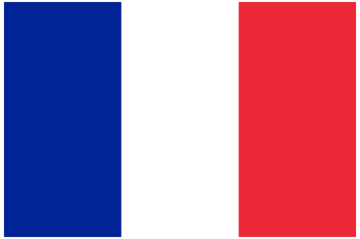
Copyright Notice

deeplearning.ai

Traditional Language models

Traditional Language Models



J'ai vu le match de foot

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

N-grams

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

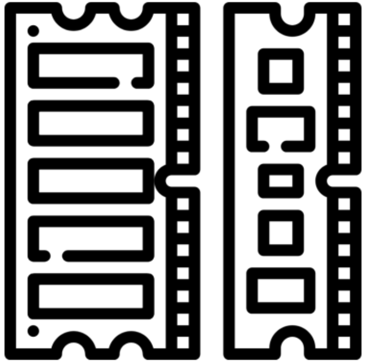
$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

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

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

Summary

- N-grams consume a lot of memory
- Different types of RNNs are the preferred alternative



deeplearning.ai

Recurrent Neural Networks

Advantages of RNNs

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

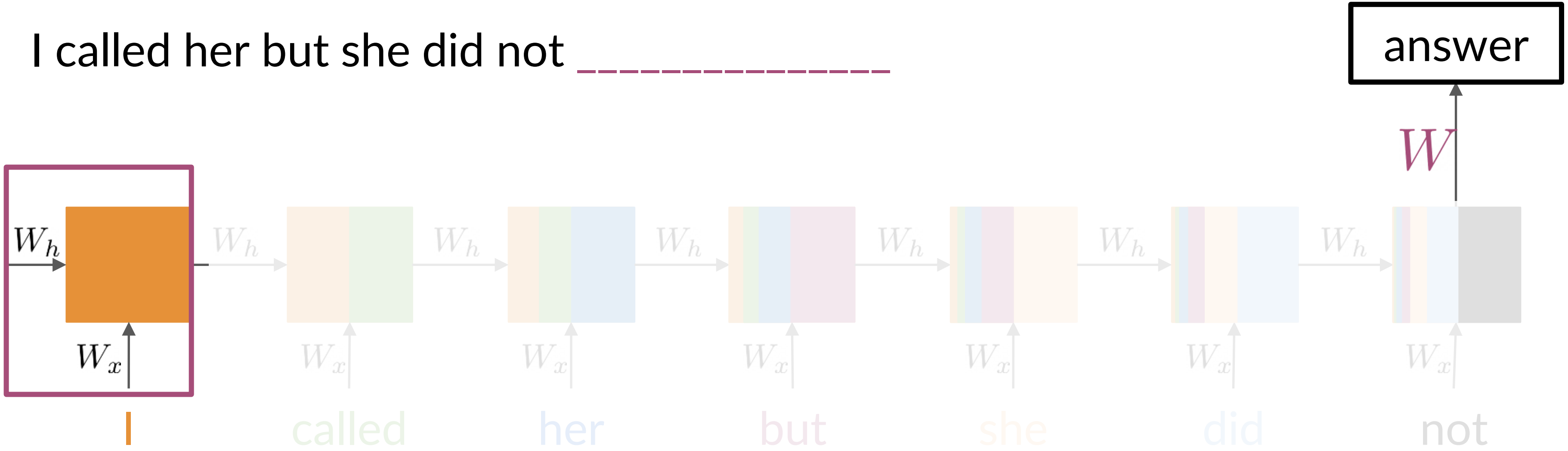
want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

Similar probabilities with trigram

RNNs Basic Structure

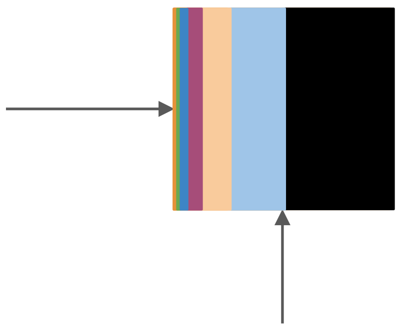
I called her but she did not _______________



