CS 839: Foundation Models
Transformers and Attention

Fred Sala

University of Wisconsin-Madison

Sept. 12, 2024

Announcements

- **Announcement:** None

- Class roadmap:

Thursday Sept. 12	Architectures I: Transformers & Attention
Tuesday Sept. 17	Architectures II: Subquadratic Architectures
Thursday Sept. 19	Language Models I
Tuesday Sept. 24	Language Models II
Thursday Sept. 26	Prompting I

Mostly Language Models

Outline

- **Mini-Intro**
  - Terminology, generative vs. discriminative, pretraining, representations vs. embeddings
- **Attention**
  - Notions of attention, self-attention, basic attention layer, QKV setup and intuition, positional encodings
- **Transformers**
  - Architecture, encoder and decoder setups

Terminology: Generative vs. Discriminative

Need a few terms to be re-used during class

- **Discriminative** model
  - Directly predict label $h(x) = y$ or compute $h(x) = p(y|x)$
  - Canonical example: **logistic regression**

$$P_\theta(y = 1|x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Terminology: Generative vs. Discriminative

Need a few terms to be re-used during class

- **Generative** model
  - Model $h(x,y) = p(x,y)$ or $h(x) = p(x)$. Can be unsupervised
  - Canonical example: **naïve Bayes**

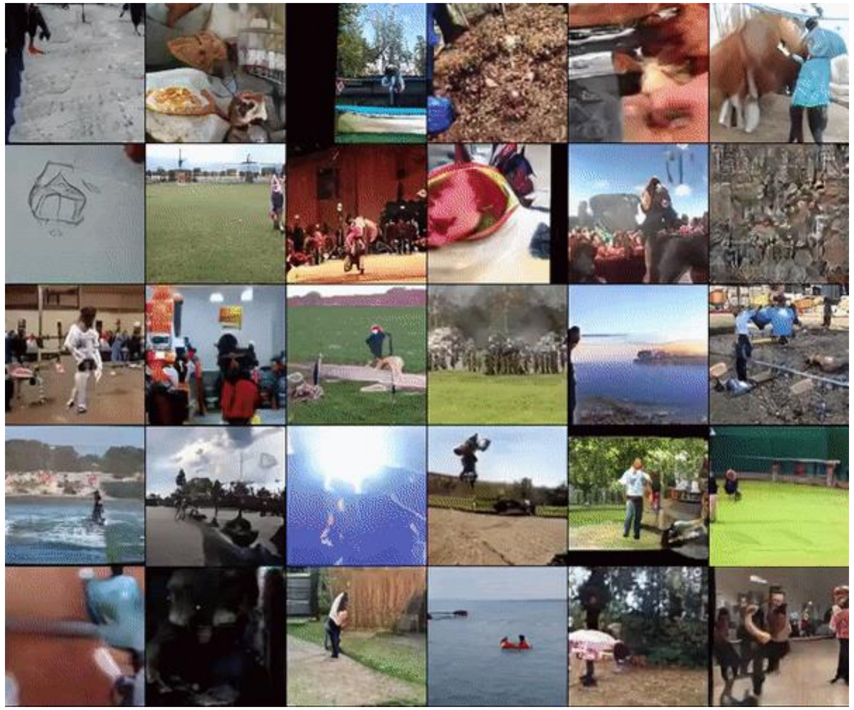
$$
\begin{aligned}
P(X_1, \ldots, X_K, Y) &= P(X_1, \ldots, X_K | Y) P(Y) \\
&= \left(\prod_{k=1}^{K} P(X_k | Y) \right) P(Y)
\end{aligned}
$$

Generative Models

Learning a distribution from samples

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

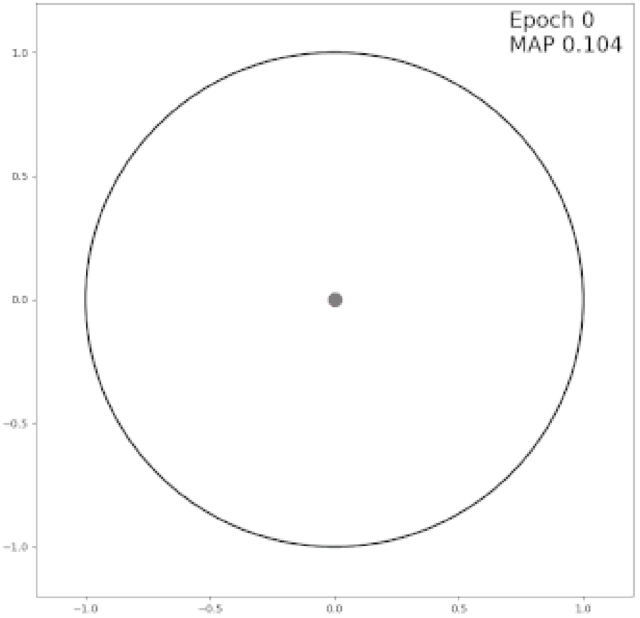
- Traditionally, want to
  - **Compute density**: compute p(x) for some x
  - **Inference**: compute p(a|b) for some a,b
  - **Sampling**: obtain a sample from p
- Modern methods: may only be able to sample/conditionally sample



Embeddings & Representations

Related terminology.

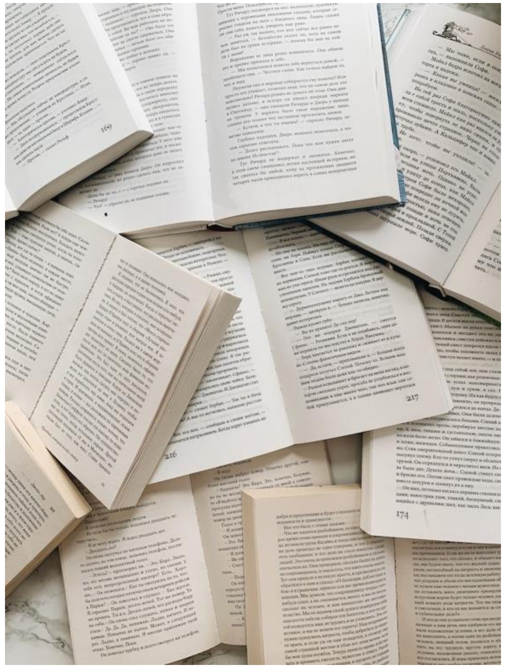
- Embeddings
  - Traditionally, goal is to take discrete objects (words, graphs, etc.) and produce vectors usable in DNNs
  - **Text**: Word2Vec **Graphs**: Hyperbolic embeddings



Embeddings & Representations

Related terminology.

