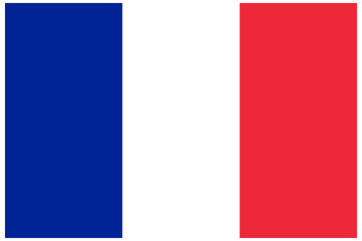
Recurrent Neural Networks For Language Modeling

FPT Education
FPT UNIVERSITY

- Traditional Language models
- Recurrent Neural Networks
- Gated Recurrent Units
- Deep and Bi-directional RNNs

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

- find several similar sentence candidates
- compute the probabilities of each sentence using a language model
- select the sequence of words with the highest probability

N-grams

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

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

N-grams consume a lot of memory

Different types of RNNs are the preferred alternative

Recurrent Neural Networks

- Advantages of RNNs
    - can capture some dependencies that could not captured with the traditional n-gram language model

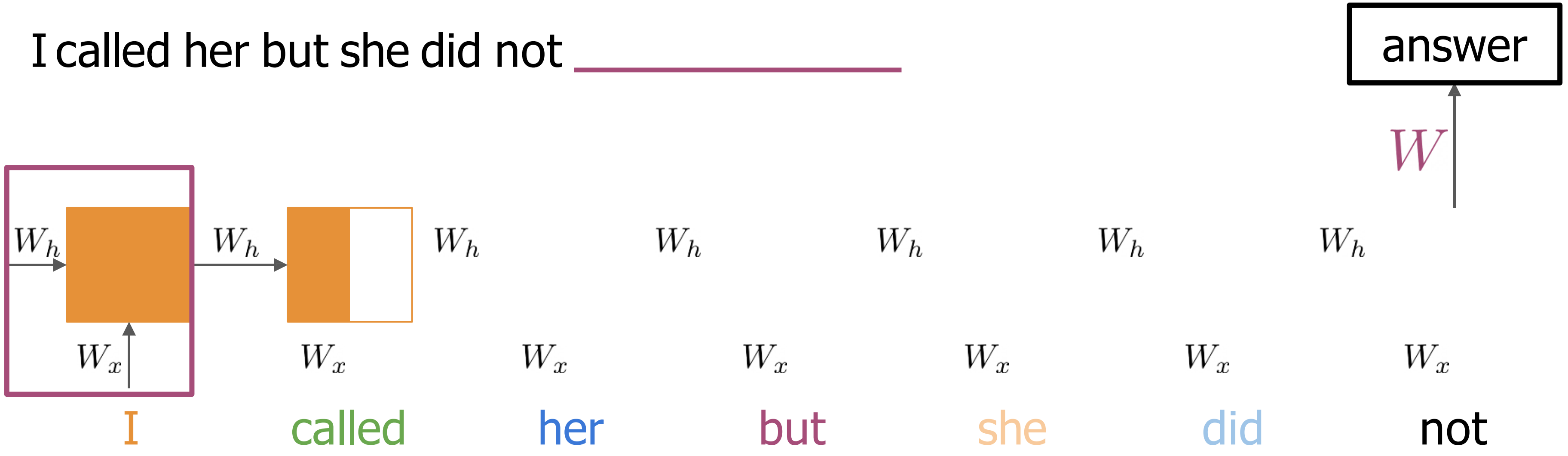
Nour was supposed to study with me. I called her but she did not answers

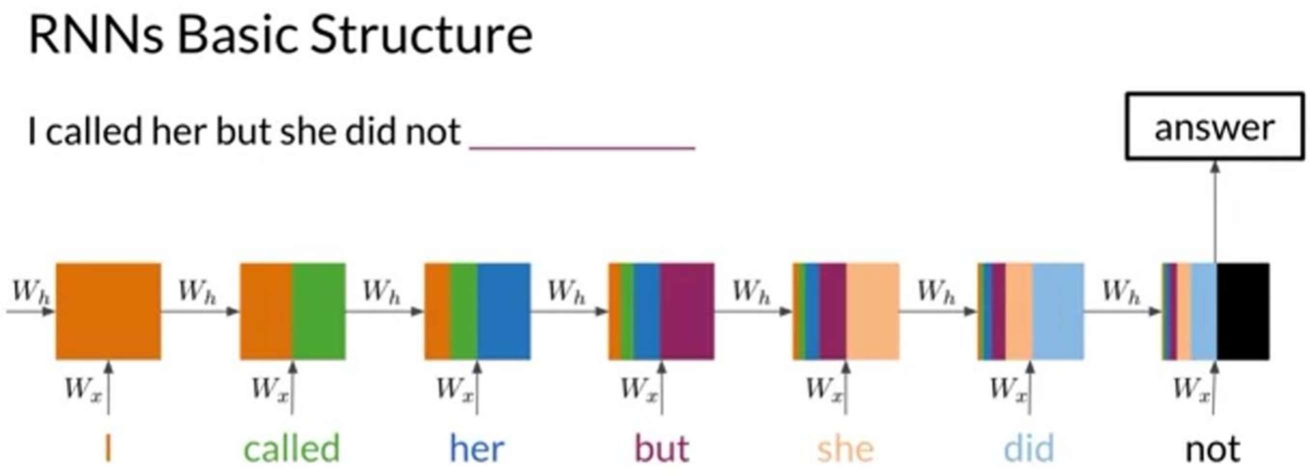
want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

