Natural Language Processing

Self Attention and Transformers

Luke Zettlemoyer
lsz@cs.washington.edu

Adapted from slides from by Vidhisha Balachandran, Emma Strubell, Graham Neubig, Robert Minneker

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Readings

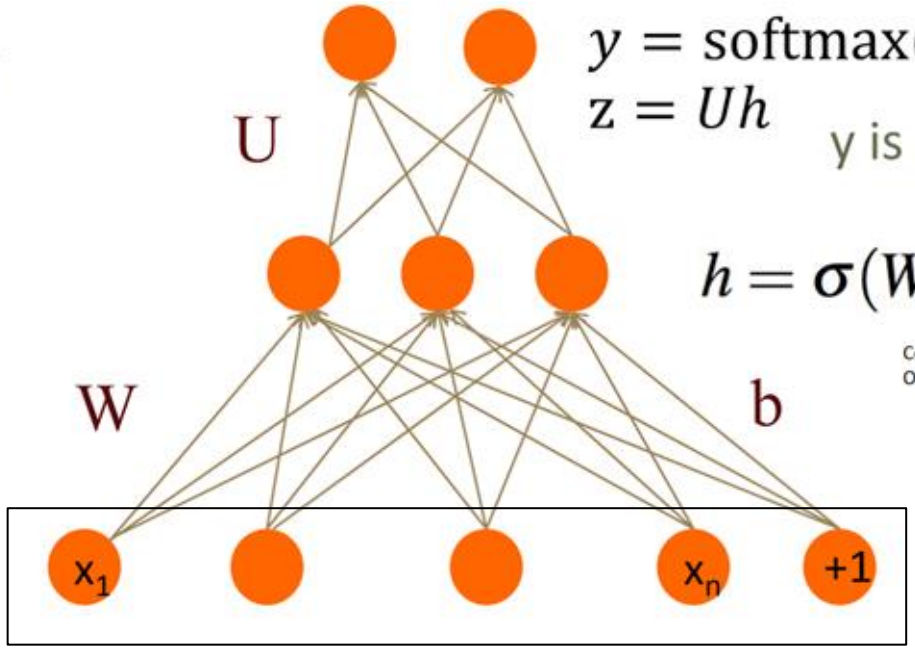
- Attention Is All You Need
- The Illustrated Transformer
- The Annotated Transformer
- Language Modeling with Transformers and PyTorch

Recap - 2 Layer MLP

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



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

$z = Uh$

y is a vector

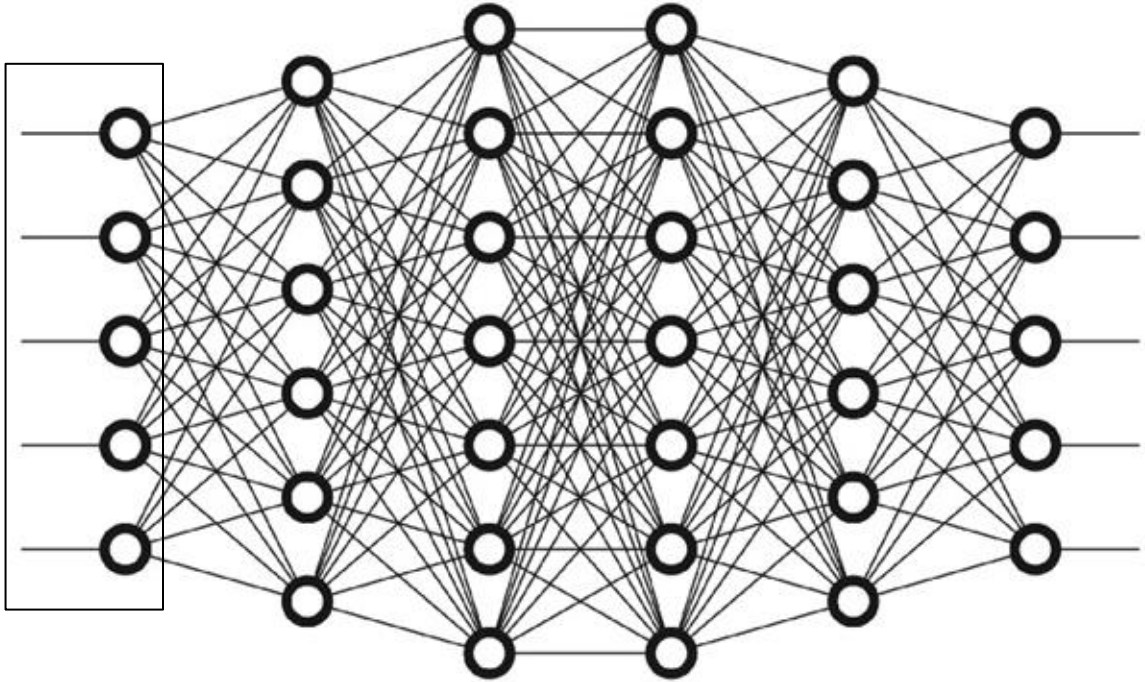
$h = \sigma(Wx + b)$

Could be ReLU
Or tanh

Fixed-Size Inputs

Deep MLP

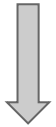
Fixed-Size Inputs



Fixed size representations for Natural Language

How to turn this

Learning NLP is fun!



Into this

[x_1, x_2, …, x_d]

(fixed length d-dimensional vector)

Fixed size representations for Natural Language

Remember Homework 1

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	ln(word count of doc)	ln(66) = 4.19

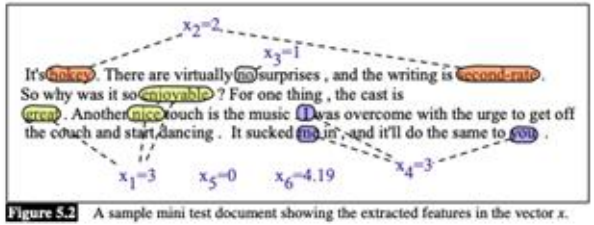


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Linguistic Features

As an example consider we have 2 examples present in our dataset:

x1: john likes to watch movies mary likes movies too

x2: mary also likes to watch football games

Based on these two documents we can get the list of all words that occur in this dataset which will be:

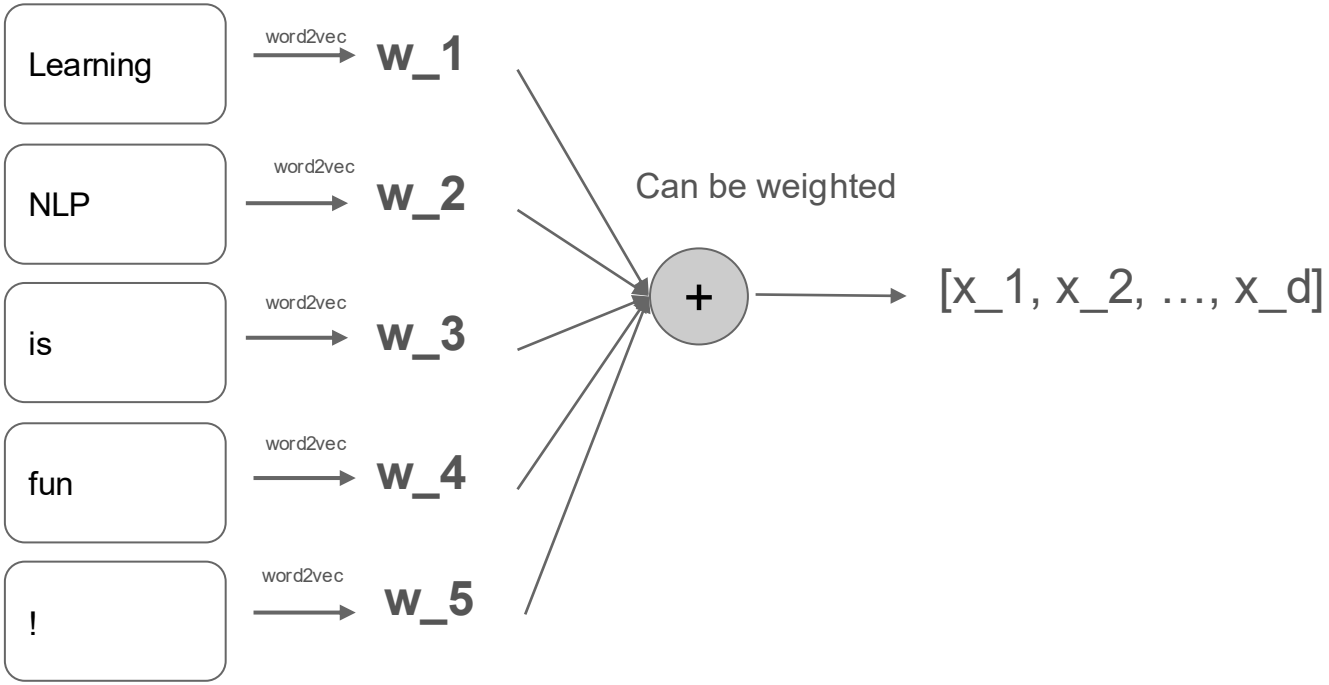
index	word
0	also
1	football
2	games
3	john
4	likes
5	mary
6	movies
7	to
8	too
9	watch

We can then define features for the two x1 and x2 as follows:

	also	football	games	john	likes	mary	movies	to	too	watch
x1	0	0	0	1	2	1	2	1	2	1
x2	1	1	1	0	1	1	0	0	0	0

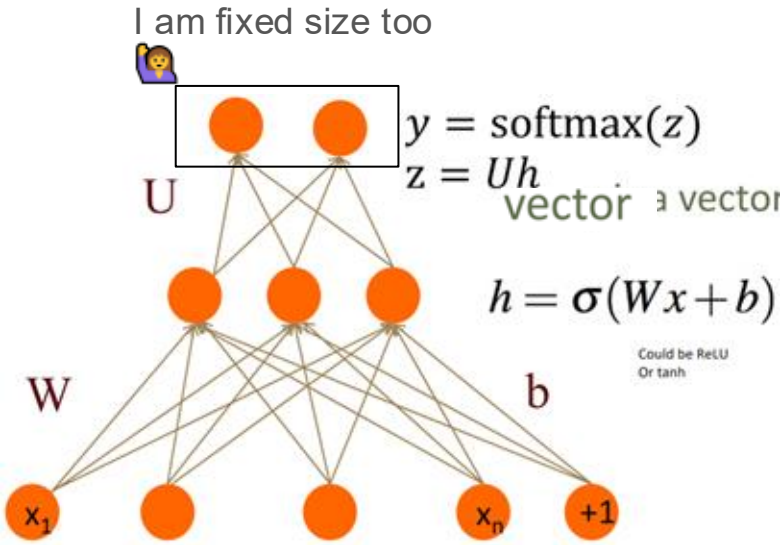
Bag of words with counts

Fixed size representations for Natural Language



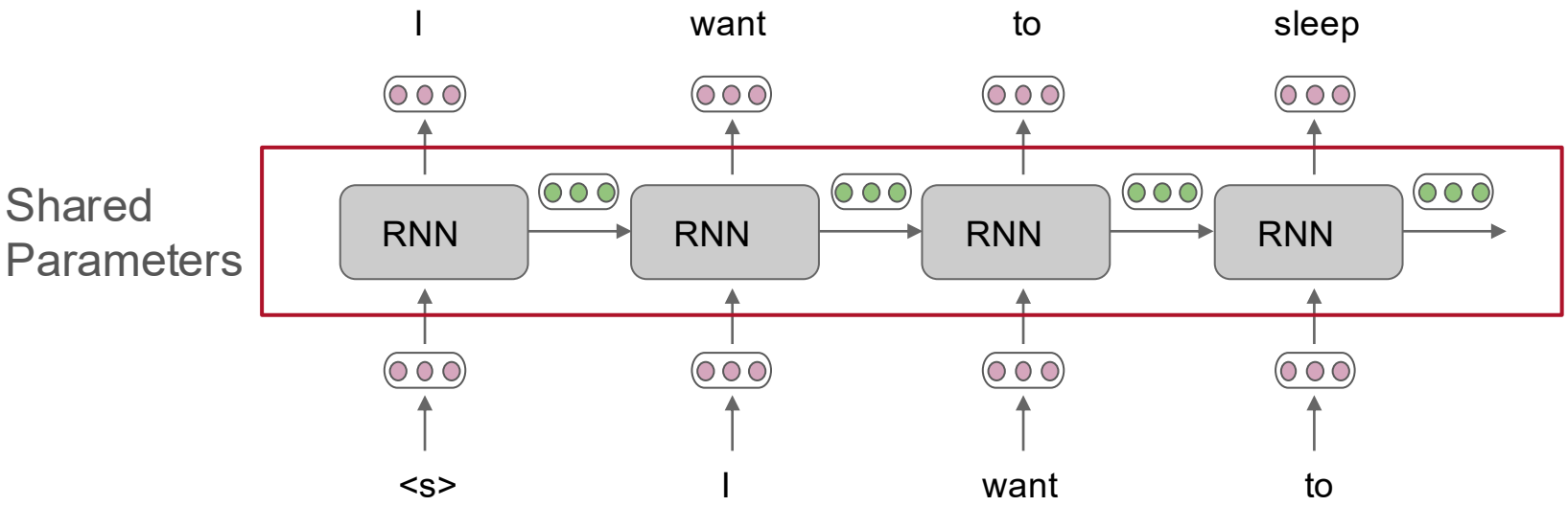
Problems with Fixed-Size Representations

1. Loss of important contextual information
2. Order invariant
3. Often Poor Generalization (Especially in case of Linguistic Features)
4. What if the output is a sequence too? Think about tasks like machine translation, good luck getting the sentence back from the fixed representation :)



Sequence Models To Rescue

Recurrent Neural Networks - RNNs



Recurrent Neural Networks - RNNs

For language it can be **sequence of words, characters, bytes** etc.

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

Both fixed size vectors

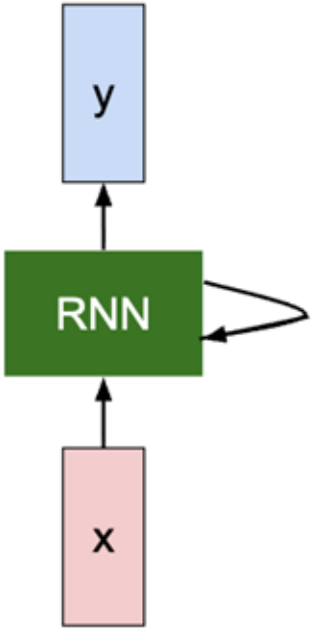
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

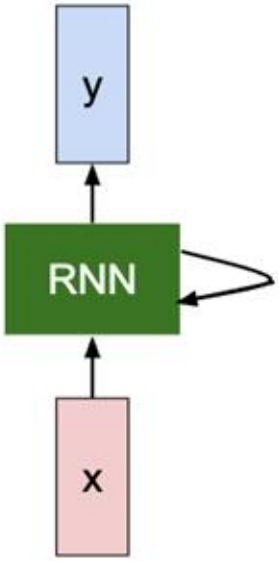
input vector at some time step



Think of x as word-vector

From CS231n slides. Lecture 8 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

Recurrent Neural Networks - RNNs



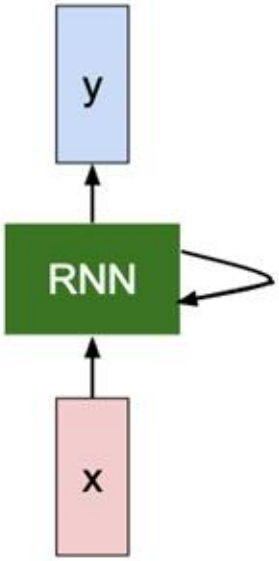
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \quad h_{t-1} + \quad x_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman

From CS231n slides. Lecture 8 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

Recurrent Neural Networks - RNNs



$$h_t = f_W(h_{t-1}, x_t)$$

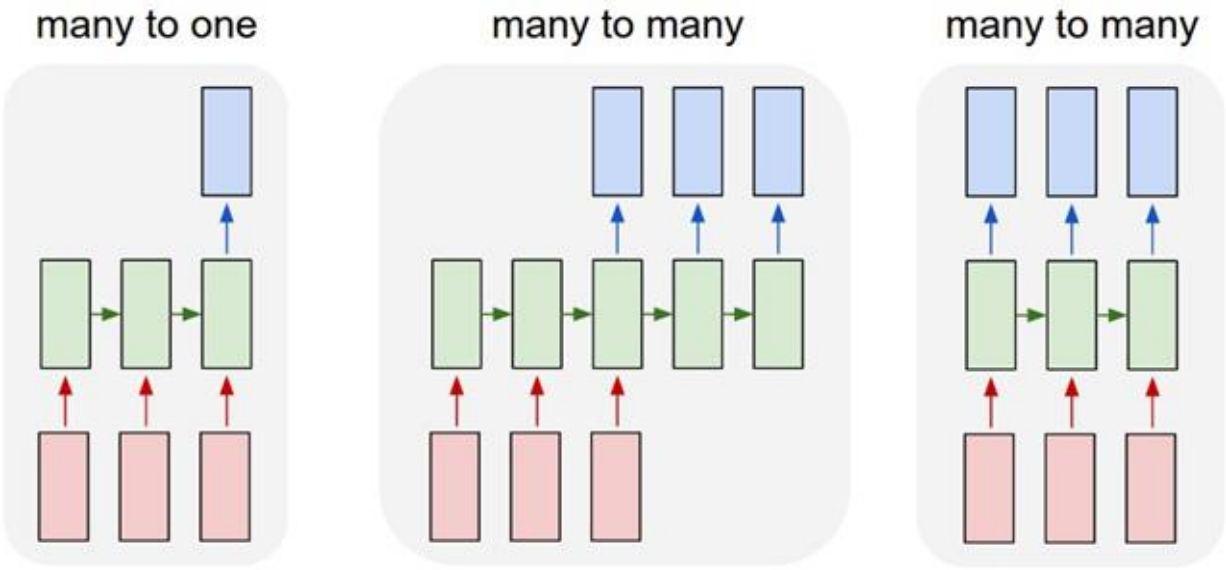
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman

From CS231n slides. Lecture 8 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