- Embeddings
  - Often trained based on some custom loss (no “task”)
  - Word2Vec: word co-occurrences ⟷ embedding distances/ips

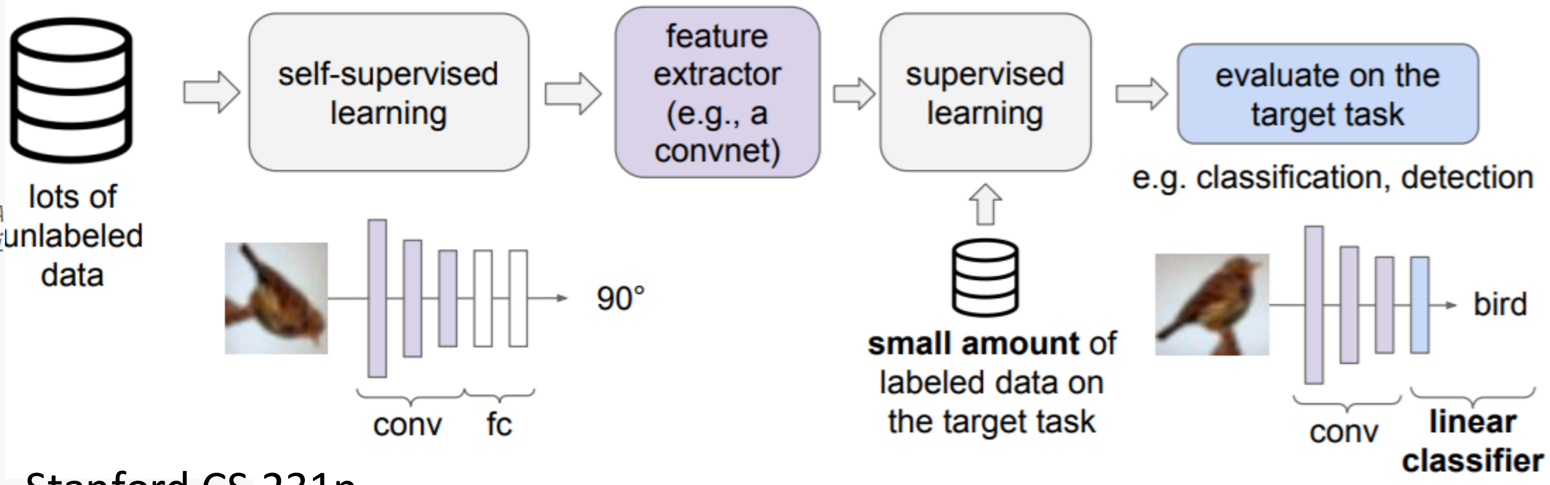


Embeddings & Representations

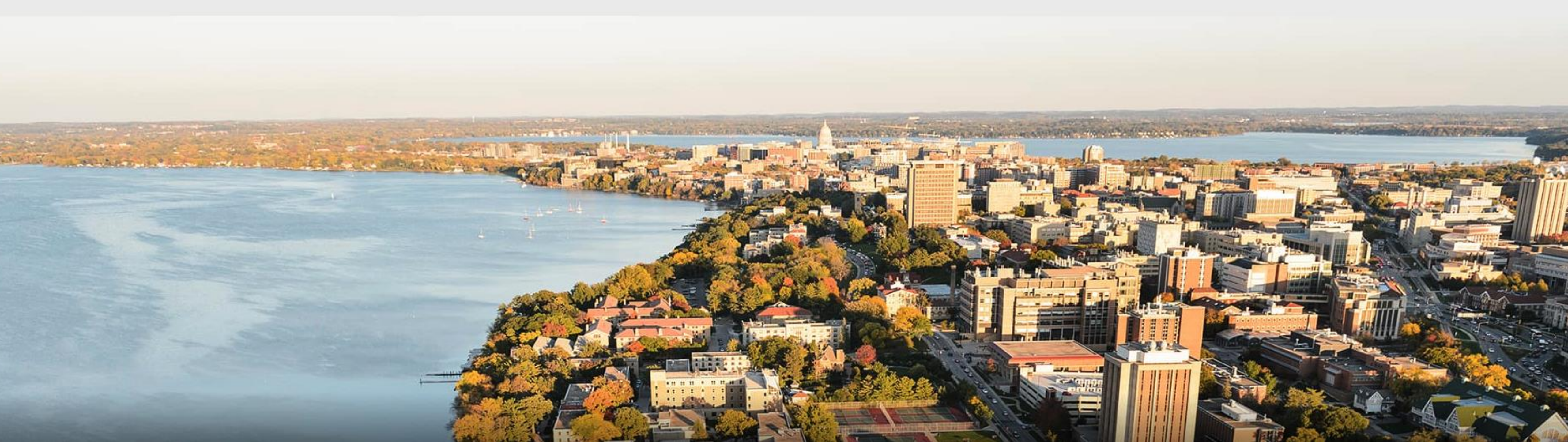
Related terminology.

