Traditional Language Models

J’ai vu le match de foot

Traditional Language Models

Traditional Language Models

J’ai vu le match de foot

I saw the game of soccer

Traditional Language Models

J'ai vu le match de foot

I saw the game of soccer

I saw the soccer game

I saw the soccer match

Traditional Language Models

J'ai vu le match de foot

I saw the game of soccer

I saw the soccer game

I saw the soccer match

Traditional Language Models

J'ai vu le match de foot

- I saw the game of soccer
- I saw the soccer game
- I saw the soccer match
- Saw I the game of soccer

Traditional Language Models

J'ai vu le match de foot

Sequence	$P(\text{Sequence})$
I saw the game of soccer	4.5 e-5
I saw the soccer game	6.0 e-5
I saw the soccer match	4.6 e-5
Saw I the game of soccer	2.6 e-9

Traditional Language Models

J'ai vu le match de foot

Sequence	$P(\text{Sequence})$
I saw the game of soccer	4.5 e-5
I saw the soccer game	6.0 e-5
I saw the soccer match	4.6 e-5
Saw I the game of soccer	2.6 e-9

N-grams

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

Bigrams

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

- Large N-grams to capture dependencies between distant words

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

- Large N-grams to capture dependencies between distant words
- Need a lot of space and RAM

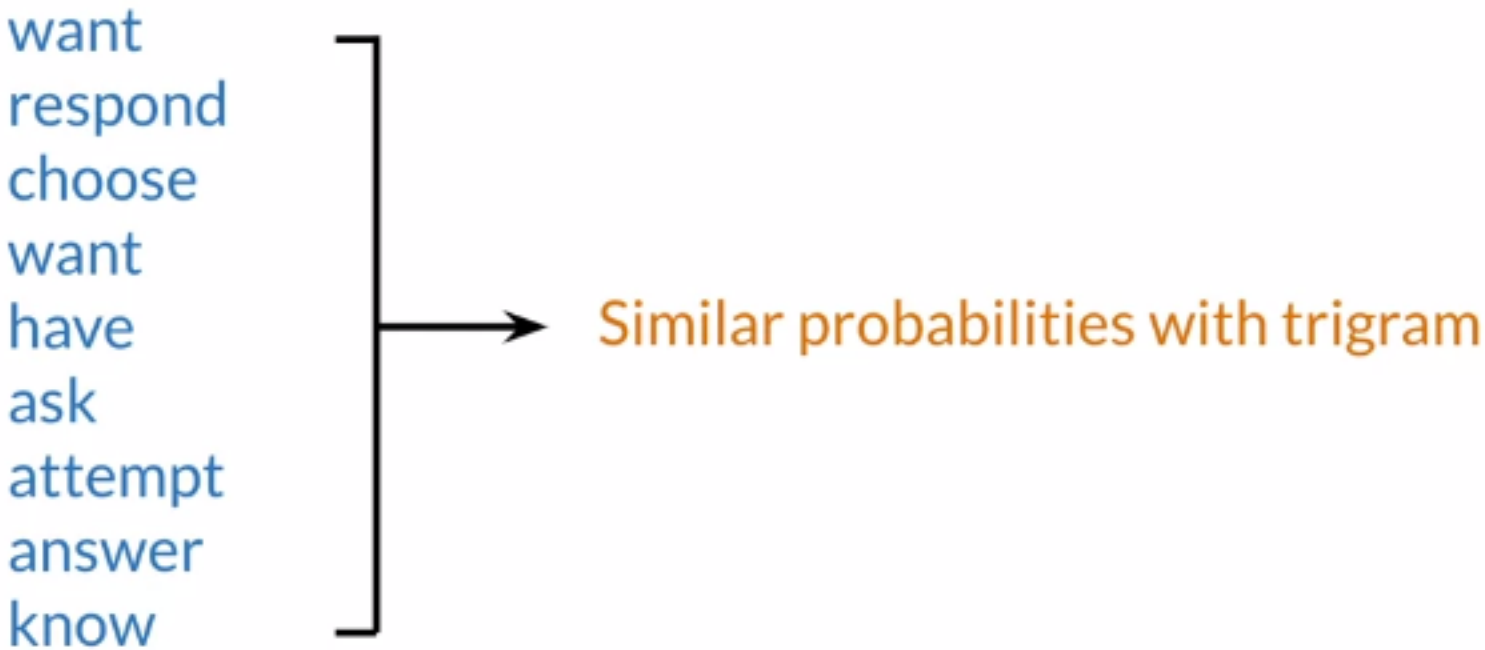
Advantages of RNNs

Advantages of RNNs

Nour was supposed to study with me. I called her but she did not _____

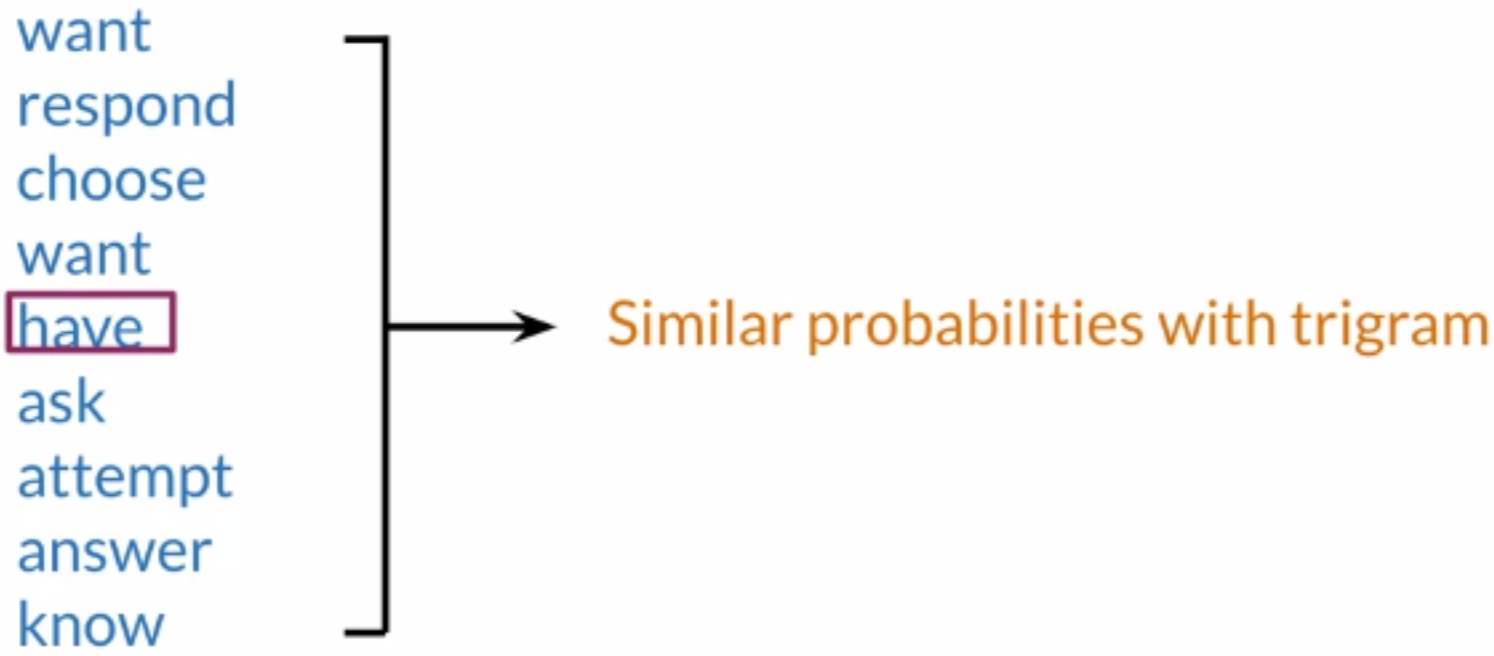
Advantages of RNNs

Nour was supposed to study with me. I called her but she did not _____



Advantages of RNNs

Nour was supposed to study with me. I called her but she did not have



Advantages of RNNs

Nour was supposed to study with me. I called her but she did not

want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

Similar probabilities with trigram

Advantages of RNNs

Nour was supposed to study with me. I called her but she did not

want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

Similar probabilities with trigram

Advantages of RNNs

Nour was supposed to study with me. I called her but she did not answer

want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

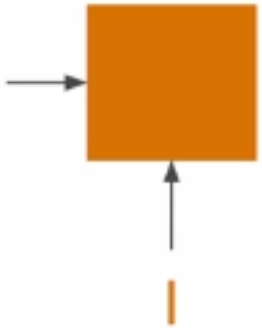
Similar probabilities with trigram

RNNs Basic Structure

I called her but she did not _____

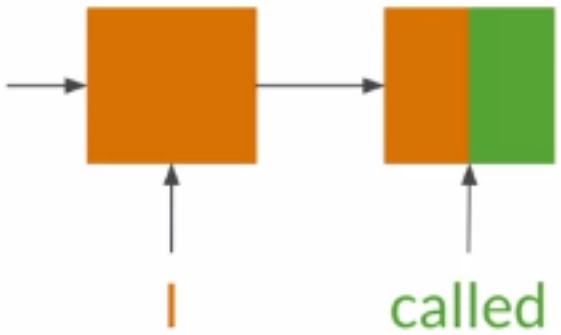
RNNs Basic Structure

I called her but she did not _____



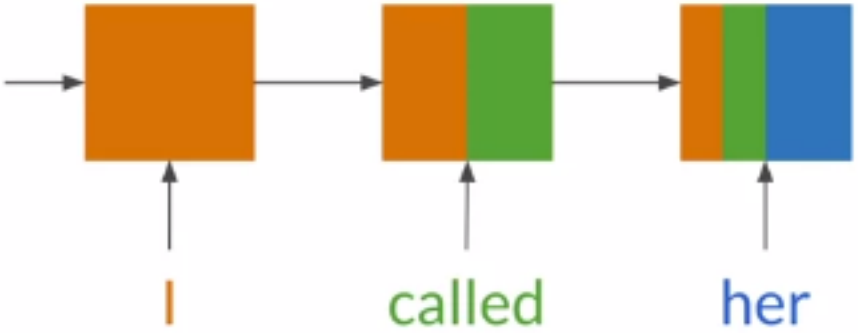
RNNs Basic Structure

I called her but she did not _____



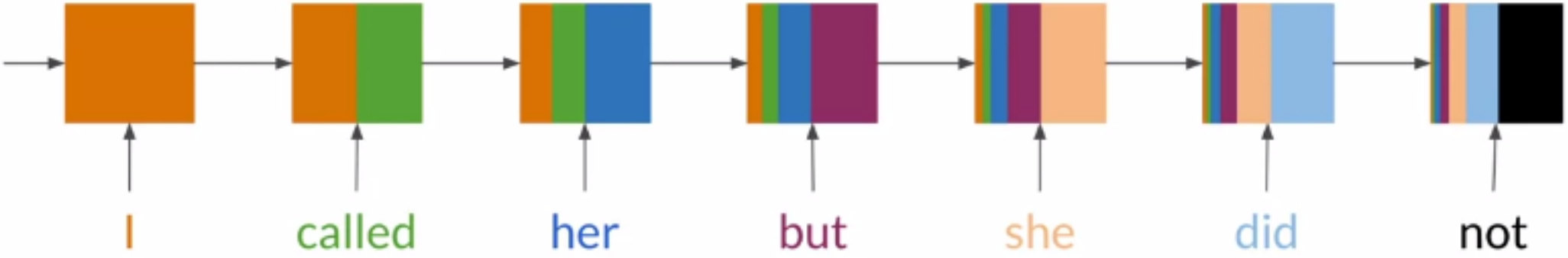
RNNs Basic Structure

I called her but she did not _____



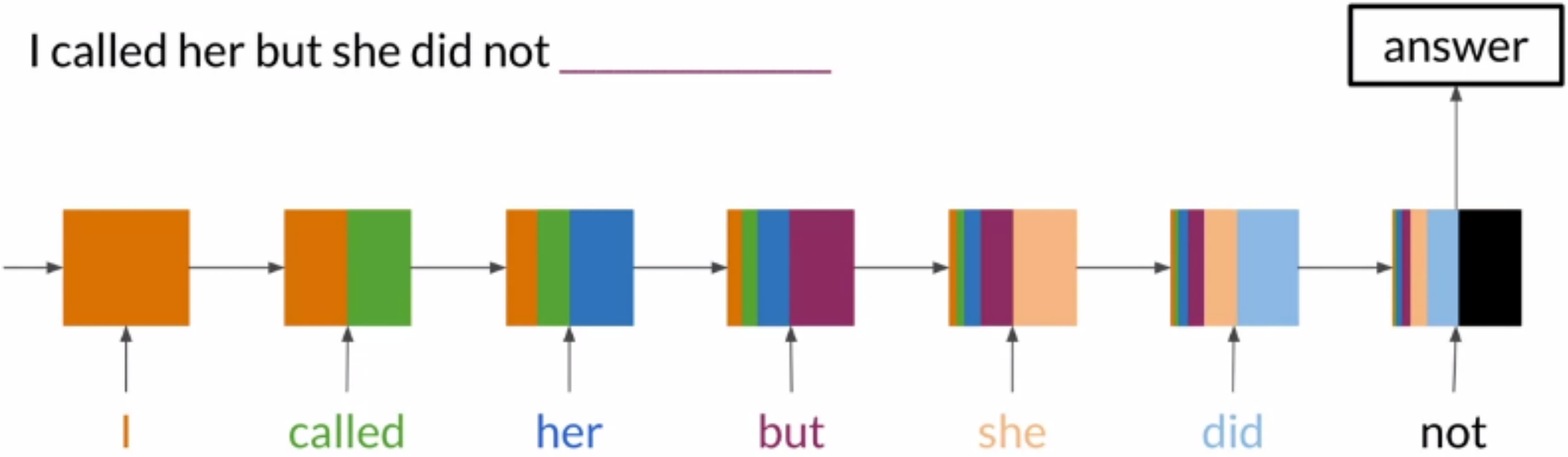
RNNs Basic Structure

I called her but she did not _____



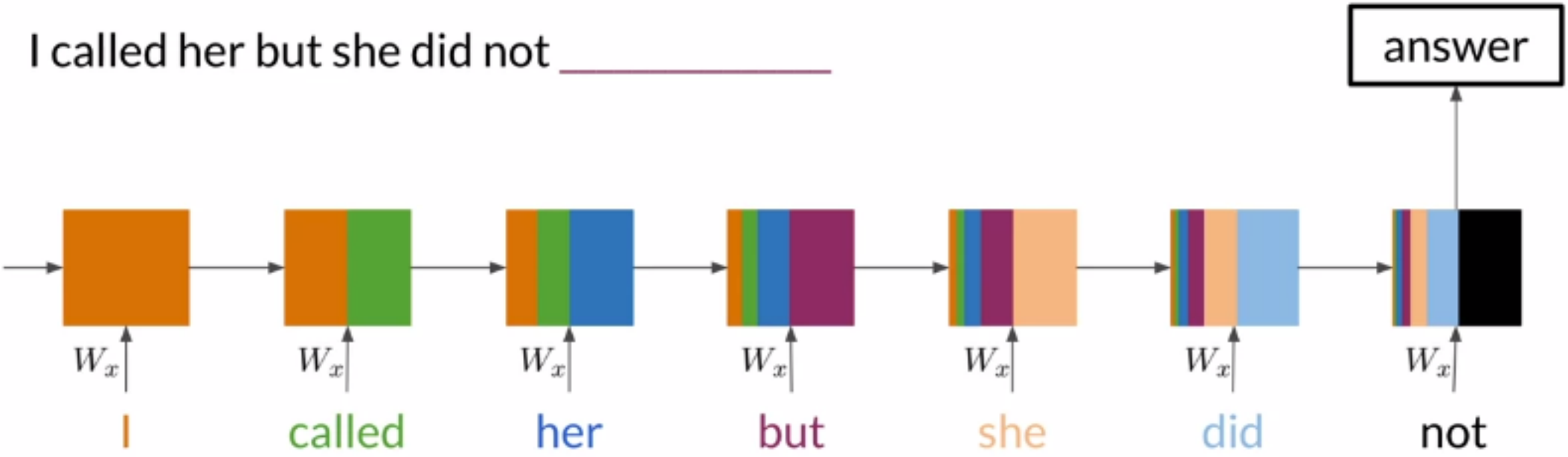
RNNs Basic Structure

I called her but she did not _____



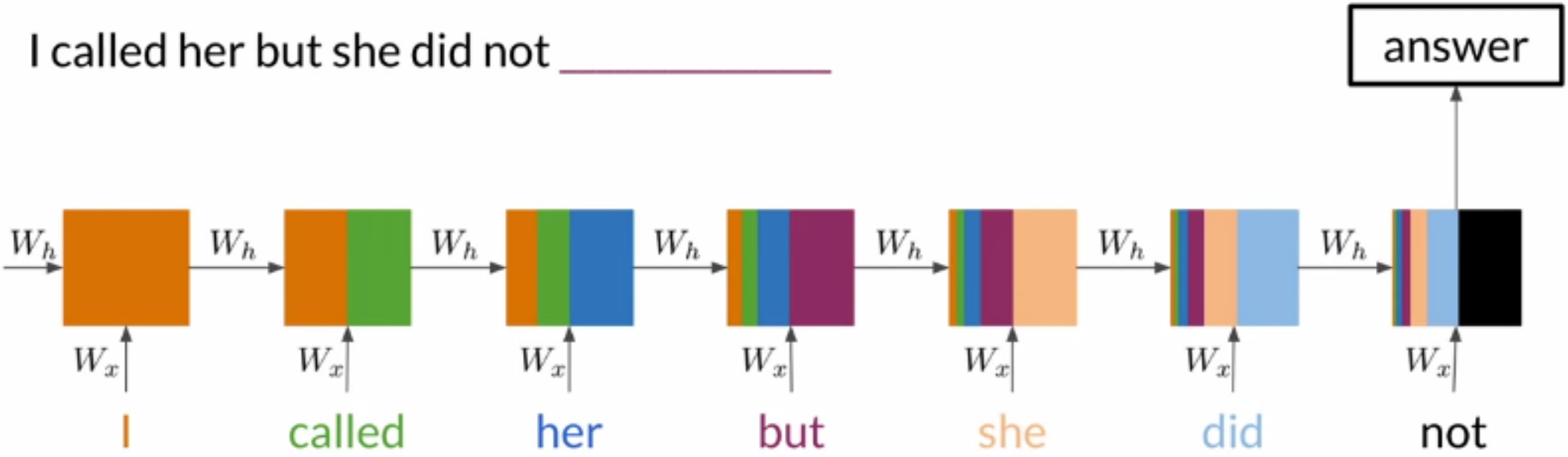
RNNs Basic Structure

I called her but she did not _____



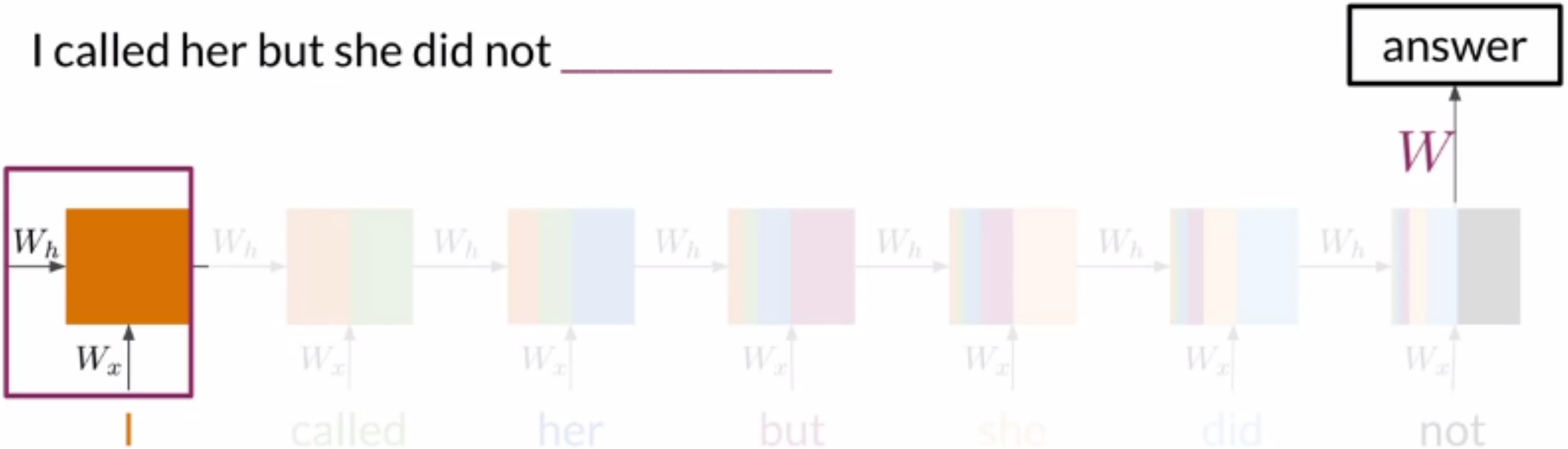
RNNs Basic Structure

I called her but she did not _____



RNNs Basic Structure

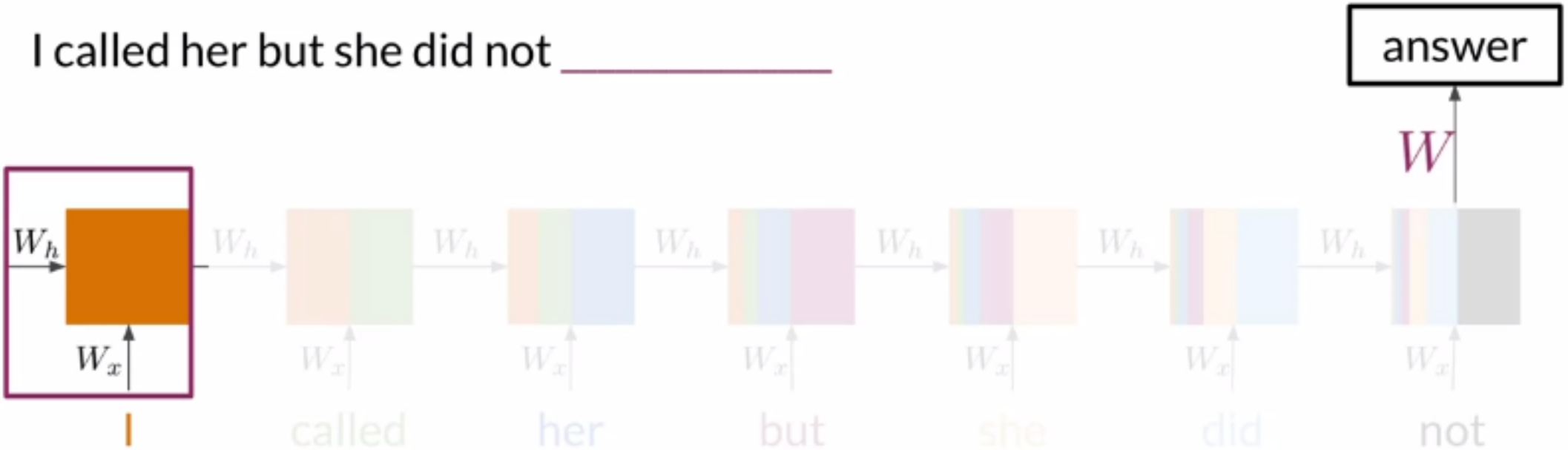
I called her but she did not _____



Learnable parameters

RNNs Basic Structure

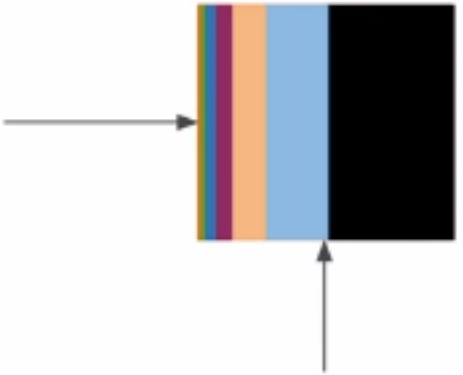
I called her but she did not _____



Learnable parameters

Summary

- RNNs model relationships among distant words
- In RNNs a lot of computations share parameters



