

Neural Language Modeling

CSCI 1460: Computational Linguistics

Lecture 10

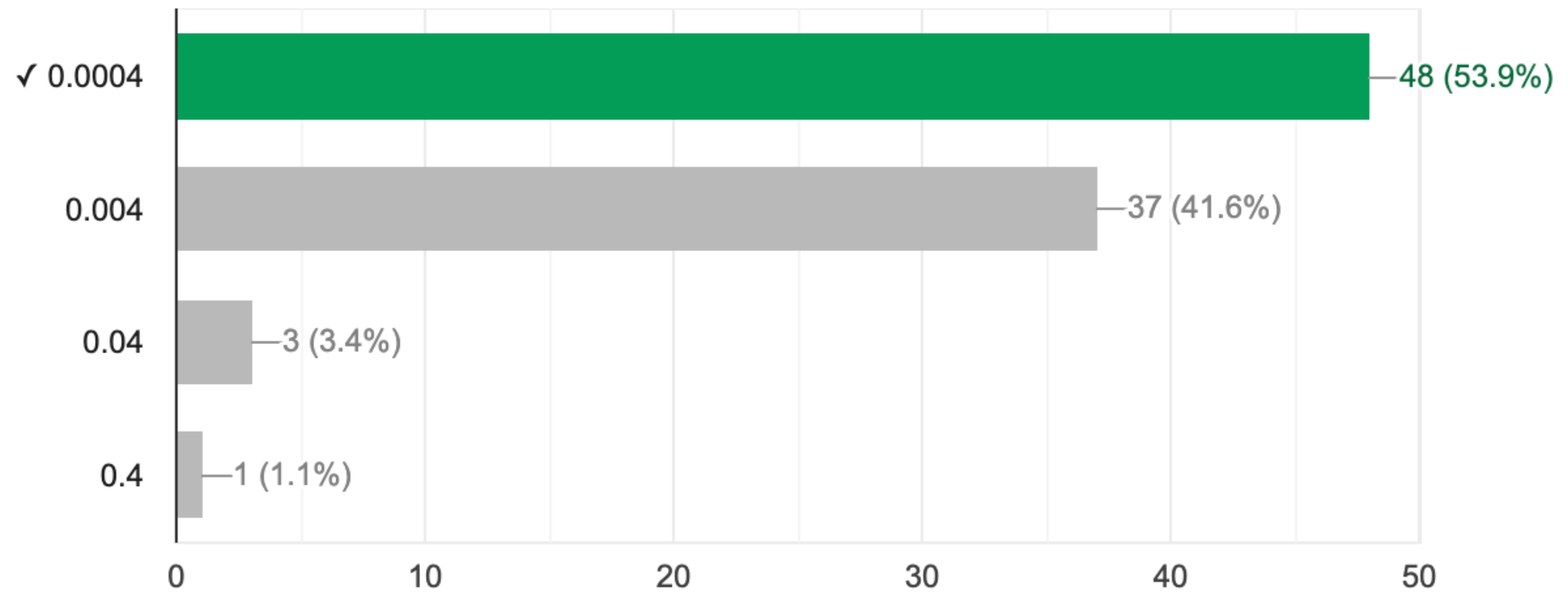
Ellie Pavlick

Fall 2022

Use the below corpus to train a unigram language model. Given your language model, what is the probability of the sentence "i say potato"? (Tip: You absolutely can do this by hand, but writing a short python program is easier, more fun, and you learn just as much! =D)



48 / 89 correct responses



```
[ ] corpus = [
    '<s> i say tomato </s>',
    '<s> you say tomato </s>',
    '<s> i like potatoes </s>',
    '<s> you like potatoes </s>',
    '<s> tomato tomato </s>',
    '<s> potato potato </s>'
]

corpus = [s.split() for s in corpus]
print(corpus)
```

```
▶ ugram_probs = {}
  for s in corpus:
    for w in s:
        if w not in ugram_probs:
            ugram_probs[w] = 0
            ugram_probs[w] += 1

    print(ugram_probs)
    total = sum(ugram_probs.values())
    print(total)
```

```
☞ {'<s>': 6, 'i': 2, 'say': 2, 'tomato': 4, '</s>': 6, 'you': 2, 'like': 2, 'potatoes': 2, 'potato': 2}
28
```

```
▶ from math import log

    new_sent = "i say potato"

    prob = 1
    for w in new_sent.split():
        prob *= ugram_probs[w]/total
    print(prob)
```

```
☞ 0.0003644314868804664
```

Are we supposed to count <s> and </s> as words????

```
[ ] corpus = [
    '<s> i say tomato </s>',
    '<s> you say tomato </s>',
    '<s> i like potatoes </s>',
    '<s> you like potatoes </s>',
    '<s> tomato tomato </s>',
    '<s> potato potato </s>'
]

corpus = [s.split() for s in corpus]
print(corpus)
```

```
▶ ugram_probs = {}
for s in corpus:
    for w in s:
        if w not in ugram_probs:
            ugram_probs[w] = 0
        ugram_probs[w] += 1

print(ugram_probs)
total = sum(ugram_probs.values())
print(total)
```

```
☞ {'<s>': 6, 'i': 2, 'say': 2, 'tomato': 4, '</s>': 6, 'you': 2, 'like': 2, 'potatoes': 2, 'potato': 2}
28
```

```
▶ from math import log

new_sent = "i say potato"

prob = 1
for w in new_sent.split():
    prob *= ugram_probs[w]/total
print(prob)
```

```
☞ 0.0003644314868804664
```

usually don't include <s> and </s> in unigram models, but I had done so in computing this answer. 🙄

Topics

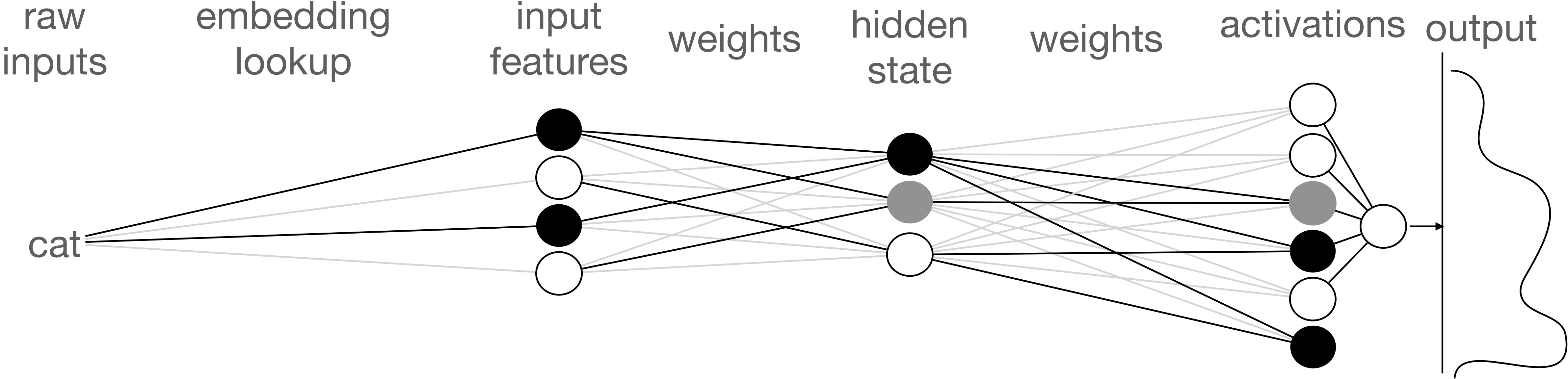
- NN Architectures for Language Modeling
 - MLP
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - Transformer

Topics

- NN Architectures for Language Modeling
 - **MLP**
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - Transformer

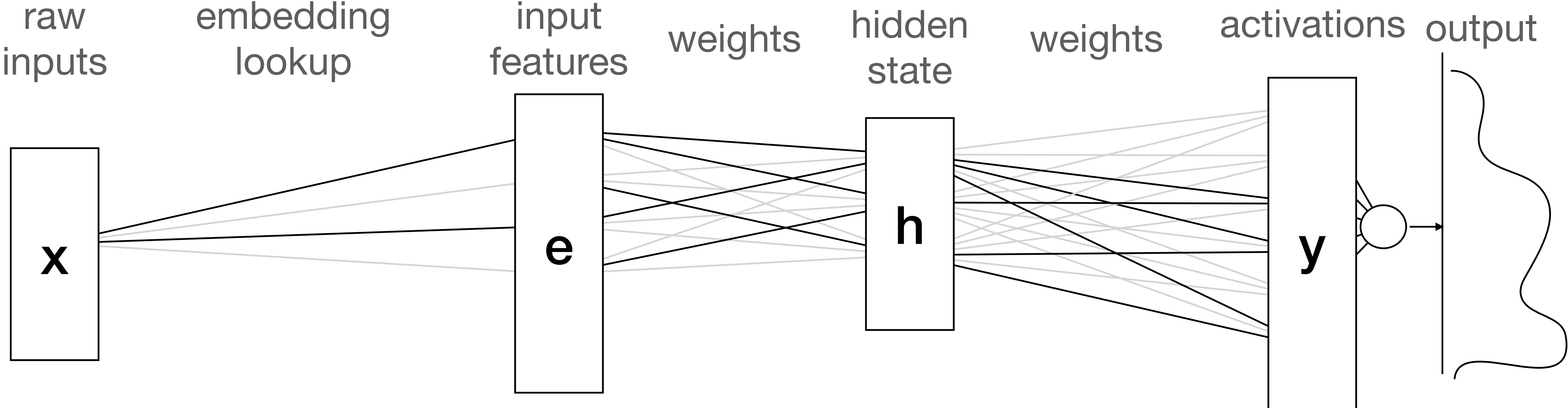
Multilayer Perceptron

Task: Predict the next word
Input: cat
Expected: sat



Multilayer Perceptron

Task: Predict the next word
Input: cat
Expected: sat



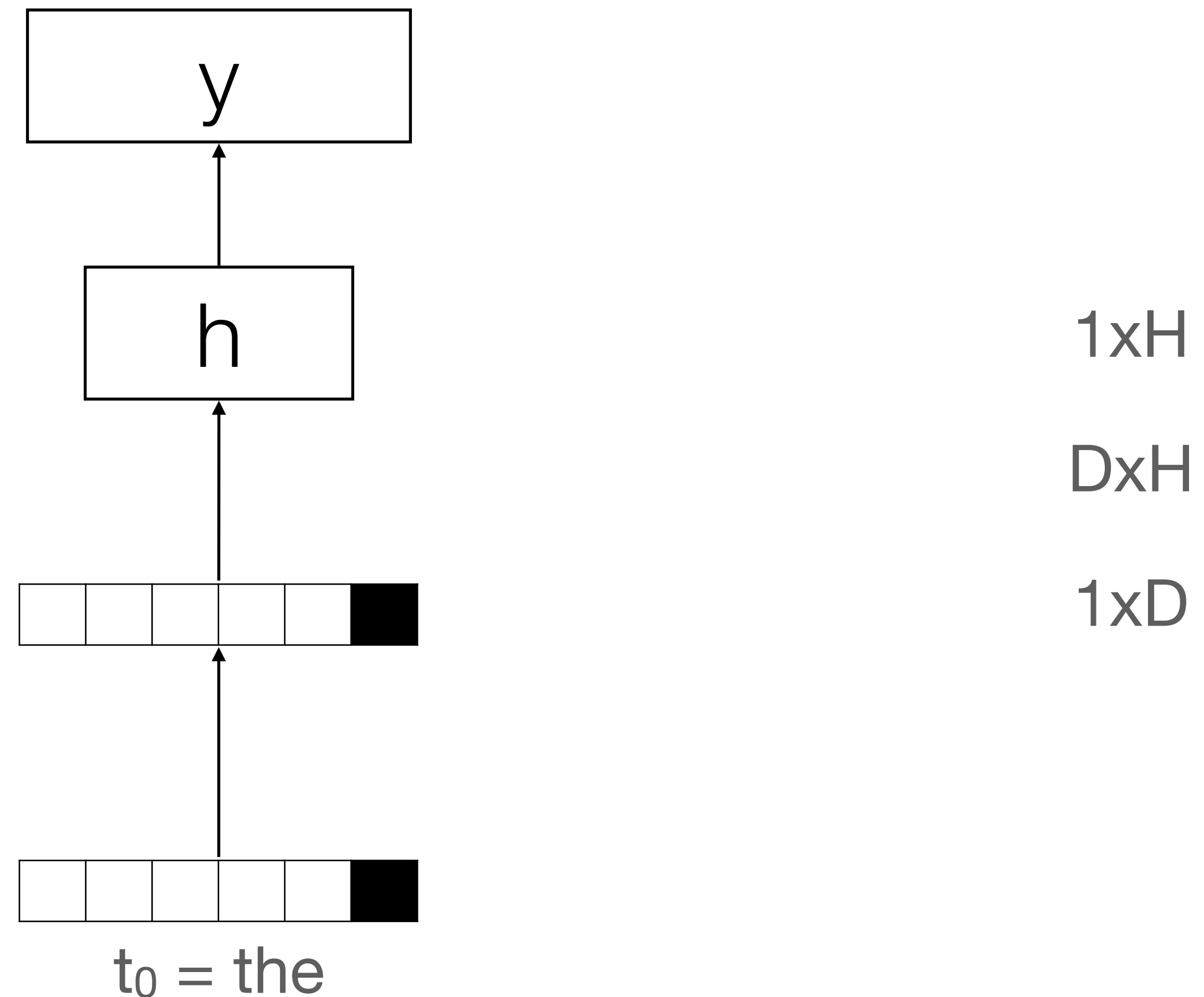
Multilayer Perceptron

MLP for Language Modeling

Task: Predict the next word

Input: the

Expected: cat



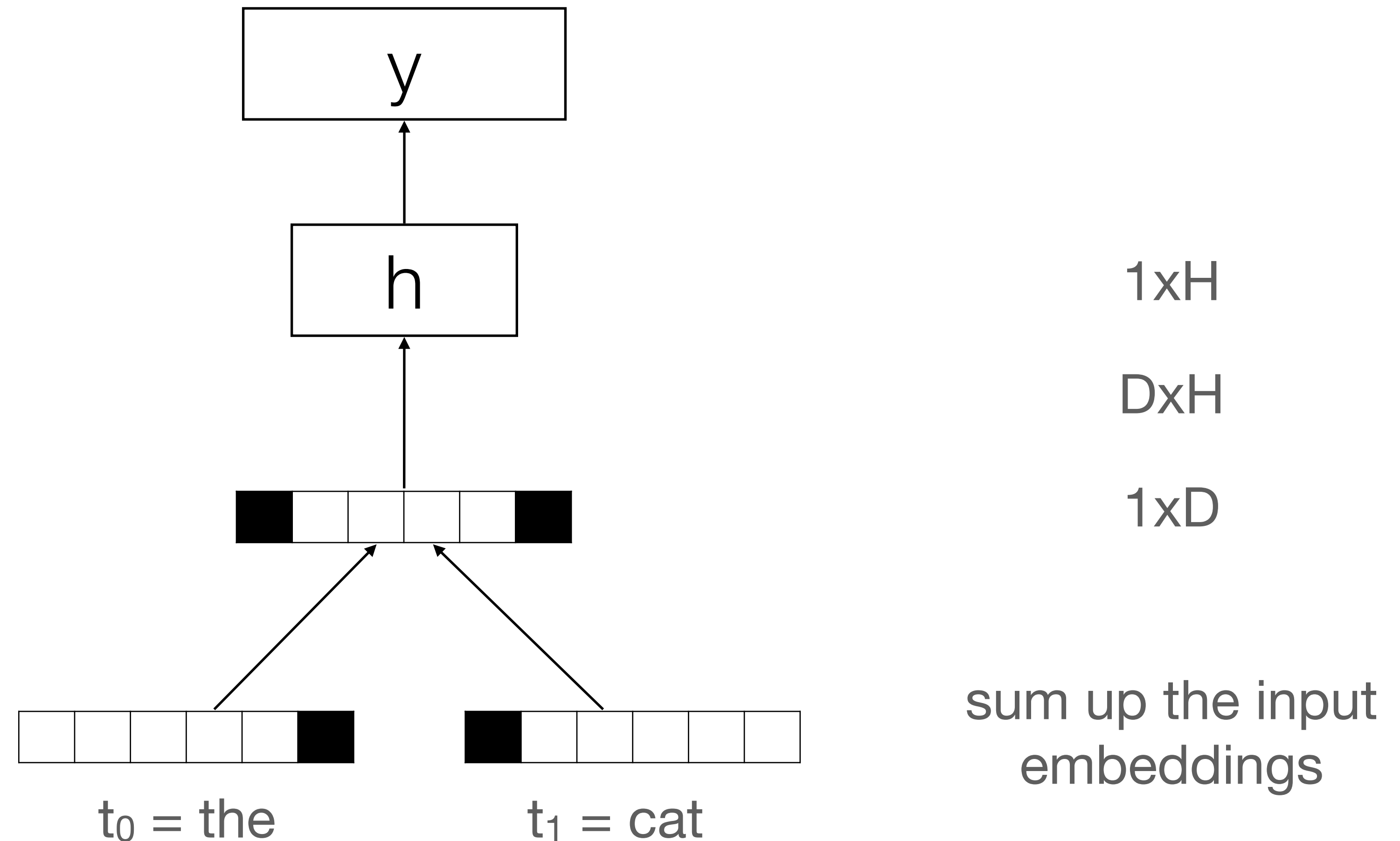
Multilayer Perceptron

MLP for Language Modeling

Task: Predict the next word

Input: the cat

Expected: sat



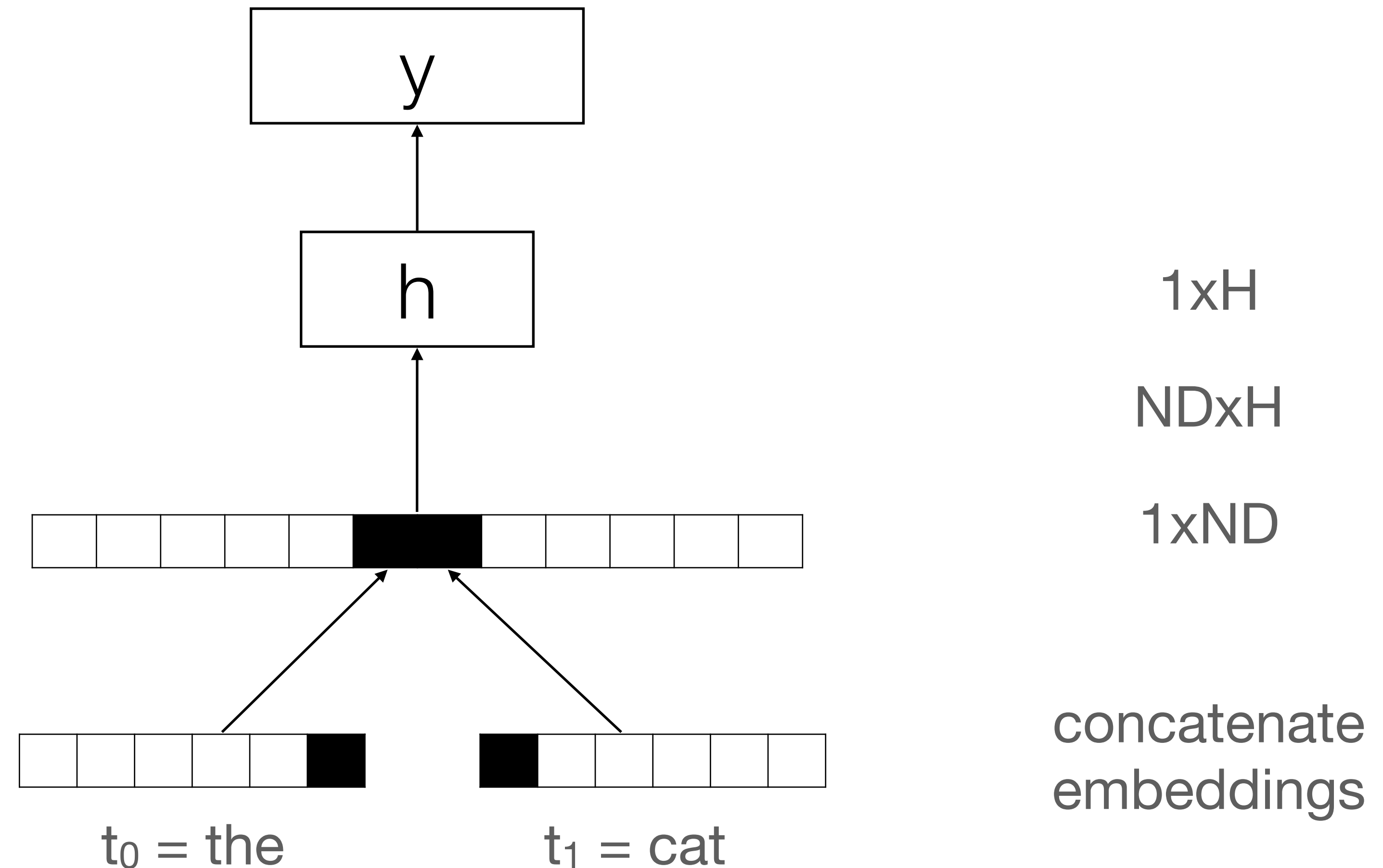
Multilayer Perceptron

MLP for Language Modeling

Task: Predict the next word

Input: the cat

Expected: sat



Multilayer Perceptron

MLP for Language Modeling

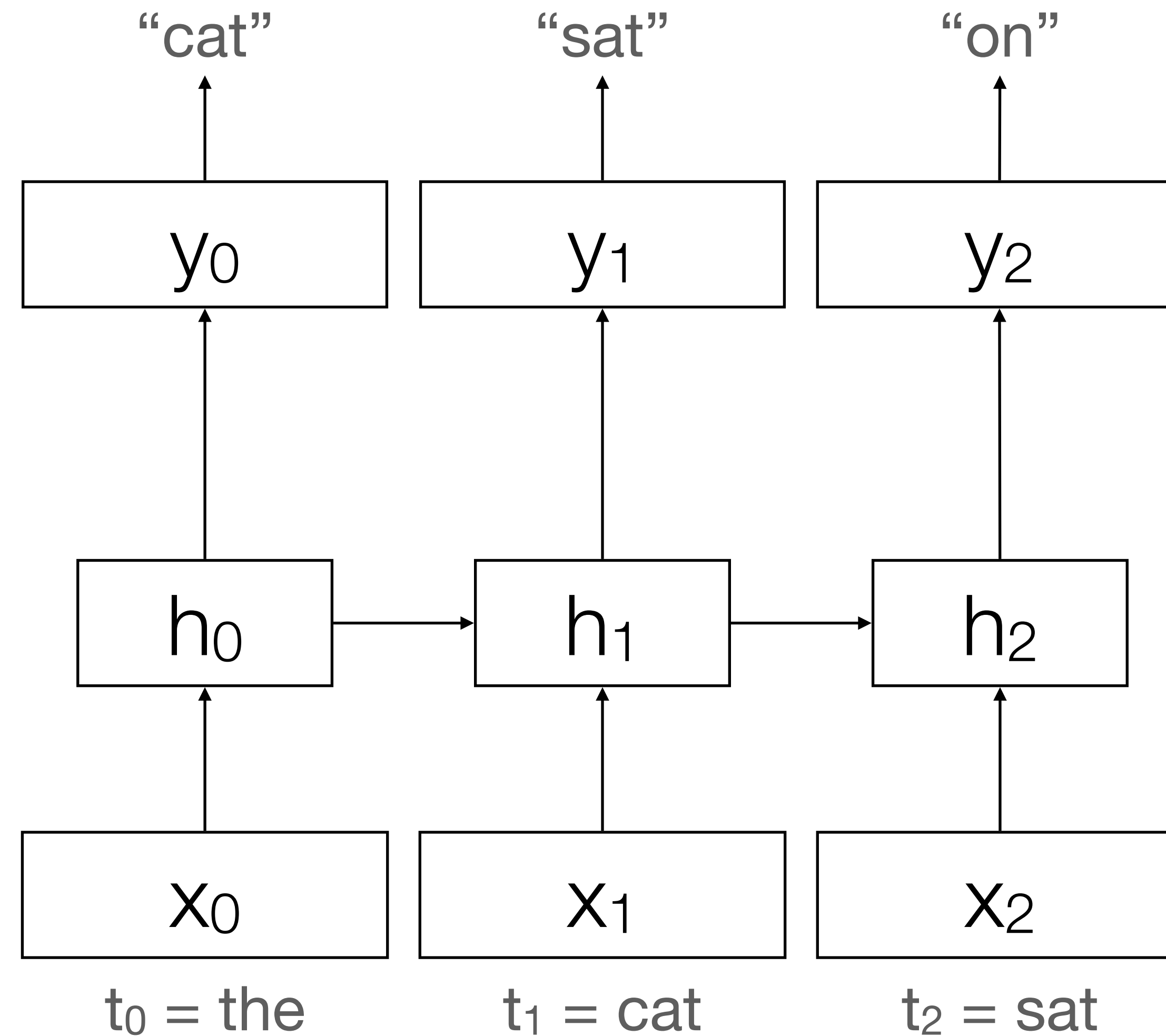
- MLP doesn't readily support long, sequential inputs
- No way of encoding word order
 - Essentially a BOW model
- Inputs either become muddy (adding everything together, i.e., “bag of vectors”) or too large (concatenating everything)
- Still, “bag-of-vectors” classifiers are common and often work well for basic applications

Topics

- NN Architectures for Language Modeling
 - ~~MLP~~
 - **Recurrent Neural Network (RNN)**
 - Long-Short Term Memory Network (LSTM)
 - Transformer

