Извличане на знания от текст

19 март 2025 г.

REVIEW

Review

- What is a neural network?

Neural Networks

- A neural network is a network of small computational units:
  - A neural unit is a function that takes an input vector **x**, performs a computation, and produces an output
  - **Example**?

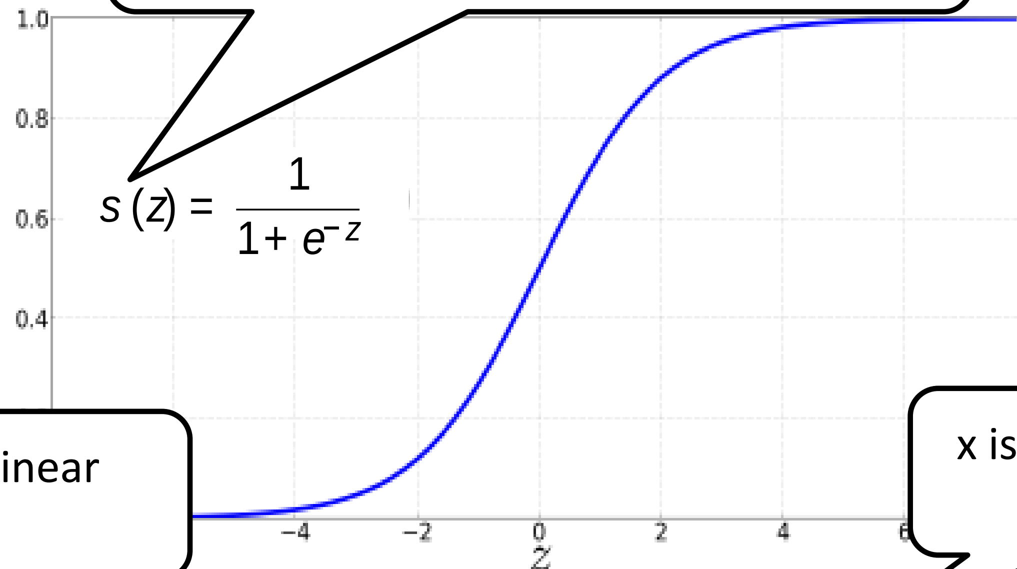
Neural Networks

- A neural network is a network of small computational units:
  - A neural unit is a function that takes an input vector **x**, performs a computation, and produces an output
  - **Example**: Binary Logistic regression
    - **Input** *x* is a feature vector
    - **Output** *y* is a number (0 or 1)

Review: Logistic Regression

This is a function that maps any real number to a value from 0 to 1

$$s(z) = \frac{1}{1 + e^{-z}}$$



This is basically linear regression

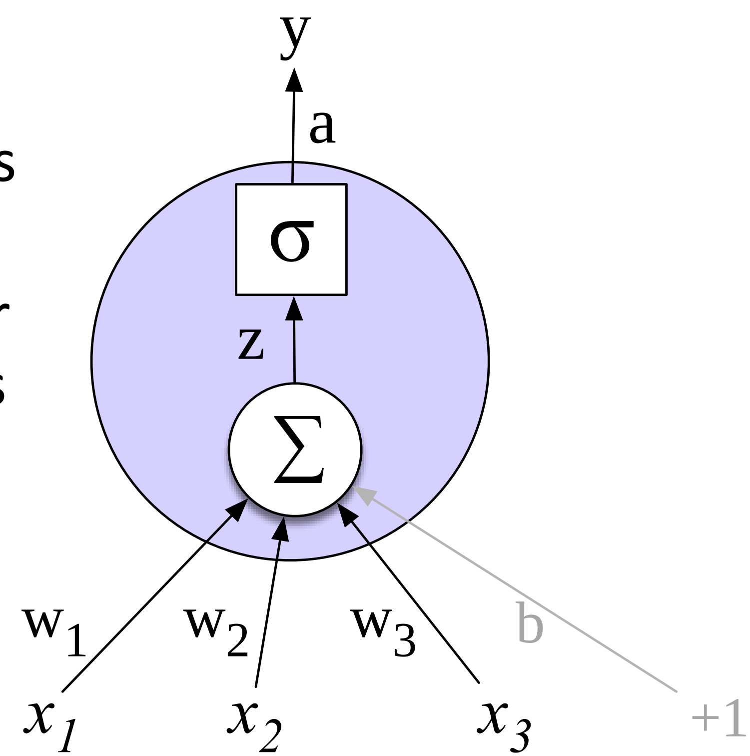
x is the input vector

$$z = \sum_{i=1}^{n} w_i x_i + b = w \cdot x + b$$

w is a set of values (weights) that are optimized to fit the data

Visualizing a Neural Unit

- A neural unit is a function of a vector of inputs. Each unit produces one output, called **activation**
- The output symbol $\mathtt{y}$ is reserved for the final output of the network. *a* is the activation or output of a single neuron
  - In this case where we have just one neural unit, $y=a$

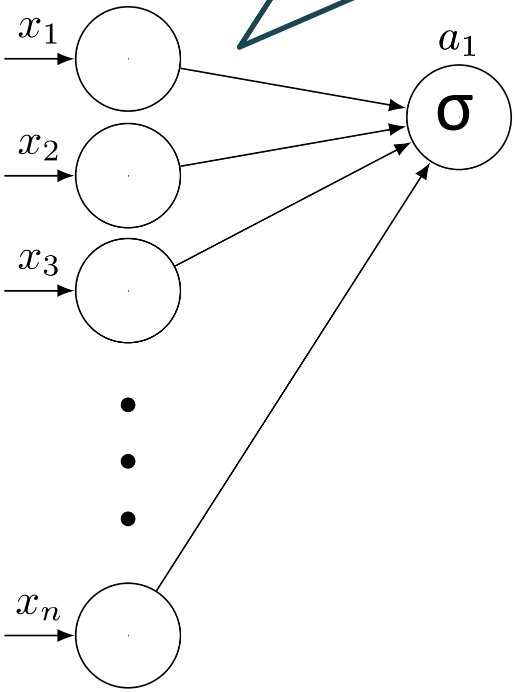


Neural Networks

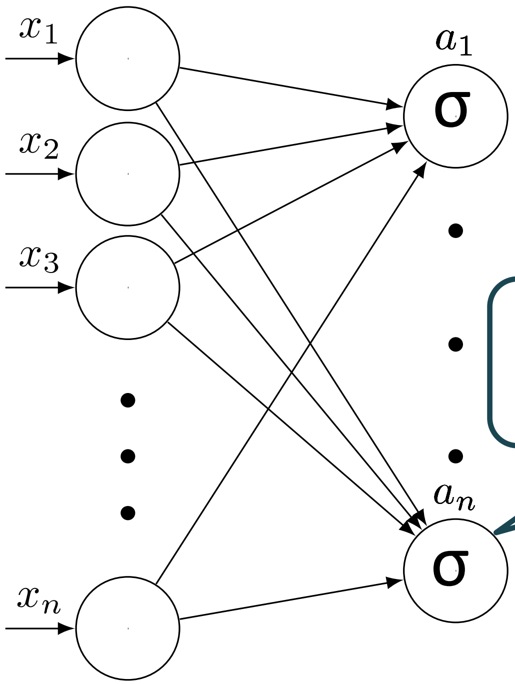
- Combining neural units into larger networks to model complex relationships between inputs and outputs.

Neural Networks

The lines represent the weights ‘w’ in the function, which are initialized randomly, then updated based on data



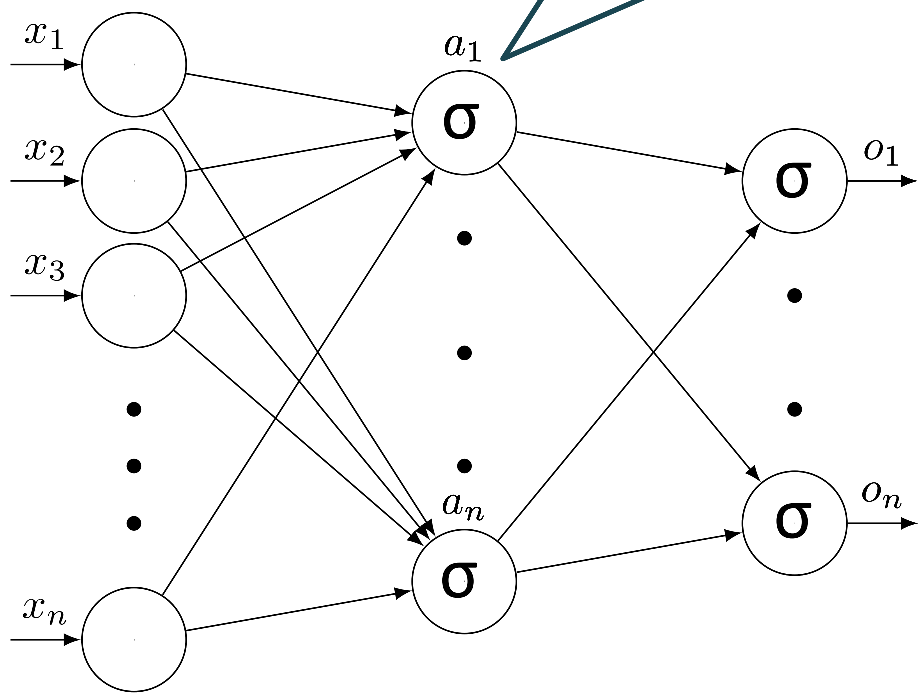
Neural Networks



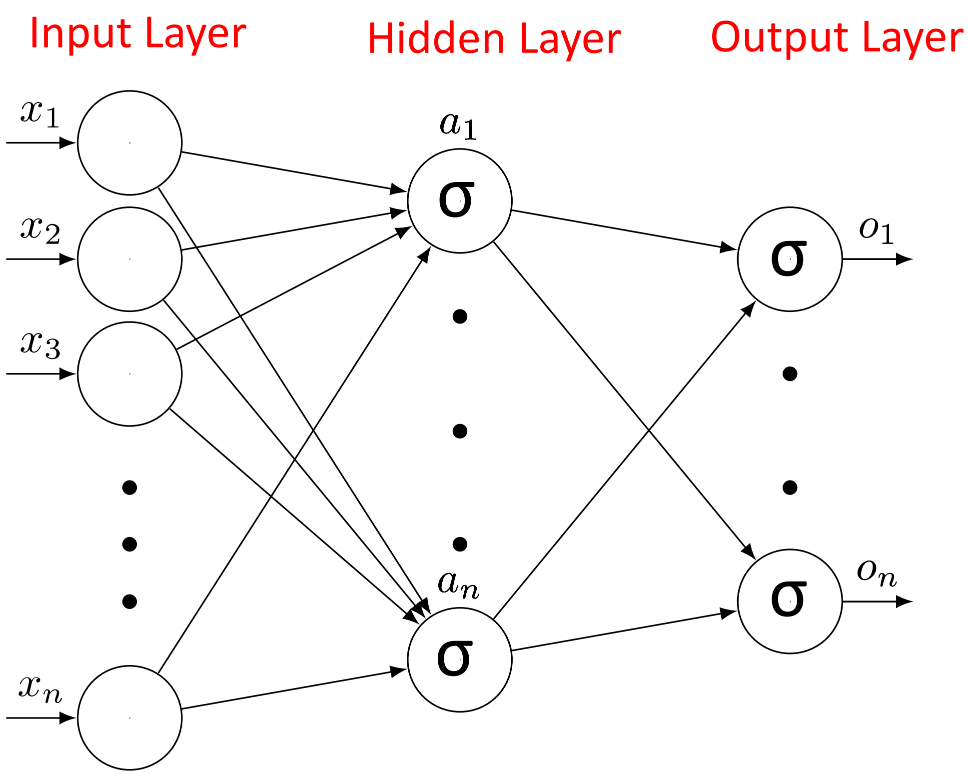
We can have multiple identical functions connected to the same input, each with a different set of weights.

Neural Networks

The outputs here can serve as input in for the next layer, and the same process is repeated …



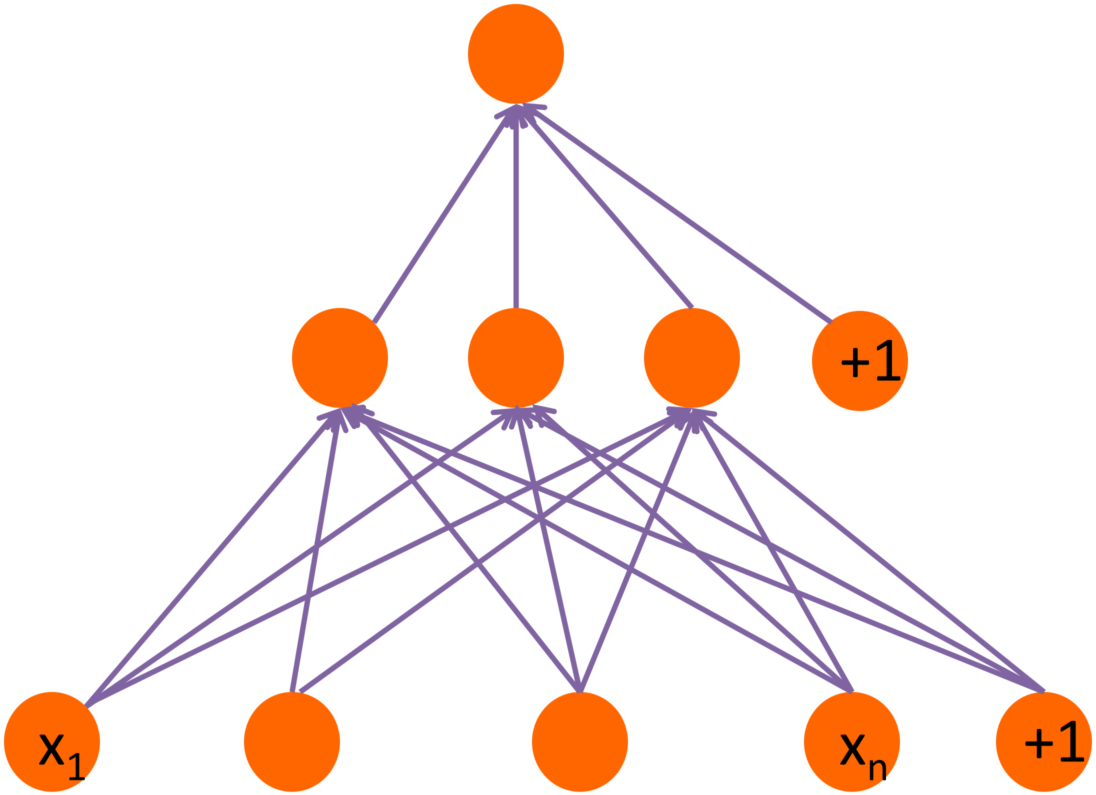
Neural Networks



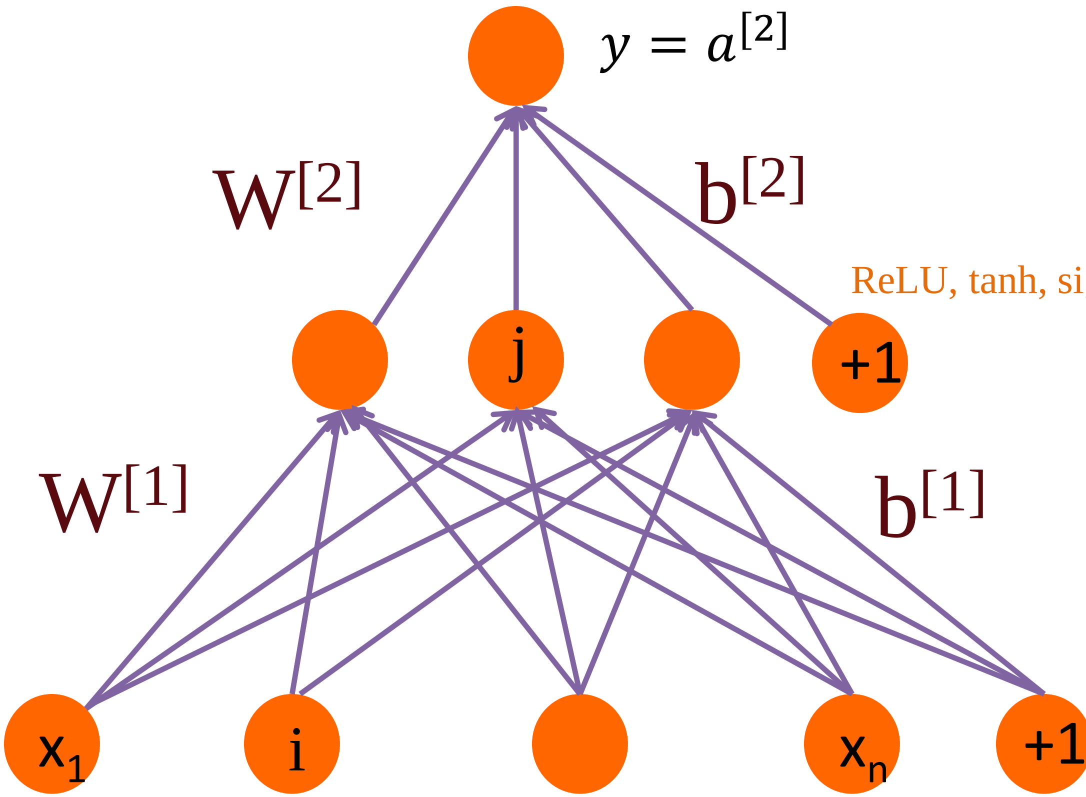
Review

- What is an activation function?
- What activation function would you use in the output layer for the following cases:
  - 5-layer network for binary classification
  - 3-layer network for multi-class classification (e.g. 5 classes)

Review: Notation & Formulae



Review



sigmoid or softmax

$$a^{[2]} = g^{[2]}(z^{[2]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

ReLU, tanh, sigmpod

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$a^{[0]}$

Exercise

- Design a feed-forward neural network for sentiment classification
  - **Input**: one sentence
  - **Output**: positive or negative

Exercise

- Design a feed-forward neural network for sentiment classification
  - **Input**: one sentence
  - **Output**: positive or negative

- Design a feed-forward neural network for entity classification:
  - **Input**: sentence
  - **Output:** One tag per word, 4 classes: person, organization, location, NONE

Exercise

- Design a feed-forward neural network for sentiment classification
  - **Input**: one sentence
  - **Output**: positive or negative

- Design a feed-forward neural network for entity classification:
  - **Input**: sentence
  - **Output:** One tag per word, 4 classes: person, organization, location, NONE

What are the problems?

