Neural Network Language Models

Mittul Singh

Recap: N-gram Language Models

- N-gram language model

$$P(w_i|w_{i-1}, w_{i-2}, w_{i-3}, w_{i-4})$$

Neural Network Classifier for Language Modelling

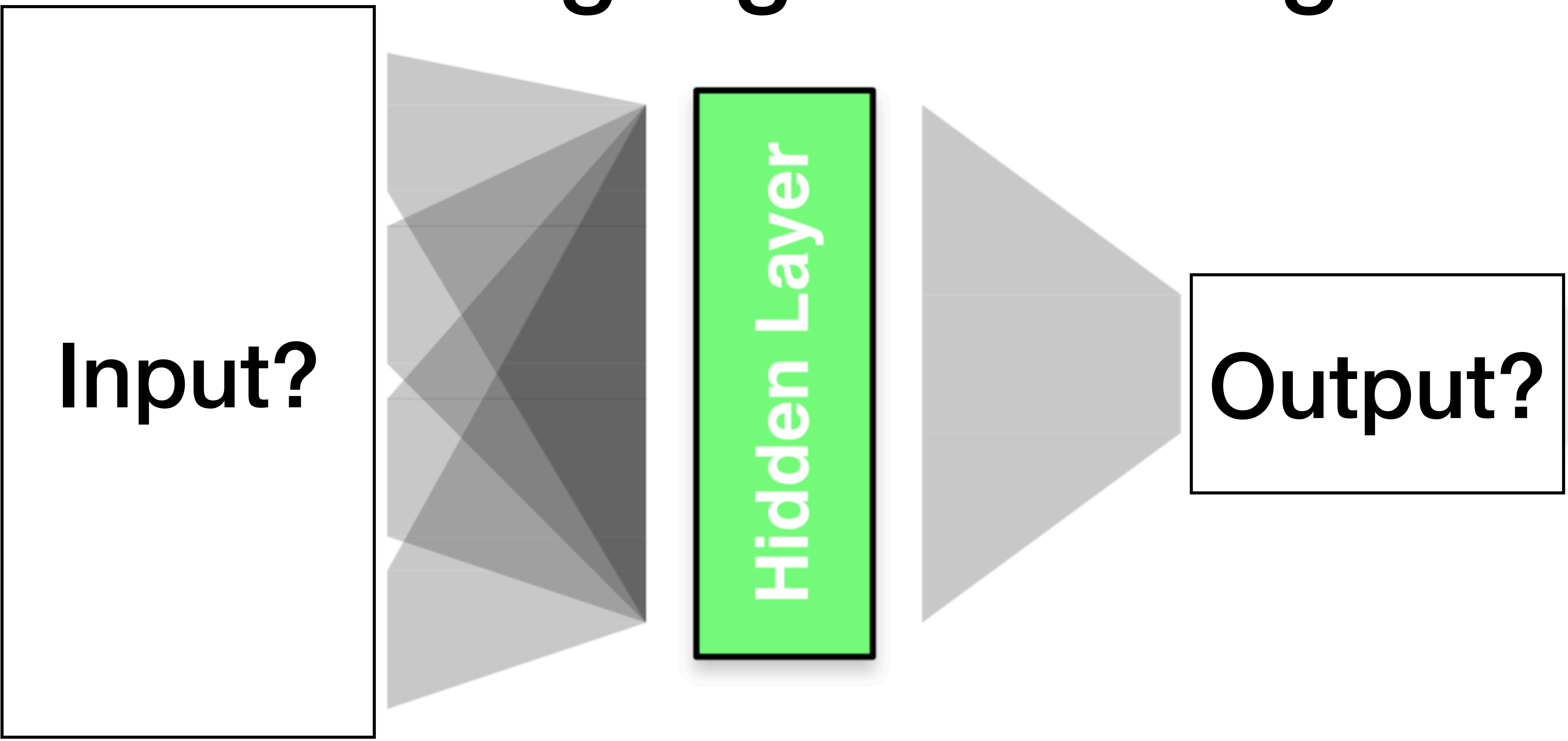


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

Neural Network Classifier for Language Modelling

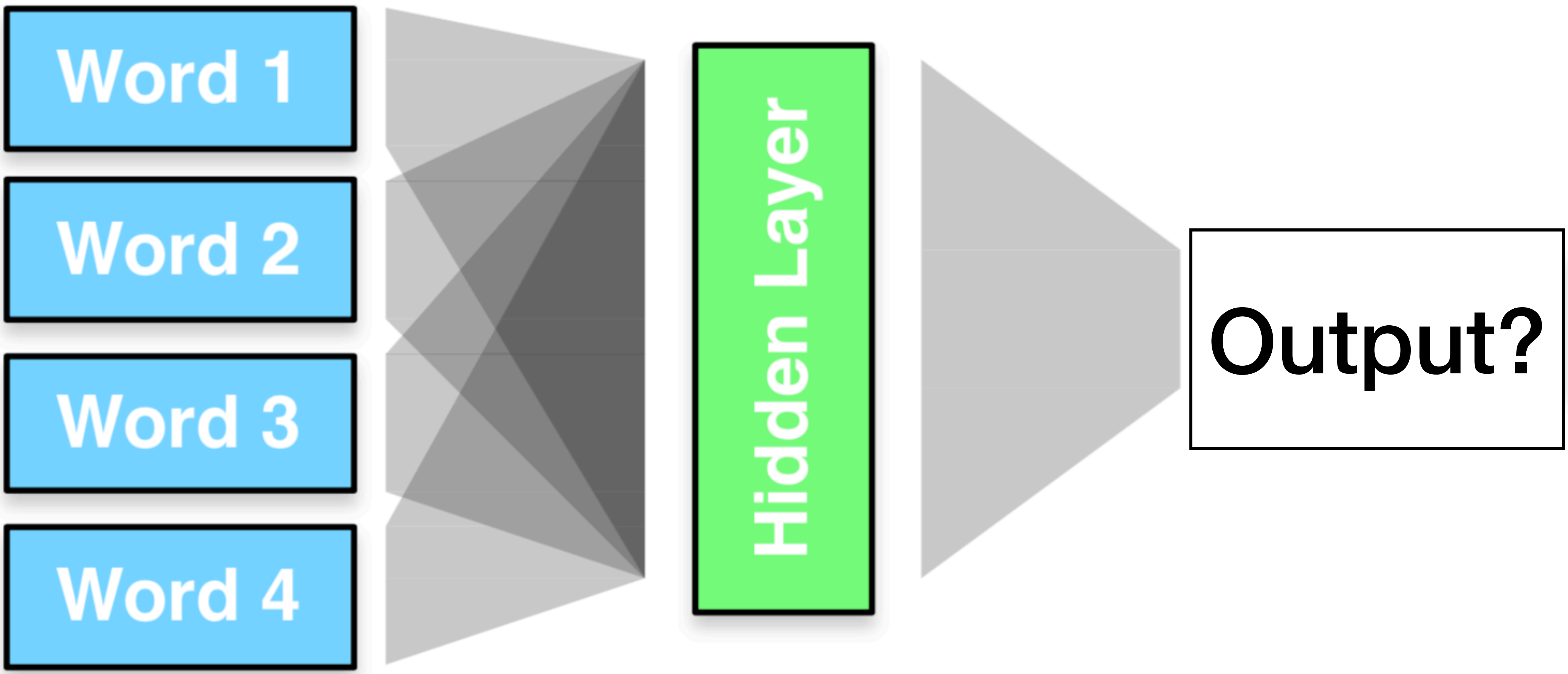


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

Neural Network Classifier for Language Modelling

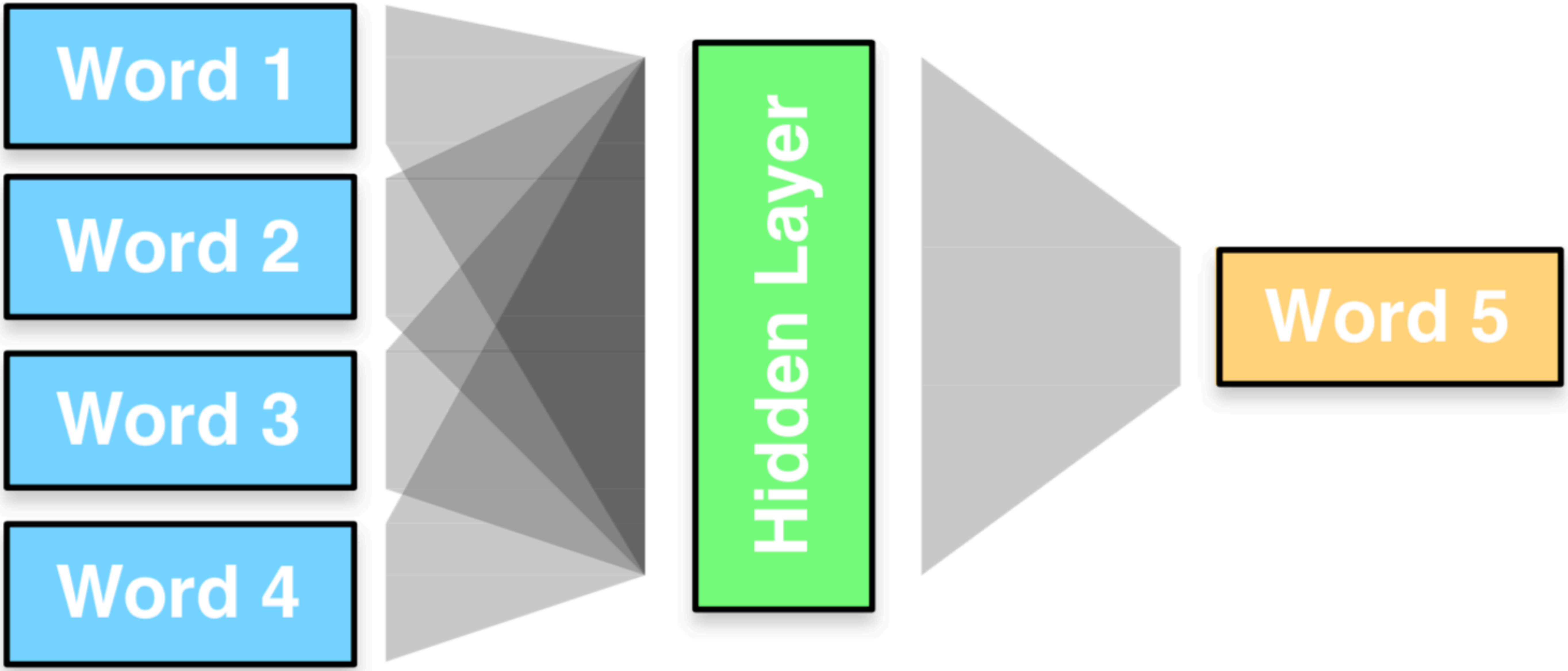


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

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

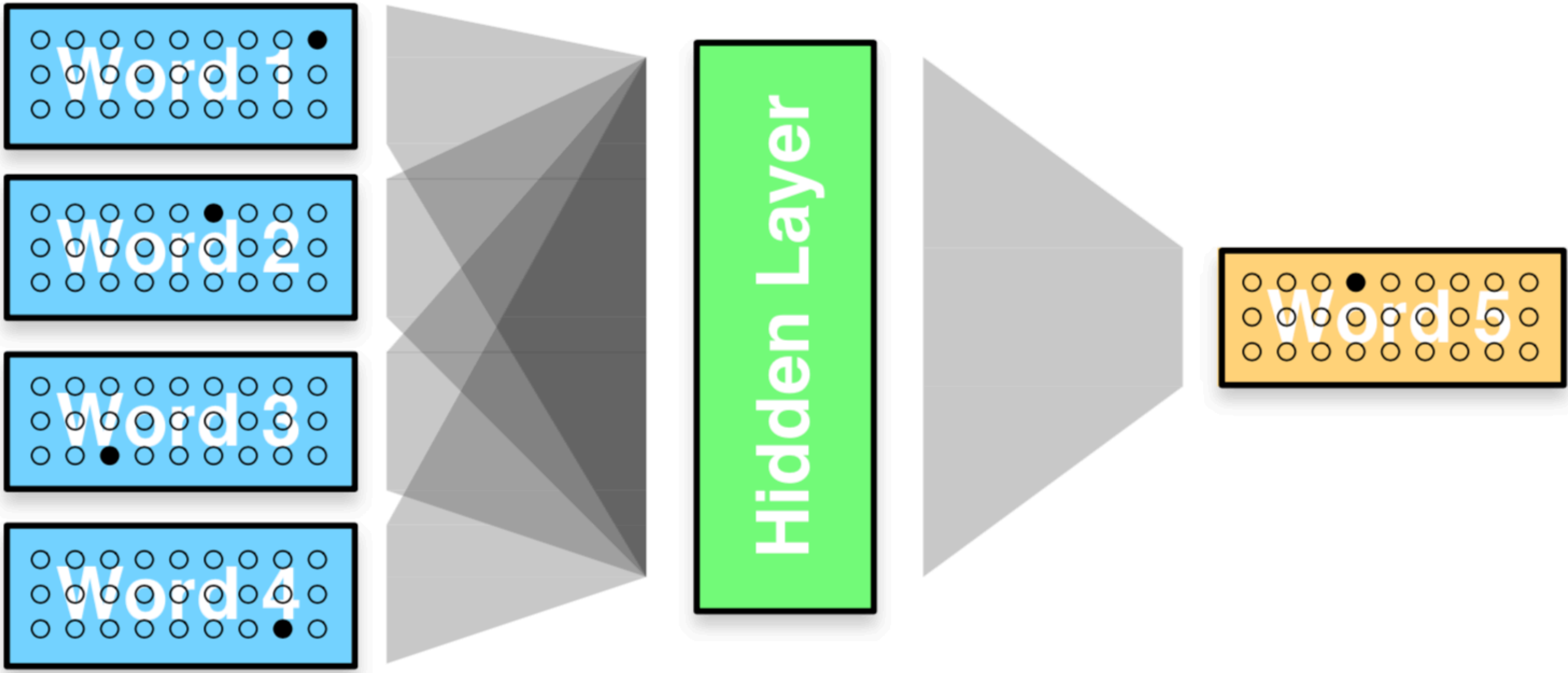
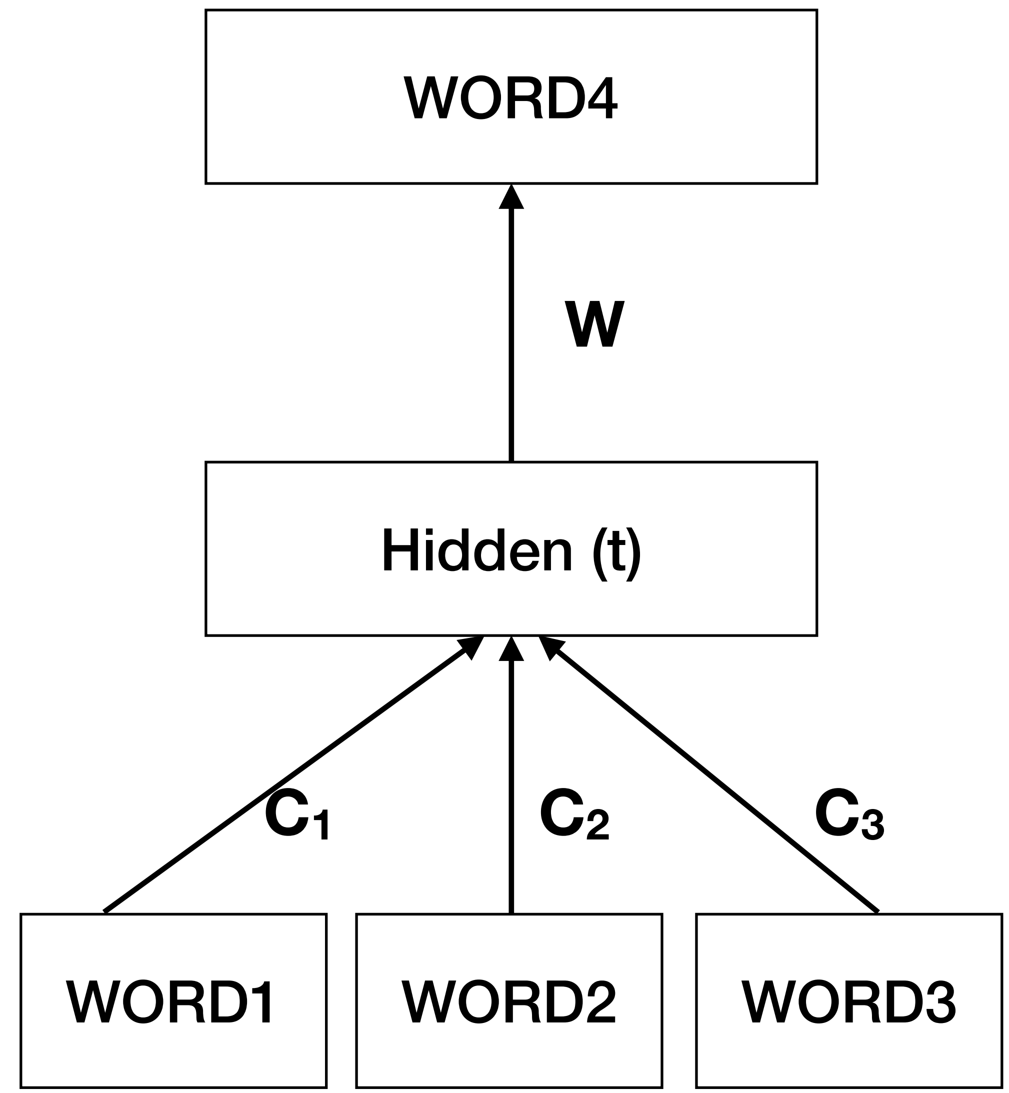


Image: http://mt-class.org/jhu/slides/lecture-nn-lm.pdf

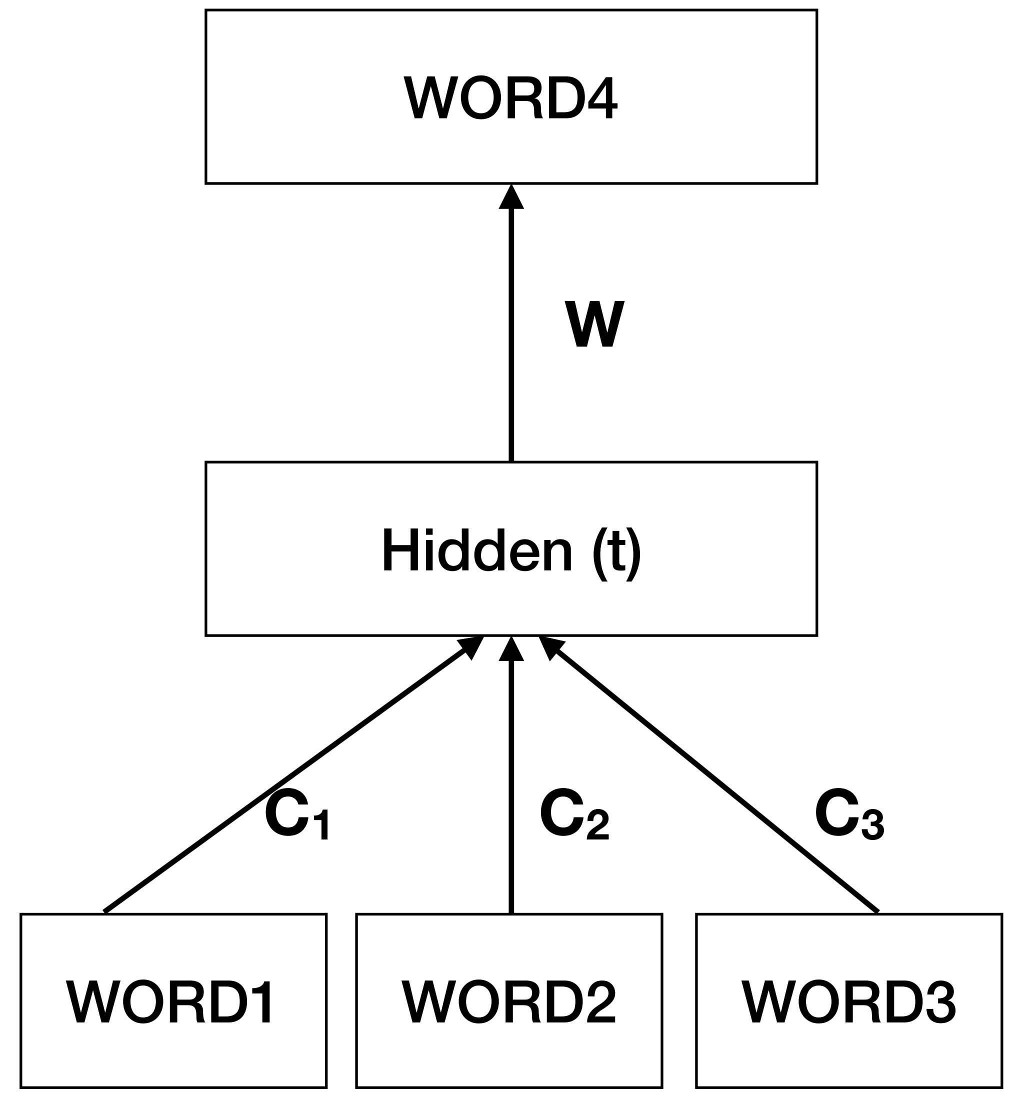
Feedforward Neural Network LM (FFNN)



FFNN

Feedforward Neural Network LM (FFNN)

