Recurrent Neural Networks

Recap

Natural language data is inherently **sequential**

- Sequences of characters, words, sentences, paragraphs, chapters; audio signals; etc.

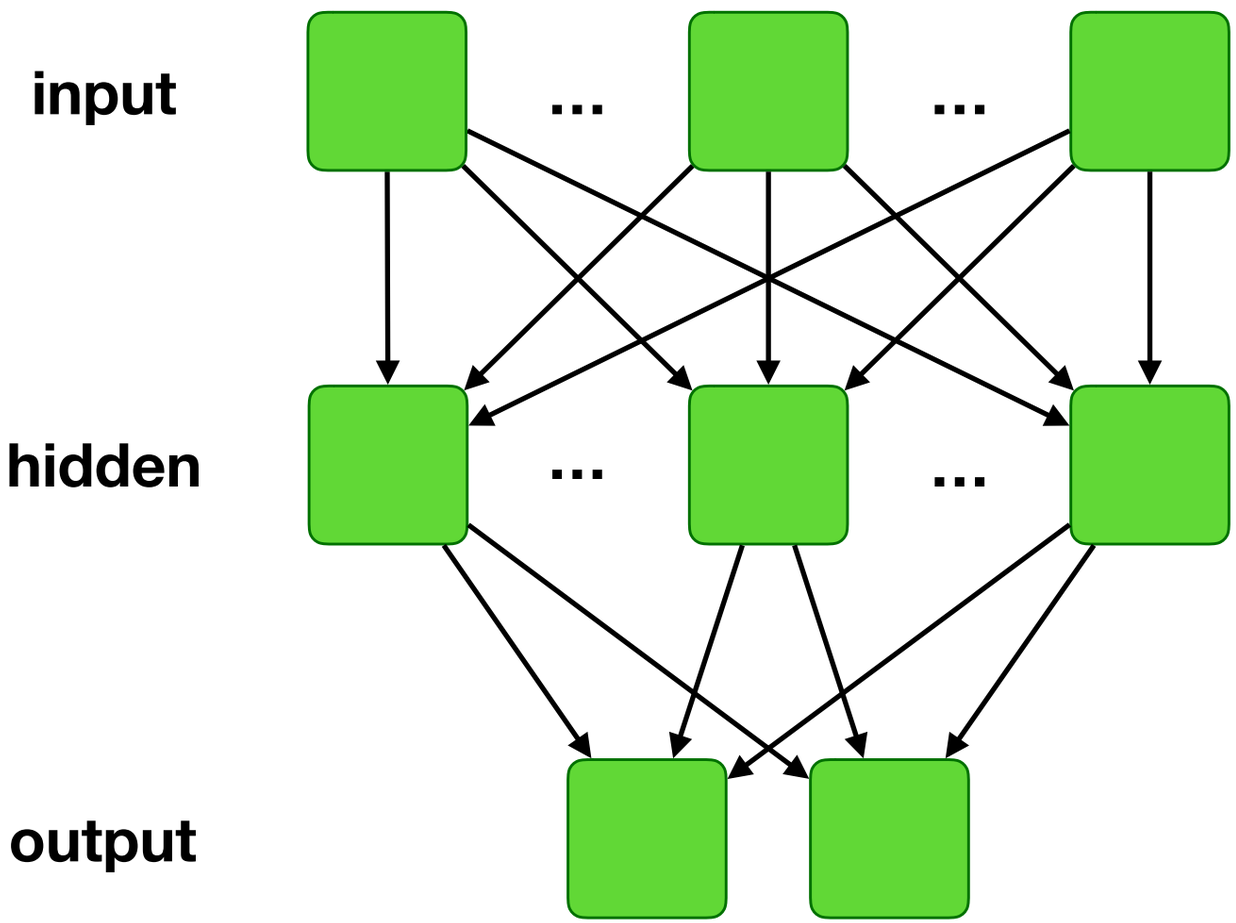
Information on the sequence is what allows us to differentiate e.g. “**dog bites man**” (boring) from “**man bites dog**” (news!)

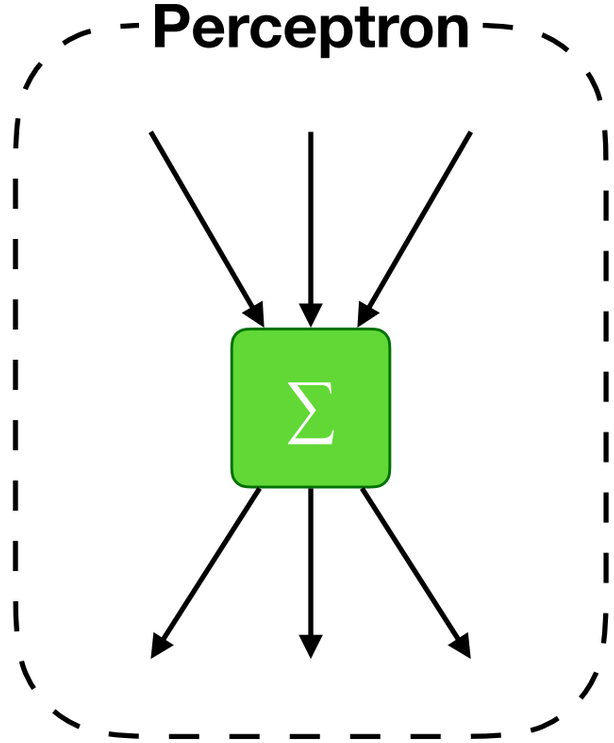
Models discussed so far

- Bag-of-words MLP: discard word order, hope for the best
- Convolutional NN: capture local word order through filters