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks

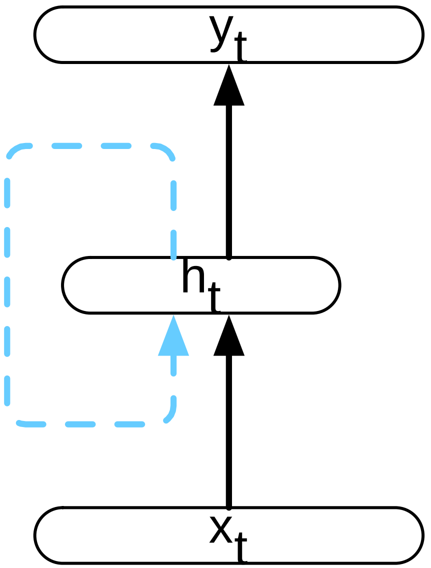
- The problem with feed-forward NN is that the input and the output lengths are fixed
  - Not all problems can be converted into one with fixed-length inputs and outputs

- Recurrent neural networks allow us to work with variable input length

Recurrent Neural Networks

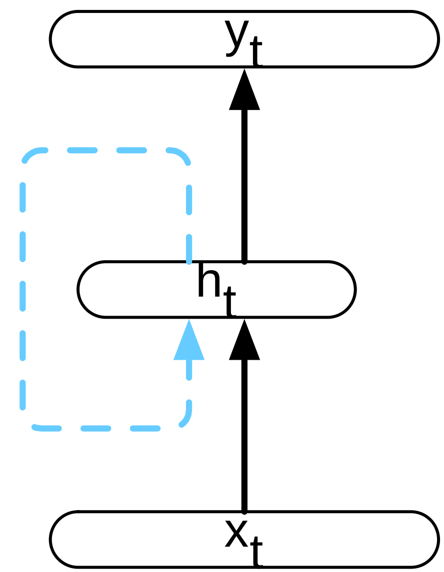
- RNNs contain **cycles** with the network connections
  - The output of a unit is used as input to itself
  - This means the **activation** from one time step will augment the input in the next time step

What do you think this is?



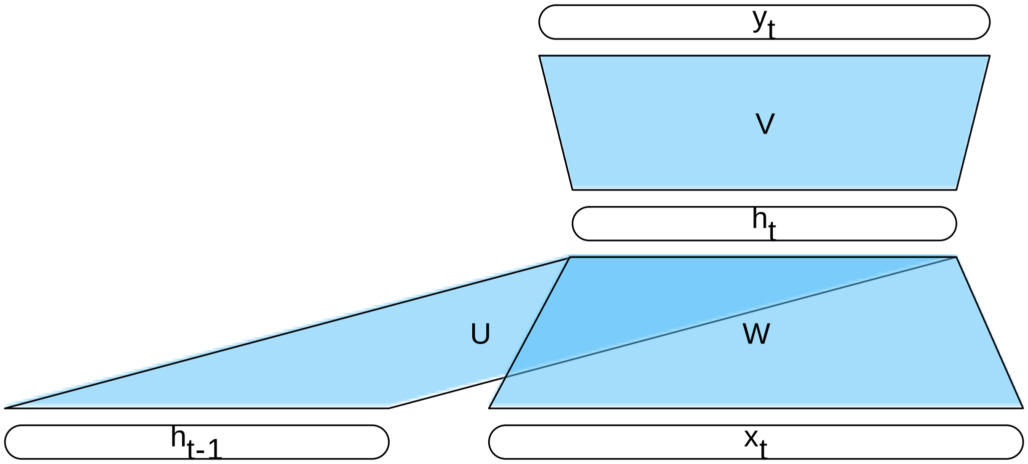
Recurrent Neural Networks

- RNNs contain **cycles** with the network connections
  - The output of a unit is used as input to itself
  - This means the **activation** from one time step will augment the input in the next time step
    - In a way, this is a form of **memory**. The network remembers the previous inputs through the recurrence



Recurrent Neural Networks

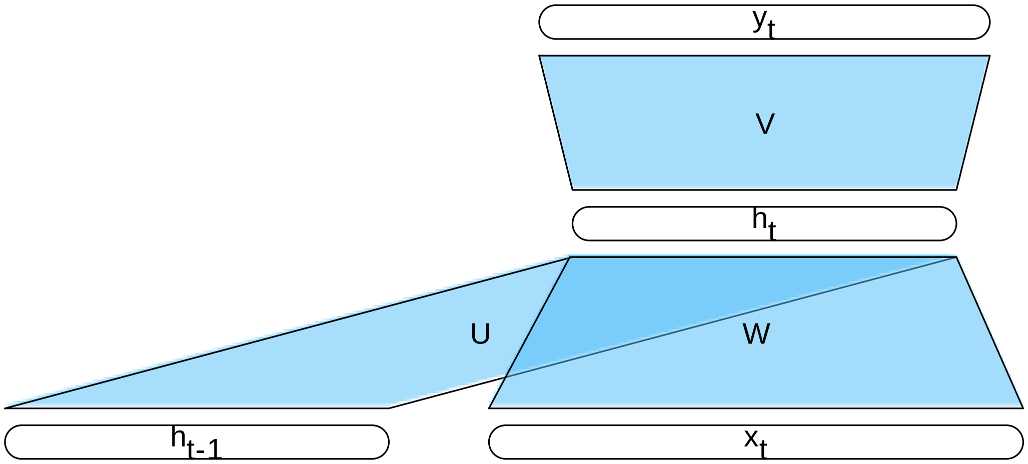
- The content of the previous hidden layer encodes information about all previous inputs



Recurrent Neural Networks

- The content of the previous hidden layer encodes information about all previous inputs

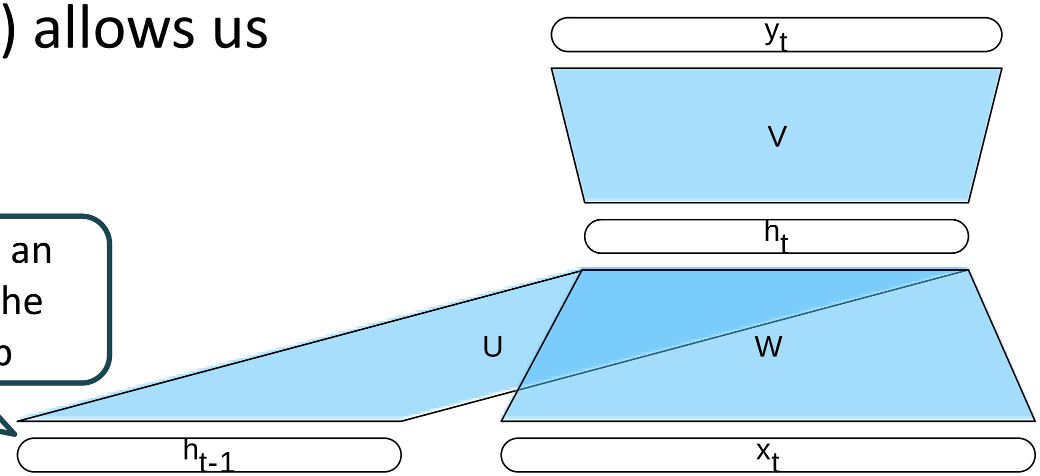
How?



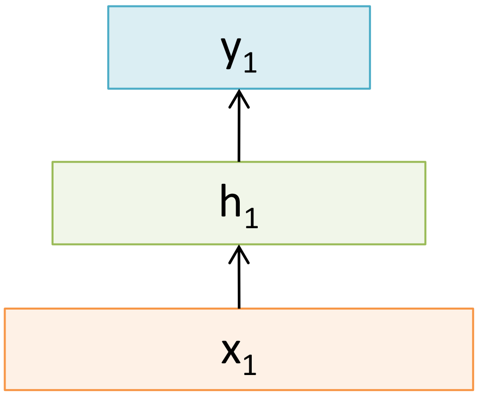
Recurrent Neural Networks

- The content of the previous hidden layer encodes information about all previous inputs
  - Through the recursion
- There is no fixed-length limit of the prior context, which (in theory) allows us to consider the full input

In practice, recursion is implemented as an additional input, which is copied from the hidden state of the previous time step