Similar probabilities with trigram

RNNs Basic Structure

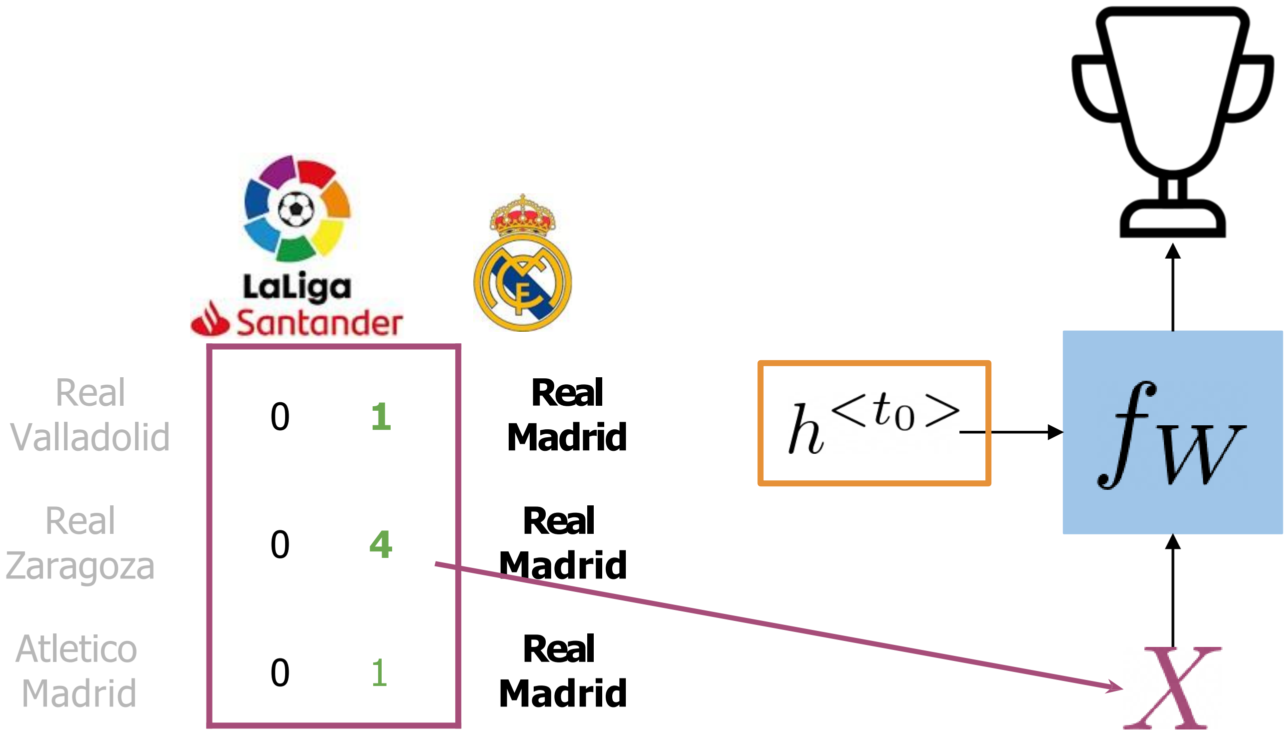




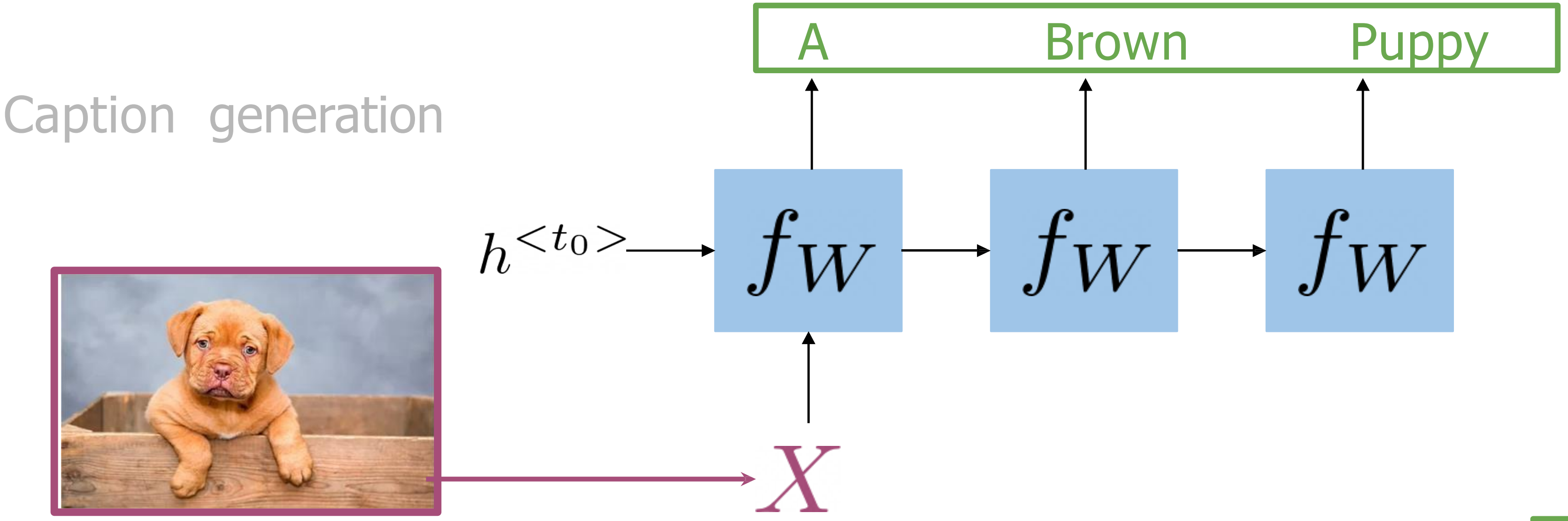
Learnable parameters

Applications of RNNs

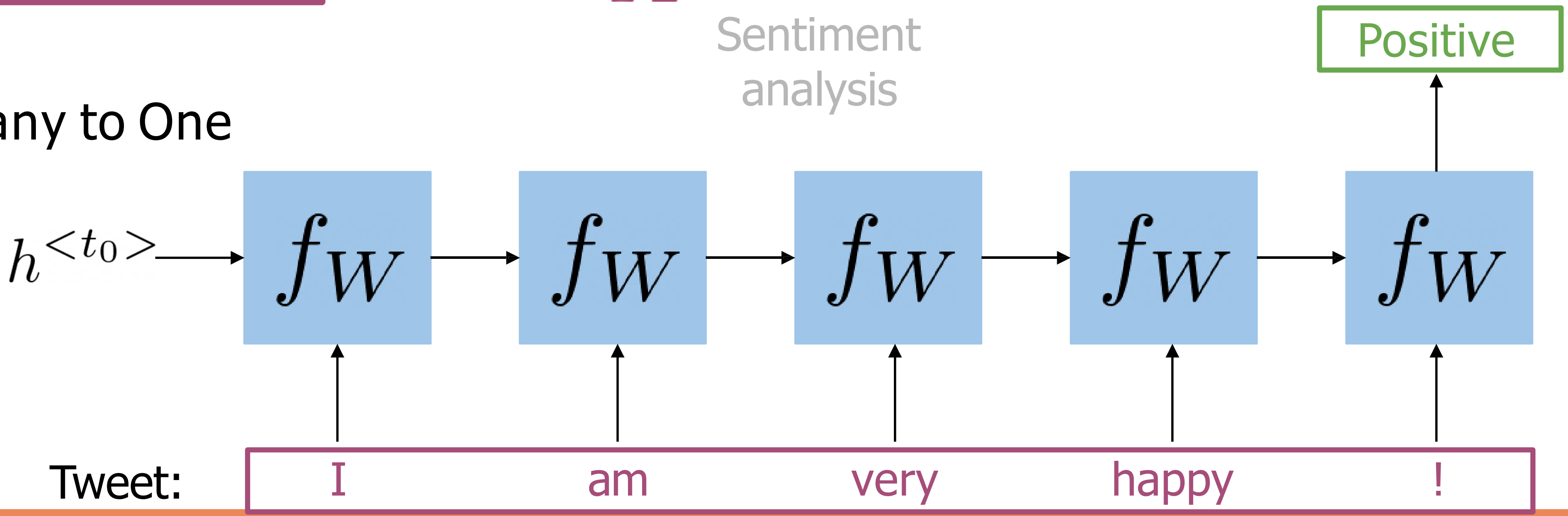
- One to One