Recap

Multi-layer perceptron

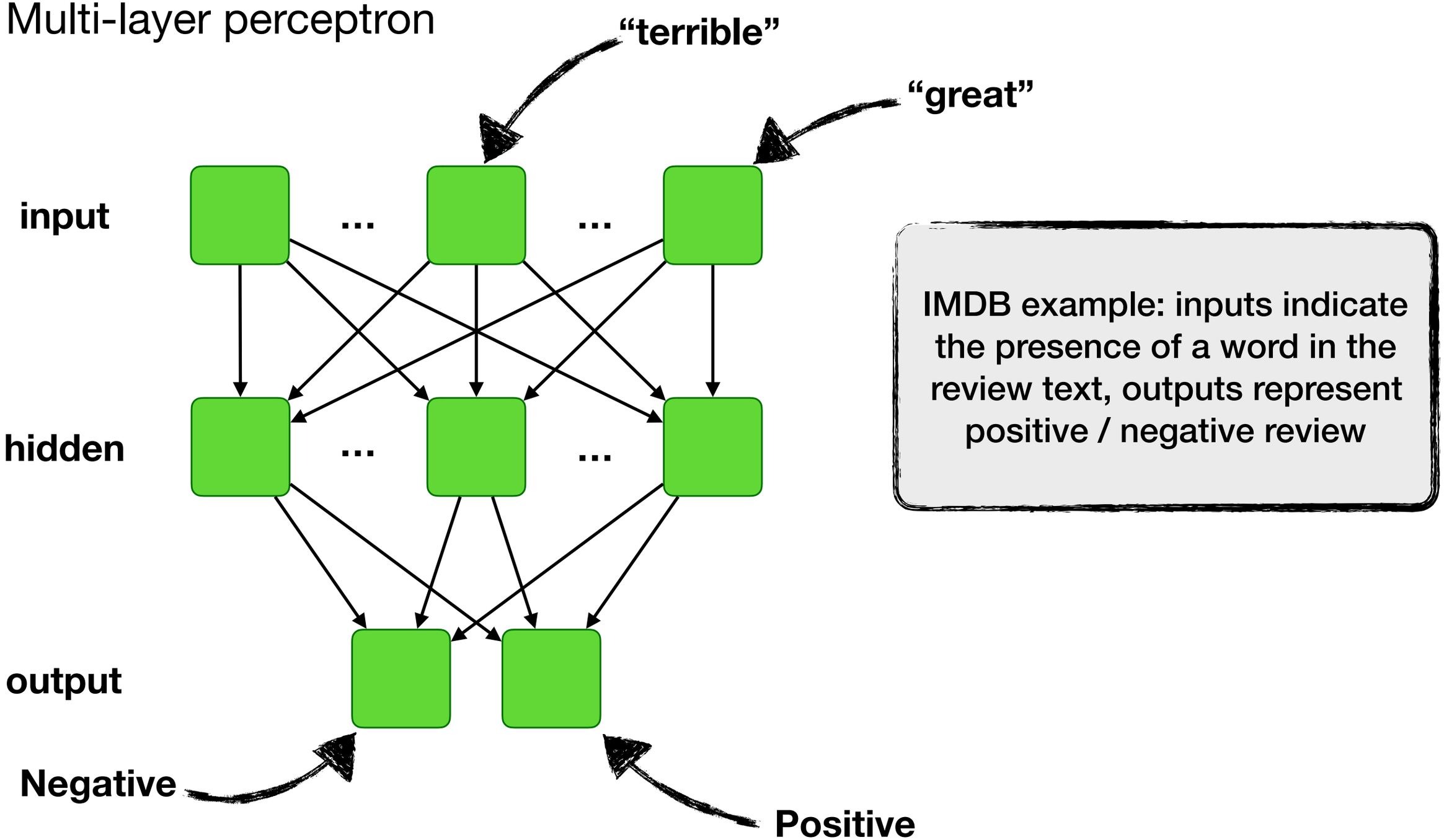




Rosenblatt (1957) *The Perceptron—a perceiving and recognizing automaton*

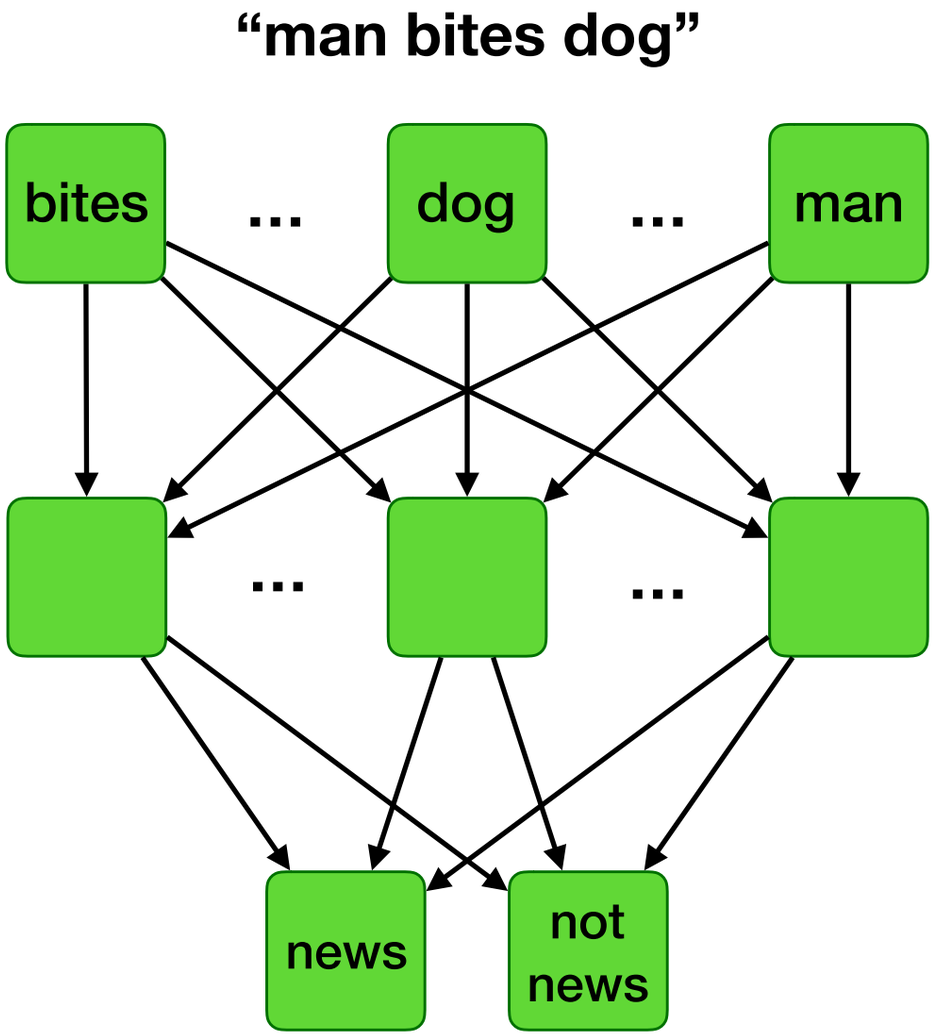
Recap

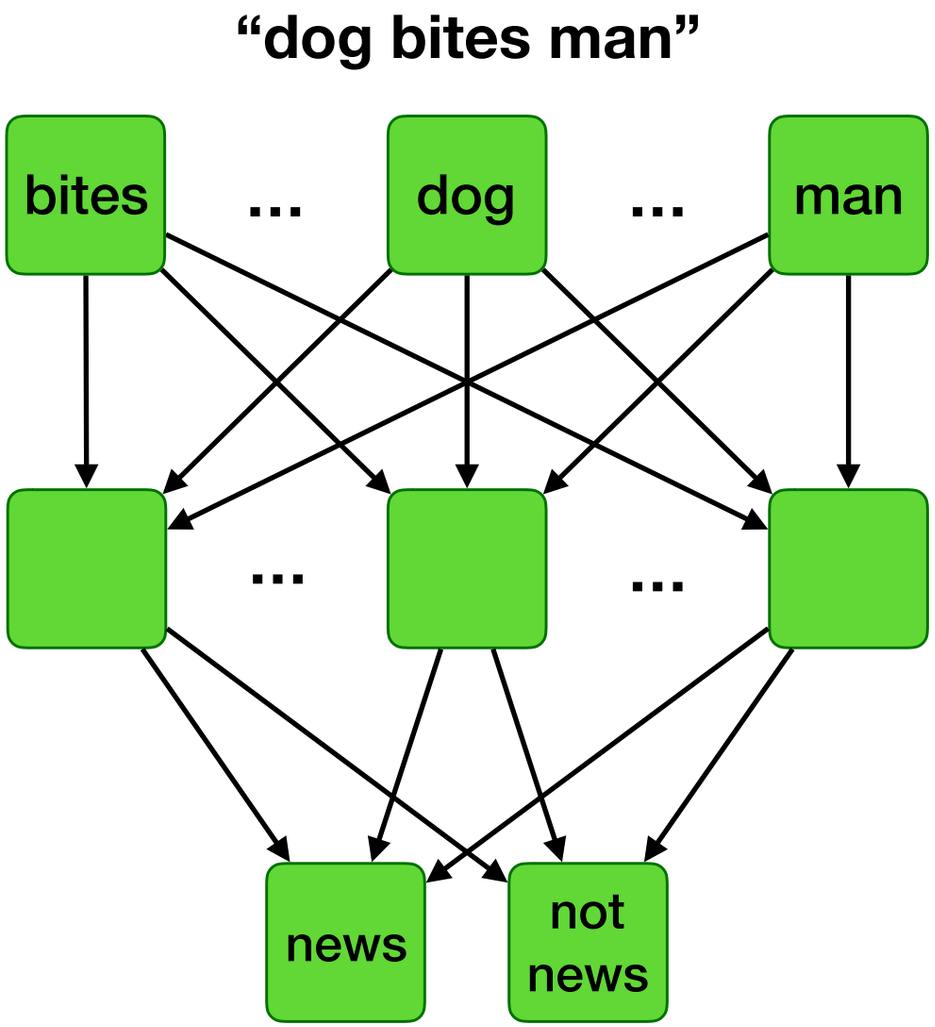
Multi-layer perceptron



Recap

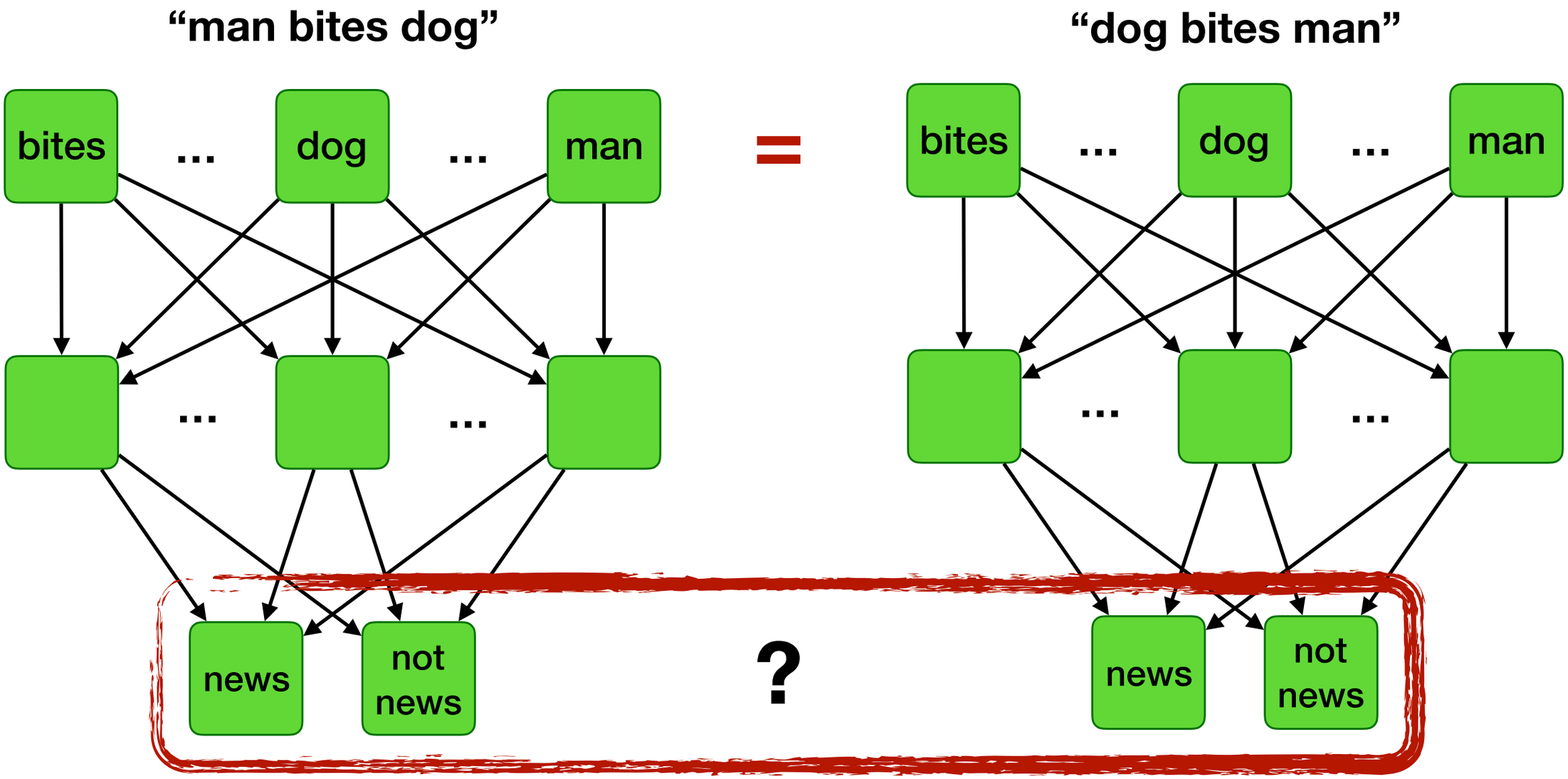
Bag-of-words multi-layer perceptron





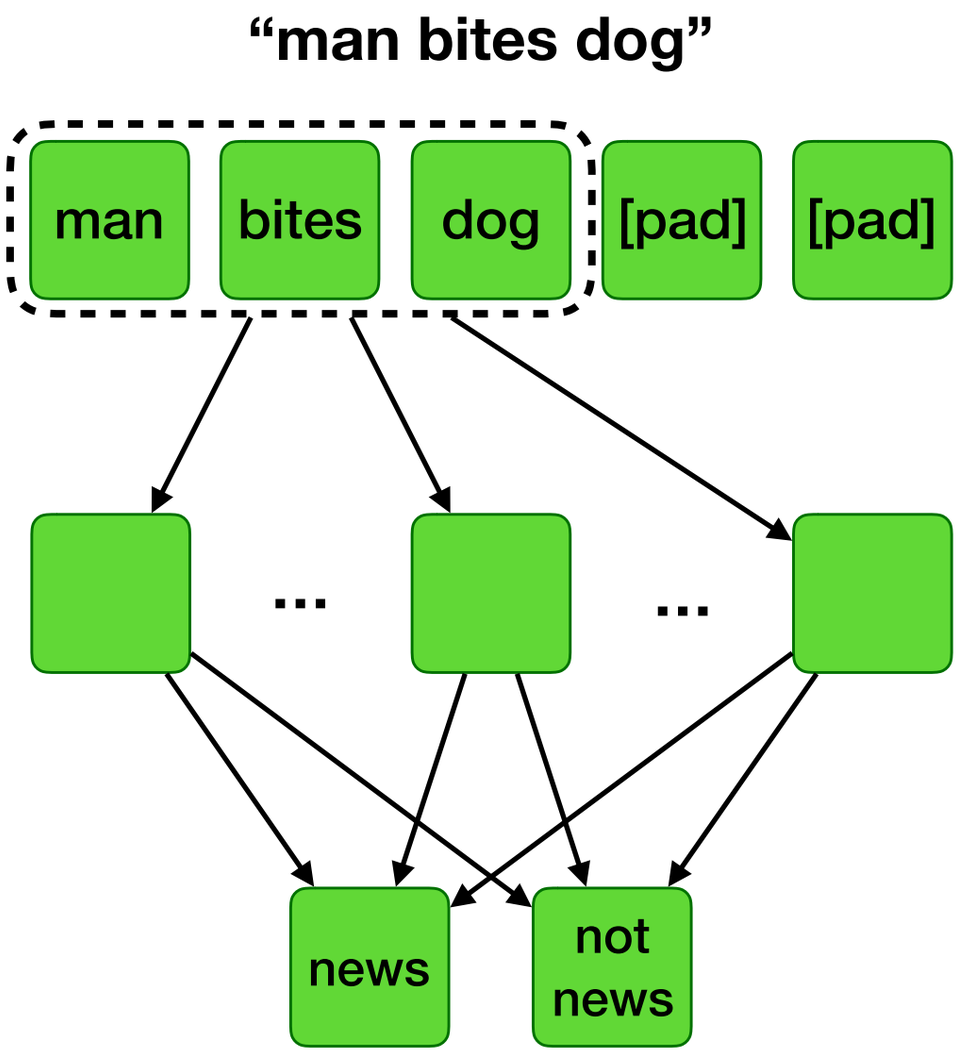
Recap

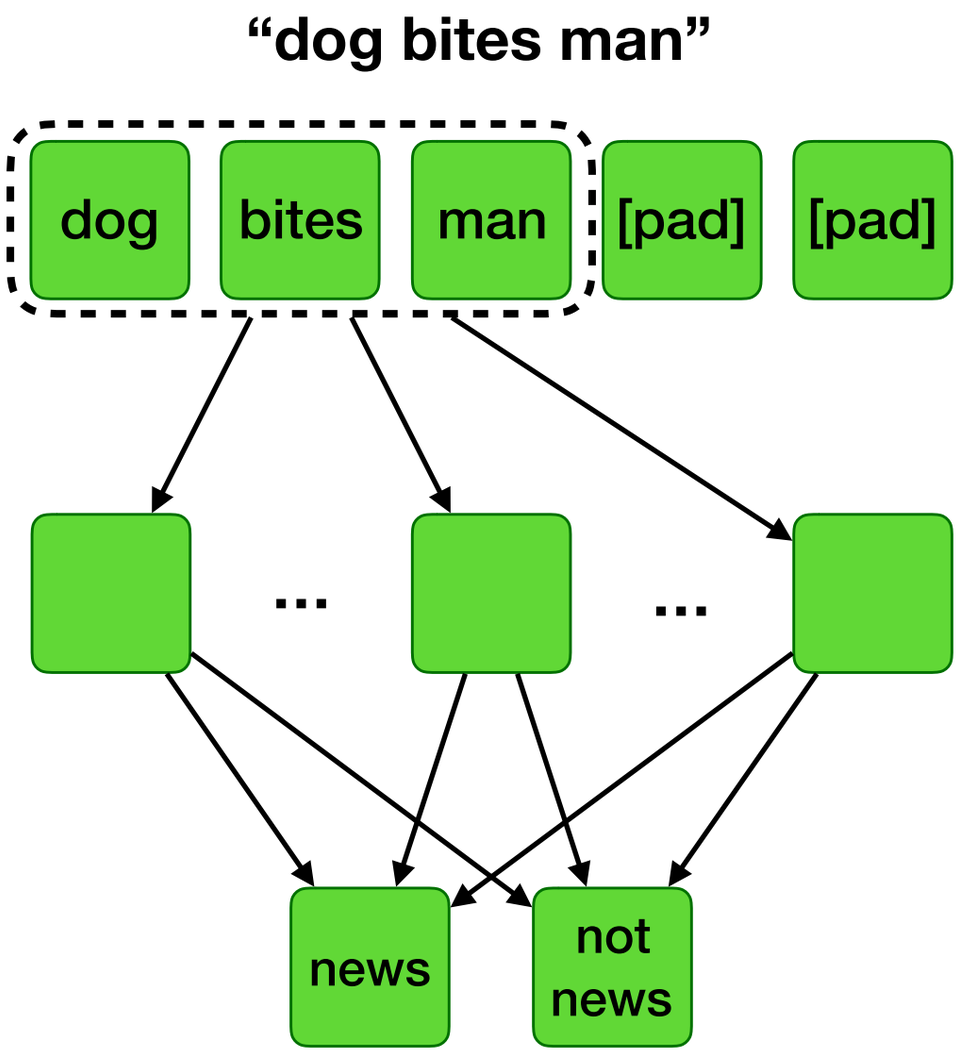
Bag-of-words multi-layer perceptron



Recap

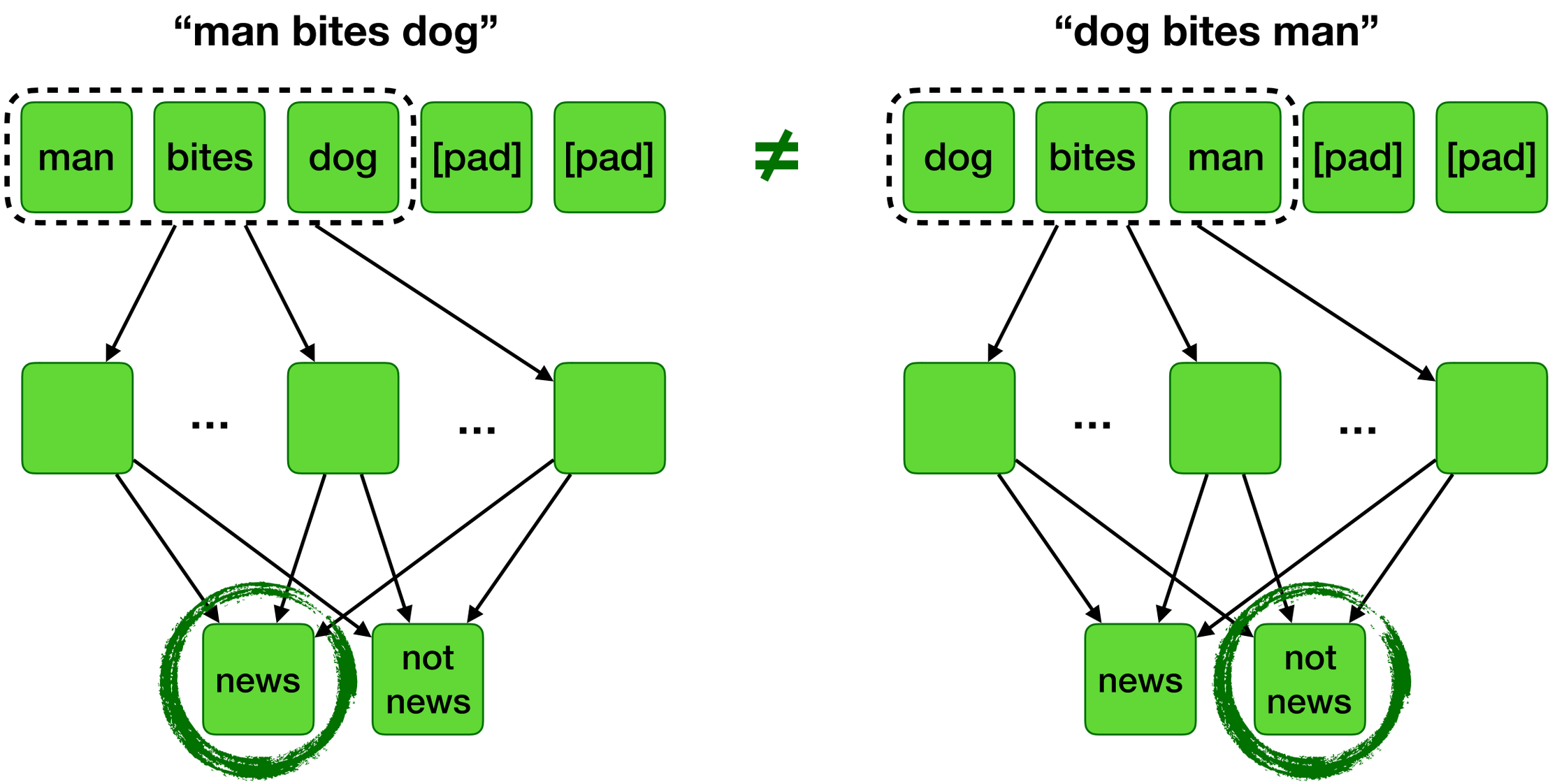
Convolutional neural network





Recap

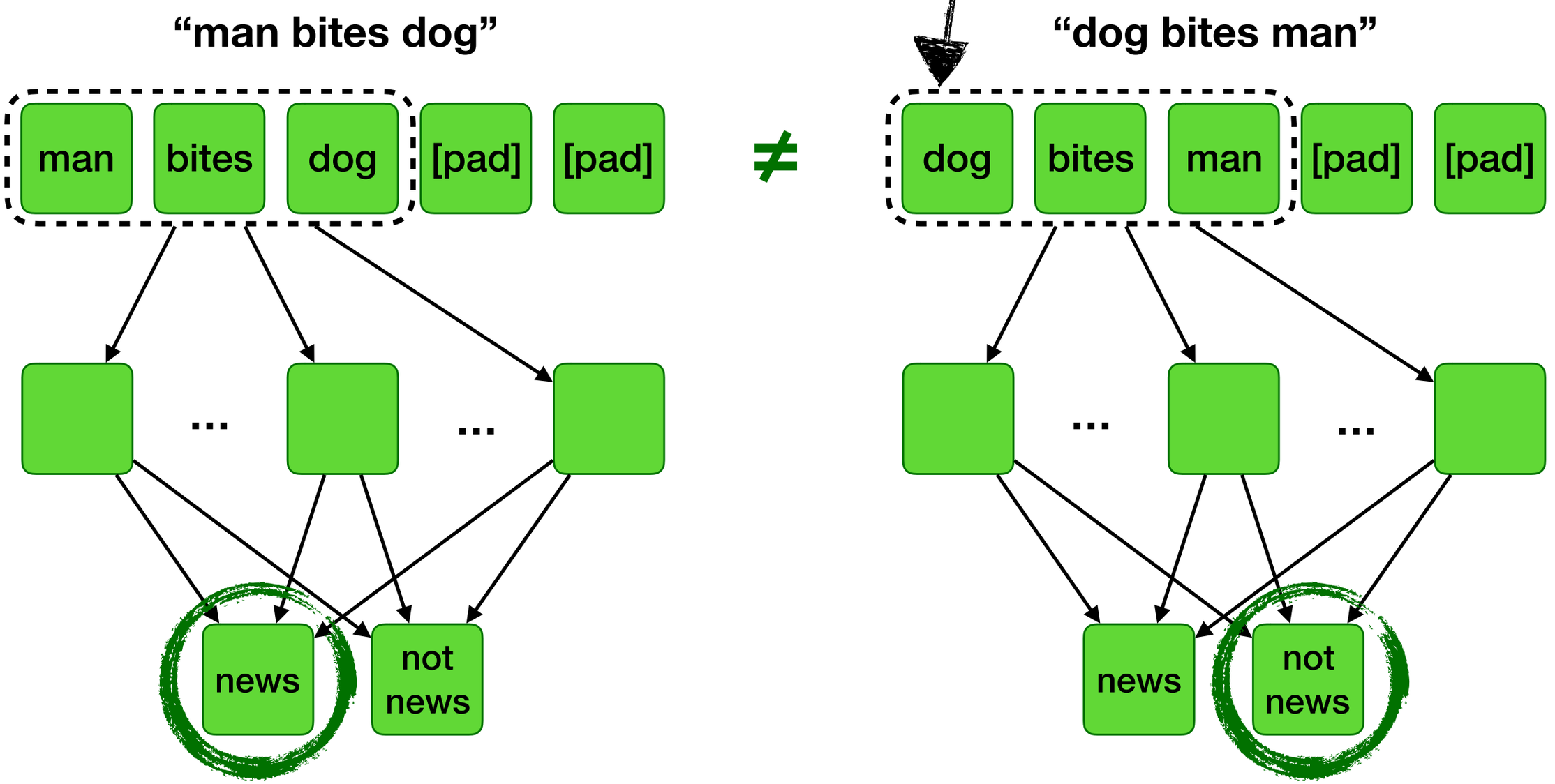
Convolutional neural network



Recap

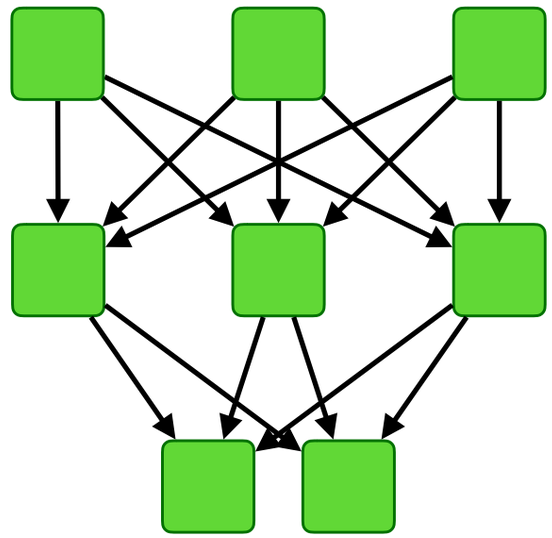
Convolutional neural network

NB: sequence information only captured **locally** up to filter width (frequently < 10)