Multiple Ways of Stacking RNNs



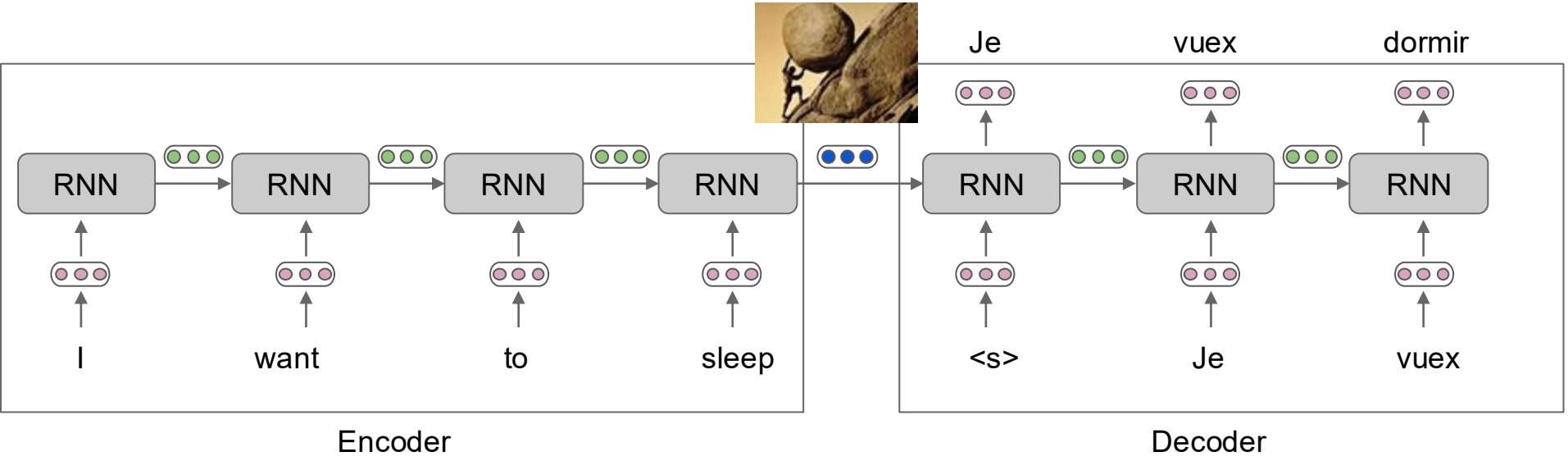
For tasks like classification

For tasks like machine translation

For tasks like language modeling

From The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy

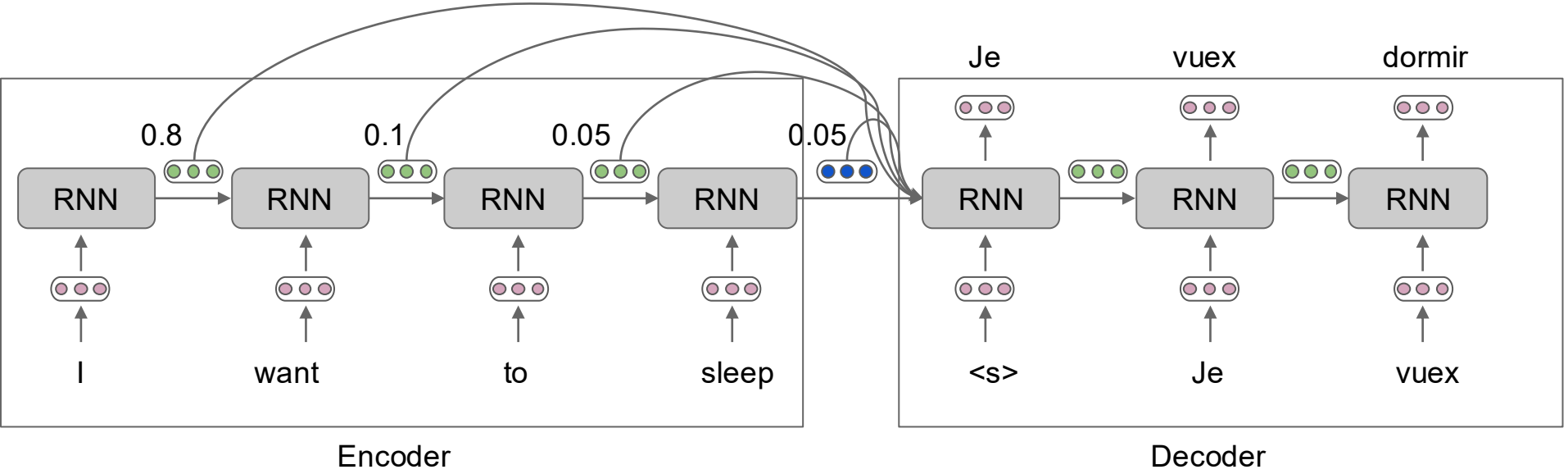
Encoder-Decoder Models



Typically Encoder and Decoder would be separate networks i.e. have their own separate weights

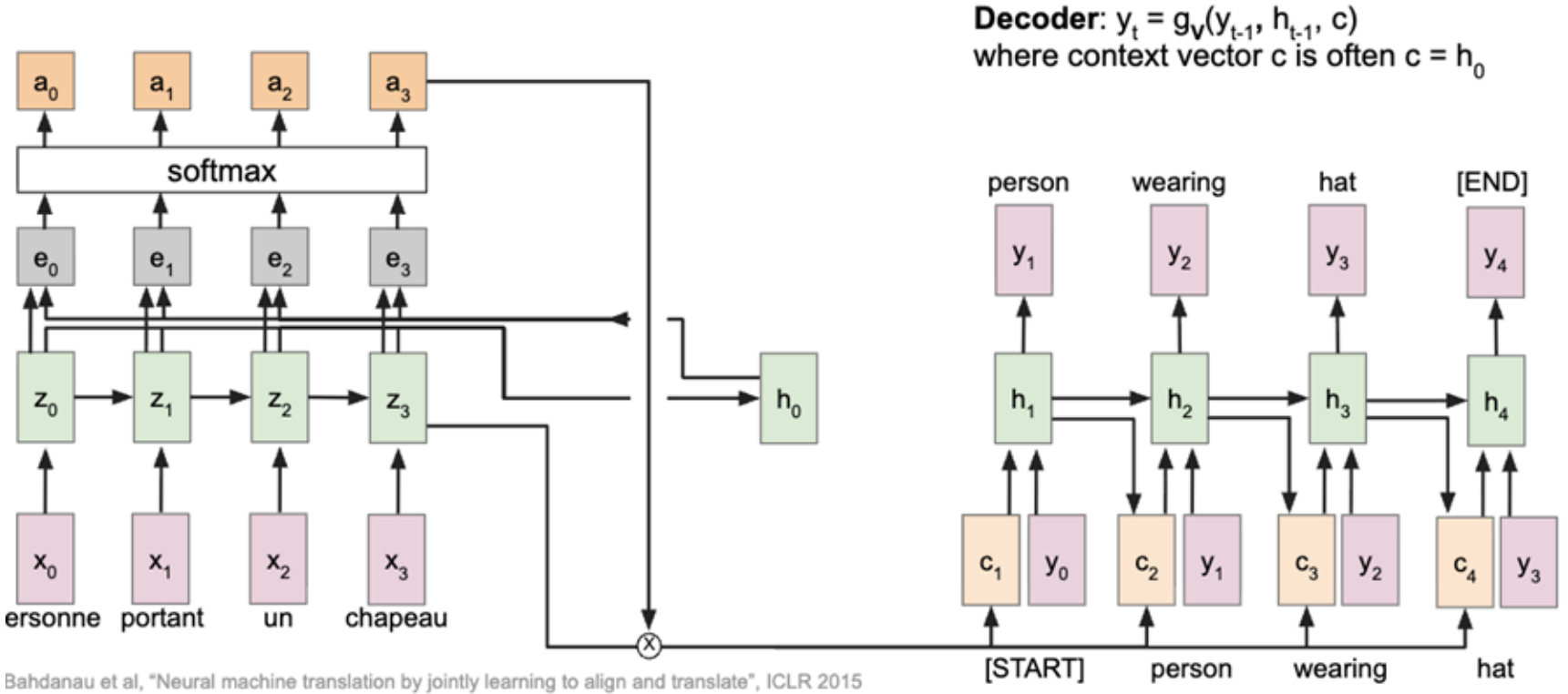
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks.

Encoder-Decoder Models With Attention



Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al, 2015

Encoder-Decoder Models With Attention



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

From CS231n slides. Lecture 9 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

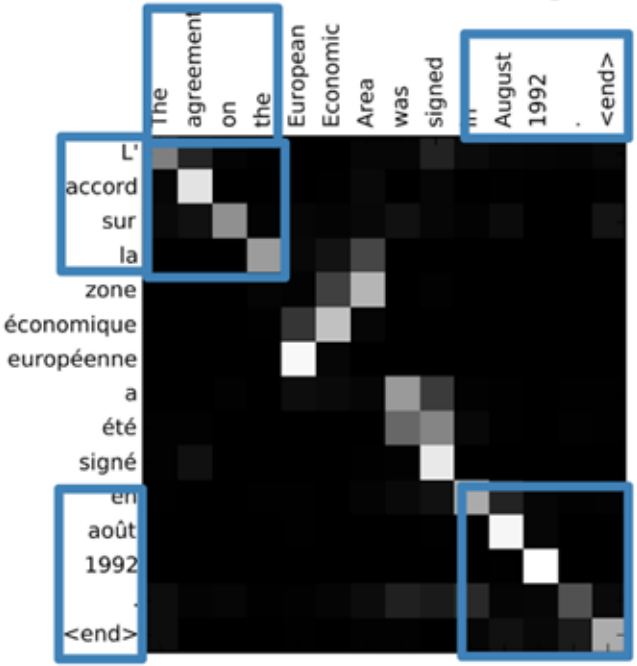
Encoder-Decoder Models With Attention

Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed **in August 1992**."

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

From CS231n slides. Lecture 9 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

What's wrong with RNNs?

- Long Range Dependencies
- Gradient vanishing / explosion
- Long time to converge
- Expensive computation

Long Range Dependencies

I'm want to watch Wicked! How does the weather in NYC look next week?

It looks sunny with some light rain during the weekend.

Oh! But I don't have a rain jacket :(Is there a store nearby?

There's a marshall's a mile away. They have the navy blue jacket you have been eyeing for a while!

Long Range Dependencies

I'm want to watch Wicked! How does the weather in NYC look next week?

It looks sunny with some light rain during the weekend.

Oh! But I don't have a rain jacket :(Is there a store nearby?

There's a marshall's a mile away. They have the navy blue jacket you have been eyeing for a while!

Ok! Looks like I can actually go! Book **the tickets** for next Wed!

Long Range Dependencies

I'm want to watch **Wicked**! How does the weather in NYC look next week?

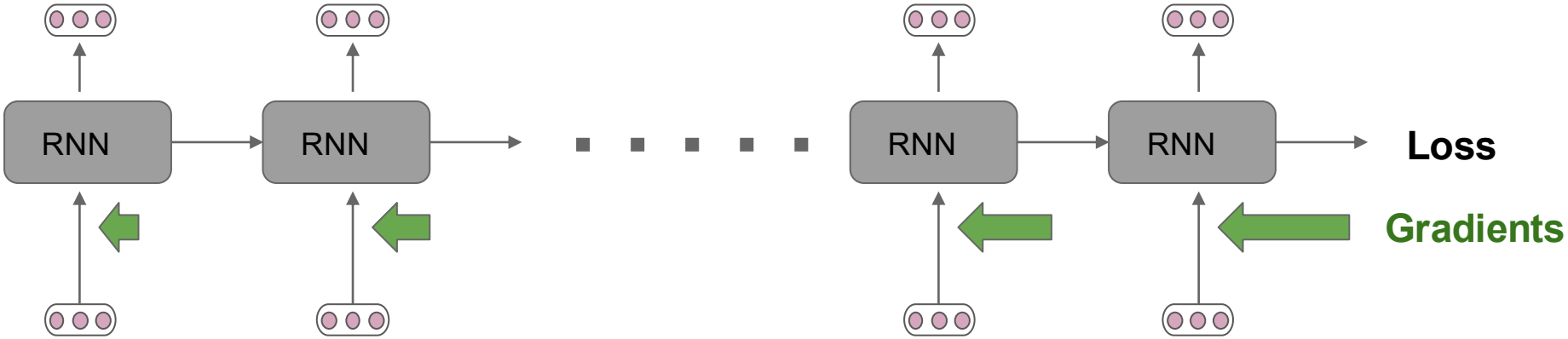
It looks sunny with some light rain during the weekend.

Oh! But I don't have a rain jacket :(Is there a store nearby?

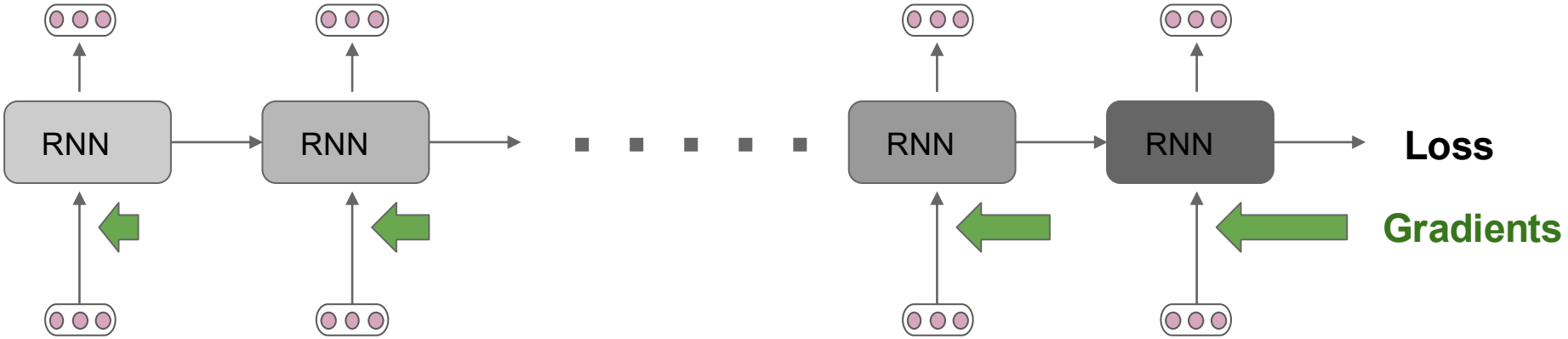
There's a marshall's a mile away. They have the navy blue jacket you have been eyeing for a while!

Ok! Looks like I can actually go! Book **the tickets** for next Wed!

Gradient vanishing / explosion



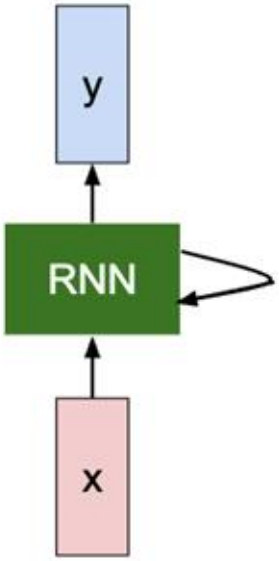
Gradient vanishing / explosion



What's wrong with RNNs?

- Long Range Dependencies
- Gradient vanishing / explosion
- Long time to converge
- Expensive computation

Recurrent Neural Networks - RNNs



$$h_t = f_W(h_{t-1}, x_t)$$

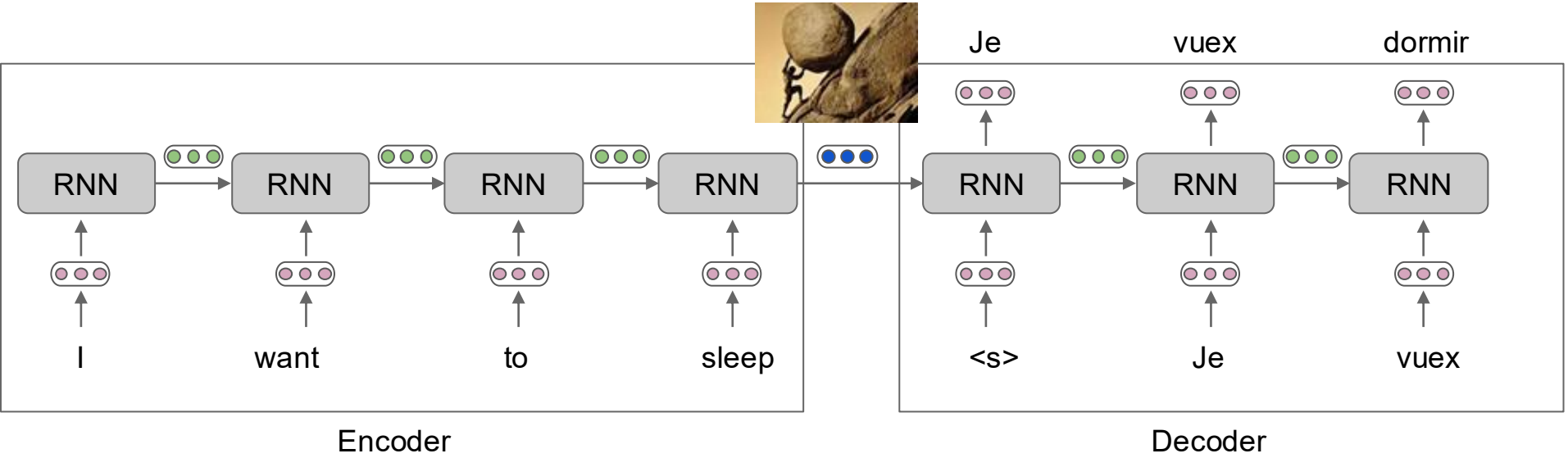
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman

From CS231n slides. Lecture 8 by Fei-Fei Li, Yunzhu Li, Ruohan Gao

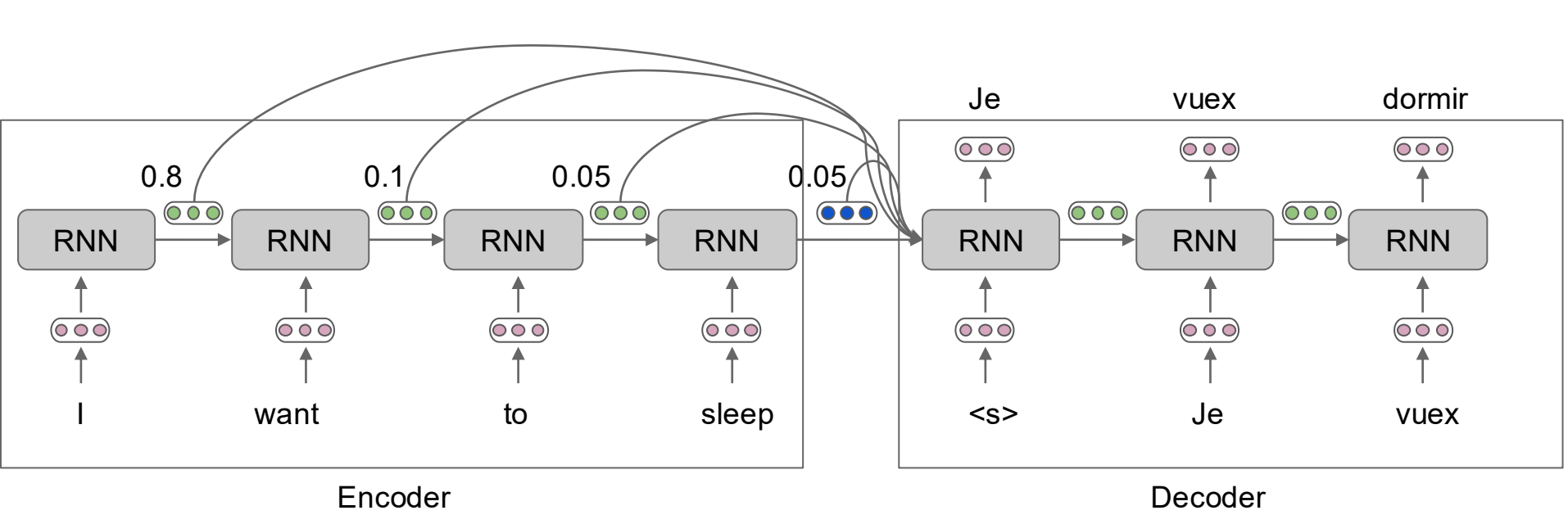
Encoder-Decoder Models



Typically Encoder and Decoder would be separate networks i.e. have their own separate weights

Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks.

