SILO AI

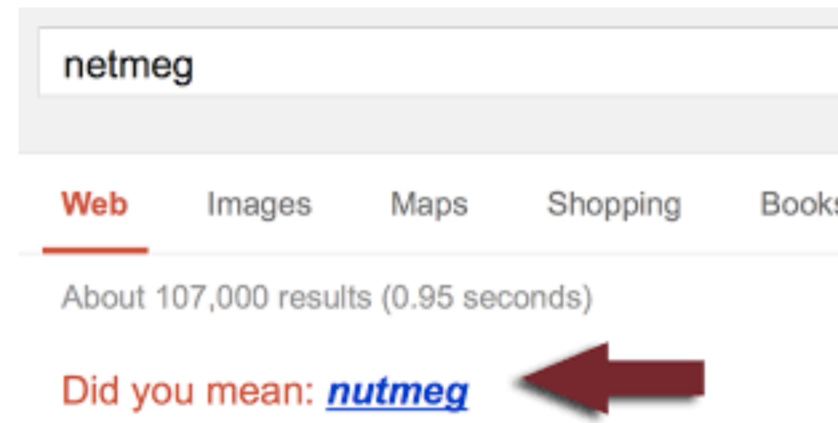
Neural Network Language Models & BERT

Mittul Singh

Language Model Applications

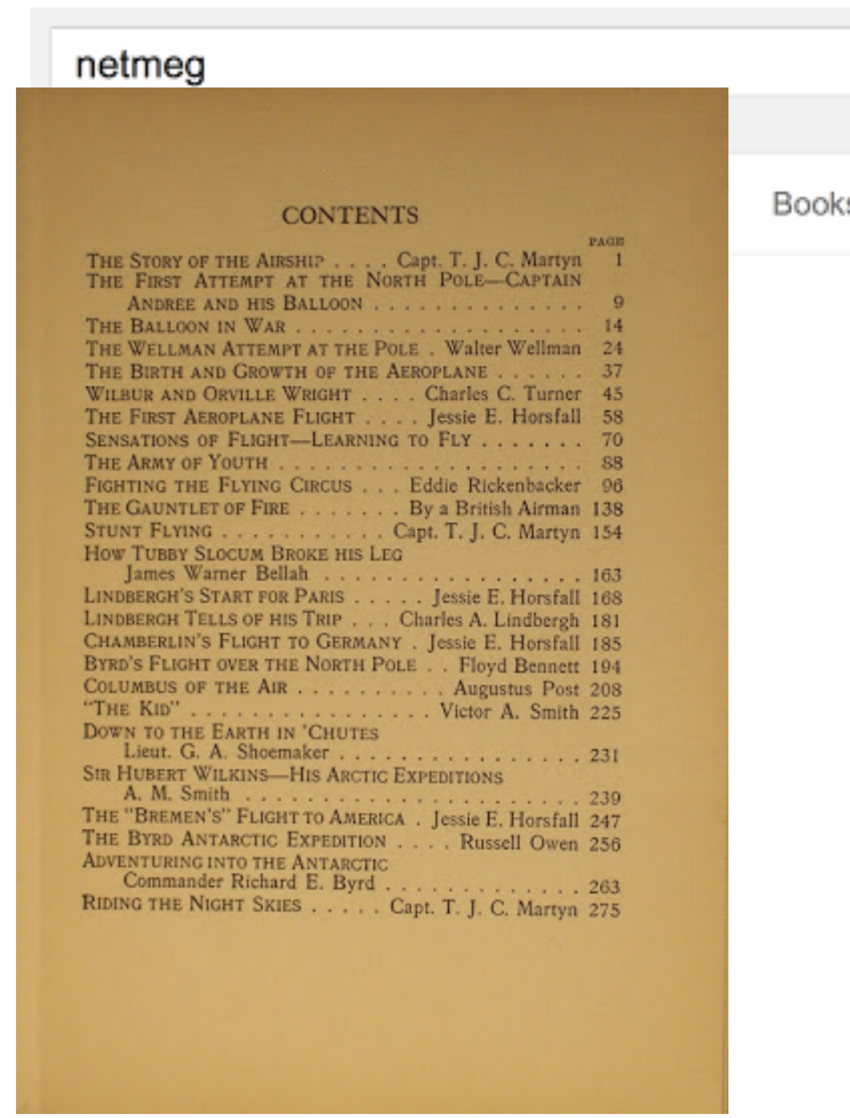
Language Model Applications

- Spelling correction, text input
    - Search Query Completion



Language Model Applications

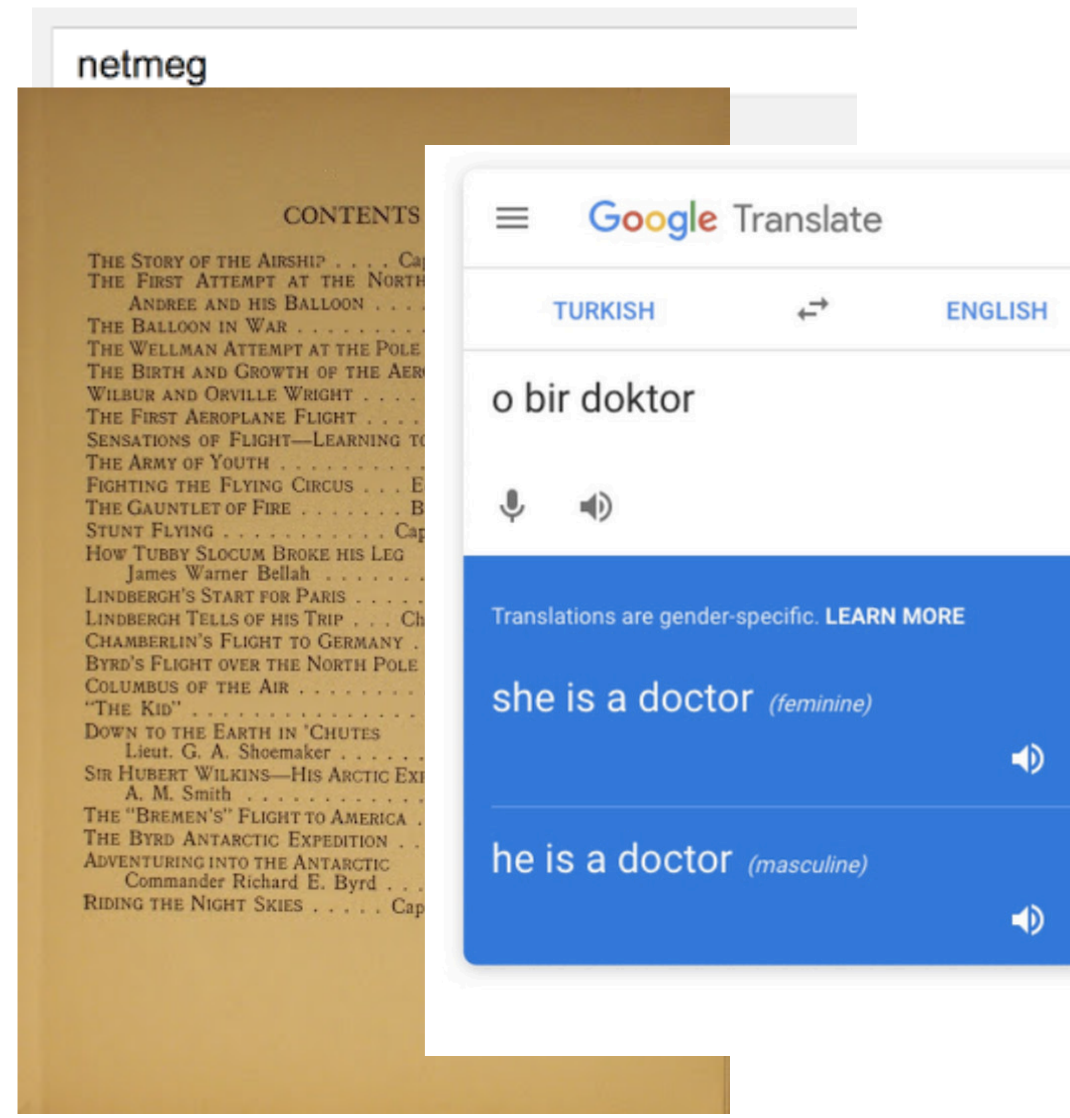
- Spelling correction, text input
    - Search Query Completion
- Optical character recognition
    - e.g. scanning old books



CONTENTS

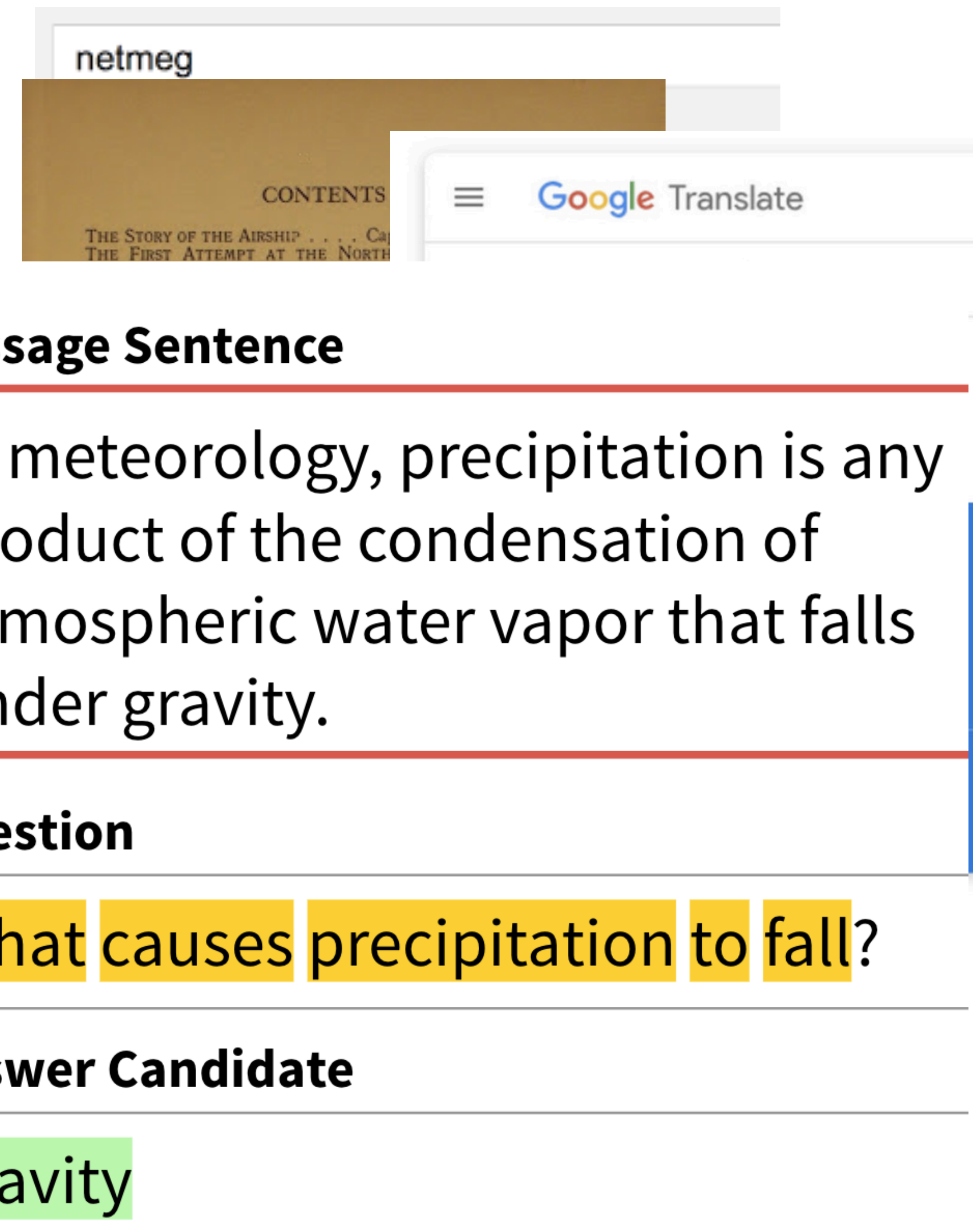
Language Model Applications

- Spelling correction, text input
  - Search Query Completion
- Optical character recognition
  - e.g. scanning old books
- Statistical machine translation



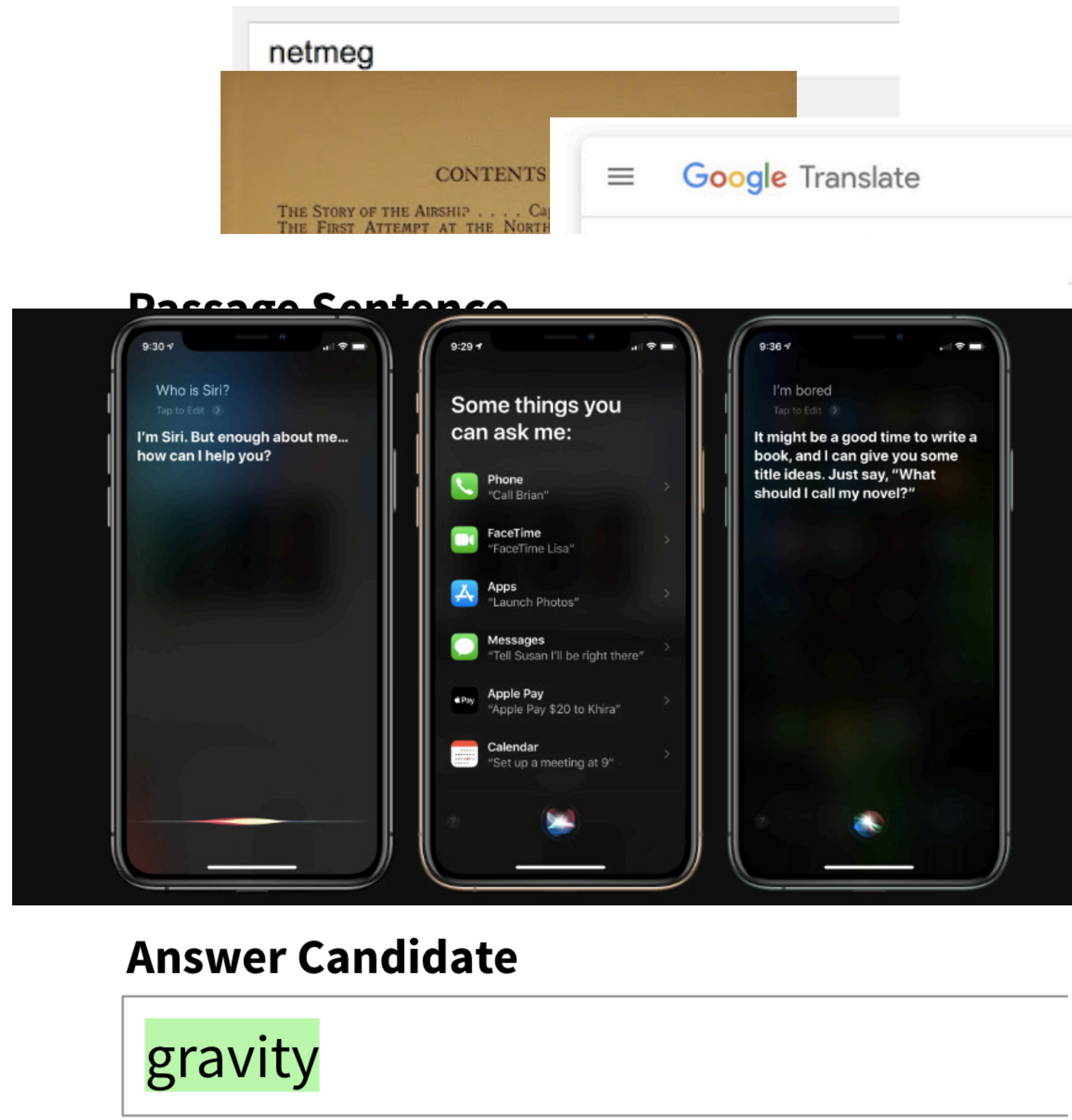
Language Model Applications

- Spelling correction, text input
    - Search Query Completion
- Optical character recognition
    - e.g. scanning old books
- Statistical machine translation
- Information retrieval
    - Question Answering



Language Model Applications

- Spelling correction, text input
    - Search Query Completion
- Optical character recognition
    - e.g. scanning old books
- Statistical machine translation
- Information retrieval
    - Question Answering
- Automatic speech recognition
- …



Recap: N-gram Language Models

Recap: N-gram Language Models

- We wanted to calculate

$$p(W) = p(w_1, w_2, \ldots, w_n) \tag{1}$$

$$p(w_i|w_{i-1}, w_{i-2}, \ldots, w_{n-1}) \approx p(w_i|w_{i-1}, w_{i-2}, w_{i-3}, w_{i-4}) \tag{2}$$

Neural Network Classifier for Language Modelling

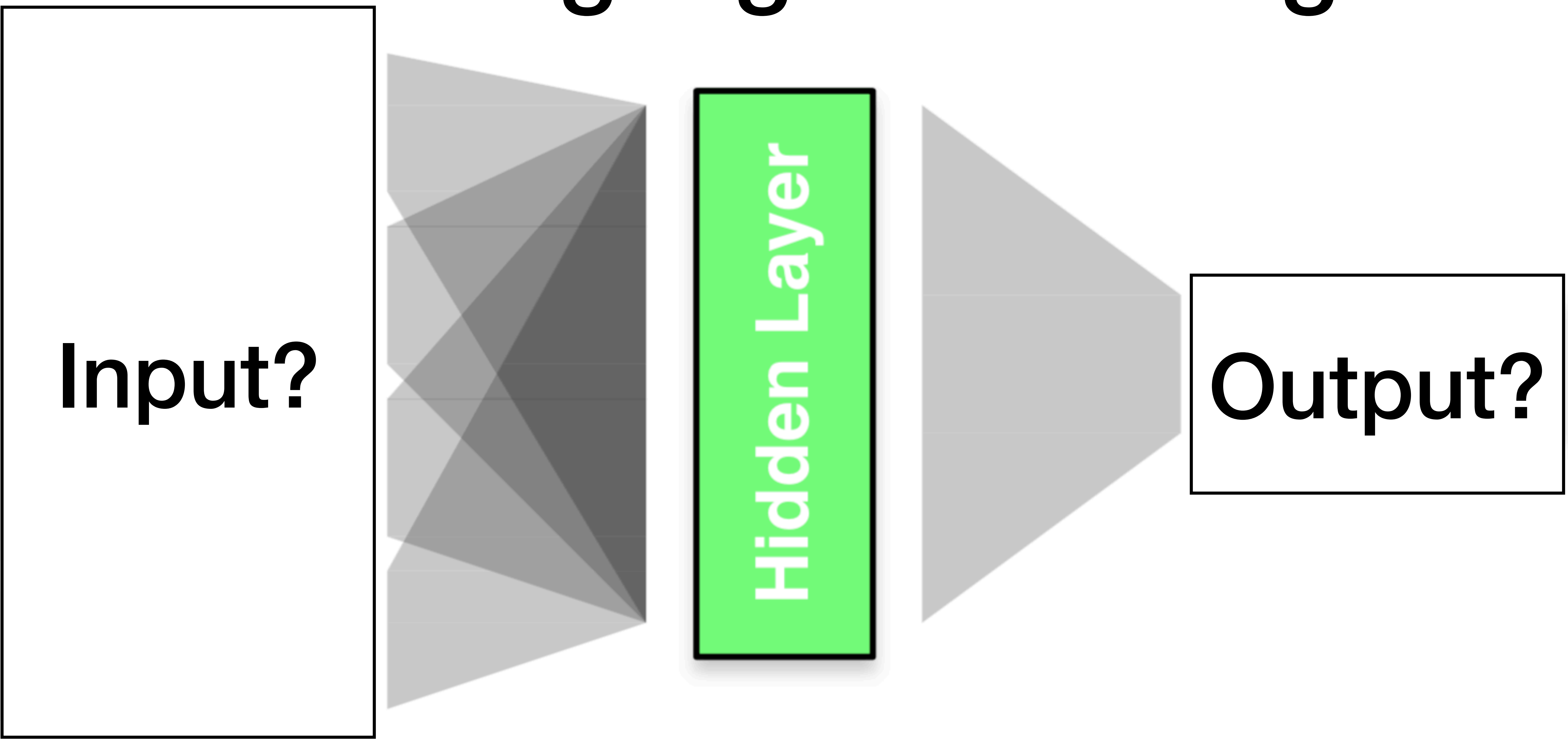


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

Neural Network Classifier for Language Modelling

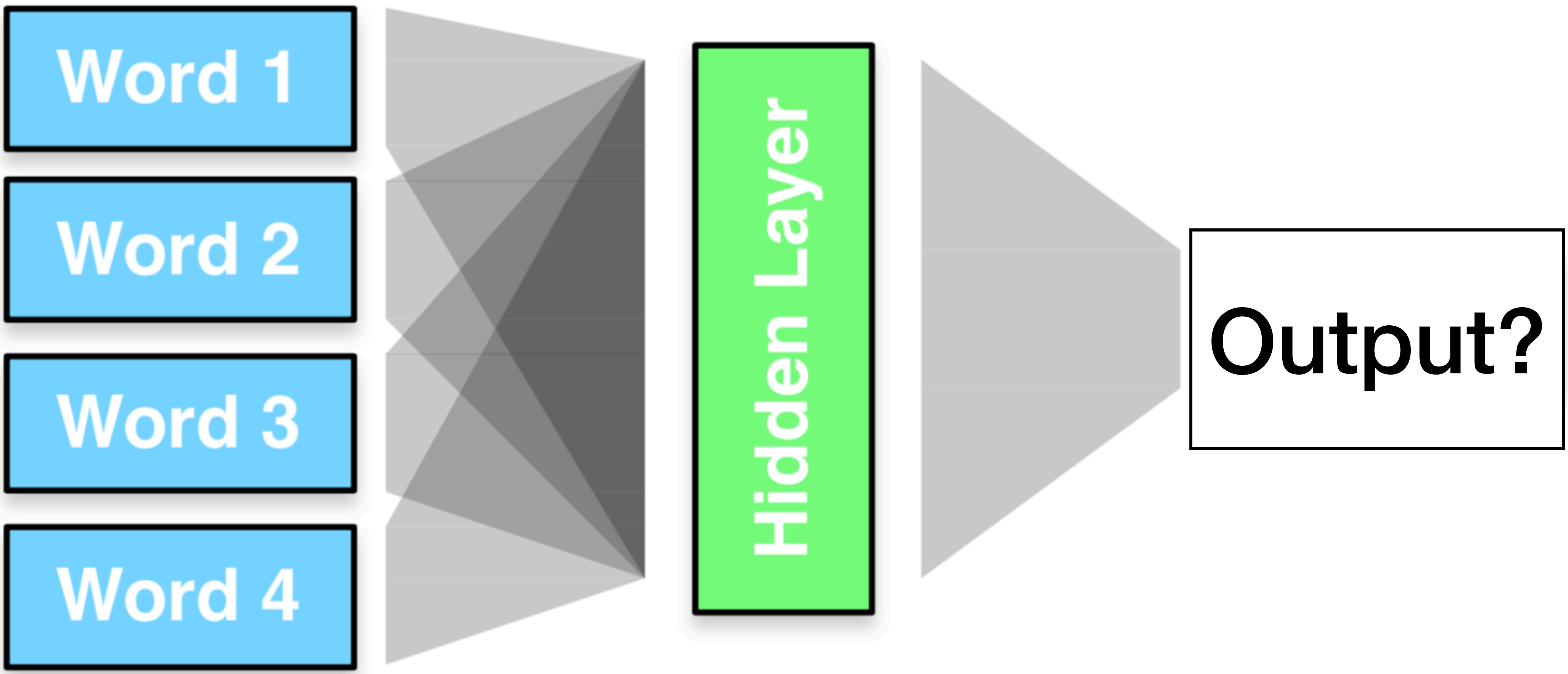


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

Neural Network Classifier for Language Modelling

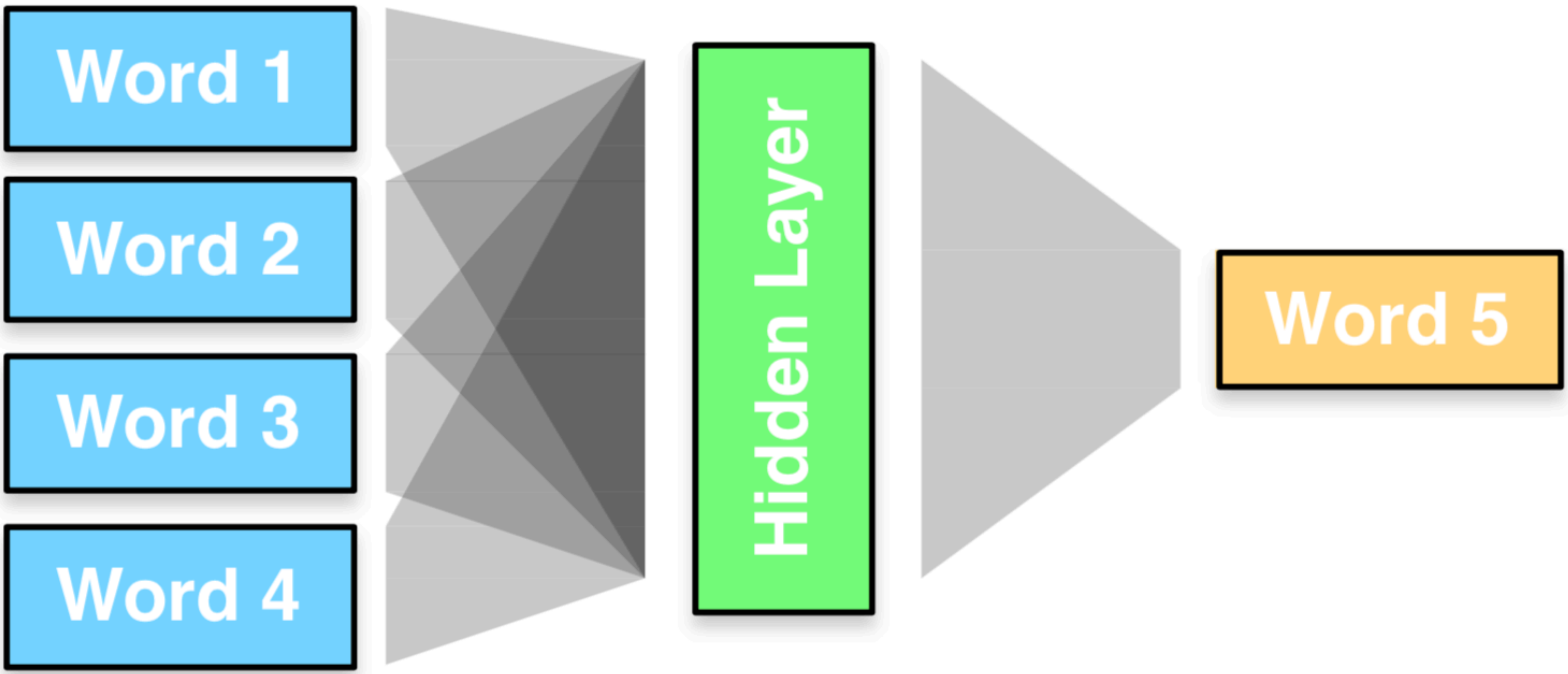


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

Representing Words

Representing Words

- Words are represented with one-hot vector, e.g.,
  - dog = (0, 0, 0, 1, 0, 0, …)
  - cat = (0, 0, 0, 0, 0, 1, …)
  - eat = (0, 1, 0, 0, 0, 0, …)

Second Sketch

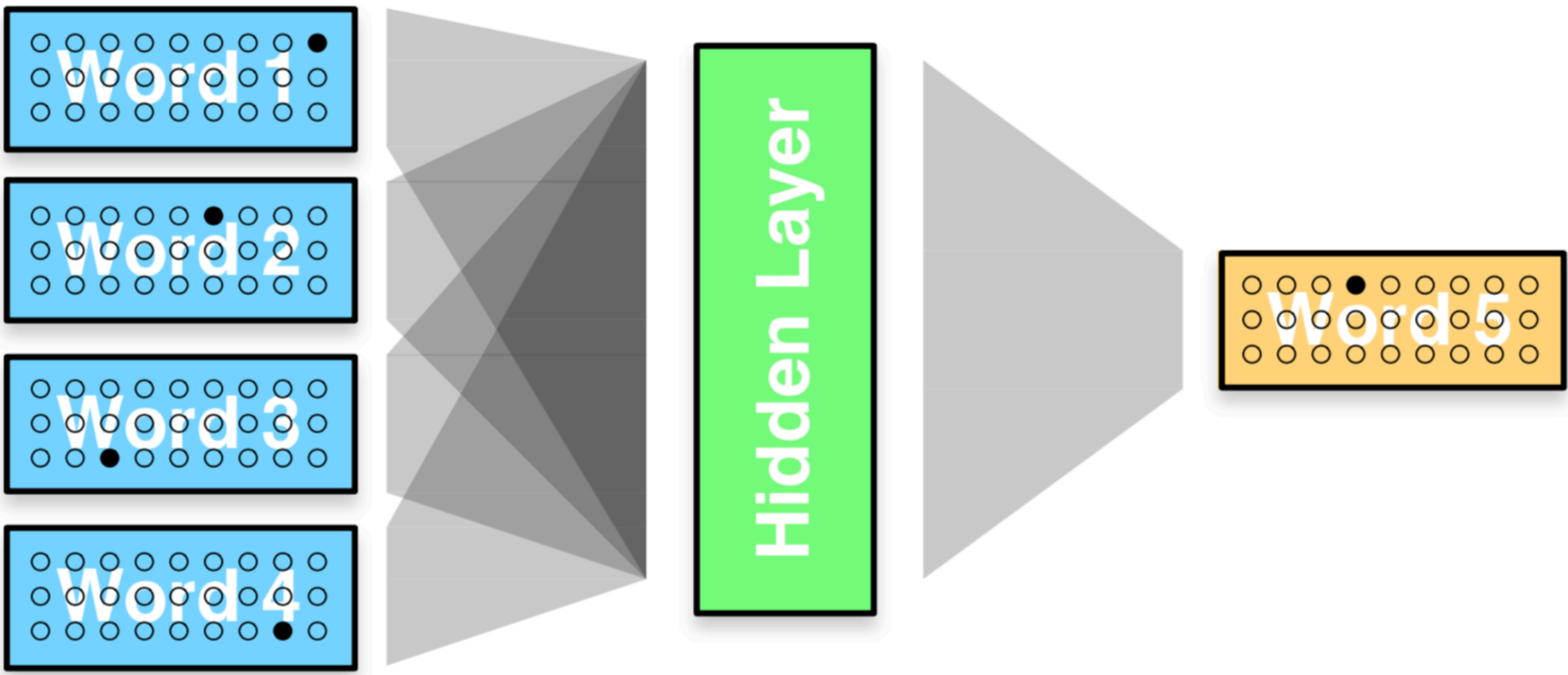
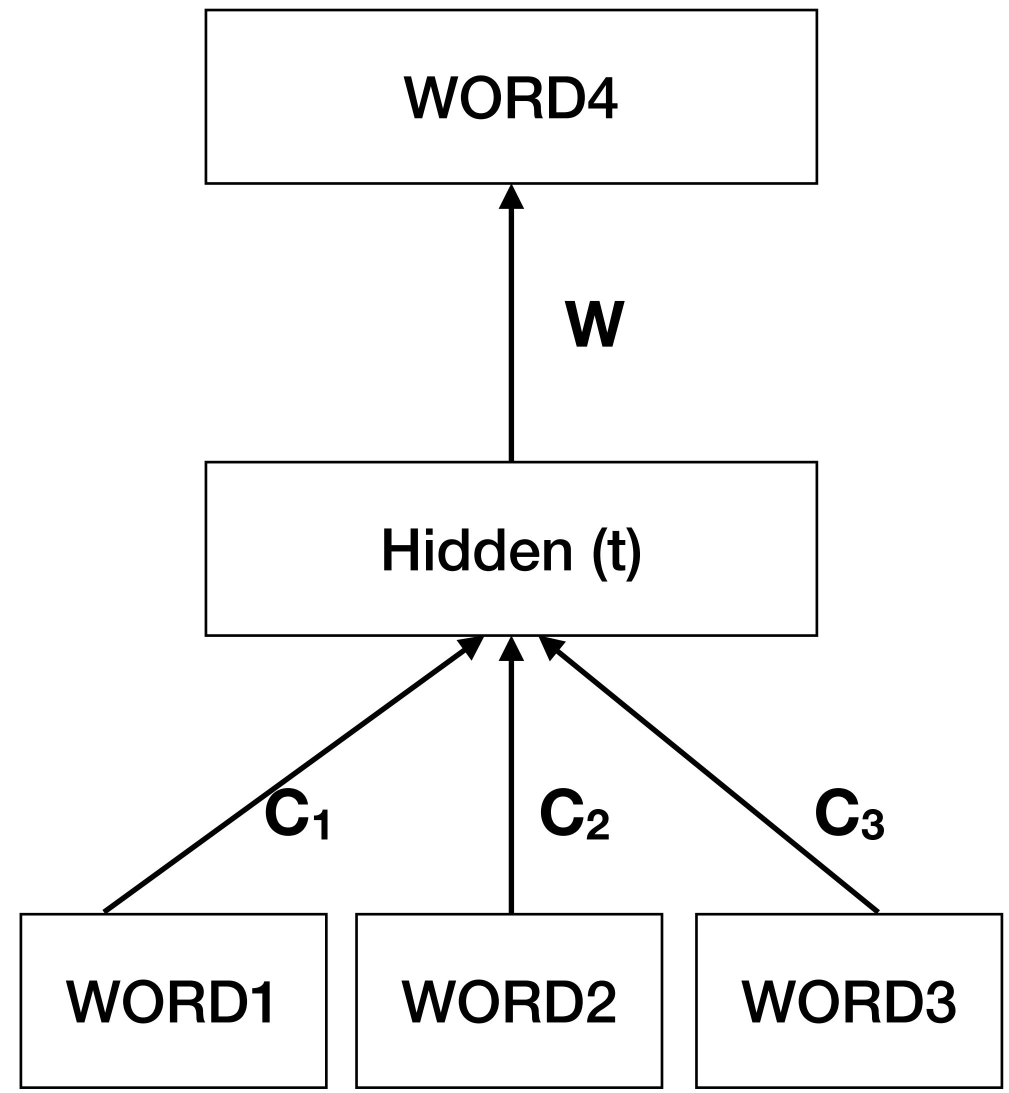