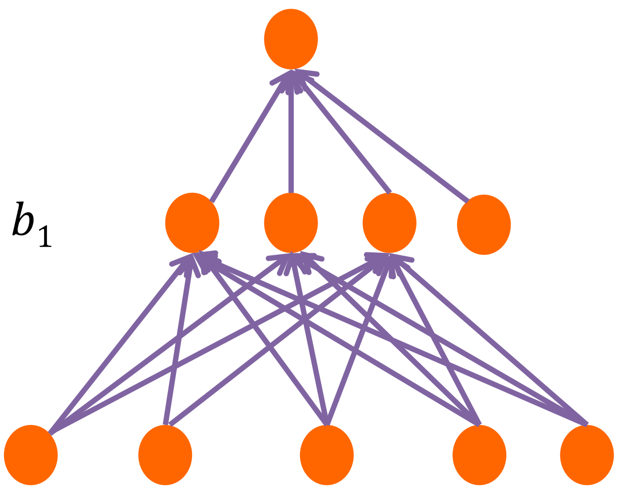


Sample Feed-forward Network

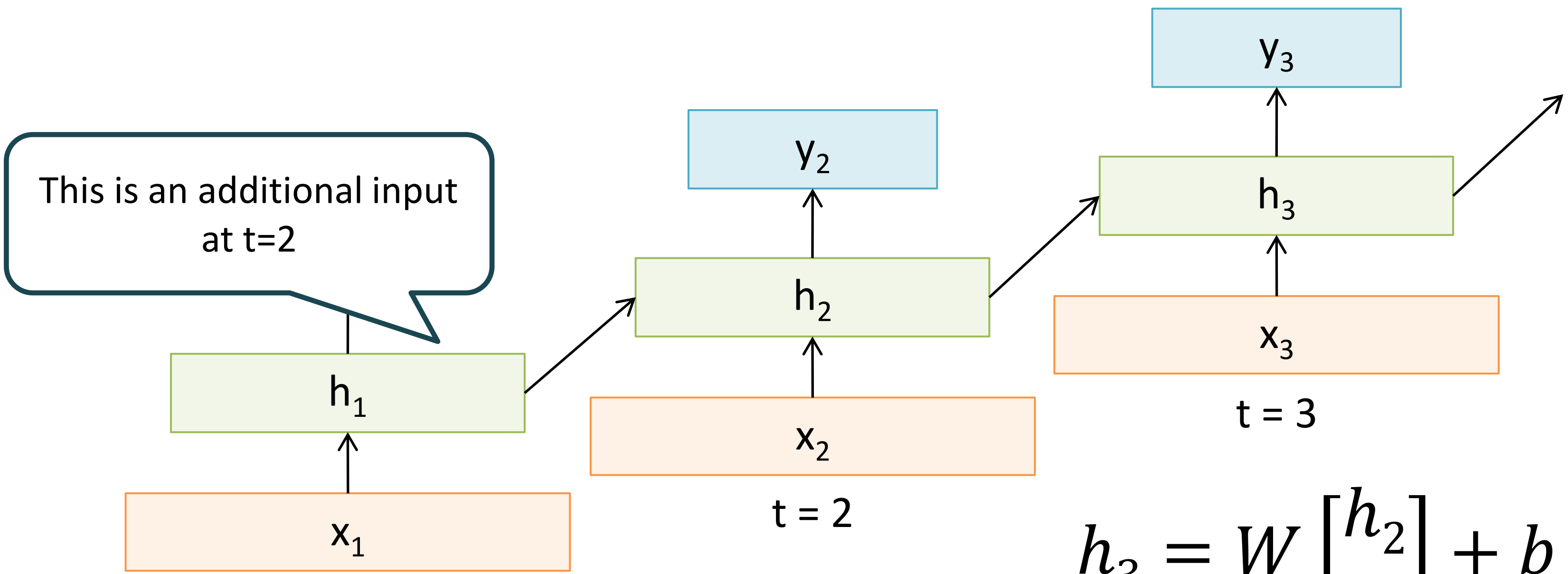


t = 1

$$h_1 = Wx_1 + b_1$$



Sample RNN

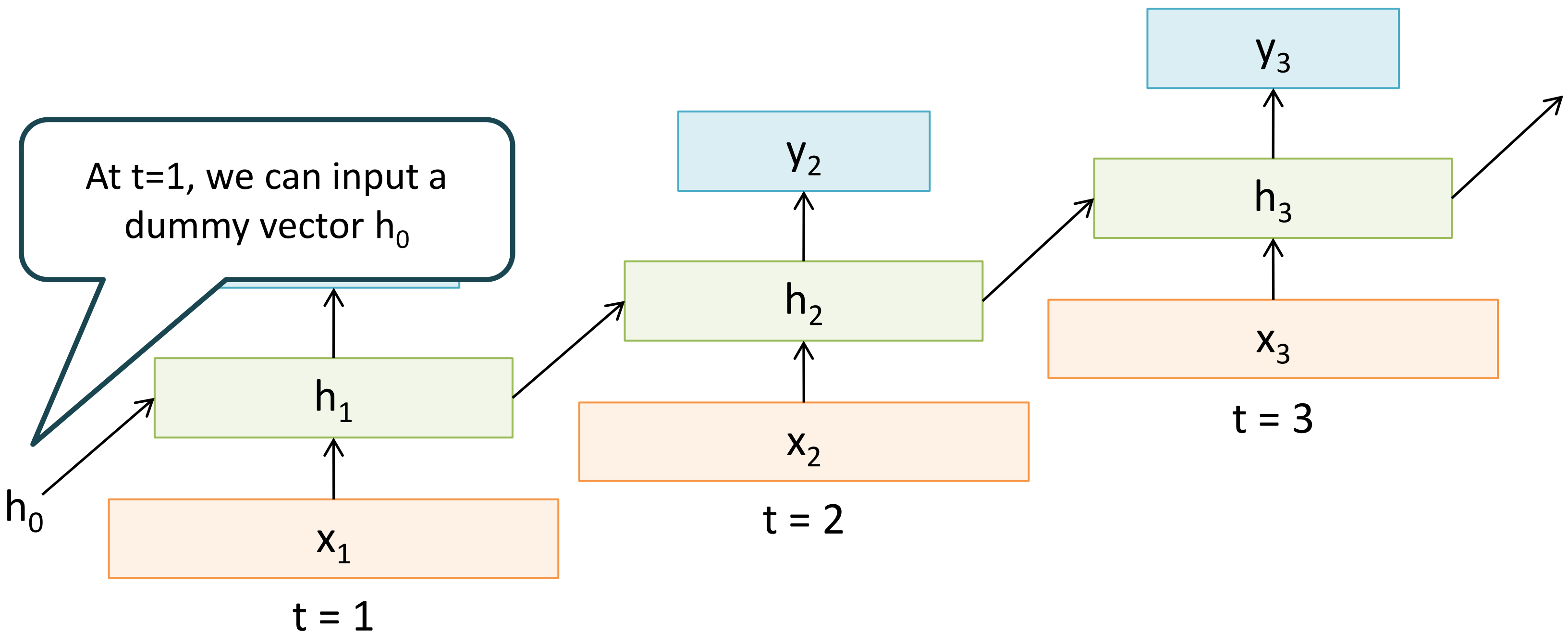


$$h_3 = W \begin{bmatrix} h_2 \\ x_3 \end{bmatrix} + b$$

$$h_2 = W \begin{bmatrix} h_1 \\ x_2 \end{bmatrix} + b$$

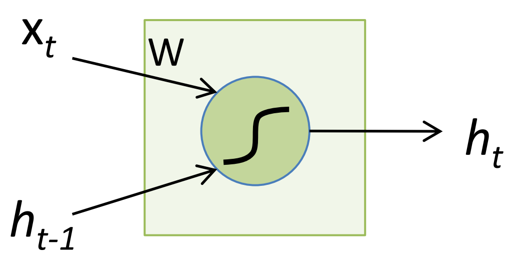
$$h_1 = Wx_1 + b$$

Sample RNN



$$h_t = W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b$$

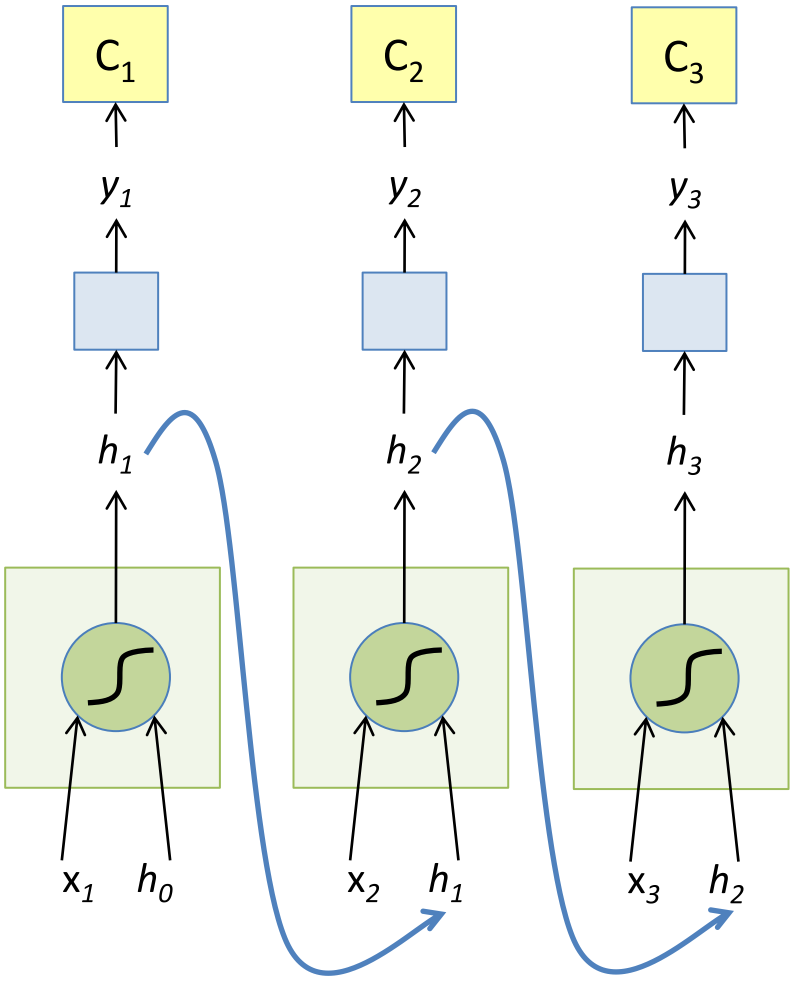
The RNN Cell



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

Nonlinearity

The RNN Forward Pass

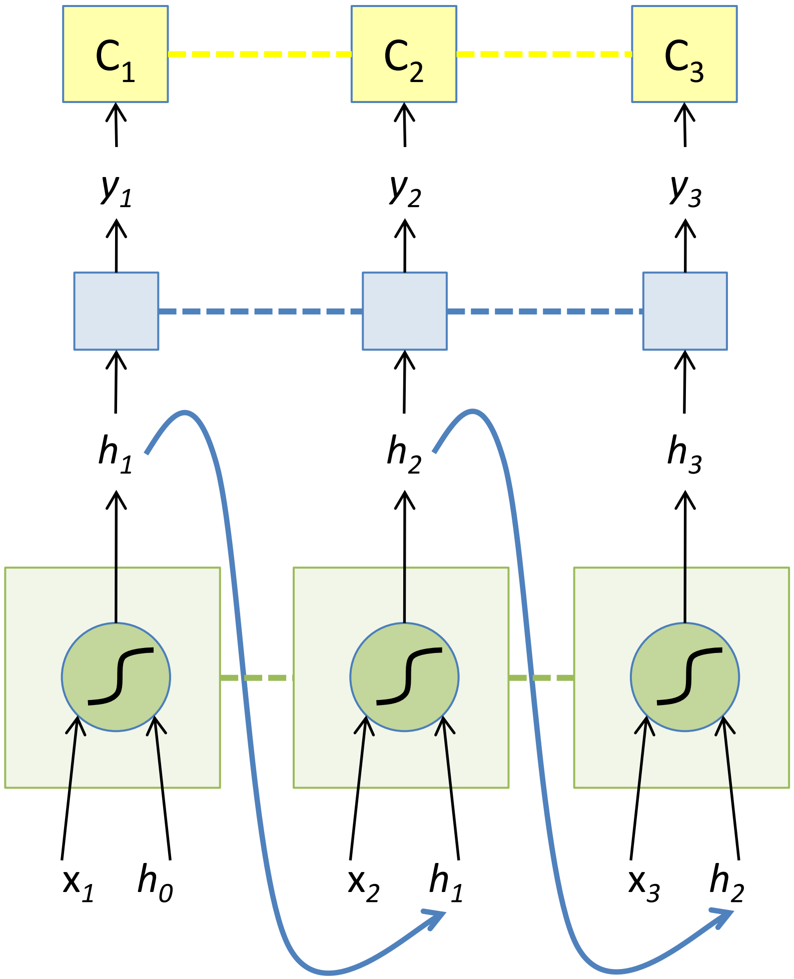


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

The RNN Forward Pass



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

------ indicates shared weights

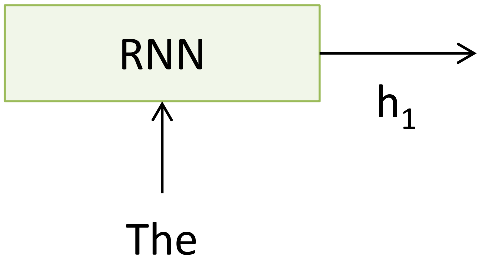
Recurrent Neural Networks (RNNs)

- Note that the weights are shared over time steps
  - Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

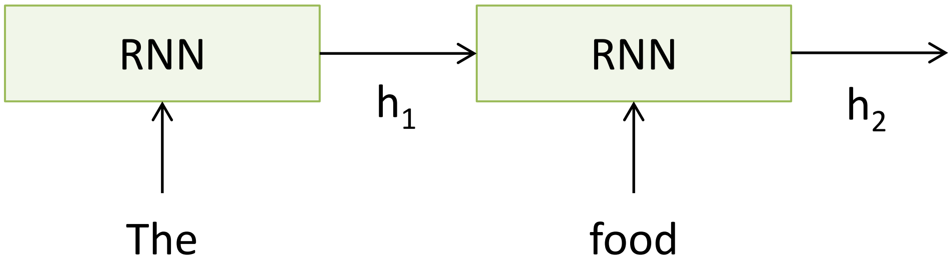
Exercise

- Design a recurrent neural network for sentiment classification
  - **Input**: one sentence
  - **Output**: positive or negative