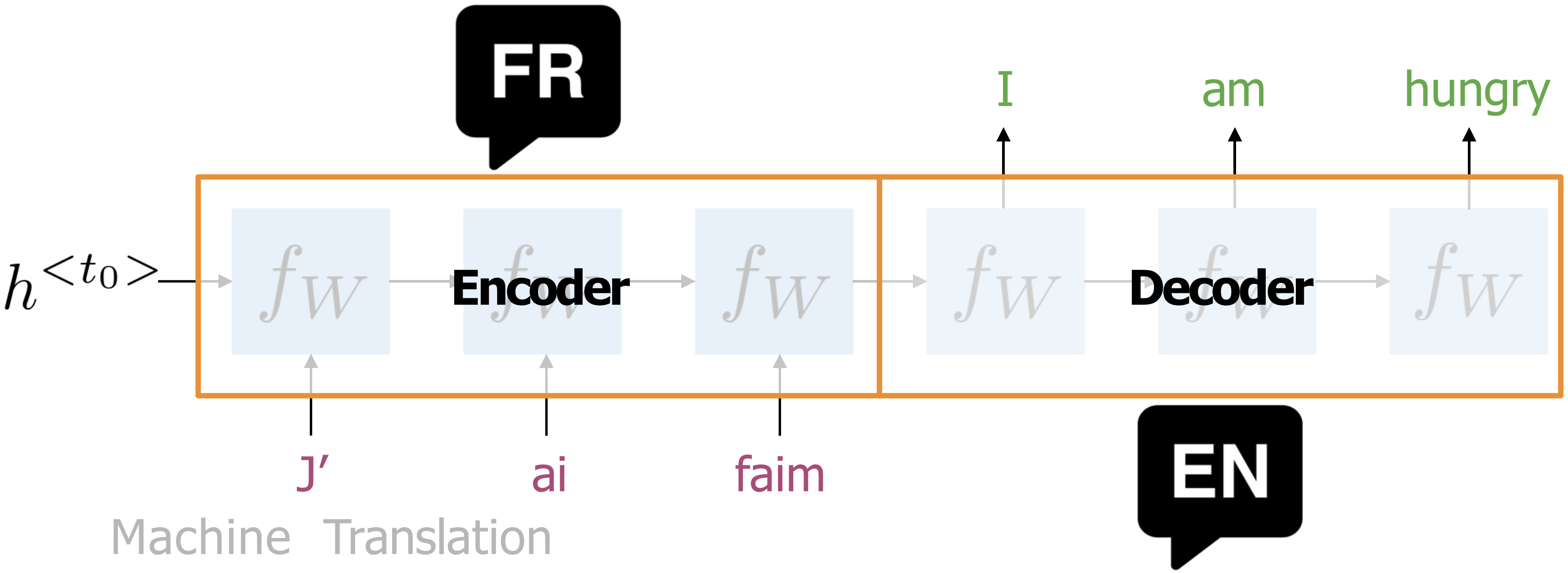
One to Many



- Many to One



Many to Many



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

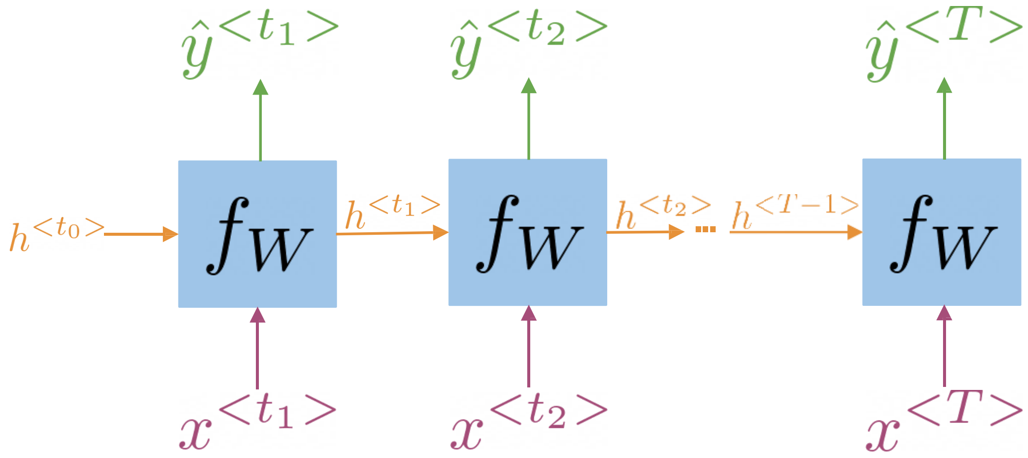
Math in Simple RNNs

FPT Education
FPT UNIVERSITY

How RNNs propagate information (Through time!)