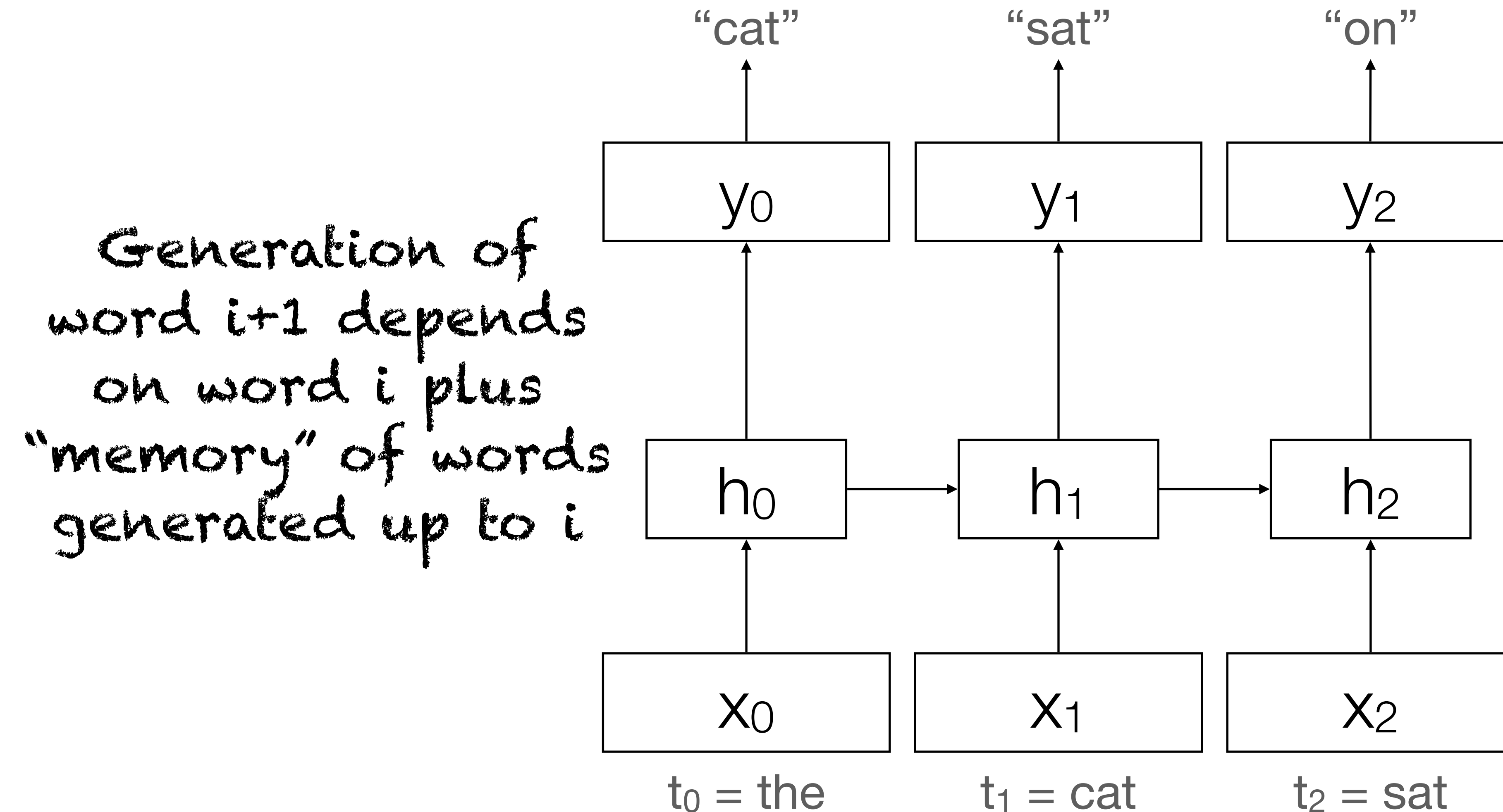
Recurrent Neural Networks (RNNs)

Architecture



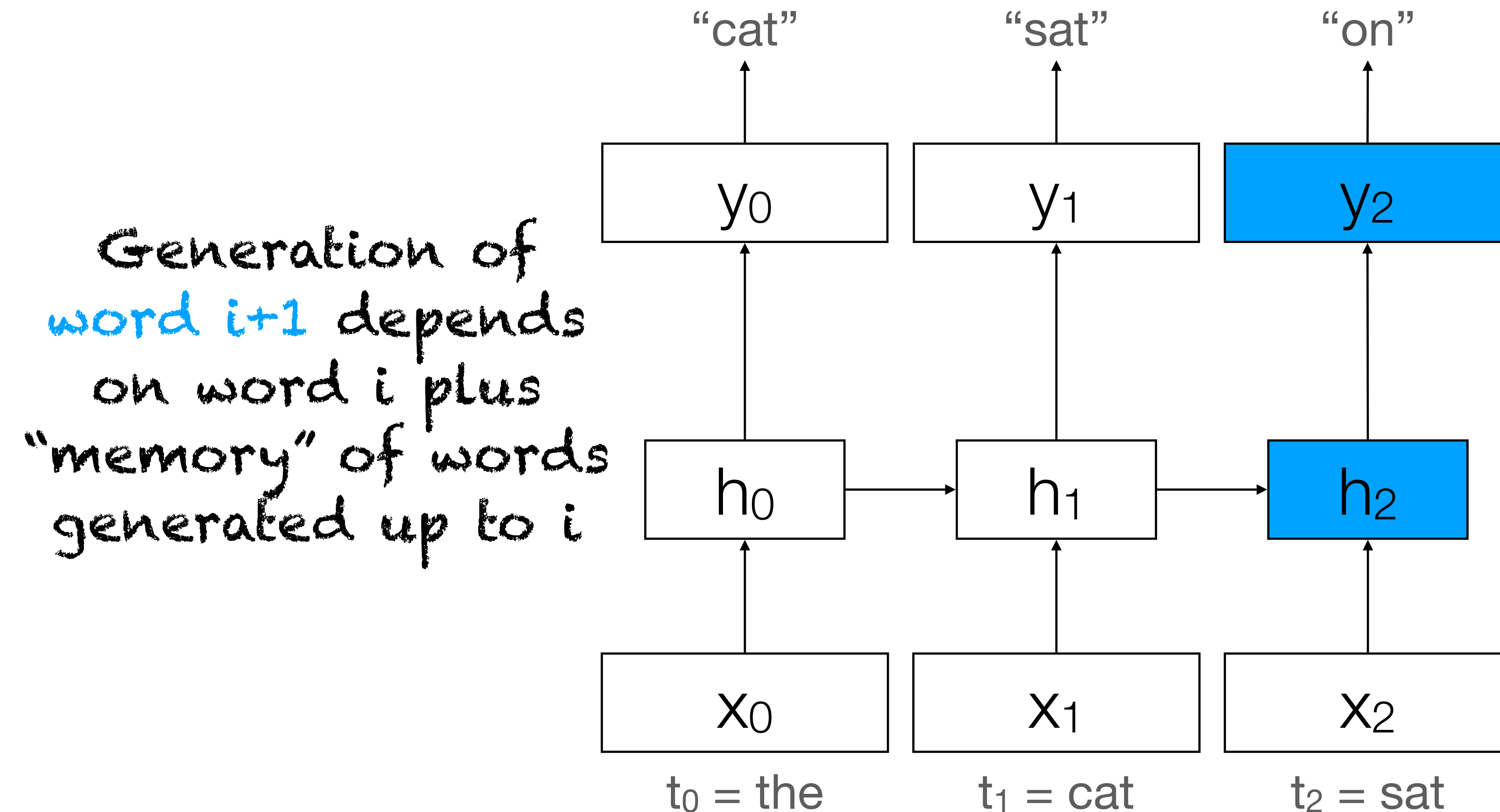
Recurrent Neural Networks (RNNs)

Architecture



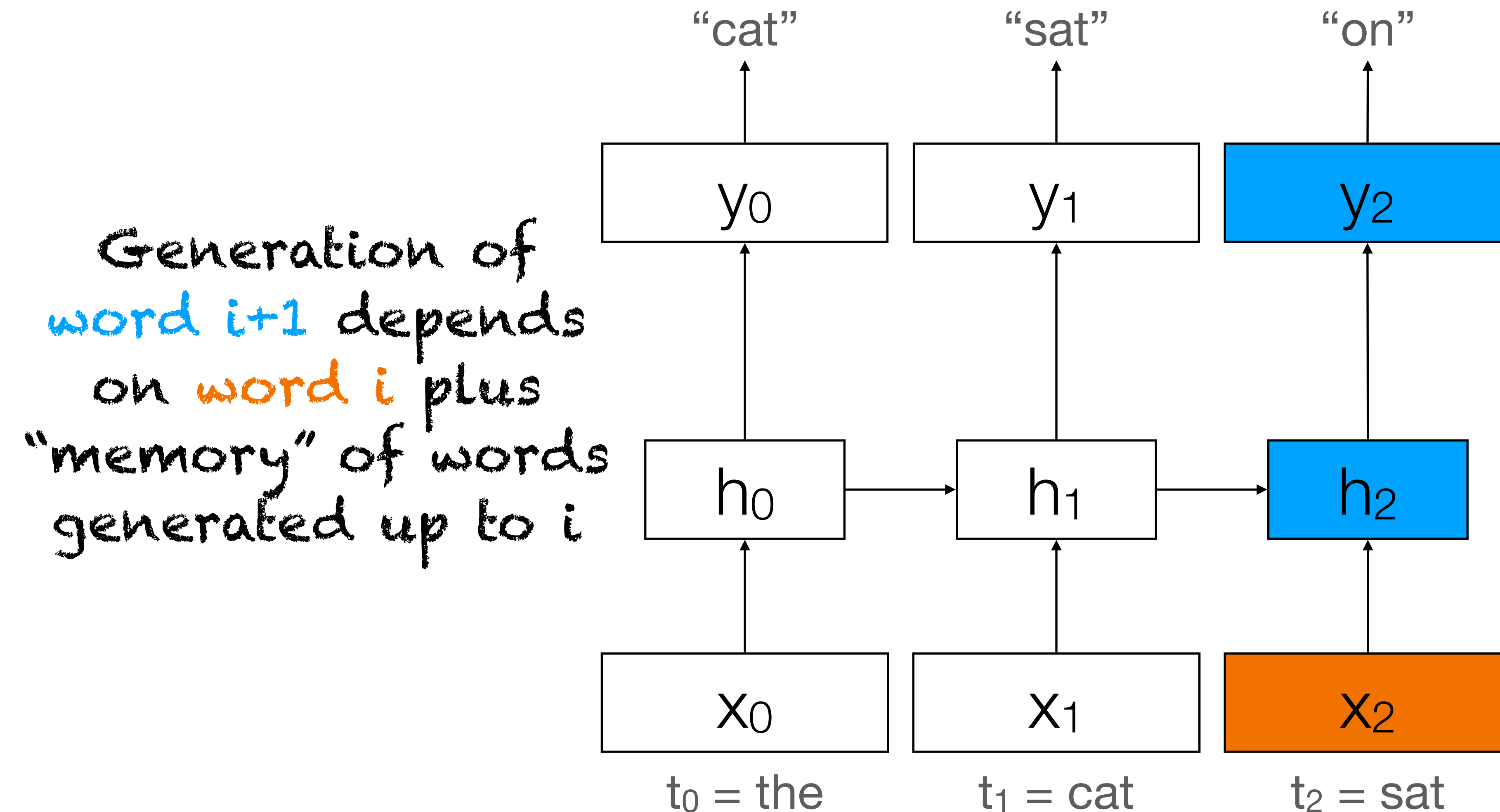
Recurrent Neural Networks (RNNs)

Architecture



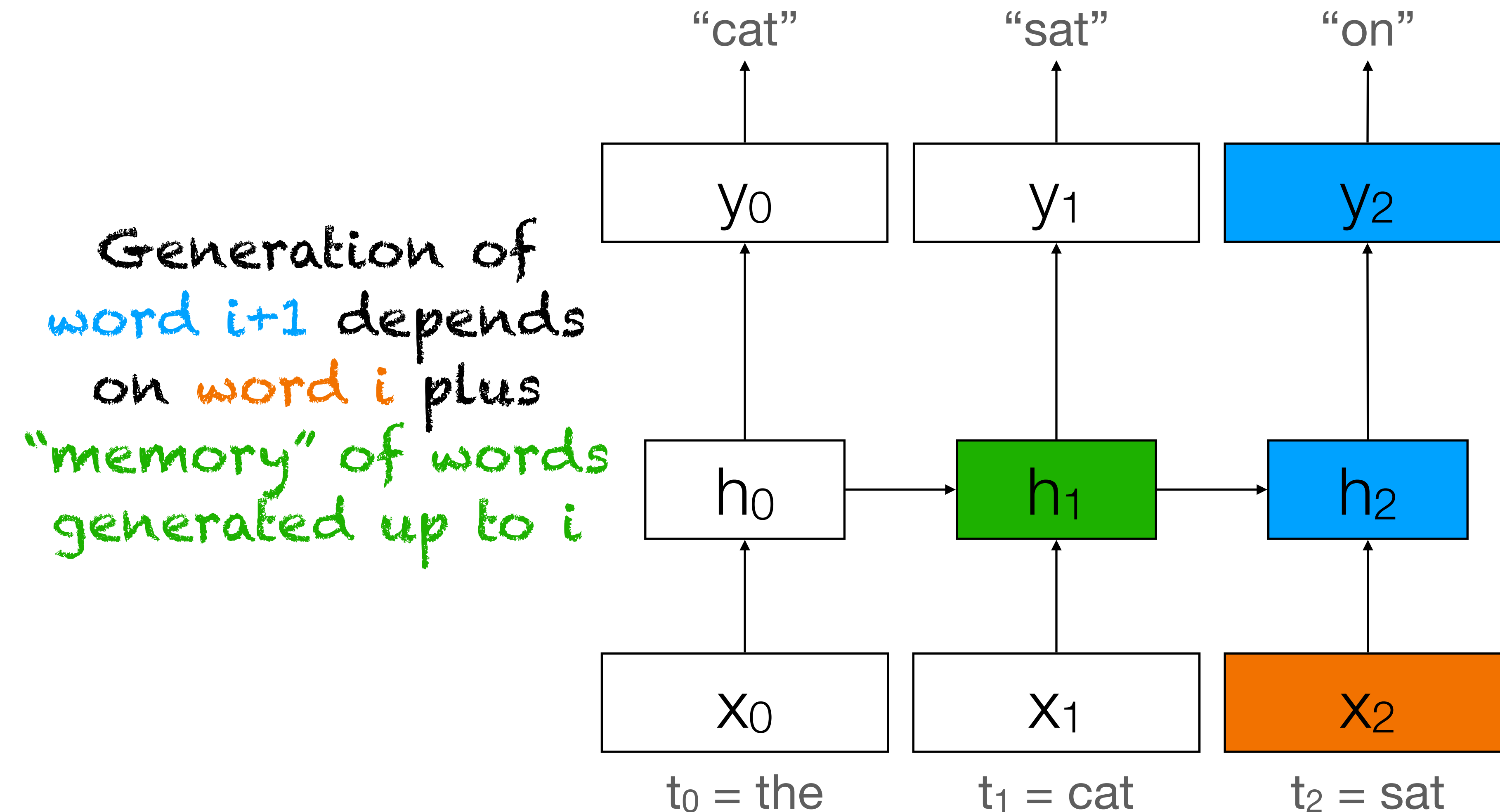
Recurrent Neural Networks (RNNs)

Architecture



Recurrent Neural Networks (RNNs)

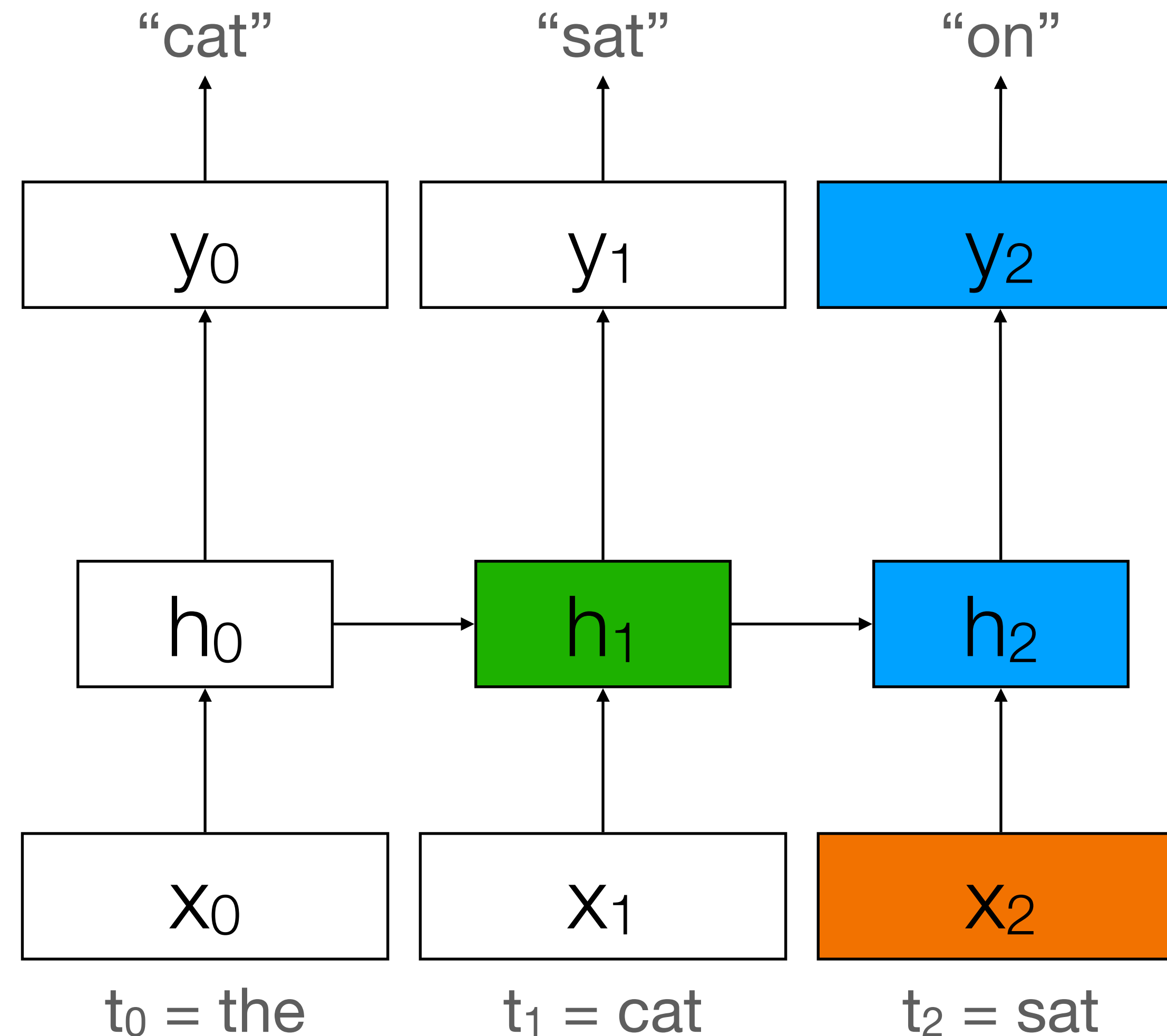
Architecture



Recurrent Neural Networks (RNNs)

Architecture

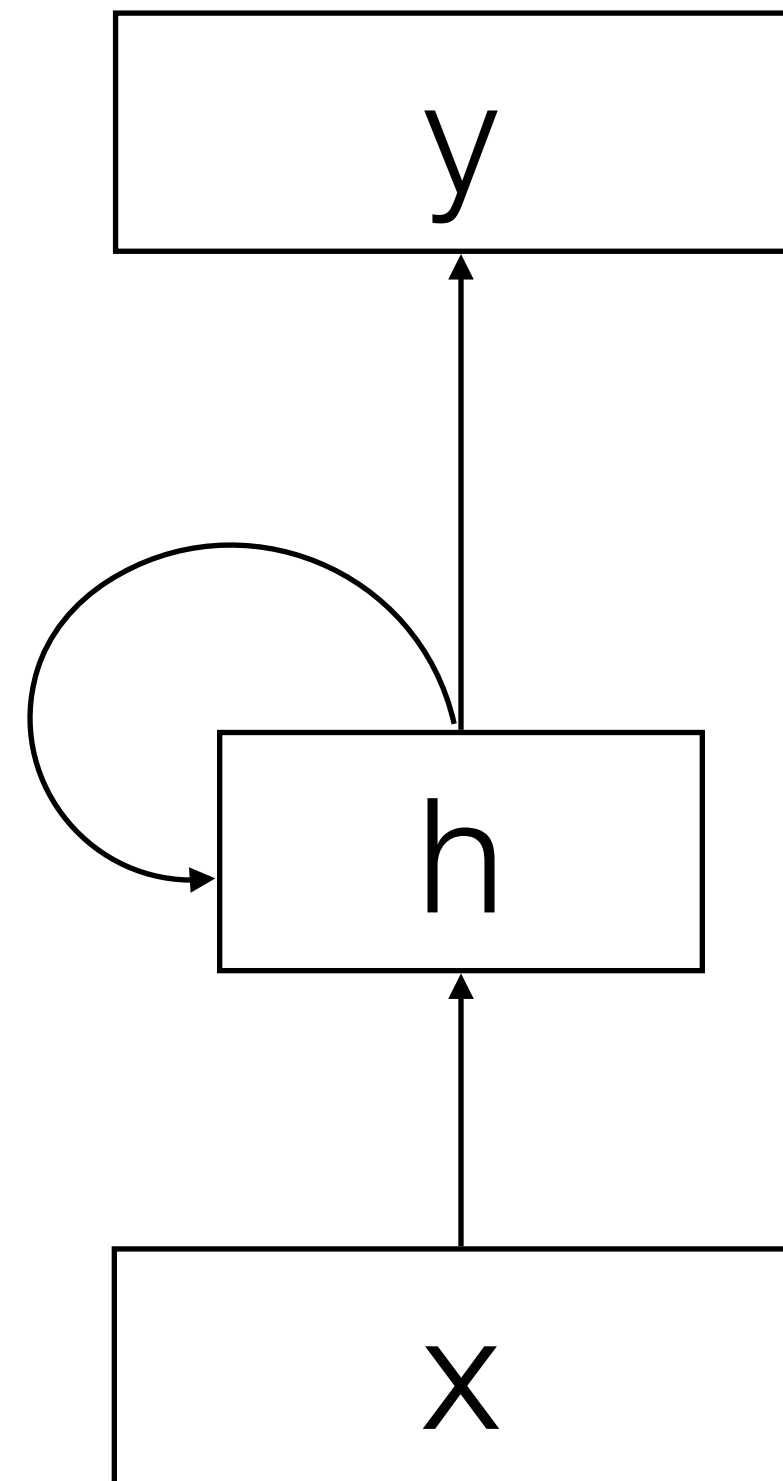
View #1:
"Unrolled"



Recurrent Neural Networks (RNNs)

Architecture

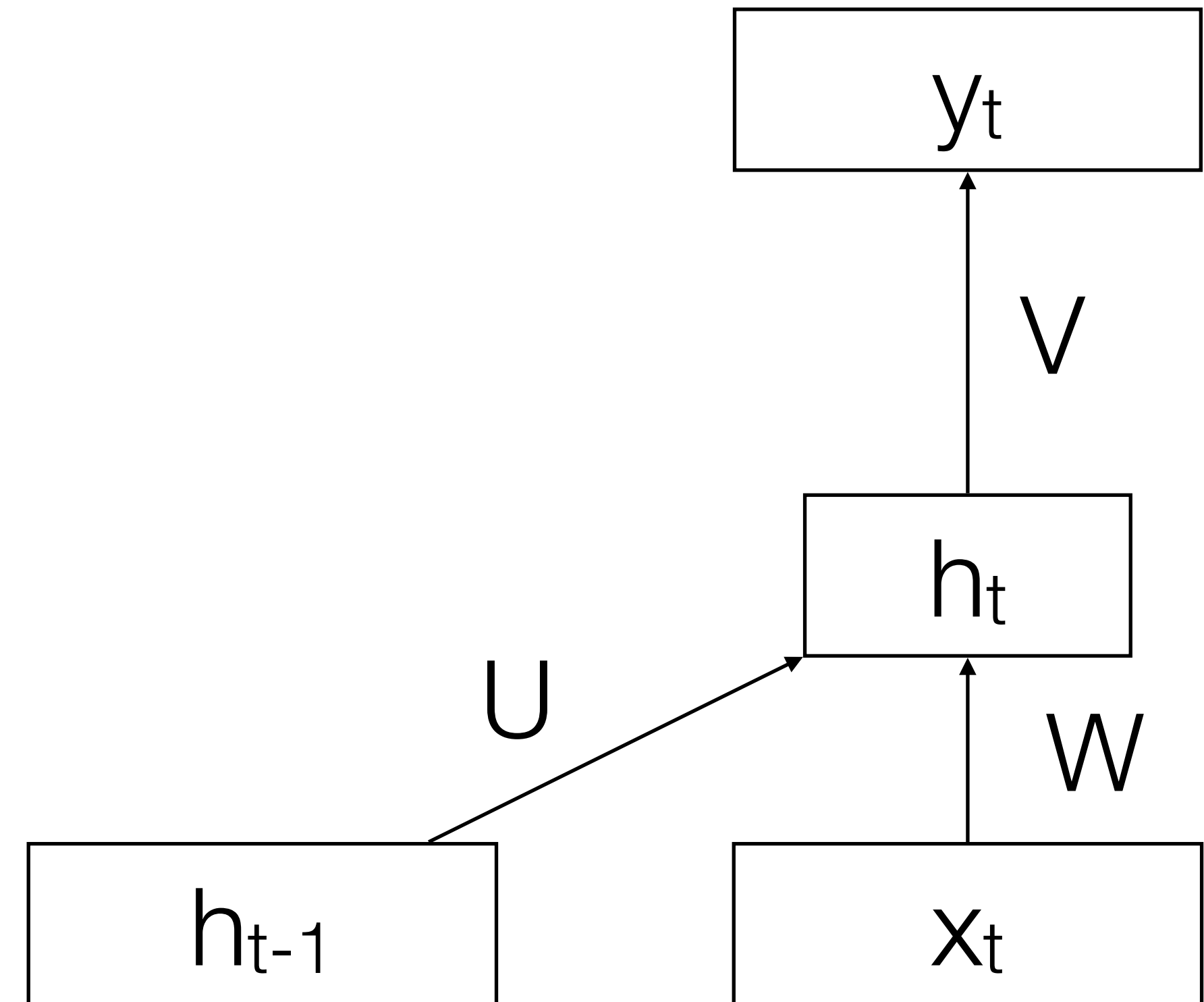
View #2:
Recurrent/
Recursive



Recurrent Neural Networks (RNNs)

Architecture

View #3:
(A single step of)
recurrent/recursive



Recurrent Neural Networks (RNNs)

