Natural Language Processing

Self Attention and Transformers

Kabir Ahuja
kahuja@cs.washington.edu

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

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Readings

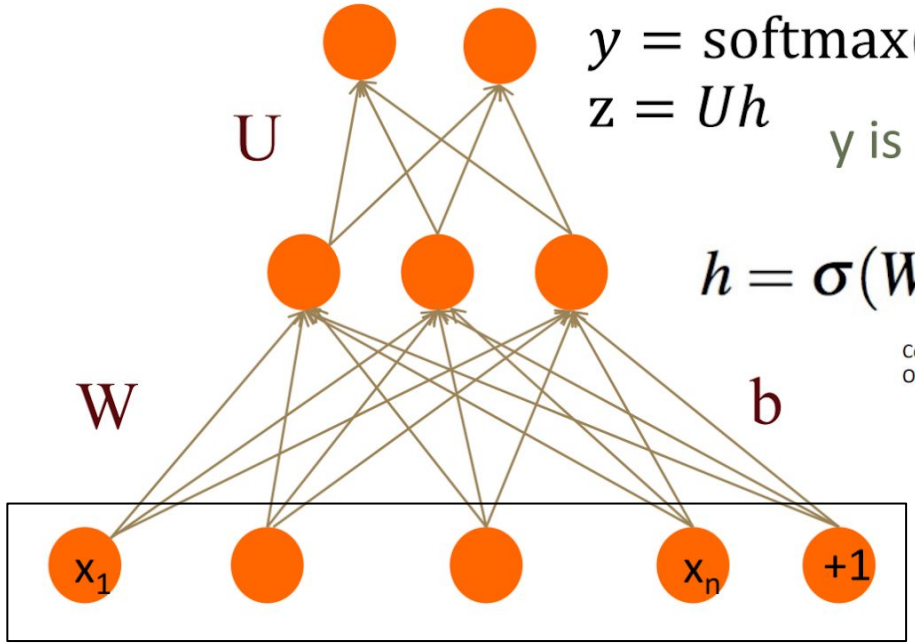
- Attention Is All You Need
- The Illustrated Transformer
- The Annotated Transformer
- Language Modeling with Transformers and PyTorch
- RoFormer: Enhanced Transformer with Rotary Position Embedding
- How Rotary Position Embedding Supercharges Modern LLMs [RoPE]

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

$h = \sigma(Wx + b)$

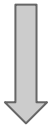
Could be ReLU
Or tanh

Fixed-Size Inputs

Fixed size representations for Natural Language

How to turn this

Learning NLP is fun!



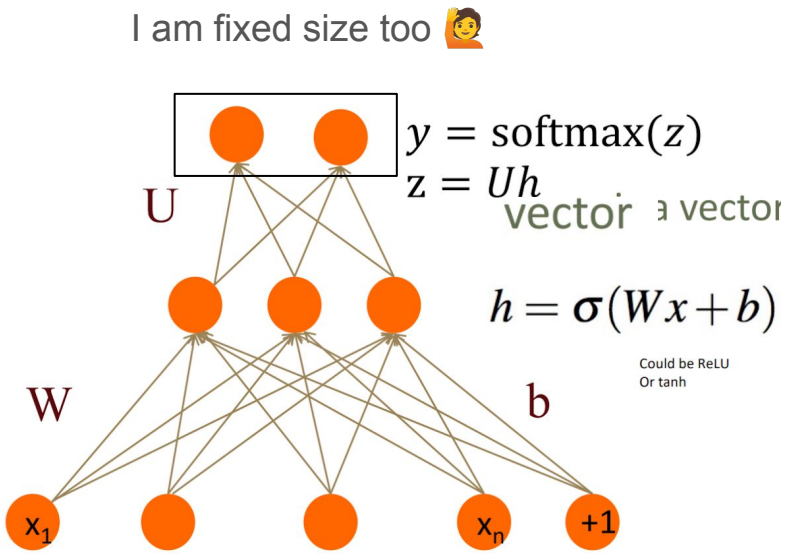
Into this

[x_1, x_2, …, x_d]

(fixed length d-dimensional vector)

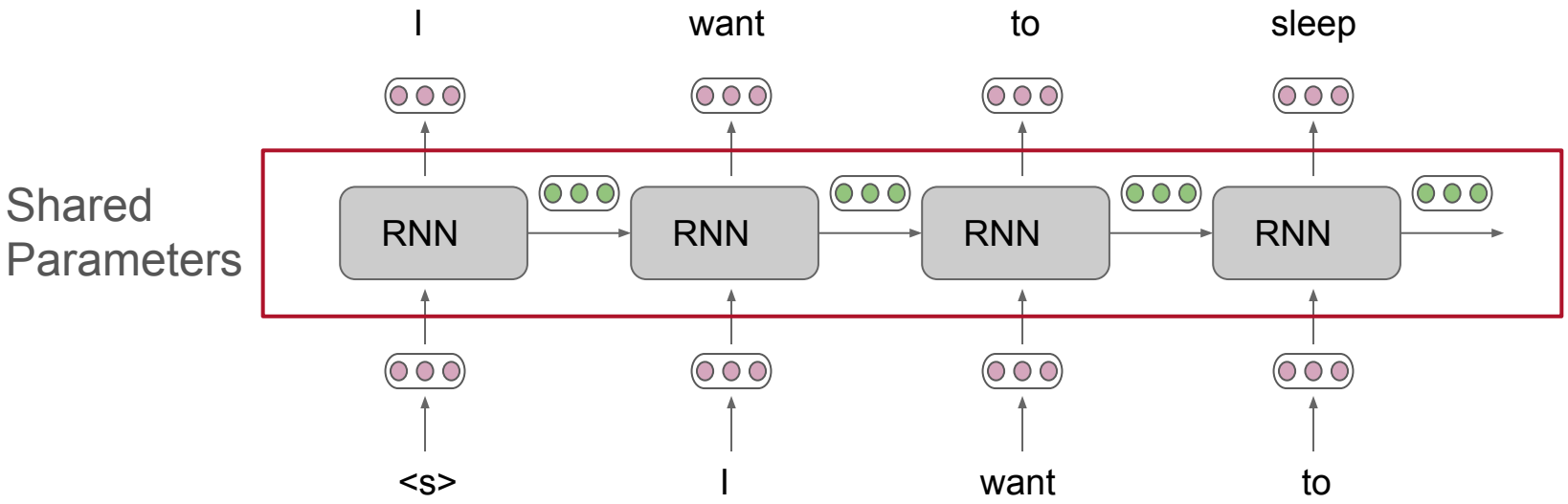
Problems with Fixed-Size Representations

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 :)

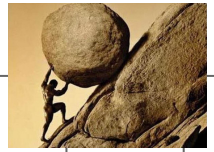


Sequence Models To Rescue

Recurrent Neural Networks - RNNs



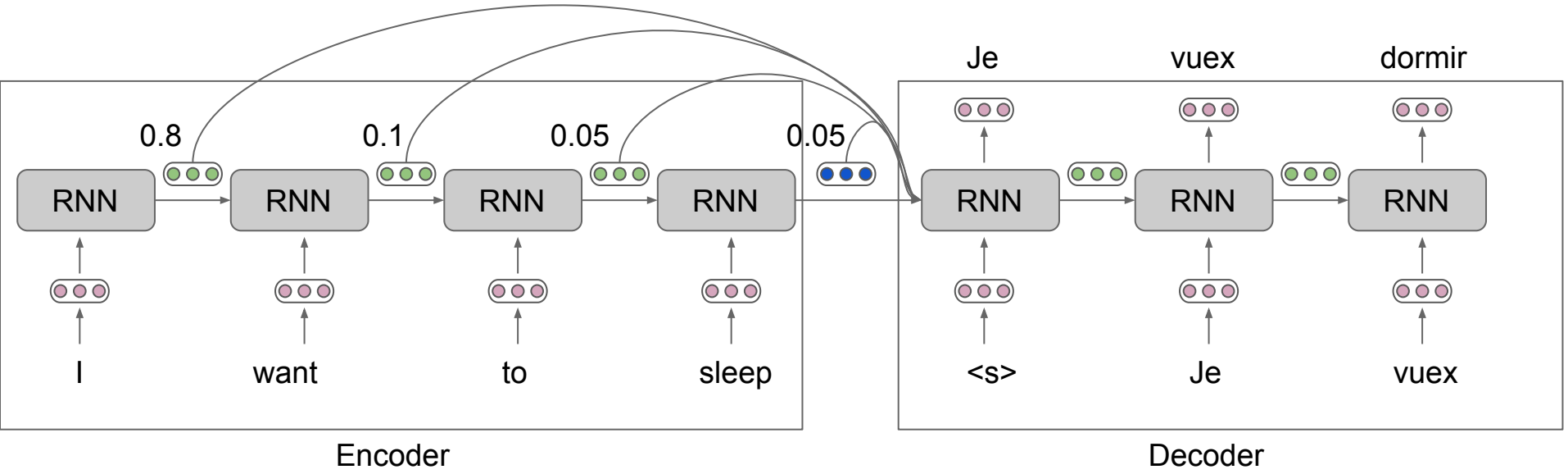
Encoder-Decoder Models



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

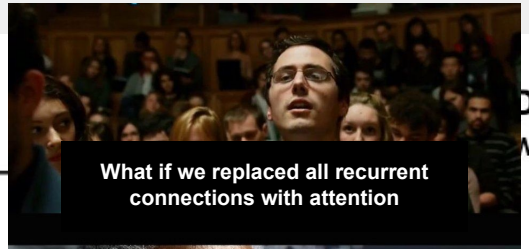


Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al, 2015

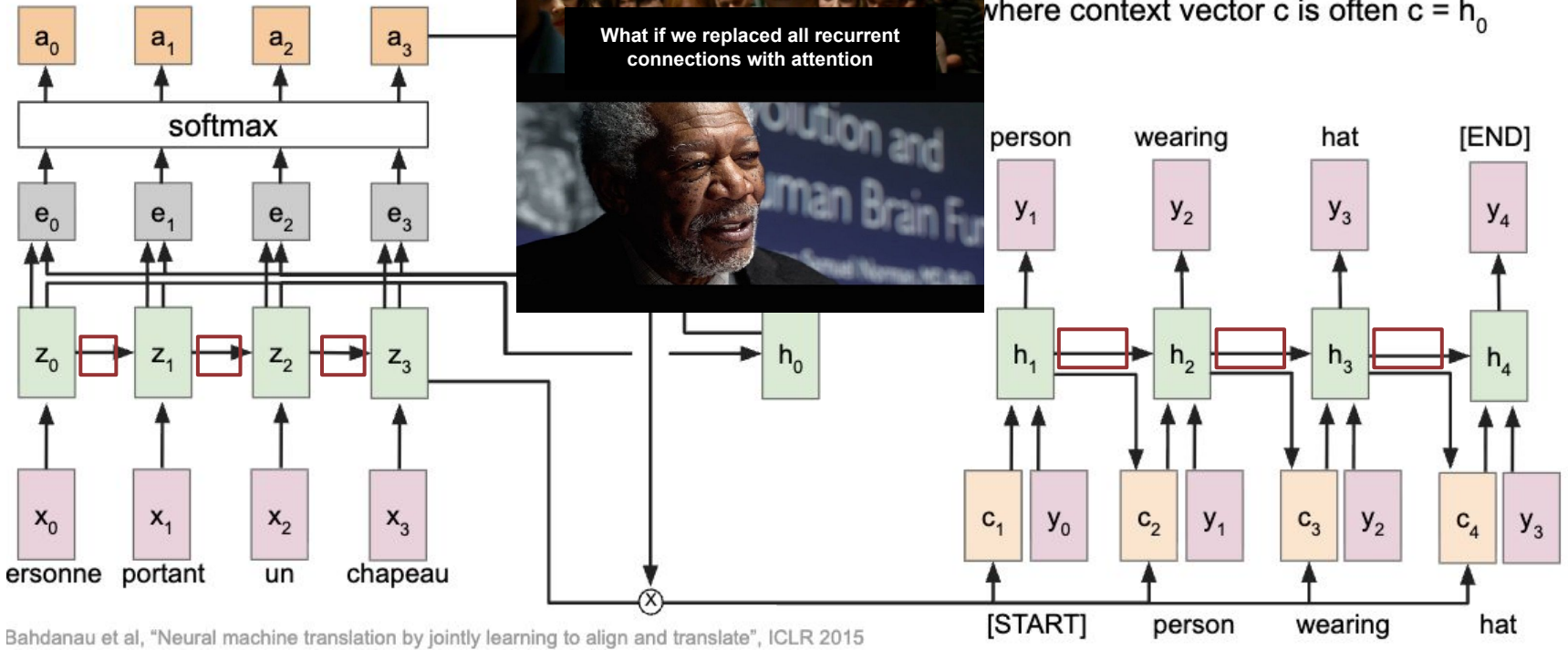
What's wrong with RNNs?

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

Encoder-Decoder Models With Attention



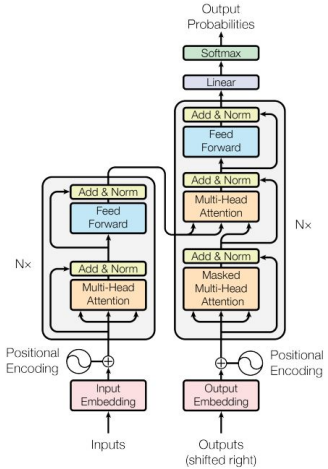
Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



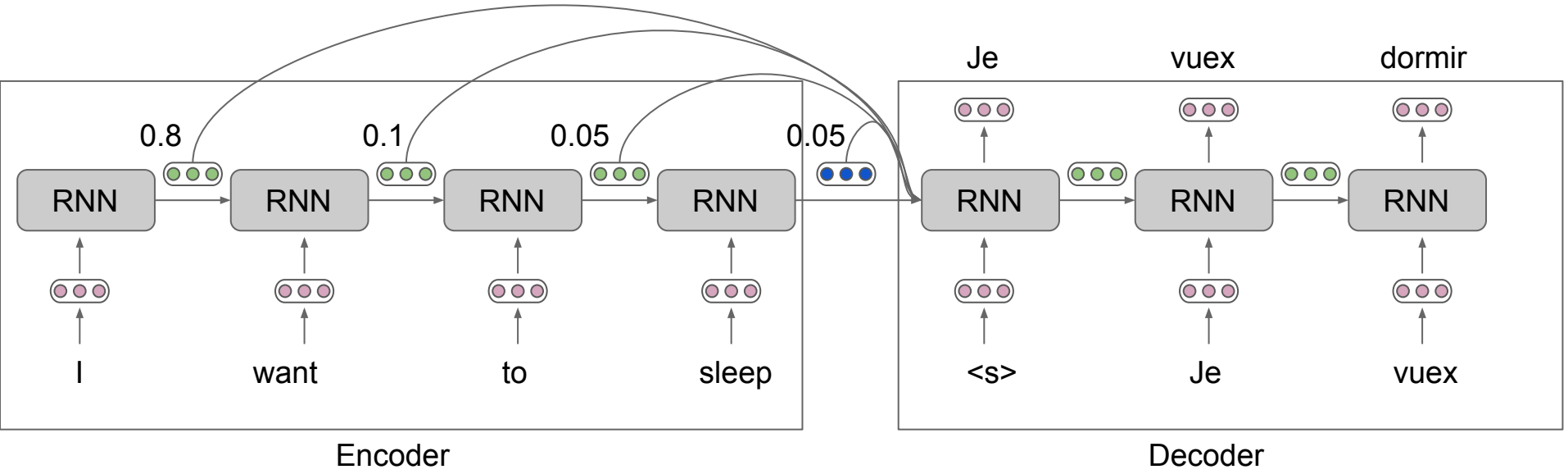
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

