Recurrent Neural Networks

- Recurrent neural networks (RNNs) are a class of neural network that are helpful in modeling sequence data.
- Derived from feedforward networks, RNNs exhibit similar behavior to how human brains function.
- Simply put: recurrent neural networks produce predictive results in sequential data that other algorithms can’t.

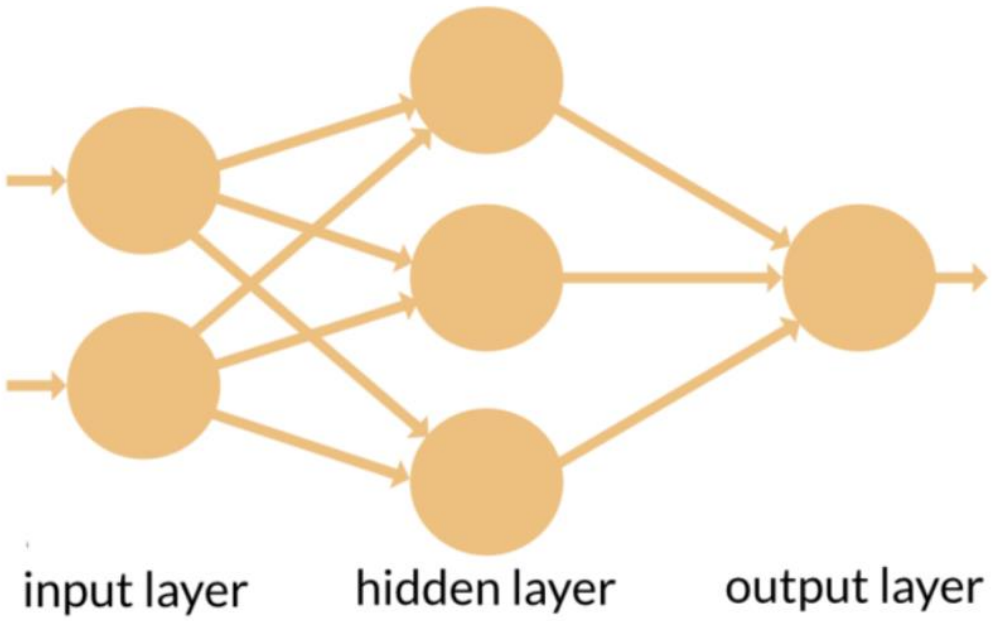
- Because of their internal memory, RNNs can remember important things about the input they received, which allows them to be very precise in predicting what's coming next.
- This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more.
- Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.
- Sequential data is basically just ordered data in which related things follow each other.
- Examples are financial data or the DNA sequence.
- The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

- But when do you need to use an RNN?

“Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame.” – Lex Fridman (MIT)

Since RNNs are being used in the software behind Siri and Google Translate, recurrent neural networks show up a lot in everyday life.

RECURRENT VS. FEED-FORWARD NEURAL NETWORKS

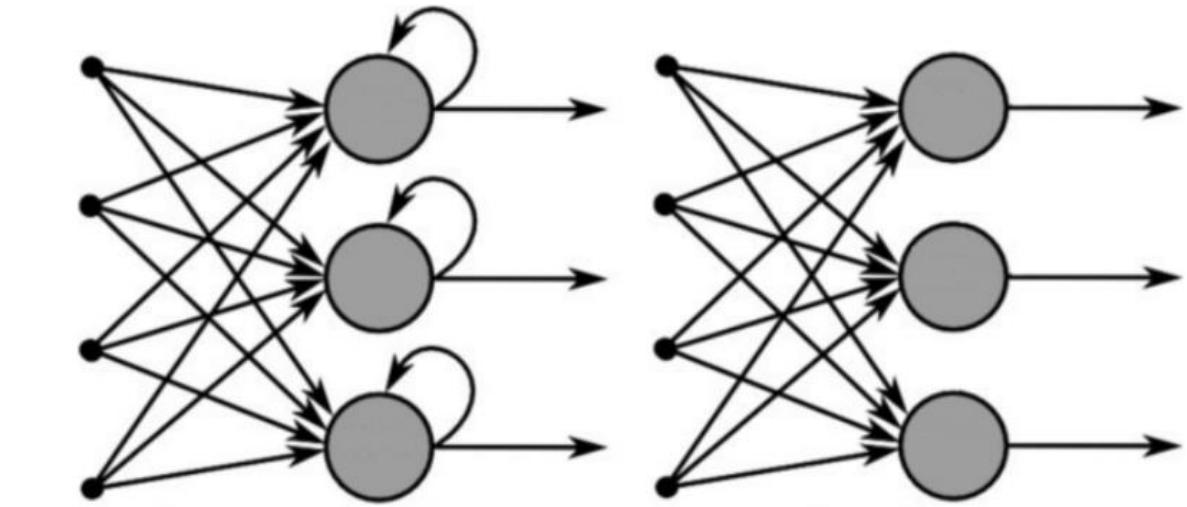


- In a feed-forward neural network, the information only moves in one direction — from the input layer, through the hidden layers, to the output layer.
- The information moves straight through the network.

Feed-forward neural networks have no memory of the input they receive and are bad at predicting what's coming next.
Because a feed-forward network only considers the current input, it has no notion of order in time.

It simply can't remember anything about what happened in the past except its training.

- In an RNN, the information cycles through a loop.
- When it makes a decision, it considers the current input an also what it has learned from the inputs it received previously.
- A usual RNN has a short-term memory. In combination with an LSTM they also have a long-term memory



Recurrent Neural Network

Feed-Forward Neural Network

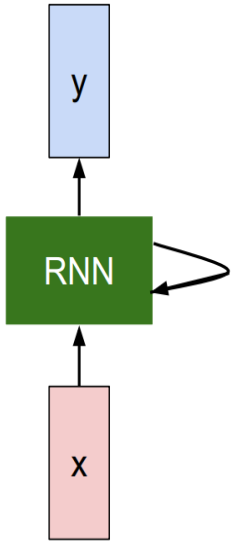
A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network

Imagine you have a normal feed-forward neural network and give it the word “neuron” as an input and it processes the word character by character.

By the time it reaches the character “r,” it has already forgotten about “n,” “e” and “u,” which makes it almost impossible for this type of neural network to predict which character would come next.

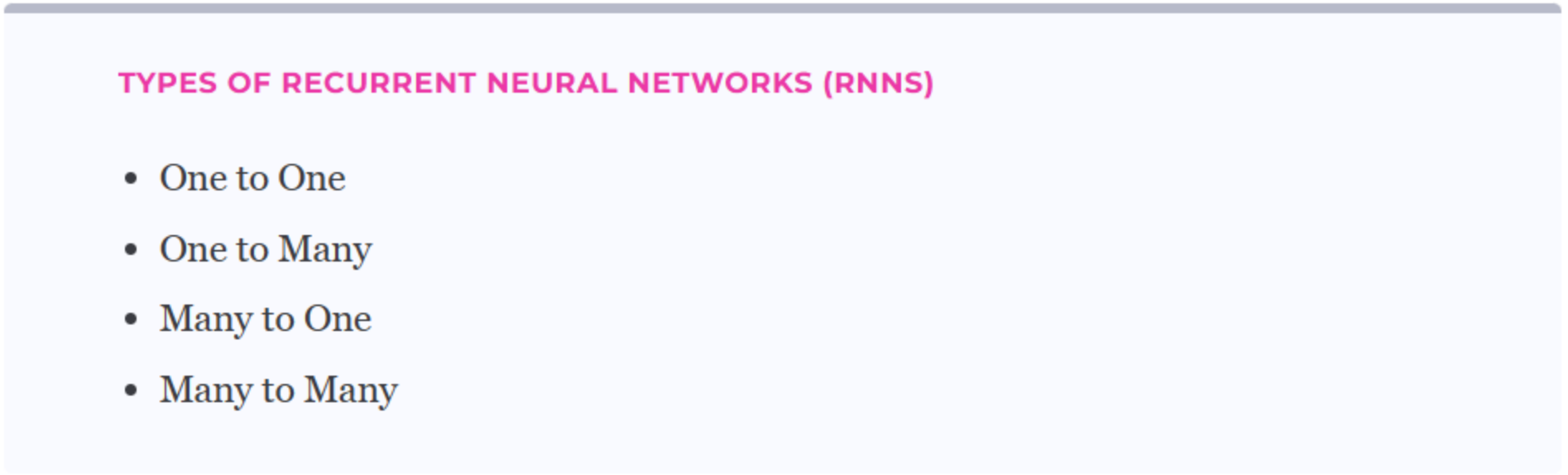
Simply put: Recurrent neural networks add the immediate past to the present.

Recurrent Neural Network



- An RNN has two inputs: the present and the recent past.
- This is important because the sequence of data contains crucial information about what is coming next, which is why an RNN can do things other algorithms can't.
- A feed-forward neural network assigns, like all other deep learning algorithms, a weight matrix to its inputs and then produces the output.
- RNNs apply weights to the current and also to the previous input. Furthermore, a recurrent neural network will also tweak the weights for both gradient descent and backpropagation through time.

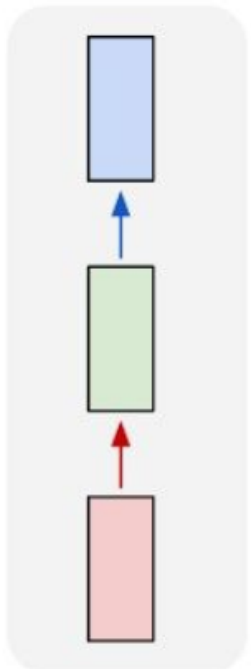
Types of Recurrent Neural Networks



- Feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).