Encoder-Decoder Models With Attention



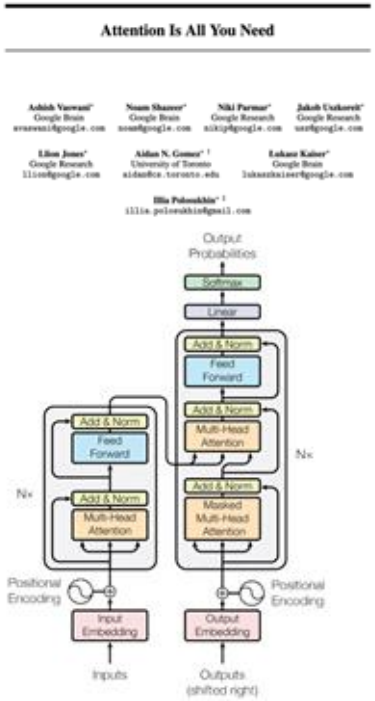
Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al, 2015

Encoder-Decoder Models With Attention

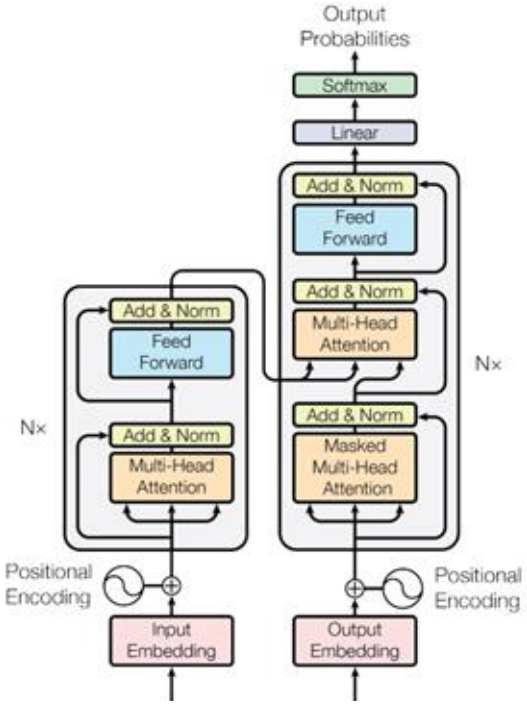
Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015


The BEATLES
All You Need Is Attention
Baby, You're a Rich Man
Capitol
5964


Attention Is All You Need
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Transformer Model



Google Research

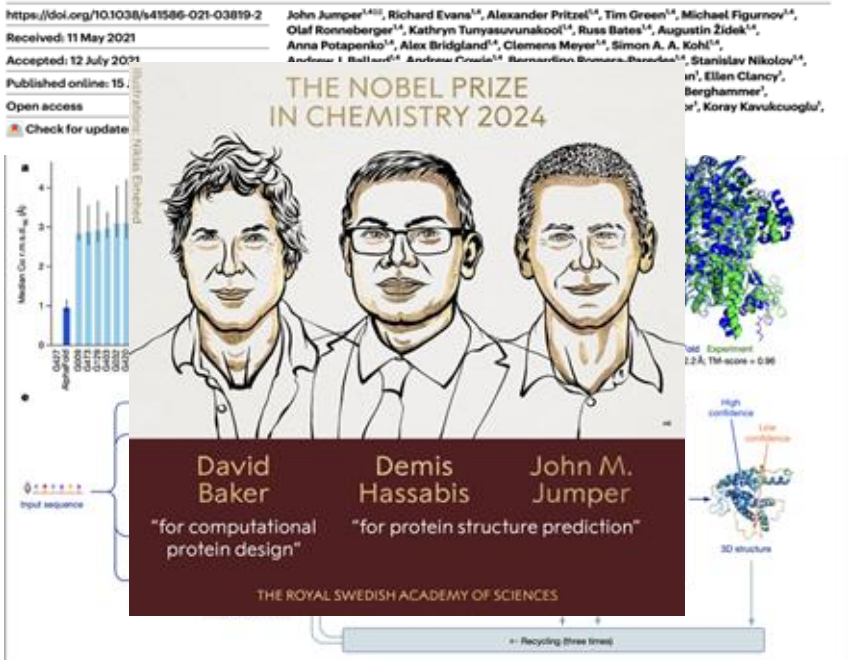
Attention is all you need (Vaswani et.al, 2017)

Real World Impact

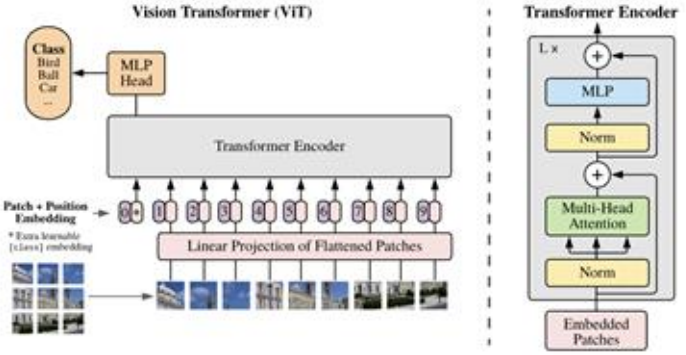
ChatGPT

Real World Impact

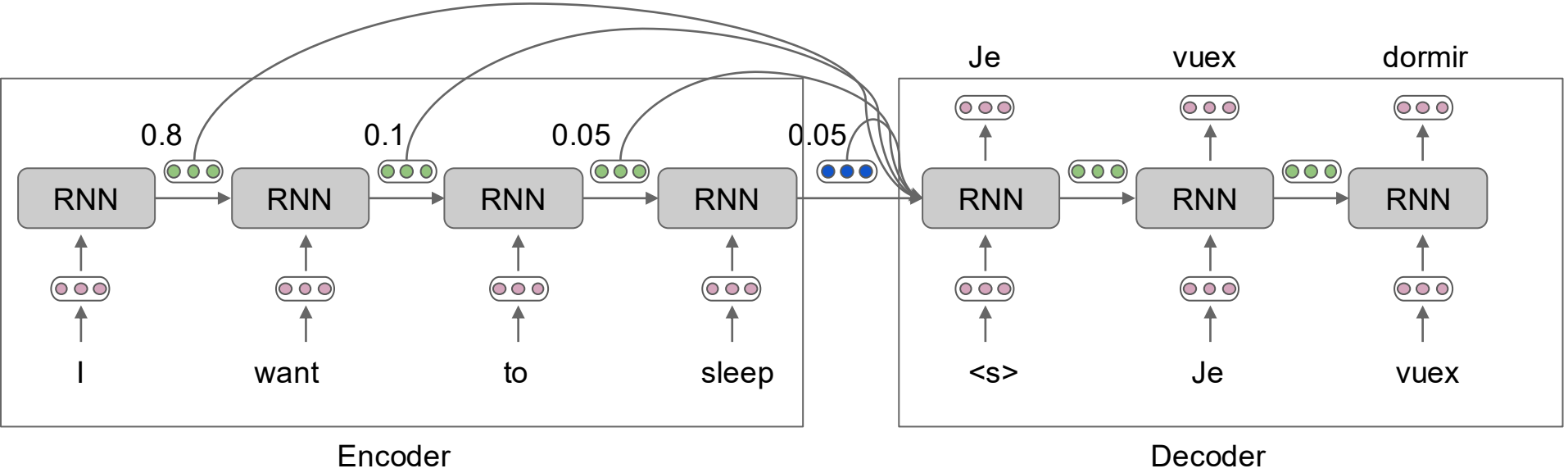
Highly accurate protein structure prediction with AlphaFold



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

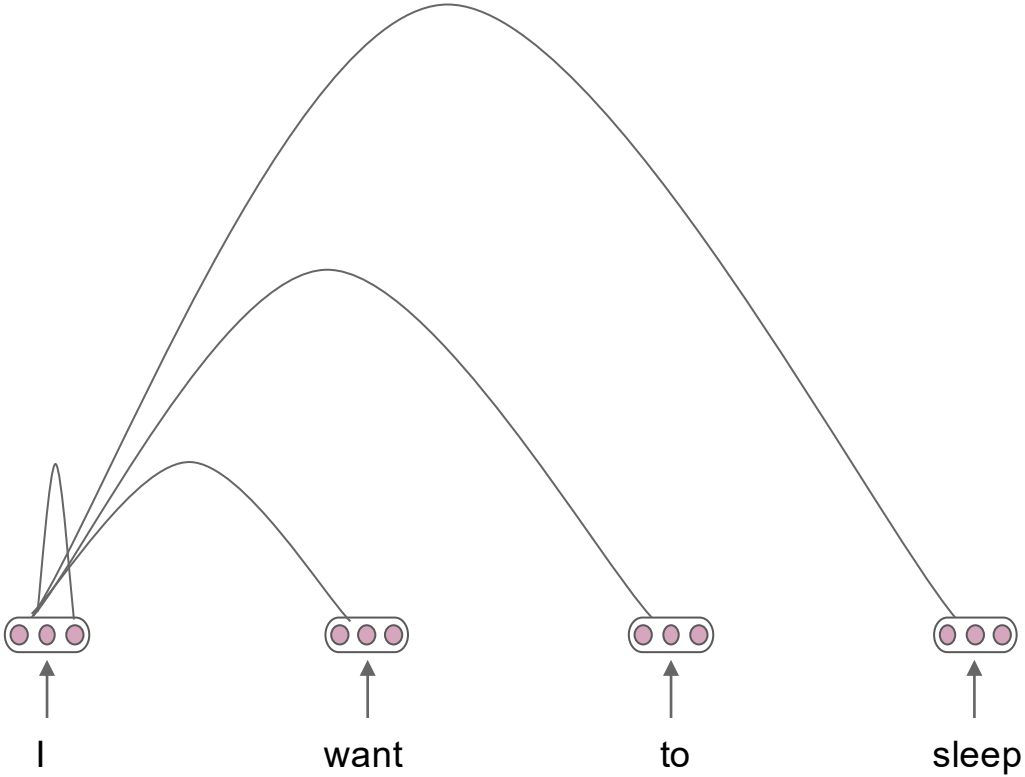


Last look at RNNs

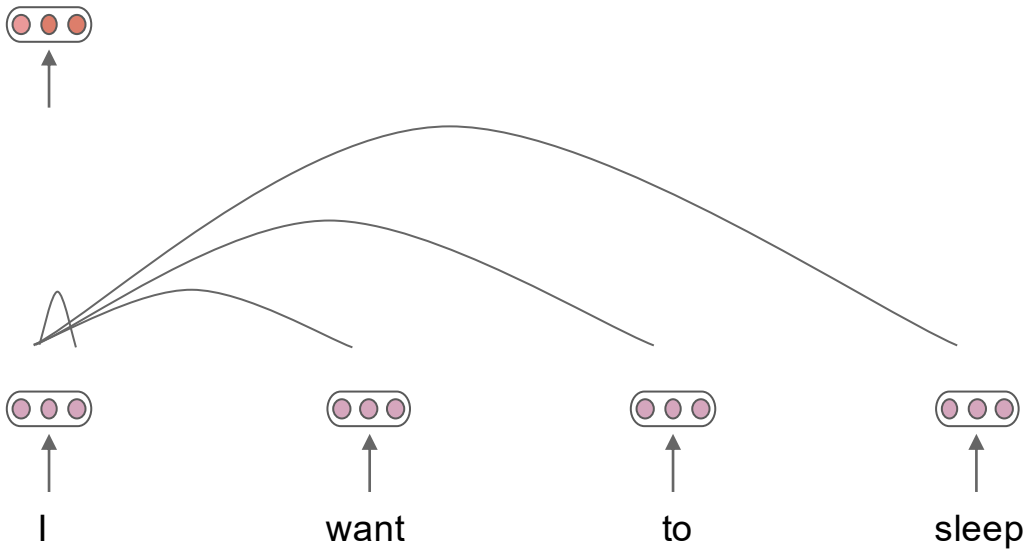


Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al, 2015

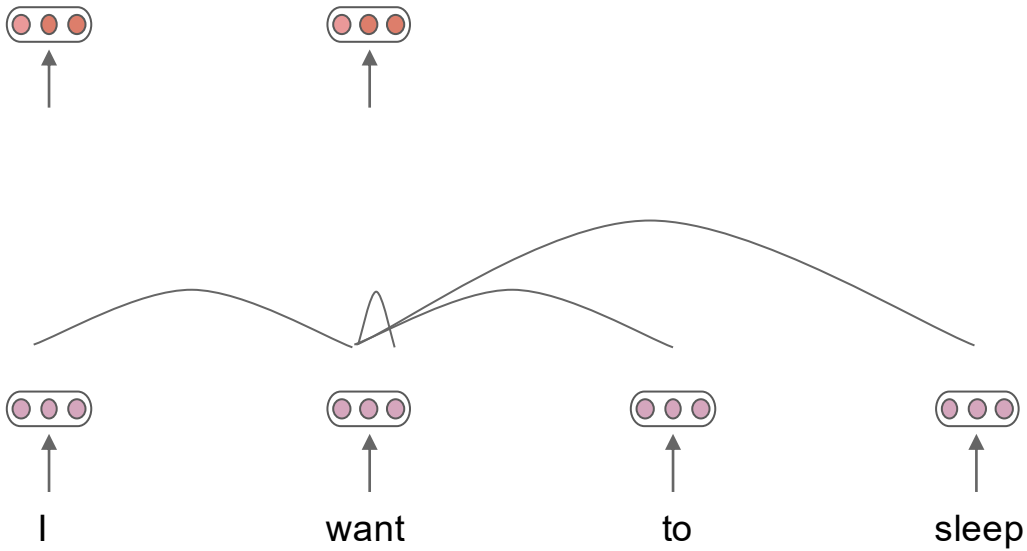
Self Attention



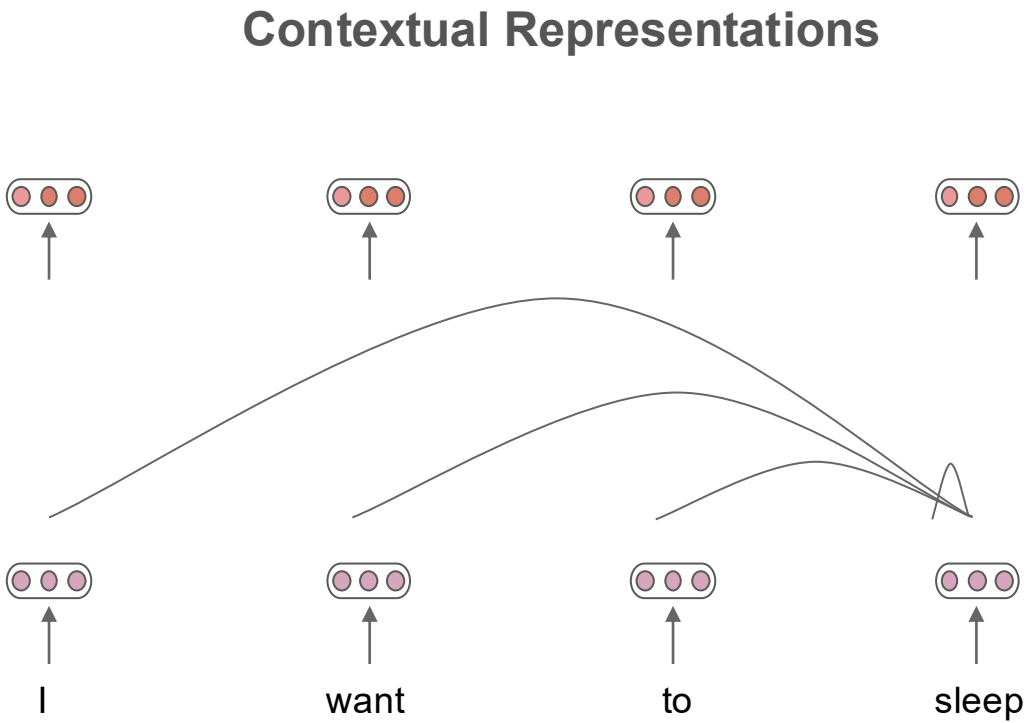
Self Attention - No more recurrence



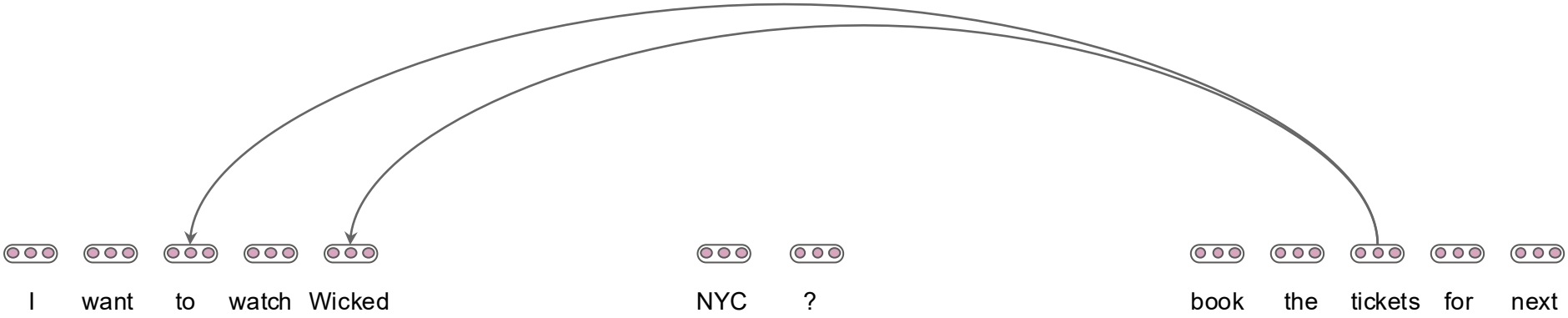
Self Attention - No more recurrence



Self Attention - No more recurrence



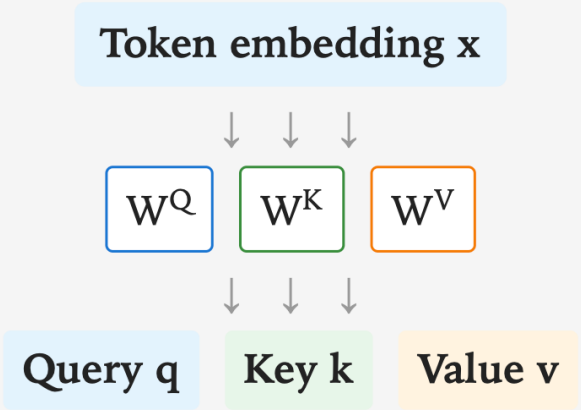
Self Attention for long sequences



Self Attention

For input embedding $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{x}_i W^Q, \quad \mathbf{k}_i = \mathbf{x}_i W^K, \quad \mathbf{v}_i = \mathbf{x}_i W^V$$