One to One

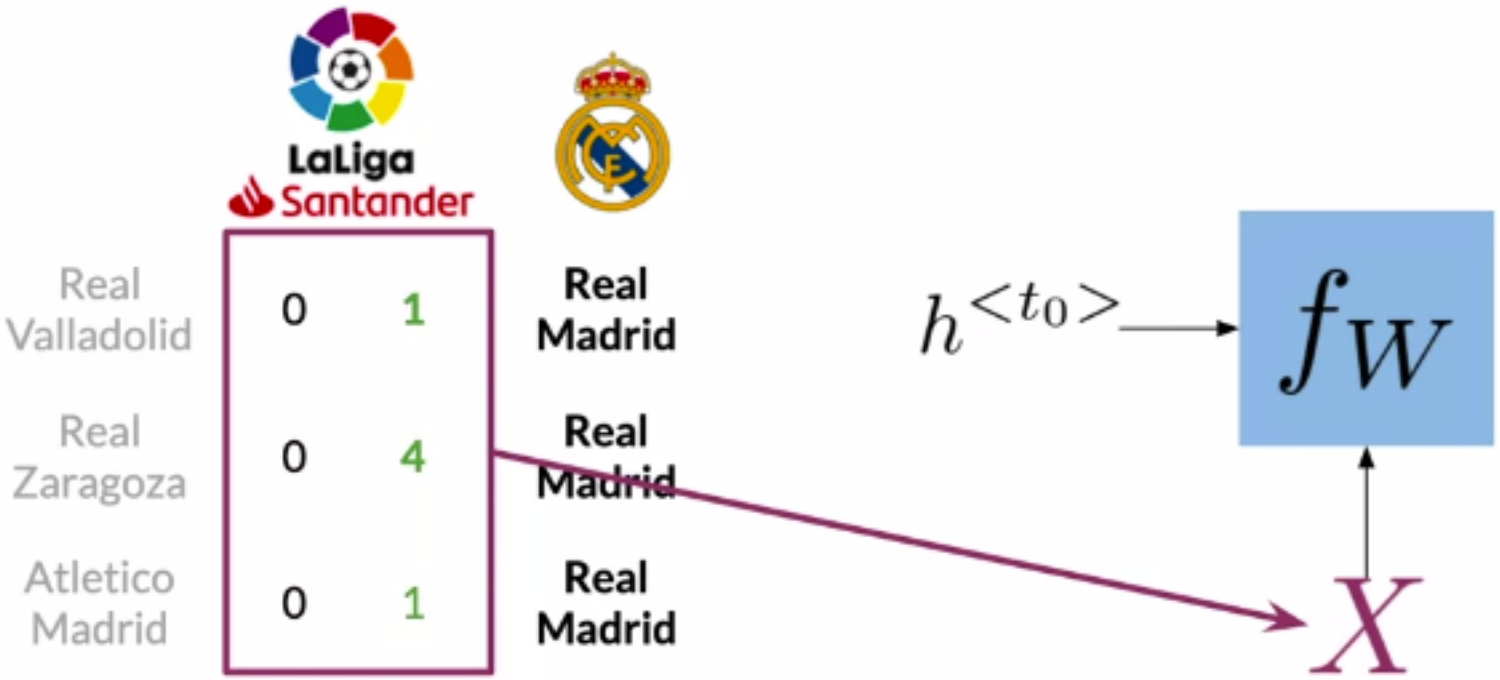
One to One

Real Valladolid	0	1	**Real Madrid**
Real Zaragoza	0	4	**Real Madrid**
Atletico Madrid	0	1	**Real Madrid**

One to One


LaLiga
Santander
Real Valladolid
0
1
Real Madrid
Real Zaragoza
0
4
Real Madrid
Atletico Madrid
0
1
Real Madrid
X

One to One


LaLiga
Santander
Real Valladolid
0
1
Real Madrid
Real Zaragoza
0
4
Real Madrid
Atletico Madrid
0
1
Real Madrid
$h^{<t_0>}$
f_W
X

One to One


LaLiga
Santander
Real Valladolid
0
1
Real Madrid
Real Zaragoza
0
4
Real Madrid
Atletico Madrid
0
1
Real Madrid
$h^{<t_0>}$
f_W
X

One to Many

One to Many

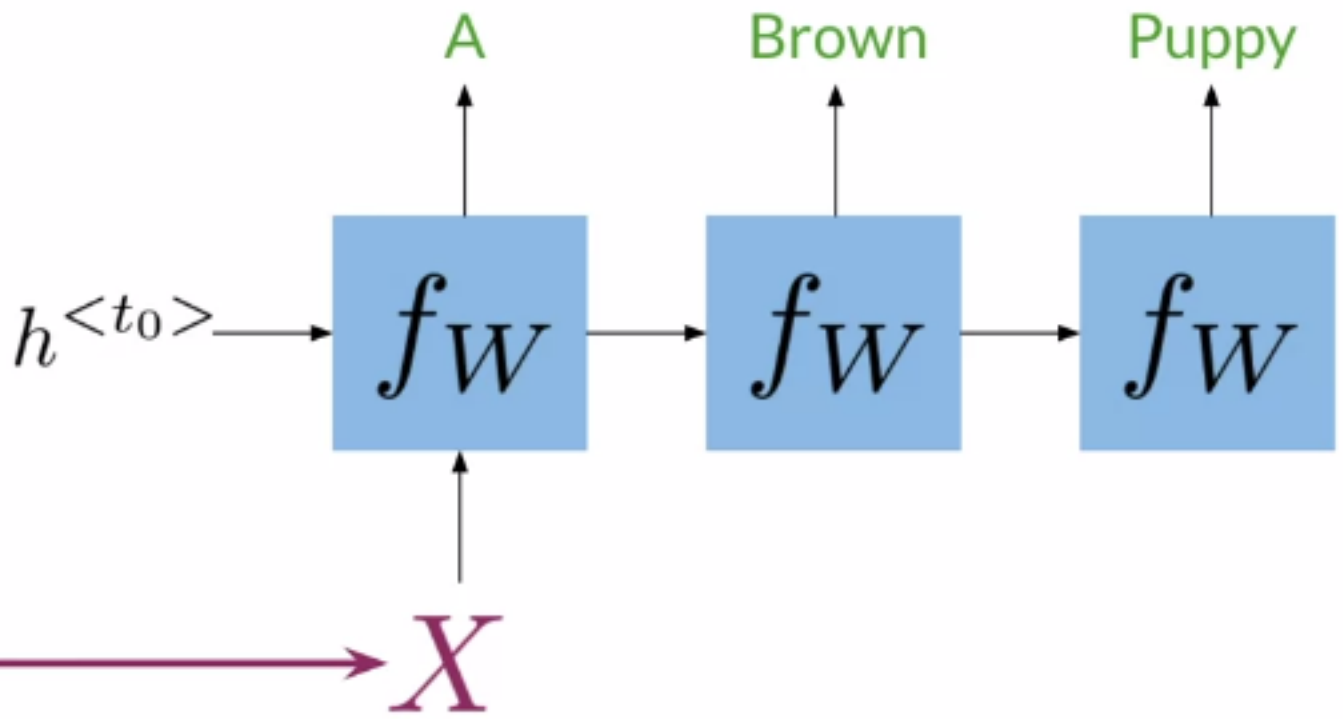
Caption generation

X

A Brown Puppy

One to Many

Caption generation



Many to One

Many to One

Sentiment analysis

Tweet: I am very happy !

Many to One

Sentiment analysis

Tweet: I am very happy !

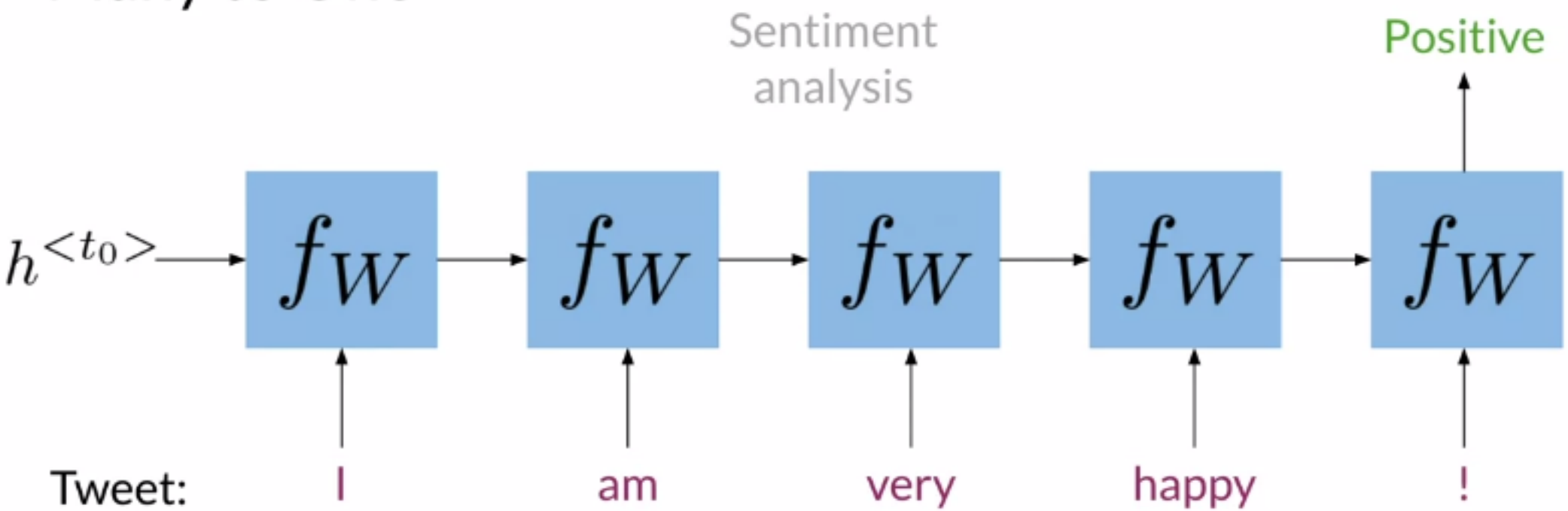
Many to One

Sentiment analysis

Positive

Tweet: I am very happy !

Many to One


Sentiment analysis
Positive
$h^{<t_0>}$
fw
fw
fw
fw
fw
Tweet:
I
am
very
happy
!

Many to Many

Many to Many

J’ ai faim

Machine Translation

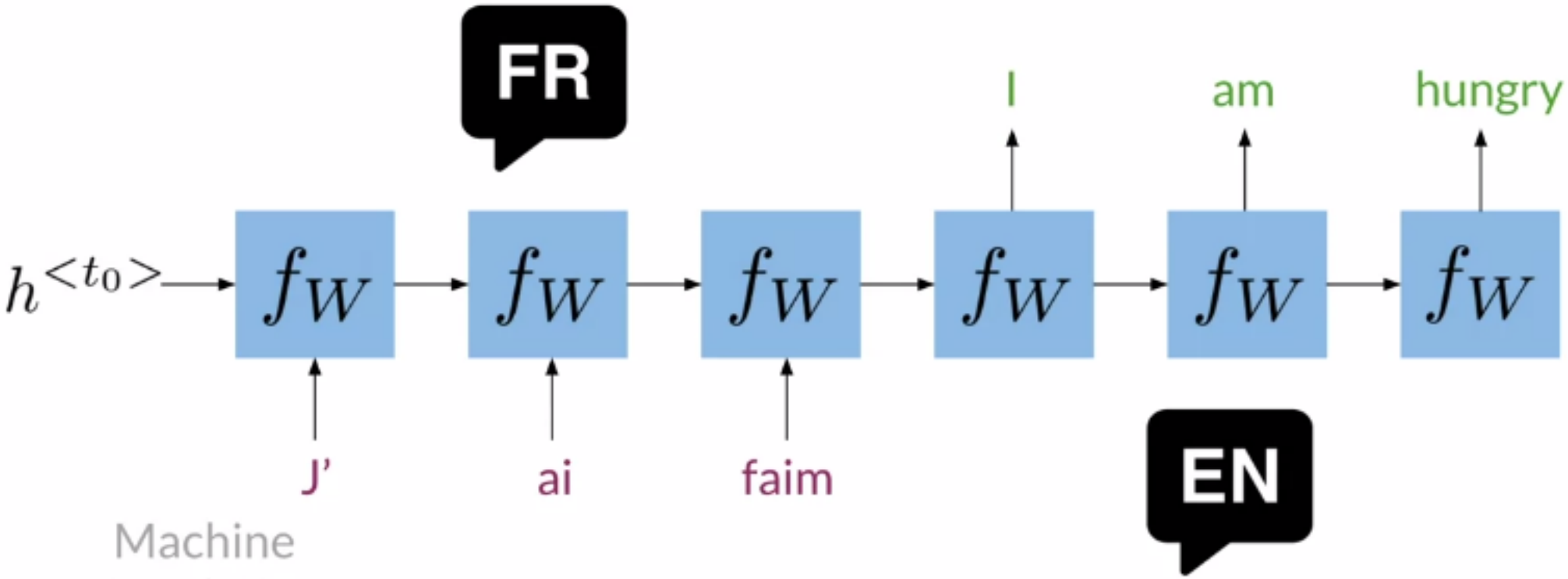
Many to Many

I am hungry

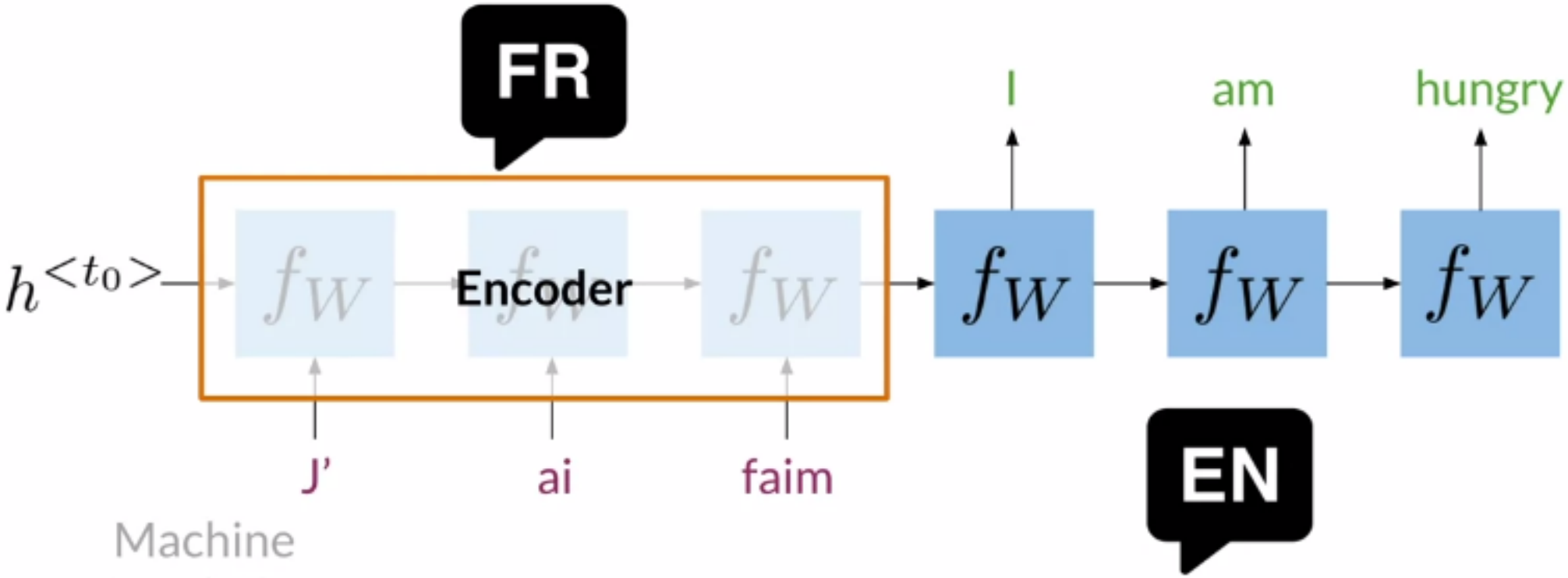
J' ai faim

Machine Translation

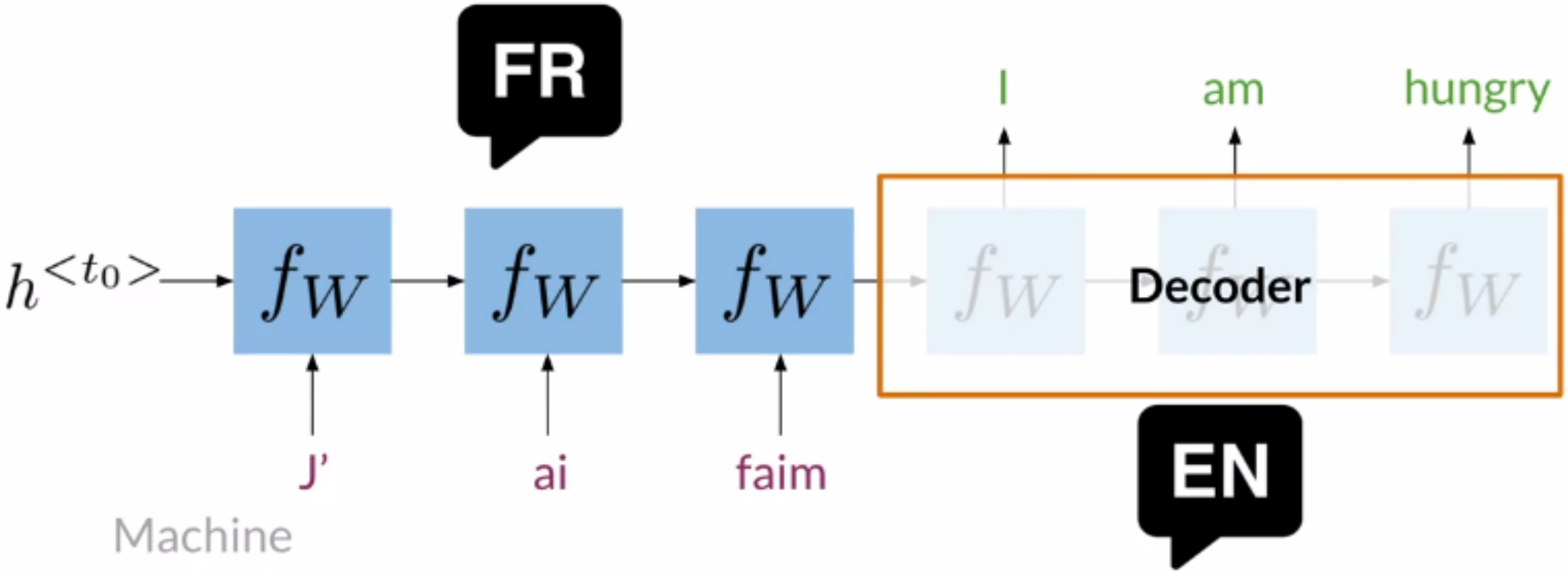
Many to Many


FR
I
am
hungry
$h^{<t_0>}$
f_W
f_W
f_W
f_W
f_W
f_W
J'
ai
faim
EN
Machine Translation

Many to Many


FR
I
am
hungry
$h^{<t_0>}$
f_W
Encoder
f_W
f_W
f_W
f_W
f_W
J'
ai
faim
EN
Machine
Translation

Many to Many



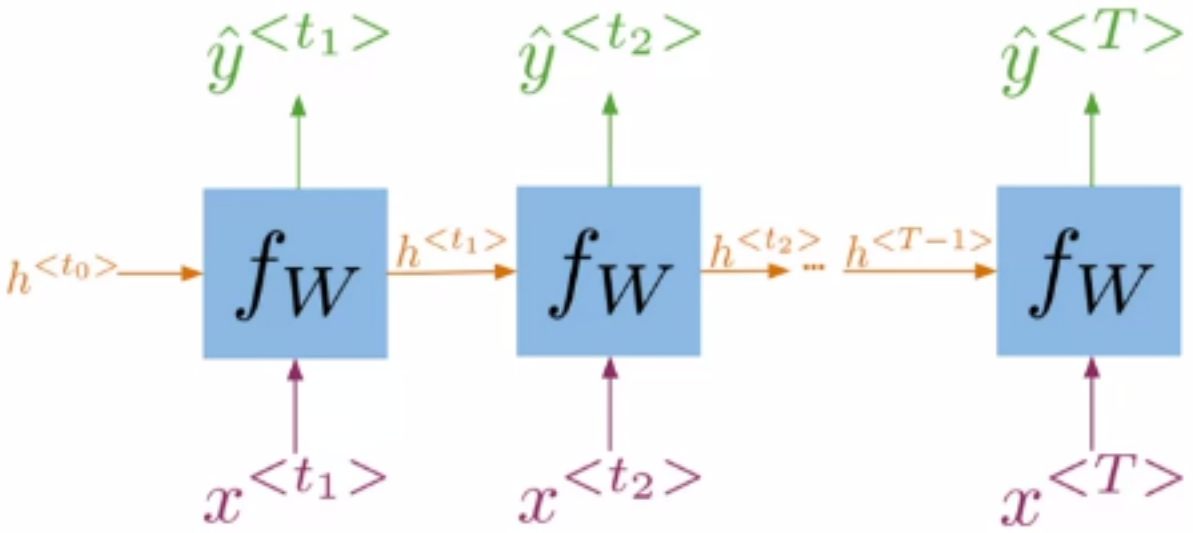
Machine Translation

Summary

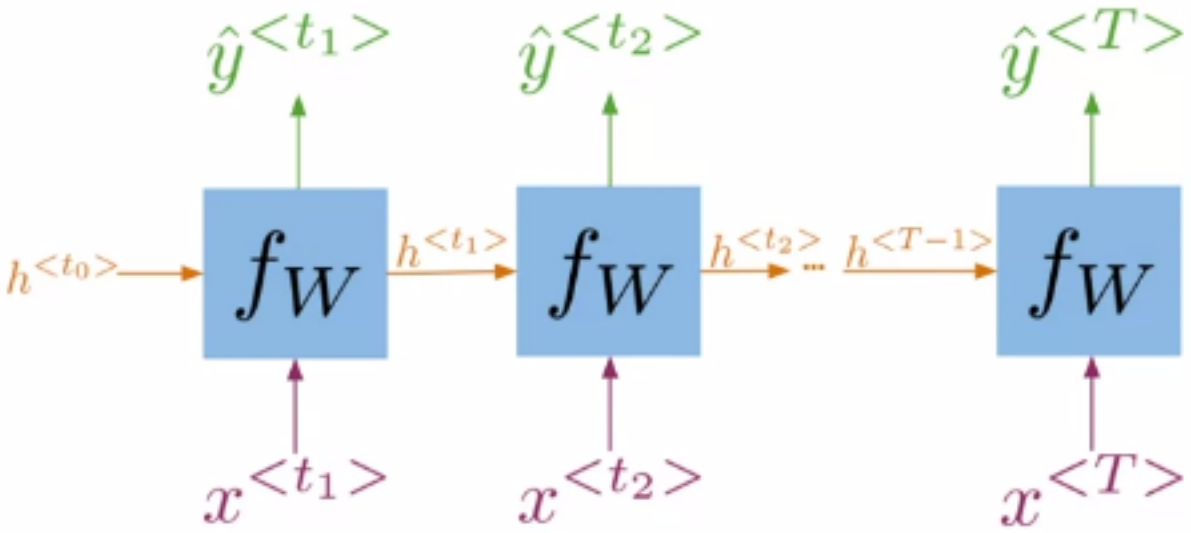
- RNNs can be implemented for a variety of NLP tasks
- Applications include Machine translation and caption generation