“Vanilla” Neural Network

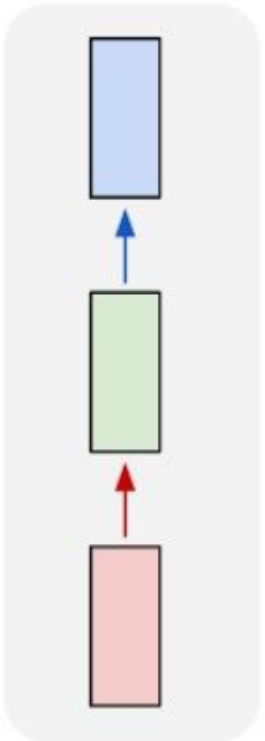
one to one



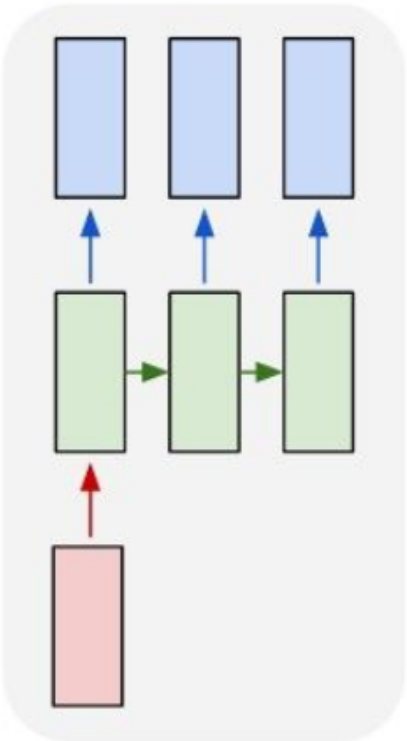
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

one to one

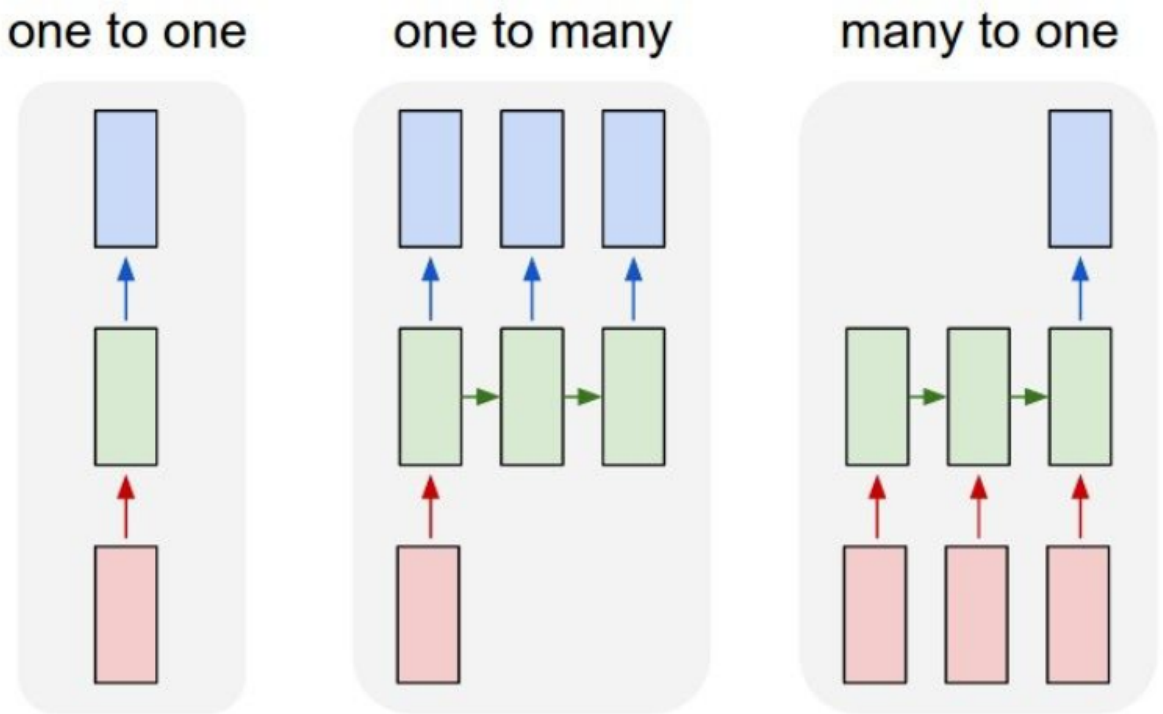


one to many



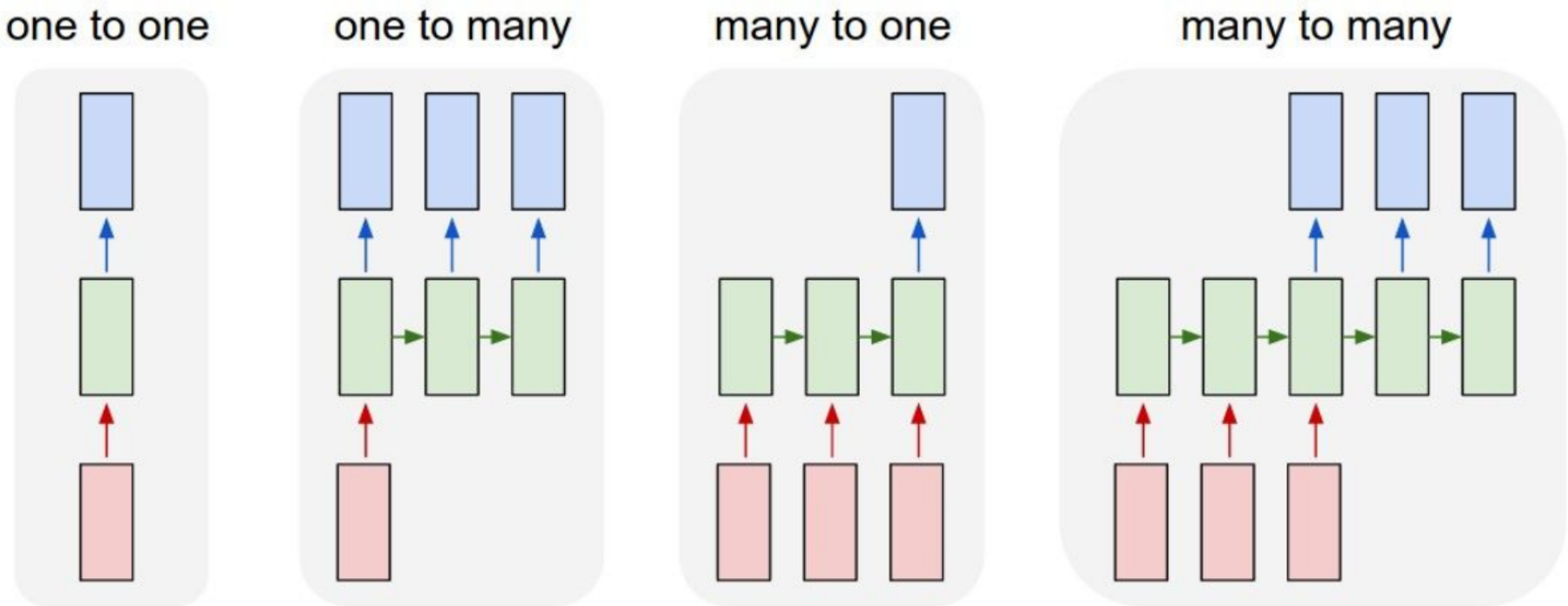
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



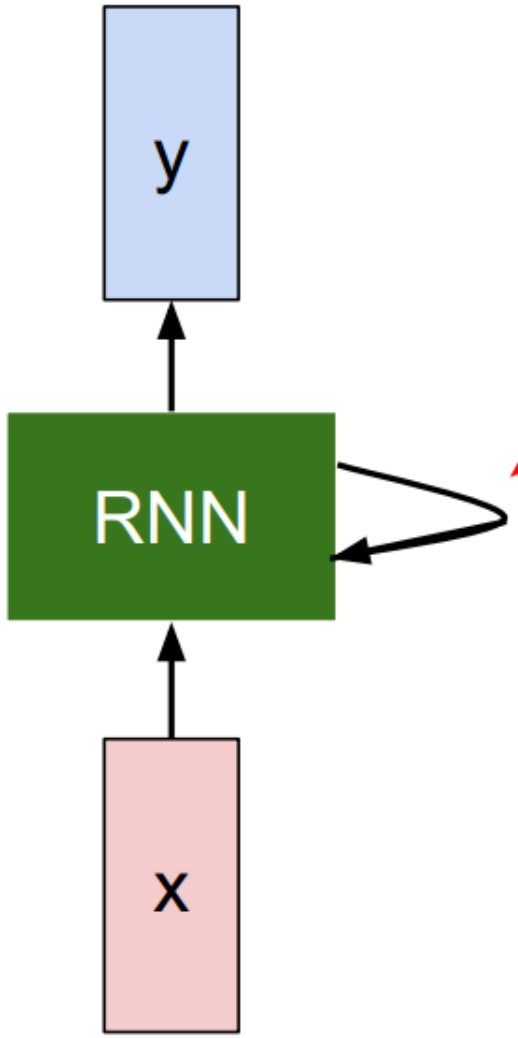
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