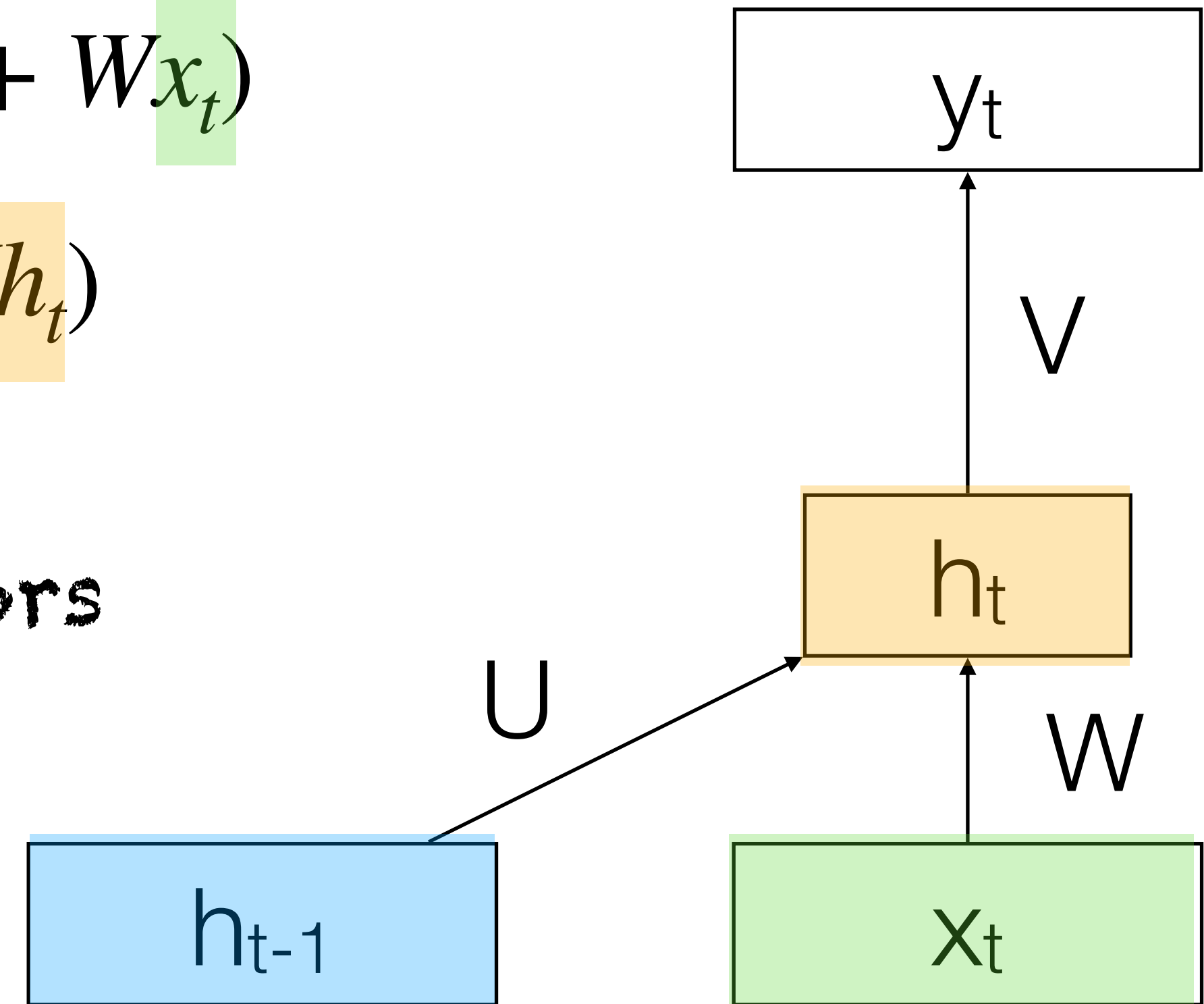
Architecture

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

View #3:
(A single step of)
recurrent/recursive

Vectors



Recurrent Neural Networks (RNNs)

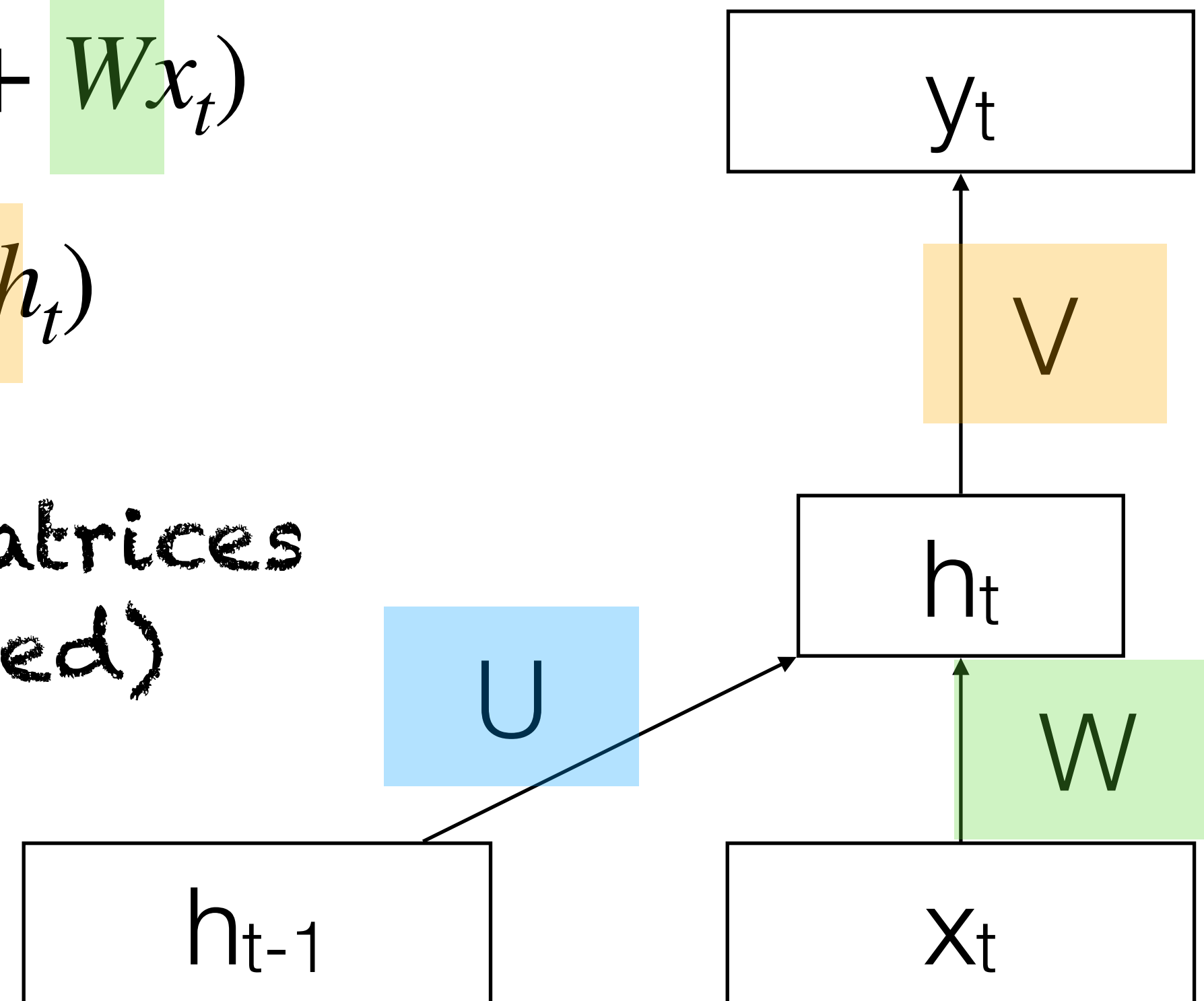
Architecture

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

View #3:
(A single step of)
recurrent/recursive

Weight Matrices
(Learned)



Recurrent Neural Networks (RNNs)

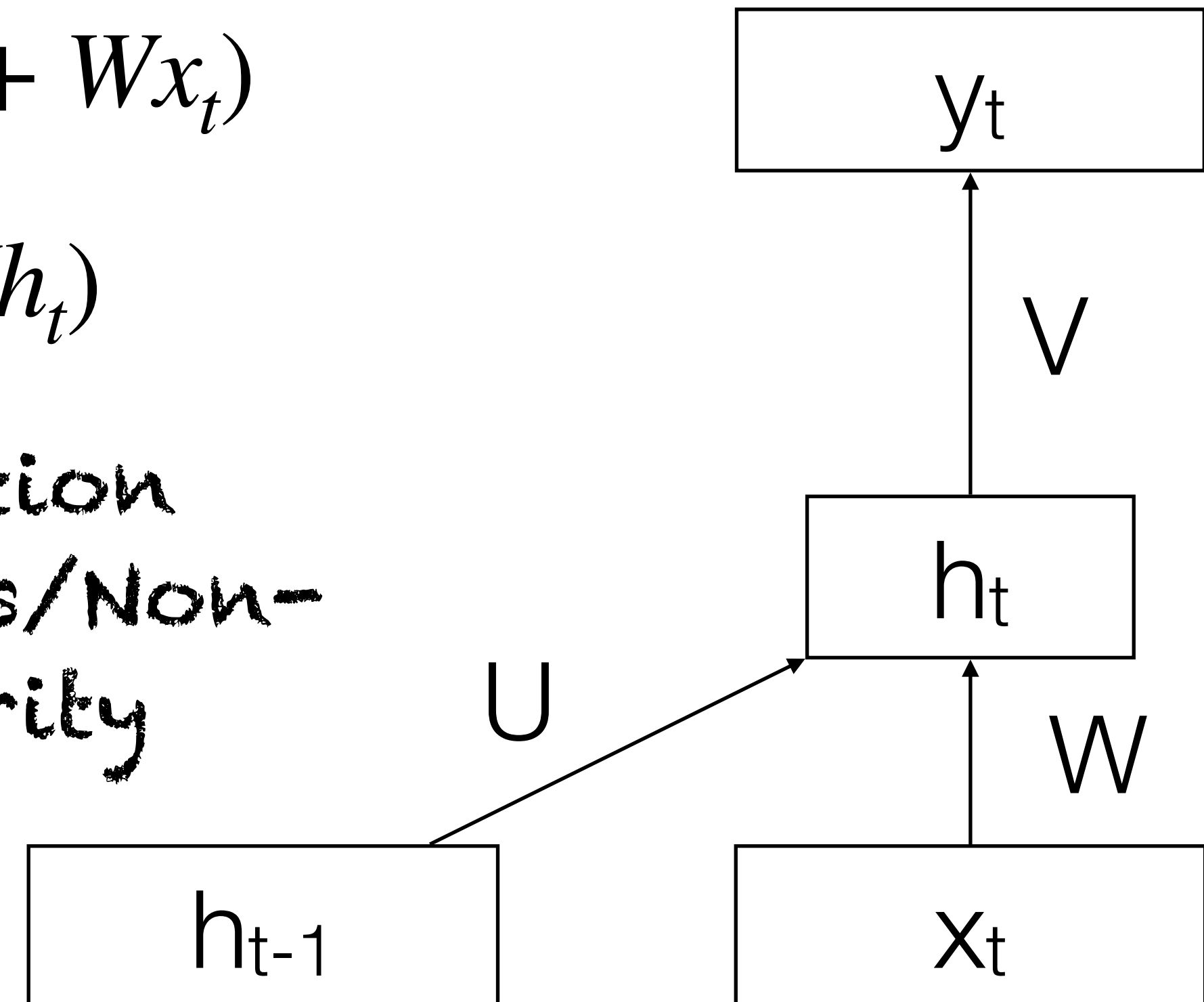
Architecture

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

View #3:
(A single step of)
recurrent/recursive

Activation
Functions/Non-
Linearity



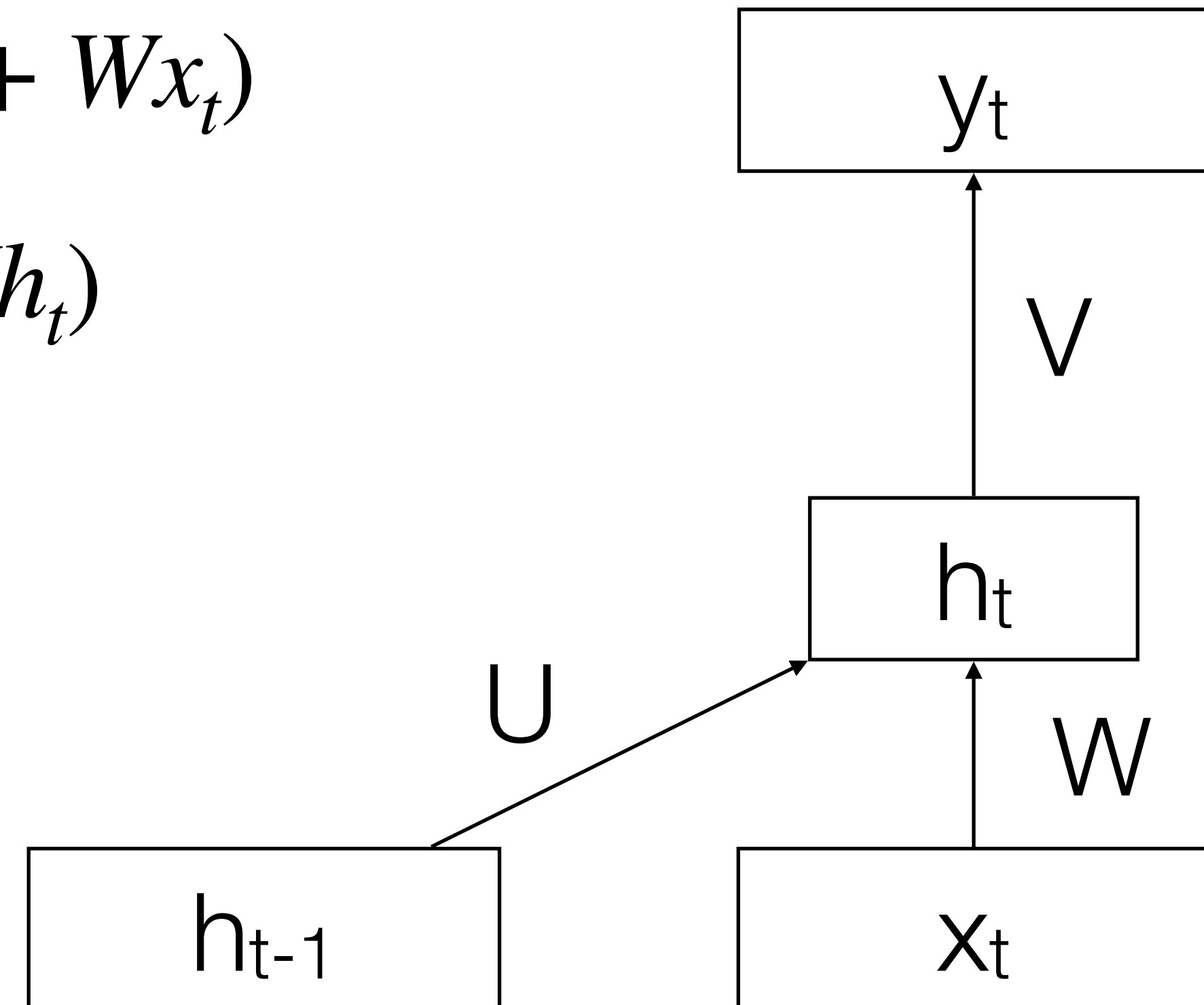
Recurrent Neural Networks (RNNs)

Architecture

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

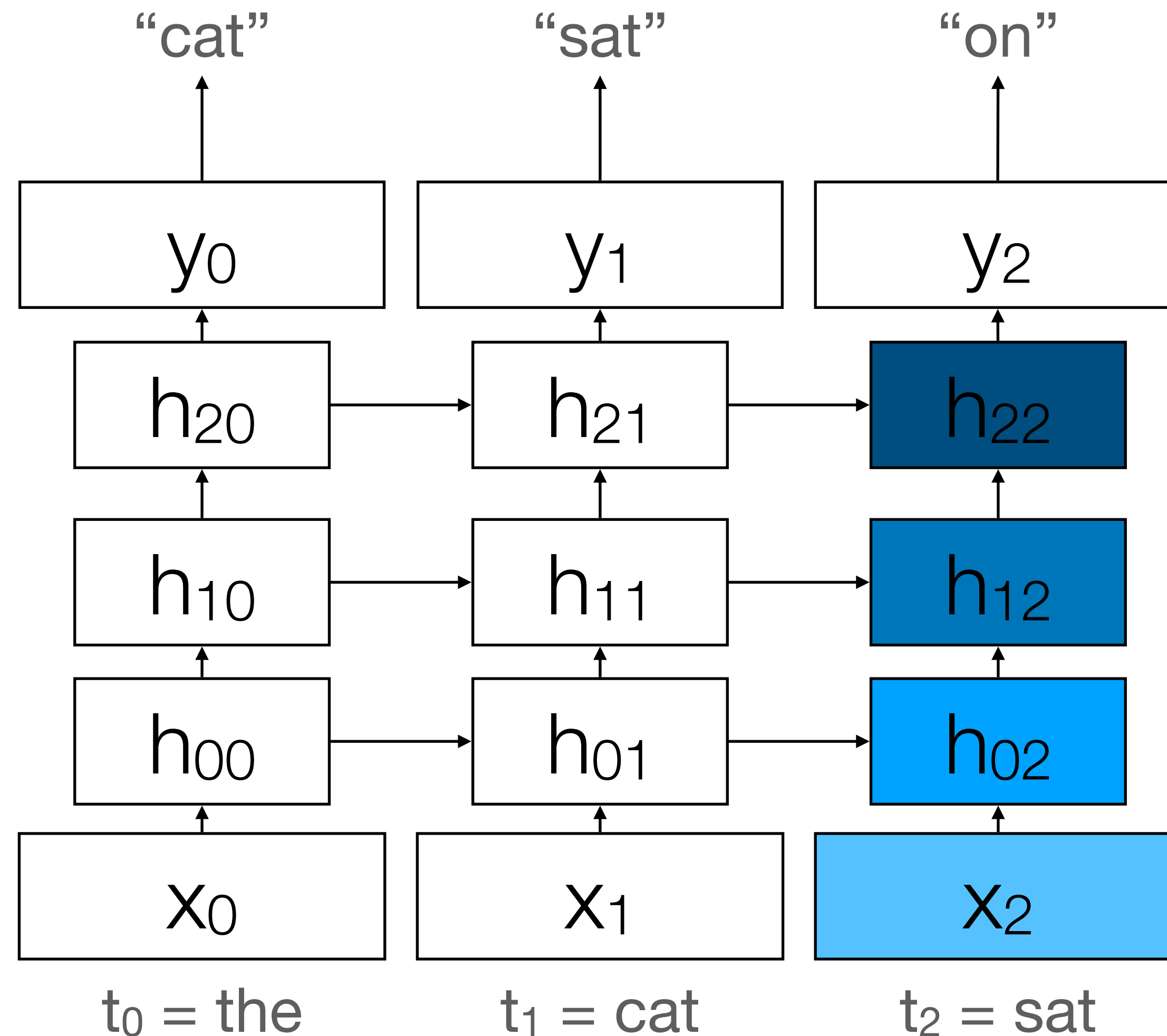
View #3:
(A single step of)
recurrent/recursive



Recurrent Neural Networks (RNNs)

Architecture

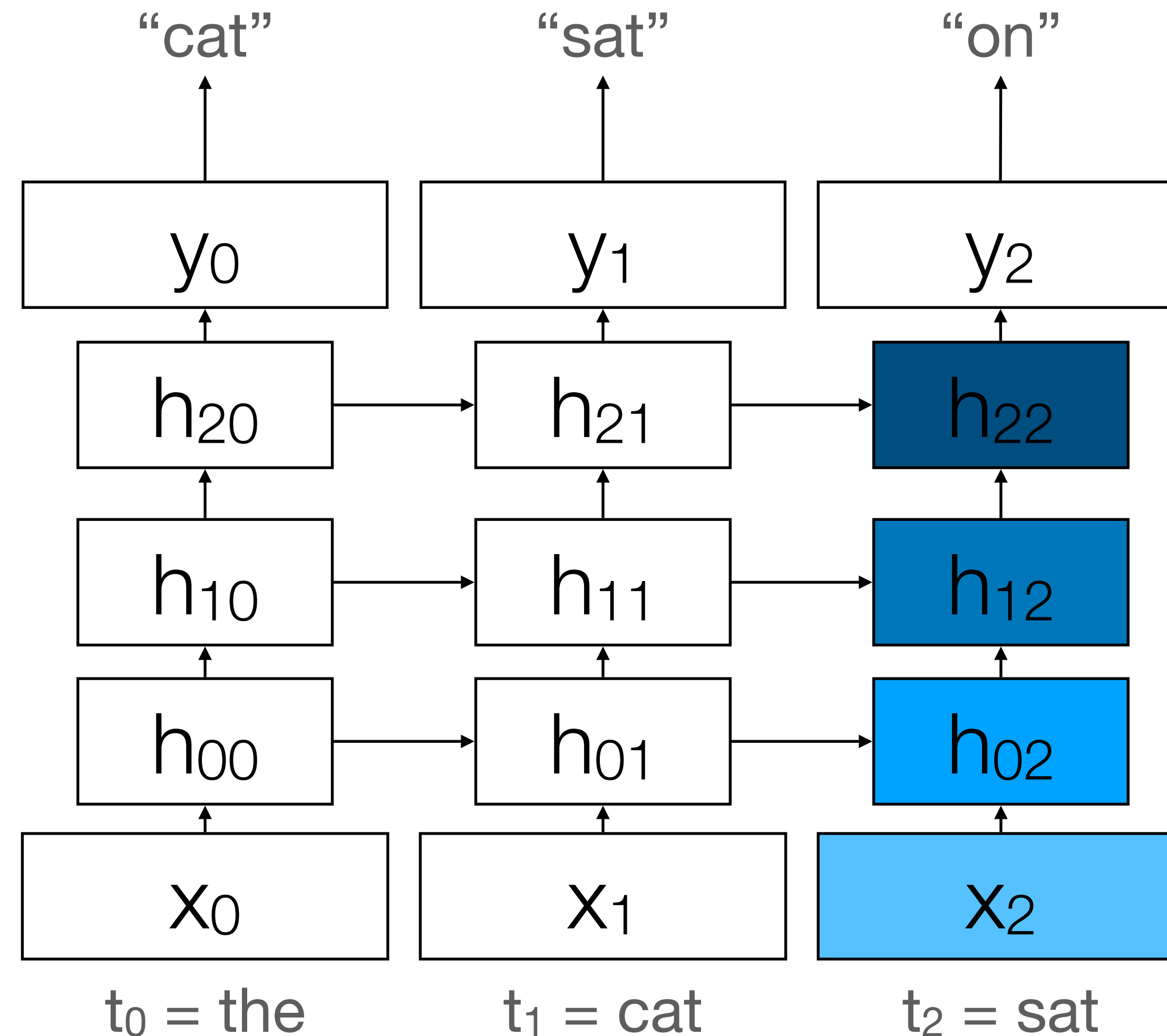
"Stacked" RNNs
(more later)



Recurrent Neural Networks (RNNs)

Architecture

"Stacked" RNNs
(more later)



allow
increasingly
abstract in-
context
representations

Recurrent Neural Networks (RNNs)

Inference

function FORWARDRNN(\mathbf{x} , *network*) **returns** output sequence \mathbf{y}

$\mathbf{h}^0 \leftarrow 0$

for $i \leftarrow 1$ **to** LENGTH(\mathbf{x}) **do**

$\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$

$\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$

return \mathbf{y}

Recurrent Neural Networks (RNNs)

Training Considerations

- Recurrent or Unrolled? Typically, in practice, unrolled and padded to a fixed length
 - Better for batching
- “Teacher Forcing”
 - When producing word i , predict based on the *real* $i-1$, not the predicted $i-1$ (which is likely wrong)
 - Student forcing = use the predicted $i-1$
 - Sometimes people mix teacher and student forcing

Topics

- NN Architectures for Language Modeling
 - ~~MLP~~
 - Recurrent Neural Network (RNN)
 - **Long-Short Term Memory Network (LSTM)**
 - Transformer

Long-Short Term Memory Network (LSTM)

Motivation

- RNNs struggle with “long range dependencies”
 - “The flights the airline was cancelling were full”
- Some challenges:
 - h trying to do too much
 - “vanishing gradients” make it hard to update early hidden states for long sequences

Long-Short Term Memory Network (LSTM)

Architecture

- Introduce a “gating” mechanisms which manages the hidden state/memory
- Break this up into two processes:
 - *forget gate* which removes information no longer needed
 - *add gate* which adds new information likely to be useful in the future
- Also adds explicit previous “context” in addition to prior hidden state

Long-Short Term Memory Network (LSTM)