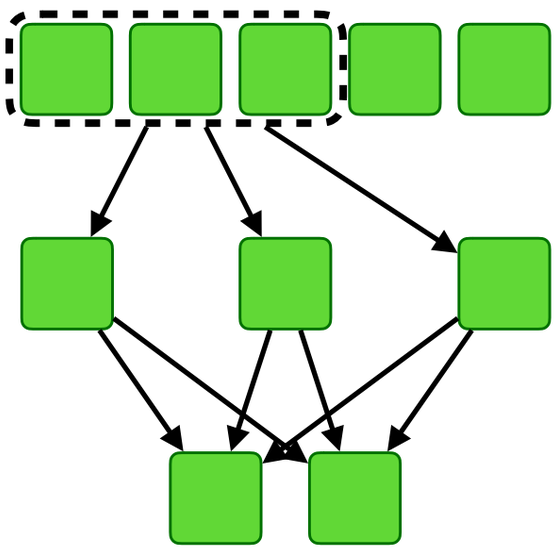
Architectures

The MLP and CNN models share the same high-level architecture



One-to-one mapping: fixed-length input to fixed-length output

One-directional information flow: input → hidden → output



Fixed number of computational steps

These are instances of **feed-forward** neural networks

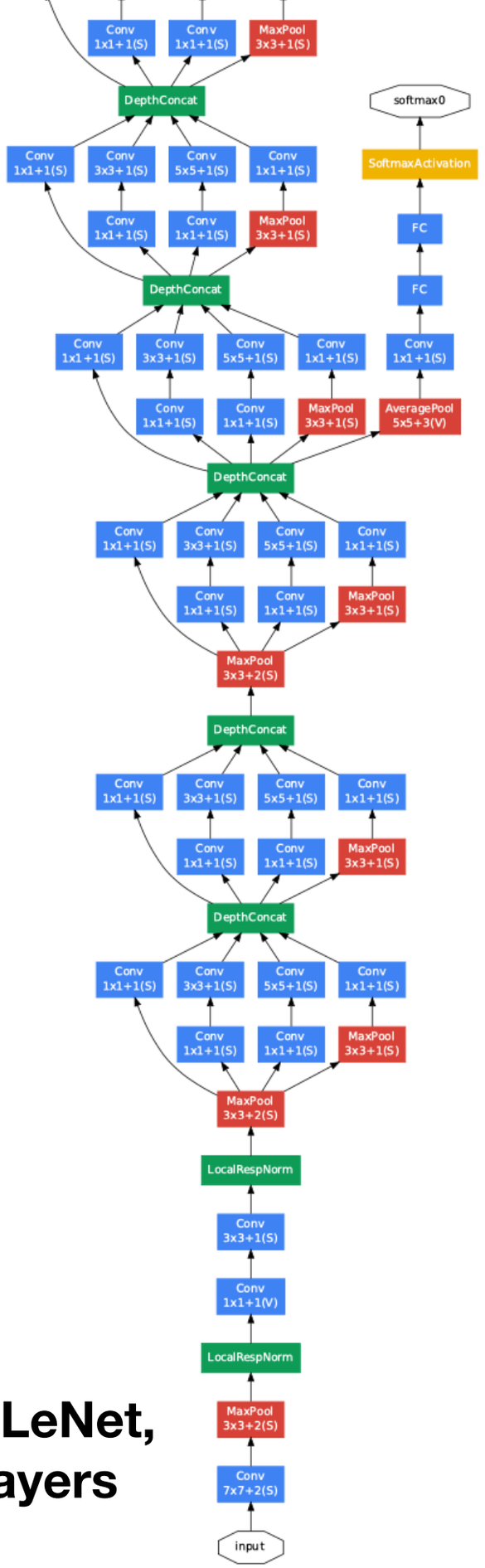
Architectures

Generally, any neural net **without cycles** is a feed-forward neural network

(Not about network depth or “simplicity”)

One-directional information flow, one-to-one mapping, fixed computational steps

By contrast, networks with cycles are called **recurrent**



GoogLeNet, 22 layers

Szegedy et al. (2015) *Going deeper with convolutions*

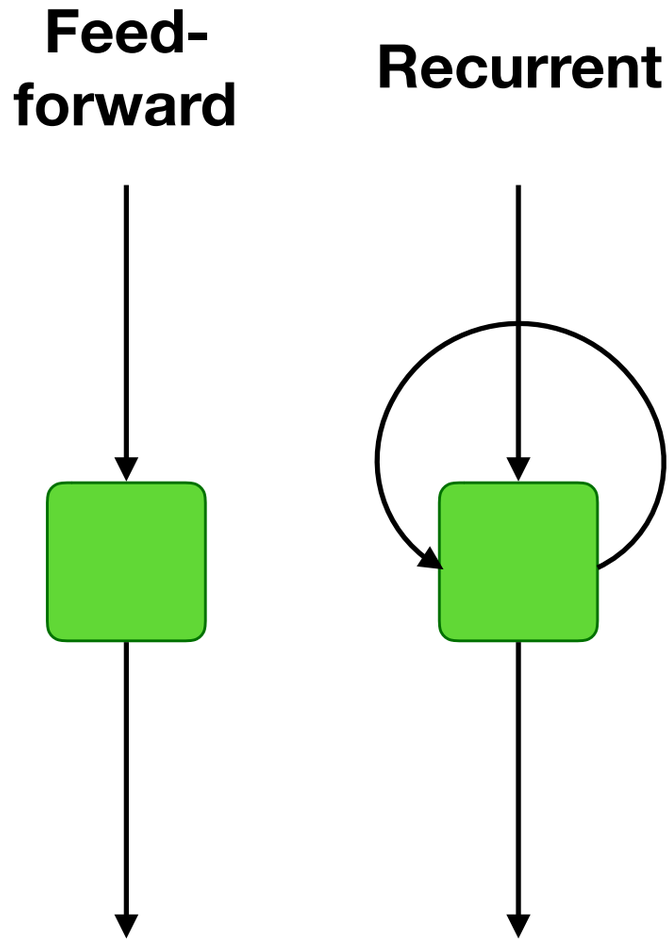
Recurrent neural networks

Neural networks with cycles

Information flowing back provides a form of memory of previous inputs → state

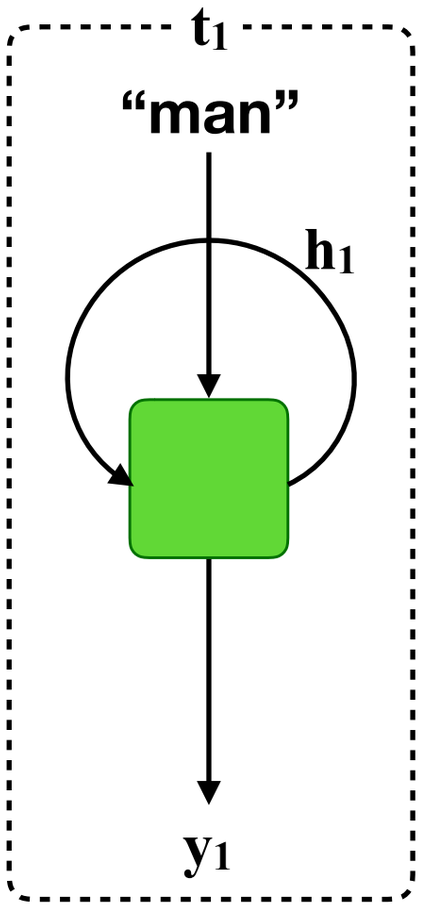
Recursive processing allows for variable-length inputs and outputs → one-to-many, many-to-many mappings etc.

Can be viewed (and implemented!) as applying in a sequence of time steps



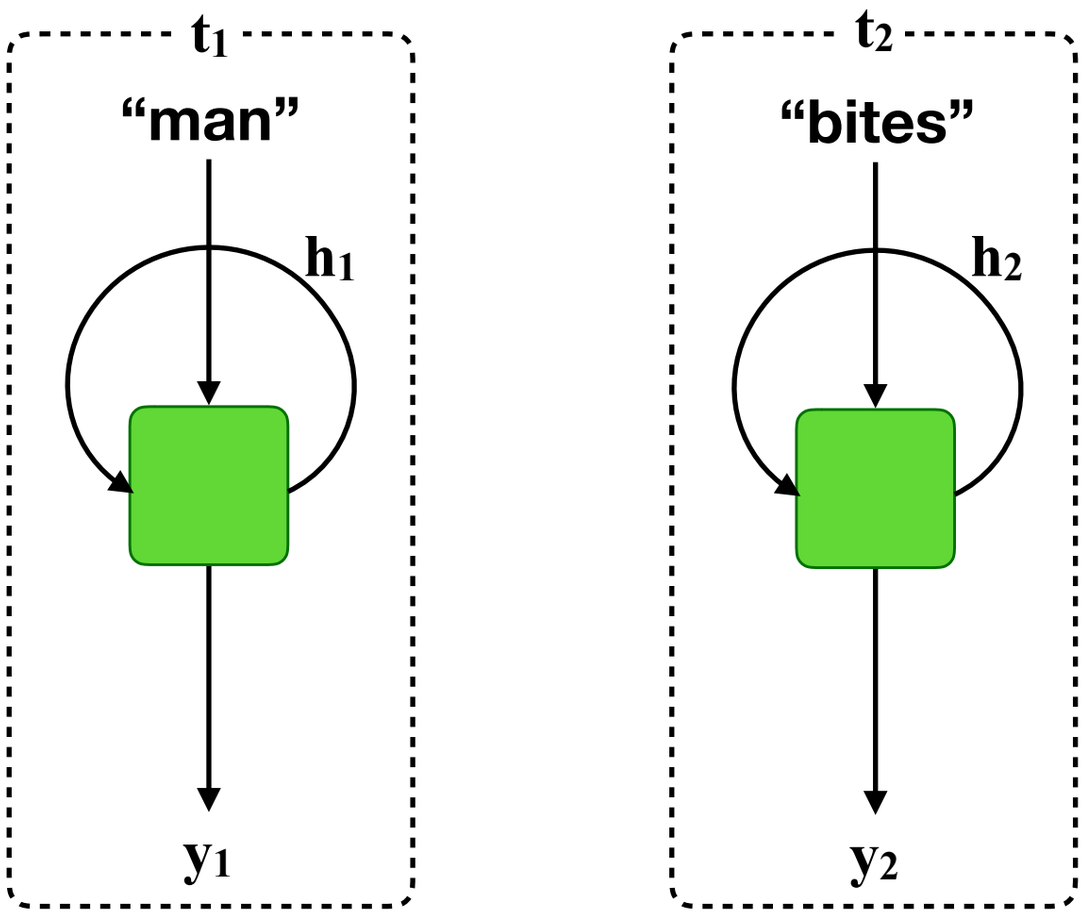
Recurrent neural networks

Time step 1: input “man”, output y_1, hidden state h_1



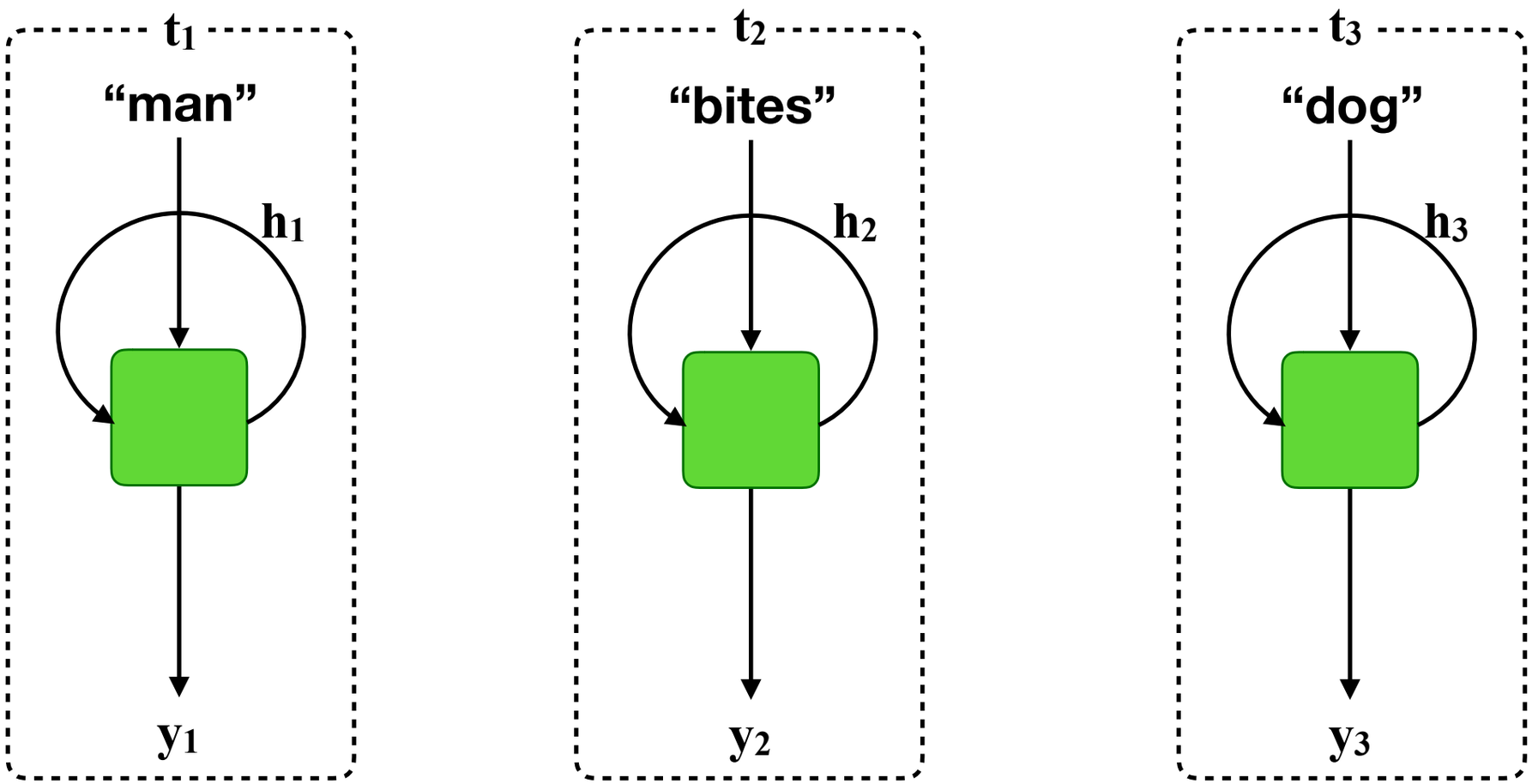
Recurrent neural networks

Time step 2: initial hidden state h_1, input “bites”, output y_2, hidden state h_2