- Representations
  - Often trained based on related task OR pretext task
  - Contain “deeper” information about each sample
  - Come from “pretrained” models

```
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMA
# Note that these weights may change across versi
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```



Stanford CS 231n



Break & Questions

History of Attention

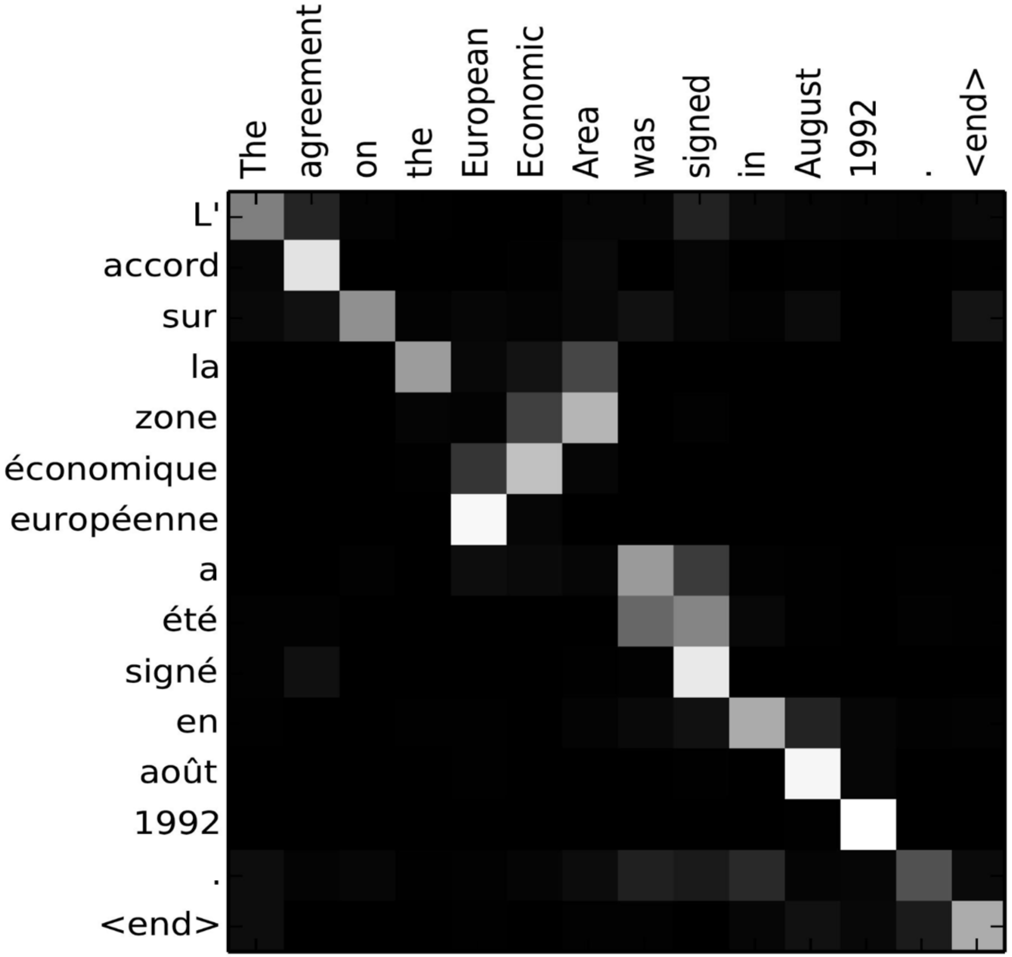
Basic motivation: in NLP *fixed* context vector **not** enough

- Why?
  - Words depend on each other
  - Dependencies are complex

- Need: mechanism to help model **focus** on the right "part"

Lots of approaches from 2014 on

- Bahdanau et al, 2014

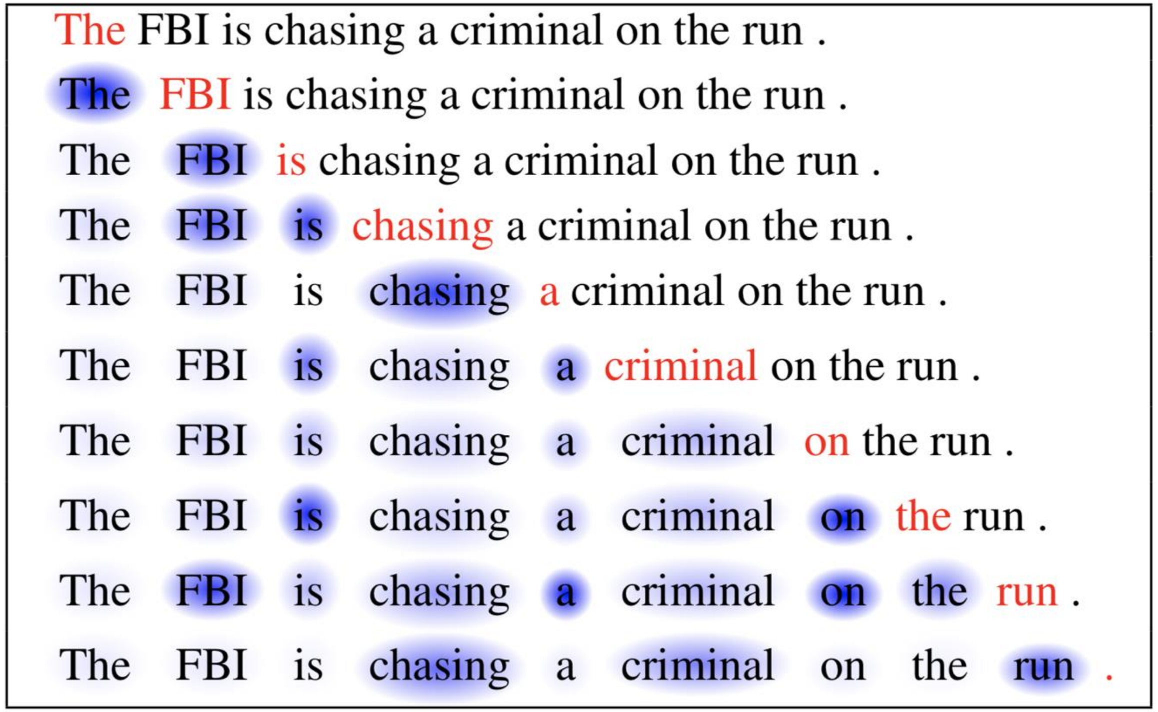


Bahdanau et al, 2014

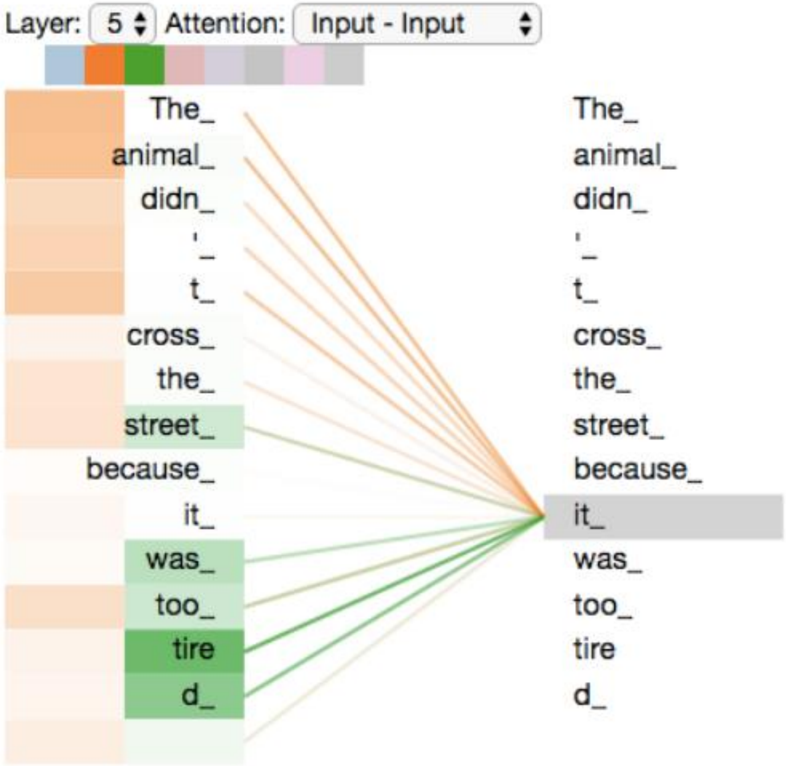
Self-Attention: Motivation

Popularized from 2017 on…

- From bottom-up. Let’s design a basic layer.
  - Intuition: dependencies **within** same sentence



