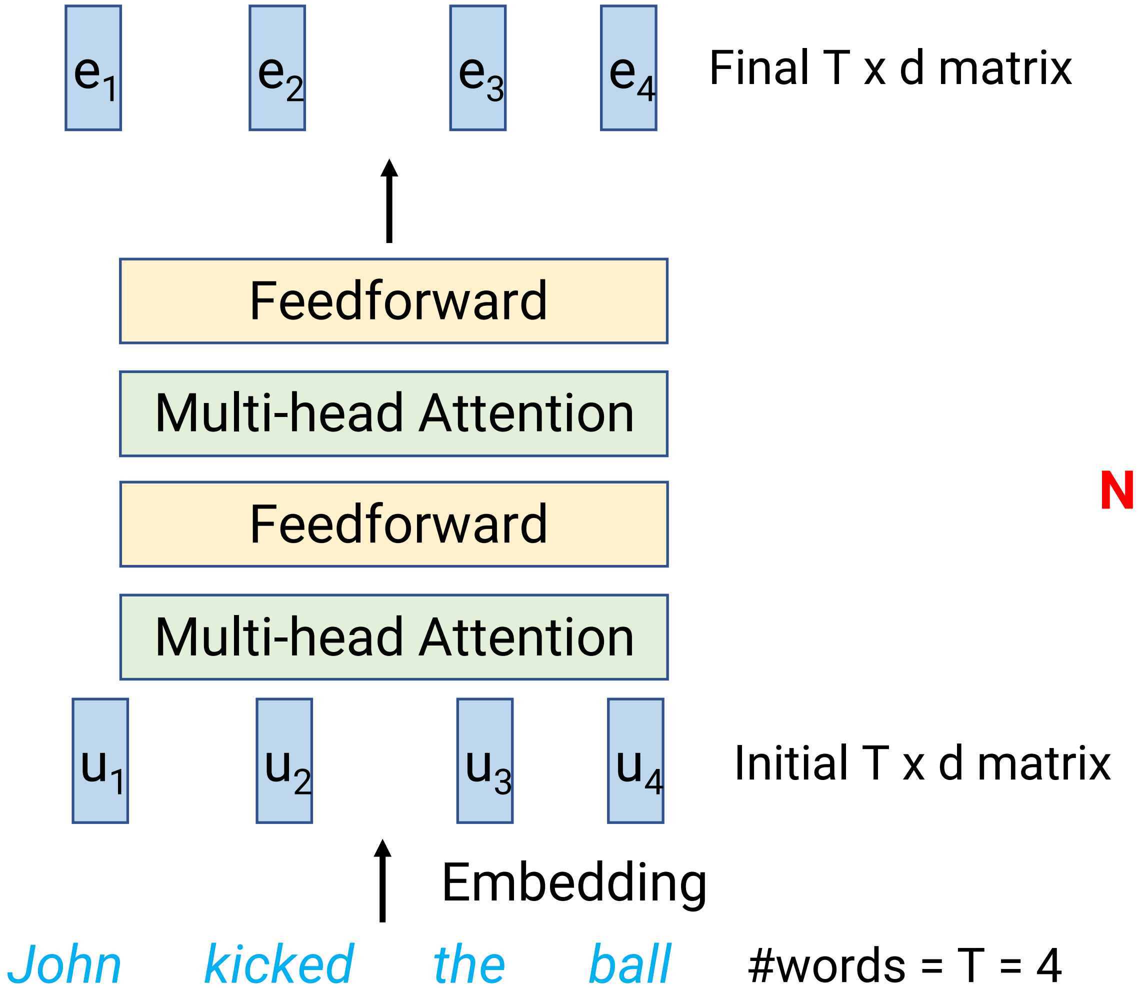
Transformers, Pretraining

Robin Jia
USC CSCI 467, Fall 2023
October 19, 2023

Review: Transformer at a high level

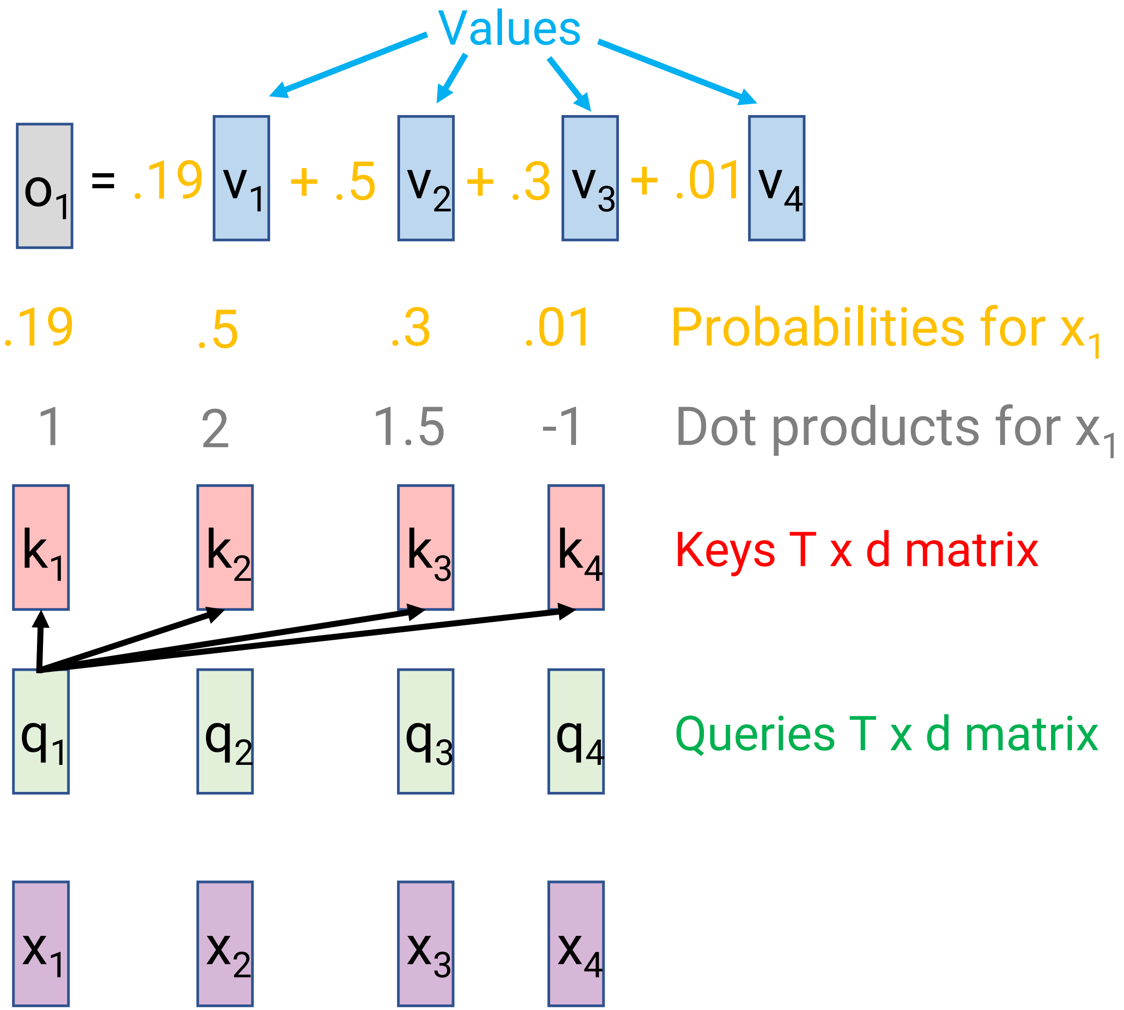


- One transformer consists of
  - Initial embeddings for each word of size d
    - Let T =#words, so initially we have a T x d matrix
  - Alternating layers of
    - **"Multi-headed" attention layer**
    - Feedforward layer
    - Both take in T x d matrix and output a new T x d matrix
  - Plus some bells and whistles…

New!