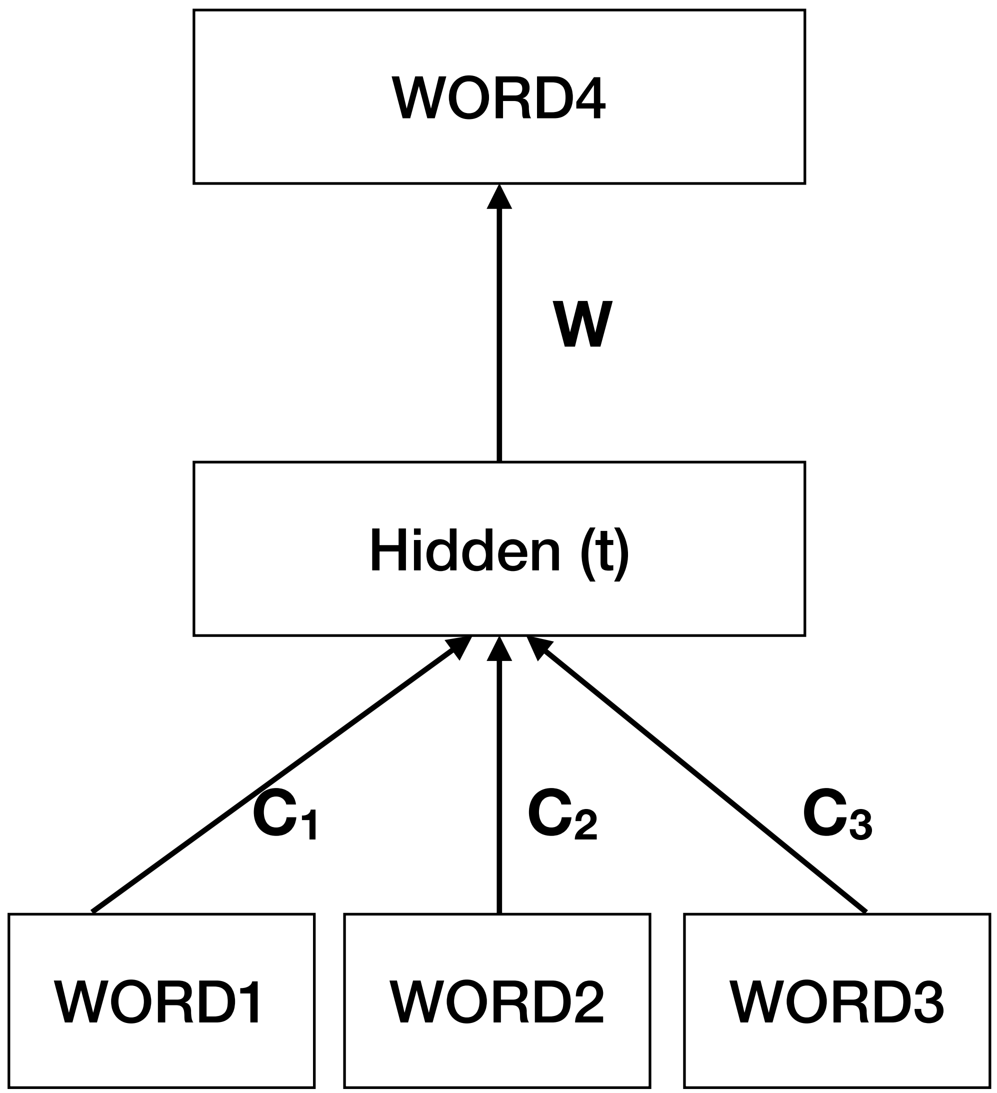
- Loop through the entire corpus



FFNN

Feedforward Neural Network LM (FFNN)

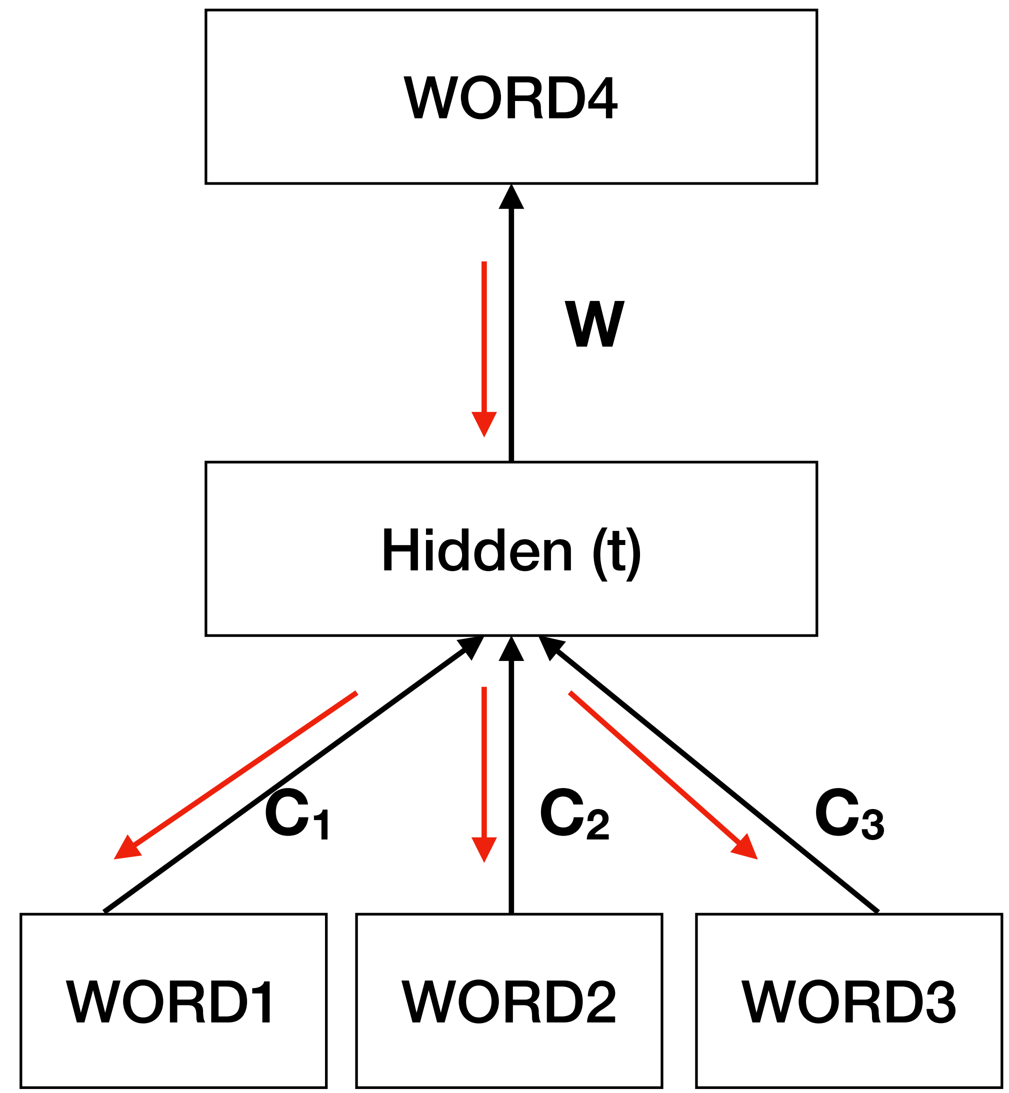
- Loop through the entire corpus
- Calculate error or loss (cross-entropy loss)



FFNN

Feedforward Neural Network LM (FFNN)

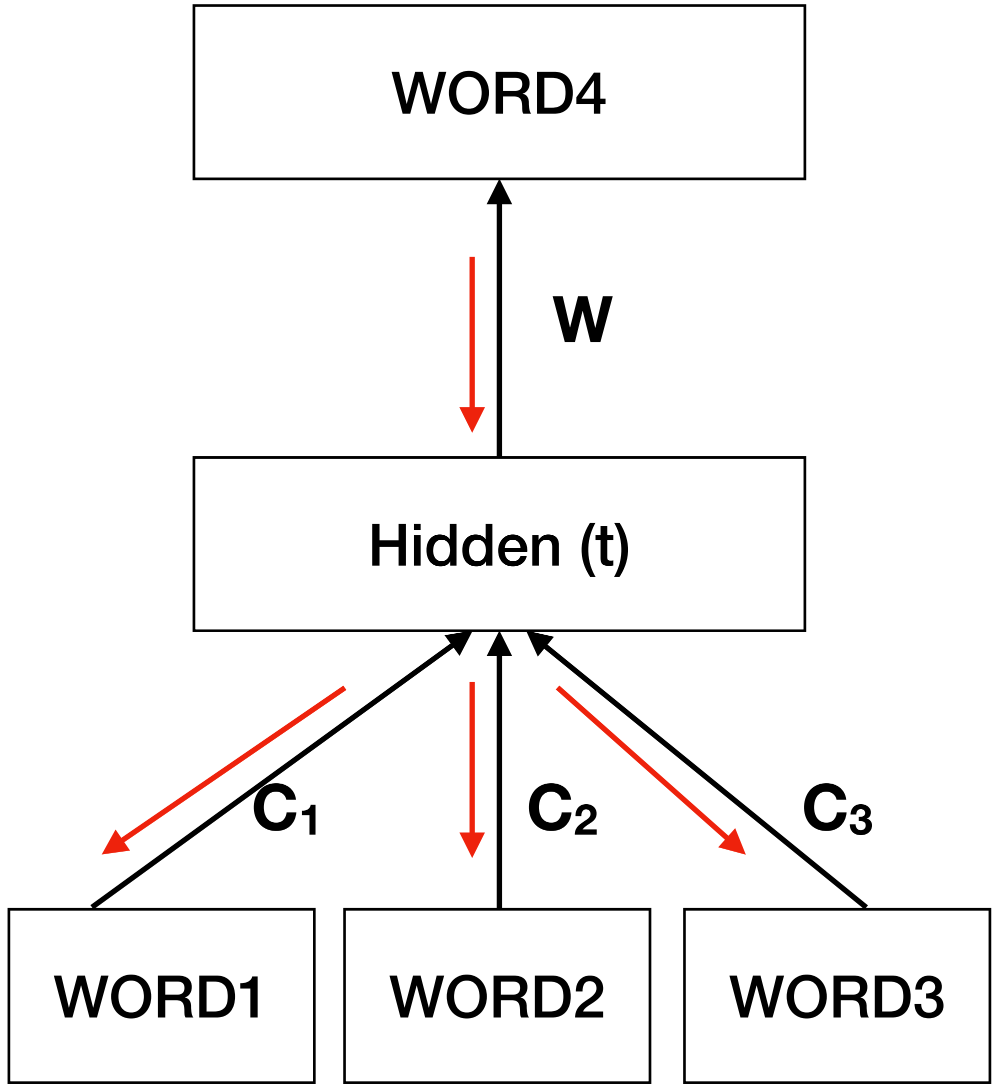
- Loop through the entire corpus
- Calculate error or loss (cross-entropy loss)
- Propagate the error through network to update the weight matrices



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 ________.”

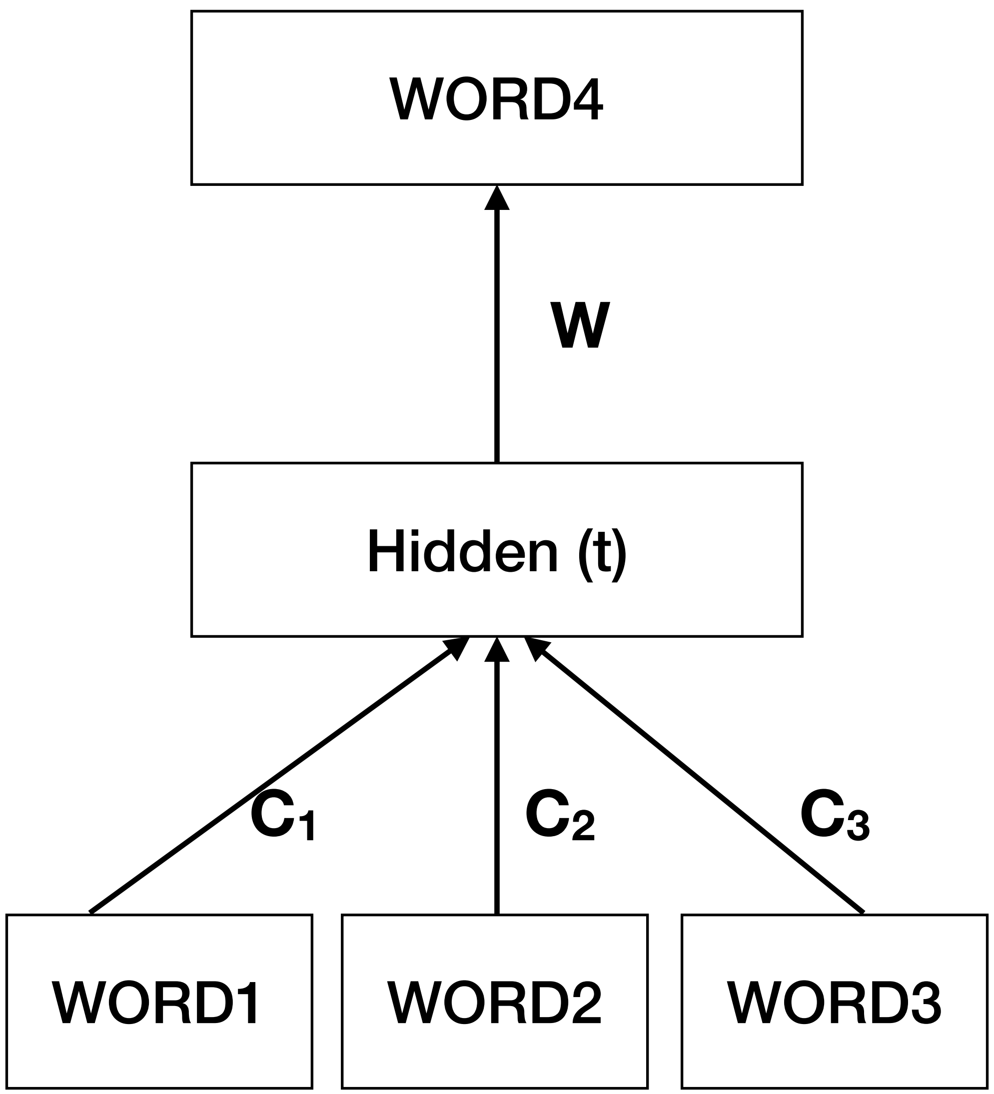
Feedforward: Long-term information

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

Feedforward: Long-term information

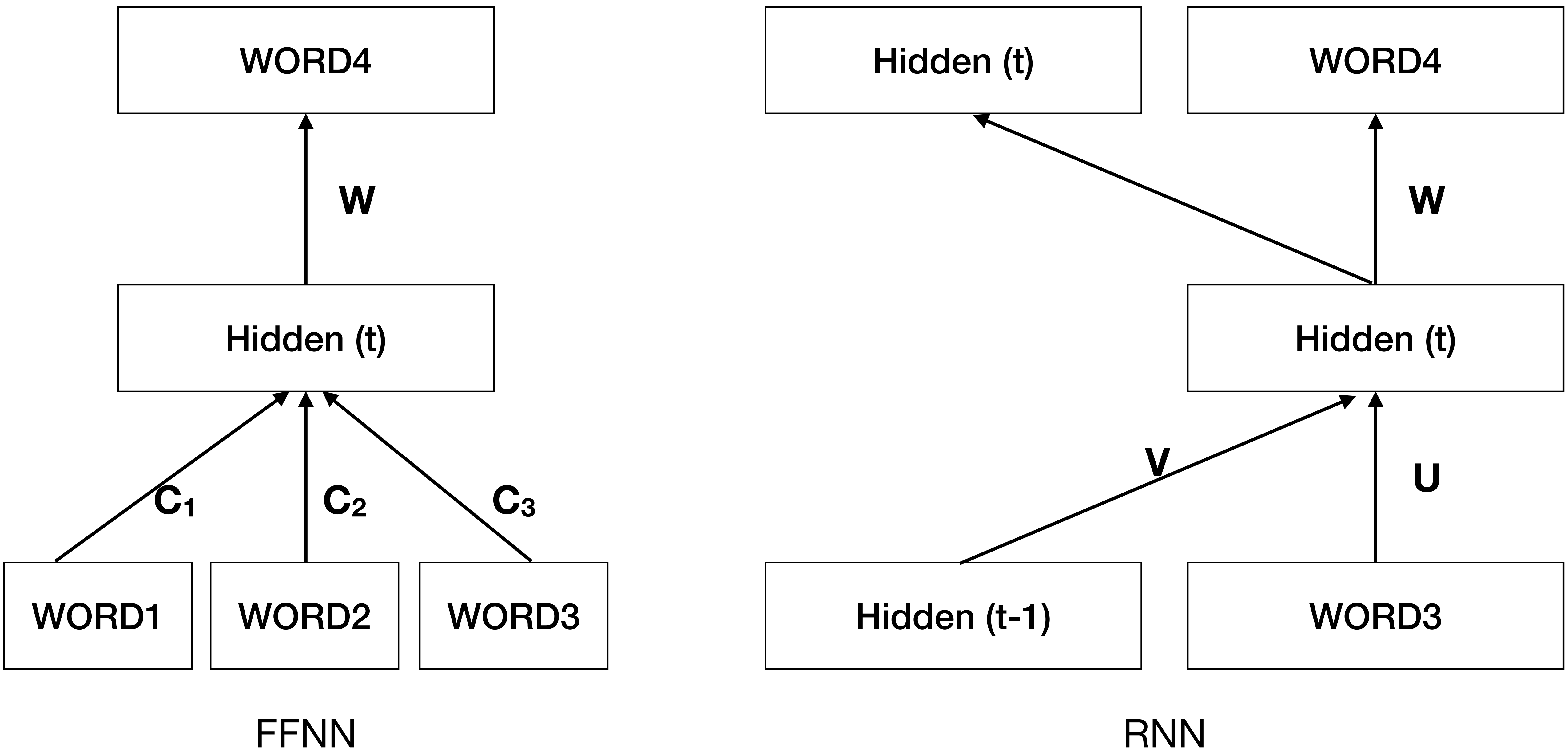
- “I grew up in France… I speak fluent _*French*_.”
- Feedforward Neural Network (FFNN) has limited context size

Recurrent Neural Networks (RNN)

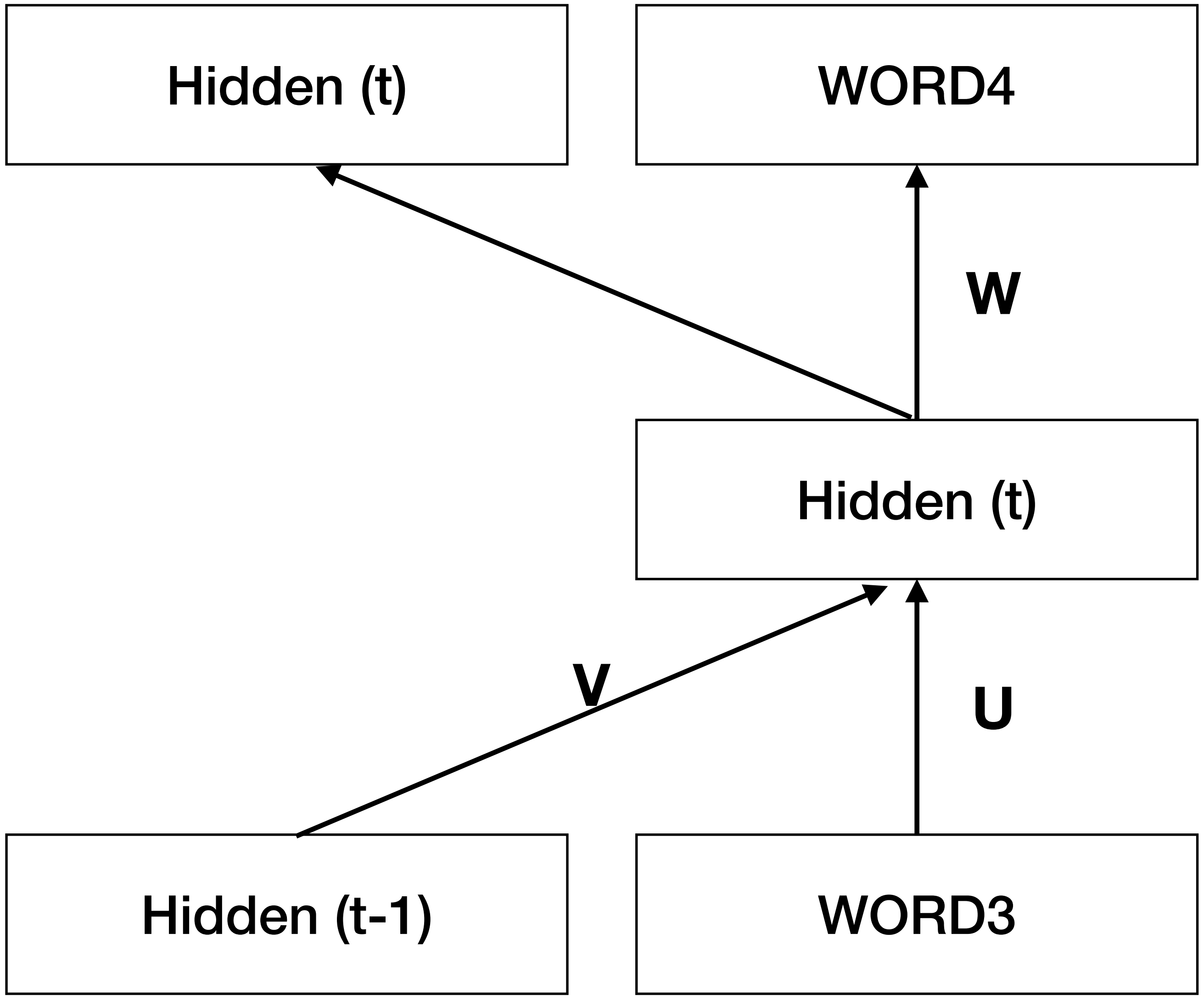


FFNN

Recurrent Neural Networks (RNN)

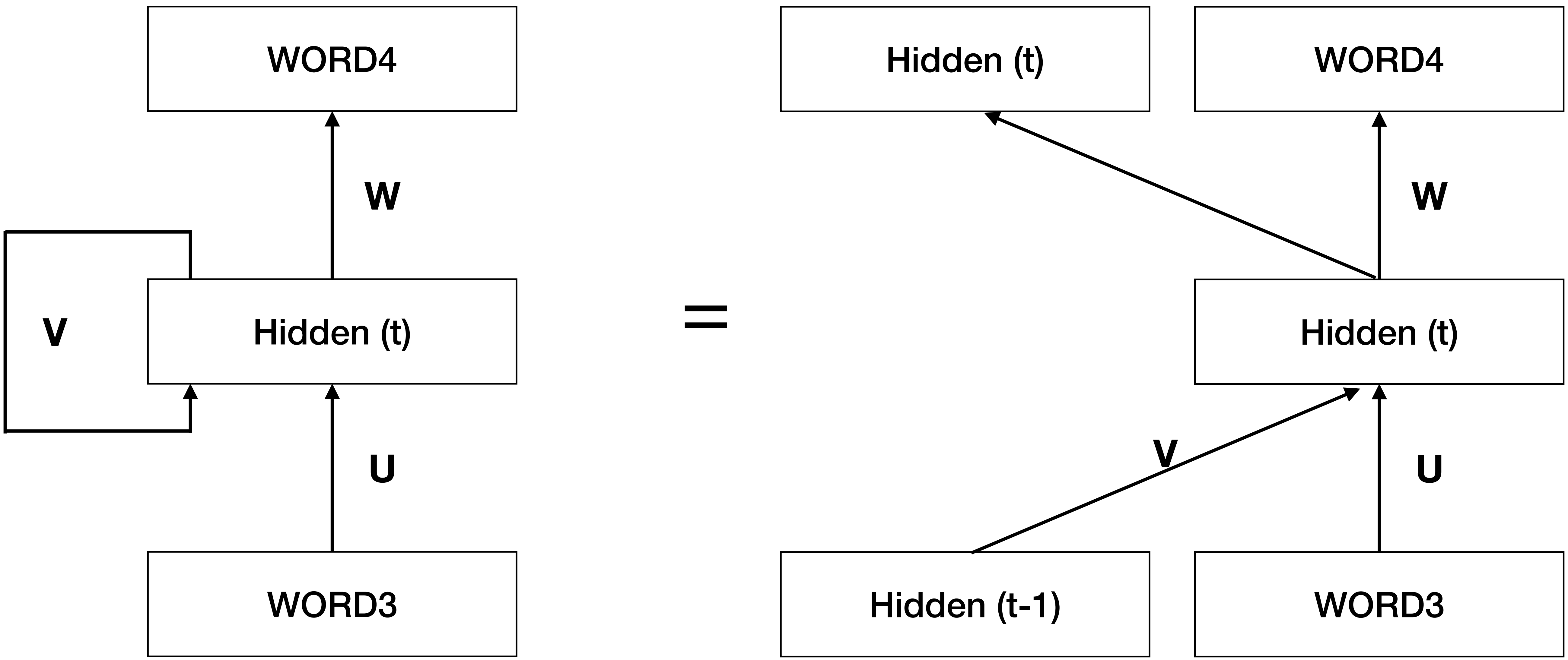


Recurrent Neural Networks (RNN)