Learnable parameters

Summary

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



deeplearning.ai

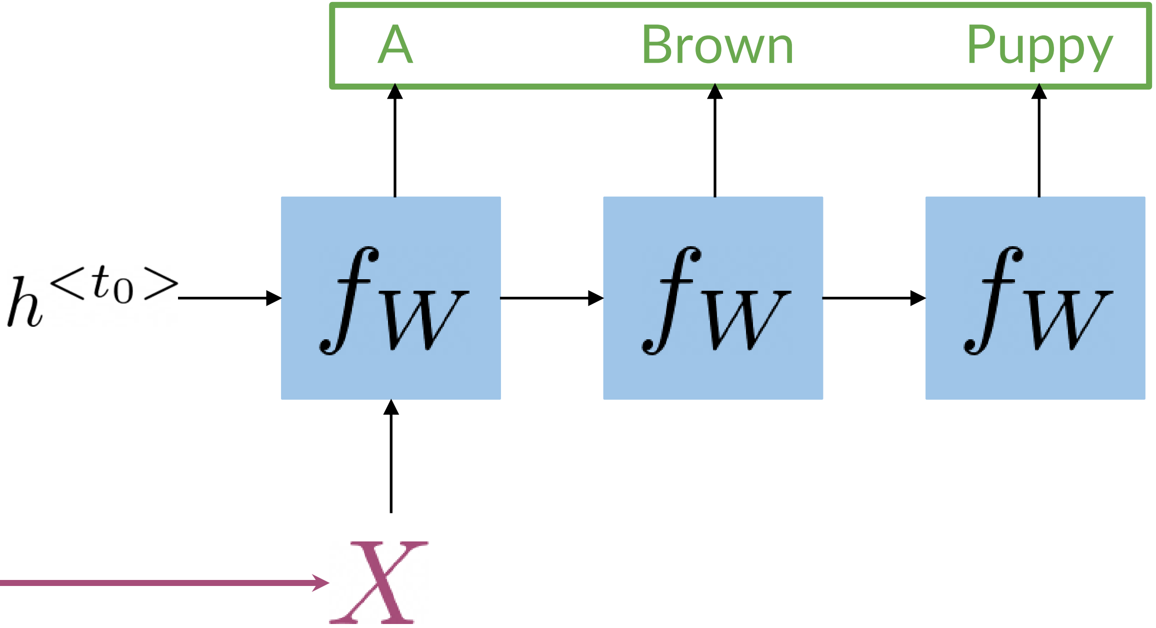
Applications of RNNs

One to One

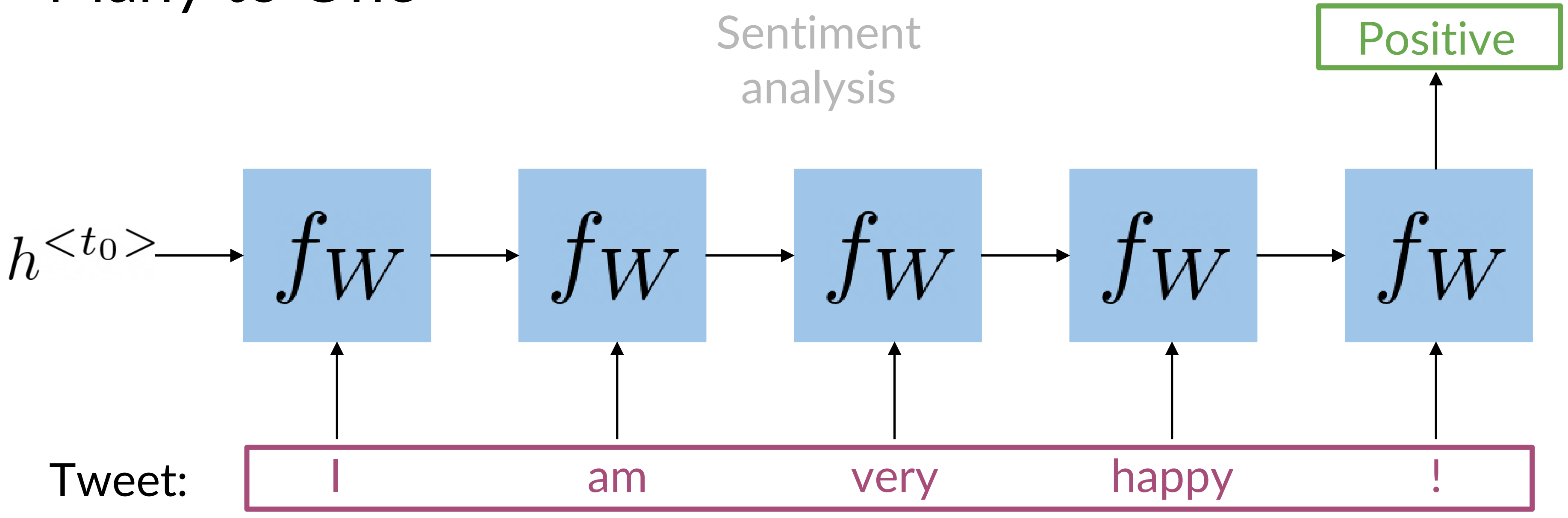

LaLiga
Santander
Real Valladolid
0 1
Real Madrid
Real Zaragoza
0 4
Real Madrid
Atletico Madrid
0 1
Real Madrid
$h^{<t_0>}$
f_W
X