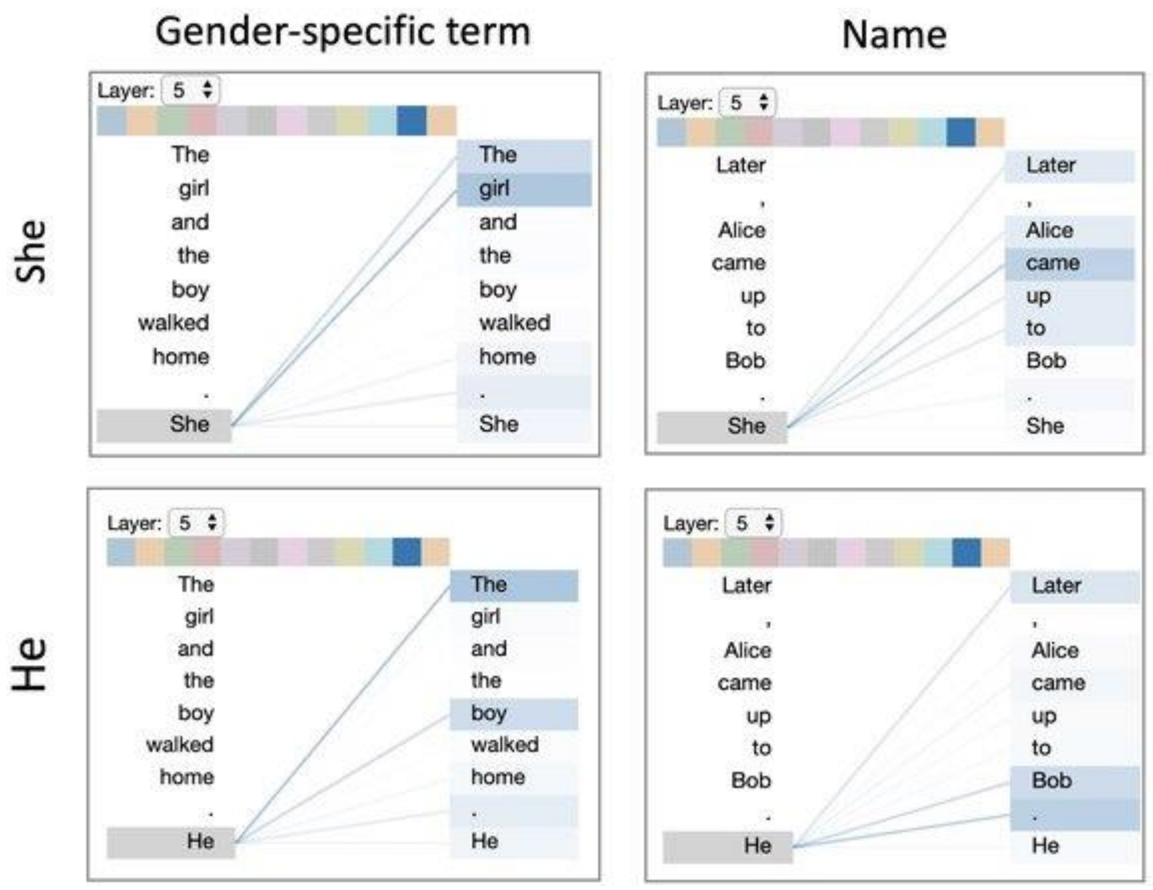
Familiar

Review: Multi-headed Attention



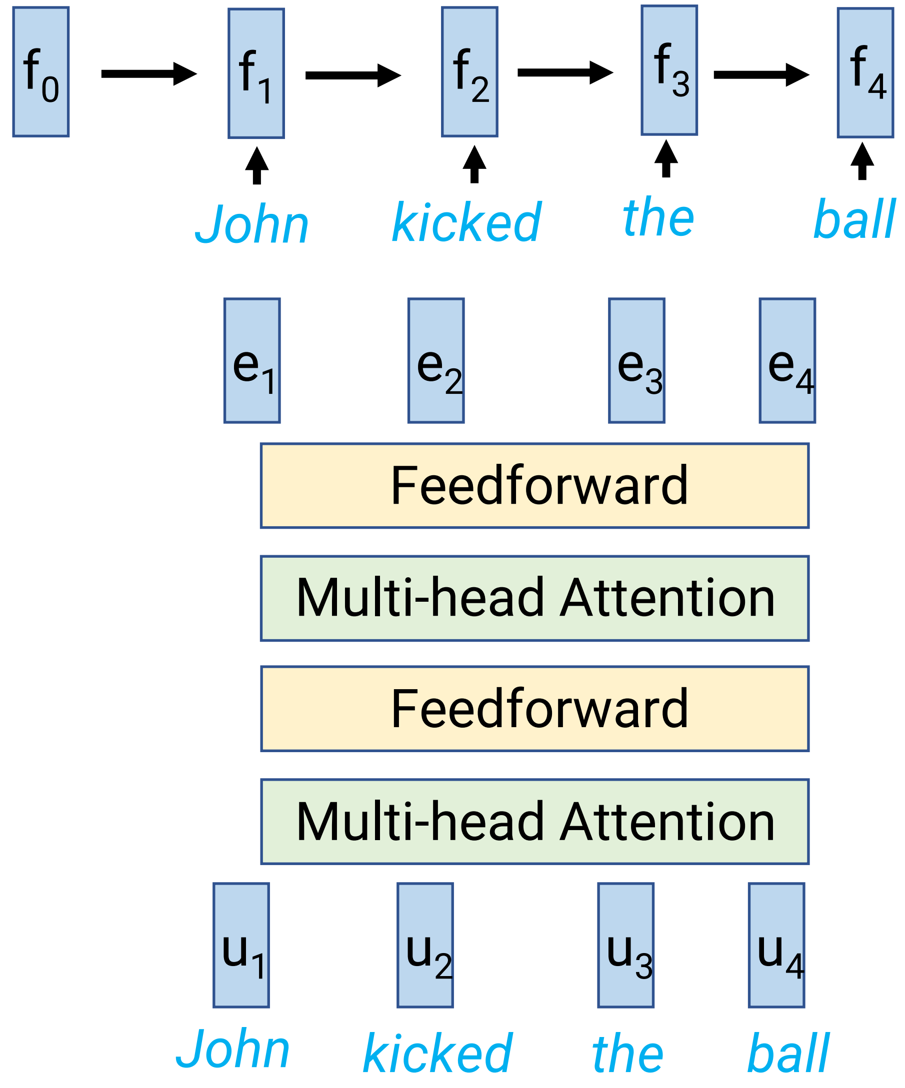
- Input: T vectors $x_1, \ldots, x_T$ each of dimension d
- At each head, apply 3 separate linear layers to each x_t:
  - Query vectors $q_t = W^Q * x_t$
  - Keys vectors $k_t = W^K * x_t$
  - Value vectors $v_t = W^V * x_t$
  - Each linear layer has its own parameters maps from dimension d to dimension d_{attn}
- To compute output o_t:
  - Dot product q_t with each key vector k_i
  - Apply softmax to get probabilities p_i
  - Compute $o_t = \sum_{i=1}^{T} p_i * v_i$
- Have n heads with n different sets of parameters, then concatenate results
  - Choose $d_{attn} = d/n$ so output is also dimension d
- Parameters W^Q, W^K, W^V for each head must be learned by gradient descent
- **Multi-headed attention is the most important idea of Transformers**

What do attention heads learn?



- This attention head seems to go from a pronoun to its antecedent (who the pronoun refers to)
- Other heads may do more boring things, like point to the previous/next word
  - In this way, can do RNN-like things as needed
  - But attention also can reach across long ranges

Runtime comparison

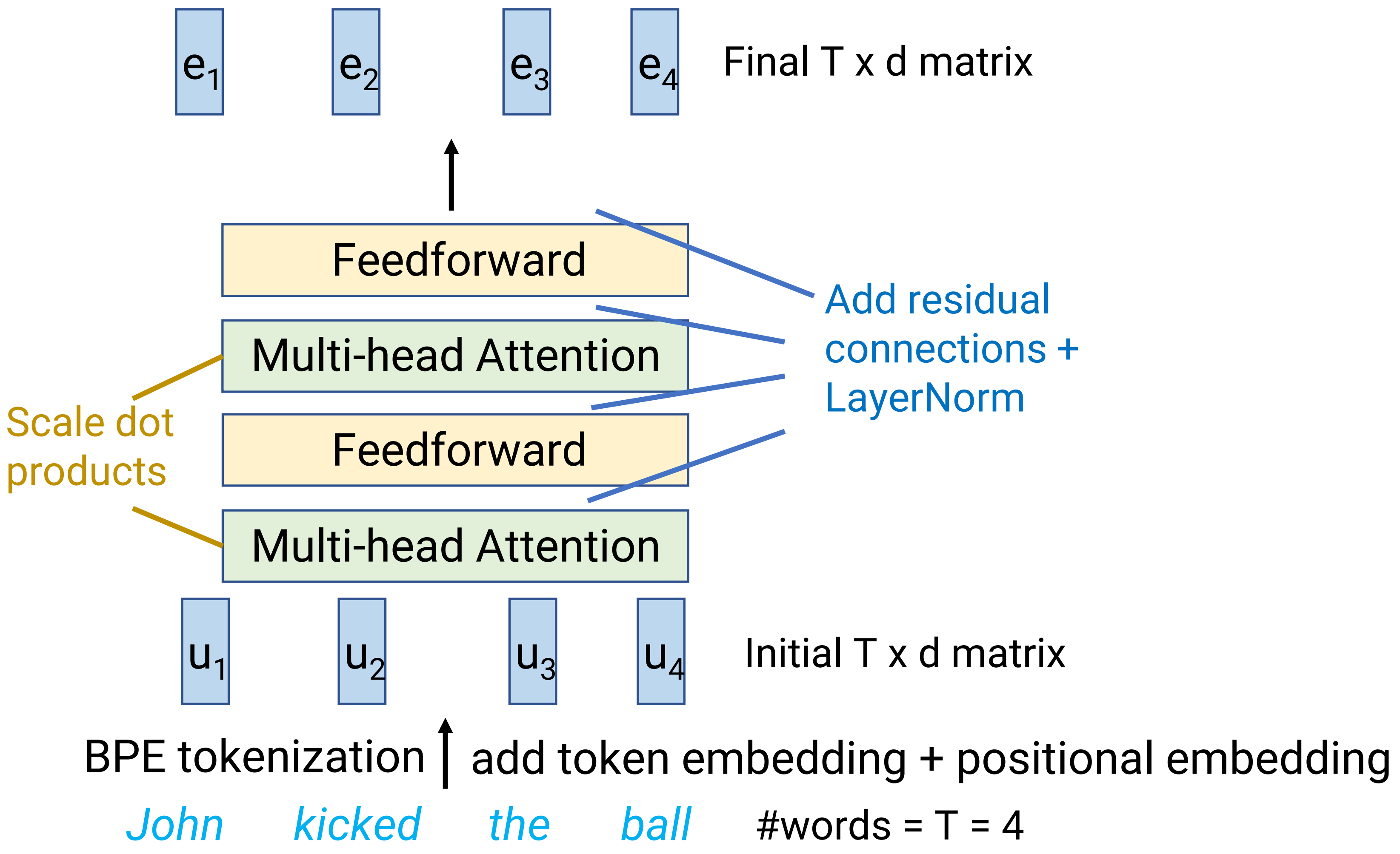


- RNNs
  - Linear in sequence length
  - But all operations have to happen in series
- Transformers
  - Quadratic in sequence length (T x T matrices)
  - But can be parallelized (big matrix multiplication)