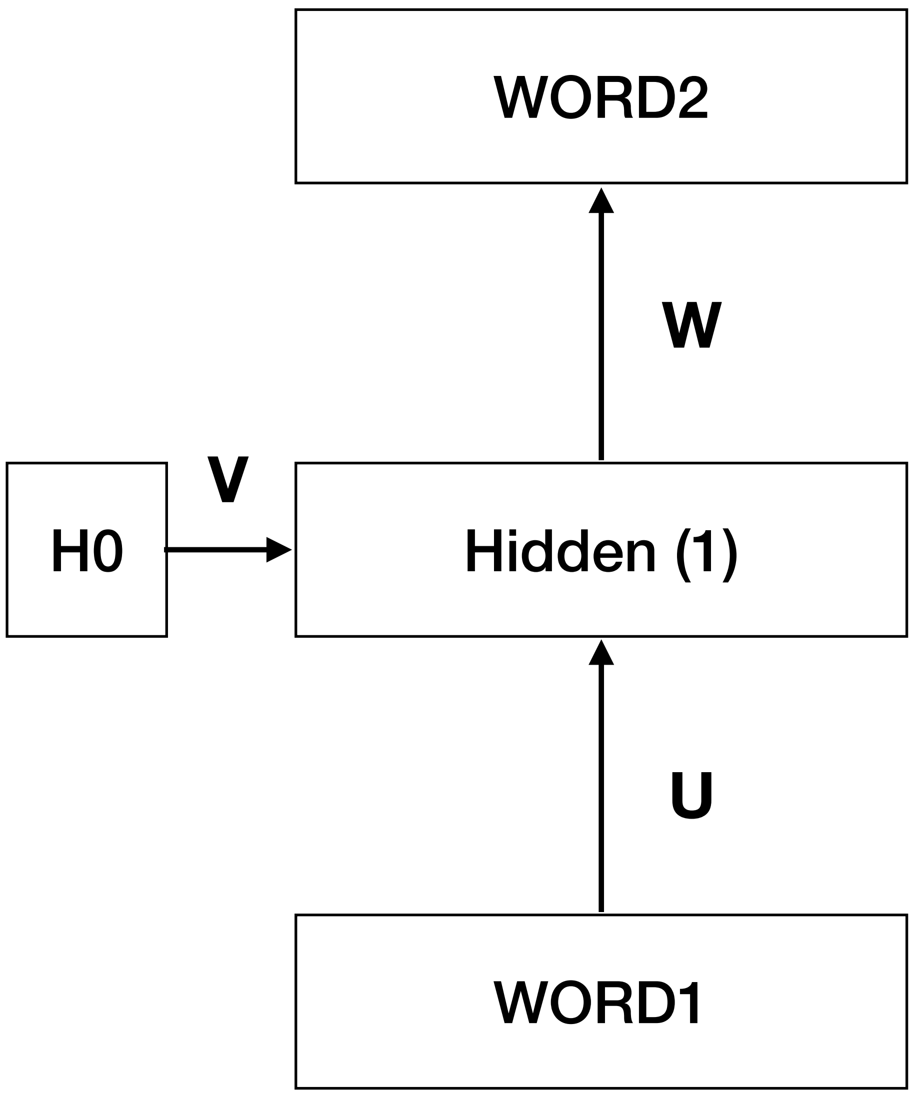
RNN

Recurrent Neural Networks (RNN)

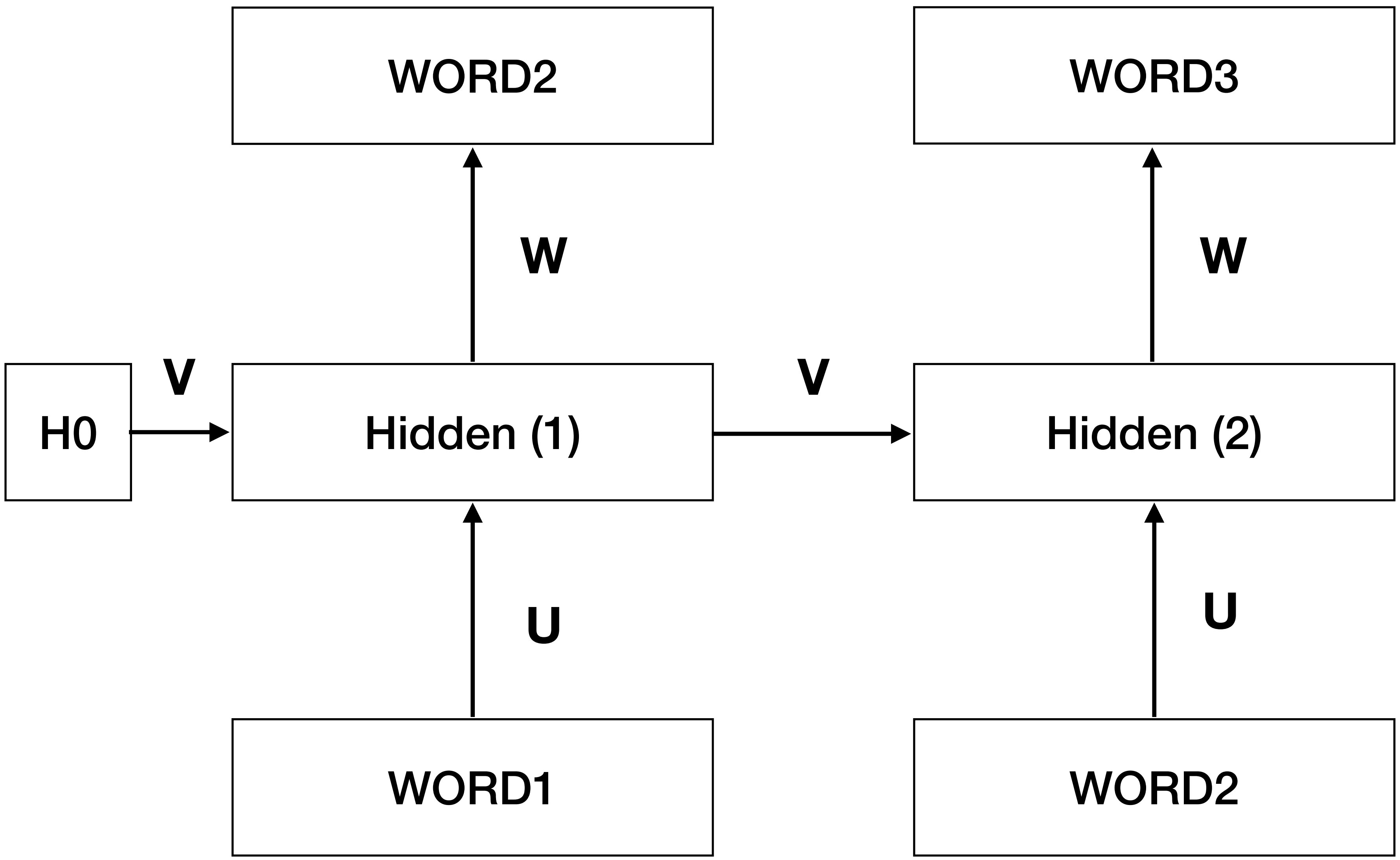


RNN

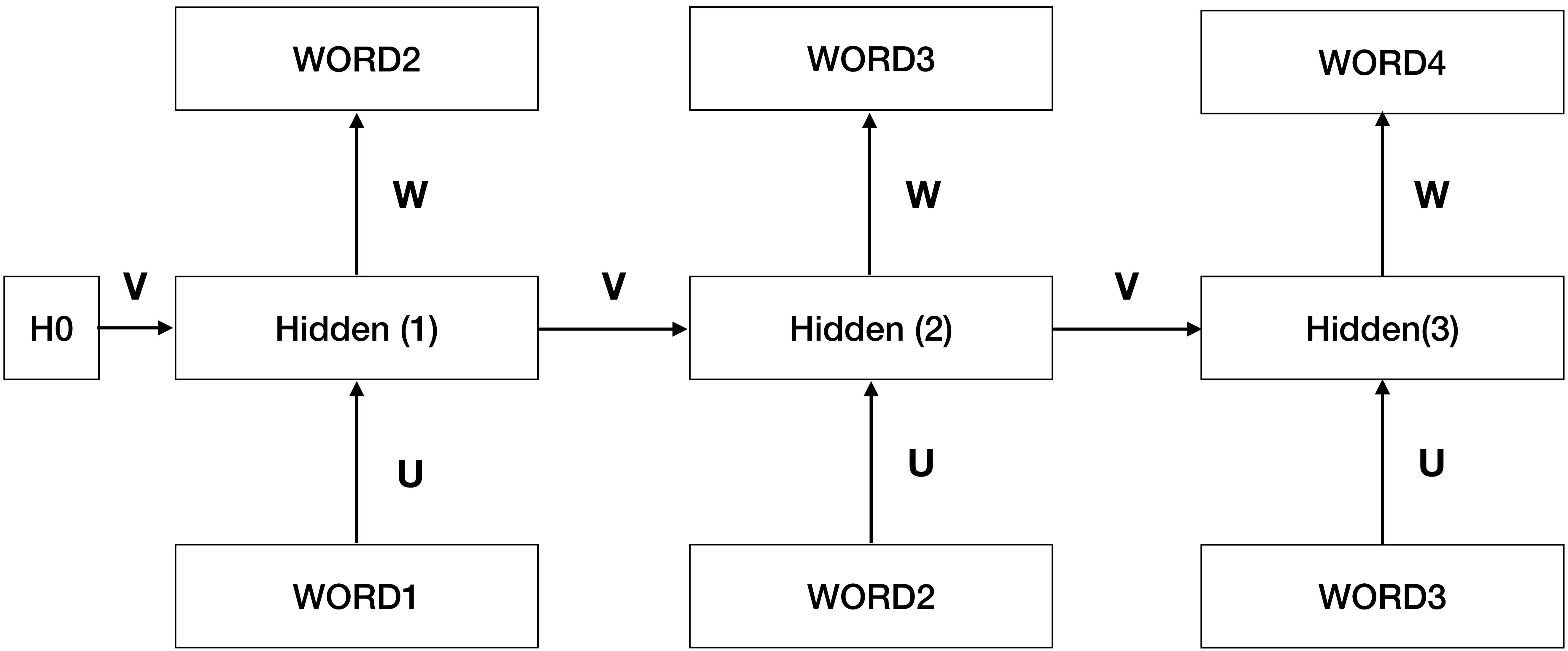
RNN: Timestep 1



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_."

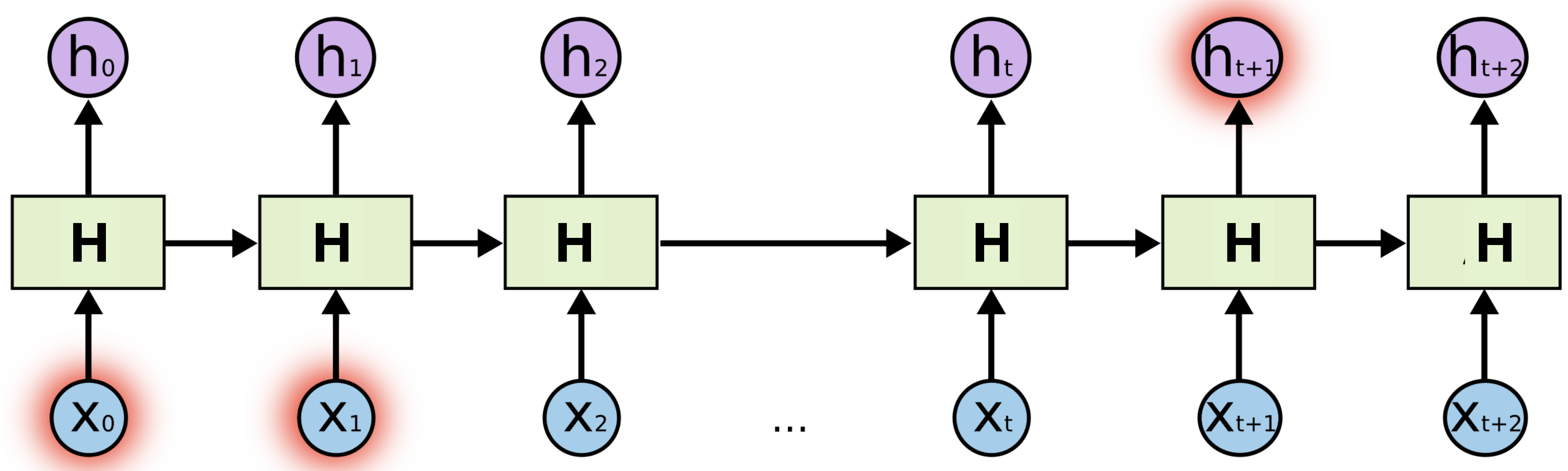


Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

- "I grew up in France… I speak fluent _French_."
- As the gap grows, RNNs become unable to learn to connect information

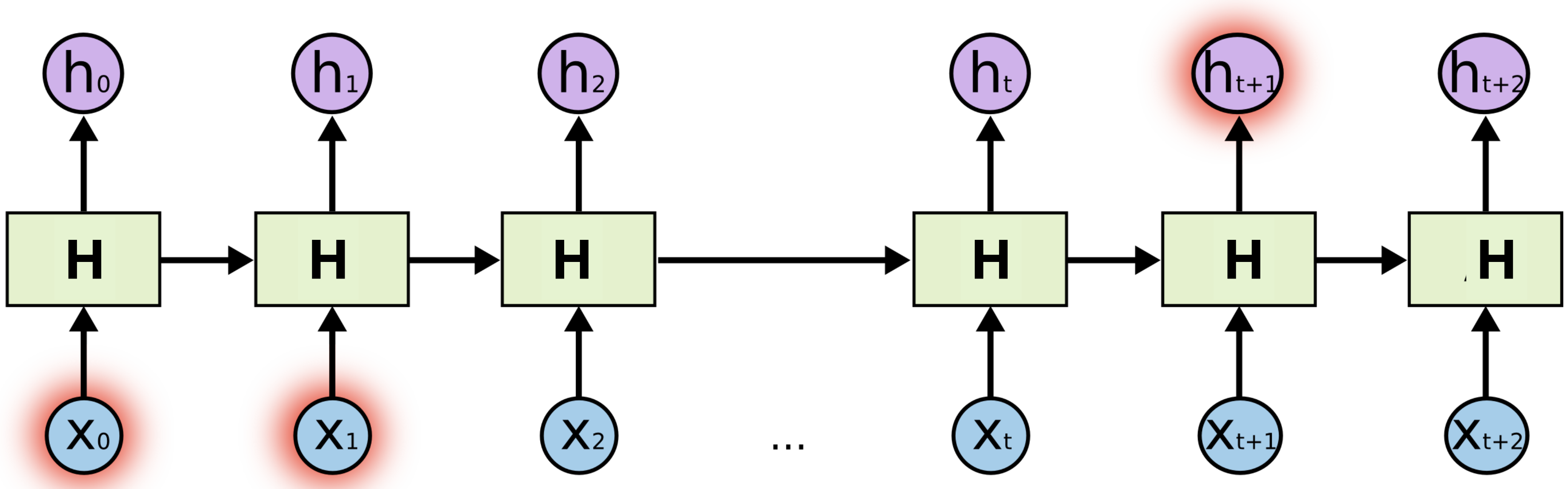
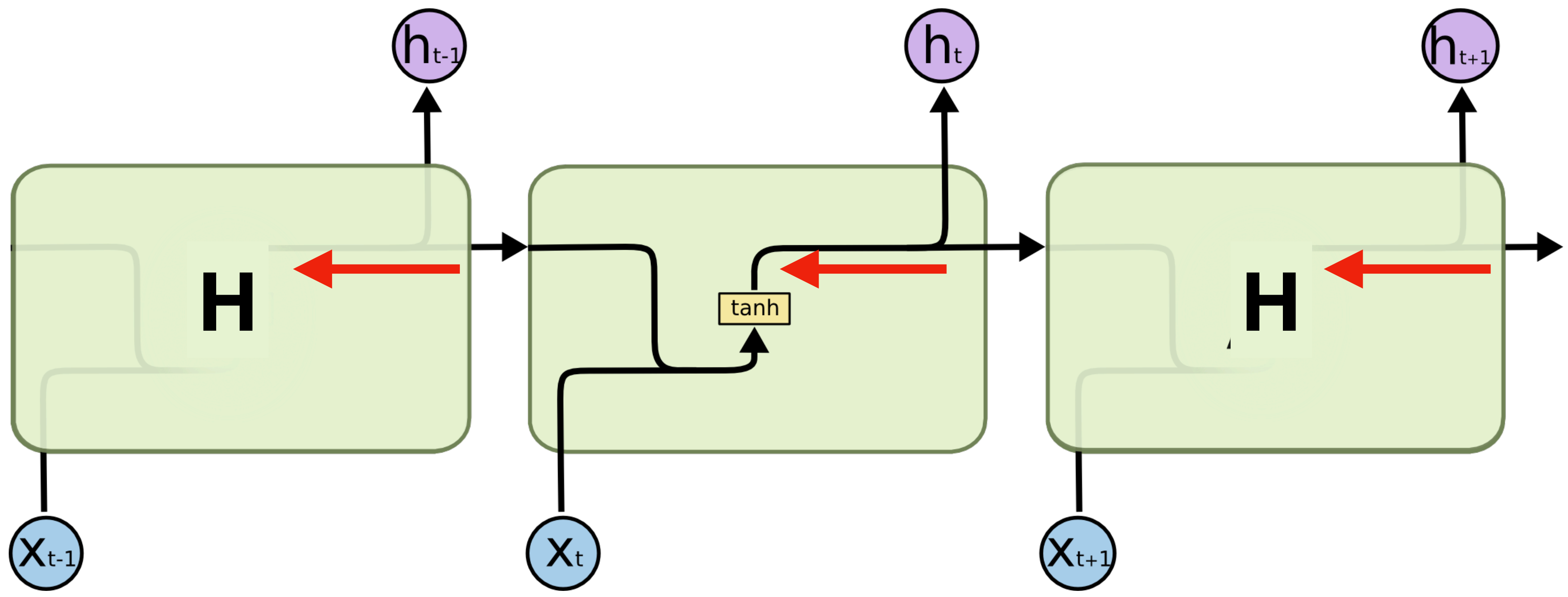


Image: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

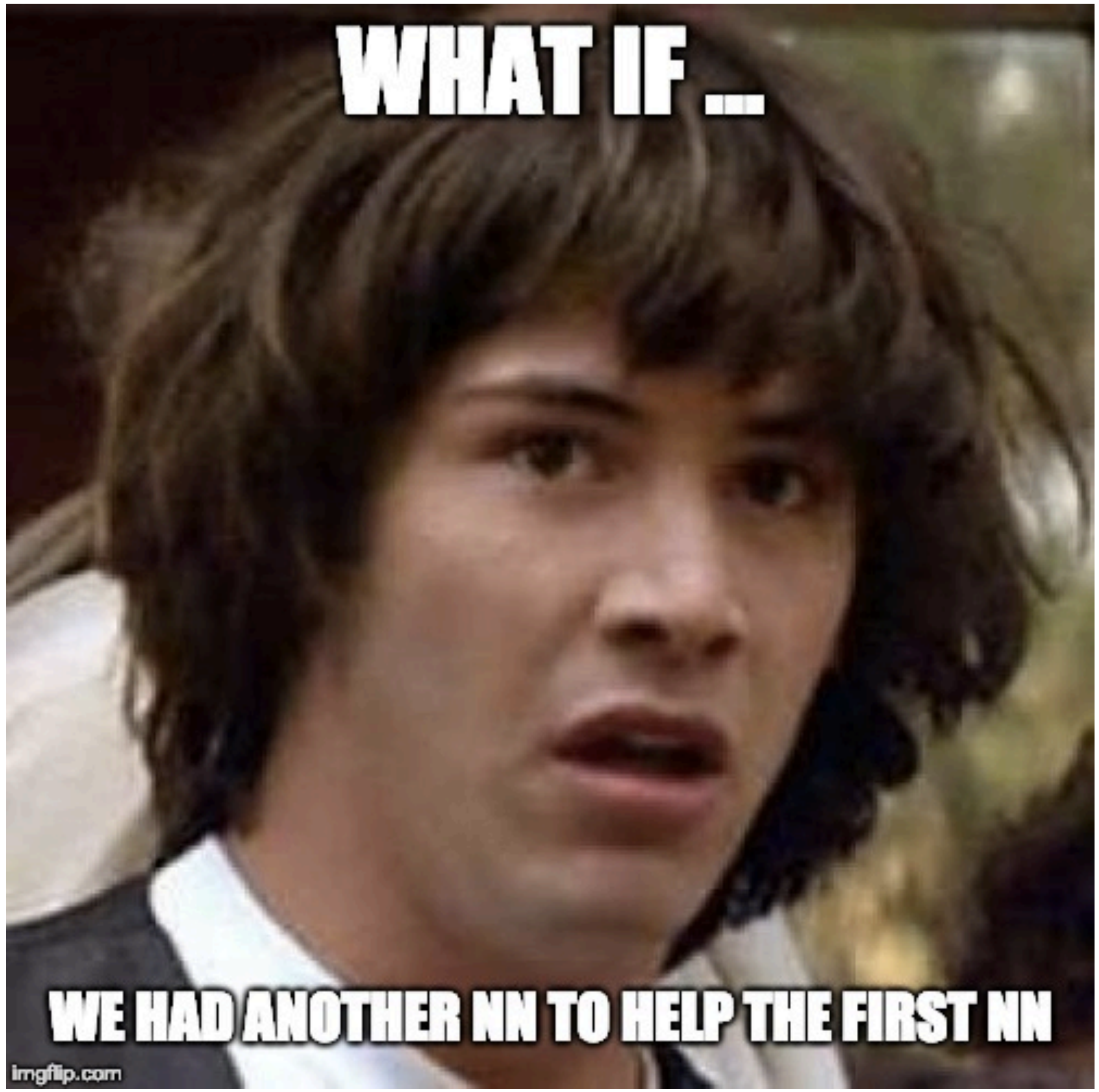
- The main problem with RNNs is that gradients less than 1 become exponentially small over time
- Known as the vanishing gradient problem
- Gradients greater than 1 become exponentially large over time (the exploding gradient problem)*
- This leads to training instability, and bad results