E.g. **Video Captioning**
Sequence of video frames -> caption

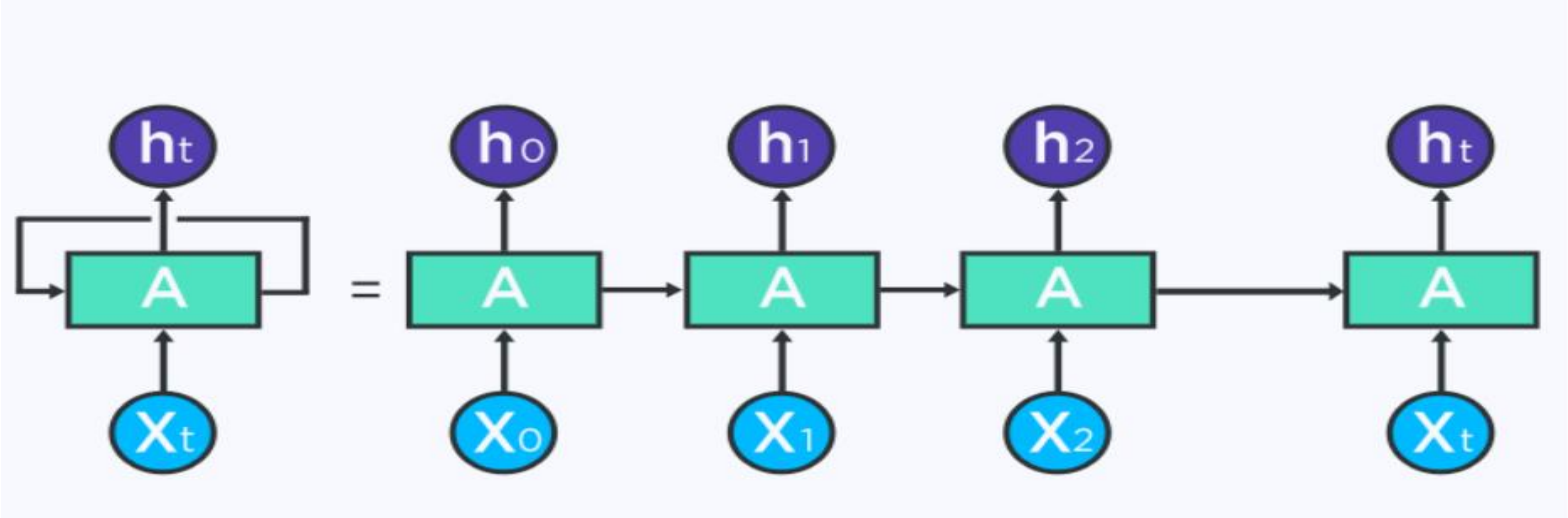
Recurrent Neural Network



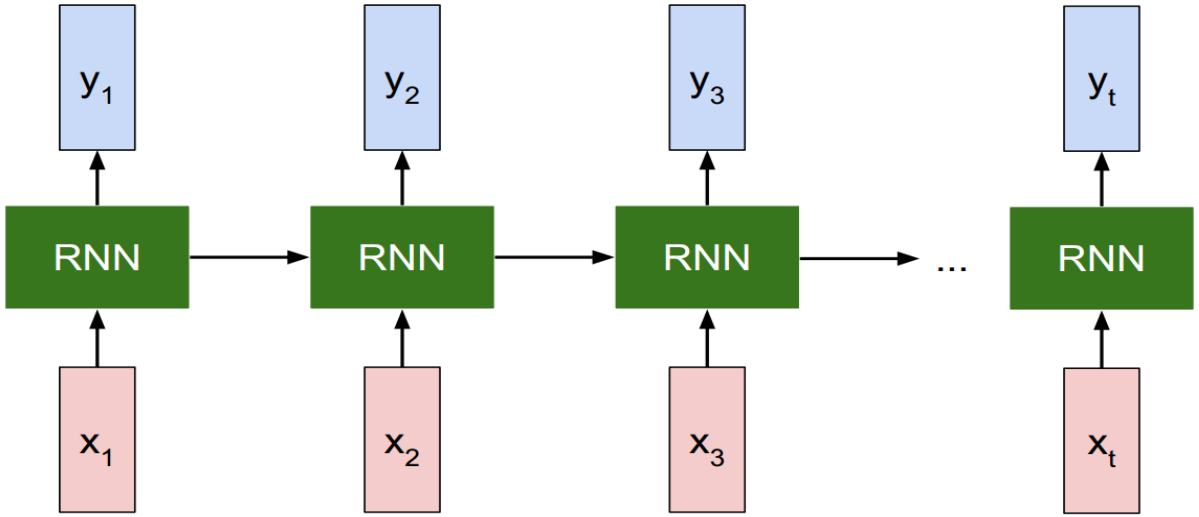
Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Backpropagation Through Time

- BPTT is basically just a fancy buzzword for doing backpropagation on an unrolled recurrent neural network.
- Unrolling is a visualization and conceptual tool, which helps you understand what's going on within the network.
- You can view an RNN as a sequence of neural networks that you train one after another with backpropagation.



Unrolled RNN



- Within BPTT the error is backpropagated from the last to the first time step, while unrolling all the time steps.
- This allows calculating the error for each time step, which allows updating the weights.
- Note that BPTT can be computationally expensive when you have a high number of time steps

RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

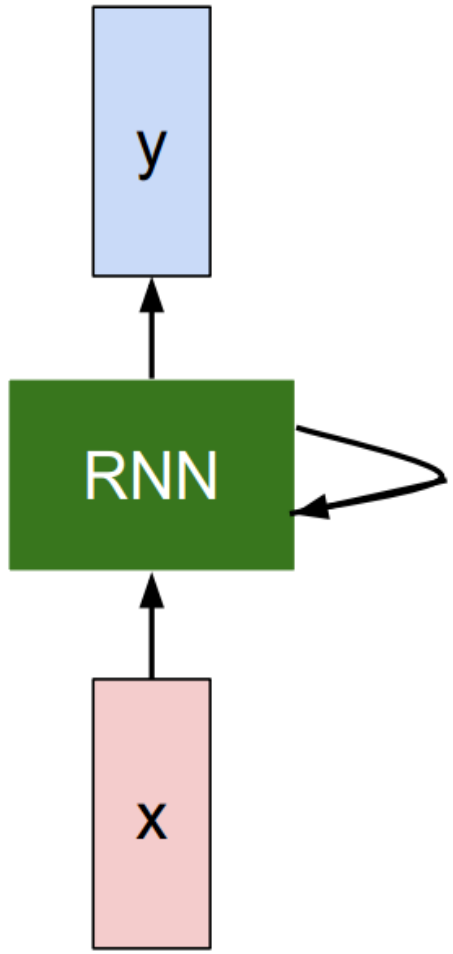
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN output generation

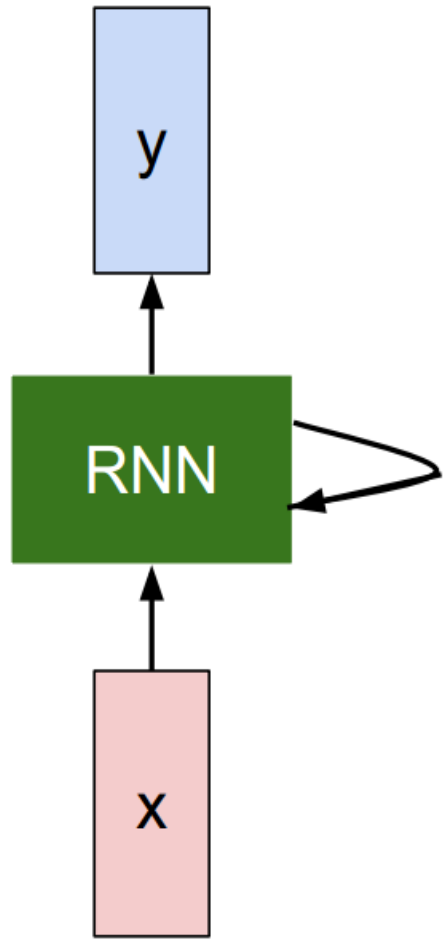
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