Today's Plan

- Transformers in full detail
- Transformer decoders
- Pre-training

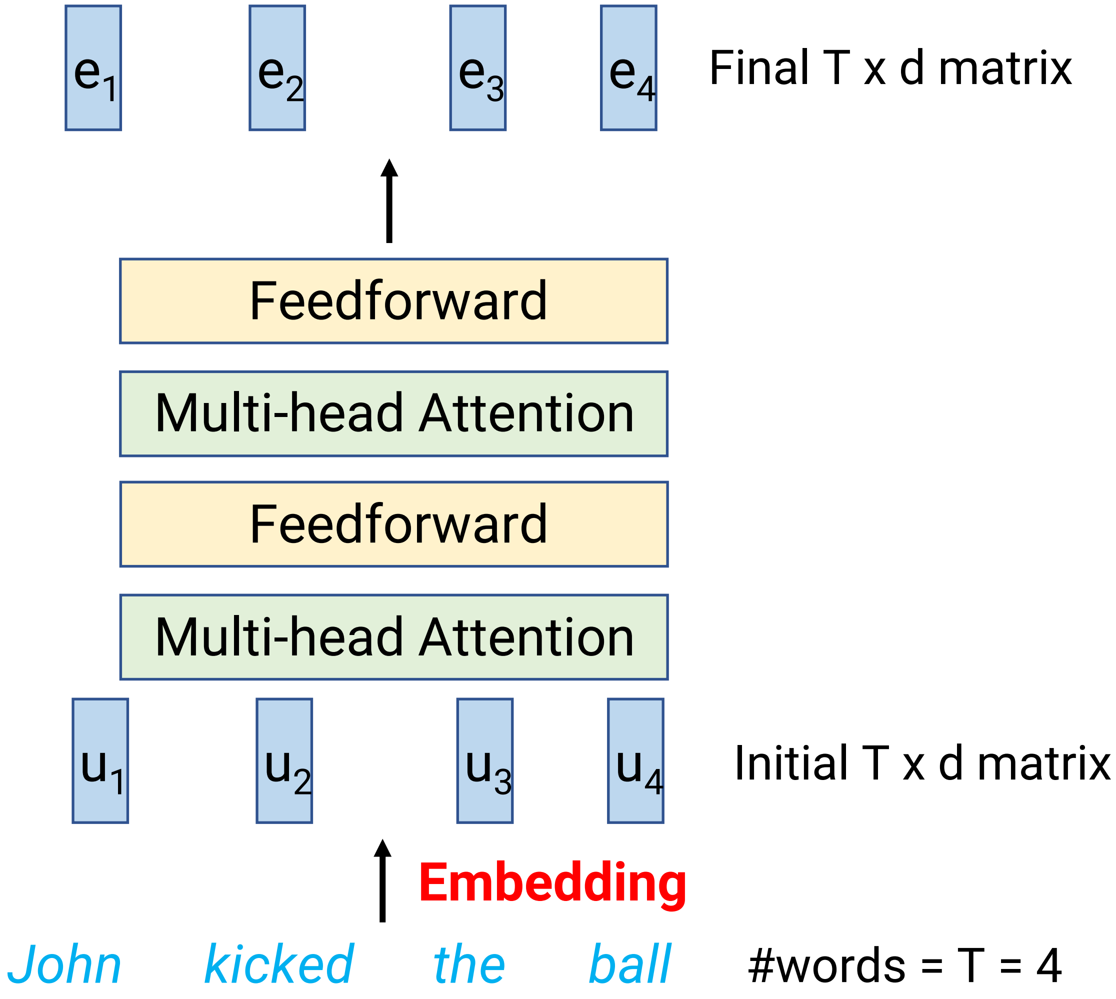
The Full Transformer



Full Transformer also includes:

- Positional embeddings
- Byte pair encoding
- Scaled dot product attention
- Residual connections between layers
- LayerNorm

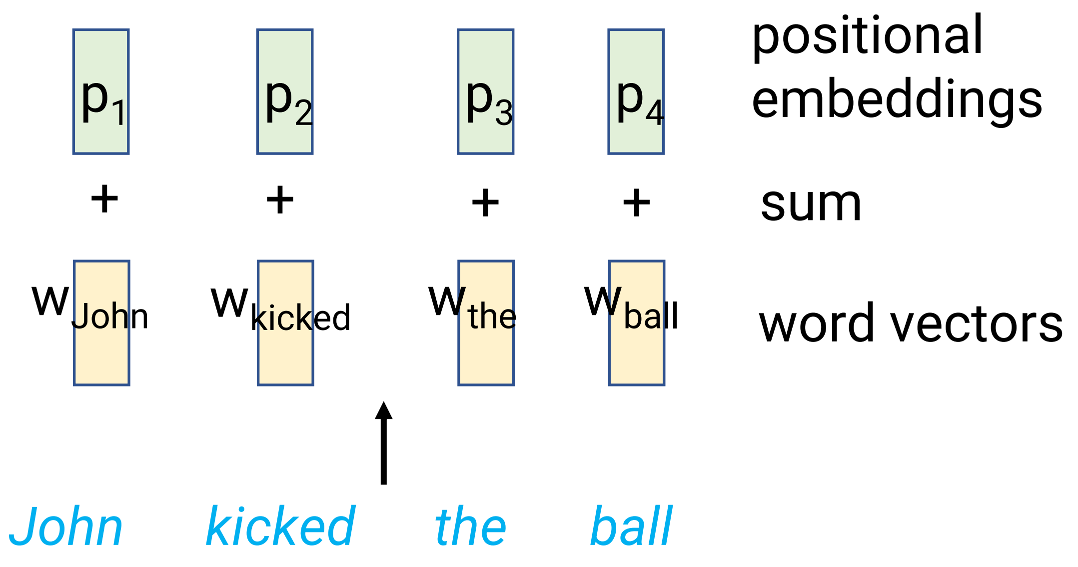
Transformer internals



- One transformer consists of
  - **Initial embeddings** for each word of size d
    - Let T =#words, so initially we have a T x d matrix
  - Alternating layers of
    - "Multi-headed" attention layer
    - Feedforward layer
    - Both take in T x d matrix and output a new T x d matrix
  - Plus some bells and whistles…

Embedding layer

- As before, learn a vector for each word in vocabulary
- Is this enough?
  - Both attention and feedforward layers are **order invariant**
  - Need the initial embeddings to also encode order of words!
    - Otherwise, every occurrence of the same word would be treated the same
- Solution: **Positional embeddings**
  - Learn a different vector for each index
  - Gets added to word vector at that index
  - Note: This means a Transformer model has some maximum sequence length it knows how to process