Architecture

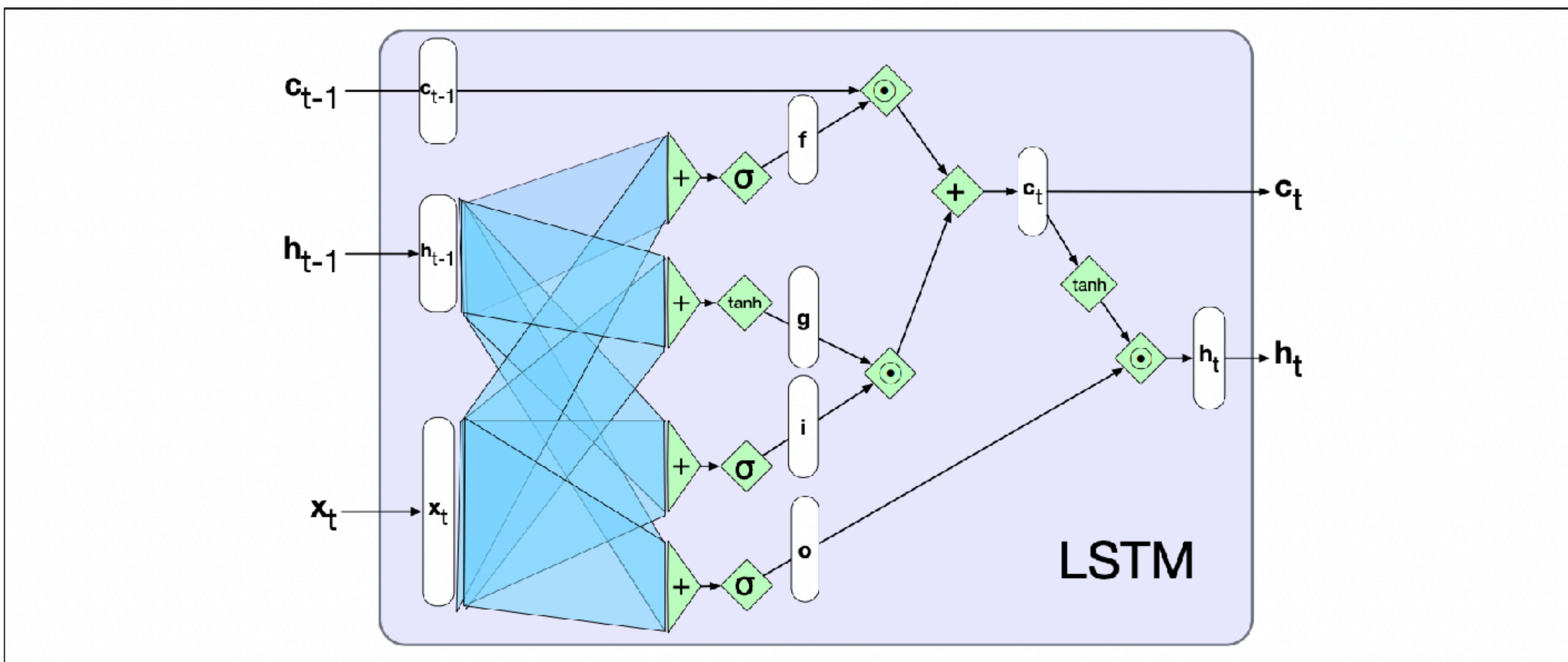


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

input and
hidden
state, same
as RNN

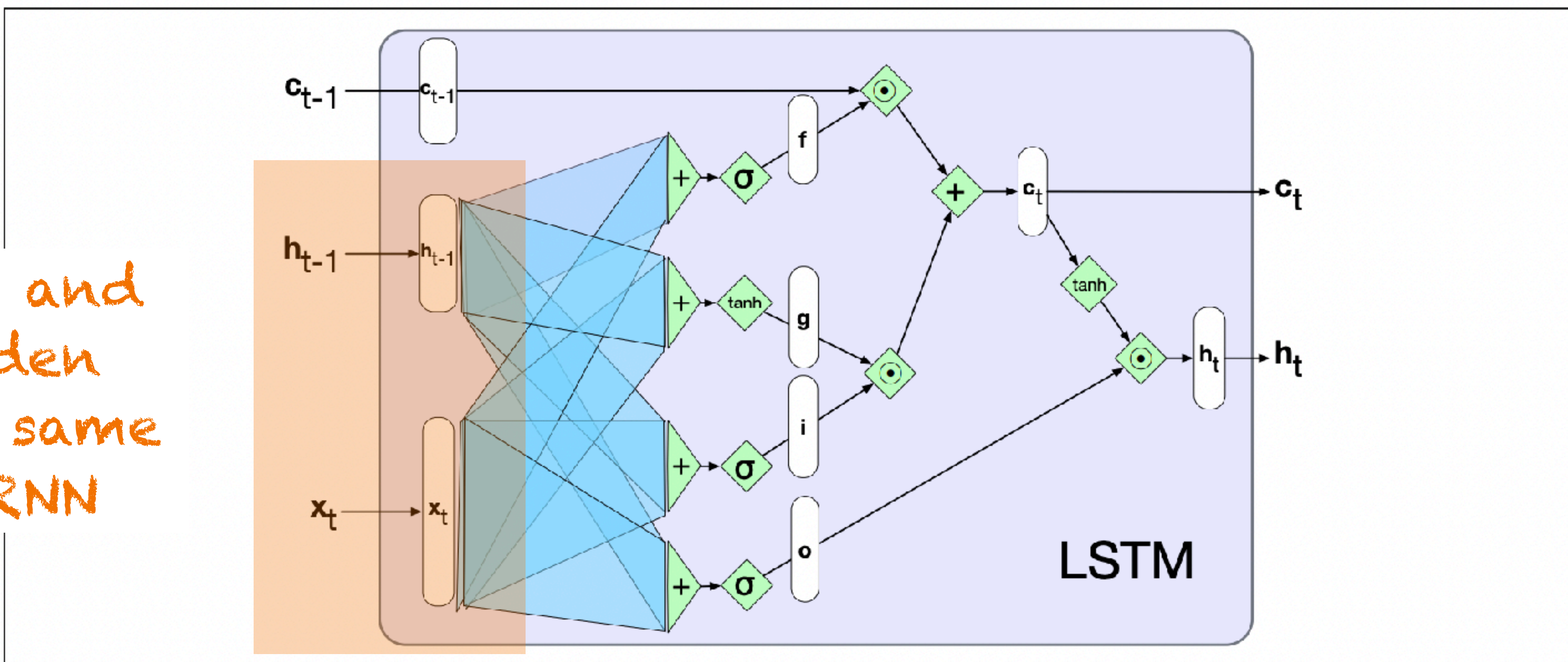


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

processing of current input,
same as in RNN

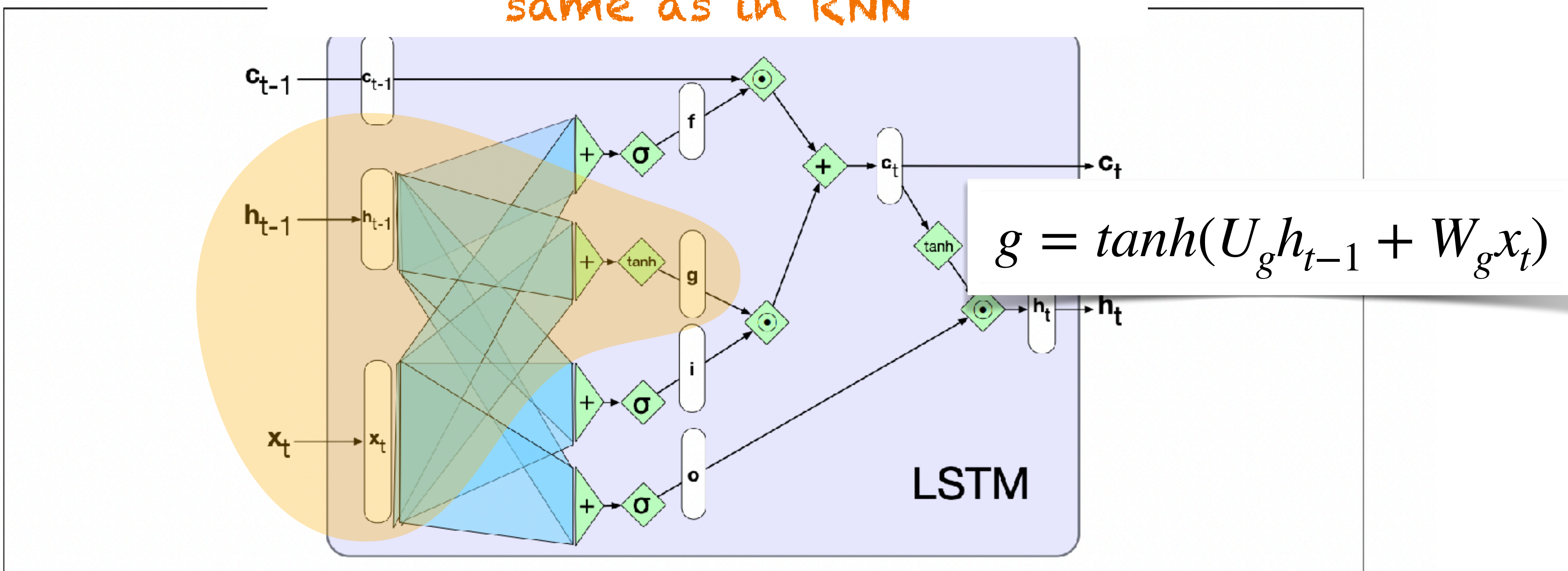


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

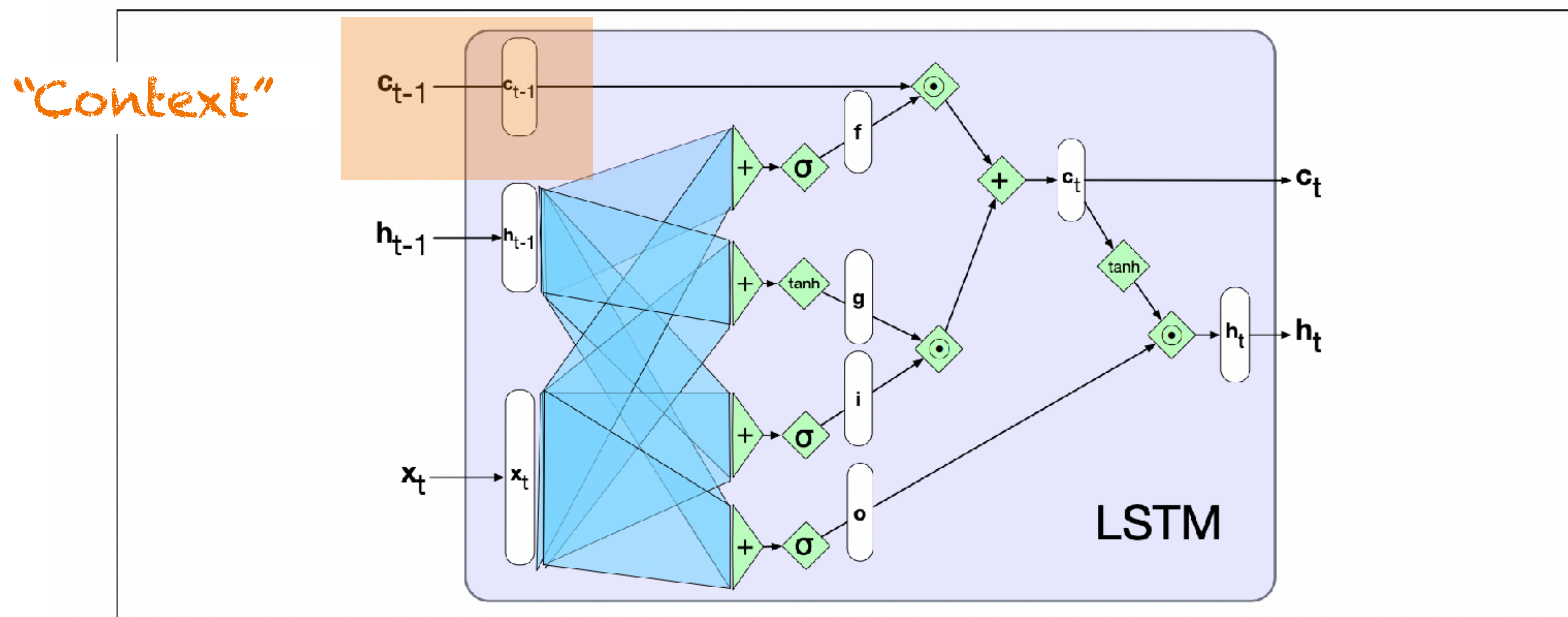


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

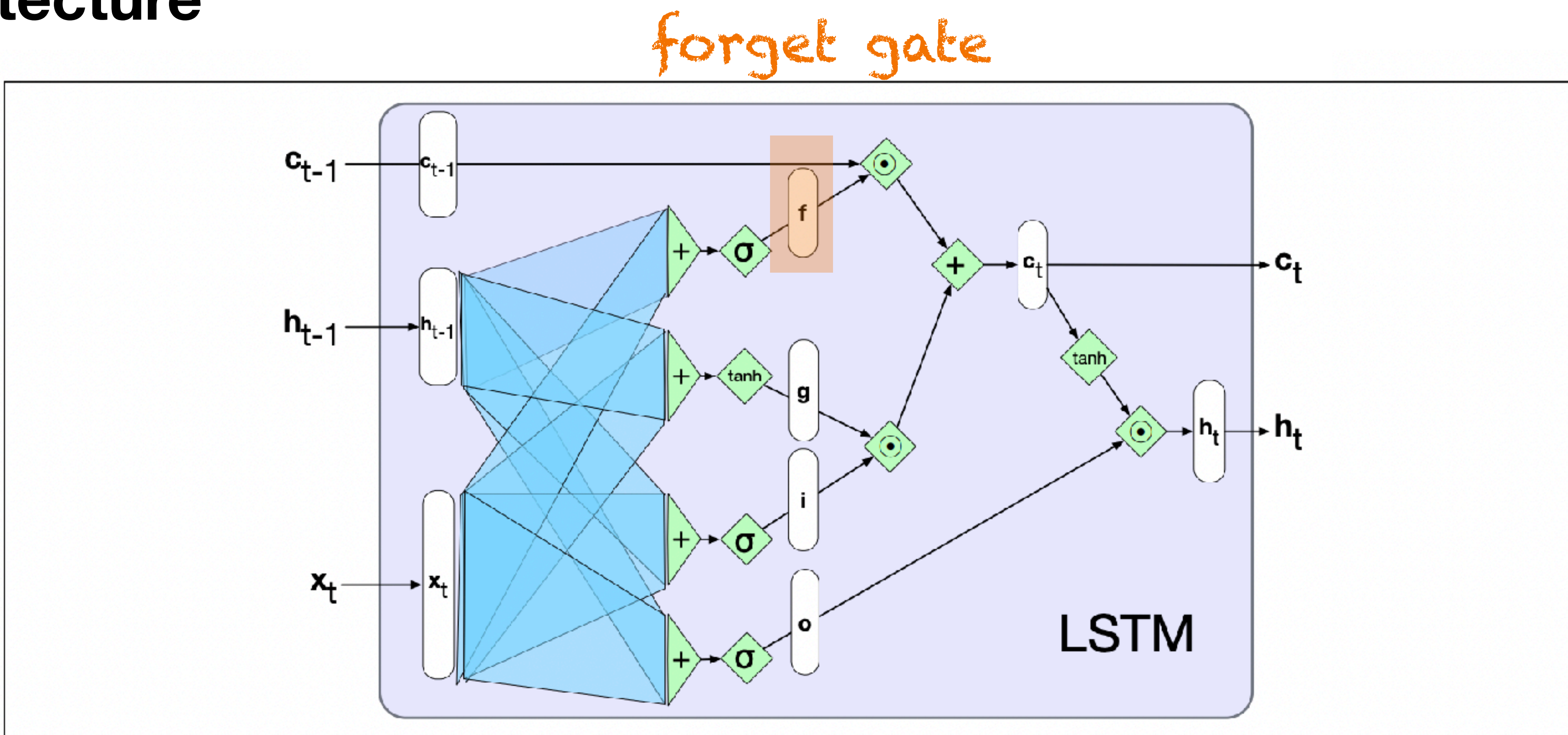


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

h_{t-1} and x also used to determine what to "forget"

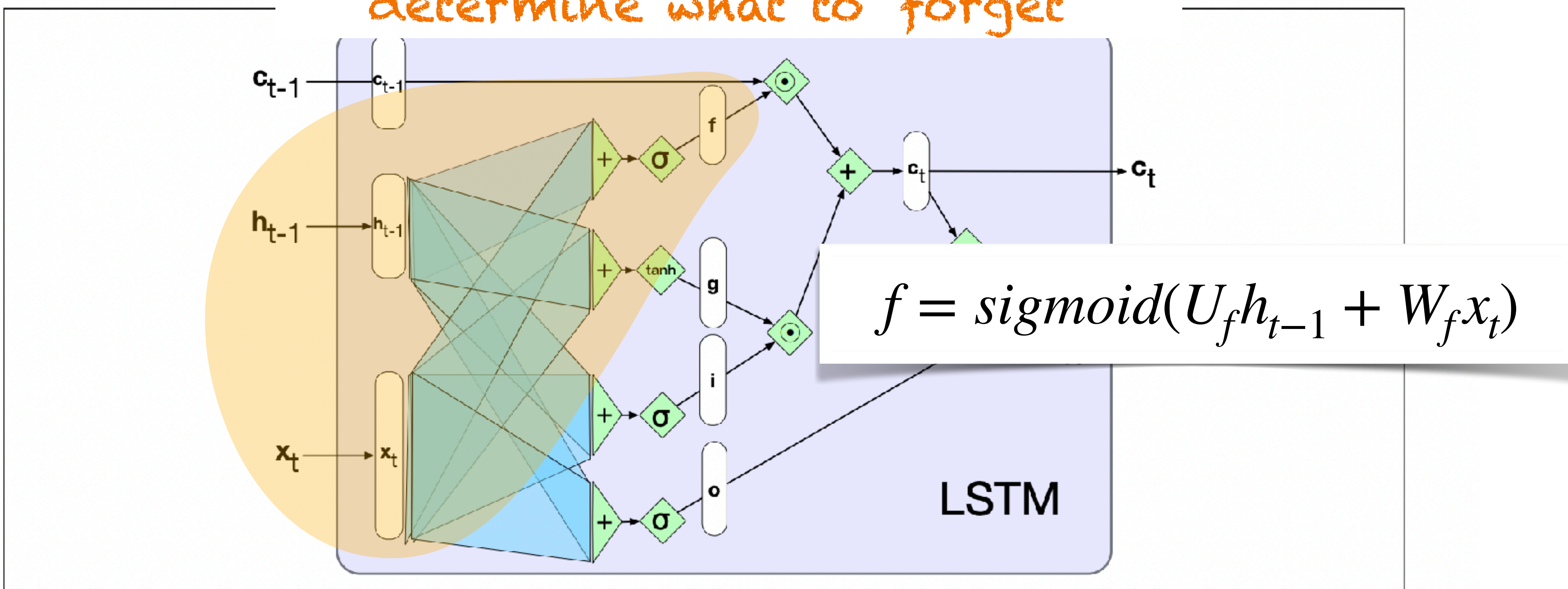


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

- “Gate” just means:
 - Learn some *mask* (i.e., vector)
 - Apply the mask to (i.e., elementwise multiplication aka Hadamard product) to some hidden state
- As always, mask is learned via backprop

3	7	2	9
---	---	---	---

⊙

0.1	0.8	0.9	0.2
-----	-----	-----	-----

=

0.3	5.6	1.8	1.8
-----	-----	-----	-----

hidden state

mask

result

Long-Short Term Memory Network (LSTM)

Architecture

f is used to "gate" the context

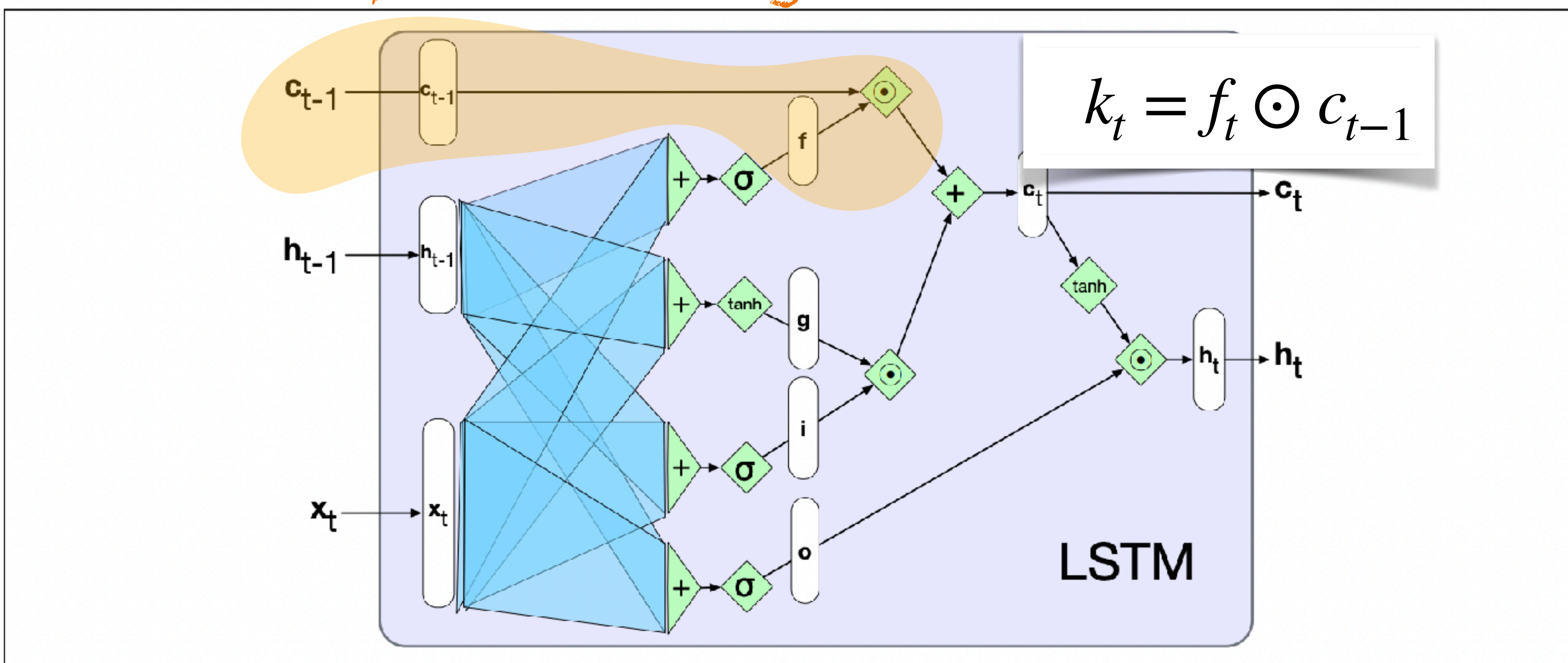


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

add gate

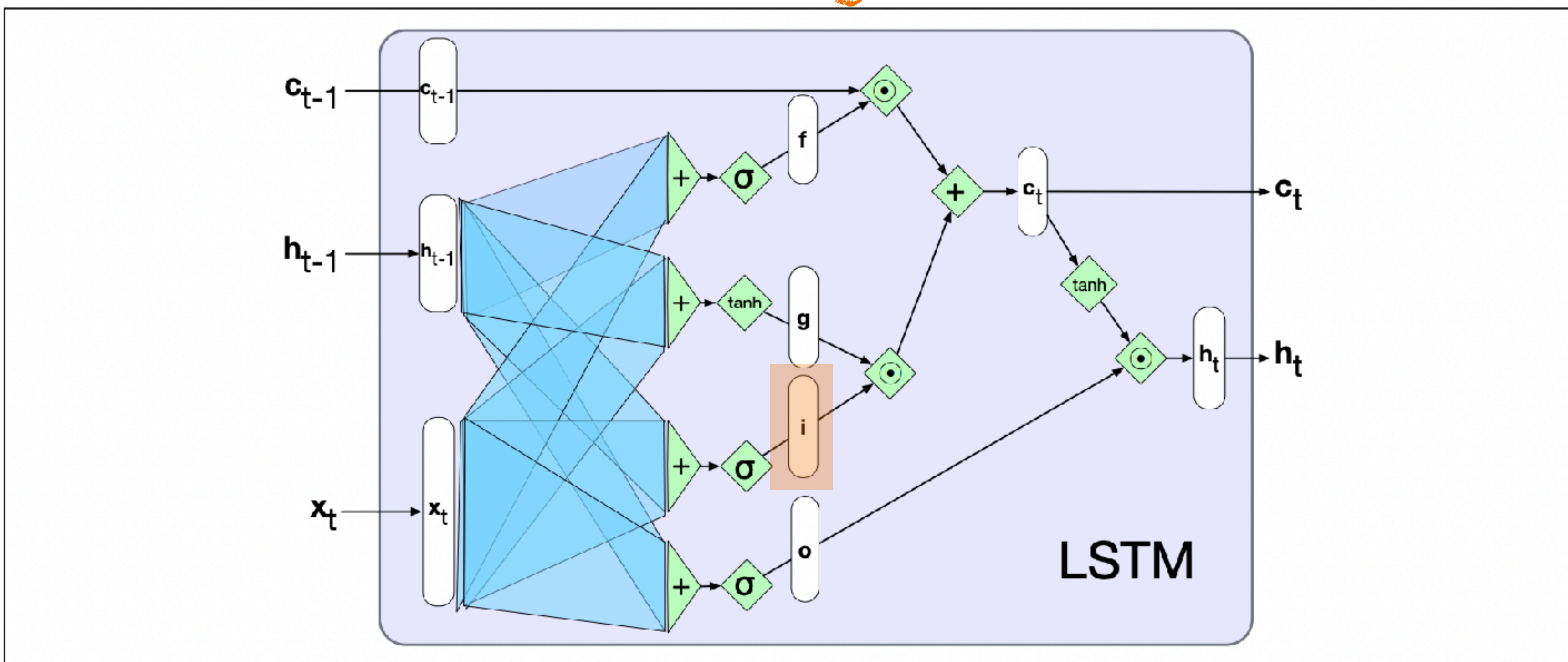


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

h_{t-1} and x also used to determine what to "add"