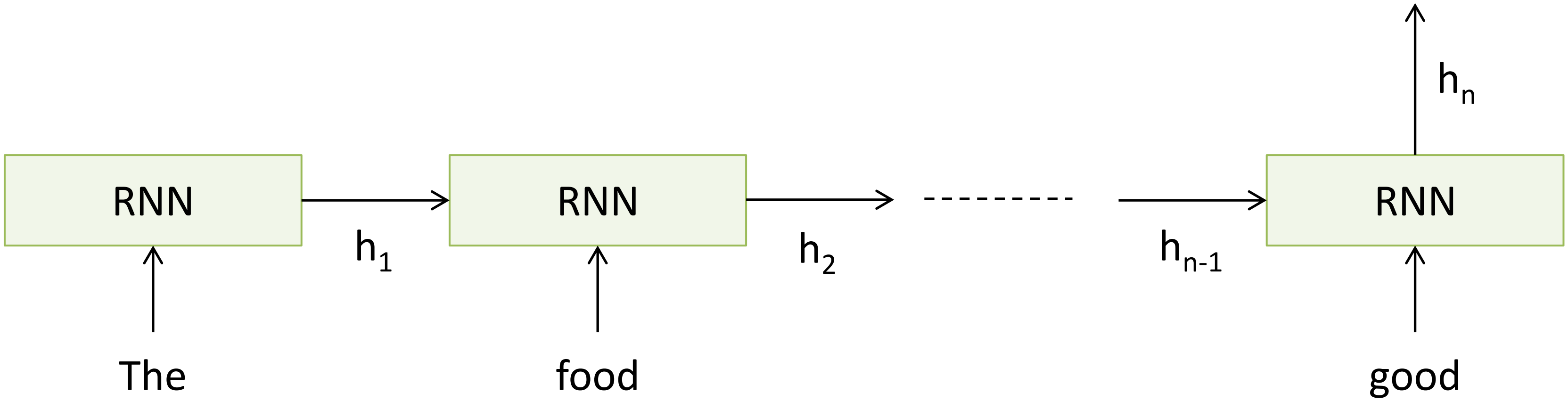
Sentiment Classification



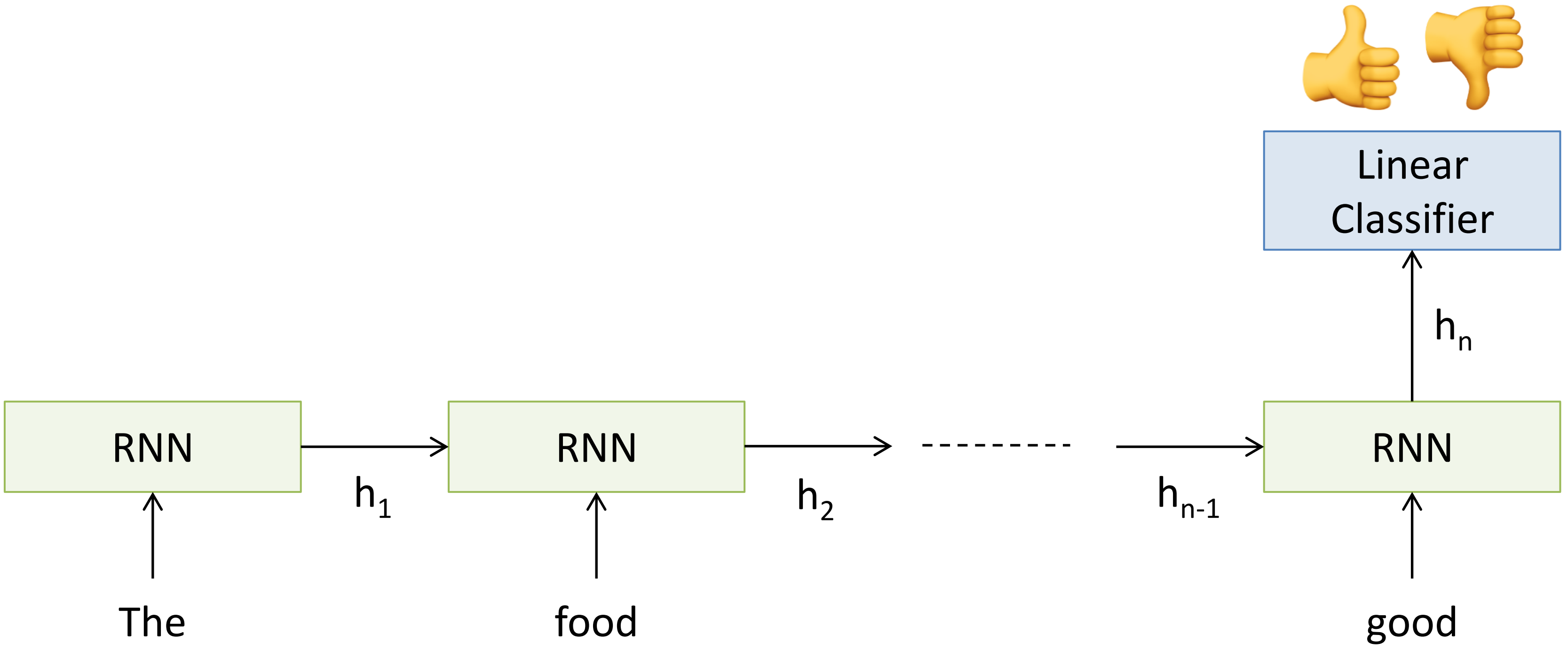
Sentiment Classification



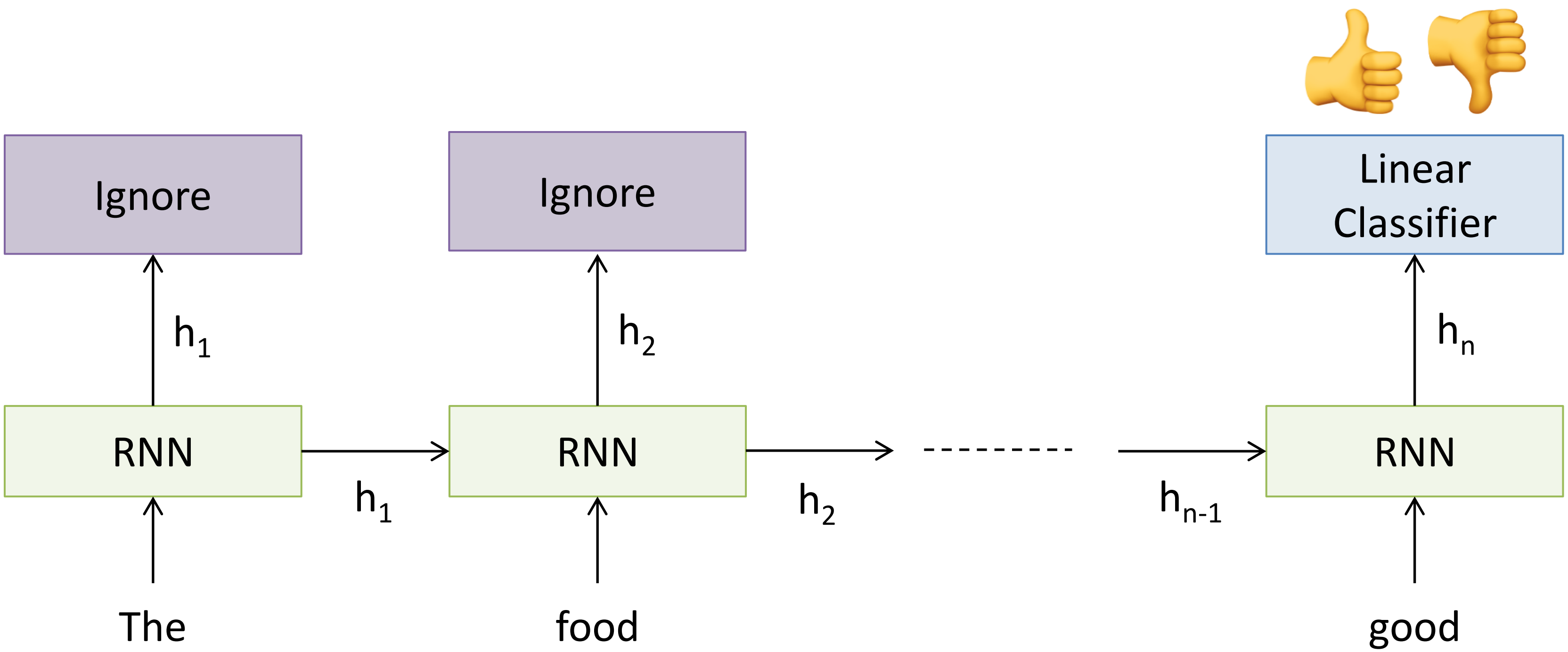
Sentiment Classification



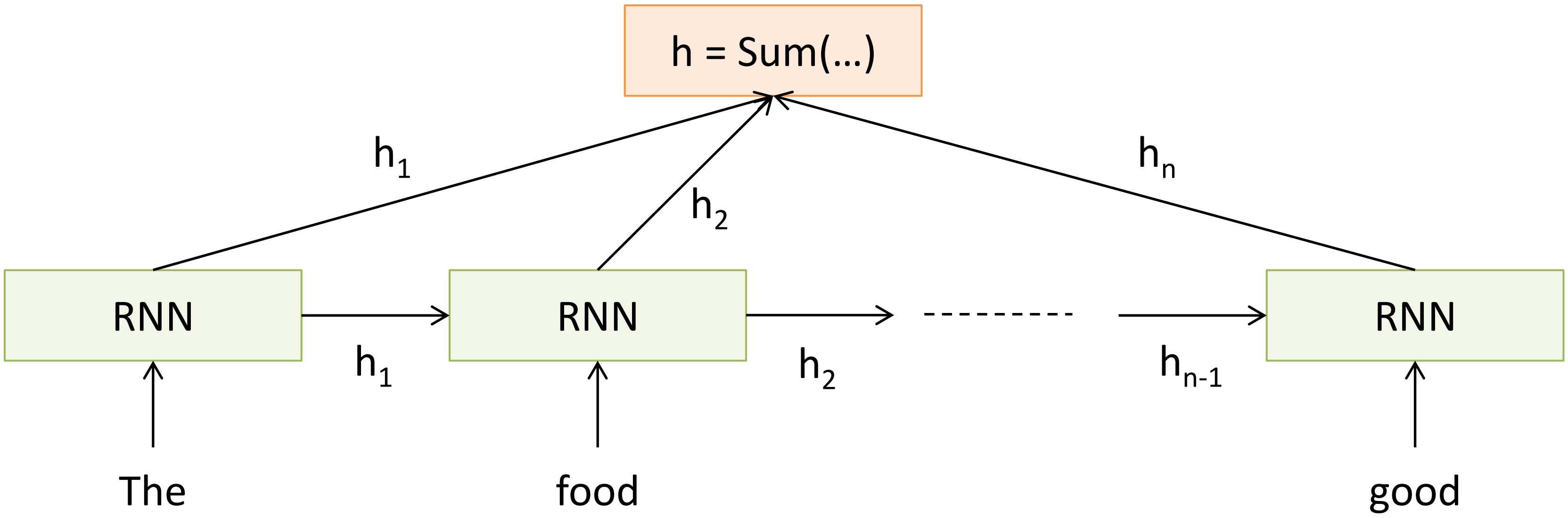
Sentiment Classification



Sentiment Classification



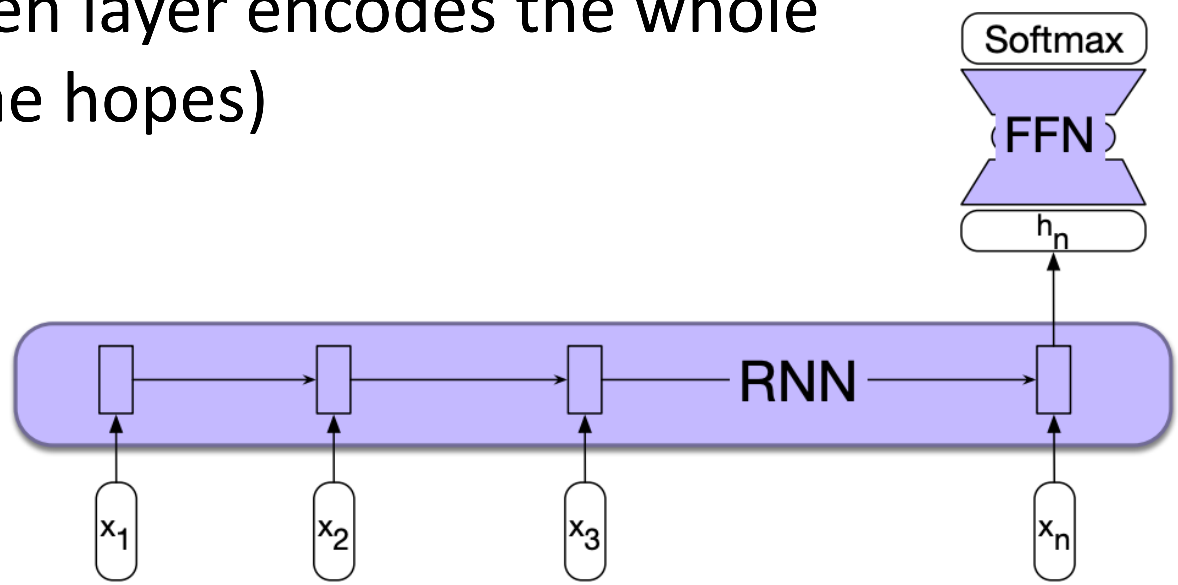
Sentiment Classification



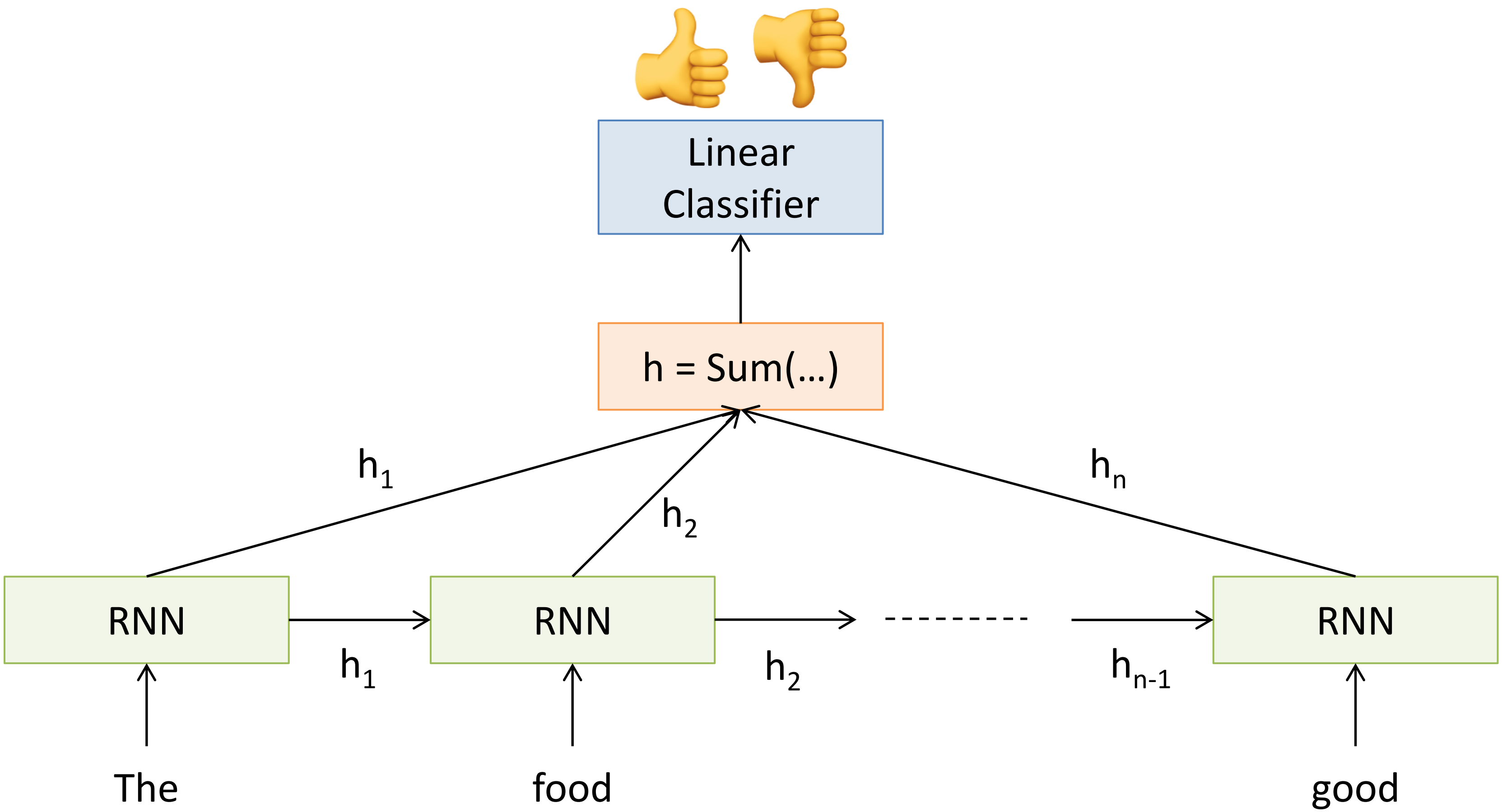
http://deeplearning.net/tutorial/lstm.html

Text classification with RNNs

- The output layer is only applied after the last word in the sentence
- The last hidden layer encodes the whole sequence (one hopes)



Sentiment Classification



http://deeplearning.net/tutorial/lstm.html

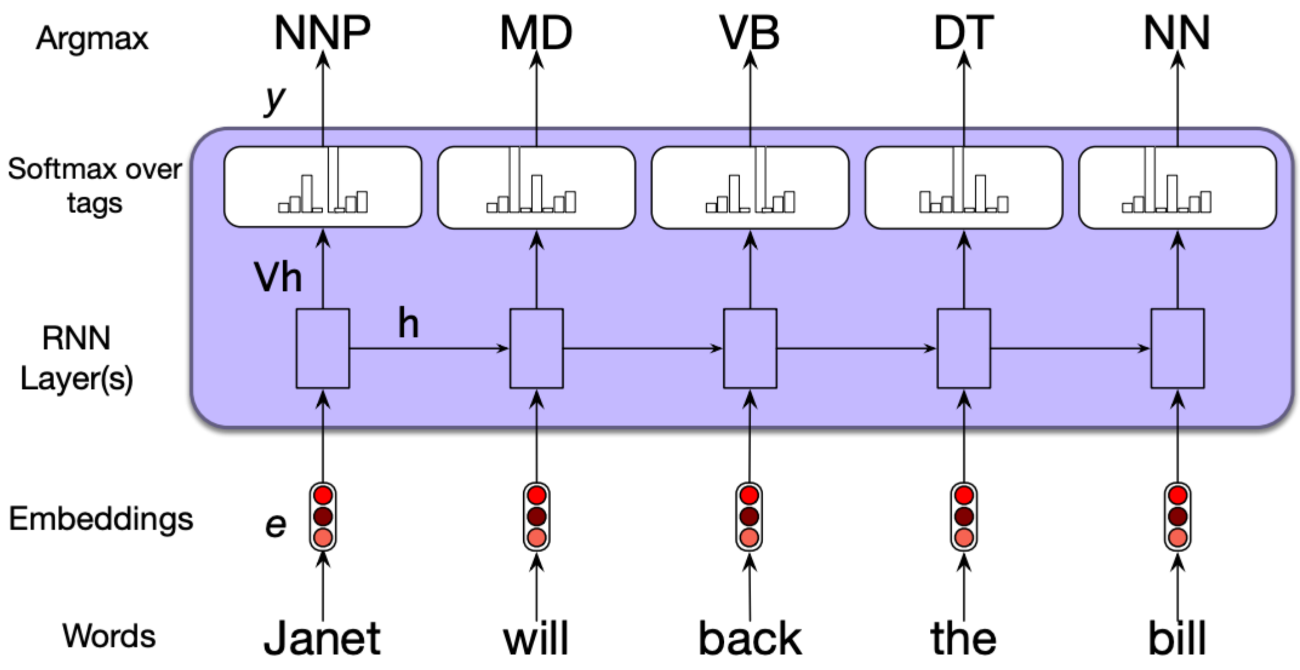
What is the motivation for each of the two models?

Exercise

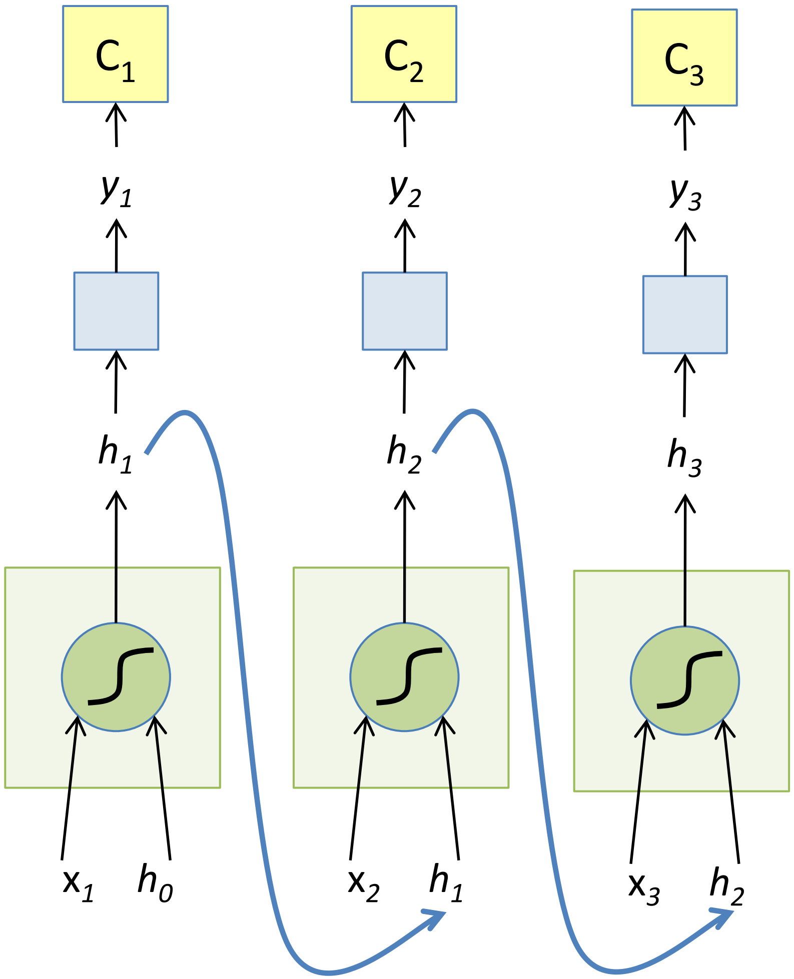
- Design a recurrent neural network for entity classification:
  - **Input**: sentence
  - **Output:** One tag per word, 4 classes: person, organization, location, NONE

Sequence Labeling with RNNs

- **Example**: Parts-of-speech tagging



The RNN Forward Pass



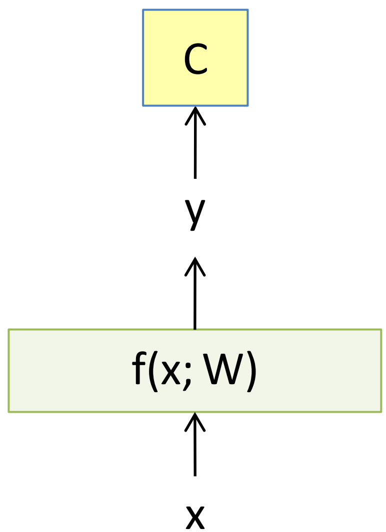
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

"Unfold" network through time by making copies at each time-step

Backpropagation Refresher



$$y = f(x; W)$$

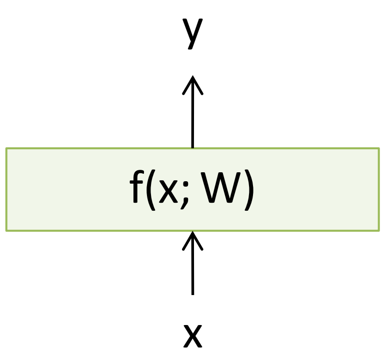
$$C = \text{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right)$$

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

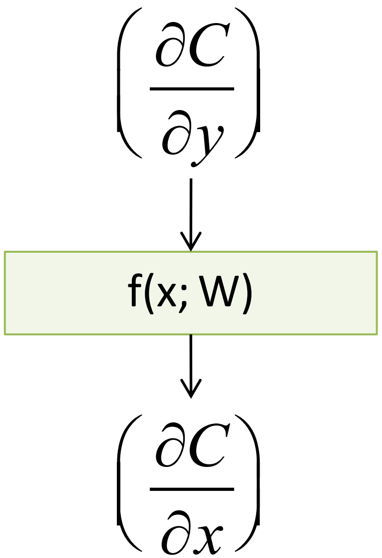
Intrinsic to the layer are:

$\left(\frac{\partial y}{\partial W}\right)$ - How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ - How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

Intrinsic to the layer are:

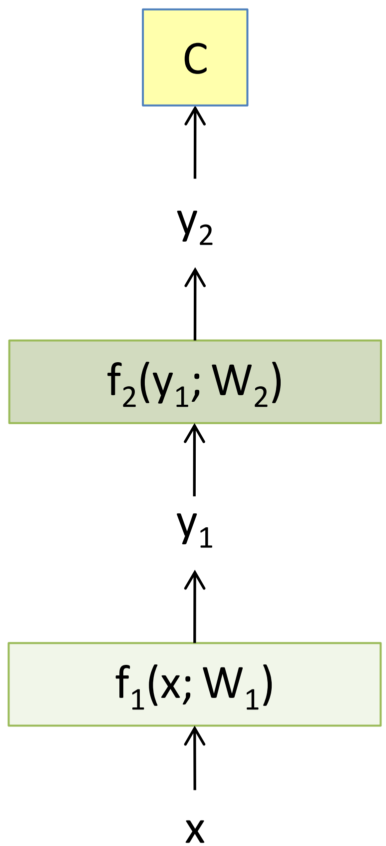
$\left(\frac{\partial y}{\partial W}\right)$ - How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ - How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Equations for common layers: http://arunmallya.github.io/writeups/nn/backprop.html

Multiple Layers



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

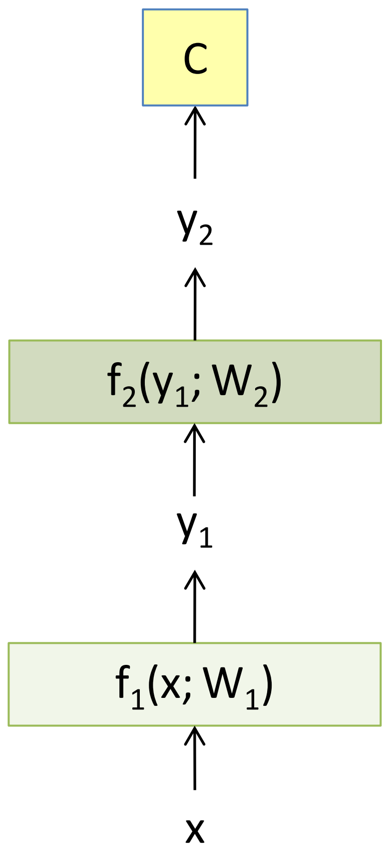
$$C = \mathrm{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