- $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are **learned** projection matrices

Self Attention

Query (Q)

"*What am I looking for?*"

Represents the current token's information needs

Key (K)

"*What do I contain?*"

Represents what information this token offers

Value (V)

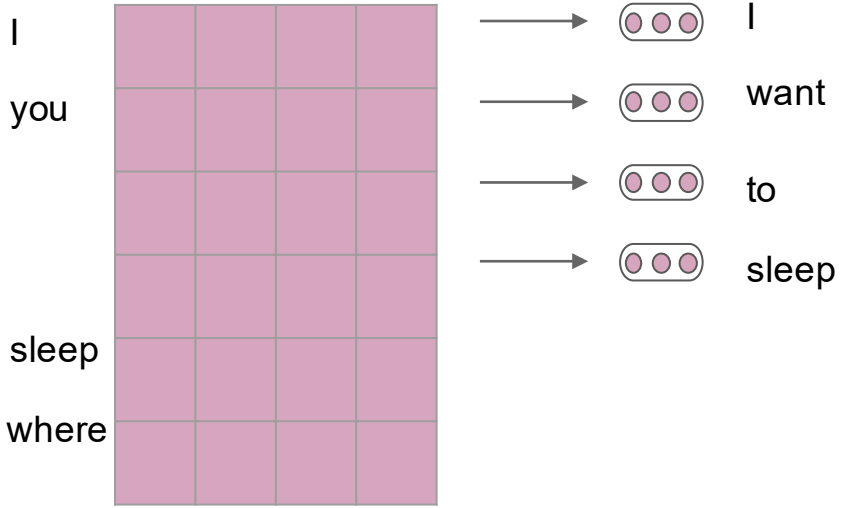
"*What do I provide?*"

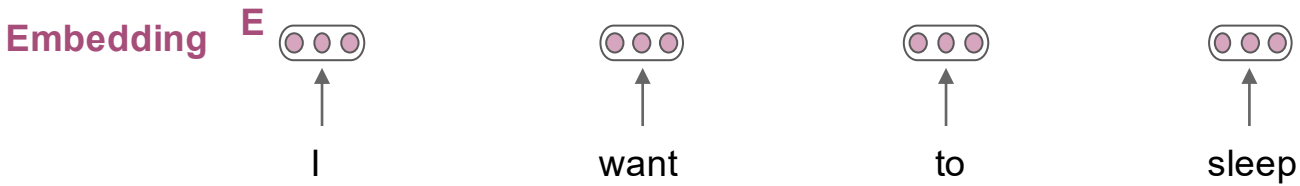
The actual information to be retrieved

Analogy: Like a search engine where:

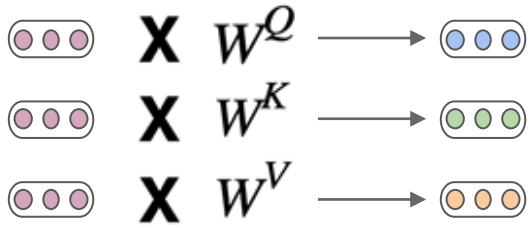
- Query = your search terms
- Key = document titles/metadata
- Value = document contents

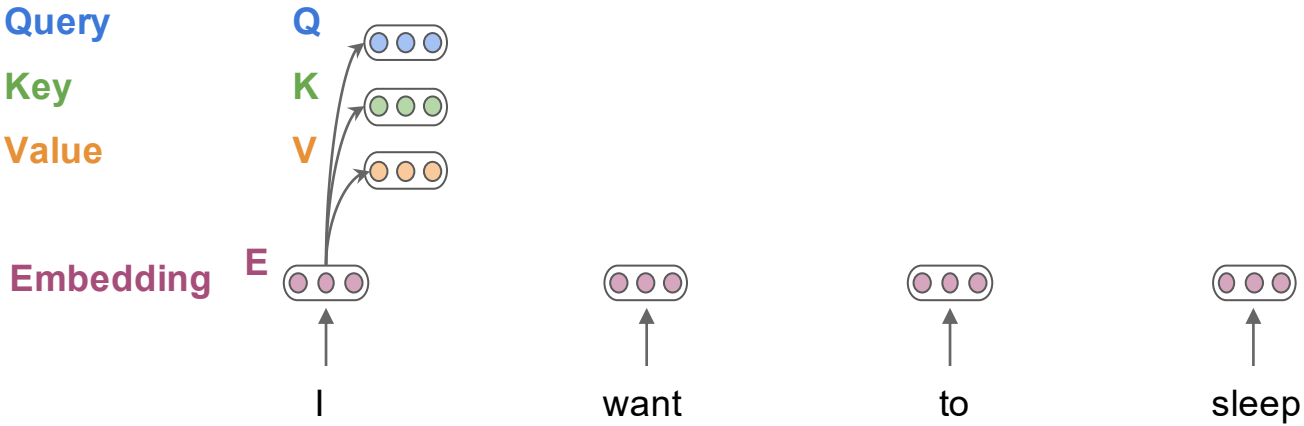
Self Attention - Word Embedding



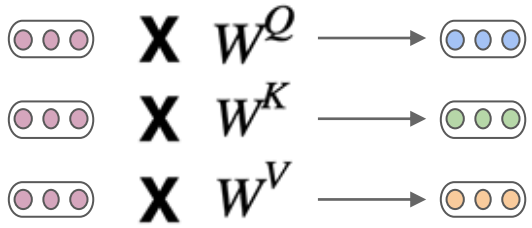


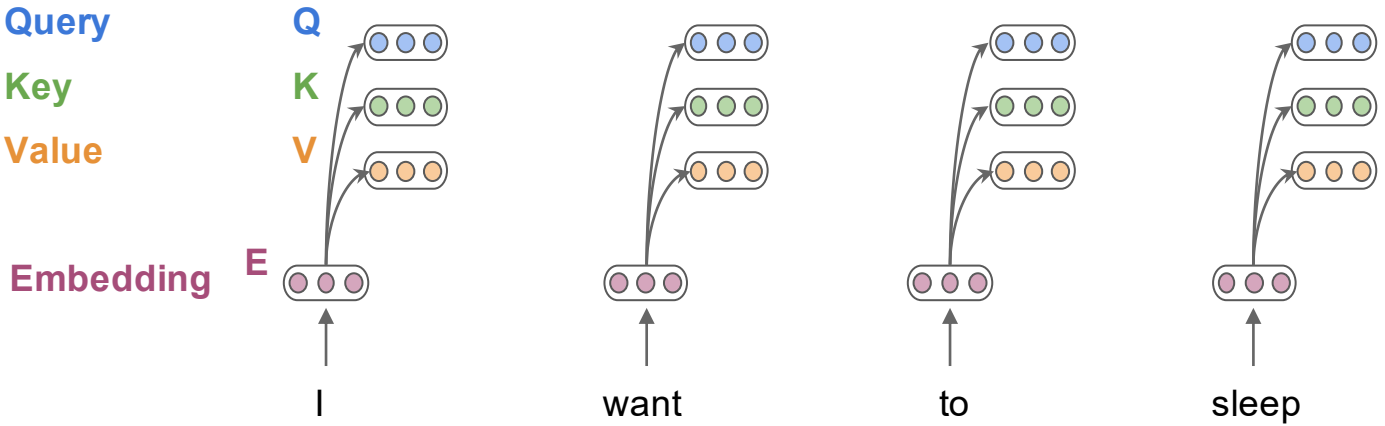
Self Attention - Projection Layer



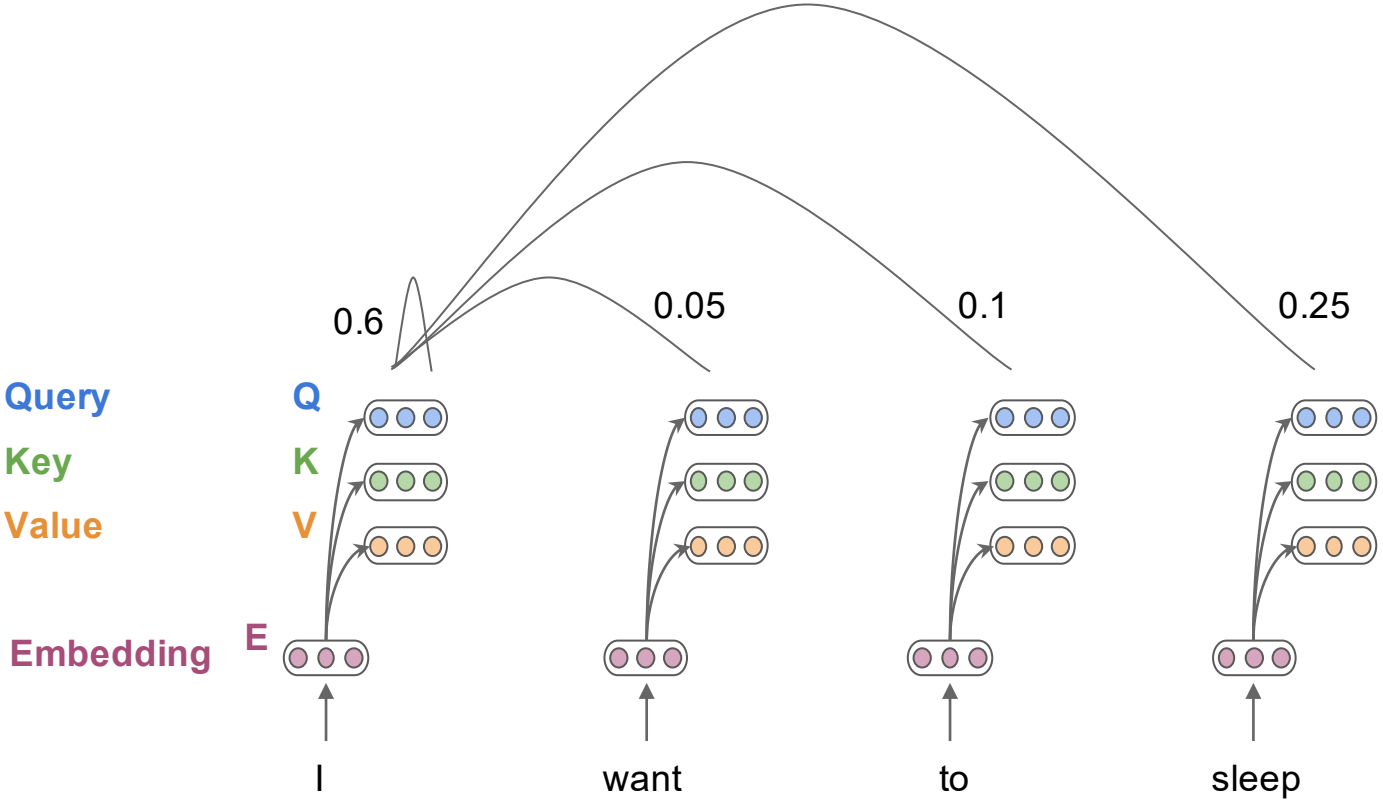


Self Attention - Projection Layer

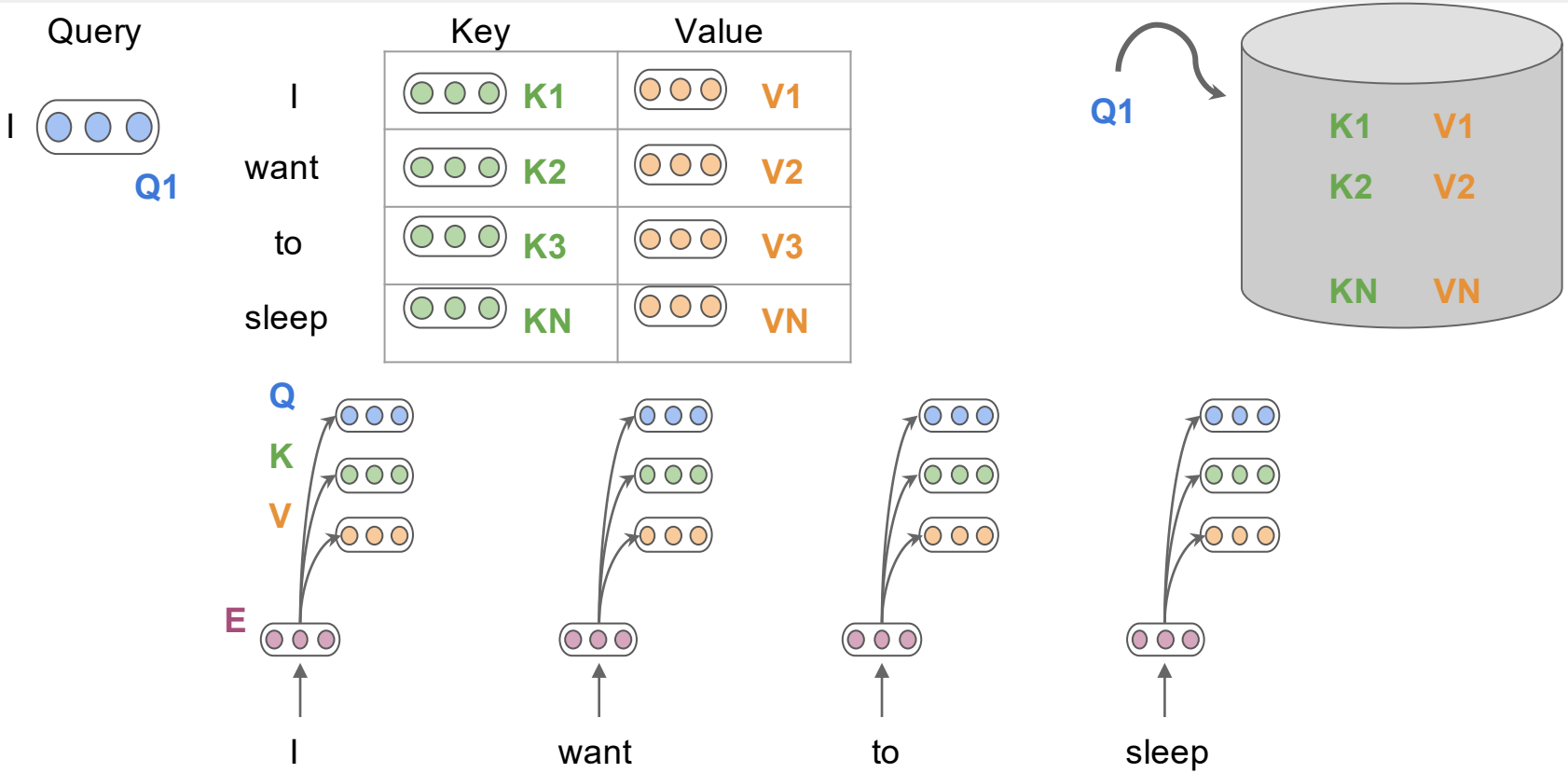




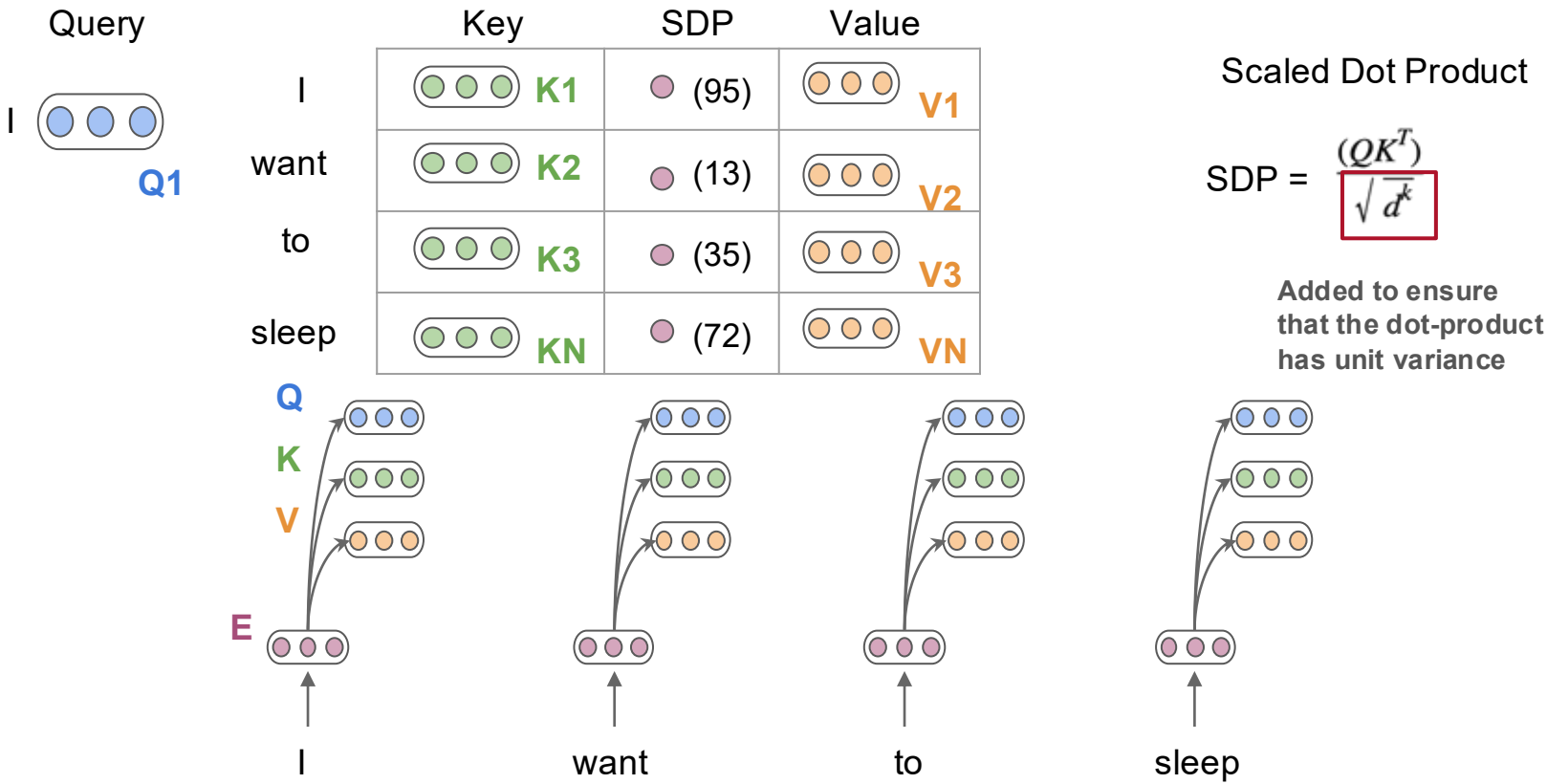
Self Attention - Attention Scores



Self Attention



Self Attention - Scaled Dot Product



Scaled Dot Product

$$\text{SDP} = \frac{(QK^T)}{\sqrt{d^k}}$$

Added to ensure that the dot-product has unit variance

Scaling Prevent Saturation

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

Why divide by $\sqrt{d_k}$?