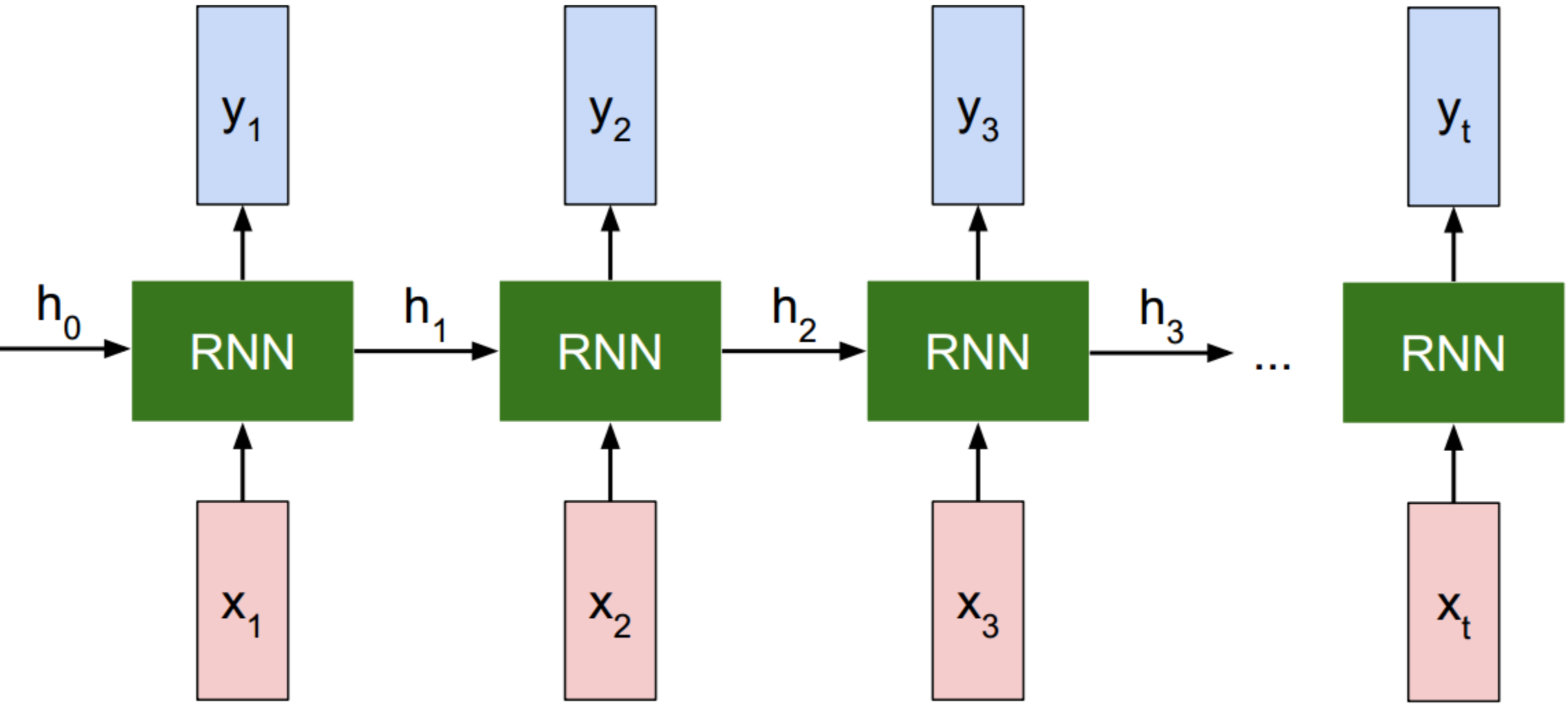
output

new state

another function with parameters W_o

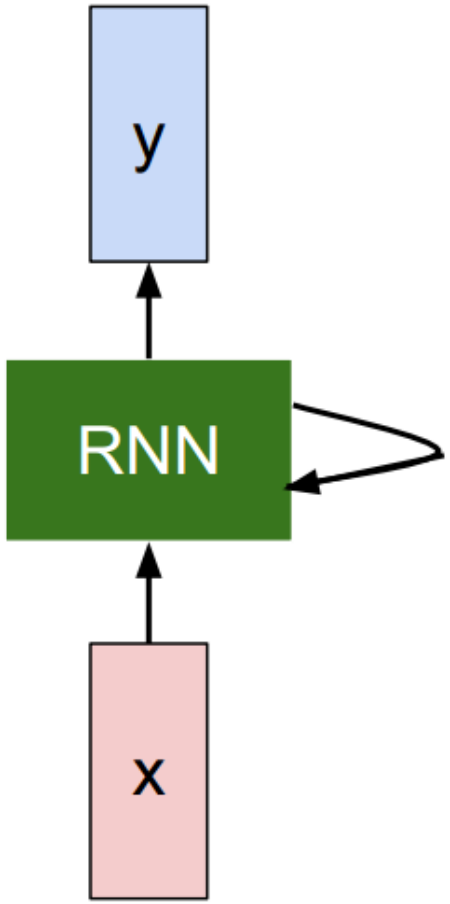


Recurrent Neural Network



(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

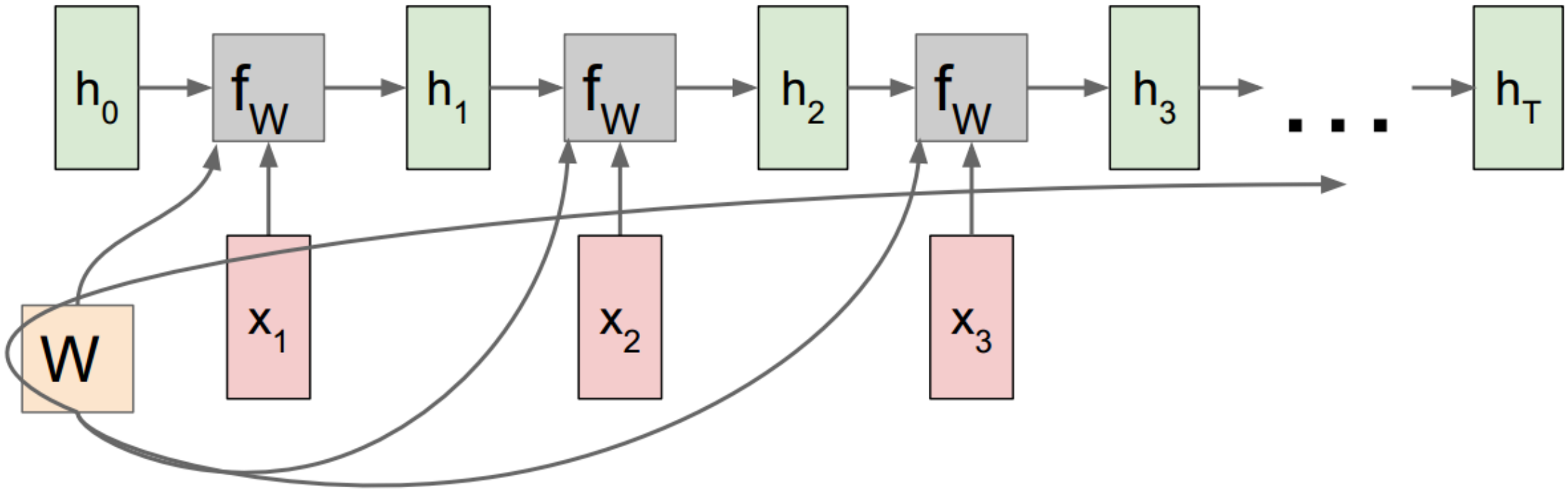
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN: Computational Graph

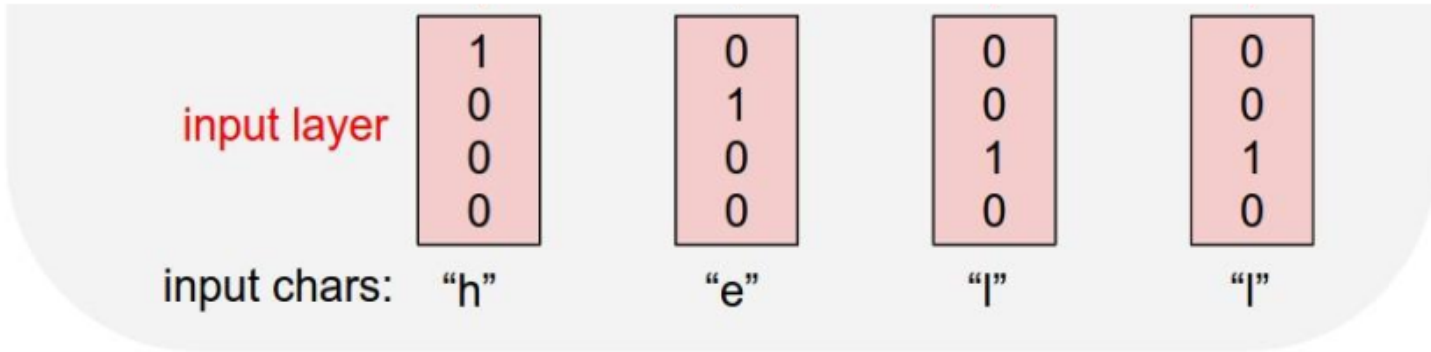
Re-use the same weight matrix at every time-step