One to Many

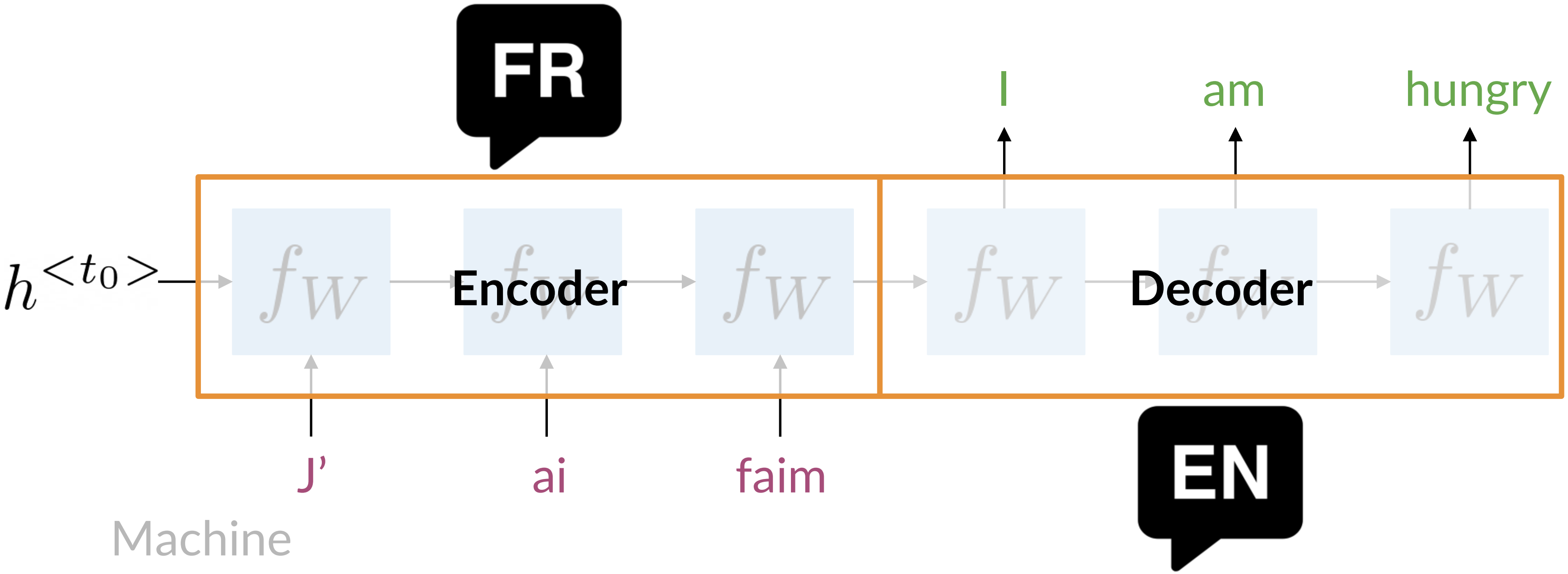
Caption generation



Many to One


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

Many to Many



Machine Translation

Summary

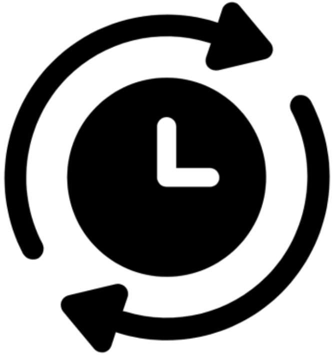
- RNNs can be implemented for a variety of NLP tasks
- Applications include Machine translation and caption generation

