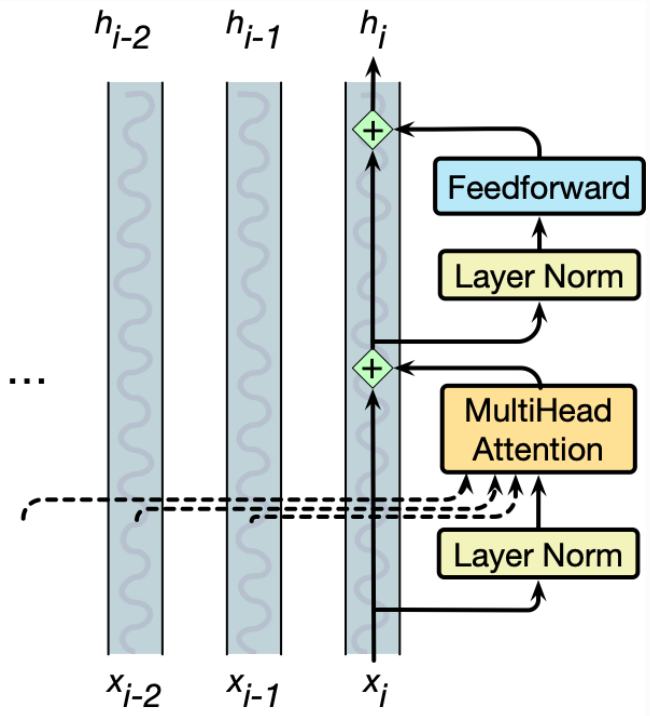
$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$



Slide adapted from Jurafsky and Martin, John Hewitt

Putting together a single transformer block



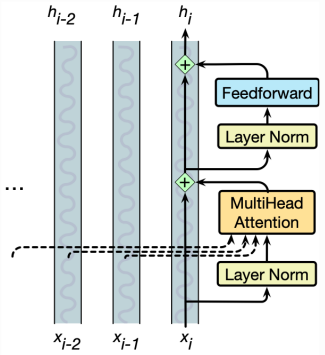
$$
\begin{aligned}
\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \cdots, \mathbf{x}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$

Slide adapted from Jurafsky and Martin

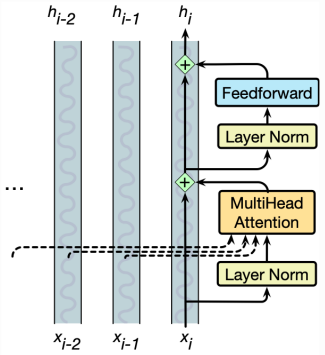
A transformer is a stack of these blocks

- Vectors all have the same dimensionality *d*
- *d* is usually around 512 or 768
- Output of one transformer block becomes the input of another

Block 2



Block 1



Slide adapted from Jurafsky and Martin

Transformer input and output

Token and Position Embeddings

- The matrix X (of shape $[N \times d]$) has an embedding for each word in the context.
  - N is the total number of word instances (tokens)
  - d is the dimension of word embeddings, a hyperparameter
- This embedding is created by adding two distinct embedding for each input: **token** and **position** embeddings
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values

Slide adapted from Jurafsky & Martin

Token Embeddings

Embedding matrix E has shape $|V| \times d$

- One row for each of the $|V|$ tokens in the vocabulary.
- Each word is a row vector of d dimensions

Given: string "*Thanks for all the*"

1. Tokenize with BPE and convert into vocab indices

w = [5,4000,10532,2224]

2. Select the corresponding rows from E, each row an embedding

(row 5, row 4000, row 10532, row 2224).