Without Scaling

`scores = [8.5, 42, 3.2, 42]`

`softmax ≈ [0, 0.55, 0, 0.45]`

Extreme values → near one-hot → vanishing gradients

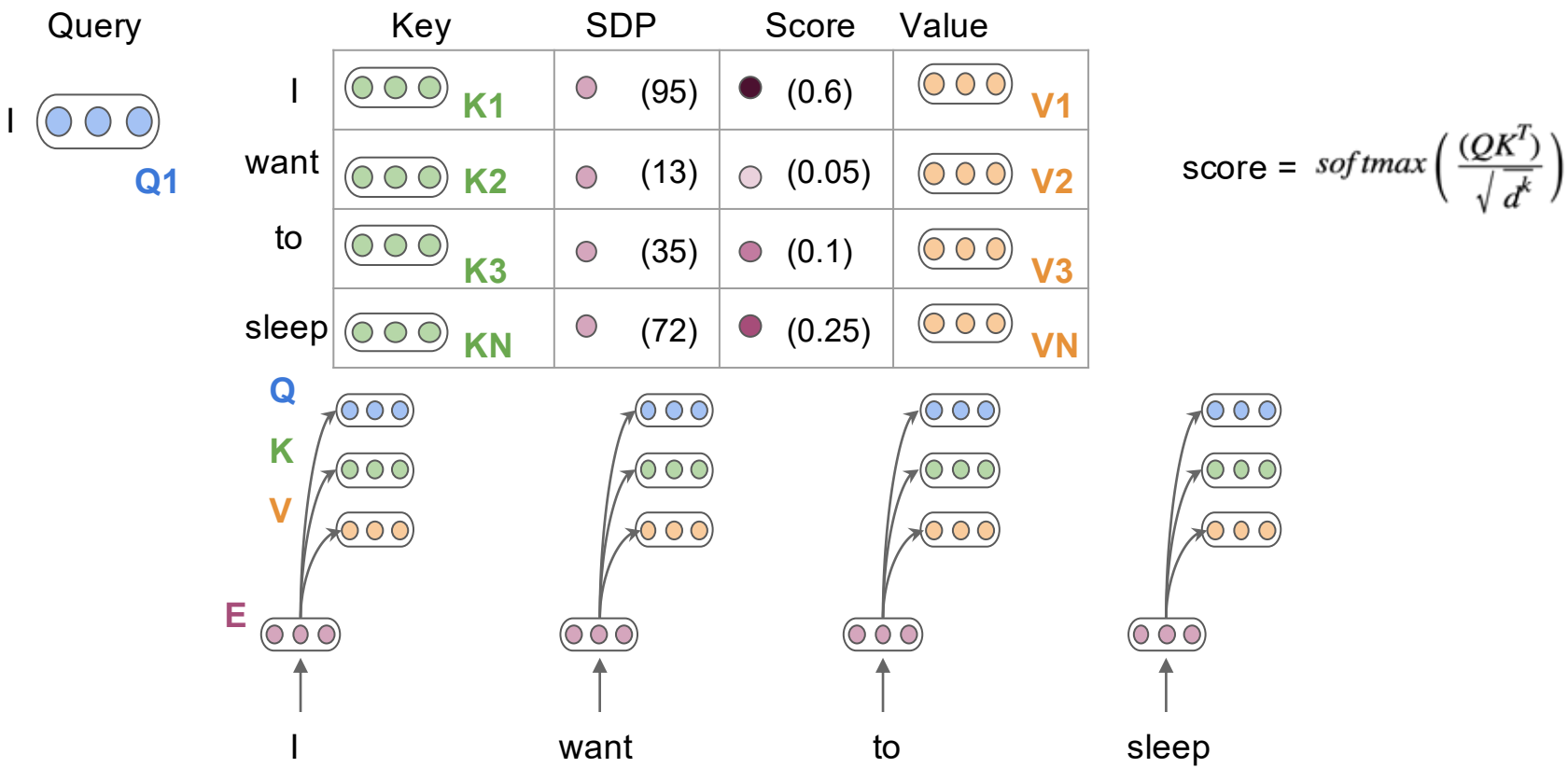
With Scaling (d_k=64)

`scores/8 = [1.1, 5.3, 0.4, 5.2]`

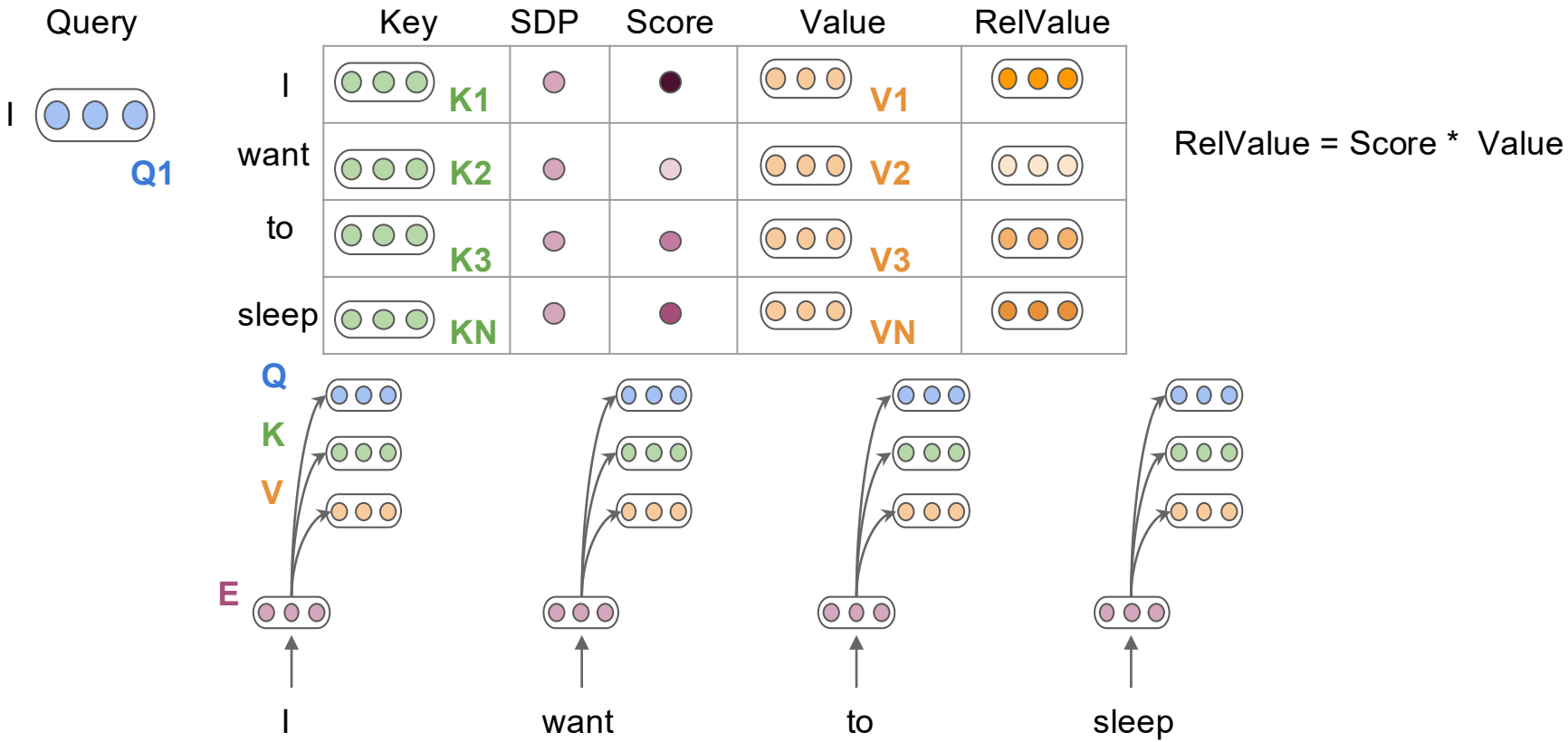
`softmax ≈ [0.01, 0.49, 0, 0.49]`

Moderate values → smooth distribution → healthy gradients

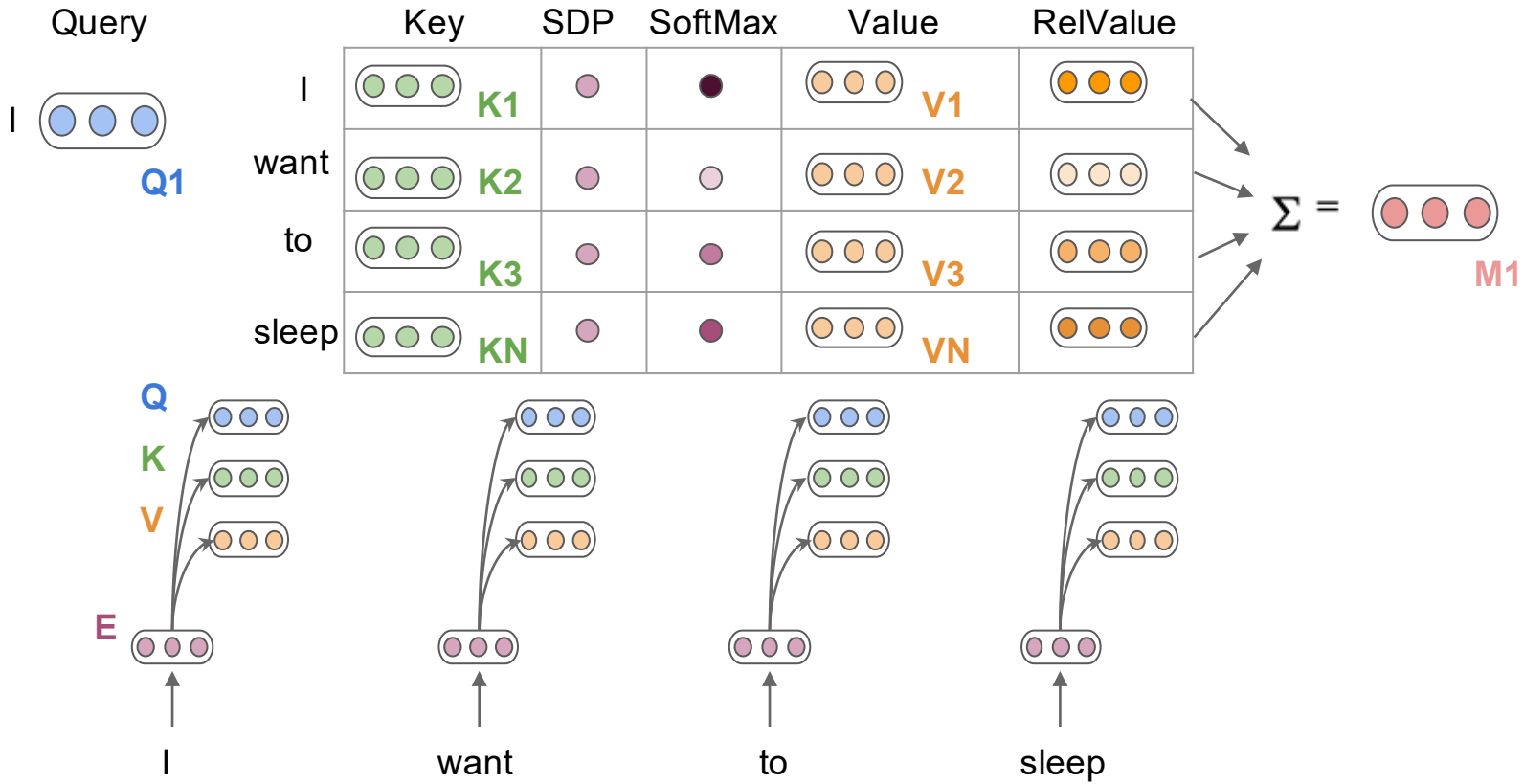
Self Attention - SoftMax



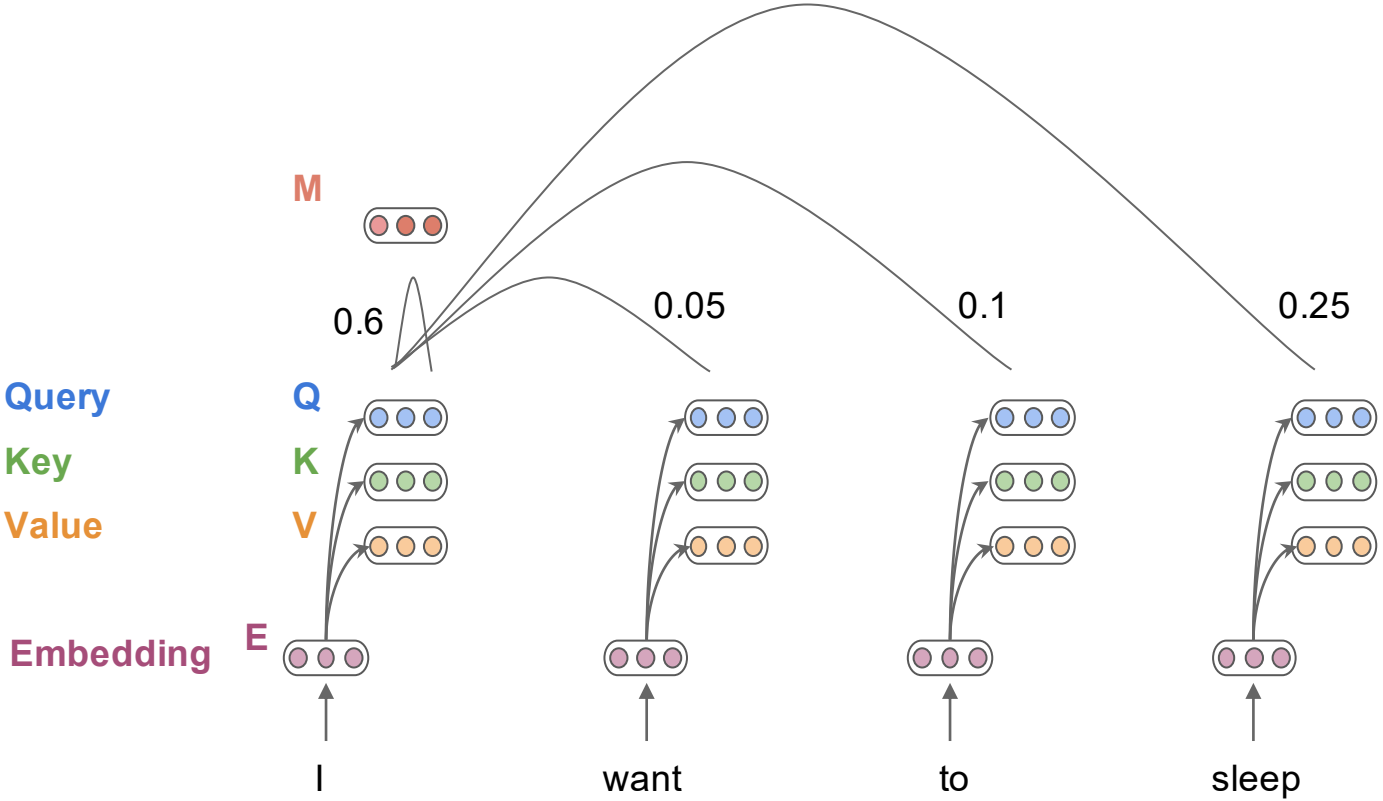
Self Attention - Soft (Relative) Values



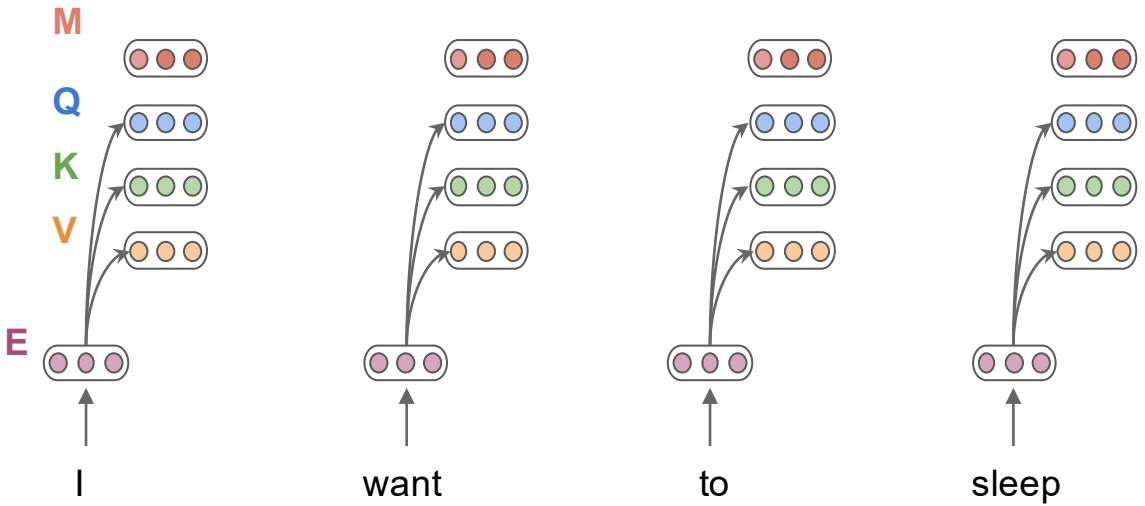
Self Attention - Attended Repr



Self Attention - Attended Contextual Rep



Self Attention - Attended Contextual Rep



All of that in a single line of linear algebra…

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

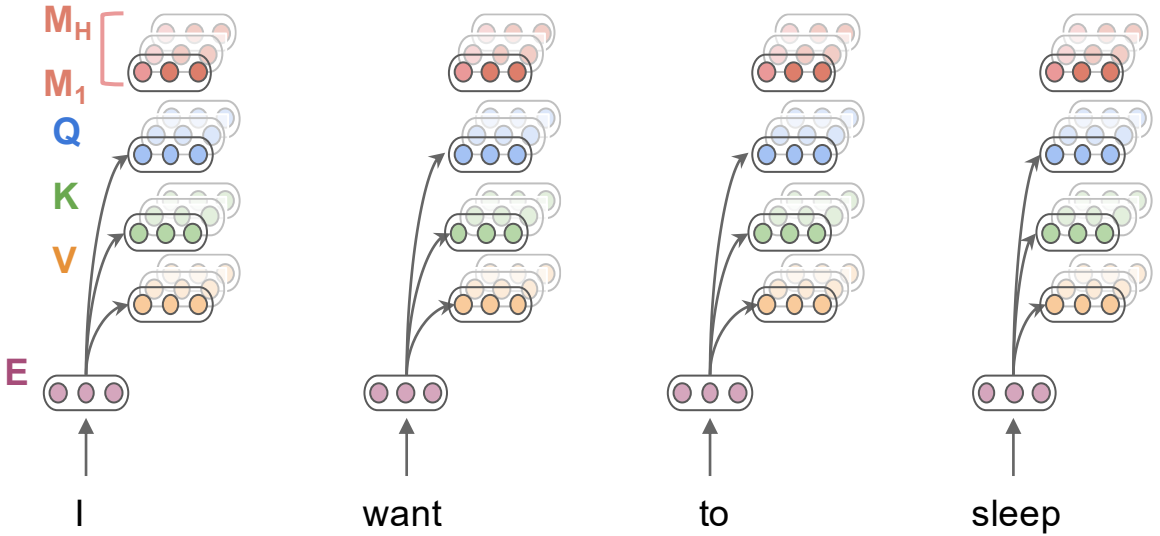
Q		K^T		Scores		softmax		V		Output
$[n \times d_k]$	×	$[d_k \times n]$	=	$[n \times n]$	→	$[n \times n]$	×	$[n \times d_v]$	=	$[n \times d_v]$