Cheng et al, 2016



Jay Alammar

Self-Attention: Goals and Inputs

From bottom-up. Let's design a basic layer.

- Two criteria
  - *Transform* incoming word vectors,
  - Enable *interactions* between words

- Input: vectors for words

Note: All visualizations are due to Jay Alammar

Excellent resource: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Retrieval Intuition

- How should we design the interactions?
  - Analogy: **search**

“Which restaurants near me are open at 9:00 pm?”

Query

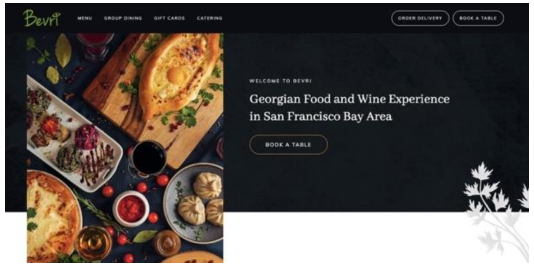
Key

Value

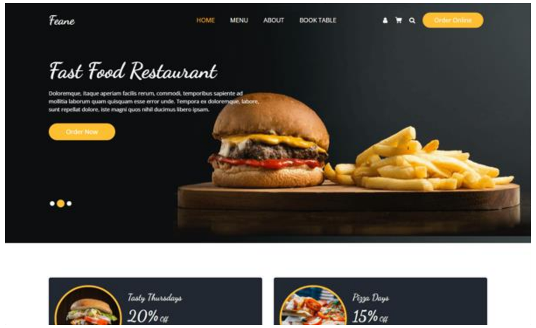
Objects:

Query
Key
Value

Score 0.3



Score 0.7



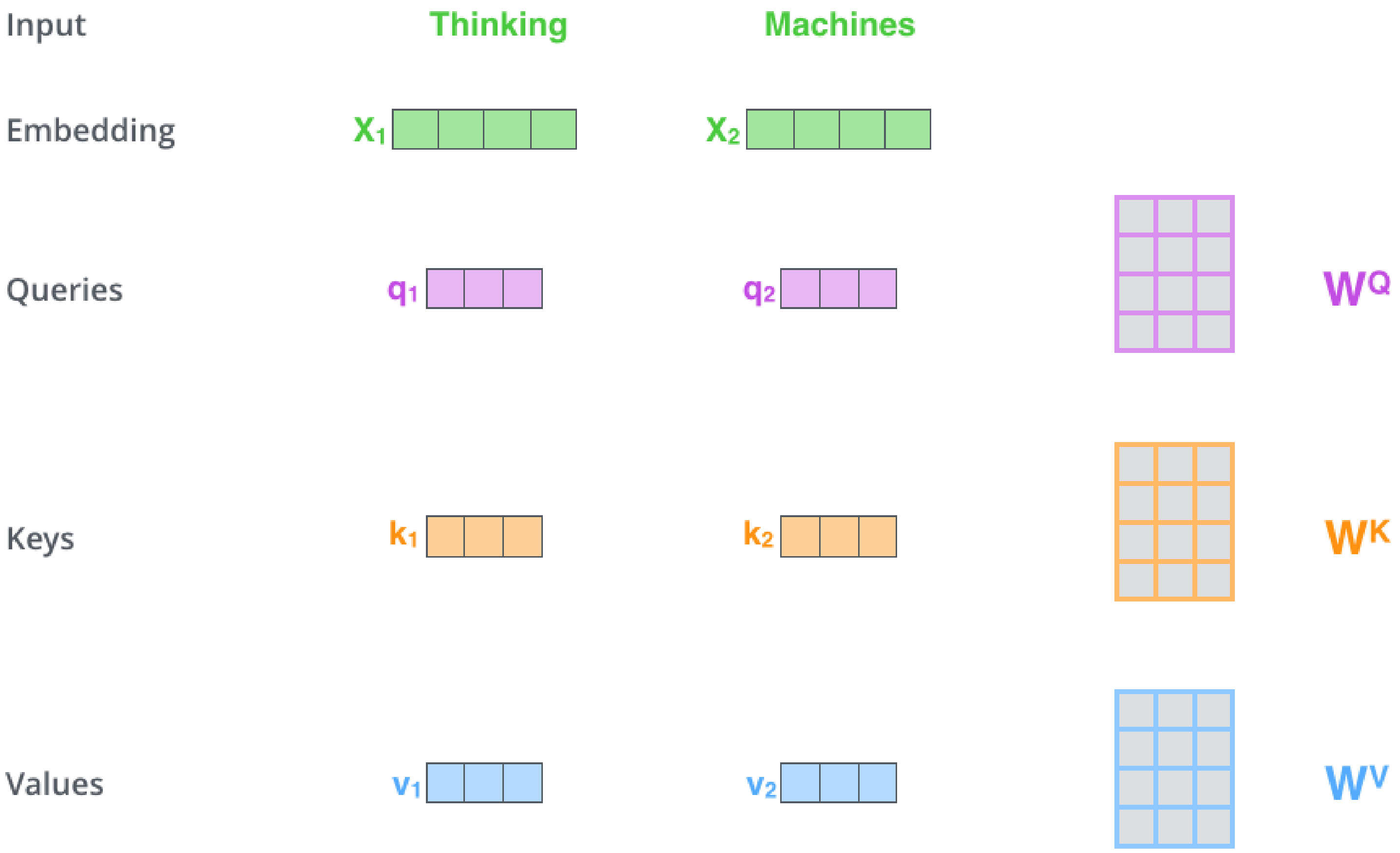
Self-Attention: Query, Key, Value Vectors

- *Transform* incoming word vectors,
- Enable *interactions* between words

- Get our **query, key, value** vectors via weight matrices: linear transformations!