A Vanilla RNN

A Vanilla RNN

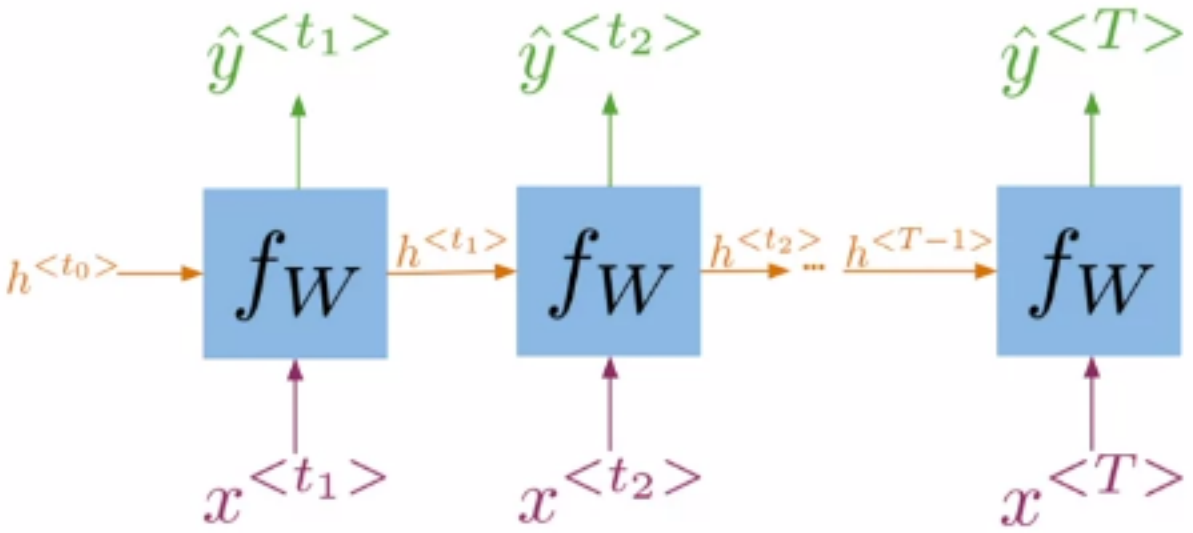


A Vanilla RNN



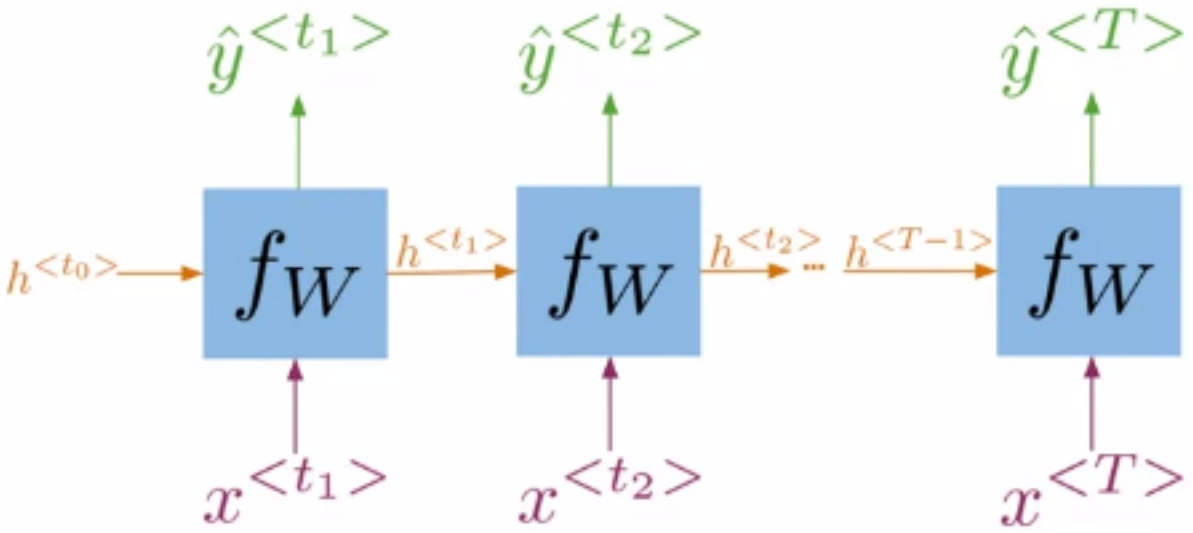
$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

A Vanilla RNN



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

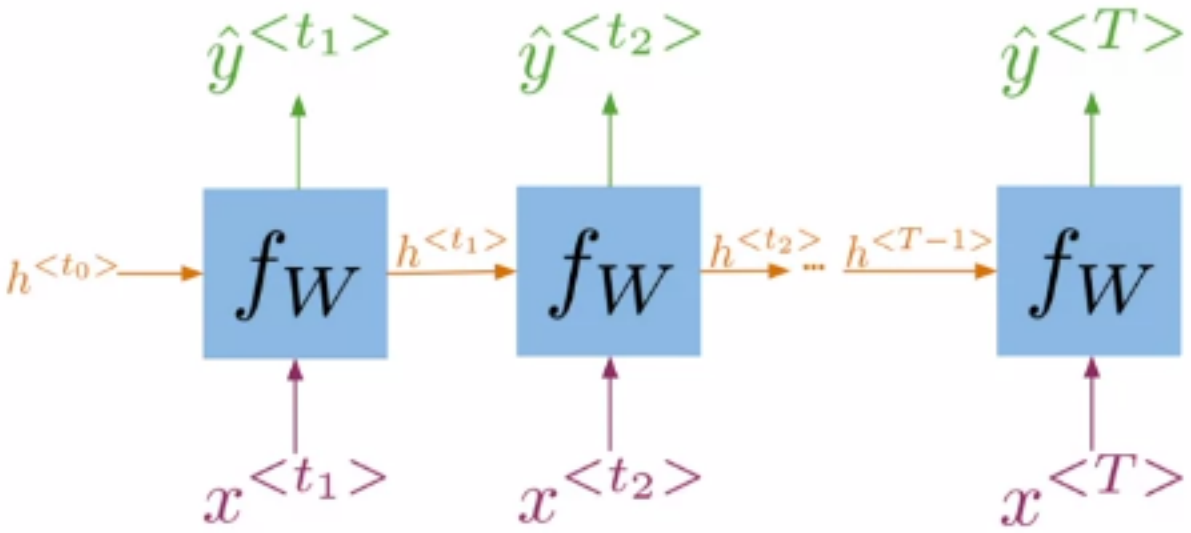
A Vanilla RNN



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$

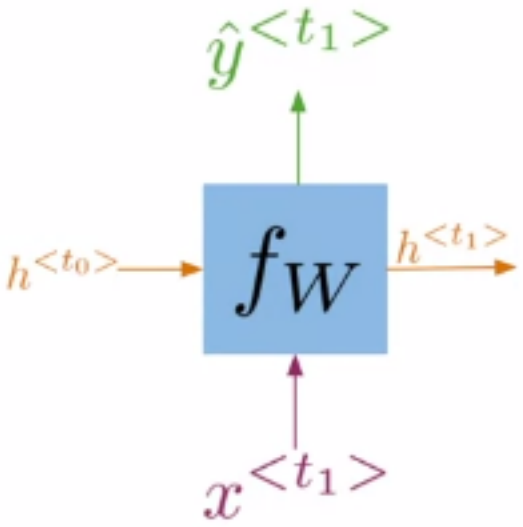
A Vanilla RNN



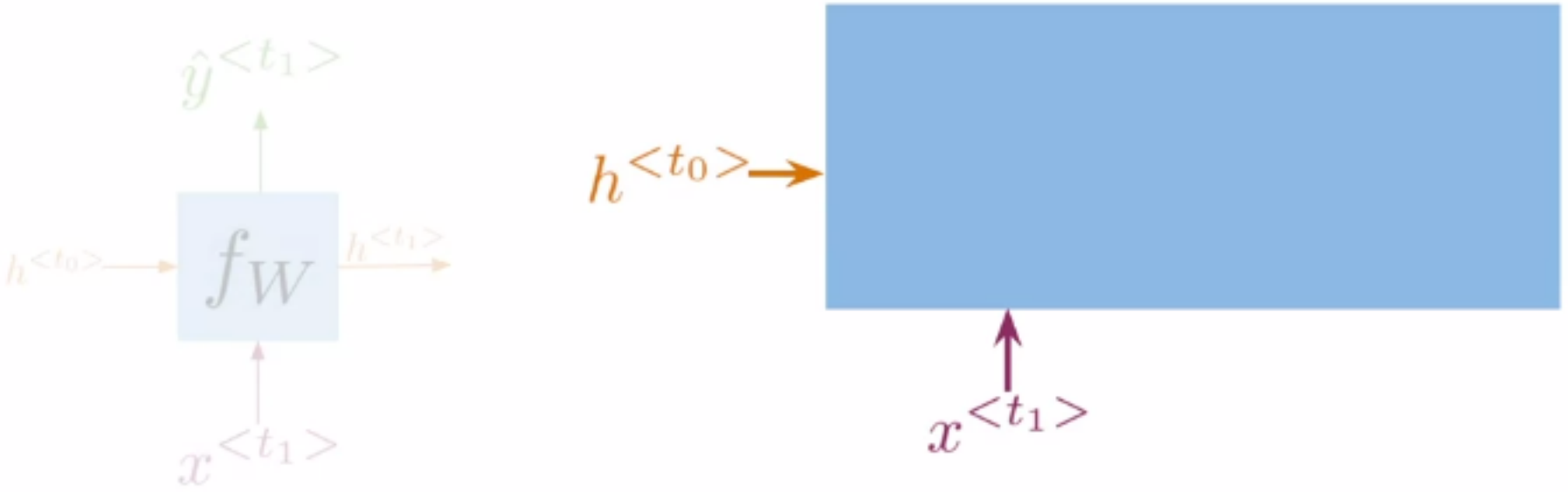
$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$

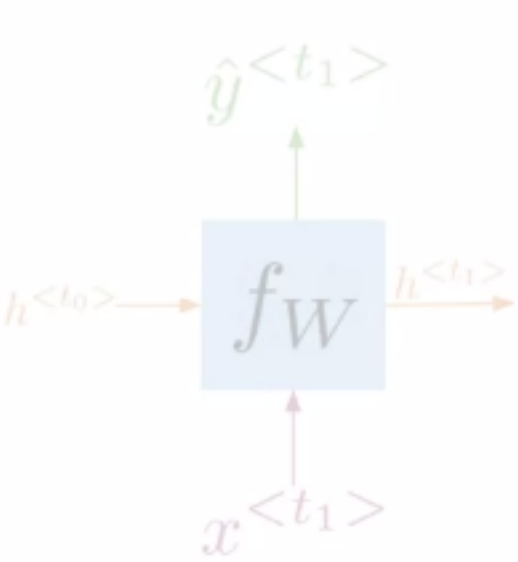
A Vanilla RNN

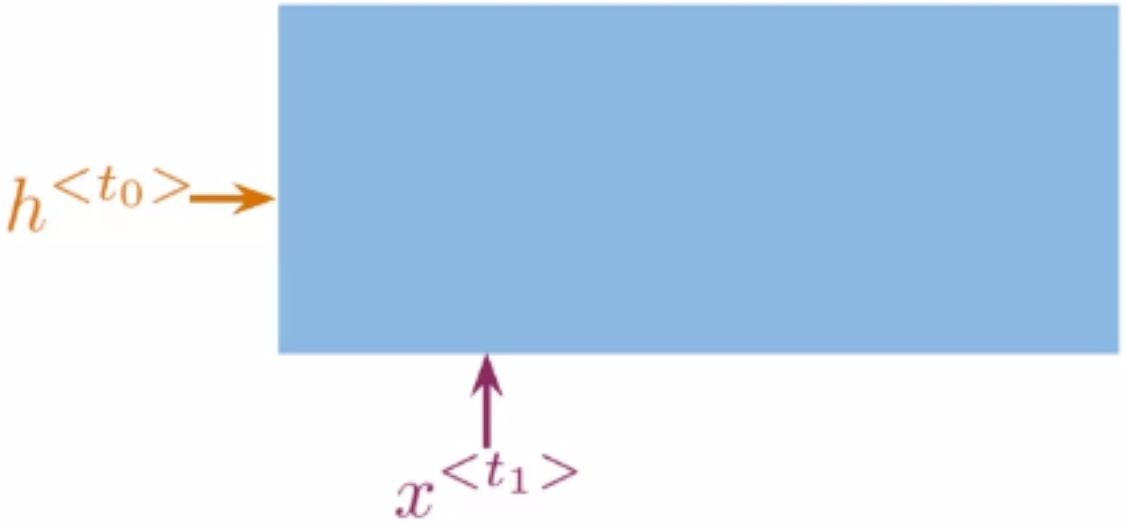


A Vanilla RNN



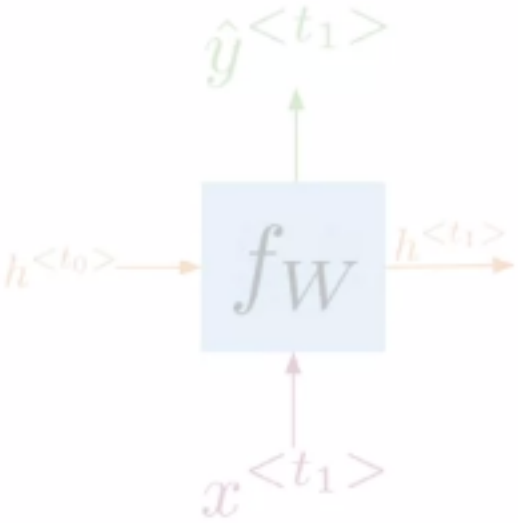
A Vanilla RNN

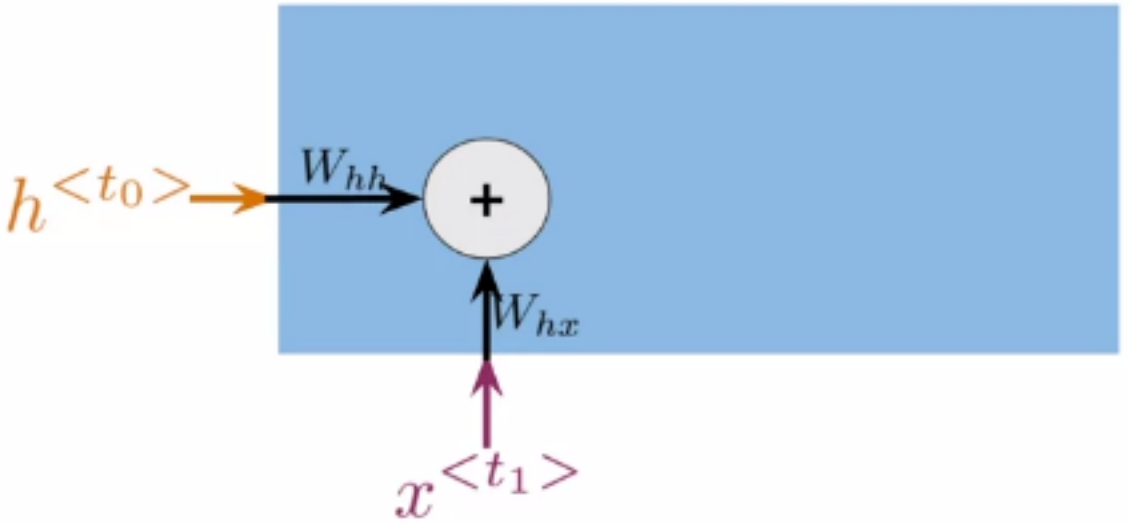




$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$

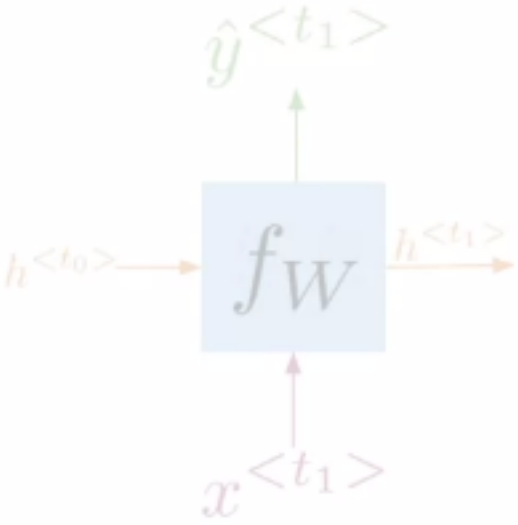
A Vanilla RNN

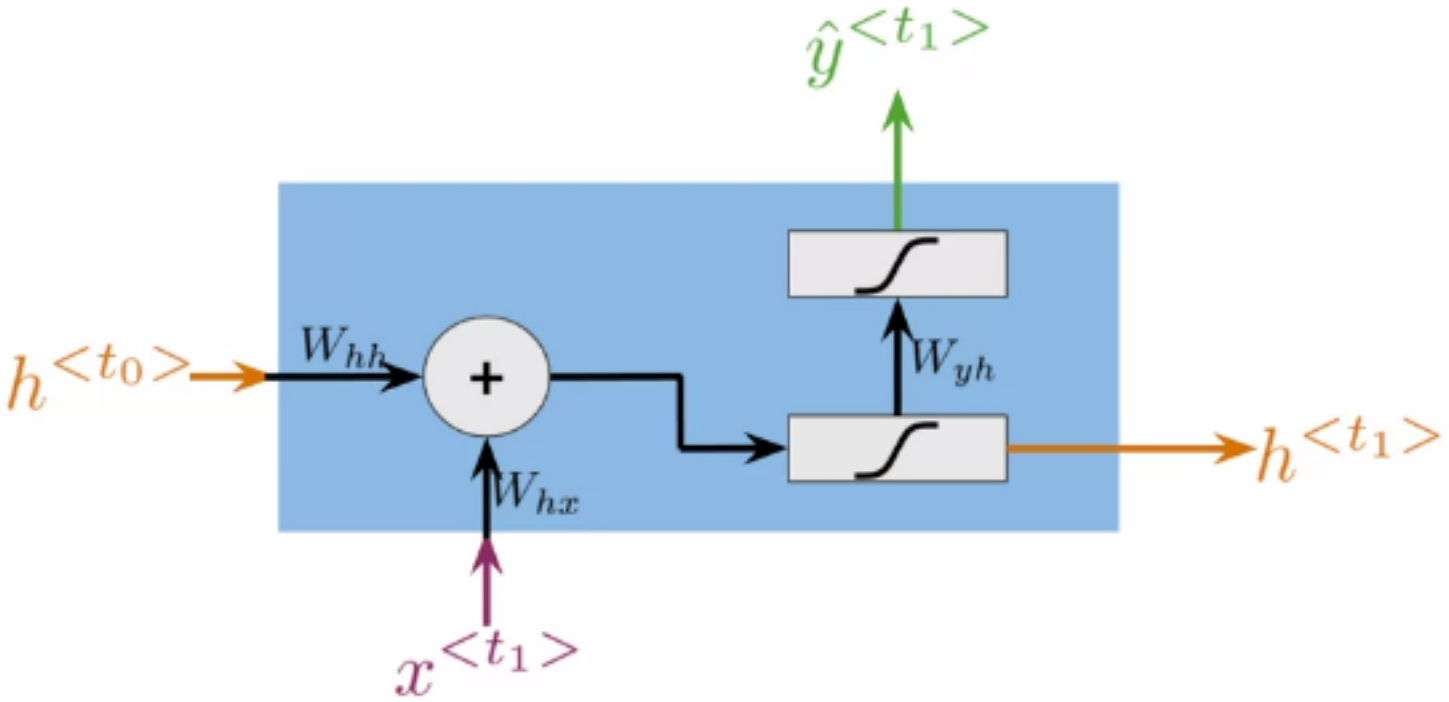




$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$

A Vanilla RNN





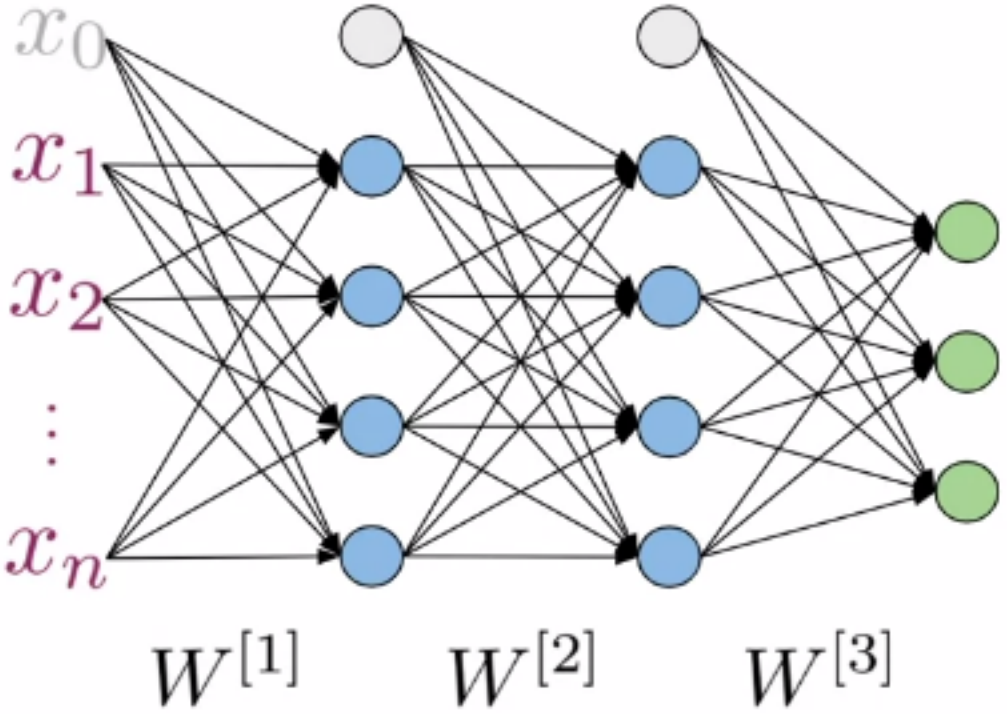
$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

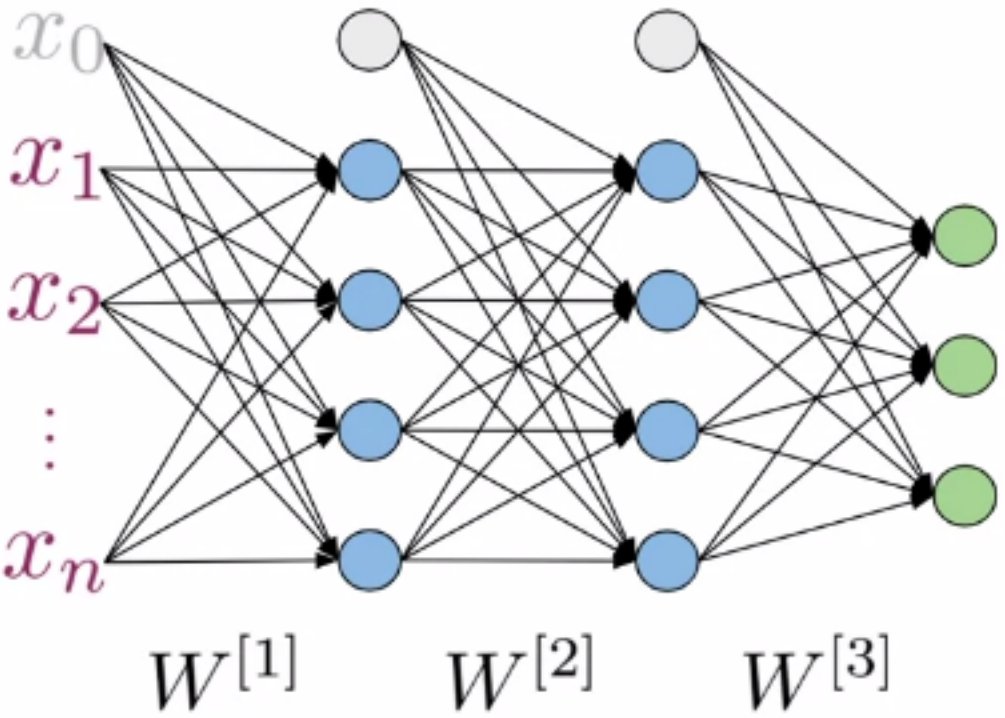
Summary

- Hidden states propagate information through time
- Basic recurrent units have two inputs at each time: $h^{<t-1>}$, $x^{<t>}$