How RNNs make predictions

- A Vanilla RNN

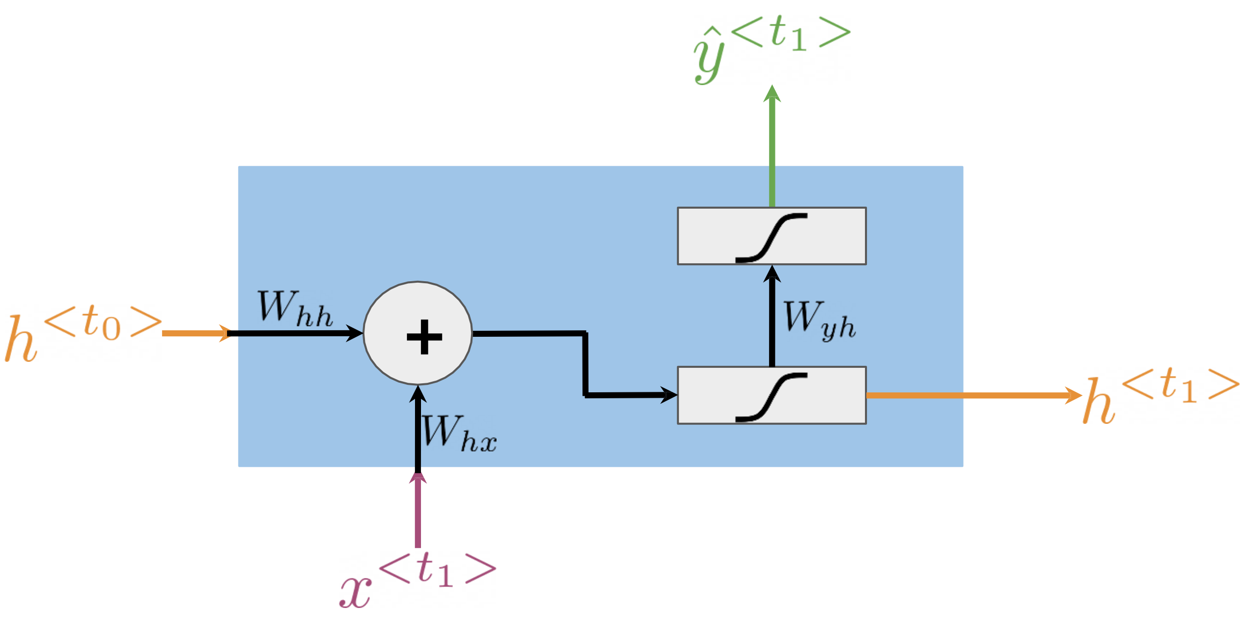


$$h^{<t>} = g(\boxed{W_h[h^{<t-1>}, x^{<t>}]}\boxed{+ b_h})$$

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

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

A Vanilla RNN



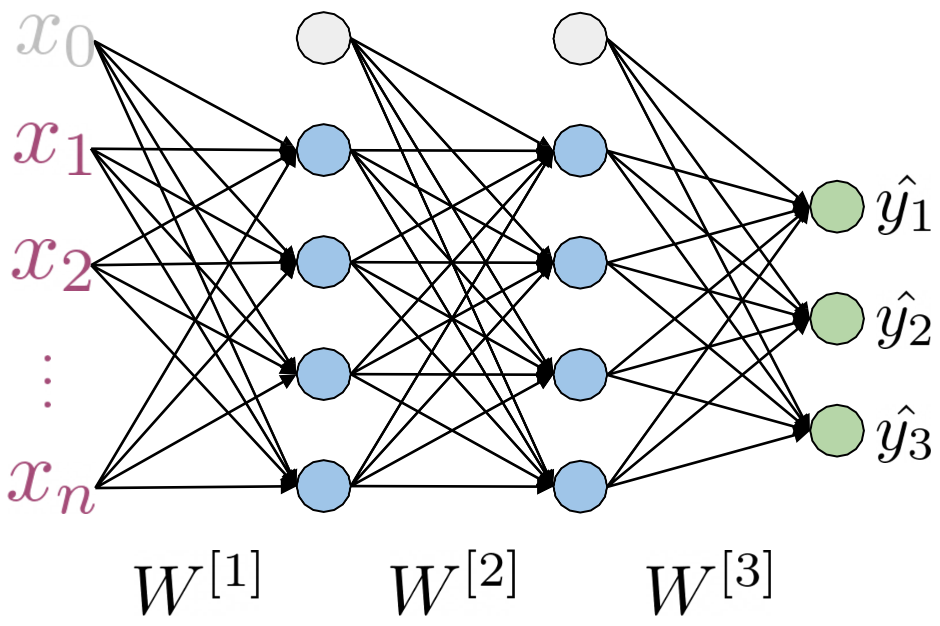
$$h^{<t>} = g(\boxed{W_{hh}h^{<t-1>} + W_{hx}x^{<t>}} + b_h)$$