Objects:

Query
Key
Value



Self-Attention: Score Functions

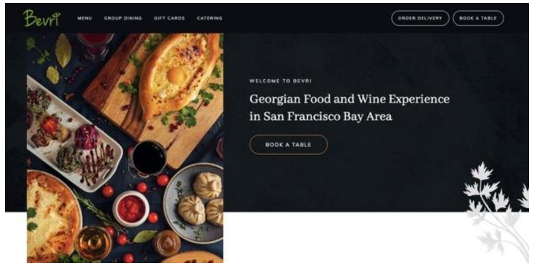
Have **query, key, value** vectors

- Next, get our **score**

q_1 k_2

Query Key

Score 0.3



- Lots of things we could do --- **simpler** is usually better!
- Dot product $q_1 \bullet k_2 = 96$

- Then we'll do **softmax**

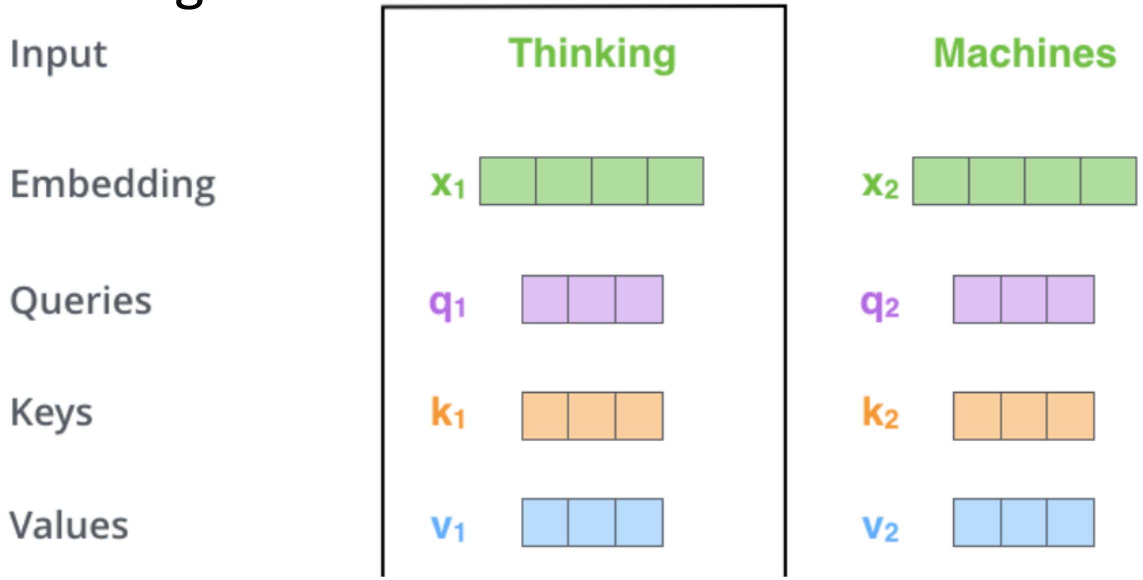
Self-Attention: Scoring and Scaling

- *Transform* incoming word vectors,
- Enable *interactions* between words

- Get our **query, key, value** vectors via weight matrices: linear transformations!
- Compute scores

Objects:

Query
Key
Value

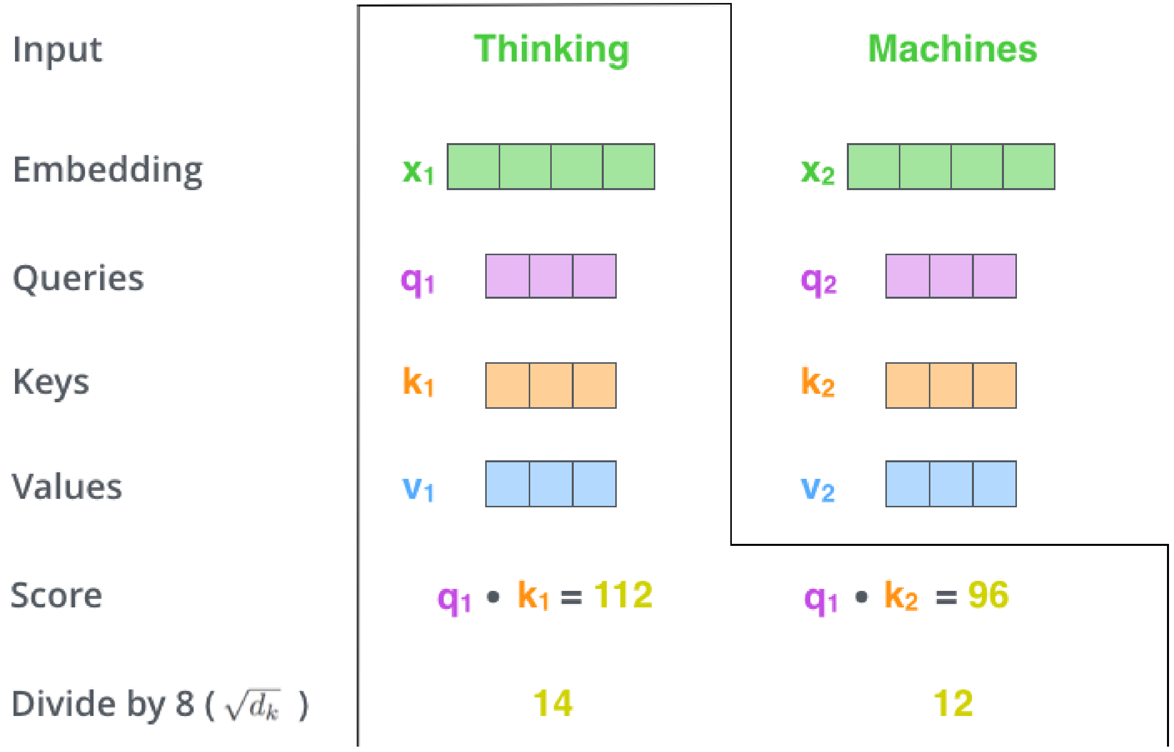


Self-Attention: Putting it Together

- Have **query, key, value** vectors via weight matrices: linear transformations!
- Have softmax score outputs (**focus**)
- Add up the values!

Objects:

Query
Key
Value



Self-Attention: Matrix Formulas