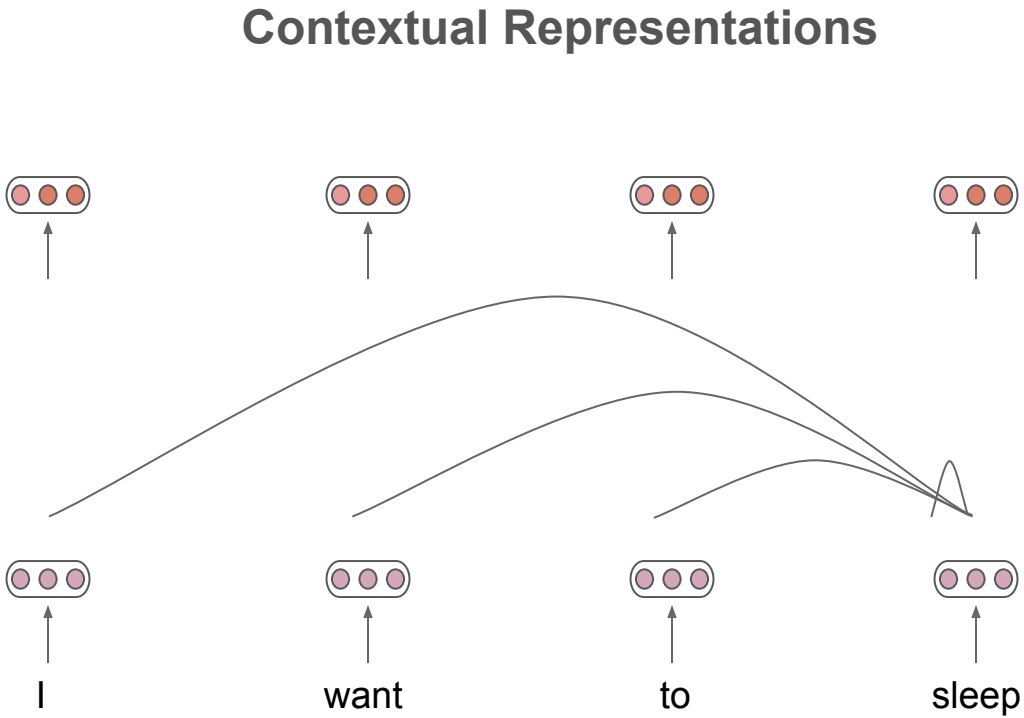


Last look at RNNs

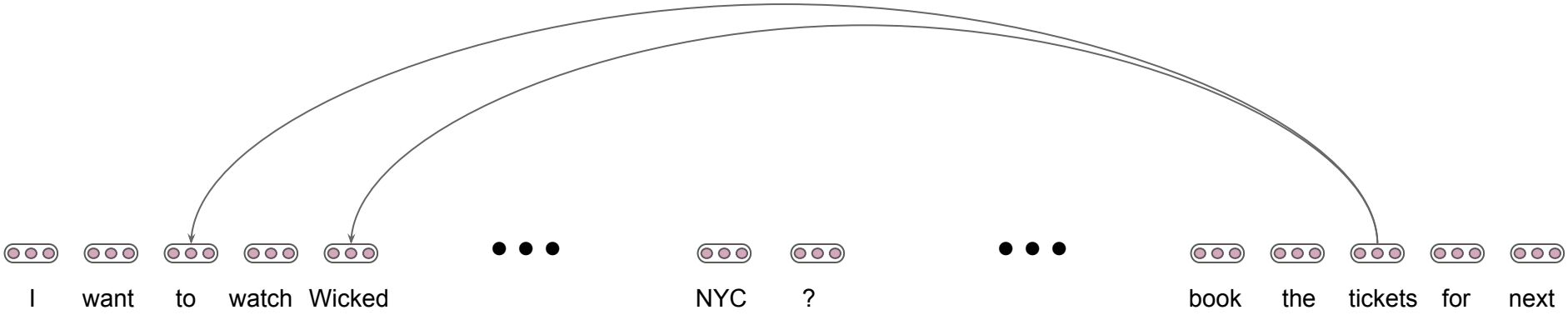


Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al, 2015

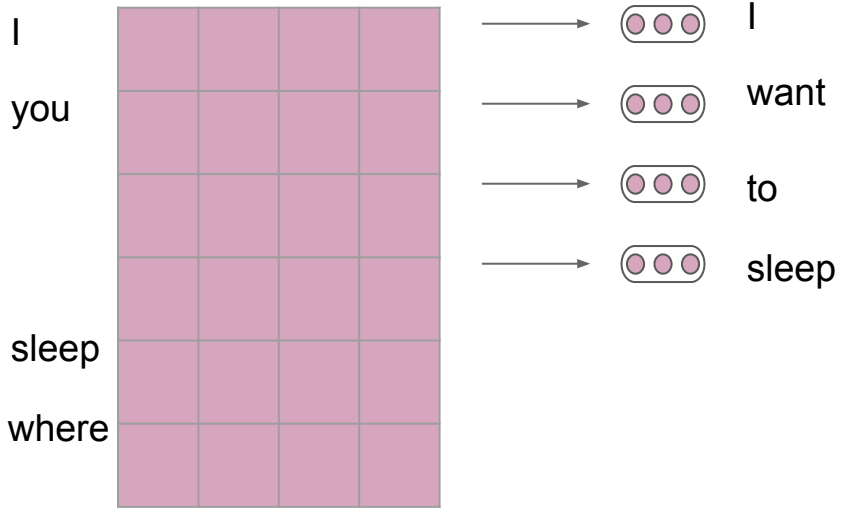
Self Attention - No more recurrence

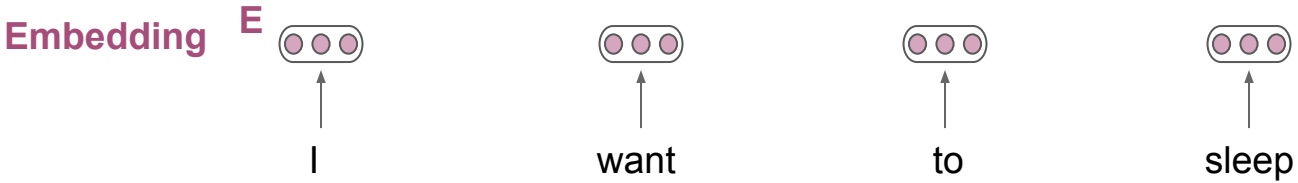


Self Attention for long sequences

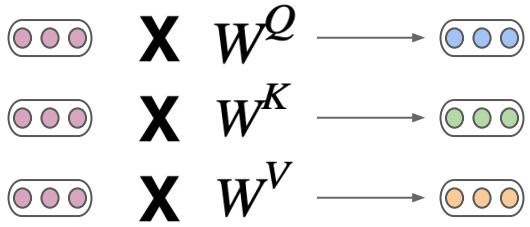


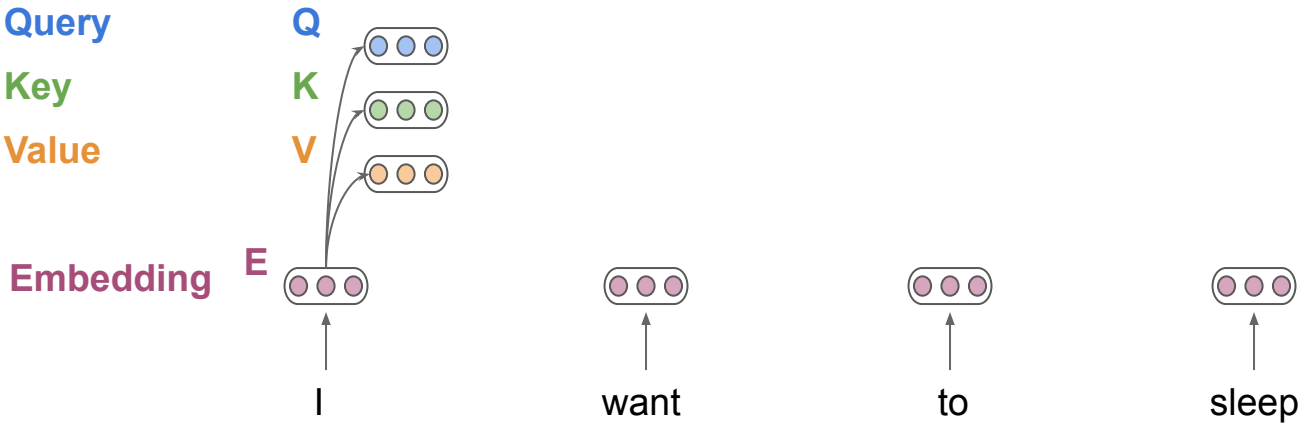
Self Attention - Word Embedding



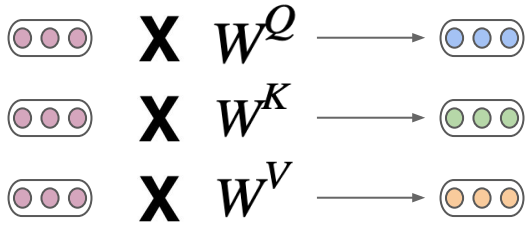