- **One matrix multiply** computes all n^2 pairwise attention scores
- GPUs are highly optimized for matrix multiplication
- This enables massive parallelism

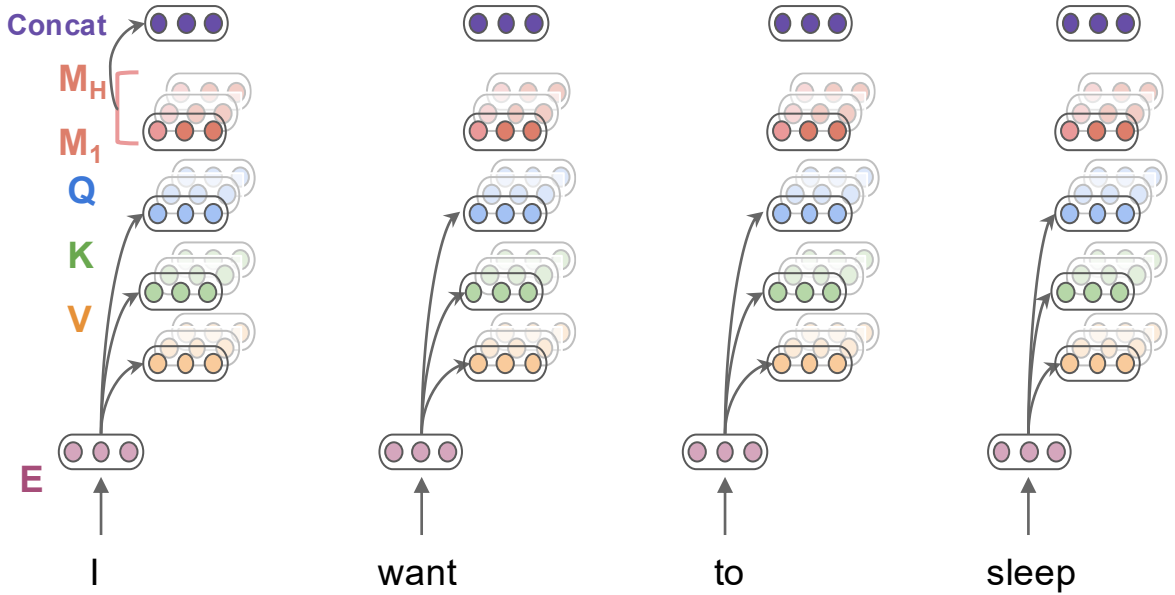
Multi-Headed Self Attention

Love me some representation power!

H (no: of heads) Different versions of Q,K,V
Each different repr -> Different attended repr



Multi-Headed Self Attention



Multi-Headed Self Attention

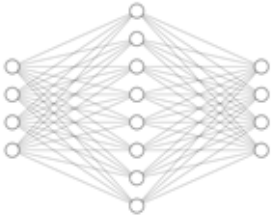
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

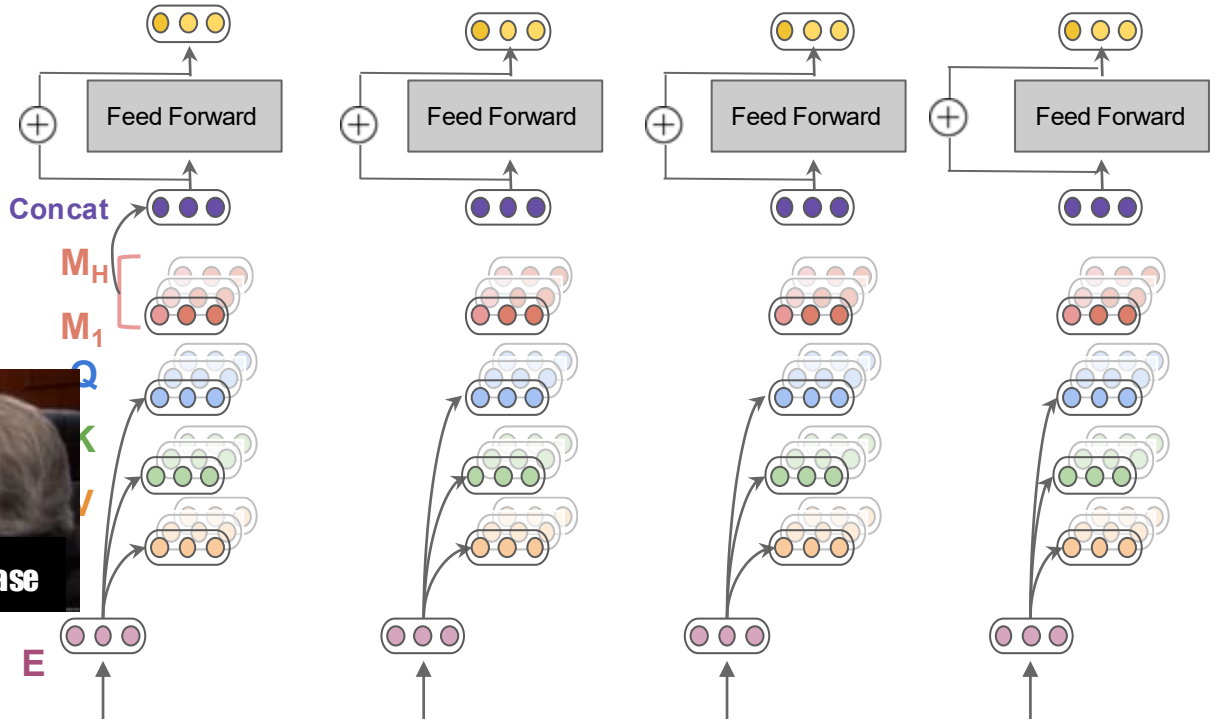
Head 1	**Head 2**	**Head 3**	**Head 4**	**...**
Subject-verb	Adjective-noun	Coreference	Positional	Other patterns

- Each head has its **own** W^Q, W^K, W^V matrices
- Heads learn to specialize in different relationship types
- Typical: 8-16 heads with $d_k = d_{\text{model}}/h$

Multi-Headed Self Attention



It's Me MLP



Residual Connections : Help with cleaner gradient flows during back-prop

FF independent for each position

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

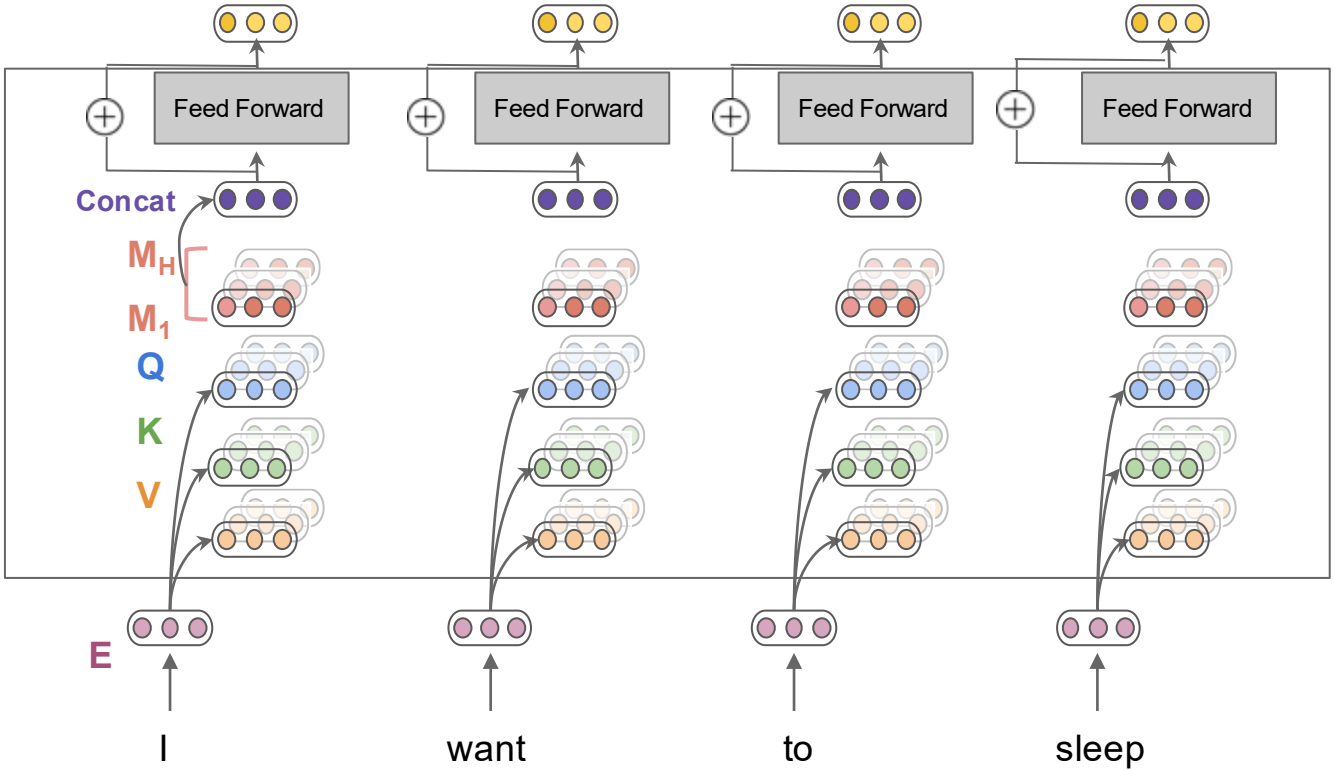
or with GELU (used in GPT-2+):

$$\text{FFN}(x) = \text{GELU}(xW_1)W_2$$

- **No interaction between positions**—applied identically to each token
- Expands to 4× dimension, then projects back

Multi-Headed Self Attention

Multi-Headed Self Attention + Feed Forward



Residuals + Layer Norm Stabilize Training

Residual connection:

$$\text{output} = \text{sublayer}(x) + x$$

Layer normalization:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sigma + \epsilon} + \beta$$

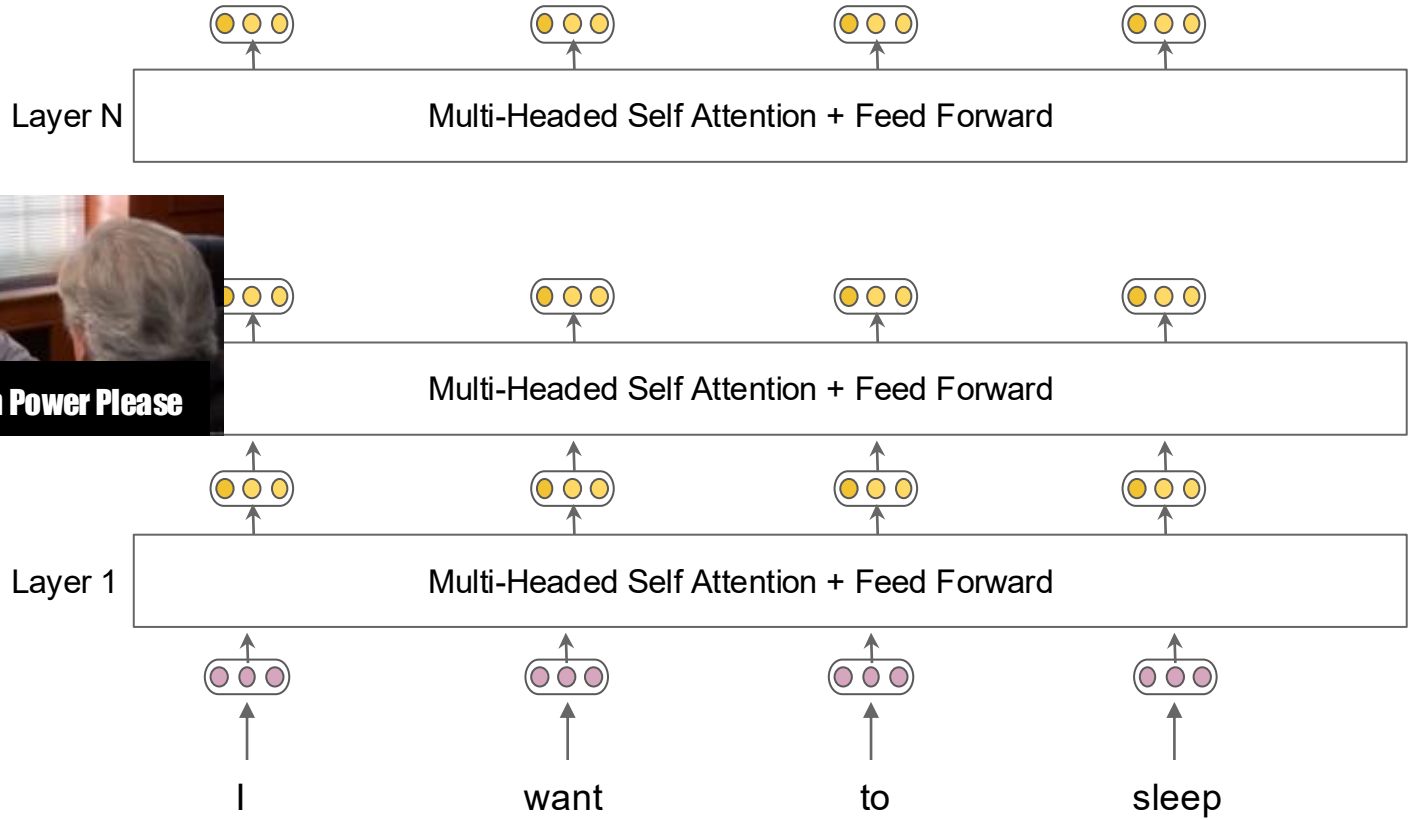
Why Residuals?

- Gradient flows directly through addition
- Enables training very deep networks
- Each layer learns a "delta"

Why LayerNorm?

