Lecture 22: Recurrent Neural Networks

COMP 411, Fall 2021
Victoria Manfredi

WESLEYAN
UNIVERSITY

These slides are based on figures and info from Andrej Karpathy's blog and slides created by Justin Johnson (U of Michigan), and Geoffrey Hinton (U of Toronto)

Today's Topics

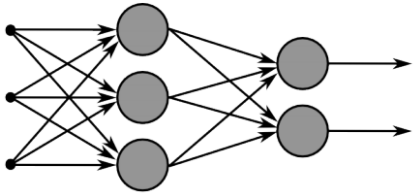
Recurrent Neural Networks

- Overview
- Computation graph
- Language modeling example

Recurrent Neural Networks

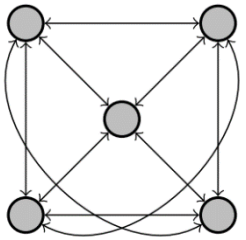
OVERVIEW

Recurrent Neural Networks



Multi-layer feed-forward NN: **DAG**

- Just computes a fixed sequence of non-linear learned transformations to convert an input pattern into an output pattern

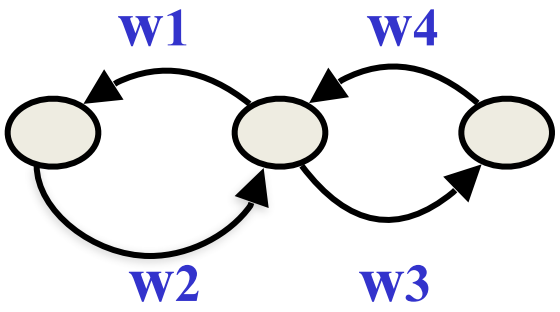


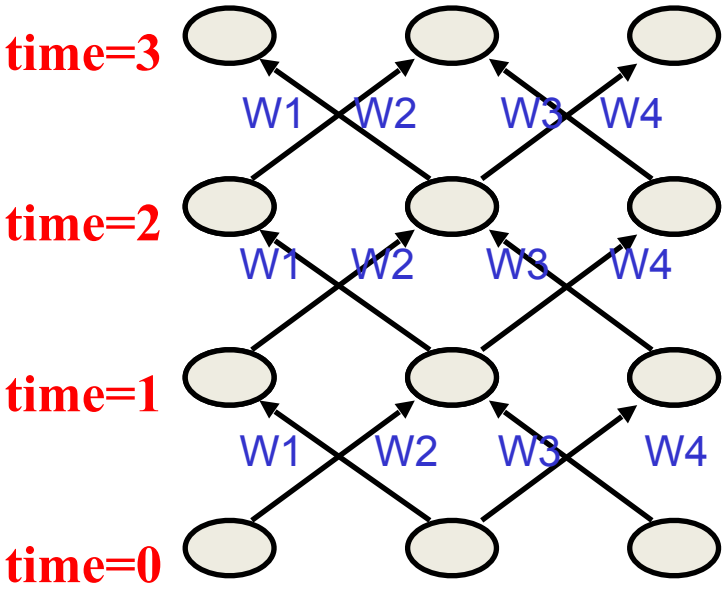
Recurrent Neural Network: **Digraph**

- Has cycles
- Cycle can act as a memory
- The hidden state of a recurrent net can carry along information about a "potentially" unbounded number of previous inputs
- They can model sequential data in a much more natural way

Equivalence between RNN and Feed-forward NN

Assume that there is a time delay of 1 in using each connection. Then the recurrent net is just a layered net that keeps reusing the same weights.





Getting targets when modeling sequences

When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.

- E. g. turn a sequence of sound pressures into a sequence of word identities.

When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.

- The target output sequence is the input sequence with an advance of 1 step.
- This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
- For temporal sequences there is a natural order for the predictions.

Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.

- It uses methods designed for supervised learning, but it doesn't require a separate teaching signal.

Recurrent neural networks

RNNs are very powerful, because they combine two properties:

- Distributed hidden state that allows them to store a lot of information about the past efficiently
- Non-linear dynamics that allows them to update their hidden state in complicated ways

With enough neurons and time, RNNs can compute anything that can be computed by your computer.

What the network learns

A recurrent network can emulate a finite state automaton, but it is exponentially more powerful.

With N hidden neurons it has 2^N possible binary activity vectors (but only N^2 weights)

- This is important when the input stream has two separate things going on at once.
- A finite state automaton needs to square its number of states.
- An RNN needs to double its number of units.

Recurrent neural networks

What kinds of behaviour can RNNs exhibit?

- They can oscillate. Good for motor control?
- They can settle to point attractors. Good for retrieving memories?
- They can behave chaotically. Bad for information processing?
- RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.

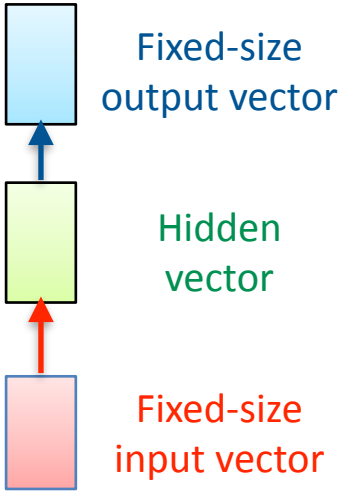
But the computational power of RNNs makes them very hard to train.

- For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer).