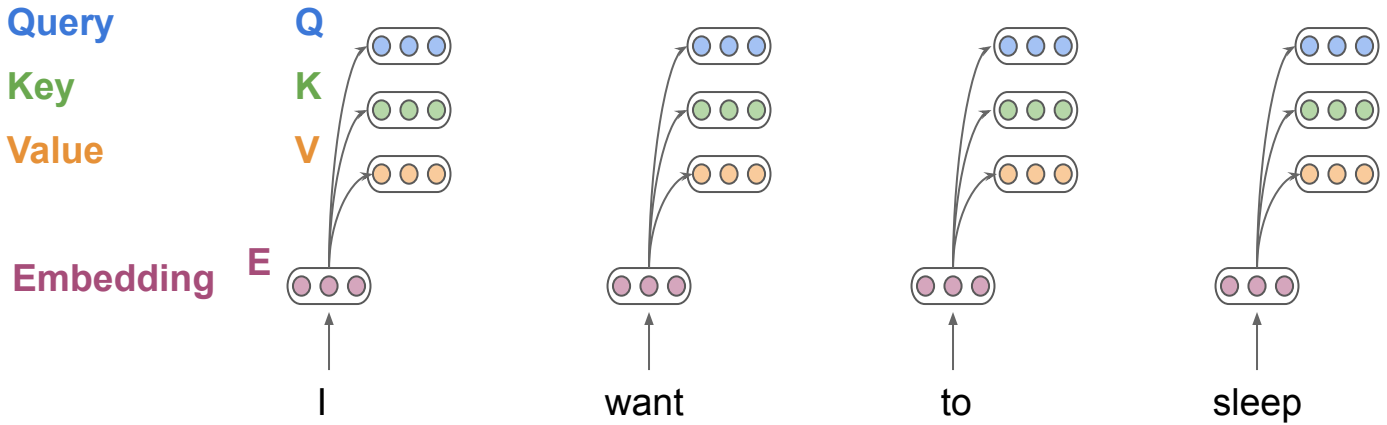
Self Attention - Projection Layer



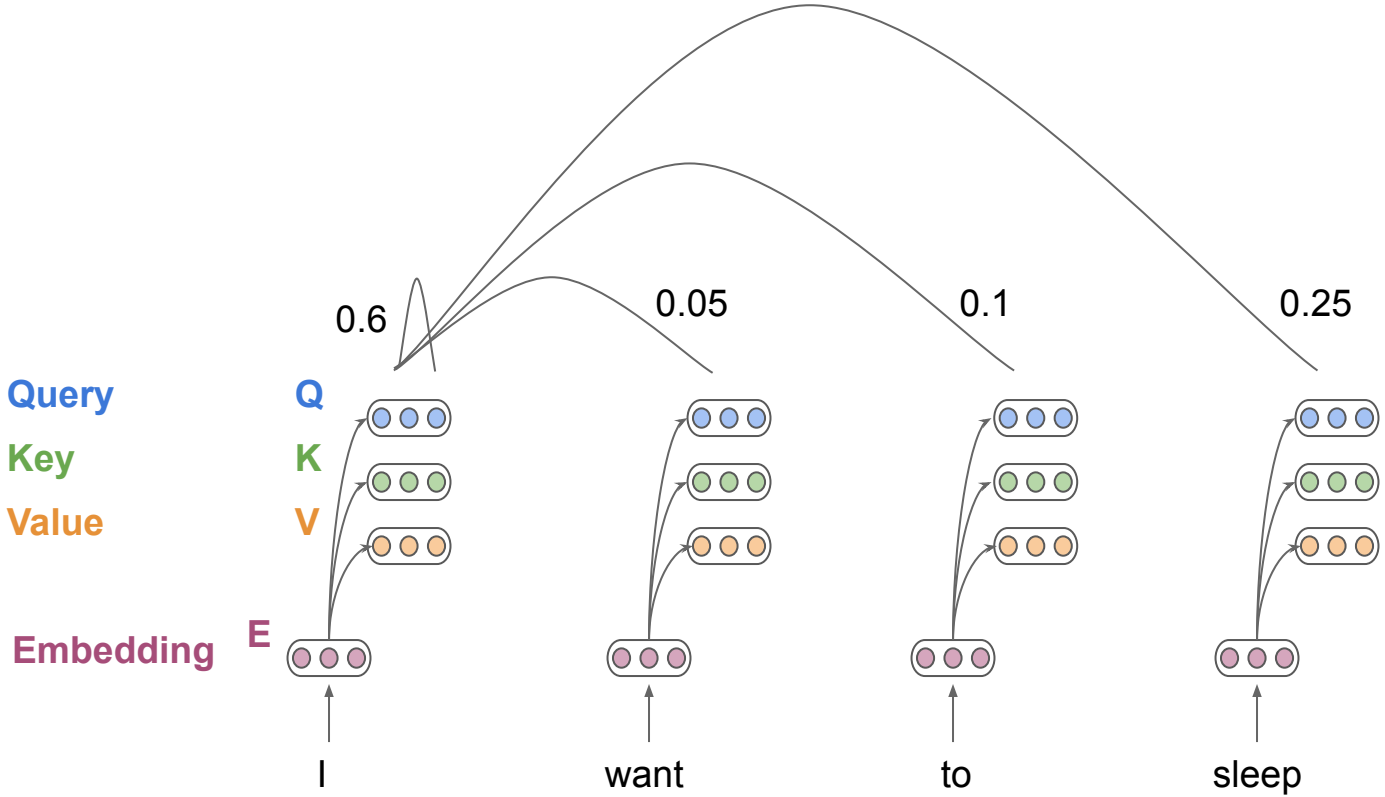


Self Attention - Projection Layer

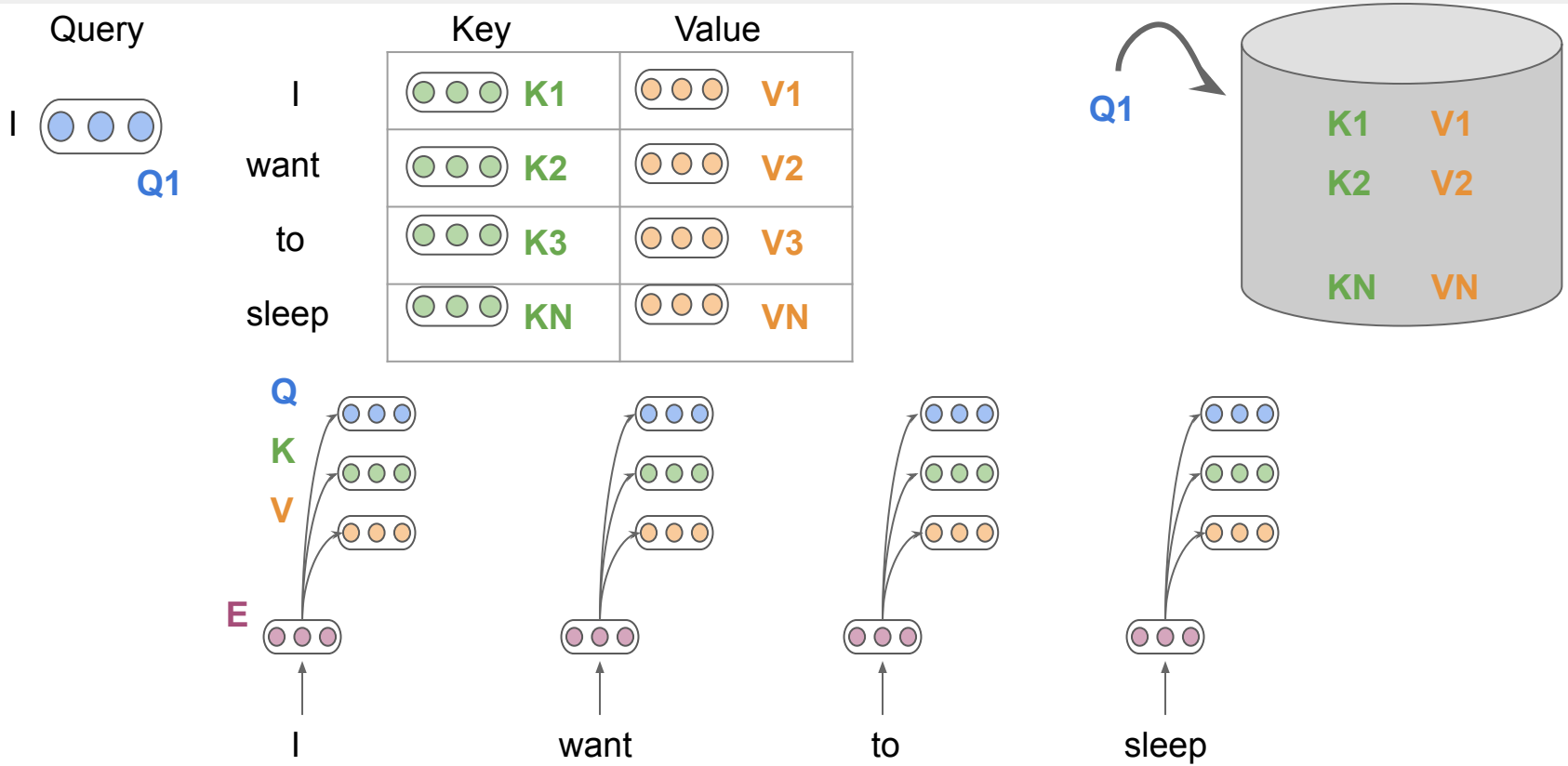




Self Attention - Attention Scores

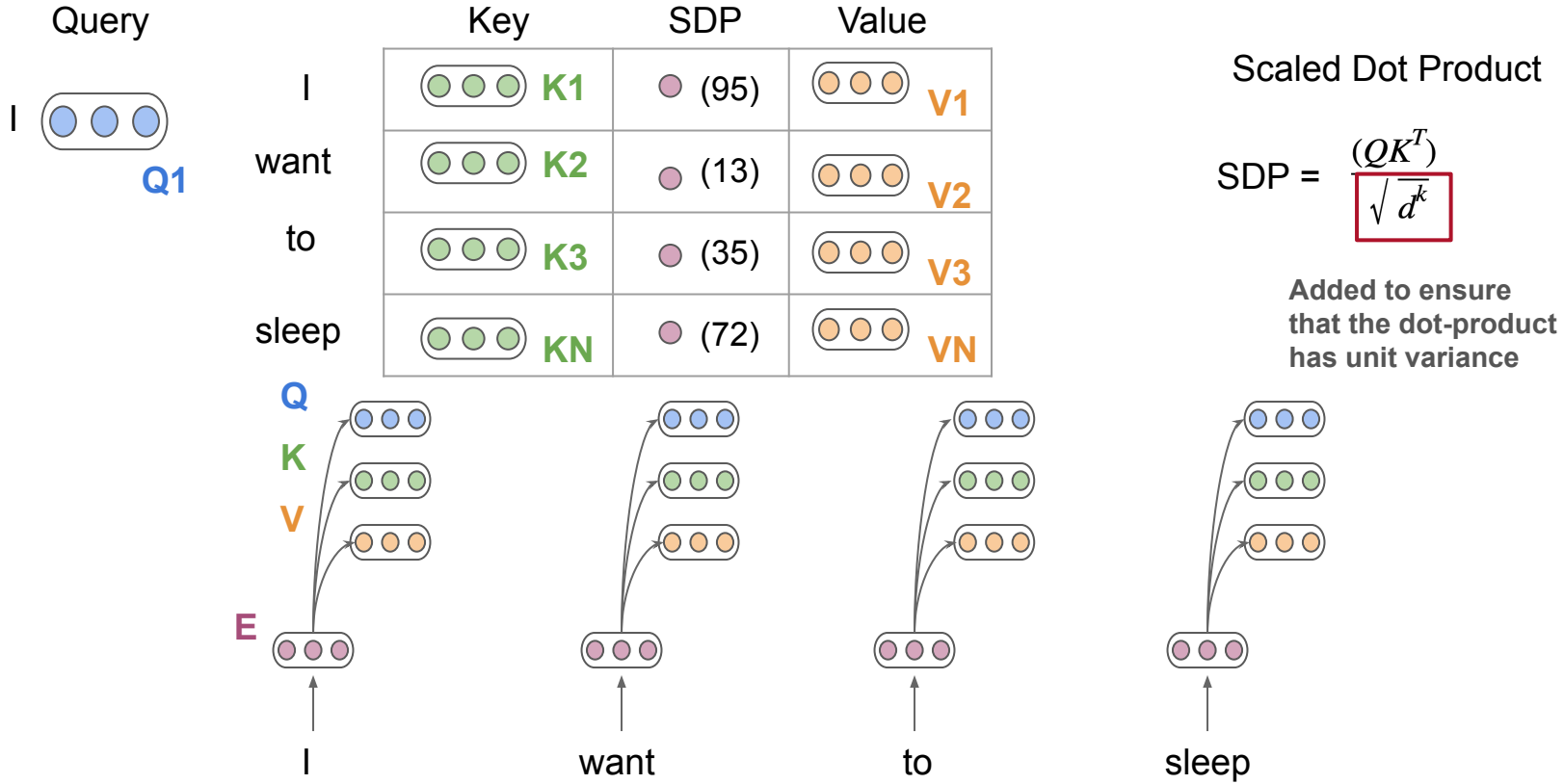


Self Attention



Questions?

Self Attention - Scaled Dot Product

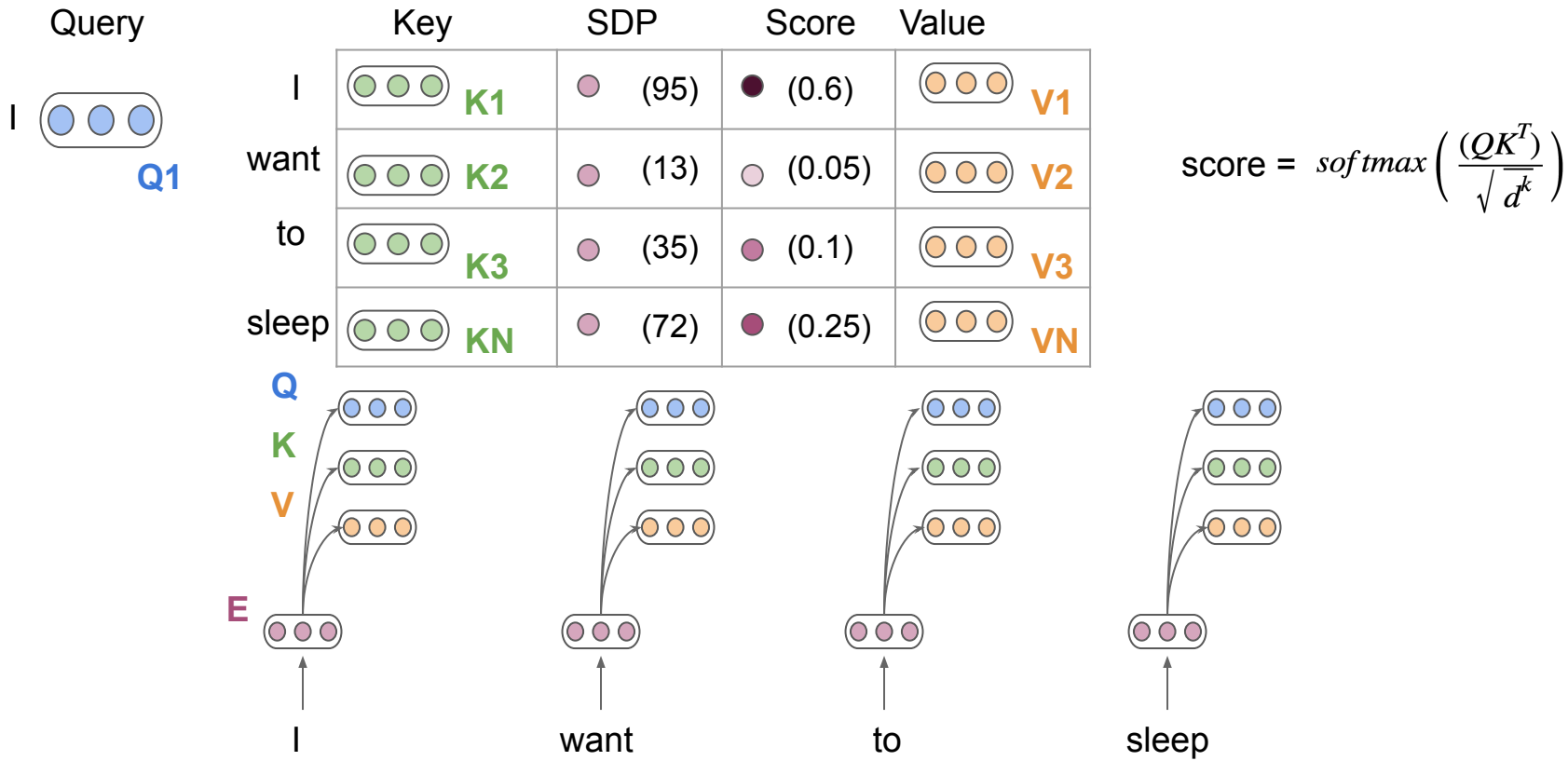


Scaled Dot Product

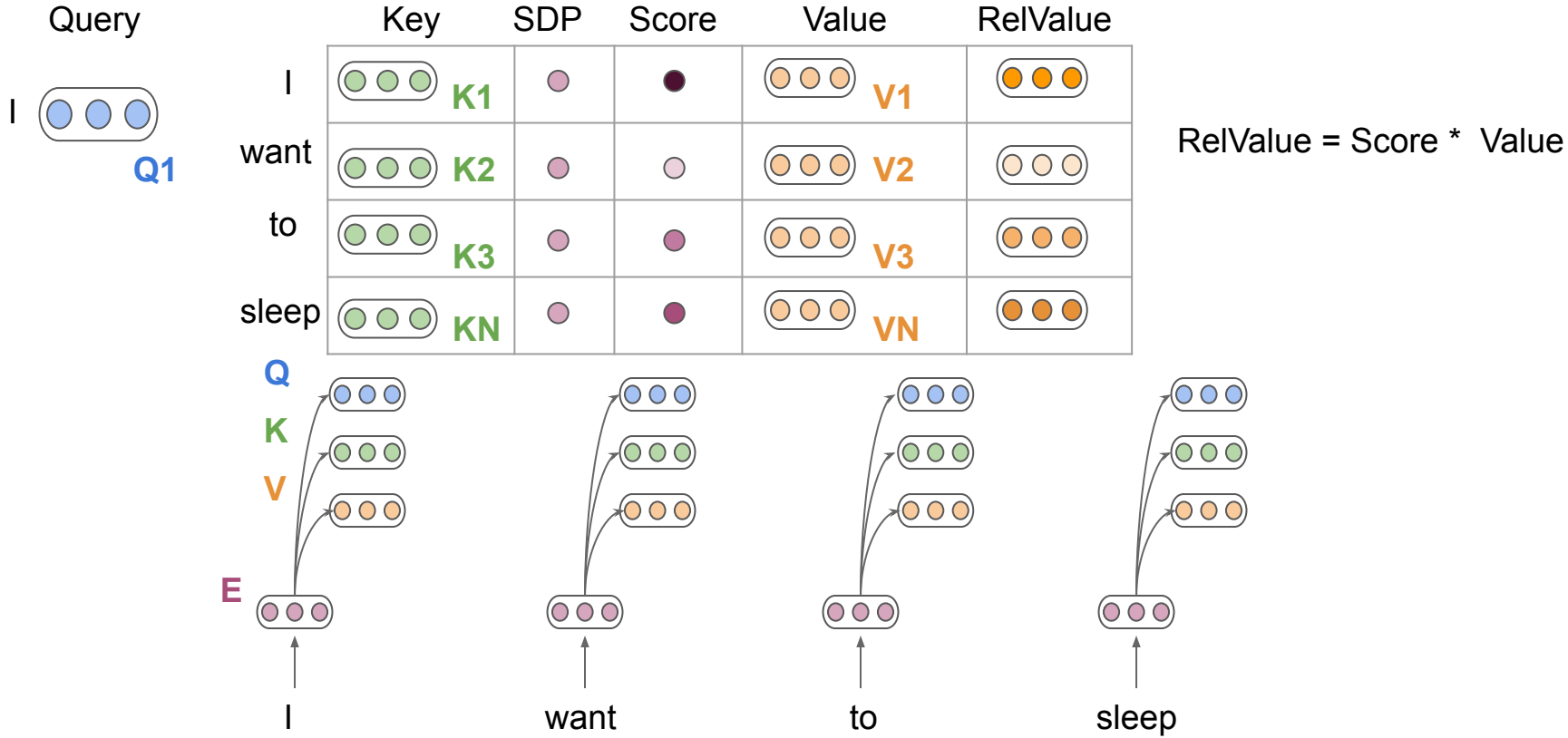
$$\text{SDP} = \frac{(QK^T)}{\sqrt{d^k}}$$