deeplearning.ai

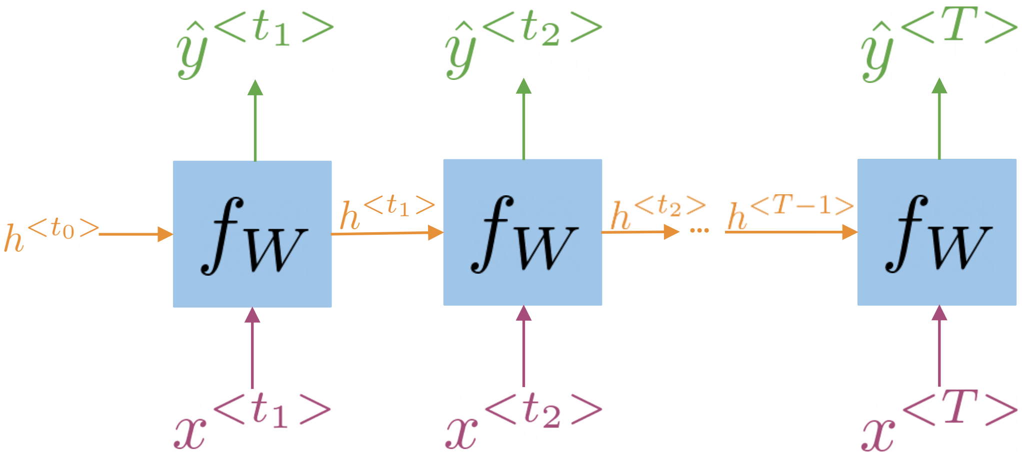
Math in Simple RNNs

Outline

- How RNNs propagate information (Through time!)
- How RNNs make predictions



A Vanilla RNN

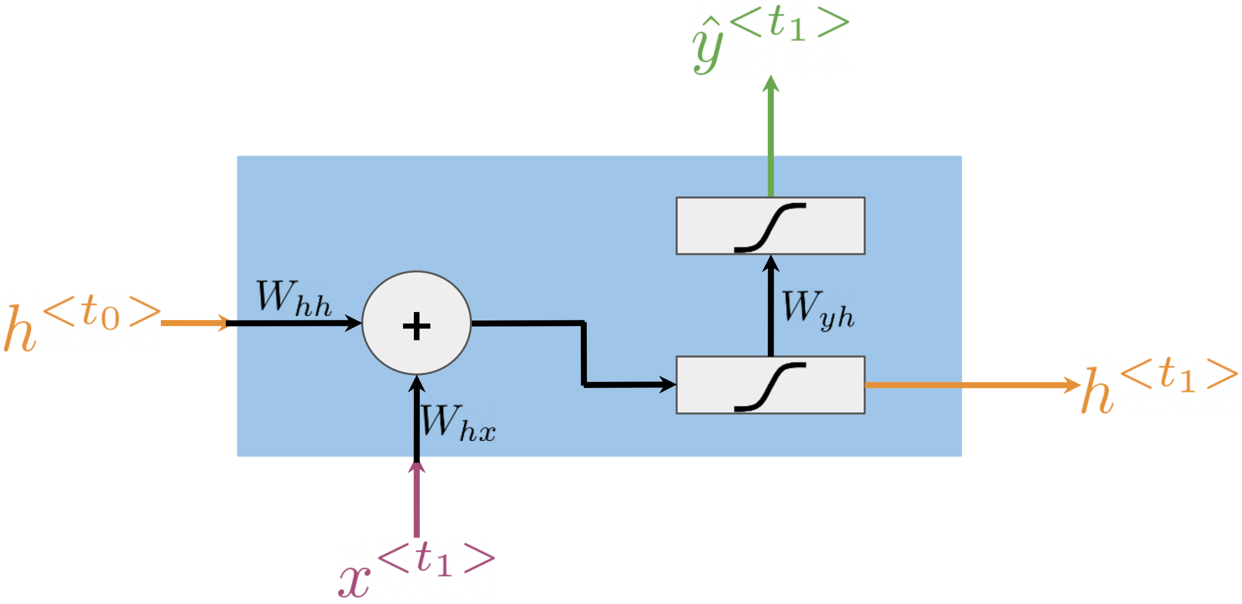


$$h^{<t>} = g(\boxed{W_h[h^{<t-1>}, x^{<t>}]} + \boxed{b_h})$$

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

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

A Vanilla RNN

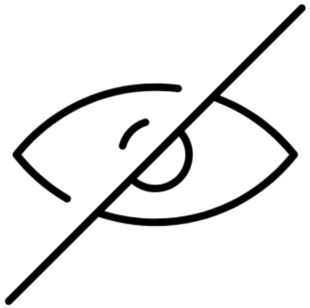


$$h^{<t>} = g(\boxed{W_{hh}h^{<t-1>} + W_{hx}x^{<t>}} + b_h)$$

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

Summary

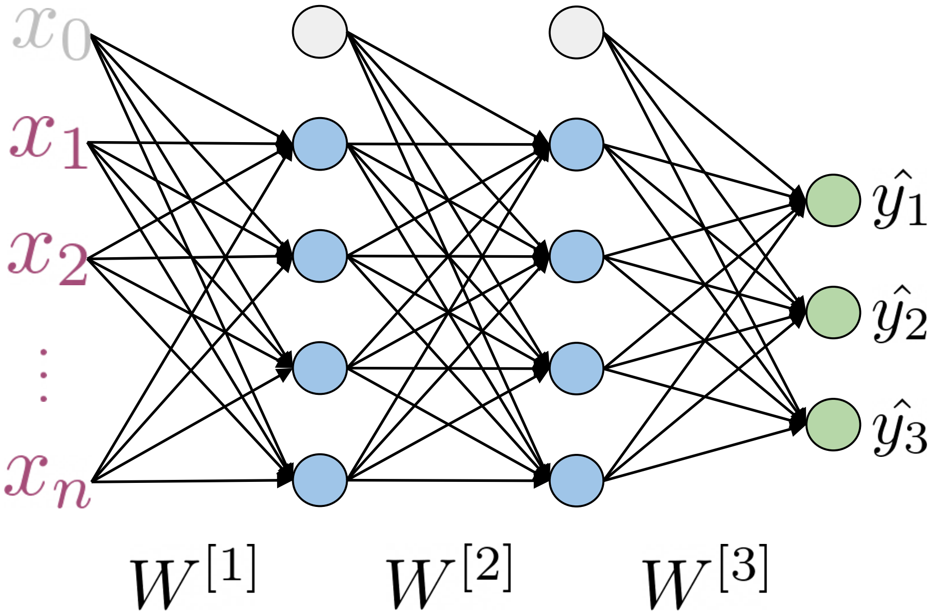
- Hidden states propagate information through time