Recurrent Neural Networks

COMPUTATION GRAPH

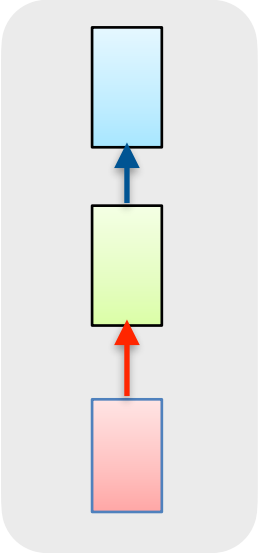
So far: “Feed-forward” Neural Networks



Arrows represent functions
like matrix multiply

So far: “Feed-forward” Neural Networks

one to one

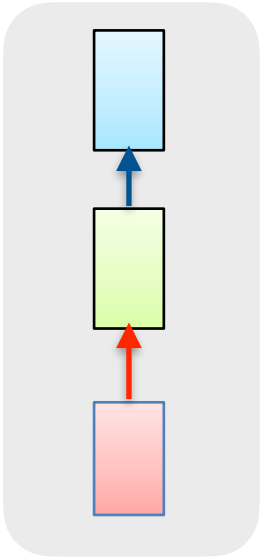


E.g., Image classification

Image → Label

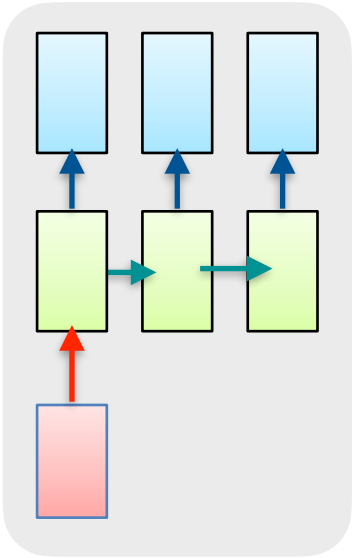
Recurrent Neural Networks: Process Sequences

one to one



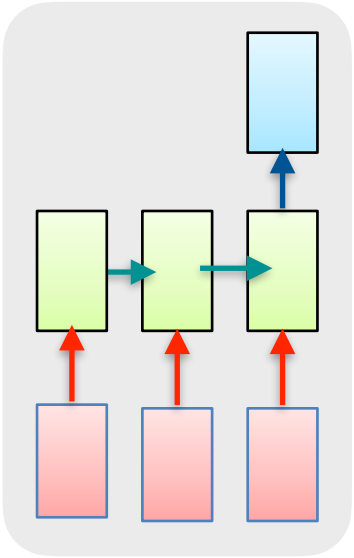
E.g., Image classification
Image → Label

one to many



E.g., Image captioning
Image → Sequence of words

many to one

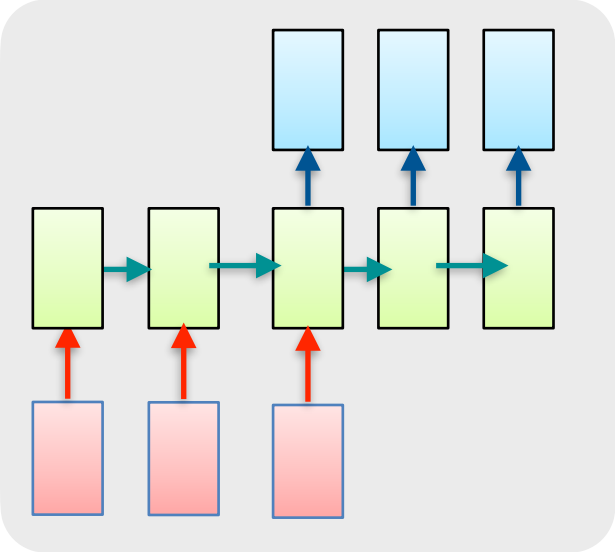


E.g., Video classification
Sequence of images → label

No pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like

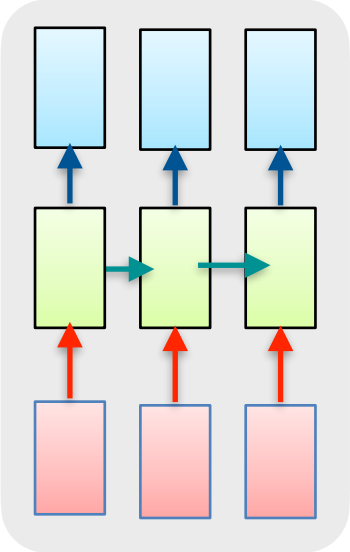
Recurrent Neural Networks: Process Sequences

many to many



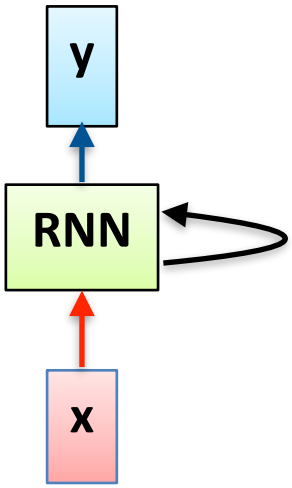
E.g., Machine Translation
Sequence of words → Sequence of words

many to many



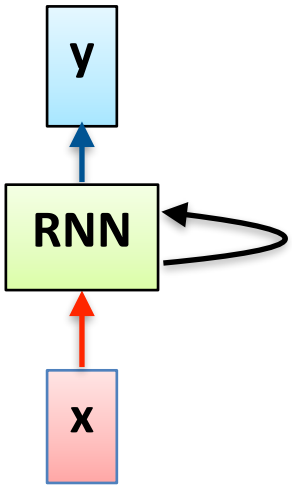
E.g., Per-frame video classification
Sequence of images → sequence of labels