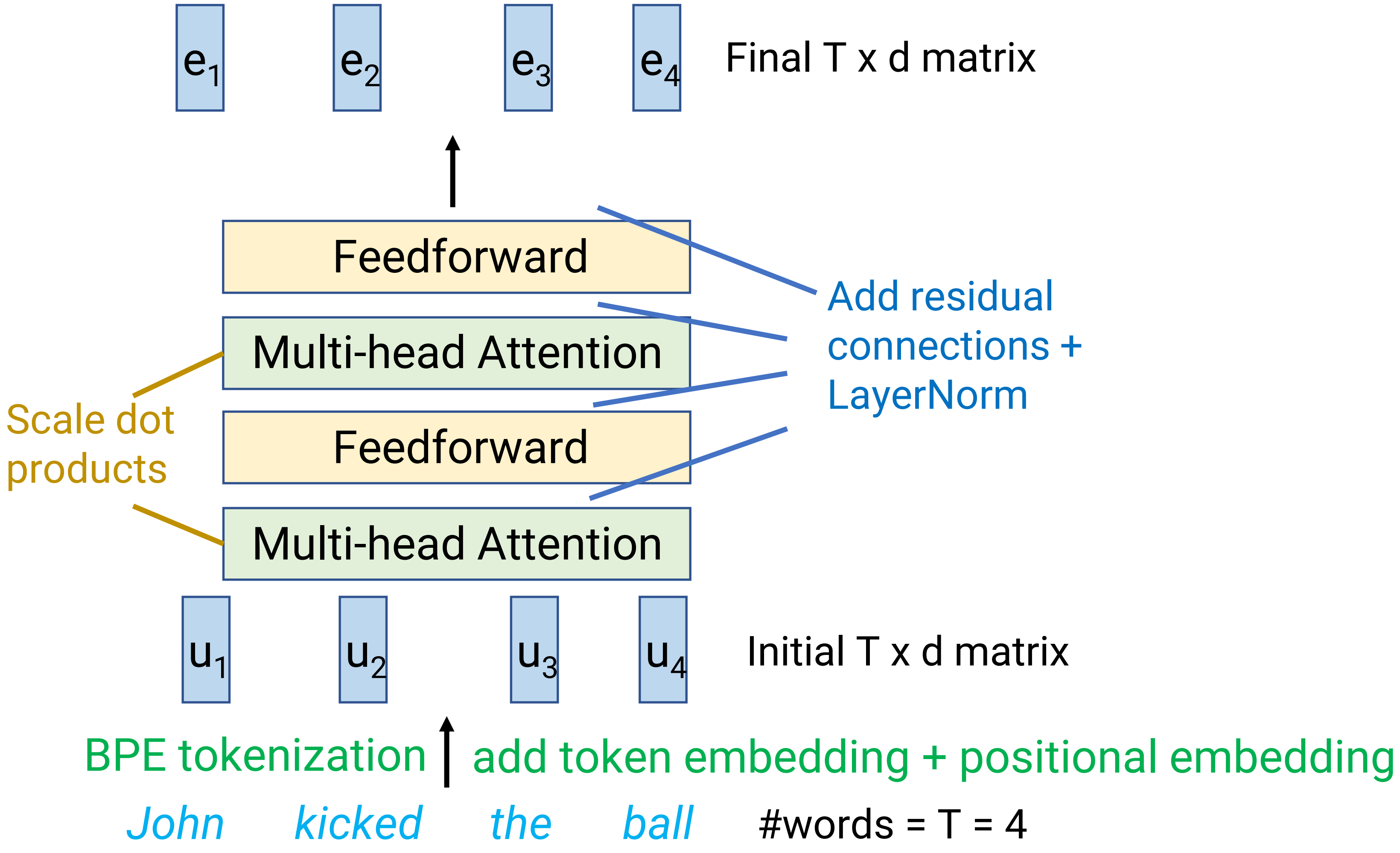


Byte Pair Encoding

- Normal word vectors have a problem: How to deal with super rare words?
  - Names? Typos?
  - Vocabulary can't contain literally every possible word…
- Solution: Tokenize string into "subword tokens"
  - Common words = 1 token
  - Rare words = multiple tokens

Aragorn told Frodo to mind Lothlorien	6 words
'Ar', 'ag', 'orn', ' told', ' Fro', 'do', ' to', ' mind', ' L', 'oth', 'lor', 'ien'	12 subword tokens

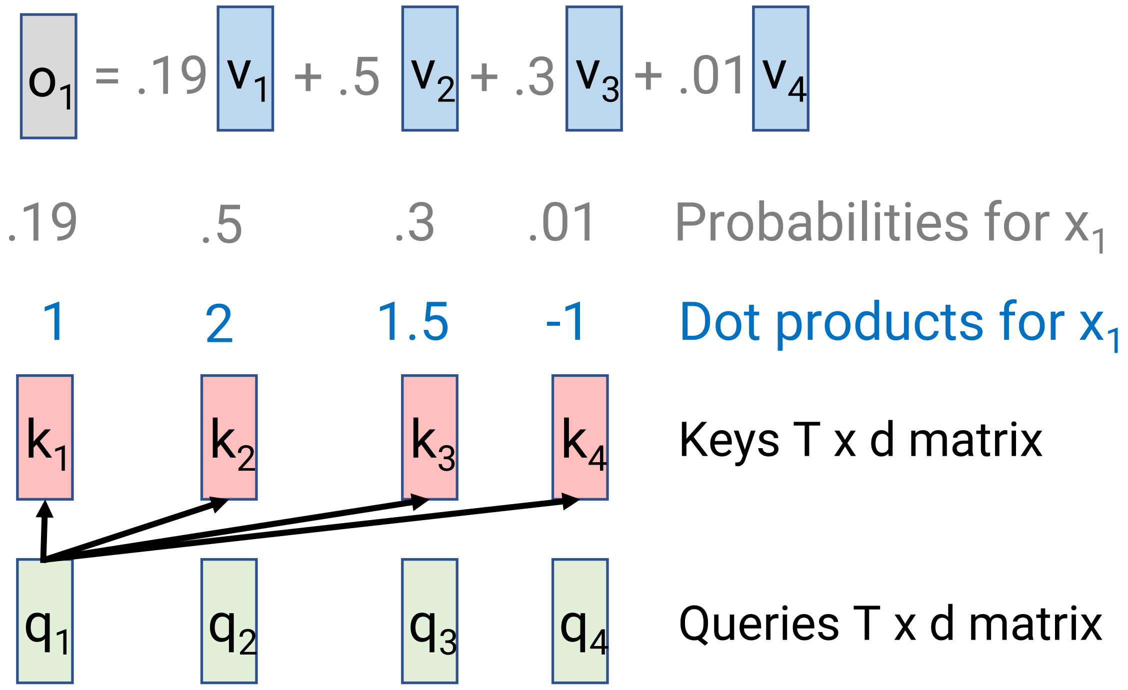
The Full Transformer



Full Transformer also includes:

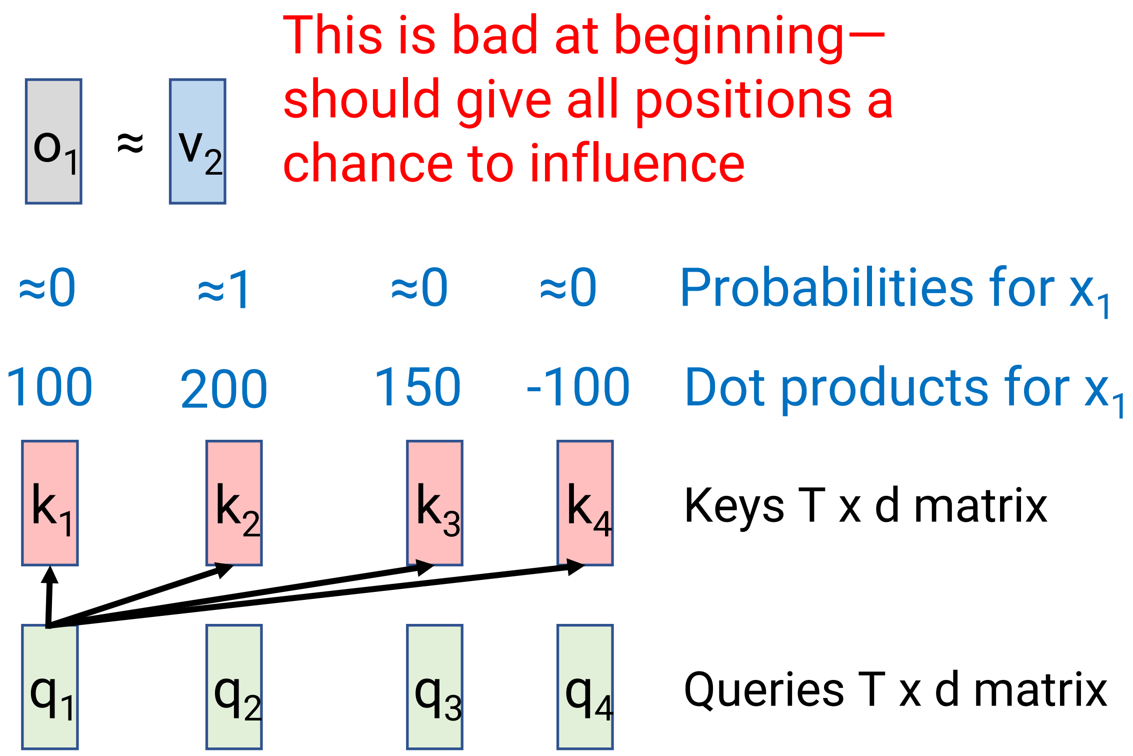
- Positional embeddings
- Byte pair encoding
- Scaled dot product attention
- Residual connections between layers
- LayerNorm