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

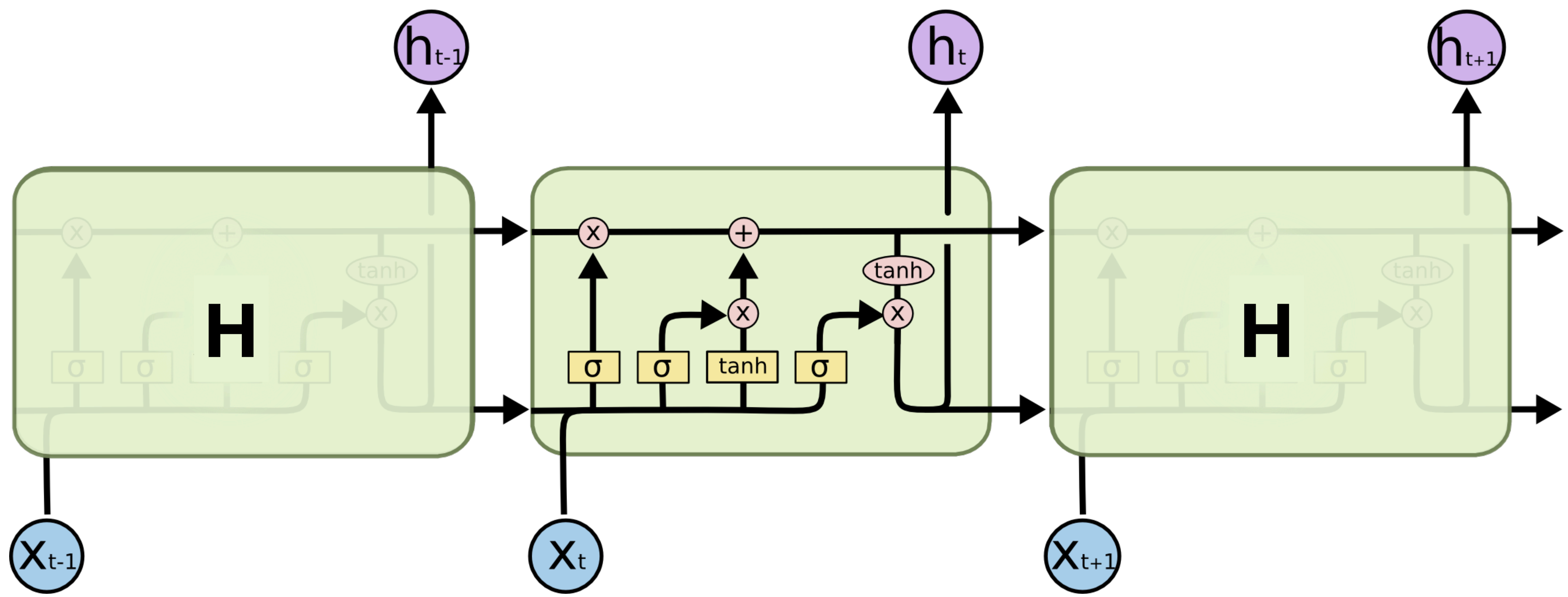
Problems with RNN

- 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

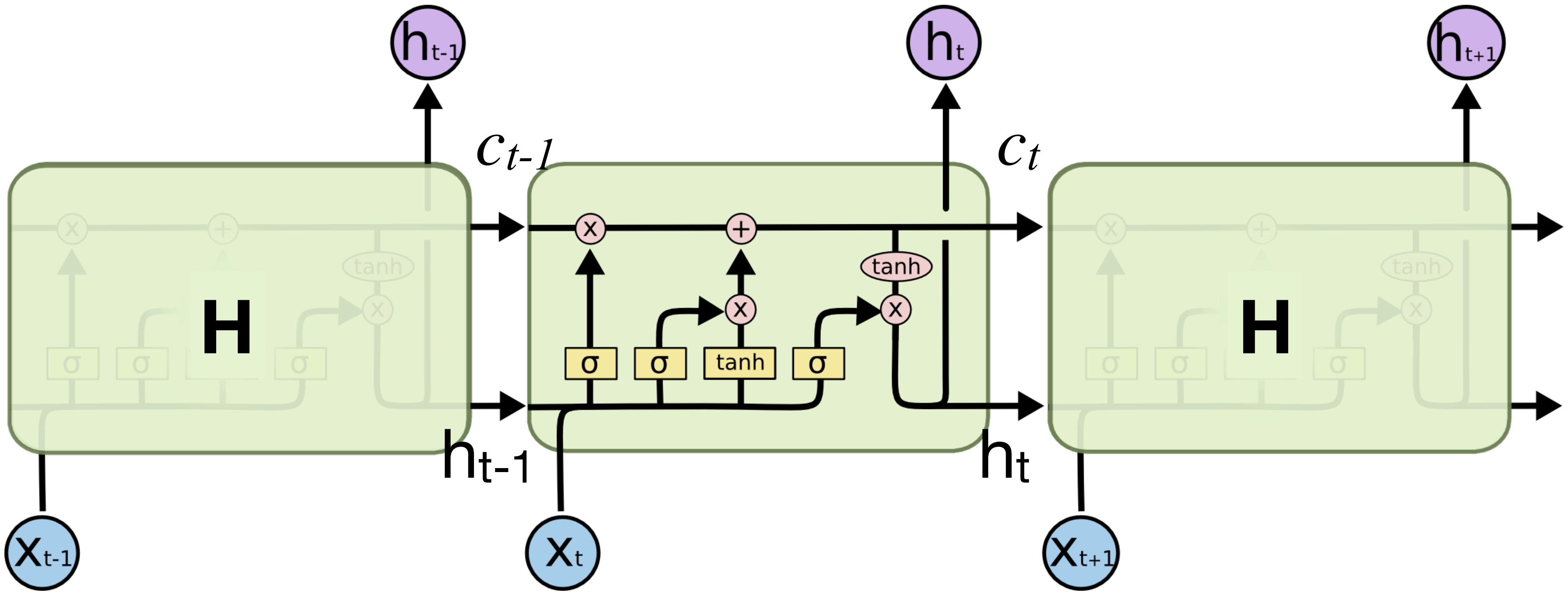


- Lets add another neural network help the first network learn long-distance relationships

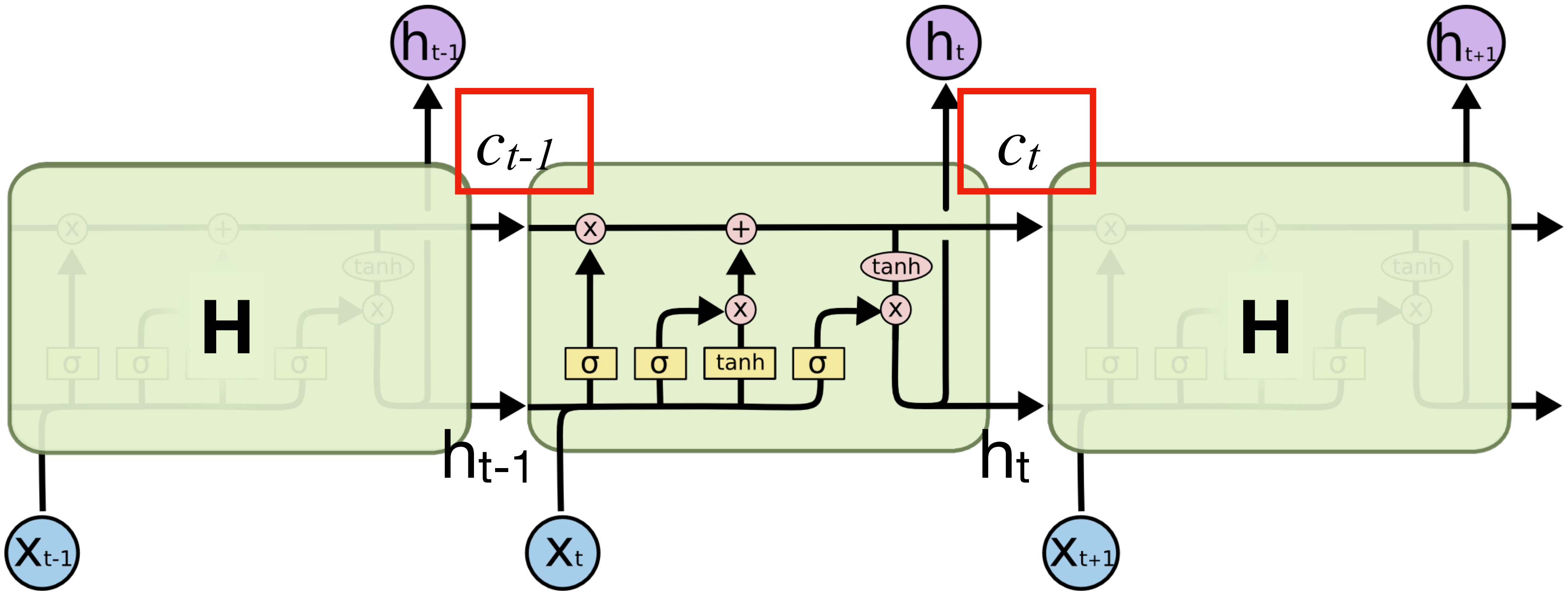
- That's basically what we do when we add more weight matrices to a neural network

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

LSTM: States

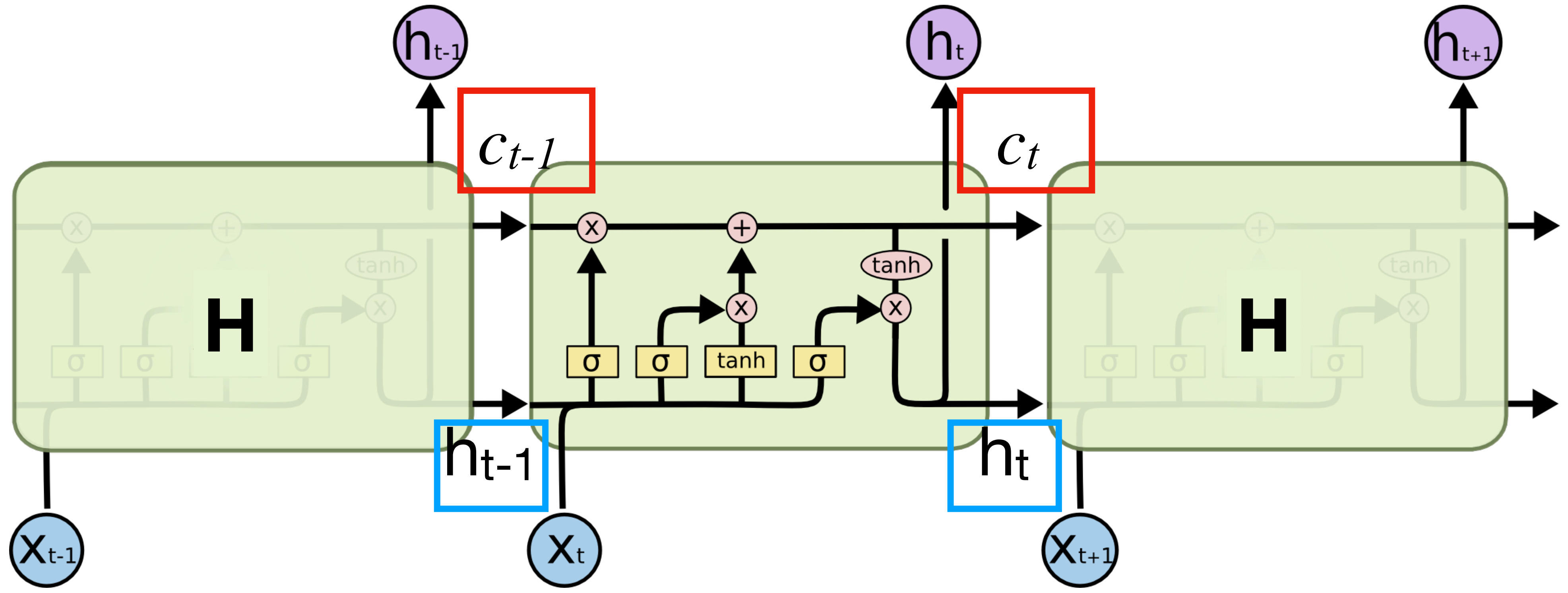


LSTM: States



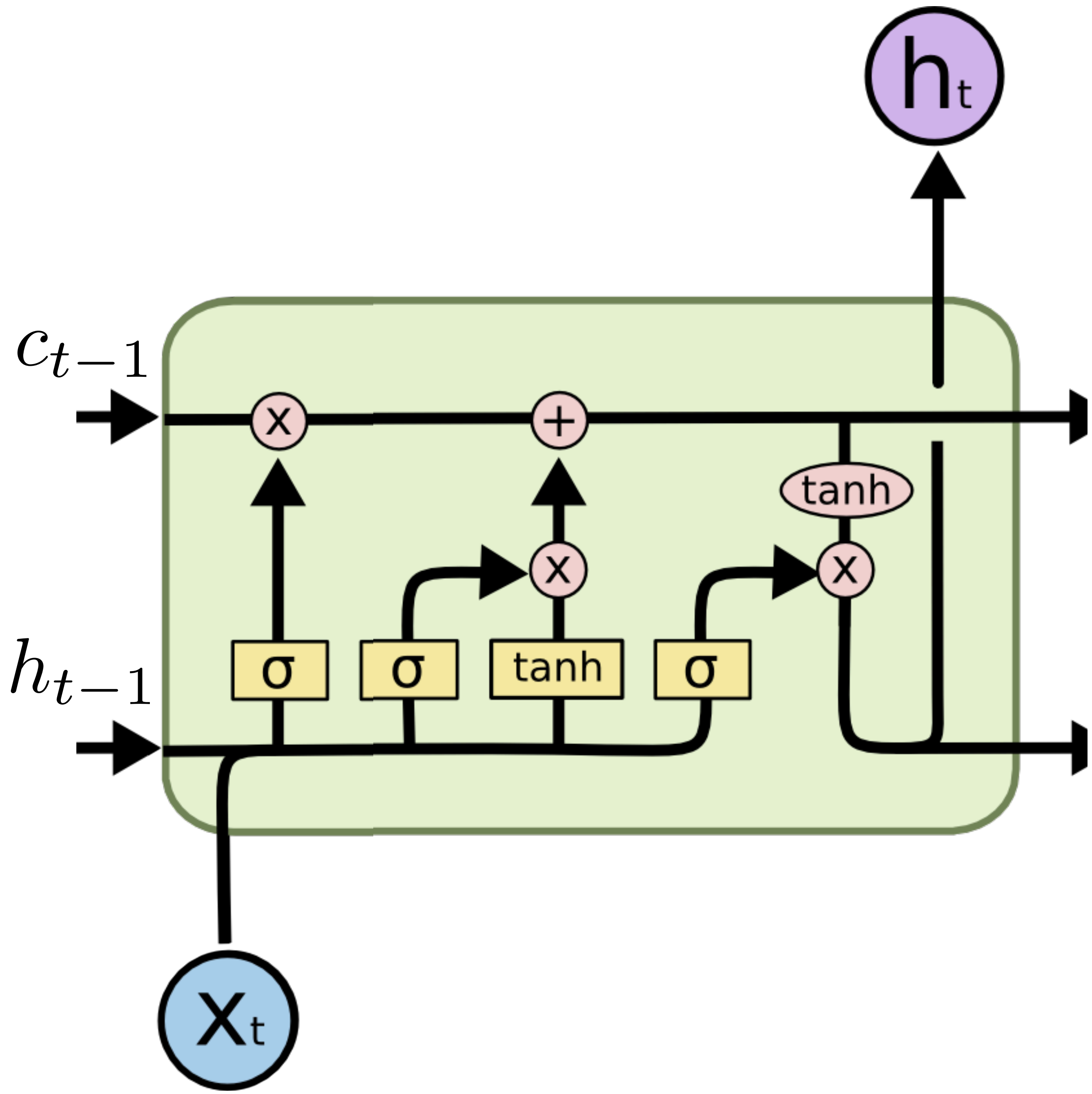
- Global State c captures global information at the document/ sentence level

LSTM: States

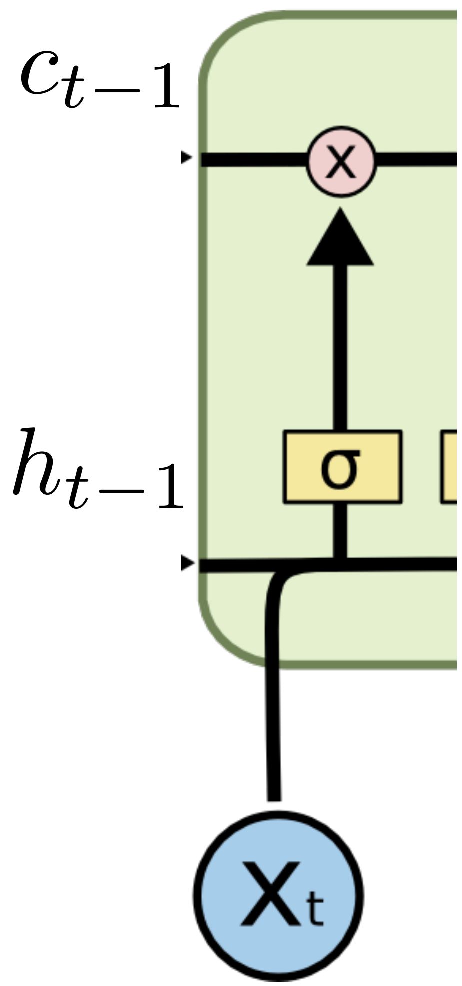


- Global State c captures global information at the document/ sentence level

- LSTM hidden state h_t interacts with this global state to predict the next word



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



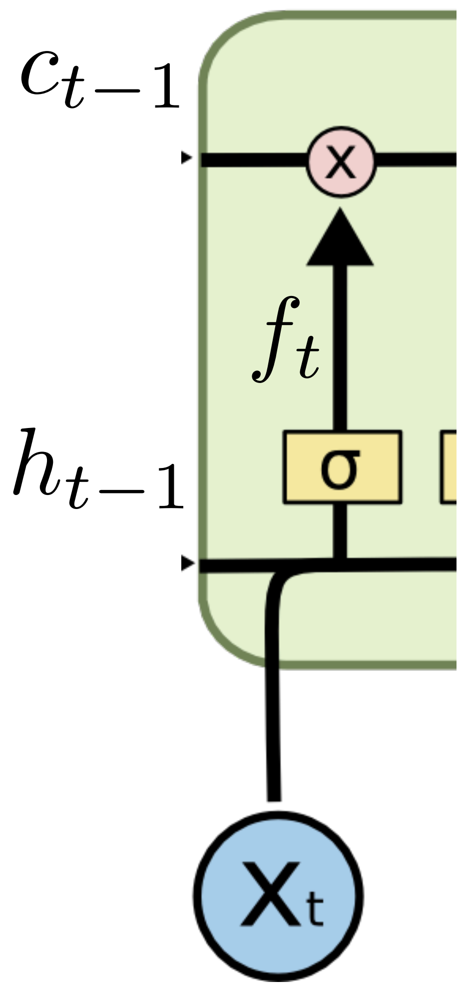
σ sigmoid function

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



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

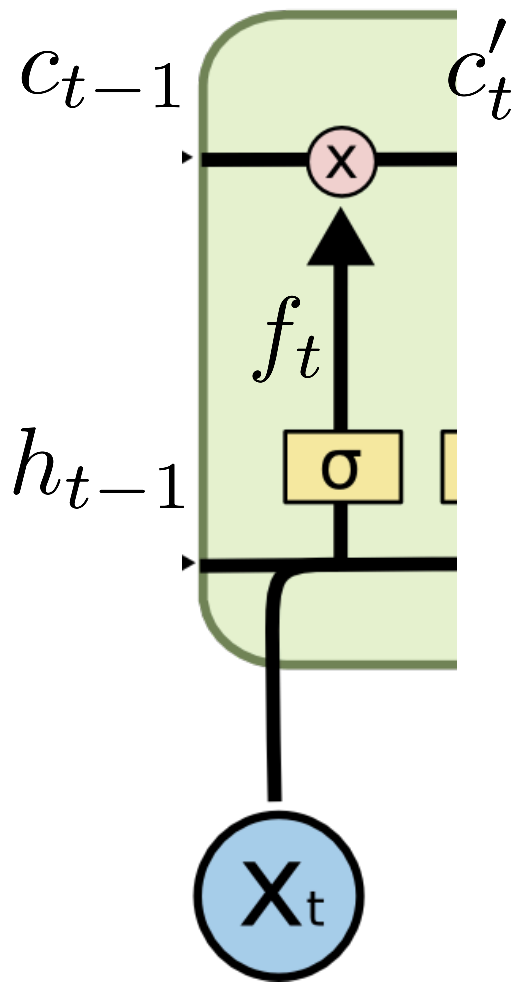
σ sigmoid function

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
- 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

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

$$b_f = 0$$

- σ : sigmoid fn * : pointwise multiplication

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

- calculate: c'_t

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

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

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

- 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

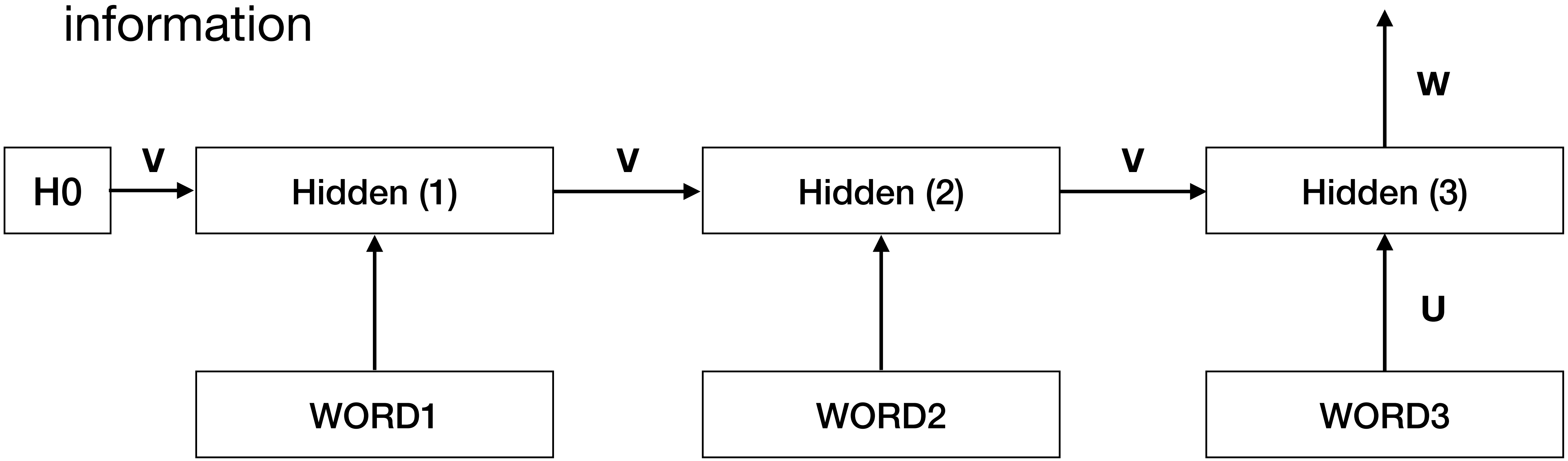
$$w_f[h_{t-1}, x_t] + b_f = [1 \quad 1] \times \begin{bmatrix} 1 \\ 0.2 \end{bmatrix} = [1.2]$$