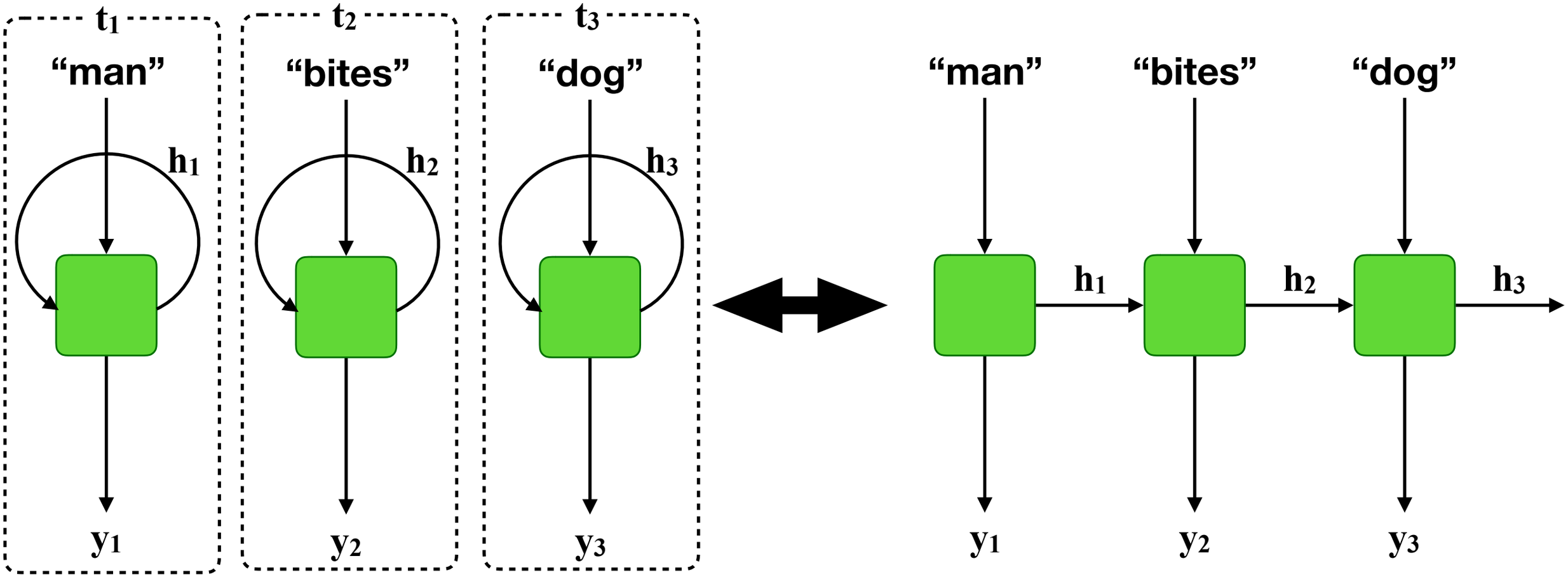
Recurrent neural networks

Time step 3: initial hidden state h_2, input “dog”, output y_3, hidden state h_3



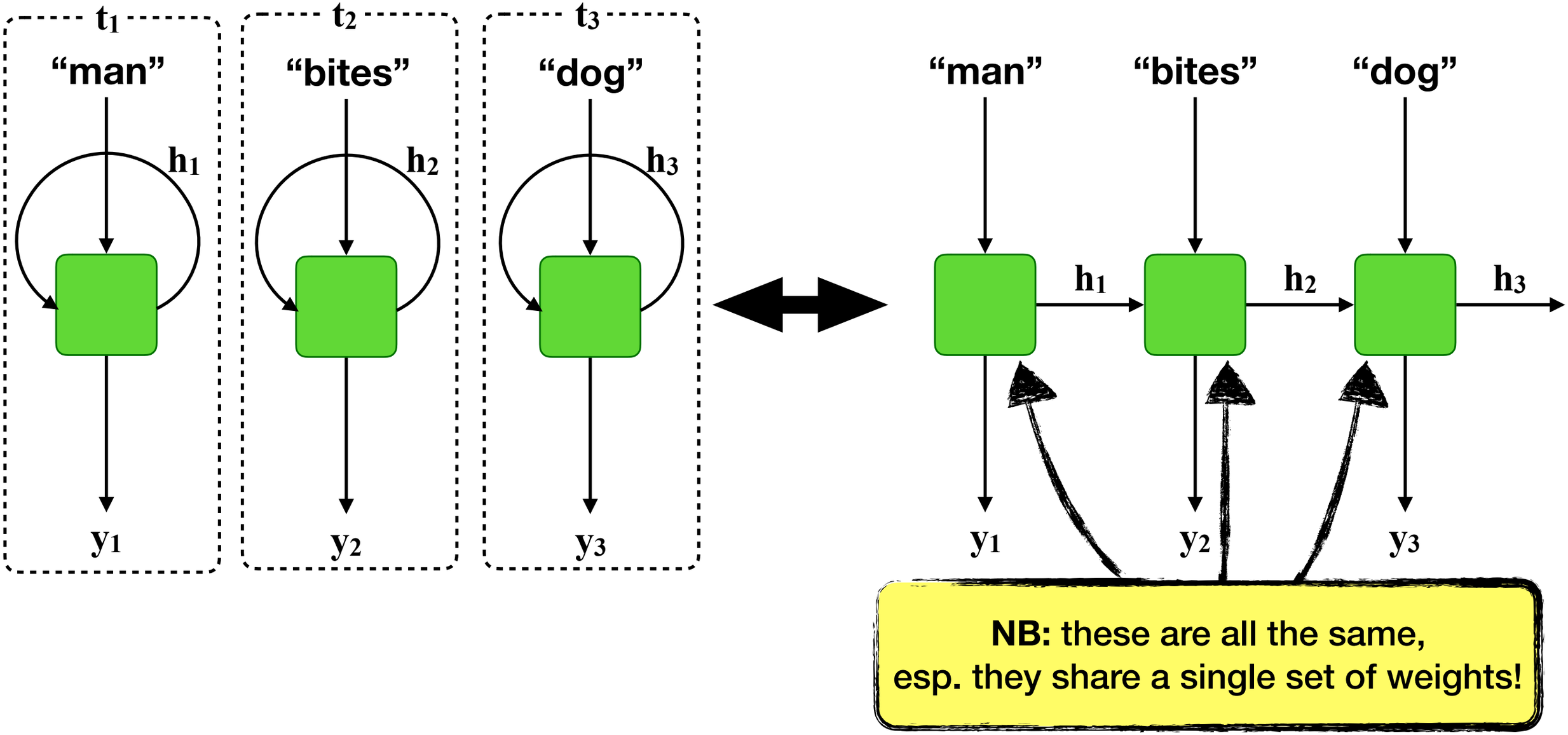
Recurrent neural networks

Step sequence (left) is equivalent to unfolded / unrolled network (right)



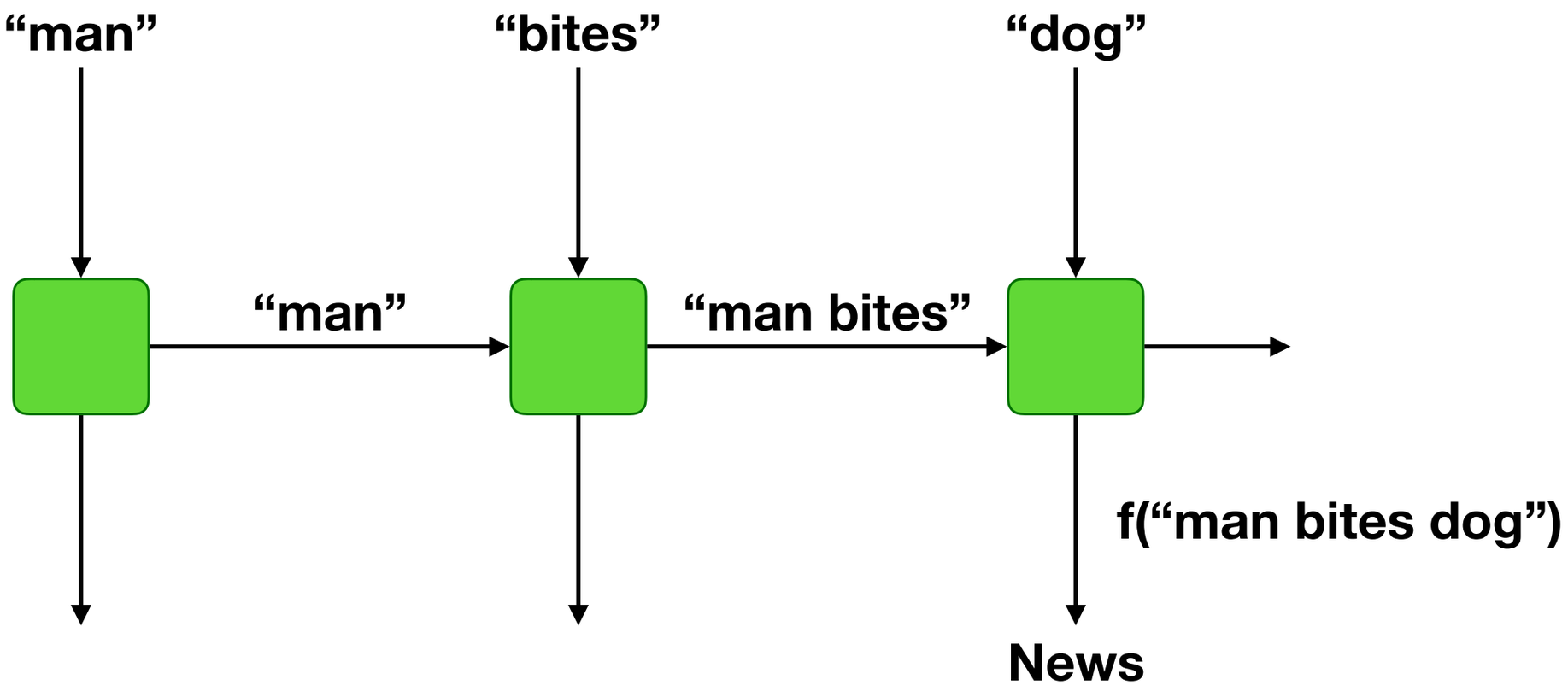
Recurrent neural networks

Step sequence (left) is equivalent to unfolded / unrolled network (right)



Recurrent neural networks

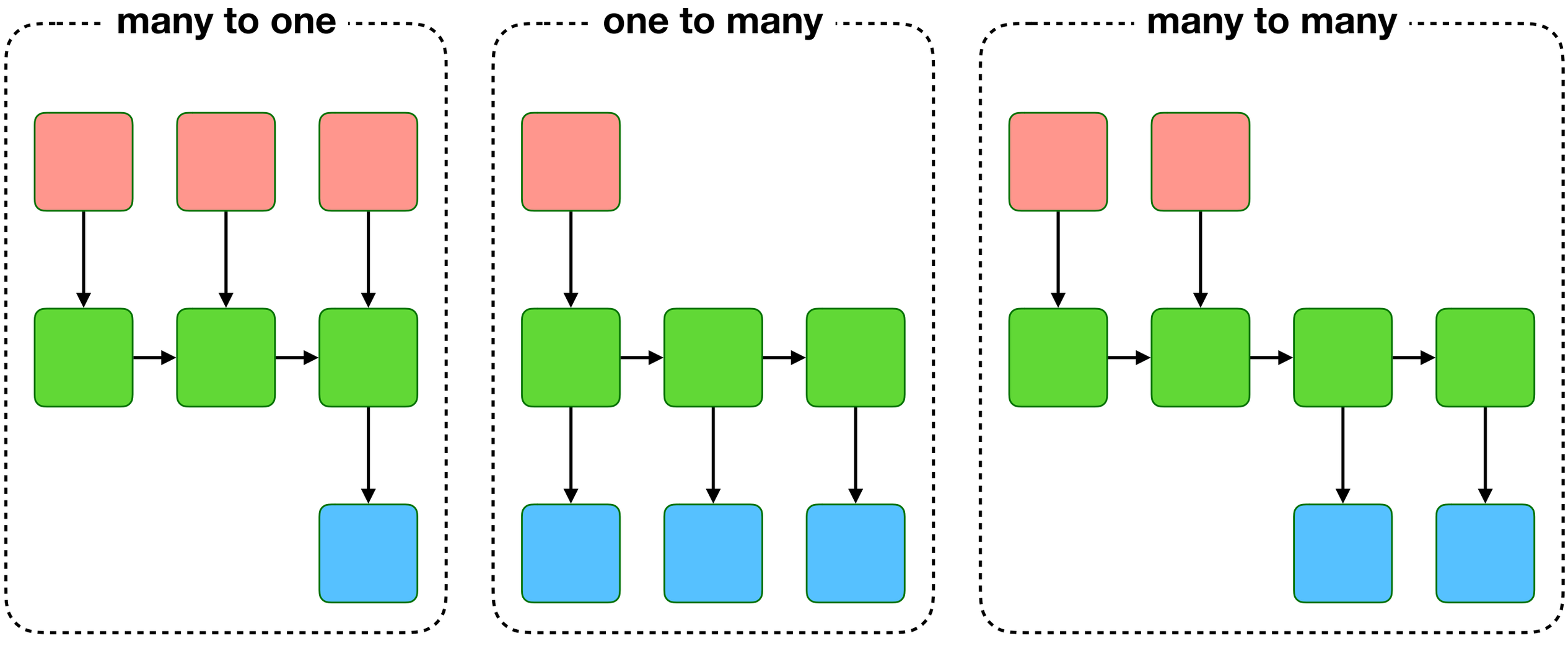
Hidden state allows the network to remember sequence of inputs



→ can theoretically capture arbitrarily long input sequences

Recurrent neural networks

Mappings between variable-length inputs and outputs



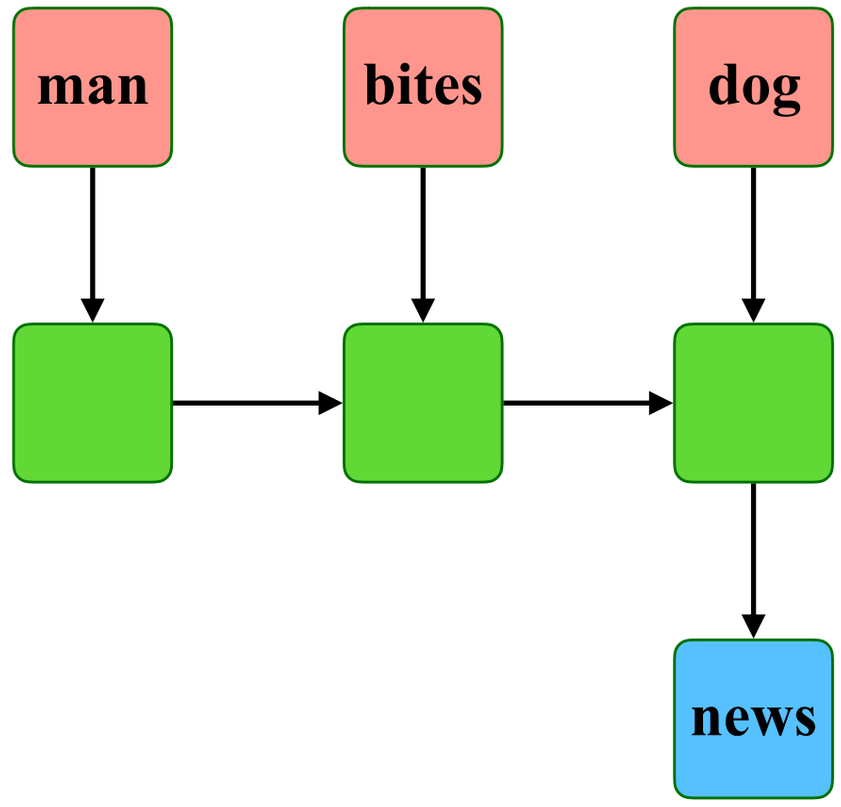
Applications

Many to one: text classification

Input: sequence of words

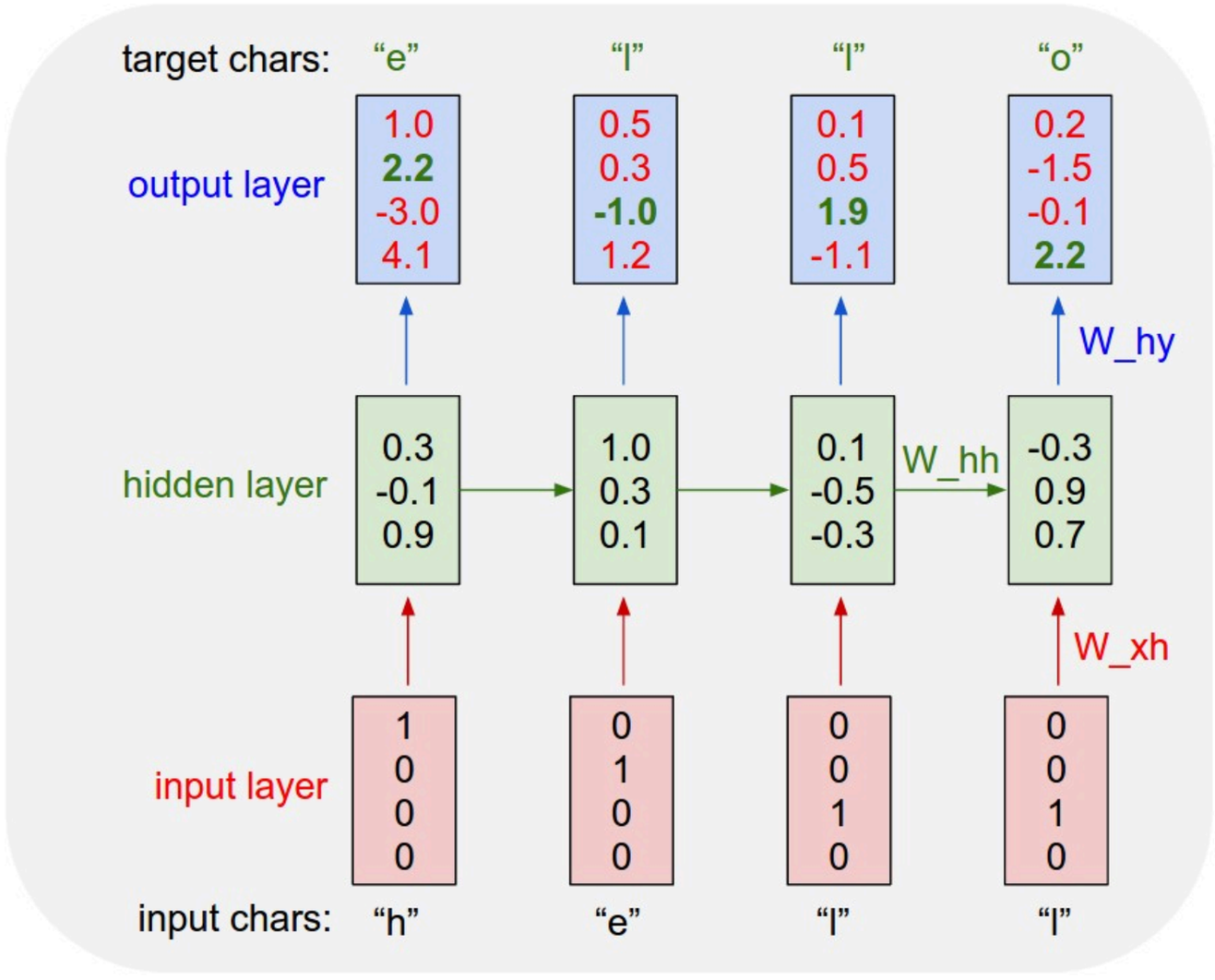
Output: target label

Similarly: language modeling
(labels = words)



Applications

Text generation: input preceding sequence, output continuation



VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are han
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds
So drop upon your lordship's head, and your opinion
Shall be against your honour.