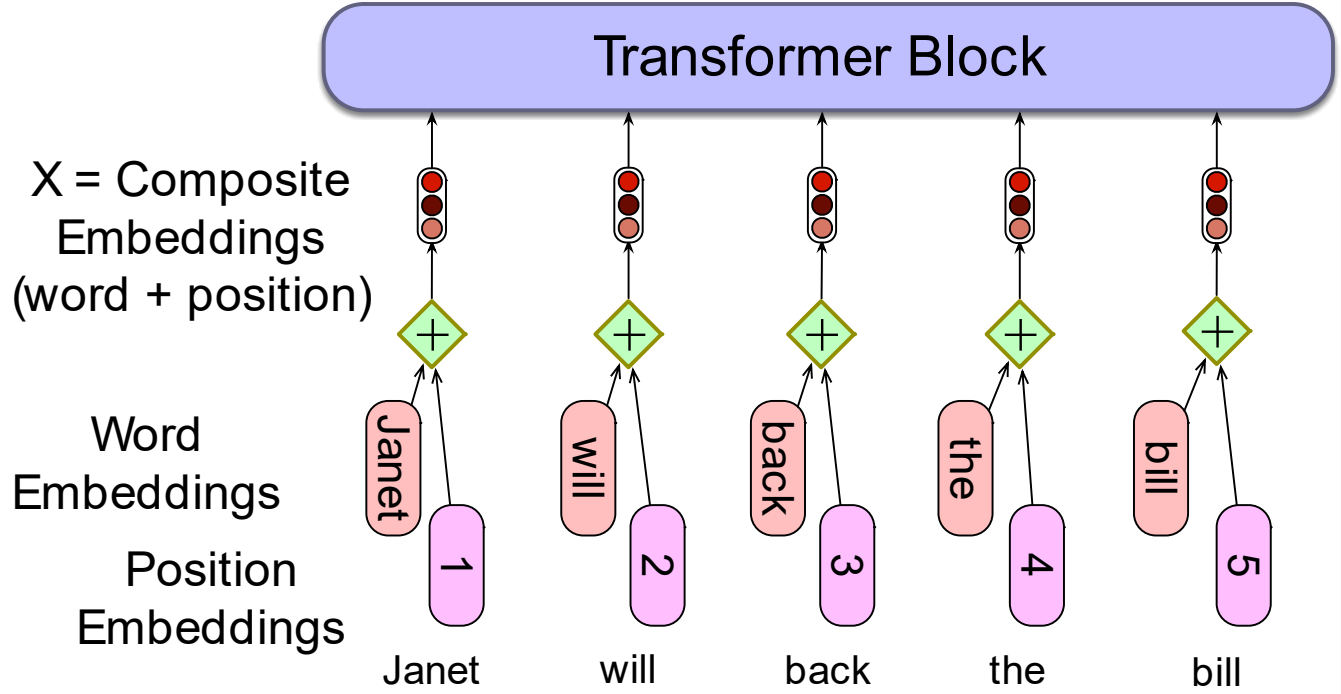
Slide adapted from Jurafsky & Martin

Position Embeddings

- There are many methods, but we'll just describe the simplest: absolute position.
- Learn a position embedding matrix E_{pos} of shape $d \times N$
- Start with randomly initialized embeddings
  - one for each integer up to some maximum length.
  - i.e., just as we have an embedding for the word *fish*, we'll have an embedding for position 3 and position 17.
- As with word embeddings, these position embeddings are learned along with other parameters during training.