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

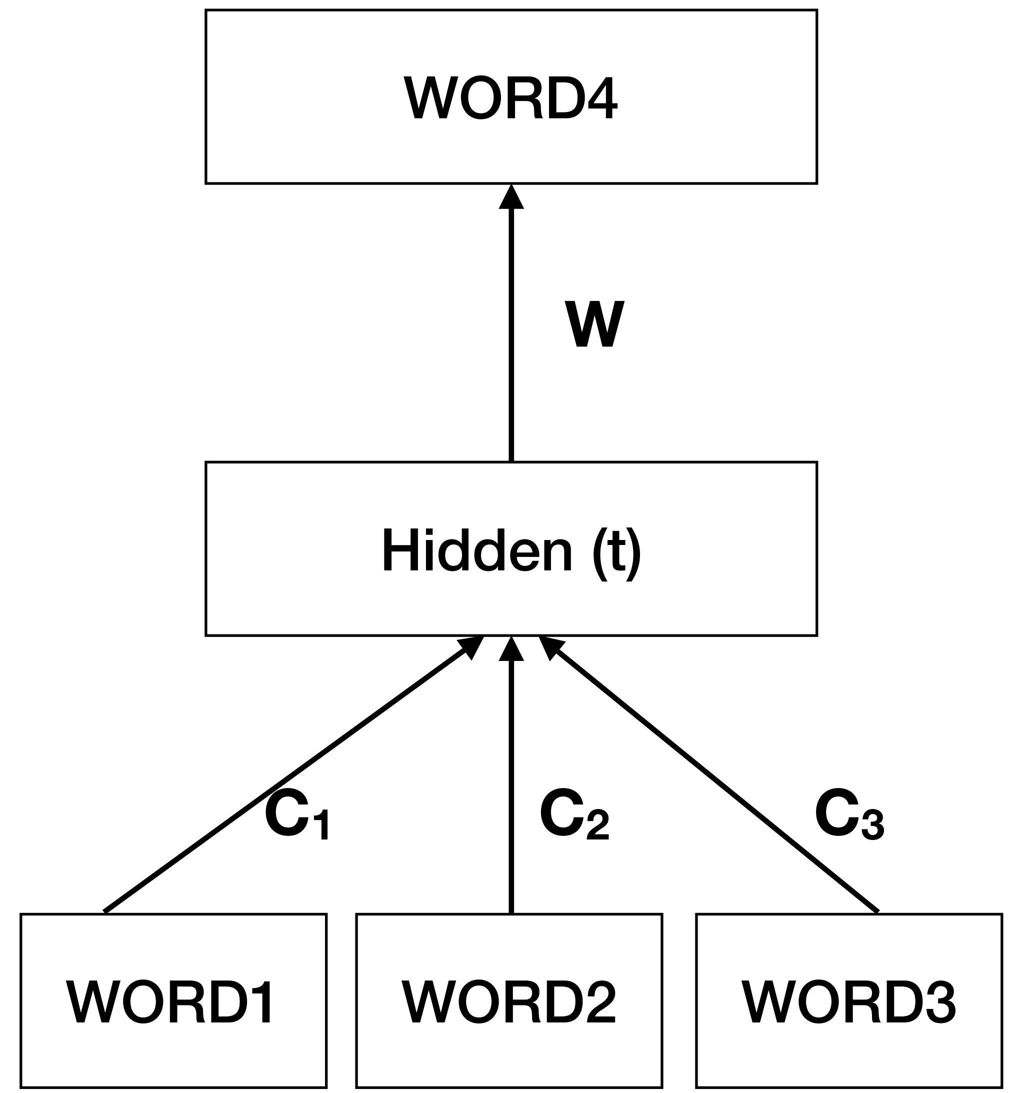
Feedforward Neural Network LM (FFNN)



FFNN

Feedforward Neural Network LM (FFNN)

- Loop through the entire corpus

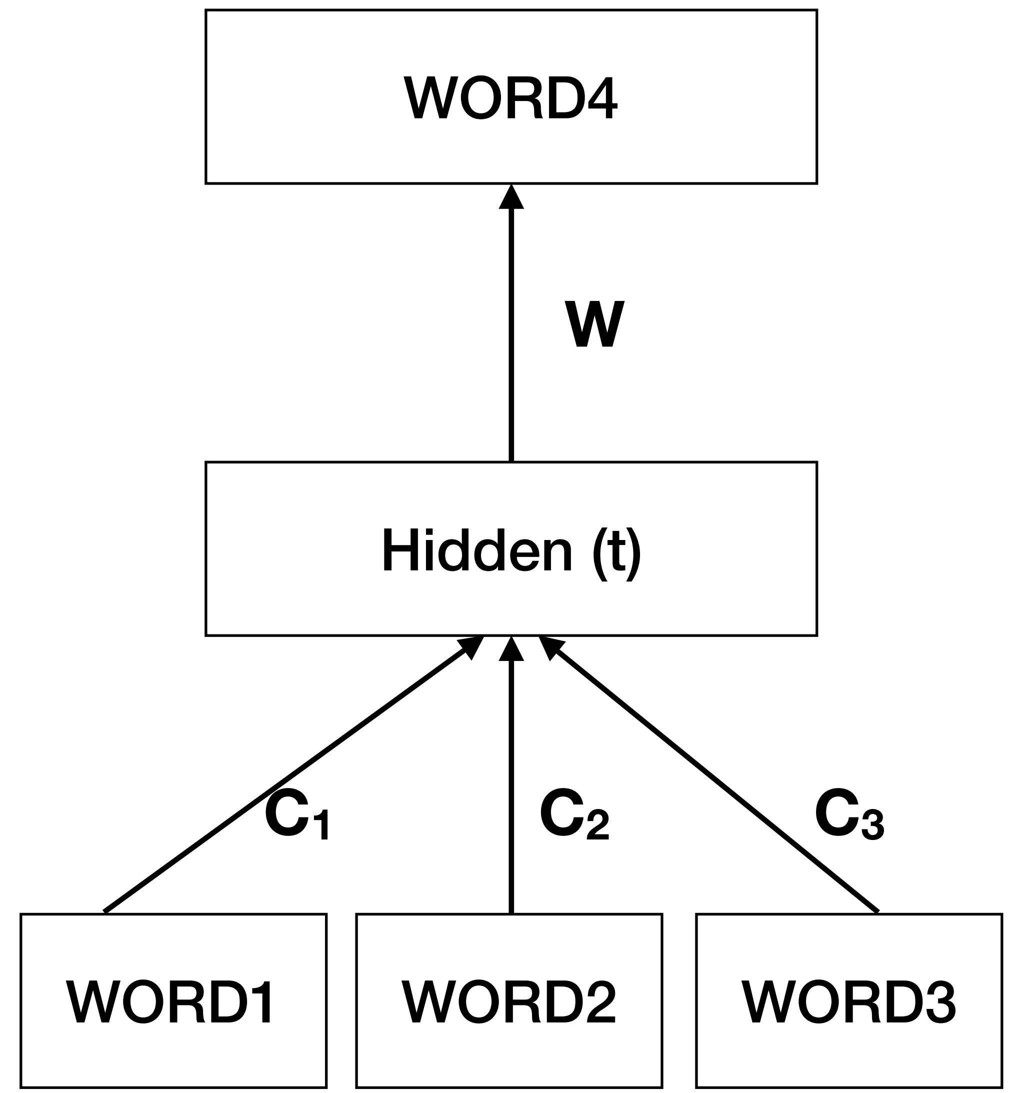


FFNN

Feedforward Neural Network LM (FFNN)

- Loop through the entire corpus

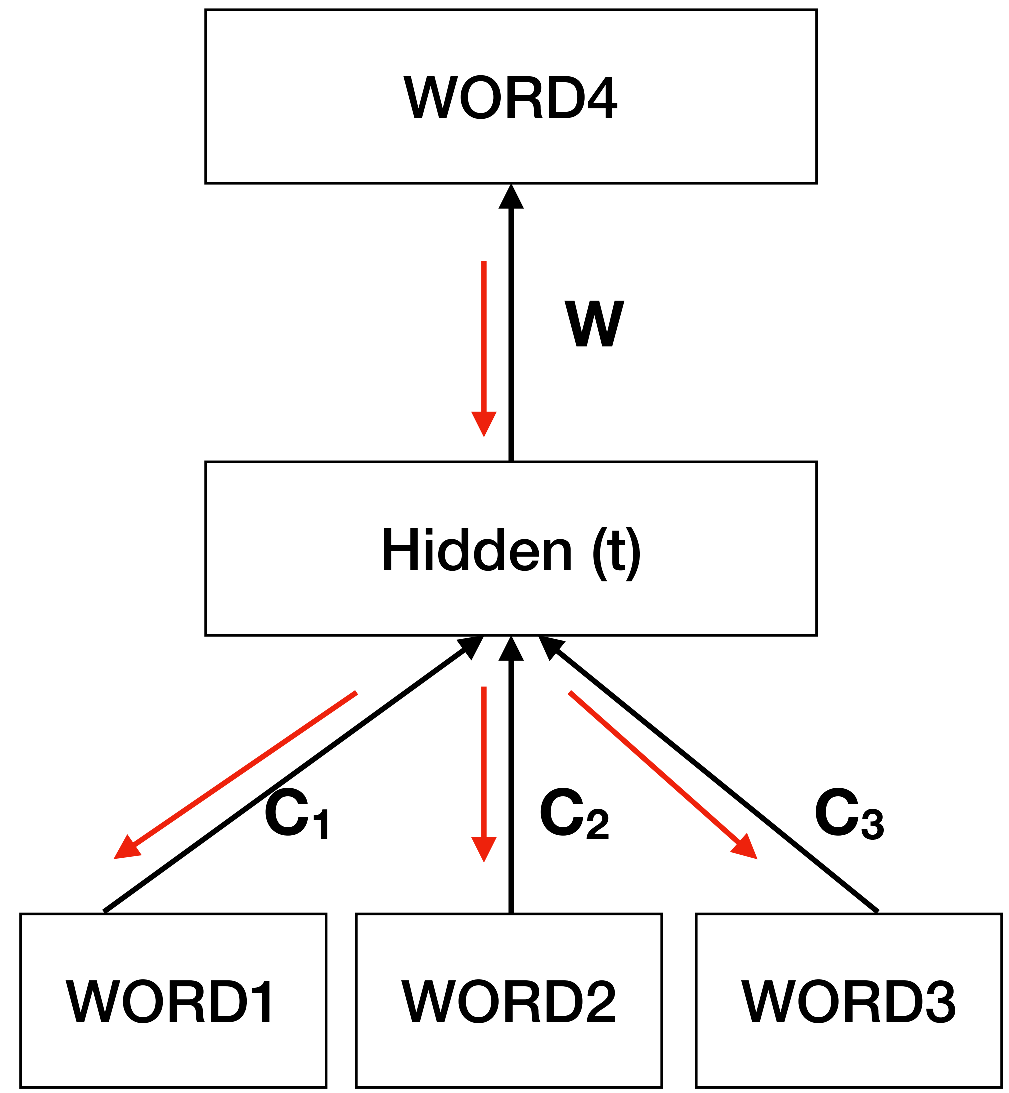
- Calculate error or loss (cross-entropy loss)



FFNN

Feedforward Neural Network LM (FFNN)

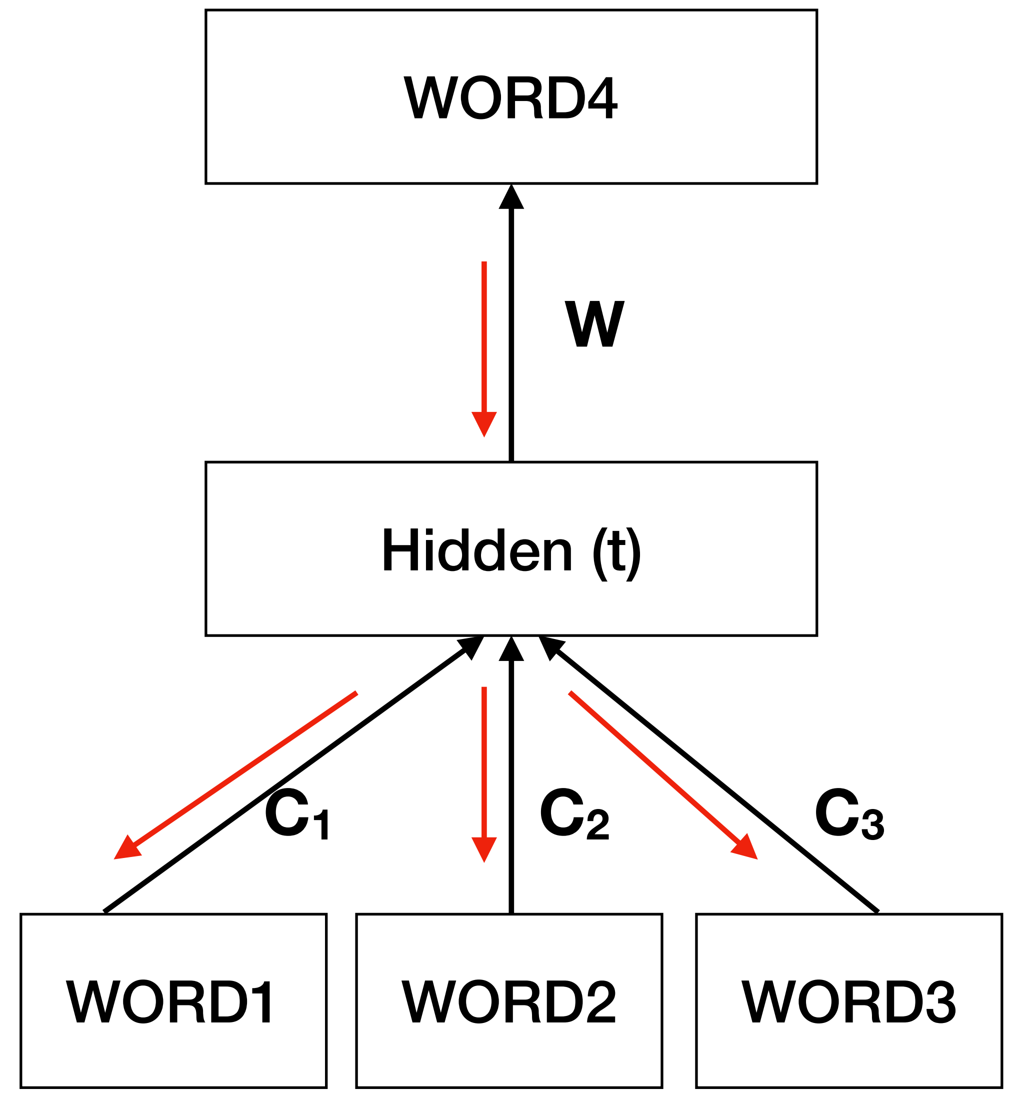
- Loop through the entire corpus
- Calculate error or loss (cross-entropy loss)
- Propagate the error through network to update the weight matrices



FFNN

Feedforward Neural Network LM (FFNN)

- Loop through the entire corpus
- Calculate error or loss (cross-entropy loss)
- Propagate the error through network to update the weight matrices
- Back Propagation



FFNN

Why NNs for LMs

Why NNs for LMs

The cat is walking in the bedroom

A dog was running in a room

Why NNs for LMs

The cat is walking in the bedroom

A dog was running in a room

=>

The cat is running in a room

A dog is walking in a bedroom

The dog was walking in the room

Why NNs for LMs

The cat is walking in the bedroom

A dog was running in a room

=>

The cat is running in a room

A dog is walking in a bedroom

The dog was walking in the room

- NNLM generalizes in such a way that **similar** words have **similar** vectors

Why NNs for LMs

The cat is walking in the bedroom

A dog was running in a room

=>

The cat is running in a room

A dog is walking in a bedroom

The dog was walking in the room

- NNLM generalizes in such a way that **similar** words have **similar** vectors
- Presence of only one such sentence in the training set helps improve the probability of its combinations

Types of NNLM

- Feedforward Neural Network Language Model
- Recurrent Neural Network Language Model
- Long-Short Term Memory LM
- Transformer-based LM
- ..

NNLM: Questions

- What might be some challenges that you might face while training or applying NNLMs?

NNLMs Challenges

NNLMs Challenges

- Long-Range Dependencies

NNLMs Challenges

- Long-Range Dependencies
- Training Speed

NNLMs Challenges

- Long-Range Dependencies
- Training Speed
- On-disk Size

NNLMs Challenges

- Long-Range Dependencies
- Training Speed
- On-disk Size
- Rare Context

NNLMs Challenges

- Long-Range Dependencies
- Training Speed
- On-disk Size
- Rare Context
- …

Feedforward: Long-term information

- “I grew up in France… I speak fluent ________.”

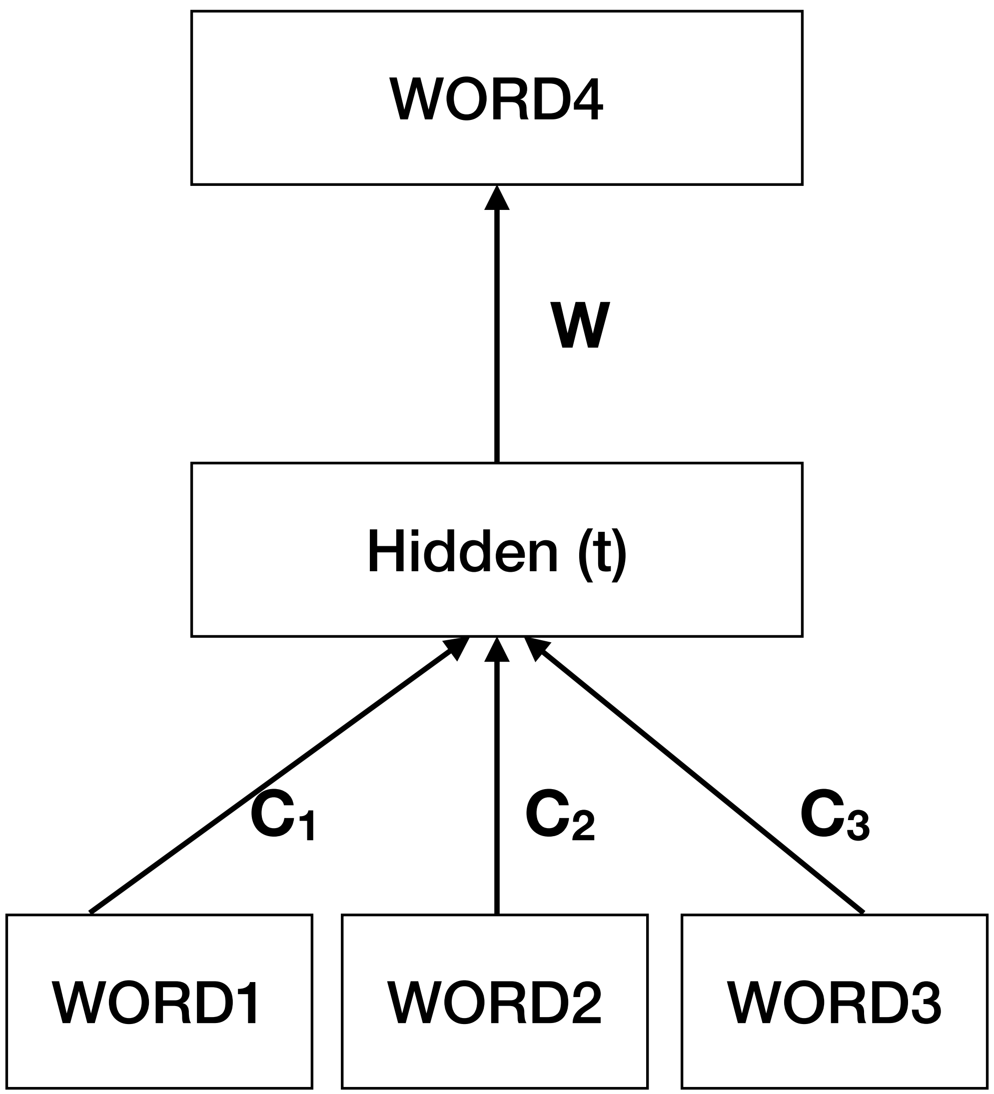
Feedforward: Long-term information

- “I grew up in France… I speak fluent ___*French*___.”

Feedforward: Long-term information

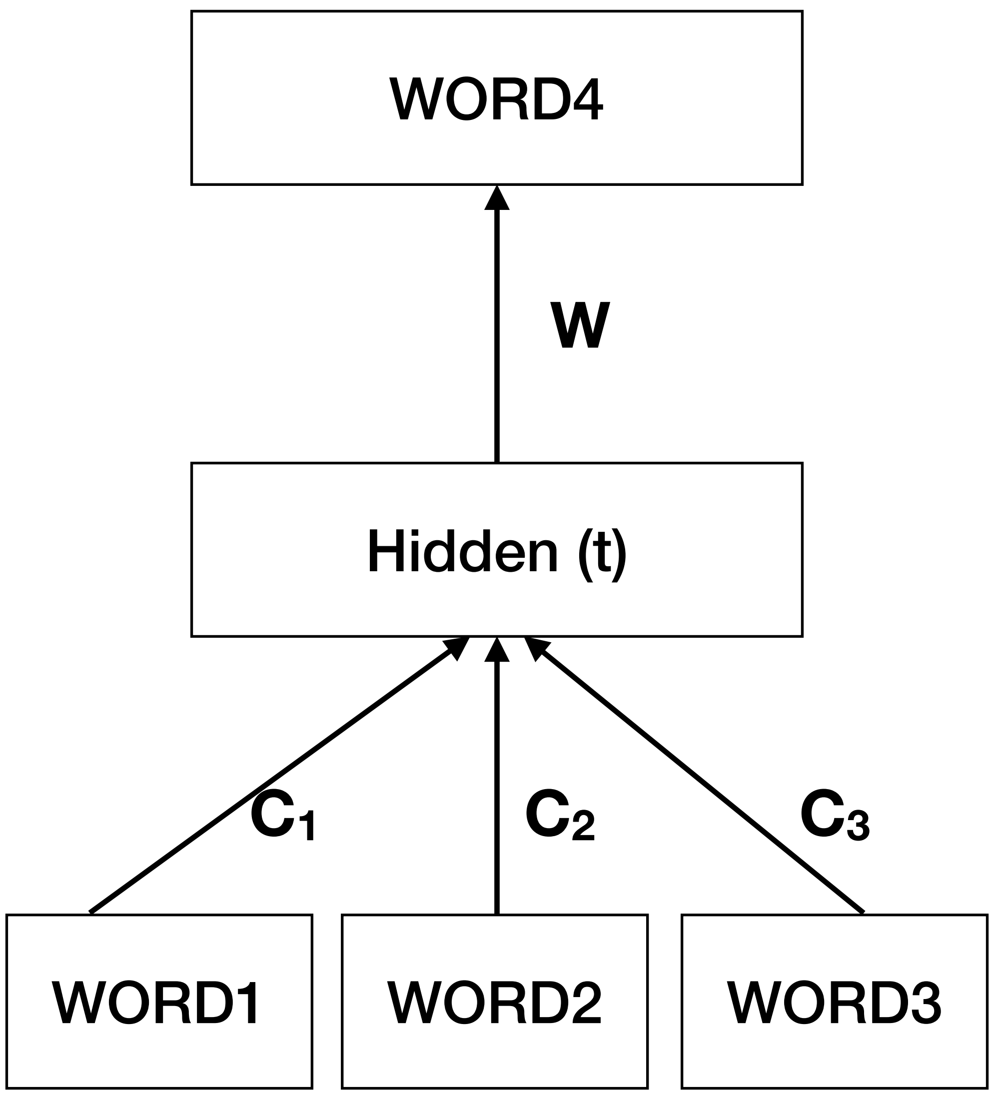
- “I grew up in France… I speak fluent _*French*_.”
- Feedforward Neural Network (FFNN) has limited context size

Recurrent Neural Networks (RNN)

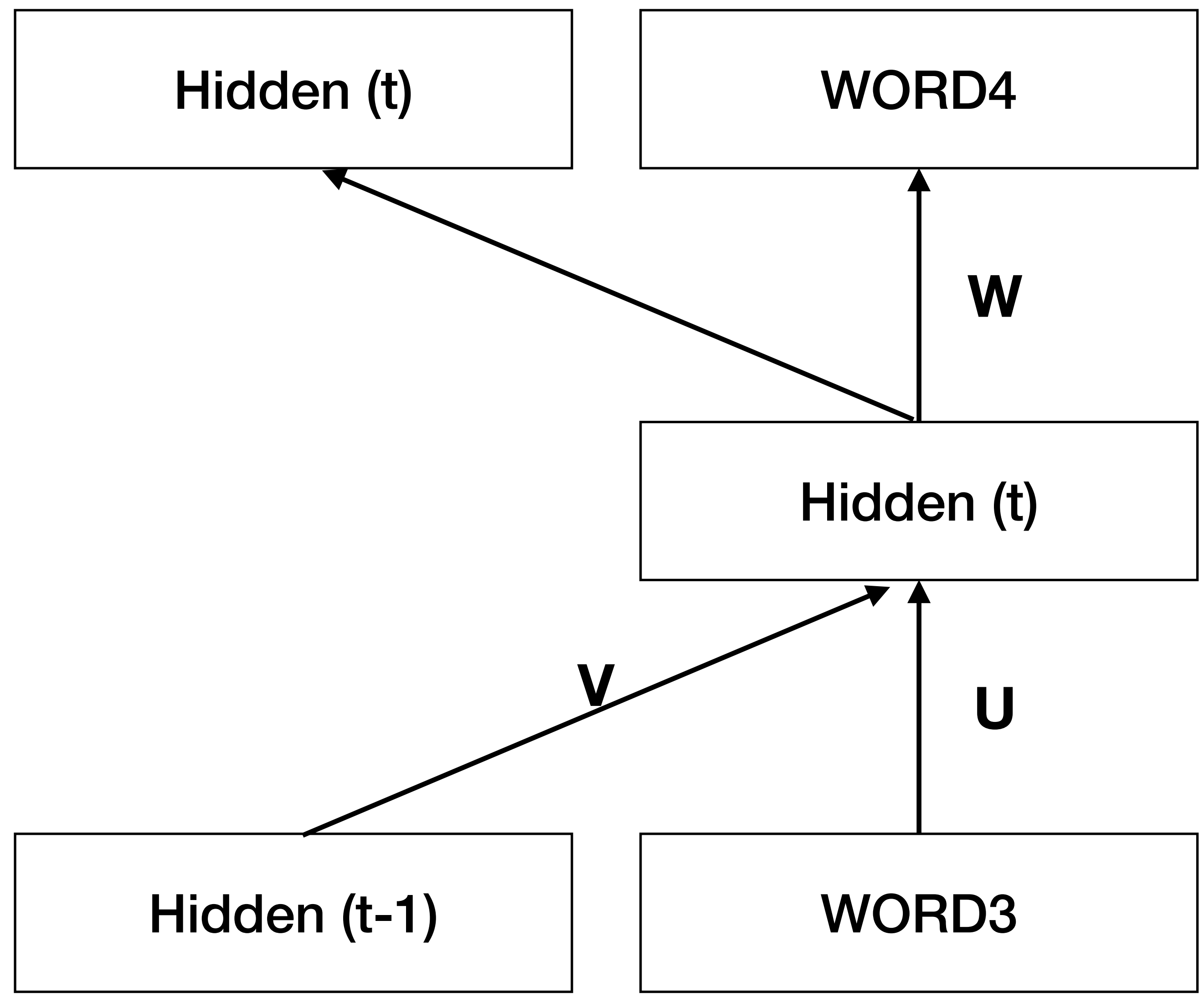


FFNN

Recurrent Neural Networks (RNN)

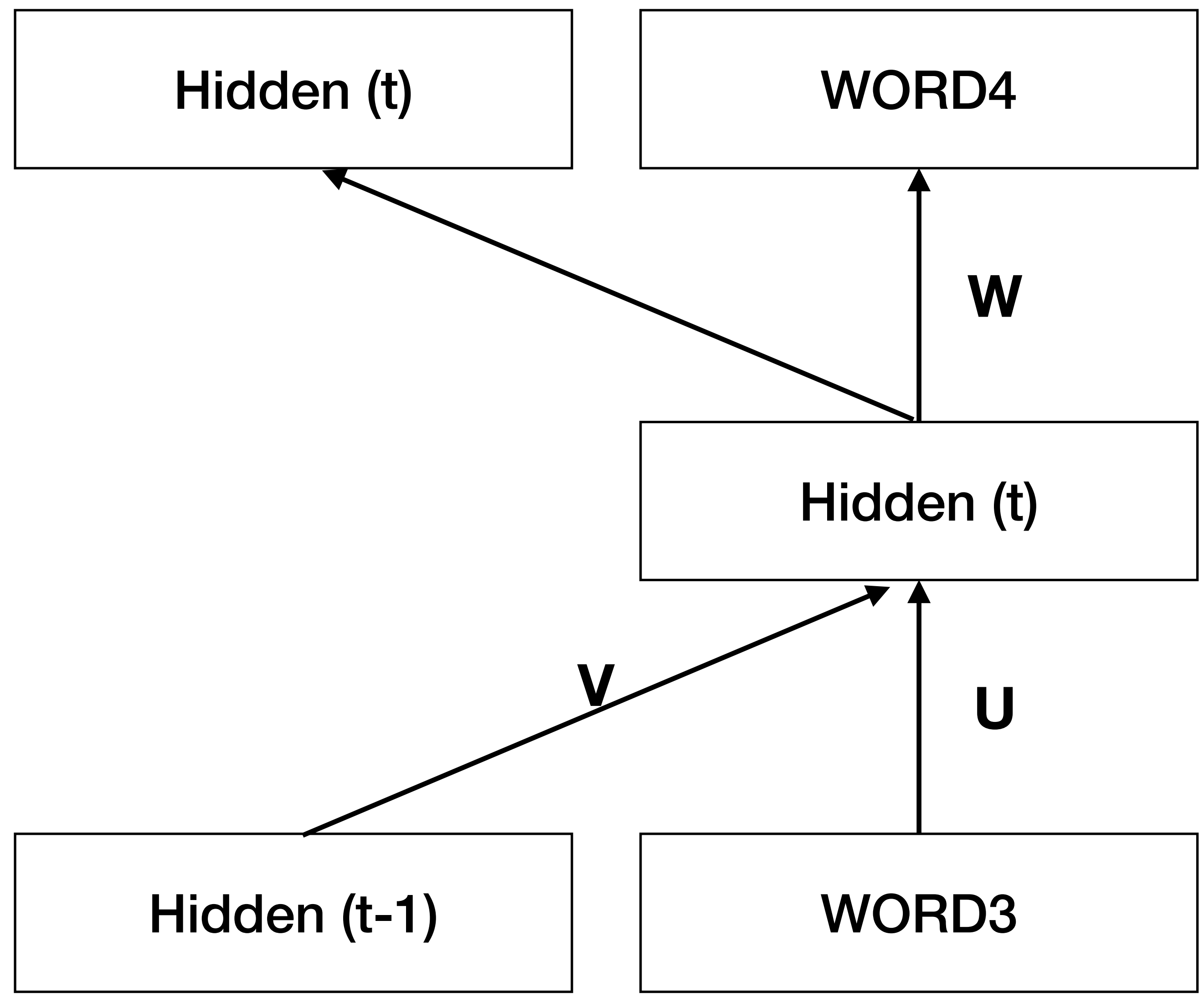


FFNN



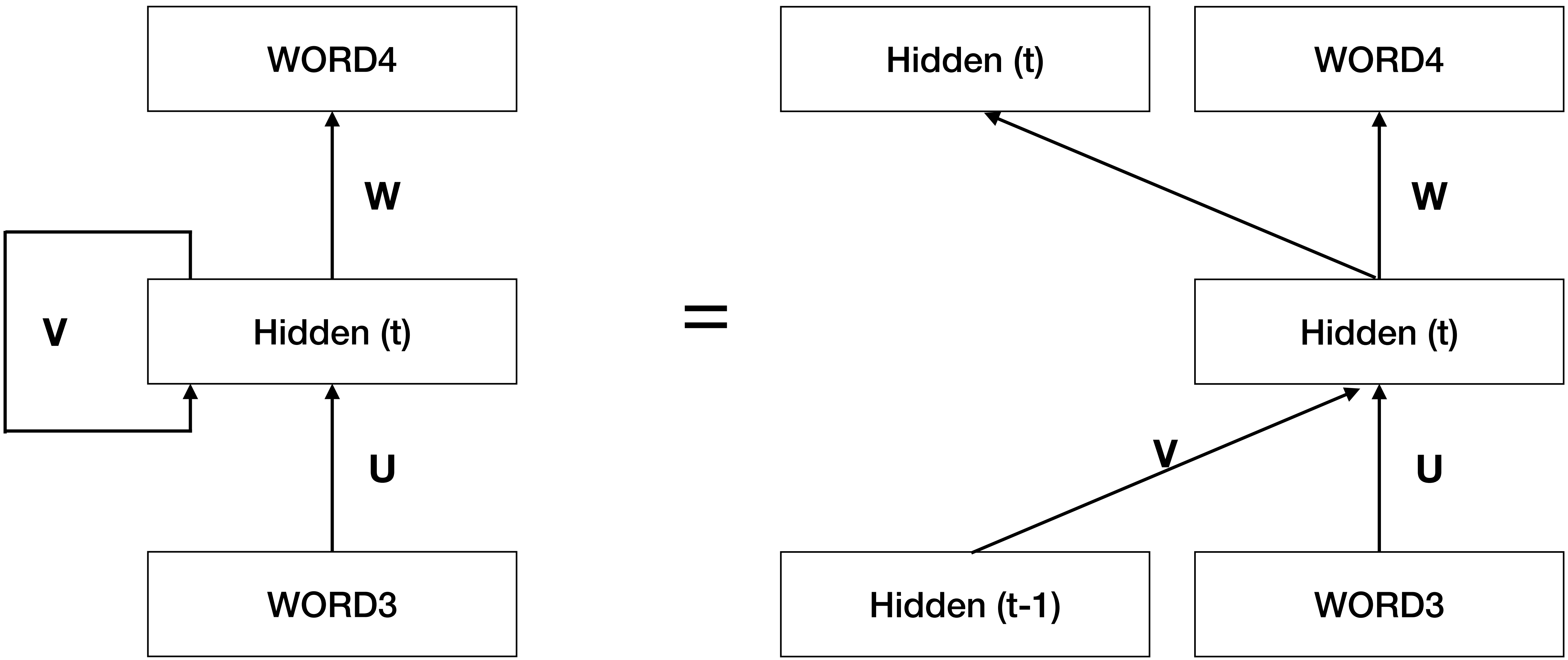
RNN

Recurrent Neural Networks (RNN)