Scaled dot product attention



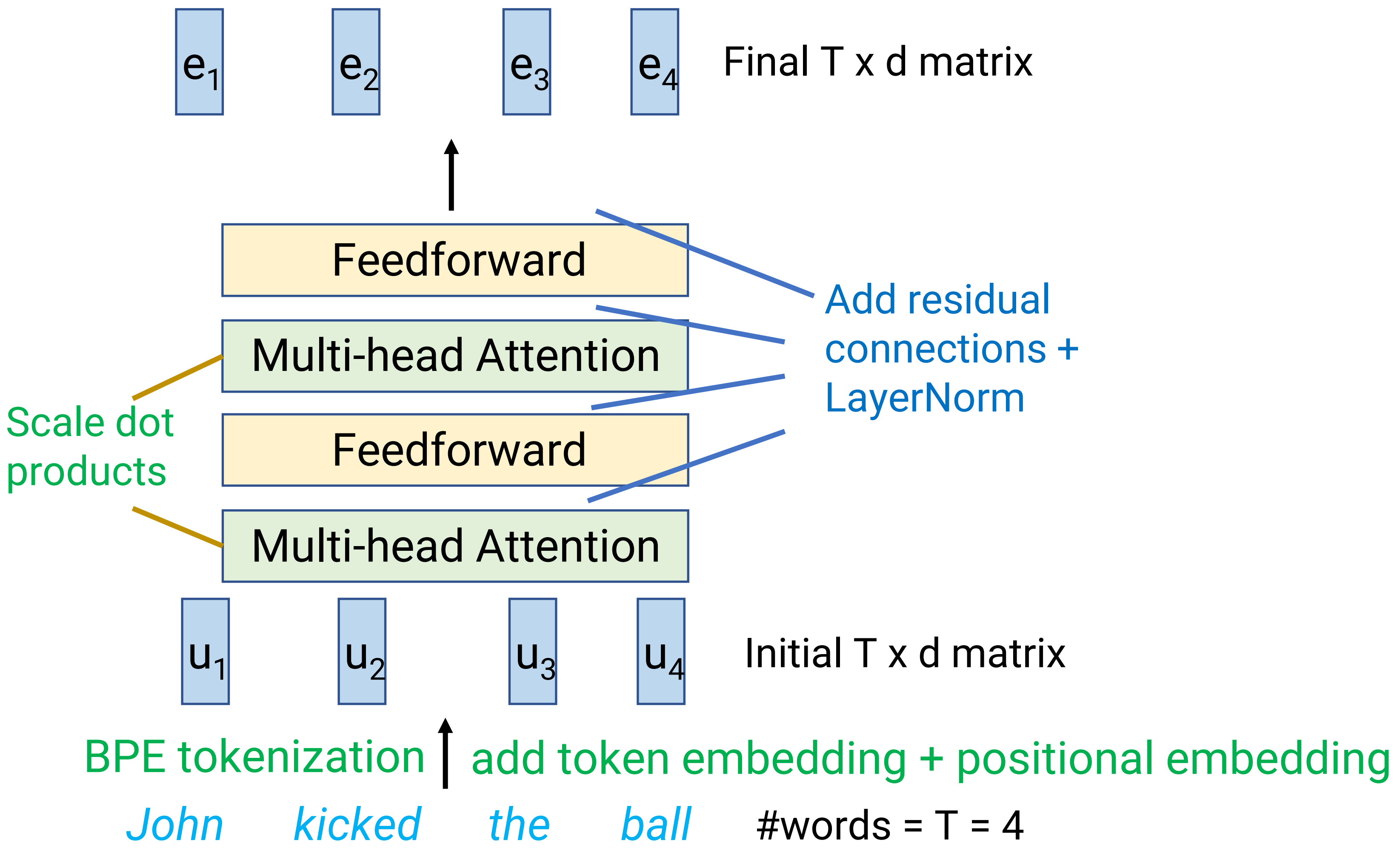
- Earlier I said, "Dot product q_t with $[k_1, \ldots, k_T]$"
- Actually, you take dot product and then divide by $\sqrt{d_{attn}}$
- Why?
  - If d large, dot product between random vectors will be large
  - This makes probabilities close to 0/1
  - Scaling dot products down encourages more even attention at beginning

Scaled dot product attention



- Earlier I said, "Dot product q_t with $[k_1, \ldots, k_T]$"
- Actually, you take dot product and then divide by $\sqrt{d_{attn}}$
- Why?
  - If d large, dot product between random vectors will be large
  - This makes probabilities close to 0/1
  - Scaling dot products down encourages more even attention at beginning

The Full Transformer

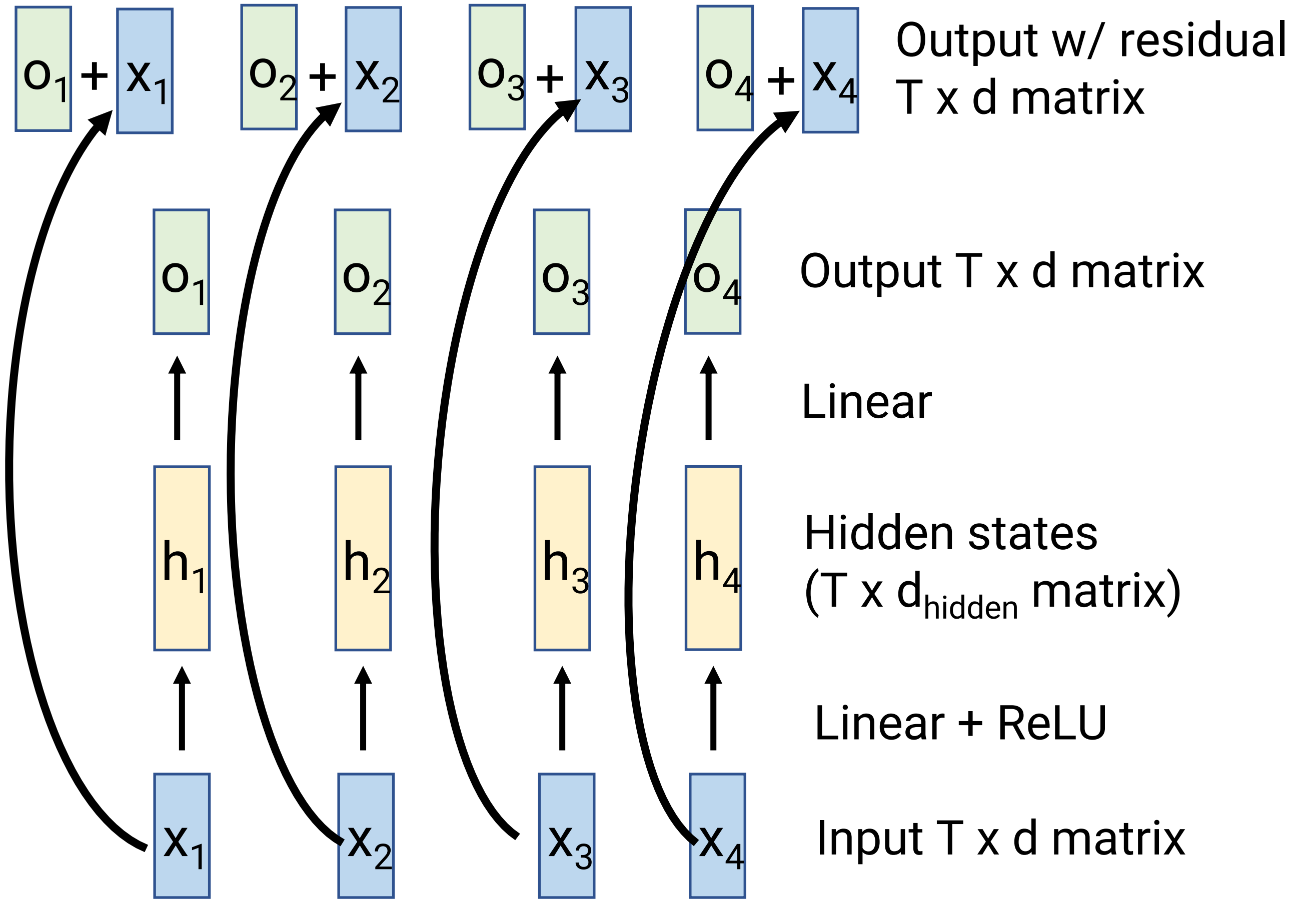


Full Transformer also includes:

- Positional embeddings
- Byte pair encoding
- Scaled dot product attention
- Residual connections between layers
- LayerNorm

Residual Connections

- Feedforward and multi-headed attention layers
  - Take in T x d matrix X
  - Output T x d matrix O
- We add a “residual” connection: we actually use X + O as output
  - Makes it easy to copy information from input to output
  - Think of O as how much we **change** the previous value
- Same idea also common in CNNs!
  - Reduces vanishing gradient issues



Layer Normalization ("LayerNorm")

- LayerNorm is just another type of layer/building block that "normalizes" a vector
- Input x: vector of size d
- Output y: vector of size d
- Formula:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$ Mean of components of x

$$\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$$ Variance of components of x

$$y = a \cdot \underbrace{\boxed{\frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}}}}_{\text{Normalized x}} + b$$

1. Normalize: Subtract by mean, divide by standard deviation
2. Rescale: Multiply by a, add b

- Parameters
  - a & b are scalar parameters, let model learn good scale/shift
    - Without these, all vectors forced to have mean=0, variance=1
  - ε is hyperparameter: Some small number to prevent division by 0

x = [100, 200, 100, 0]