Added to ensure that the dot-product has unit variance

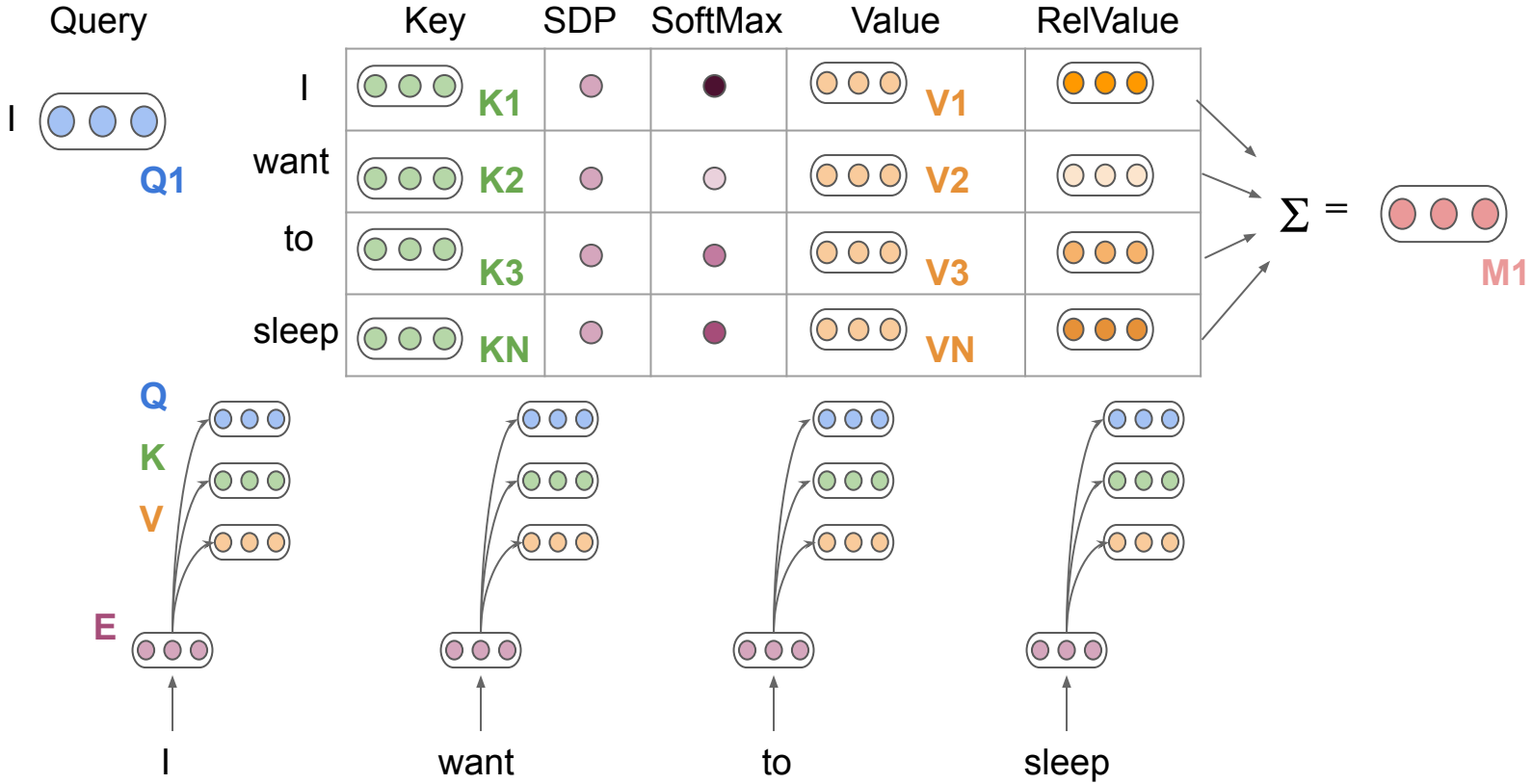
Self Attention - SoftMax



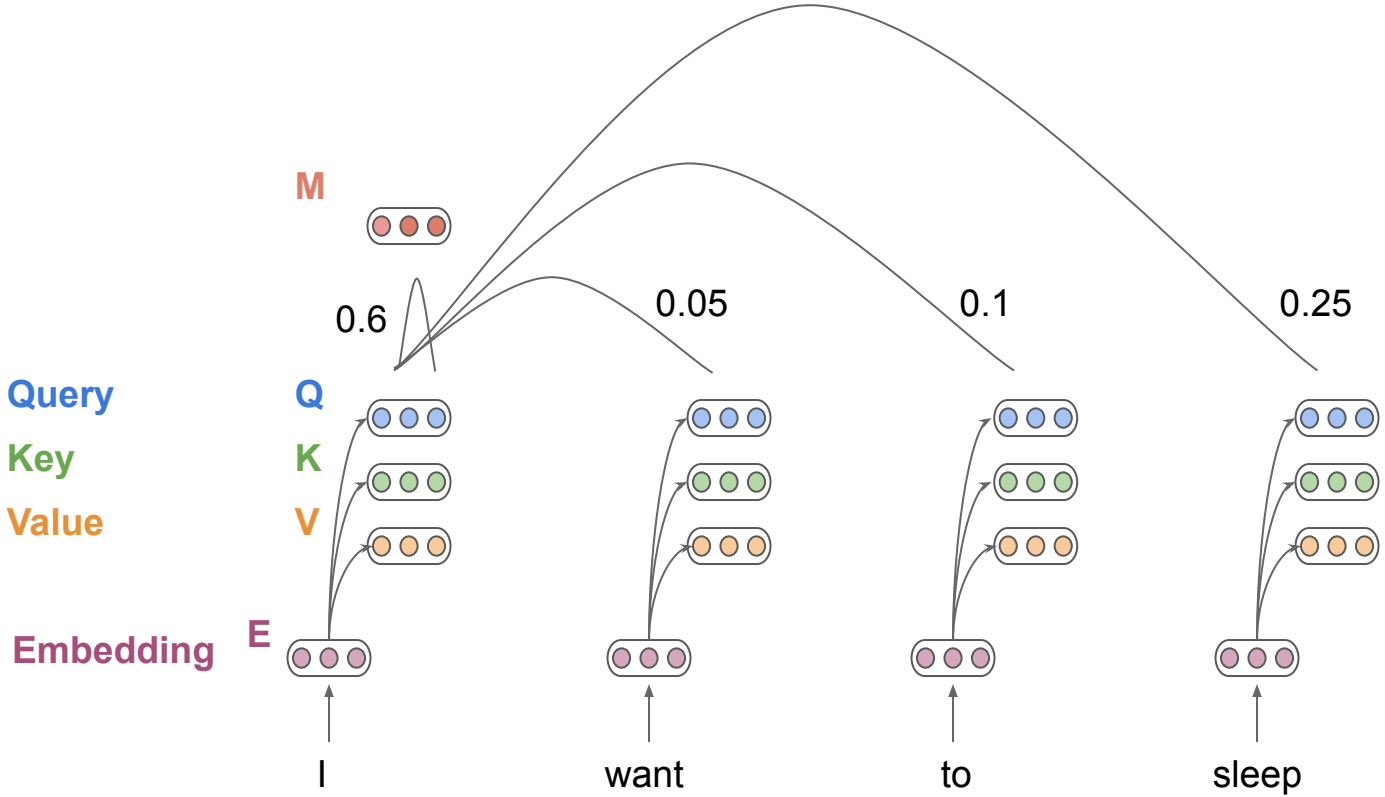
Self Attention - Soft (Relative) Values



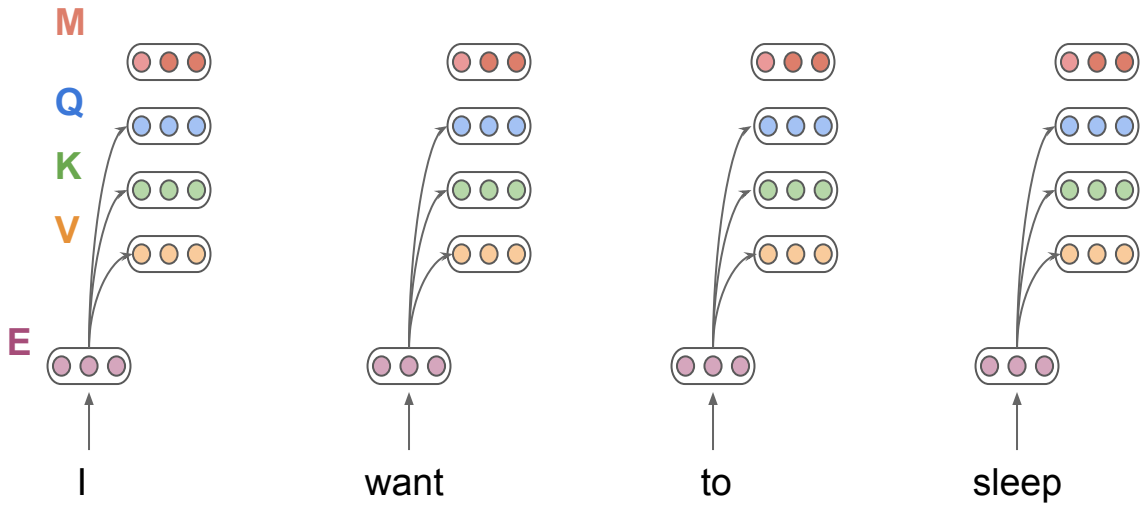
Self Attention - Attended Repr



Self Attention - Attended Contextual Rep



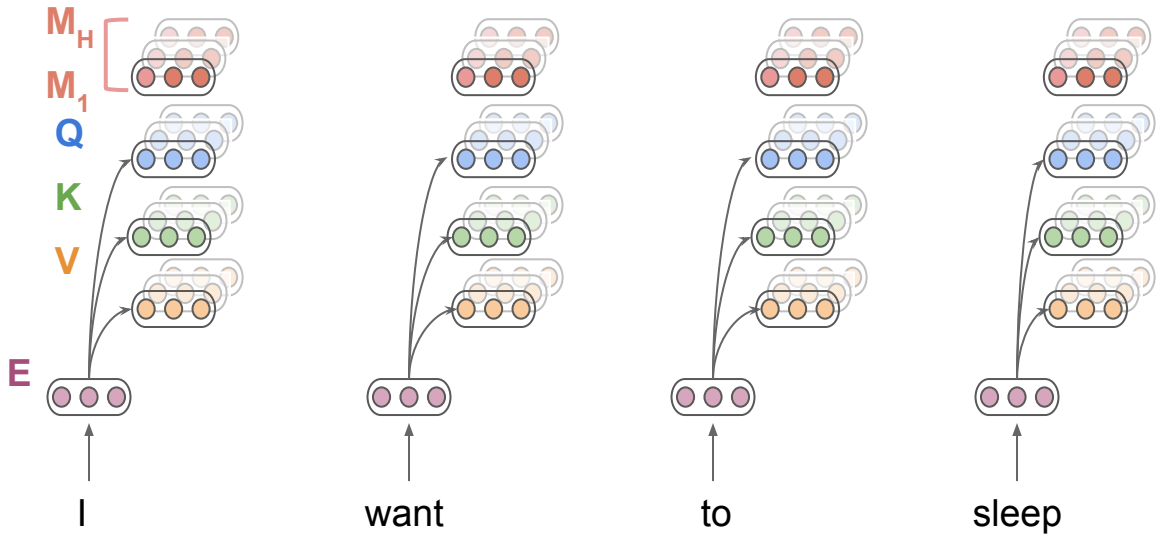
Self Attention - Attended Contextual Rep



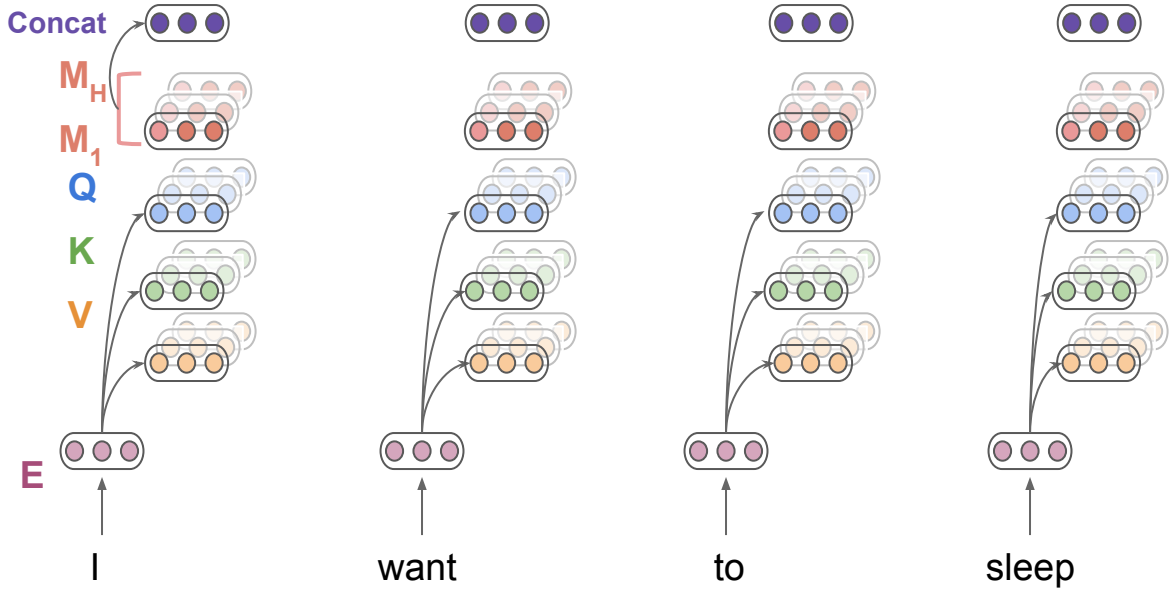
Multi-Headed Self Attention

Love me some representation power!