- Normalizes activations per token
- Stabilizes training dynamics
- γ, β are learned parameters

Multi-Headed Self Attention



Final Transformer

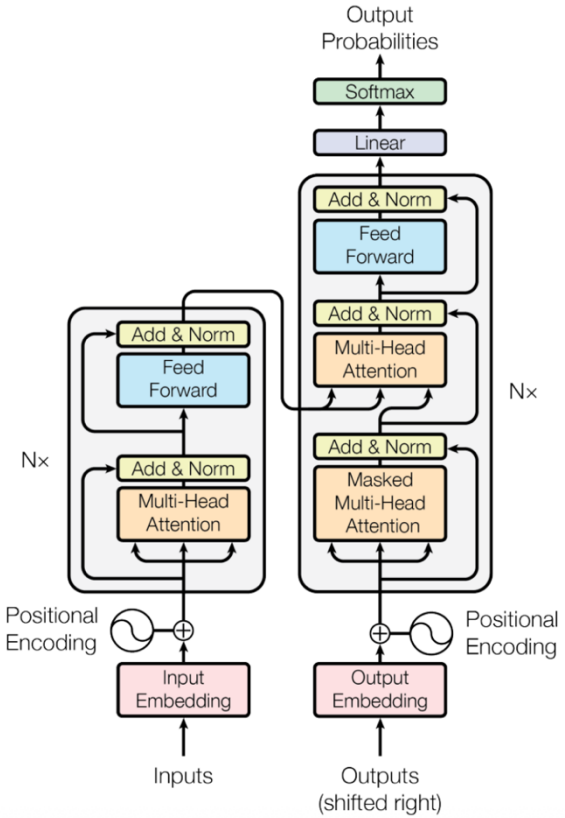
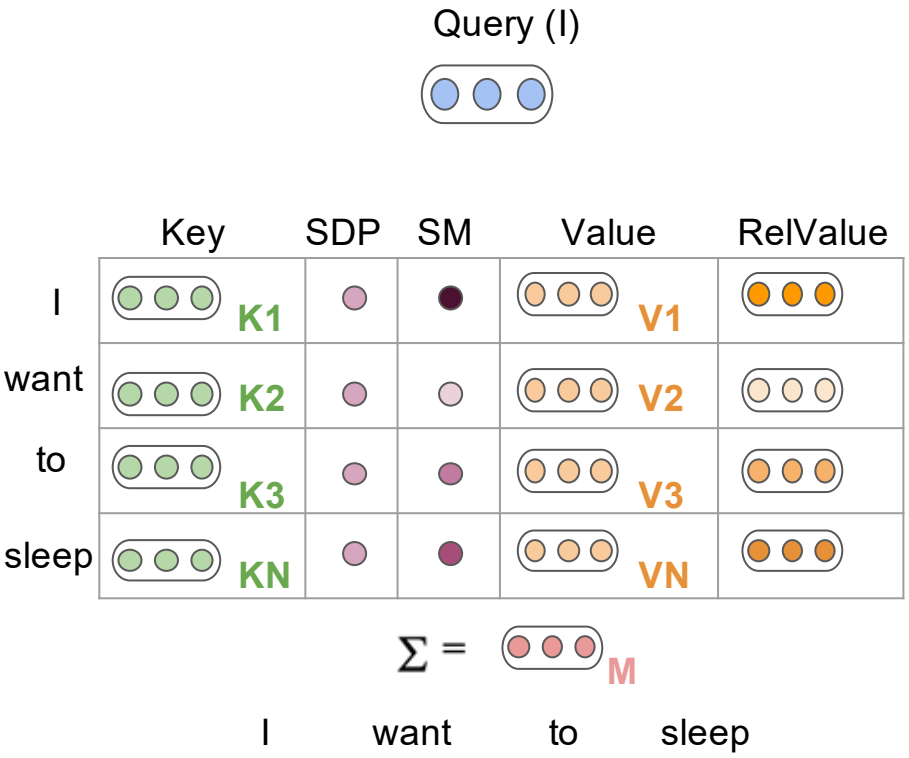
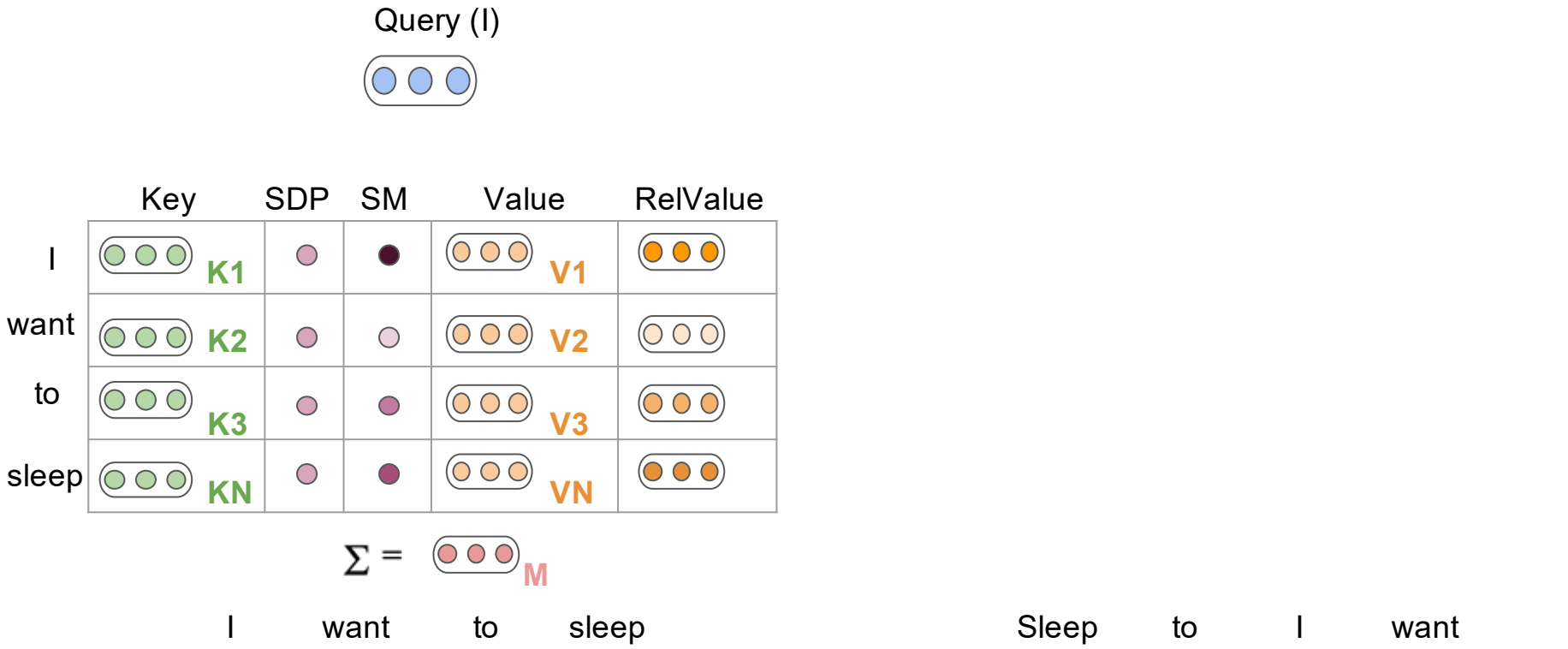


Figure 1: The Transformer - model architecture.

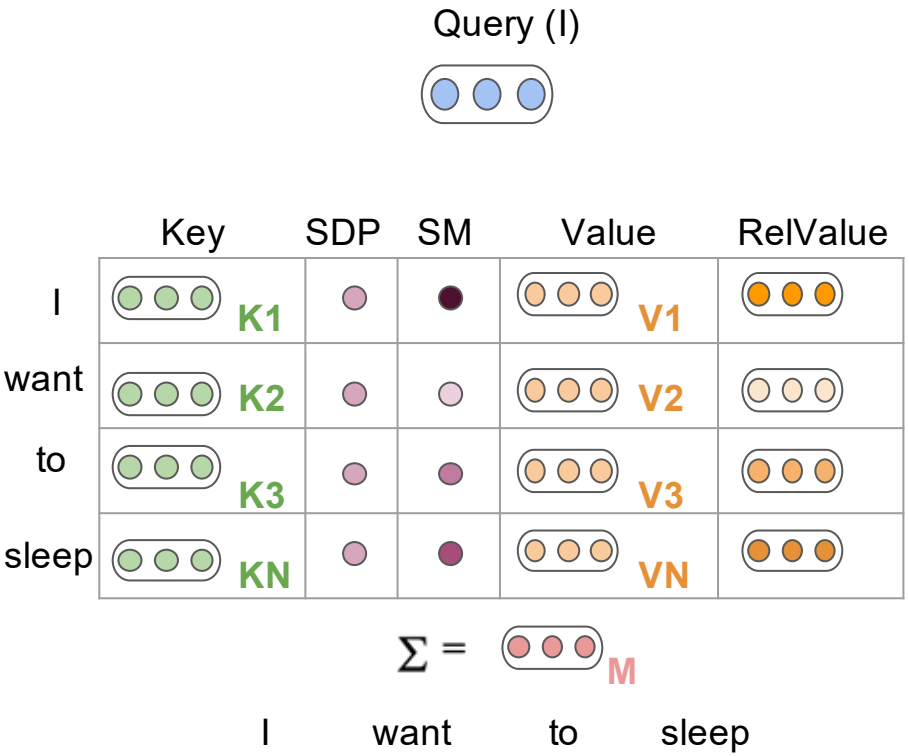
Revisiting Self Attention



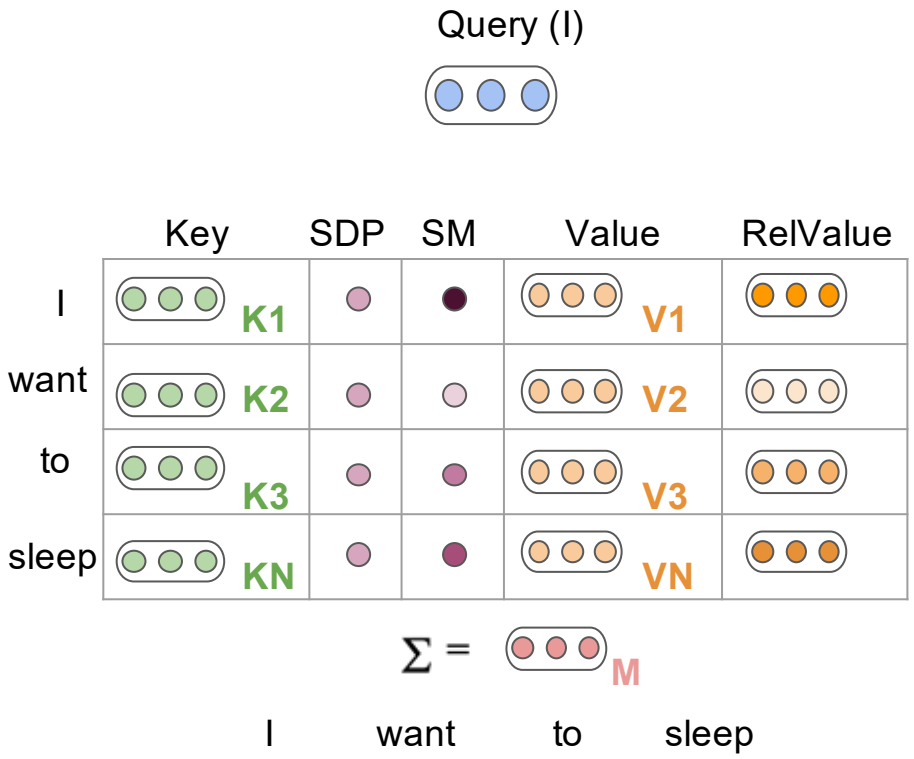
Revisiting Self Attention

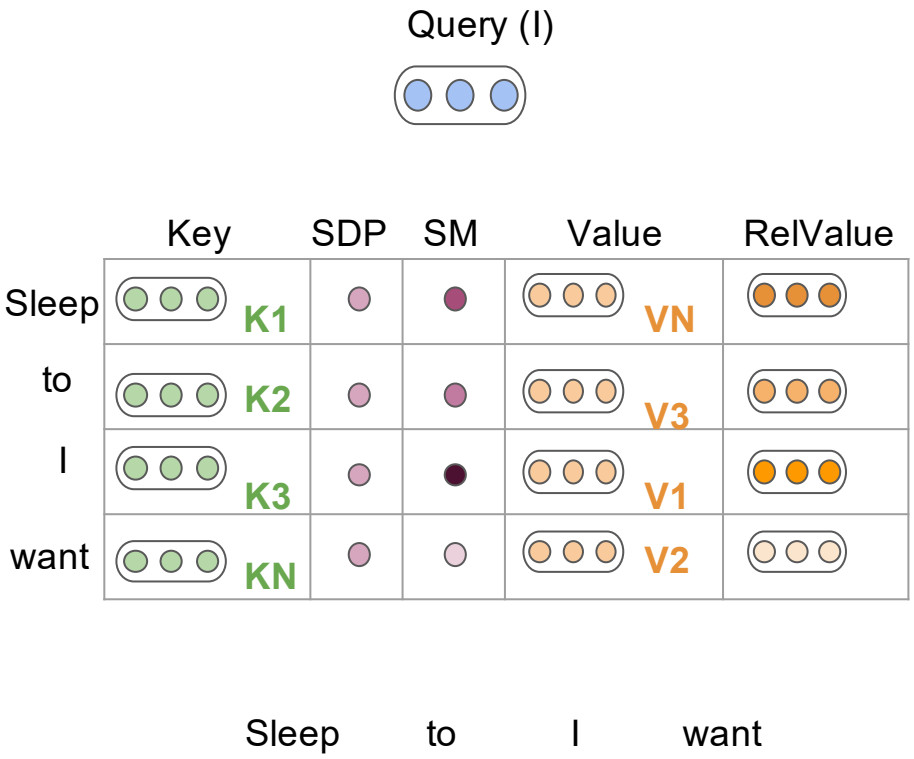


Revisiting Self Attention

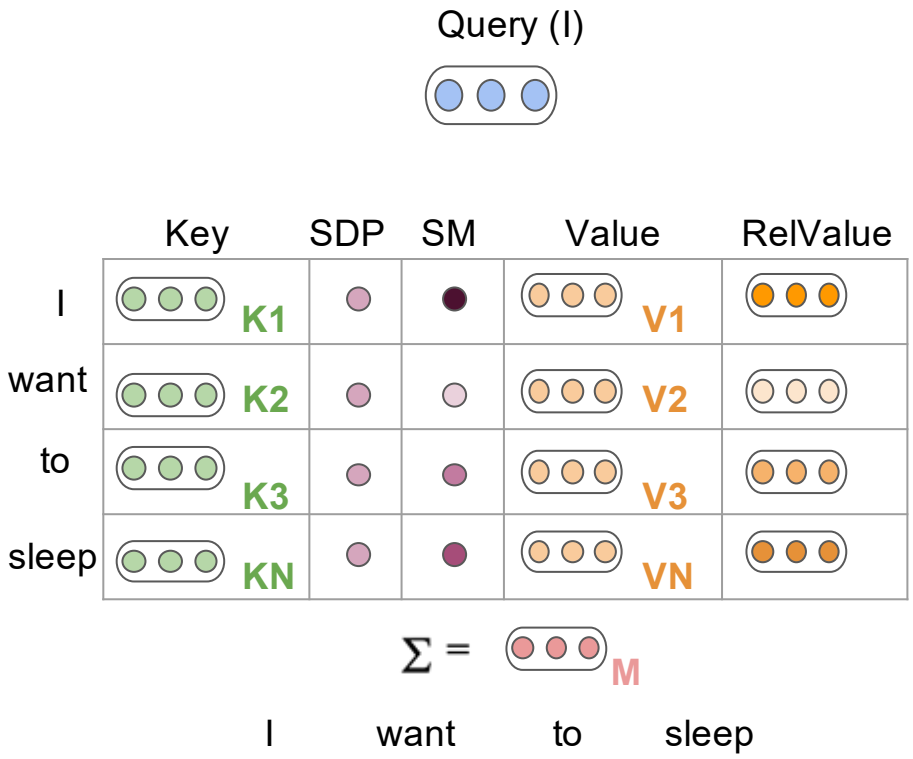


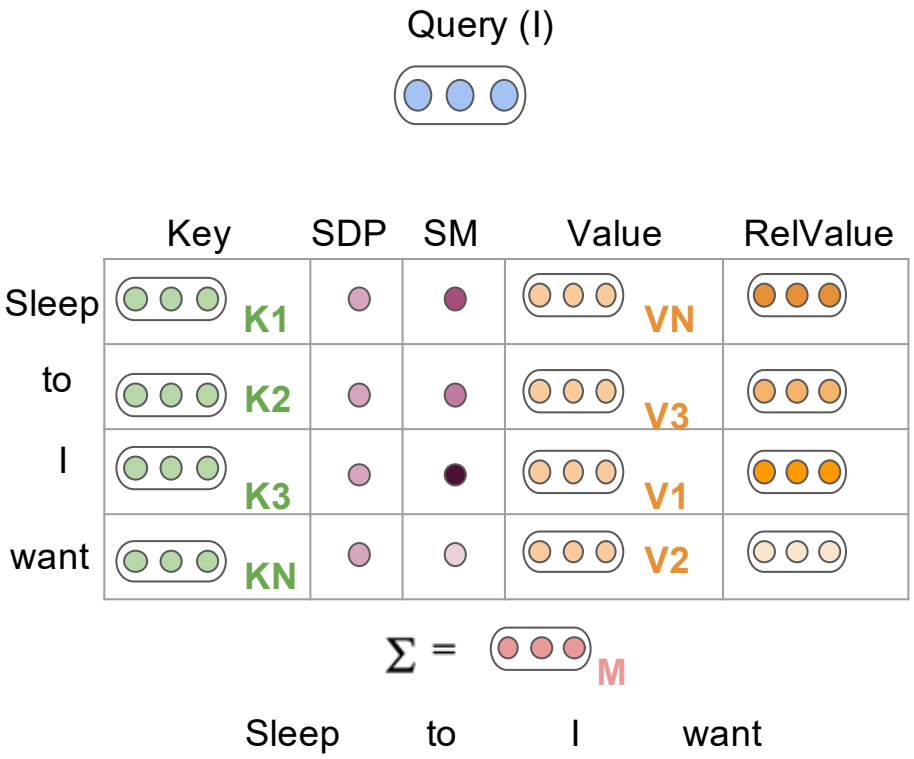
Revisiting Self Attention





Revisiting Self Attention



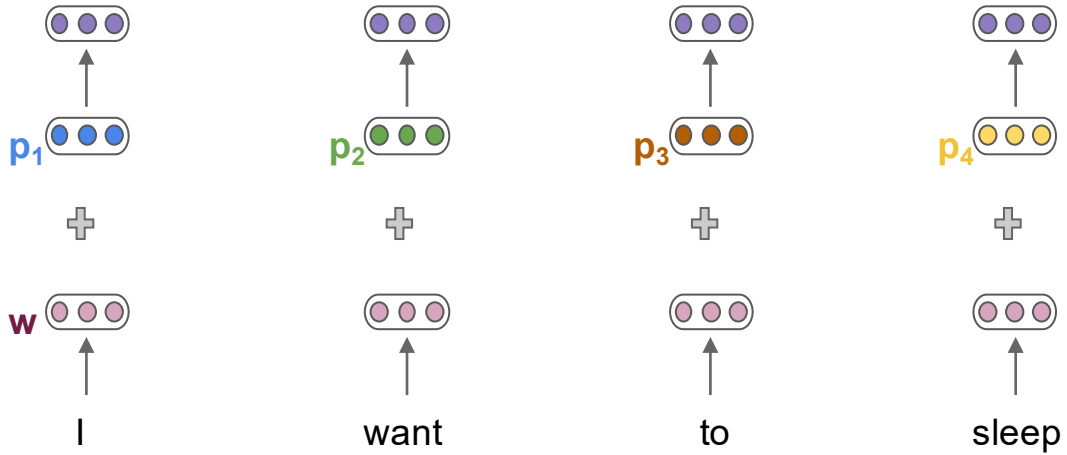


Revisiting Self Attention

Same representation for both sentences - But positions matter!