- Basic recurrent units have two inputs at each time: $h^{<t-1>}, x^{<t>}$



deeplearning.ai

Cost Function for RNNs

Cross Entropy Loss



K - classes or possibilities

Either 0 or 1

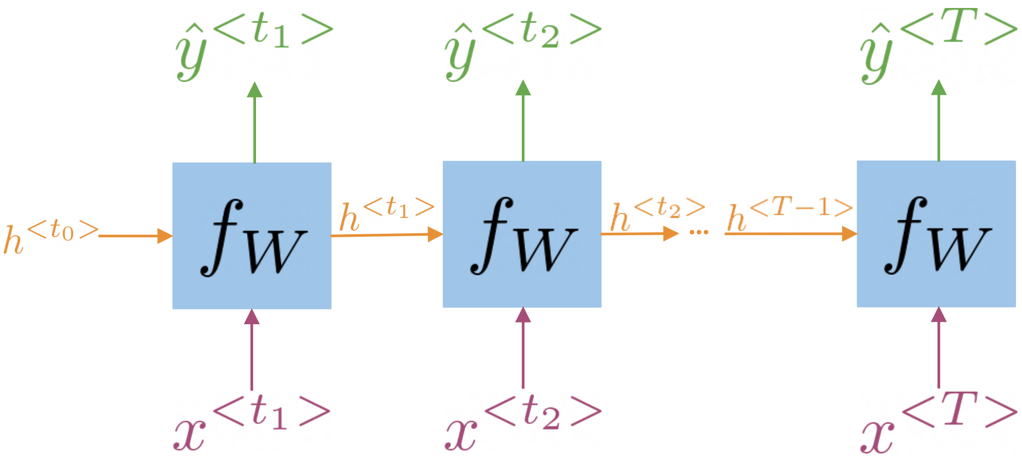
$$J = -\sum_{j=1}^{\boxed{K}} \boxed{y_j} \log \hat{y}_j$$

Looking at a single example (x, y)

Cross Entropy Loss

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

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



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

Average with respect to time

Summary

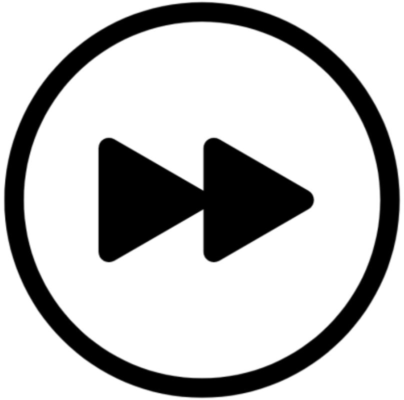
For RNNs the loss function is just an average through time!