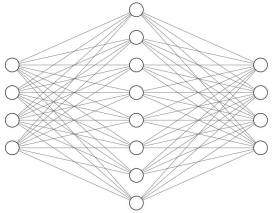
H (no: of heads) Different versions of Q,K,V
Each different repr -> Different attended repr



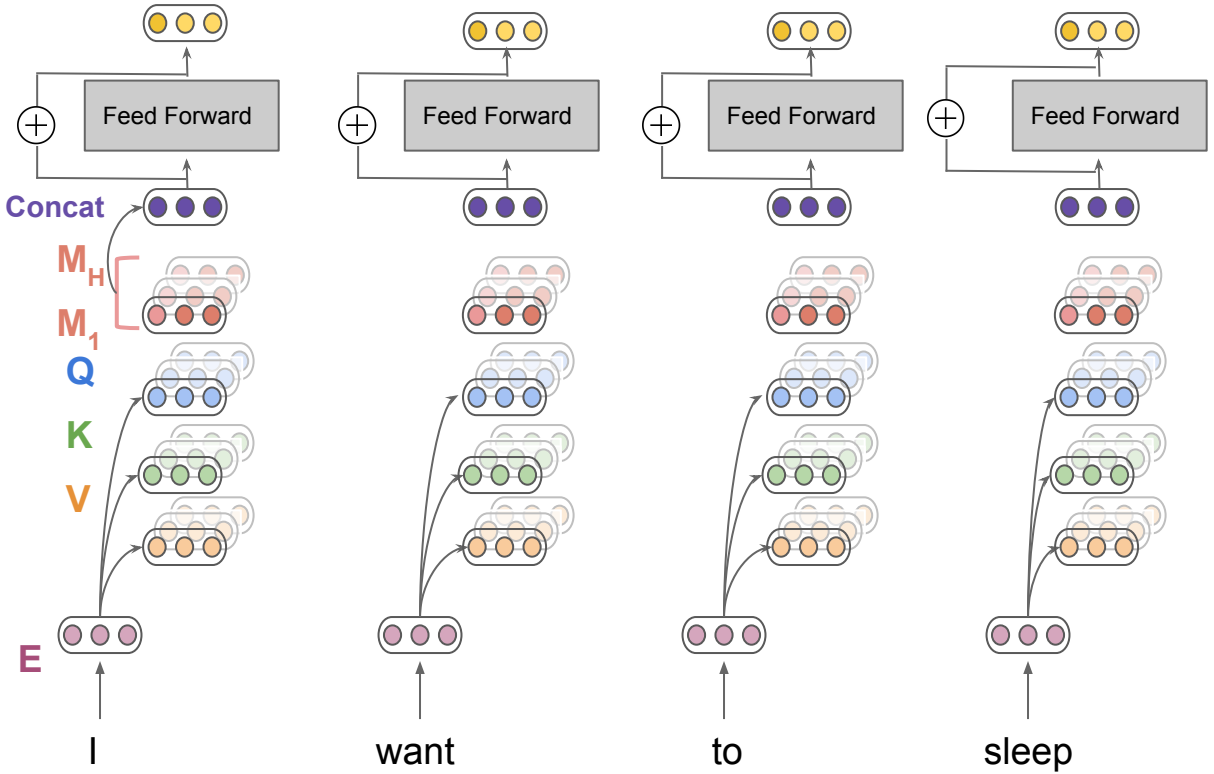
Multi-Headed Self Attention



Multi-Headed Self Attention



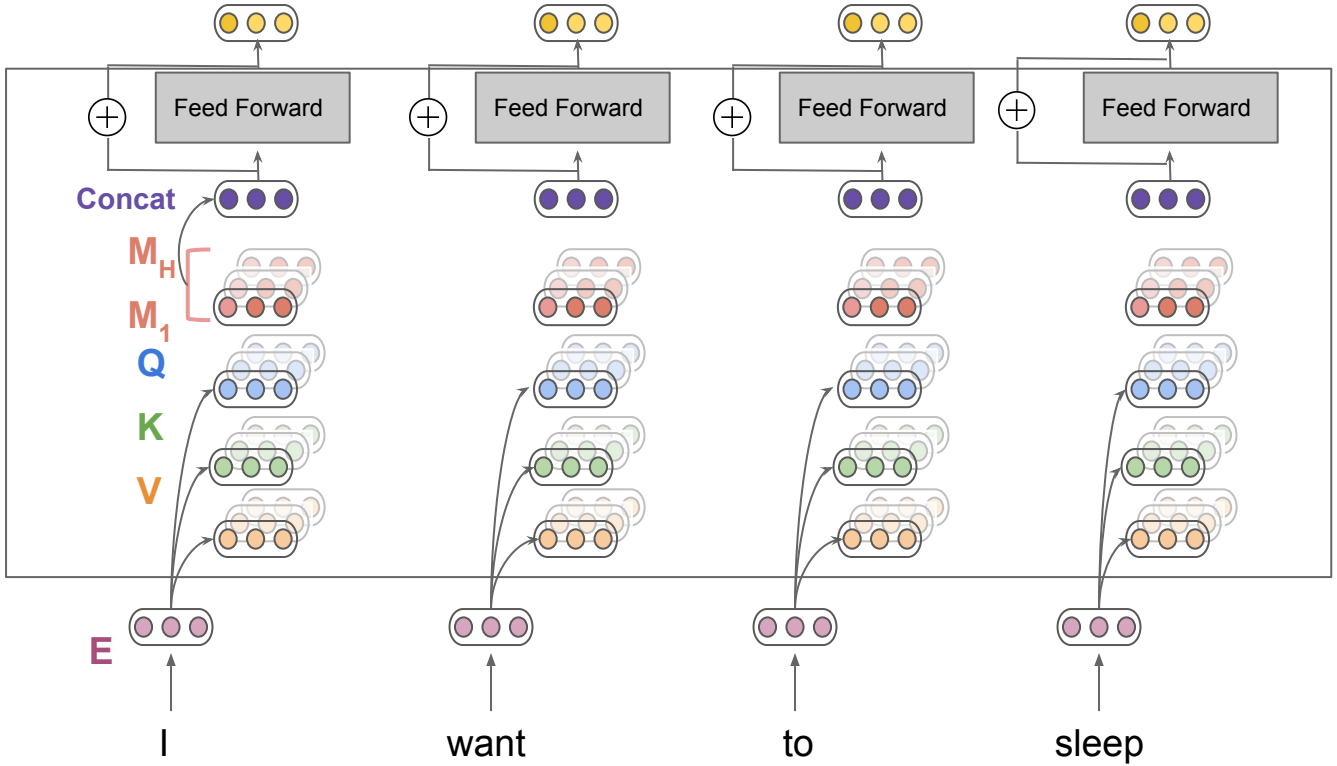
It's Me MLP



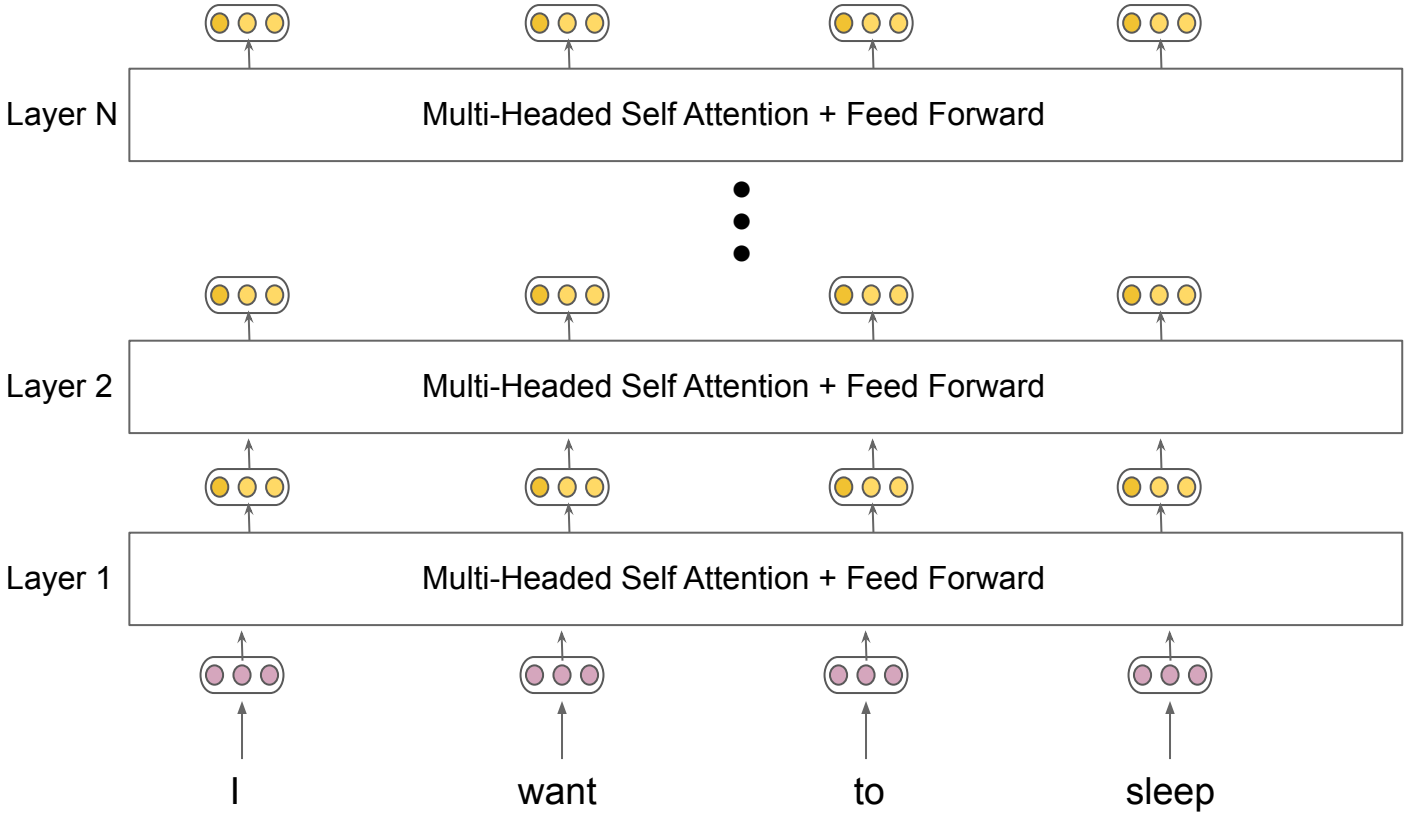
Residual Connections : Help with cleaner gradient flows during back-prop

Multi-Headed Self Attention

Multi-Headed Self Attention + Feed Forward

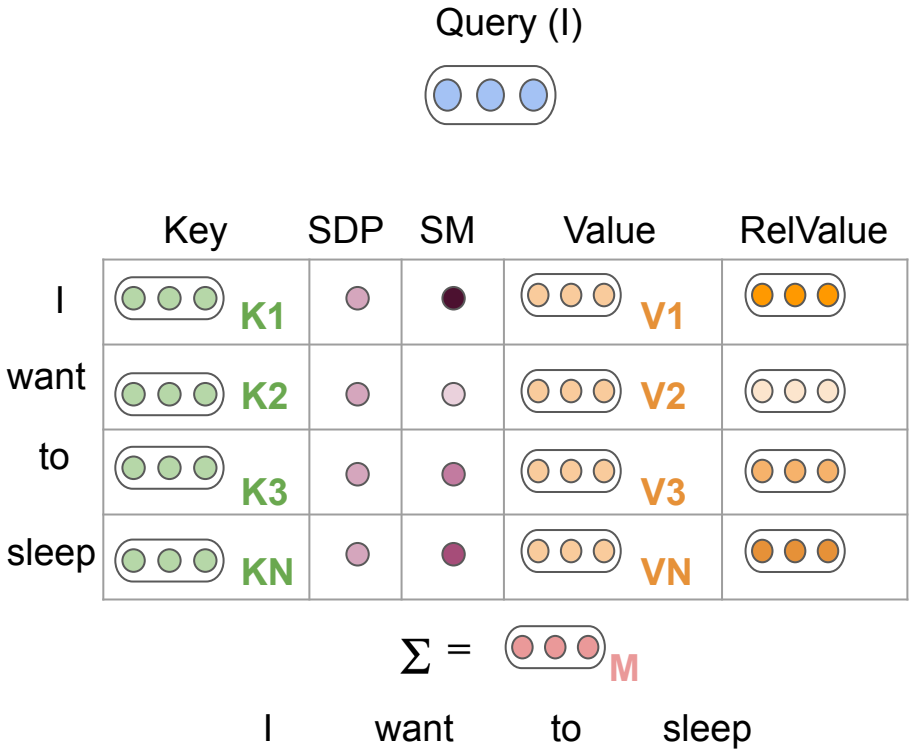


Multi-Headed Self Attention

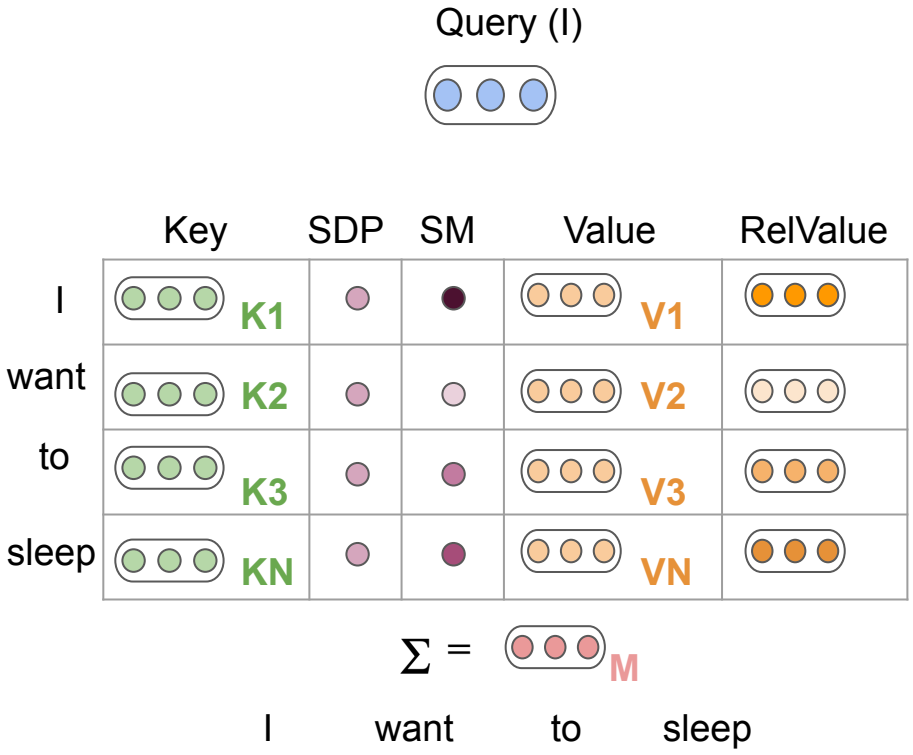


Questions?