Recurrent Neural Networks



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

Recurrent Neural Networks



We can process a sequence of vectors $\mathbf{x}$ by applying a **recurrence formula** at every timestep

$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

New state

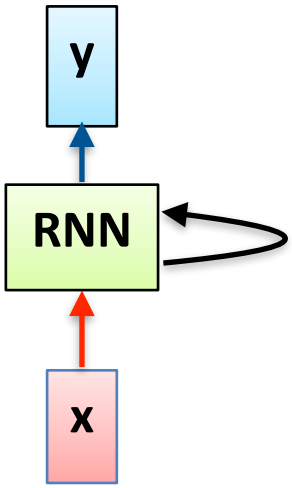
Some nonlinear, differentiable function with parameters W

Old state

Input vector at some timestep

Notice the same function and the se set of parameters are used at every timestep

(Vanilla) Recurrent Neural Networks



The state consists of a single "hidden" vector $\mathbf{h}$

$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

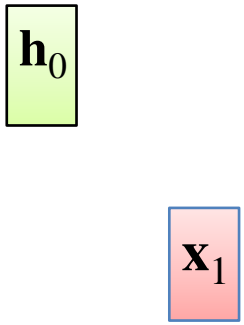
$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t$$

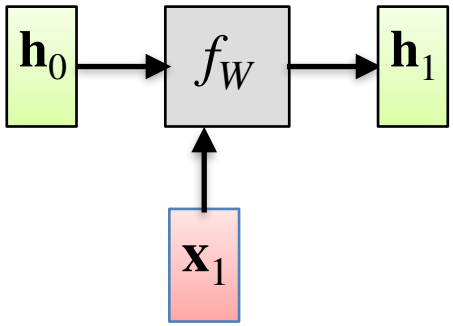
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN computational graph

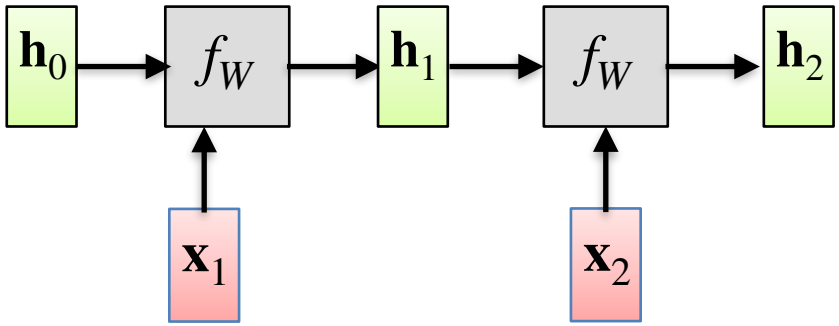
Initial hidden state: either set to all 0, or learn it