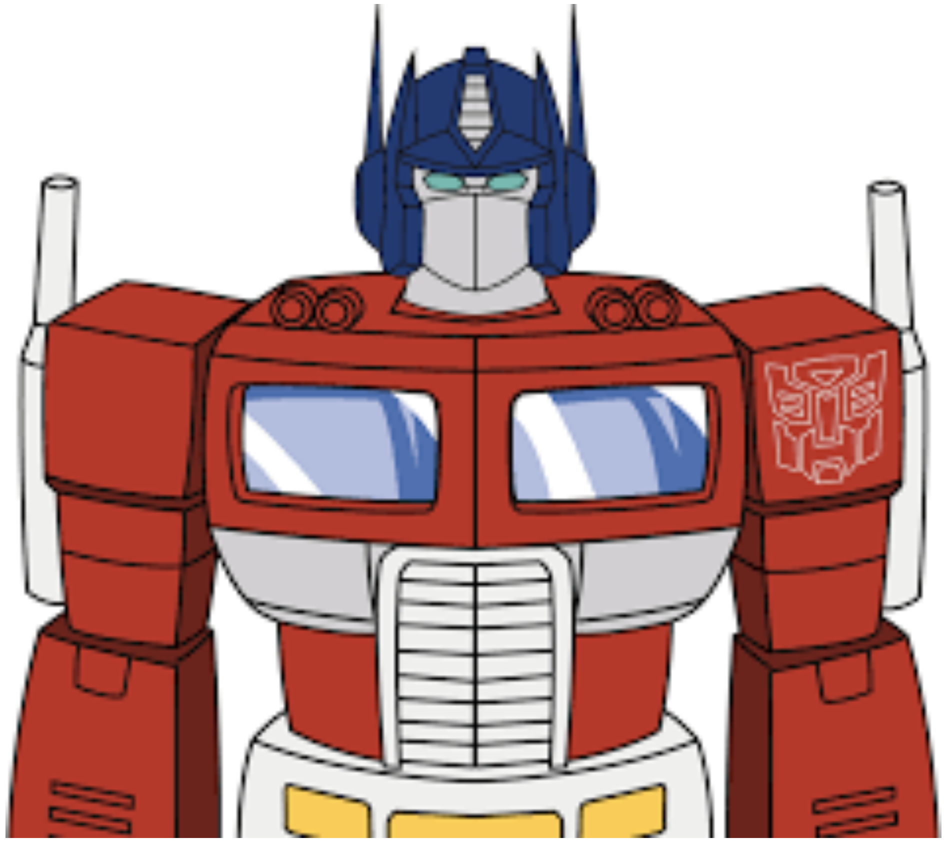
$$f_t = [\sigma(1.2)] = [0.77]$$

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

LSTM Problems

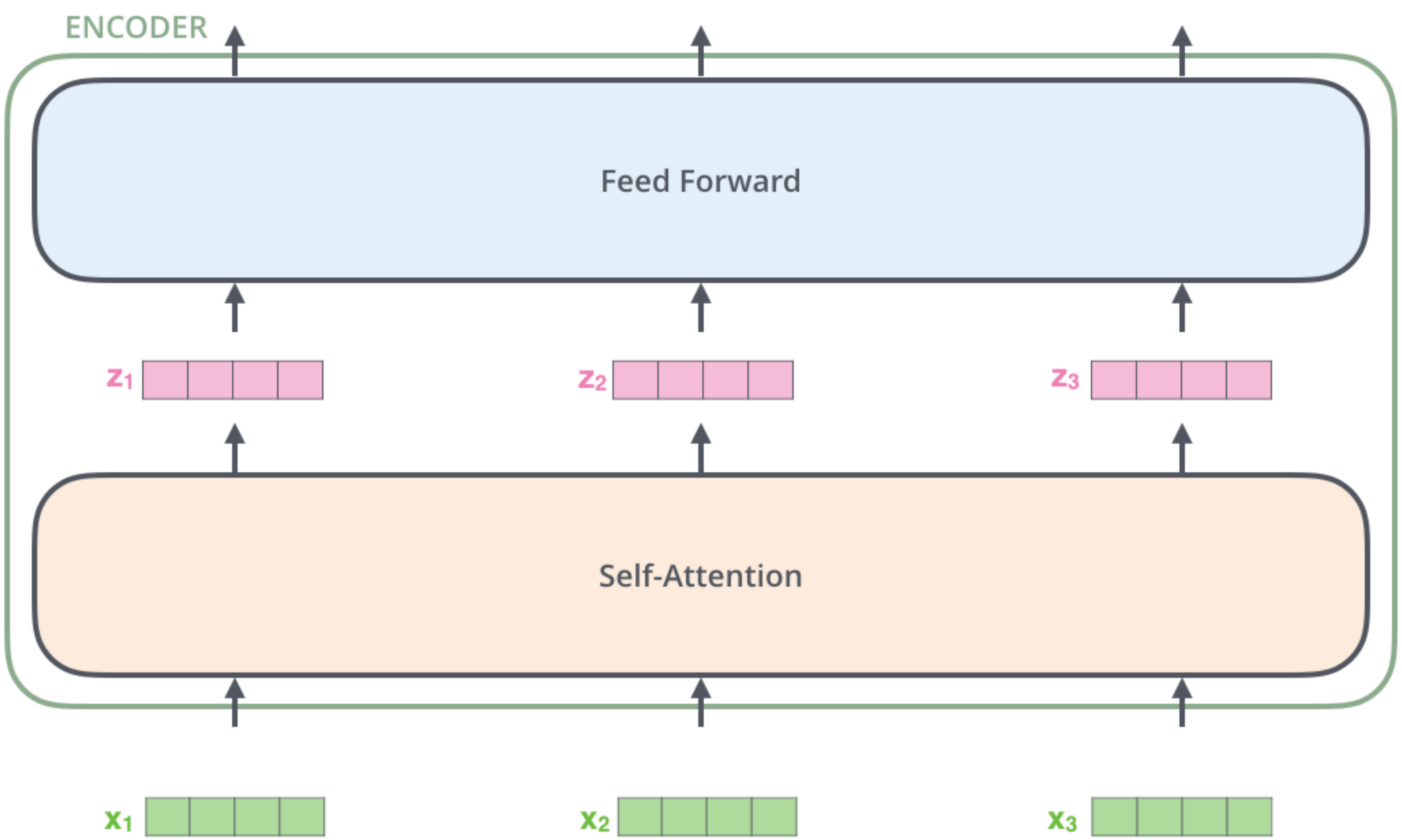
- Forget gate: removes information from the Global Cell state (*C*)
  - this information might be be useful at a later stage
- Implicit representation of long-term information
  - Cell state and previous hidden state summarise the prior information





Transformers for Language Modelling

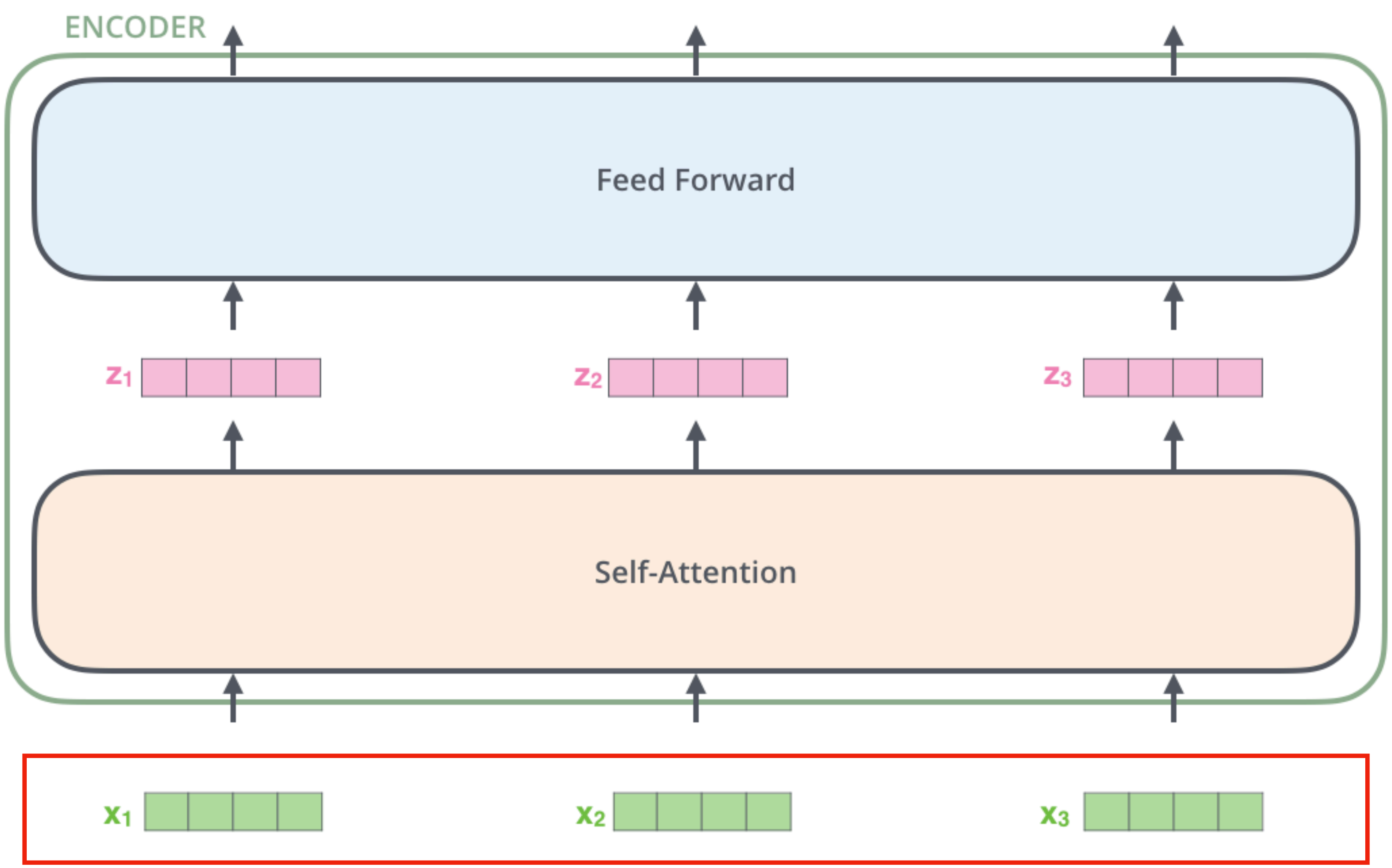
Transformers: Simplified



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

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

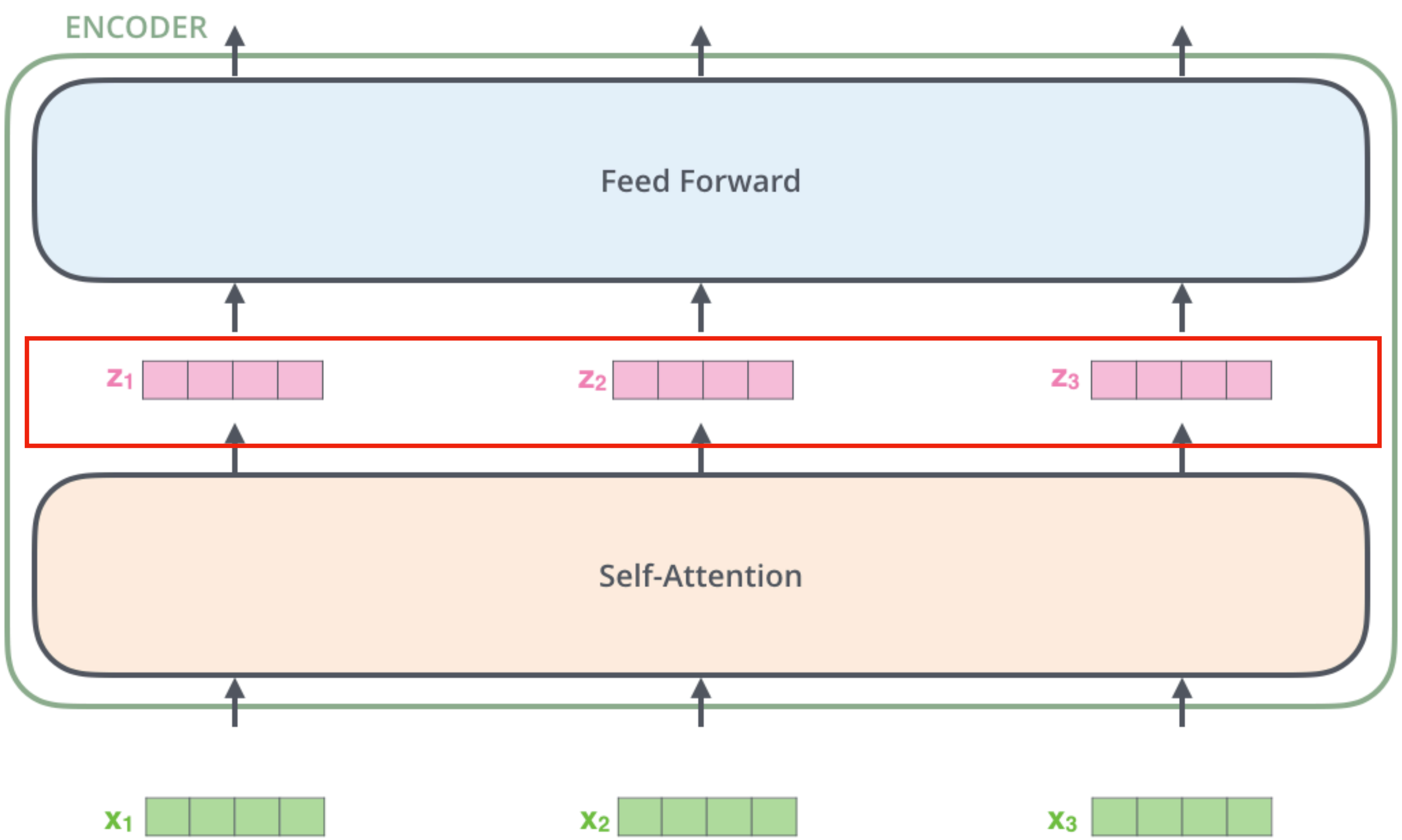
Transformers: Simplified



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

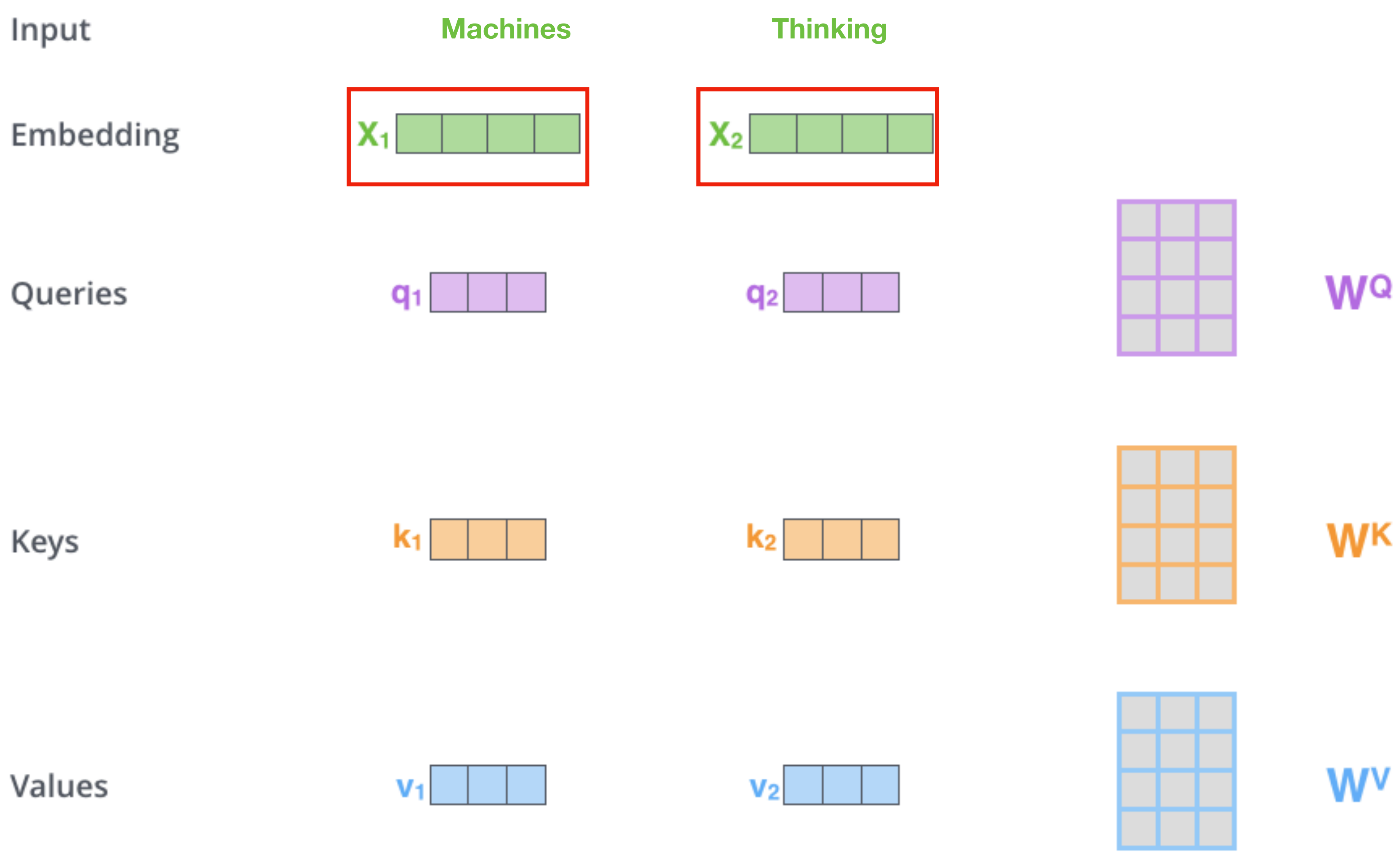
Credit: http://jalammar.github.io/illustrated-transformer/

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"

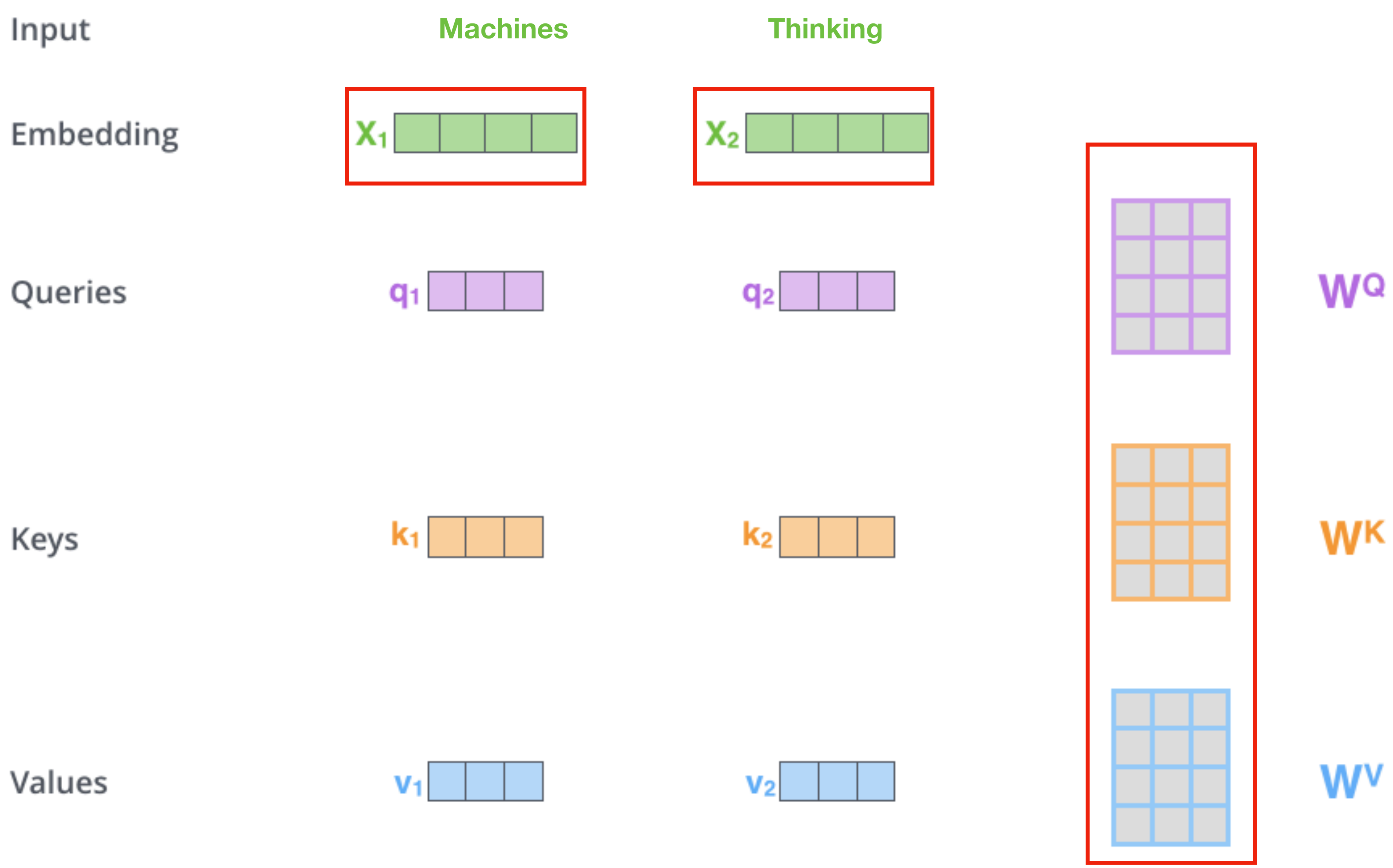
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