Cross Entropy Loss



Cross Entropy Loss

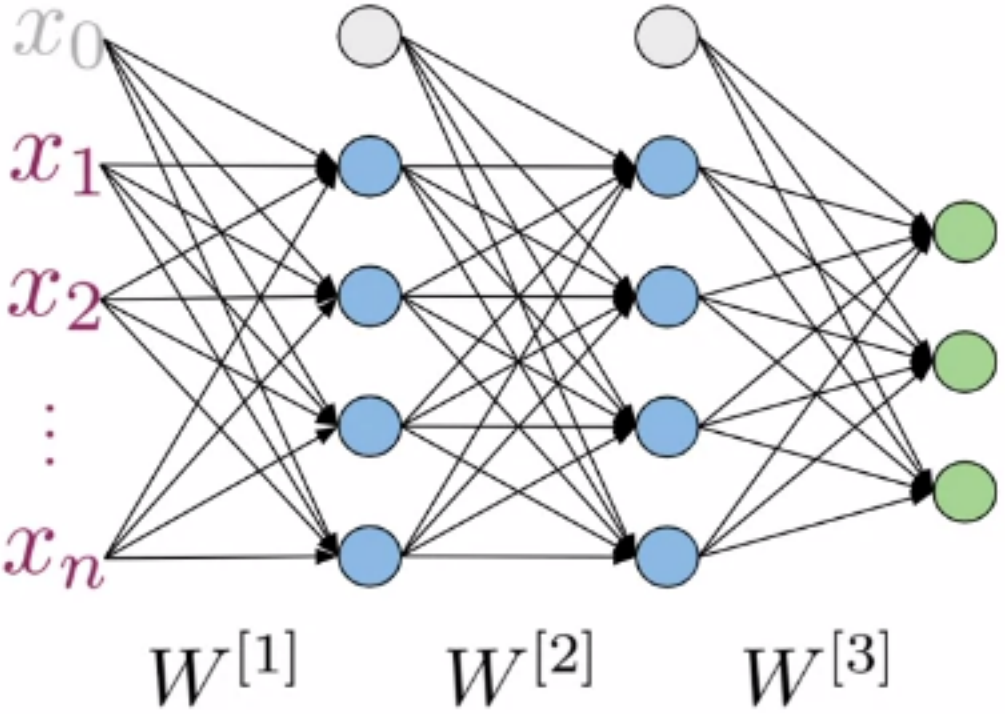


K - classes or possibilities

$$J = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$

Looking at a single example (x, y)

Cross Entropy Loss



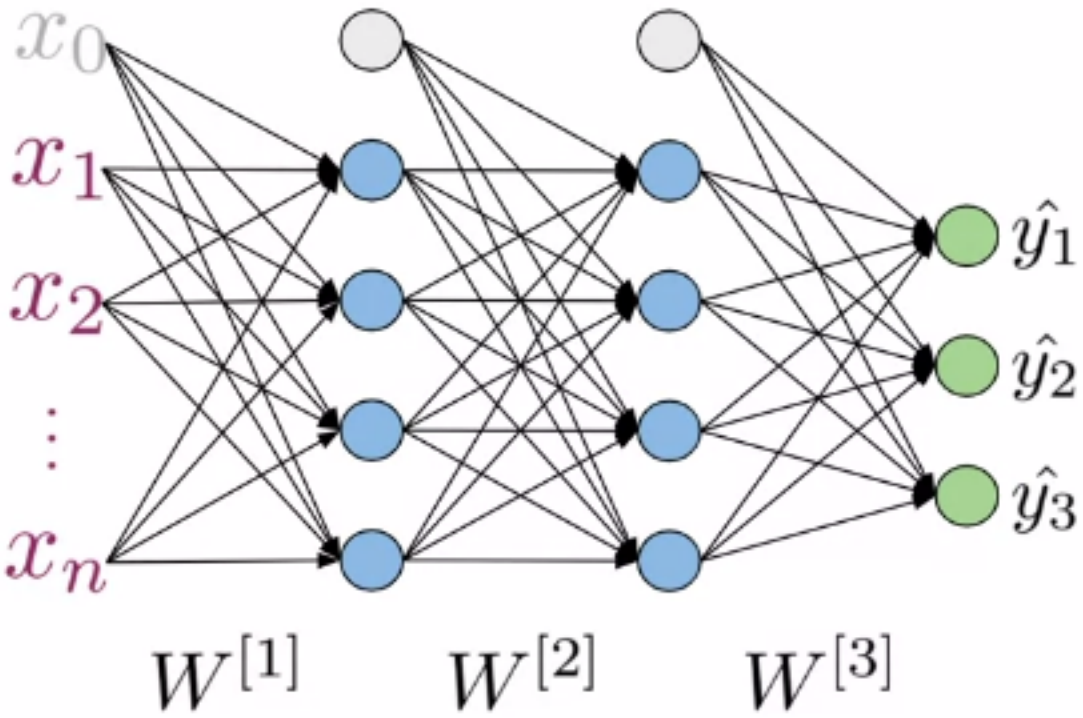
K - classes or possibilities

Either 0 or 1

$$J = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$

Looking at a single example (x, y)

Cross Entropy Loss



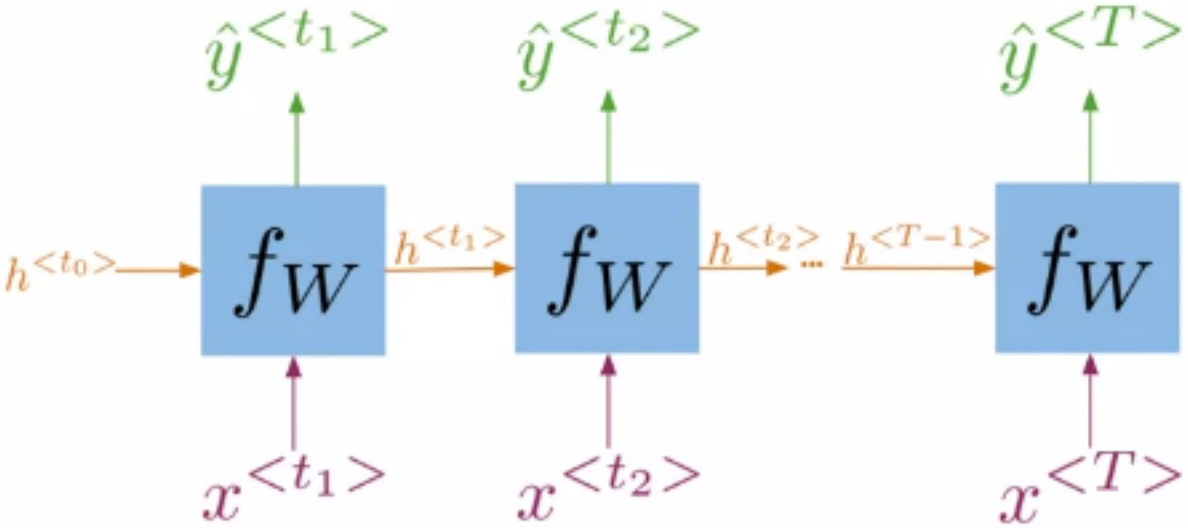
K - classes or possibilities

Either 0 or 1

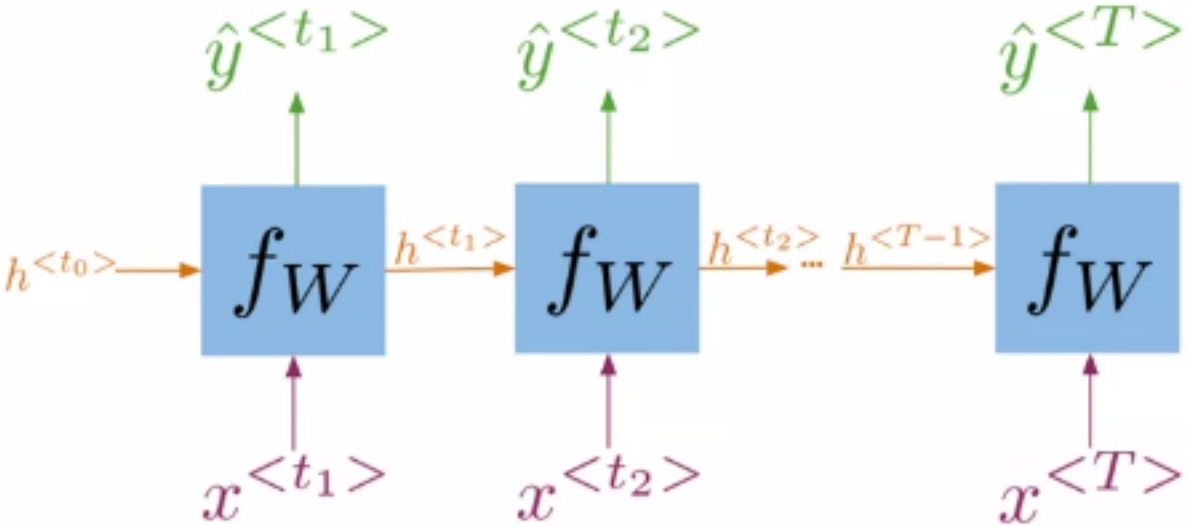
$$J = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$

Looking at a single example (x, y)

Cross Entropy Loss



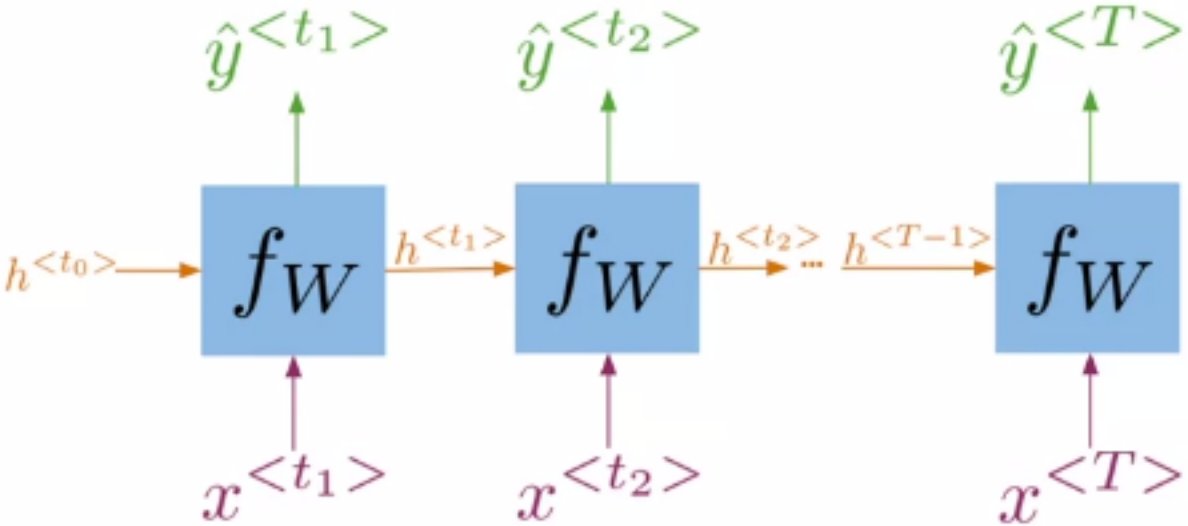
Cross Entropy Loss



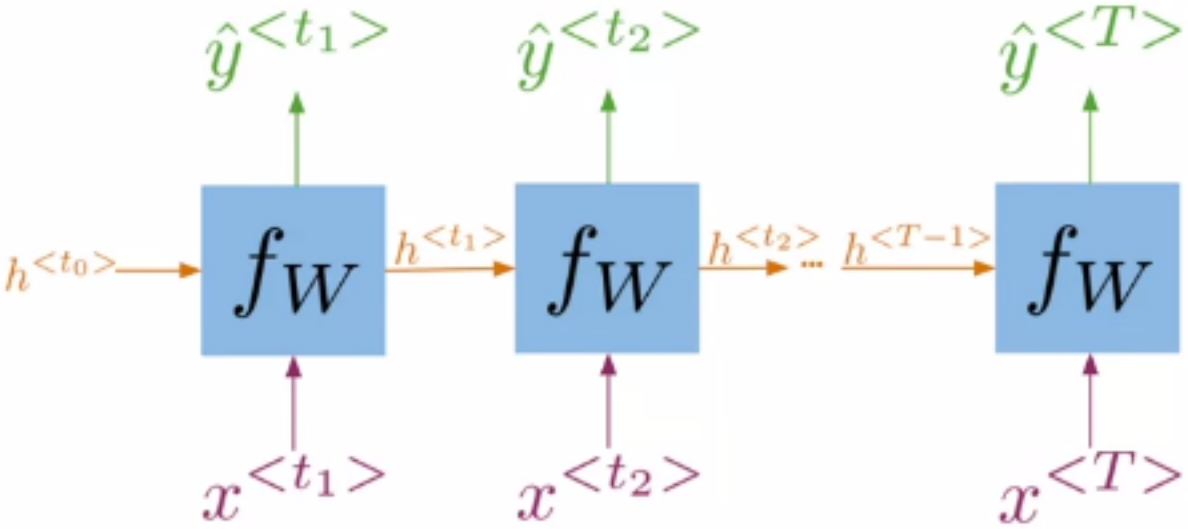
Cross Entropy Loss

$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



Cross Entropy Loss

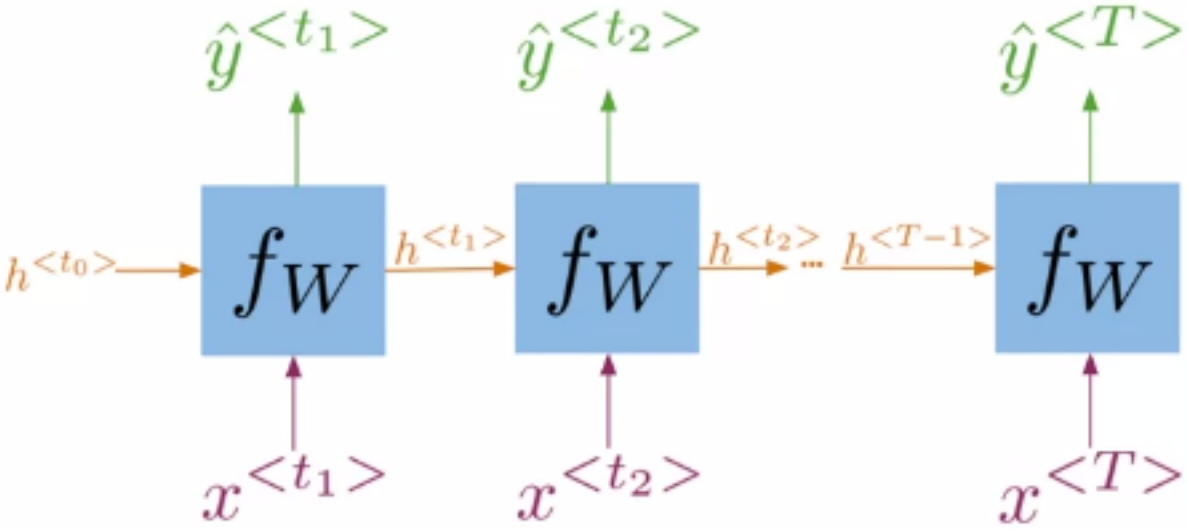


$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

Cross Entropy Loss



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

Average with respect to time

Summary

For RNNs the loss function is just an average through time!

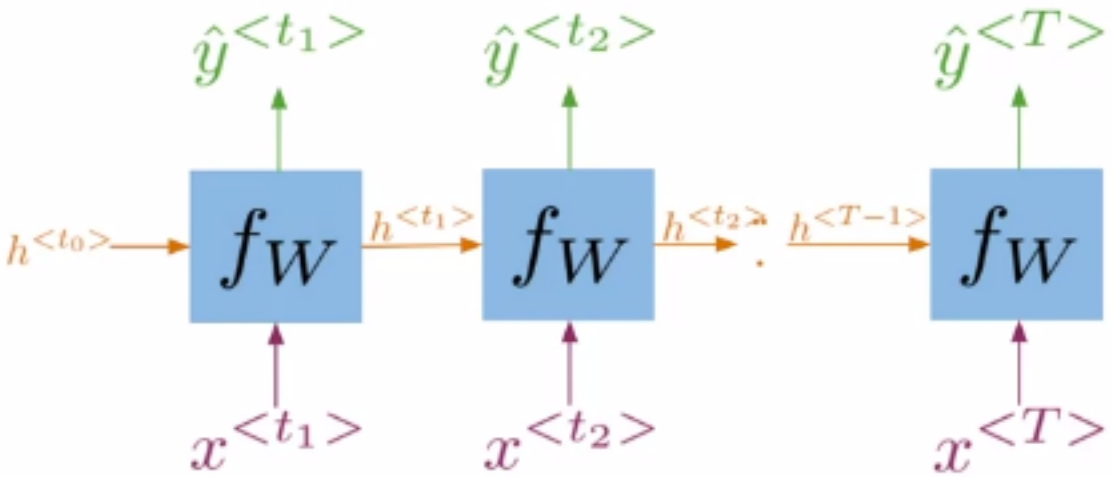
Outline

- scan() function in tensorflow
- Computation of forward propagation using abstractions

tf.scan() function

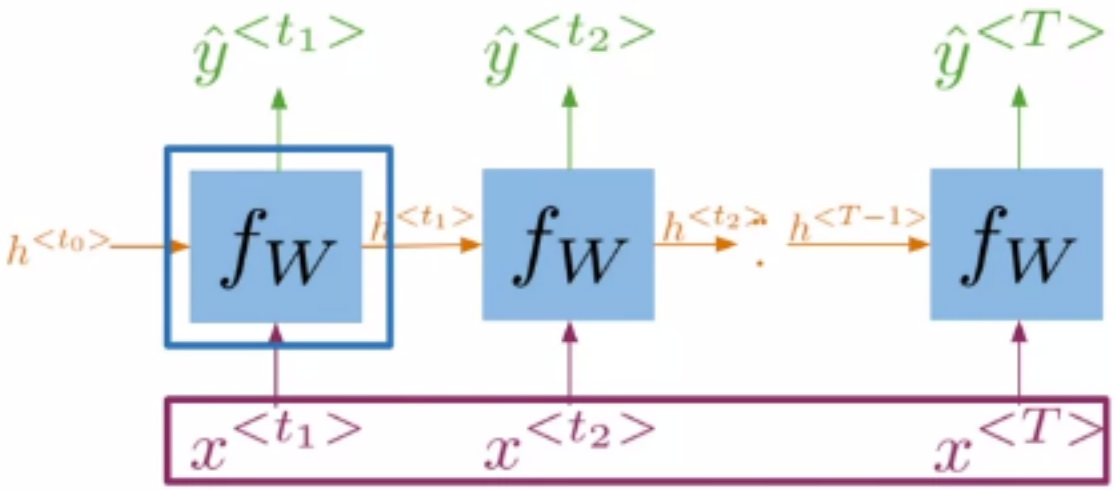
```
def scan(fn, elems, initializer=None, ...):
```

tf.scan() function



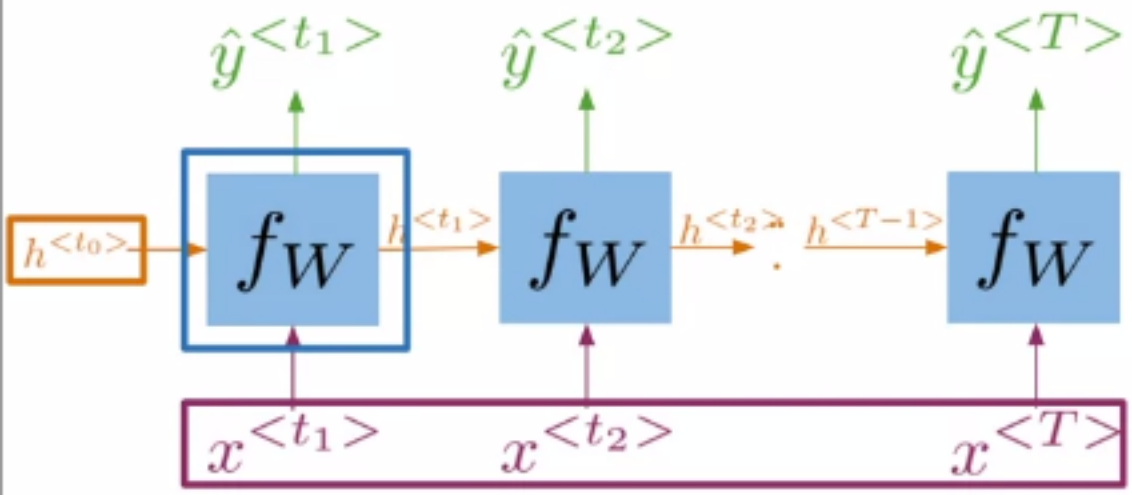

```
def scan(fn, elems, initializer=None, ...):
```

tf.scan() function



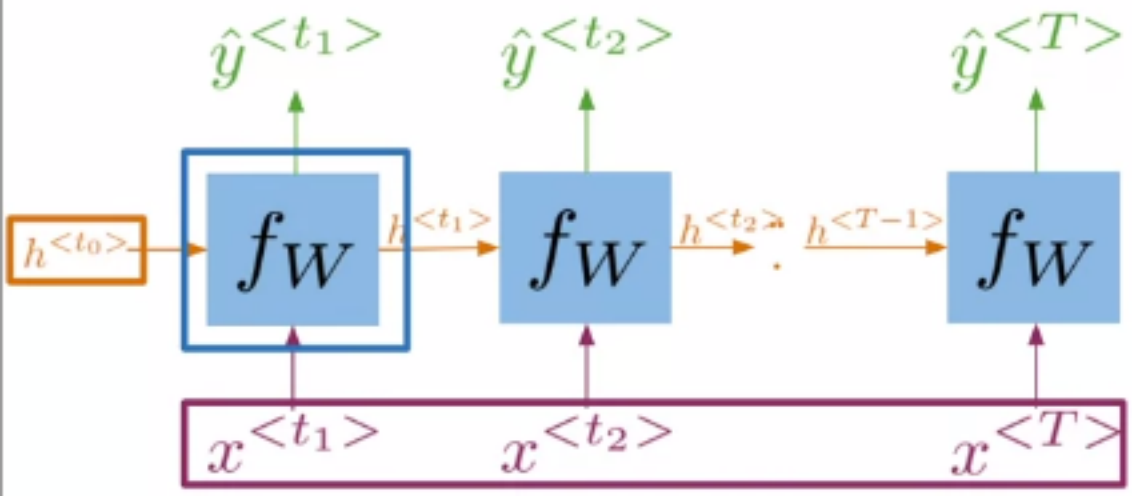

```
def scan(fn, elems, initializer=None, ...):
```

tf.scan() function



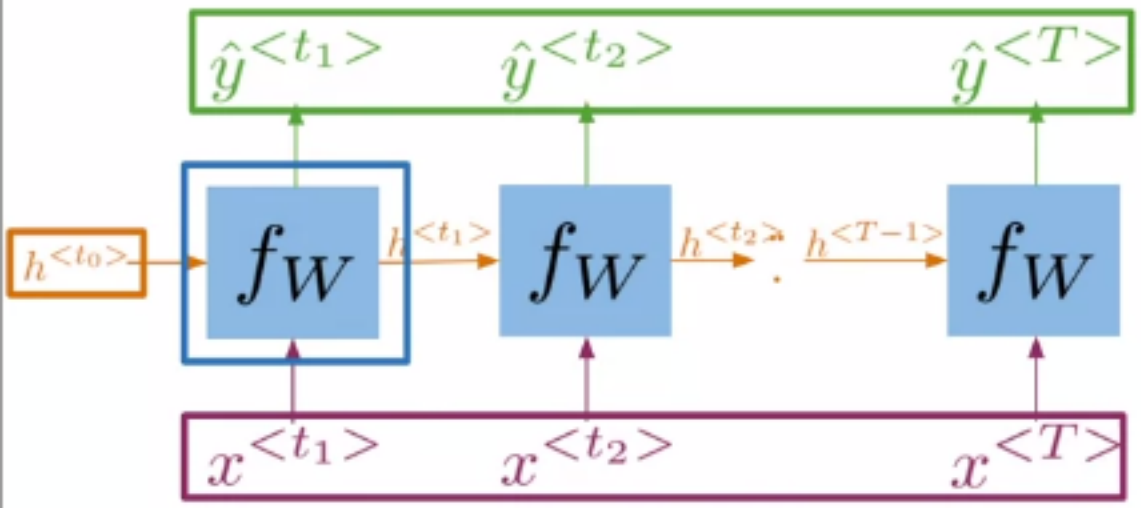

```
def scan(fn, elems, initializer=None, ...):
```