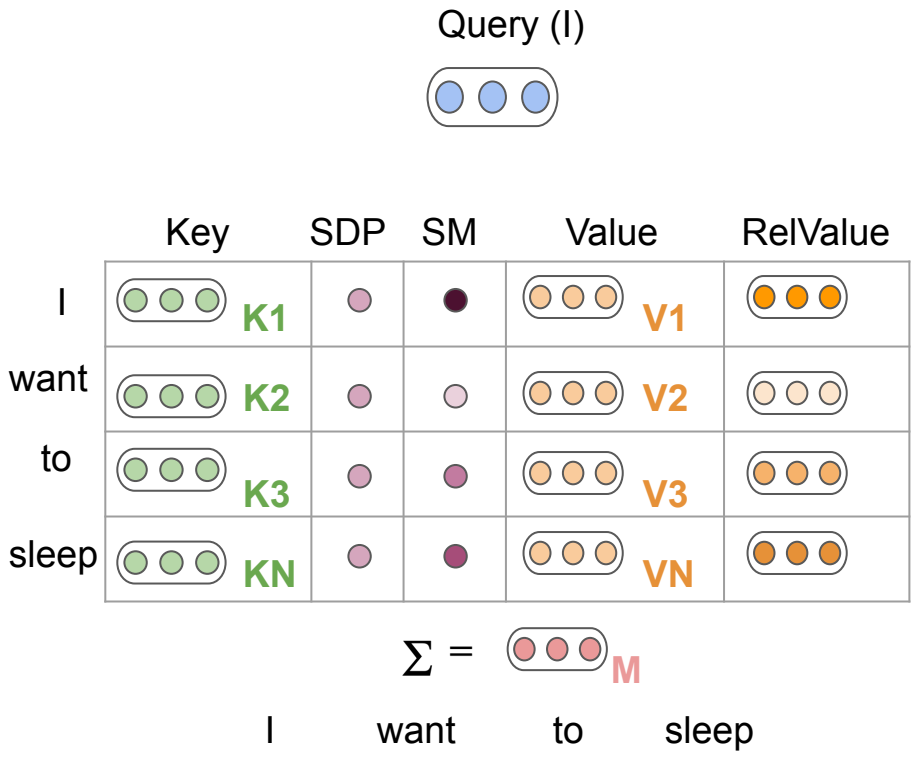
Revisiting Self Attention



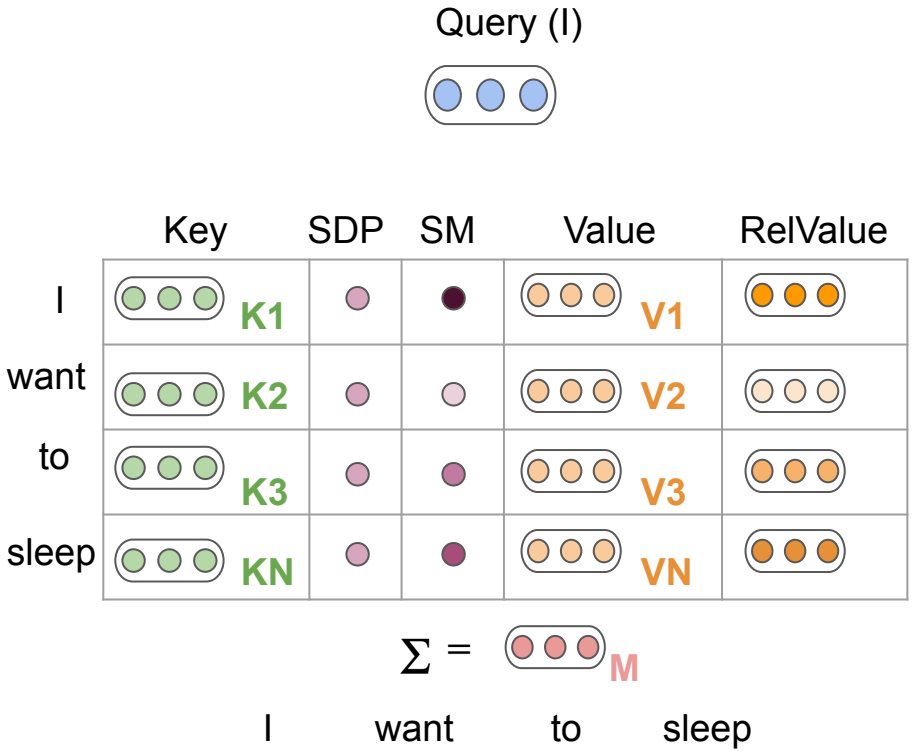
Revisiting Self Attention

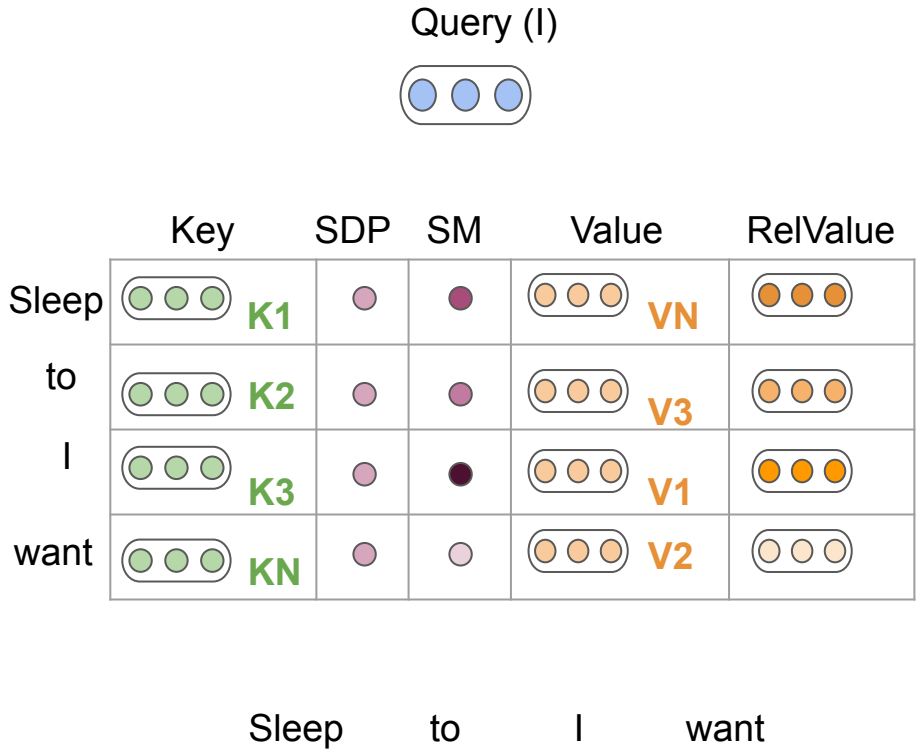


Revisiting Self Attention

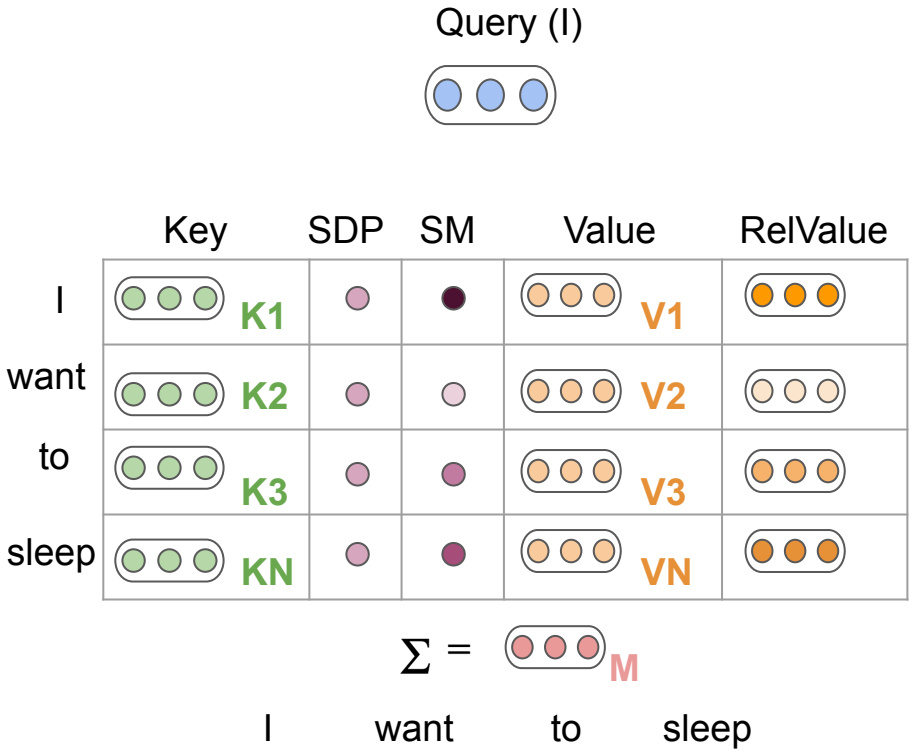


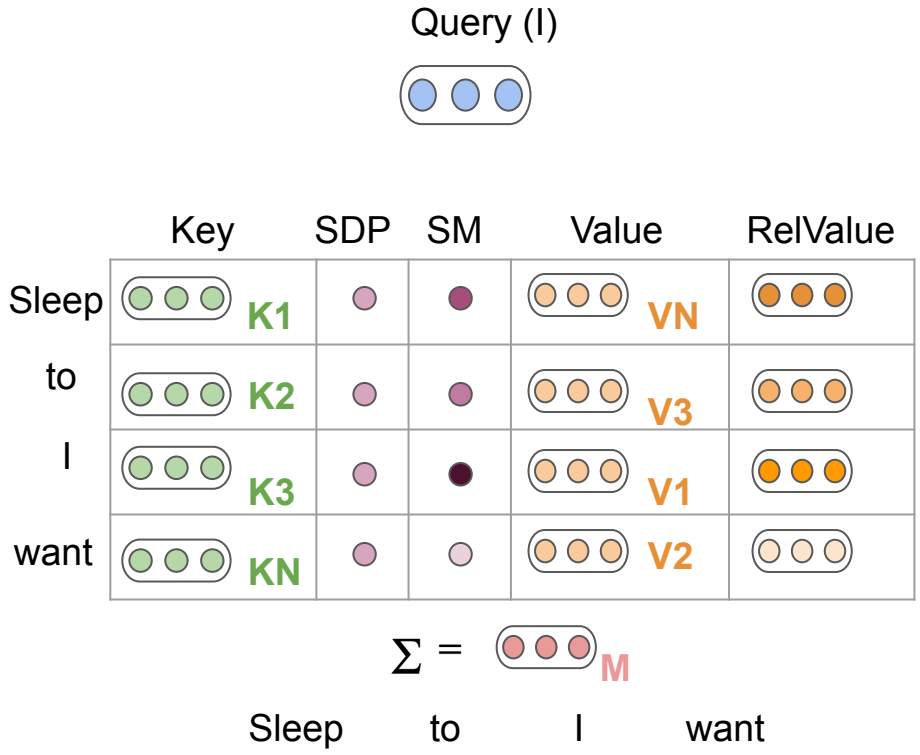
Revisiting Self Attention



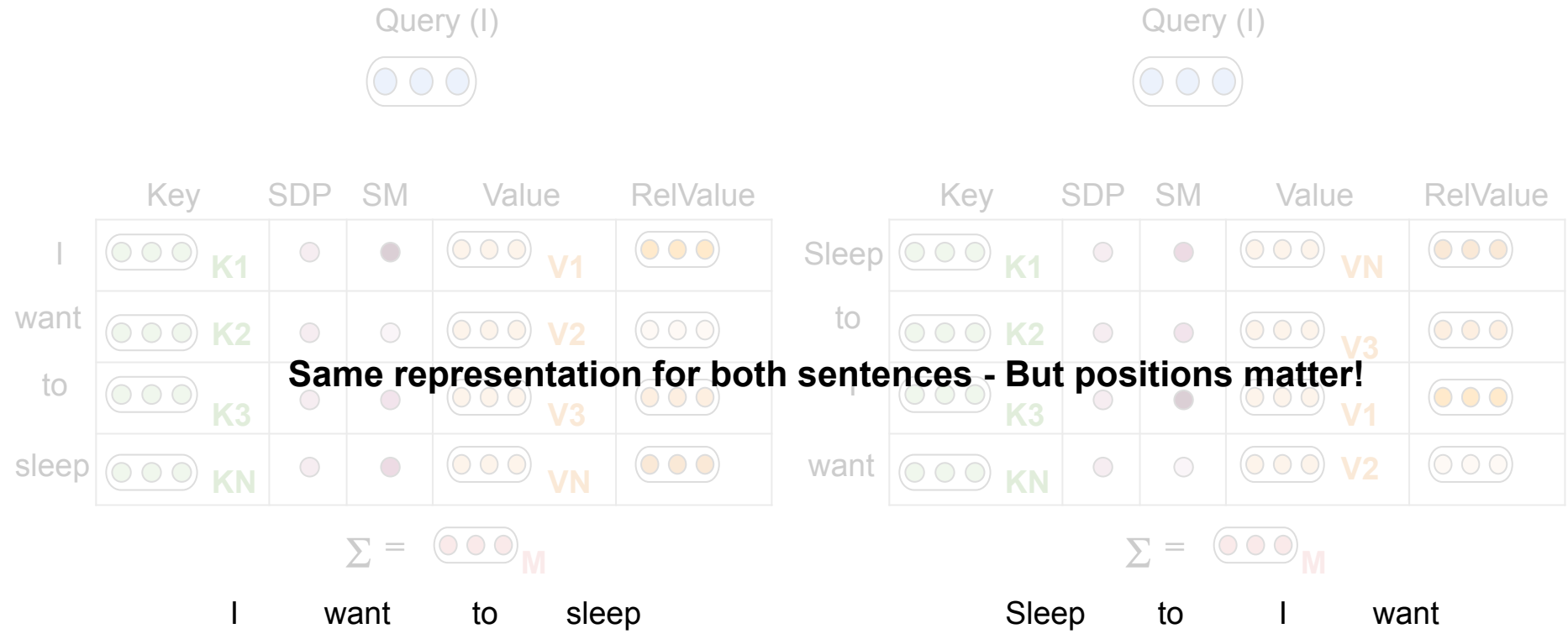


Revisiting Self Attention





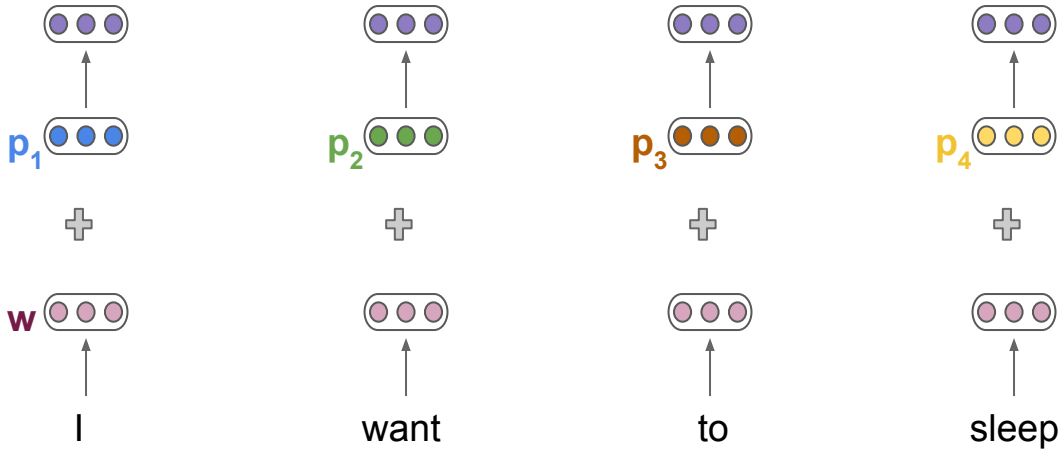
Revisiting Self Attention



Same representation for both sentences - But positions matter!