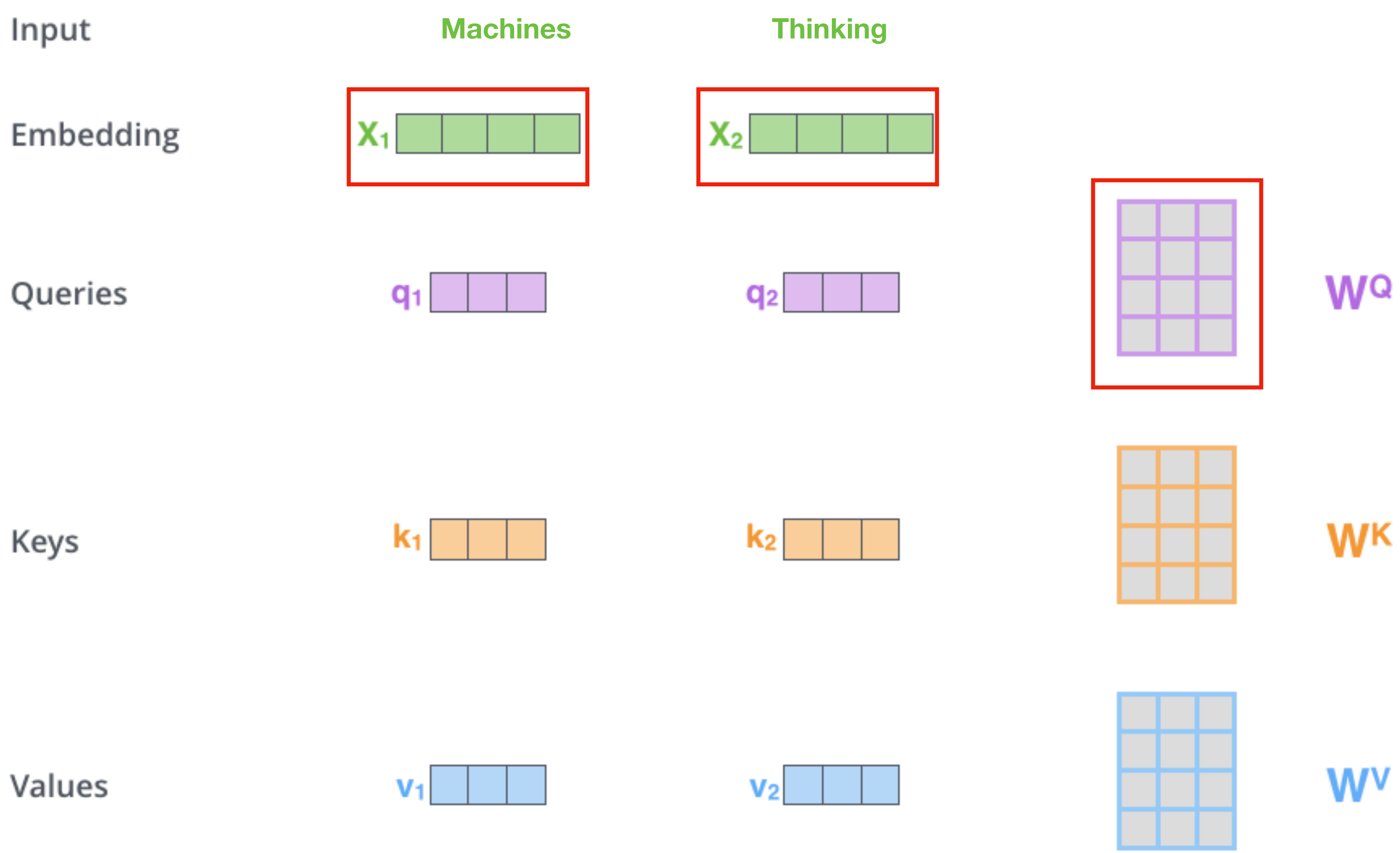
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



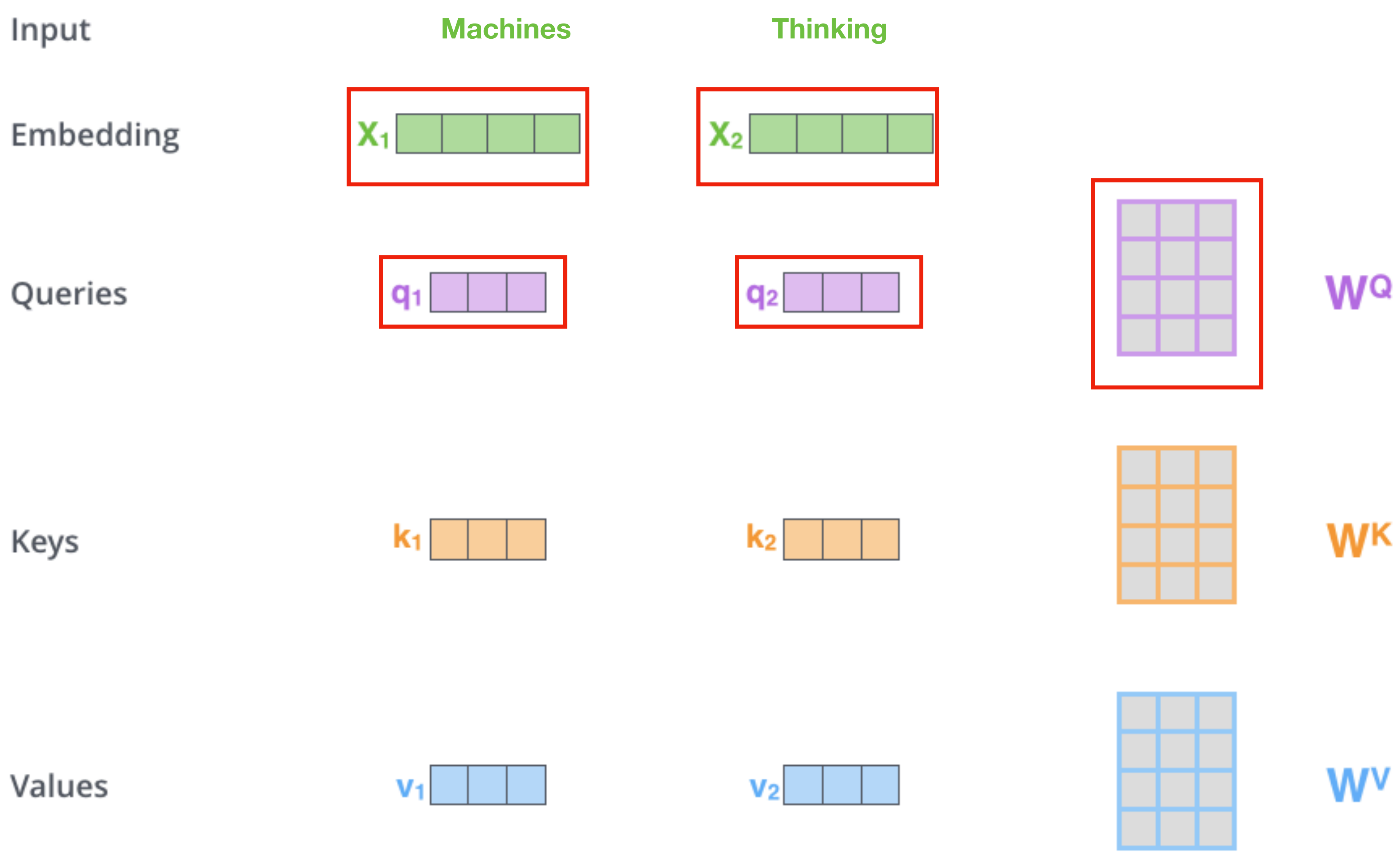
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



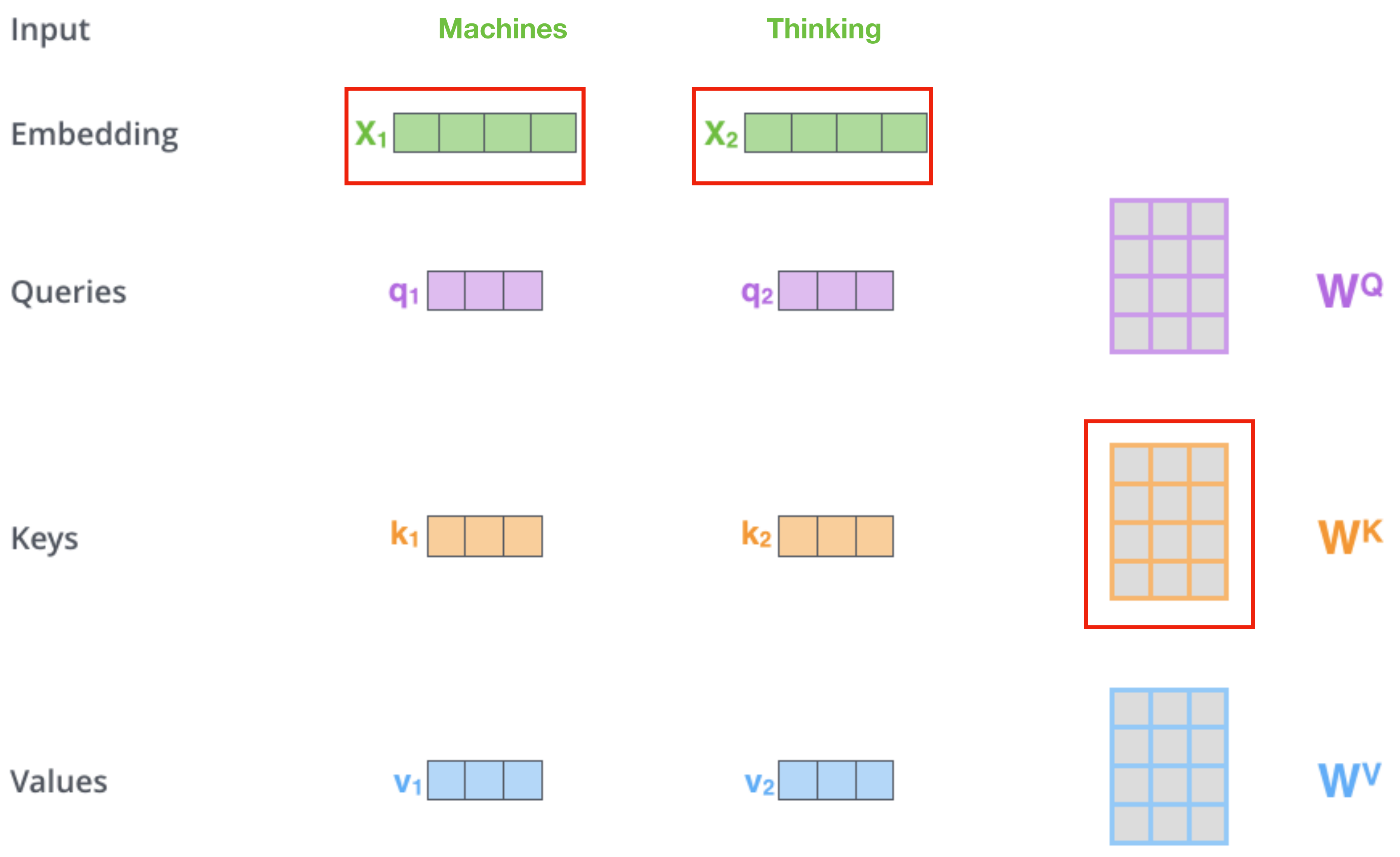
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



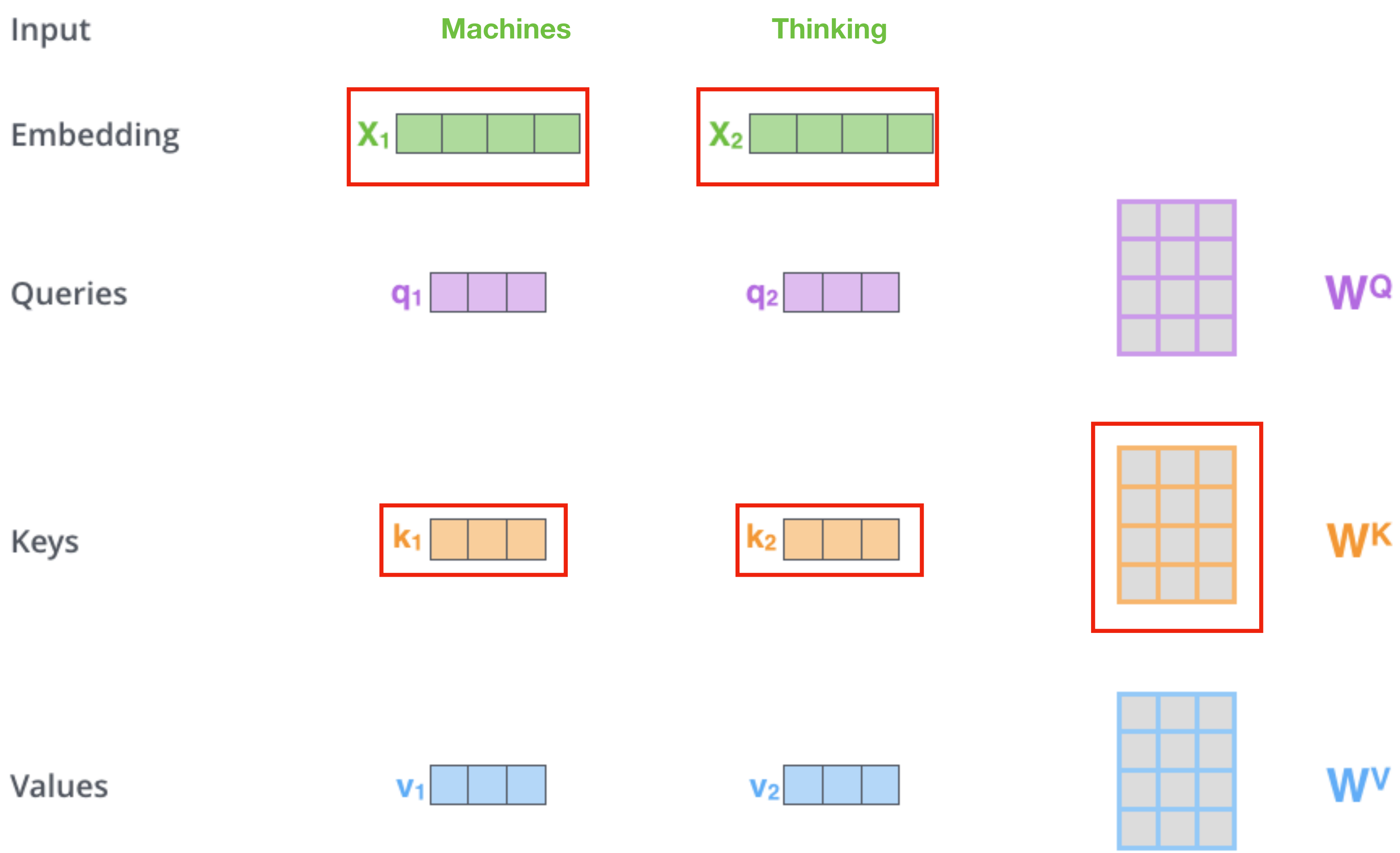
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0



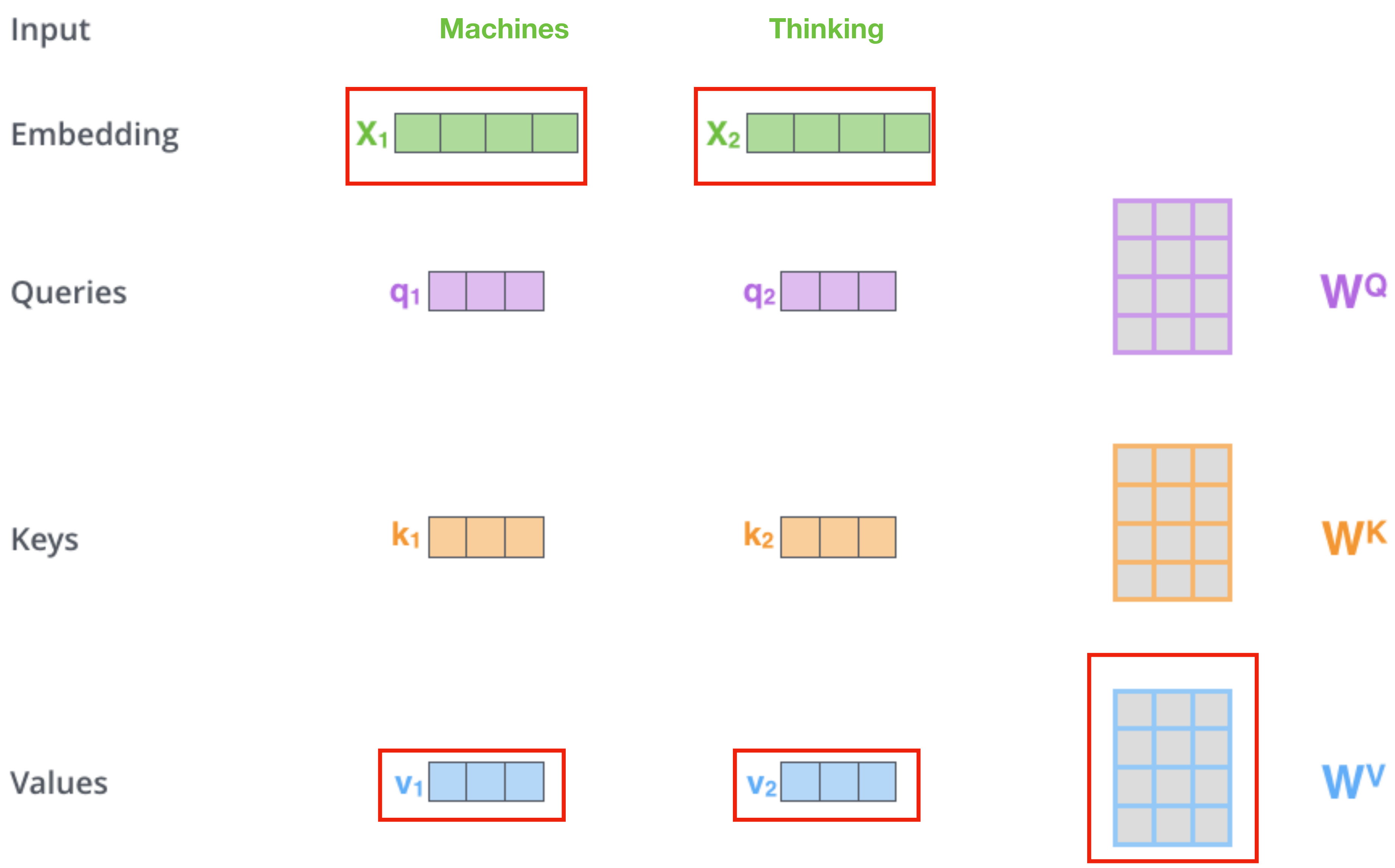
Credit: http://jalammar.github.io/illustrated-transformer/