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

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

Cost Function for RNNs

- Cross Entropy Loss



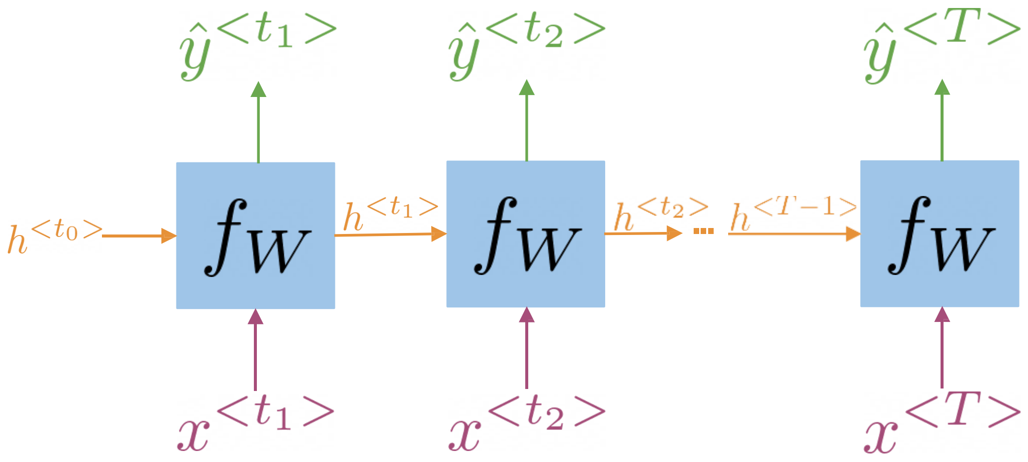
K - classes or possibilities

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

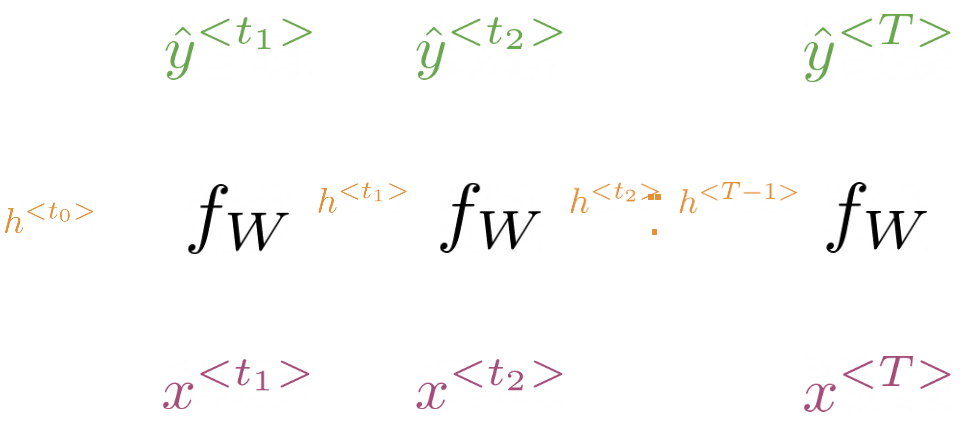
- For RNNs the loss function is just an average through time!