Positional Encoding

Position embeddings - each position number has an associated embedding



Positional Encoding

Sinusoidal (original transformer):

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Sinusoidal (fixed)

- No learned parameters
- Can extrapolate to longer sequences
- Used in original transformer

Learned

- Embedding lookup by position
- More flexible
- Used in GPT, BERT

Rotary (RoPE)

- Rotation in embedding space
- Better length generalization
- Used in LLaMA, GPT-NeoX

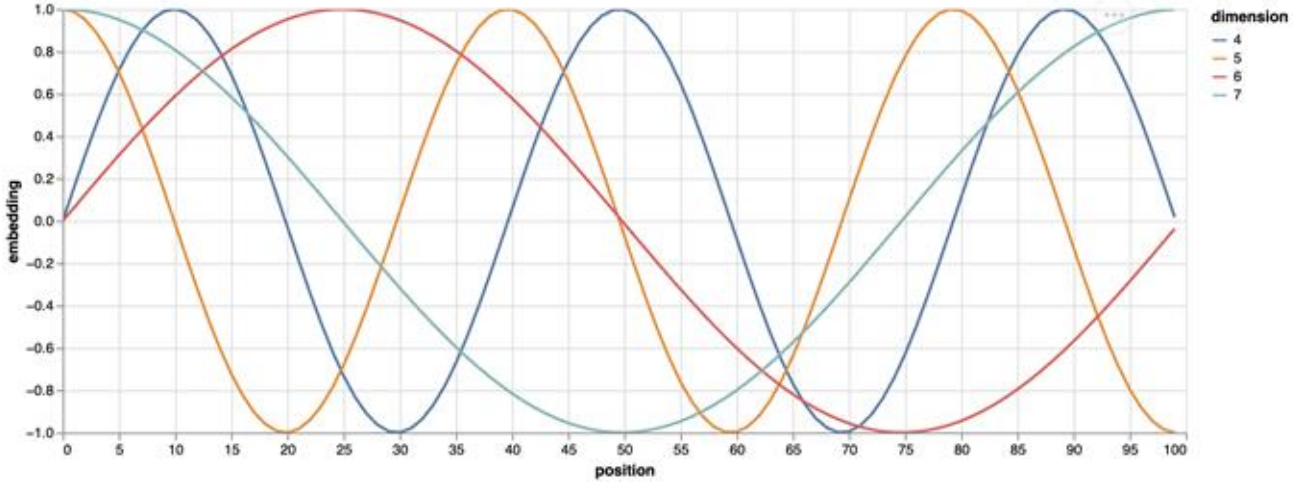
Learnable Positional Encoding

- Simplest type of positional encodings
- Initialize position encodings as random vectors for each position from 0 to MAX LENGTH
- Used in GPT-1, GPT-2, GPT-3
- CONS: Doesn't generalize to sequences of length greater than MAX LENGTH

Sinusoidal Positional Encoding

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

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

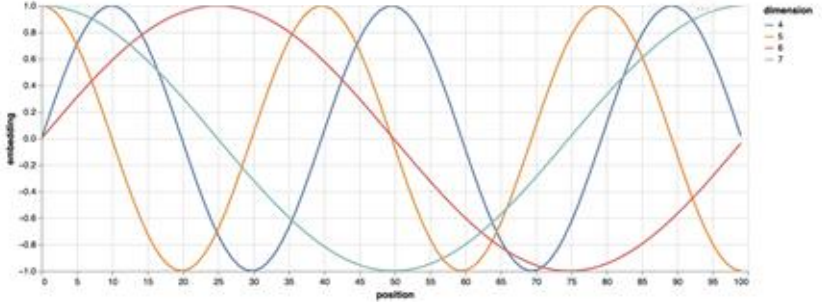


Sinusoidal Positional Encoding

- Introduced in the original Transformers paper and was used in models like BERT
- Major promise of this embeddings over using learnable encodings was that these might lead to length generalization beyond maximum length seen during training
- Alas, that's not the case and sinusoidal embeddings are usually as bad as learnable embeddings when it comes to length generalization

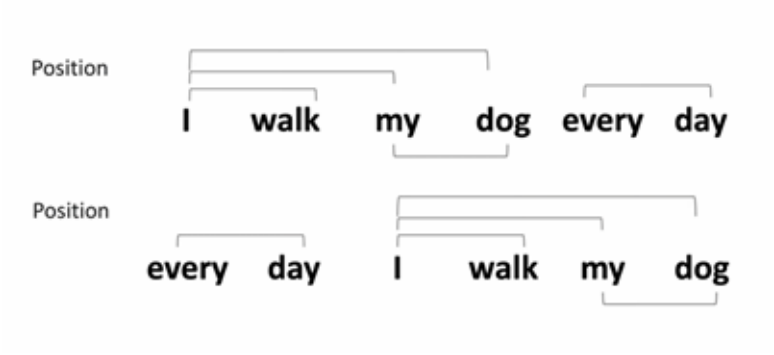
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

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



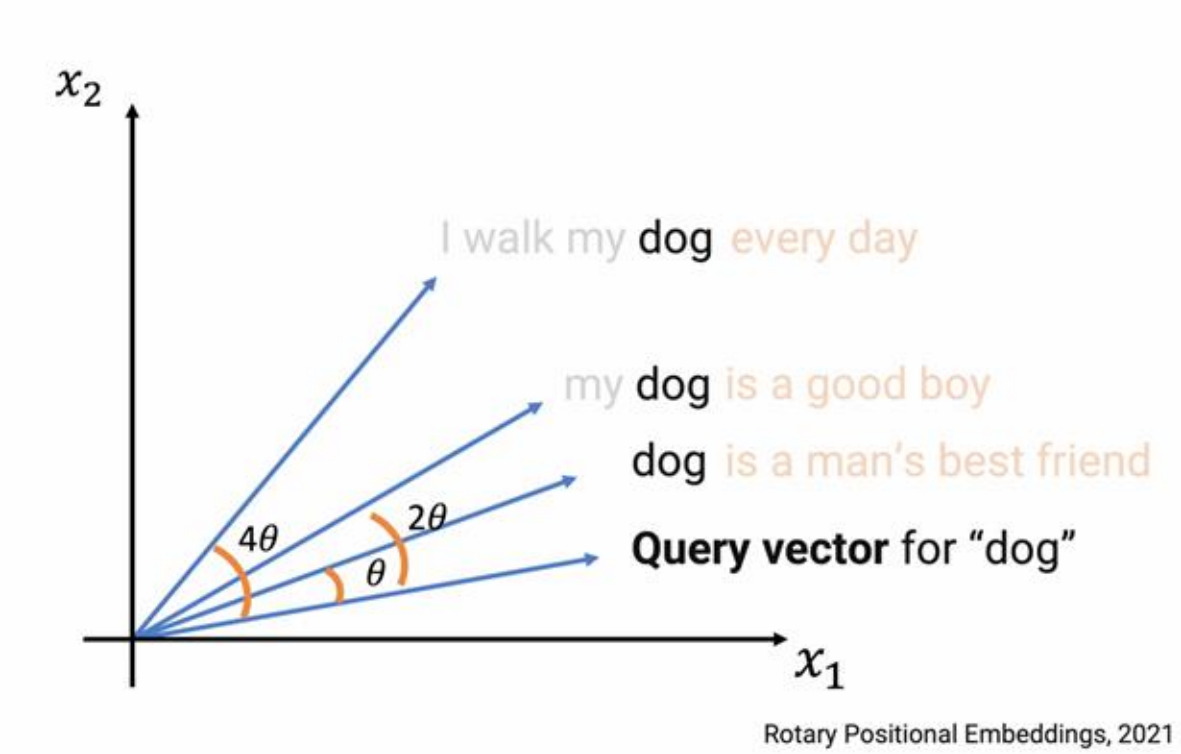
Relative Position Encodings

- Both learnable and sinusoidal position encodings were examples of Absolute Position Encodings
- i.e. positional information depends on the absolute position of the token
- In relative position encodings we provide the positional information in form of the relative difference between token position
- This information is added while taking the dot product between the query and key vectors



From Jia-Bin Huan's Youtube Video:
https://www.youtube.com/watch?v=SMBkImDWOyQ&t=683s

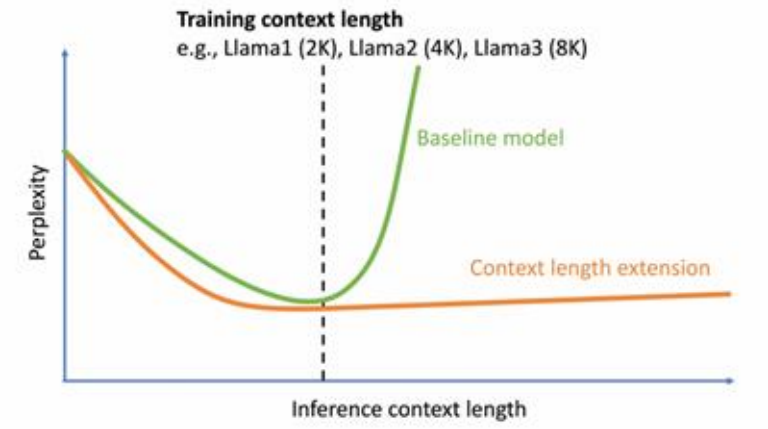
Rotary Positonal Encodings (RoPE)



From Jia-Bin Huan's Youtube Video:
https://www.youtube.com/watch?v=SMBkImDWOyQ&t=683s

Rotary Positonal Encodings (RoPE)

- Generally show faster convergence than Absolute Position Encoding methods
- The length generalization is still not guaranteed!
- Unseen relative distances i.e. the ones that exceed the maximum sequence length during training remain an issue
- Neural-Tangent-Kernel (NTK) RoPE is an extension to RoPE that enables generalization to long sequences



From Jia-Bin Huan's Youtube Video:
https://www.youtube.com/watch?v=SMBkImDWOyQ&t=683s

Different Archtectures

Encoder-Only (BERT)

Bidirectional attention

- See all tokens at once
- Good for classification, NER
- Cannot generate text

Decoder-Only (GPT)

Causal attention

- See only past tokens
- Good for generation, chat
- Dominant for modern LLMs

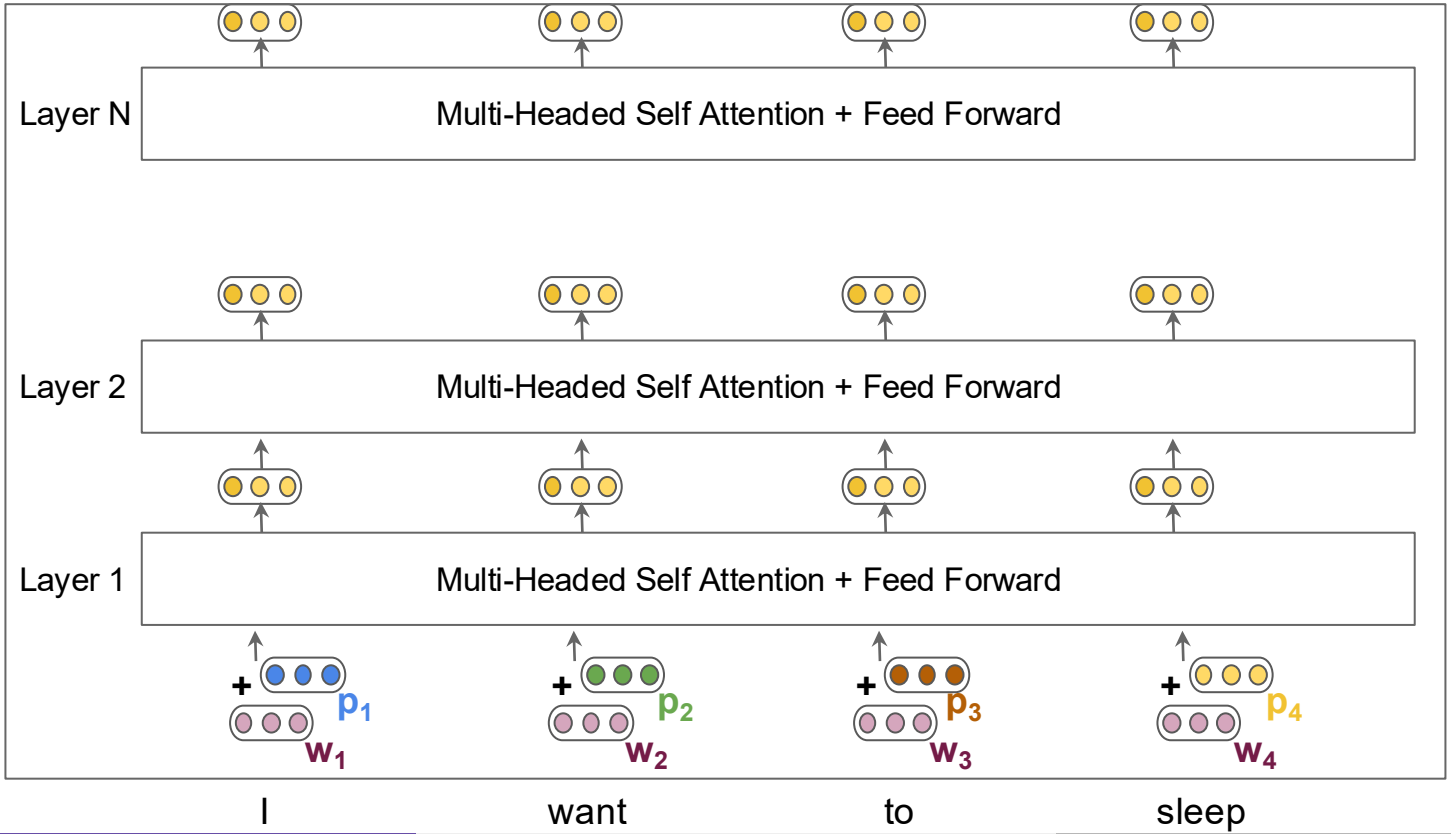
Encoder-Decoder (T5)

Both + cross-attention

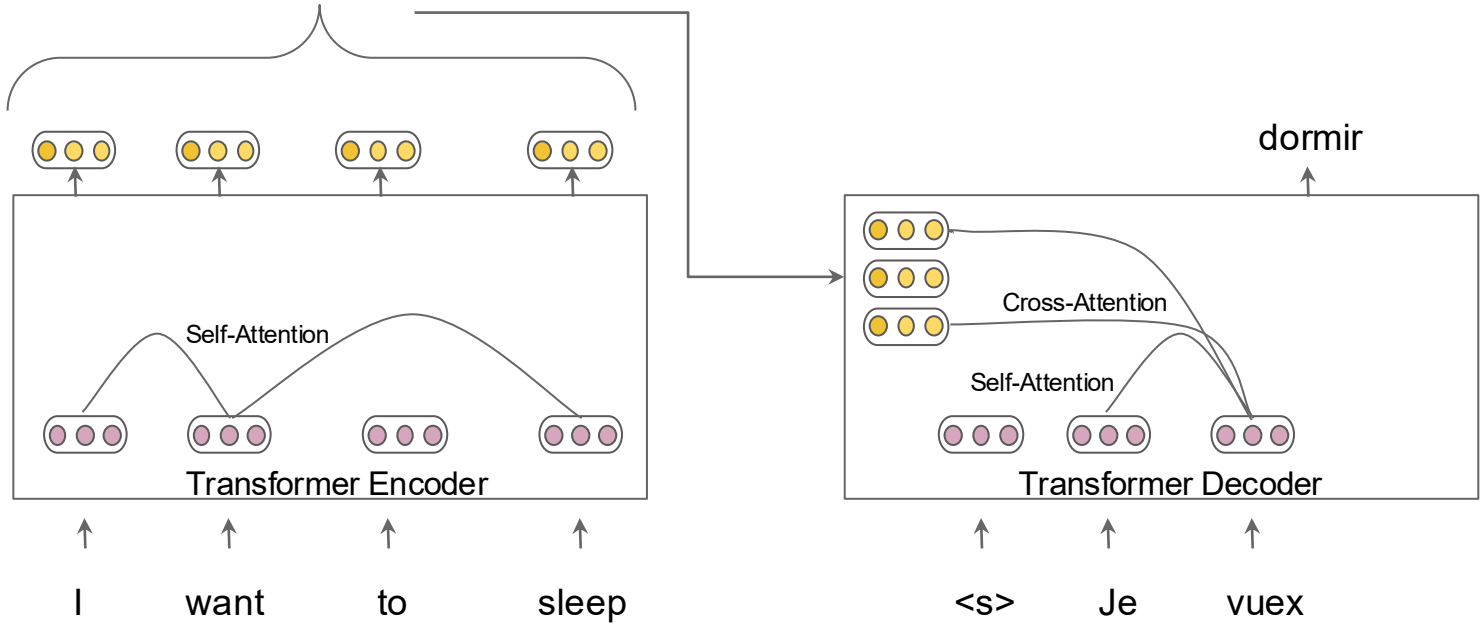
- Encoder sees all input
- Decoder generates output
- Good for translation, summarization

Transformer Encoder

N-Layer Transformer Encoder



Transformer Encoder - Decoder

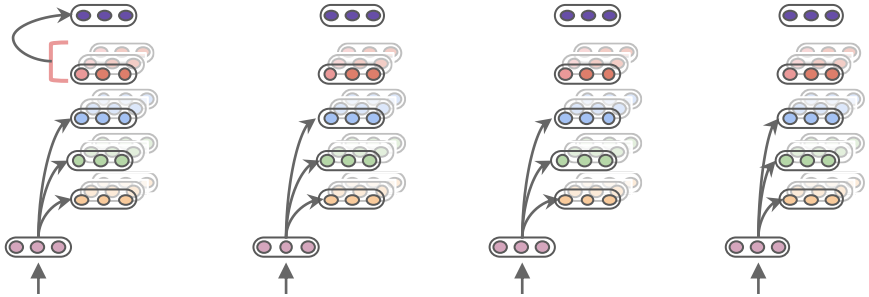


What's so great about Transformers?

- Parallelizable computation
  - Entire sequence, All queries, all attention heads computed in parallel
  - Benefits from fast matrix multiplication on GPUs
- Rich expressive power
  - Every token connected to every other token
  - Can form long range dependencies
- Depth not proportional to seq length
  - Reduces exploding/vanishing gradient problem
  - Converges faster

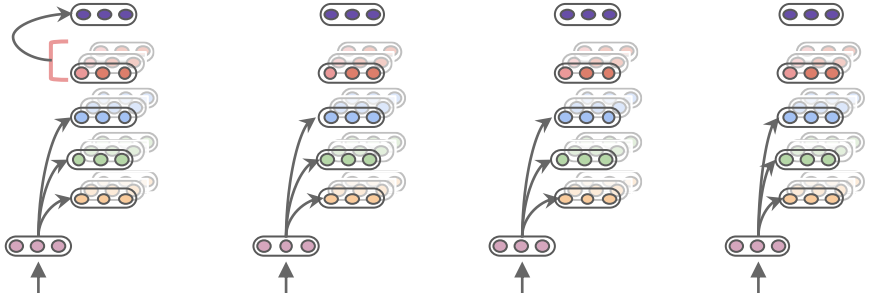
What's so great about Transformers?

- Parallelizable computation - Entire sequence can be processed in parallel



What's so great about Transformers?

- Parallelizable computation - Entire sequence can be processed in parallel



- Rich expressive power - long range dependencies