Chain Rule for Gradient Computation



$$y_1 = f_1(x;W_1)$$

$$y_2 = f_2(y_1;W_2)$$

$$C = \mathrm{Loss}(y_2, y_{GT})$$

$$\text{Find } \frac{\P C}{\P W_1}, \frac{\P C}{\P W_2}$$

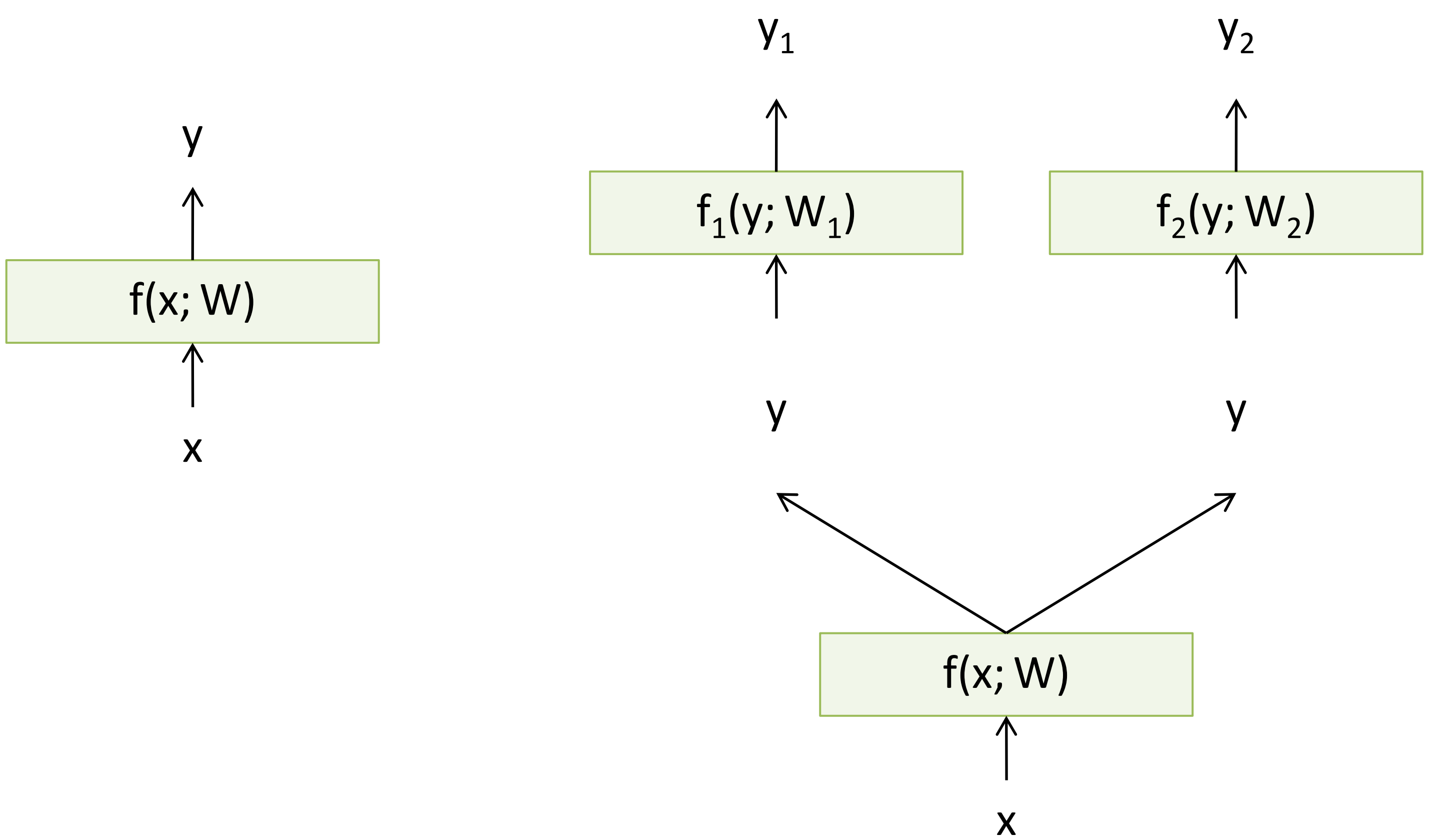
$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

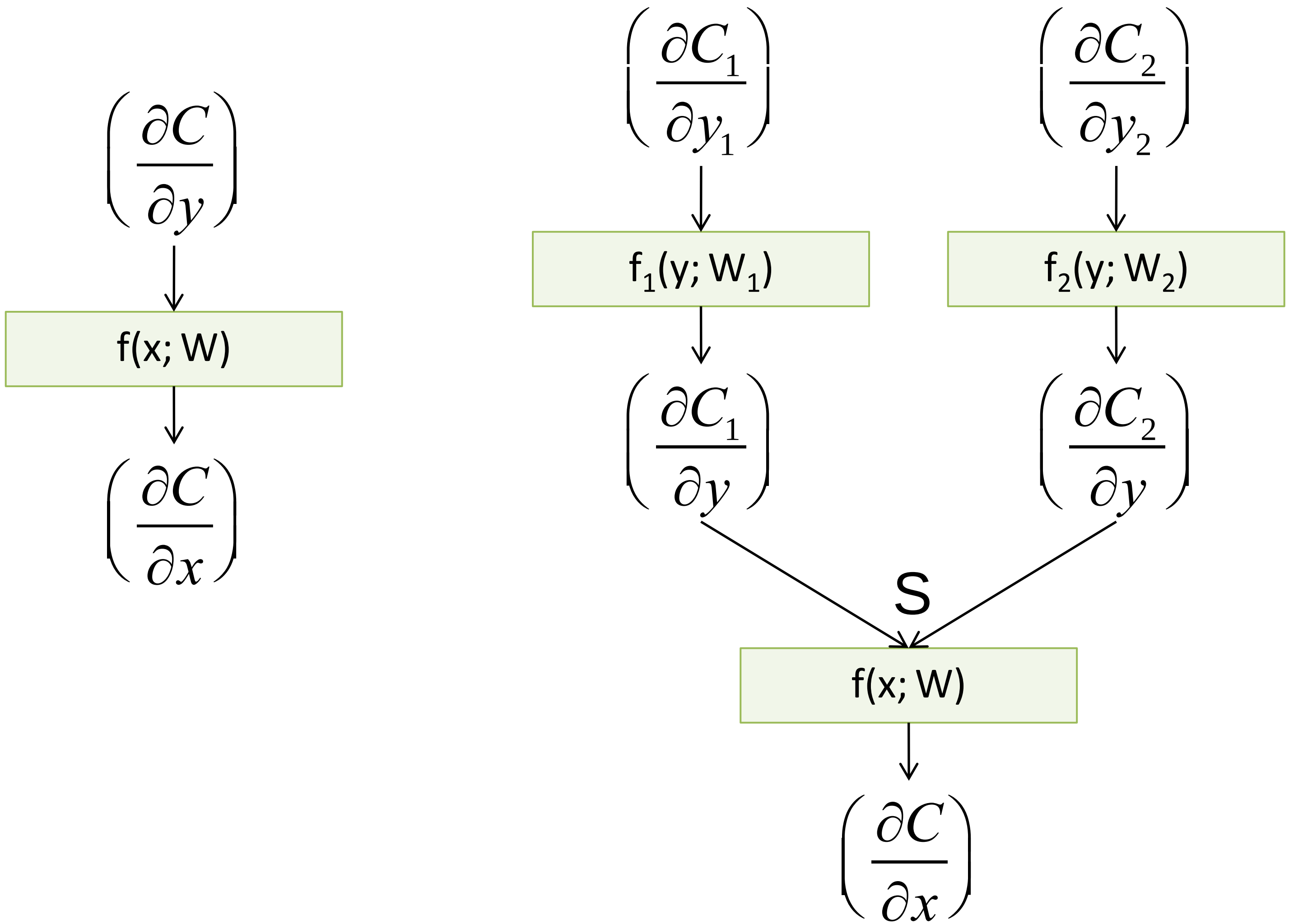
$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

Application of the Chain Rule

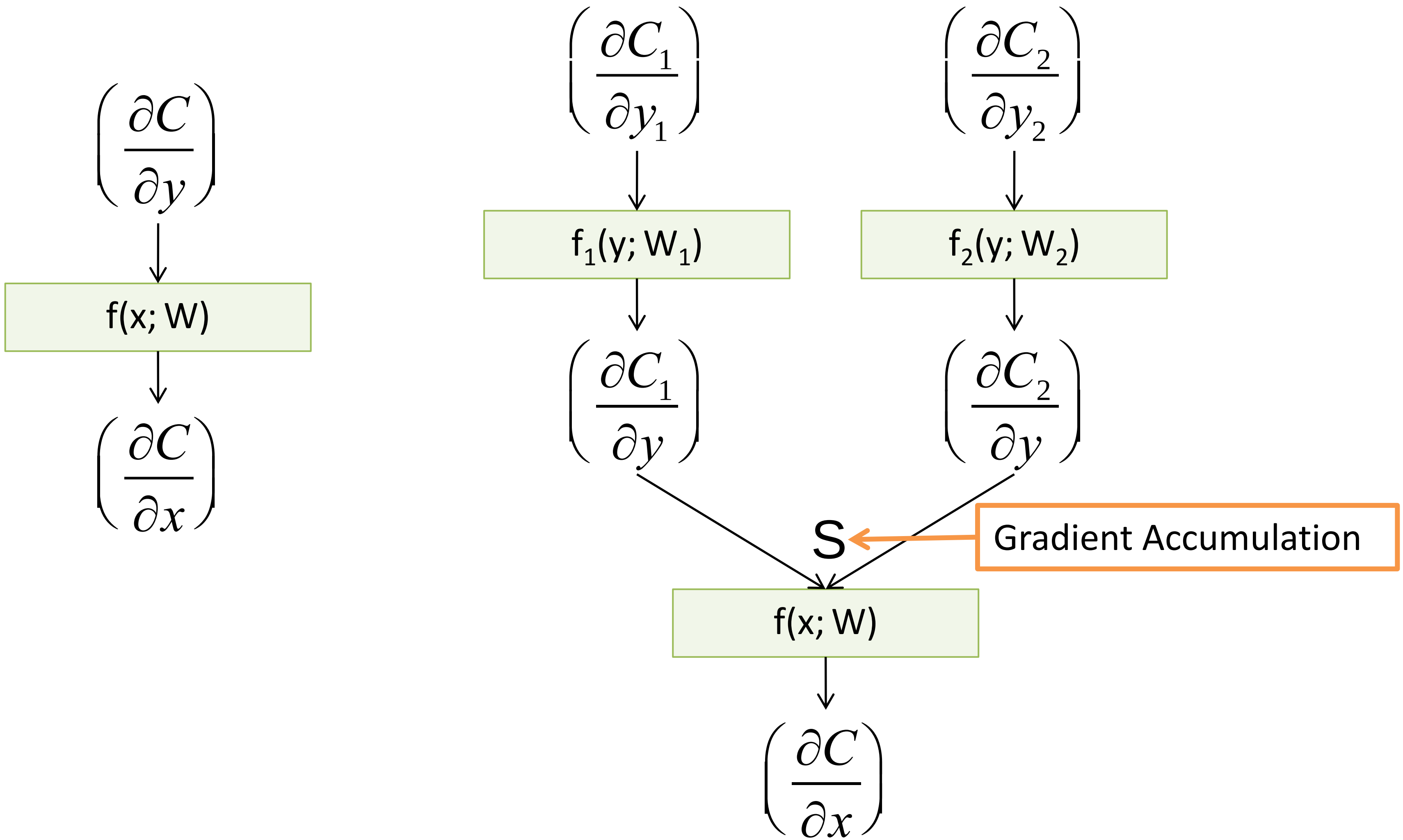
Extension to Computational Graphs



Extension to Computational Graphs



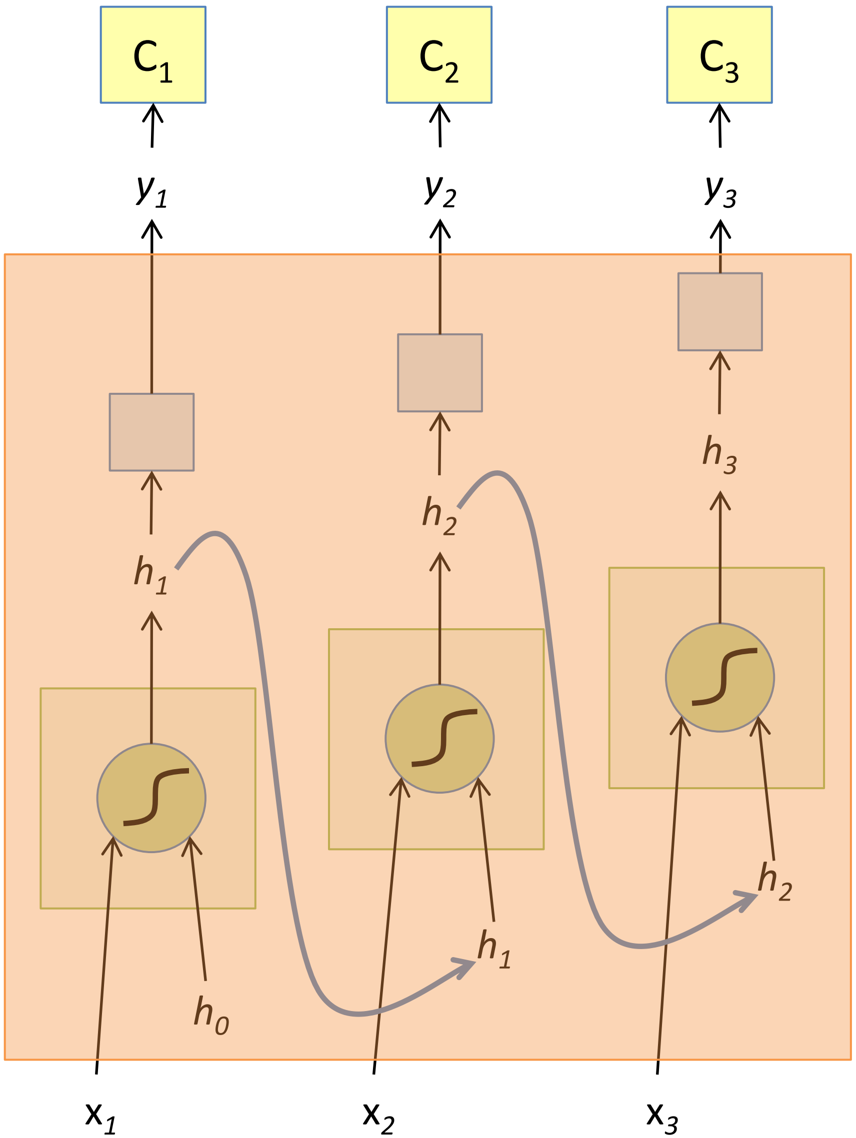
Extension to Computational Graphs



Backpropagation Through Time (BPTT)

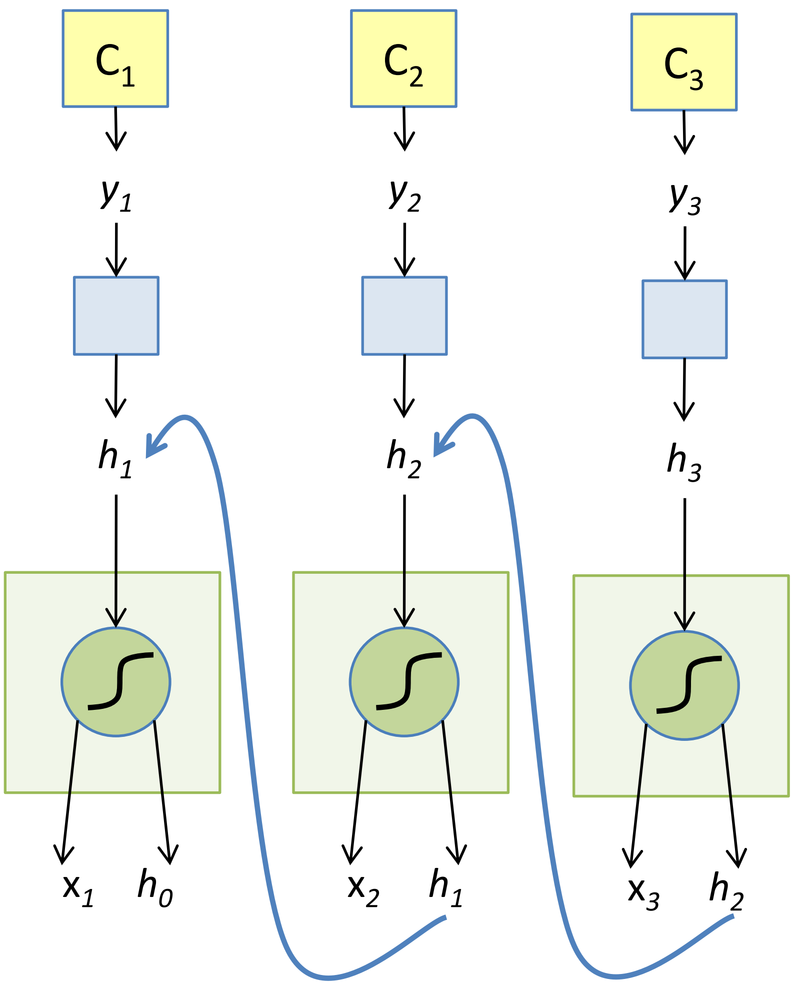
- BPTT is used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
  - This unfolded network accepts the whole time series as input
  - The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded RNN



- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

The Unfolded RNN Backward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

$$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

Problems with RNNs

- In NLP, using RNNs for classification of long sequences has many drawbacks
  - Simple RNNs typically encode recent words; information from more distant context can be lost
  - The errors backpropagated to earlier time steps tend to either get too small or too large, which complicates training

Why is this?

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.

The problem of exploding or vanishing gradients

- In an RNN trained on long sequences (*e.g.,* 100 time steps) the gradients can easily explode or vanish.
  - So RNNs have difficulty dealing with long-range dependencies.