Implementation Note

scan() function in tensorflow

Computation of forward propagation using abstractions

- tf.scan() function




```python
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

Gated Recurrent Units

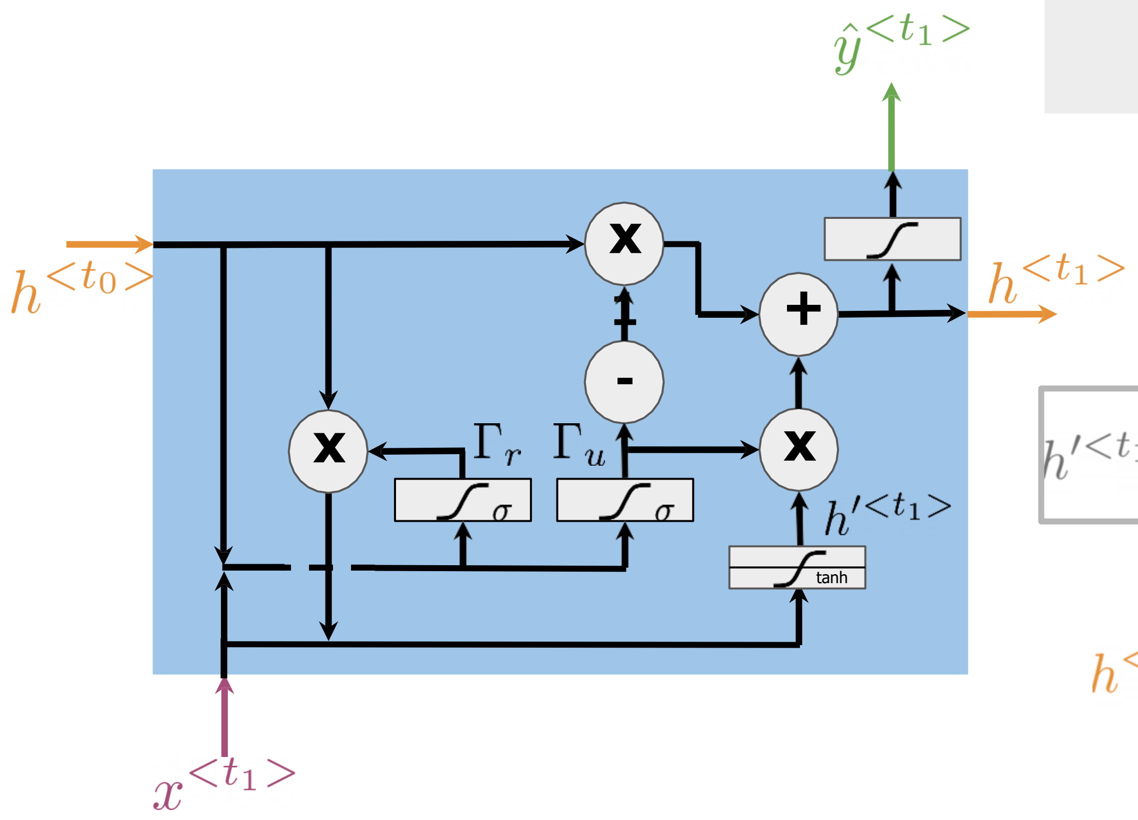
- Gated recurrent unit (GRU) structure
- Comparison between GRUs and vanilla RNNs

"Ants are really interesting. They are everywhere."

Plural

Relevance and update gates to remember important prior information

Comparison between GRUs and vanilla RNNs



Gates to keep/update relevant information in the hidden state