μ = 100

$\sigma^2 = ¼ * (0^2 + 100^2 + 0^2 + 100^2) = 5000$

Normalized x =

[0, 100, 0, -100] / $\sqrt{5000}$

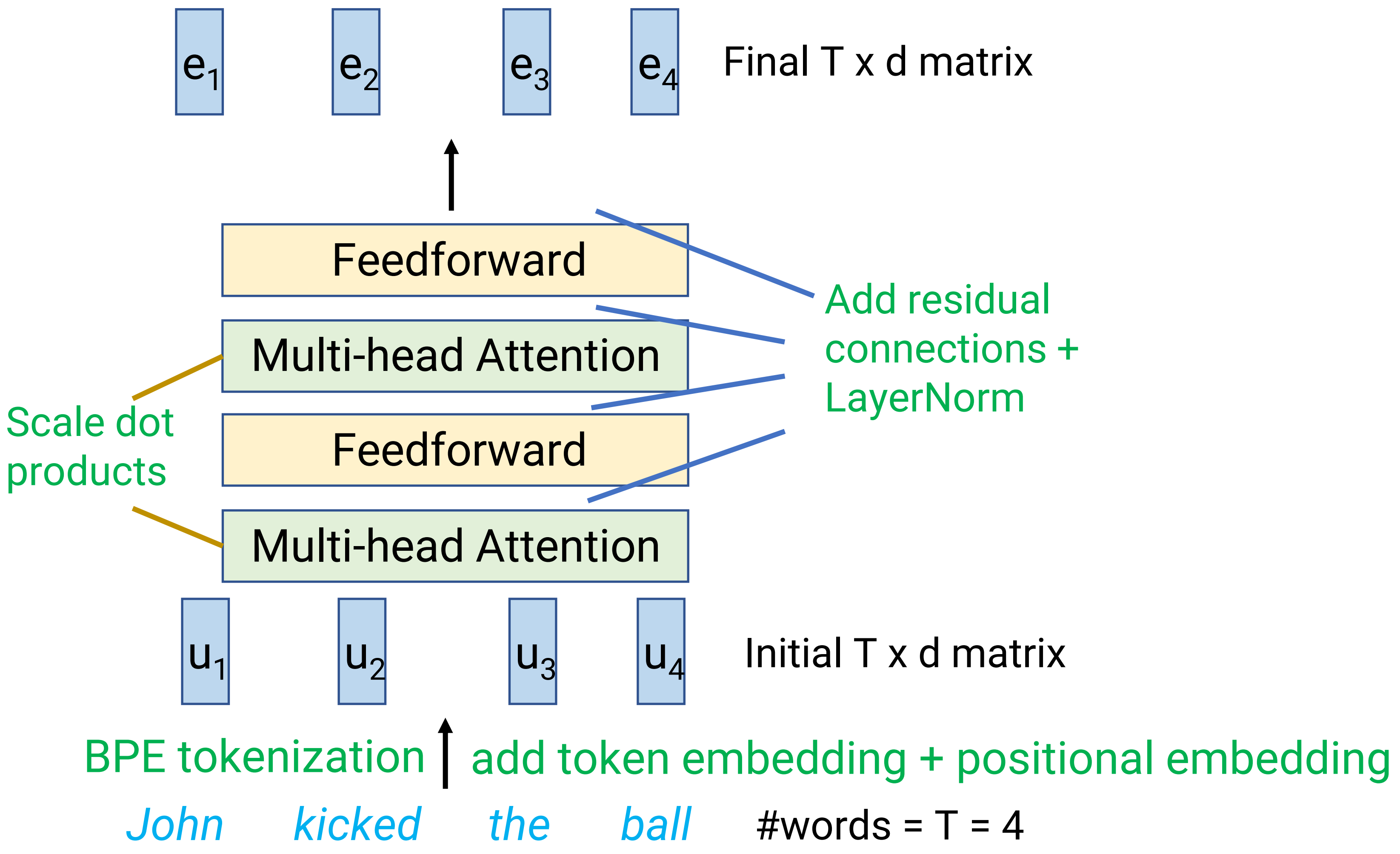
= [0, 1.4, 0, -1.4] (If $\varepsilon \approx 0$)

Output = [b, 1.4a+b, b, -1.4a+b]

LayerNorm in Transformers

- After every feedforward & multi-headed attention layer, we also add Layer Normalization
  - Input: vectors $x_1, \ldots, x_T$
  - Compute μ and σ^2 for each vector
  - Normalize each vector
  - Use the same a and b to rescale each vector
- Is applied after residual connection
  - Output of each layer is $\mathrm{LayerNorm}(x + \mathrm{Layer}(x))$
- Why? Stabilizes optimization by avoiding very large values

The Full Transformer



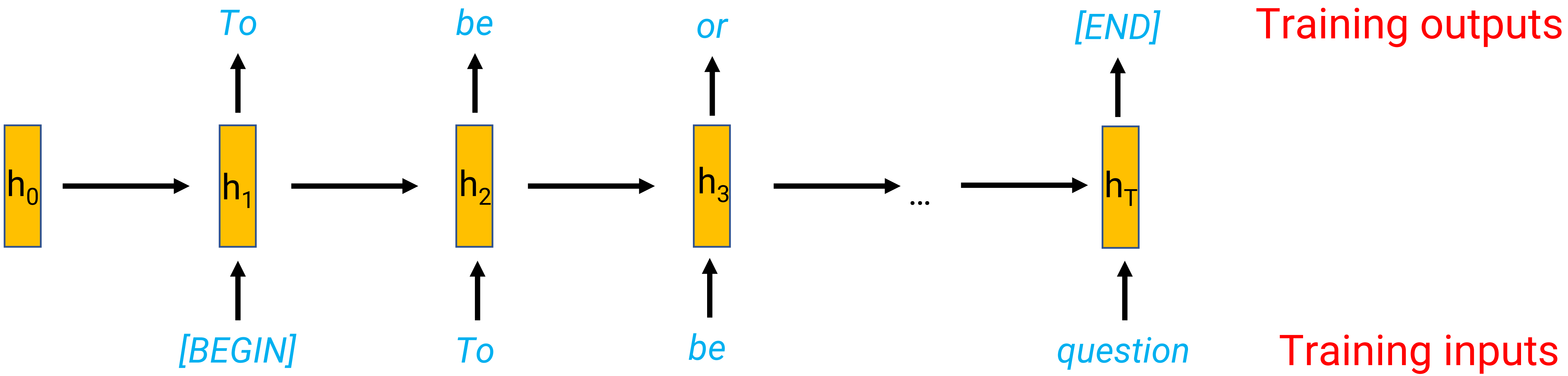
Full Transformer also includes:

- Positional embeddings
- Byte pair encoding
- Scaled dot product attention
- Residual connections between layers
- LayerNorm

Announcements

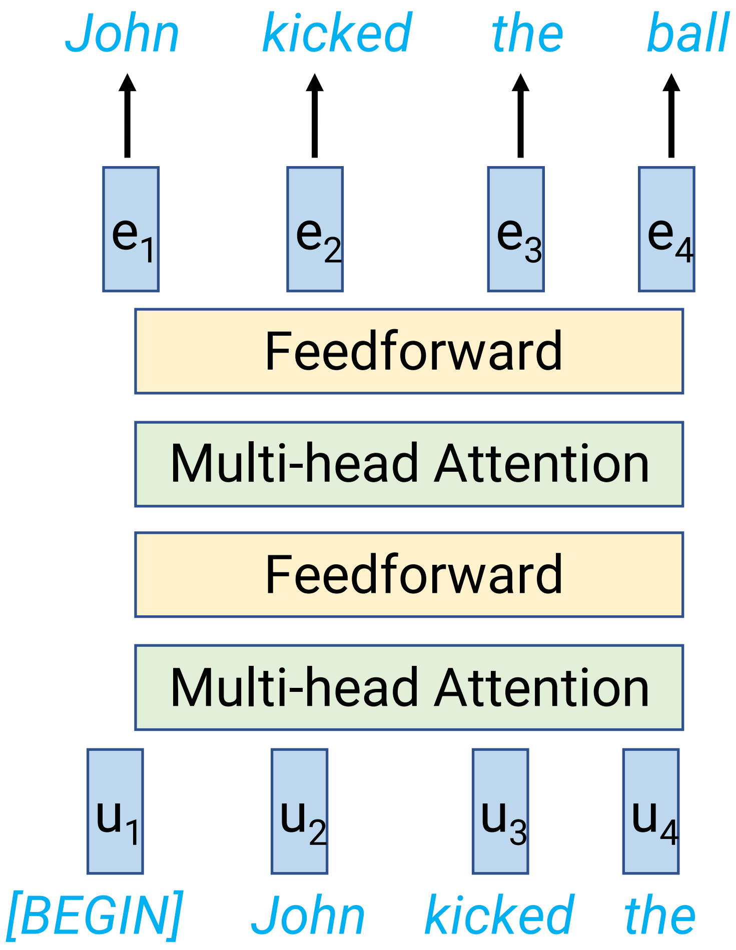
- Project midterm report due October 31
- HW3 to be released early next week
- Tomorrow’s section: RNNs in pytorch
  - How does an RNN decoder work?
  - What do the gradients look like?