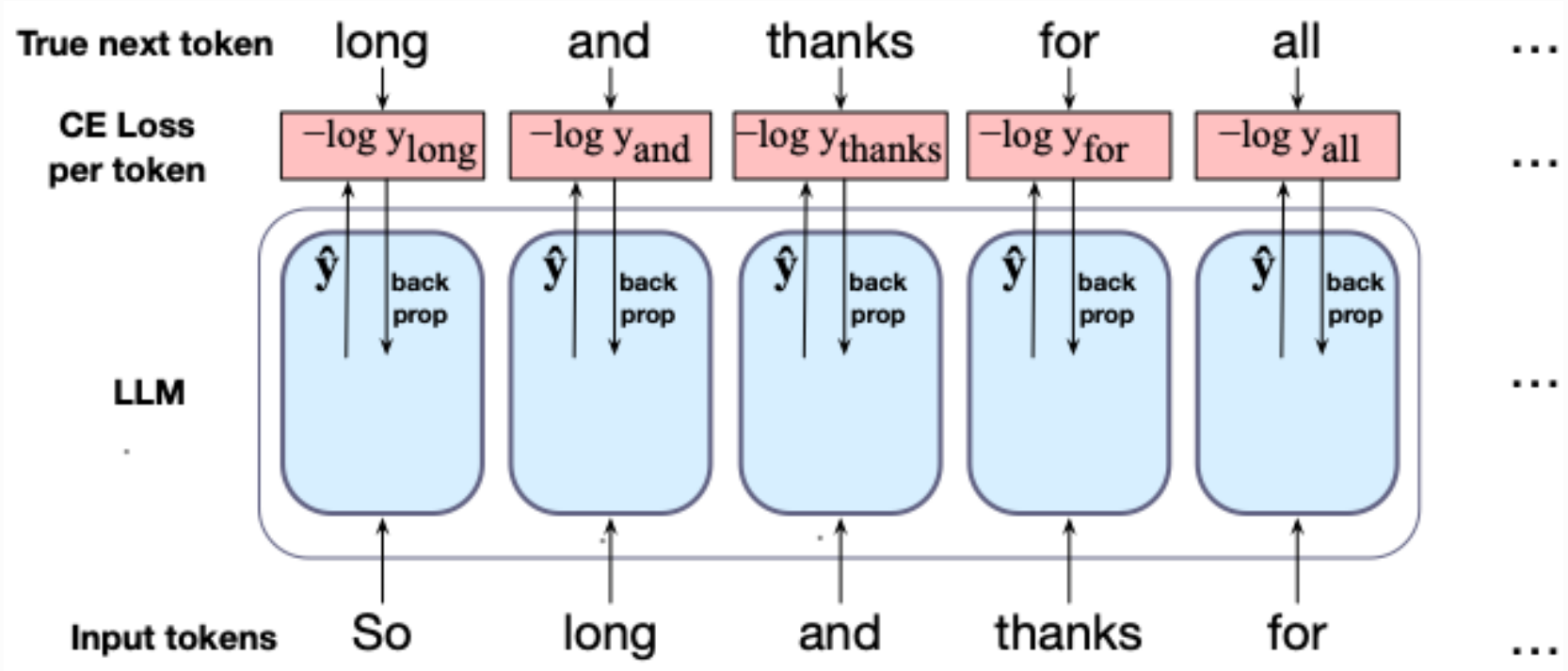
Slide adapted from Jurafsky & Martin

Each **x** is just the sum of token (word) and position embeddings



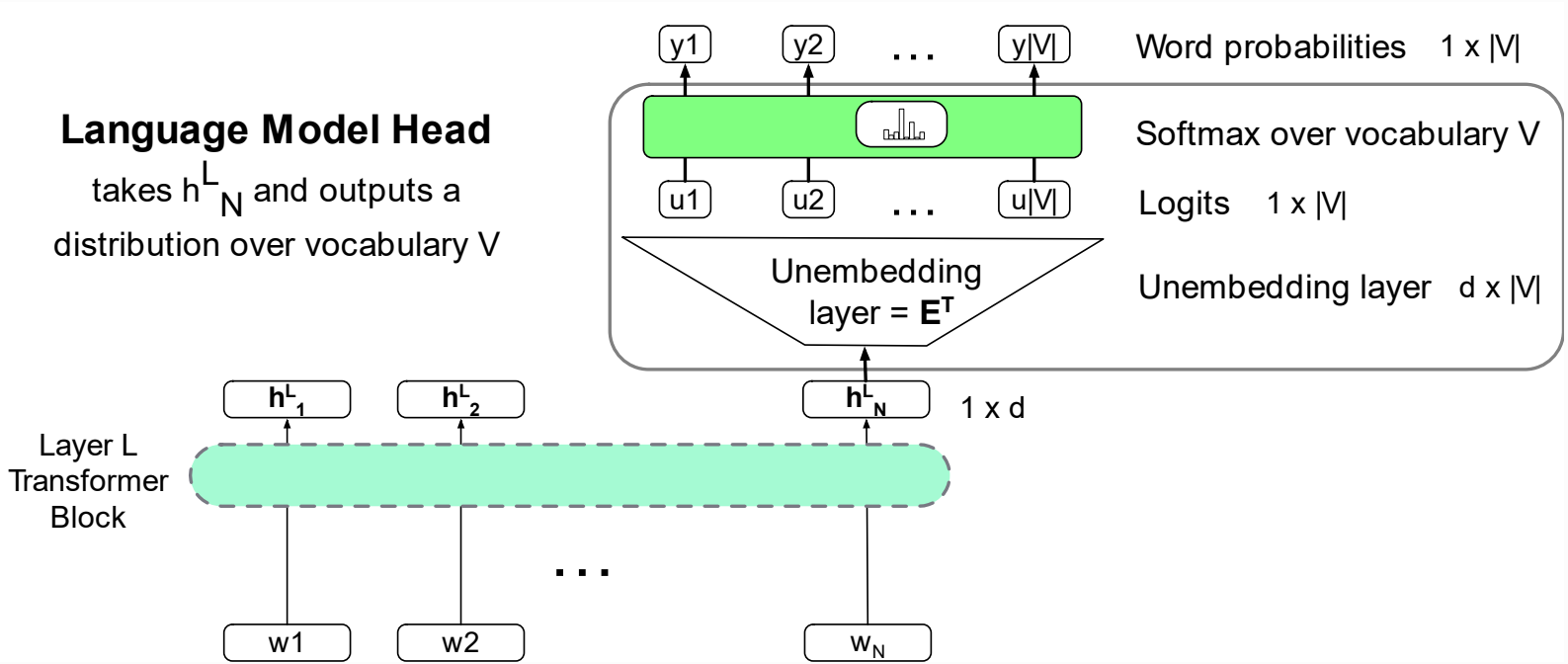
Slide adapted from Jurafsky & Martin

Reminder: training a transformer language model