tf.scan() function



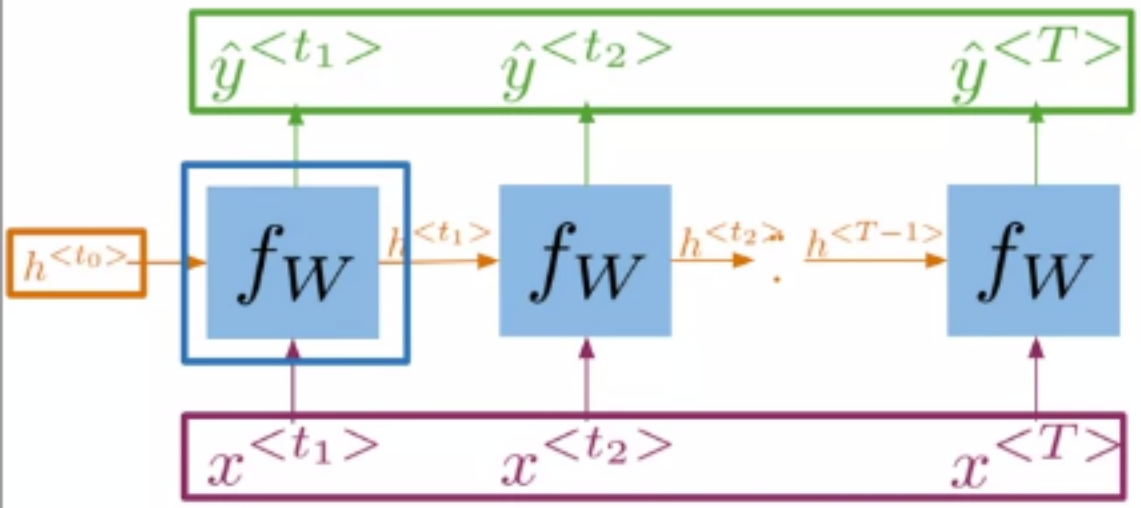

```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []
```

tf.scan() function



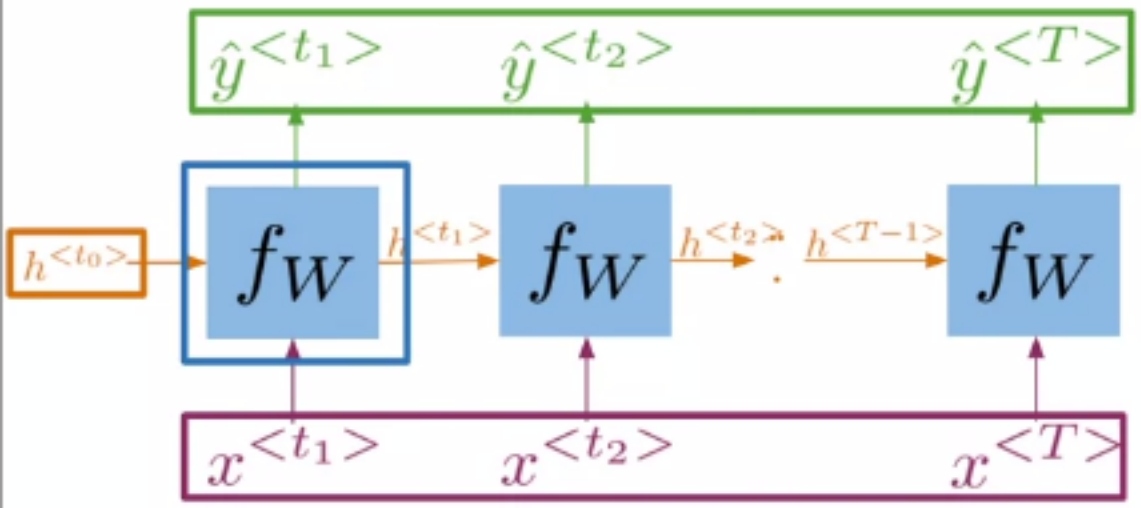

```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []
```

tf.scan() function



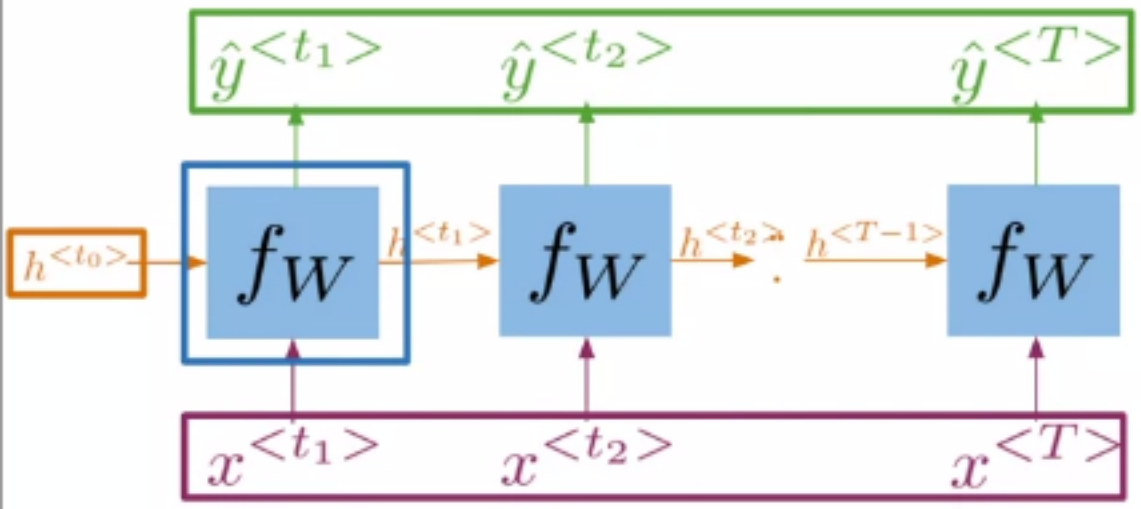

```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []

    for x in elems:
```

tf.scan() function




```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []

    for x in elems:
        y, cur_value = fn(x, cur_value)
        ys.append(y)
```

tf.scan() function




```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []

    for x in elems:
        y, cur_value = fn(x, cur_value)
        ys.append(y)

    return ys, cur_value
```

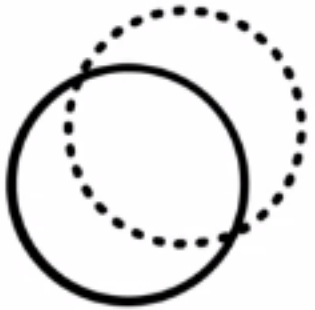
tf.scan() function


```
def scan(fn, elems, initializer=None, ...):

    cur_value = initializer
    ys = []

    for x in elems:
        y, cur_value = fn(x, cur_value)
        ys.append(y)

    return ys, cur_value
```

Frameworks like Tensorflow need this type of abstraction

Summary

- Frameworks require abstractions
- tf.scan() mimics RNNs



Outline

- Gated recurrent unit (GRU) structure
- Comparison between GRUs and vanilla RNNs

Outline

- Gated recurrent unit (GRU) structure
- Comparison between GRUs and vanilla RNNs

Gated Recurrent Units

"Ants are really interesting. _____ are everywhere."

Gated Recurrent Units

"Ants are really interesting. _____ are everywhere."

Gated Recurrent Units

"Ants are really interesting. They are everywhere."

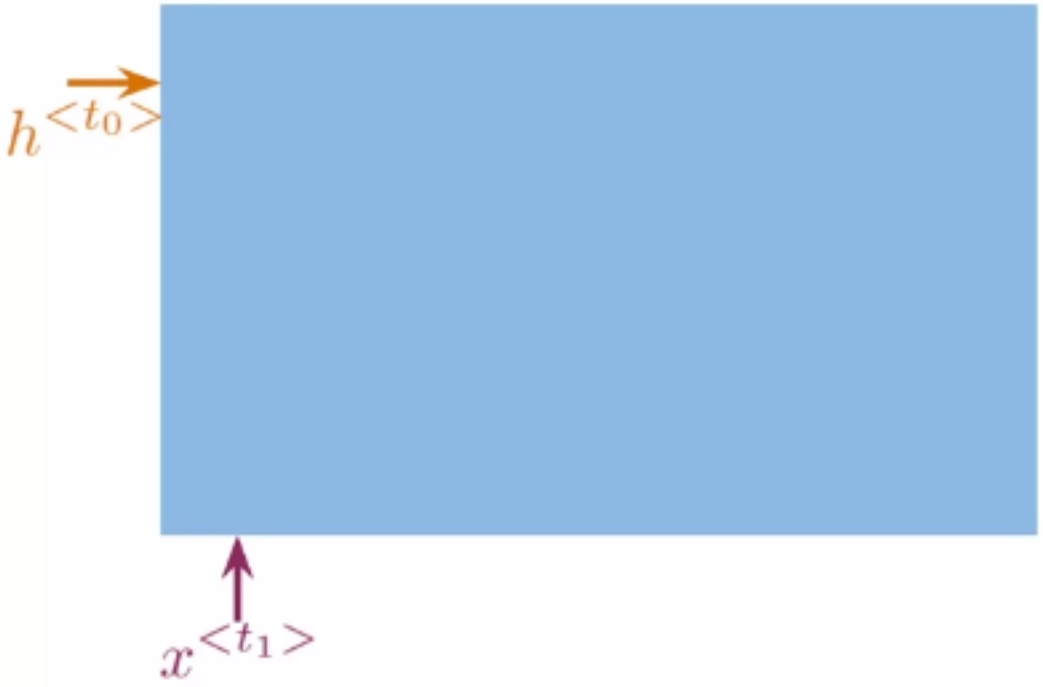
Gated Recurrent Units

Gated Recurrent Units

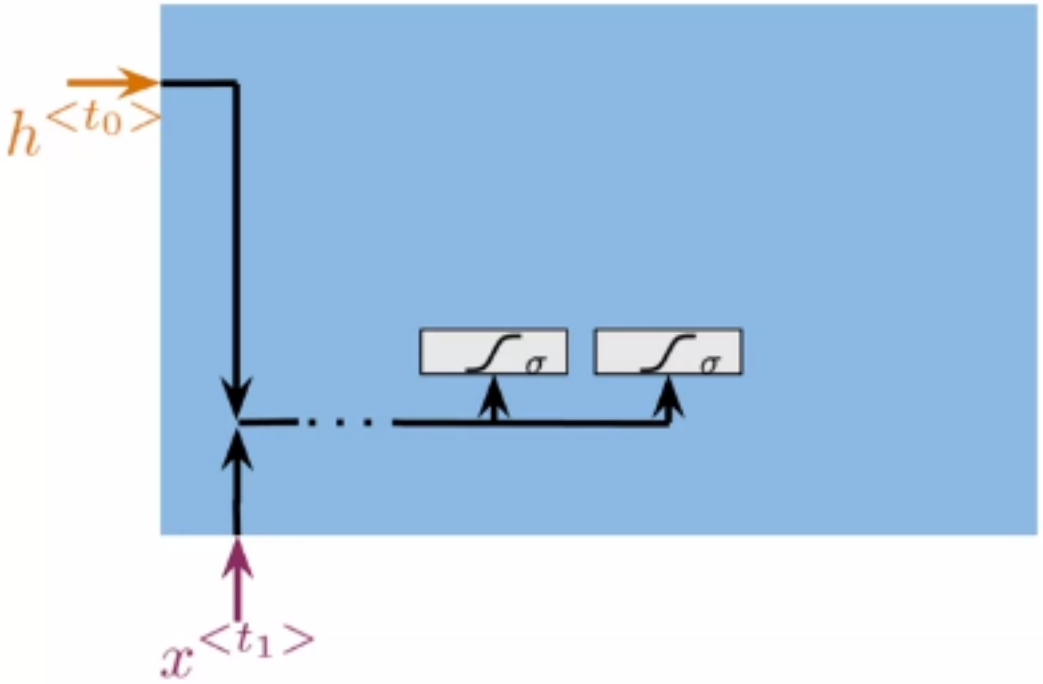
Relevance and update gates to remember important prior information

Gated Recurrent Unit

Gated Recurrent Unit

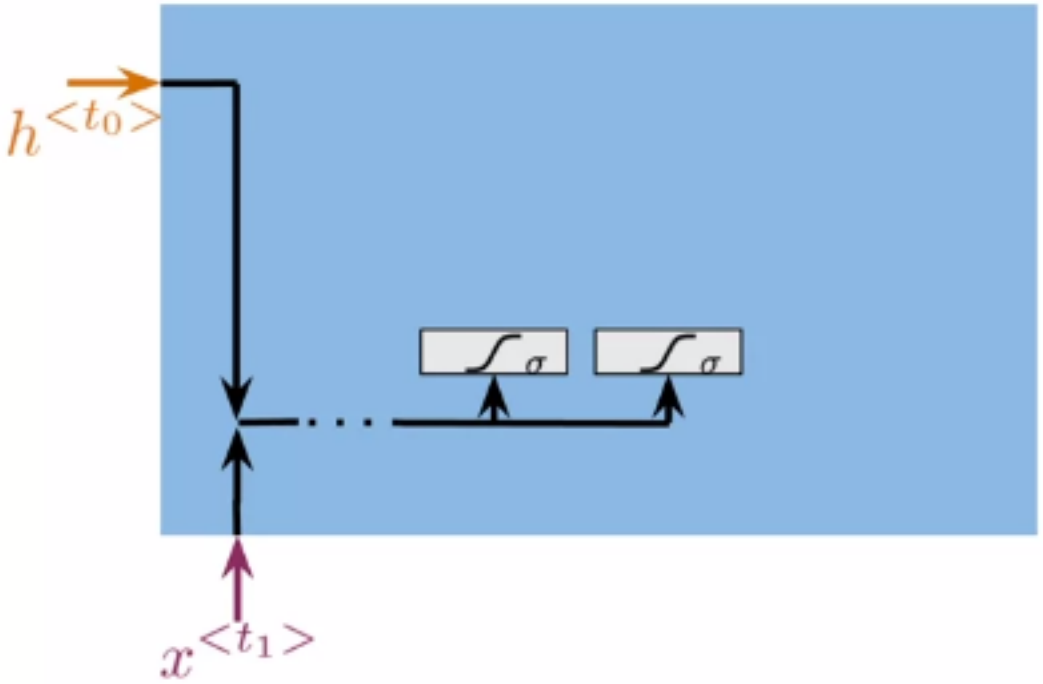


Gated Recurrent Unit



$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

Gated Recurrent Unit

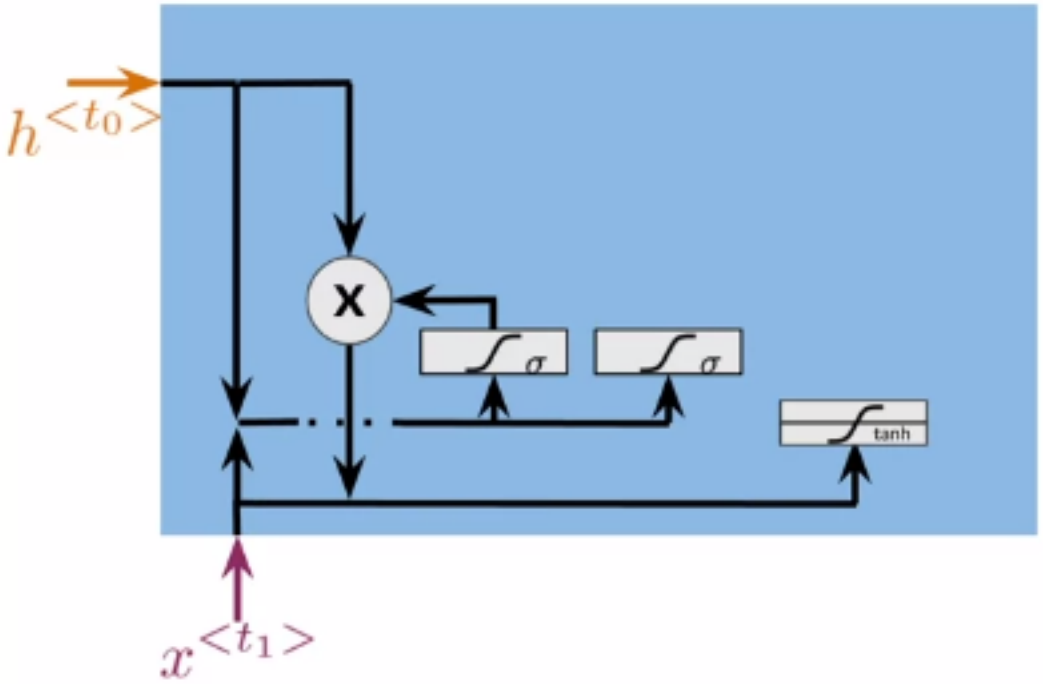


Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

Gated Recurrent Unit

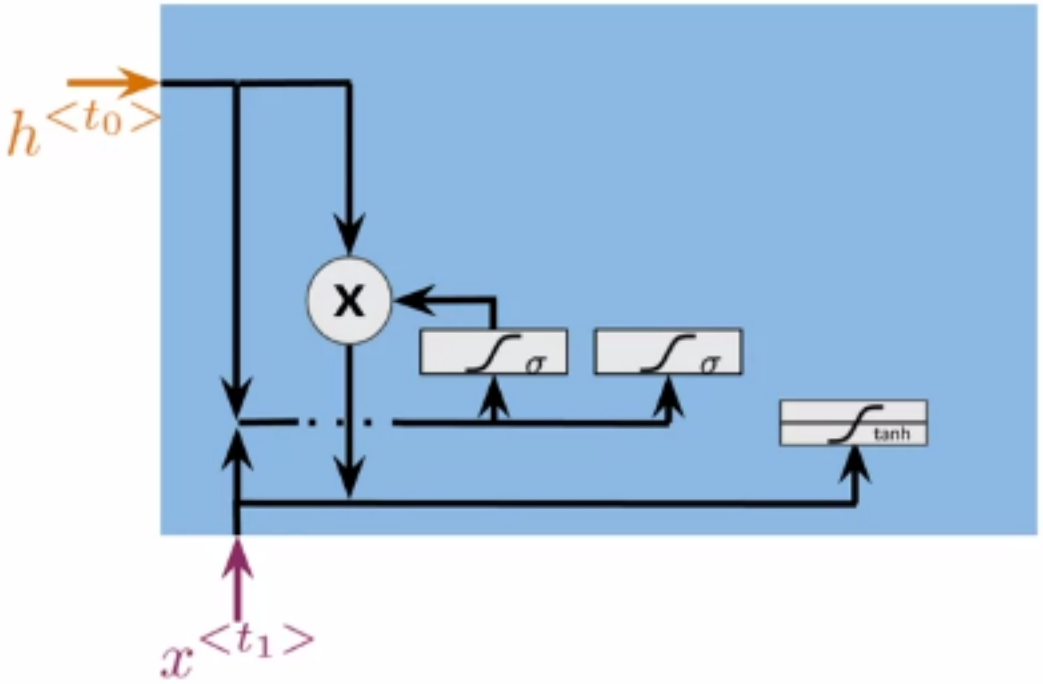


Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Gated Recurrent Unit

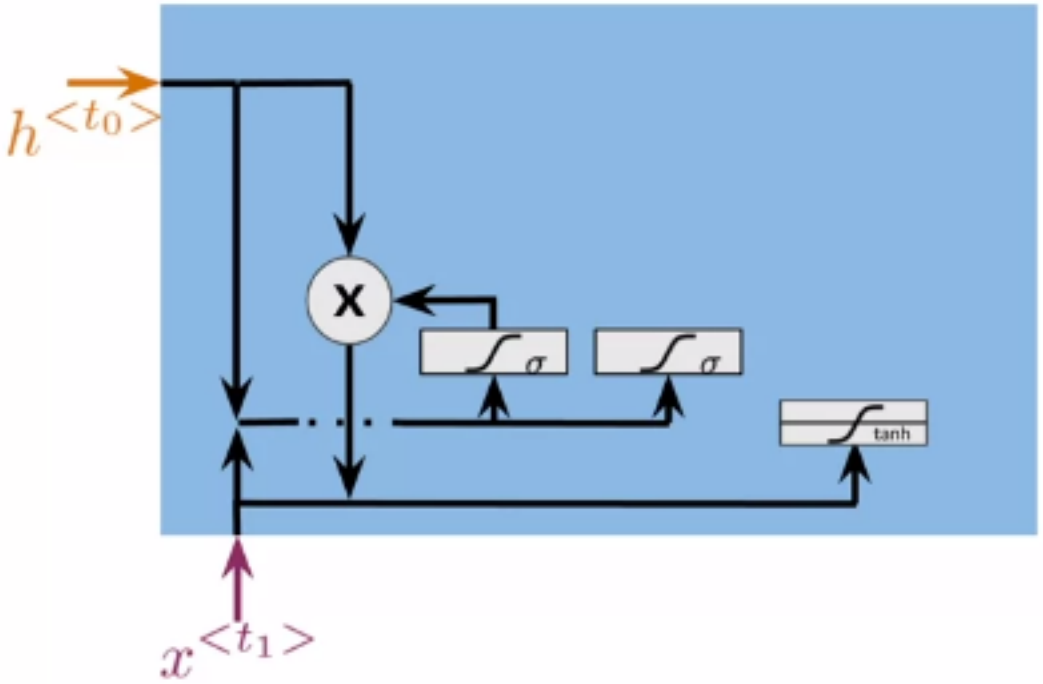


Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Gated Recurrent Unit



Gates to keep/update relevant information in the hidden state

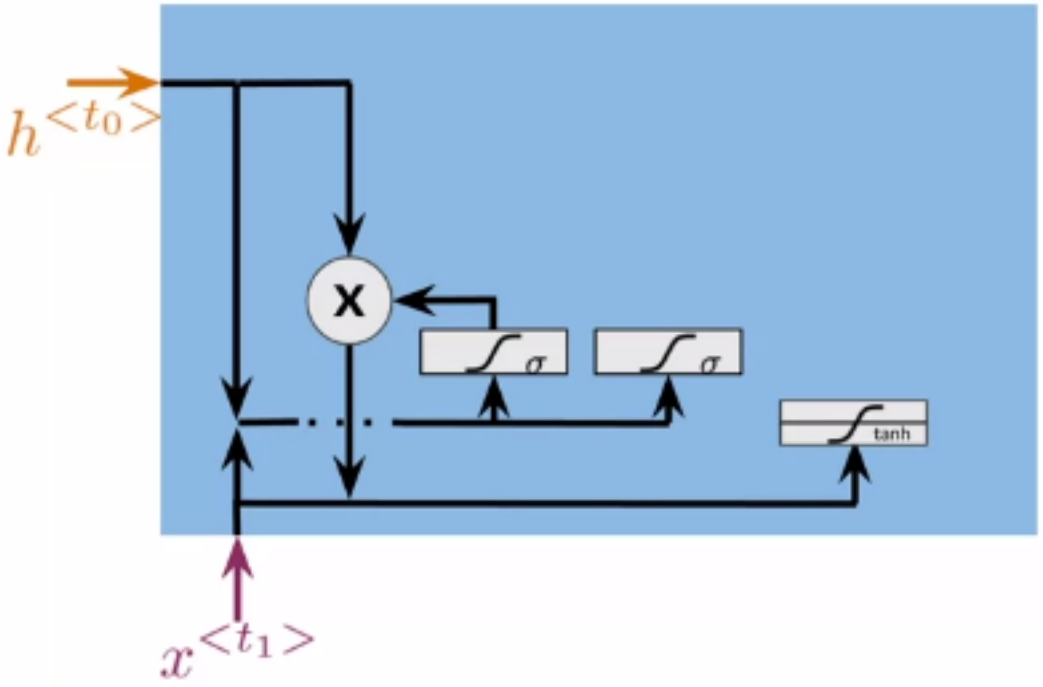
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Hidden state candidate