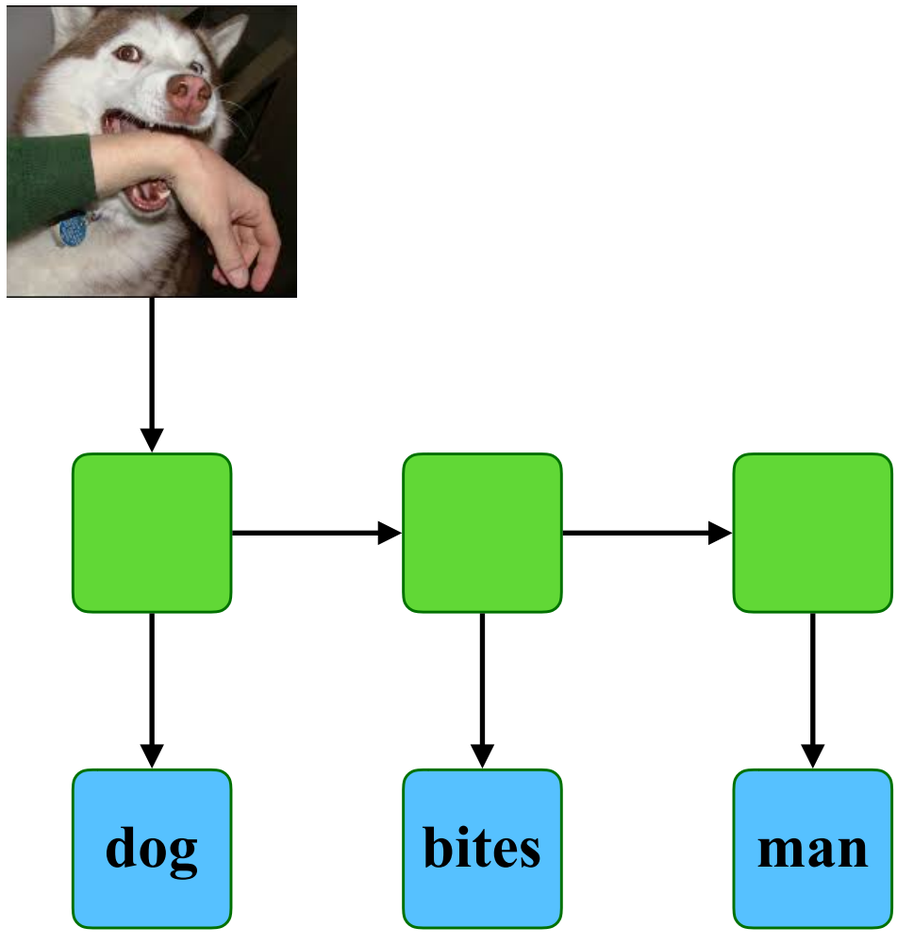
Karpathy (2015) *The Unreasonable Effectiveness of Recurrent Neural Networks*

Applications

One to many: image captioning

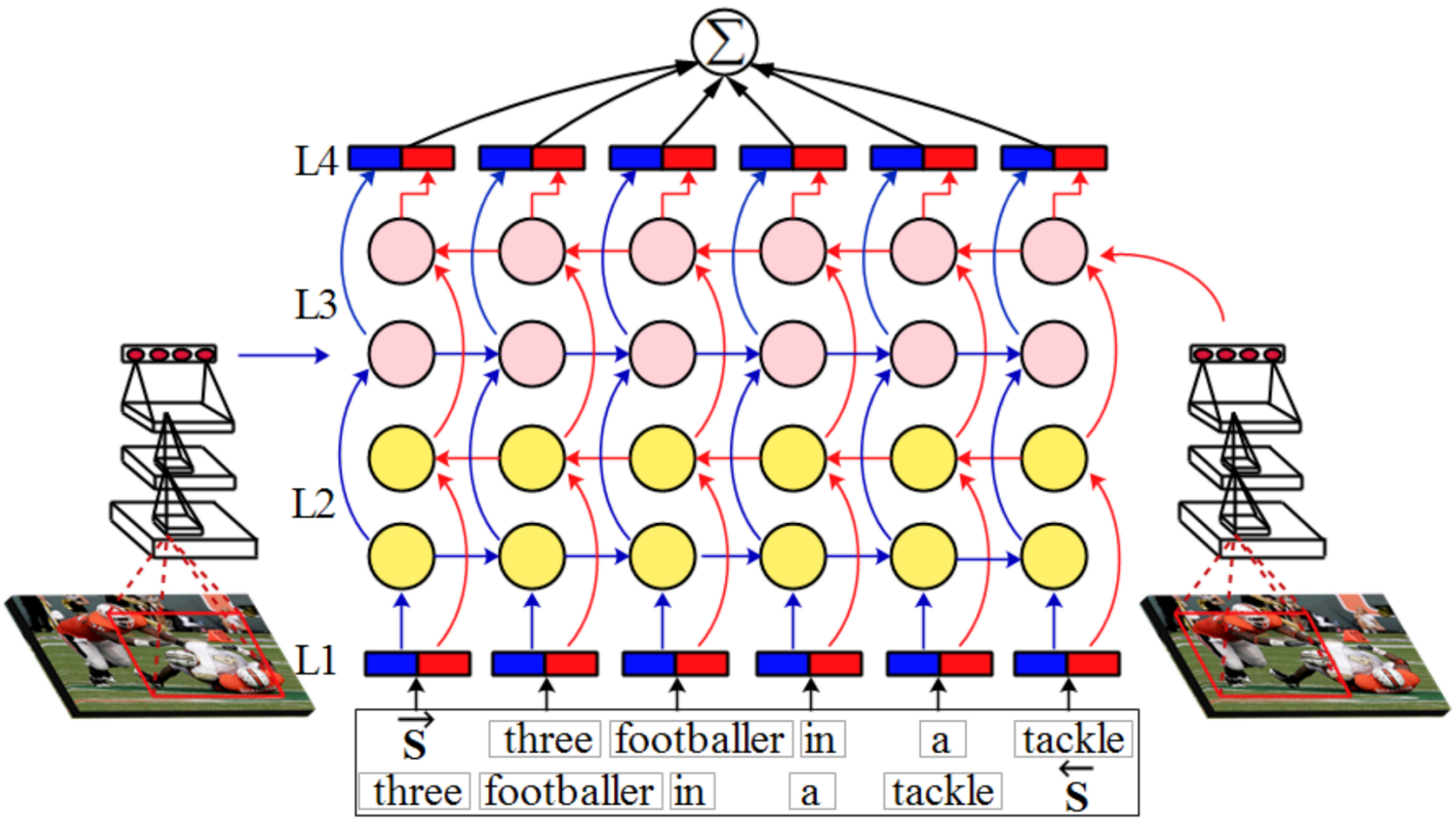
Input: vector representation of image (generated e.g. with a CNN)

Output: sequence of words



Applications

Image captioning: input picture (e.g. w/CNN), output word sequence



(b)

A living room with a couch and a table.

Wang *et al*. (2016) *Image Captioning with Deep Bidirectional LSTMs*

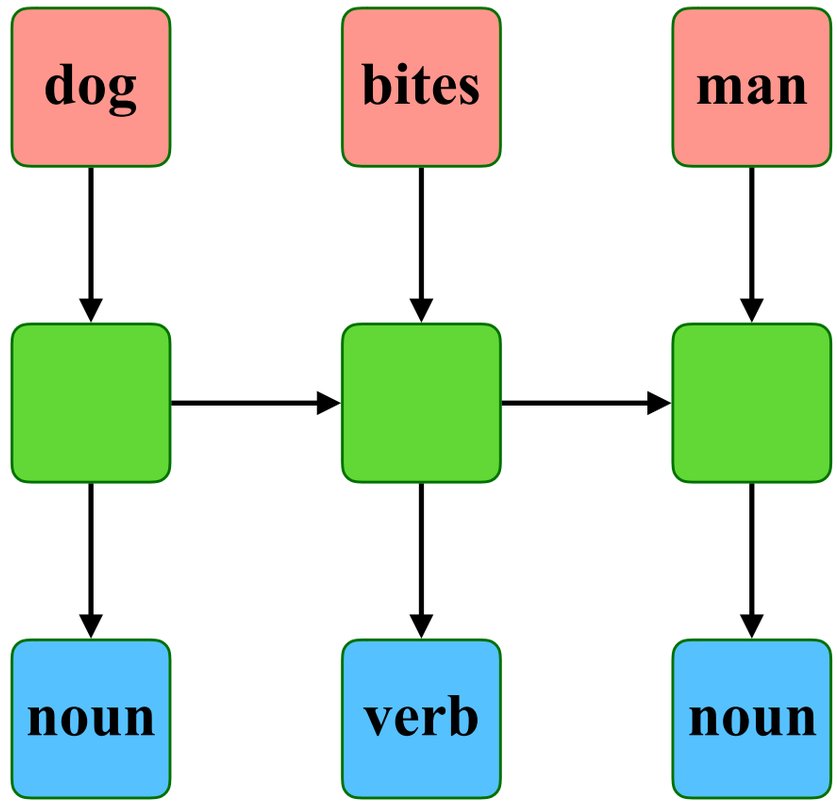
Applications

Many to many: sequence labeling
(e.g. part-of-speech tagging, NER)

Input: sequence of words

Output: sequence of labels

(Variable length, but number of outputs matches number of inputs)



Applications

Named entity recognition: input word sequence, output tag sequence

Turun yliopisto (lyhenne TY tai UTU) on ensimmäinen täysin suomenkielinen yliopisto, joka perustettiin 1920. Elokuussa 2019 yliopiston rehtorina aloitti Jukka Kola. Yliopiston viimeisenä kanslerina ennen instituution lakkauttamista toimi vuosina 2010–2013 Pekka Puska. Yliopiston Turun kampus sijaitsee kaupungin keskustan alueella Yliopistonmäen läheisyydessä. Yliopisto toimii Turun lisäksi Raumalla (lastentarha-, luokanopettaja- ja käsityöopettajakoulutus) ja Porissa (kauppakorkeakoulu, kulttuurituotanto ja maisemantutkimus).
Turun Akatemia
Turkuun perustettiin jo vuonna 1640 yliopisto, Turun Akatemia, joka on nykyisen Helsingin yliopiston edeltäjä. Se oli kolmas Ruotsin valtakuntaan perustettu yliopisto ja Suomen ainoa. Turun palon jälkeen vuonna 1827 Akatemia siirrettiin Helsinkiin. Akatemian siirtämisen jälkeen yliopistollinen opetus Turussa loppui vuoteen 1917 asti.

```
Erik          B-PER
Justander     I-PER
oli           O
Turun         B-ORG
akatemian     I-ORG
professori    O
.             O
```

Luoma *et al*. (2020) *A Broad-coverage Corpus for Finnish Named Entity Recognition*

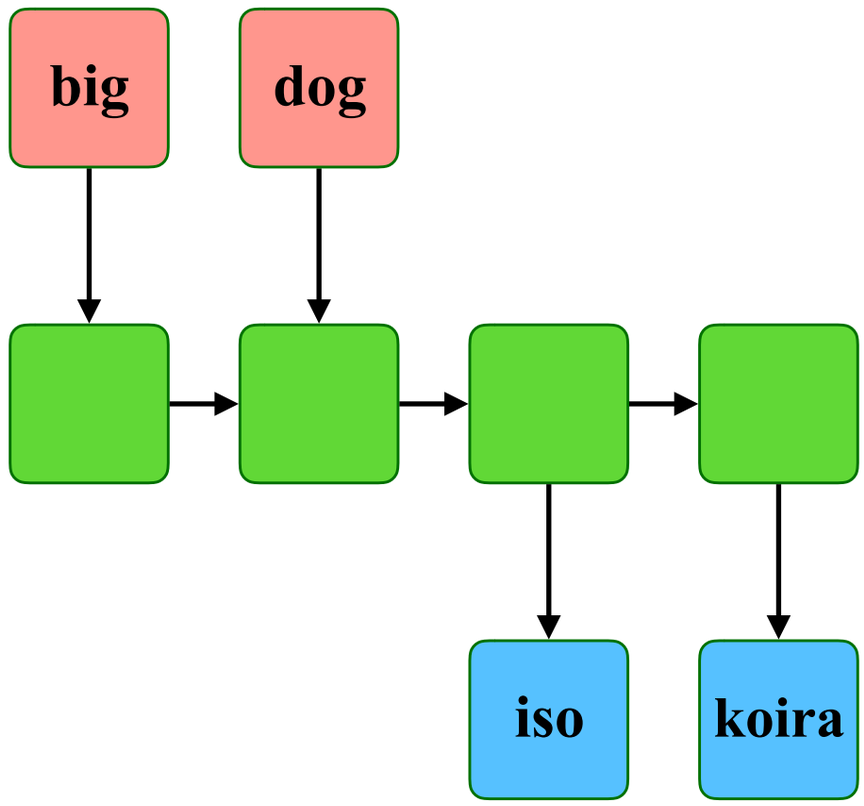
Applications

Many to many: machine translation

Input: sequence of words in one language

Output: sequence of words in another language

Similarly: text summarization, dialogue systems / chatbots, etc.



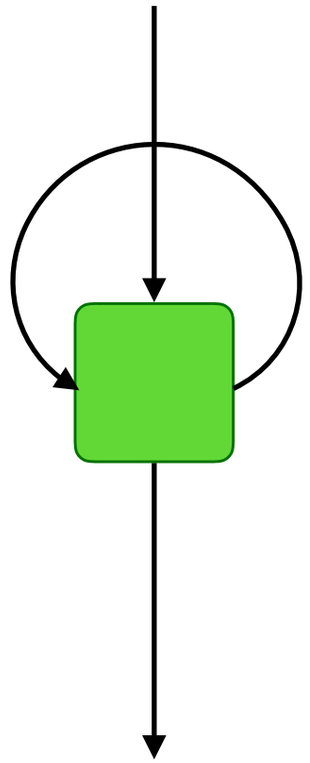
Sutskever *et al*. (2014) *Sequence to Sequence Learning with Neural Networks*

Summary

Recurrent neural networks are neural networks with **cycles**

“Memory” of previous inputs through **hidden state**

Support for **variable-length** inputs and outputs of arbitrary length



Applicable to tasks with **many-to-one**, **one-to-many** and **many-to-many** mappings between inputs and outputs

Applications in text classification, sequence labeling, machine translation, image captioning, text generation

(So, what’s the catch?)

Recurrent Neural Networks in Practice

Vanilla RNN

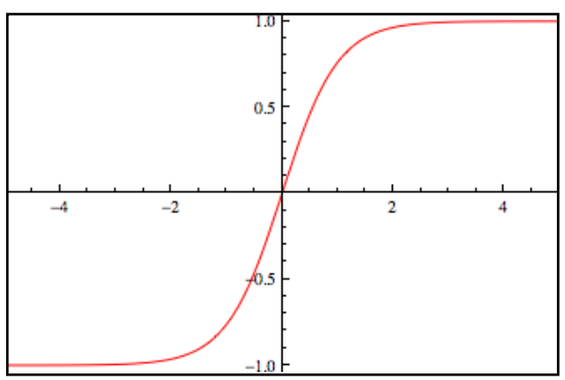
Reminder: perceptron

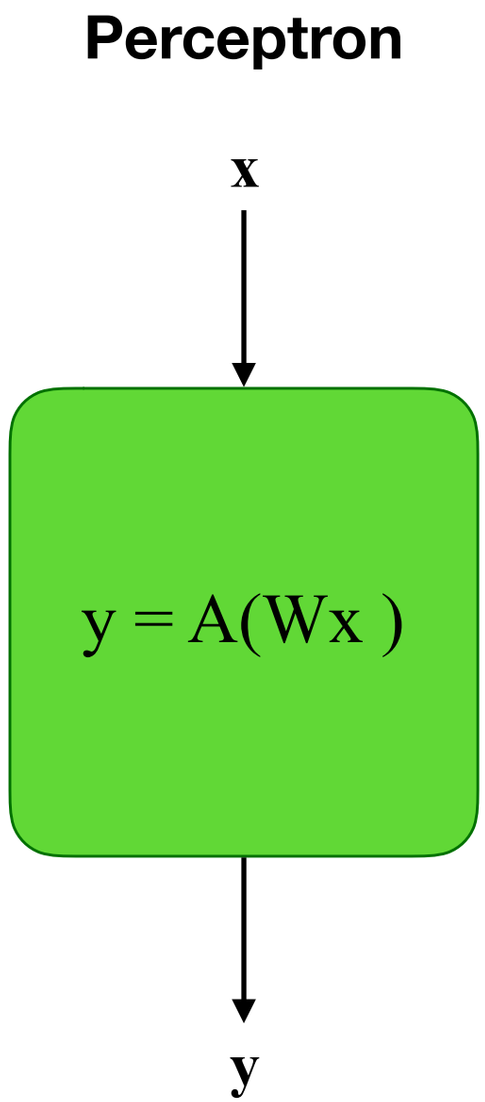
x : input

W: weights

A : activation function (e.g. tanh)

y : output





Vanilla RNN

Vanilla RNN cell

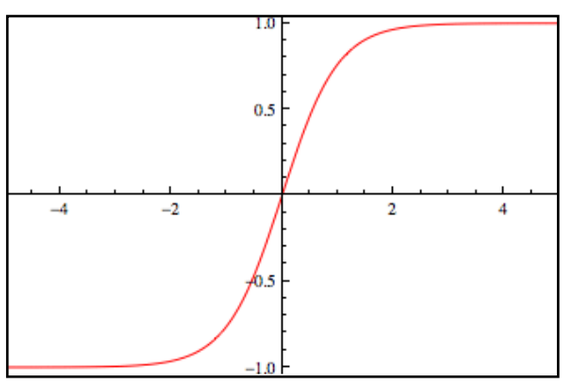
h0 : hidden state for step 0

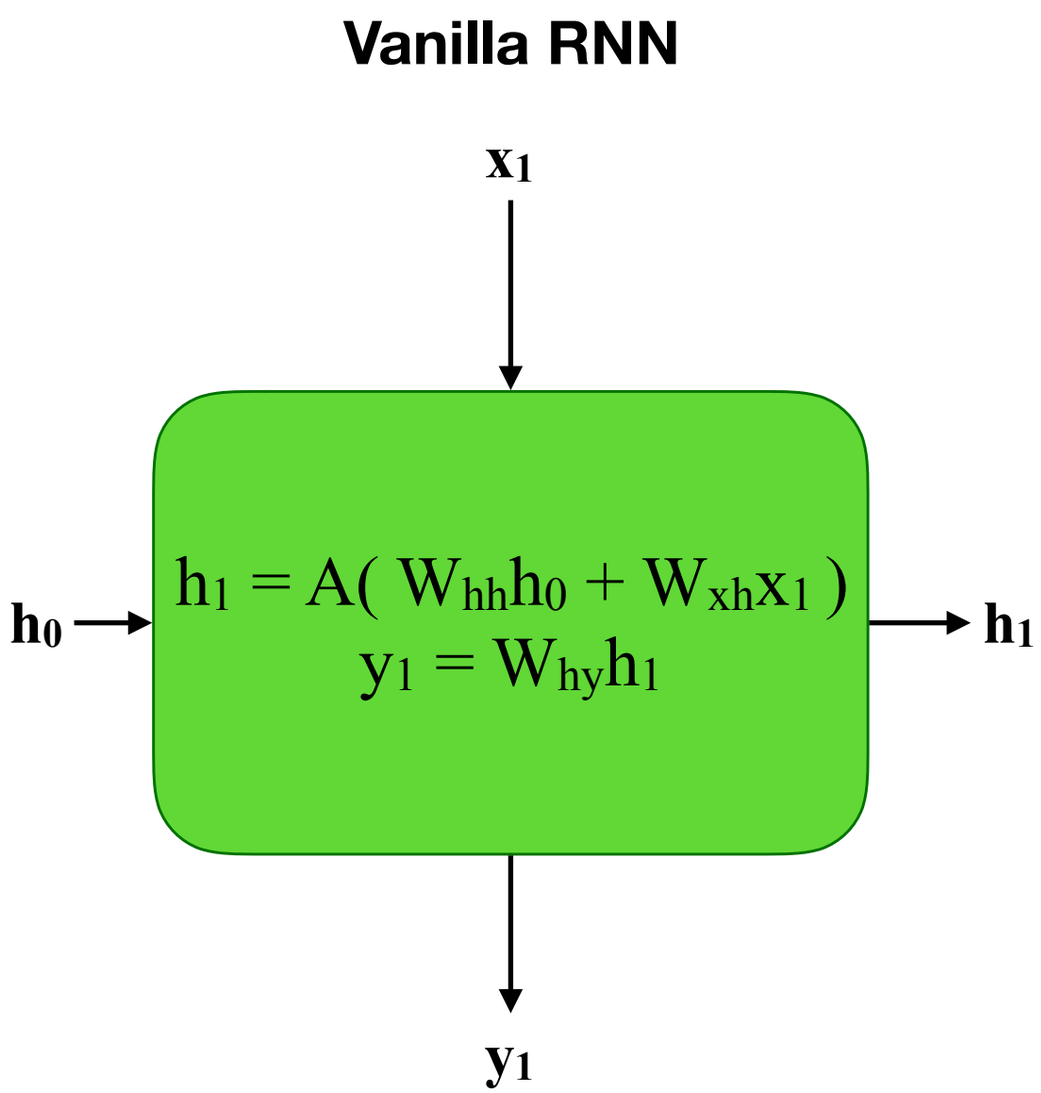
x1 : input for step 1

W_{hh}, W_{xh}, W_{hy}: weights

A : activation function (e.g. tanh)

y1 : output





A code perspective

```
rnn = RNN()
y = rnn.step(x)
```

```
class RNN:
  def __init__(self, h_dim):
    self.h = zeros(h_dim)
    …

  def step(self, x):
    self.h = tanh(dot(self.W_hh, self.h) +
                  dot(self.W_hx, x))
    return dot(self.W_hy, self.h)
```