Slide adapted from Jurafsky and Martin

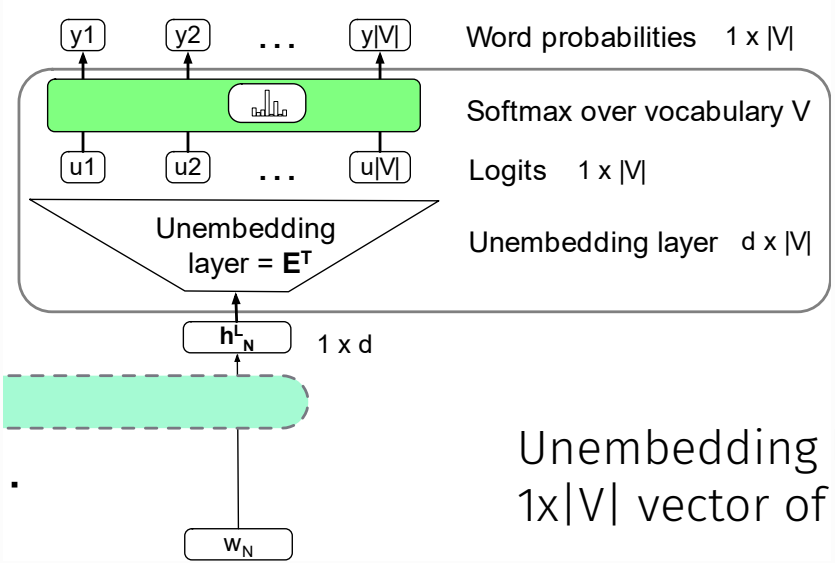
Output: Language modeling head



Slide adapted from Jurafsky & Martin

Language modeling head

Unembedding layer: FFN layer projects from h^L_N (shape $1 \times d$) to probability distribution vector over the vocabulary