Self-Attention: Step 0

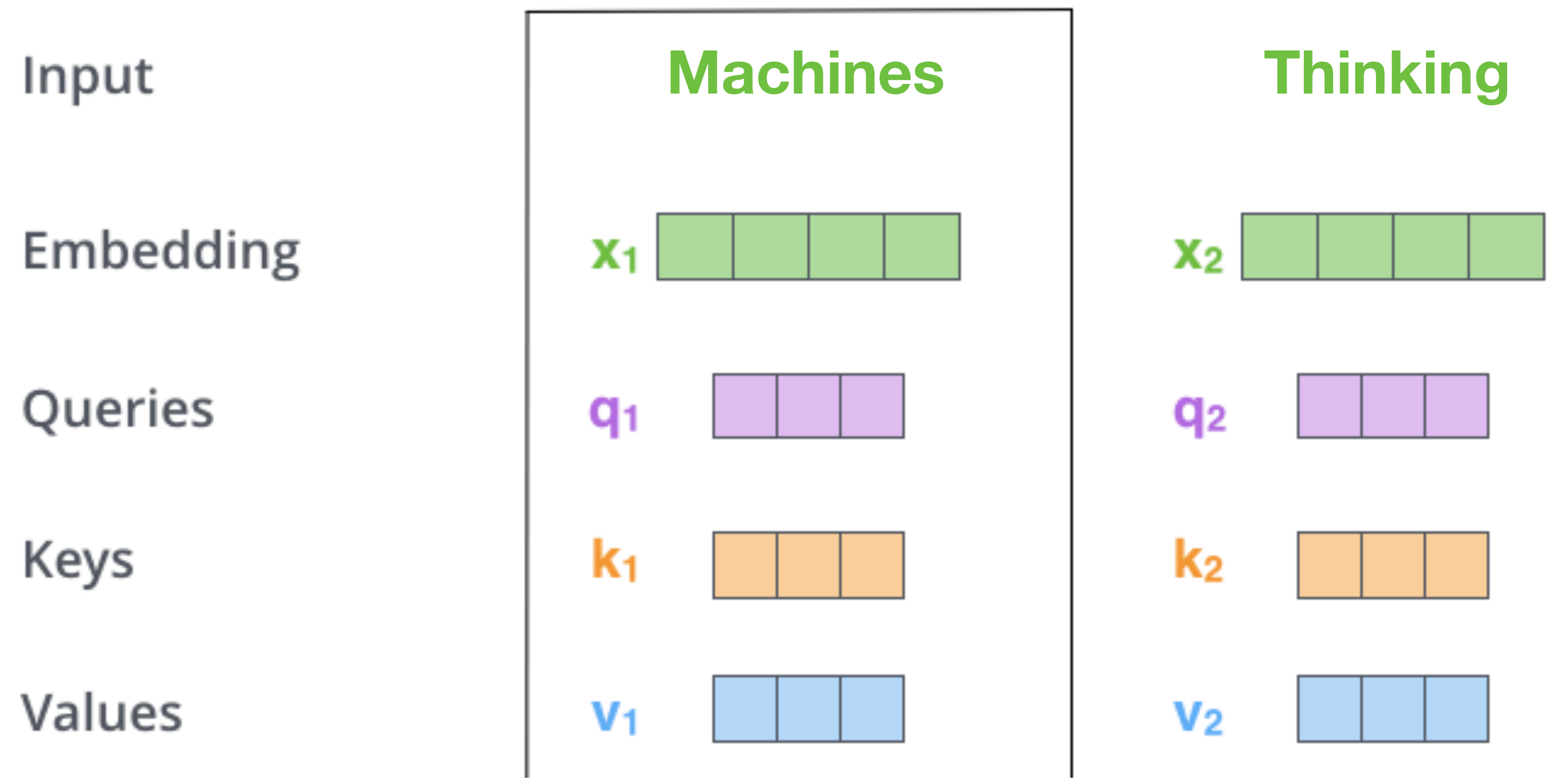


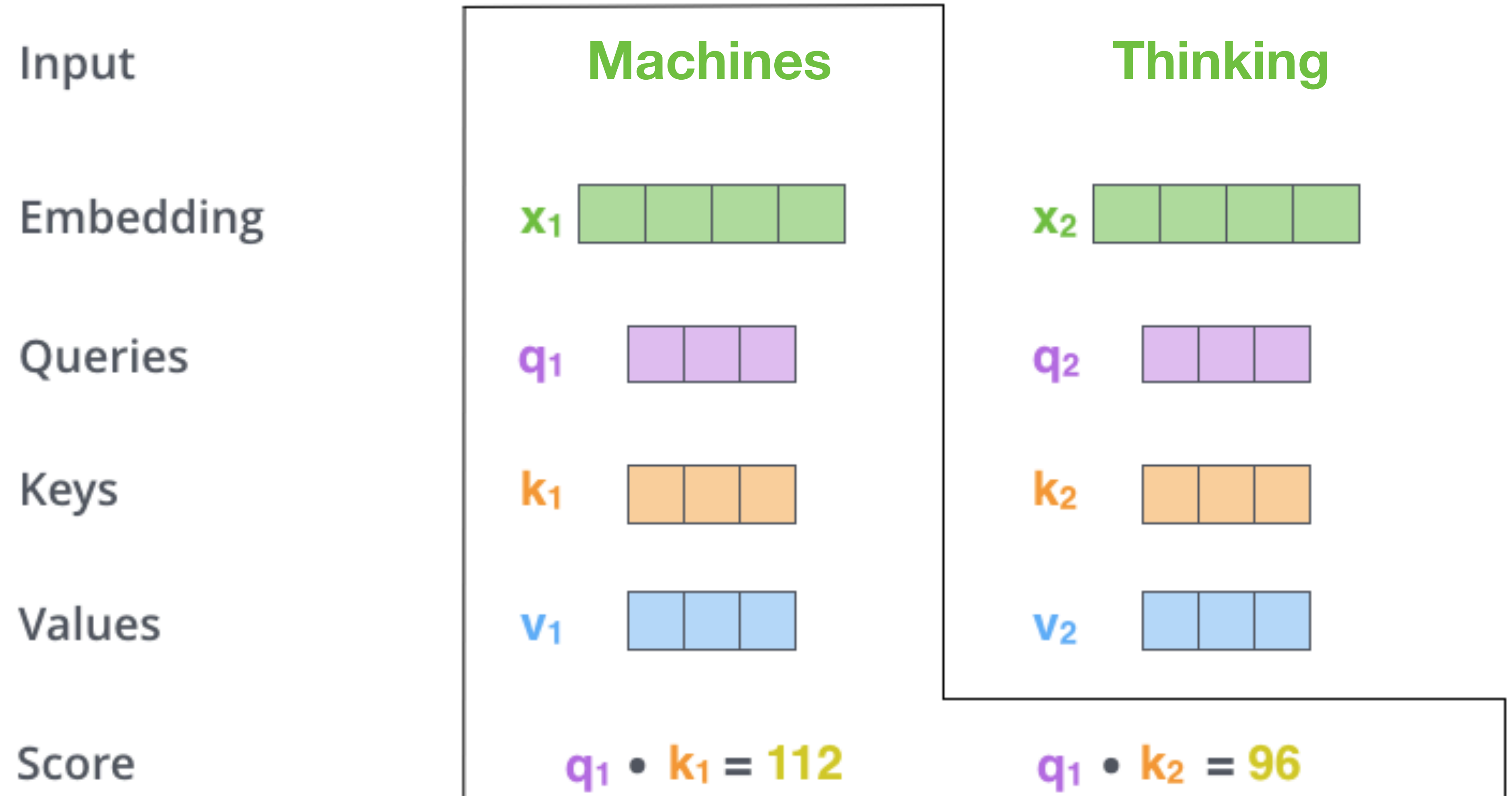
Credit: http://jalammar.github.io/illustrated-transformer/

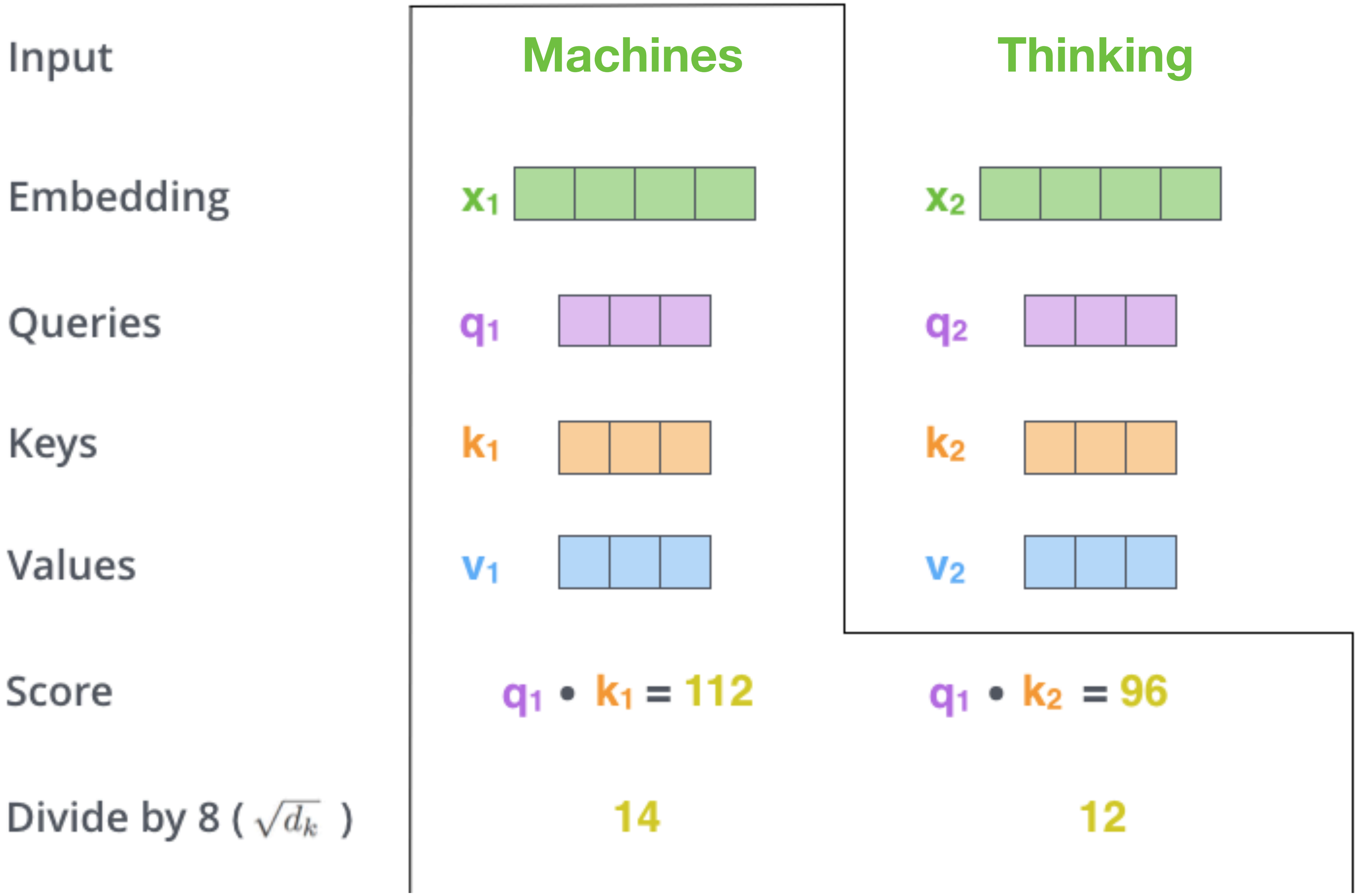
Self-Attention: Step 0

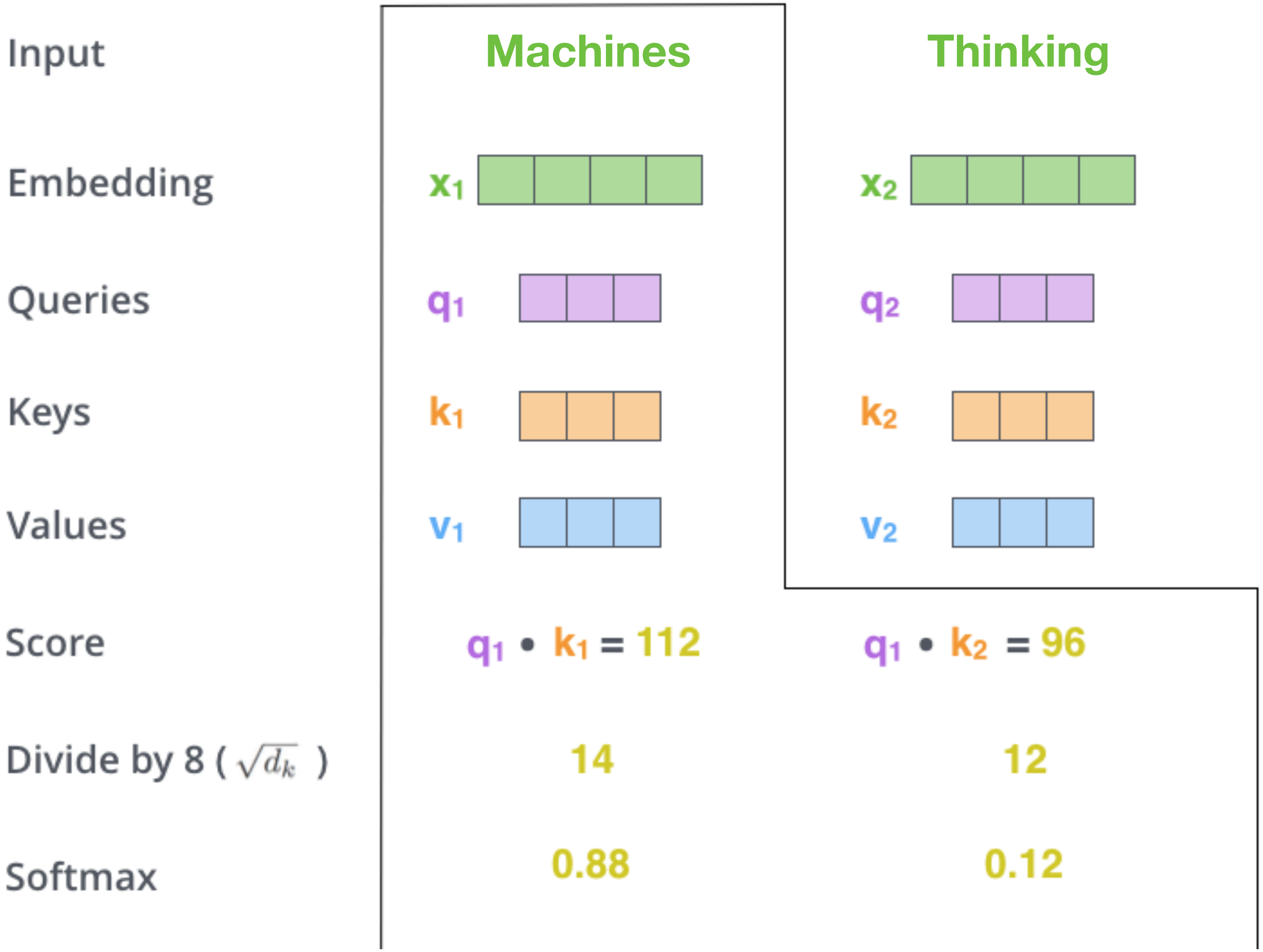


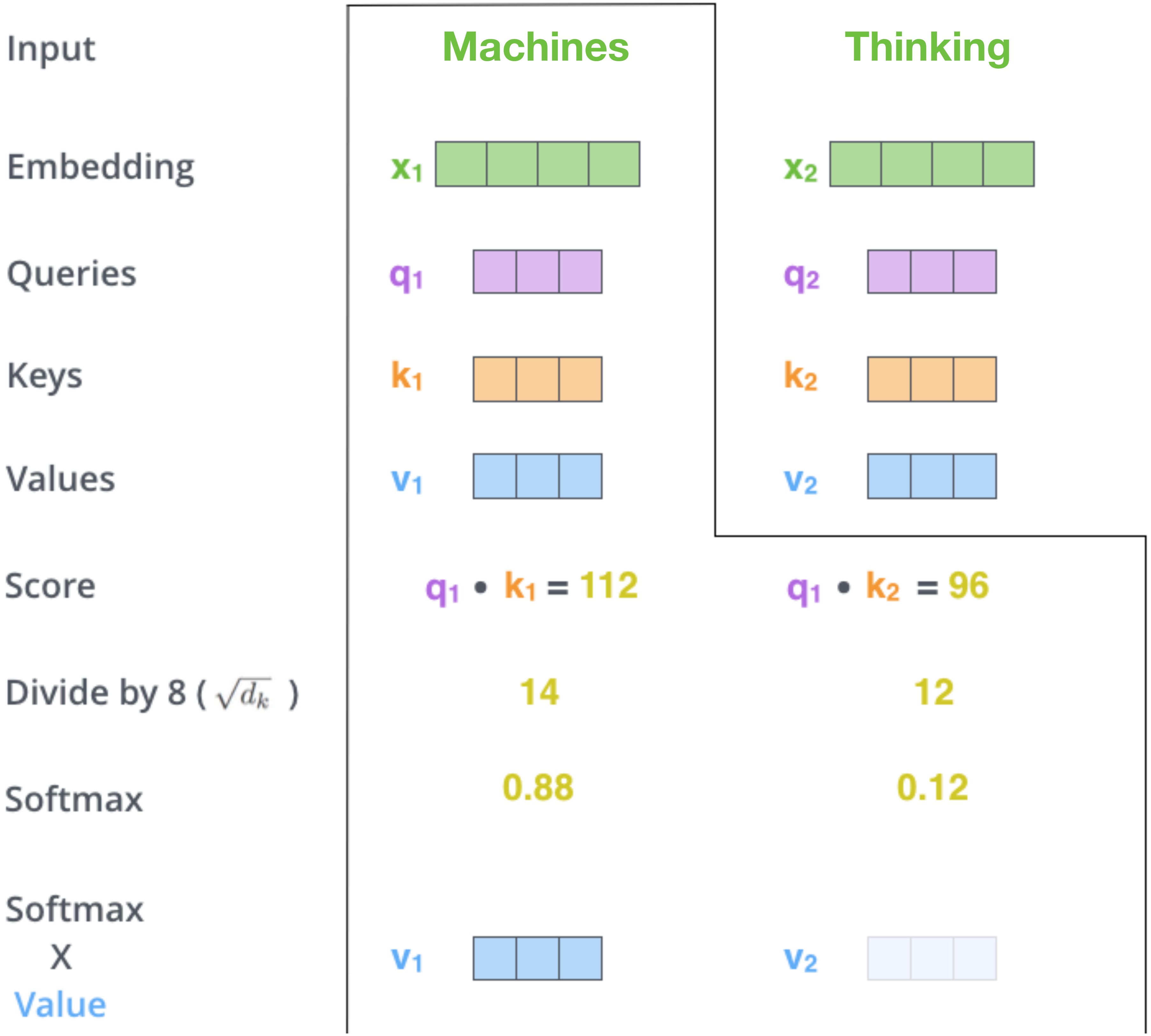
Credit: http://jalammar.github.io/illustrated-transformer/

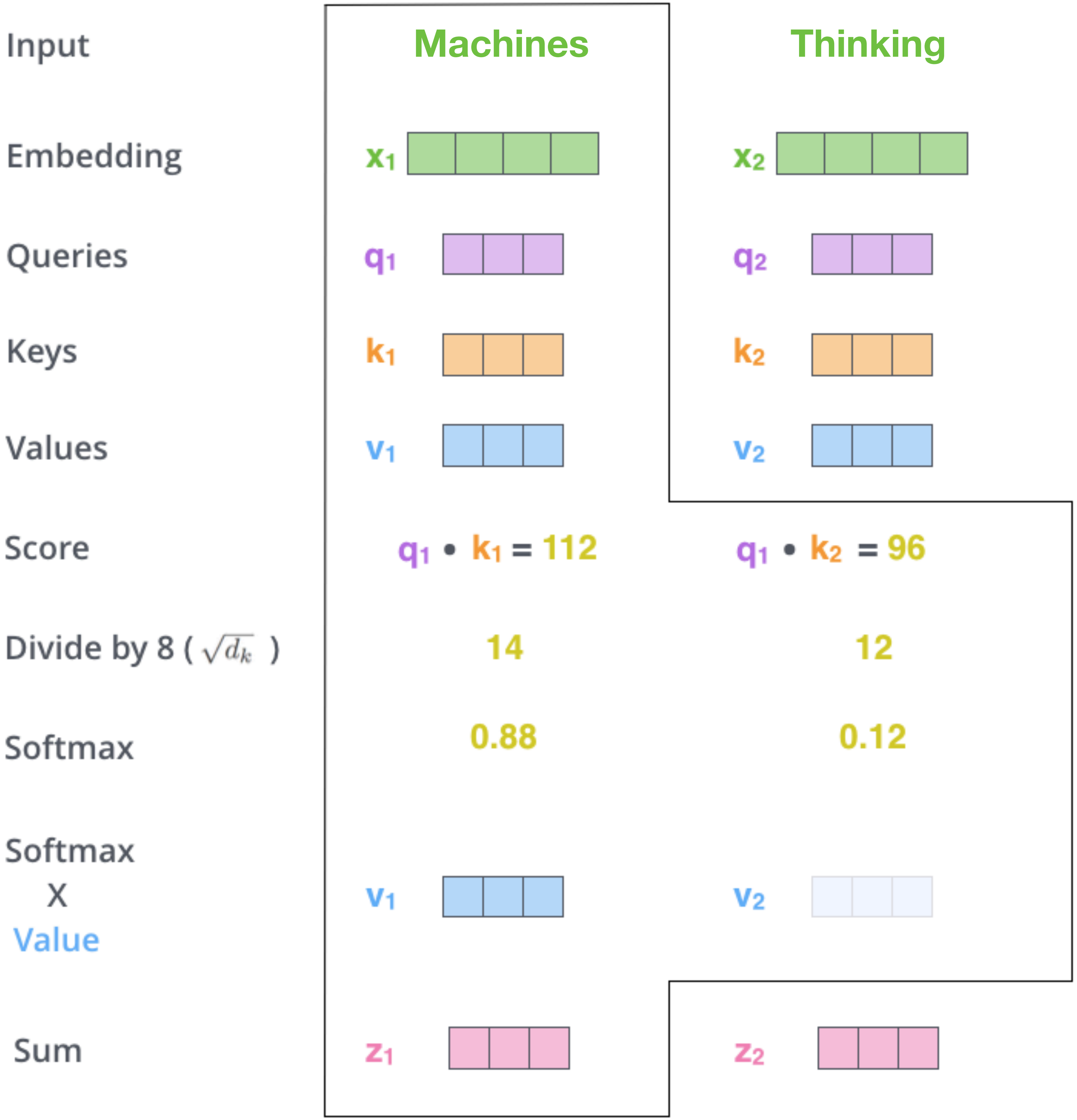

Input
Embedding
Queries
Keys
Values
Machines
Thinking
x1
x2
q1
q2
k1
k2
v1
v2


Input
Machines
Thinking
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96


Input
Machines
Thinking
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

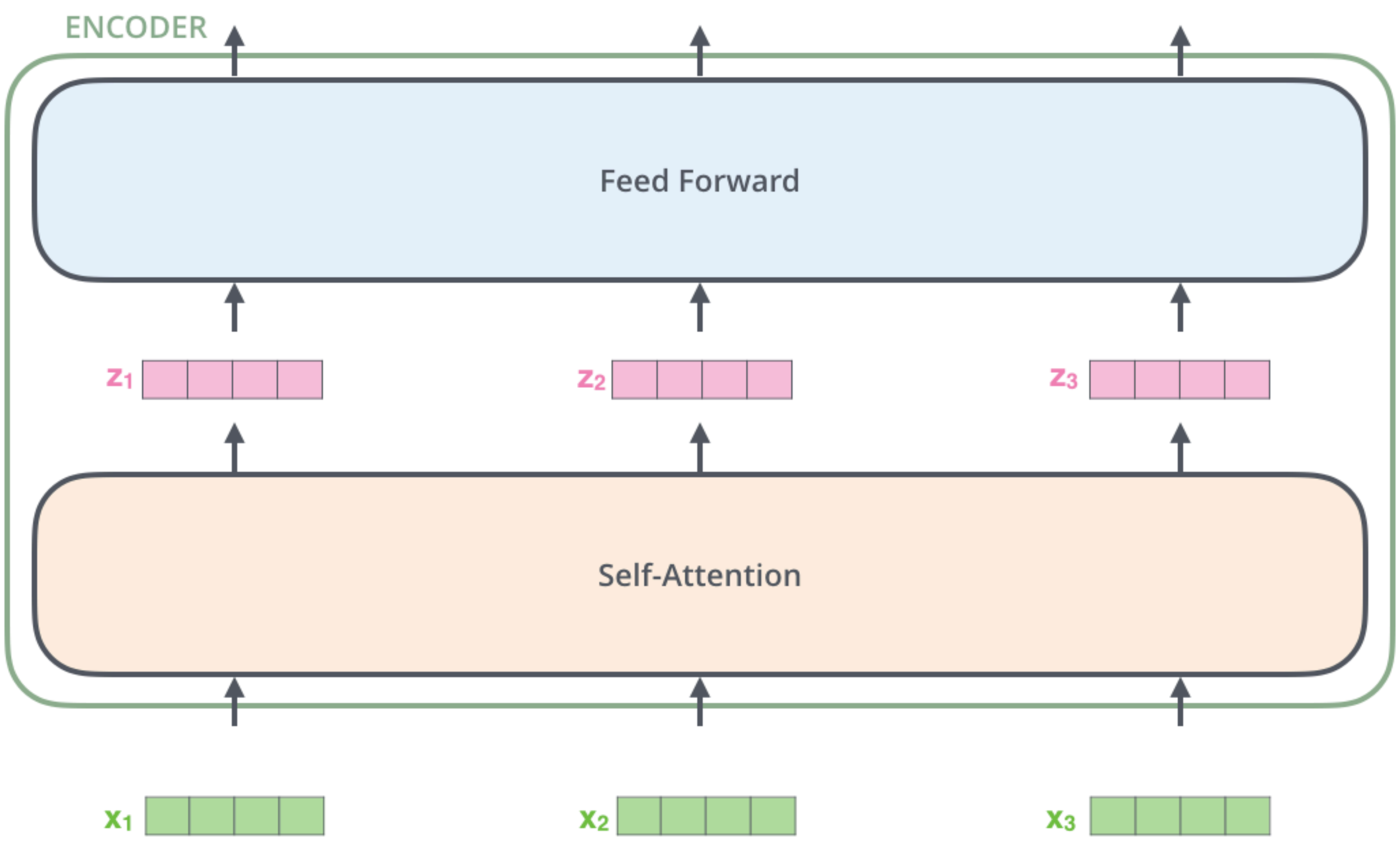

Input
Machines
Thinking
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12


Input
Embedding
Queries
Keys
Values
Score
Divide by 8 ($\sqrt{d_k}$)
Softmax
Softmax
X
Value
Machines
Thinking
x_1
x_2
q_1
q_2
k_1
k_2
v_1
v_2
$q_1 \bullet k_1 = 112$
$q_1 \bullet k_2 = 96$
14
12
0.88
0.12
v_1
v_2