deeplearning.ai

Implementation Note

Outline

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



tf.scan() function

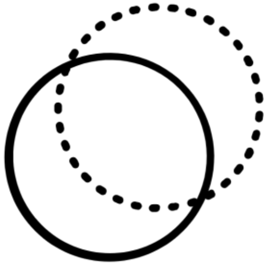

```
def scan(fn, elems, initializer=None,
...):
    cur_value = initializer
    ys = []

    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

Parallel computations and GPU usage

Summary

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

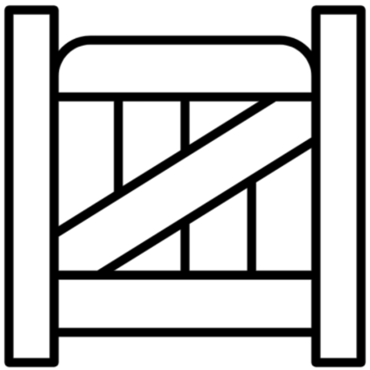


deeplearning.ai

Gated Recurrent Units

Outline

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



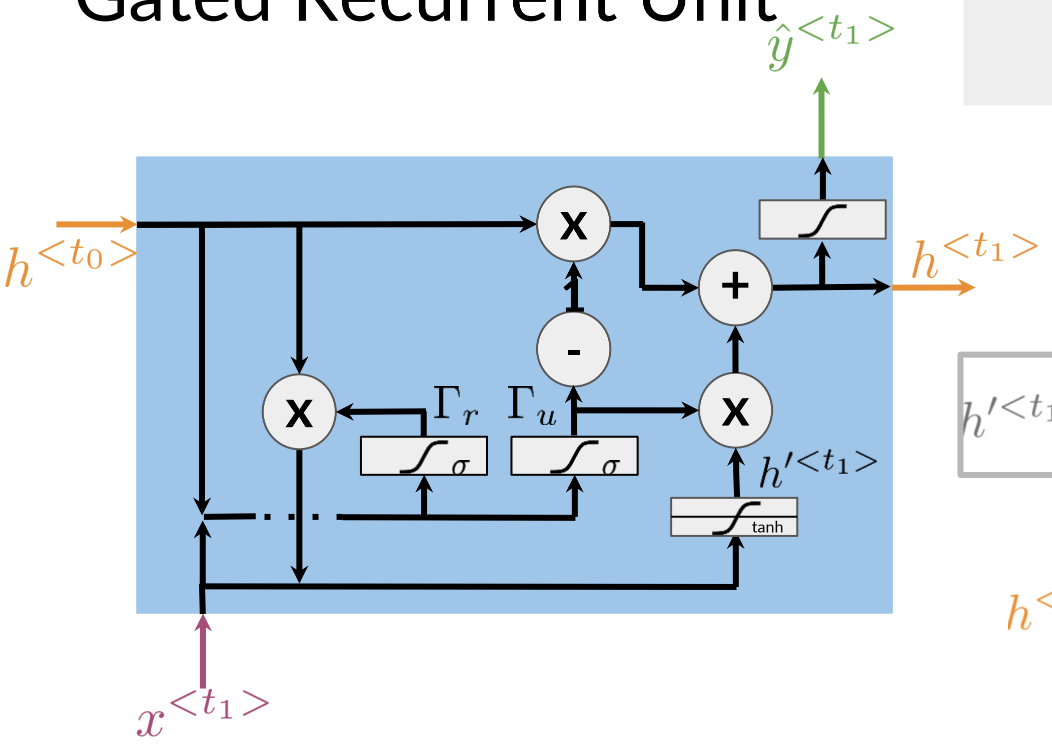
Gated Recurrent Units

“Ants are really interesting. __They_____ are everywhere.”

Plural

Relevance and update gates to remember important prior information

Gated Recurrent Unit