Karpathy (2015) *The Unreasonable Effectiveness of Recurrent Neural Networks*

A code perspective

```
rnn = RNN()
y = rnn.step(x)
```

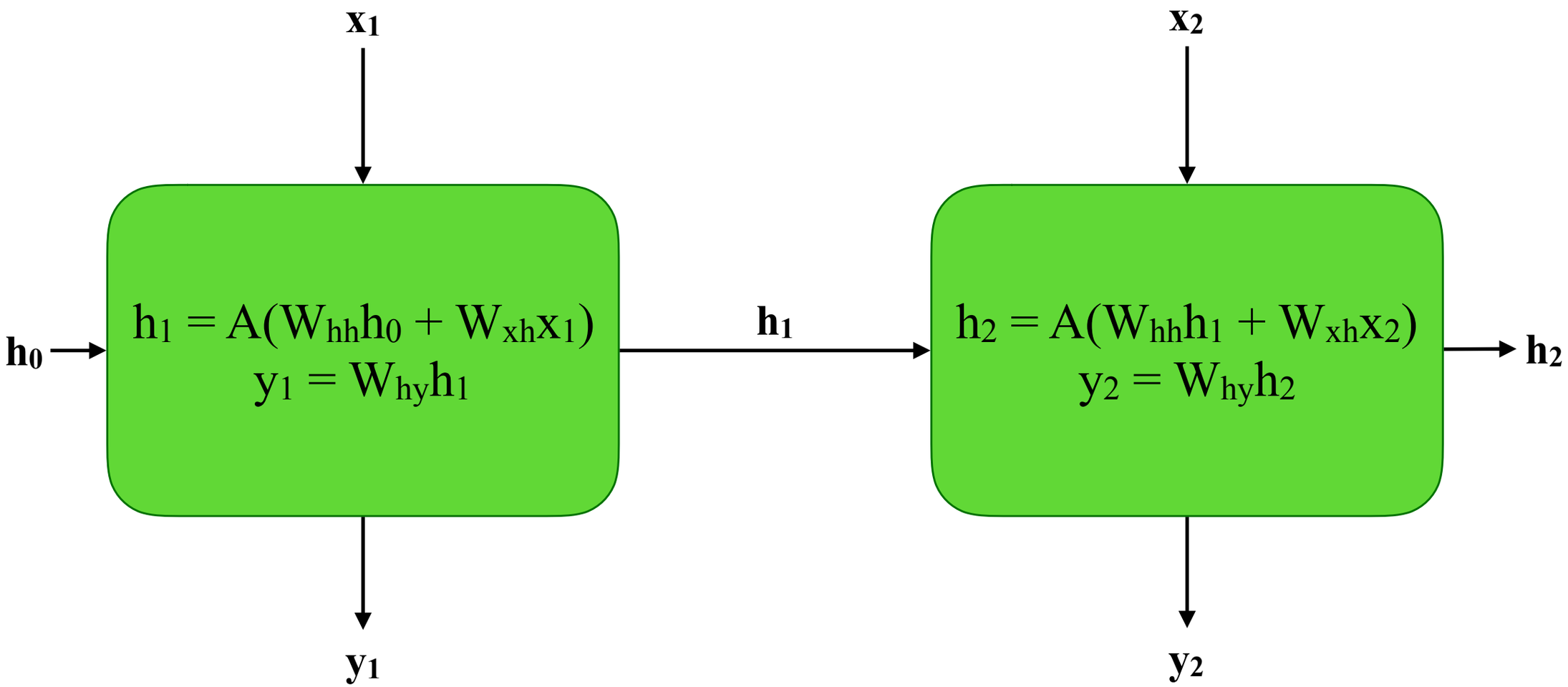
```
class RNN:
  def __init__(self, h_dim):
    self.h = zeros(h_dim)
    …

  def step(self, x):
    self.h = tanh(dot(self.W_hh, self.h) +
                  dot(self.W_hx, x))
    return dot(self.W_hy, self.h)
```

Note keras / numpy conventions, esp. `dot` is matrix multiplication

Karpathy (2015) *The Unreasonable Effectiveness of Recurrent Neural Networks*

Vanilla RNN

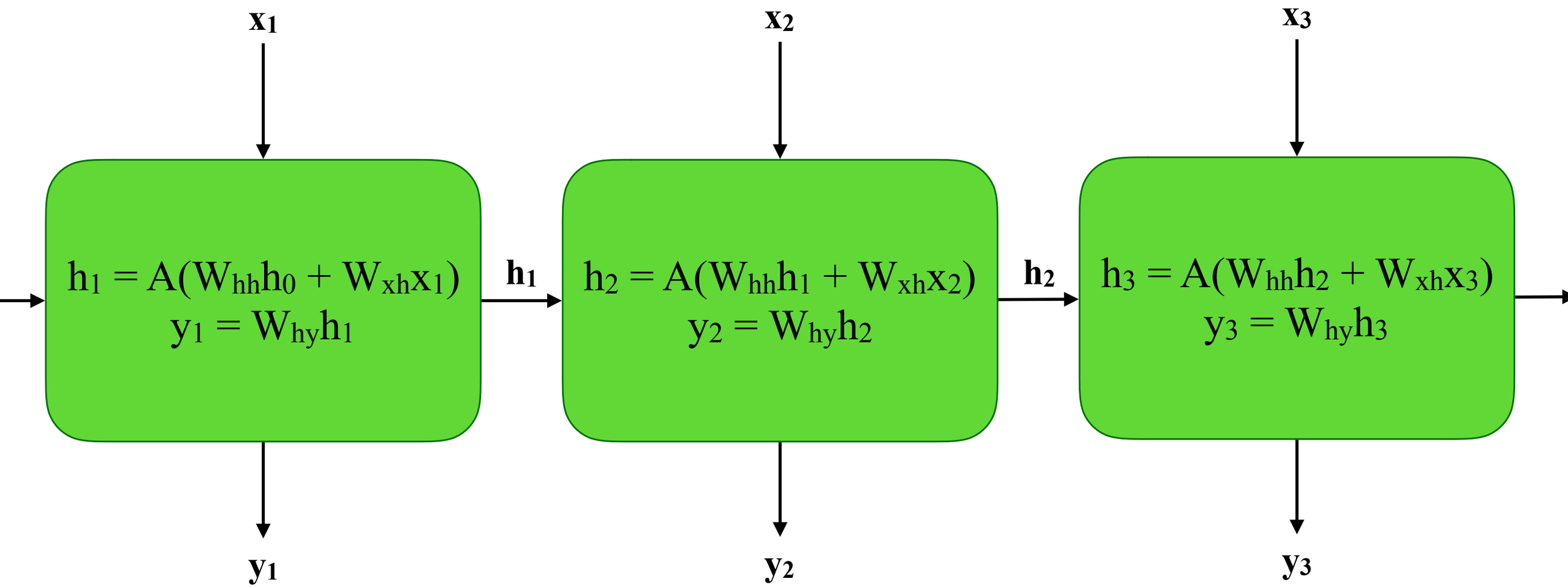
Like any NN, an RNN is just a function



$$h_2 = A(W_{hh}h_1 + W_{xh}x_2) = A(W_{hh}(A(W_{hh}h_0 + W_{xh}x_1)) + W_{xh}x_2)$$

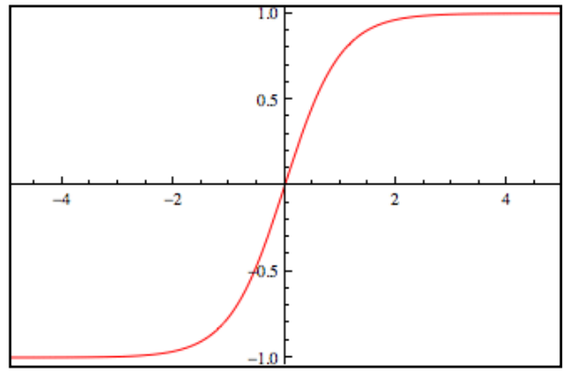
Vanilla RNN

Like any NN, an RNN is just a function

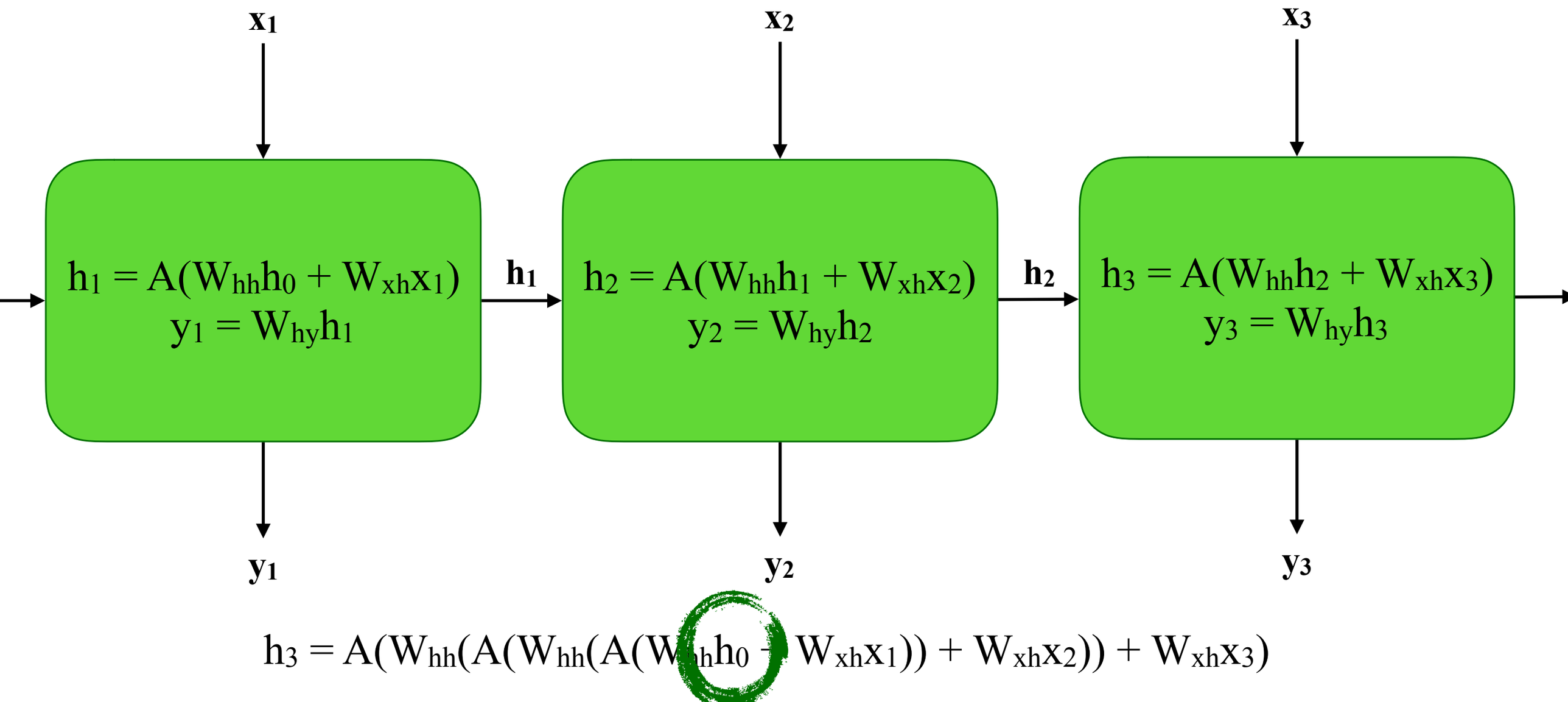


$$h_3 = A(W_{hh}(A(W_{hh}(A(W_{hh}h_0 + W_{xh}x_1)) + W_{xh}x_2)) + W_{xh}x_3)$$

Vanilla RNN



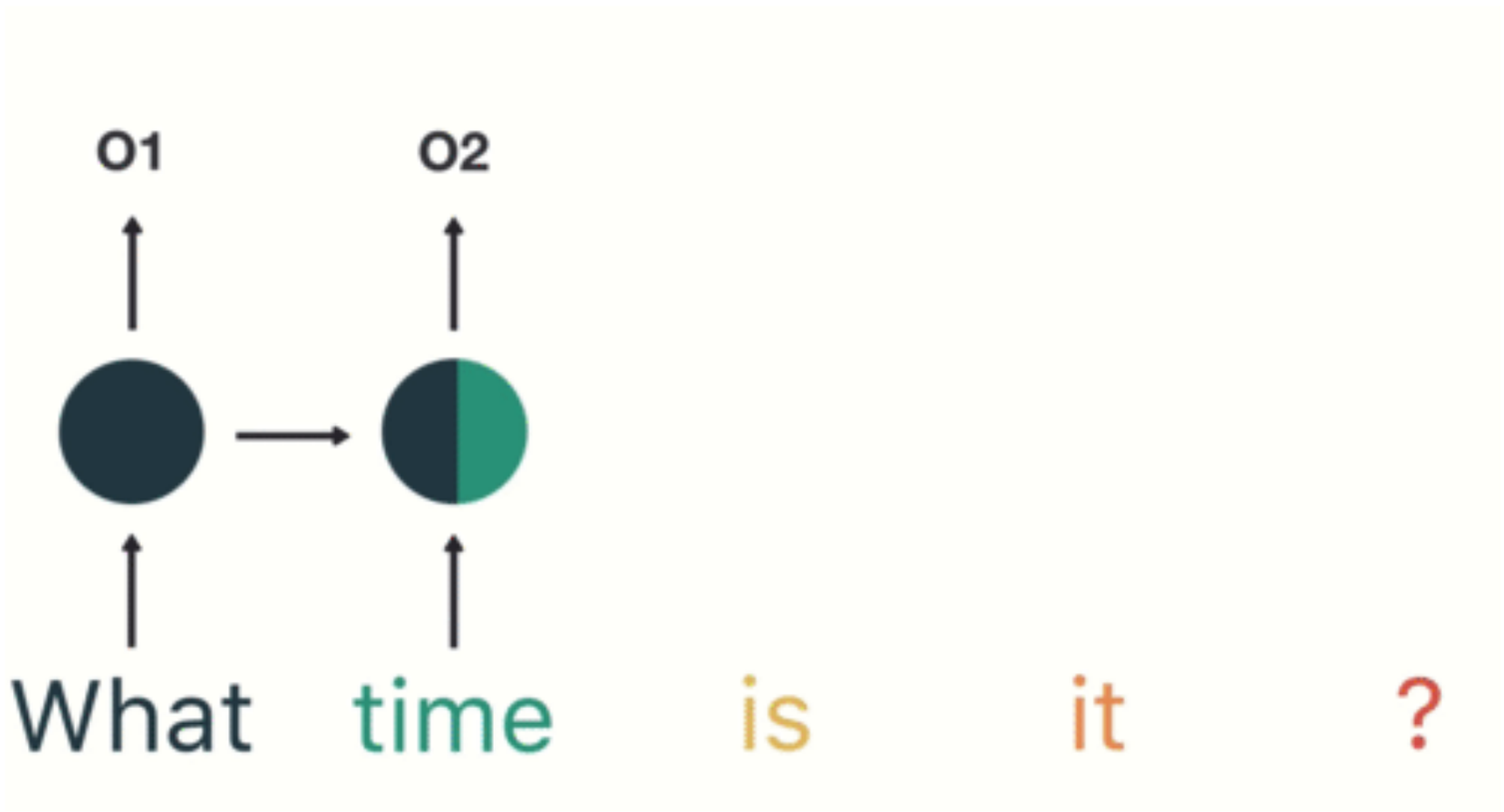
h repeatedly “squished” together with input by activation function