Issues with the Vanilla RNNs

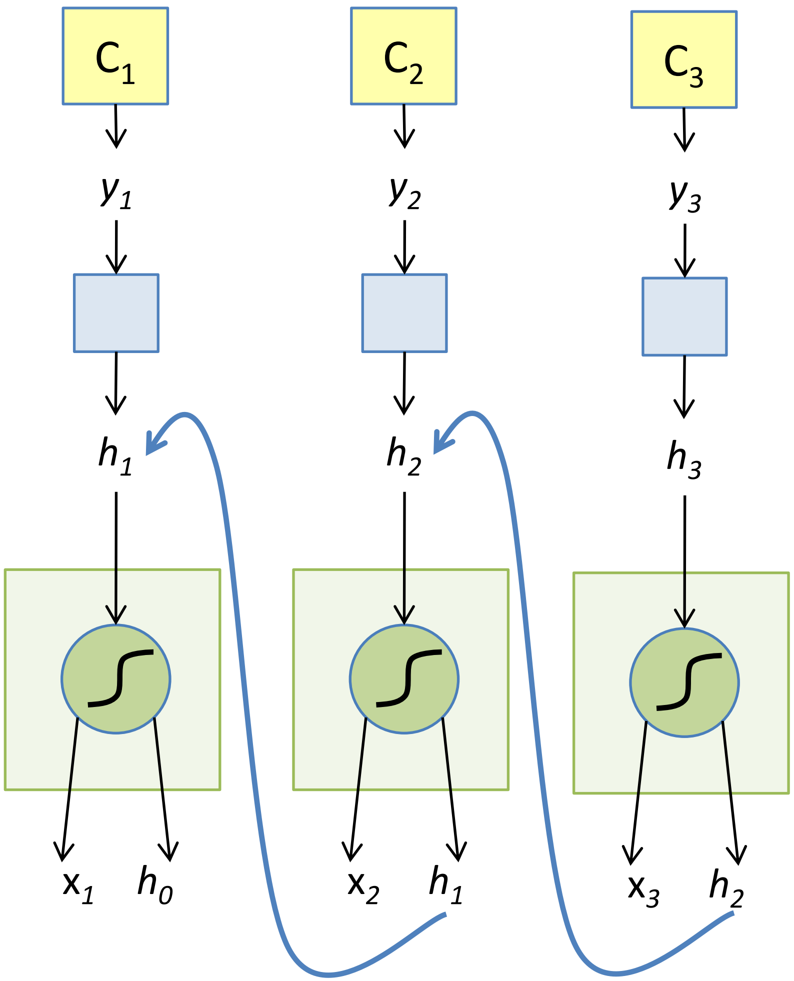
- In the same way, a product of *k* real numbers can shrink to zero or explode to infinity, so can a product of matrices

- Exploding gradients are often controlled with gradient element-wise or norm clipping

[1] On the difficulty of training recurrent neural networks, Pascanu *et al.*, 2013

The Un… ckward

h_t and ${}_{ht\text{-}1}$ are related through matrix multiplication



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t\text{-}1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

These can be very small very gradients corresponding to many hidden layers

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

The Identity Relationship

- Recall $$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$ $$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

What is $\frac{\partial h_t}{\partial h_{t-1}}$

- Suppose that **instead of a matr**… we had an **identity relationship** between the hidden states

$$h_t = h_{t-1} + F(x_t)$$

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

- The gradient does not decay as the error is propagated all the way back aka “Constant Error Flow”

Long Short-Term Memory (LSTM)[1]

- The LSTM uses this idea of “Constant Error Flow” for RNNs to ensure that gradients don’t decay

- The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time
  - which prevents the vanishing gradient issue

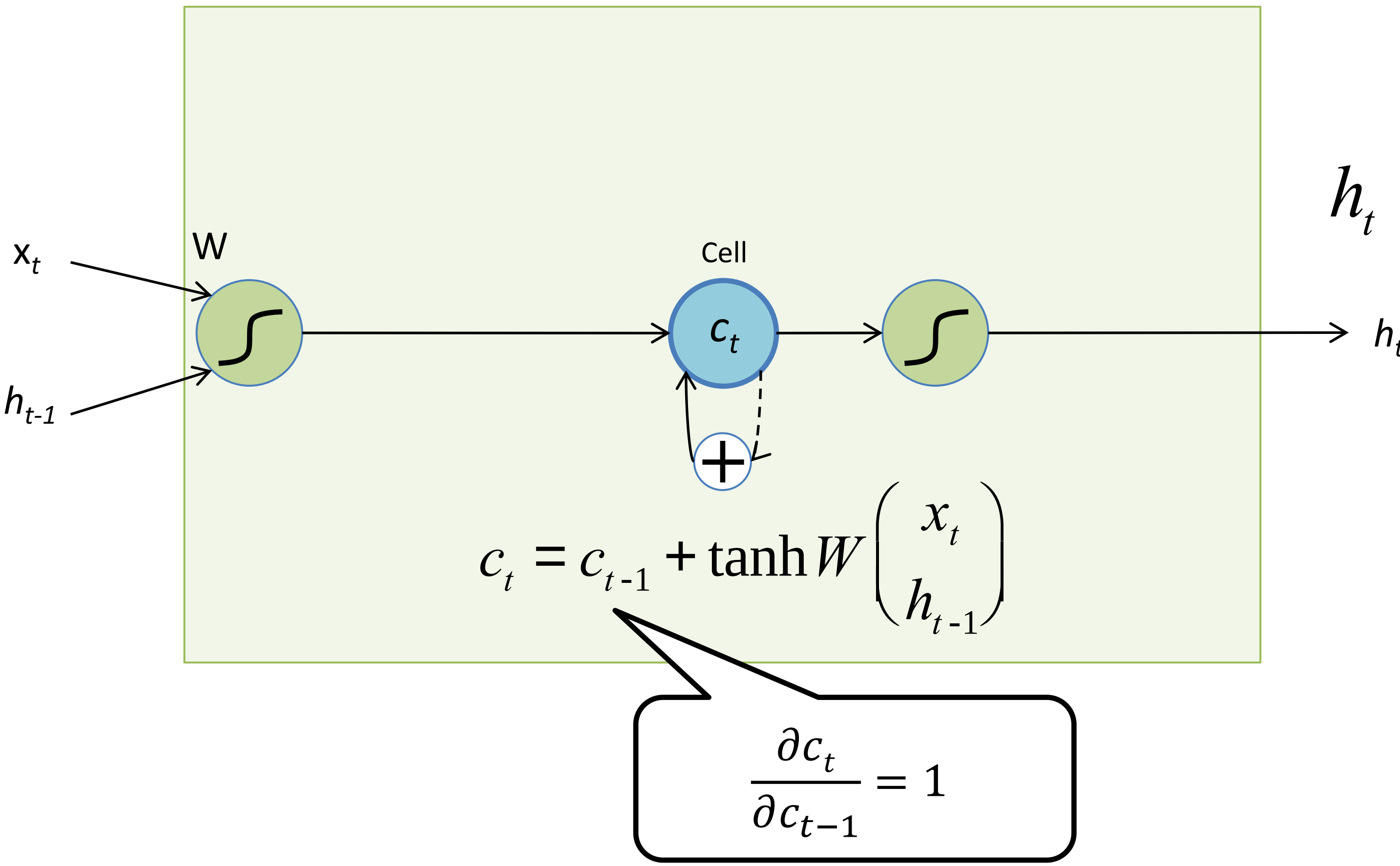
[1] Long Short-Term Memory, Hochreiter *et al*., 1997

Long Short-Term Memory

- **LSTM** cells use additional functions to manage information flow from hidden units to the subsequent time steps
  - These additional units are called **gates.** An LSTM cell has an **input** gate, a **forget** gate, and an **output** gate
- LSTMs are the most widely used architecture in NLP neural models.

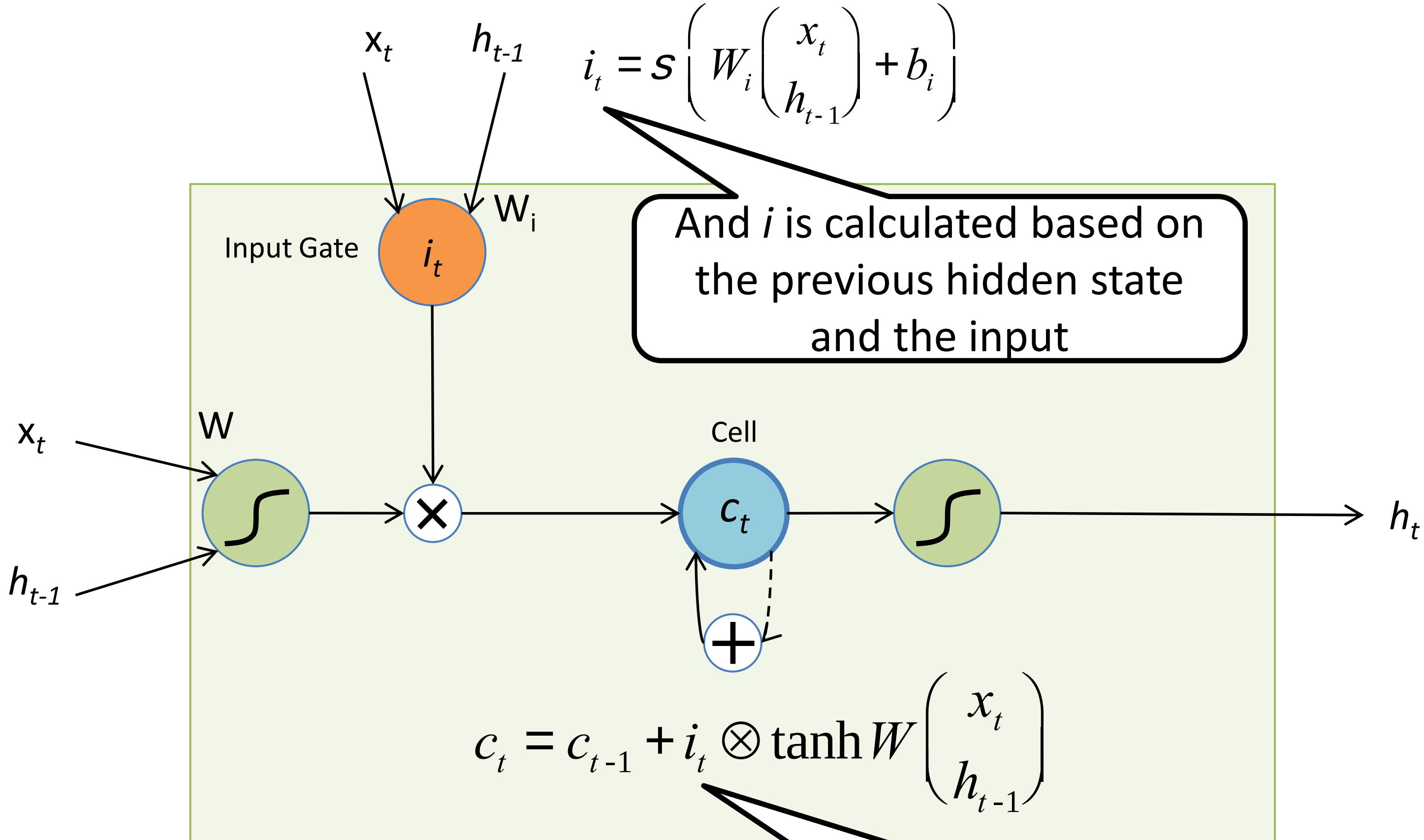
The LSTM Idea

$$h_t = \tanh c_t$$



* Dashed line indicates time-lag

The Original LSTM Cell



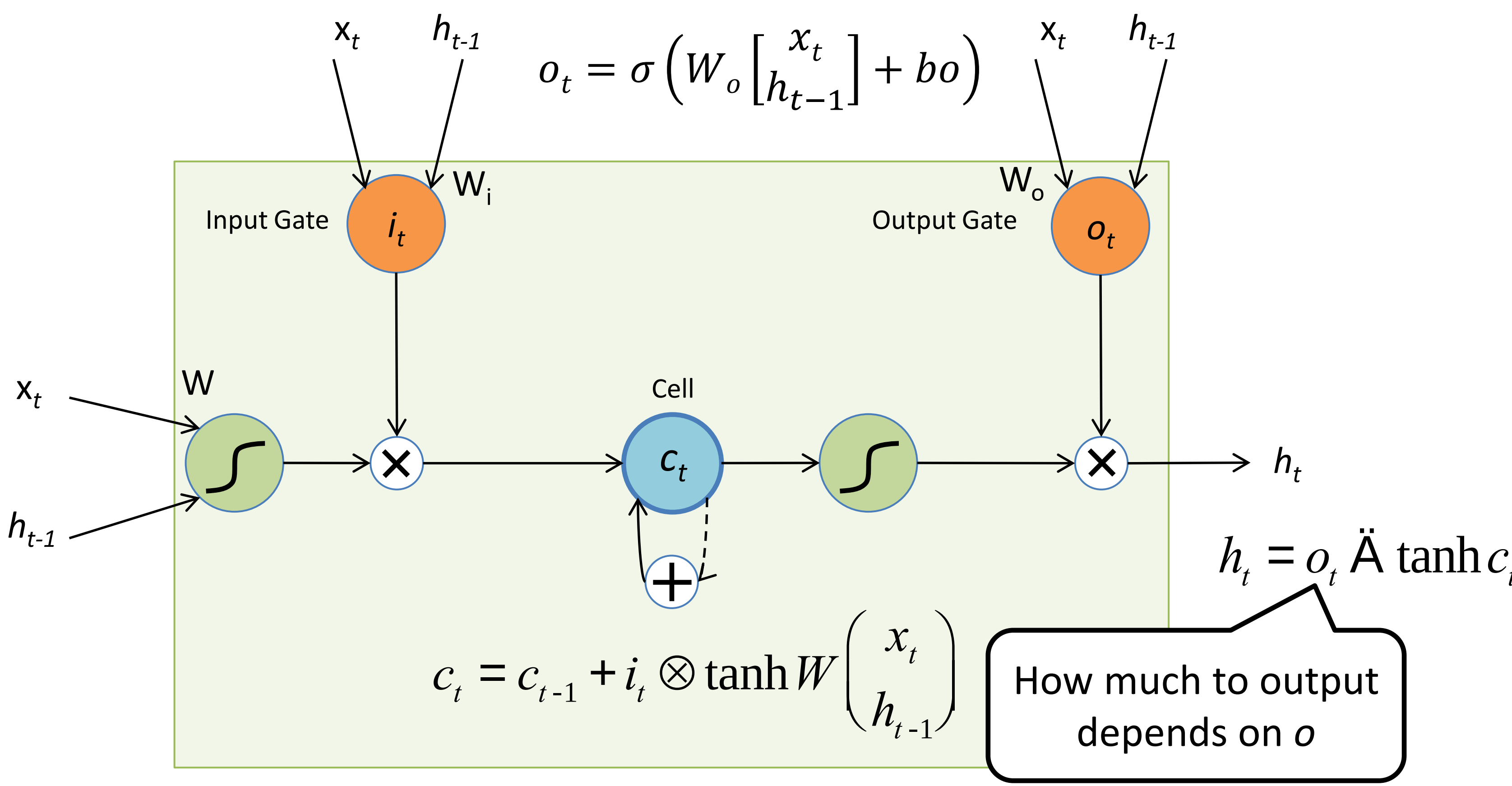
$$i_t = s\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

And *i* is calculated based on the previous hidden state and the input

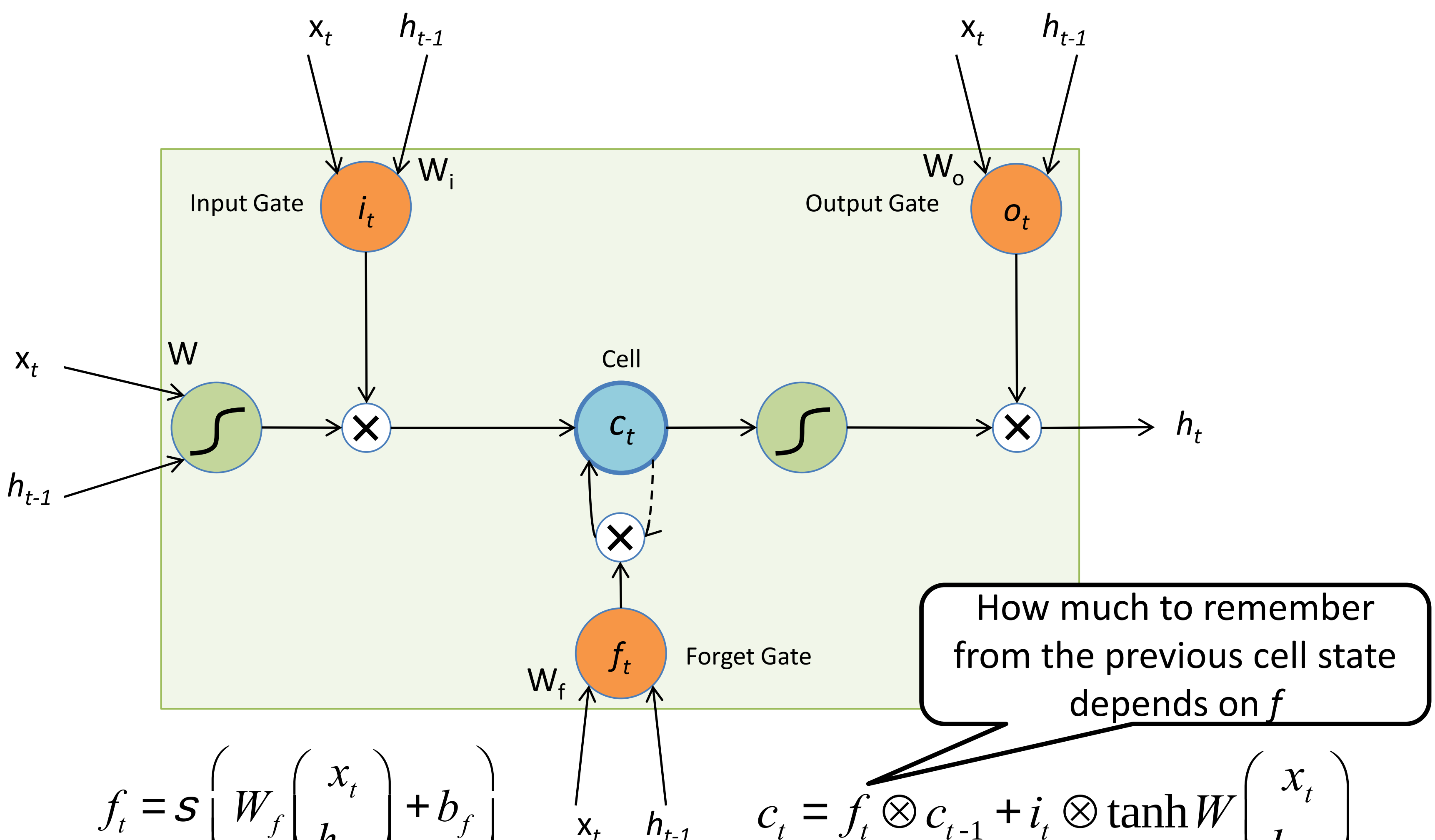
$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