RNN computational graph

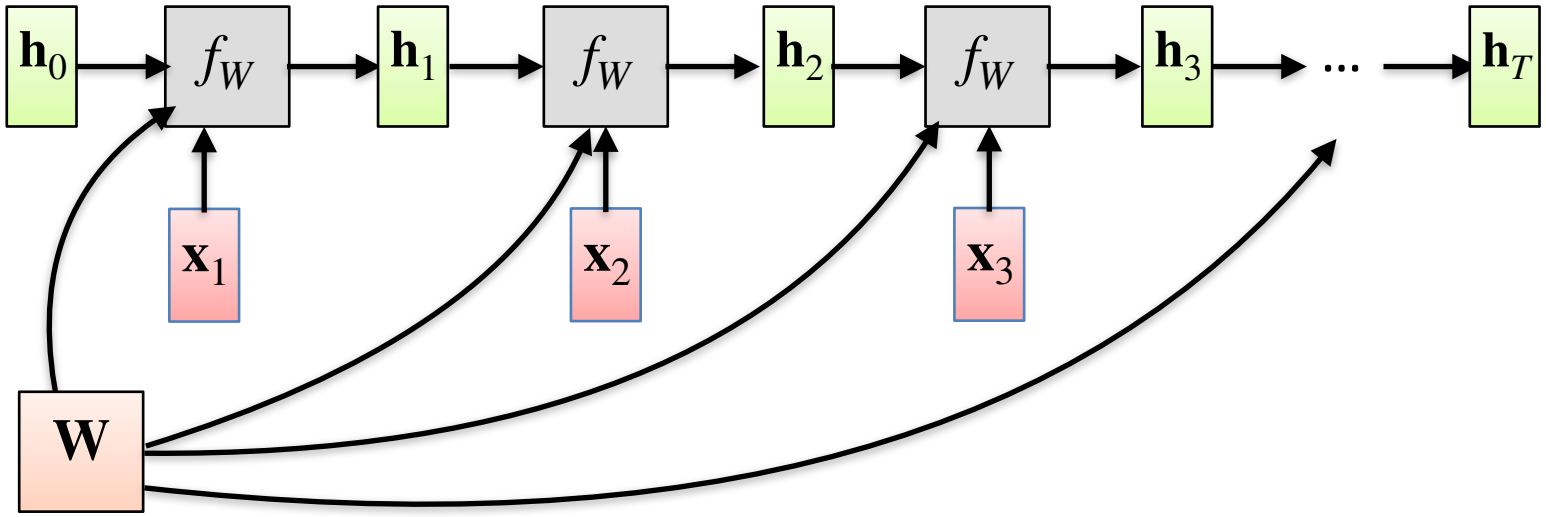


RNN computational graph



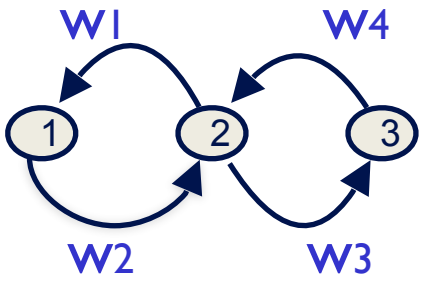
RNN computational graph

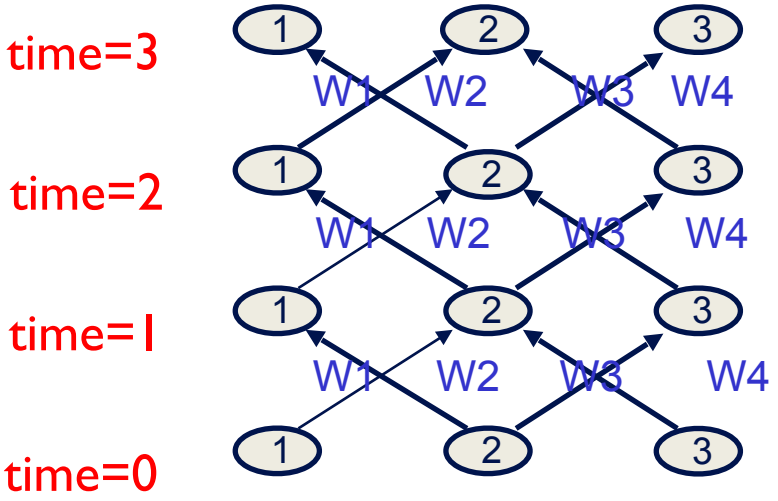
Re-use the same weight matrix at every timestep



Equivalence between RNN and Feed-forward NN

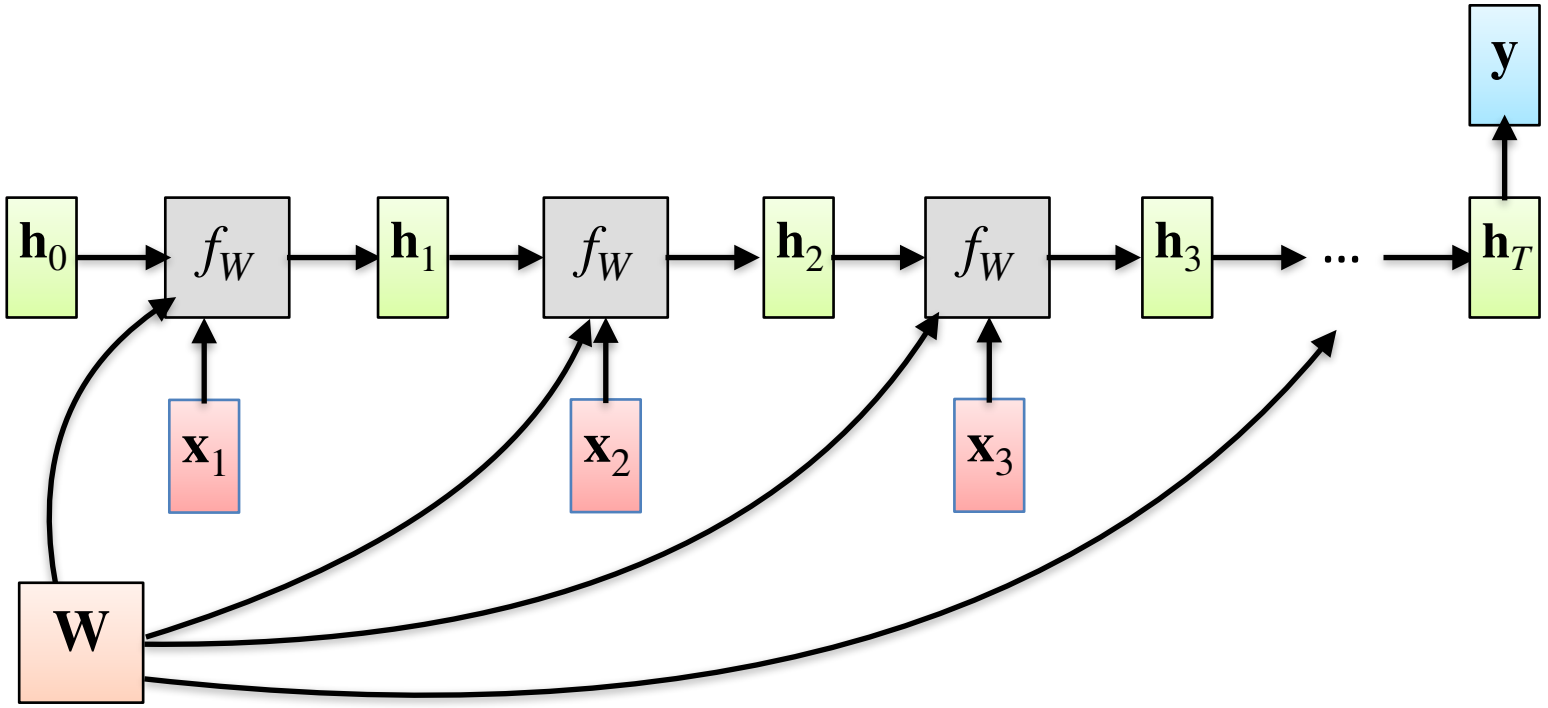
- Assume that there is a time delay of 1 in using each connection.
- The recurrent net is just a layered net that keeps reusing the same weights.