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

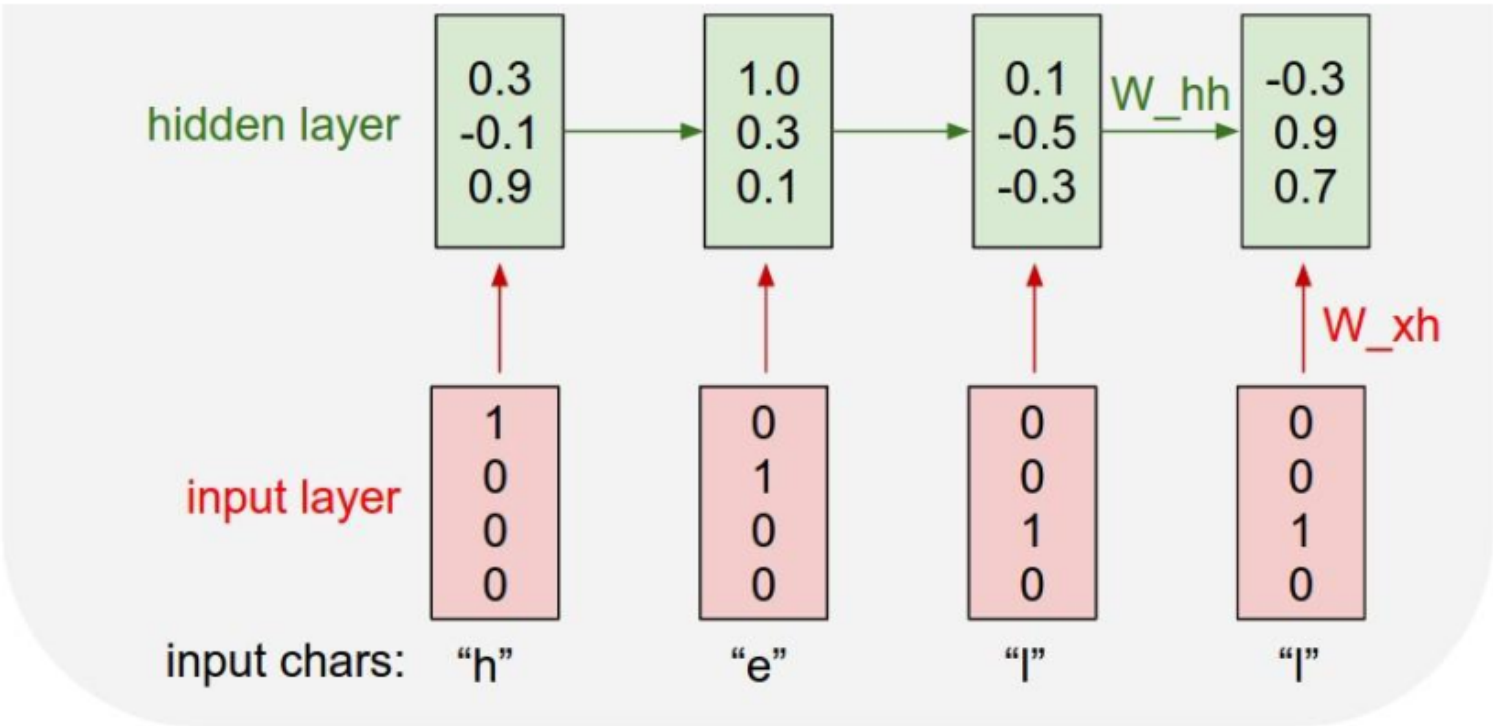


Example: Character-level Language Model

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

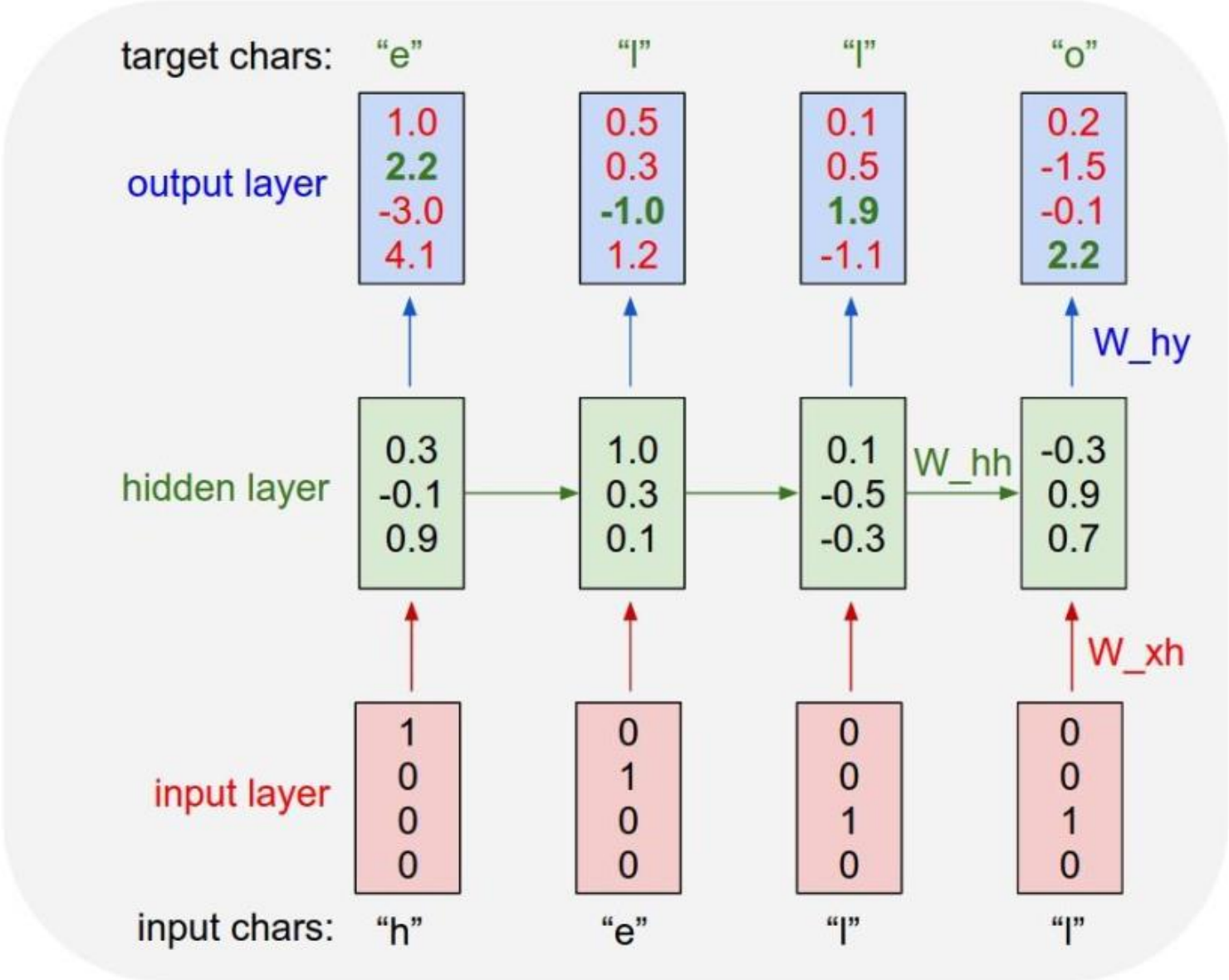
Example training sequence:
“hello”



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

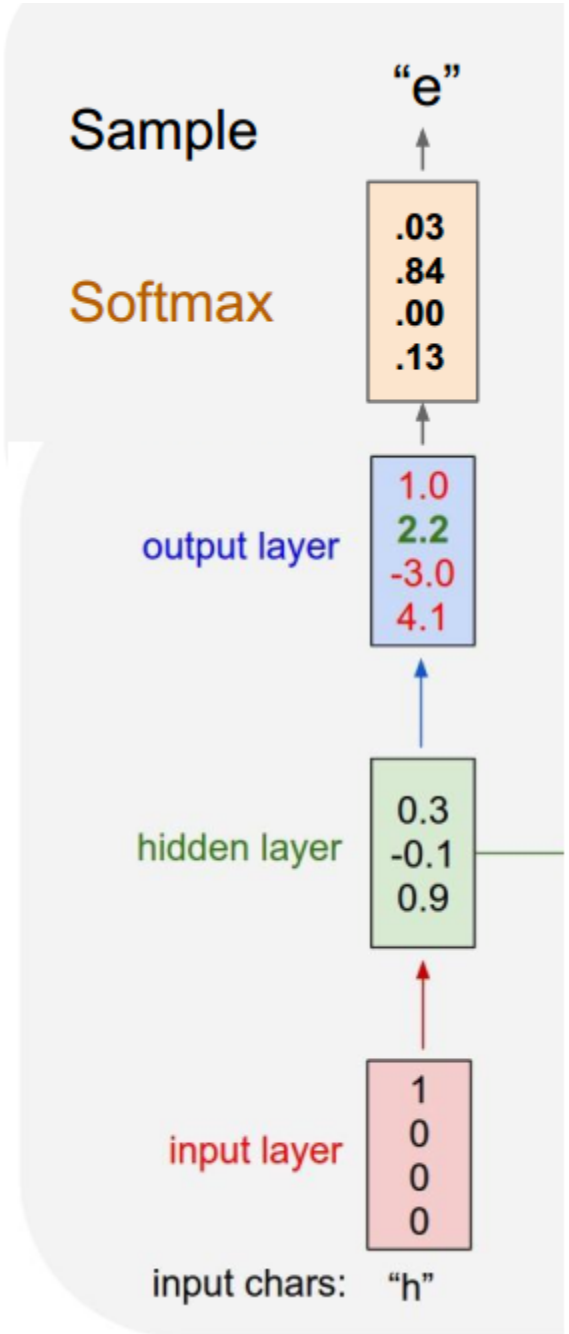
Example training sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

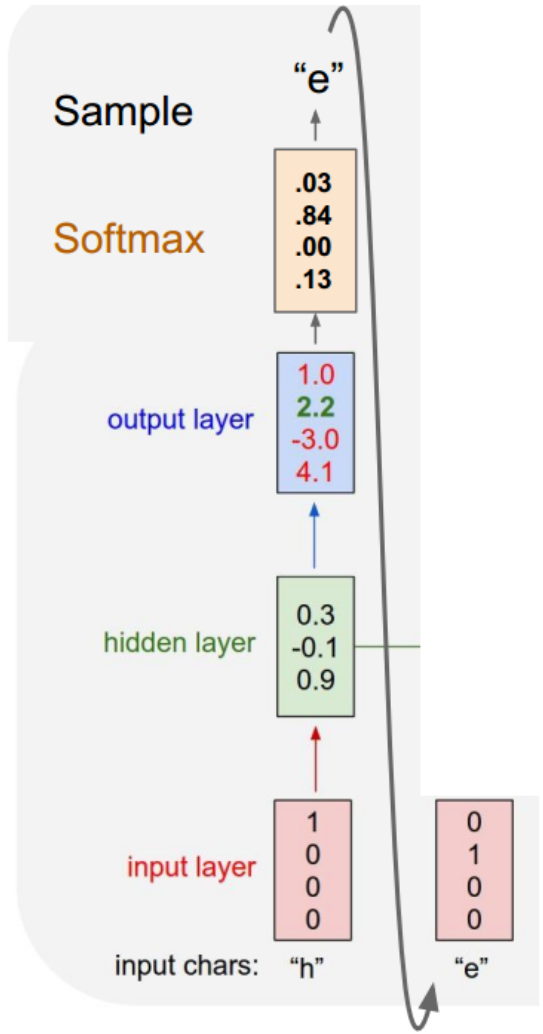
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