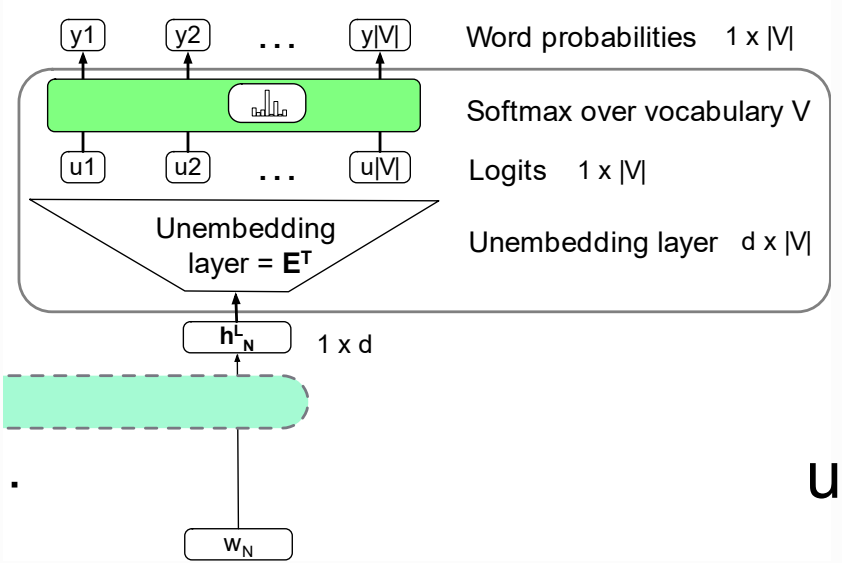
Why "unembedding"? **Tied** to E^T

Weight tying, we use the same weights for two different matrices

Unembedding layer maps from an embedding to a 1x|V| vector of logits

Slide adapted from Jurafsky & Martin

Language modeling head



Logits, the score vector u

One score for each of the $|V|$ possible words in the vocabulary V. Shape $1 \times |V|$.

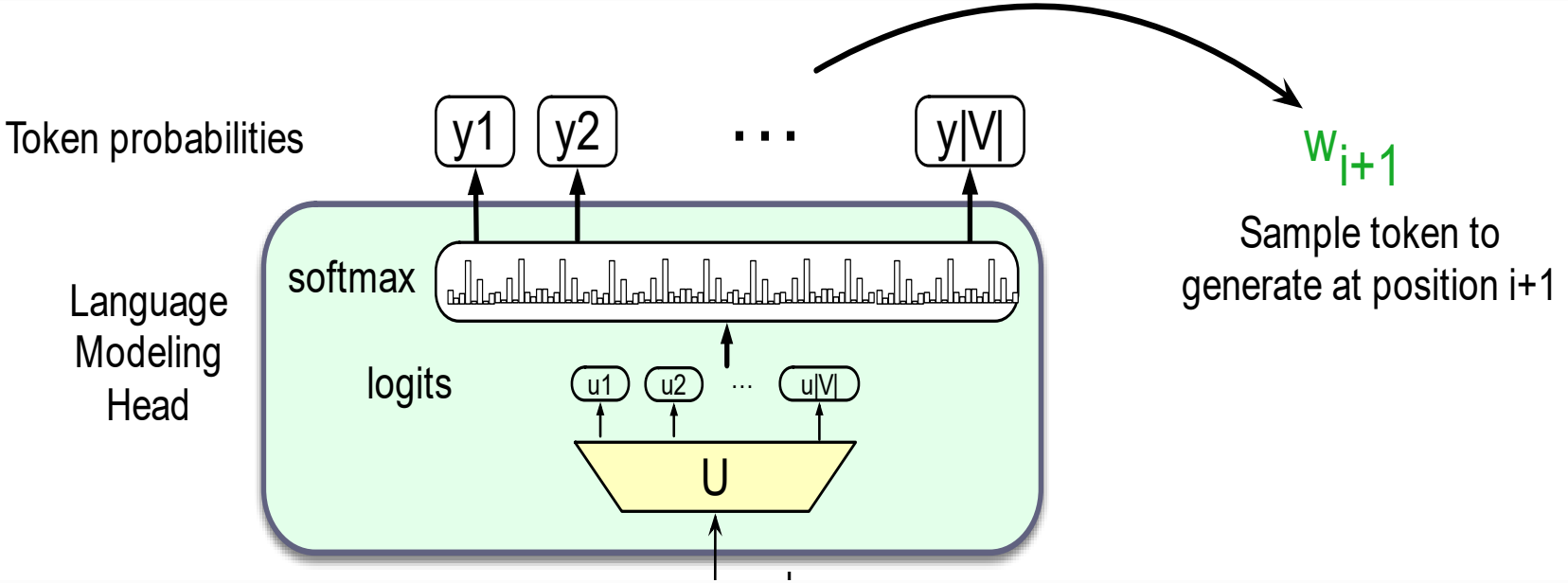
Softmax turns the logits into probabilities over vocabulary. Shape $1 \times |V|$.