$$h_3 = A(W_{hh}(A(W_{hh}(A(W_{hh}h_0 + W_{xh}x_1)) + W_{xh}x_2)) + W_{xh}x_3)$$

Limitations

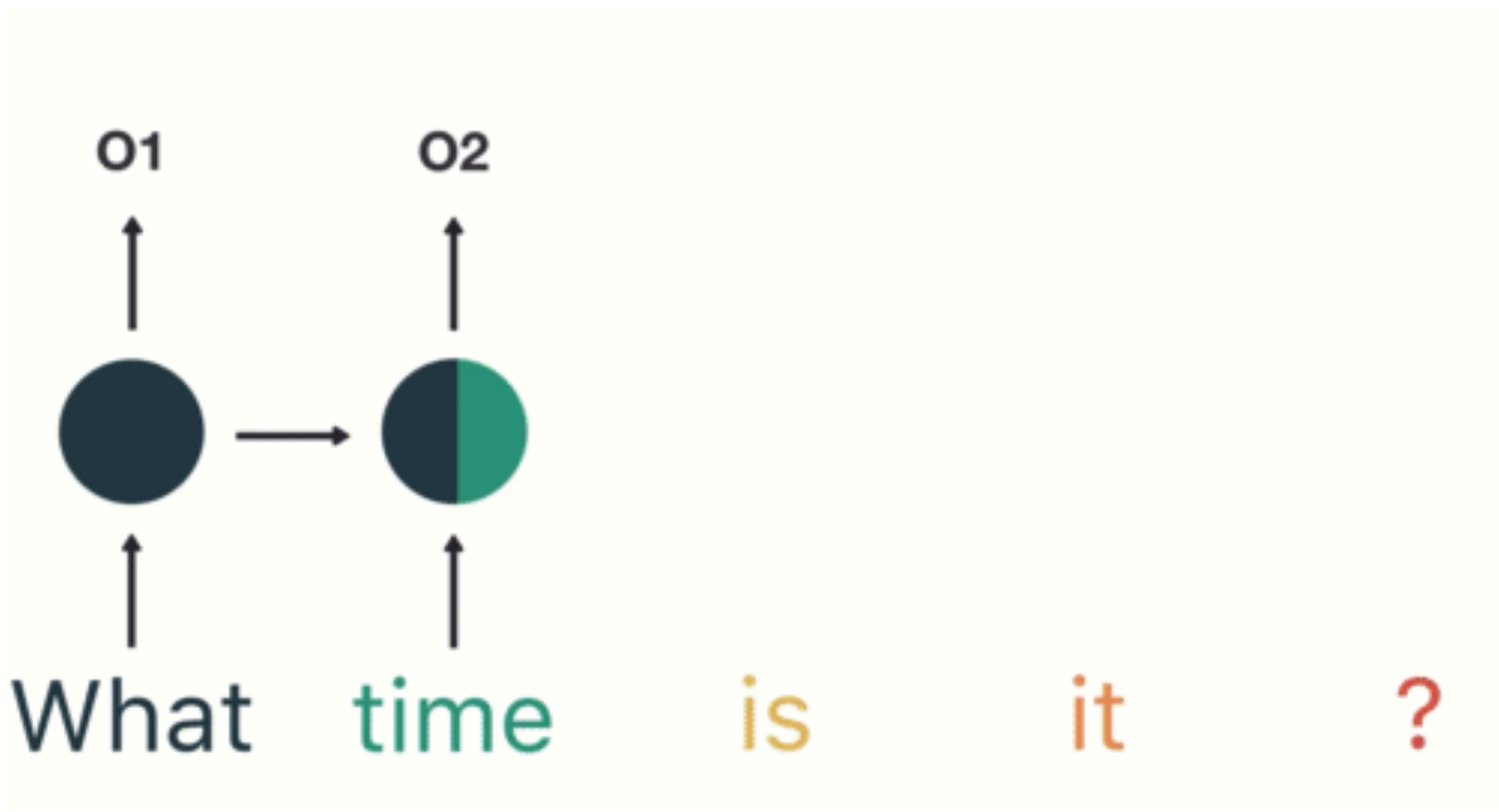
Vanilla RNNs have bad short-term memory



Nguyen (2018) *Illustrated Guide to Recurrent Neural Networks*

Limitations

Vanilla RNNs have bad short-term memory

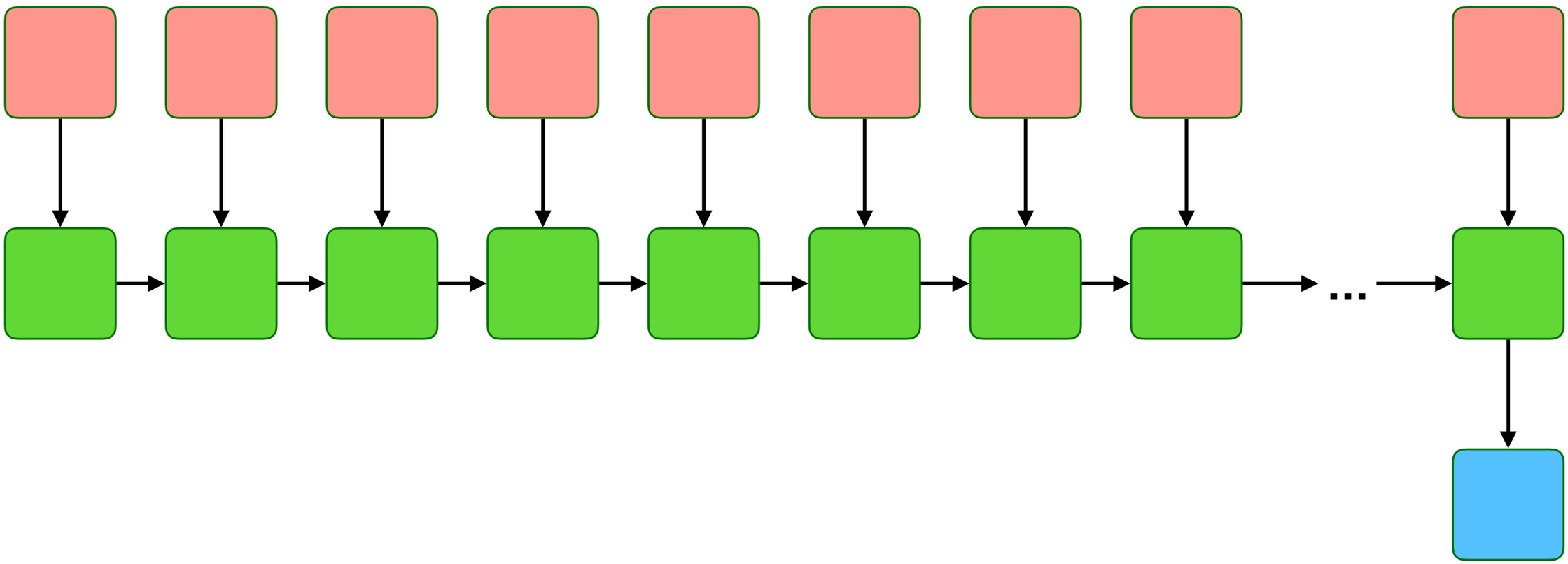


Hidden state at step 5

Nguyen (2018) *Illustrated Guide to Recurrent Neural Networks*

Limitations

Vanilla RNNs are hard to train

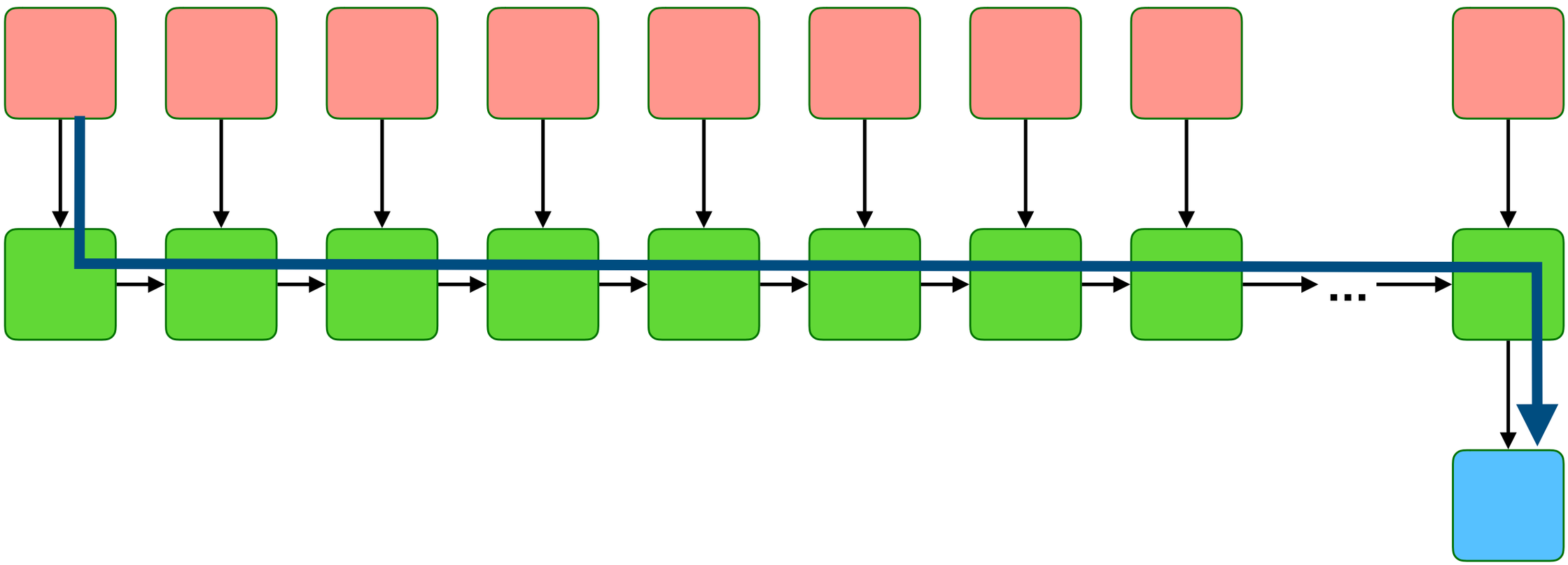


Bengio *et al*. (1994) *Learning Long-Term Dependencies with Gradient Descent is Difficult*

Limitations

Forward pass

Vanilla RNNs are hard to train



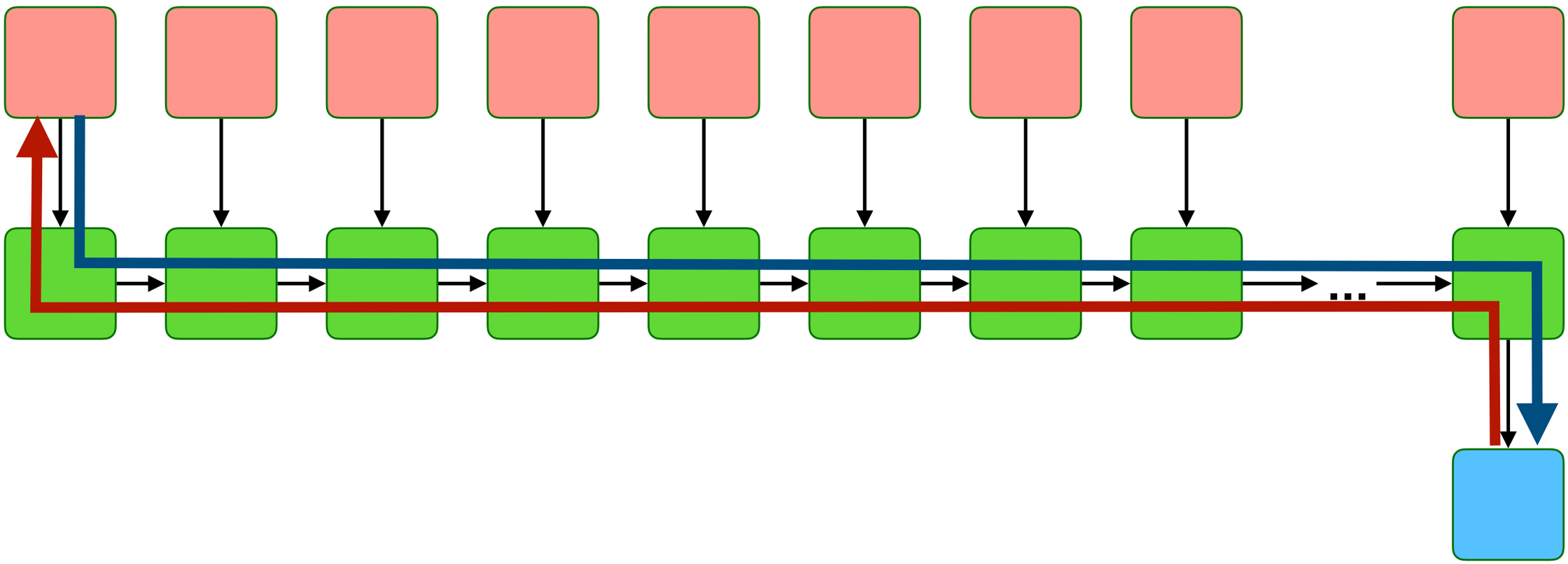
Bengio *et al*. (1994) *Learning Long-Term Dependencies with Gradient Descent is Difficult*

Limitations

Forward pass

Backpropagation

Vanilla RNNs are hard to train



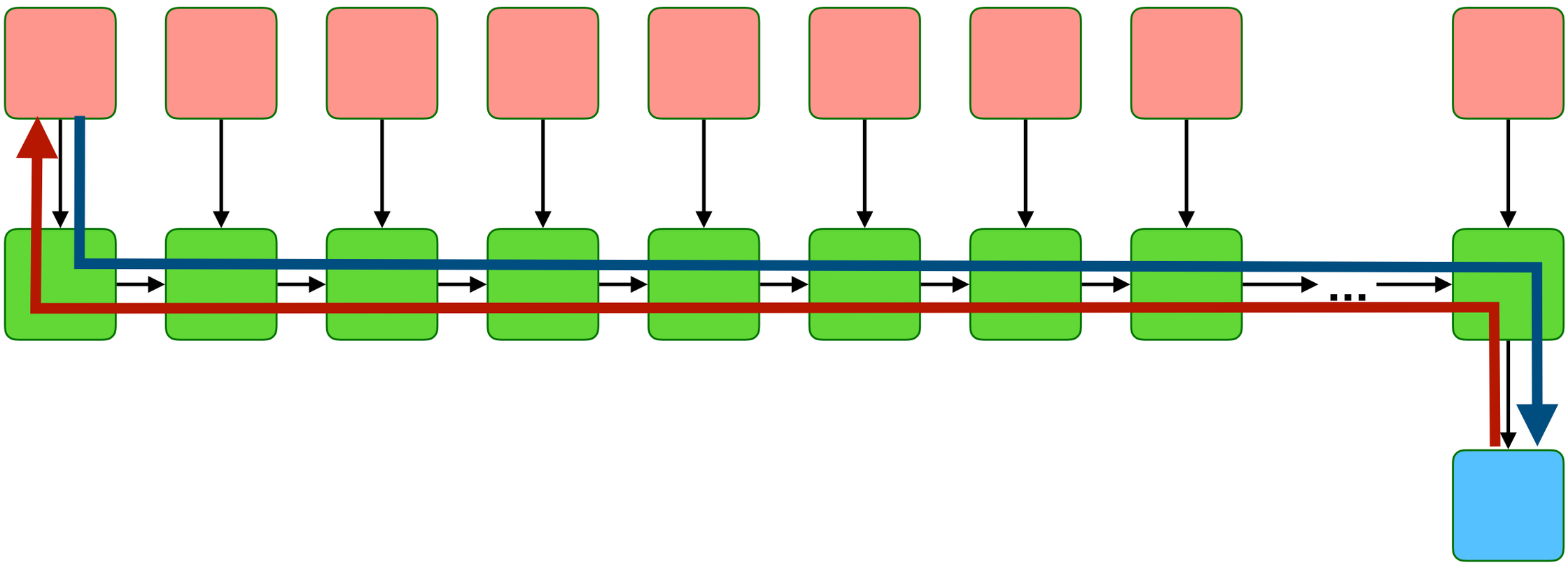
Bengio *et al*. (1994) *Learning Long-Term Dependencies with Gradient Descent is Difficult*

Limitations

Forward pass

Backpropagation

Vanilla RNNs are hard to train



Bengio *et al*. (1994) *Learning Long-Term Dependencies with Gradient Descent is Difficult*

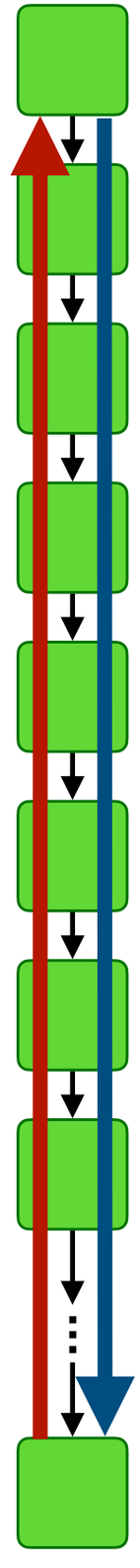
Limitations

Gradient calculated over repeated applications of W_{hh} and A, values multiplied together (chain rule)

- Many values > 1 → **exploding gradients**
- Many values < 1 → **vanishing gradients**