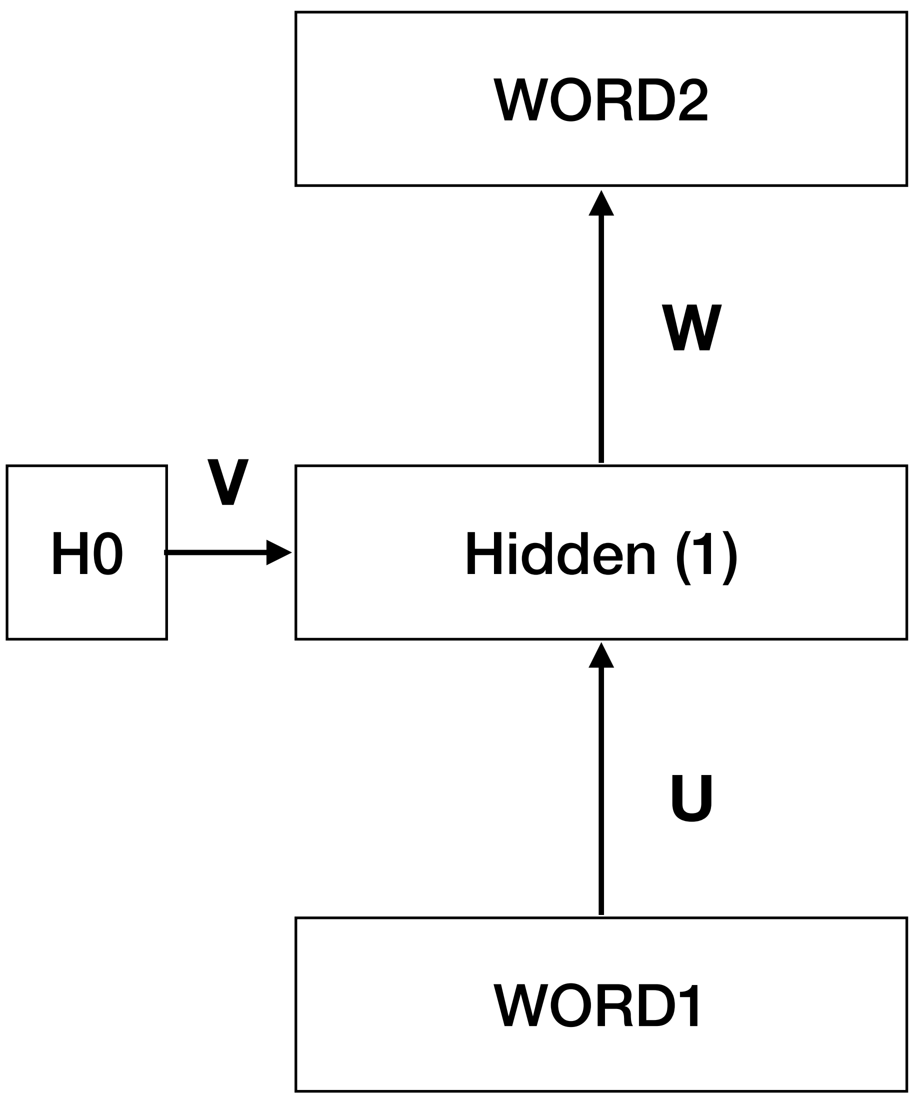
RNN

Recurrent Neural Networks (RNN)

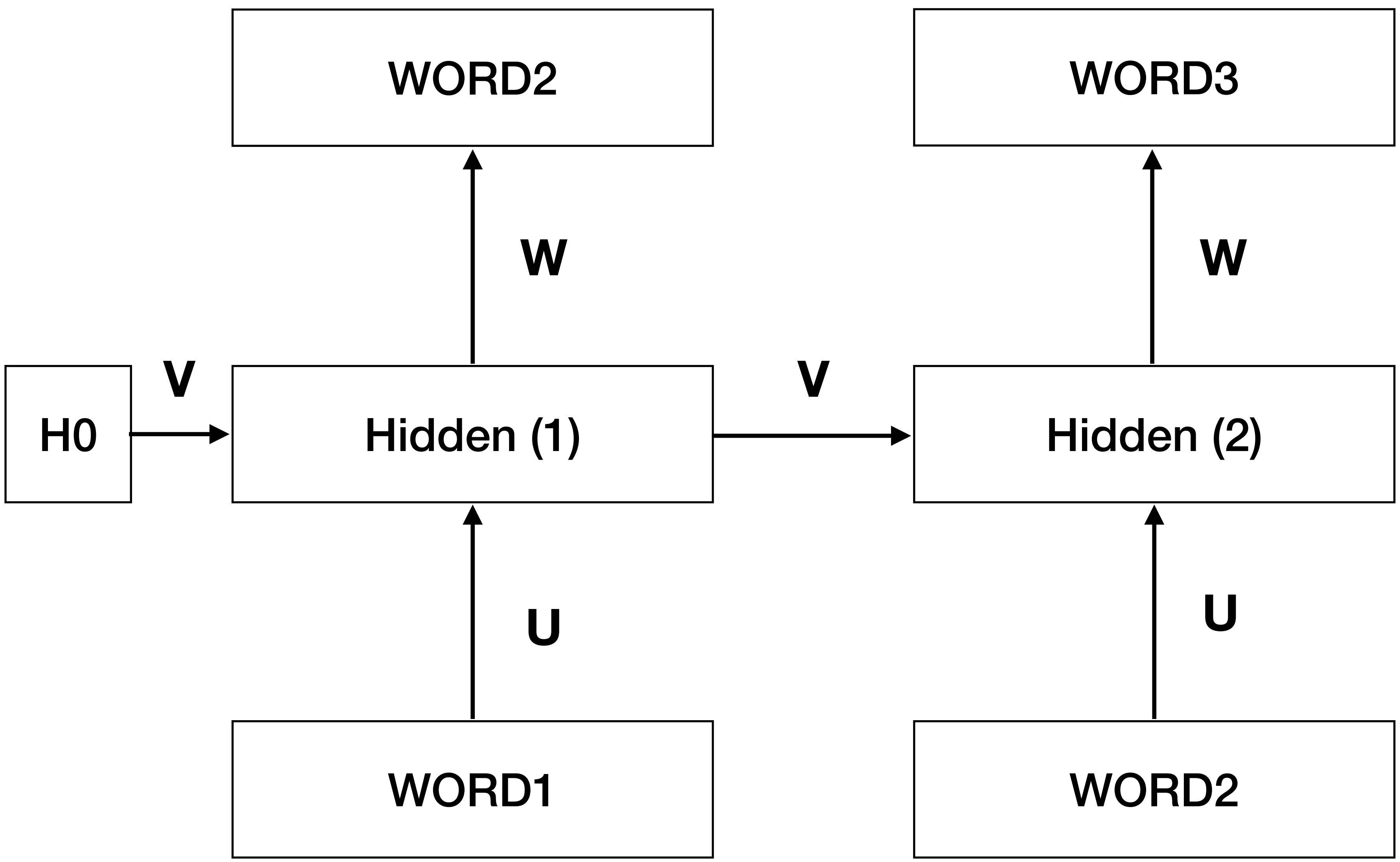


RNN

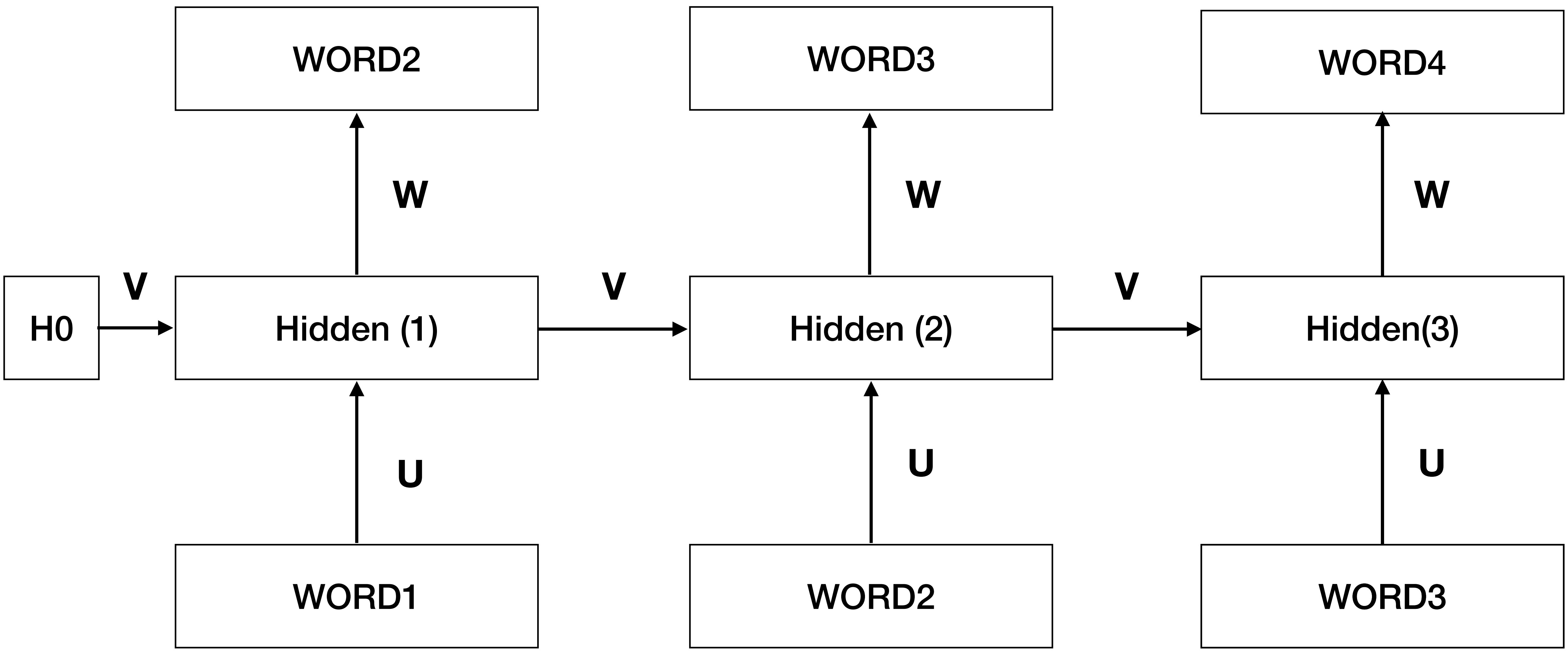
RNN: Timestep 1



RNN: Timestep 2



RNN: Timestep 3



Theoretically information from first step is available to the present timestep

RNN

- "I grew up in France… I speak fluent _*French*_."

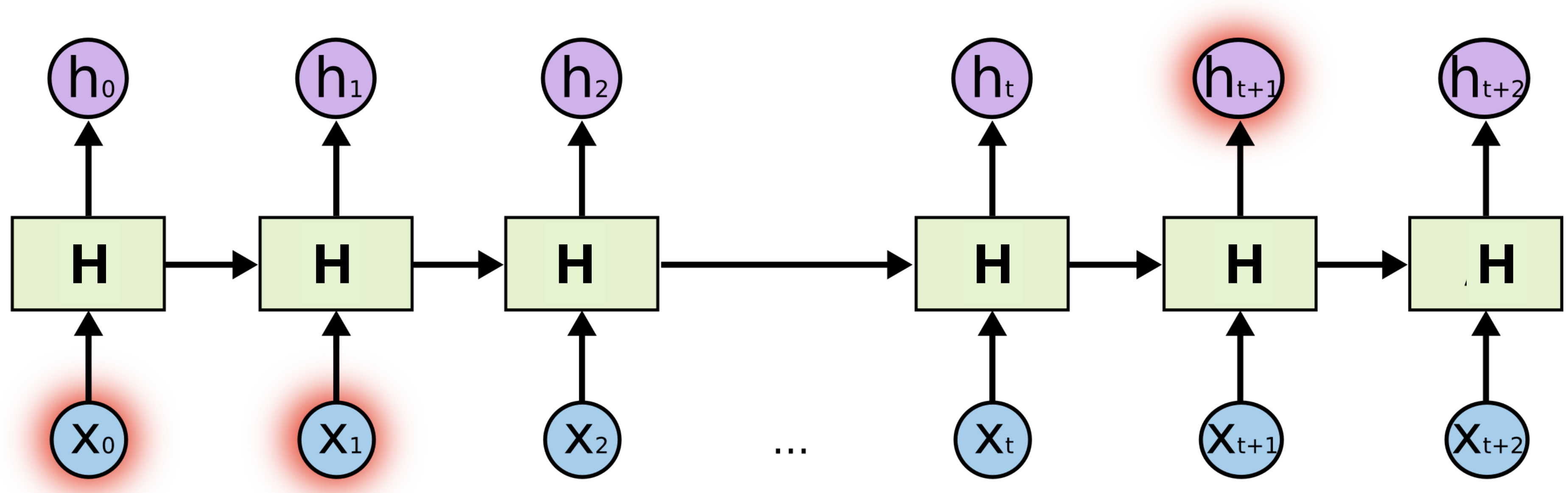


Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

- "I grew up in France… I speak fluent _French_."

- As the gap grows, RNNs become unable to learn to connect information

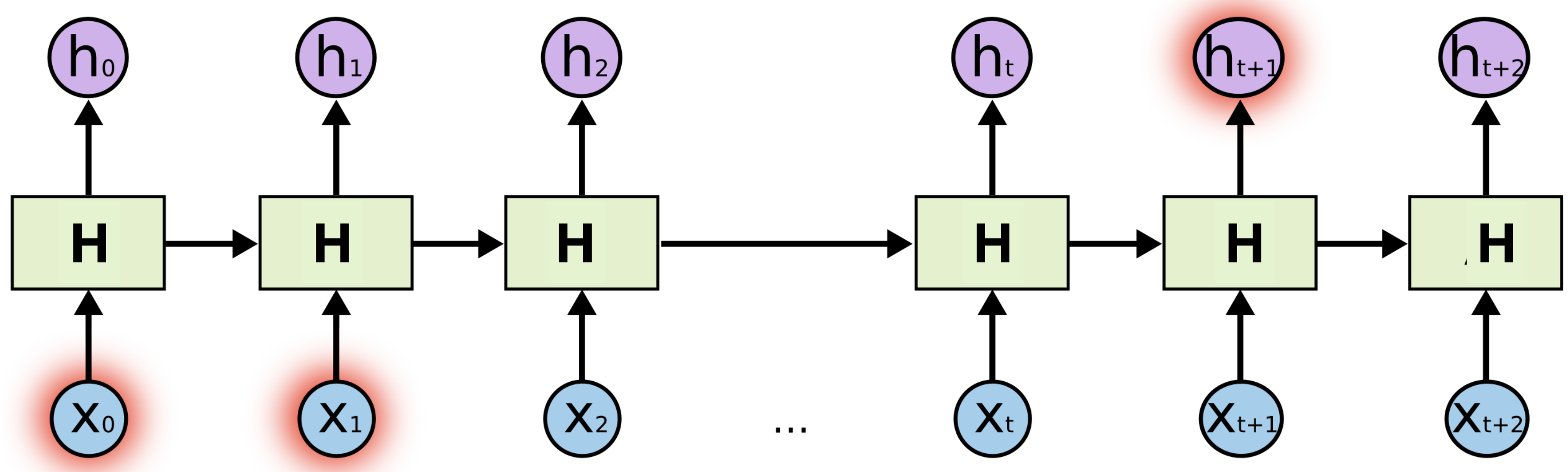
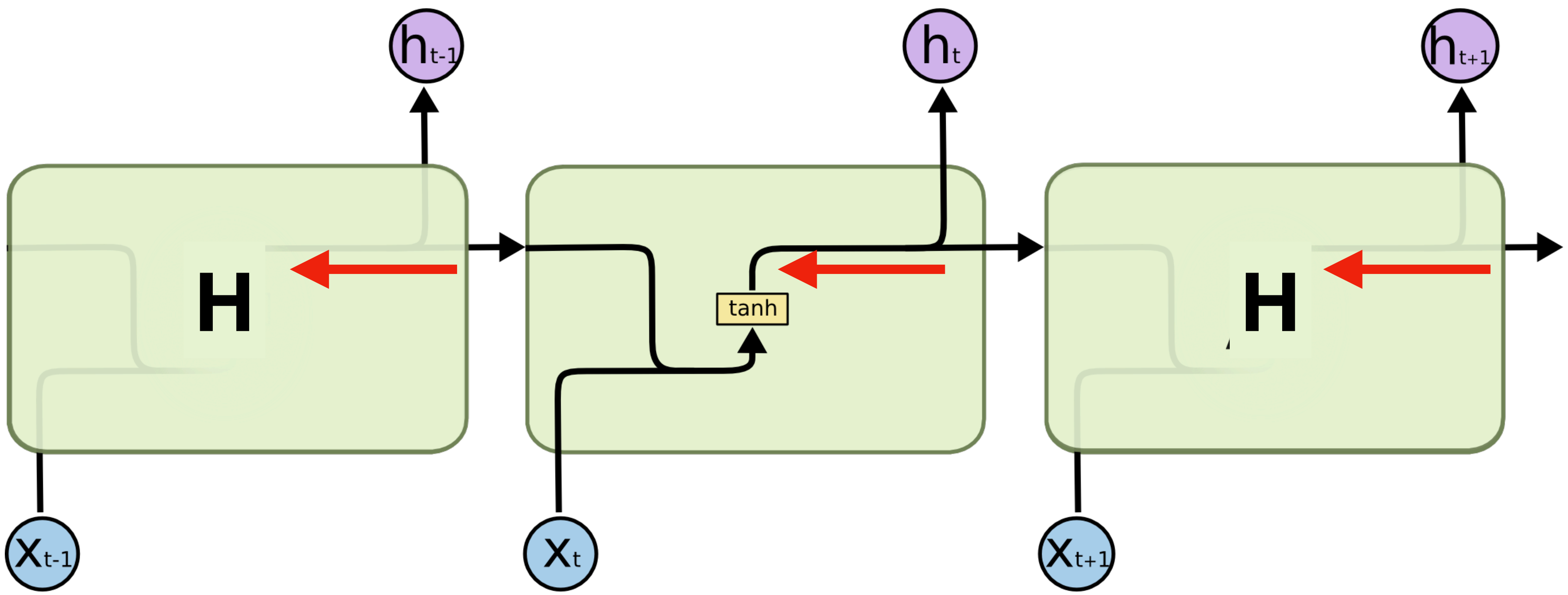


Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN



- Error (red arrow) is passed through a chain of hidden states
- Error passing through multiple of these functions can vanish

Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problems with RNN

- The main problem with RNNs is that gradients less than 1 become exponentially small over time

Problems with RNN

- The main problem with RNNs is that gradients less than 1 become exponentially small over time
- Known as the vanishing gradient problem

Problems with RNN

- The main problem with RNNs is that gradients less than 1 become exponentially small over time
- Known as the vanishing gradient problem
- Gradients greater than 1 become exponentially large over time (the exploding gradient problem)*

* The exploding gradient problem can be alleviated by clipping large gradient values to some maximum number

Problems with RNN

- The main problem with RNNs is that gradients less than 1 become exponentially small over time

- Known as the vanishing gradient problem

- Gradients greater than 1 become exponentially large over time (the exploding gradient problem)*

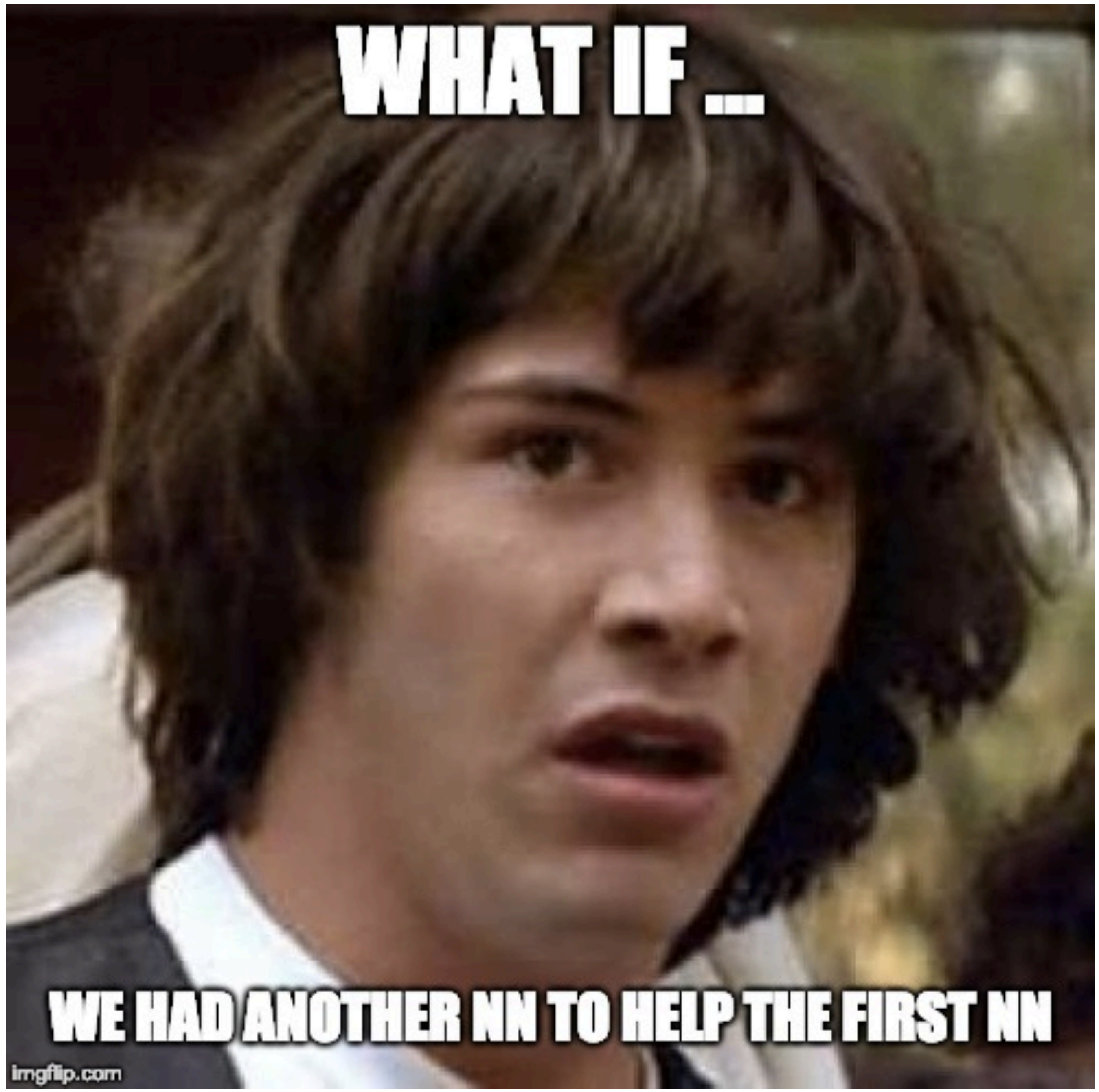
- This leads to training instability, and bad results

* The exploding gradient problem can be alleviated by clipping large gradient values to some maximum number

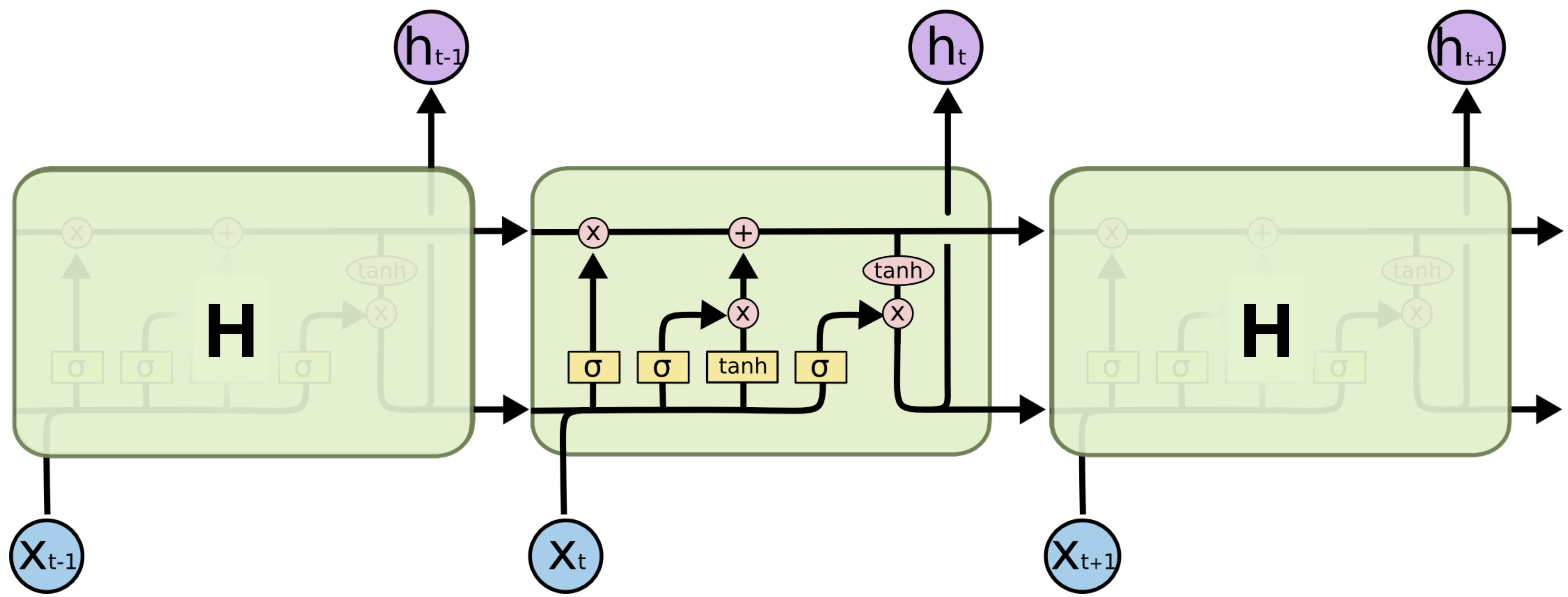
Problems with RNN

- The main problem with RNNs is that gradients less than 1 become exponentially small over time
- Known as the vanishing gradient problem
- Gradients greater than 1 become exponentially large over time (the exploding gradient problem)*
- This leads to training instability, and bad results
- Sequence Modeling: https://www.deeplearningbook.org/contents/rnn.html

* The exploding gradient problem can be alleviated by clipping large gradient values to some maximum number


WHAT IF ...
WE HAD ANOTHER NN TO HELP THE FIRST NN
imgflip.com

Long-Short Term Memory

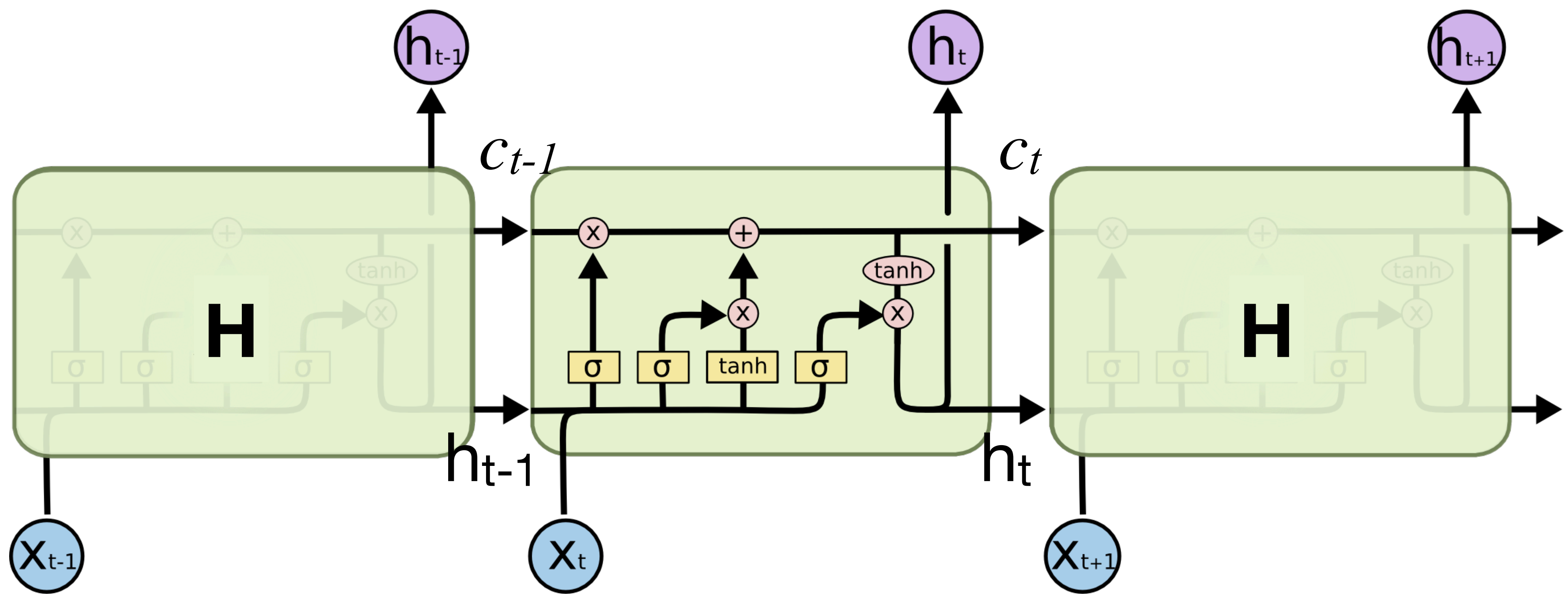


- Lets add another neural network help the first network learn long-distance relationships

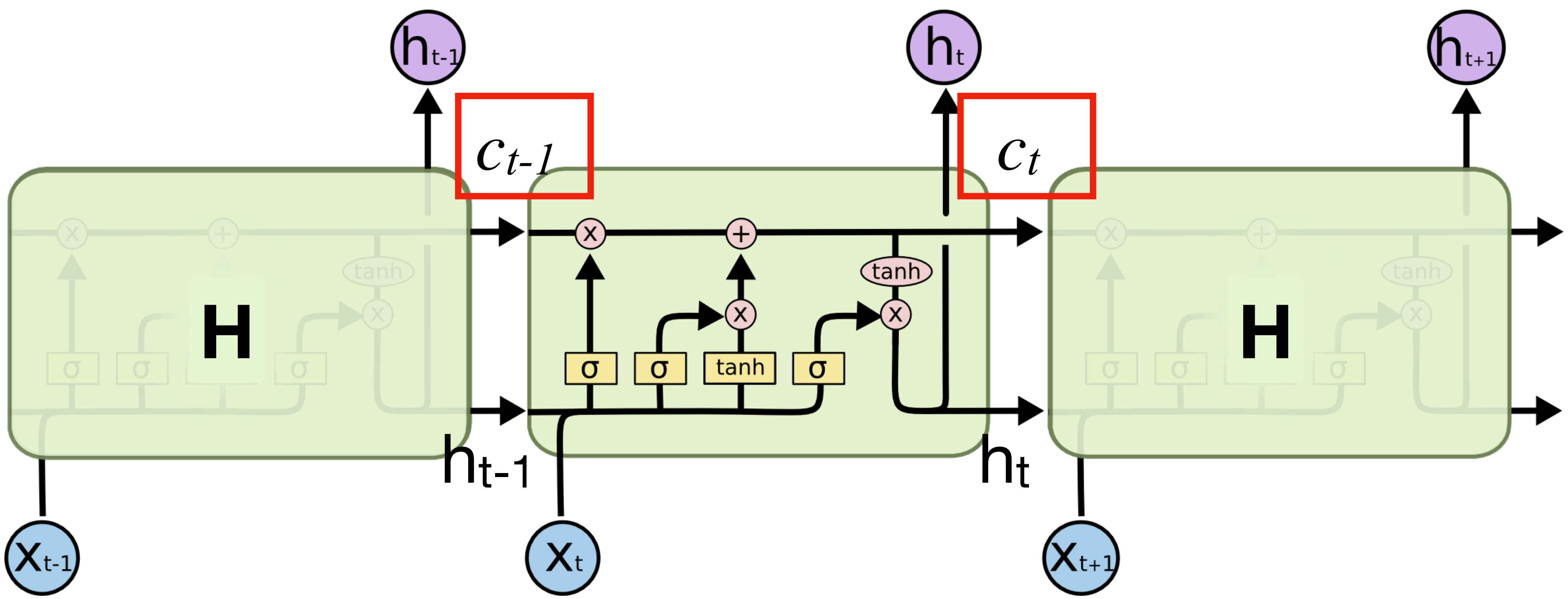
- That's basically what we do when we add more weight matrices to a neural network

Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: States

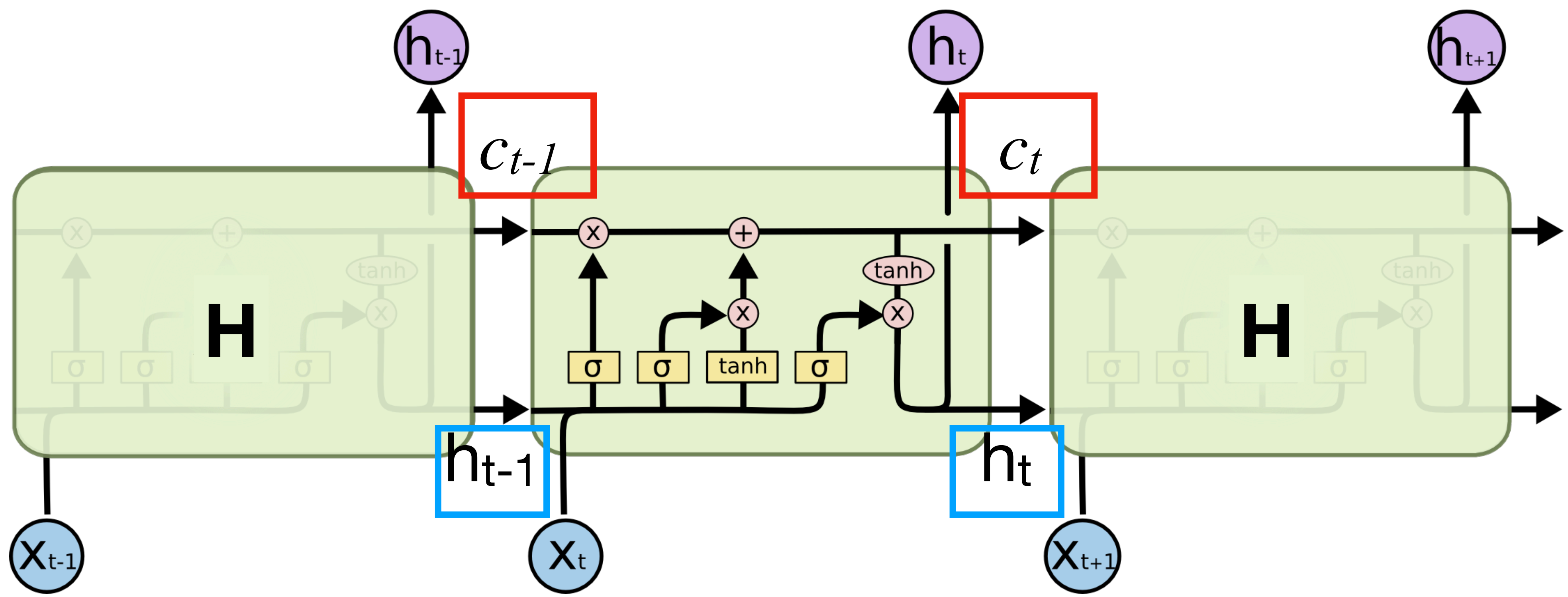


LSTM: States

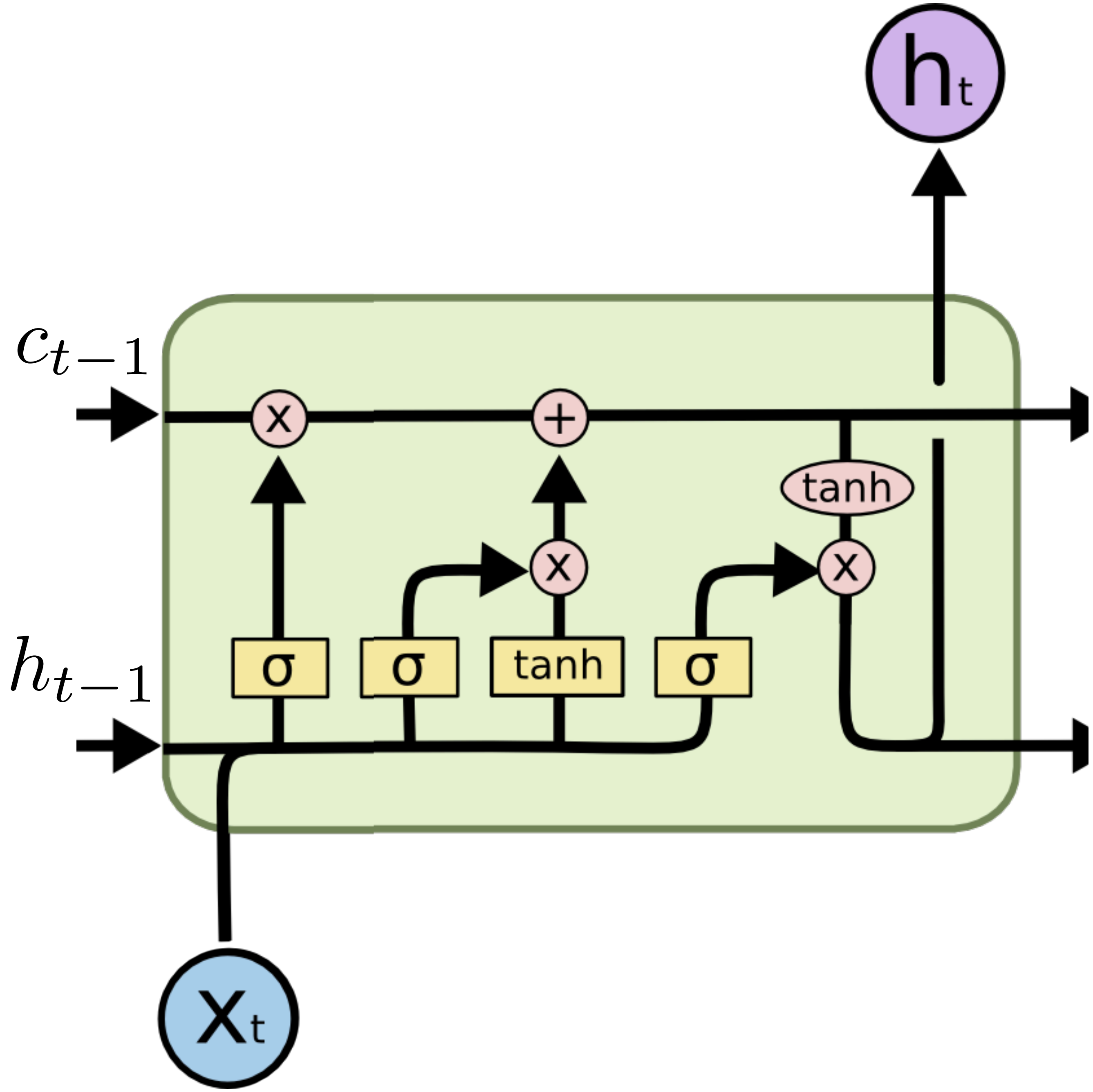


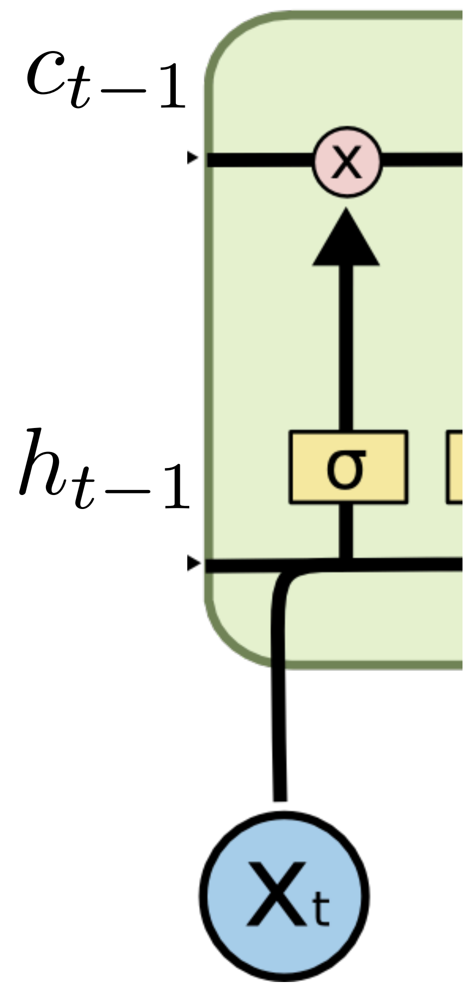
- Global State $\boxed{c}$ captures global information at the document/ sentence level

LSTM: States

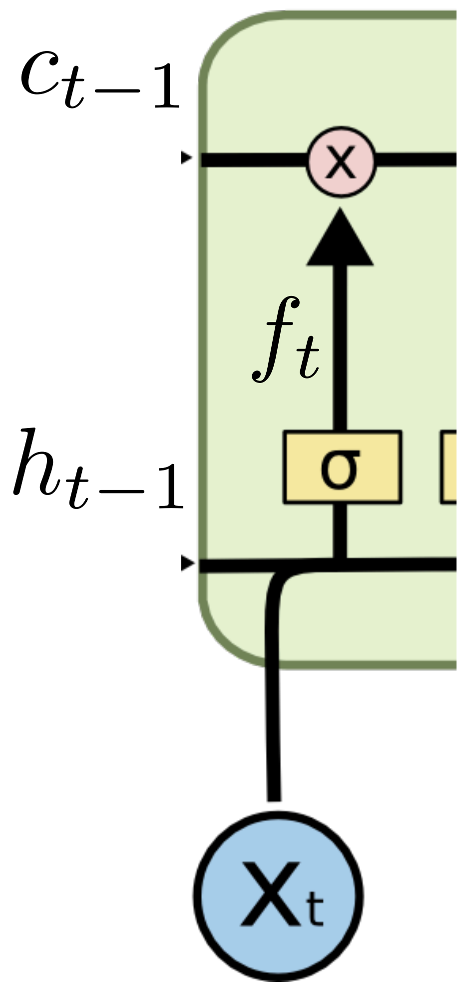


- Global State $\boxed{c}$ captures global information at the document/ sentence level
- LSTM hidden state $\boxed{h_t}$ interacts with this global state to predict the next word


c_{t-1}
h_{t-1}
h_t
x
+
tanh
x
x
σ
σ
tanh
σ
x_t

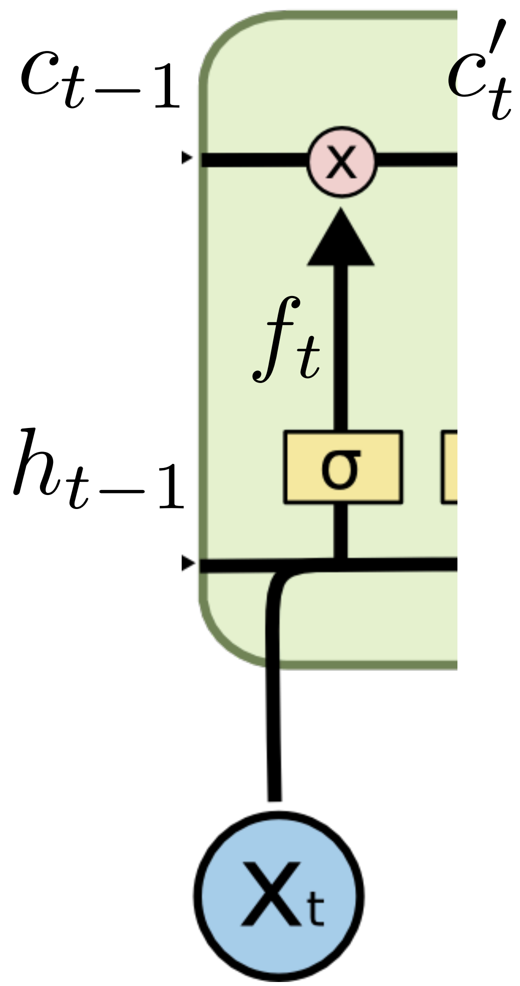


σ	sigmoid function
w_x	weight of the respective gate(x)
b_x	bias of the respective gate(x)
h_{t-1}	output of the previous LSTM
x_t	input at current timestamp



$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

σ	sigmoid function
w_x	weight of the respective gate(x)
b_x	bias of the respective gate(x)
h_{t-1}	output of the previous LSTM
x_t	input at current timestamp



$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

σ sigmoid function

w_x weight of the respective gate(x)

b_x bias of the respective gate(x)

h_{t-1} output of the previous LSTM

x_t input at current timestamp

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

- weights and bias

  $w_f = [1 \quad 1]$

  $b_f = 0$
- σ : sigmoid fn * : pointwise multiplication
- "," is vector concatenation
- h_{t-1} = [1], c_{t-1} = [2], x_t = [0.2]
- calculate: c'_t

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

- weights and bias

  $w_f = [1 \quad 1]$

  $b_f = 0$

- σ : sigmoid fn * : pointwise multiplication
- h_{t-1} = [1], c_{t-1} = [2], x_t = [0.2]
- calculate: c'_t

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

- weights and bias

$$w_f = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$b_f = 0$$

- σ : sigmoid fn * : pointwise multiplication

- h_{t-1} = [1], c_{t-1} = [2], x_t = [0.2]

- calculate: c'_t

$$w_f[h_{t-1}, x_t] + b_f = \begin{bmatrix} 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0.2 \end{bmatrix} = [1.2]$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

- weights and bias

$$w_f = [1 \quad 1]$$

$$b_f = 0$$

- σ : sigmoid fn * : pointwise multiplication
- h_{t-1} = [1], c_{t-1} = [2], x_t = [0.2]
- calculate: c'_t

$$w_f[h_{t-1}, x_t] + b_f = [1 \quad 1] \times \begin{bmatrix} 1 \\ 0.2 \end{bmatrix} = [1.2]$$

$$f_t = [\sigma(1.2)] = [0.77]$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$c'_t = c_{t-1} * f_t$$

- weights and bias

$$w_f = [1 \quad 1]$$

$$b_f = 0$$

- σ : sigmoid fn * : pointwise multiplication

- h_{t-1} = [1], c_{t-1} = [2], x_t = [0.2]

- calculate: c'_t

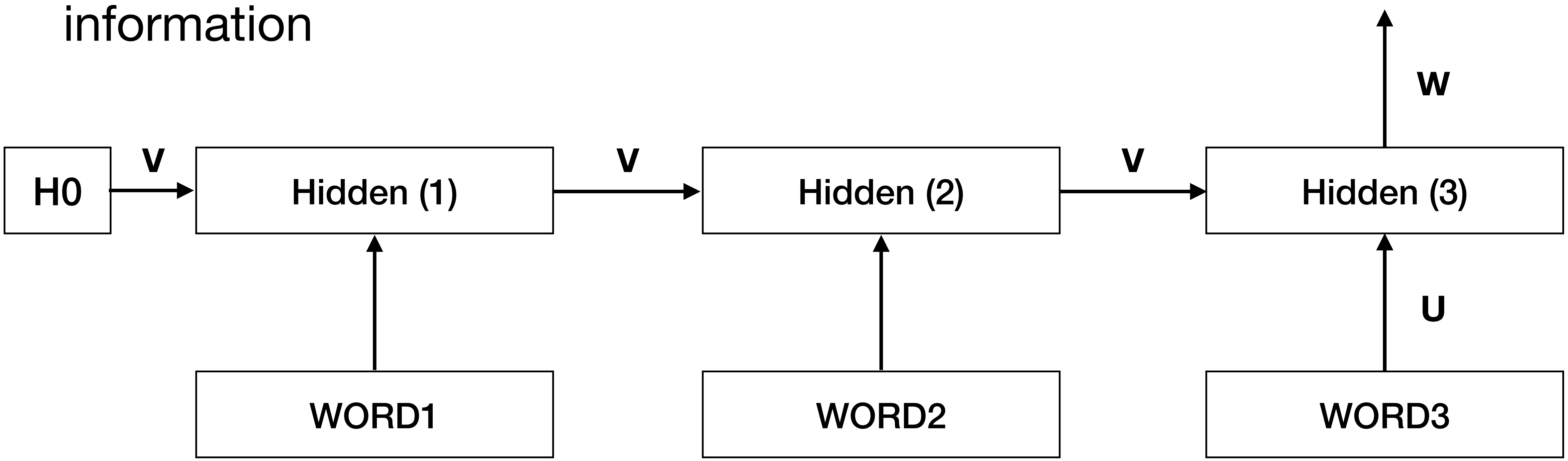
$$w_f[h_{t-1}, x_t] + b_f = [1 \quad 1] \times \begin{bmatrix} 1 \\ 0.2 \end{bmatrix} = [1.2]$$

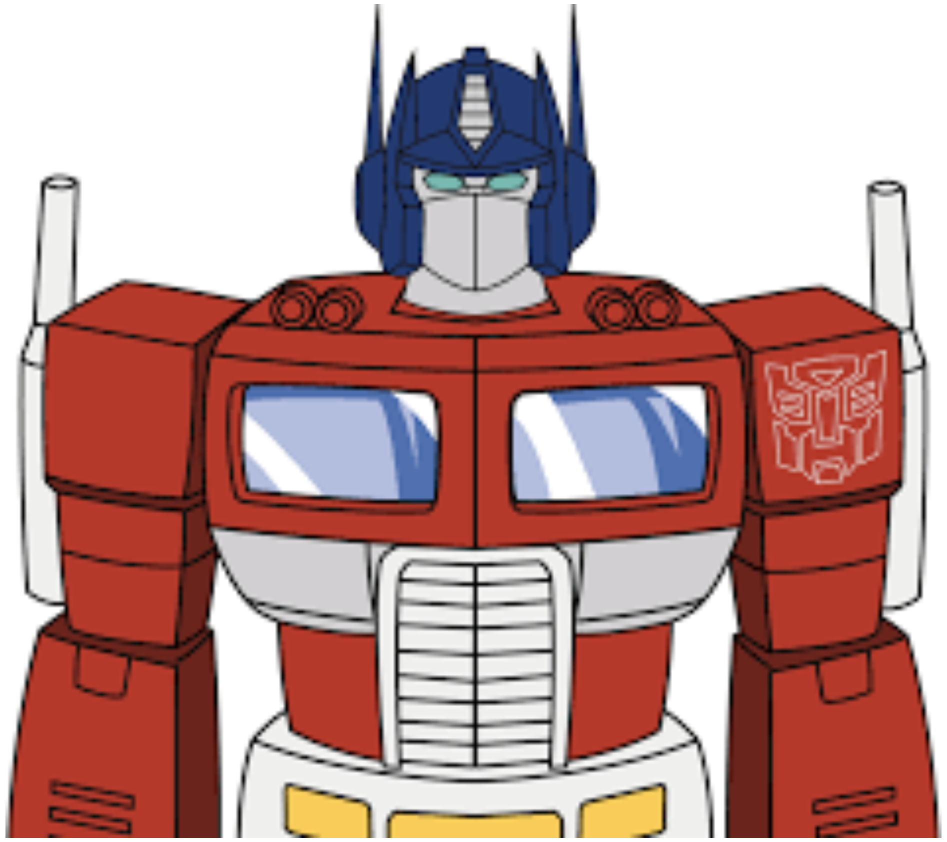
$$f_t = [\sigma(1.2)] = [0.77]$$

$$c'_t = c_{t-1} * f_t = [2] * [0.77] = [1.54]$$

LSTM Problems

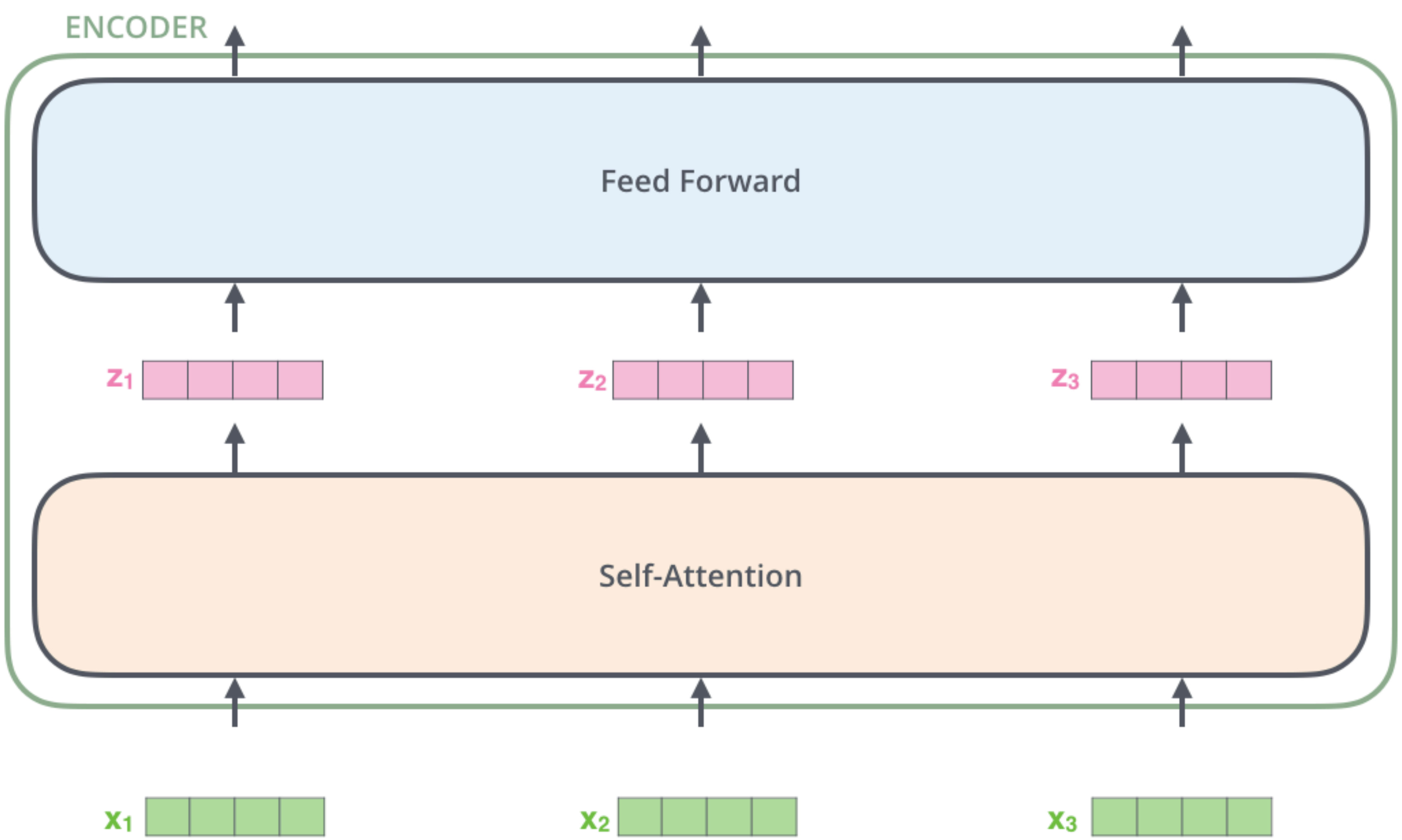
- Forget gate: removes information from the Global Cell state (C)
  - this information might be be useful at a later stage
- Implicit representation of long-term information
  - Cell state and previous hidden state summarise the prior information





Transformers for Language Modelling

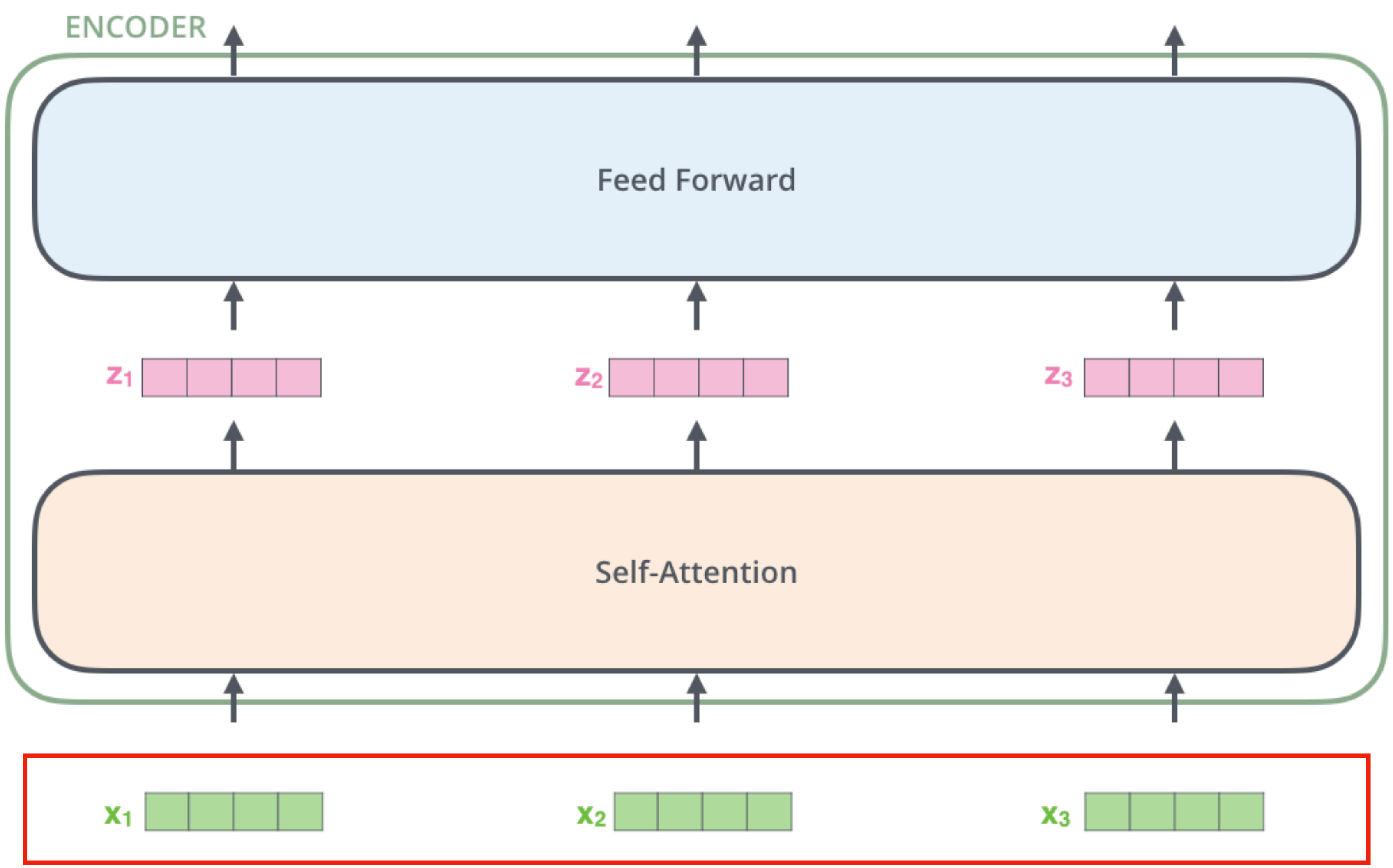
Transformers: Simplified



Multiple (50-90) such layers in a Transformer LM

Credit: http://jalammar.github.io/illustrated-transformer/

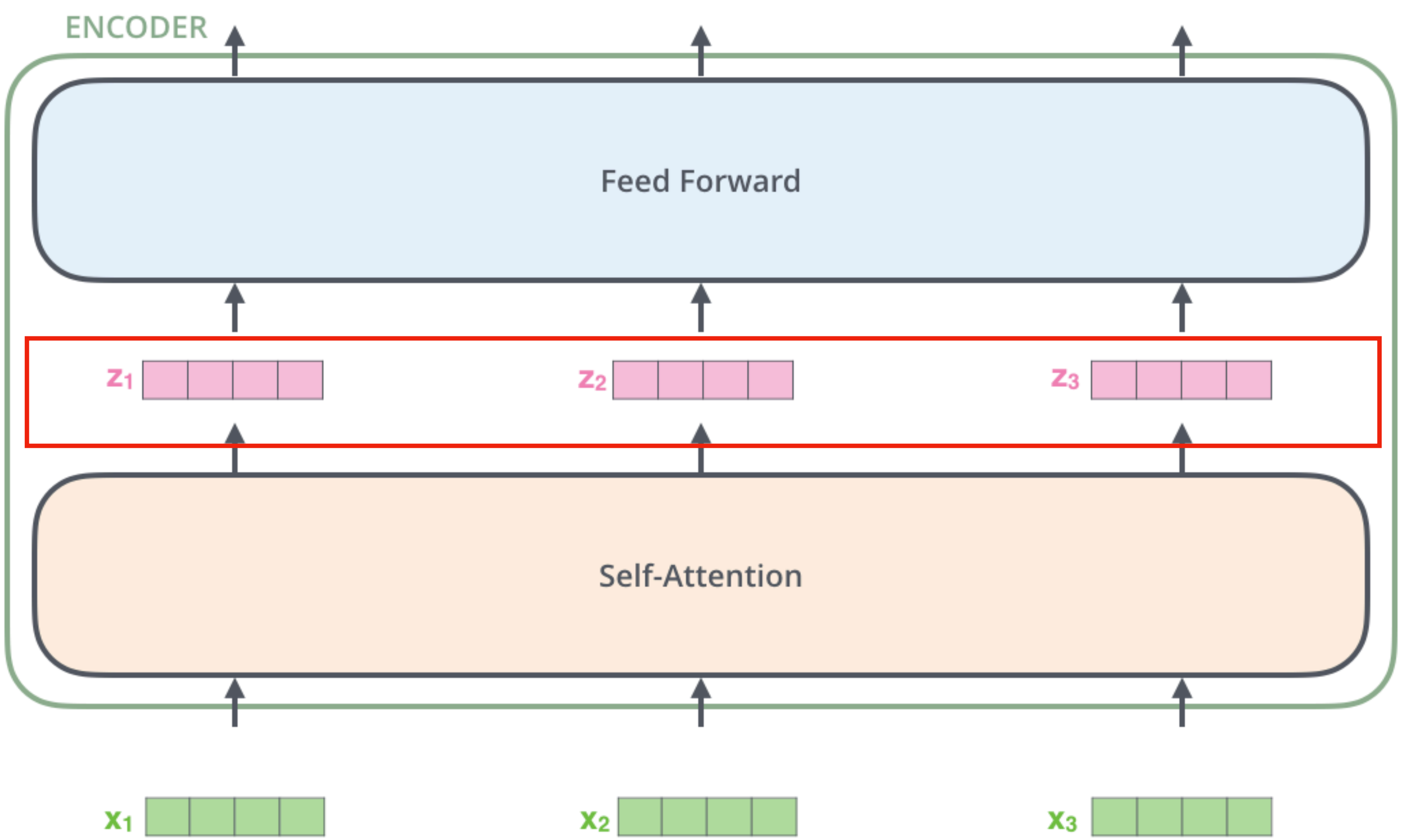
Transformers: Simplified



Multiple (50-90) such layers in a Transformer LM

Credit: http://jalammar.github.io/illustrated-transformer/

Transformers: Simplified



Multiple (50-90) such layers in a Transformer LM

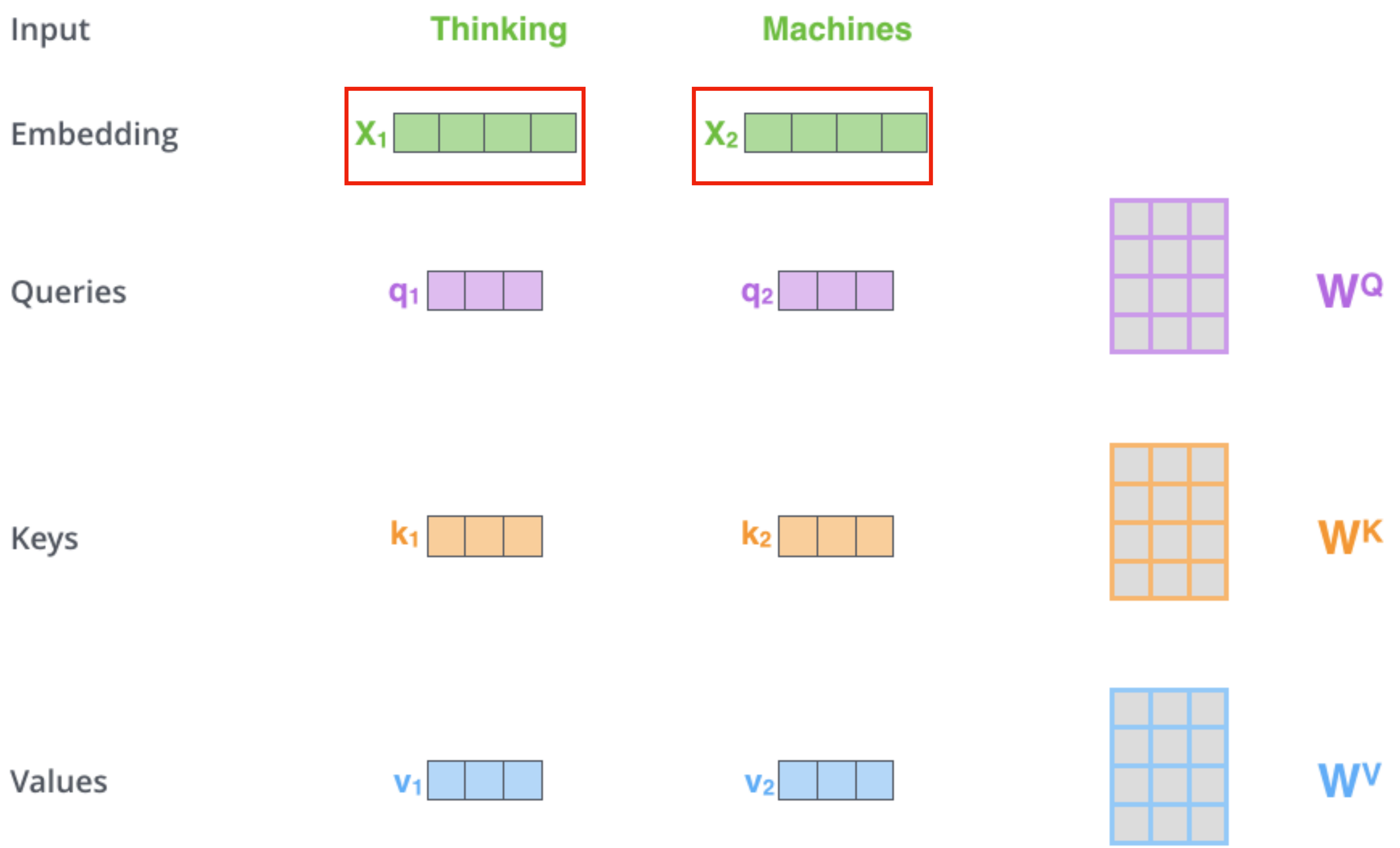
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention

- E.g. “The animal didn't cross the street because **it** was too tired”
- What does “**it**” refer to? “The animal” or “the street”
- Self-attention is the mechanism that helps LM associate:
    - “**it**” with “the animal”

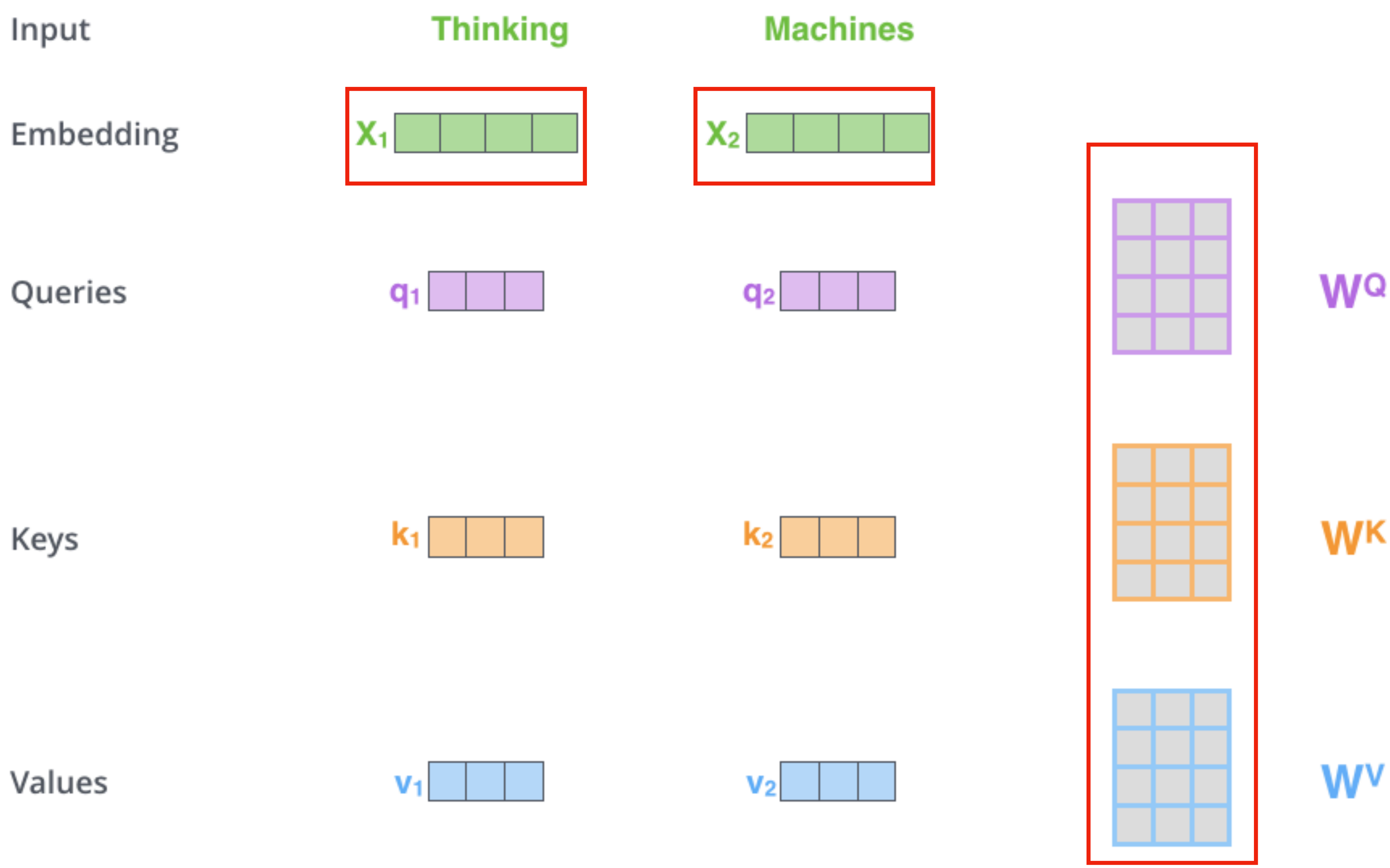
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