Gated Recurrent Unit



Gates to keep/update relevant information in the hidden state

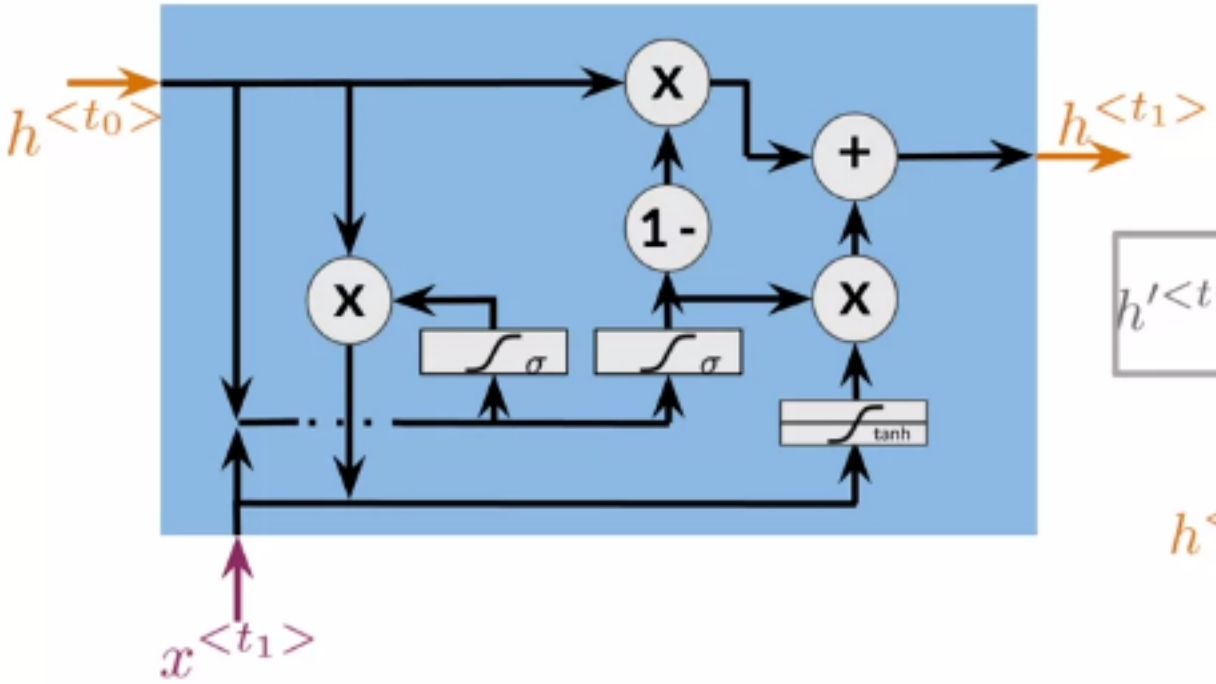
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Hidden state candidate

Gated Recurrent Unit



Gates to keep/update relevant information in the hidden state

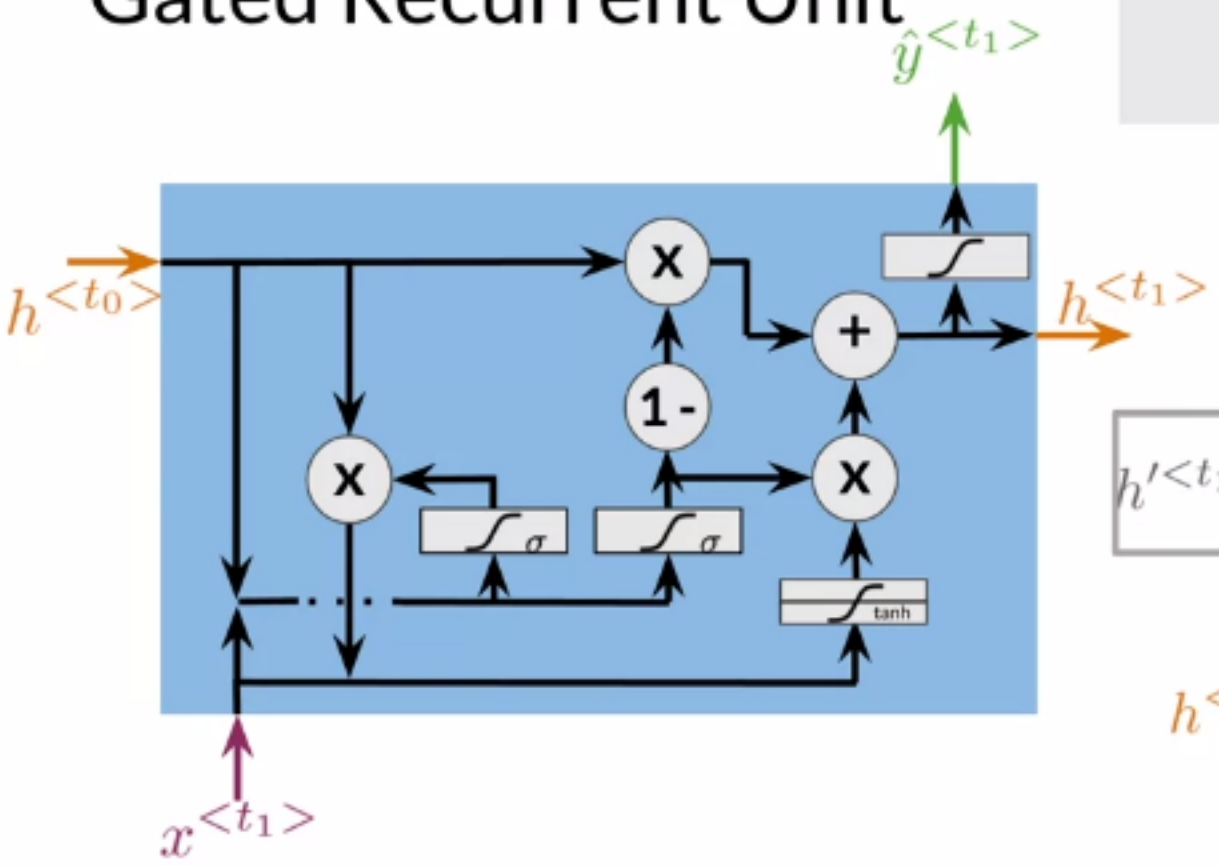
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$

Hidden state candidate

$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$

Gated Recurrent Unit



Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$

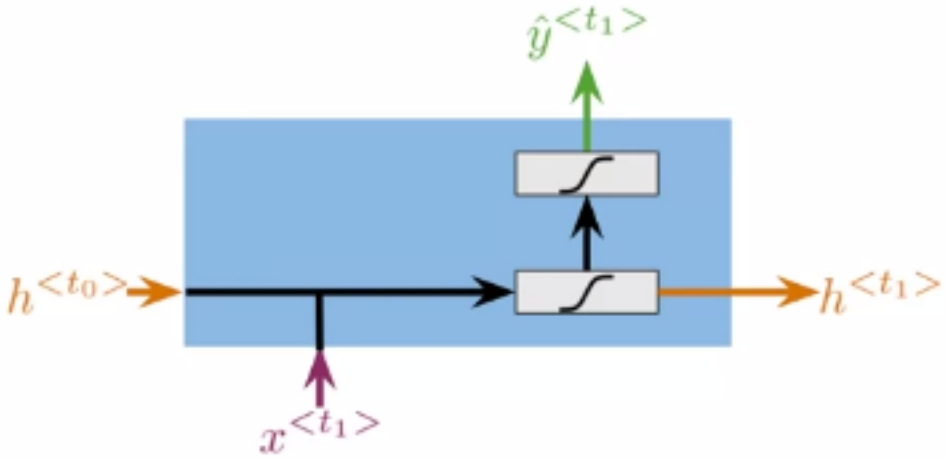
Hidden state candidate

$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$

$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

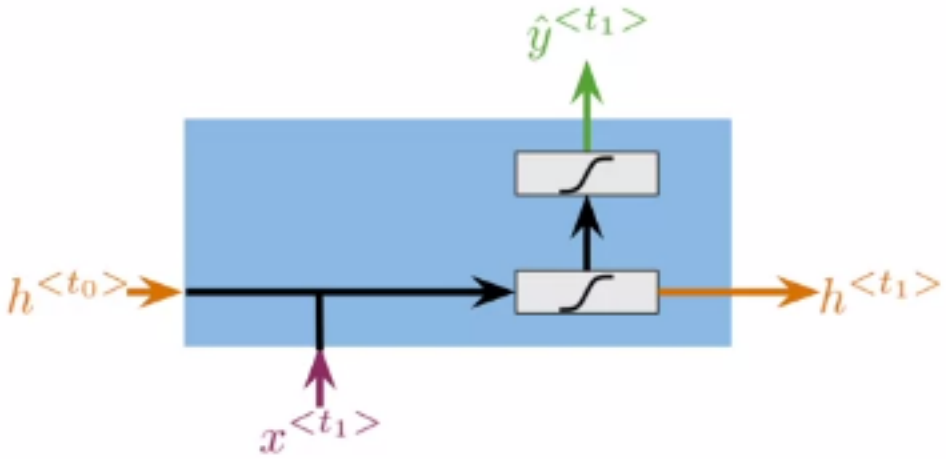
Vanilla RNN vs GRUs

Vanilla RNN vs GRUs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

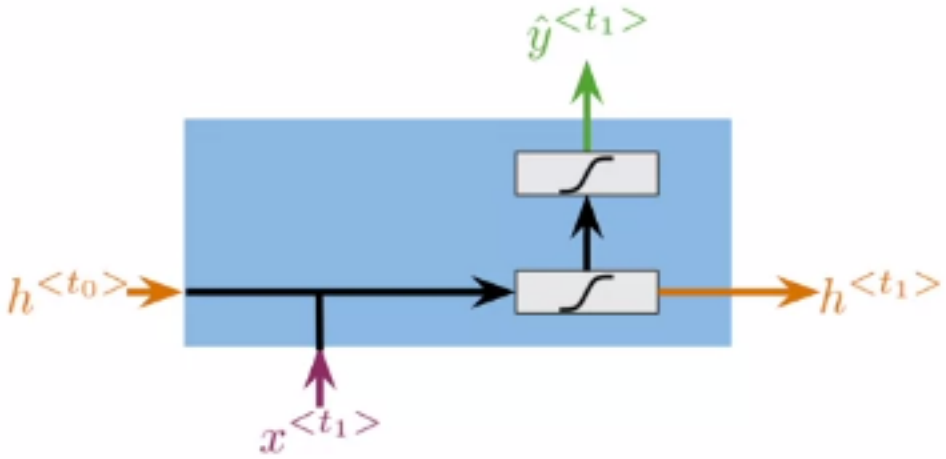
Vanilla RNN vs GRUs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

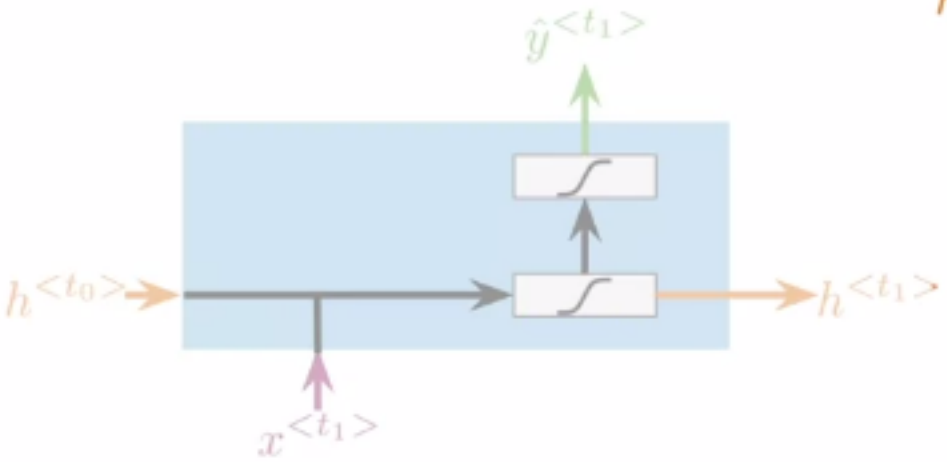
Vanilla RNN vs GRUs

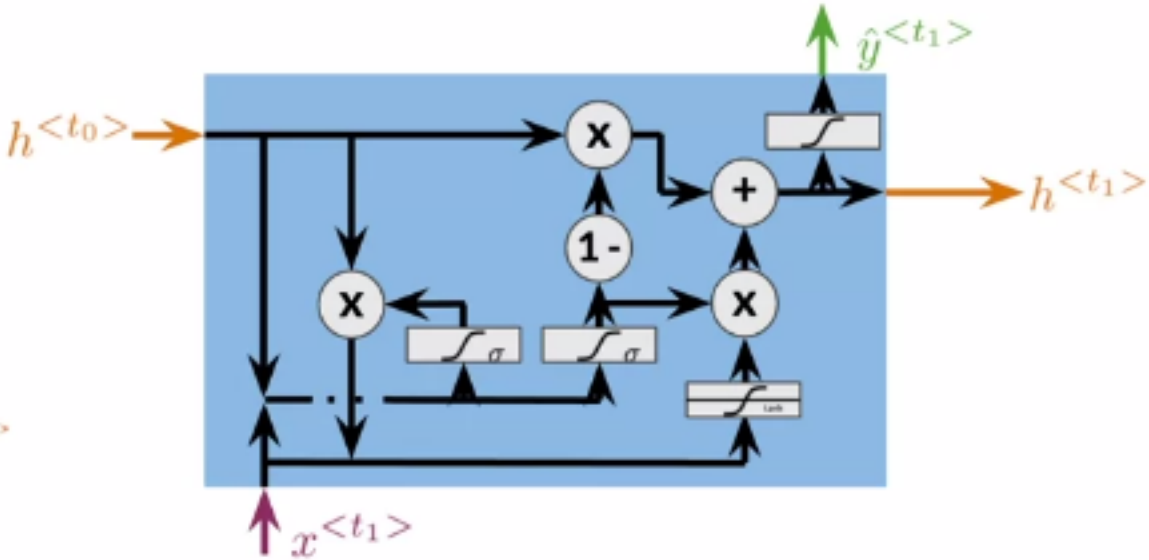


$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

Vanilla RNN vs GRUs

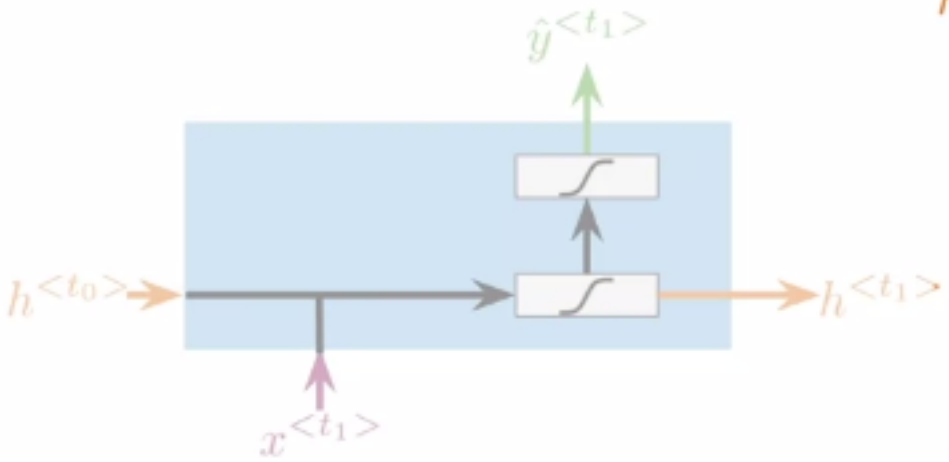




$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

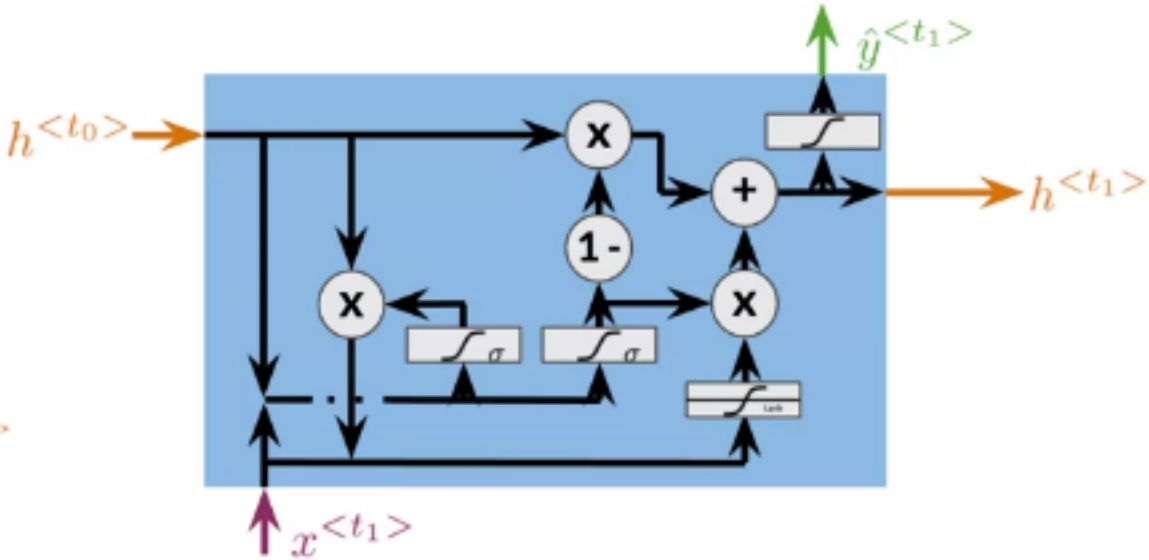
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

Vanilla RNN vs GRUs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

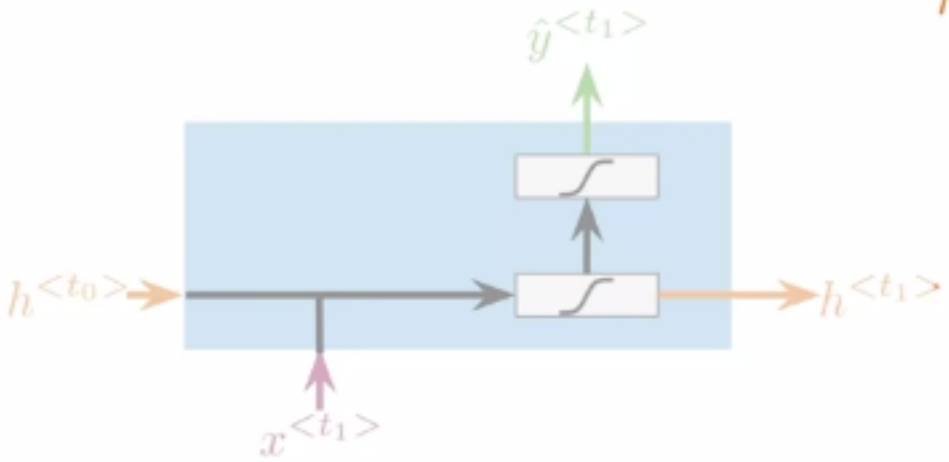
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



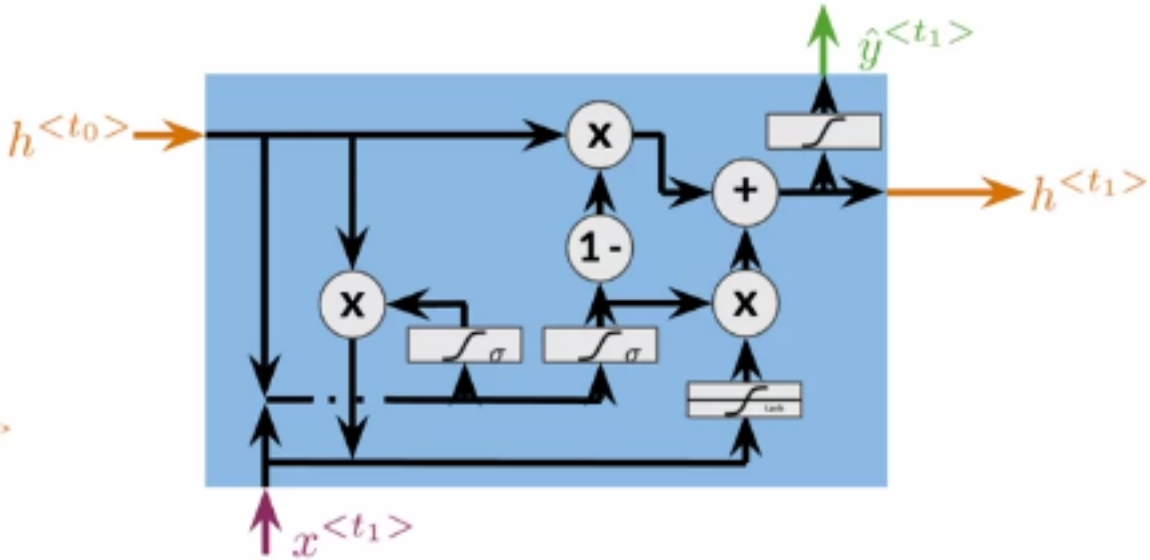
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

Vanilla RNN vs GRUs

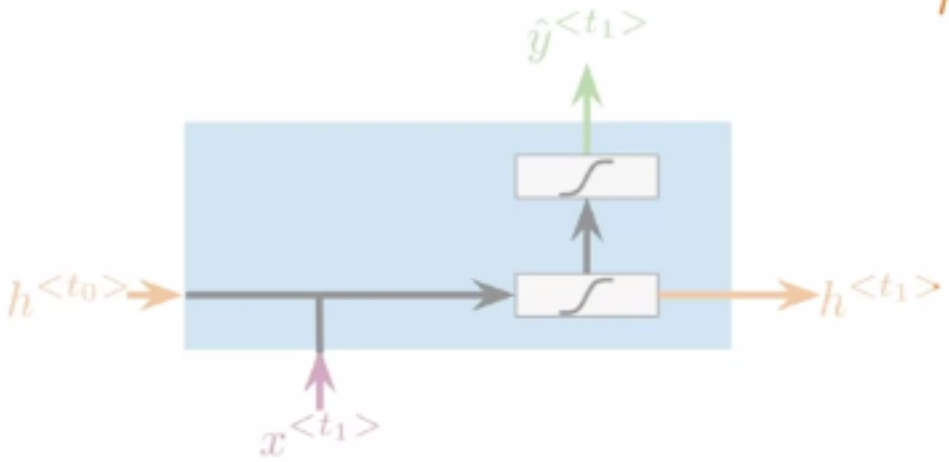


$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



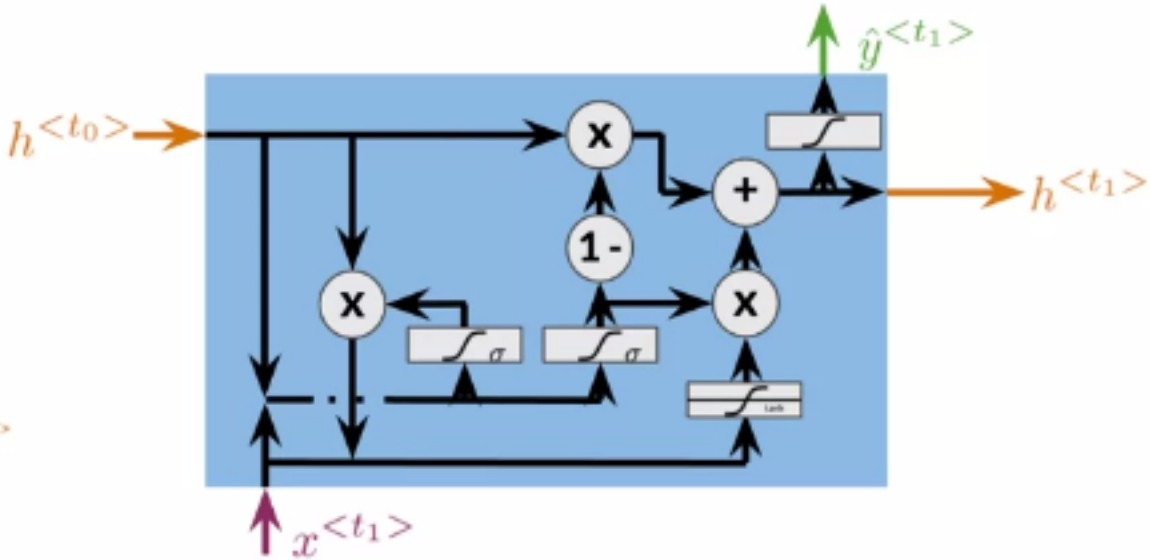
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$