Review: RNN Decoder Language Models

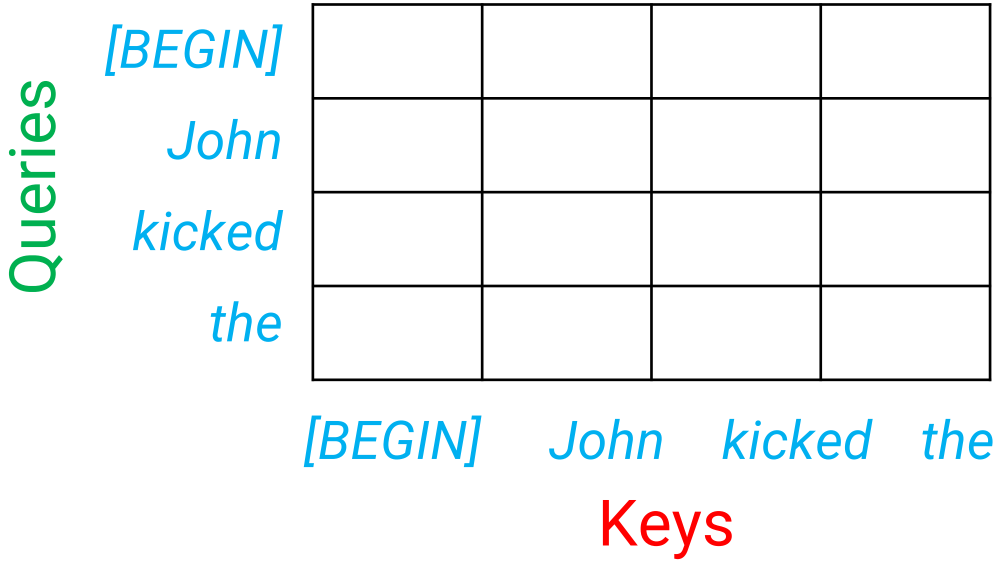


- At each step, predict the next word given current hidden state
- Test time: Model chooses a next word, that gets fed back in
- Training time: Model is fed the human-written words, tries to guess next word at every step
- RNN computations must happen in series at both training and test time
    - Each hidden state depends on the previous hidden state

Transformer autoregressive decoders



- How to do autoregressive language modeling?
- Test-time
  - At time t, attend to positions 1 through t
  - Happens in series



Transformer autoregressive decoders

- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

Queries

[BEGIN]	10	-2	6	3
John	0	7	2	-4
kicked	-3	4	5	-8
the	2	1	7	6
	[BEGIN]	*John*	*kicked*	*the*

Keys

Transformer autoregressive decoders

- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

Queries

	[BEGIN]	John	kicked	the
[BEGIN]	10	-2	6	3
John	0	7	2	-4
kicked	-3	4	5	-8
the	2	1	7	6

Keys

Transformer autoregressive decoders

- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

Queries

[BEGIN]	10	$-\infty$	$-\infty$	$-\infty$
John	0	7	$-\infty$	$-\infty$
kicked	-3	4	5	$-\infty$
the	2	1	7	6
	[BEGIN]	*John*	*kicked*	*the*

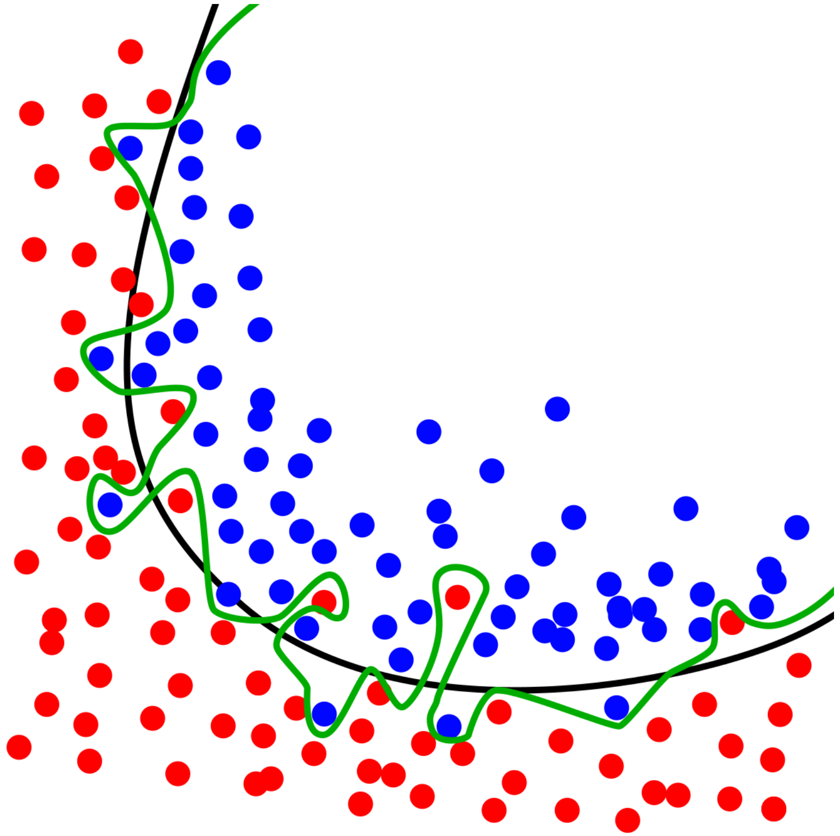
Keys

Today’s Plan

- Transformers in full detail
- Transformer decoders
- Pre-training

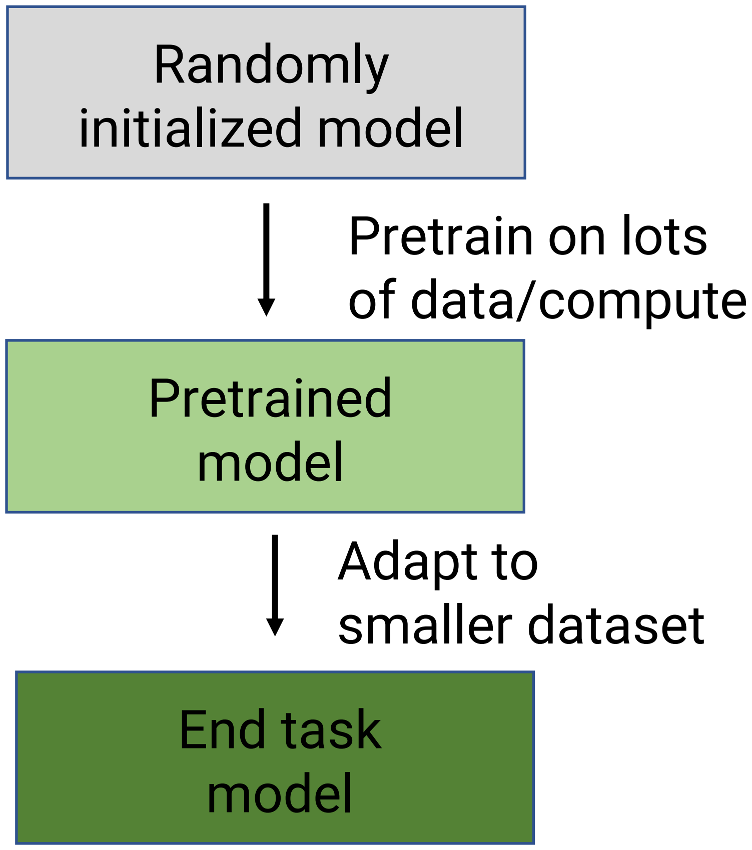
Neural Networks and Scale

- Neural networks are very expressive, but have tons of parameters
  - Very easy to overfit a small training dataset
- Traditionally, neural networks were viewed as flexible but very **"sample-inefficient"**: they need many training examples to be good
  - Computationally expensive
  - Training at scale often uses GPUs