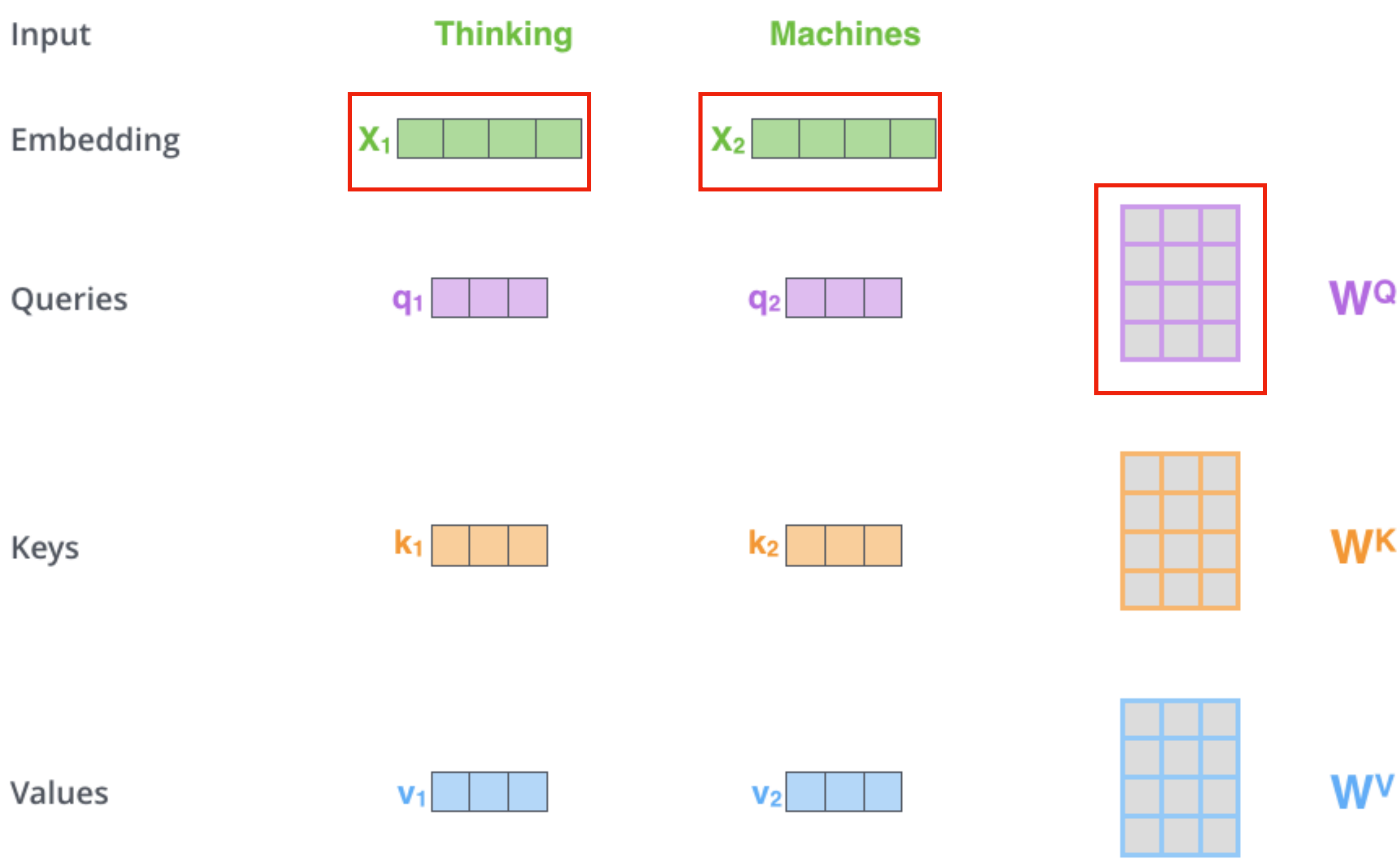
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



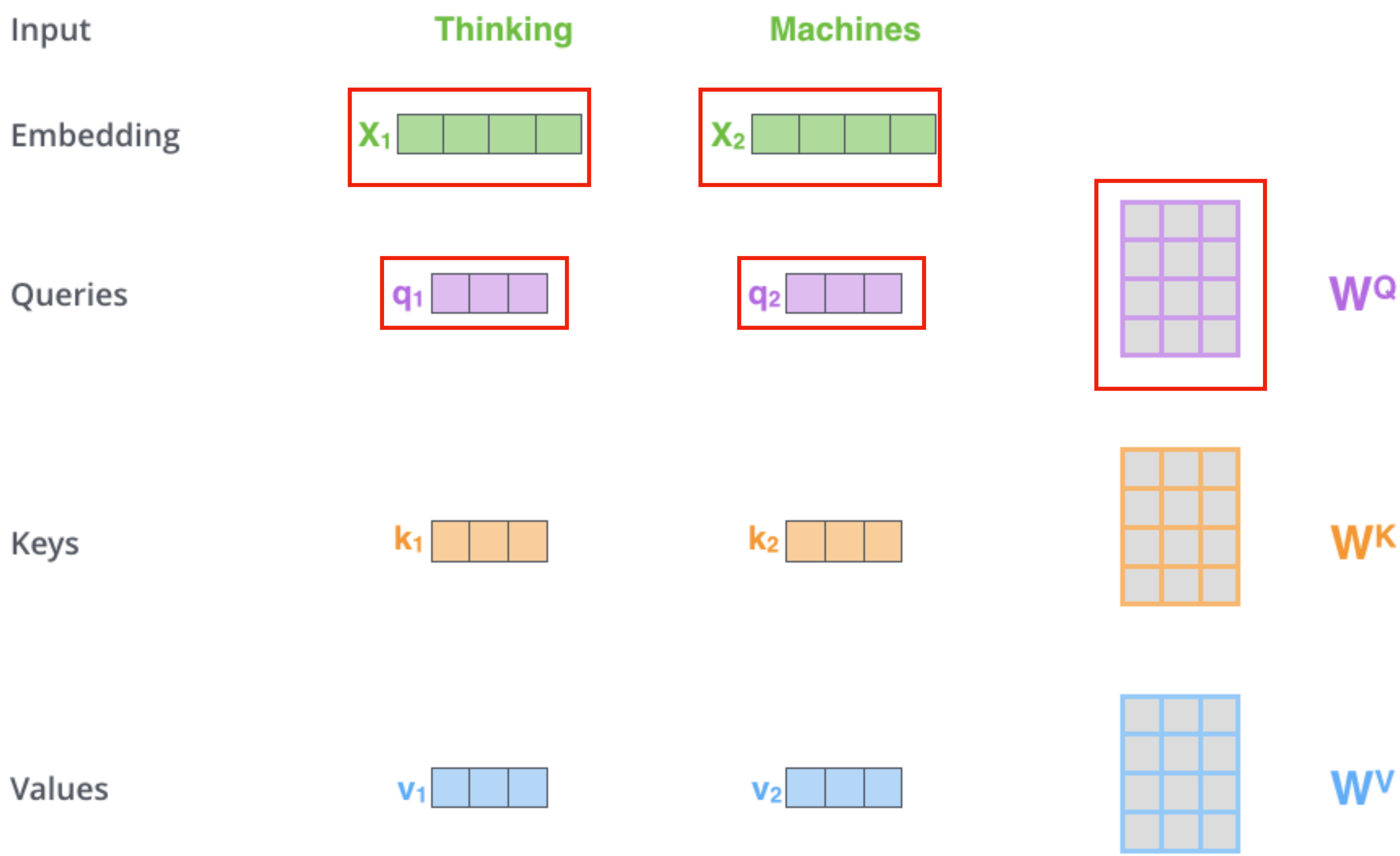
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



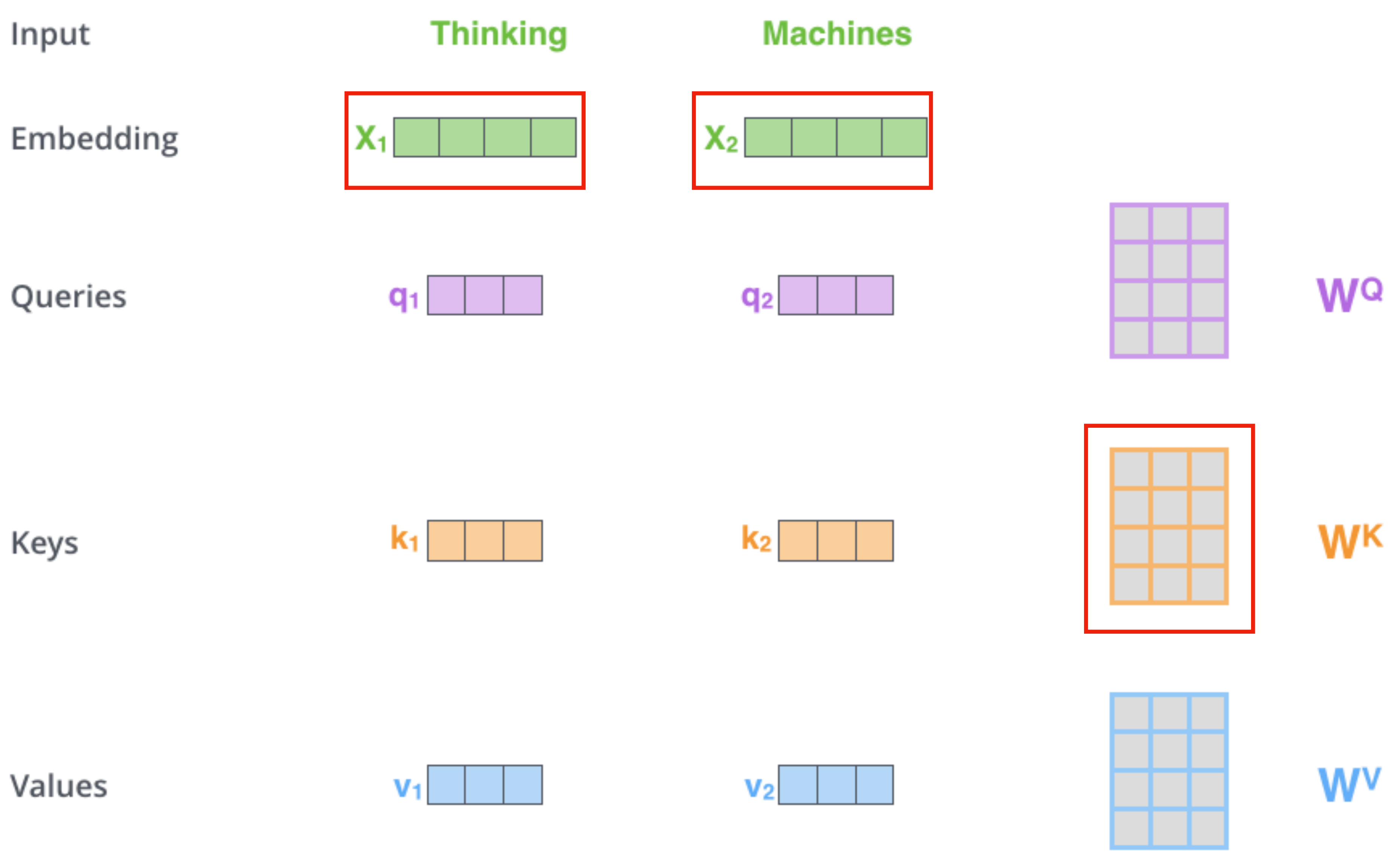
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



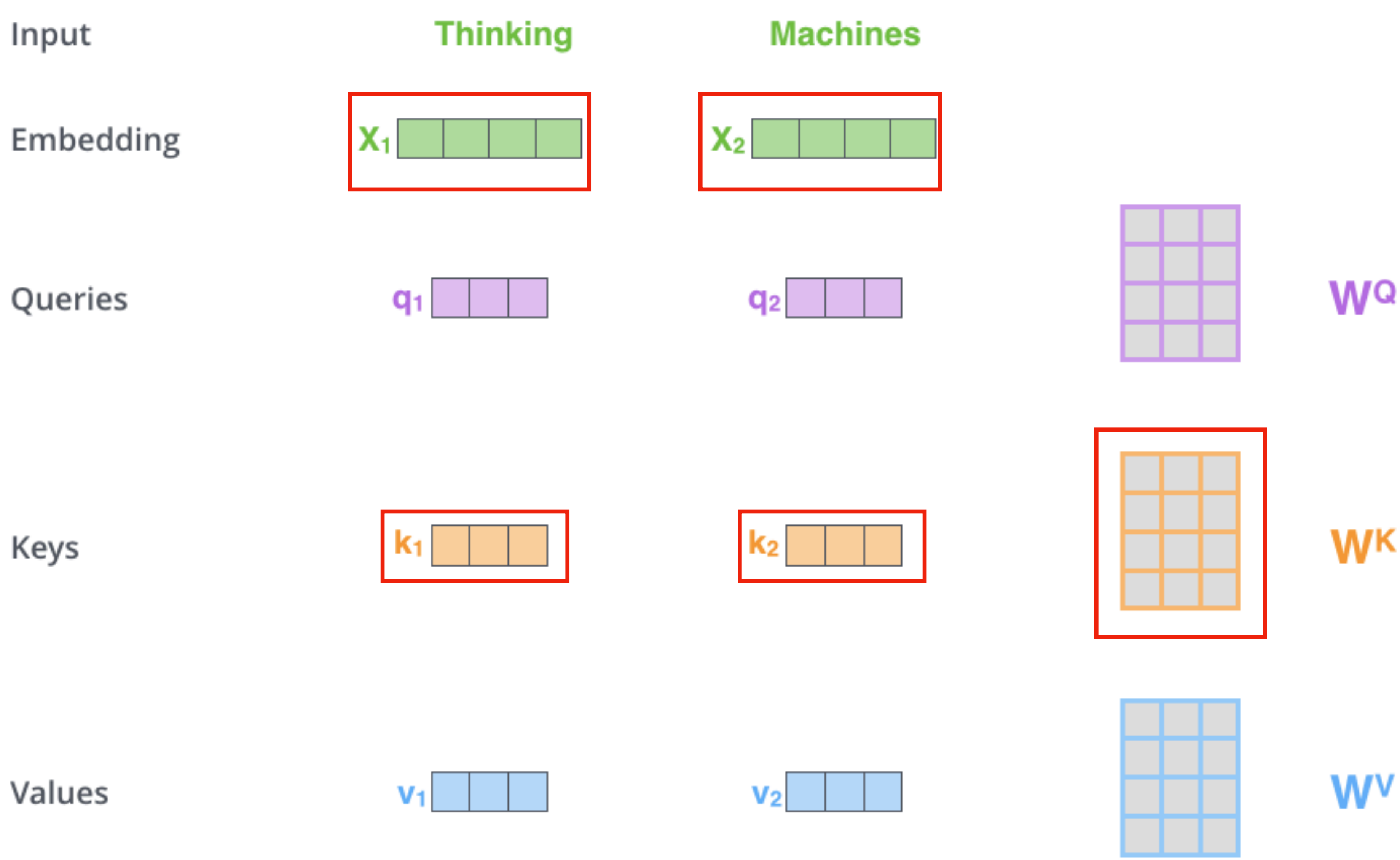
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



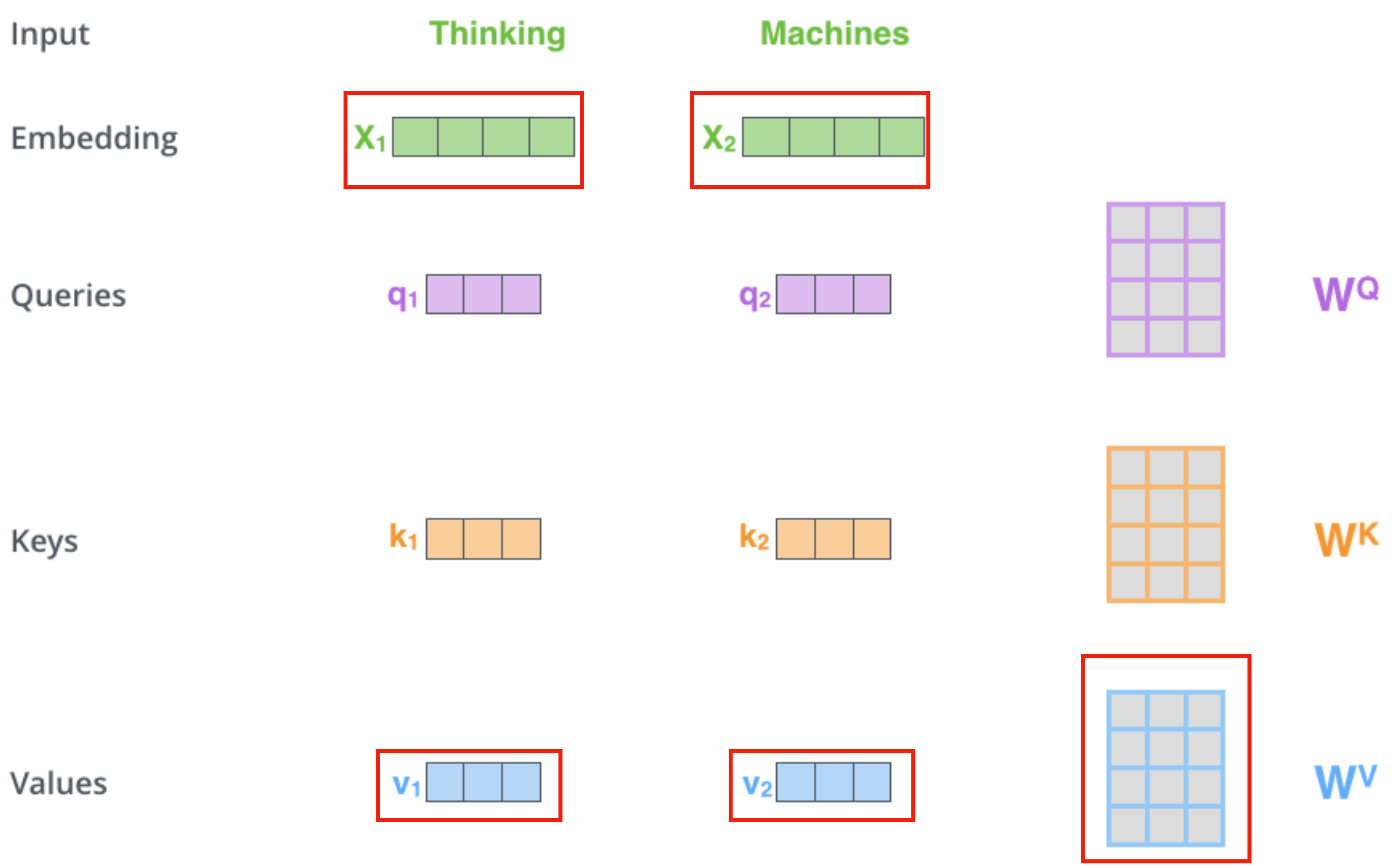
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0

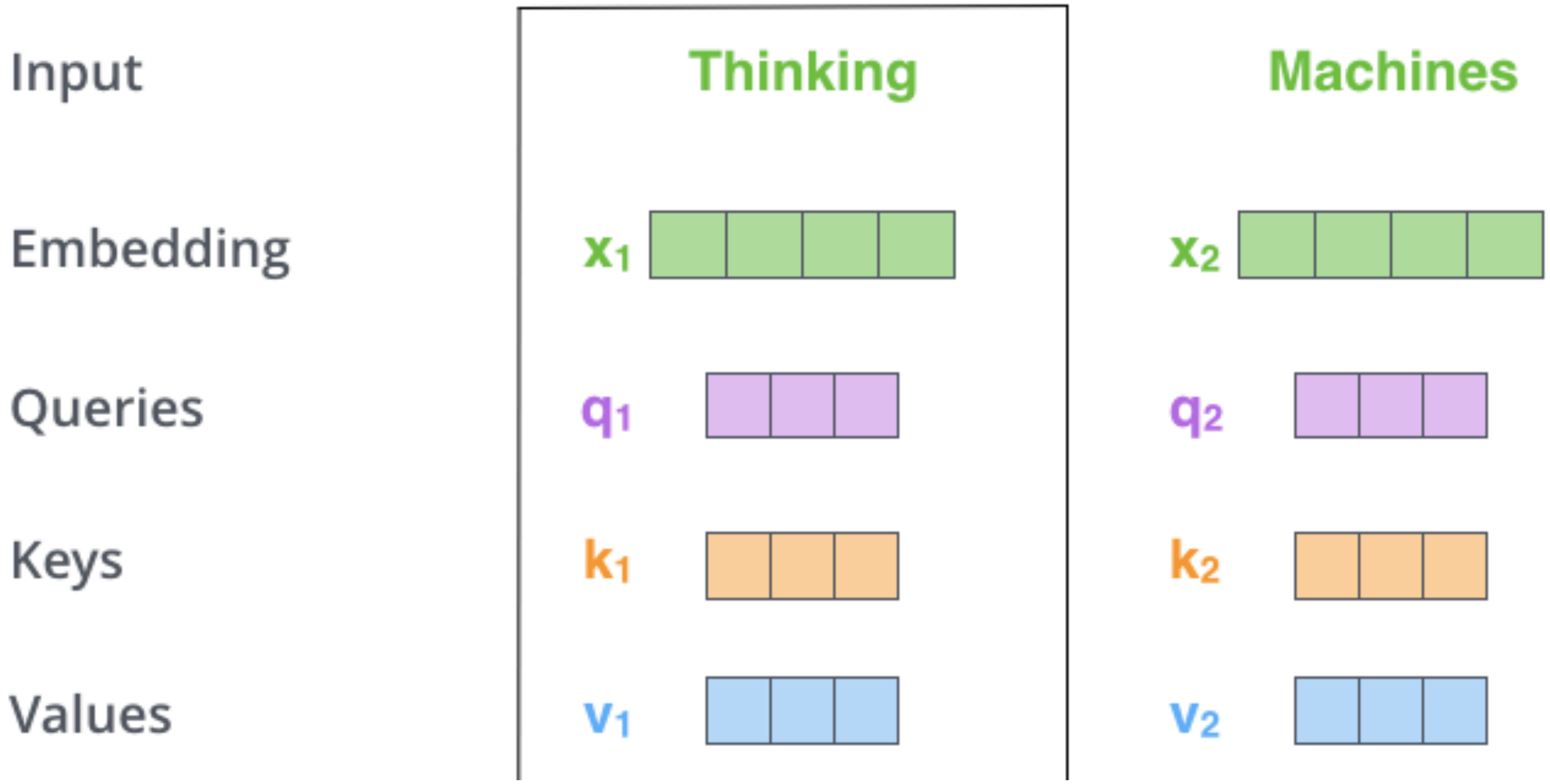


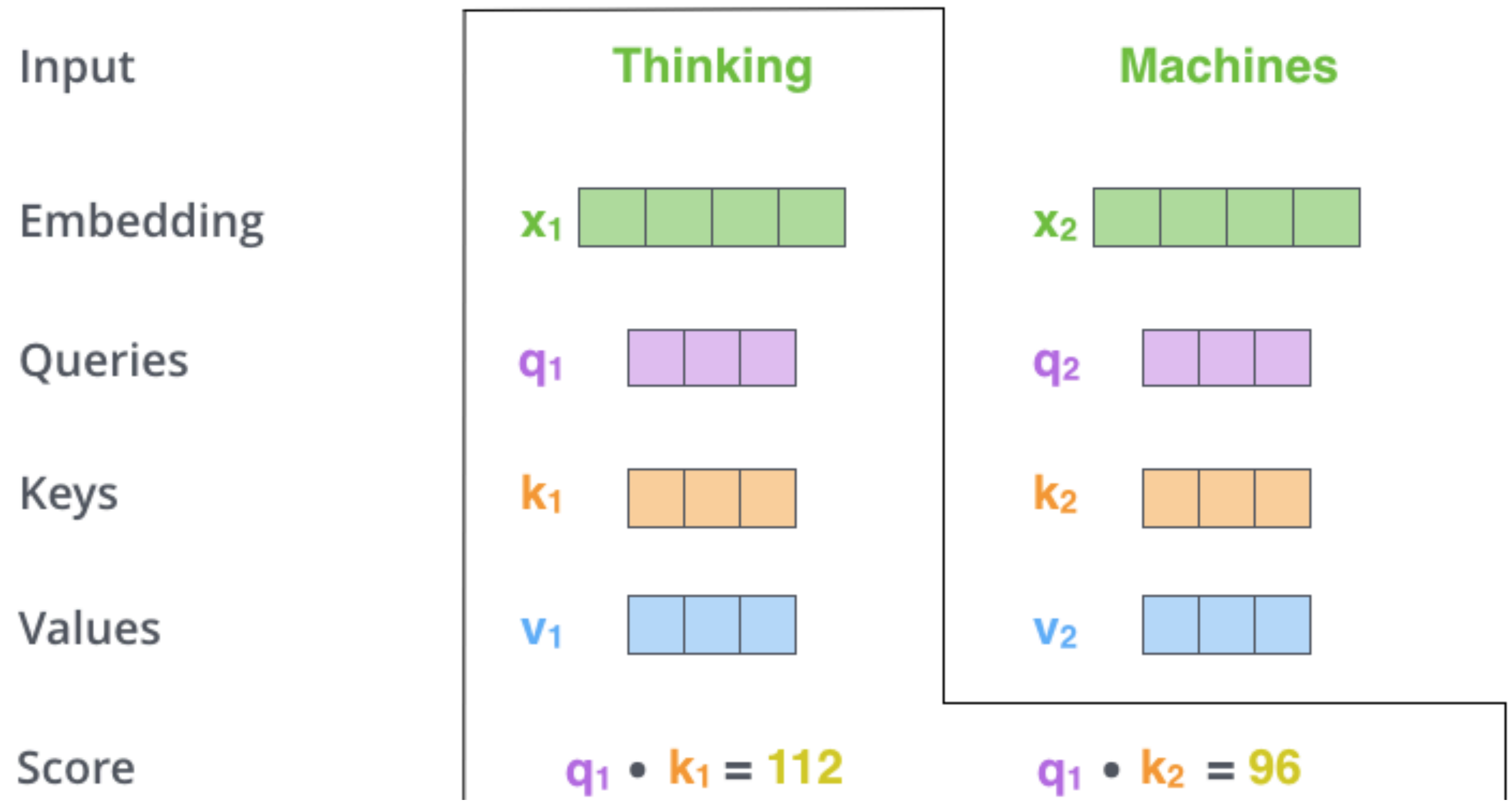
Credit: http://jalammar.github.io/illustrated-transformer/

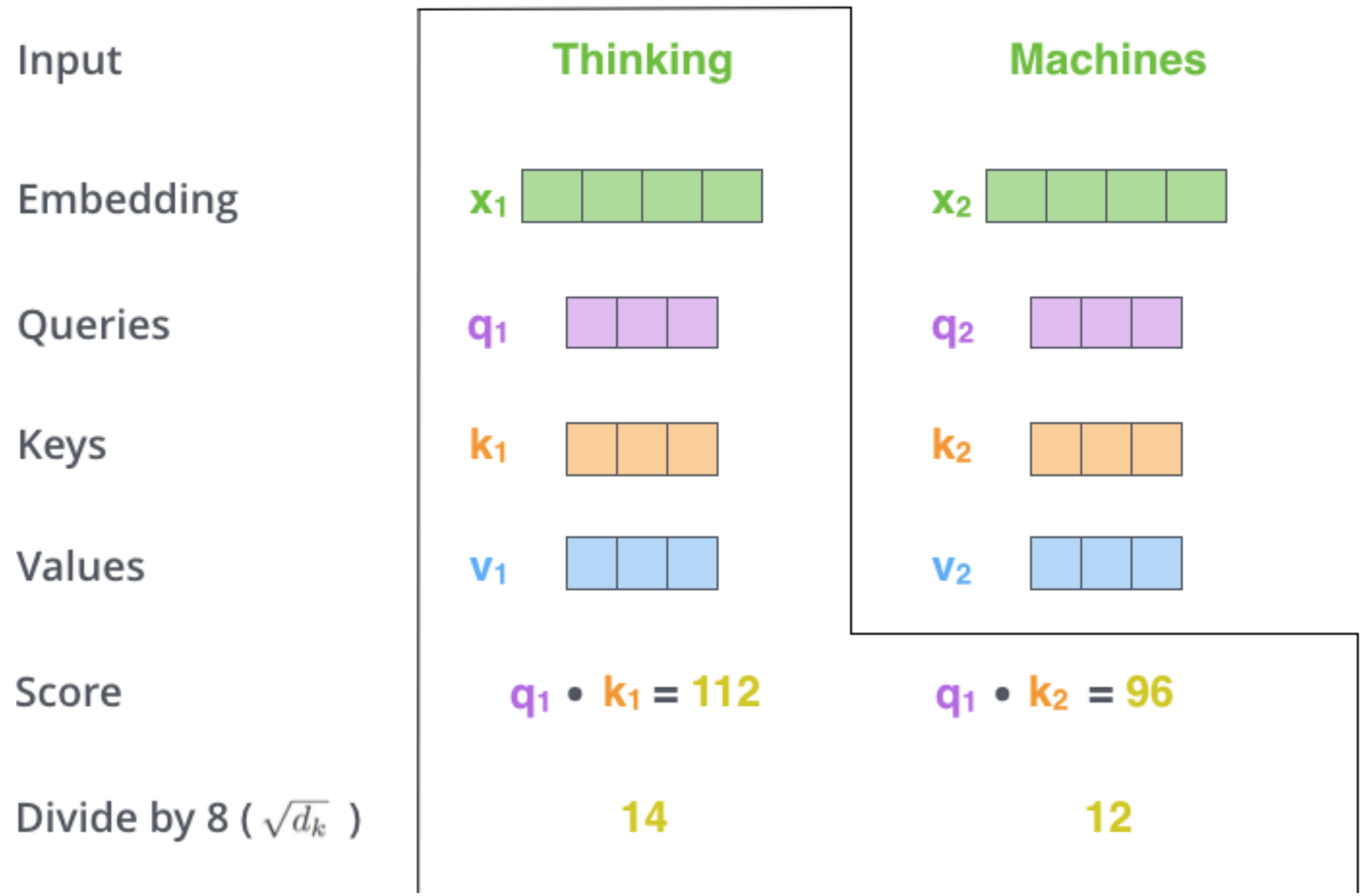
Self-Attention: Step 0

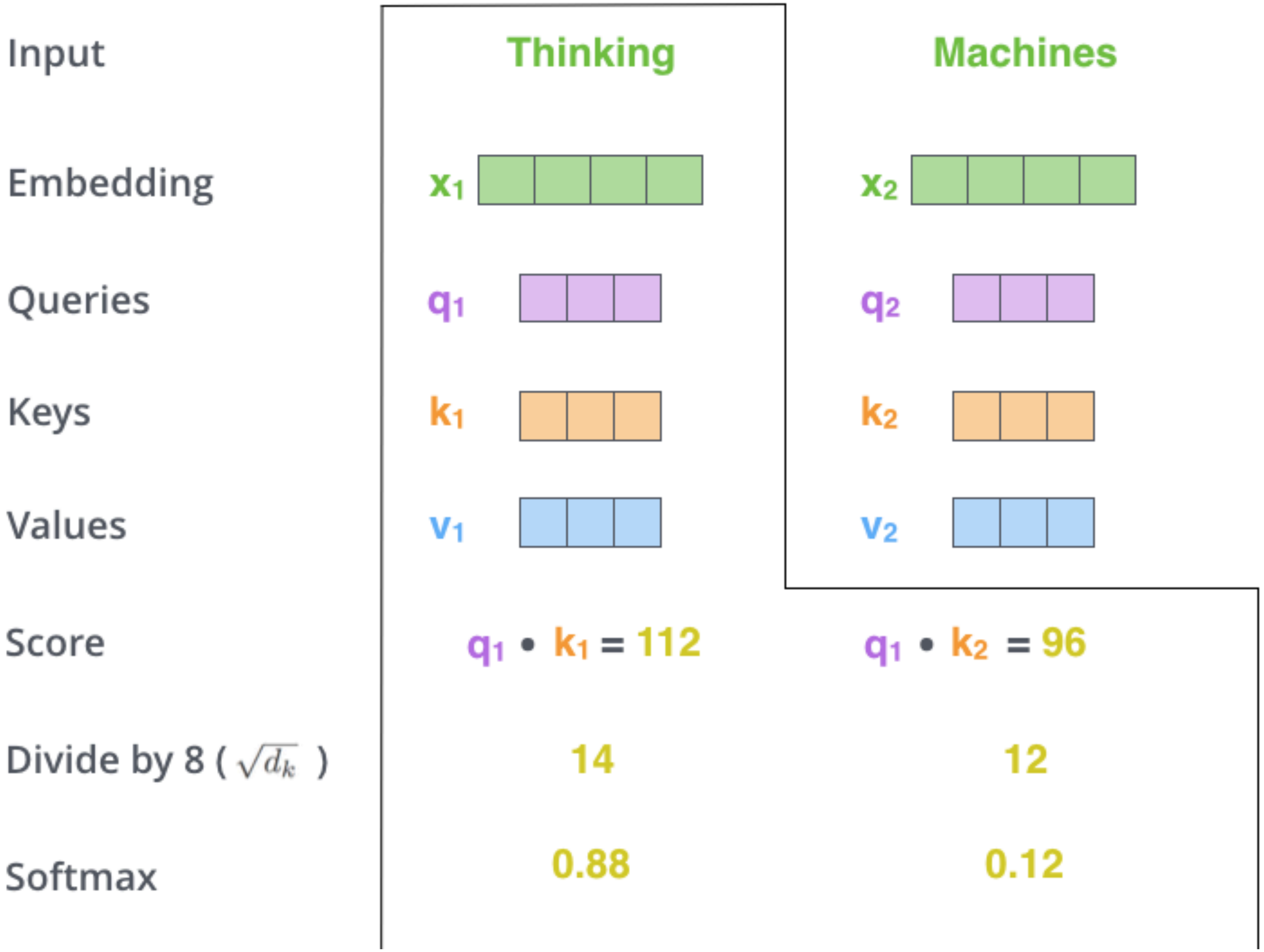


Credit: http://jalammar.github.io/illustrated-transformer/

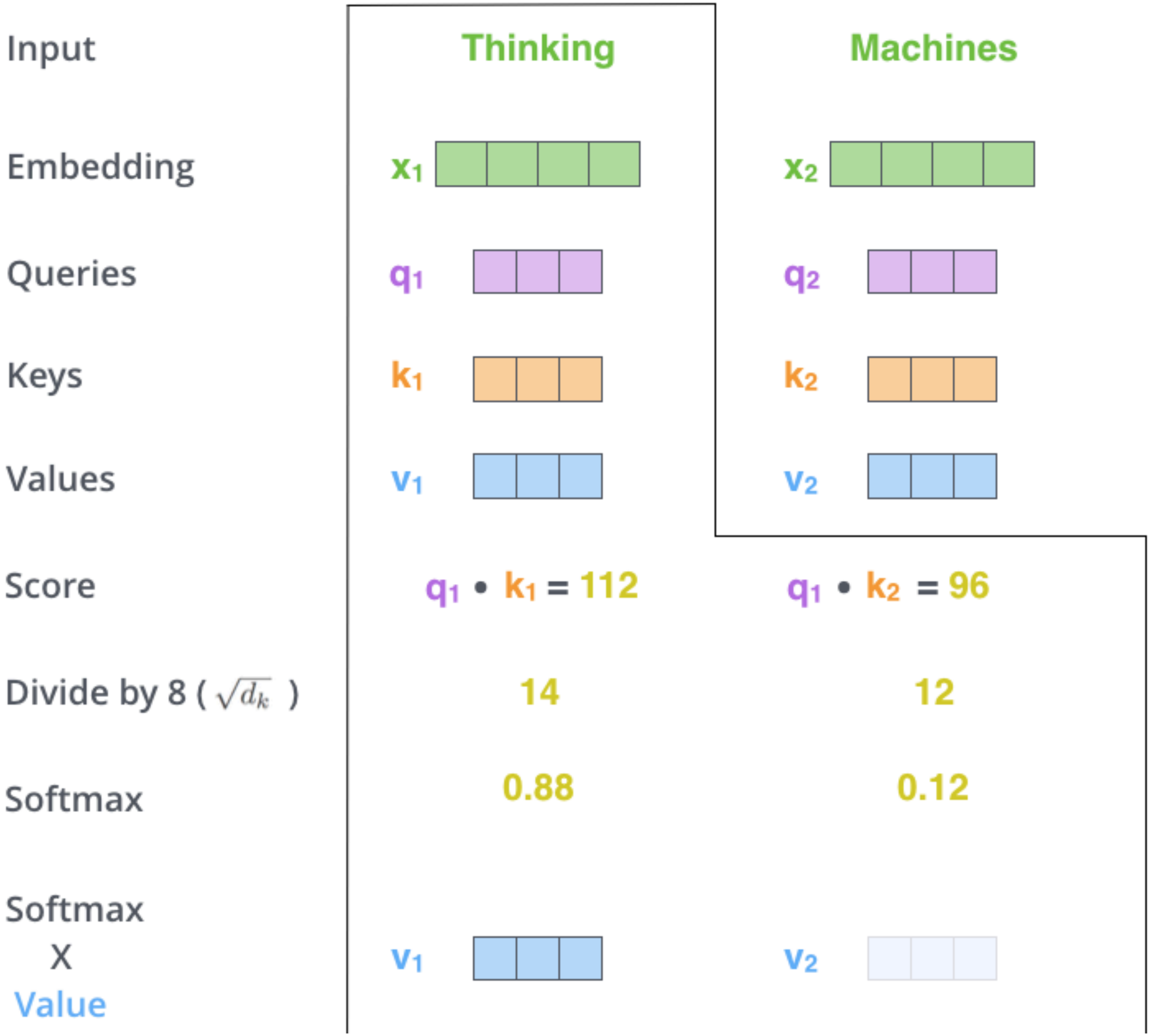

Input
Embedding
Queries
Keys
Values
Thinking
Machines
x1
x2
q1
q2
k1
k2
v1
v2


Input
Thinking
Machines
Embedding
x_1
x_2
Queries
q_1
q_2
Keys
k_1
k_2
Values
v_1
v_2
Score
$q_1 \bullet k_1 = 112$
$q_1 \bullet k_2 = 96$