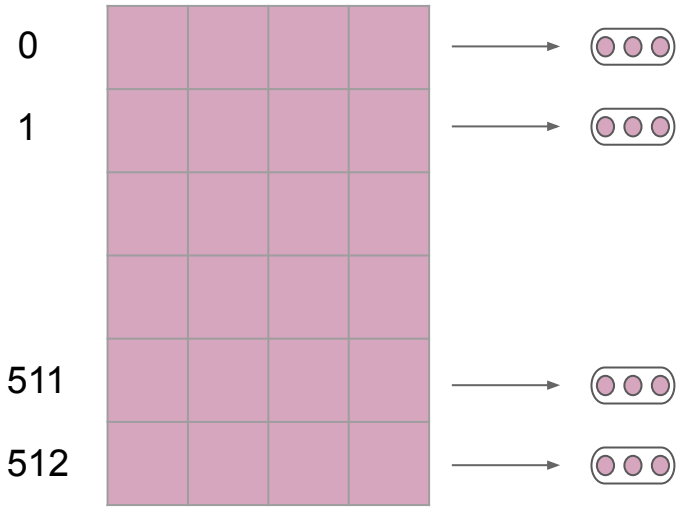
Positional Encoding

Position embeddings - each position number has an associated embedding



Learnable Positional Encoding

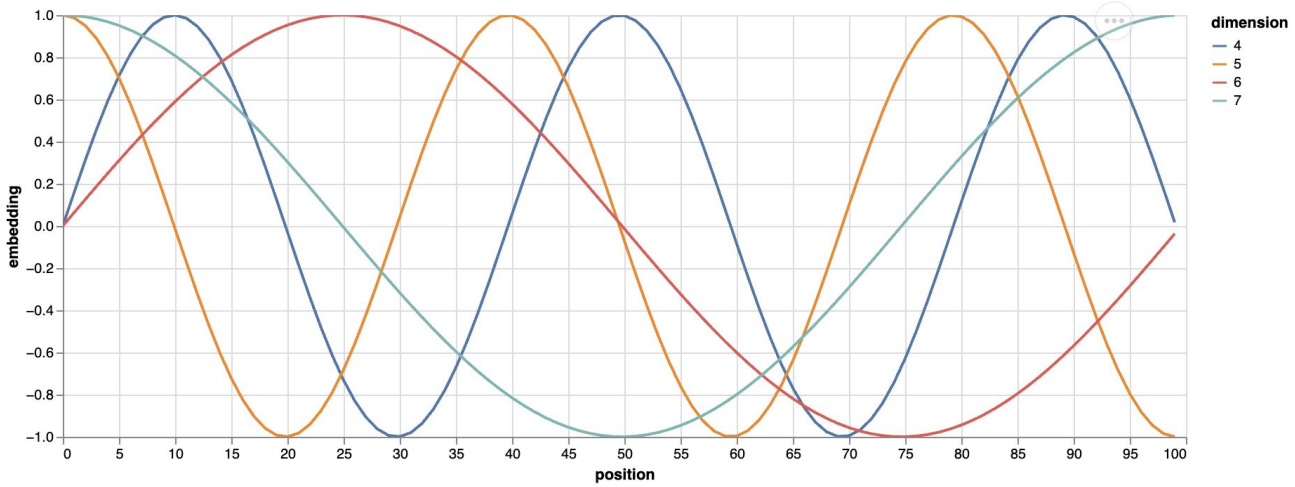
- Simplest type of positional encodings
- Initialize position encodings as random vectors for each position from 0 to MAX LENGTH
- Used in GPT-1, GPT-2, GPT-3
- CONS: Doesn't generalize to sequences of length greater than MAX LENGTH



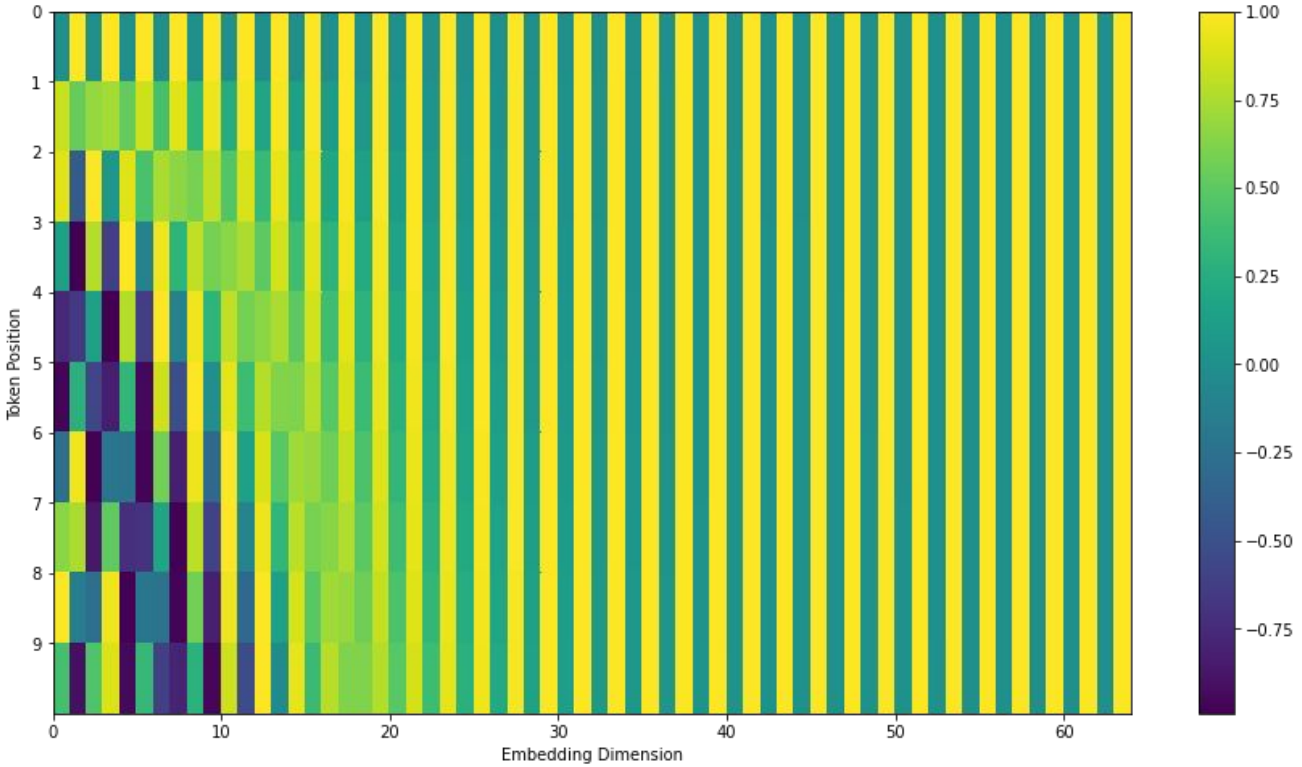
Sinusoidal Positional Encoding

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

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



Sinusoidal Positional Encoding

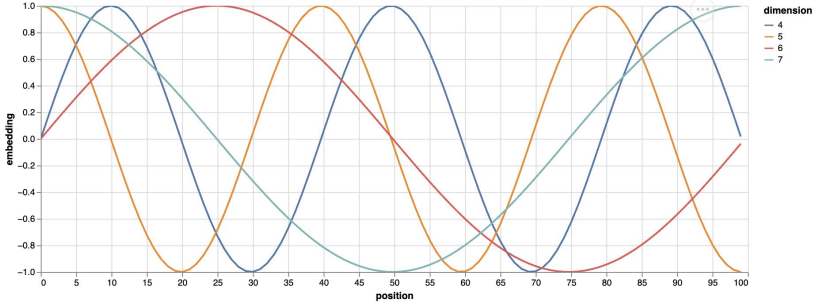


Sinusoidal Positional Encoding

- 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

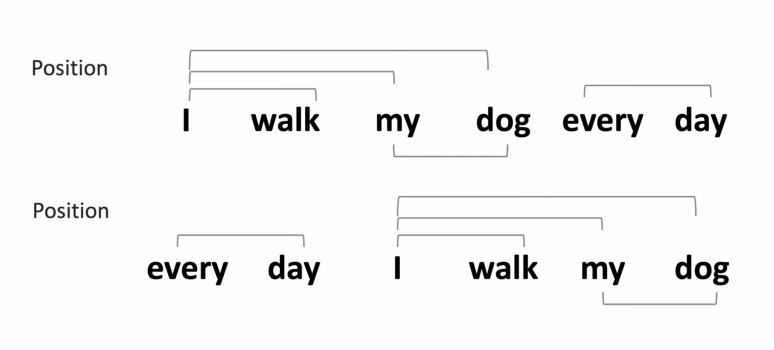
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

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



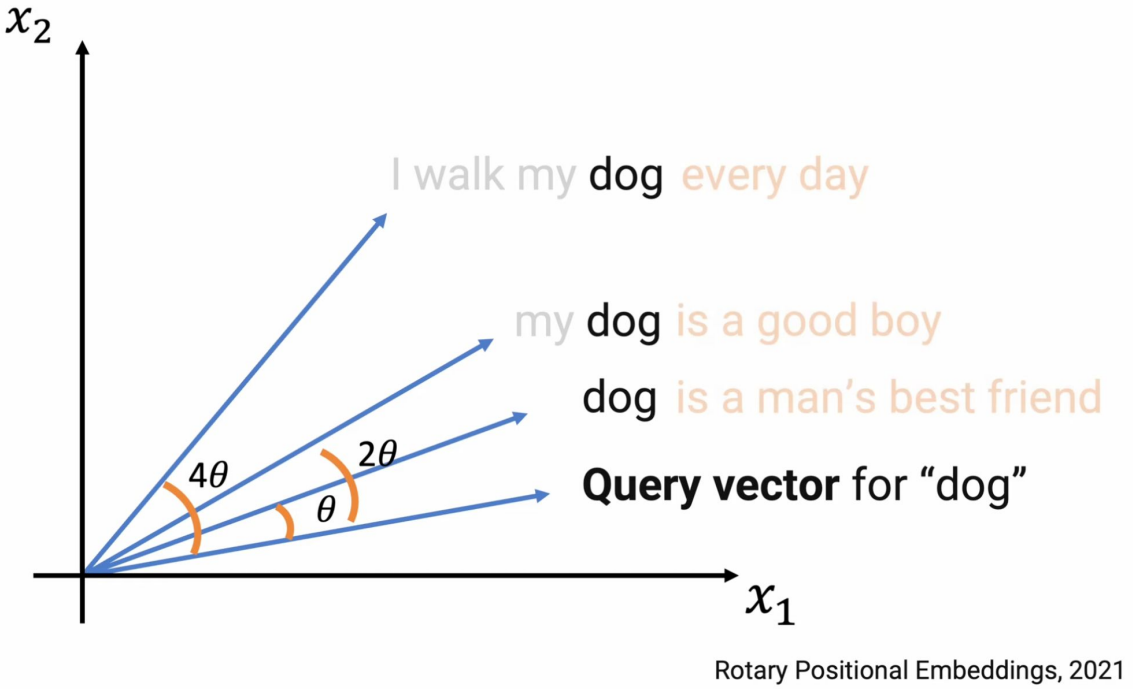
Relative Position Encodings

- 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)



Rotary Positional Embeddings, 2021

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

Rotary Positonal Encodings (RoPE)

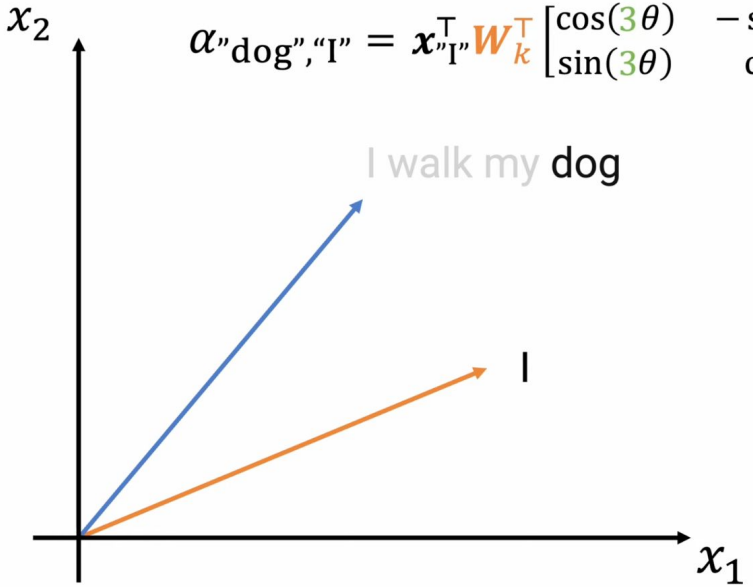
$$q_{\text{"dog"}} = \begin{bmatrix} \cos(4\theta) & -\sin(4\theta) \\ \sin(4\theta) & \cos(4\theta) \end{bmatrix} W_q x_{\text{"dog"}}$$

Query vector for "dog"

$$k_{\text{"I"}} = \begin{bmatrix} \cos(1\theta) & -\sin(1\theta) \\ \sin(1\theta) & \cos(1\theta) \end{bmatrix} W_k x_{\text{"I"}}$$

Key vector for "I"

$$\alpha_{\text{"dog","I"}} = x_{\text{"I"}}^{\top} W_k^{\top} \begin{bmatrix} \cos(3\theta) & -\sin(3\theta) \\ \sin(3\theta) & \cos(3\theta) \end{bmatrix} W_q x_{\text{"dog"}}$$



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

Rotary Positonal Encodings (RoPE)

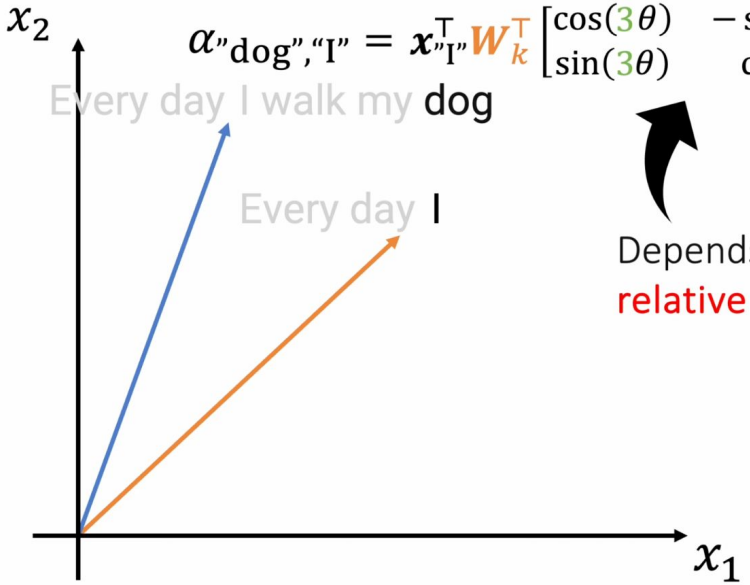
$$q_{\text{"dog"}} = \begin{bmatrix} \cos(6\theta) & -\sin(6\theta) \\ \sin(6\theta) & \cos(6\theta) \end{bmatrix} W_q \boldsymbol{x}_{\text{"dog"}}$$

Query vector for "dog"

$$k_{\text{"I"}} = \begin{bmatrix} \cos(3\theta) & -\sin(3\theta) \\ \sin(3\theta) & \cos(3\theta) \end{bmatrix} W_k \boldsymbol{x}_{\text{"I"}}$$

Key vector for "I"

$$\alpha_{\text{"dog","I"}} = \boldsymbol{x}_{\text{"I"}}^{\top} W_k^{\top} \begin{bmatrix} \cos(3\theta) & -\sin(3\theta) \\ \sin(3\theta) & \cos(3\theta) \end{bmatrix} W_q \boldsymbol{x}_{\text{"dog"}}$$