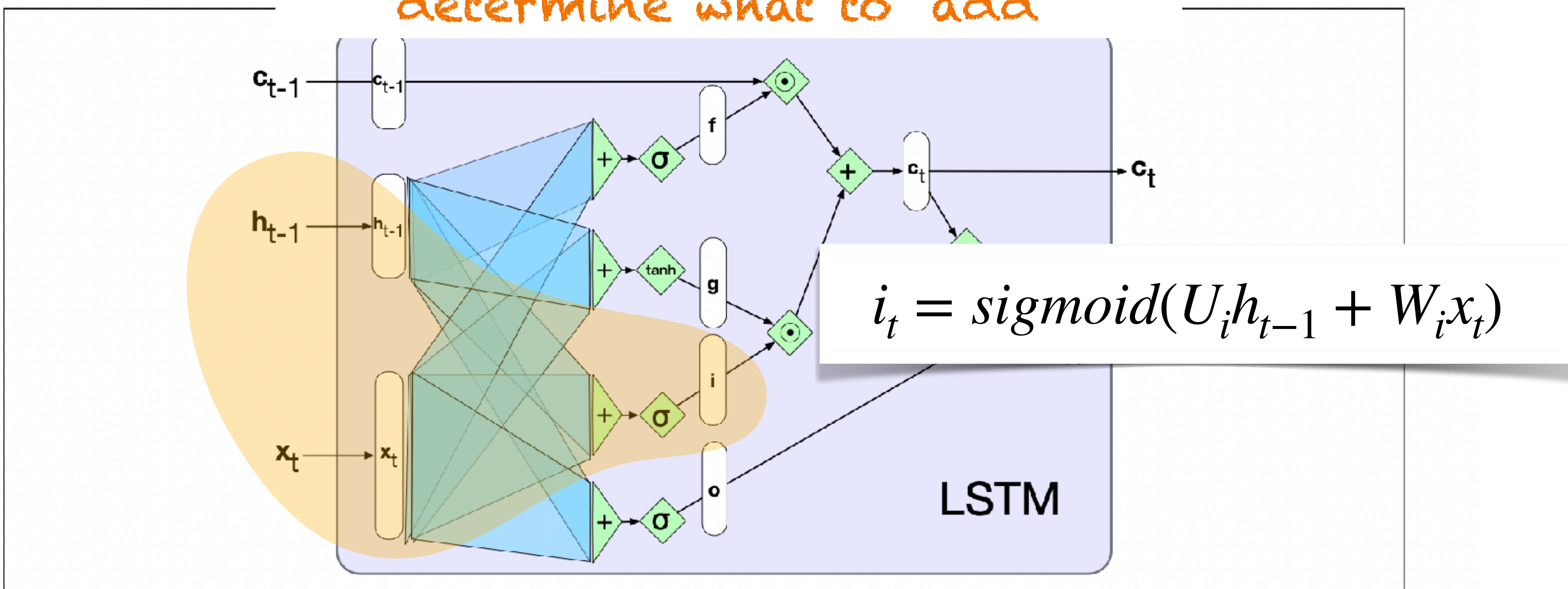


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

i is used to "gate" the current state

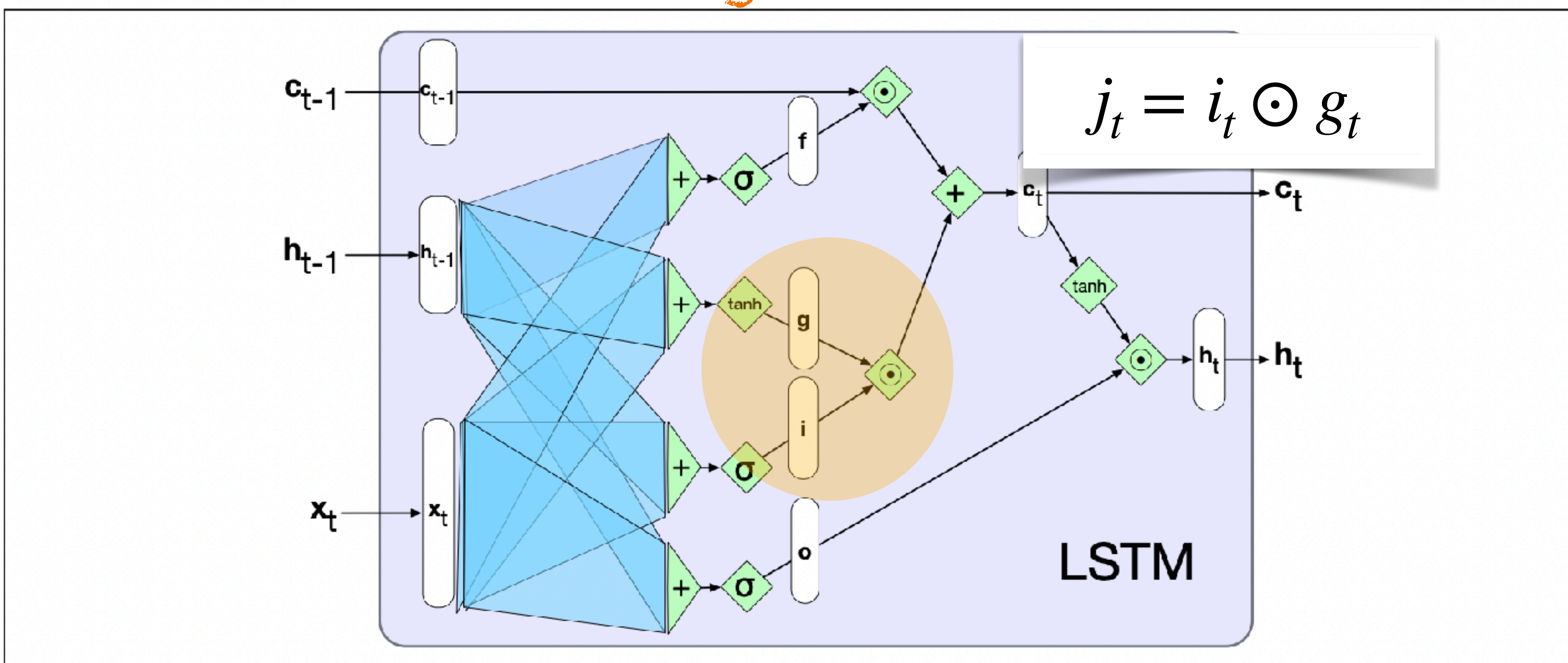


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

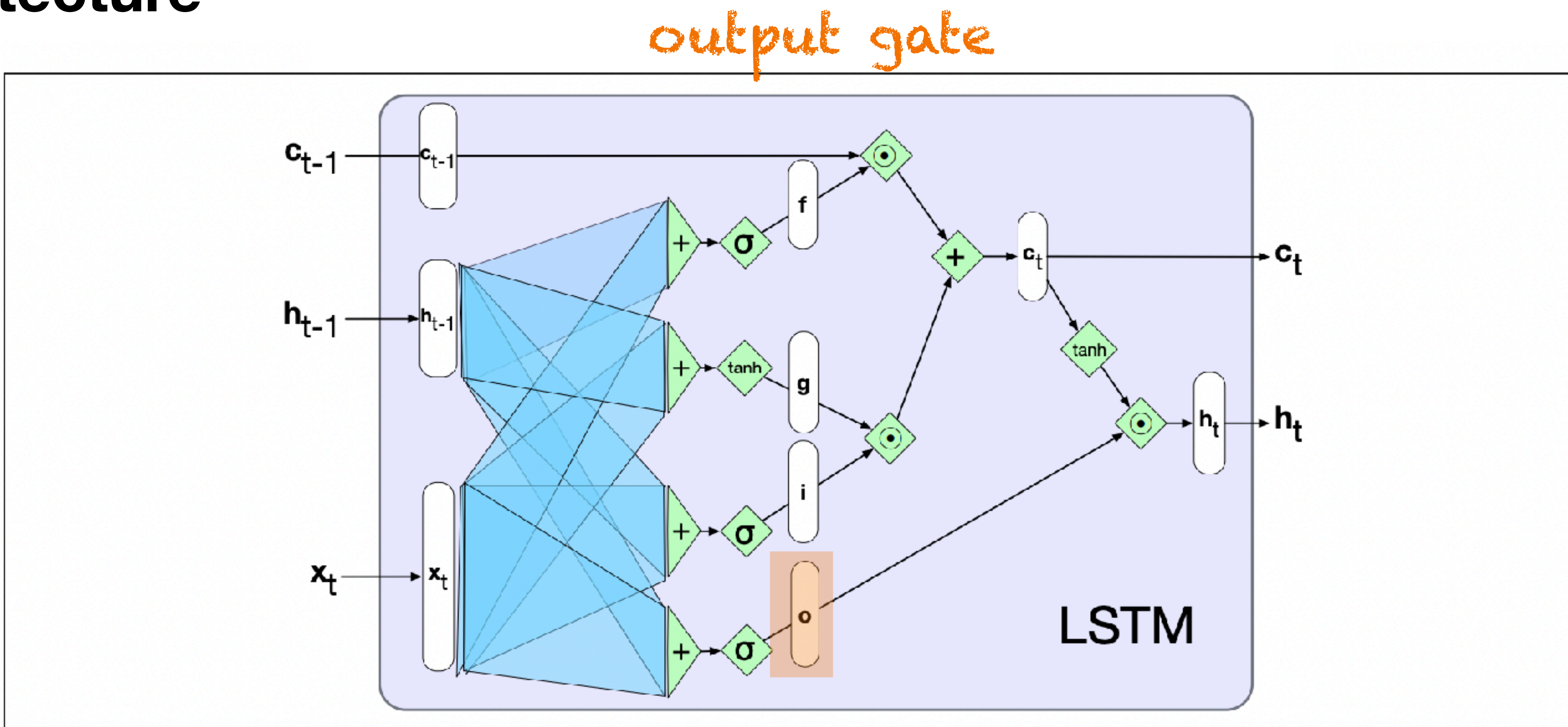


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

h_{t-1} and x also used to determine what to "add"

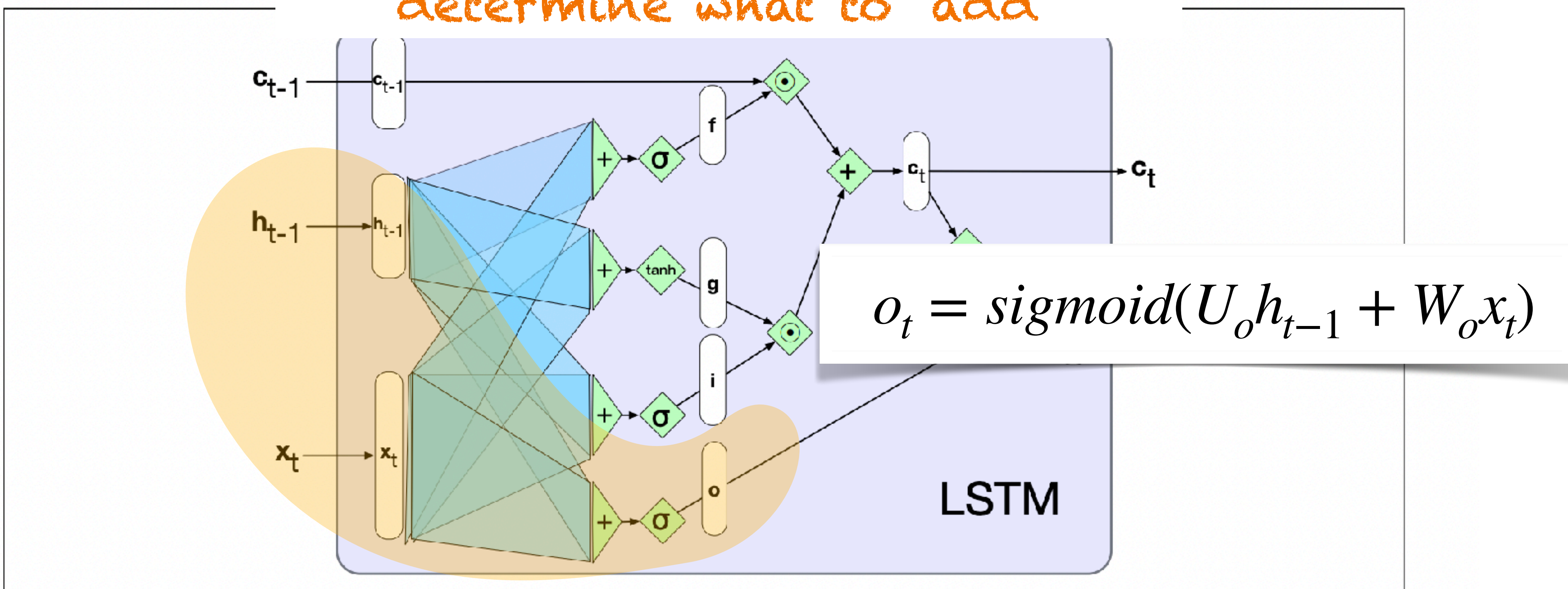
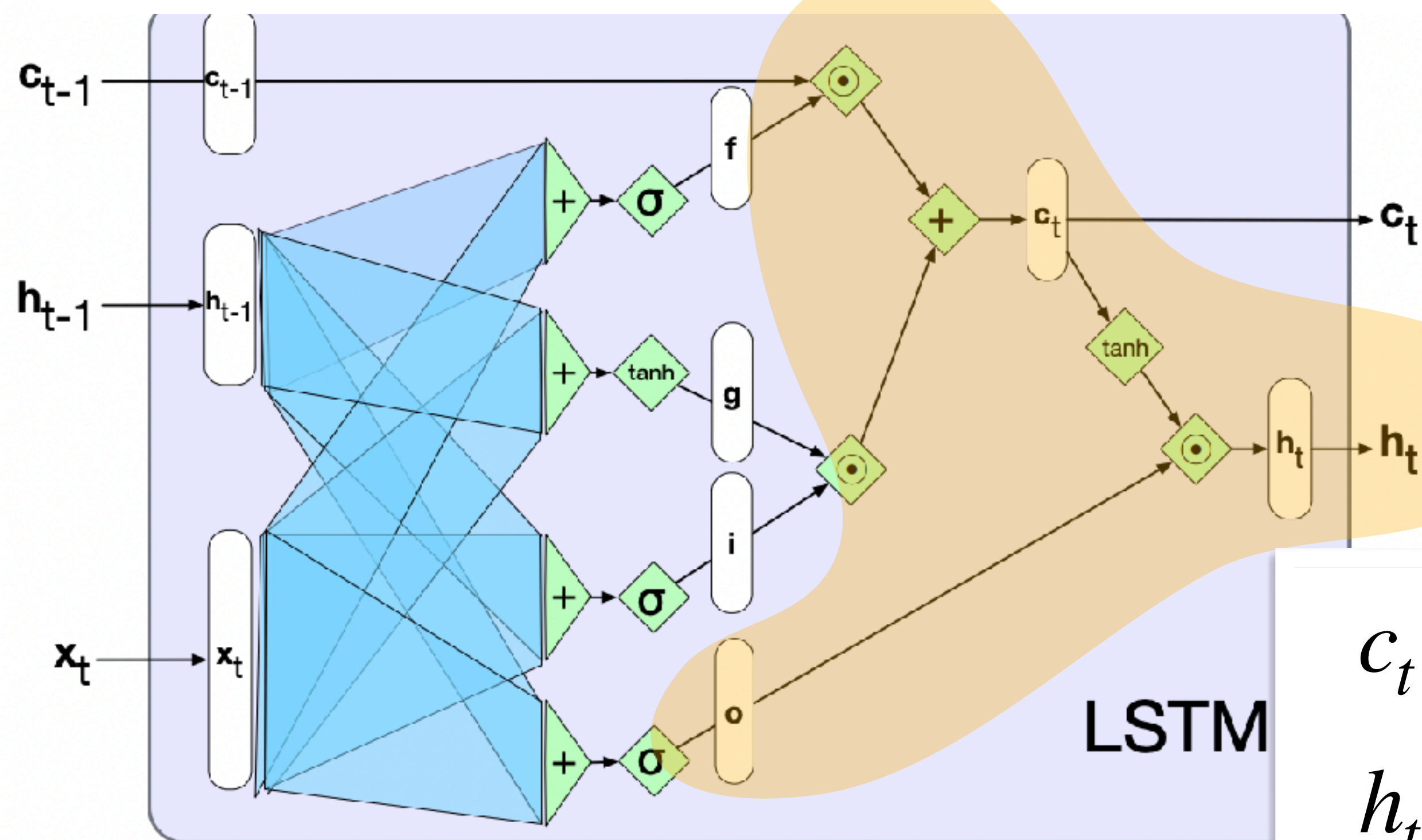


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

Architecture

h_{t-1} and x also used to determine what to "add"



$$c_t = j_t + k_t$$

$$h_t = o_t \odot \tanh(c_t)$$

Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

Long-Short Term Memory Network (LSTM)

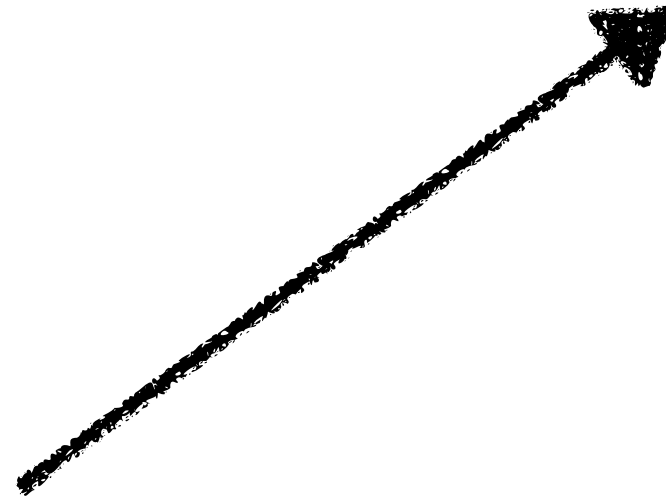
Architecture

$$\begin{aligned} g &= \tanh(U_g h_{t-1} + W_g x_t) \\ f &= \text{sigmoid}(U_f h_{t-1} + W_f x_t) \\ i_t &= \text{sigmoid}(U_i h_{t-1} + W_i x_t) \\ o_t &= \text{sigmoid}(U_o h_{t-1} + W_o x_t) \end{aligned}$$

$$\begin{aligned} k_t &= f_t \odot c_{t-1} \\ j_t &= i_t \odot g_t \\ c_t &= j_t + k_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Long-Short Term Memory Network (LSTM)

Architecture



compute current
state, add gate,
forget gate, and
output gate from
previous hidden state
and current input


$$\begin{aligned} g &= \tanh(U_g h_{t-1} + W_g x_t) \\ f &= \text{sigmoid}(U_f h_{t-1} + W_f x_t) \\ i_t &= \text{sigmoid}(U_i h_{t-1} + W_i x_t) \\ o_t &= \text{sigmoid}(U_o h_{t-1} + W_o x_t) \end{aligned}$$

$$\begin{aligned} k_t &= f_t \odot c_{t-1} \\ j_t &= i_t \odot g_t \\ c_t &= j_t + k_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Long-Short Term Memory Network (LSTM)

Architecture

combine those
things using
Hadamard
product



$$\begin{aligned} g &= \tanh(U_g h_{t-1} + W_g x_t) \\ f &= \text{sigmoid}(U_f h_{t-1} + W_f x_t) \\ i_t &= \text{sigmoid}(U_i h_{t-1} + W_i x_t) \\ o_t &= \text{sigmoid}(U_o h_{t-1} + W_o x_t) \end{aligned}$$

$$\begin{aligned} k_t &= f_t \odot c_{t-1} \\ j_t &= i_t \odot g_t \\ c_t &= j_t + k_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Long-Short Term Memory Network (LSTM)

Architecture

$$\begin{aligned} g &= \tanh(U_g h_{t-1} + W_g x_t) \\ f &= \text{sigmoid}(U_f h_{t-1} + W_f x_t) \\ i_t &= \text{sigmoid}(U_i h_{t-1} + W_i x_t) \\ o_t &= \text{sigmoid}(U_o h_{t-1} + W_o x_t) \end{aligned}$$

update context
and hidden
state for next
iteration

$$\begin{aligned} k_t &= f_t \odot c_{t-1} \\ j_t &= i_t \odot g_t \\ c_t &= j_t + k_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Long-Short Term Memory Network (LSTM)

Architecture

$$\begin{aligned} g &= \tanh(U_g h_{t-1} + W_g x_t) \\ f &= \text{sigmoid}(U_f h_{t-1} + W_f x_t) \\ i_t &= \text{sigmoid}(U_i h_{t-1} + W_i x_t) \\ o_t &= \text{sigmoid}(U_o h_{t-1} + W_o x_t) \end{aligned}$$

$$k_t = f_t \odot c_{t-1}$$

$$j_t = i_t \odot g_t$$

$$c_t = j_t + k_t$$

$$h_t = o_t \odot \tanh(c_t)$$



Long Short Term Memory Network / LSTM

g = current state (like h in the rnn)

f = forget gate

i = add gate

o = output gate

k = intermediate output which is the gated context (context after "forgetting" what f says to forget)

j = intermediate output which is the gated version of g , the stuff from g that needs to be added to the context

c = updated context

h = updated hidden state

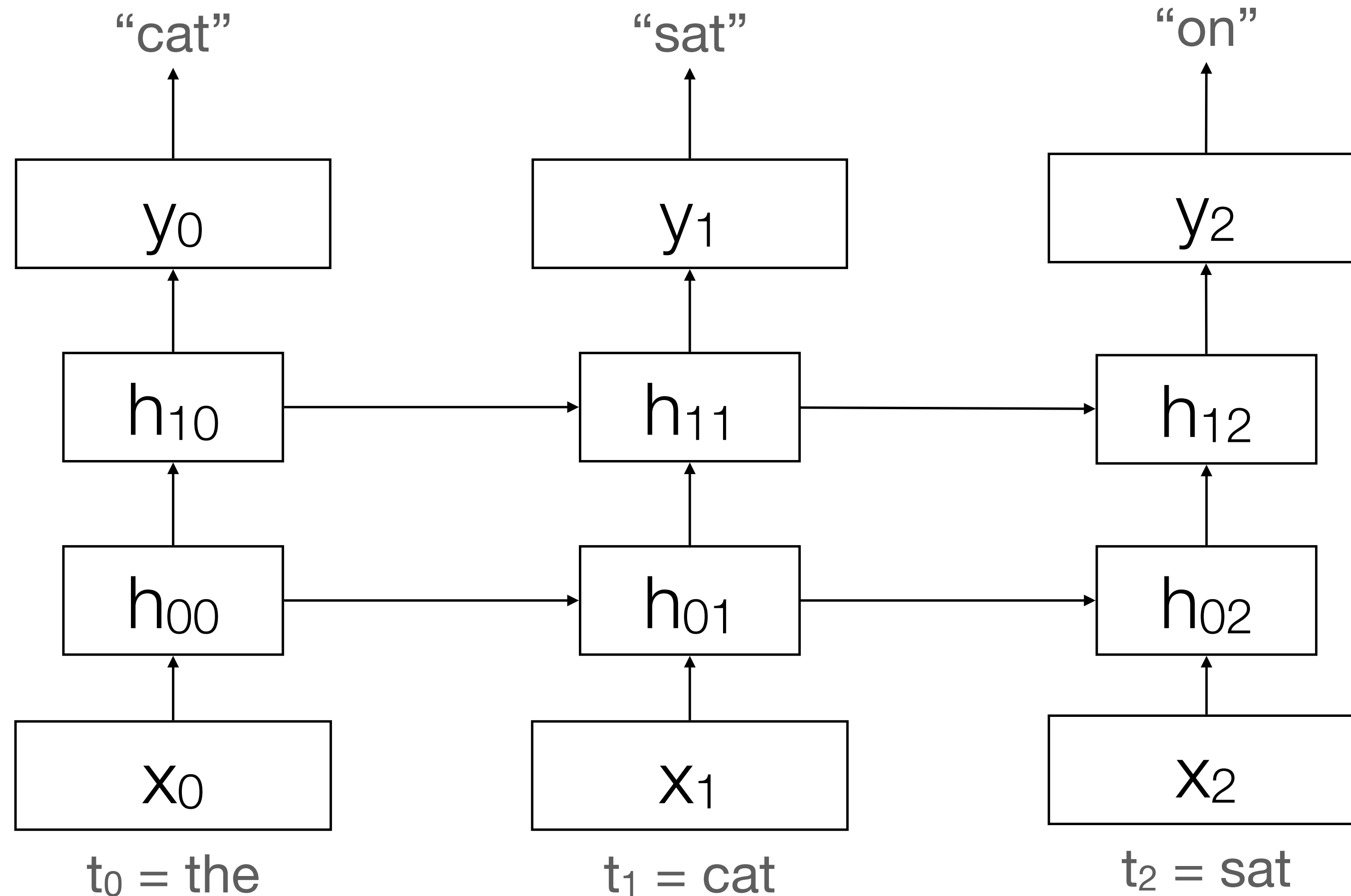
$$h_t = c_t \odot \tanh(c_t)$$

Topics

- NN Architectures for Language Modeling
 - ~~MLP~~
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformer**

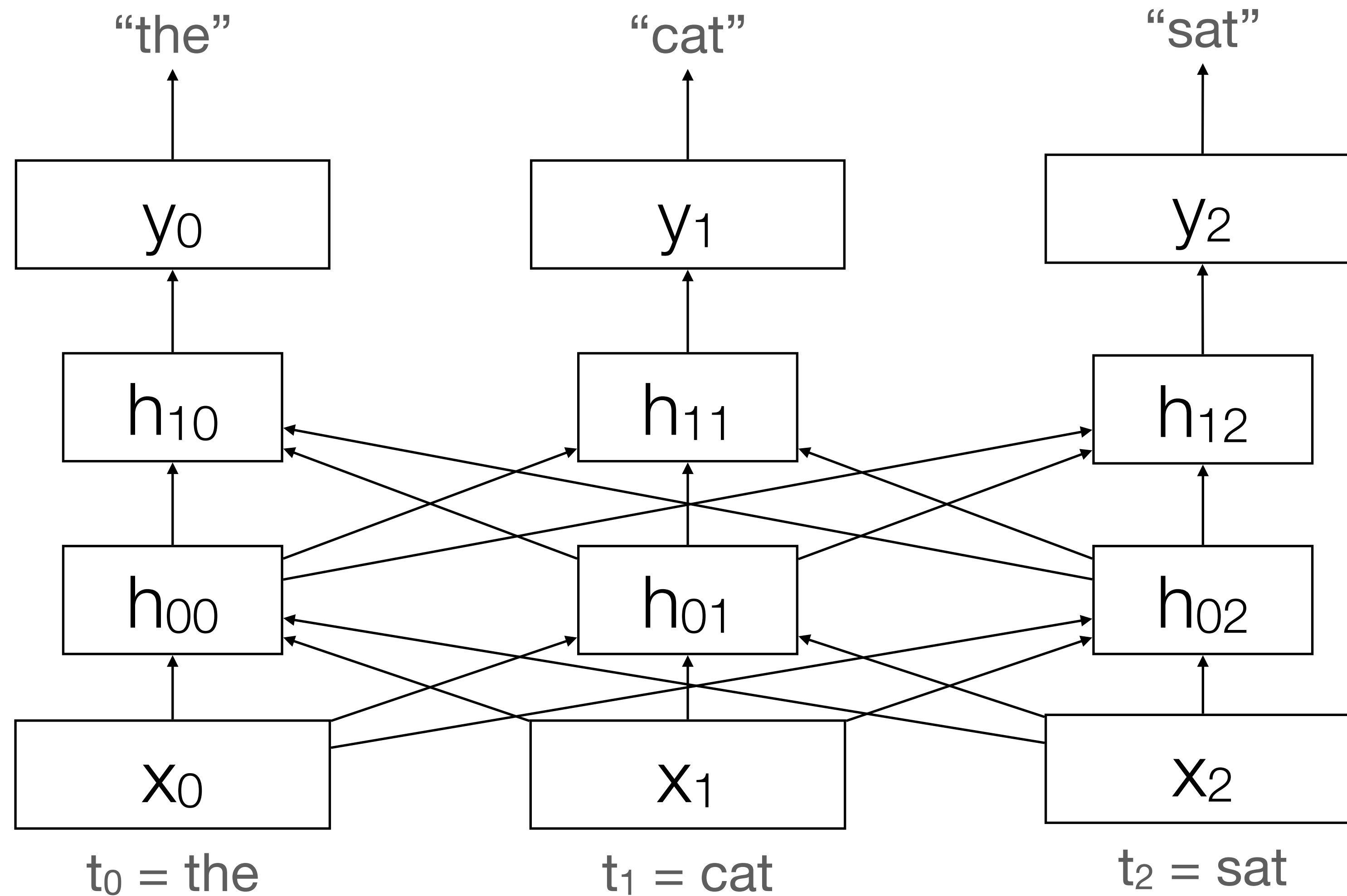
Transformers

Recap: Recurrent Neural Network (RNN)