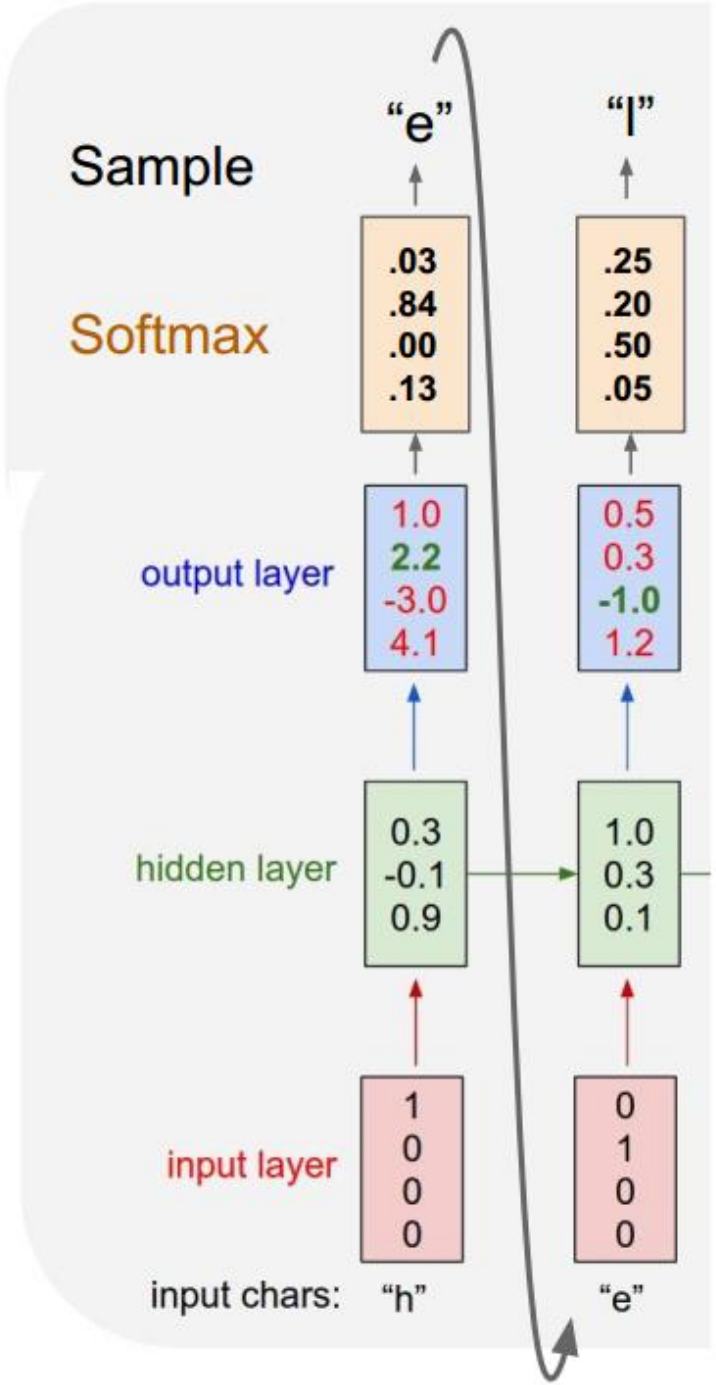
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



Common Problems of Recurrent Neural Networks

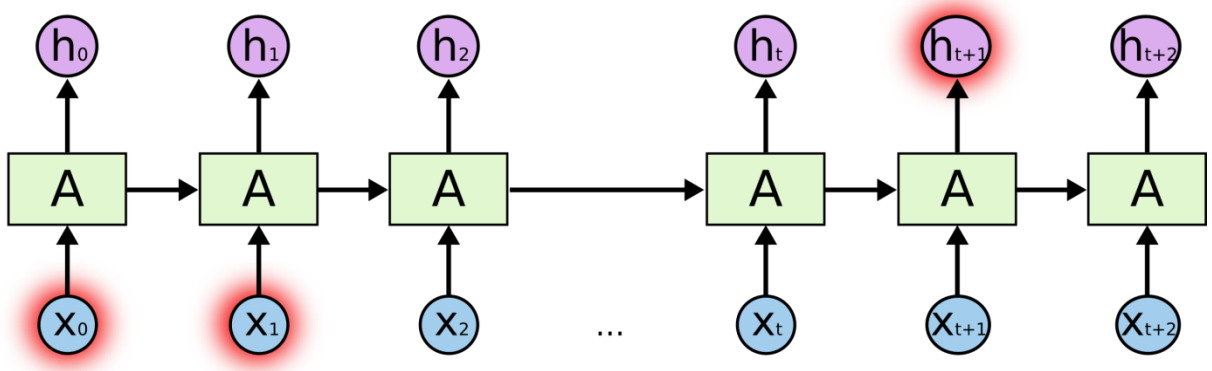
- **Exploding gradients:** This is when the algorithm, without much reason, assigns a stupidly high importance to the weights. Fortunately, this problem can be easily solved by truncating or squashing the gradients.
- **Vanishing gradients:** These occur when the values of a gradient are too small and the model stops learning or takes way too long as a result. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.
- **Complex training process:** Because RNNs process data sequentially, this can result in a tedious training process.
- **Difficulty with long sequences:** The longer the sequence, the harder RNNs must work to remember past information.
- **Inefficient methods:** RNNs process data sequentially, which can be a slow and inefficient approach.

Vanishing/Exploding Gradient Problem

- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large).
- Makes it very difficult train deep networks, or simple recurrent networks over many time steps.

Long Distance Dependencies

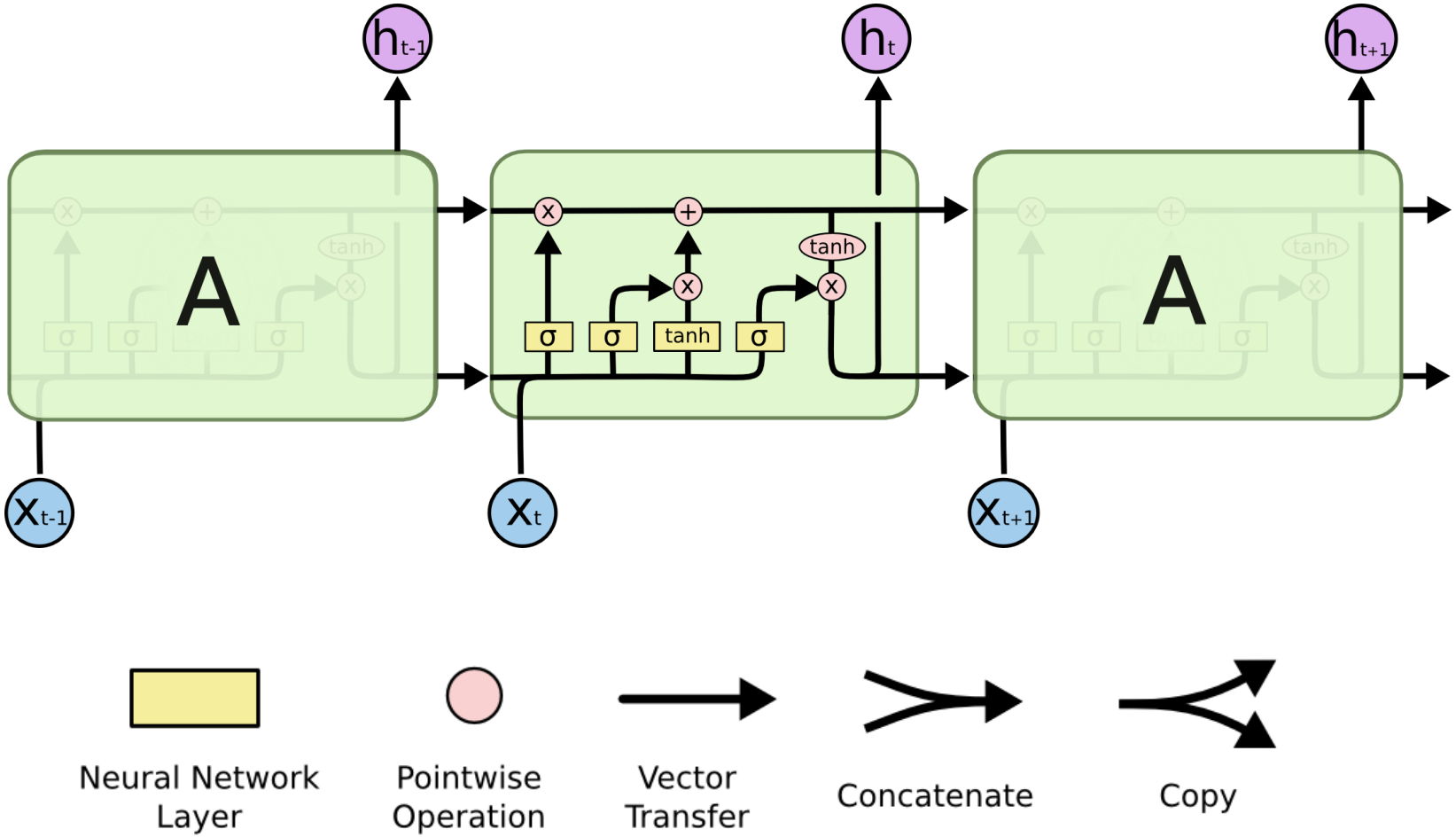
- It is very difficult to train SRNs to retain information over many time steps
- This make is very difficult to learn SRNs that handle long-distance dependencies, such as subject-verb agreement.



Long Short Term Memory

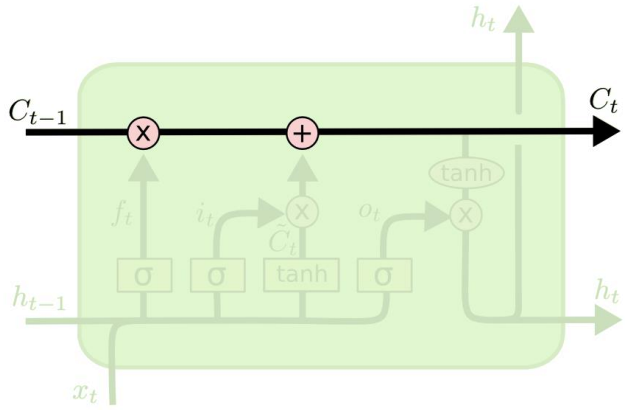
- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

LSTM Network Architecture



Cell State

- Maintains a vector C_t that is the same dimensionality as the hidden state, h_t
- Information can be added or deleted from this state vector via the forget and input gates.