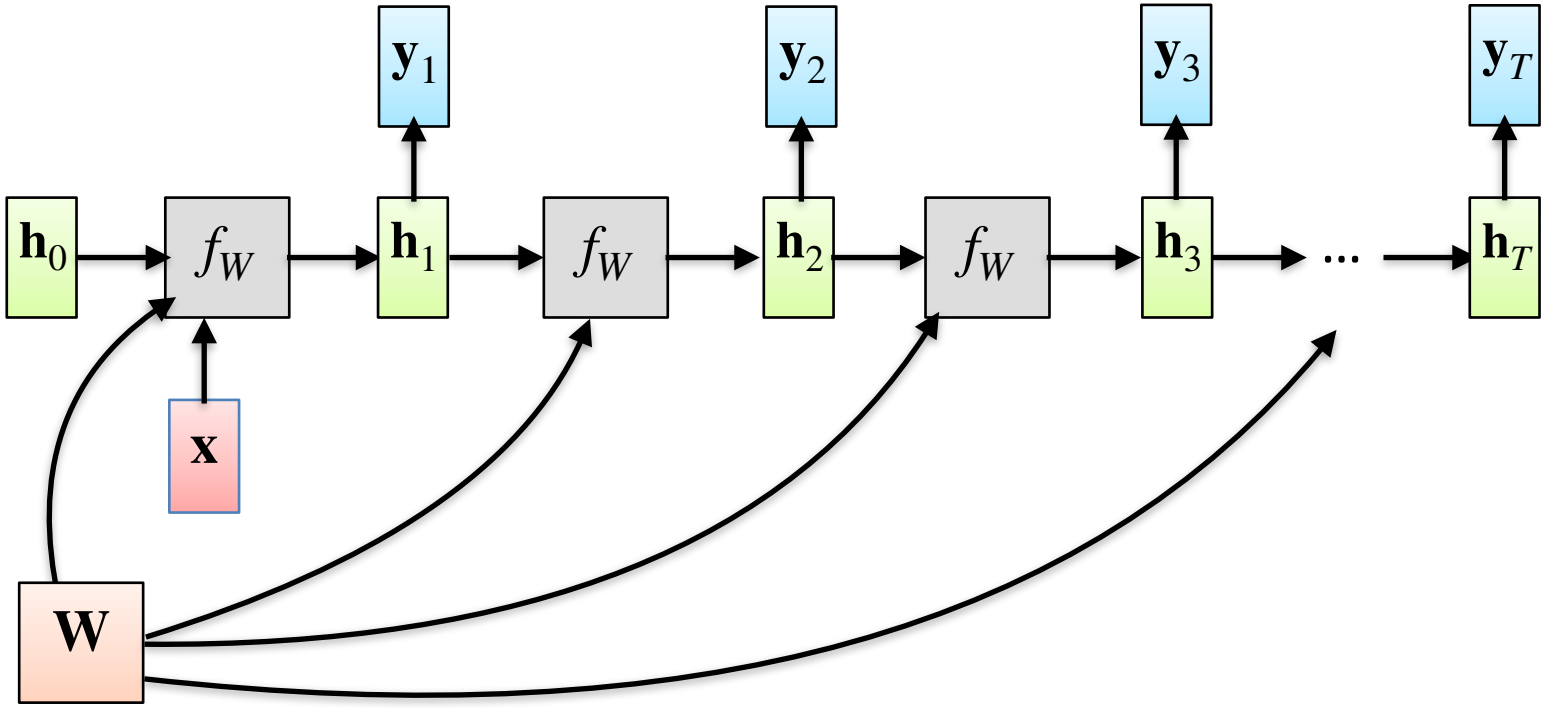


Slide Credit: Geoff Hinton

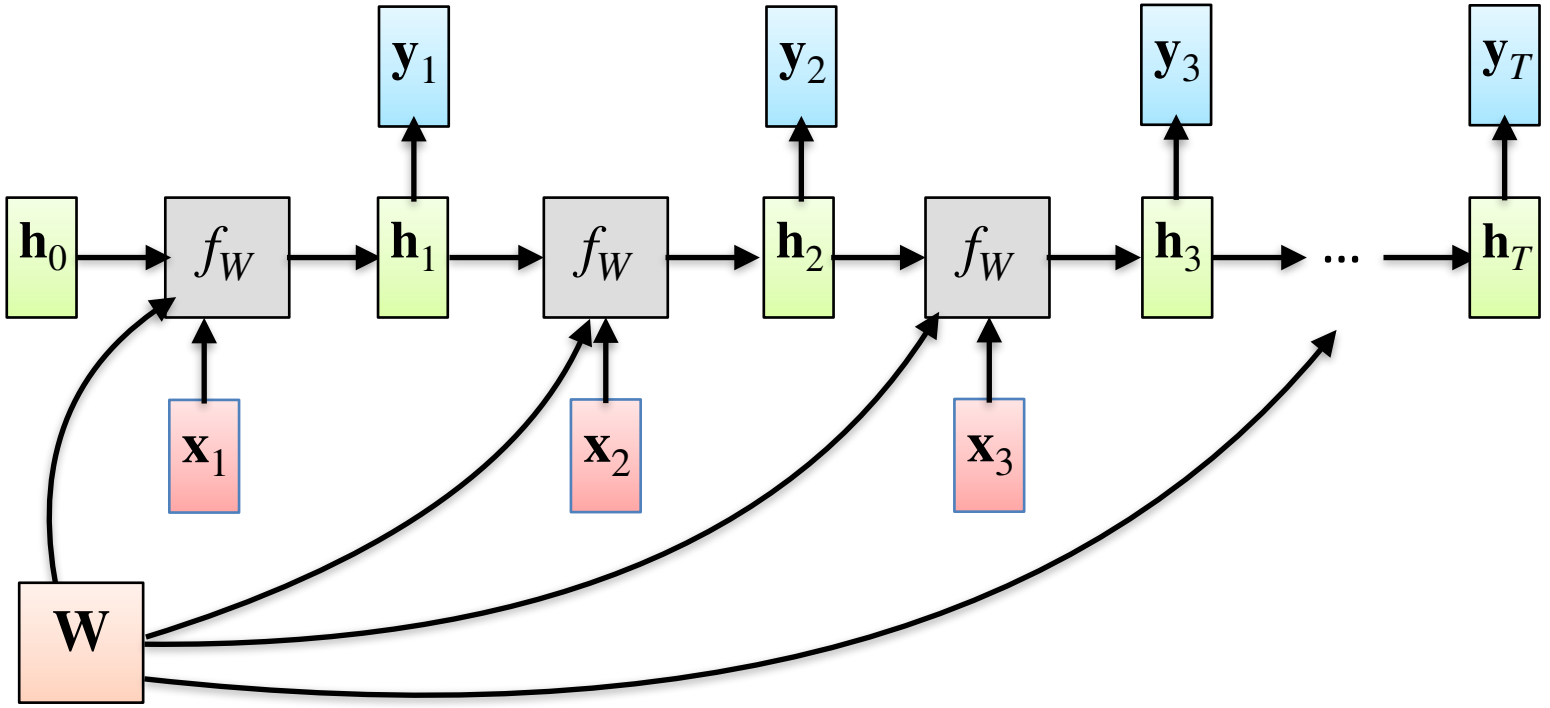
RNN computational graph: many to one



RNN computational graph: one to many



RNN computational graph: many to many