How much to update the cell will depend on the value of *i*

The Original LSTM Cell



The Popular LSTM Cell



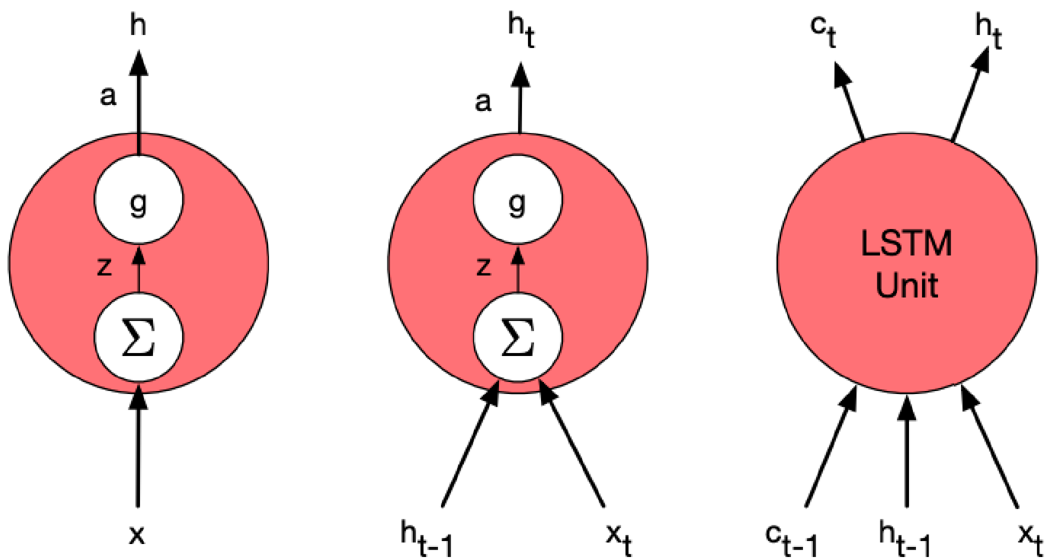
$$f_t = s\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

LSTM – Forward/Backward

Go To: [Illustrated LSTM Forward and Backward Pass]

LSTM



(a) Basic Neural Unit

(b) Recurrent Unit

(c) LSTM Unit

Gated Recurrent Unit (GRU)

- A simplified version of the LSTM
  - Merges the forget and the input gate into a single 'update' gate
  - Merges the memory cell and the hidden state

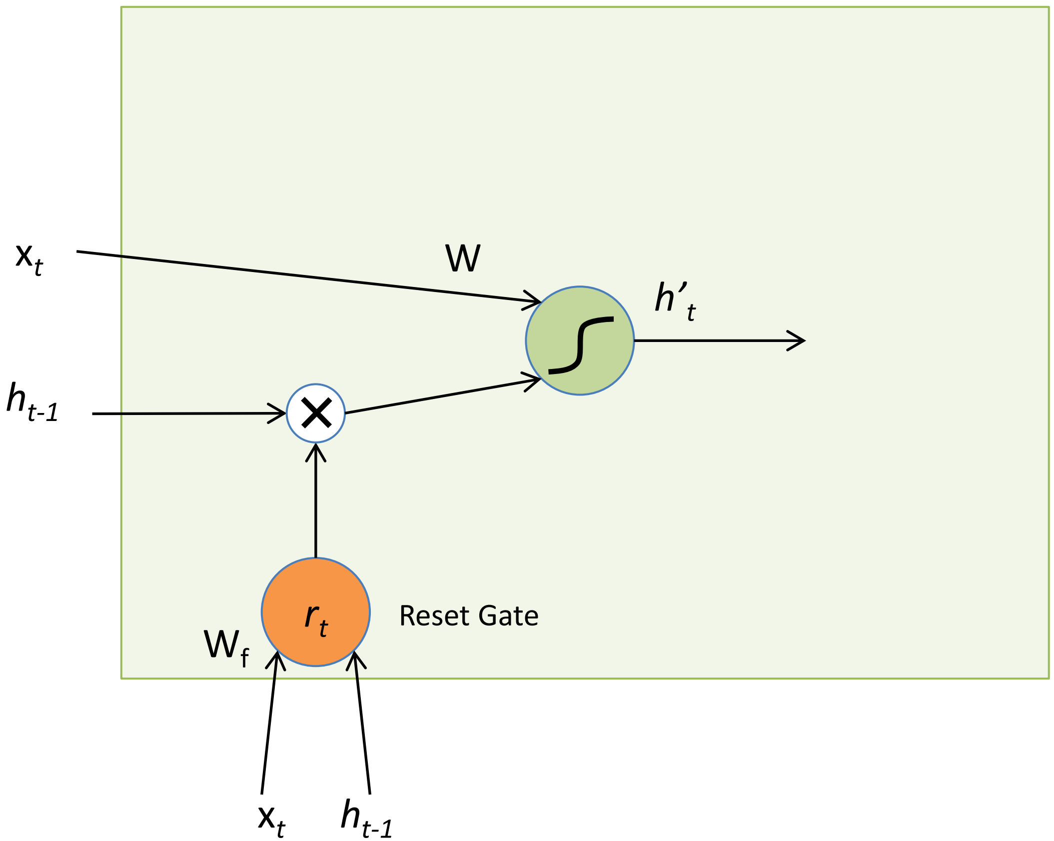
- Has fewer parameters than an LSTM and has been shown to outperform LSTM on some tasks

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, Cho *et al*., 2014

GRU

$$r_t = s\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

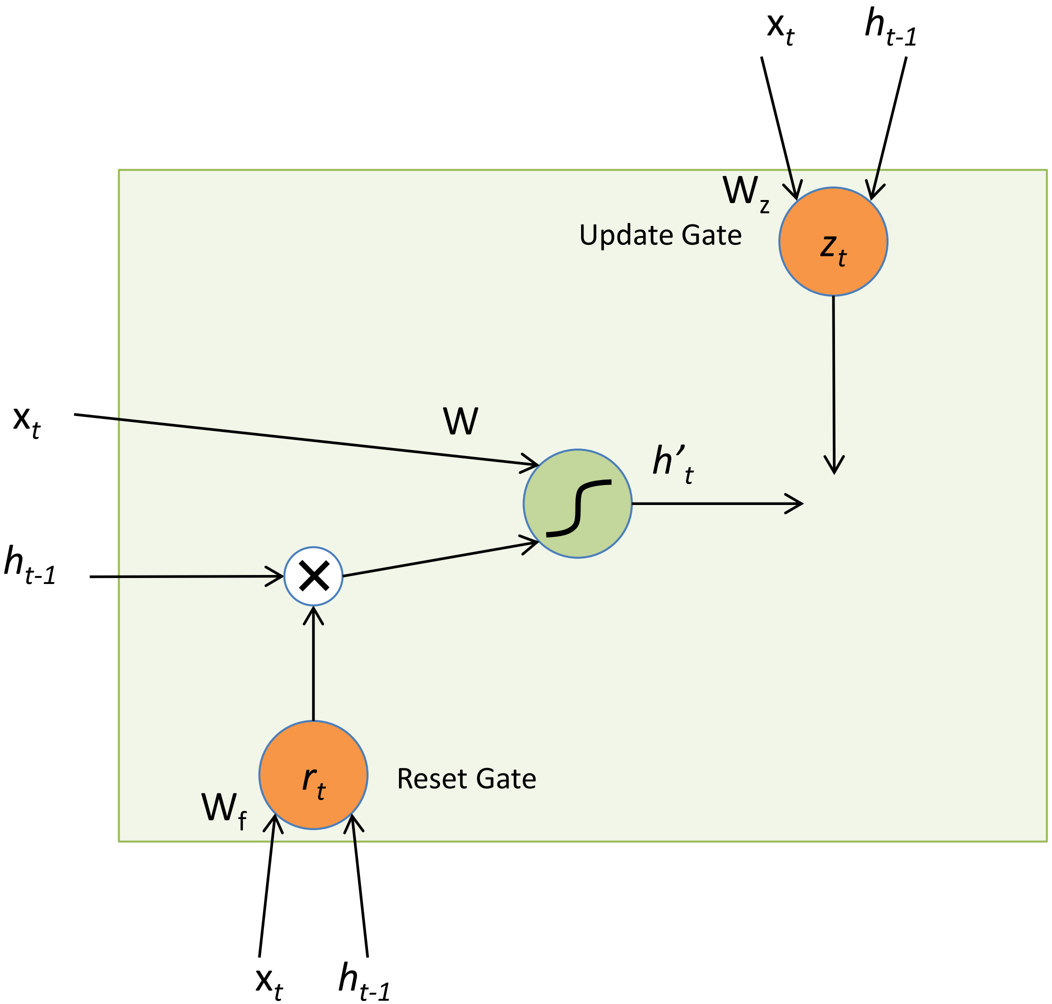
GRU



$$r_t = s\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

GRU

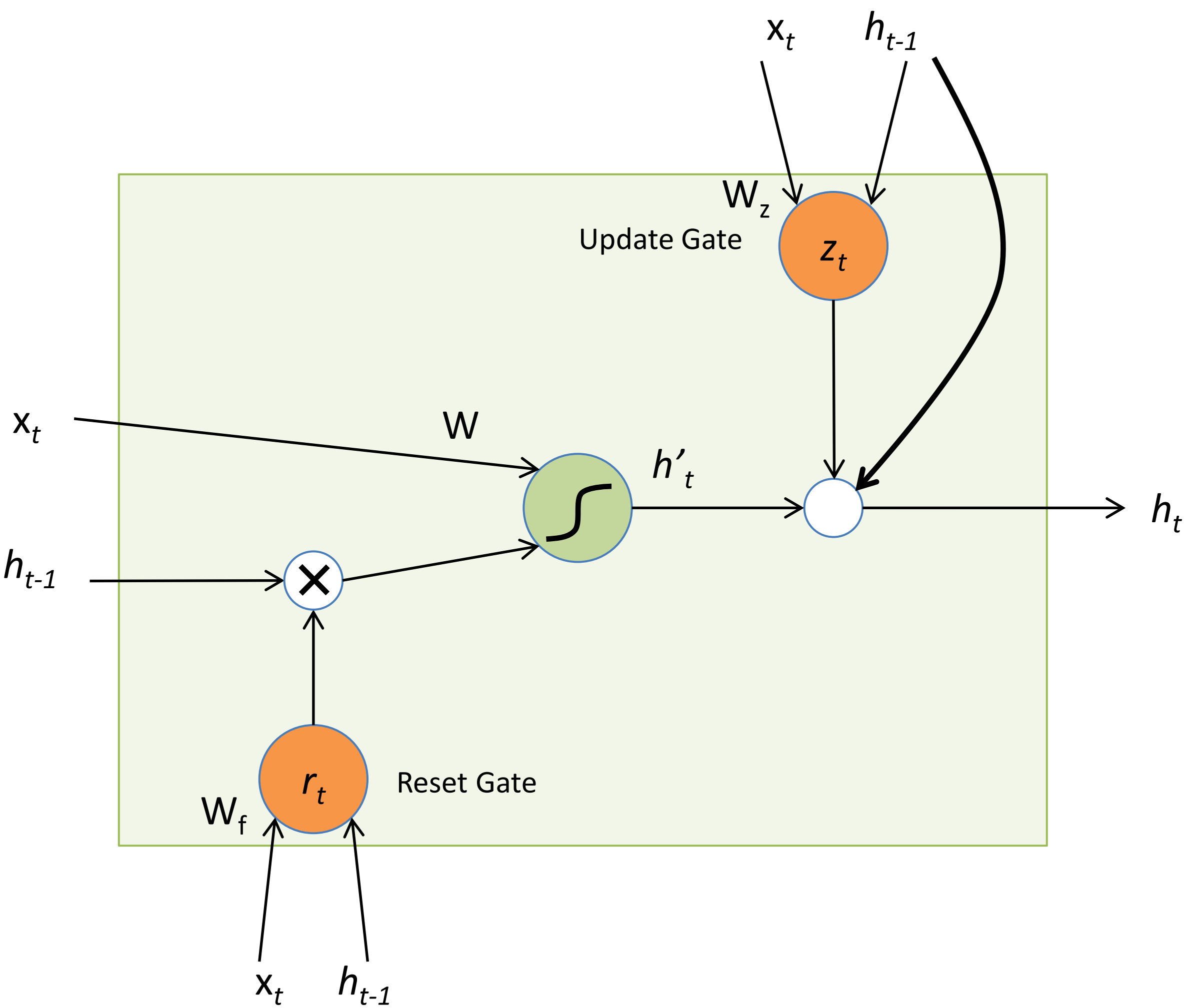


$$r_t = s\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = s\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

GRU



$$r_t = s\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

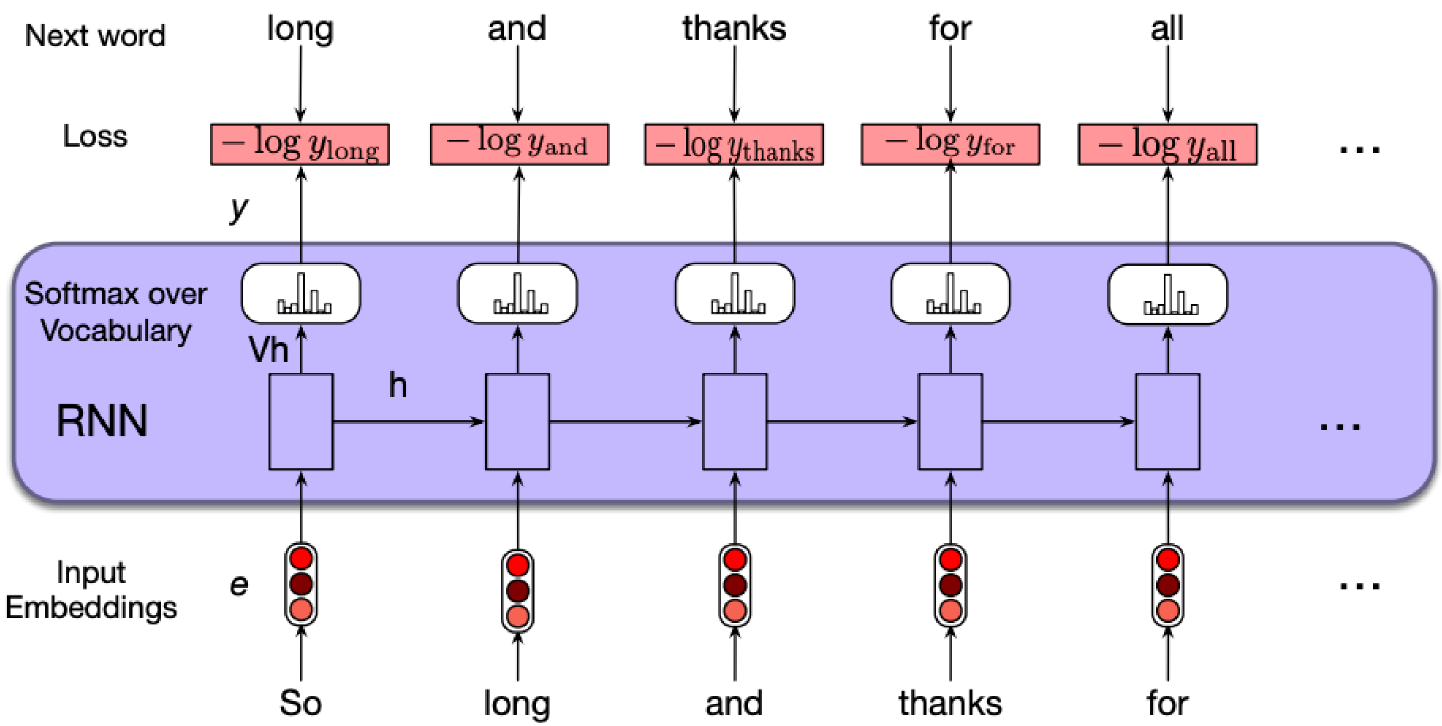
$$z_t = s\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h_t = (1 - z_t) \text{Ä} h_{t-1} + z_t \text{Ä} h'_t$$