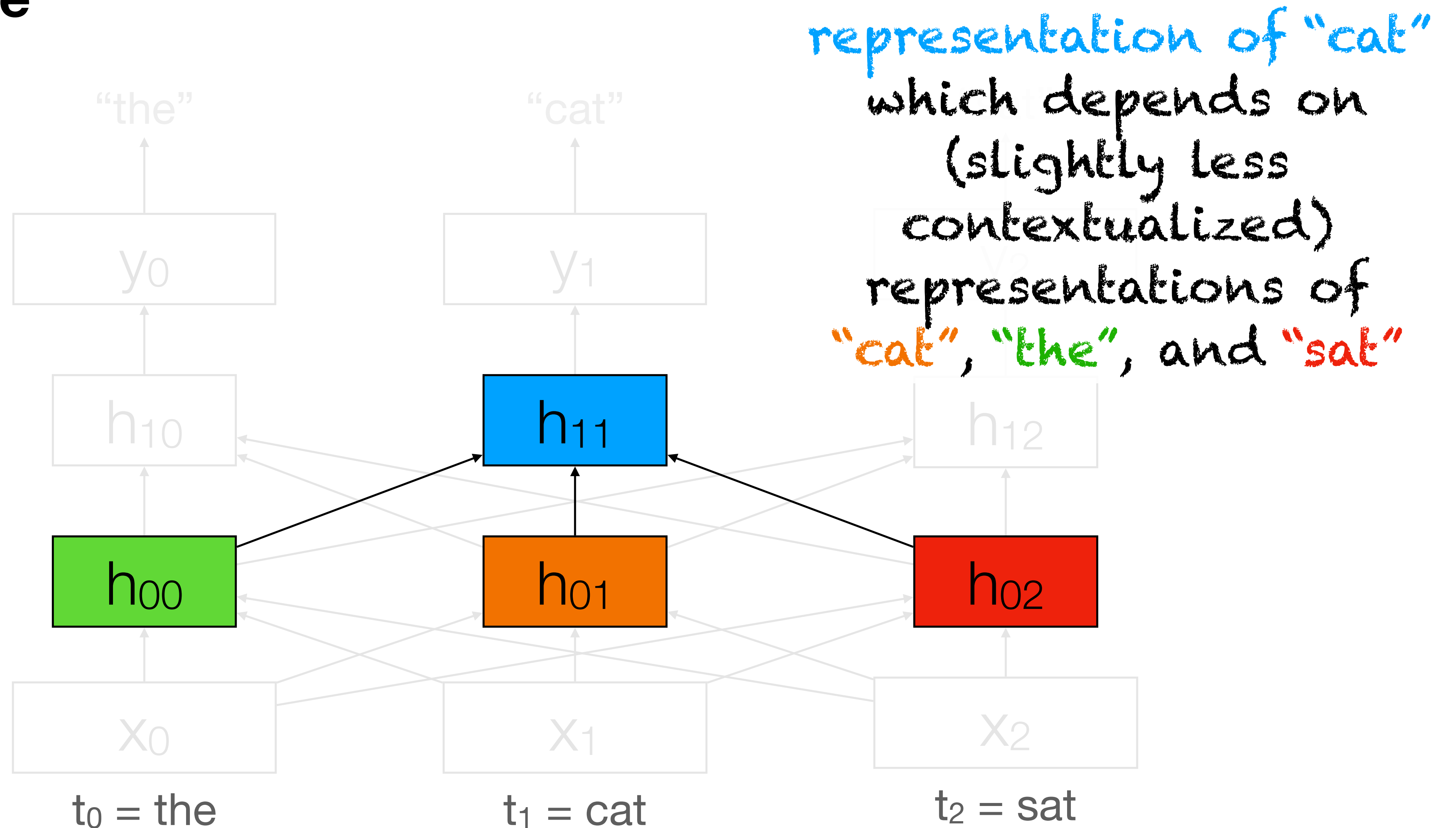
Transformers

Architecture



Transformers

Architecture



Topics

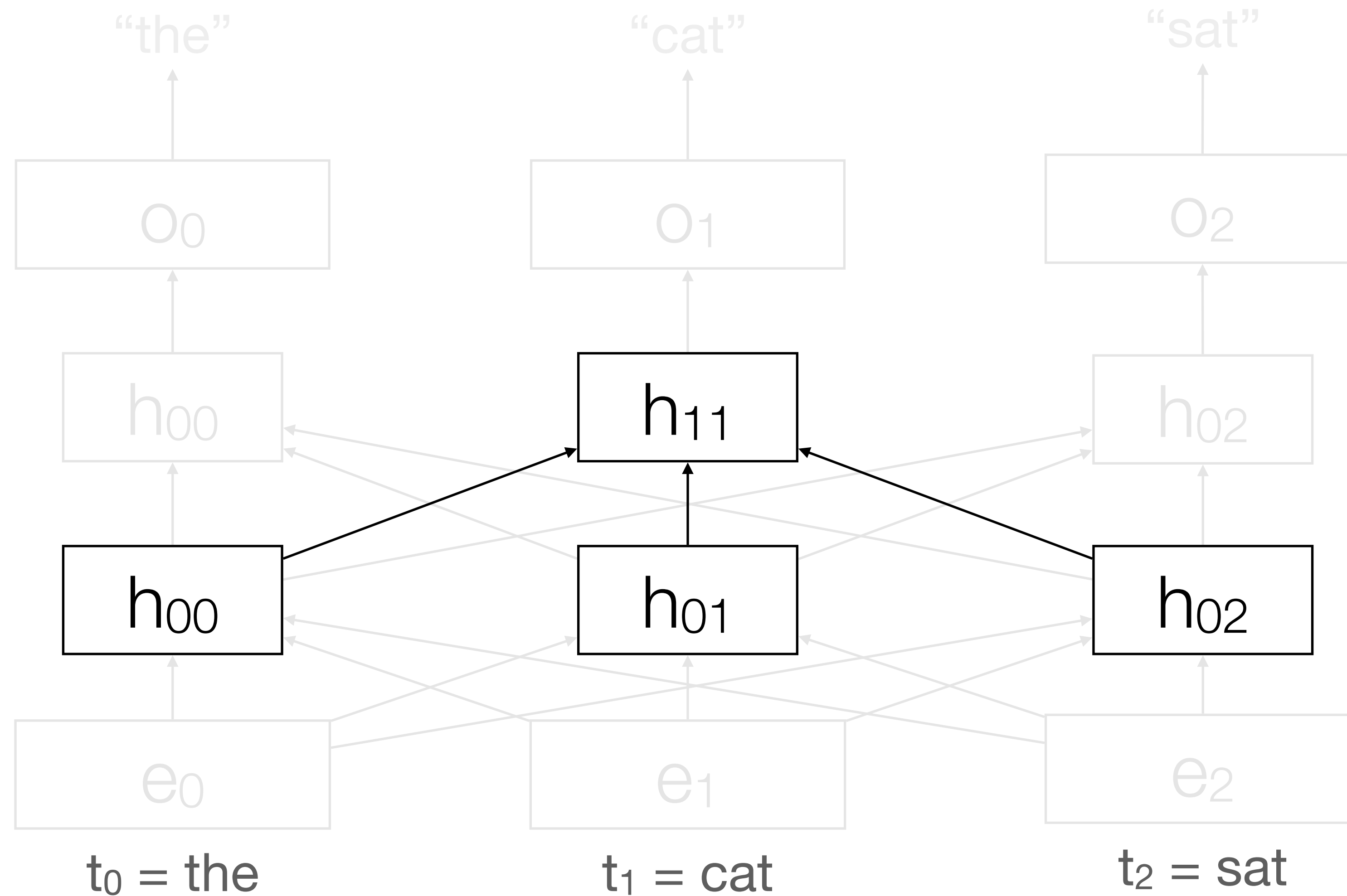
- NN Architectures for Language Modeling
 - ~~MLP~~
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformers**
 - Architecture
 - Self Attention
 - Blocks
 - Positional Encodings

Topics

- NN Architectures for Language Modeling
 - ~~MLP~~
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformers**
 - Architecture
 - **Self Attention**
 - Blocks
 - Positional Encodings

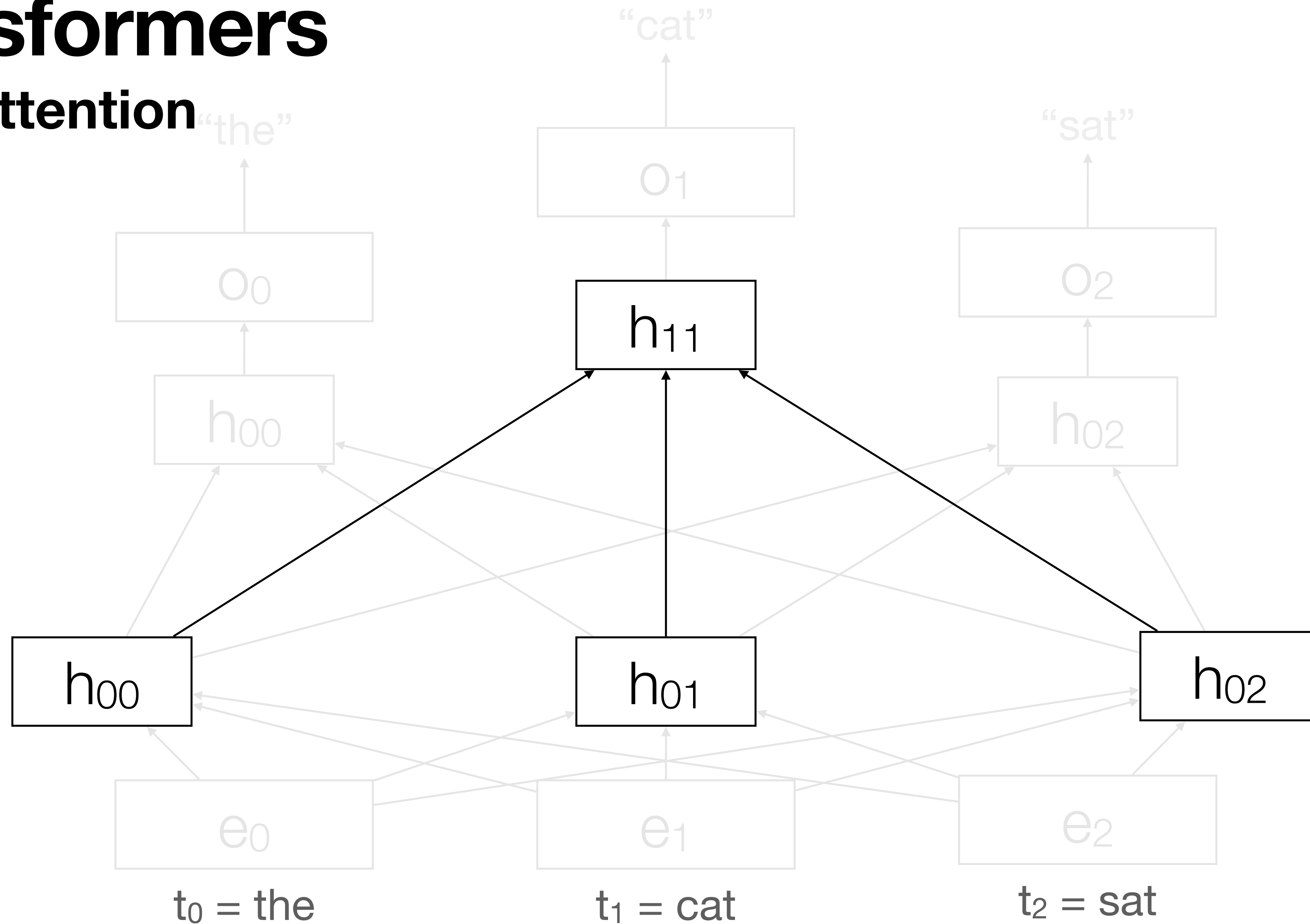
Transformers

(Self) Attention



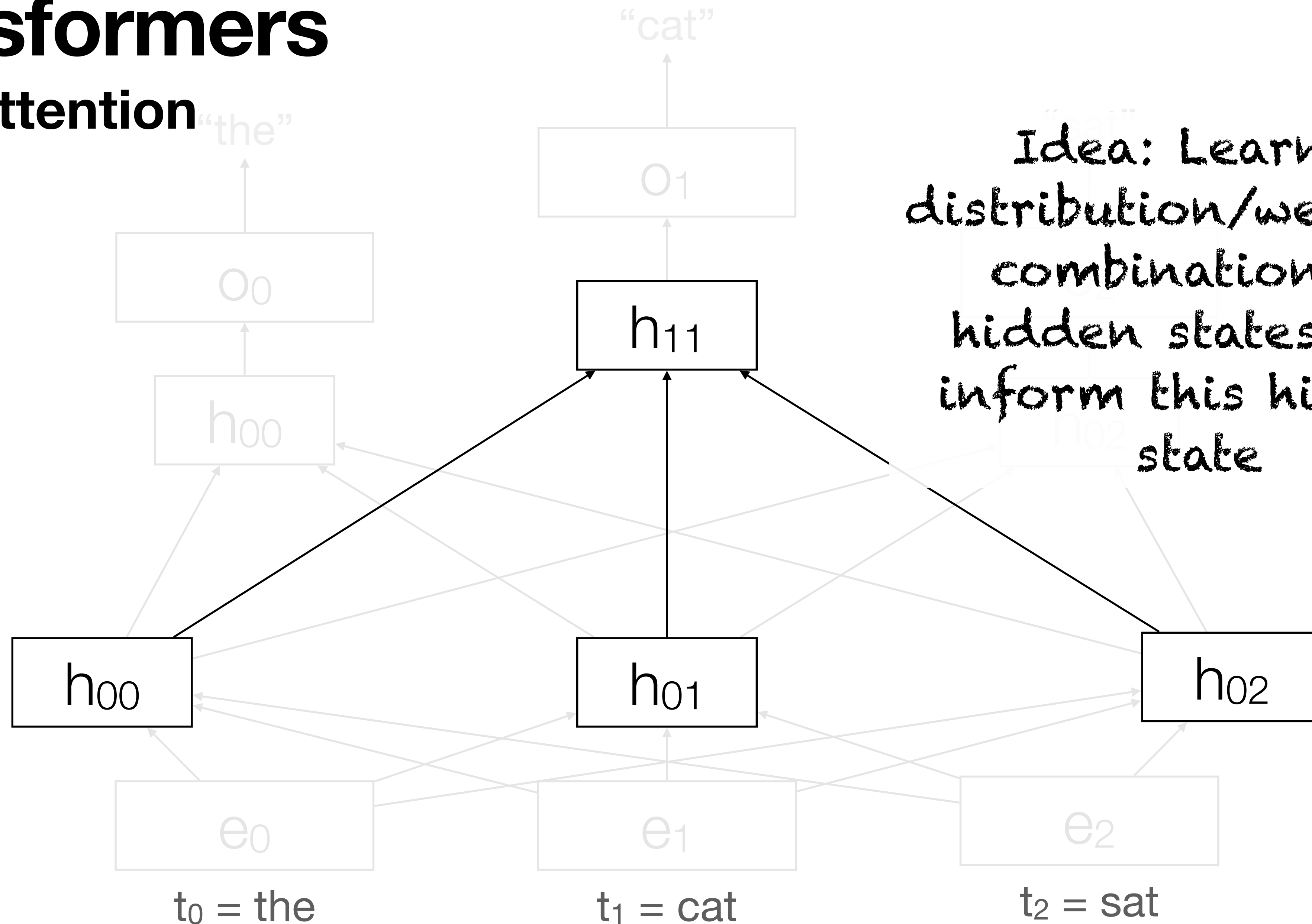
Transformers

(Self) Attention



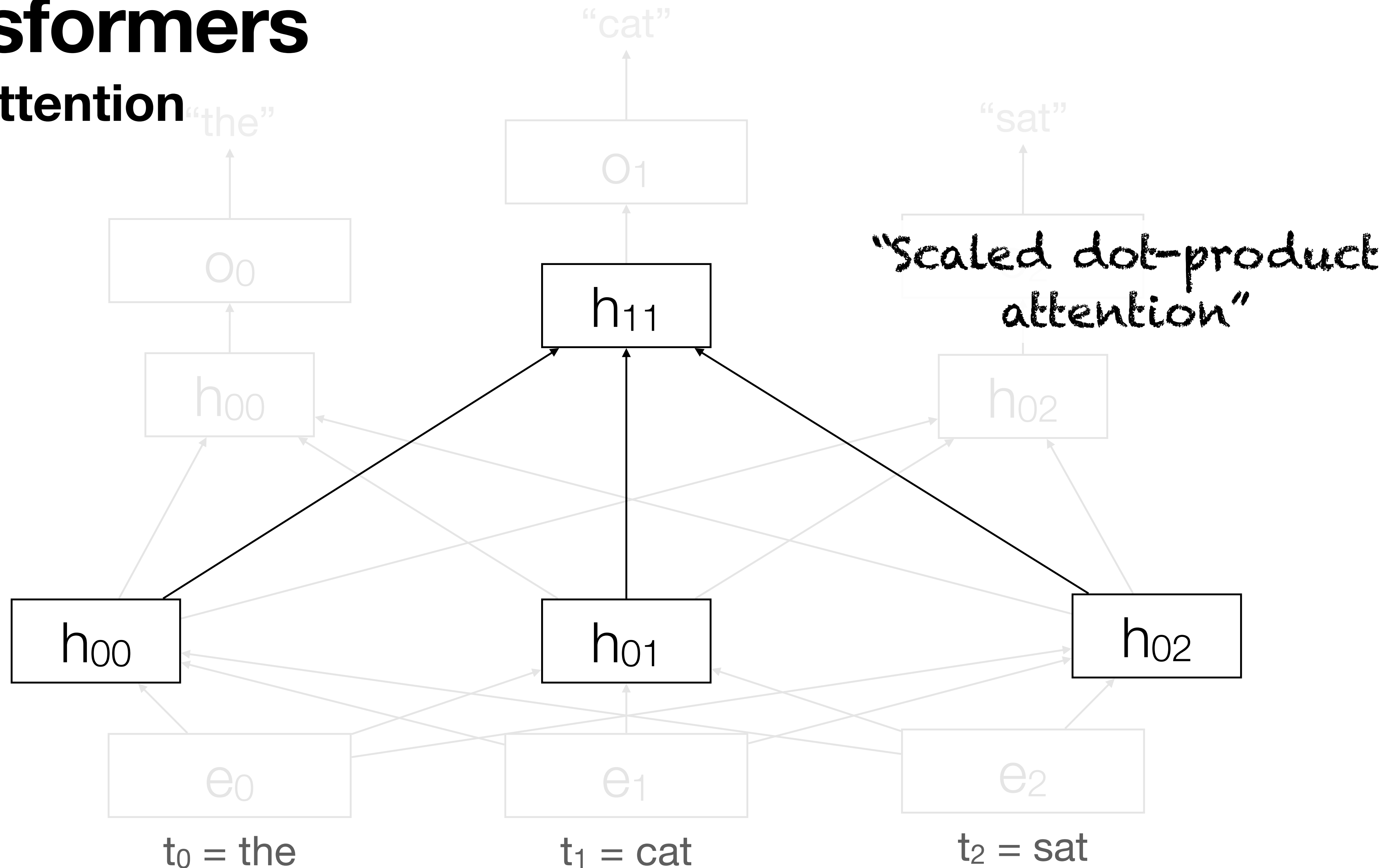
Transformers

(Self) Attention



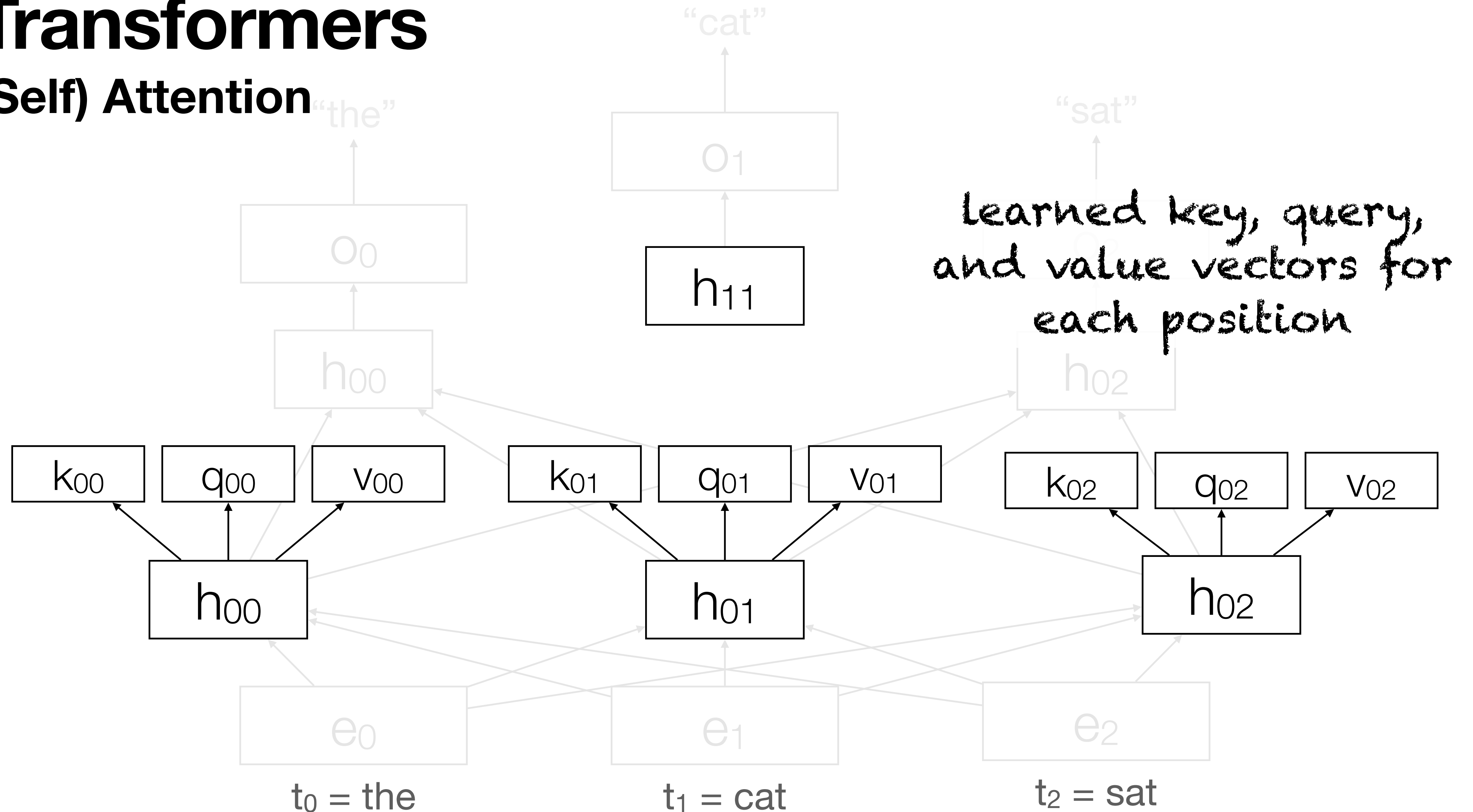
Transformers

(Self) Attention



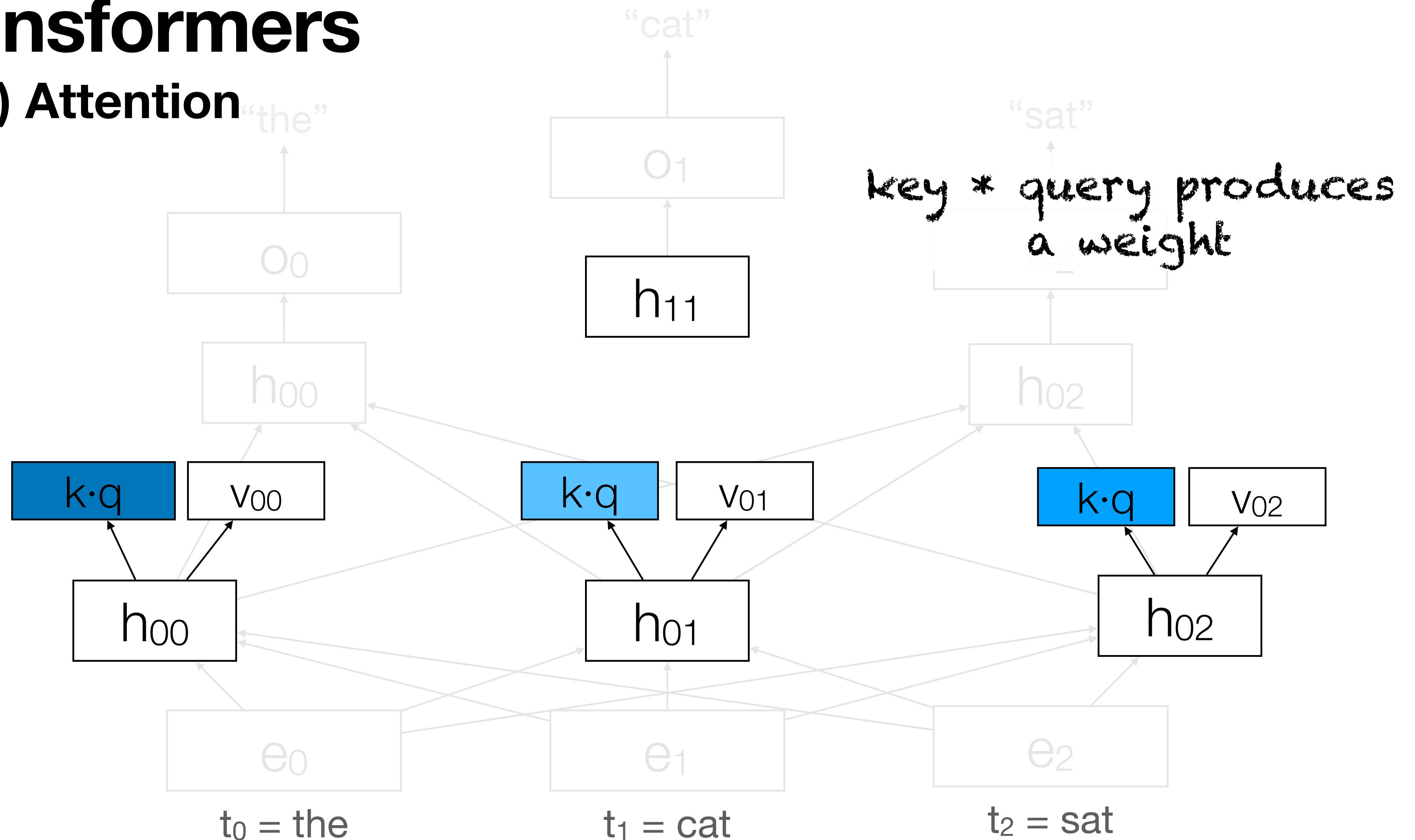
Transformers

(Self) Attention



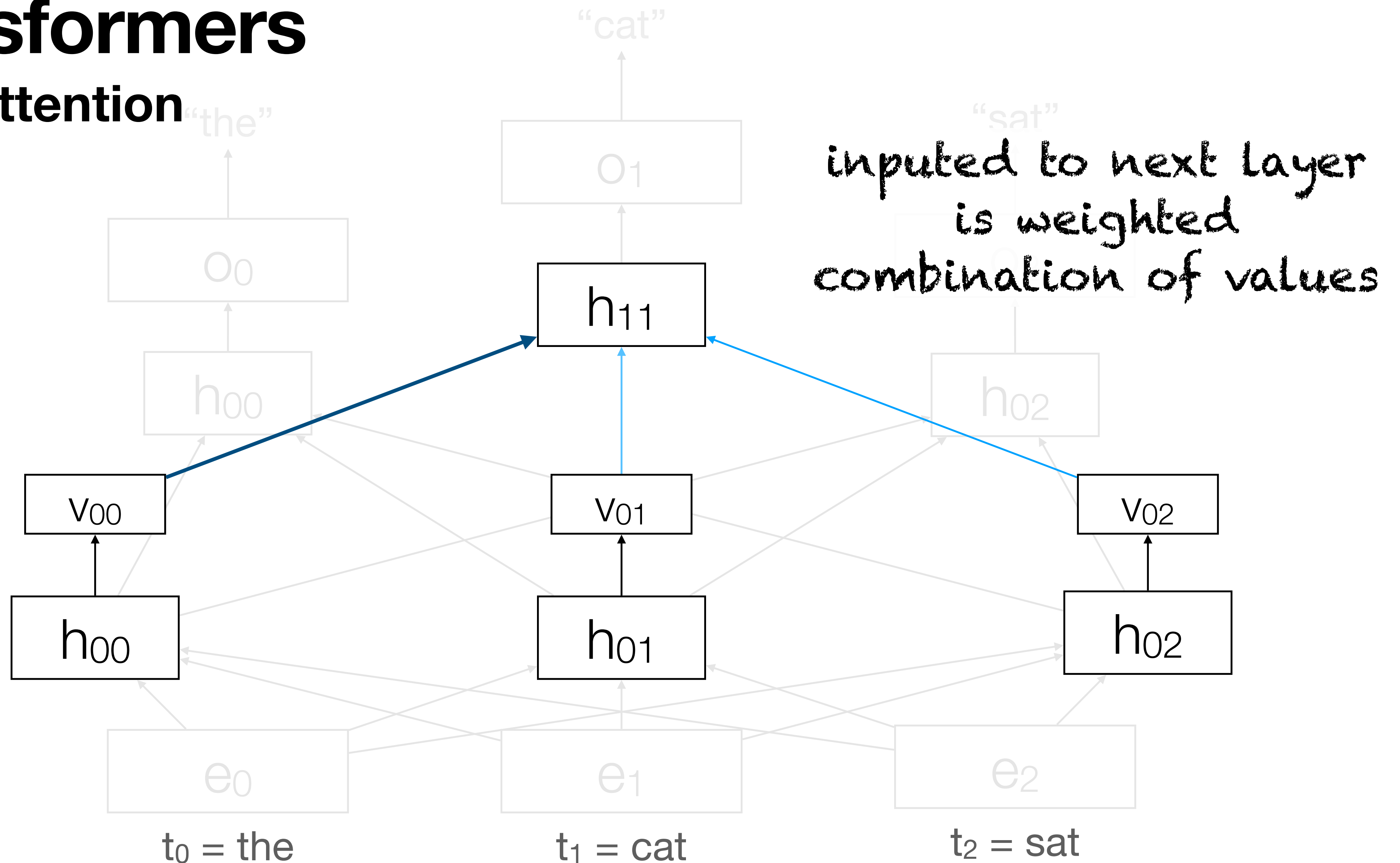
Transformers

(Self) Attention



Transformers

(Self) Attention



Transformers

(Self) Attention

- Each word has three roles:
 - Query: The word as the current focus. I.e., attention is trying to determine how to process this word.
 - Key: The word as a context word. I.e., attention is determining how to use this word to inform the query.
 - Value: The word as part of the output. I.e., attention is determining how to use this word, based on the key-query, to produce an output.
- Every word acts in all three roles at each timestep.
- We learn three weight matrices (Q,K,V) to cast each word into each role