Recurrent Neural Networks

LANGUAGE MODELING EXAMPLE

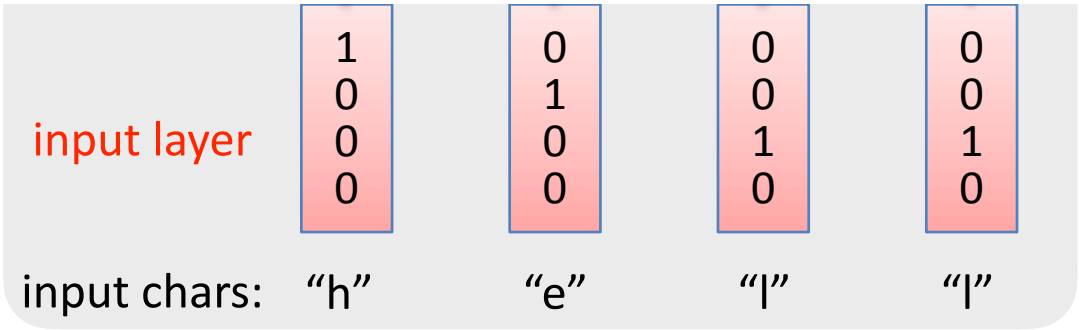
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



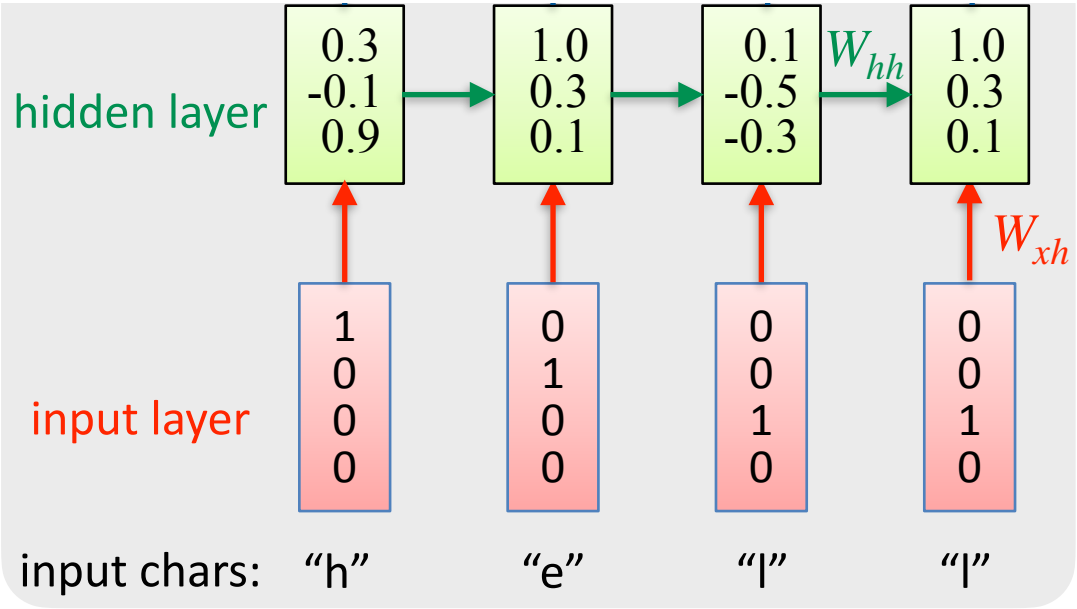
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”