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

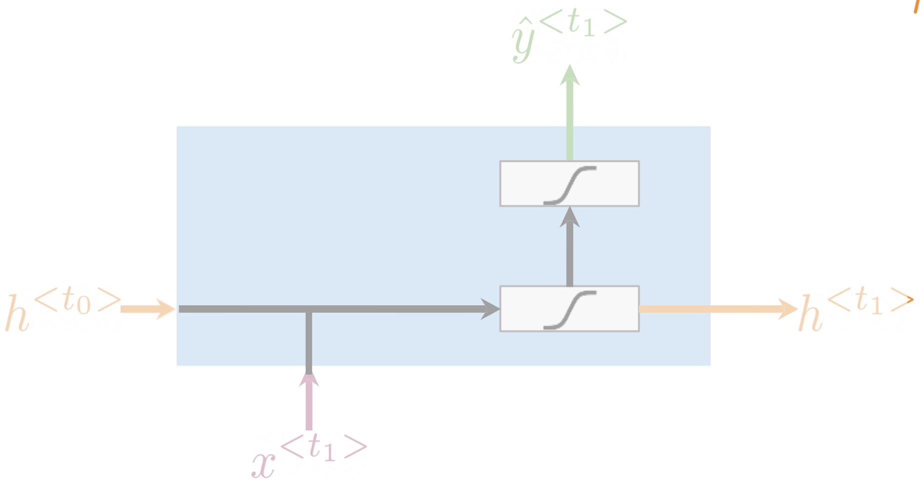
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

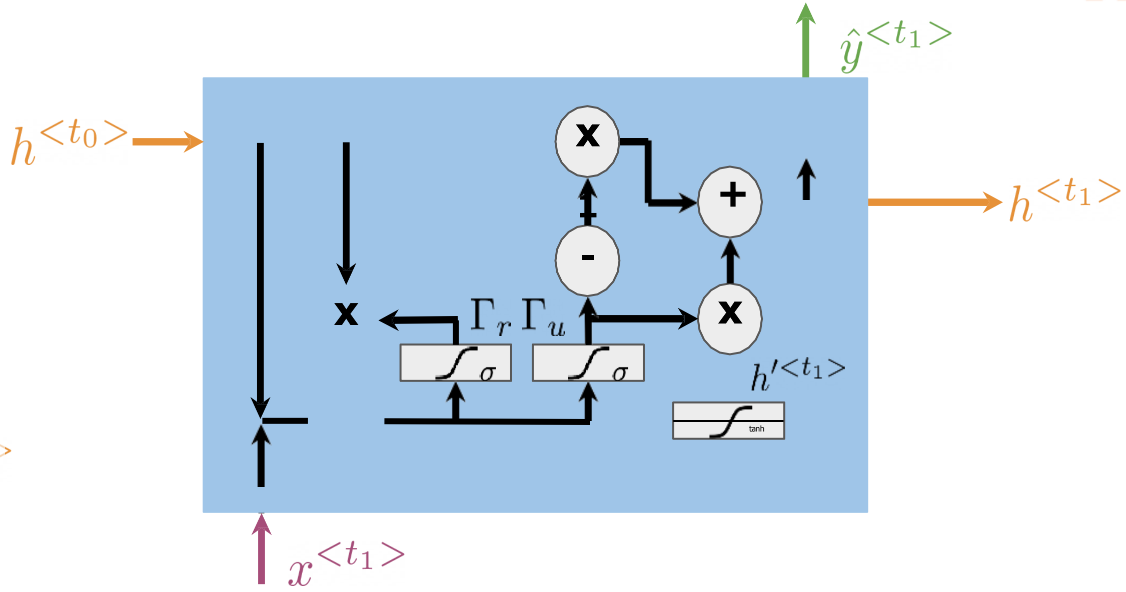
Hidden state candidate

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

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

Vanilla RNN vs GRUs





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

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

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

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