Gates to keep/update relevant information in the hidden state

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

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

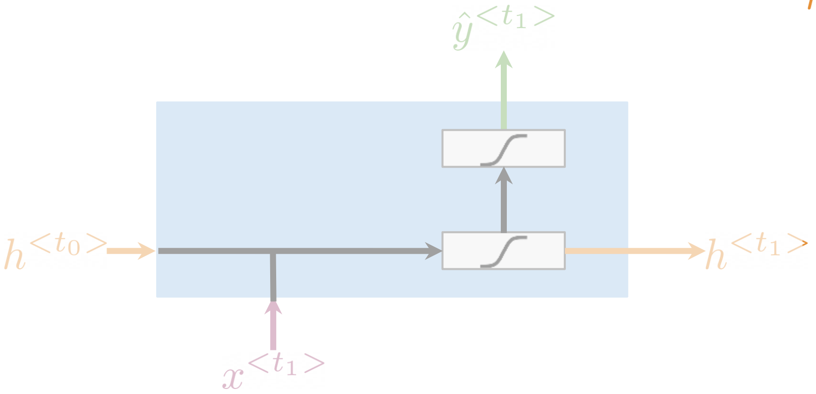
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

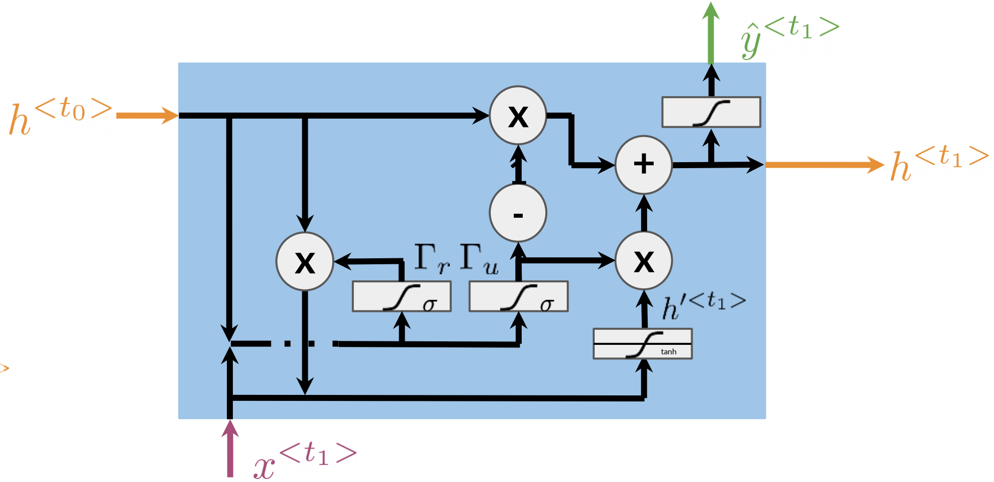
Hidden state candidate

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

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

Vanilla RNN vs GRUs





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

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

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

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

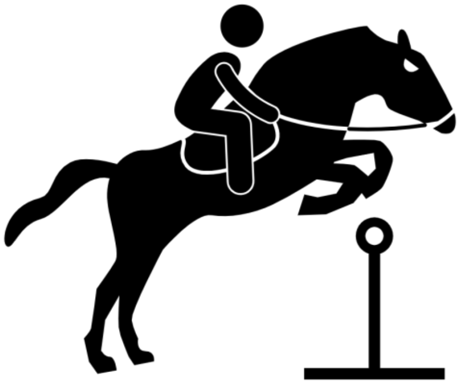
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

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