$$u = h_N^L E^T$$

$$y = \text{softmax}(u)$$

Slide adapted from Jurafsky & Martin

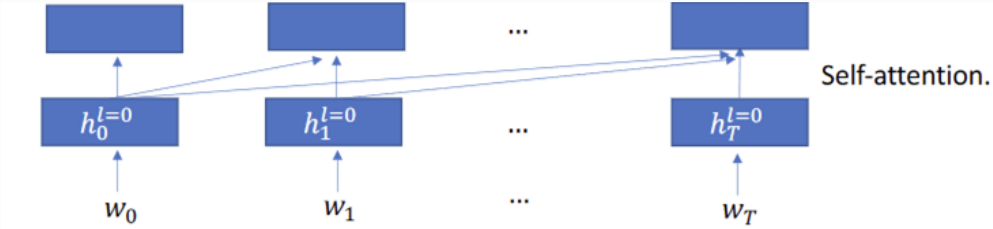
The final transformer language model

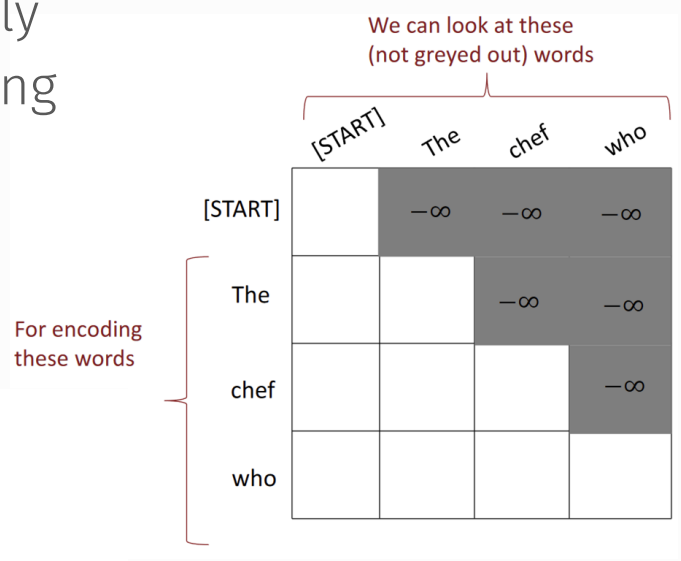


Slide adapted from Jurafsky & Martin

Decoding: apply a “causal mask” for self-attention

- To do auto-regressive LM, we need to apply a “causal” mask to self-attention, forbidding it from getting future context.
- At timestep t, we set a_i = 0 for $i > t$





Slide adapted from Tianxing He, John Hewitt

Activity: draw a diagram of a transformer-based LM

- Input all the way to predicting the next word
- Then explain its parts to a neighbor

Coding activity

Notebook: prompting SCI open-source LLMs

1. Go to this nbgitpuller link (also available on course website)
2. Start a server with **TEACH – 6 CPUs, 48 GB**
3. Load custom environment at **/ix1/cs1671-2026s/class_env**
4. Open **session18_prompting.ipynb**