Vanilla RNN vs GRUs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



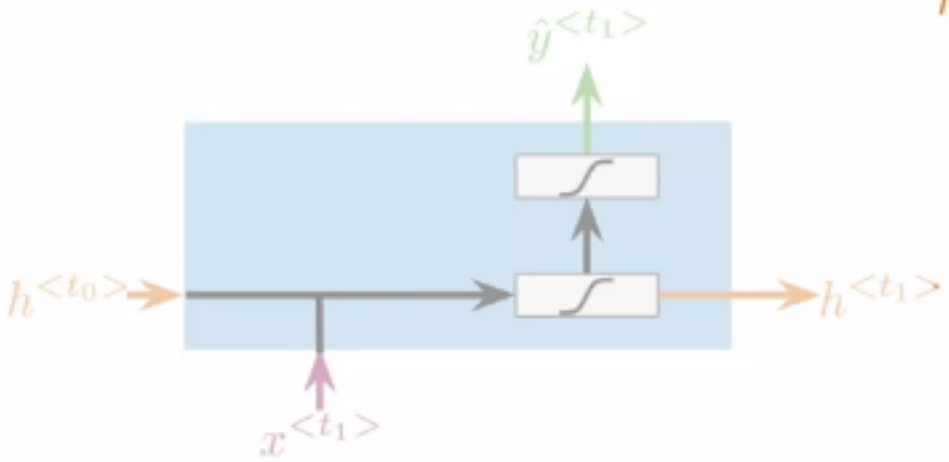
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

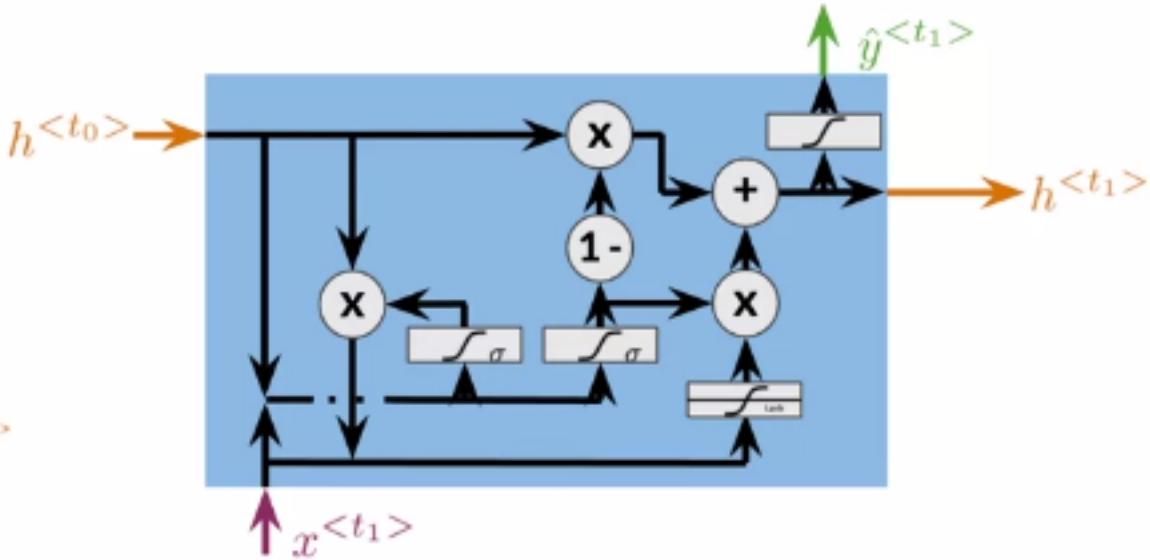
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$

Vanilla RNN vs GRUs

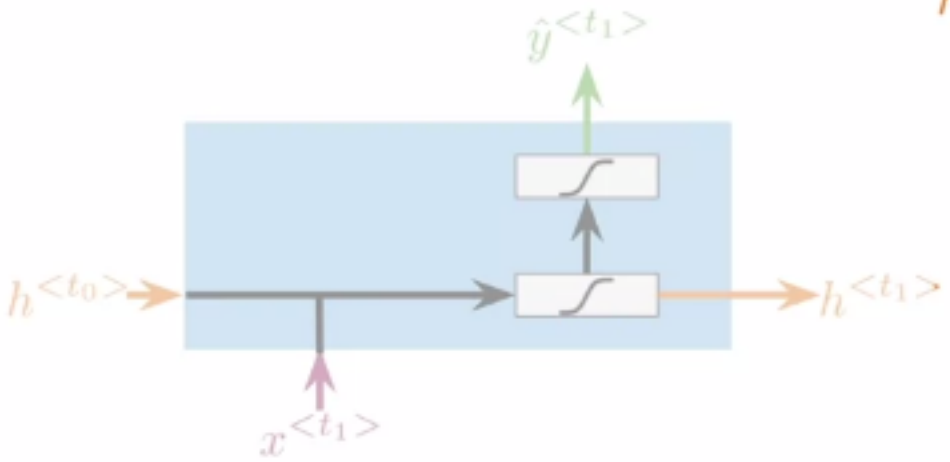


$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

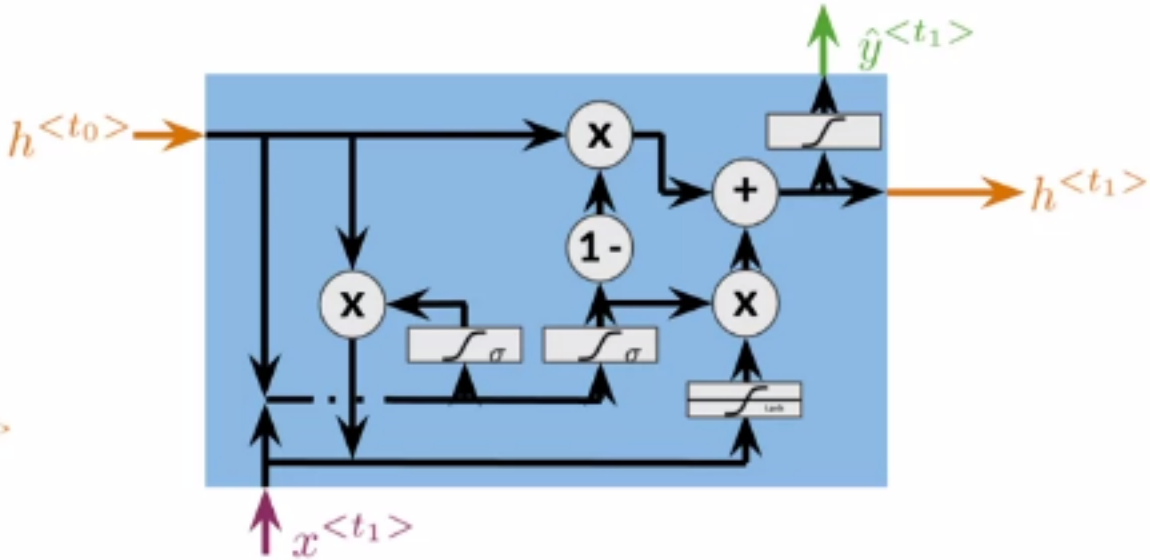


$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$
$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Vanilla RNN vs GRUs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



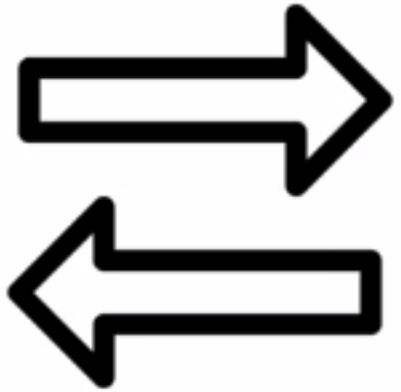
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$
$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Summary

- GRUs “decide” how to update the hidden state
- GRUs help preserve important information

Outline

- How bidirectional RNNs propagate information
- Forward propagation in deep RNNs



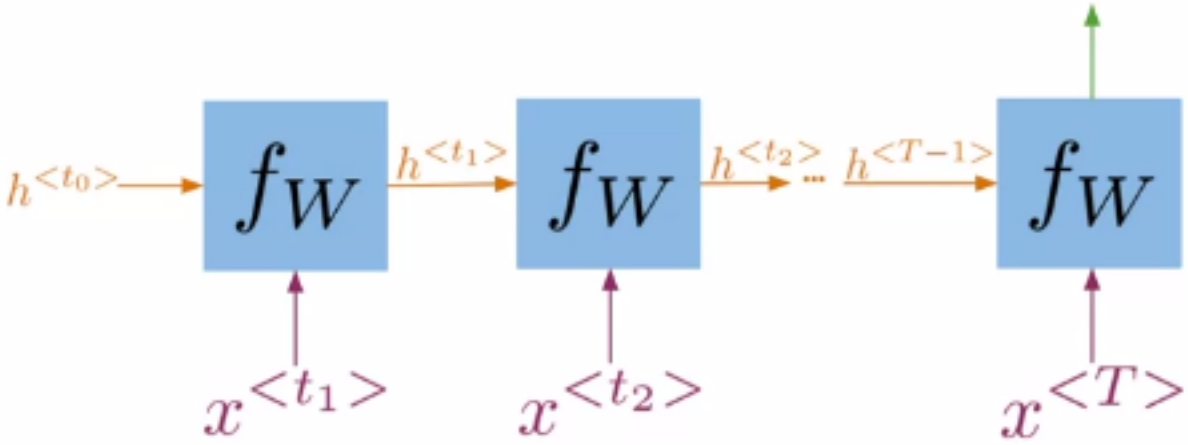
Bi-directional RNNs

Bi-directional RNNs

I was trying really hard to get a hold of _____. **Louise,** finally answered when I was about to give up.

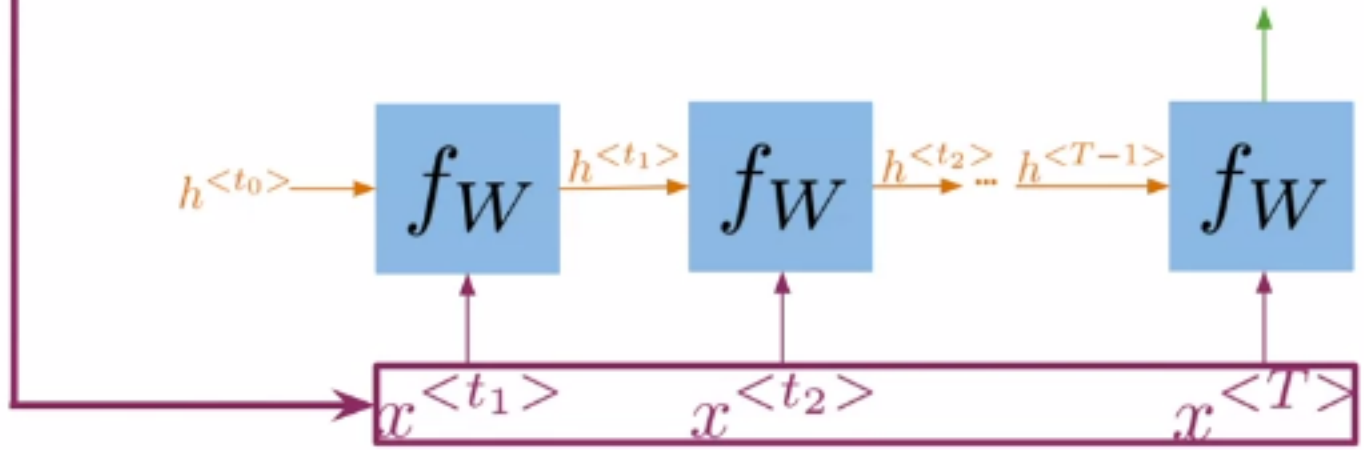
Bi-directional RNNs

I was trying really hard to get a hold of _____. **Louise,** finally answered when I was about to give up.



Bi-directional RNNs

I was trying really hard to get a hold of _____. **Louise,** finally answered when I was about to give up.