Pretraining

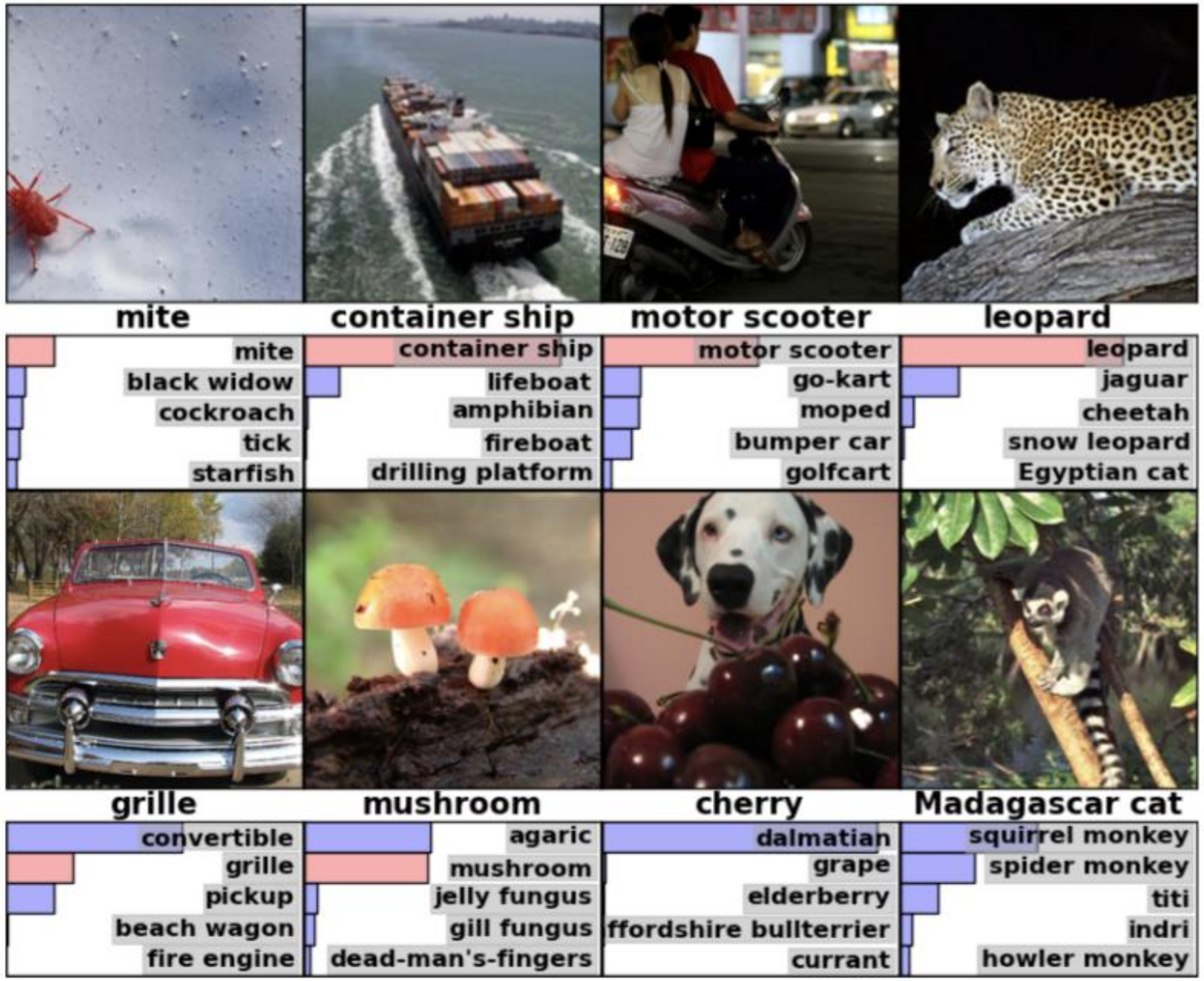
- Neural networks learn to extract features useful for some training task
  - The more data you have, the more successful this will be
- If your training task is very general, these features may also be useful for other tasks!
- Hence: **Pretraining**
  - First pre-train your model on one task with a lot of data
  - Then use model's features for a task with less data
  - Upends the conventional wisdom: You can use neural networks with small datasets now, if they were pretrained appropriately!



ImageNet Features

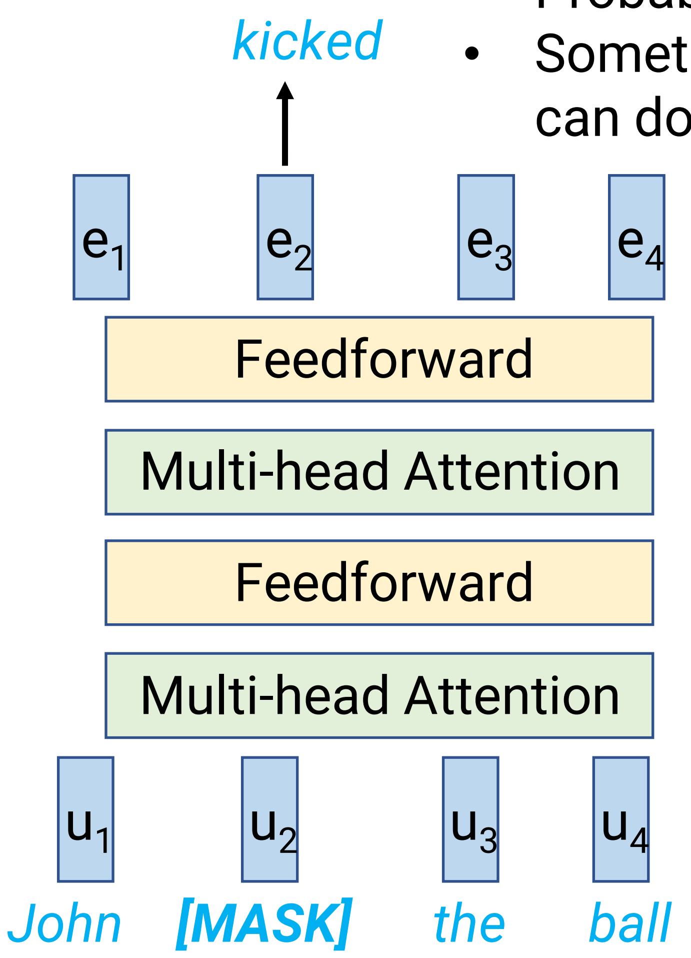
Features learned by AlexNet trained on ImageNet

ImageNet Features



- ImageNet dataset: **14M** images, 1000-way classification
- Most applications don't have this much data
- **But the same features are still useful**
- Using "frozen" pretrained features
  - Get a (small) dataset for your task
  - Generate features from ImageNet-trained model on this data
  - Train linear classifier (or shallow neural network) using ImageNet features

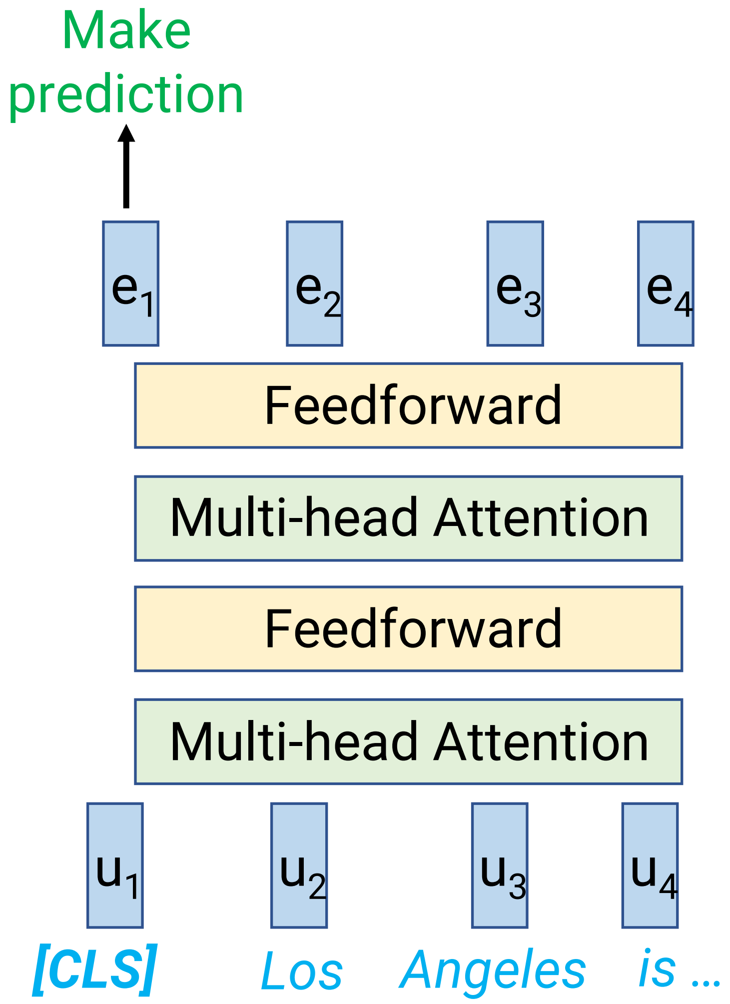
Masked Language Modeling (MLM)



- Probably a verb
- Something a person can do to a ball

- MLM: Randomly mask some words, train model to predict what's missing
  - Doing this well requires understanding grammar, world knowledge, etc.
  - Get training data just by grabbing any text and randomly delete words
  - Thus: Crawl internet for text data
- Transformers are good fit due to scalability
  - Large matrix multiplications are highly optimized on GPUs/TPUs
  - Don't need lots of operations happening in series (like RNNs)
- Most famous example: BERT

Fine-tuning



- Initialize parameters with BERT
  - BERT was trained to expect every input to start with a special token called [CLS]
- Add parameters that take in the output at the [CLS] position and make prediction
- Keep training all parameters (“fine-tune”) on the new task
- Point: BERT provides very good initialization for SGD

What about ChatGPT???

- ChatGPT appears to be a fine-tuned language model
    - Pretrained on autoregressive language modeling
    - Then fine-tuned with a method called RLHF (reinforcement learning from human feedback)
    - We’ll return to this when we talk about reinforcement learning!

Conclusion: Transformers

- “Attention is all you need”
  - Get rid of recurrent connections—all “communication” between words in sequence is handled by attention
  - Have multiple attention “heads” to learn different types of relationships between words
    - Each head has its own parameters, which enable them to learn different things
  - Plus lots of additional components to make it fit together
  - Most famous modern language models (e.g., ChatGPT) are Transformers!
- Pretraining
  - First train on large labeled or unlabeled datasets
  - Features learned are useful for other tasks with less data