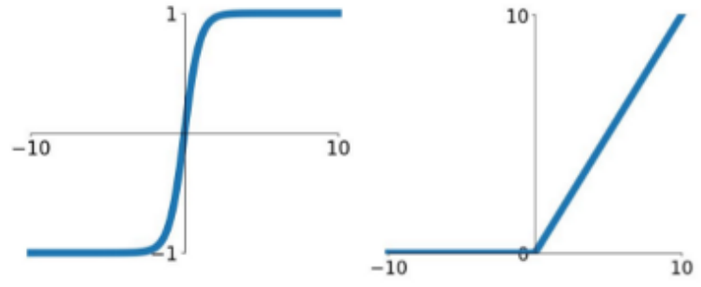
Exploding gradients can be addressed e.g. through gradient clipping

Vanishing gradients are a challenging problem not only for RNNs but for any deep architecture



Limitations

Alleviating (not solving!) vanishing gradients:

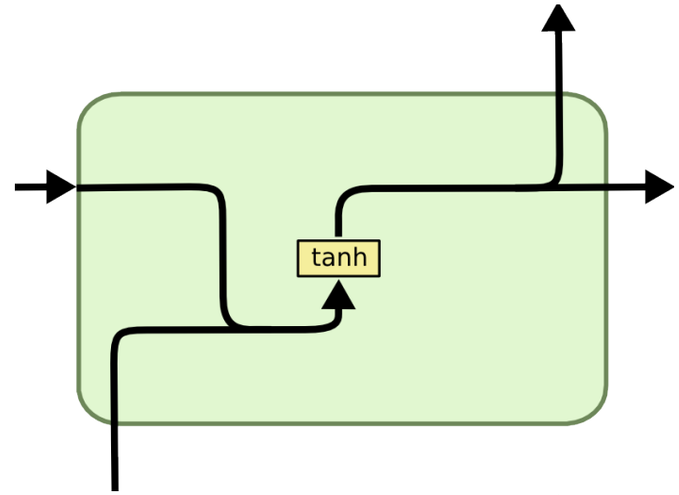


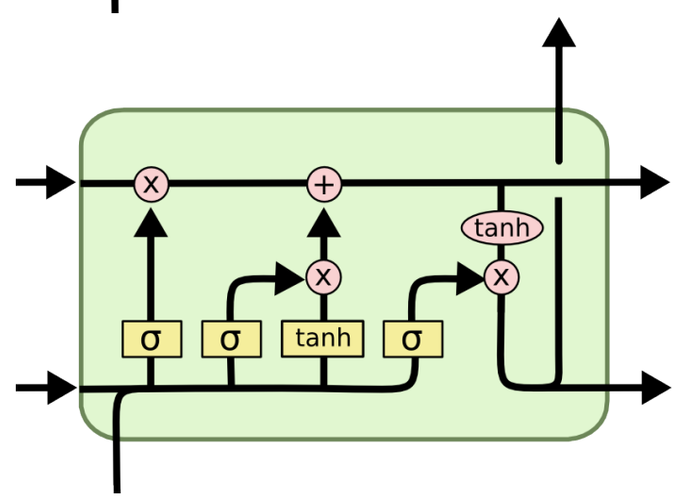
Alternative activation functions (e.g. ReLU)

Alternative optimizers and training approaches

Alternative RNN cell designs

- Long short-term memory (LSTM)
- Gated recurrent unit (GRU)



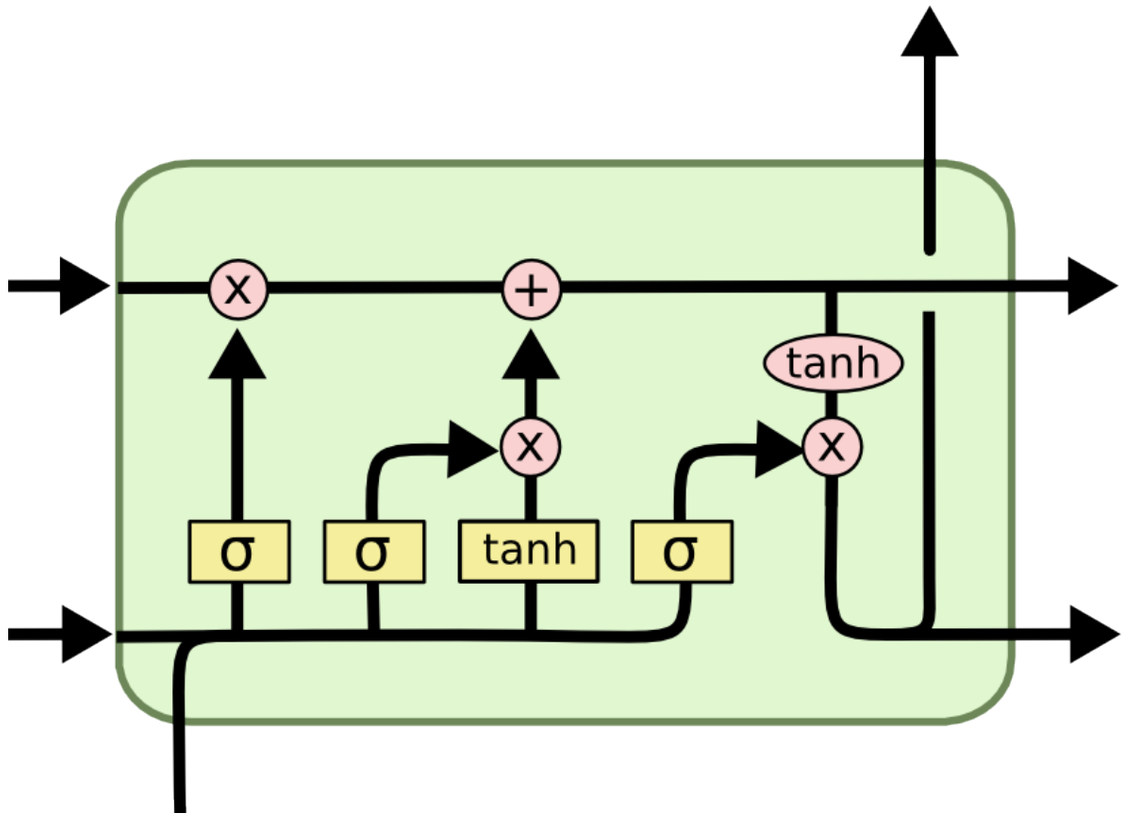


LSTM illustrations from Olah (2015) *Understanding LSTM Networks*

LSTMs

RNN cell design aiming to alleviate short-term memory issues

Key point: cell state ("memory") passed through unmodified "by default", changes controlled by series of gates



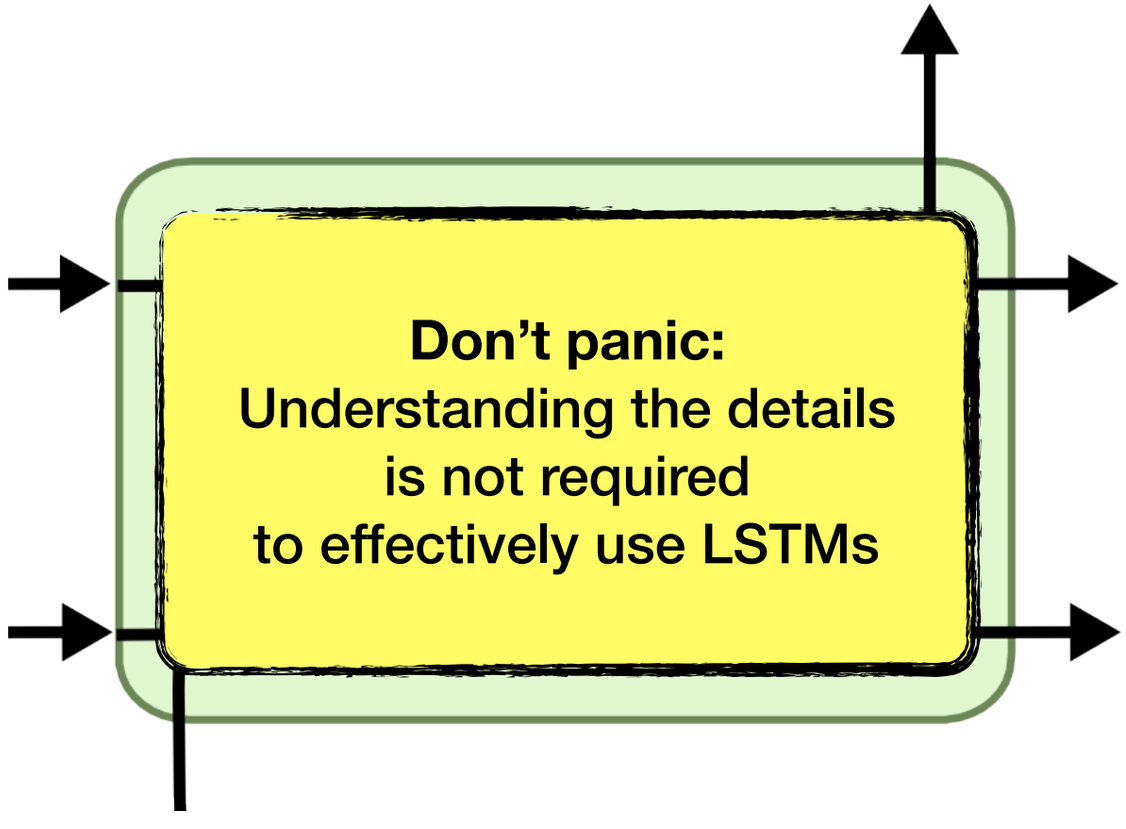
Hochreiter and Schmidhuber (1997) *Long short-term memory*

LSTM illustrations from Olah (2015) *Understanding LSTM Networks*

LSTMs

RNN cell design aiming to alleviate short-term memory issues

Key point: cell state ("memory") passed through unmodified "by default", changes controlled by series of gates



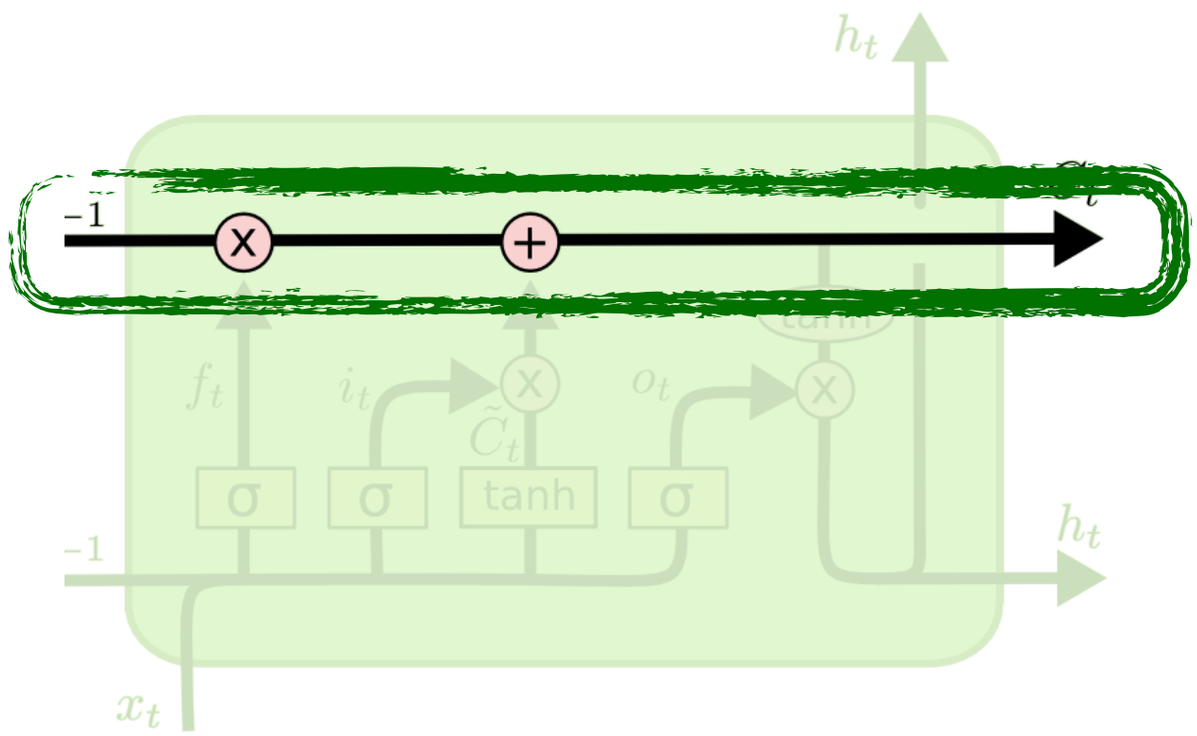
Hochreiter and Schmidhuber (1997) *Long short-term memory*

LSTM illustrations from Olah (2015) *Understanding LSTM Networks*

LSTMs

RNN cell design aiming to alleviate short-term memory issues

Key point: “memory” passed through unmodified by default, changes controlled by series of **gates**



Hochreiter and Schmidhuber (1997) *Long short-term memory*

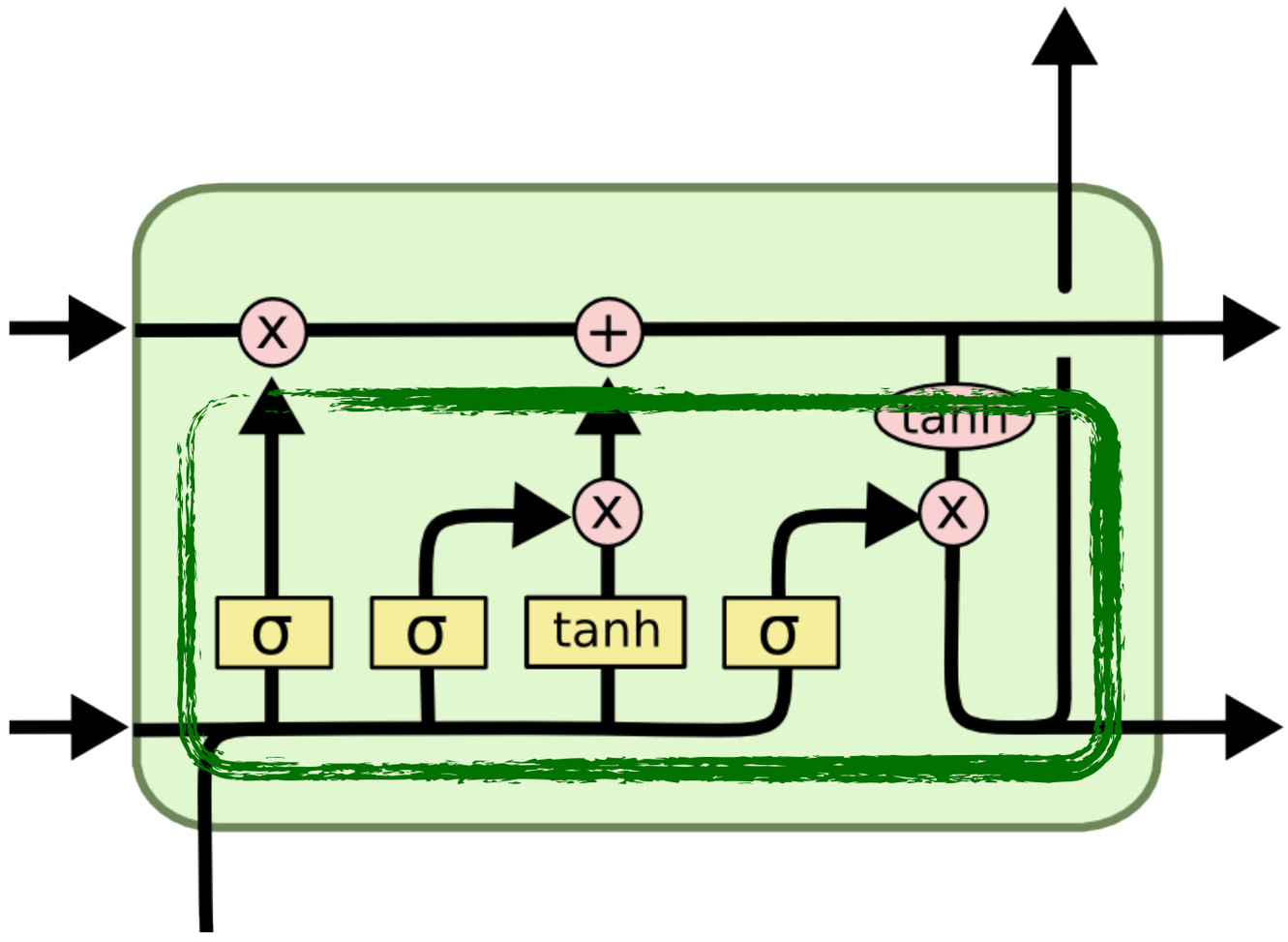
LSTM illustrations from Olah (2015) *Understanding LSTM Networks*

LSTMs

RNN cell design aiming to alleviate short-term memory issues

Key steps:

1. **Forget**
2. **Store**
3. **Update**
4. **Output**



Hochreiter and Schmidhuber (1997) *Long short-term memory*

LSTM illustrations from Olah (2015) *Understanding LSTM Networks*

LSTMs

```
rnn = RNN()
y = rnn.step(x)
```

```
rnn = LSTM()
y = rnn.step(x)
```

LSTM “API” is identical to that of vanilla RNNs

For most purposes, LSTMs, GRUs, etc. are drop-in replacements for vanilla RNNs

It is not necessary to understand the mathematical details to make effective use of these methods!

Resources

Recurrent Neural Networks | Stanford CS 231N
https://www.youtube.com/watch?v=6niqTuYFZLQ

Recurrent Neural Networks | MIT 6.S191
https://www.youtube.com/watch?v=SEnXr6v2ifU

Chris Olah: Understanding LSTM Networks
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Michael Nguyen: Illustrated Guide to LSTM's and GRU's: A step by step explanation
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21