- Have **query, key, value** vectors via weight matrices: linear transformations!
- Have softmax score outputs (**focus**)
- Add up the values!

Objects:

Query
Key
Value

$$Q = XW_Q, K = XW_K, V = XW_V$$

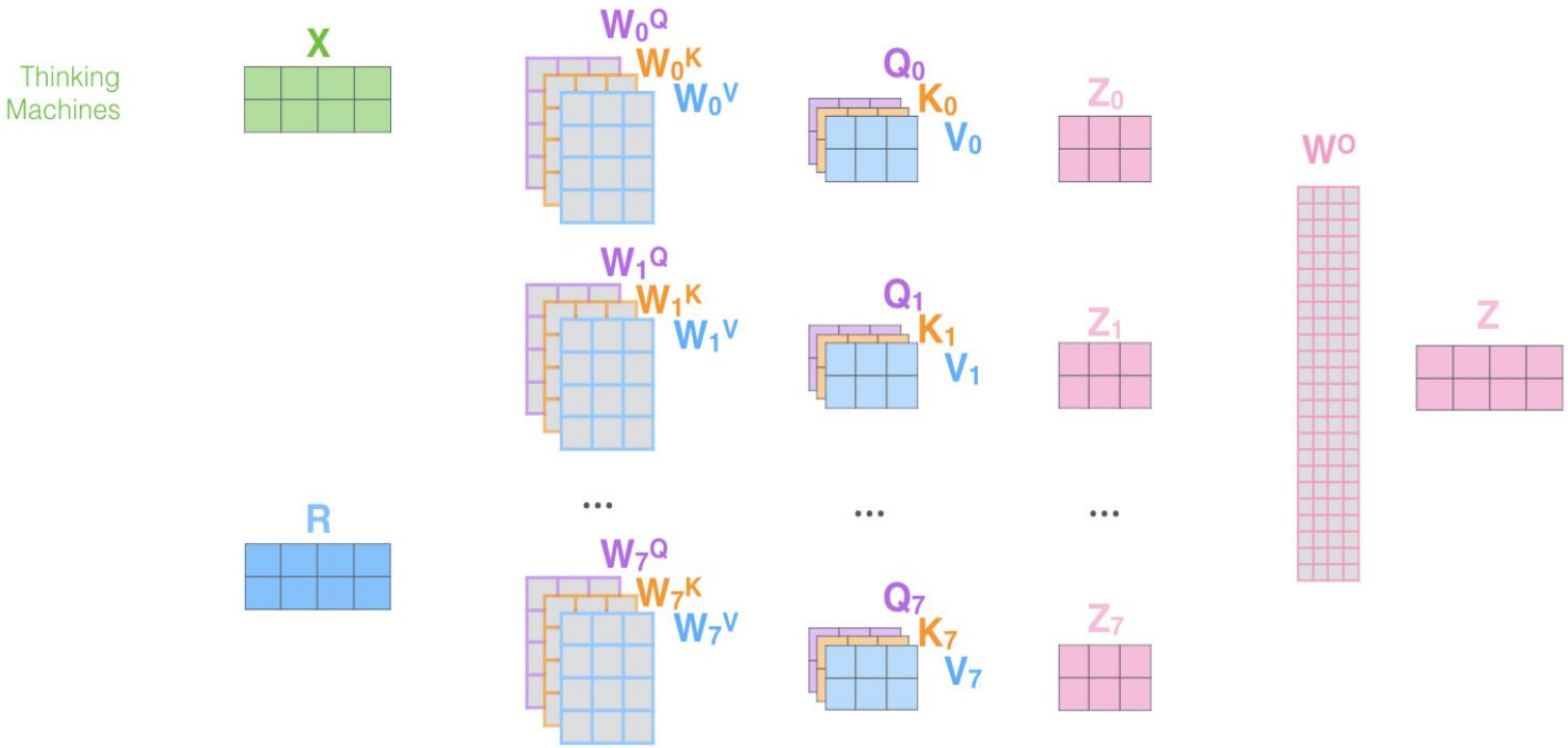
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(X\frac{W_Q {W_K}^T}{\sqrt{d_k}}X^T\right)V$$

Self-Attention: Multi-head

This is great but will we capture everything in one?

- Do we use just 1 kernel in CNNs? **No!**
- Do it many times in parallel: **multi-headed attention.** Concatenate outputs



Self-Attention: Position Encodings

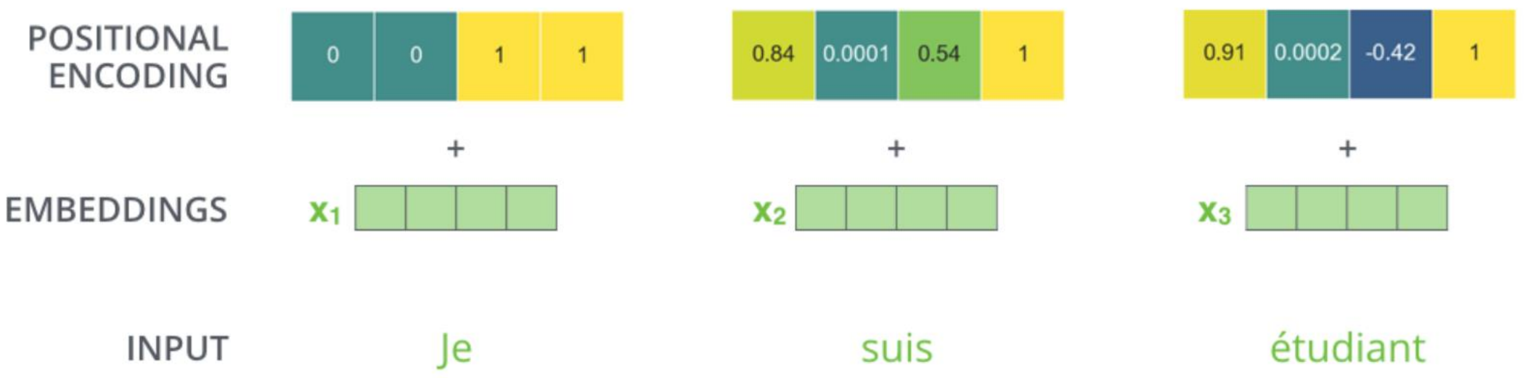
Almost have a full layer designed.

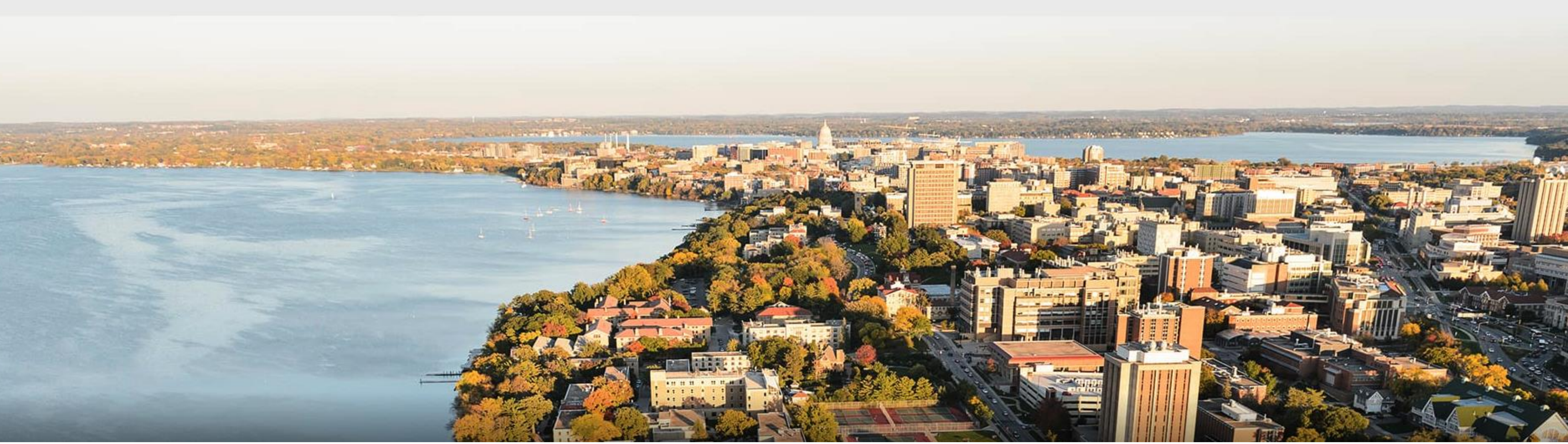
- One annoying issue: so far, order of words **(position) doesn't matter**!
- Solution: add positional encodings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Component index



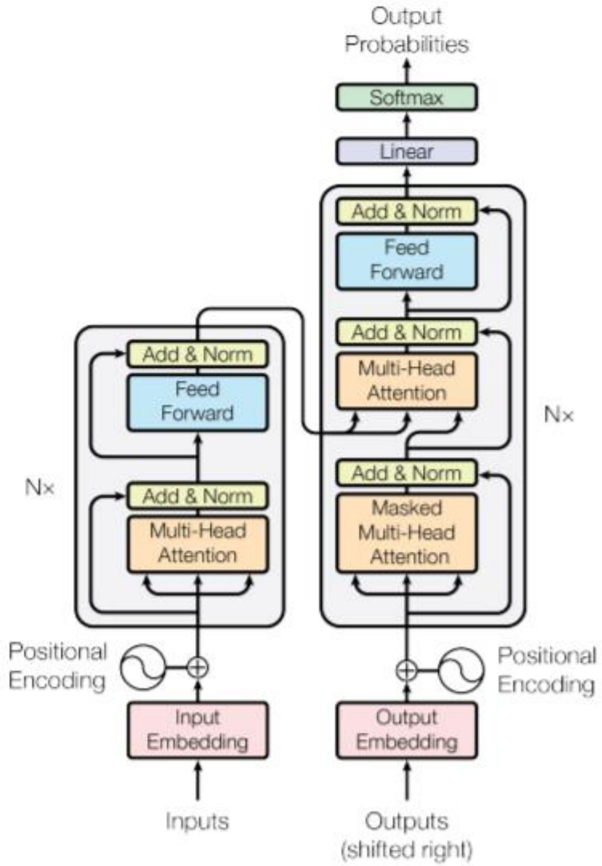


Break & Questions