Depends only on the relative positions

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

Rotary Positonal Encodings (RoPE)

$$R_\Theta^m \, W_q \, x_m$$

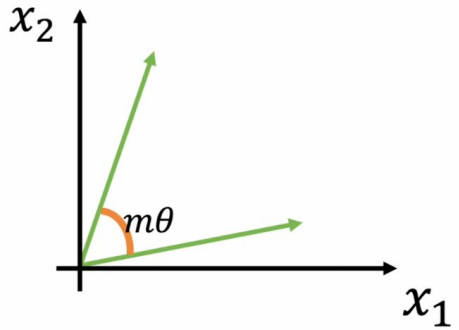
query vector for the token at position m

$$R_\Theta^m = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$

$$R_\Theta^n \, W_k \, x_n$$

key vector for the token at position n

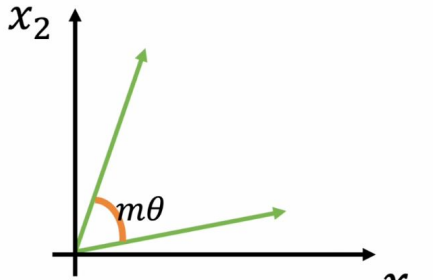
$$\alpha_{m,n} = x_n^\top \, W_k^\top \quad R_\Theta^{m-n} \quad W_q \, x_m$$

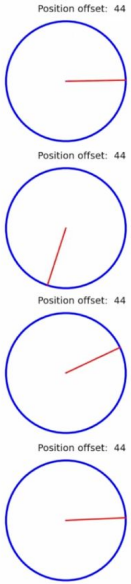


From Jia-Bin Huan's Youtube Video: https://www.youtube.com/watch?v=SMBkImDWOyQ&t=683s

Rotary Positonal Encodings (RoPE)

$$R_\Theta^m = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$





$$\begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) \\ \sin(m\theta_1) & \cos(m\theta_1) \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix}$$

$$\begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \cos(m\theta_2) & -\sin(m\theta_2) \\ \sin(m\theta_2) & \cos(m\theta_2) \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix}$$

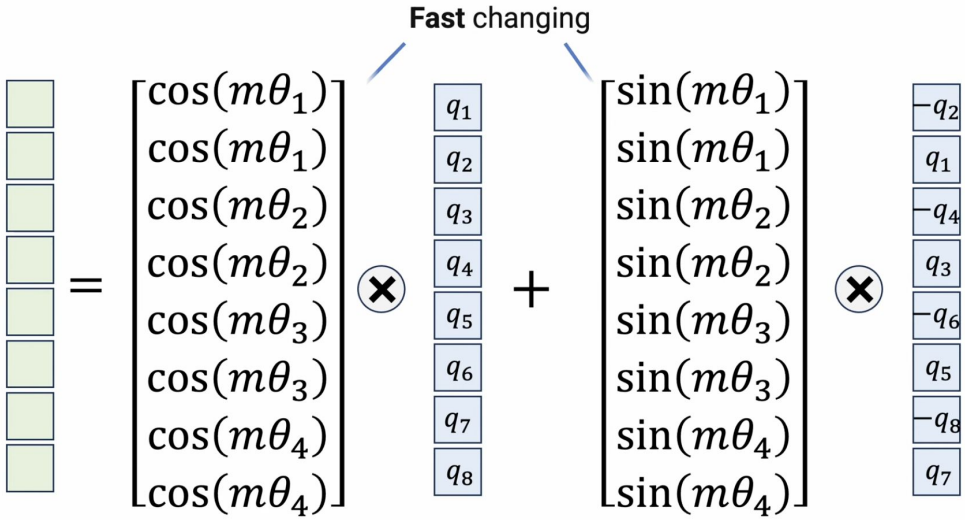
$$\begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \cos(m\theta_3) & -\sin(m\theta_3) \\ \sin(m\theta_3) & \cos(m\theta_3) \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix}$$

$$\begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \cos(m\theta_4) & -\sin(m\theta_4) \\ \sin(m\theta_4) & \cos(m\theta_4) \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix}$$

$$W_q x_m$$

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

Rotary Positonal Encodings (RoPE)



$f_q(\boldsymbol{x}_m, m)$

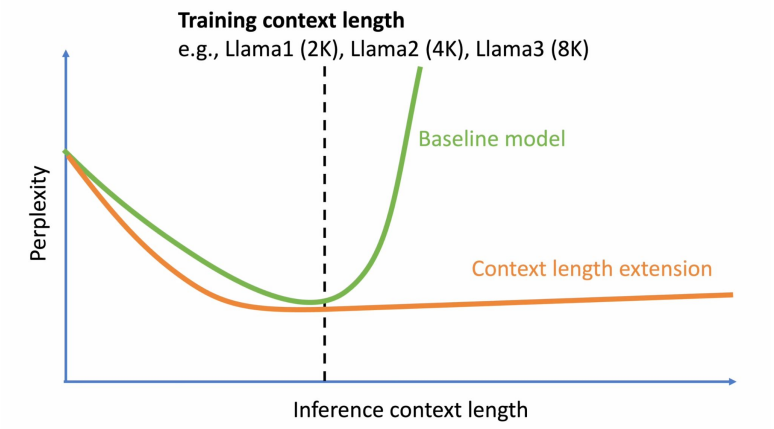
$\theta_i = N^{-2i/d}$

$N = 10{,}000$

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

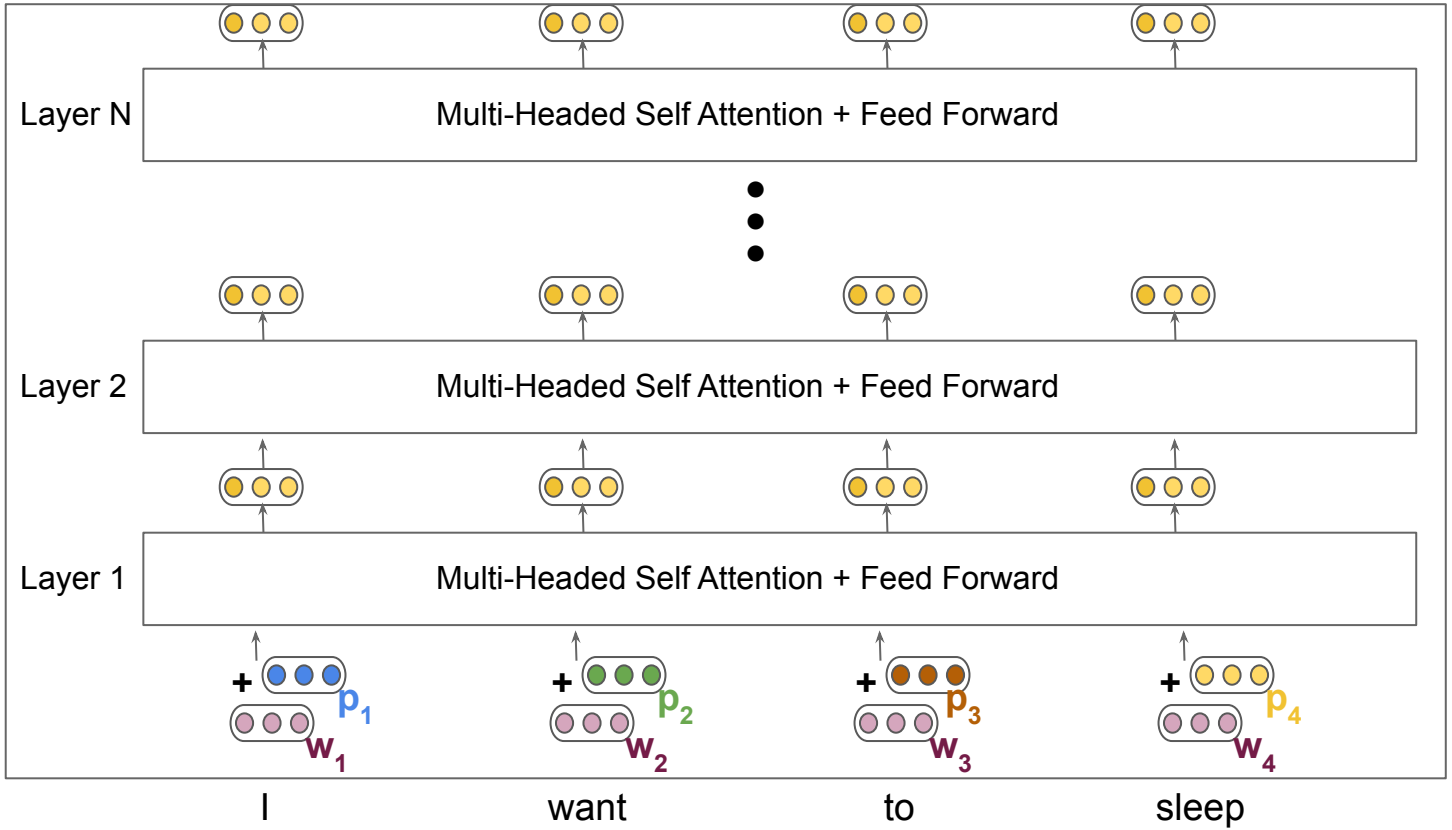


From Jia-Bin Huan's Youtube Video: https://www.youtube.com/watch?v=SMBkImDWOyQ&t=683s

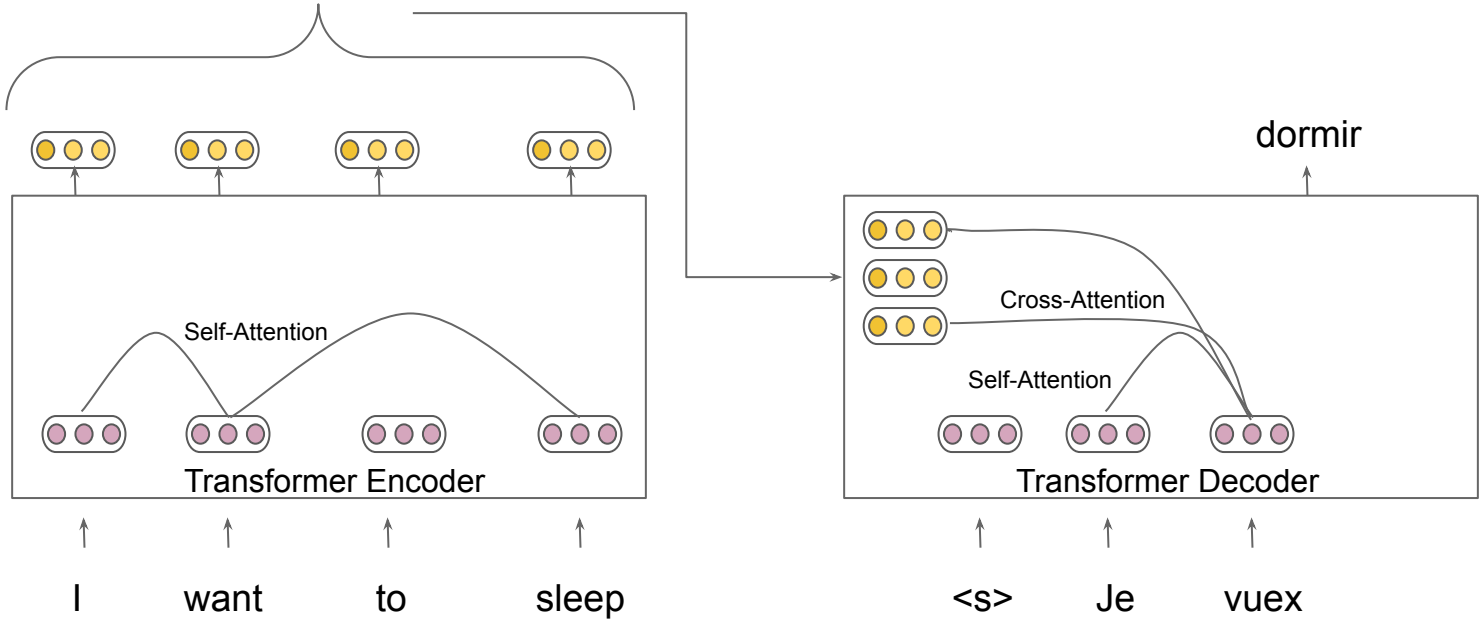
Questions?

Transformer Encoder

N-Layer Transformer Encoder



Transformer Encoder - Decoder

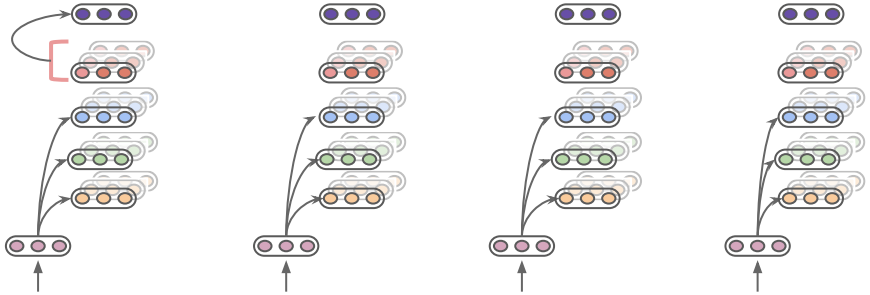


What's so great about Transformers?

- 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

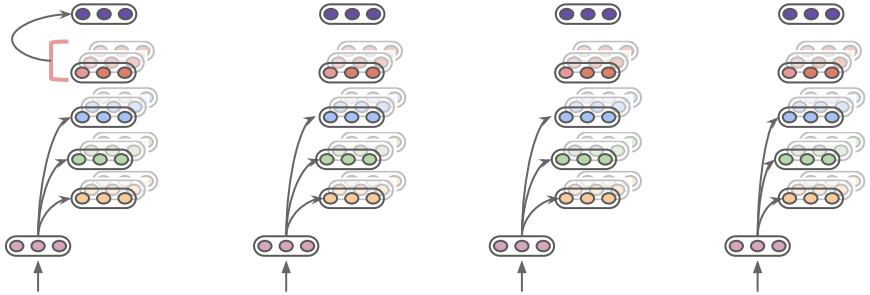
What's so great about Transformers?

- Parallelizable computation - Entire sequence can be processed in parallel

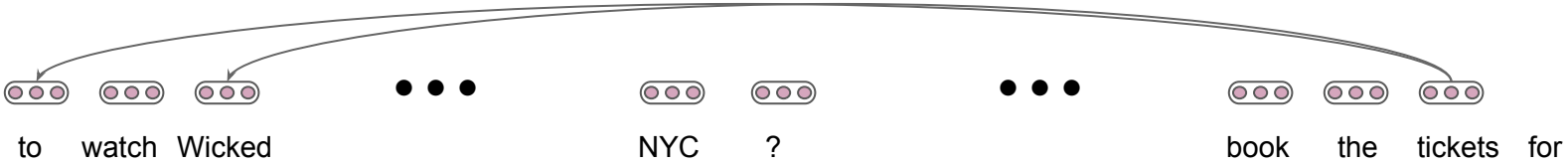


What's so great about Transformers?

- Parallelizable computation - Entire sequence can be processed in parallel



- Rich expressive power - long range dependencies



Thank you!

kahuja@cs.washington.edu

Results/Impact

- Improves results, Establishes SOTA in various tasks!
  - Machine Translation
  - Constituency Parsing
  - Language Modeling
  - and more!

- Computationally faster!
  - No sequential computation - Entire sequence processed in parallel