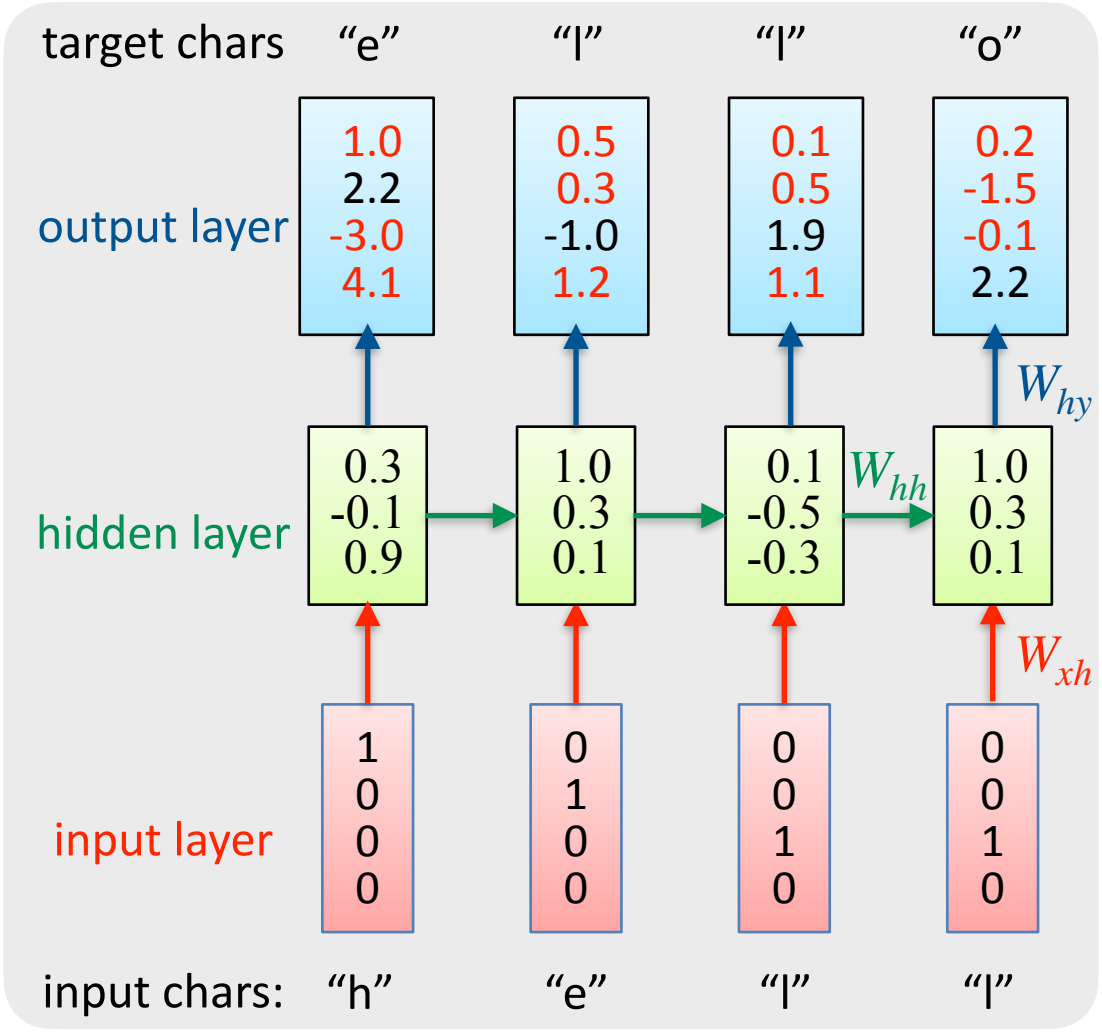
Vocabulary: [h, e, l, o]



Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$



Training sequence: “hello”

Vocabulary: [h, e, l, o]

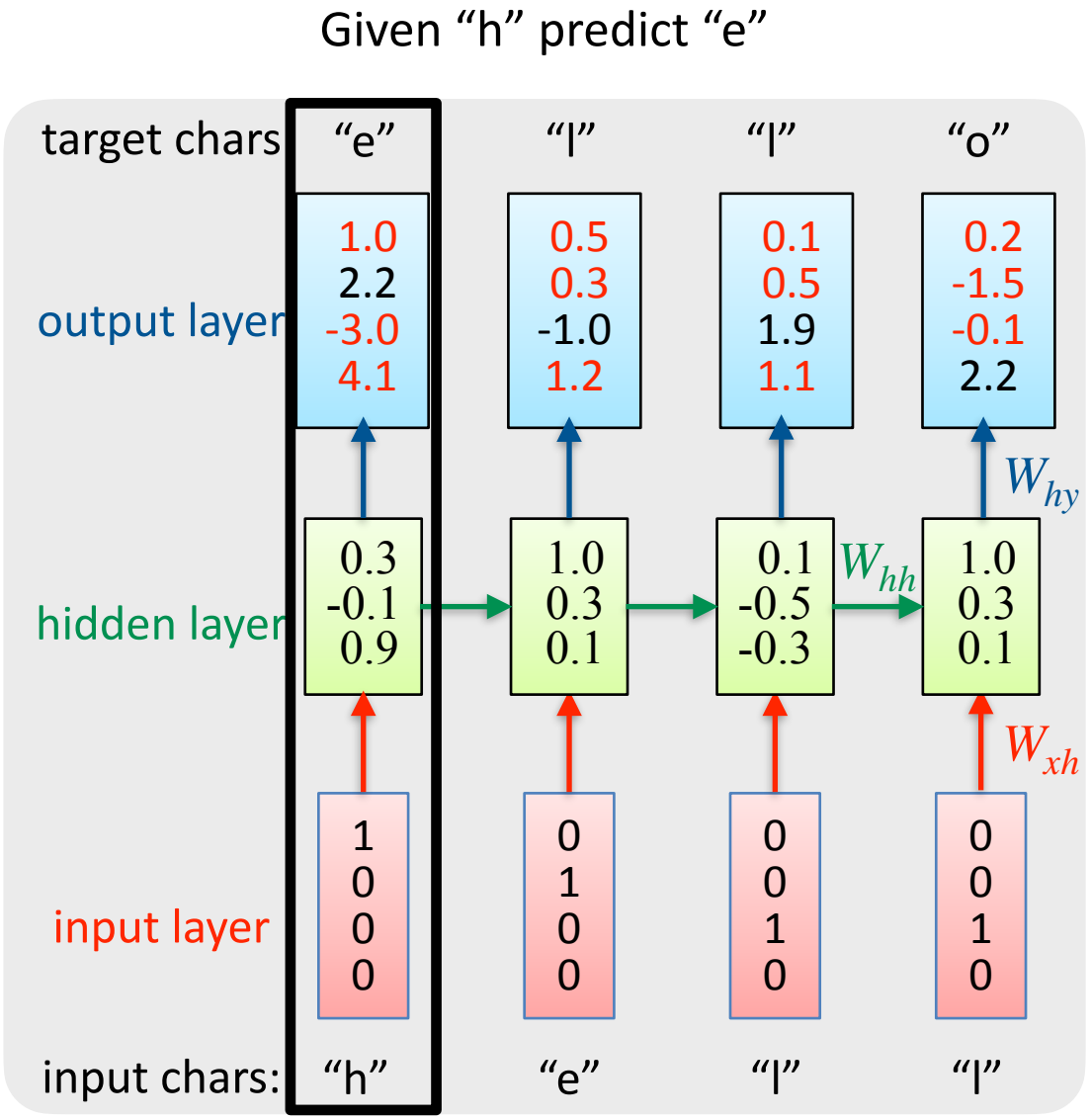
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