Transformers

(Self) Attention

- Each word has three roles:
 - Query: The word as the current focus. I.e., attention is trying to determine how to process this word.
 - Key: The word as a context word. I.e., attention is determining how to use this word to inform the query.
 - Value: The word as part of the output. I.e., attention is determining how to use this word, based on the key-query, to produce an output.
- Every word acts in all three roles at each timestep.
- We learn three weight matrices (Q,K,V) to cast each word into each role

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

Transformers

(Self) Attention

- To actually compute the attention:
 - $\text{score} = \text{dot}(k, q)$
 - score is just a scalar number
- $y = \text{weighted_sum of values} = \text{sum}(\text{score} * v)$

Transformers

(Self) Attention

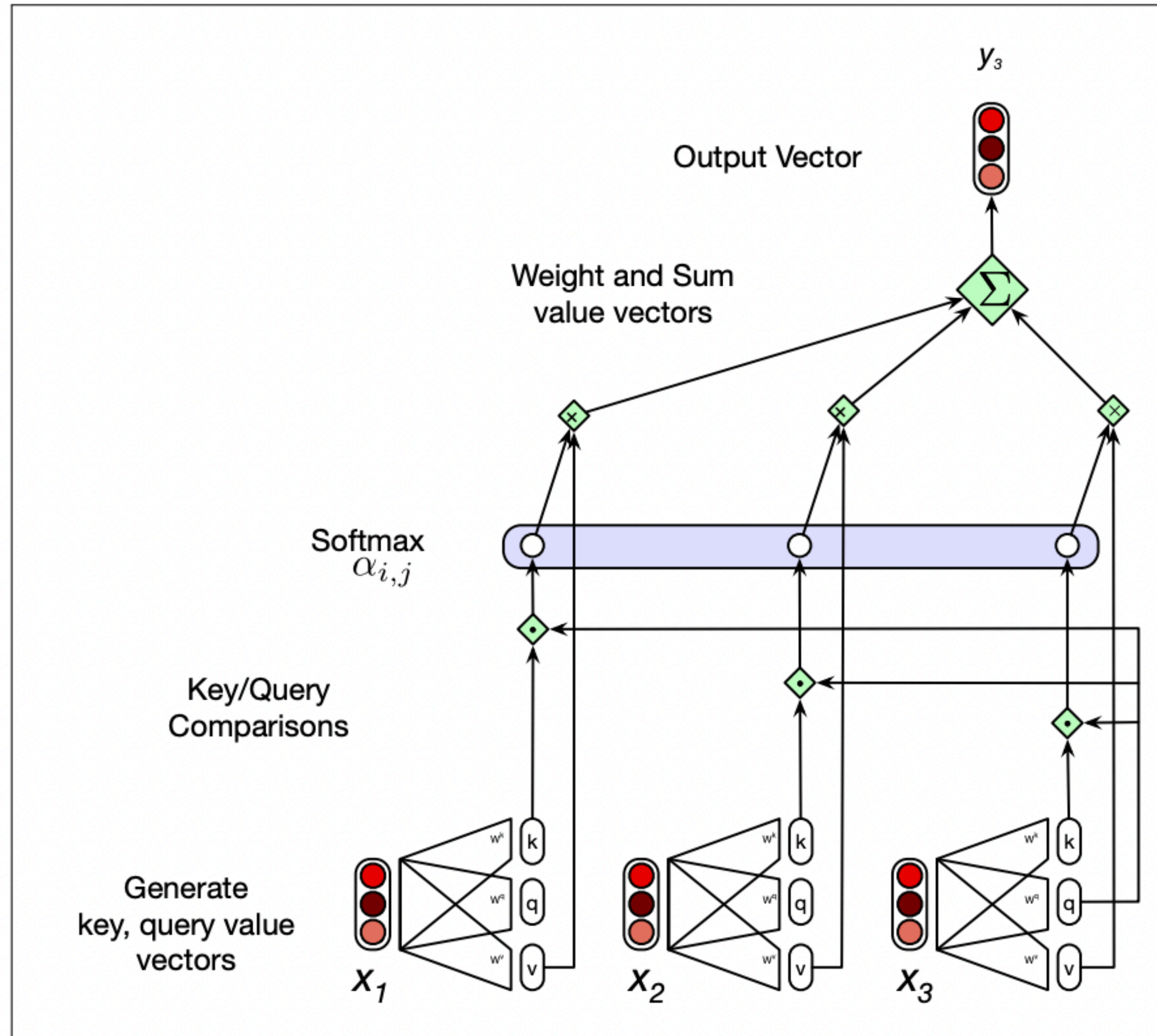


Figure 9.16 Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention.

Transformers

(Self) Attention

$\text{dot}(q, k)$

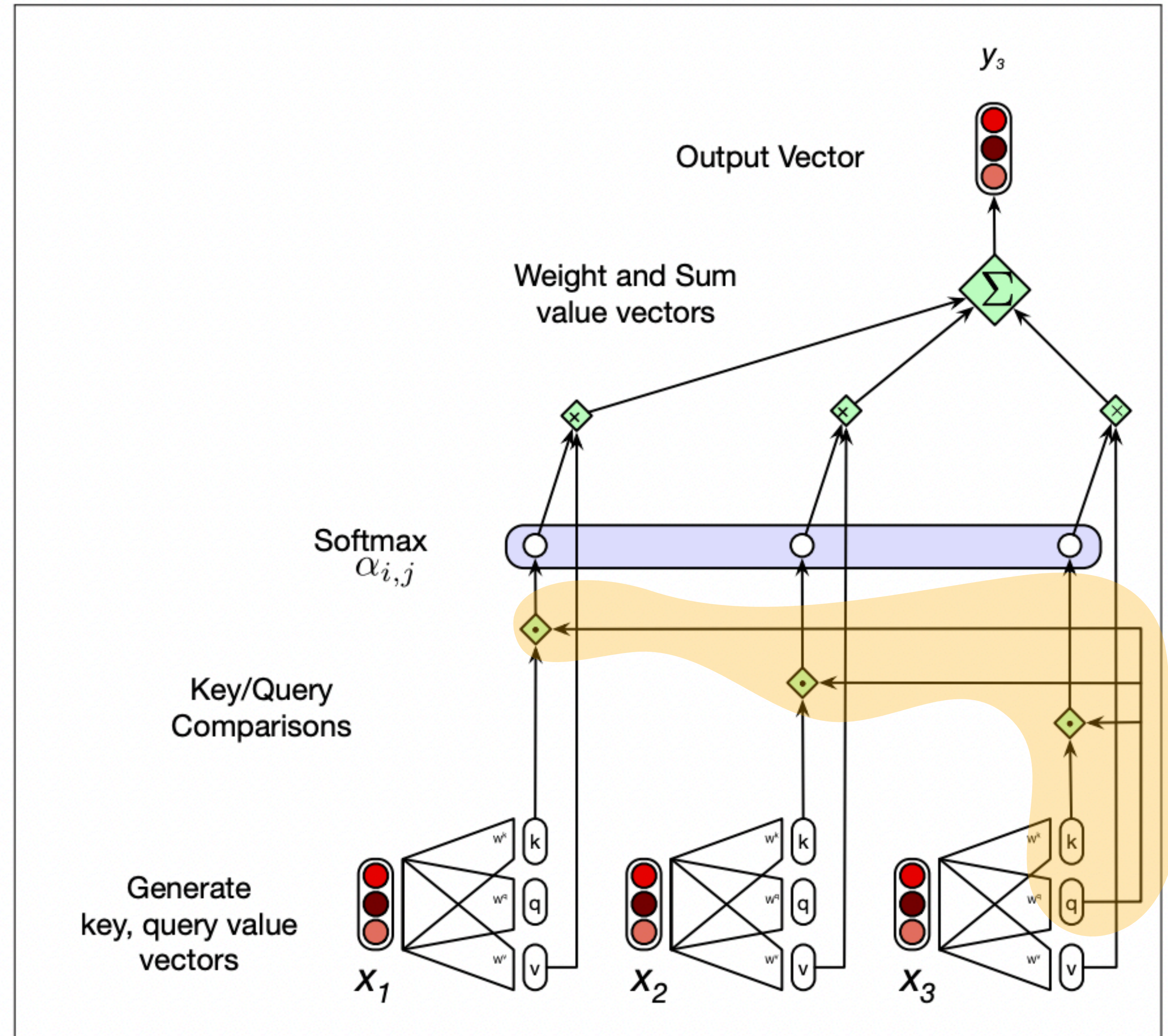


Figure 9.16 Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention.

Transformers

(Self) Attention

score*value

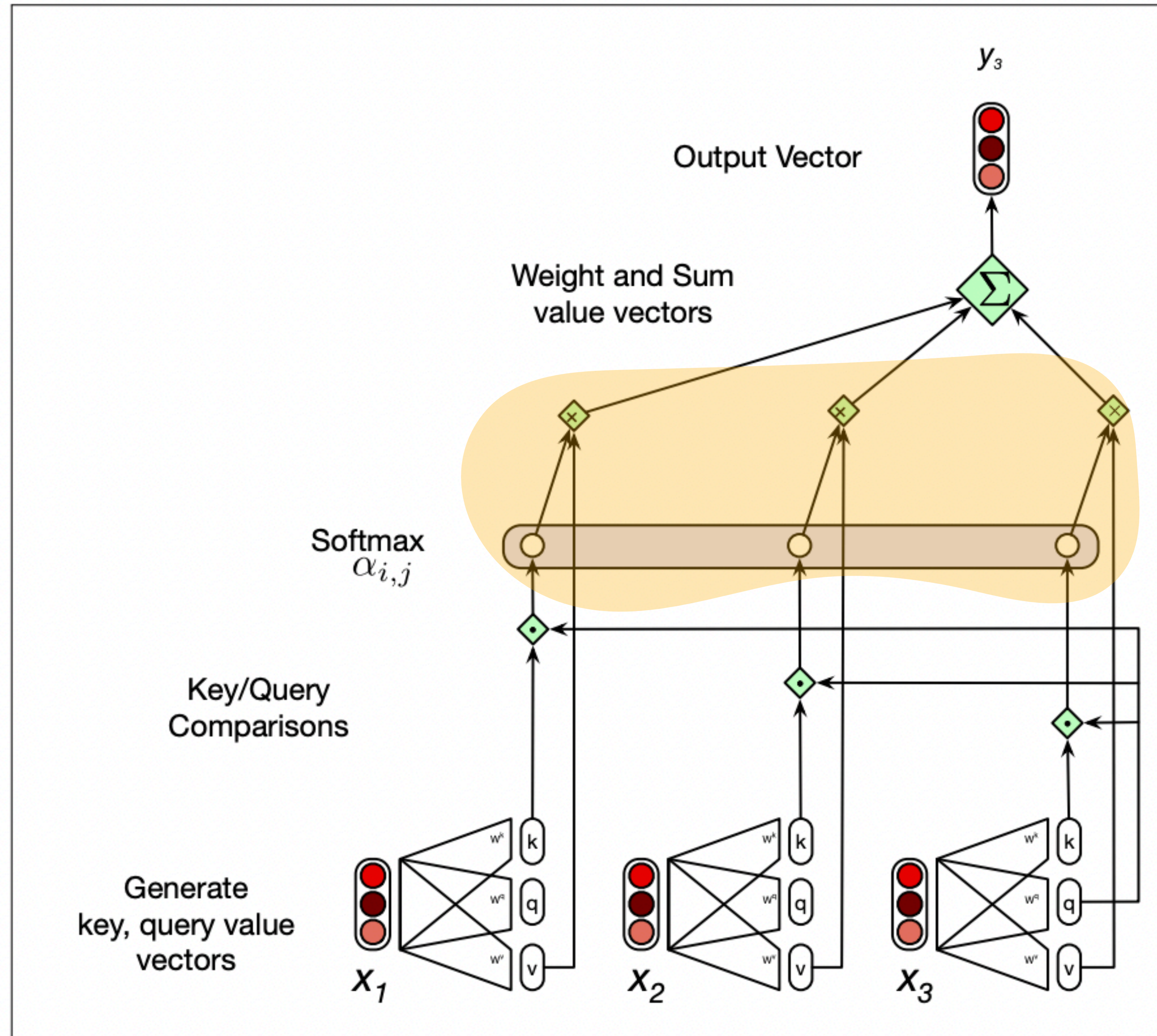
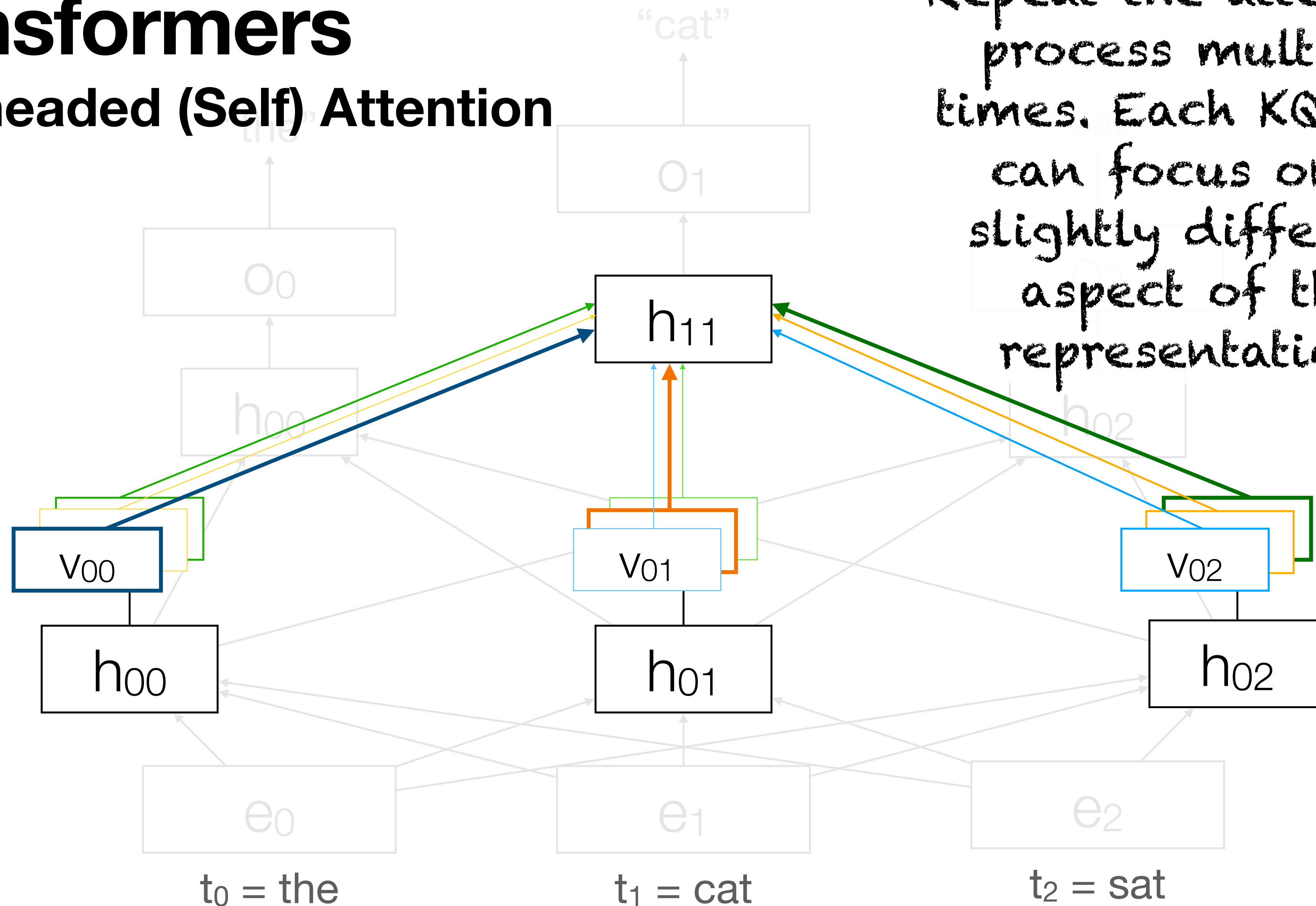


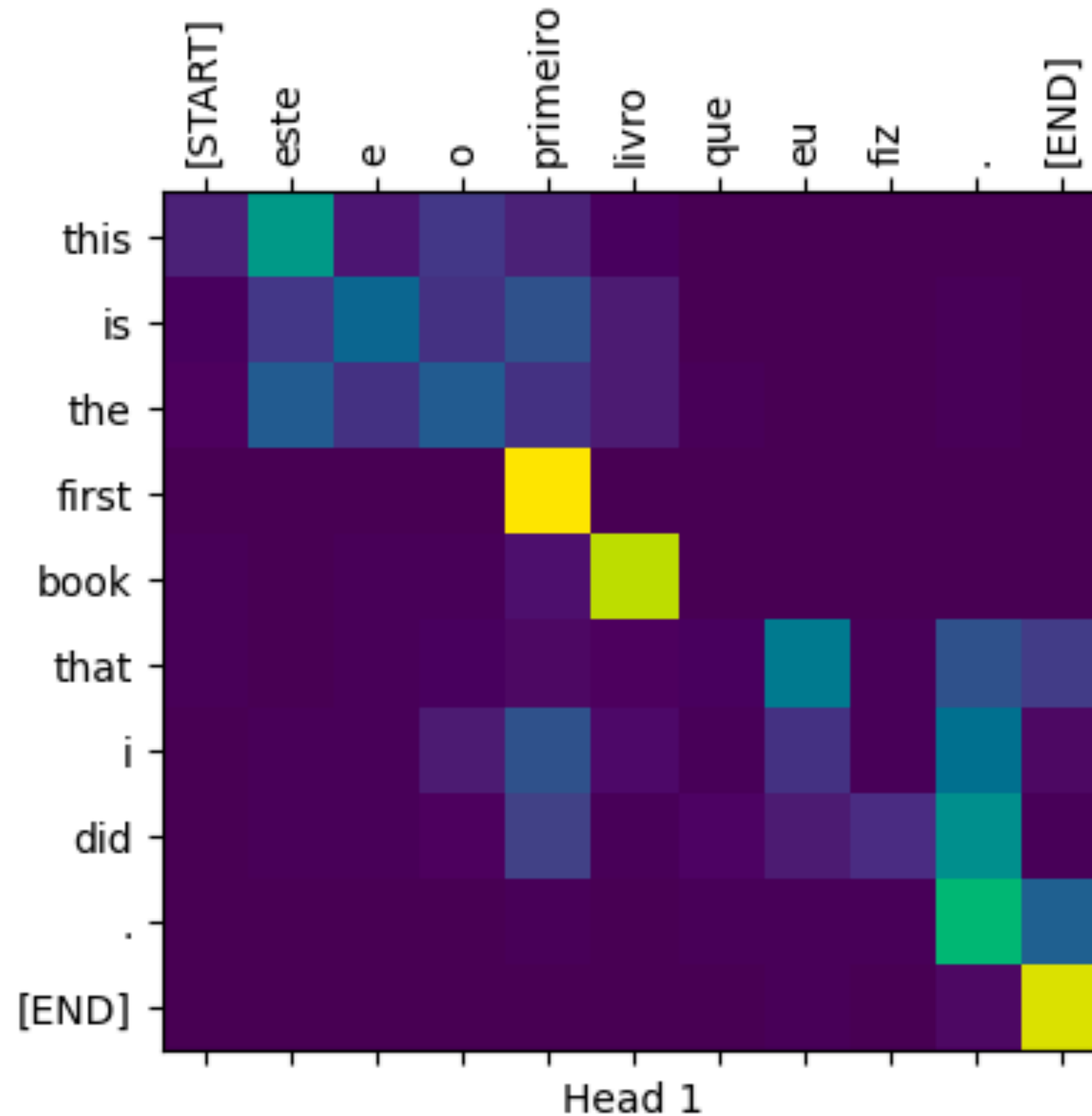
Figure 9.16 Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention.

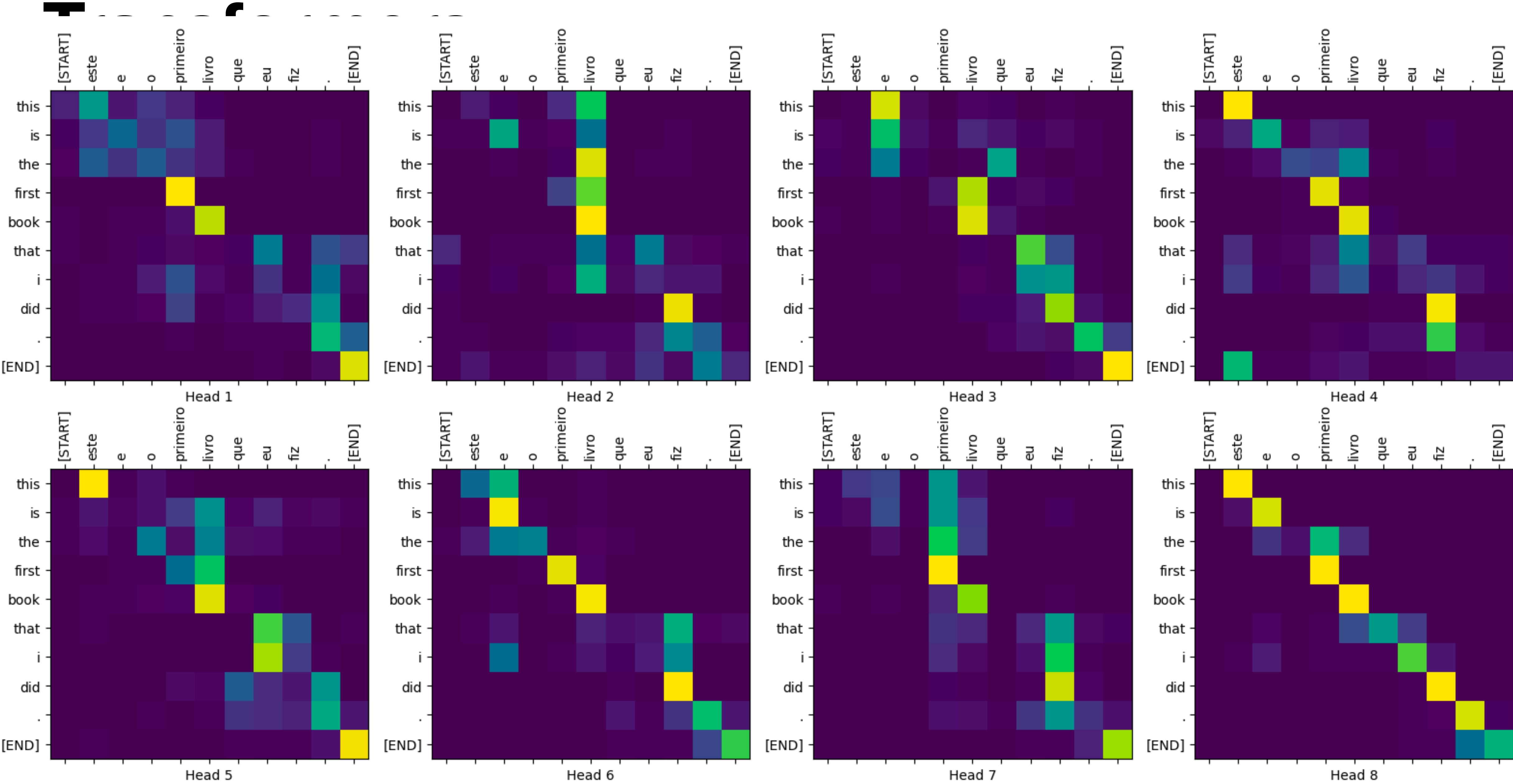
Multiheaded (Self) Attention

Multiheaded (Self) Attention

Repeat the attention process multiple times. Each KQV set can focus on a slightly different aspect of the representation.