Transformers: Model Architecture

- Initial goal for an architecture: **encoder-decoder**
  - Get **rid of recurrence**
  - Replace with **self-attention**

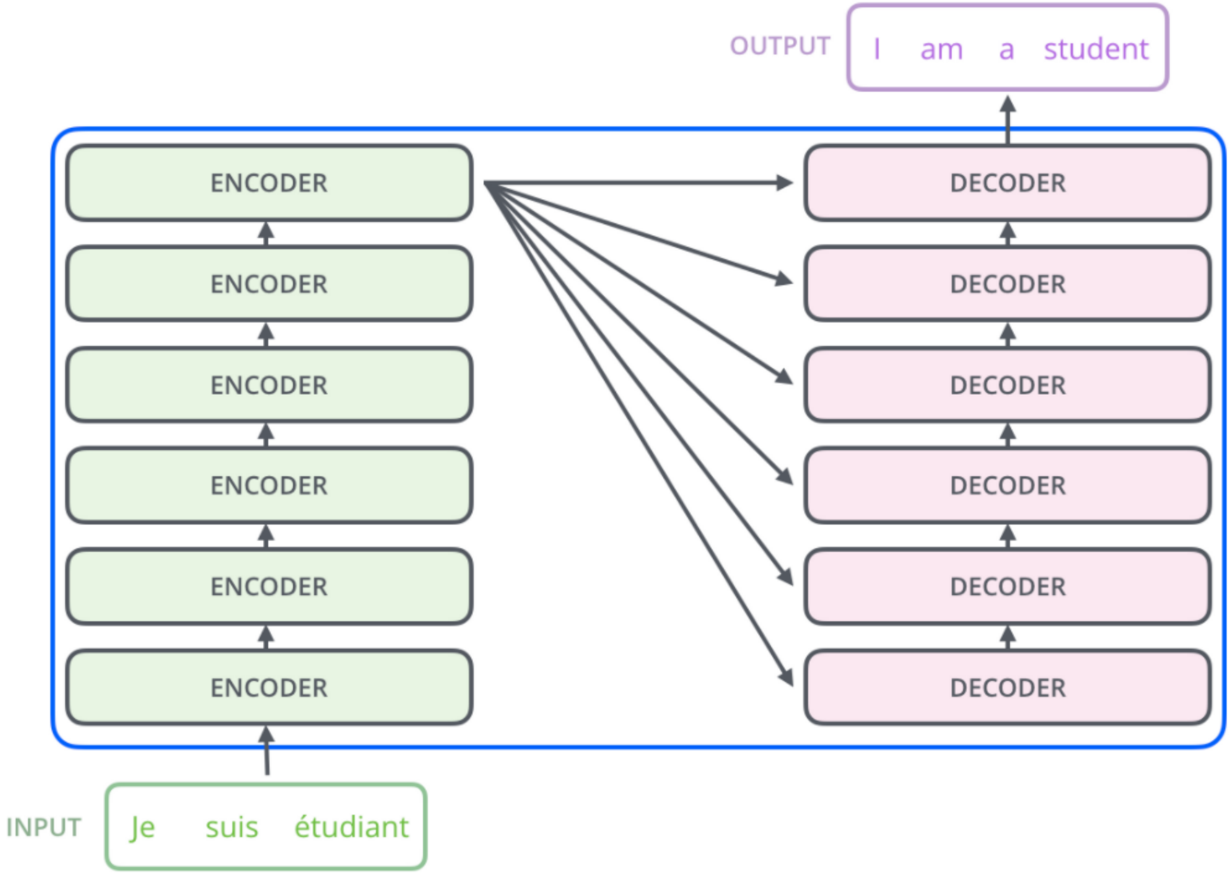
- Architecture
  - The famous picture you've seen
  - Centered on self-attention blocks



Vaswani et al. '17

Transformers: Architecture

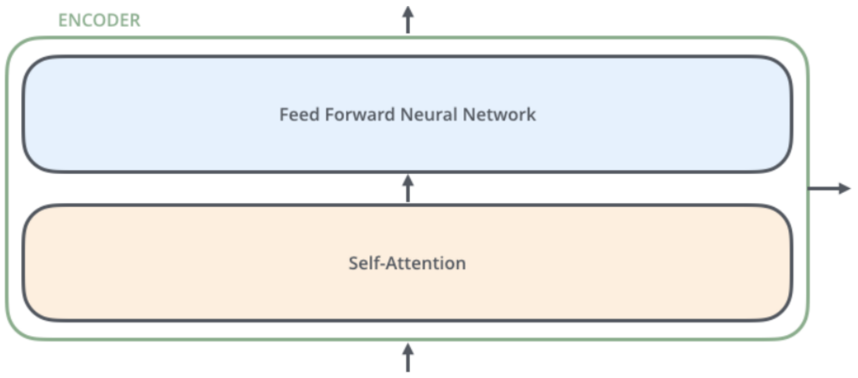
- **Sequence-sequence** model with **stacked** encoders/decoders:
  - For example, for French-English translation:

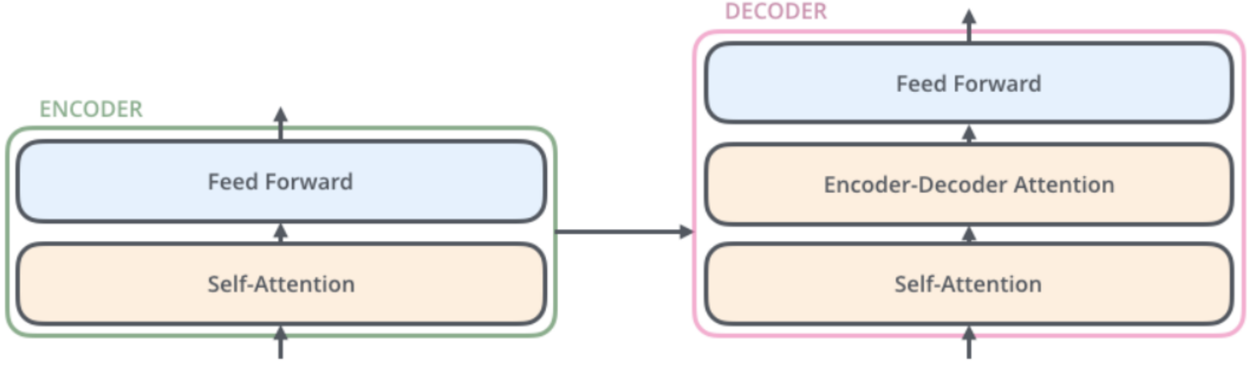


Transformers: Architecture

- Sequence-sequence model with **stacked** encoders/decoders:
  - What's inside each encoder/decoder unit?

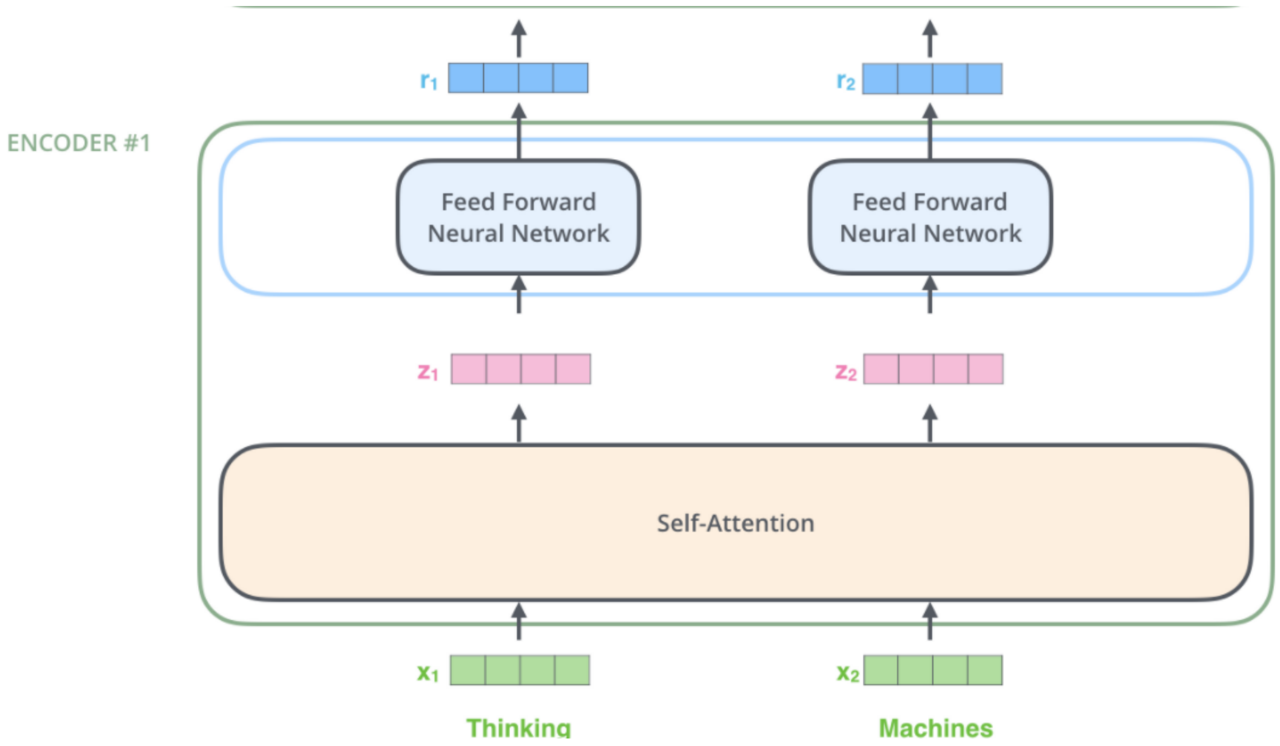
  - Focus encoder first: **pretty simple**! 2 components:
    - Self-attention block
    - Fully-connected layers (i.e., an MLP)





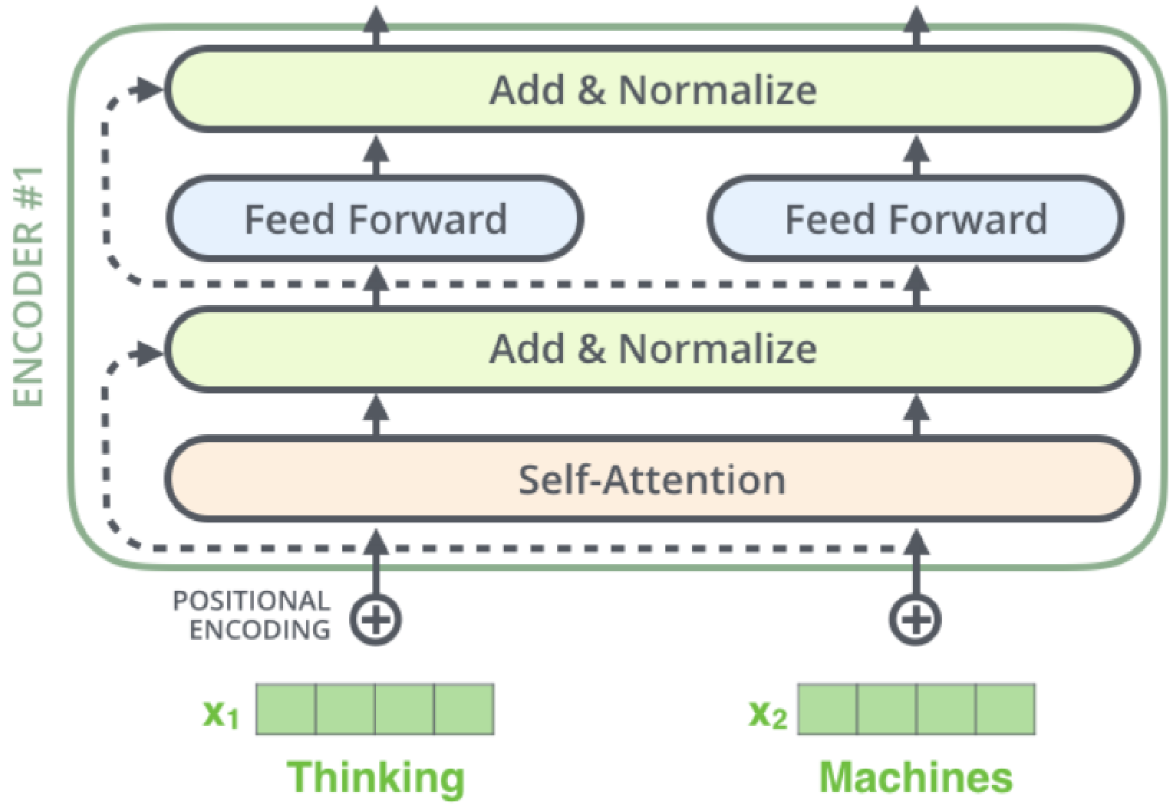
Transformers: Inside an Encoder