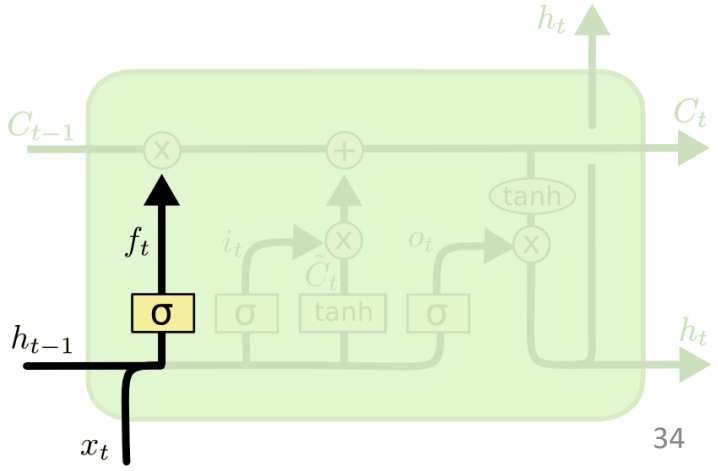


Cell State Example

- Want to remember person & number of a subject noun so that it can be checked to agree with the person & number of verb when it is eventually encountered.
- Forget gate will remove existing information of a prior subject when a new one is encountered.
- Input gate "adds" in the information for the new subject.

Forget Gate

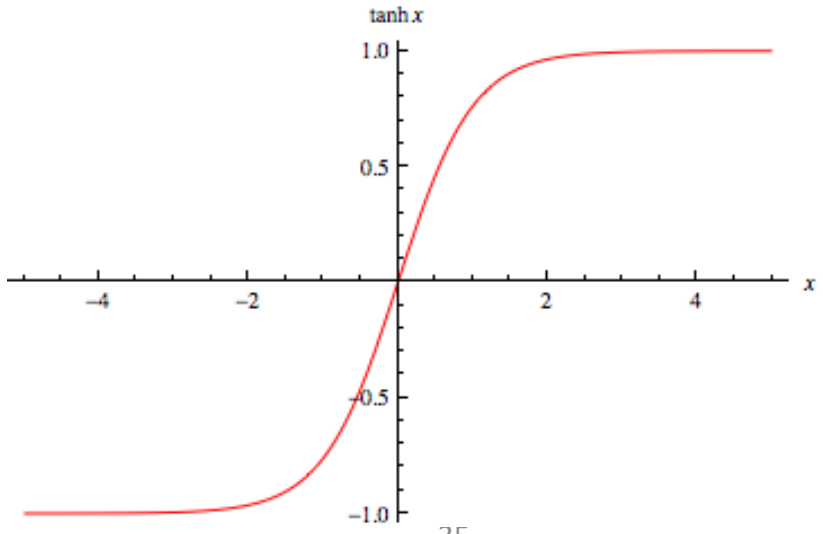
- Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, x_t, and the current hidden state, h_t:
- Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

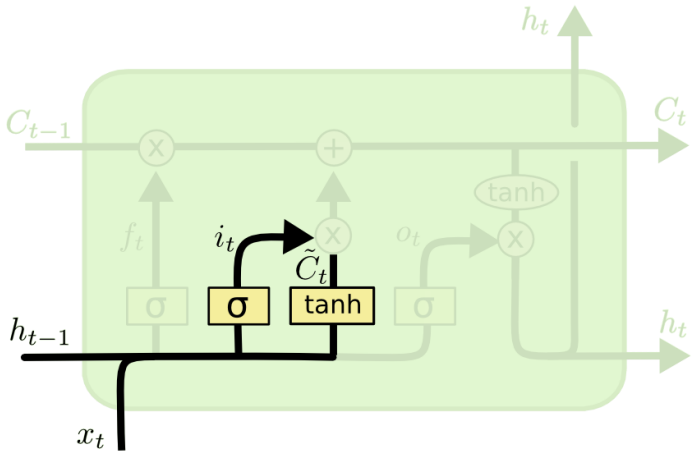
Hyperbolic Tangent Units

- Tanh can be used as an alternative nonlinear function to the sigmoid logistic (0-1) output function.
- Used to produce thresholded output between –1 and 1.



Input Gate

- First, determine which entries in the cell state to update by computing 0-1 sigmoid output.
- Then determine what amount to add/subtract from these entries by computing a tanh output (valued –1 to 1) function of the input and hidden state.

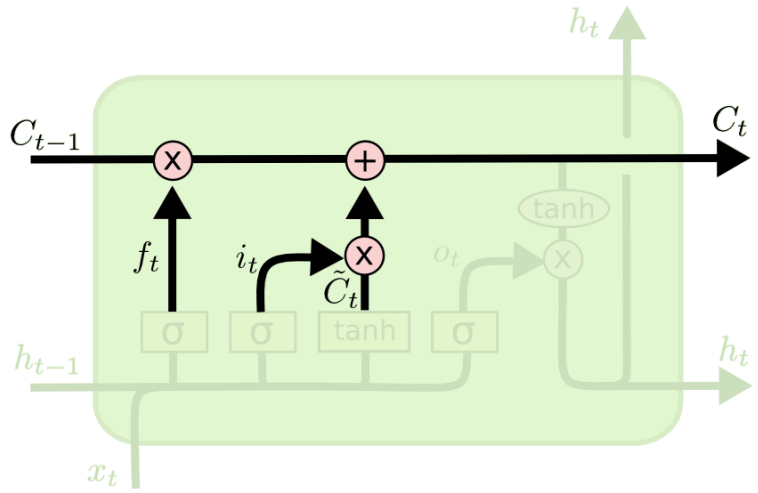


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Updating the Cell State

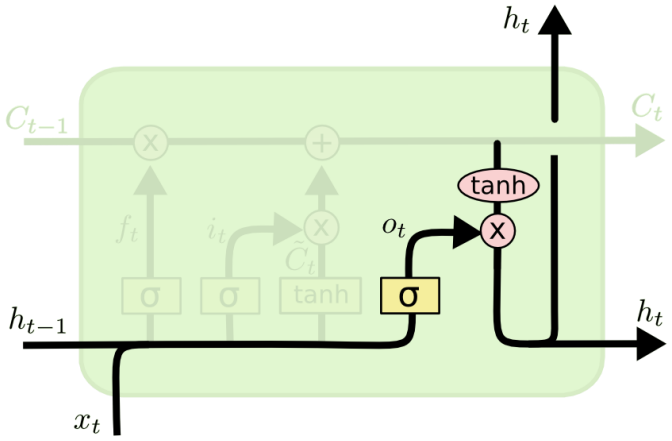
- Cell state is updated by using component-wise vector multiply to "forget" and vector addition to "input" new information.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate

- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and current hidden state to determine which elements of the cell state to "output".

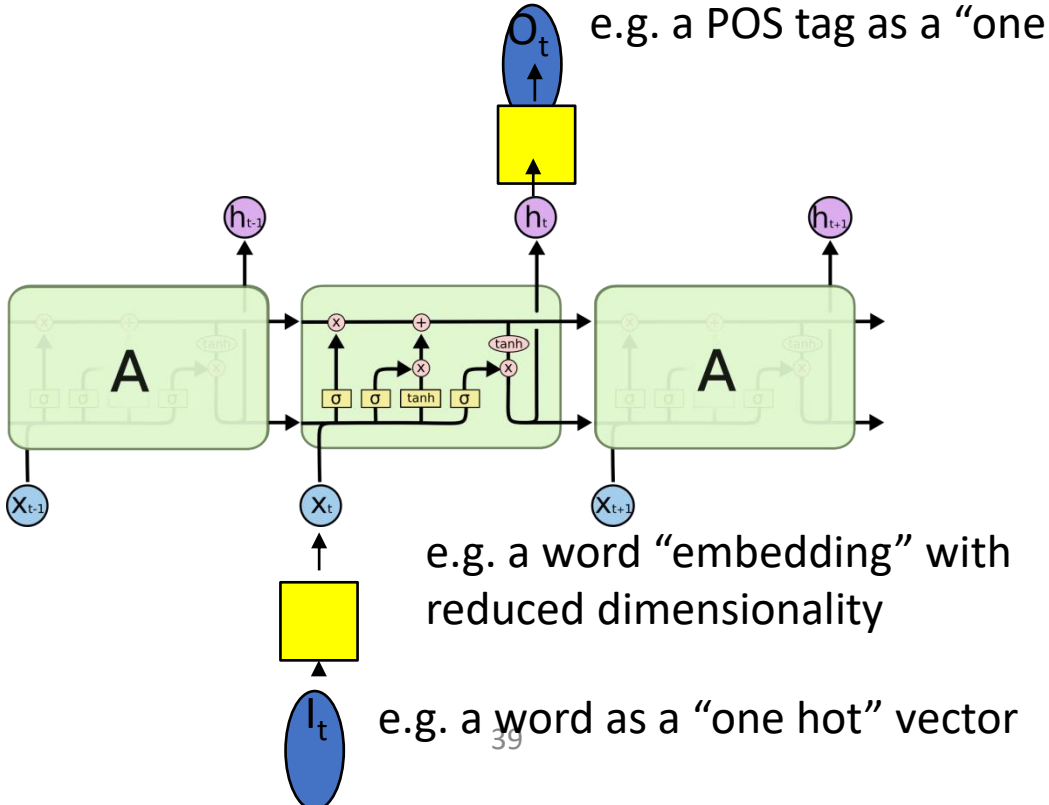


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Overall Network Architecture

- Single or multilayer networks can compute LSTM inputs from problem inputs and problem outputs from LSTM outputs.