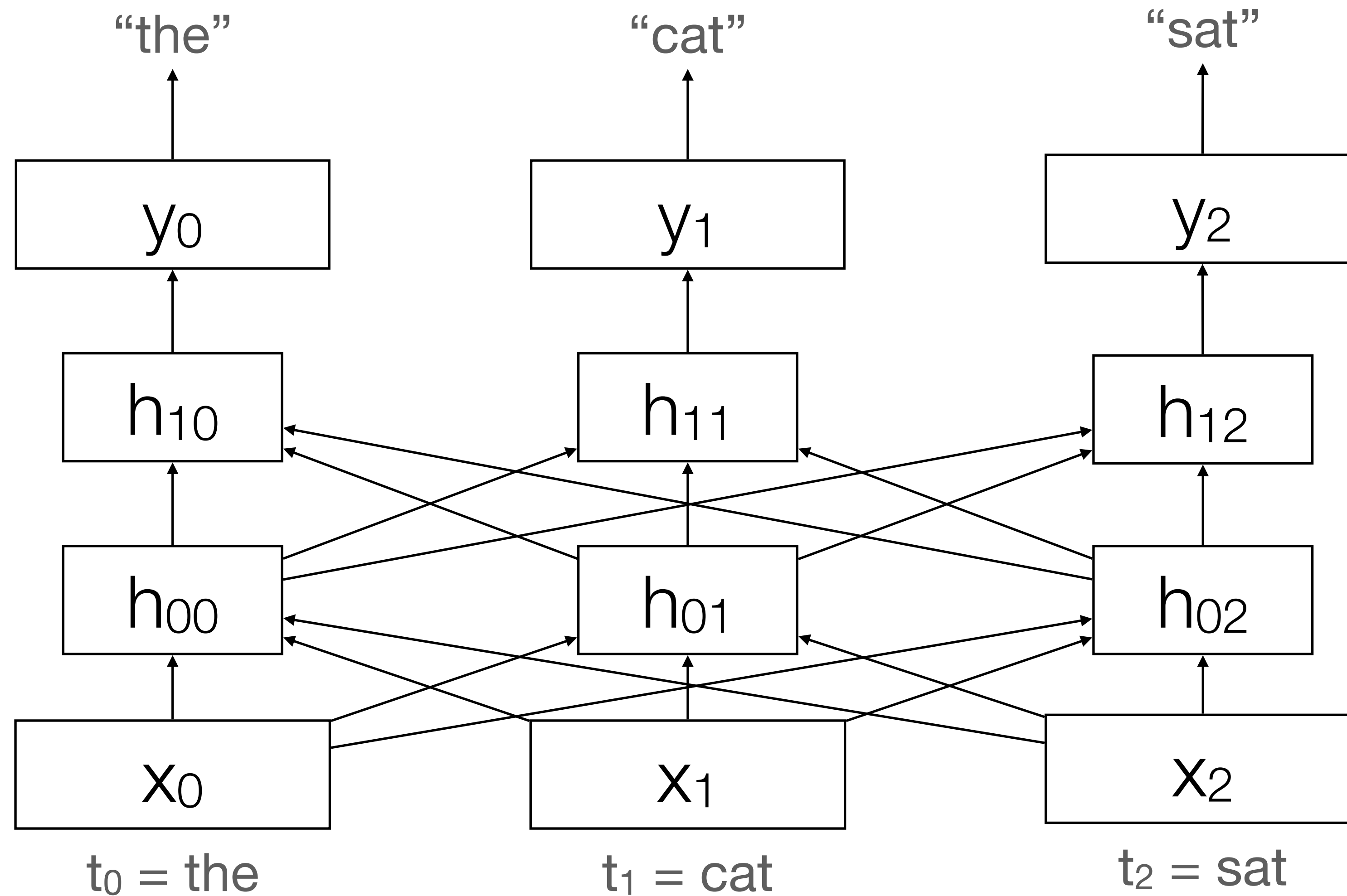


Topics

- NN Architectures for Language Modeling
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformers**
 - Architecture
 - Self Attention
 - **Blocks**
 - Positional Encodings

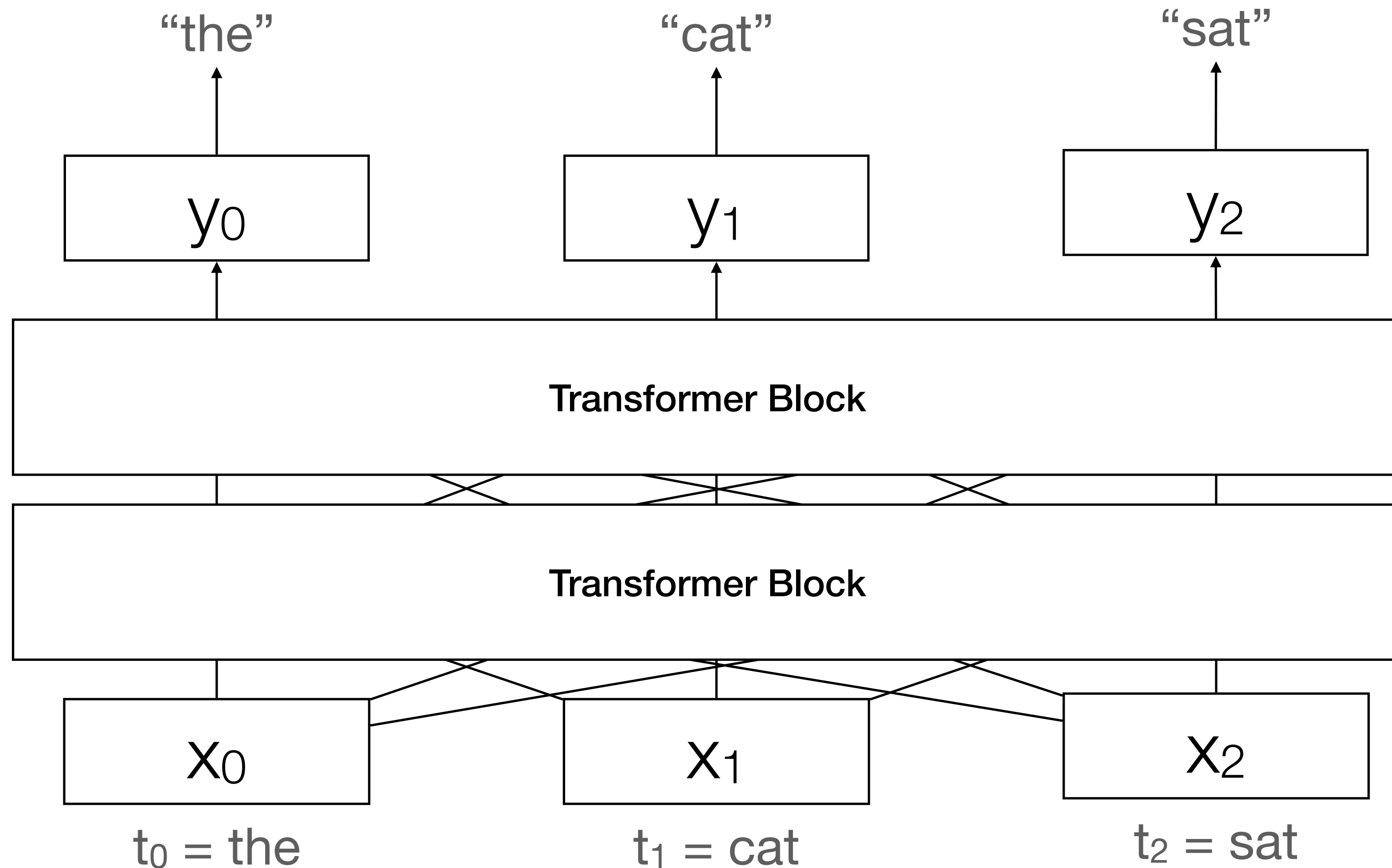
Transformers

Transformer Blocks



Transformers

Transformer Blocks



Transformers

Transformer Blocks

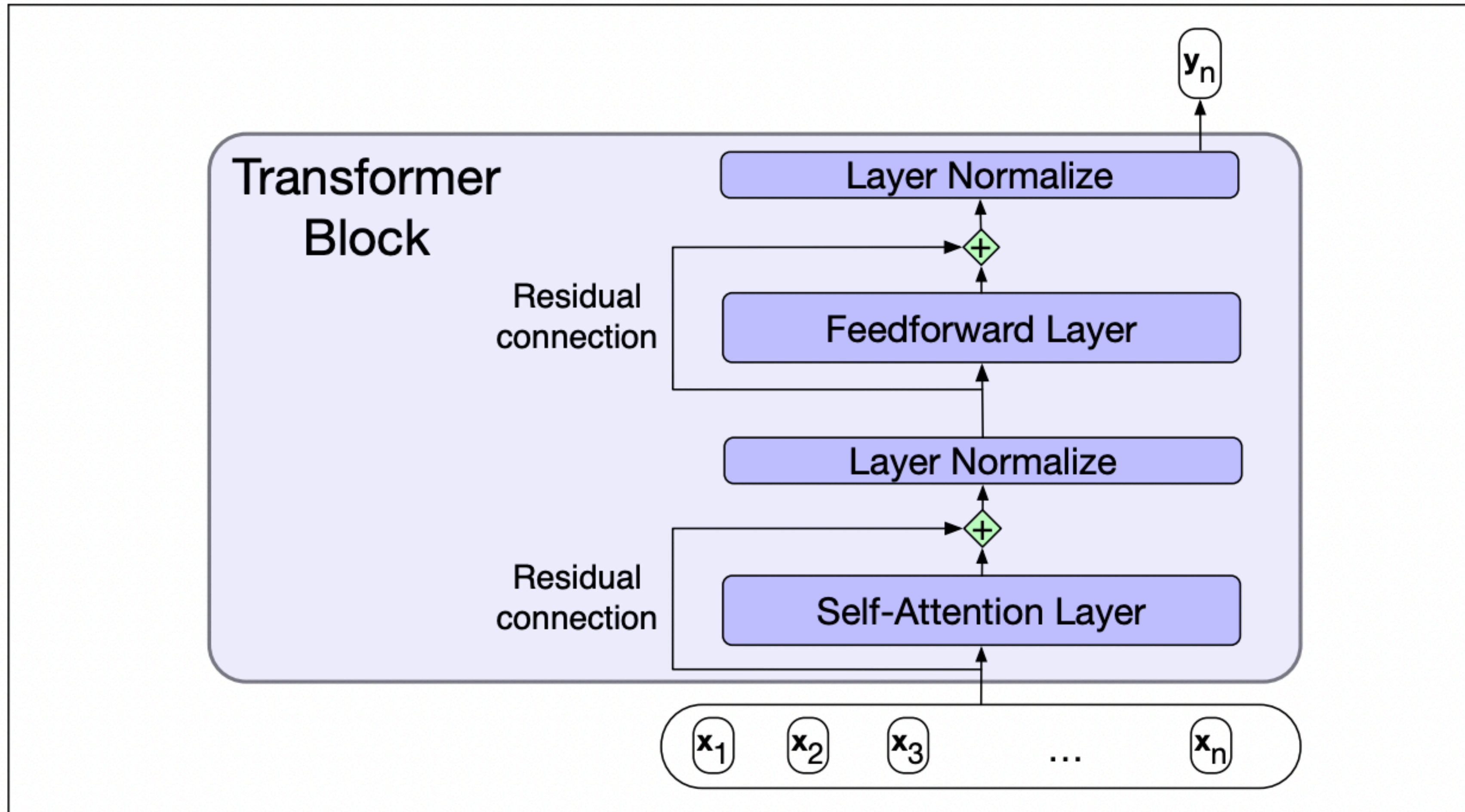


Figure 9.18 A transformer block showing all the layers.

Transformers

Transformer Blocks

just add input to output, to help
with training/vanishing
gradients

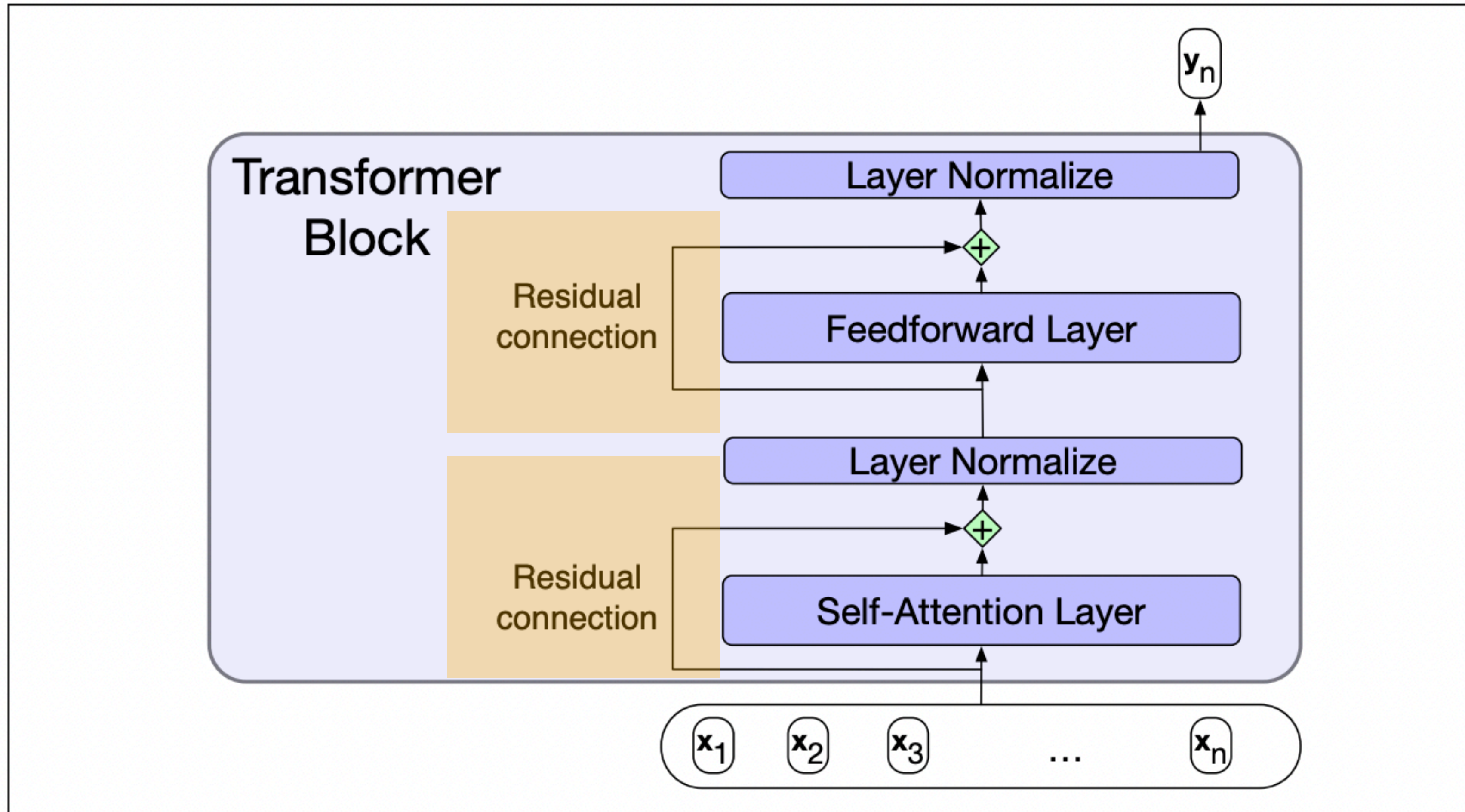


Figure 9.18 A transformer block showing all the layers.

Transformers

Transformer Blocks

same idea z-score normalization:
subtract mean, divide by standard
deviation (with some learnable
parameters, of course)

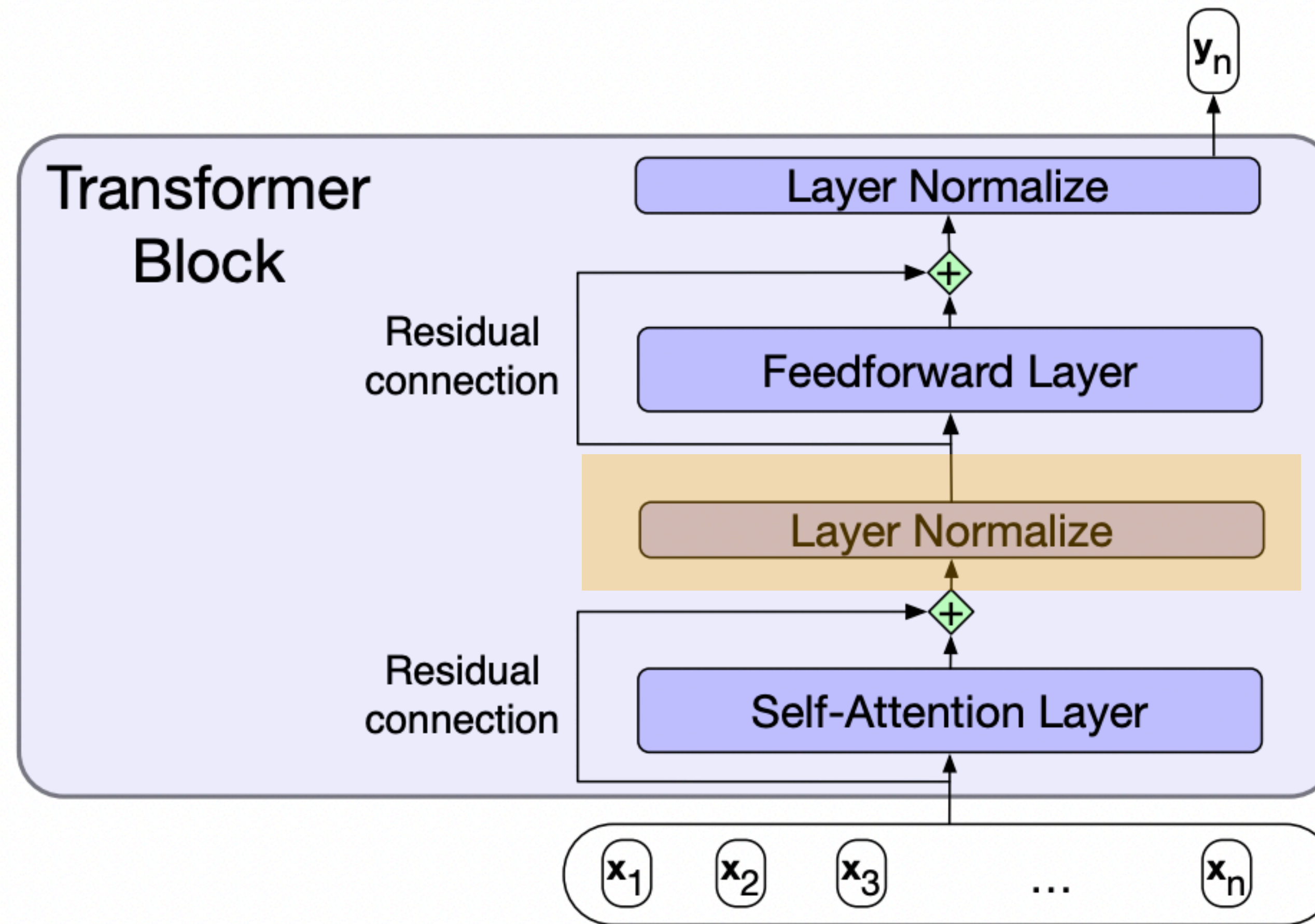


Figure 9.18 A transformer block showing all the layers.

Transformers

Transformer Blocks

simple perceptron-style layer to combine everything together

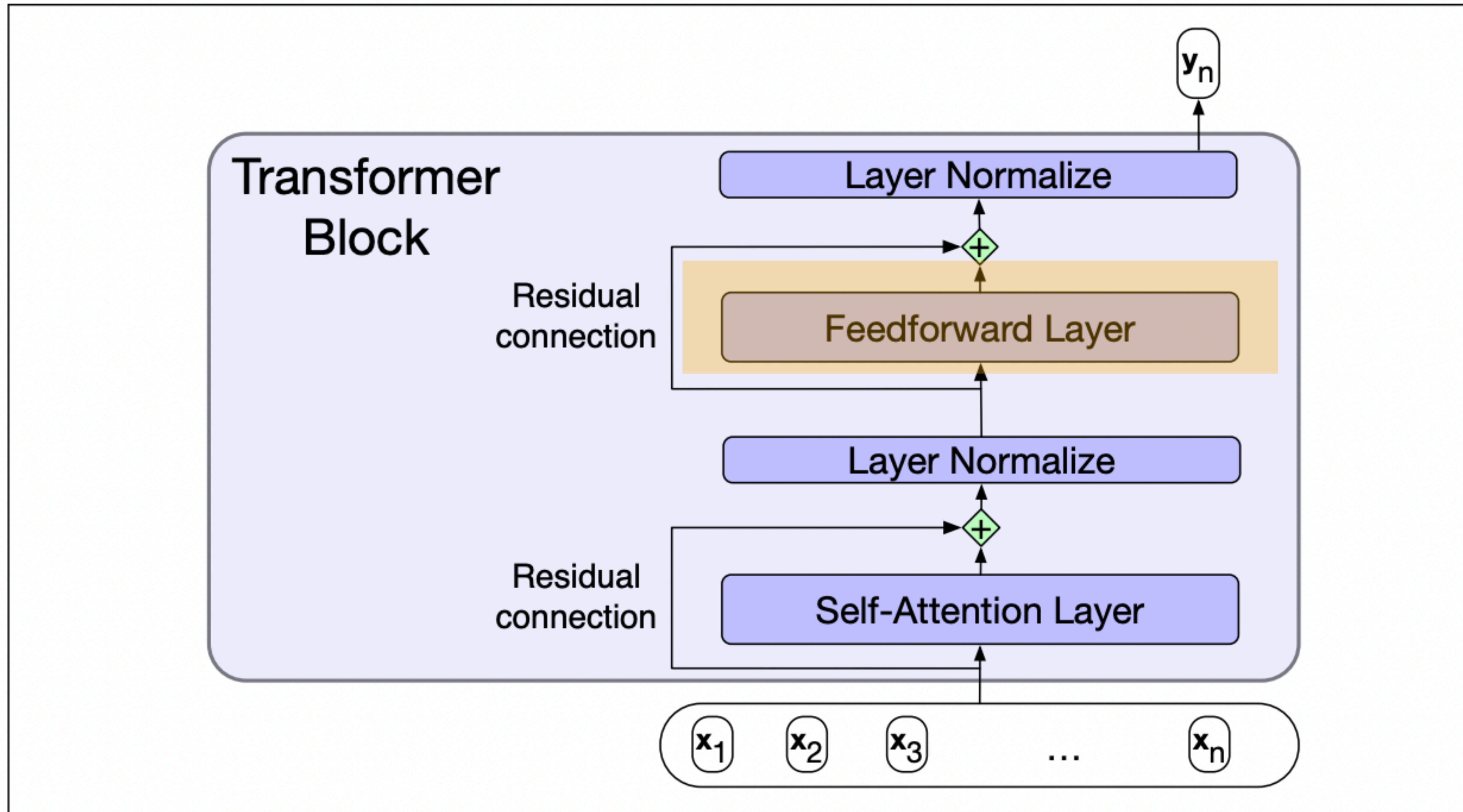


Figure 9.18 A transformer block showing all the layers.

Topics

- NN Architectures for Language Modeling
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformers**
 - Architecture
 - Self Attention
 - Blocks
 - **Positional Encodings**

Transformers

Positional Encodings

- Unlike RNNs/LSTMs—Transformers aren't actually aware of the order in which words occur!
 - Essentially, they are a (very fancy) bag of words
- Solution: Positional encodings
 - Idea: Just include an input with each word saying what position it is (e.g., “cat in the 3rd position”, “sat in the 4th position”)

Transformers

Positional Encodings

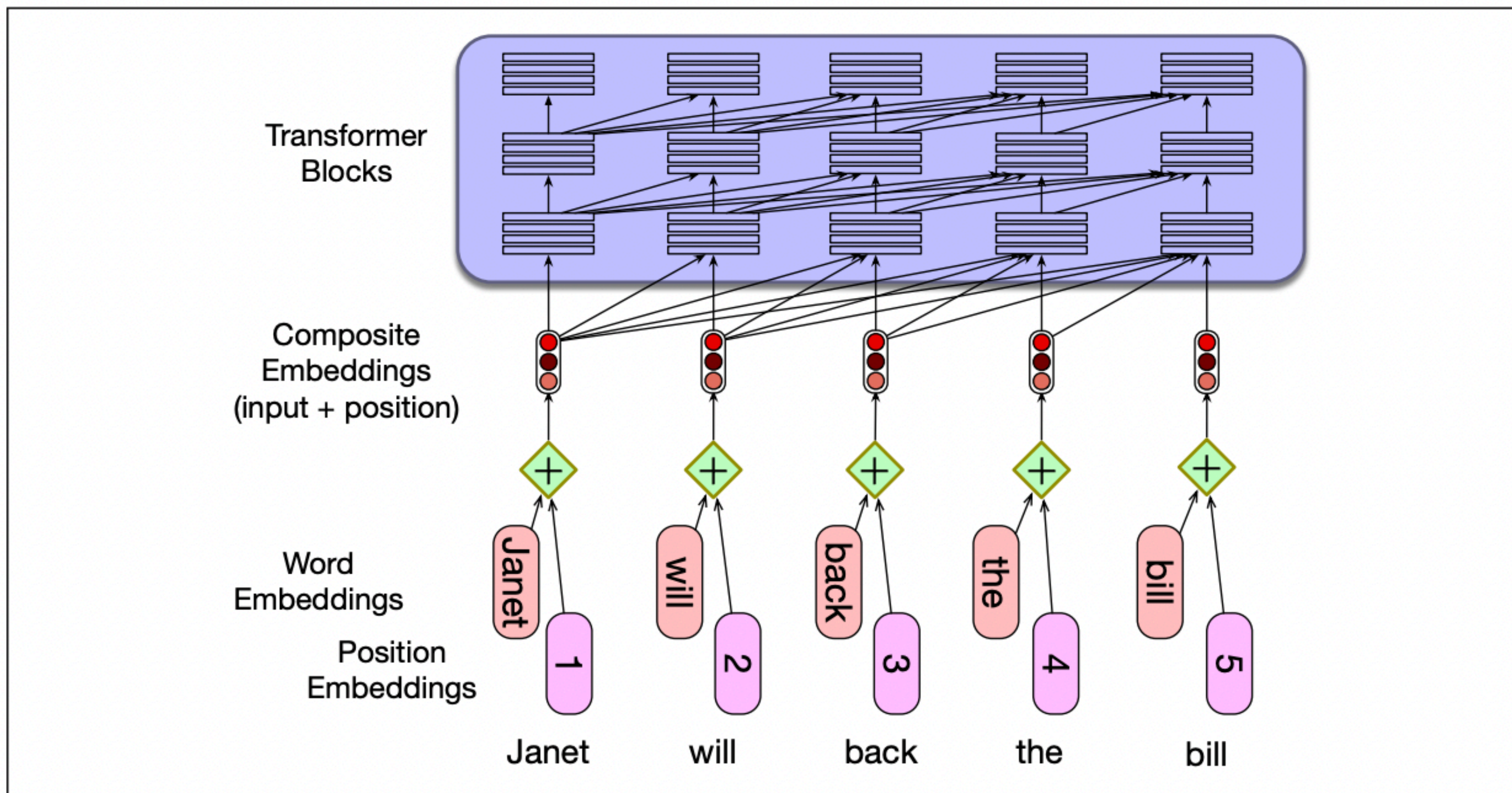


Figure 9.20 A simple way to model position: simply adding an embedding representation of the absolute position to the input word embedding.

Transformers

Positional Encodings

- Problems
 - Not the same as relative/order information (which we want in language). (Later work introduces relative positional encodings instead, and they seem to work better)
 - Less supervision for later positions
 - What about language being infinitely recursive?

Topics

- NN Architectures for Language Modeling
 - Recurrent Neural Network (RNN)
 - Long-Short Term Memory Network (LSTM)
 - **Transformers**

Transformers

What's the big deal?

- Attention
 - Very **minimal inductive bias**
 - Any **arbitrary graph structure** over the inputs can be learned
- Multiheadedness
 - Each “head” focuses on a **different subspace** of the input
 - E.g., one head can highlight syntactically connected words, while another finds pragmatically relevant information unconstrained by syntax
 - Ty is my cat. Yesterday while my husband and I weren't looking he killed a bird

Transformers

What's the big deal?

- Attention
 - Very **minimal inductive bias**
 - Any **arbitrary graph structure** over the inputs can be learned
- Multiheadedness
 - Each “head” focuses on a **different subspace** of the input
 - E.g., one head can highlight syntactically connected words, while another finds pragmatically relevant information unconstrained by syntax
 - **Ty** is my **cat**. Yesterday while my **husband** and I weren't looking **he**
killed a bird

Transformers

What's the big deal?

- Also: **very scalable!**
 - At layer N, no dependency between timesteps, so can be trained completely in parallel (unlike RNNs)
 - Faster training = bigger models + more data
 - Allows for massive **pretraining**

All done!
More questions?