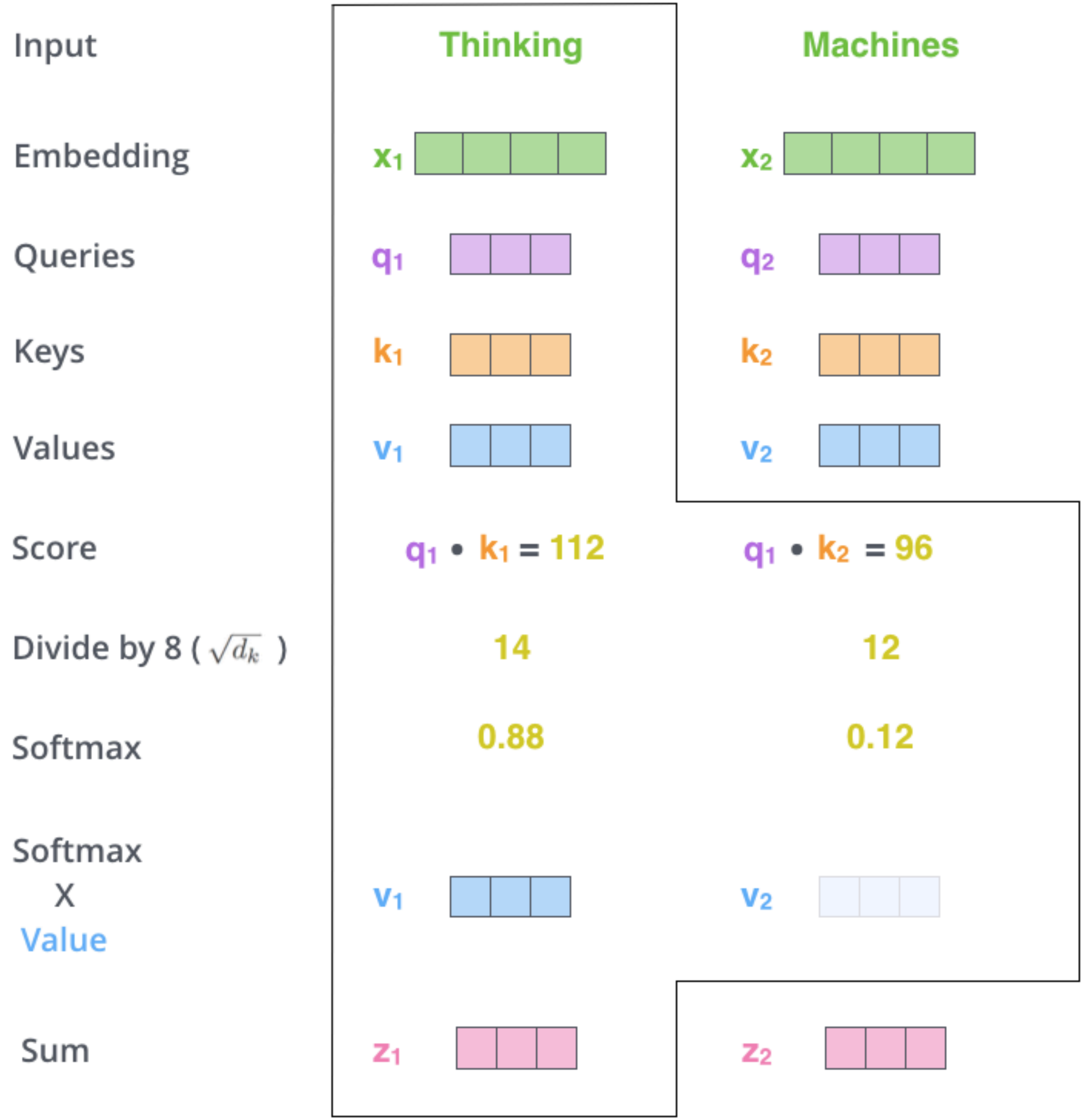

Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12



Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12

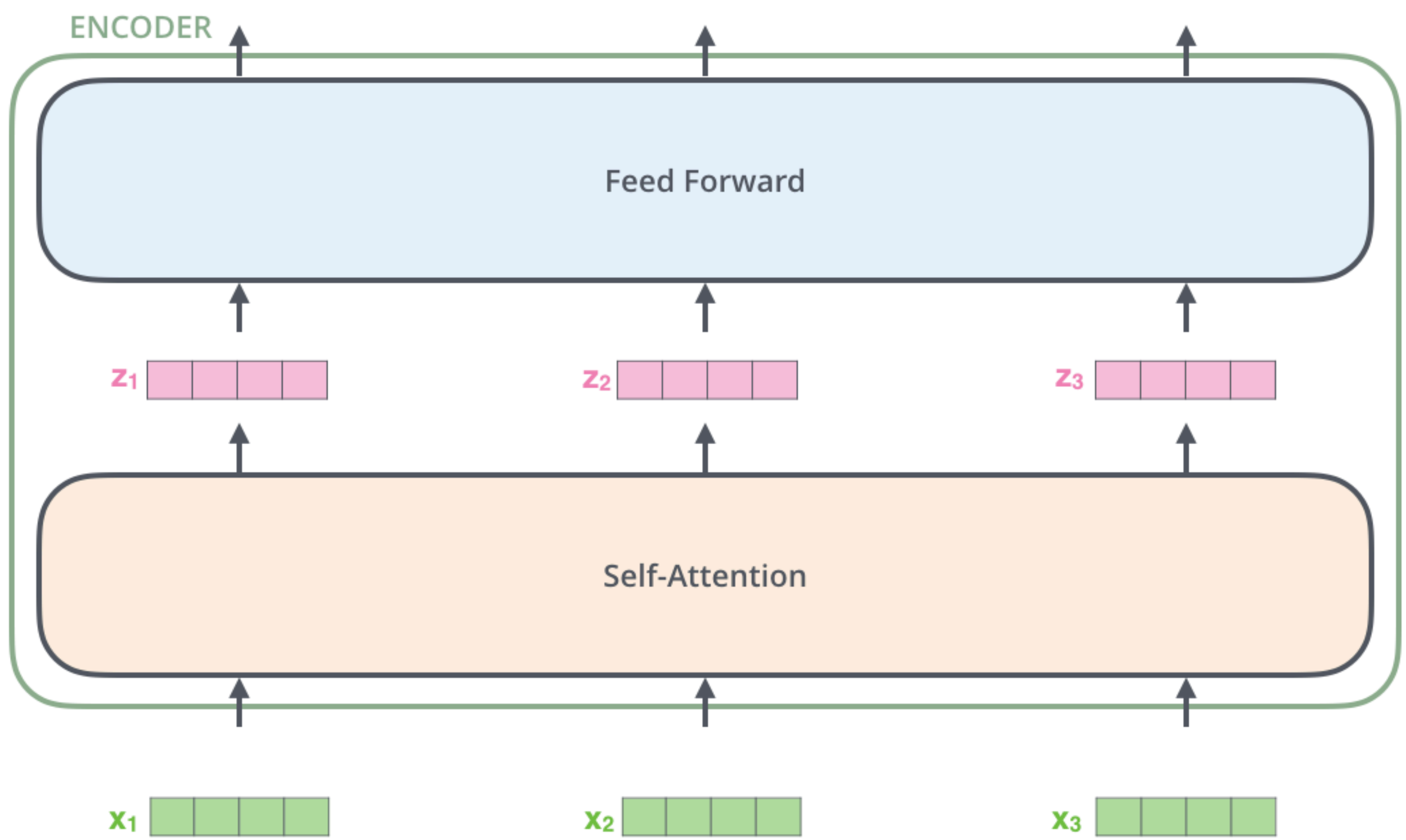


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12
Softmax
X
Value
v1
v2



Credit: http://jalammar.github.io/illustrated-transformer/

Transformers: Simplified



Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention

- Self-Attention seems to be asking an association question

Self-Attention

- Self-Attention seems to be asking an association question
- Query ~ smaller word embedding

Self-Attention

- Self-Attention seems to be asking an association question
- Query ~ smaller word embedding
- Key & Value ~ Key is the hash key that maps to Value

Self-Attention

- Self-Attention seems to be asking an association question
- Query ~ smaller word embedding
- Key & Value ~ Key is the hash key that maps to Value
- The names Query, Key and Value come from retrieval parlance

Self-Attention

- Self-Attention seems to be asking an association question
- Query ~ smaller word embedding
- Key & Value ~ Key is the hash key that maps to Value
- The names Query, Key and Value come from retrieval parlance
  - you fire a query, you compare to a key vector and return the value

Self-attention: exercise

- “Computers are thinking machines”
- Compute z for machines
- Q = K = V = $\begin{bmatrix} 0.2 & 0.8 \\ -0.2 & 0.5 \\ -0.3 & -0.4 \\ 0.7 & 0.7 \end{bmatrix}$
- Computers = [1 0 0 0], are = [0 1 0 0], thinking = [0 0 1 0], machines = [0 0 0 1]
- <u>Softmax</u>

Self-attention: exercise

- “Computers are thinking machines”
- Compute z for machines
- Q = K = V = $\begin{bmatrix} 0.2 & 0.8 \\ -0.2 & 0.5 \\ -0.3 & -0.4 \\ 0.7 & 0.7 \end{bmatrix}$
- Computers = [1 0 0 0], are = [0 1 0 0], thinking = [0 0 1 0], machines = [0 0 0 1]
- Softmax

z = [0.24   0.55]

Input	Computers	are	thinking	machines
Embedding				
Queries				
Keys				
Values				
Score	q·k			
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries				
Keys				
Values				
Score $q \cdot k$				
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Keys				
Values				
Score $q \cdot k$				
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

	Computers	are	thinking	machines
Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values				
Score $q \cdot k$				
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score $q \cdot k$				
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Keys		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)					
Softmax					
Softmax X Value					
Sum					

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	0.21	-0.49	$\boxed{0.98}$
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)					
Softmax					
Softmax X Value					
Sum					

Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys		[0.2 0.8]	[-0.2 0.5]	$\boxed{[-0.3\ -0.4]}$	[0.7 0.7]
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	0.21	$\boxed{-0.49}$	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)					
Softmax					
Softmax X Value					
Sum					

		Computers	are	thinking	machines
Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys		[0.2 0.8]	$\boxed{[-0.2\ 0.5]}$	[-0.3 -0.4]	[0.7 0.7]
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	$\boxed{0.21}$	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)					
Softmax					
Softmax X Value					
Sum					

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys	$\boxed{[0.2\ 0.8]}$	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score q·k	$\boxed{0.7}$	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)				
Softmax				
Softmax X Value				
Sum				

Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)					
Softmax					
Softmax X Value					
Sum					

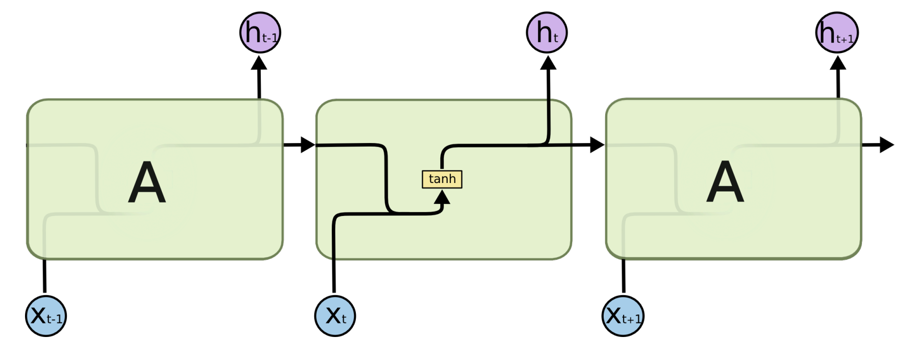
Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)	0.49	0.15	-0.35	0.69
Softmax				
Softmax X Value				
Sum				

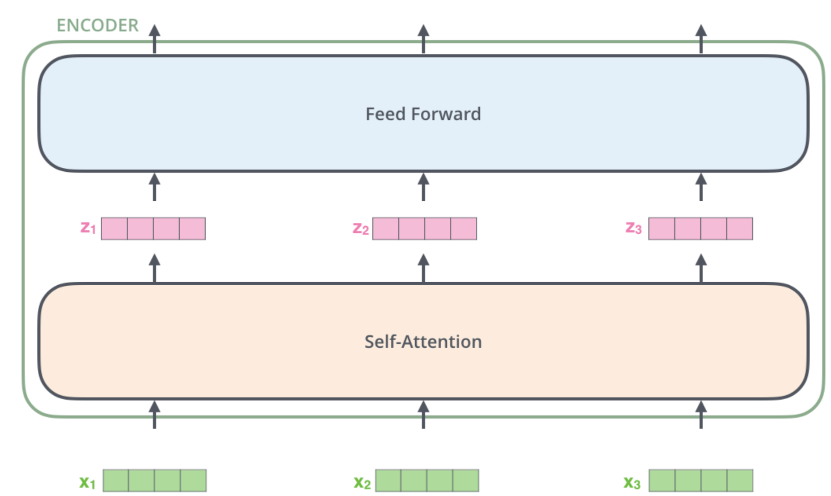
		Computers	are	thinking	machines
Input		Computers	are	thinking	machines
Embedding		[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values		[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score	q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)		0.49	0.15	-0.35	0.69
Softmax		0.30	0.21	0.13	0.36
Softmax X Value					
Sum					

	Computers	are	thinking	machines
Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)	0.49	0.15	-0.35	0.69
Softmax	0.30	0.21	0.13	0.36
Softmax X Value	[0.06 0.24]	[-0.04 0.10]	[-0.04 -0.05]	[0.25 0.25]
Sum				

Input	Computers	are	thinking	machines
Embedding	[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]
Queries	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	$\boxed{[0.7\ 0.7]}$
Keys	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Values	[0.2 0.8]	[-0.2 0.5]	[-0.3 -0.4]	[0.7 0.7]
Score q·k	0.7	0.21	-0.49	0.98
Divide by $\sqrt{2}$ ($\sqrt{d_k}$)	0.49	0.15	-0.35	0.69
Softmax	0.30	0.21	0.13	0.36
Softmax X Value	[0.06 0.24]	[-0.04 0.10]	[-0.04 -0.05]	[0.25 0.25]
Sum				[0.23 0.54]

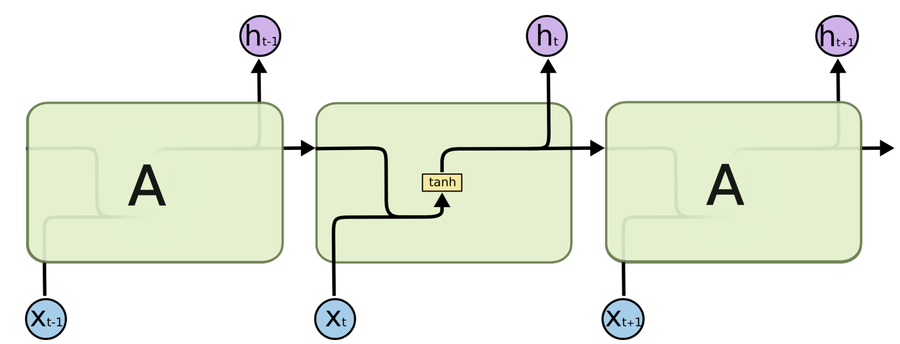
Transformers for Language Modelling

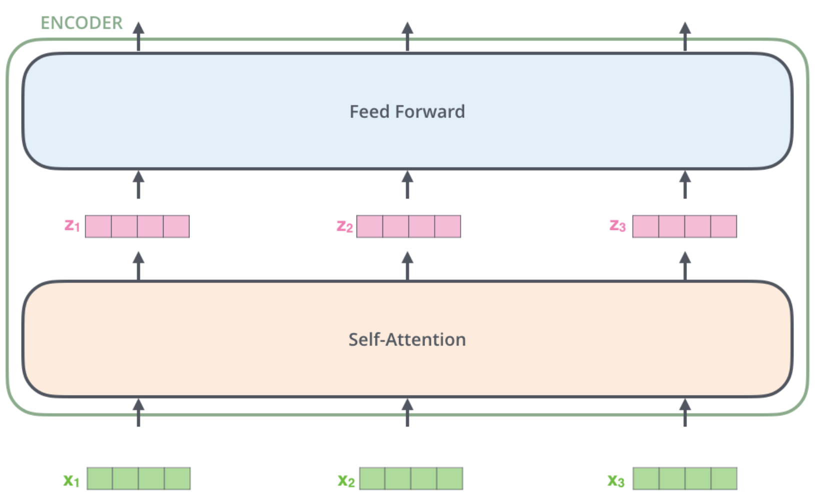




Transformers for Language Modelling

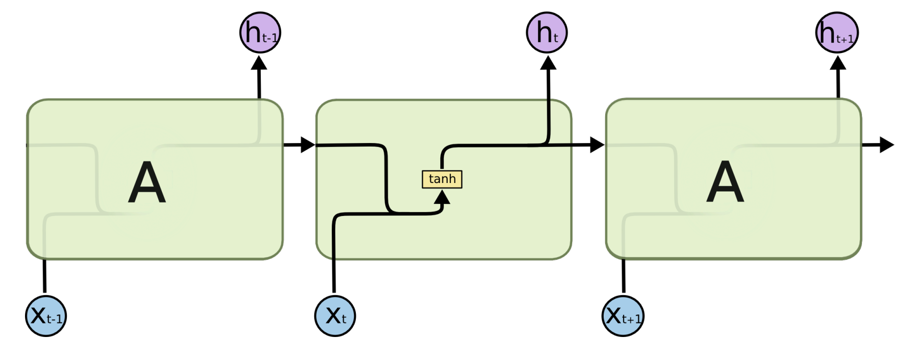
- RNNs: Process tokens one-by-one

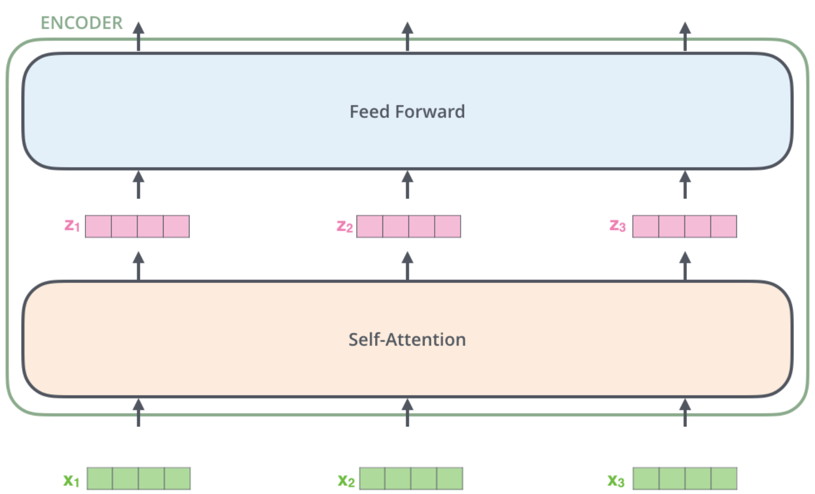




Transformers for Language Modelling

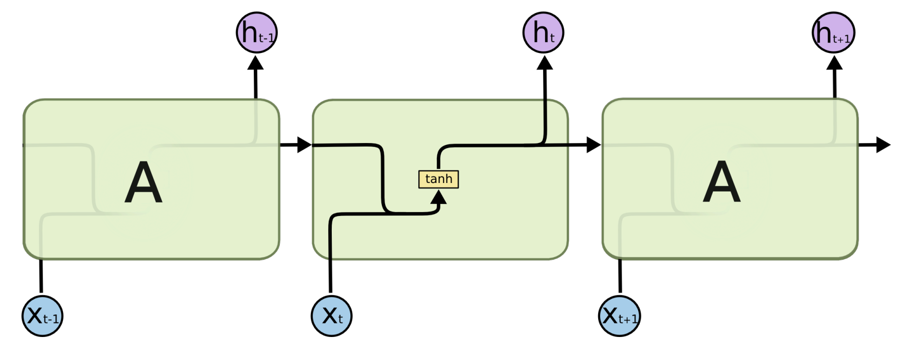
- RNNs: Process tokens one-by-one
  - Chain of dependencies built using a single token



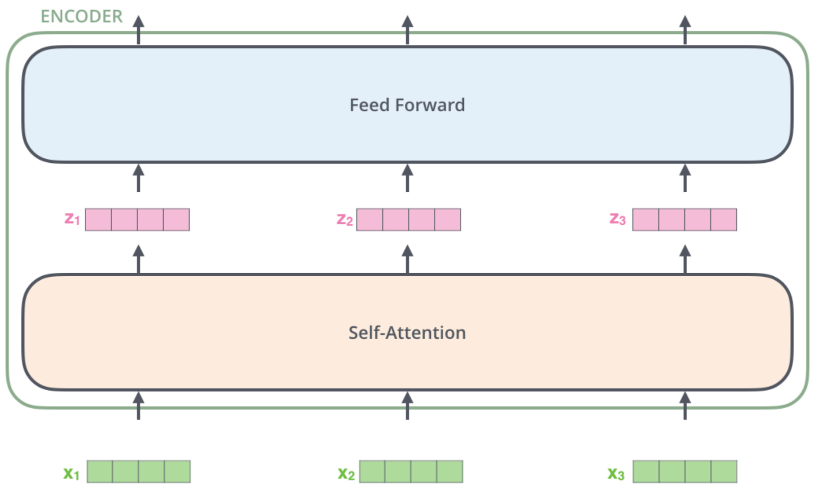


Transformers for Language Modelling

- RNNs: Process tokens one-by-one
  - Chain of dependencies built using a single token

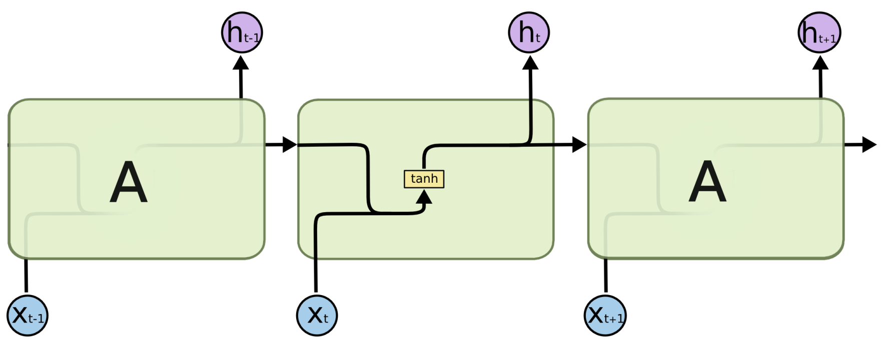


- Transformers LM: Process a segment of tokens

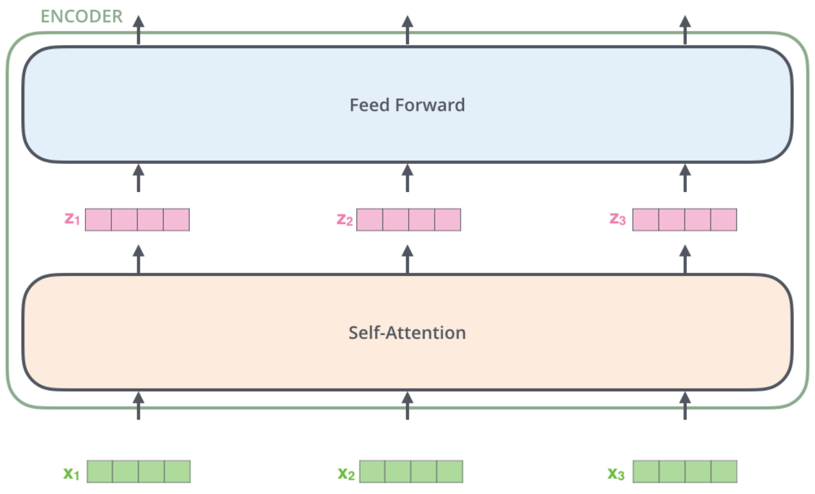


Transformers for Language Modelling

- RNNs: Process tokens one-by-one
  - Chain of dependencies built using a single token

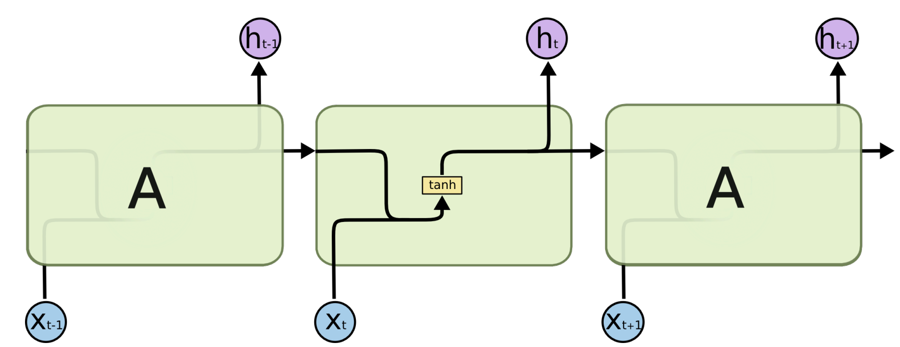


- Transformers LM: Process a segment of tokens
  - Dependencies within the segment

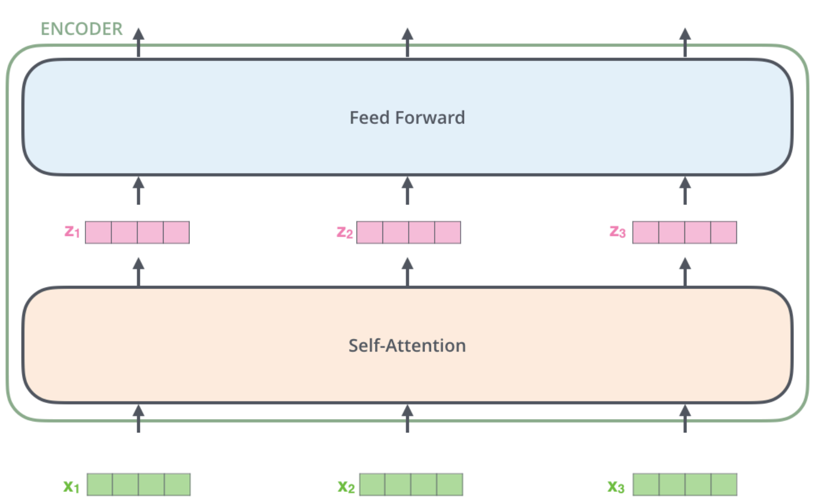


Transformers for Language Modelling

- RNNs: Process tokens one-by-one
  - Chain of dependencies built using a single token

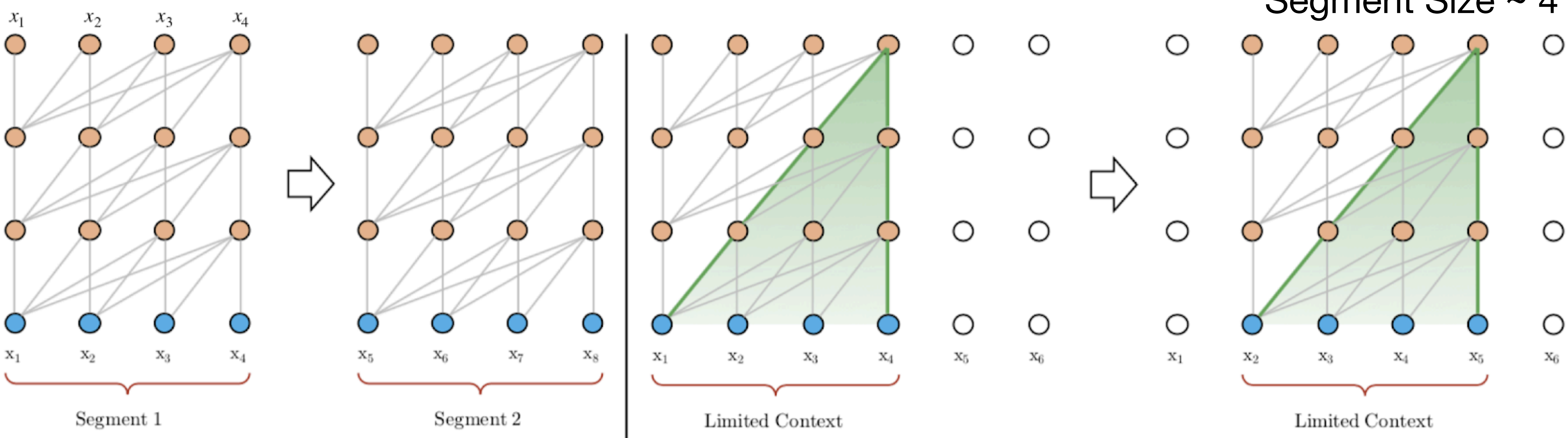


- Transformers LM: Process a segment of tokens
  - Dependencies within the segment
  - Within segment position is given by the positional encoding



Transformer LM processing of Segments

Segment Size ~ 4

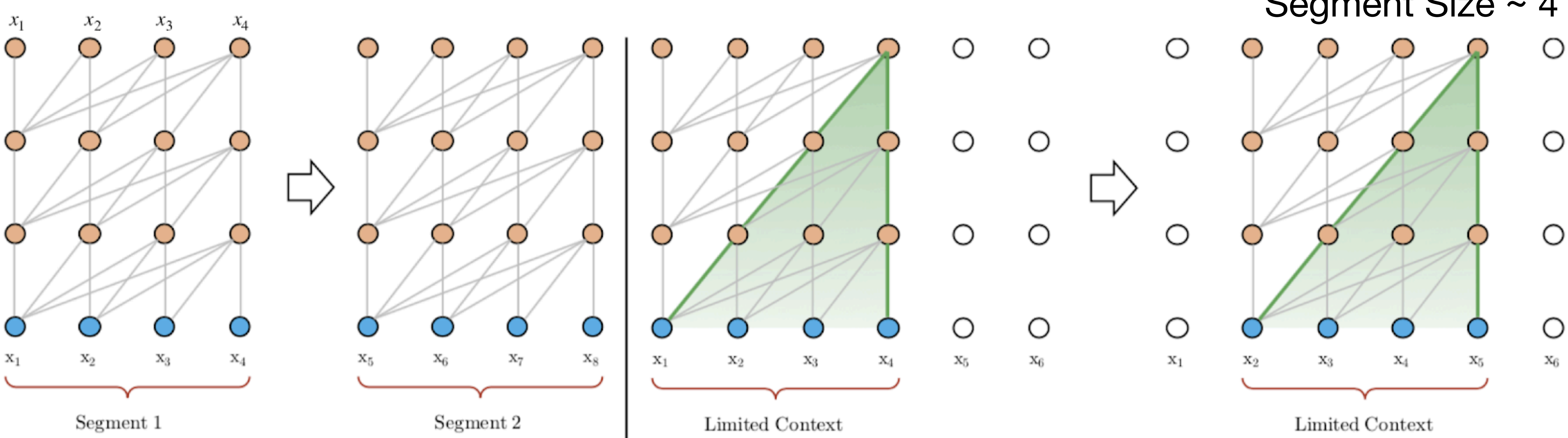


(a) Training phase. (b) Evaluation phase.