LSTM Training

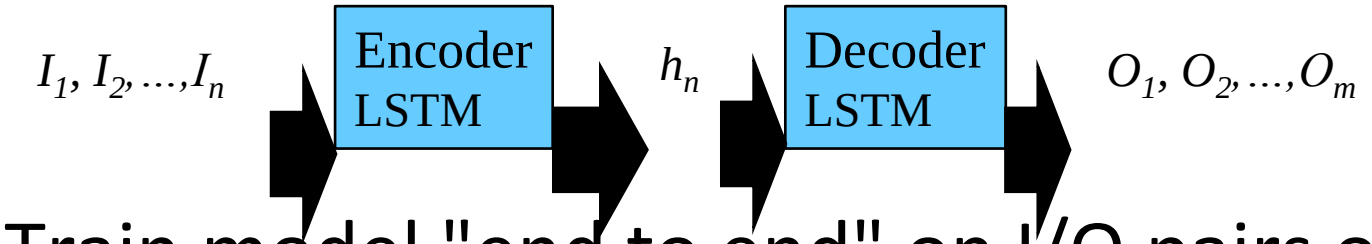
- Trainable with backprop derivatives such as:
  - Stochastic gradient descent (randomize order of examples in each epoch) with momentum (bias weight changes to continue in same direction as last update).
  - ADAM optimizer (Kingma & Ma, 2015)
- Each cell has many parameters (W_f, W_i, W_C, W_o)
  - Generally requires lots of training data.
  - Requires lots of compute time that exploits GPU clusters.

General Problems Solved with LSTMs

- Sequence labeling
  - Train with supervised output at each time step computed using a single or multilayer network that maps the hidden state (h_t) to an output vector (O_t).
- Language modeling
  - Train to predict next input ($O_t = I_{t+1}$)
- Sequence (e.g. text) classification
  - Train a single or multilayer network that maps the final hidden state (h_n) to an output vector (O).

Sequence to Sequence Transduction (Mapping)

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.



- Train model "end to end" on I/O pairs of sequences.

Summary of LSTM Application Architectures

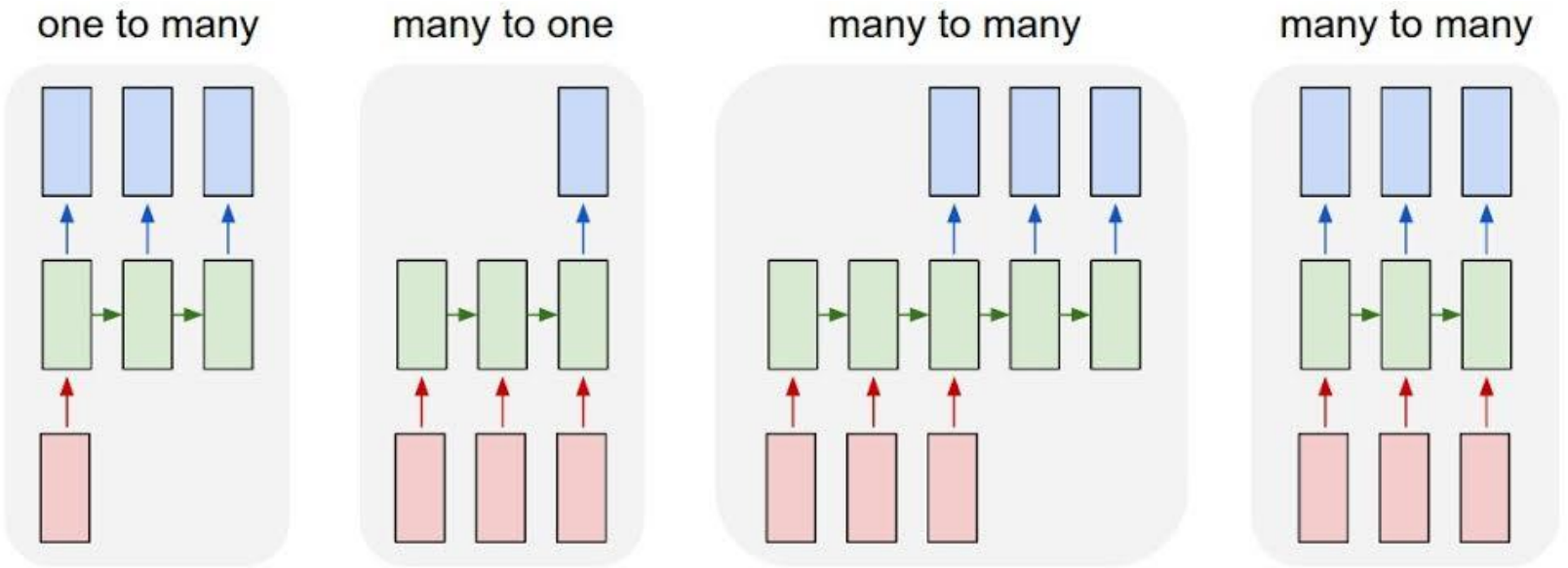


Image Captioning

Video Activity Recog
Text Classification

Video Captioning
Machine Translation

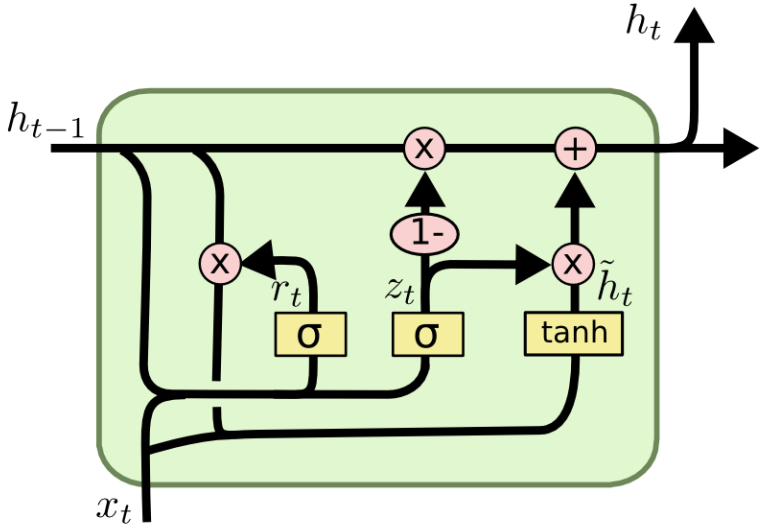
POS Tagging
Language Modeling

Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
  - POS Tagging
    https://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)
  - NER
  - Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning: CNN output vector to sequence
- Sequence to Sequence
  - Machine Translation (Sustkever, Vinyals, & Le, 2014)
  - Video Captioning (input sequence of CNN frame outputs)

Gated Recurrent Unit (GRU)

- Alternative RNN to LSTM that uses fewer gates (Cho, et al., 2014)
  - Combines forget and input gates into “update” gate.
  - Eliminates cell state vector



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU vs. LSTM

- GRU has significantly fewer parameters and trains faster.
- Experimental results comparing the two are still inconclusive, many problems they perform the same, but each has problems on which they work better.

Attention

- For many applications, it helps to add “attention” to RNNs.
- Allows network to learn to attend to different parts of the input at different time steps, shifting its attention to focus on different aspects during its processing.
- Used in image captioning to focus on different parts of an image when generating different parts of the output sentence.
- In MT, allows focusing attention on different parts of the source sentence when generating different parts of the translation.

Attention for Image Captioning

(Xu, et al. 2015)

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)

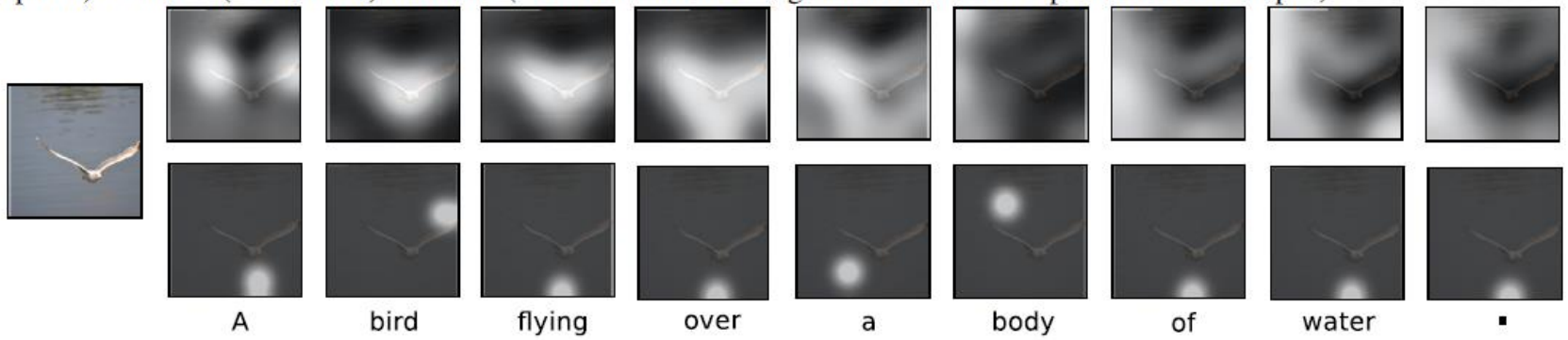


Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Conclusions

- By adding “gates” to an RNN, we can prevent the vanishing/exploding gradient problem.
- Trained LSTMs/GRUs can retain state information longer and handle long-distance dependencies.
- Recent impressive results on a range of challenging NLP problems.