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

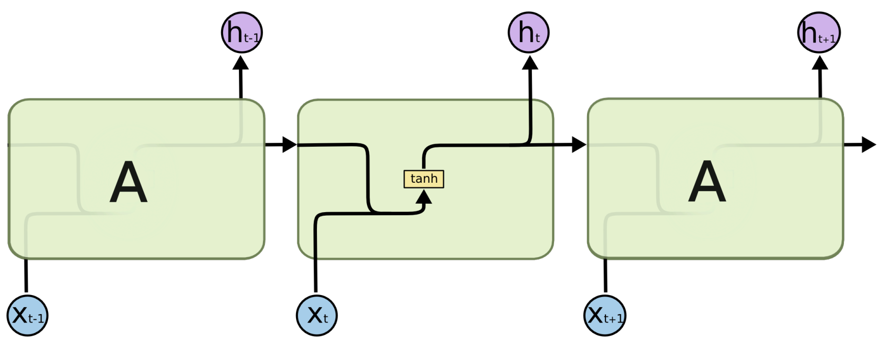
Self-Attention

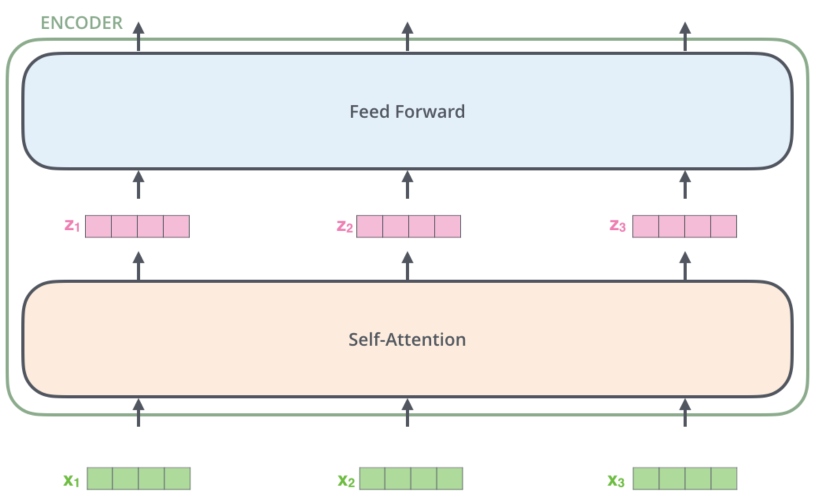
- Self-Attention seems to be asking an association question
- Query ~ smaller word embedding

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

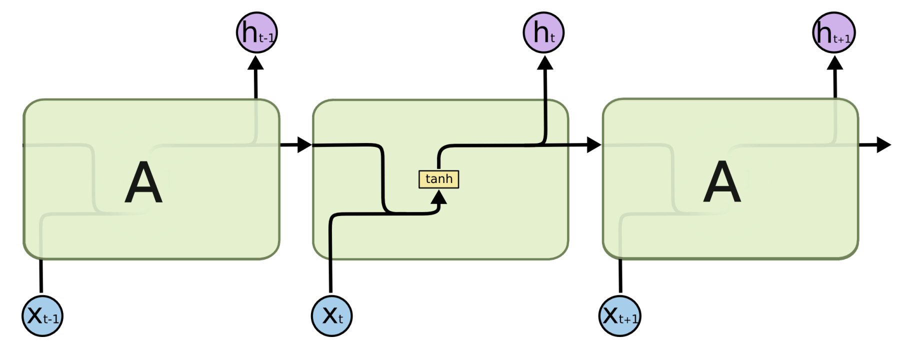
Transformers for Language Modelling

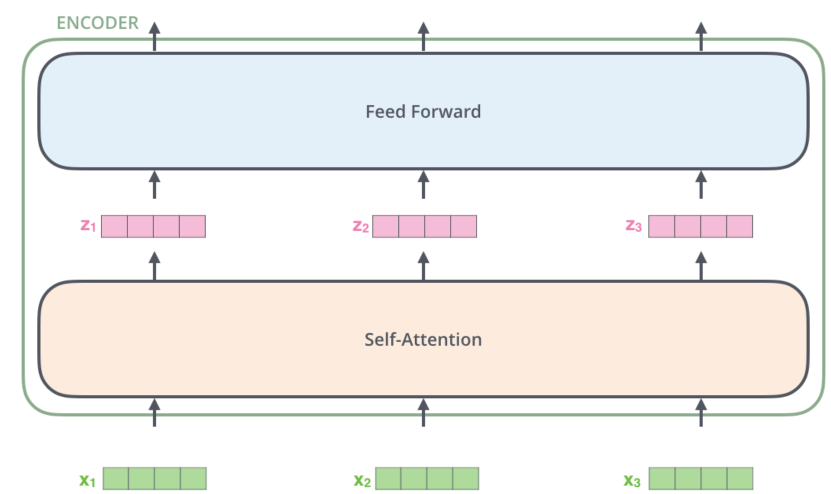




Transformers for Language Modelling

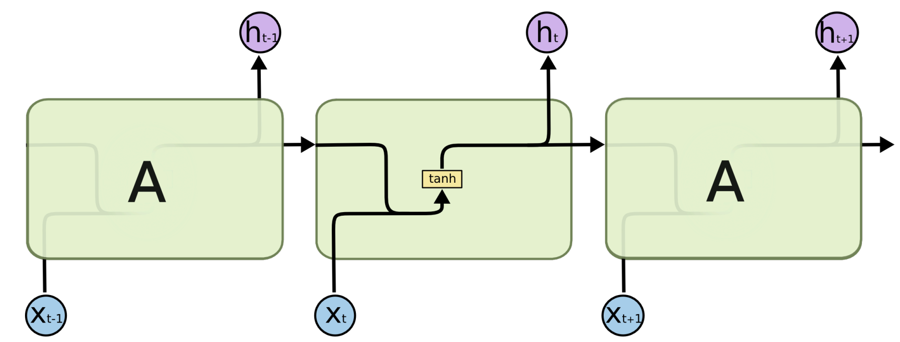
- RNNs: Process tokens one-by-one

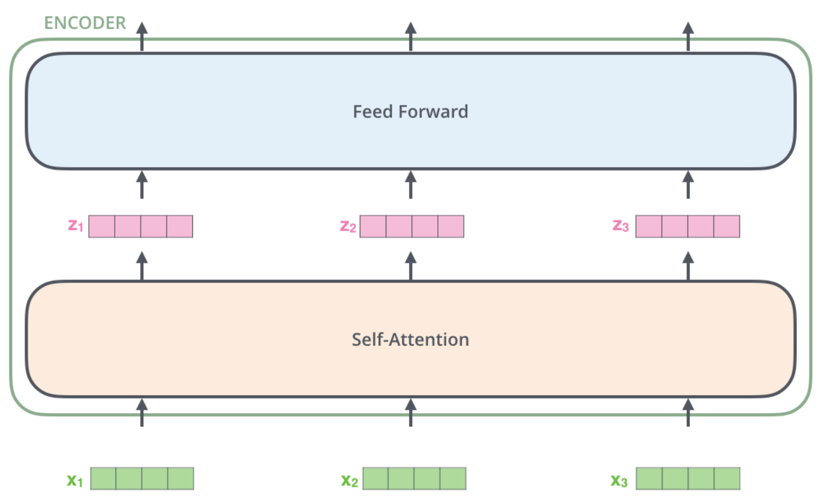




Transformers for Language Modelling

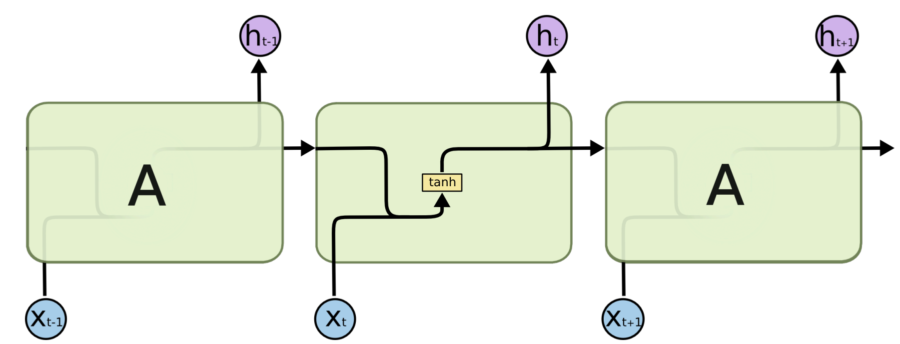
- RNNs: Process tokens one-by-one
  - Chain of dependencies built using a single token



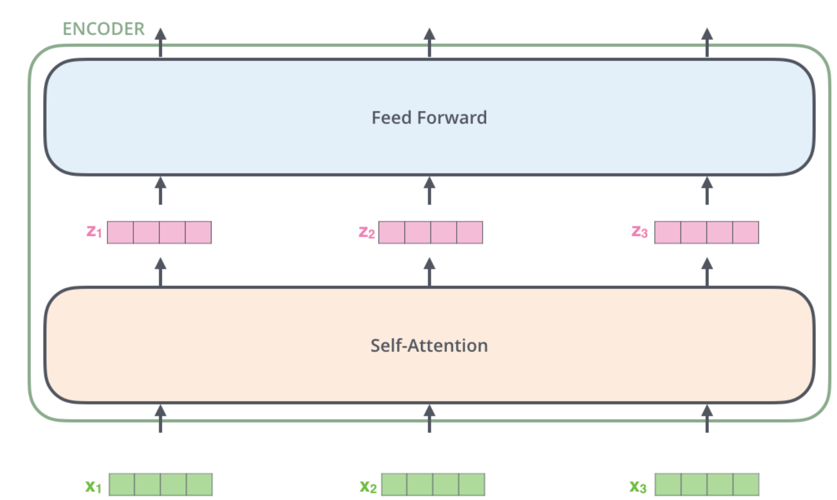


Transformers for Language Modelling

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

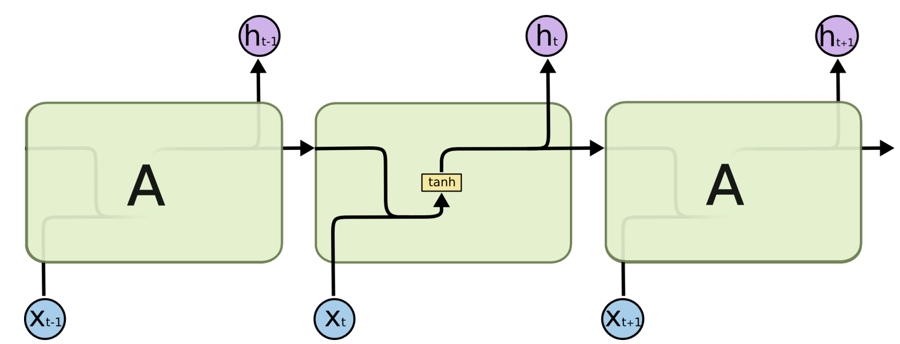


- Transformers LM: Process a segment of tokens

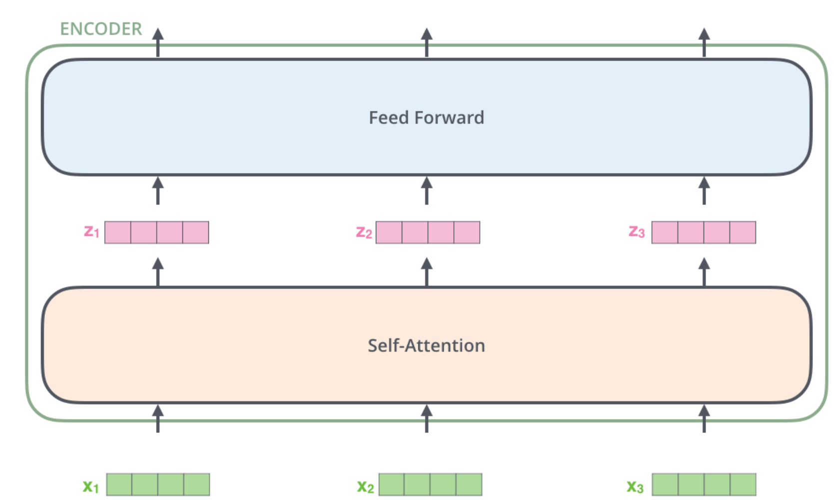


Transformers for Language Modelling

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

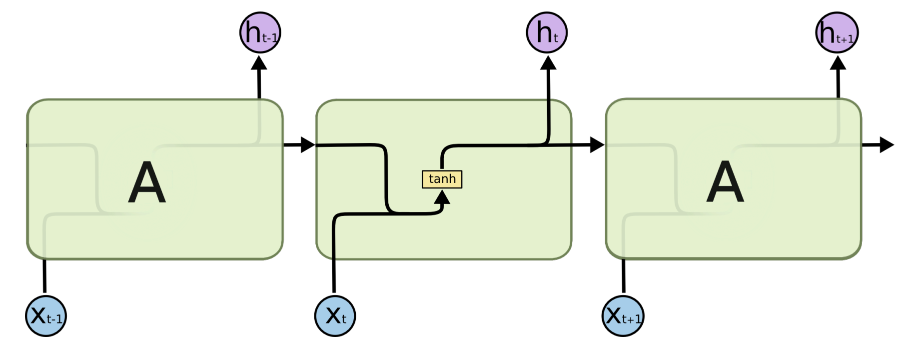


- Transformers LM: Process a segment of tokens
    - Dependencies within the segment

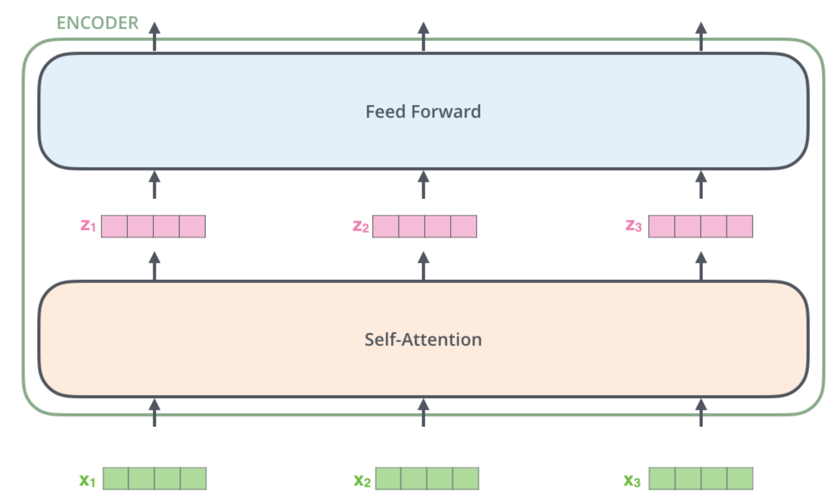


Transformers for Language Modelling

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

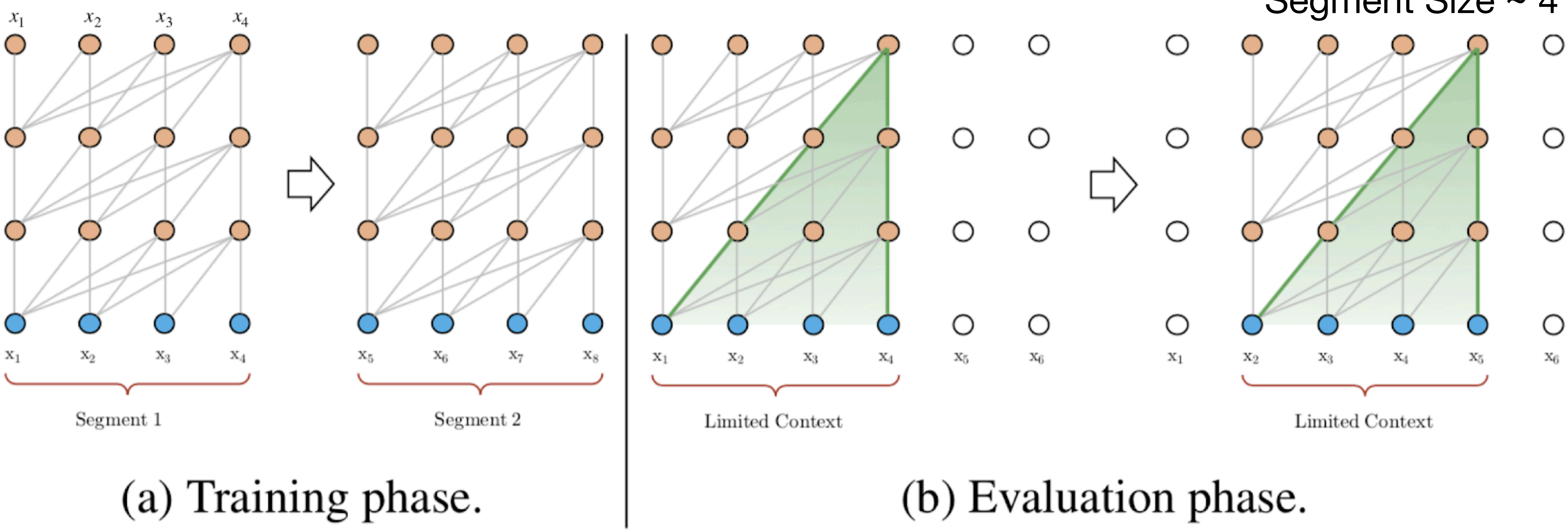


- Transformers LM: Process a segment of tokens
  - Dependencies within the segment
  - Within segment position is given by the positional encoding



Transformer LM processing of Segments

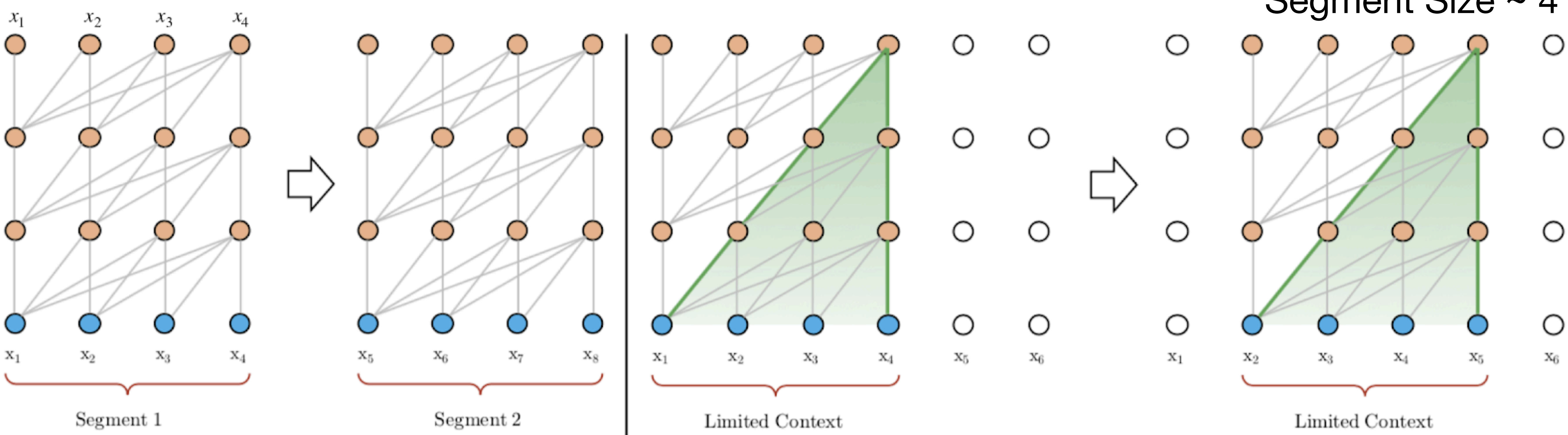
Segment Size ~ 4



Dai et al., 2019

Transformer LM processing of Segments

Segment Size ~ 4



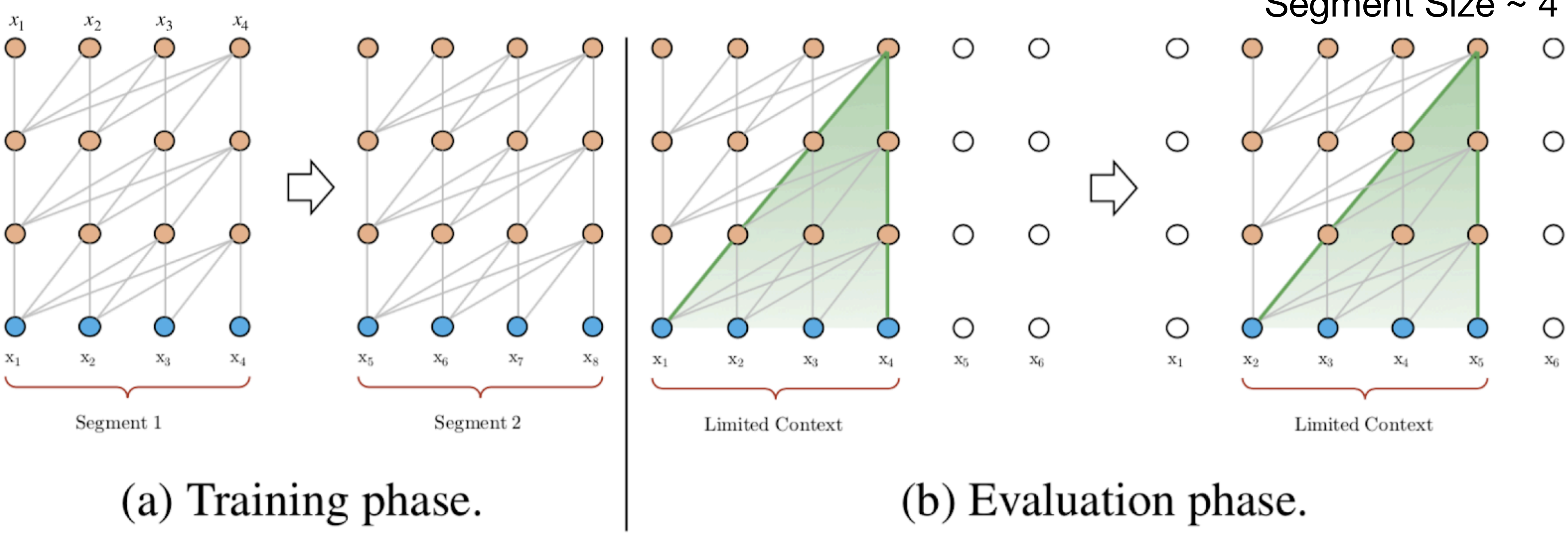
(a) Training phase. (b) Evaluation phase.

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

Summary

- NNLM:
- Challenges
  - Long-Term Dependencies
  - LSTMs
  - Transformers
    - Self Attention