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

Summary

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

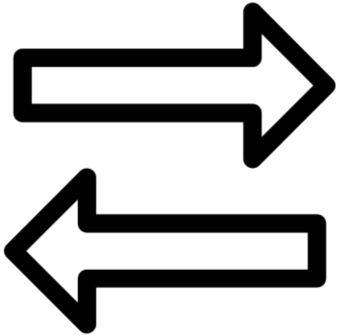


deeplearning.ai

Deep and Bi-directional RNNs

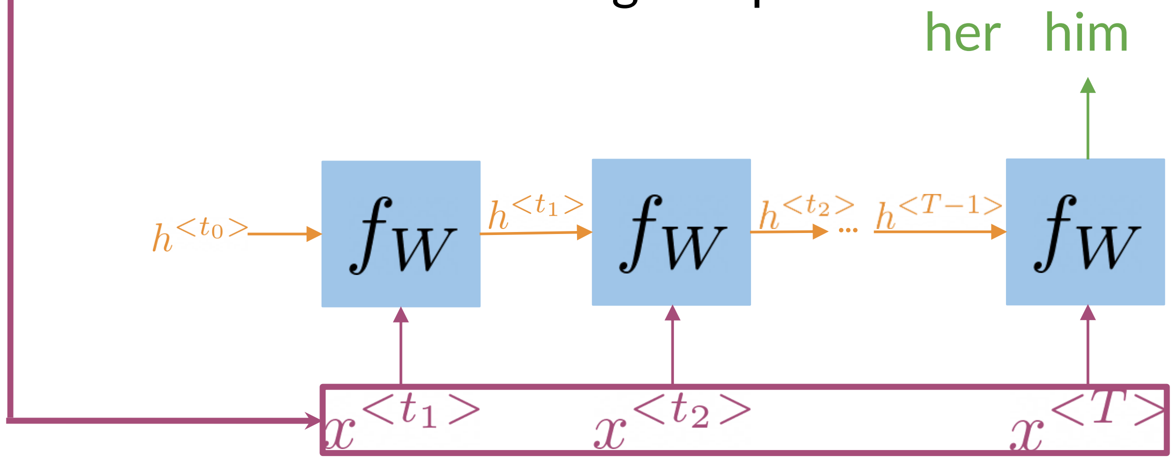
Outline

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

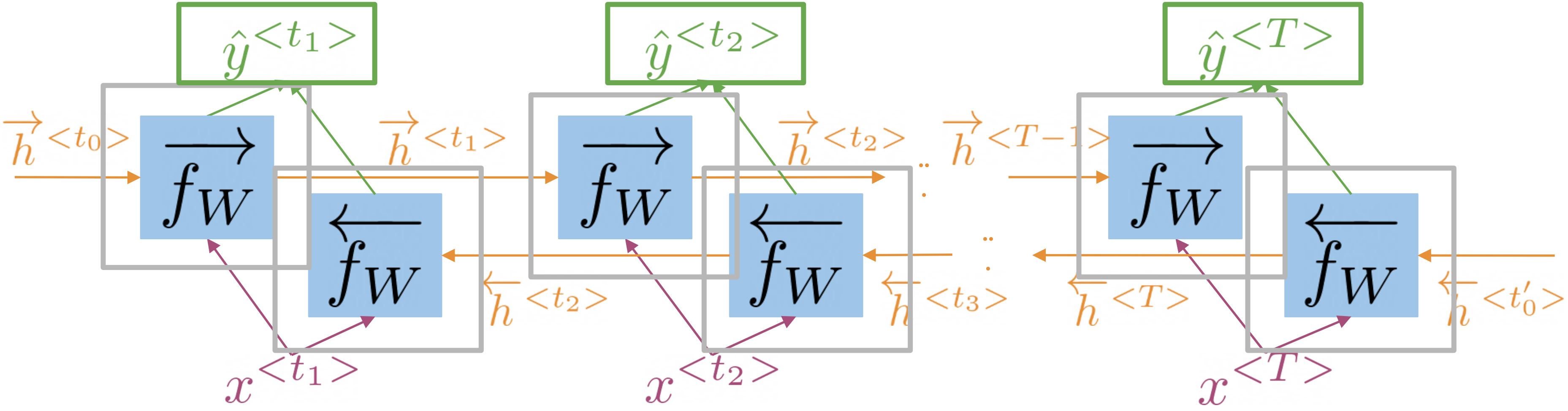


Bi-directional RNNs

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



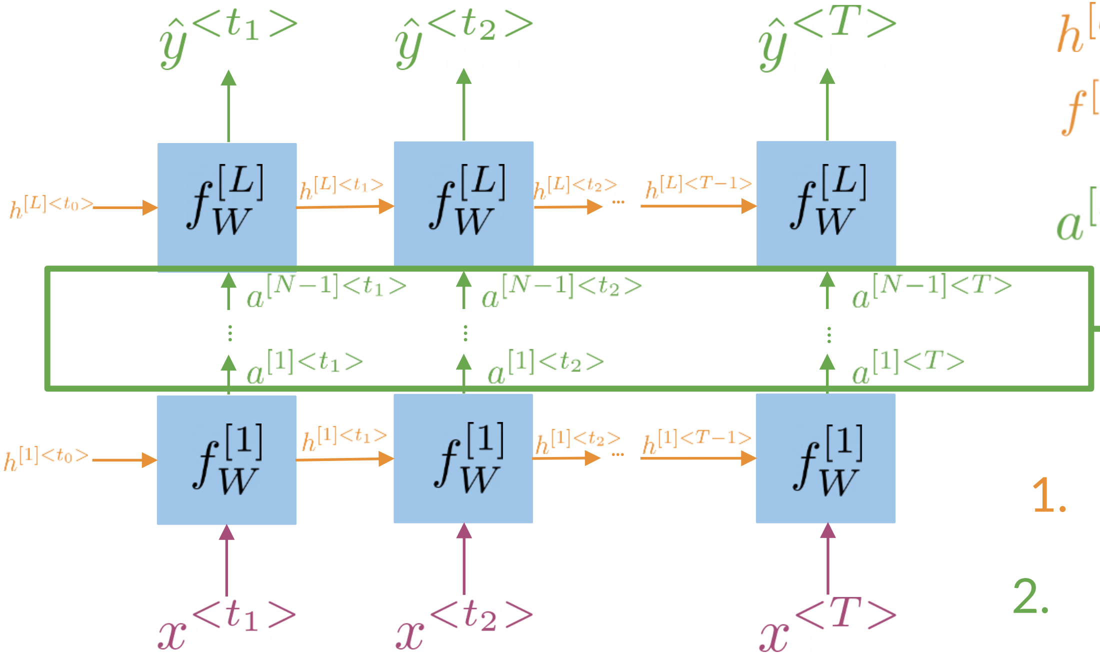
Bi-directional RNNs



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

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

Deep RNNs



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

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

Intermediate layers and activations

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