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

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

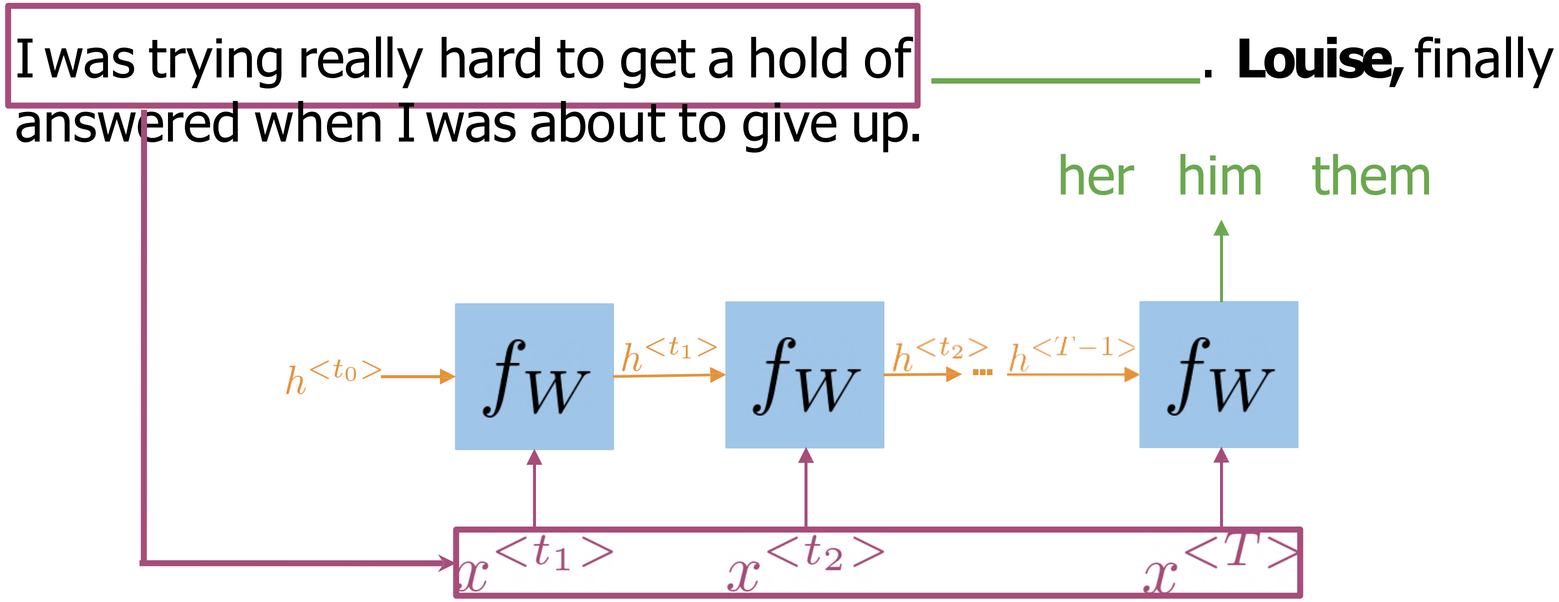
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Deep and Bi-directional RNNs

How bidirectional RNNs propagate information

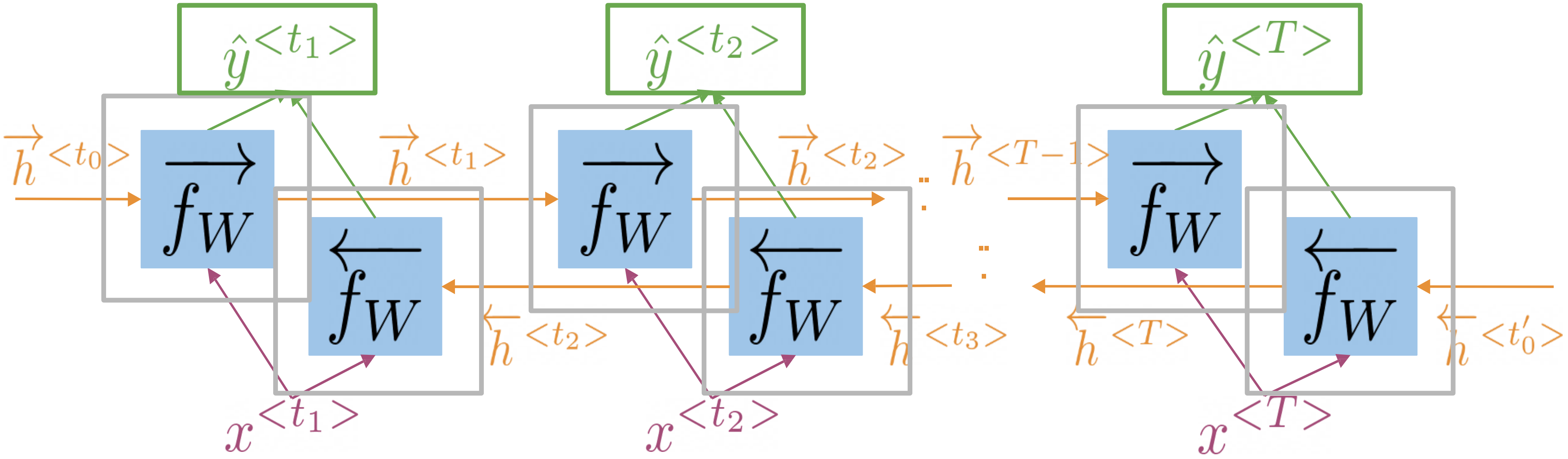
Forward propagation in deep RNNs

- Bi-directional RNNs



Bi-directional RNNs

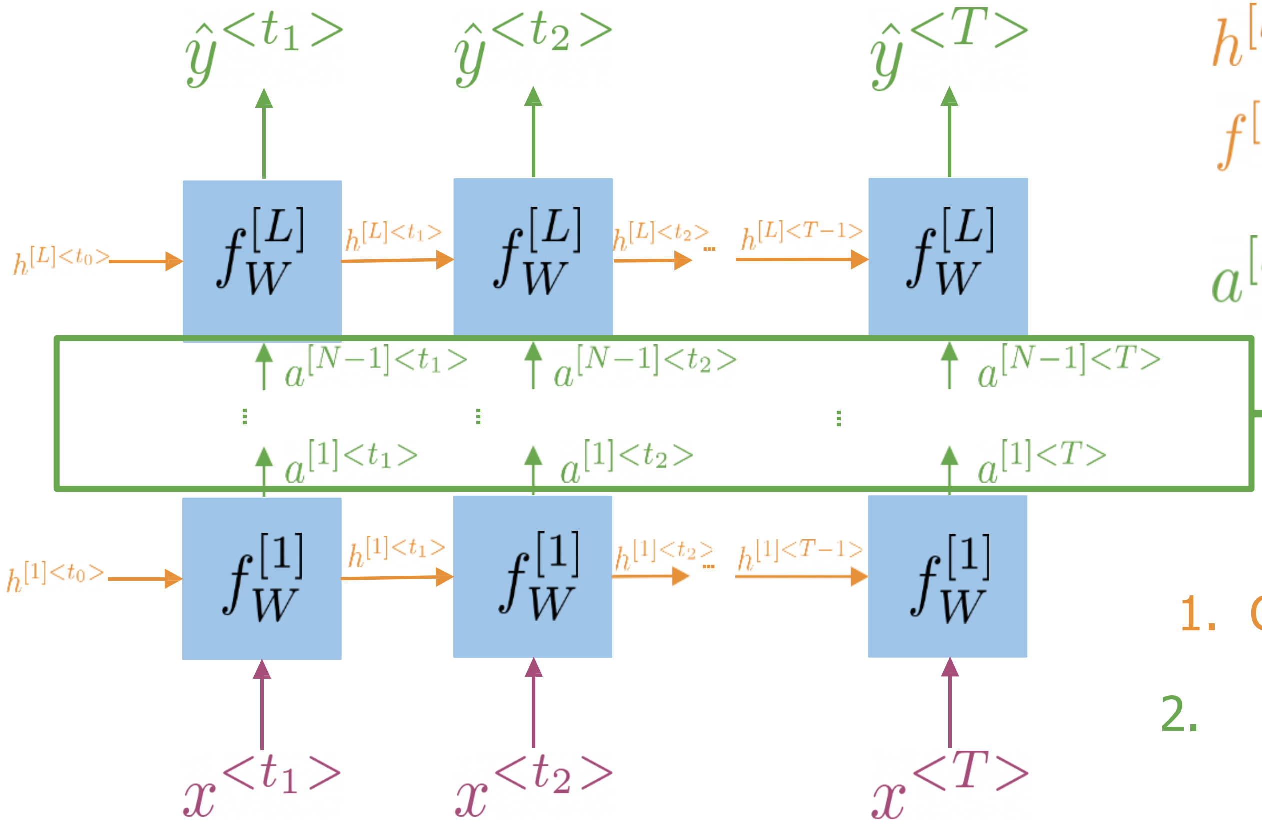
FPT Education
FPT UNIVERSITY



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

- 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