Dai et al., 2019

Transformer LM processing of Segments

Segment Size ~ 4



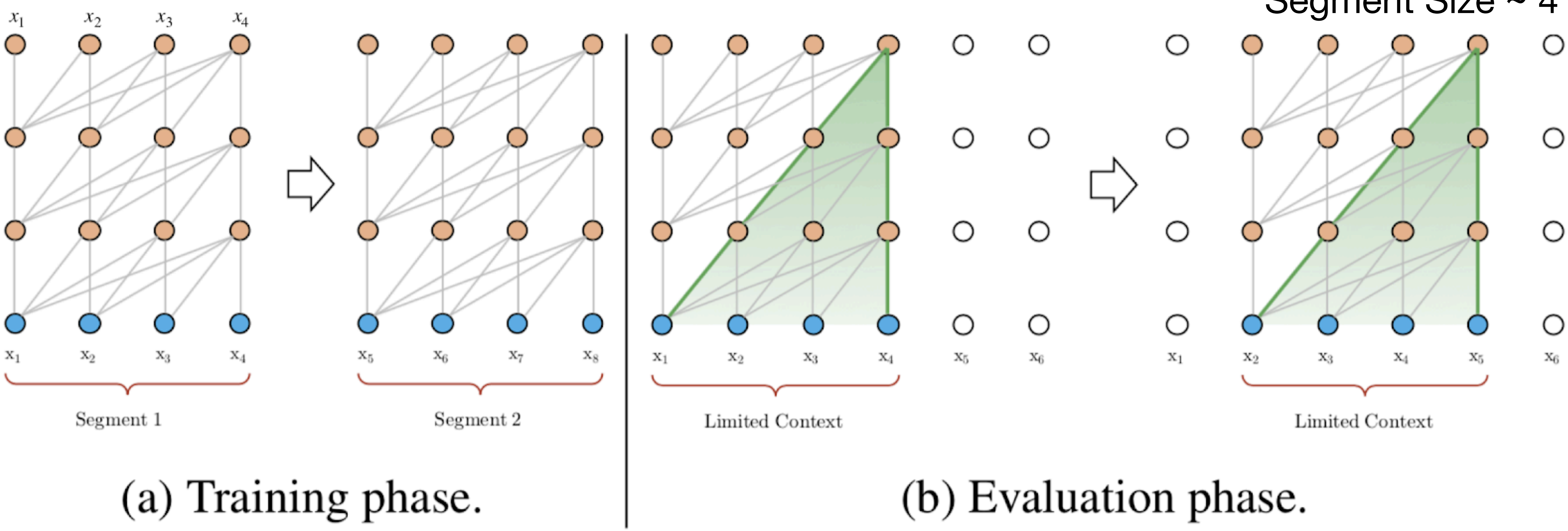
(a) Training phase. (b) Evaluation phase.

Dai et al., 2019

- Limited context-dependency
  - the model can't "use" a word that appeared several sentences ago.

Transformer LM processing of Segments

Segment Size ~ 4



Dai et al., 2019

- Limited context-dependency
  - the model can't "use" a word that appeared several sentences ago.
- Context fragmentation
  - no relationships can be leveraged across segments

[Jacob Devlin et al 2018]

Image credit: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

BERT: Bidirectional Encoder Representations from Transformers

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data
- Deep learning requires large amounts of annotated data

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data
- Deep learning requires large amounts of annotated data
- Language models for general purpose representations

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data
- Deep learning requires large amounts of annotated data
- Language models for general purpose representations
- Aim to **pretrain** general purpose representations

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data
- Deep learning requires large amounts of annotated data
- Language models for general purpose representations
- Aim to **pretrain** general purpose representations
- that can be **fine-tuned** using small task-specific dataset to obtain good performance on specialised tasks

BERT: Bidirectional Encoder Representations from Transformers

- For specialised tasks like named entity recognition, question answering there is a lack of training data
- Deep learning requires large amounts of annotated data
- Language models for general purpose representations
- Aim to **pretrain** general purpose representations
- that can be **fine-tuned** using small task-specific dataset to obtain good performance on specialised tasks
- Welcome BERT!

Transformers for Language Modelling

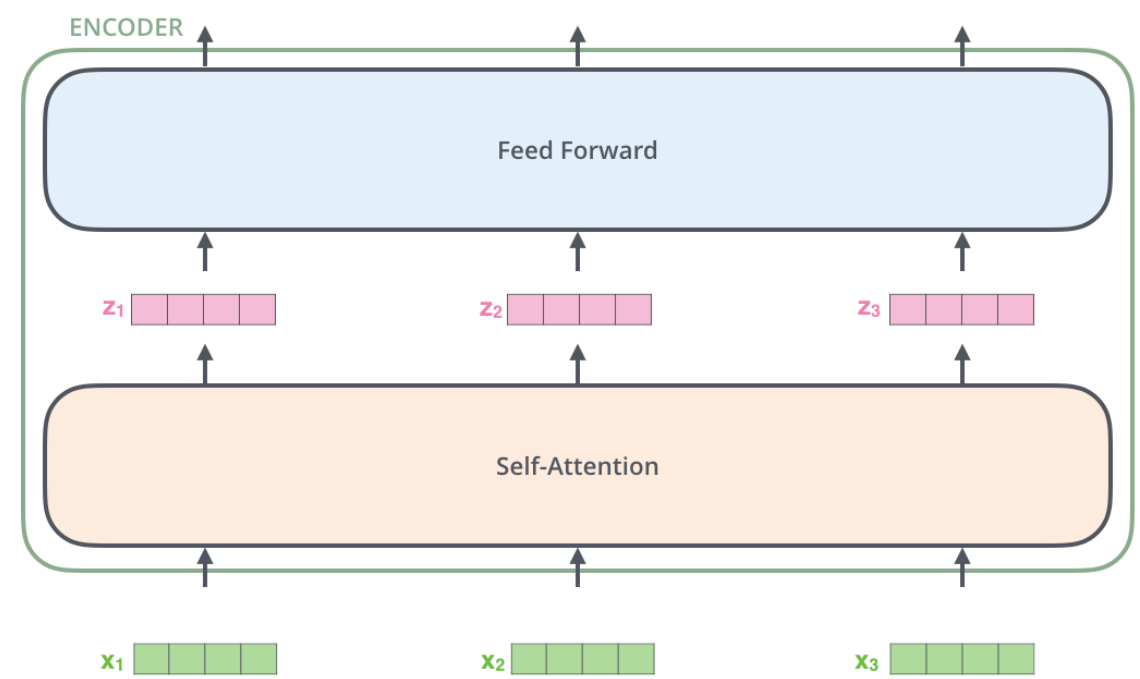


Image Credit: https://arxiv.org/pdf/1706.03762.pdf
Content Credit:TransformerXL Explained & Al-Rfou et al. 2018

Transformers for Language Modelling

- Transformers LM

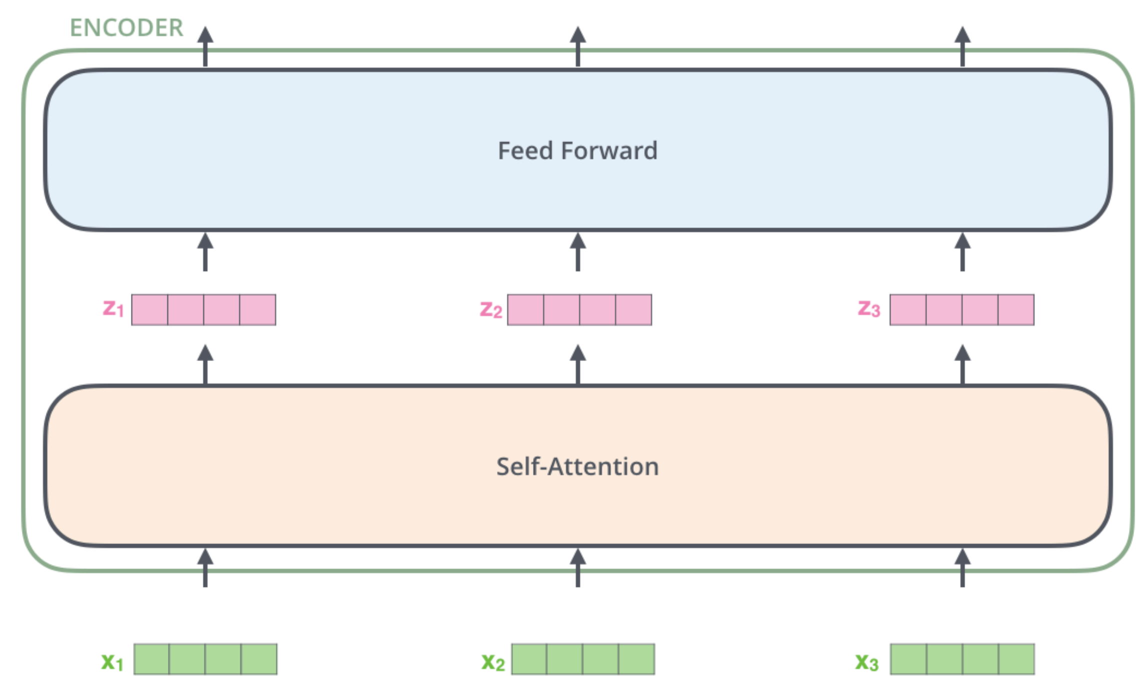


Image Credit: https://arxiv.org/pdf/1706.03762.pdf
Content Credit:TransformerXL Explained & Al-Rfou et al. 2018

Transformers for Language Modelling

- Transformers LM
  - Unidirectional

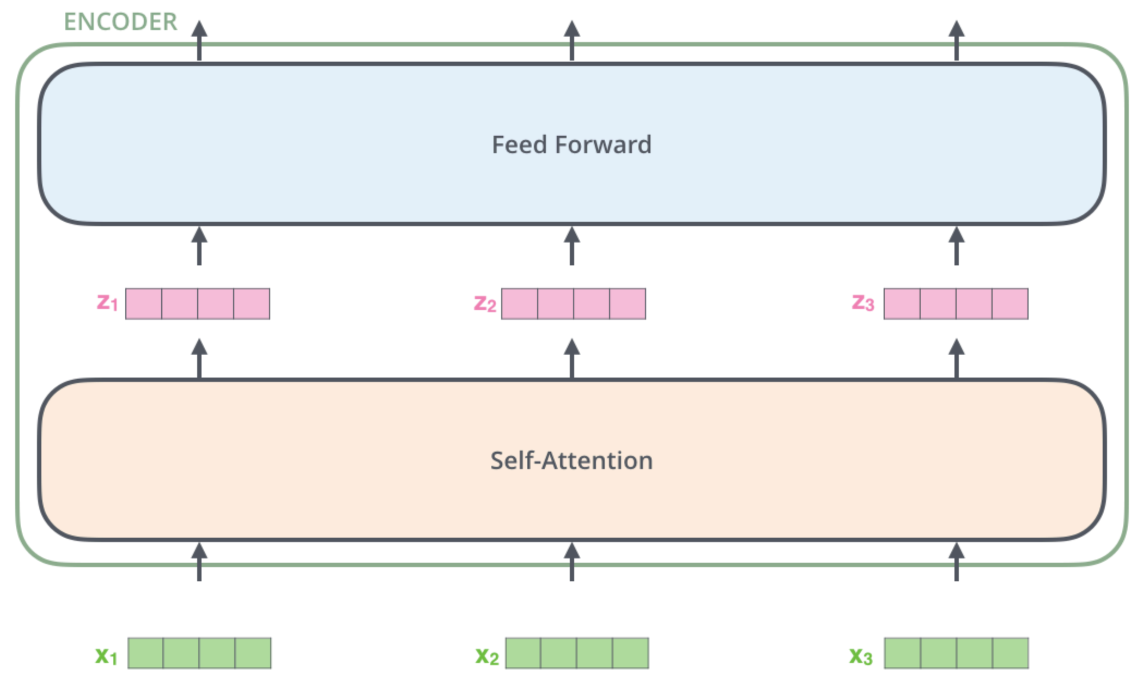


Image Credit: https://arxiv.org/pdf/1706.03762.pdf
Content Credit: TransformerXL Explained & Al-Rfou et al. 2018

Transformers for Language Modelling

- Transformers LM
  - Unidirectional
  - Segment of tokens

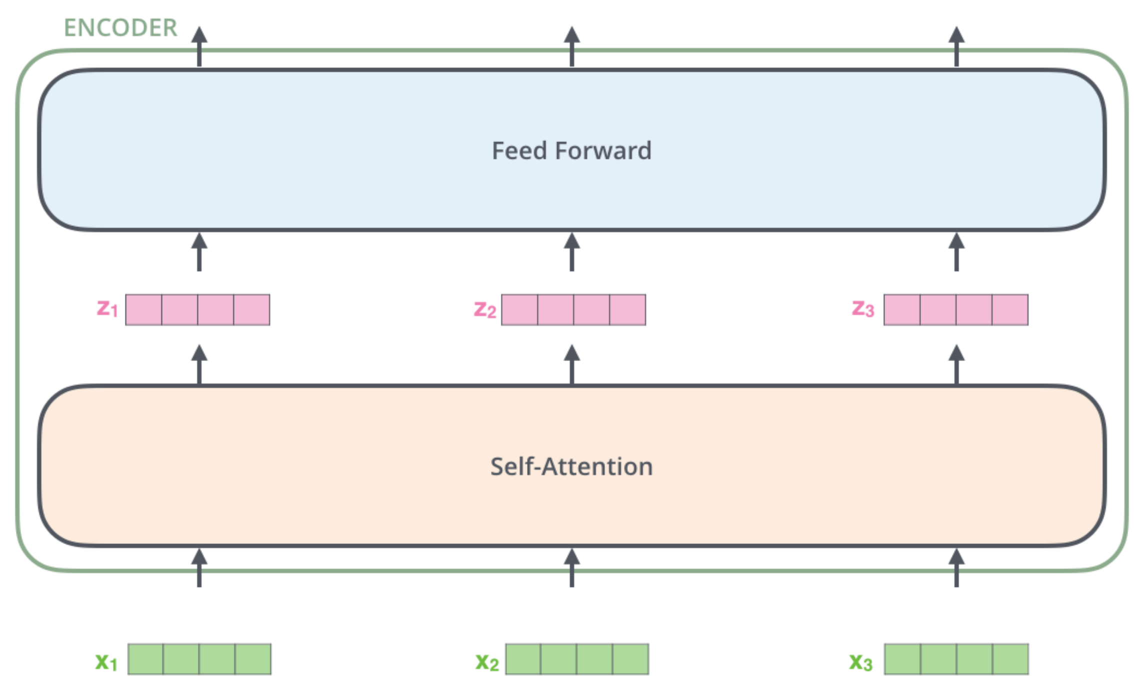


Image Credit: https://arxiv.org/pdf/1706.03762.pdf
Content Credit:TransformerXL Explained & Al-Rfou et al. 2018

Transformers for Language Modelling

- Transformers LM
  - Unidirectional
  - Segment of tokens
- Language Models predict the next word

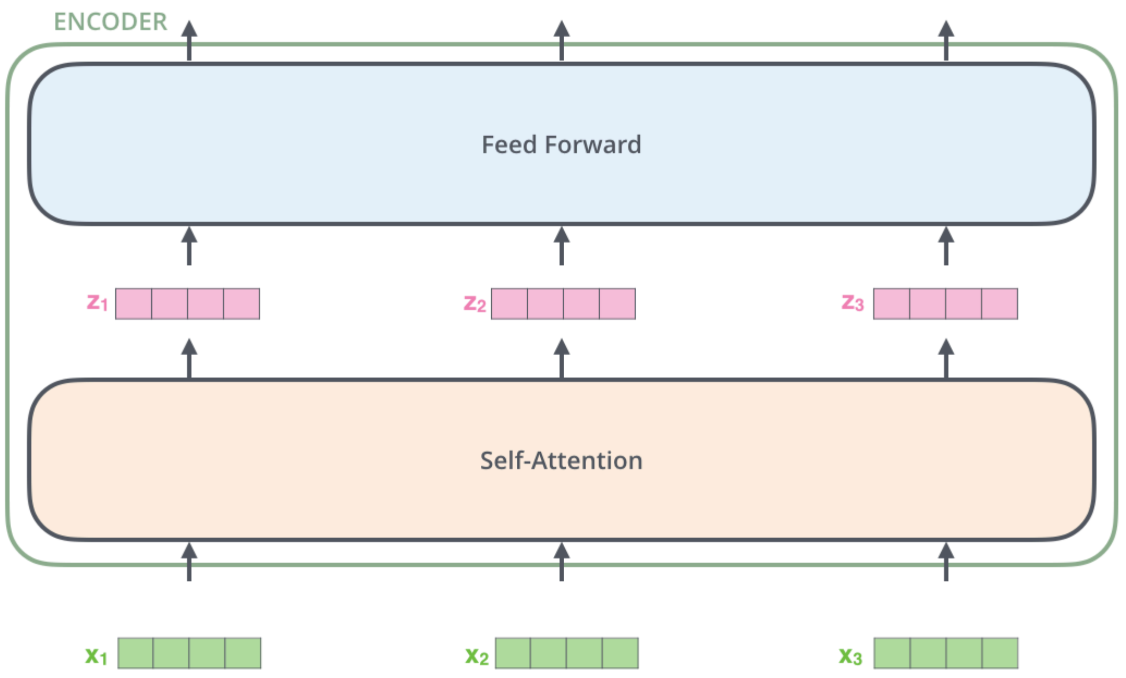


Image Credit: https://arxiv.org/pdf/1706.03762.pdf
Content Credit:TransformerXL Explained & Al-Rfou et al. 2018

Encoder Representations

- Require only the representations
- Forego of the output layer and only keep the encoder

Traditionally,

- Language Models predict the next word

Traditionally,

- Language Models predict the next word
- Or loosely, they “fill in the blank” based on the context

Traditionally,

- Language Models predict the next word
- Or loosely, they “fill in the blank” based on the context
- The man went to the store and bought a _____ of shoes

Traditionally,

- Language Models predict the next word
- Or loosely, they “fill in the blank” based on the context
- The man went to the store and bought a _____ of shoes
- Language models are mostly used as unidirectional tools

Traditionally,

- Language Models predict the next word
- Or loosely, they “fill in the blank” based on the context
- The man went to the store and bought a _____ of shoes
- Language models are mostly used as unidirectional tools
- Bidirectionality in this above example can help make a better judgement

Traditionally,

- Language Models predict the next word
- Or loosely, they “fill in the blank” based on the context
- The man went to the store and bought a _____ of shoes
- Language models are mostly used as unidirectional tools
- Bidirectionality in this above example can help make a better judgement
- In BERT, this bidirectionality is important to obtain good general purpose representations

BERT: Learning Setup

BERT: Learning Setup

- Pretraining

BERT: Learning Setup

- Pretraining
  - Takes lots and lots of sentences

BERT: Learning Setup

- Pretraining
  - Takes lots and lots of sentences
  - Masked LM

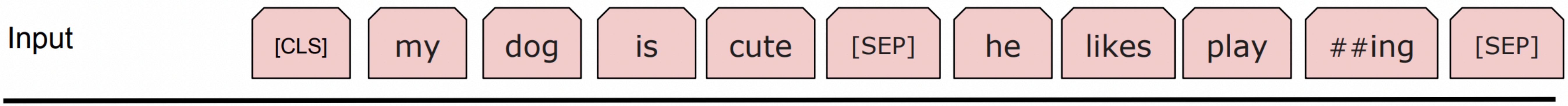
BERT: Learning Setup

- Pretraining
  - Takes lots and lots of sentences
  - Masked LM
  - Next Sentence Prediction

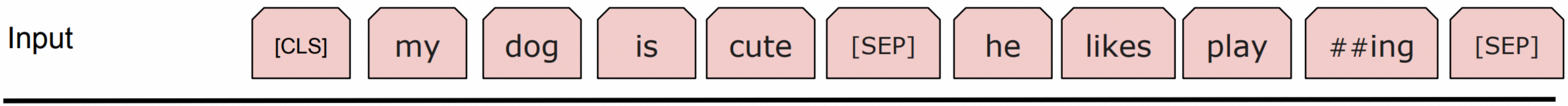
BERT: Learning Setup

- Pretraining
  - Takes lots and lots of sentences
  - Masked LM
  - Next Sentence Prediction
- Finetune

Masked LM

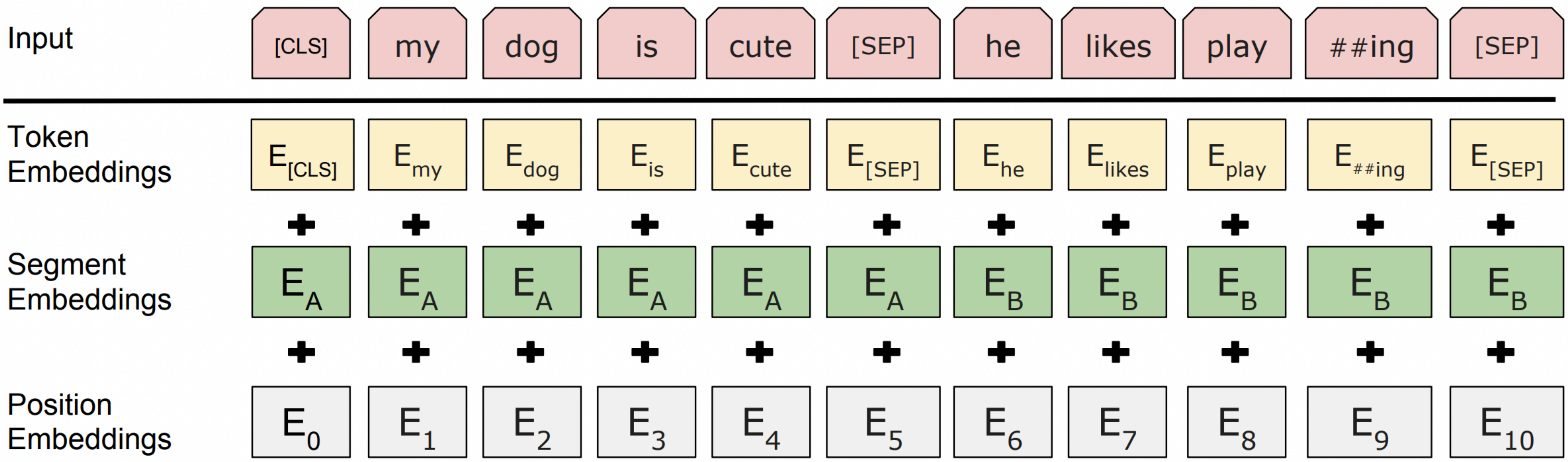


Masked LM



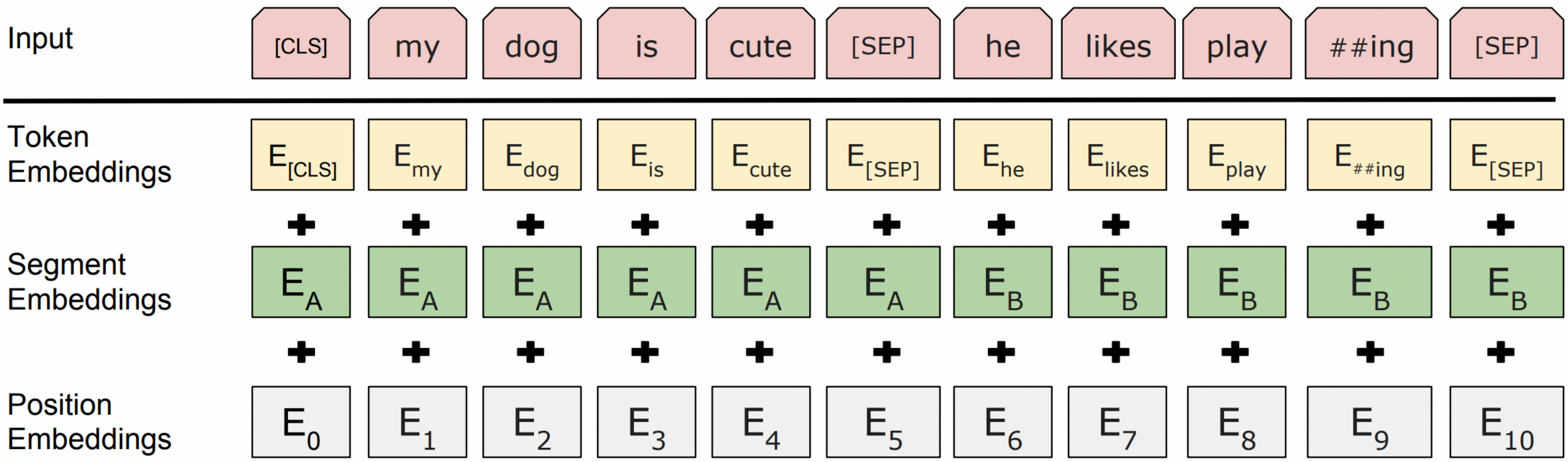
- Use specialised tokens CLS, SEP

Masked LM



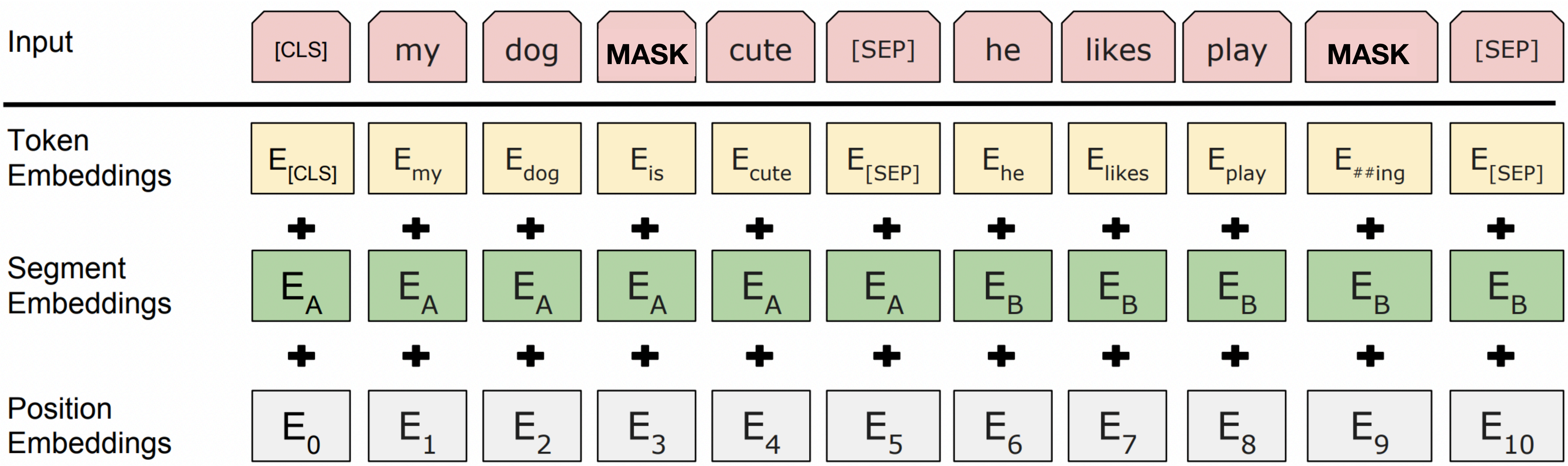
- Use specialised tokens CLS, SEP

Masked LM



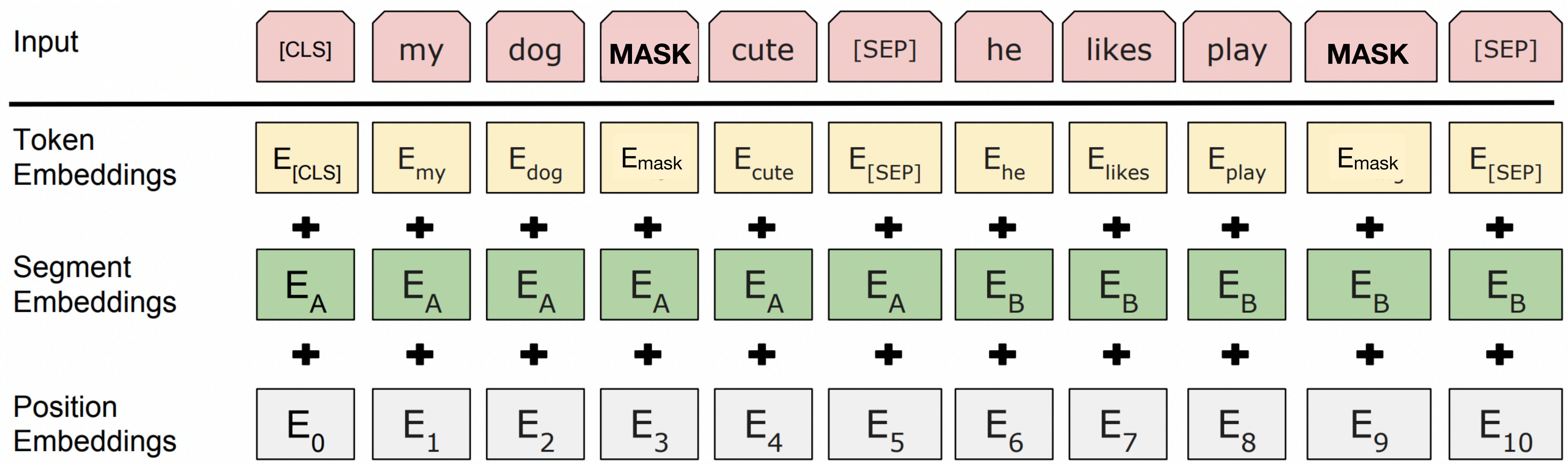
- Use specialised tokens CLS, SEP
- 15% of the tokens are randomly masked

Masked LM



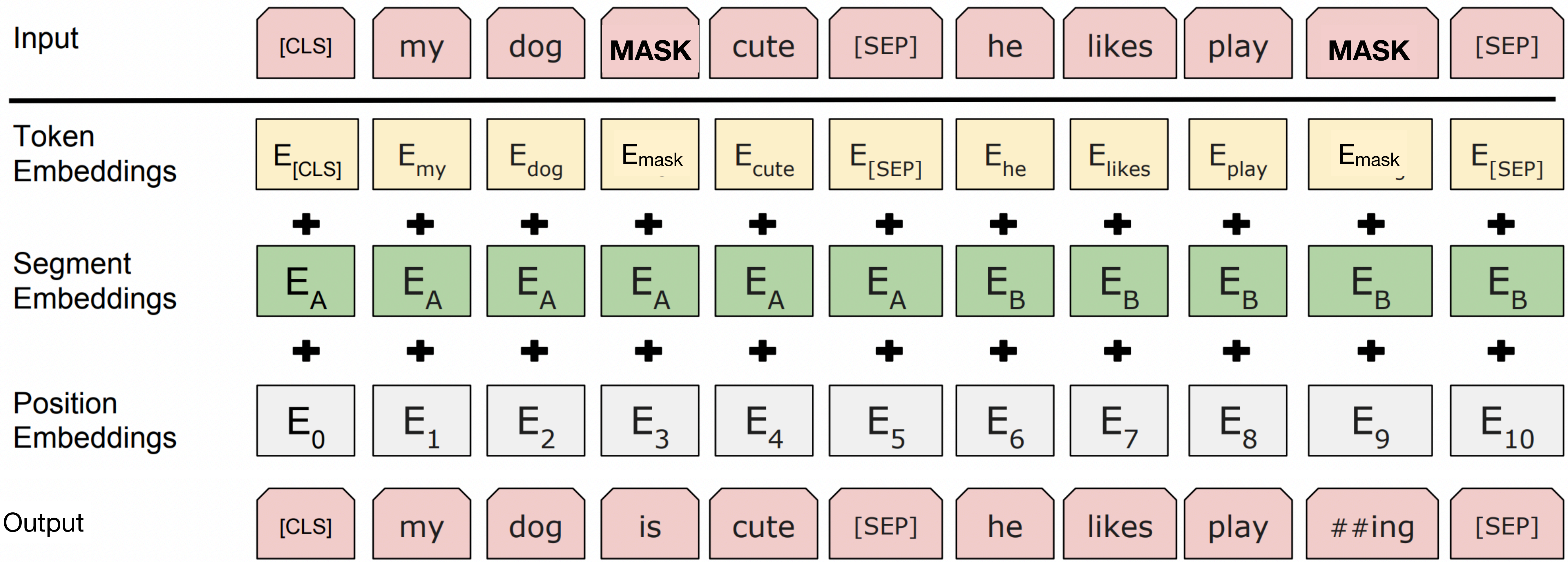
- Use specialised tokens CLS, SEP
- 15% of the tokens are randomly masked

Masked LM



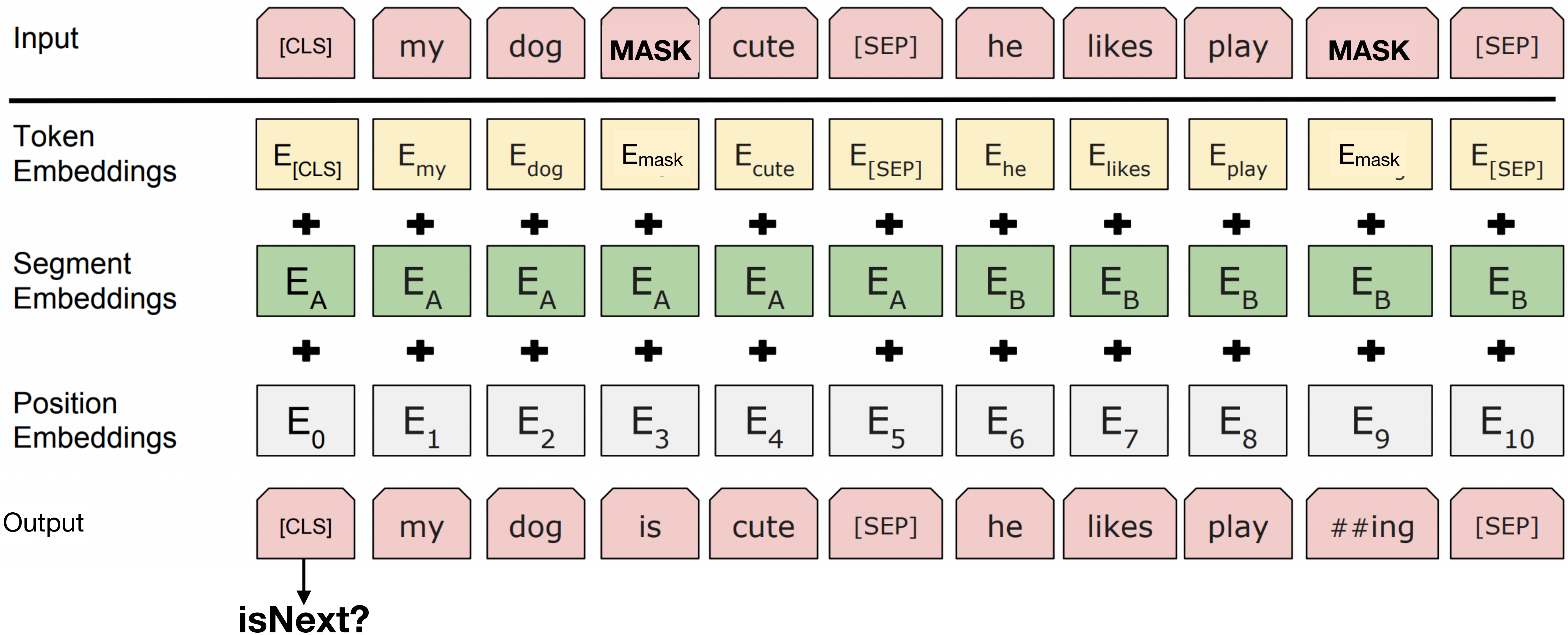
- Use specialised tokens CLS, SEP
- 15% of the tokens are randomly masked

Masked LM



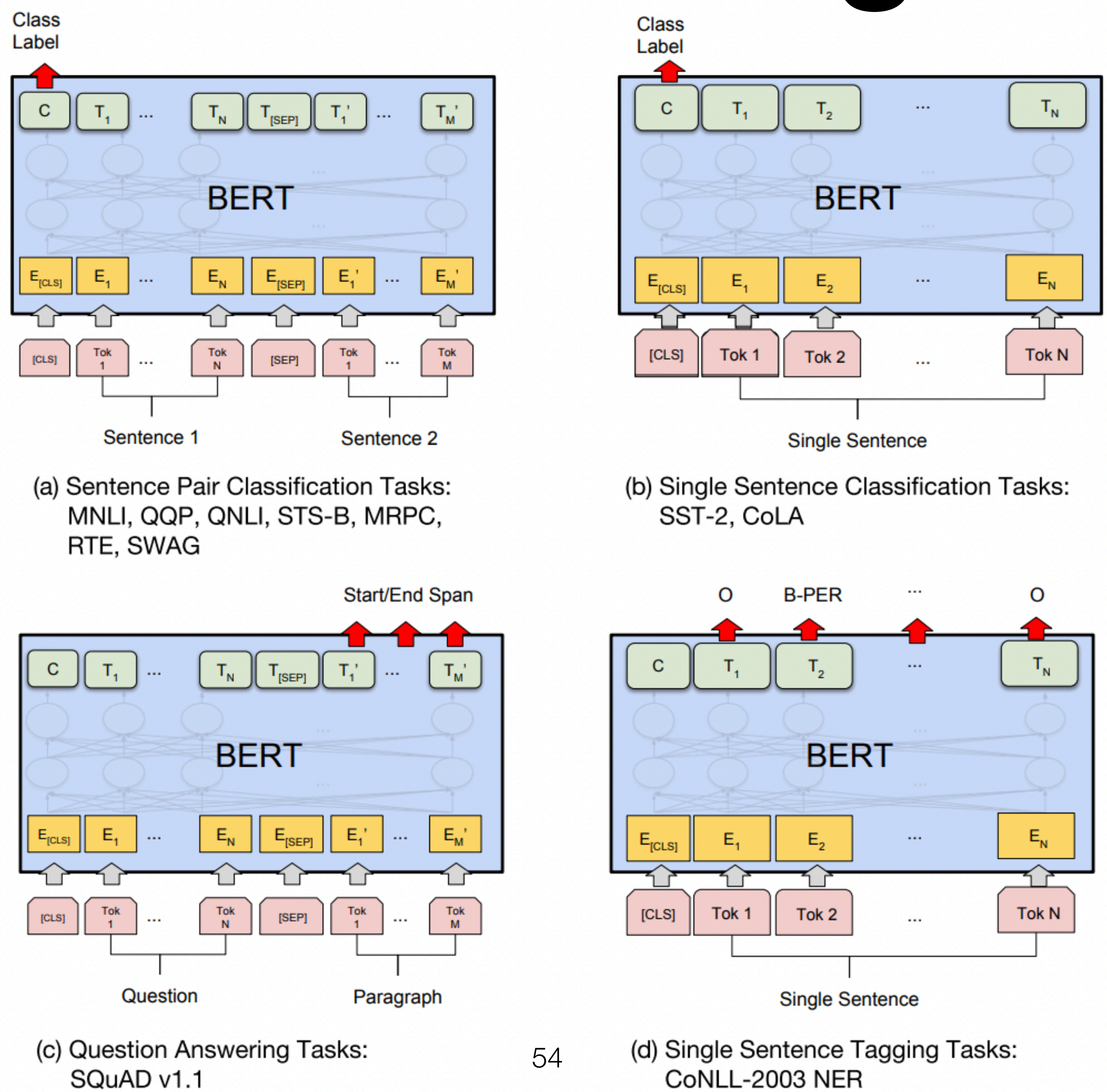
- Use specialised tokens CLS, SEP
- 15% of the tokens are randomly masked

Next Sentence Prediction



Use the CLS output embedding to predict is sentence B is the next sentence or not.