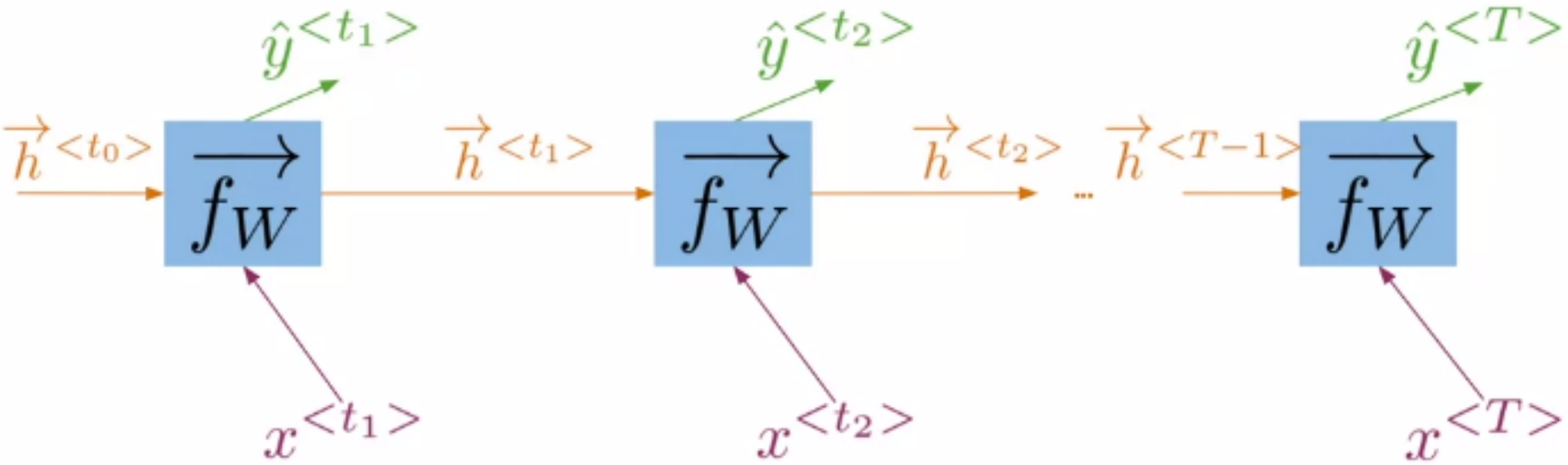
Bi-directional RNNs

Bi-directional RNNs

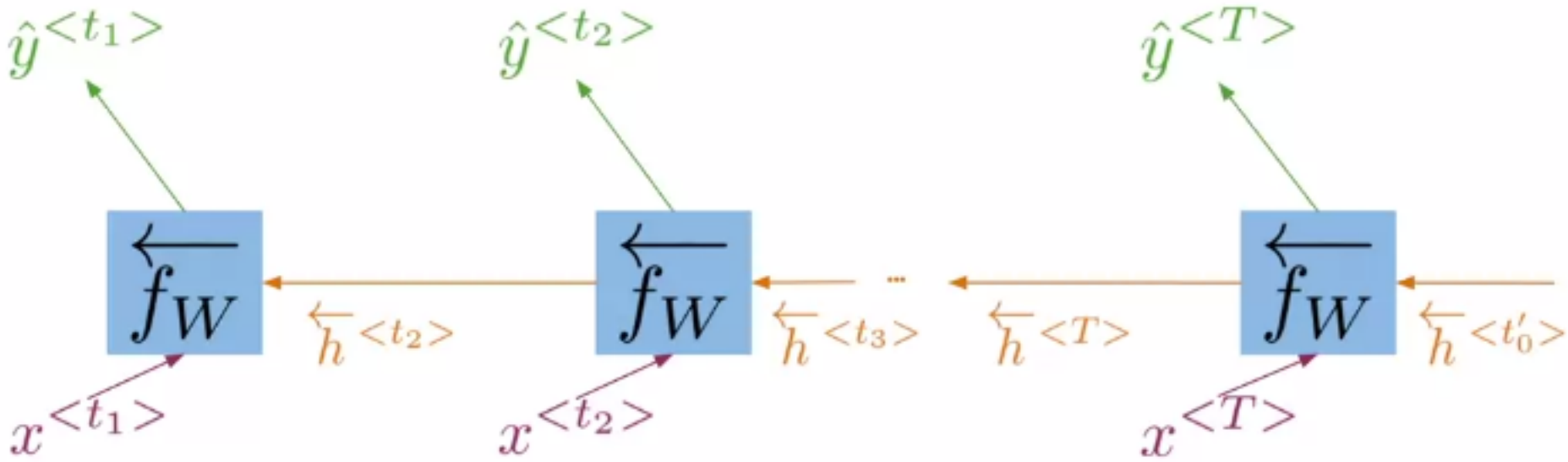
$x^{<t_1>}$ $x^{<t_2>}$ $x^{<T>}$

Bi-directional RNNs

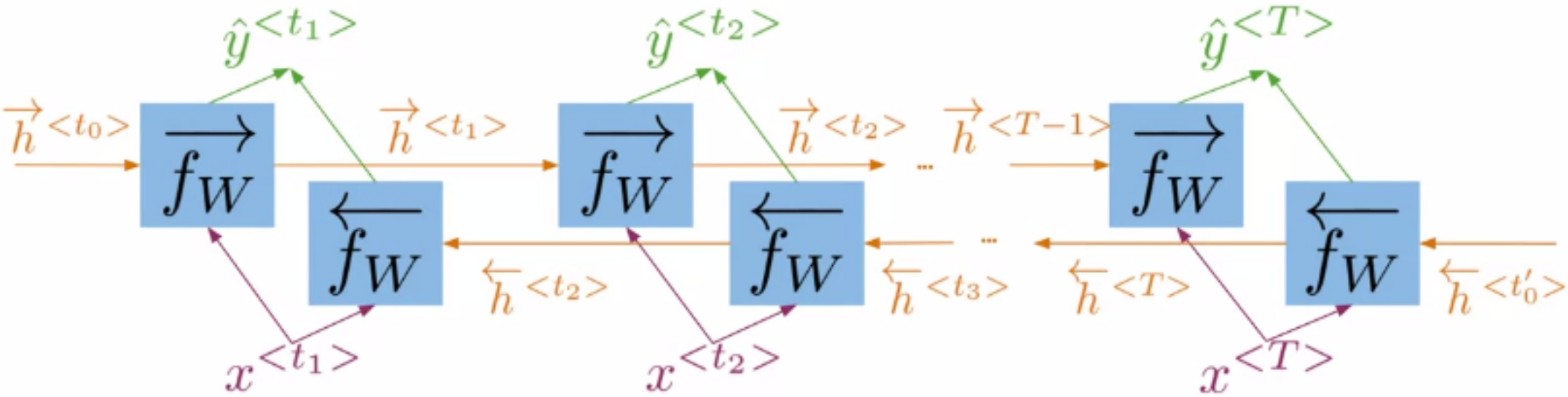
Bi-directional RNNs



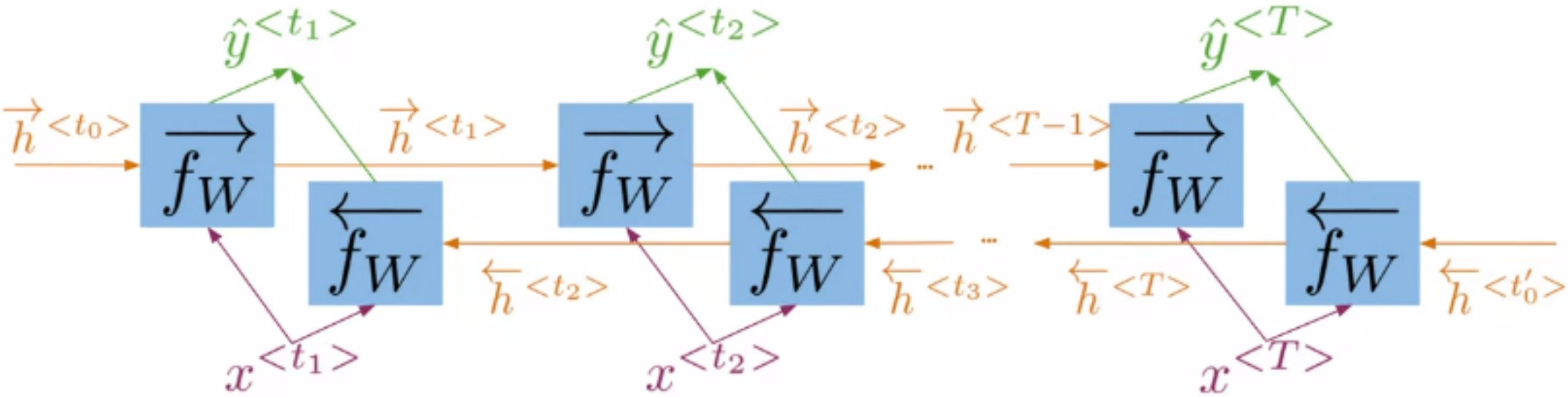
Bi-directional RNNs



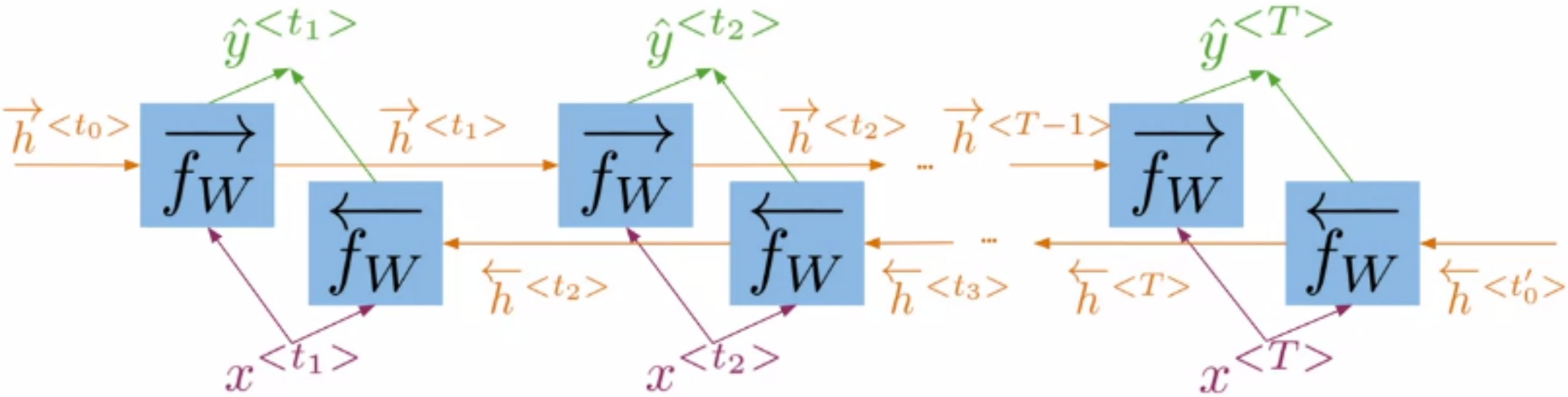
Bi-directional RNNs



Bi-directional RNNs

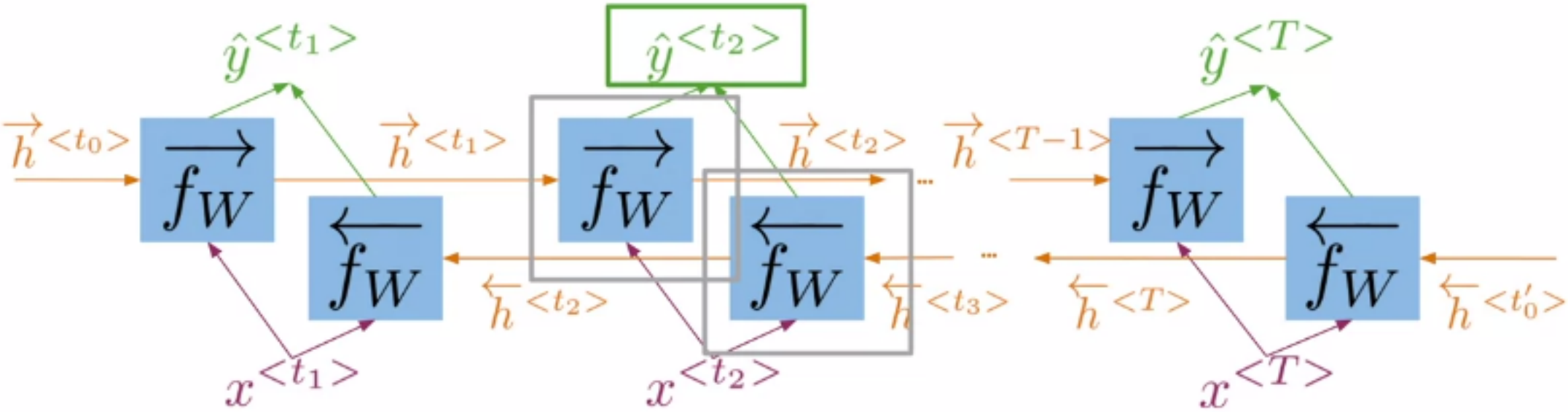


Bi-directional RNNs



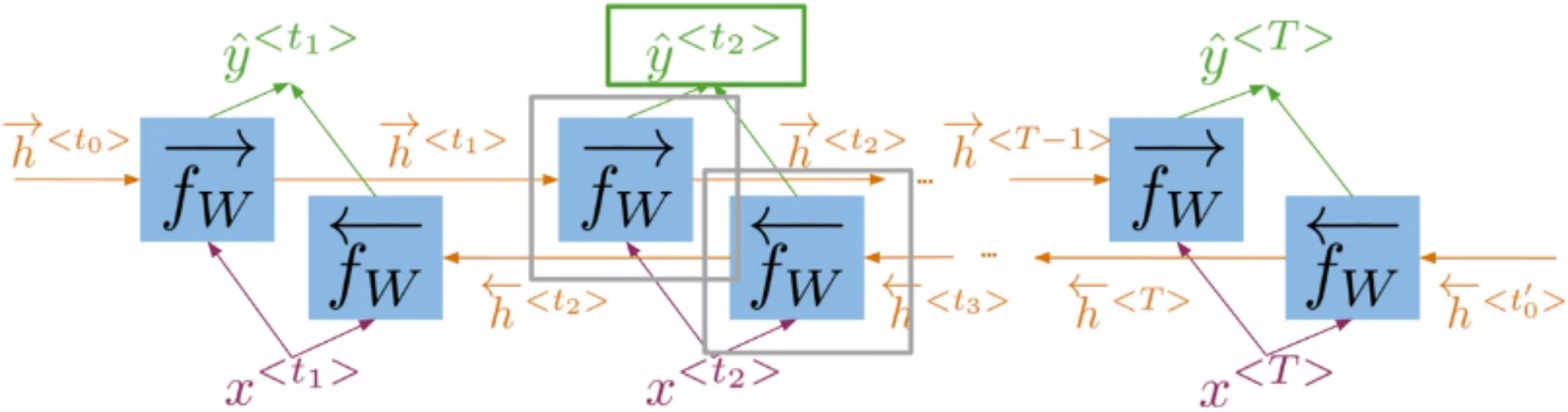
Information flows from the past and from the future **independently**

Bi-directional RNNs



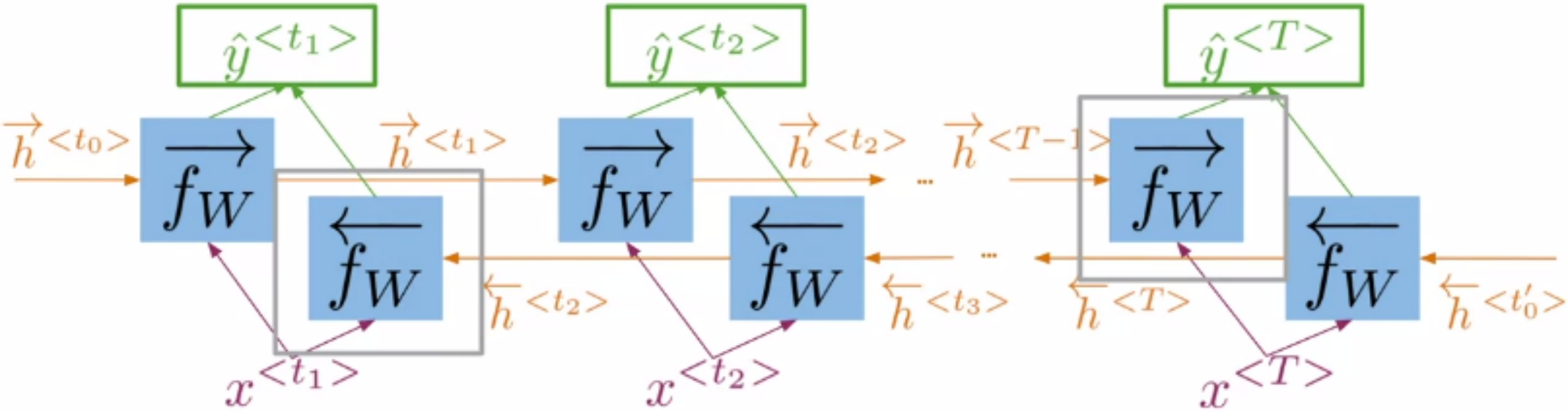
Information flows from the past and from the future **independently**

Bi-directional RNNs



$$\hat{y}^{<t>} = g(W_y[\overrightarrow{h}^{<t>}, \overleftarrow{h}^{<t>}] + b_y)$$

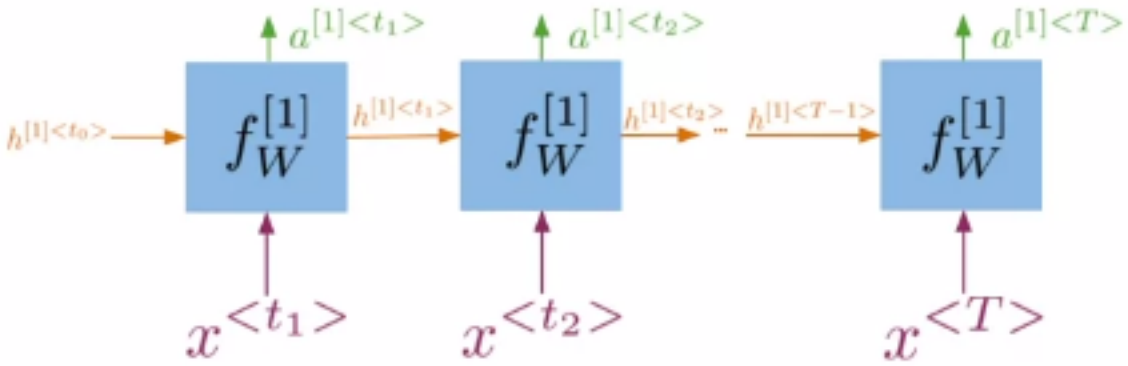
Bi-directional RNNs



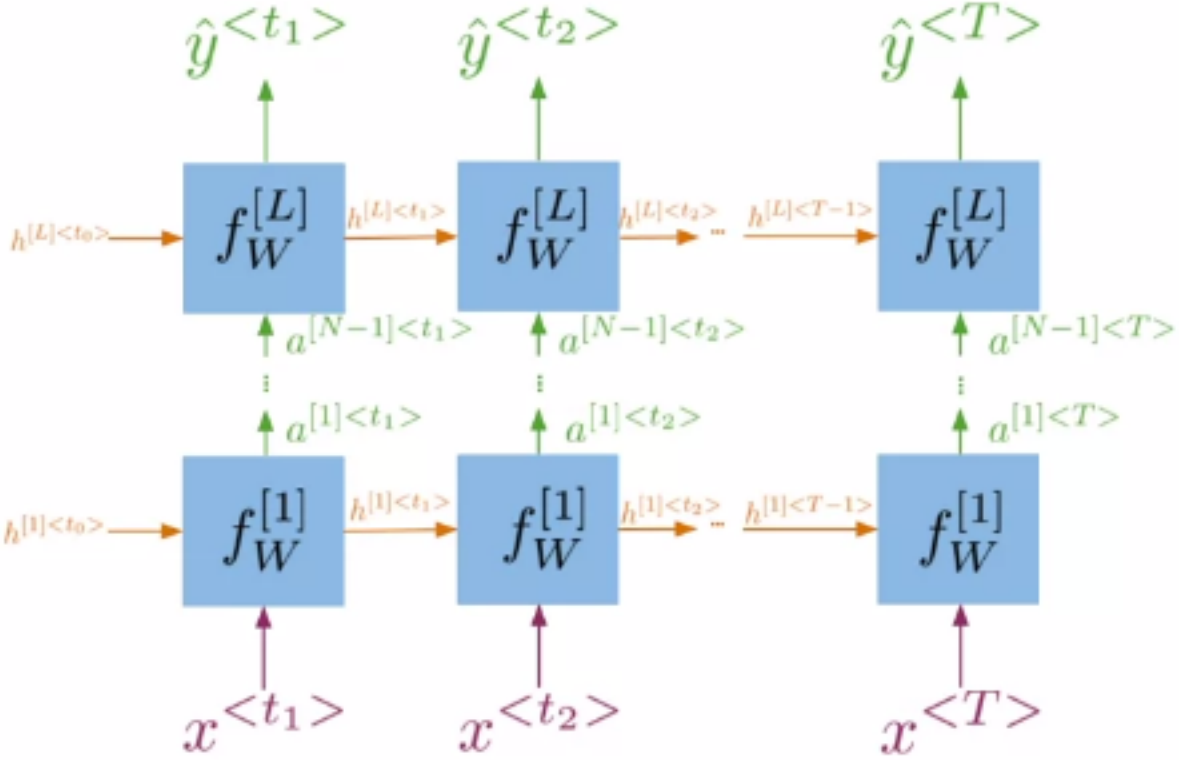
$$\hat{y}^{<t>} = g(W_y[\overrightarrow{h}^{<t>}, \overleftarrow{h}^{<t>}] + b_y)$$

Deep RNNs

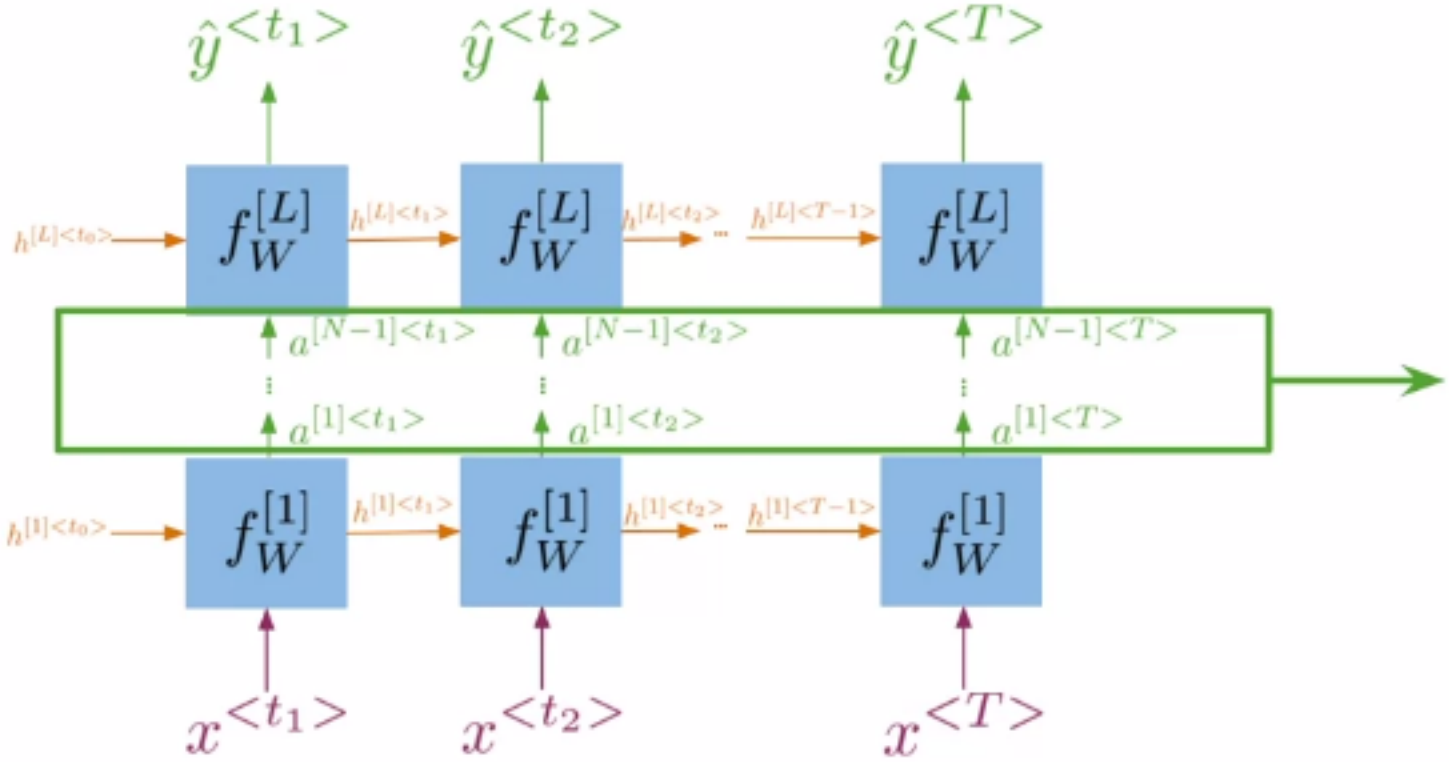
Deep RNNs



Deep RNNs

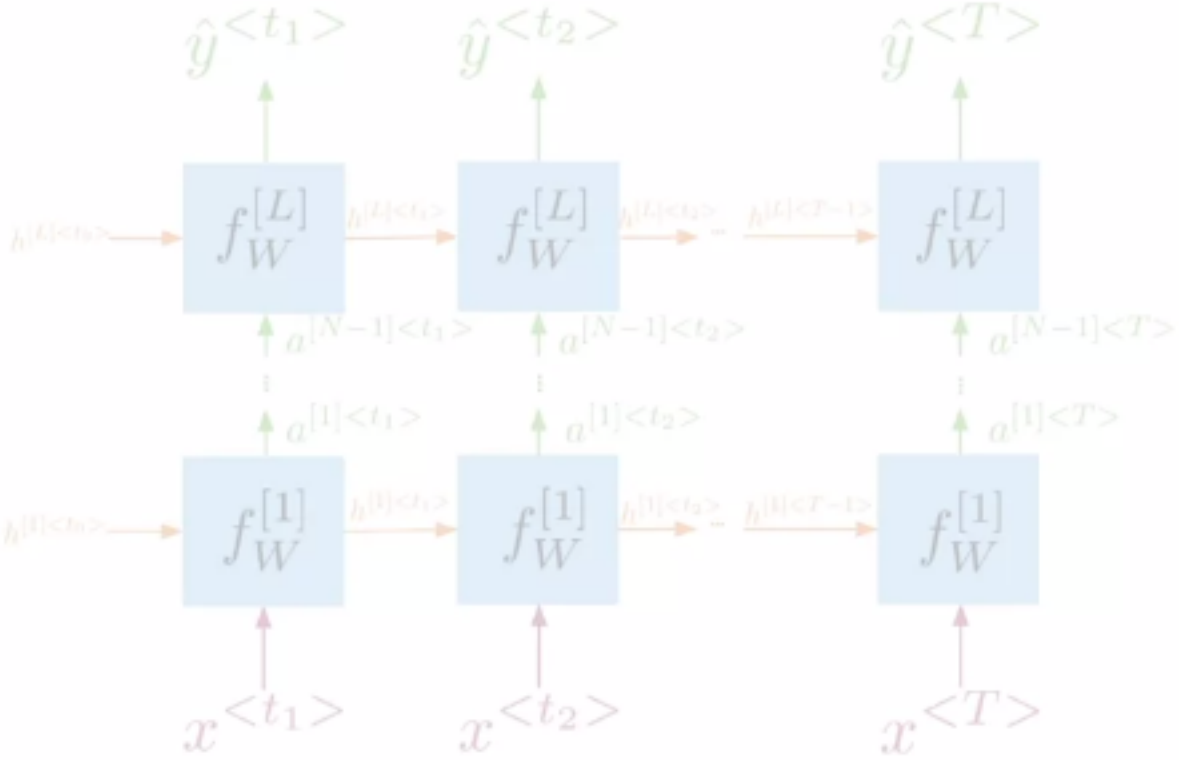

$\hat{y}^{<t_1>}$
$\hat{y}^{<t_2>}$
$\hat{y}^{<T>}$
$f_W^{[L]}$
$f_W^{[1]}$
$h^{[L]<t_0>}$
$h^{[L]<t_1>}$
$h^{[L]<t_2>}$
$h^{[L]<T-1>}$
$h^{[1]<t_0>}$
$h^{[1]<t_1>}$
$h^{[1]<t_2>}$
$h^{[1]<T-1>}$
$a^{[N-1]<t_1>}$
$a^{[N-1]<t_2>}$
$a^{[N-1]<T>}$
$a^{[1]<t_1>}$
$a^{[1]<t_2>}$
$a^{[1]<T>}$
$x^{<t_1>}$
$x^{<t_2>}$
$x^{<T>}$

Deep RNNs



Intermediate layers and activations

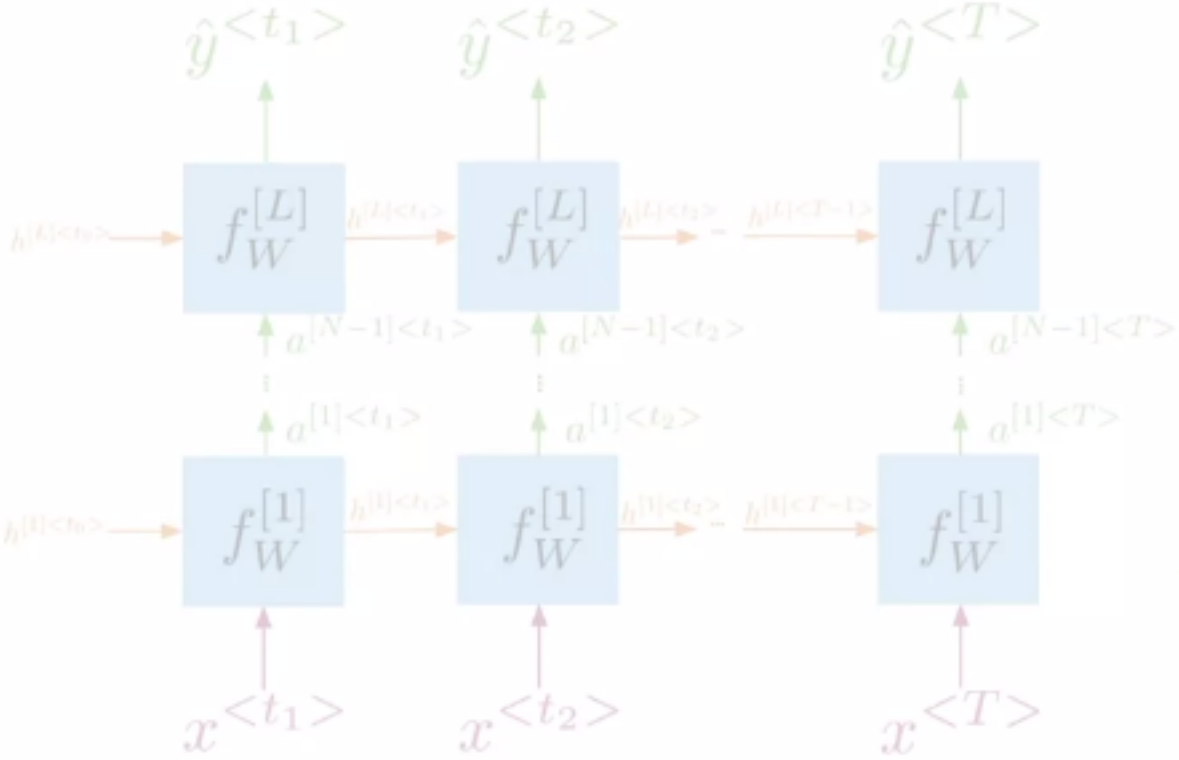
Deep RNNs



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

Deep RNNs

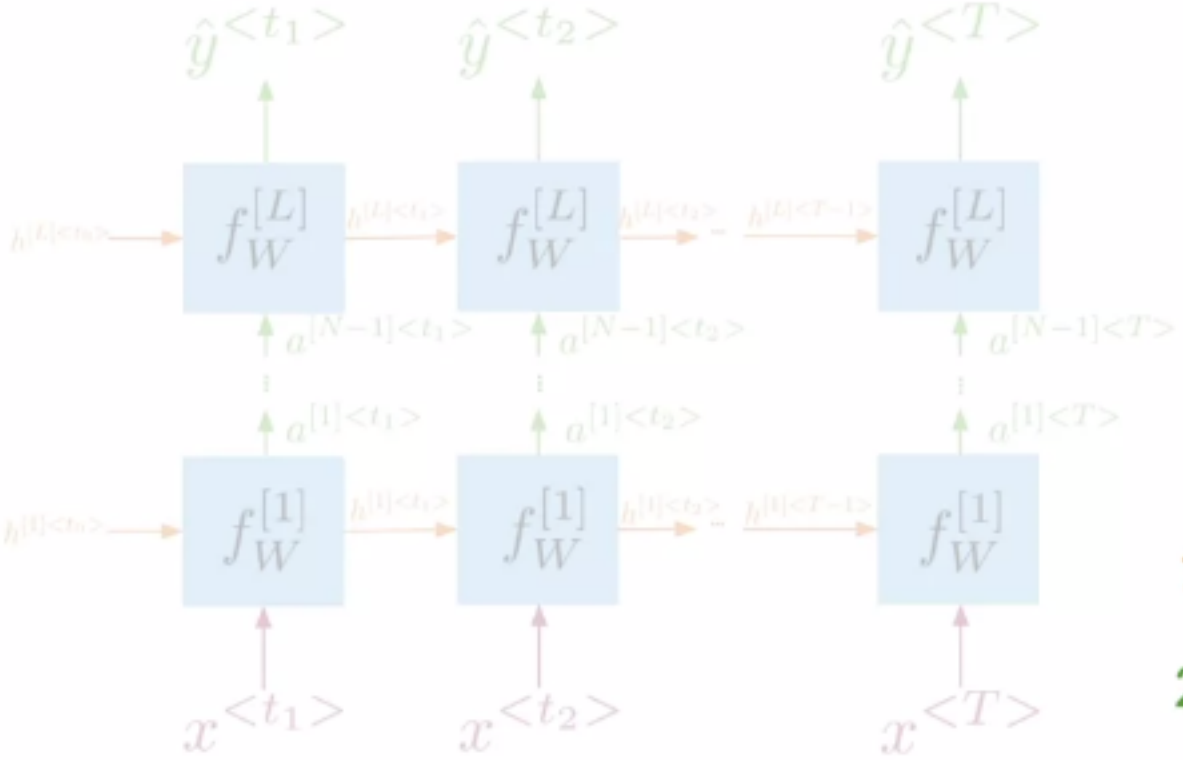


$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

1. Get hidden states for current layer

Deep RNNs



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

1. Get hidden states for current layer
2. Pass the activations to the next layer

Summary

- In bidirectional RNNs, the outputs take information from the past and the future
- Deep RNNs have more than one layer, which helps in complex tasks