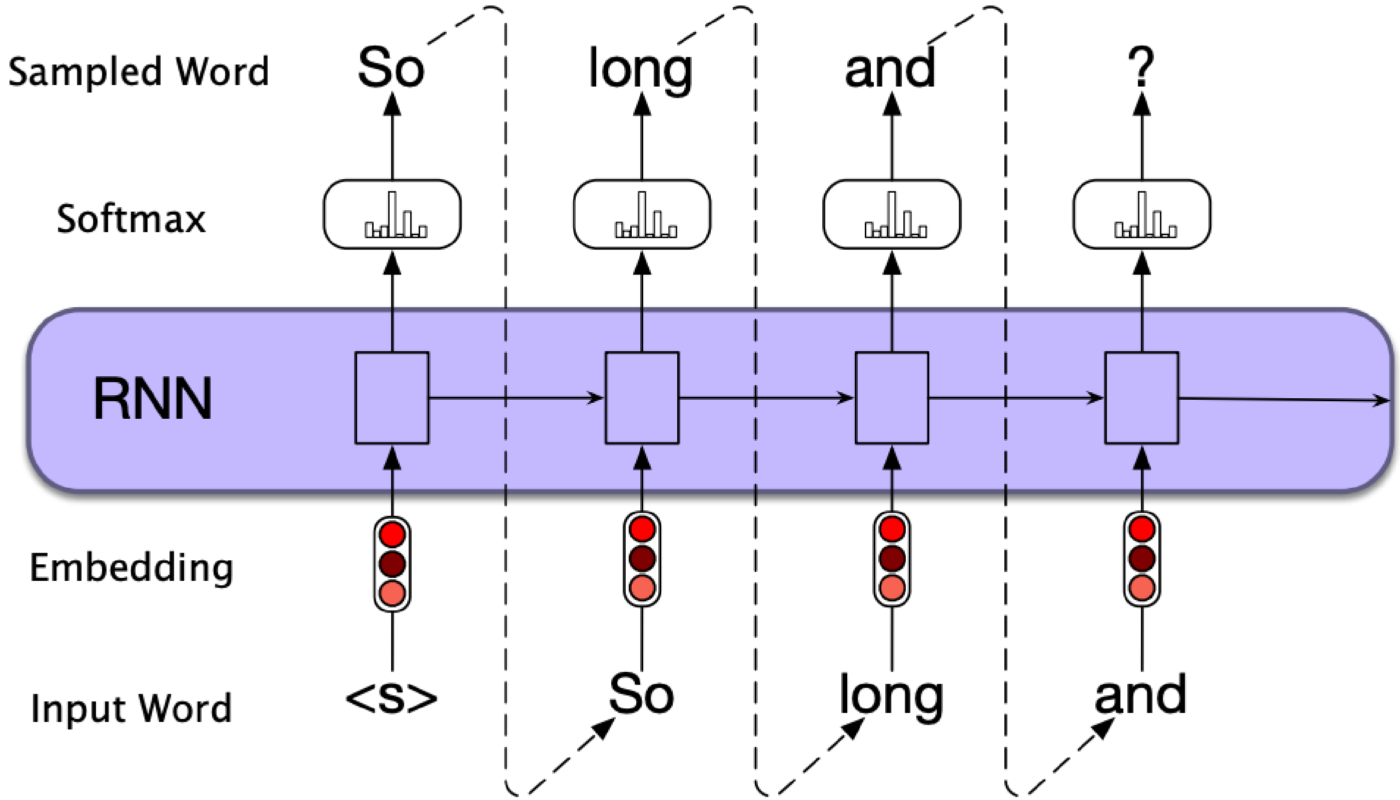
Exercise

- Design a recurrent neural network for language modeling
  - **Input**: ?
  - **Output:** ?

Neural Language Model



Text Generation



What is Encoded in *h*?

- LSTMs learned as language models generate syntactic sentences with long-range number agreement (among other consistencies)
  - How do they achieve that?

What is Encoded in h?

- LSTMs learned as language models generate syntactic sentences with long-range number agreement (among other consistencies)
  - Information in neural units could be stored in local or distributed ways
    - A single unit or multiple units could be responsible for encoding a syntactic or semantic structure

What is Encoded in h?

- **Example**: Number agreement
  - If the network uses local encoding , it is possible to identify the specific units that encode number information
  - How?

What is Encoded in h?

- **Example**: Number agreement
  - If the network uses local encoding , it is possible to identify the specific units that encode number information
  - This can be by ablating cells (setting them to zero) in the network and observing their effects in performance

What is Encoded in h?

- **Example**: Number agreement
  - Data used for probing number agreement:

the **boy** **greets** the guy
the **boy** probably **greets** the guy
the **boy** most probably **greets** the guy
the **boy** openly and deliberately **greets** the guy
the **boy** near Pat **greets** the guy
the **boy** near the car **greets** the guy
the **boy** near the car kindly **greets** the guy

From: The emergence of number and sytax units in LSTM language models, 2019

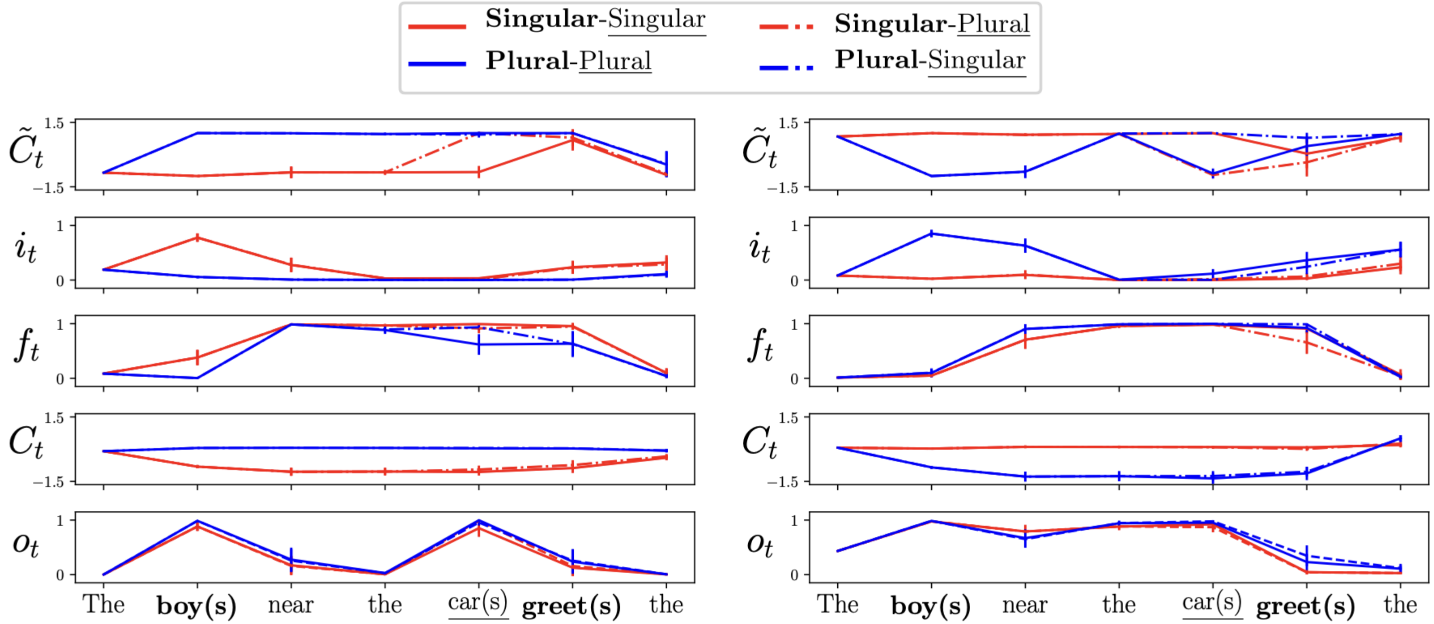
What is Encoded in h?

- **Example**: Number agreement
  - Using a neural language model with 2 hidden LSTM layers of size 650
  - Two cells were identified as responsible for storing number information
    - ablating them resulted in drastic decrease in performance

From: The emergence of number and sytax units in LSTM language models, 2019

What is Encoded in h?

- **Example**: Number agreement
  - Visualizing gate and cell dynamics:



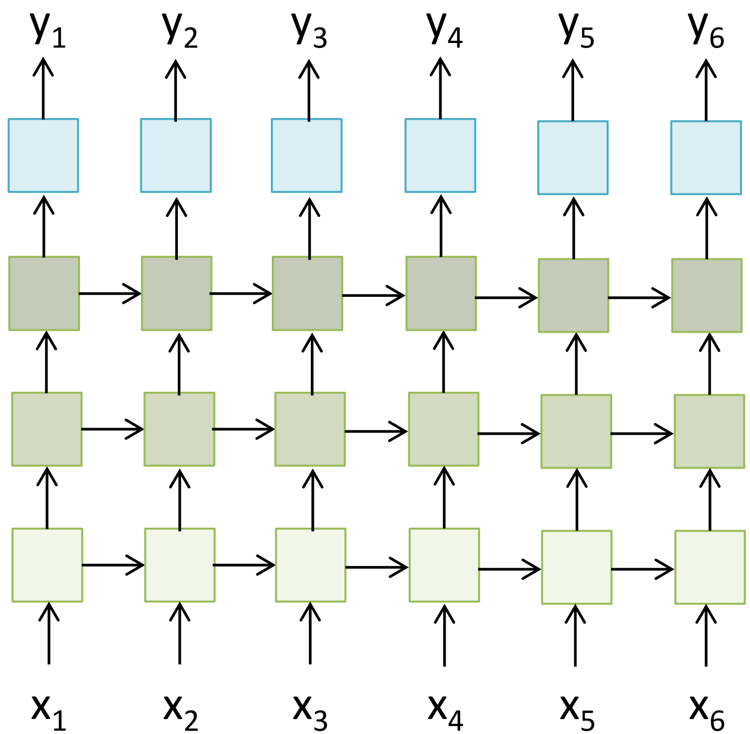
(a) 988 (singular)

(b) 776 (plural)

From: [The emergence of number and sytax units in LSTM language models, 2019]()

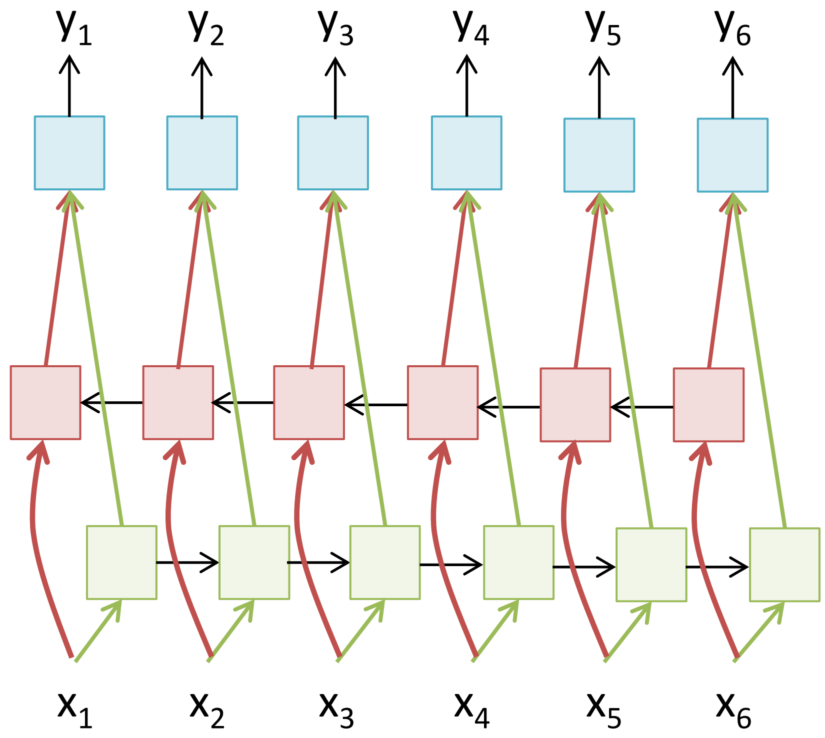
Multi-layer RNNs

- We can of course design RNNs with multiple hidden layers



Bi-directional RNNs

- RNNs can process the input sequence in forward and in the reverse direction



Contextual Embeddings

- Word embeddings obtained using word2vec and similar models are **static**
  - Each word has a fixed representation, regardless of context
  - New words cannot be easily modeled without retraining
  - The embeddings encode some syntactic and semantic features, but they’re implicit and hard to distangle

Contextual Embeddings

- ELMo (**E**mbeddings from **L**anguage **Mo**dels) is a multi-layer bidirectional LSTM trained as a language model
  - The hidden layers of the network corresponding to each input word are used as contextualized embeddings

ELMo Architecture

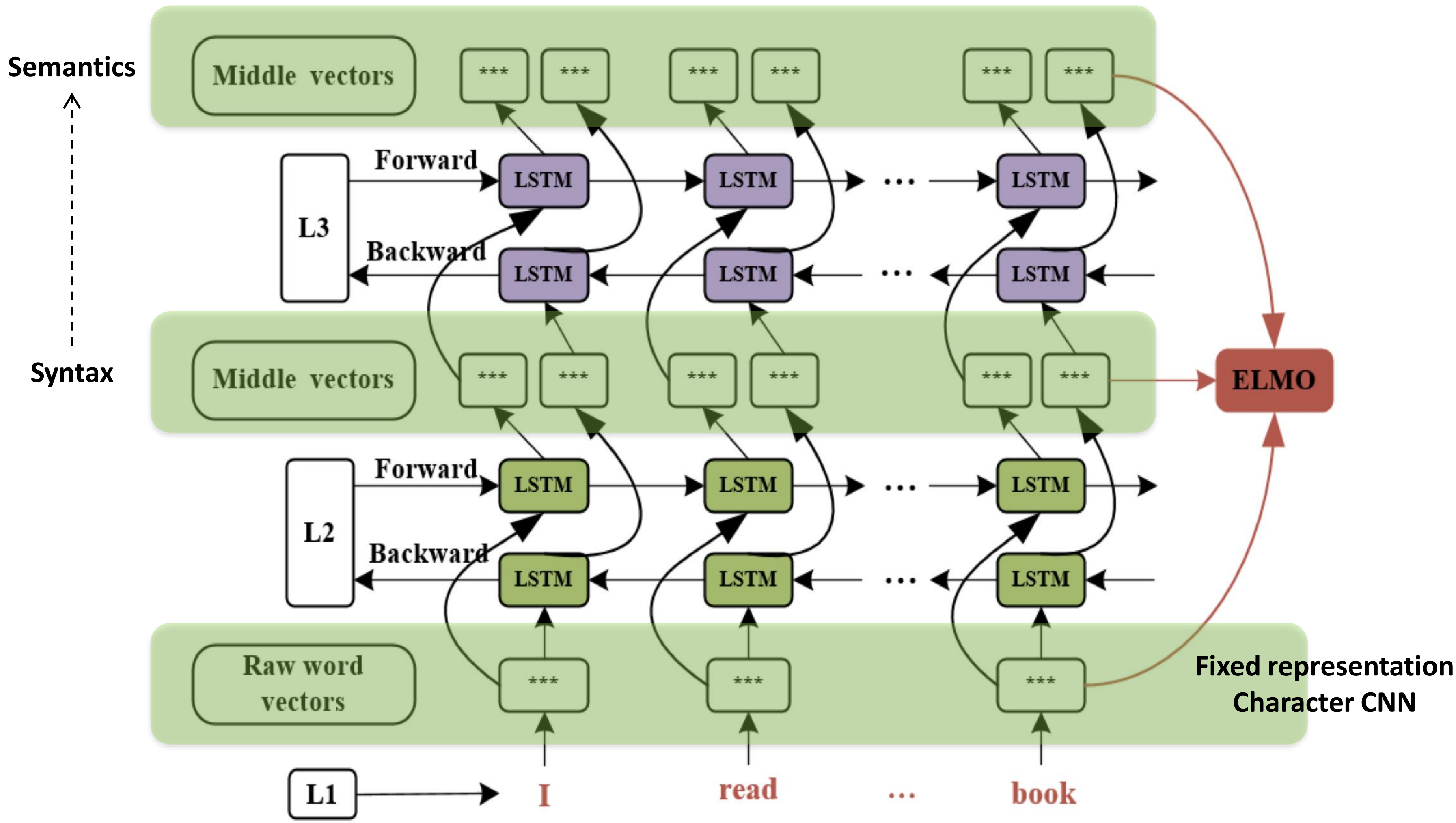


Image from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9141288

Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Backpropagation through many timesteps can be tricky due to exploding and vanishing gradients
  – LSTMs address the vanishing gradient problem by controlling the propagation of gradients through various gates
  – Exploding gradients are handled by gradient clipping
- RNNs can be stacked, or bi-directional
- Architectures like the GRU have fewer parameters than the LSTM and might perform better

Other Useful Resources / References

- http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
- http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

- R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013
- S. Hochreiter, and J. Schmidhuber, Long short-term memory, Neural computation, 1997 9(8), pp.1735-1780
- F.A. Gers, and J. Schmidhuber, Recurrent nets that time and count, IJCNN 2000
- K. Greff , R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems, 2016
- K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014
- R. Jozefowicz, W. Zaremba, and I. Sutskever, An empirical exploration of recurrent network architectures, JMLR 2015

Slide References

- The slides in this lecture are adapted from multiple other lectures from:
  - Shangsong Liang, MBZUAI
  - Hanan Aldarmaki, MBZUAI
  - “Speech & Language Processing” 3rd edition, draft:
    - https://web.stanford.edu/~jurafsky/slp3/