Fine-tuning


Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Glue Test Results

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

[Jacob Devlin et al 2018]

Reading Task

- Read this article: https://medium.com/@samia.khalid/bert-explained-a-complete-guide-with-theory-and-tutorial-3ac9ebc8fa7c

- Prepare questions to discuss!

- Time: 10 mins

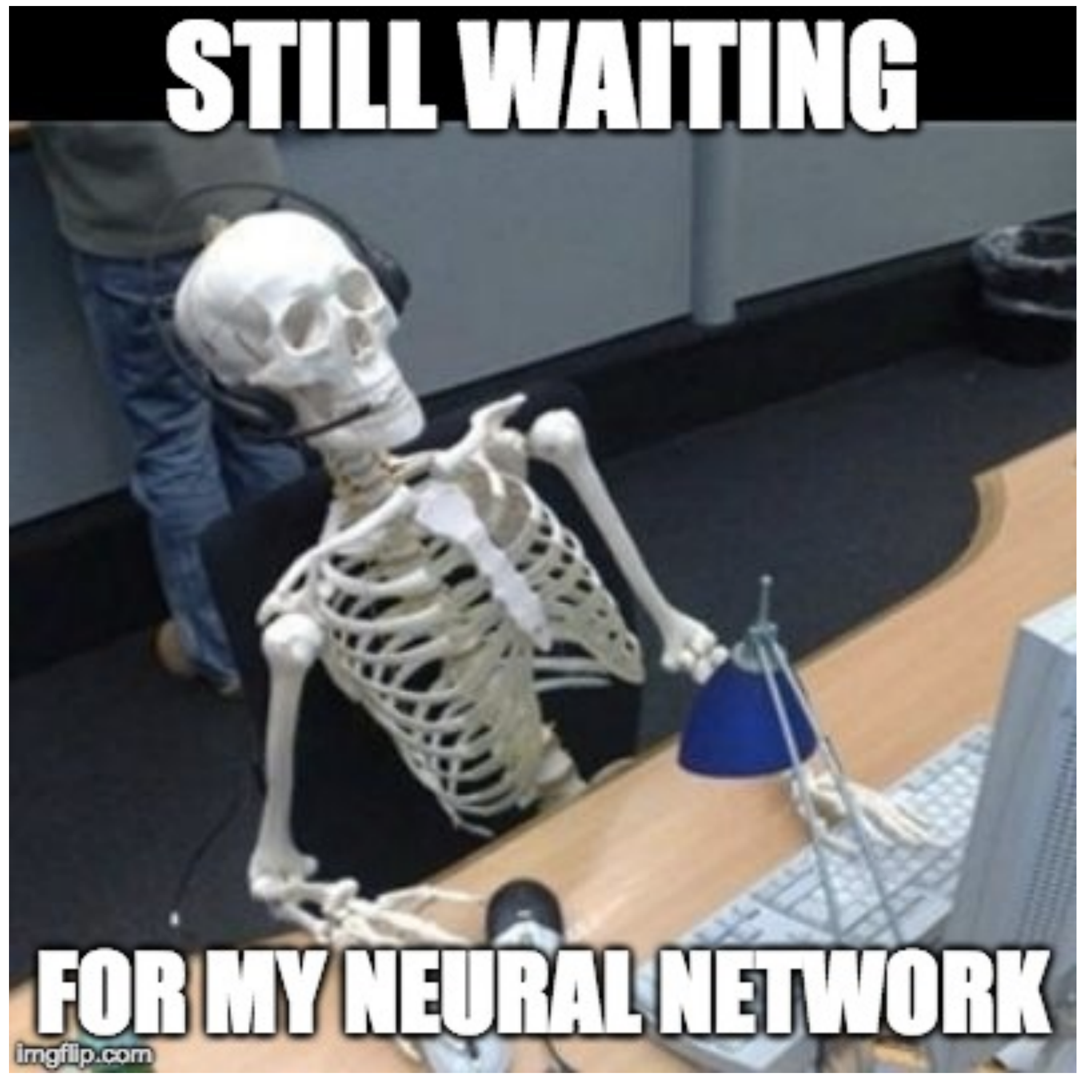
Summary

- NNLM:
  - LSTMs
  - Transformers
    - Self Attention
    - BERT
- Challenges
  - Long-Term Dependencies
  - Class-based output layer
  - Rare Words

Further Reading

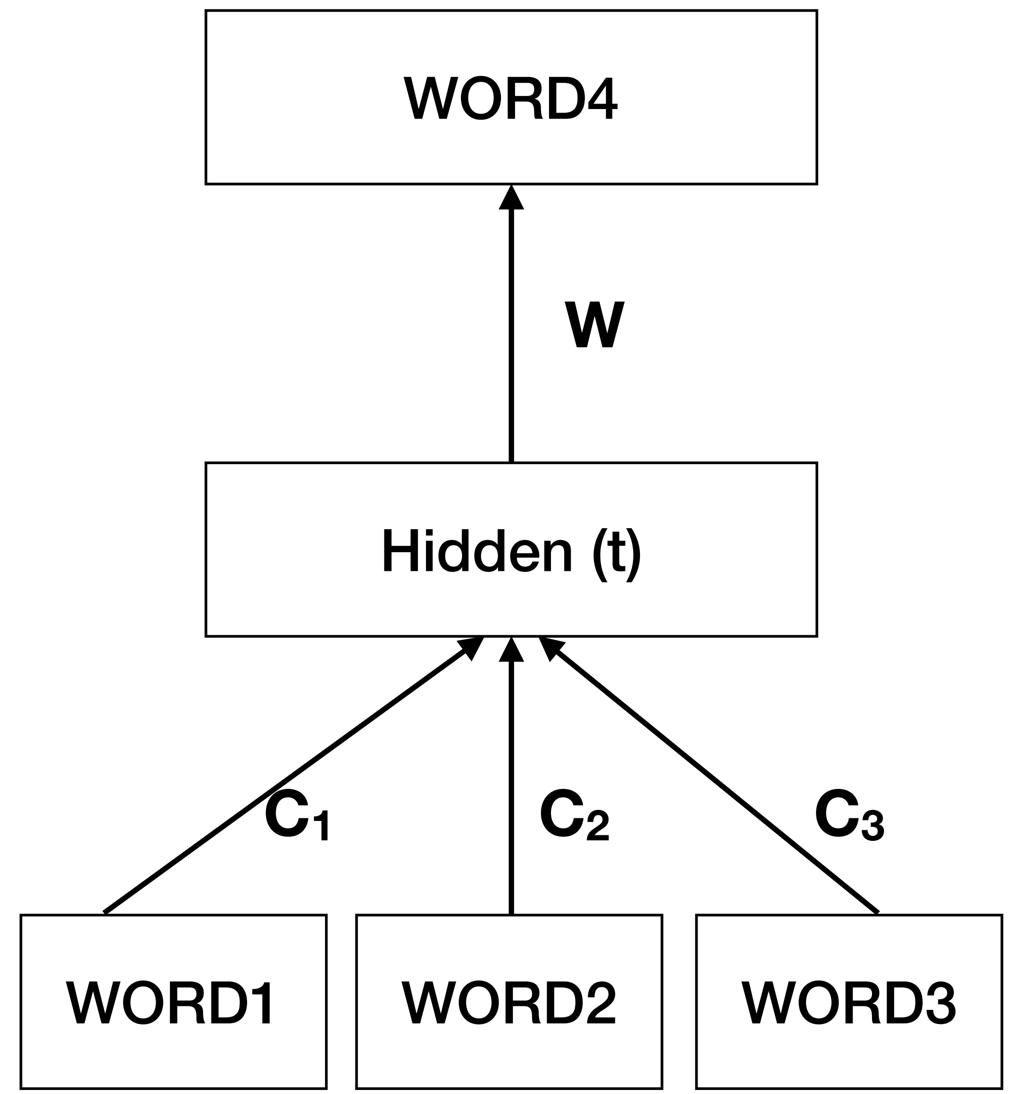
- Neural Networks and Neural Language Models: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Network Training



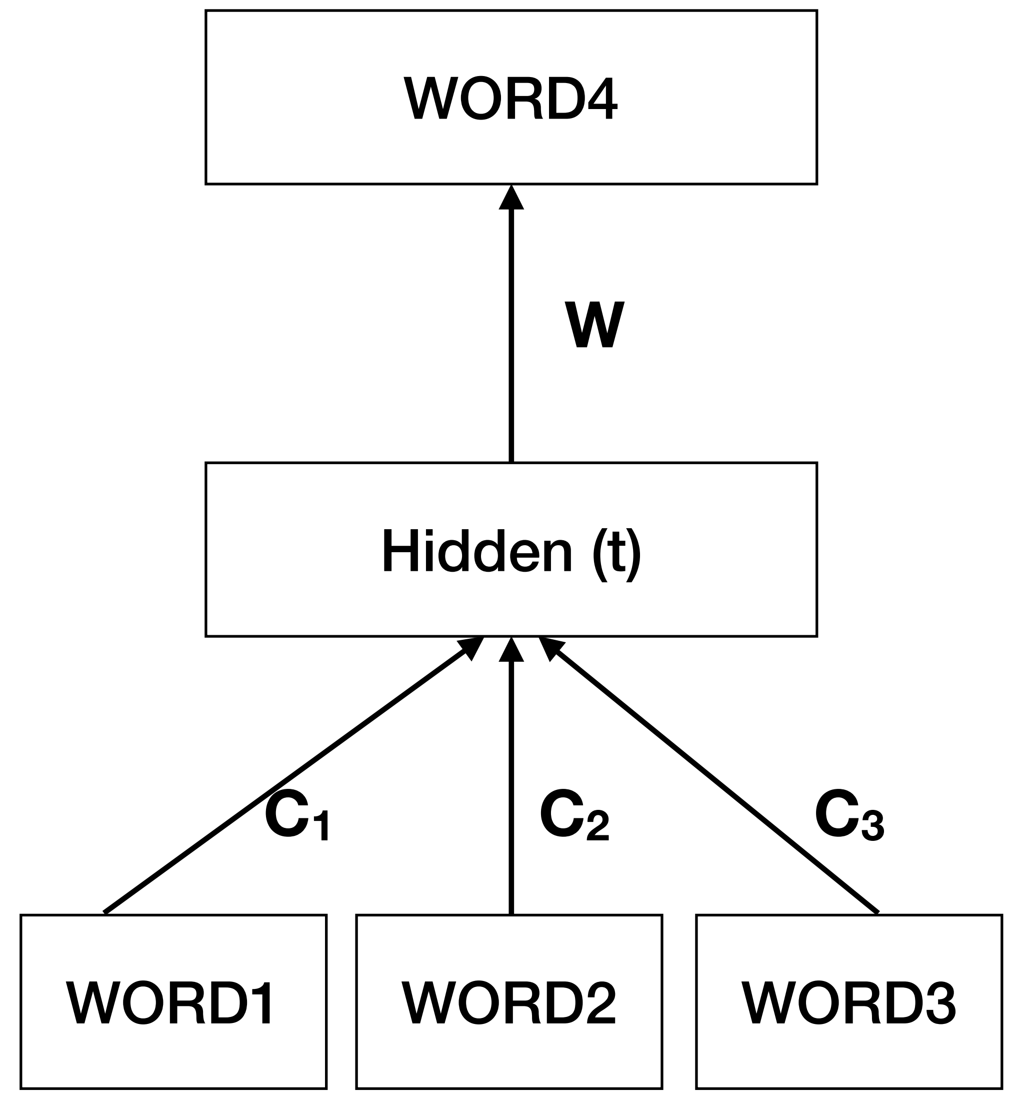
Softmax Normalization

- Slowest part of training an NNLM is softmax normalization



Softmax Normalization

- Slowest part of training an NNLM is softmax normalization
- Why?



Softmax Normalization

- Slowest part of training an NNLM is softmax normalization
- Why?
- Before the softmax layer (final layer) we just have a real number, not a probability

Softmax Normalization

- Slowest part of training an NNLM is softmax normalization
- Why?
- Before the softmax layer (final layer) we just have a real number, not a probability
- So we need to know the sum of scores for all possible words being predicted (ie. the normalization constant)

$$p(w_i|h) = \frac{e^{w_i^T \cdot h}}{Z}$$

$$Z = \sum_{k=1}^{|V|} e^{w_k^T \cdot h}$$

Softmax Normalization

- Slowest part of training an NNLM is softmax normalization
- Why?
- Before the softmax layer (final layer) we just have a real number, not a probability
- So we need to know the sum of scores for all possible words being predicted (ie. the normalization constant)
- This involves |V| steps, where |V| is the size of the vocabulary

$$p(w_i|h) = \frac{e^{w_i^T \cdot h}}{Z}$$

$$Z = \sum_{k=1}^{|V|} e^{w_k^T \cdot h}$$

Softmax Normalization

- Slowest part of training an NNLM is softmax normalization
- Why?
- Before the softmax layer (final layer) we just have a real number, not a probability
- So we need to know the sum of scores for all possible words being predicted (ie. the normalization constant)
- This involves |V| steps, where |V| is the size of the vocabulary
- Typical values of |V| are between 10K to 10M

$$p(w_i|h) = \frac{e^{w_i^T \cdot h}}{Z}$$

$$Z = \sum_{k=1}^{|V|} e^{w_k^T \cdot h}$$

Softmax Normalization

- Slowest part of training an NNLM is softmax normalization

- Why?

- Before the softmax layer (final layer) we just have a real number, not a probability

- So we need to know the sum of scores for all possible words being predicted (ie. the normalization constant)

$$p(w_i|h) = \frac{e^{w_i^T \cdot h}}{Z}$$

$$Z = \sum_{k=1}^{|V|} e^{w_k^T \cdot h}$$

- This involves |V| steps, where |V| is the size of the vocabulary

- Typical values of |V| are between 10K to 10M

- We must do this for every word in our training set (eg. 1M-1B), every epoch (> 10)

Class-based Output Layer

- Class-based output is calculated as:

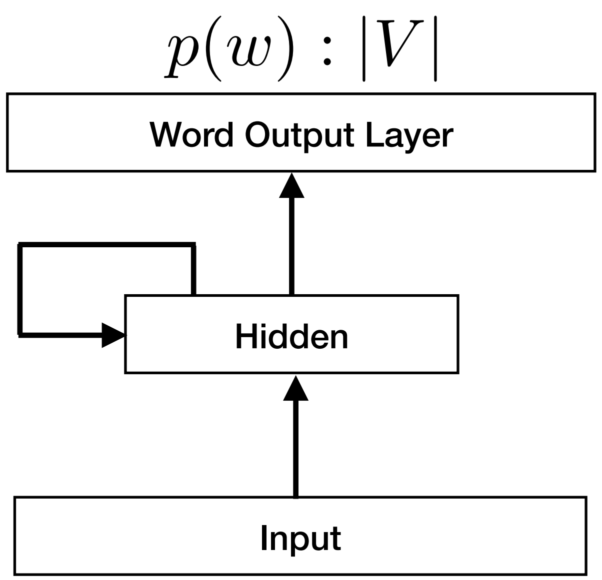
$$p(w) = \underbrace{p(w|C(w))}_{\text{membership probability}} \times \underbrace{p(C(w))}_{\text{class probability}}$$

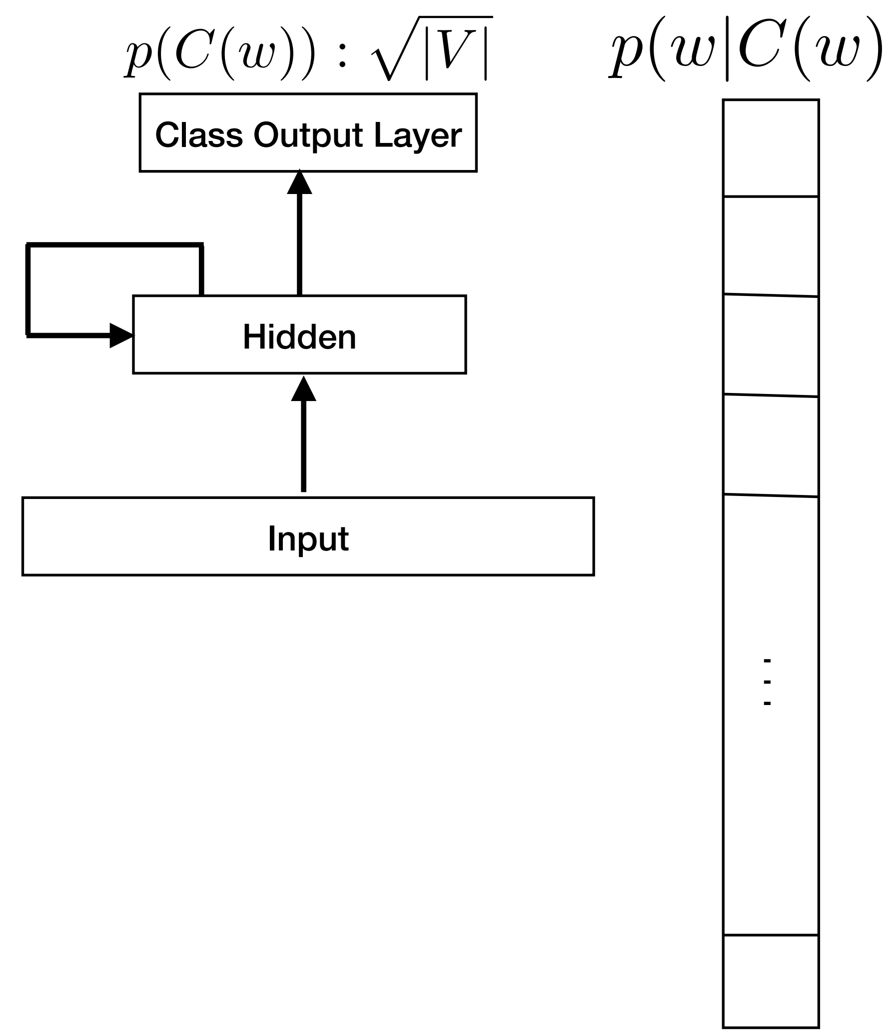
Class-based Output Layer

- Class-based output is calculated as:

$$p(w) = p(w|C(w)) \times p(C(w))$$

membership probability class probability



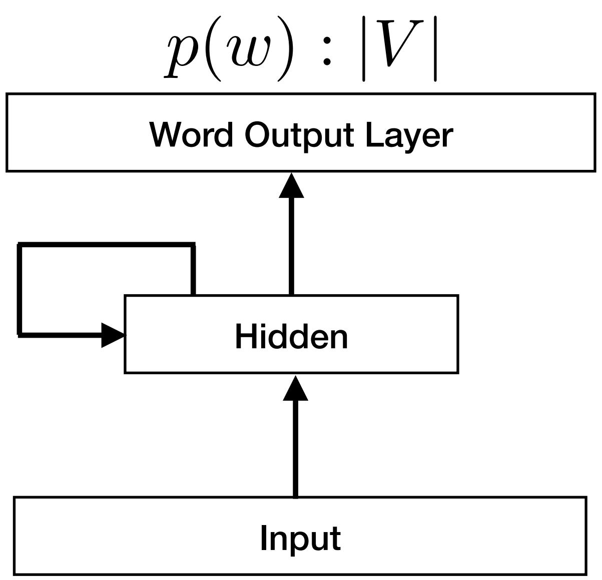


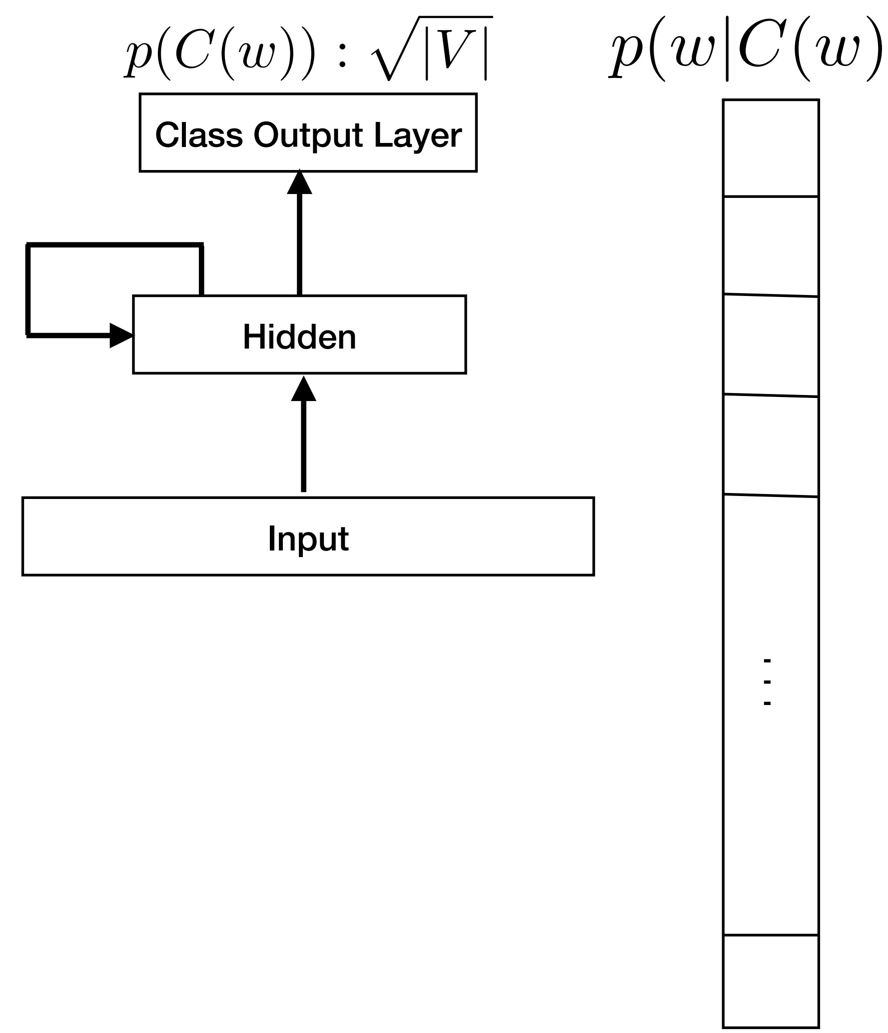
Class-based Output Layer

- Class-based output is calculated as:

$$p(w) = p(w|C(w)) \times p(C(w))$$

membership probability   class probability



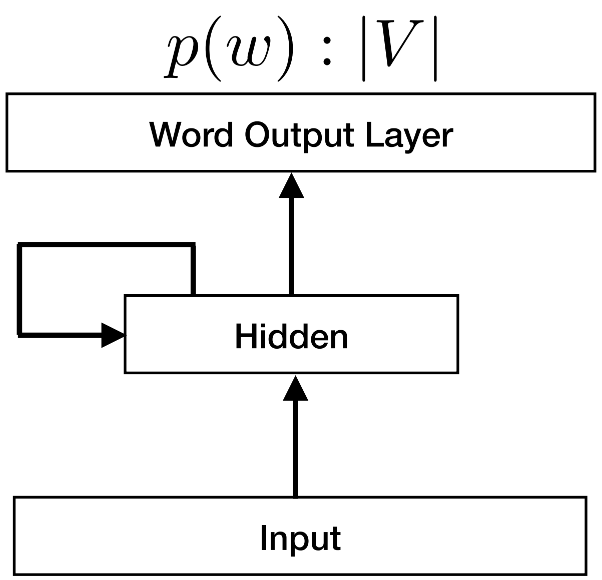


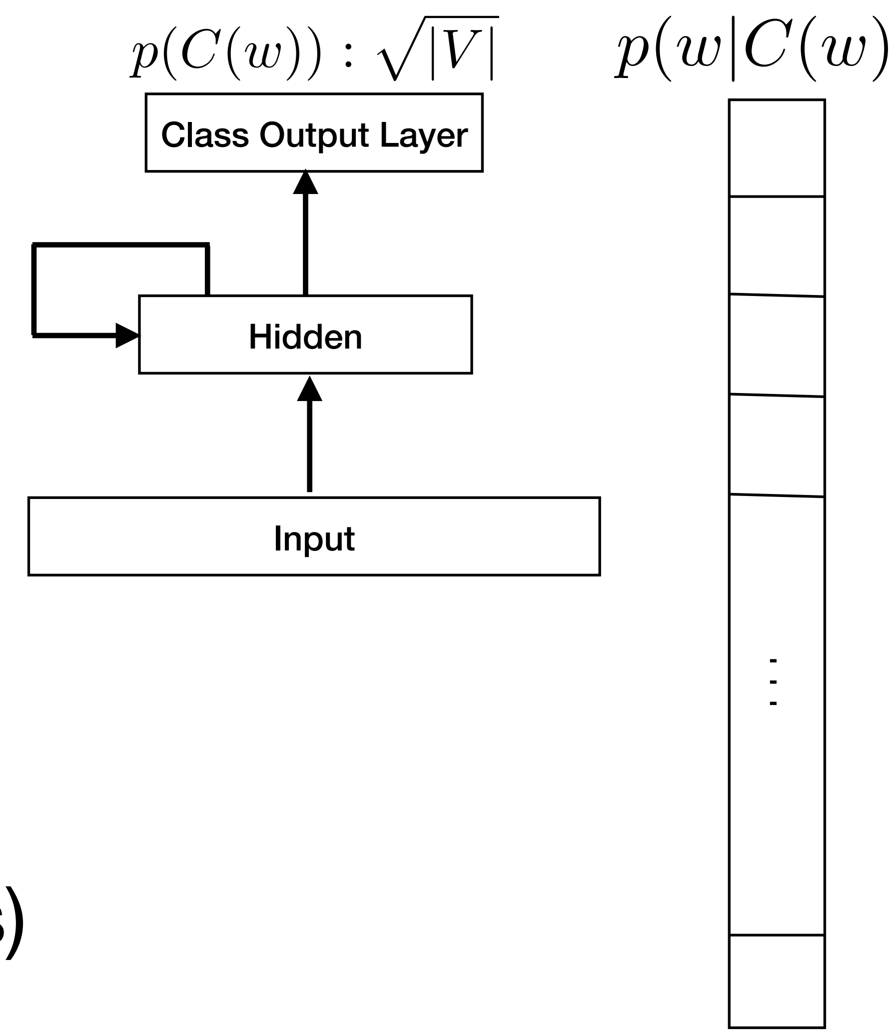
- Softmax: N × 4M (Vocabulary)

Class-based Output Layer

- Class-based output is calculated as:

$$p(w) = \underbrace{p(w|C(w))}_{\text{membership probability}} \times \underbrace{p(C(w))}_{\text{class probability}}$$





- Softmax: N × 4M (Vocabulary)

- Class-based Output: N × 2K (Classes)

Speeding up Normalization

Speeding up Normalization

- Class-based Decomposition $O(\sqrt{|V|})$

Speeding up Normalization

- Class-based Decomposition $O(\sqrt{|V|})$
- Noise Contrastive Estimation (NCE) $O(1)$

Speeding up Normalization

- Class-based Decomposition $O(\sqrt{|V|})$
- Noise Contrastive Estimation (NCE) $O(1)$
- Hierarchical Softmax $O(\log_2 |V|)$

Speeding up Normalization

- Class-based Decomposition $O(\sqrt{|V|})$
- Noise Contrastive Estimation (NCE) $O(1)$
- Hierarchical Softmax $O(\log_2 |V|)$
- Self Normalization ensures that the normalization constant Z is close to one. Slow for training, fast for test-time queries $O(1)$

Question

- What kind of words will be tough to predict by neural network language model? And what can help the NNLM to predict these?

Rare Words

- Rare words: words which occur with low frequency
  - Out-of-Vocabulary words (OOVs: Frequency 0 in training set)
  - Frequency is 1
- Problem learning good feature vectors for such words
  - Not enough data
- A predictive system is not able handle them well
  - One would want automatic Video Subtitling Systems to predict these words as well

Rare Words

Languages	
Finnish	
Swedish	
Arabic	
English	

Rare Words

Languages	Training Set Size
Finnish	170M
Swedish	130M
Arabic	460M
English	790M

Rare Words

Languages	Training Set Size	Vocabulary
Finnish	170M	4.2M
Swedish	130M	3.5M
Arabic	460M	1.3M
English	790M	760K

Rare Words

Languages	Training Set Size	Vocabulary	Rare Words (f≤1)
Finnish	170M	4.2M	55 %
Swedish	130M	3.5M	56 %
Arabic	460M	1.3M	55 %
English	790M	760K	41 %

Rare Words

Languages	Training Set Size	Vocabulary	Rare Words (f≤1)	Rare Words (f≤5)
Finnish	170M	4.2M	55 %	82 %
Swedish	130M	3.5M	56 %	82 %
Arabic	460M	1.3M	55 %	78 %
English	790M	760K	41 %	70 %

Rare Words

Languages	Training Set Size	Vocabulary	Rare Words (f≤1)	Rare Words (f≤5)
Finnish	170M	4.2M	55 %	82 %
Swedish	130M	3.5M	56 %	82 %
Arabic	460M	1.3M	55 %	78 %
English	790M	760K	41 %	70 %

Rare words form a large portion of the vocabulary

OOVs

- Morphologically-rich languages have a lot of OOVs

OOVs

- Morphologically-rich languages have a lot of OOVs
    - Creating a large training corpus can still lead to OOV rate ~ 2%

OOVs

- Morphologically-rich languages have a lot of OOVs
  - Creating a large training corpus can still lead to OOV rate ~ 2%
- Some languages (low-resource) can have OOV rates as high as 40%

Subwords

- Segment words into subwords
  - Words: This sentence simply has words

Subwords

- Segment words into subwords
  - Words: This sentence simply has words
  - Morphemes: Th is sent ence in to morph eme s

Subwords

- Segment words into subwords
  - Words: This sentence simply has words
  - Morphemes: Th is sent ence in to morph eme s
  - Characters: T h i s o n e i n t o c h a r a c t e r s

Summary

- NNLM:
  - LSTMs
  - Transformers
    - Self Attention
    - BERT
- Challenges
  - Long-Term Dependencies
  - Class-based output layer
  - Rare Words