- Let’s take a look at the encoder. Two components:
  - 1. **Self-attention** layer (covered this)
  - 2. “Independent” **feedforward nets** for each head



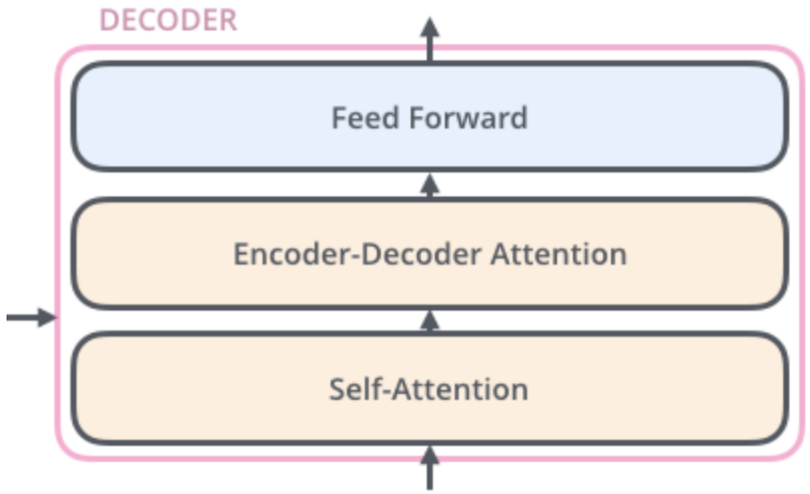
Transformers: More Tricks

- Recall a big innovation for ResNets: residual connections
  - And also layer normalizations
  - Apply to our encoder layers



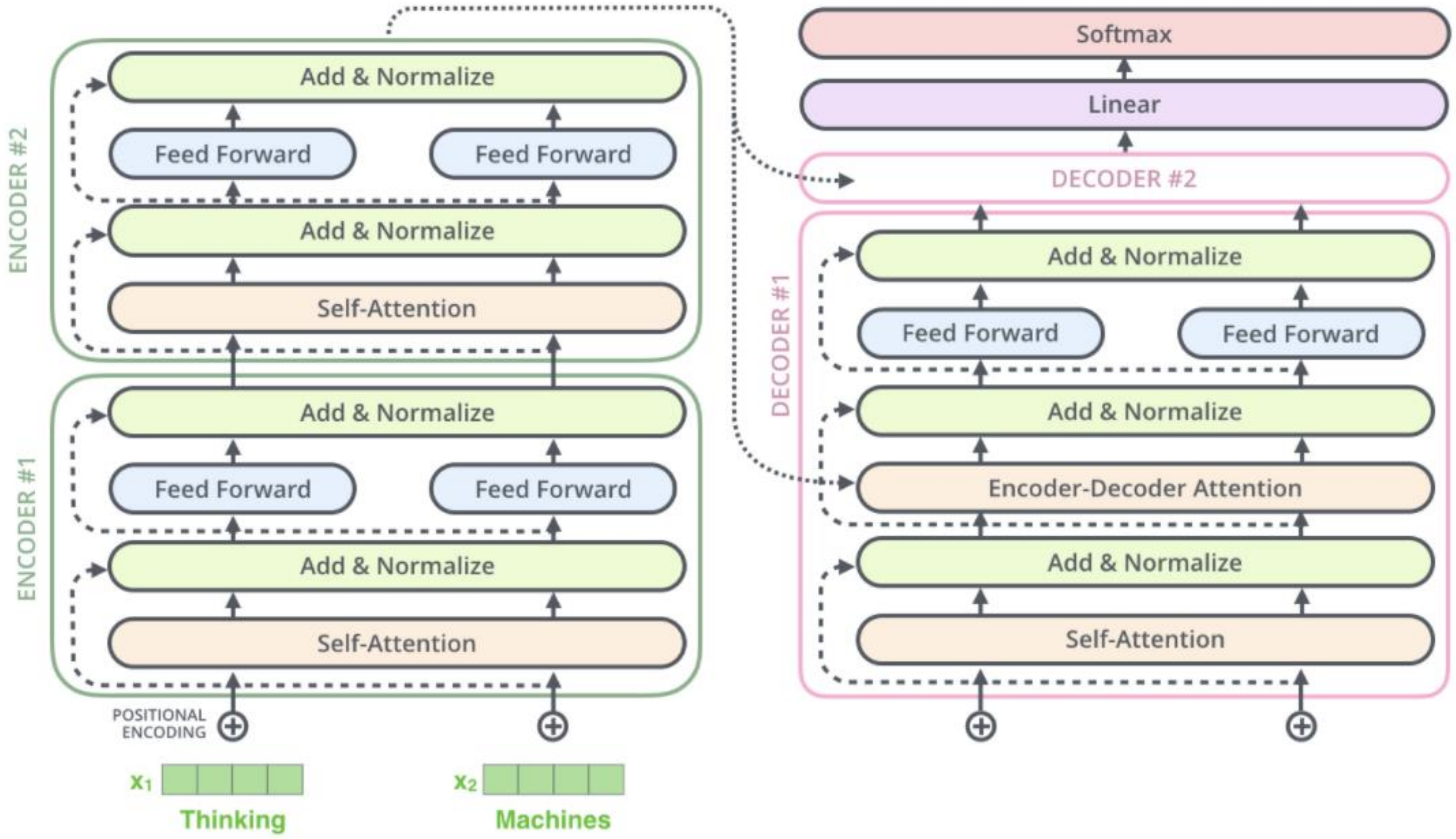
Transformers: Inside a Decoder

- Let’s take a look at the decoder. Three components:
  - 1. **Self-attention** layer (covered this)
  - 2. Encoder-decoder attention (same, but K, V come from encoder)
  - 3. “Independent” **feedforward nets** for each head



Transformers: Putting it All Together

- What does the full architecture look like?



Transformers: The Rest

- Next time: we'll talk about
  - How to **use** it (i.e., outputs)
  - How to **train** it
  - How to **rip** it apart and build other models with it.