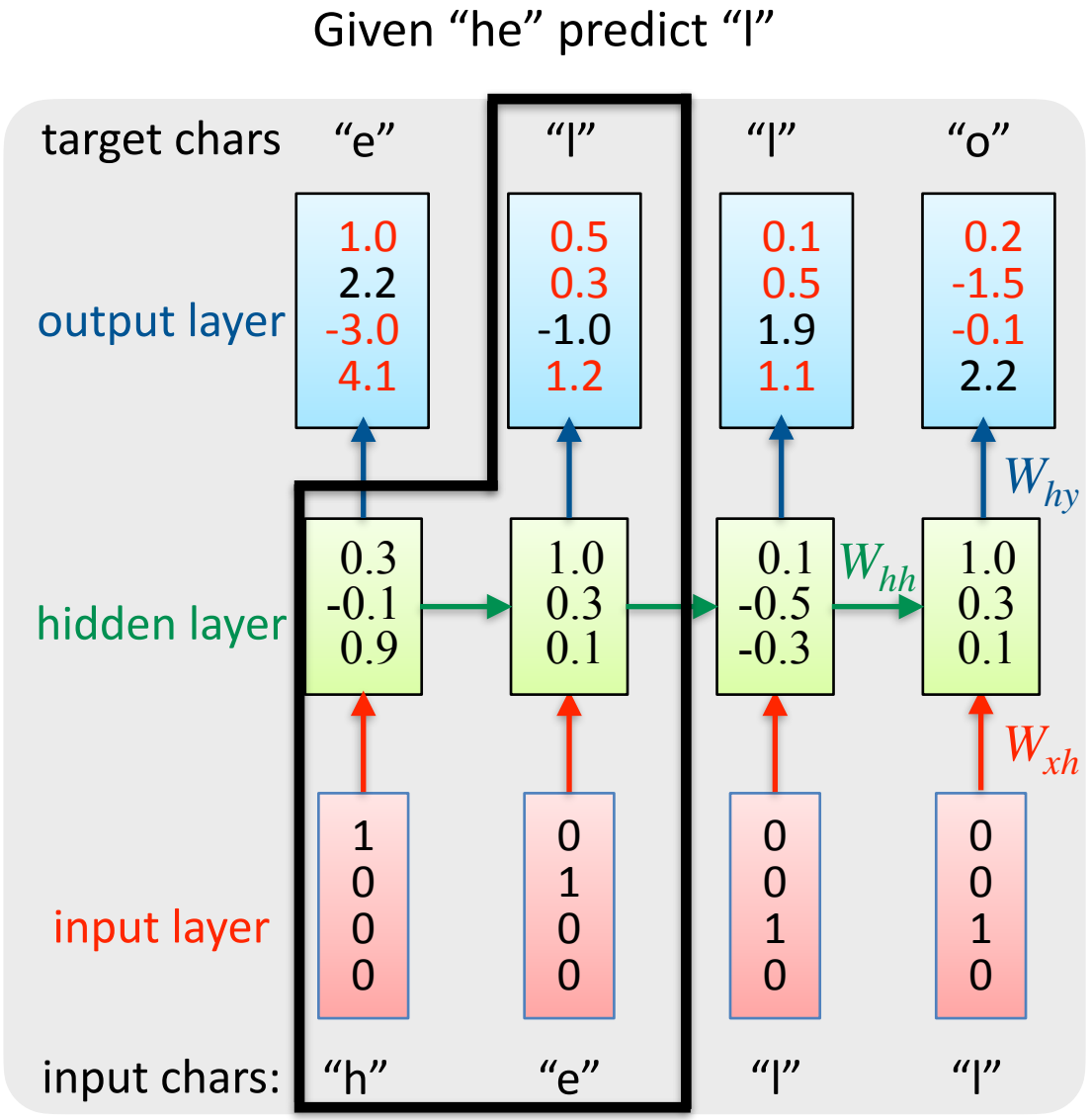
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



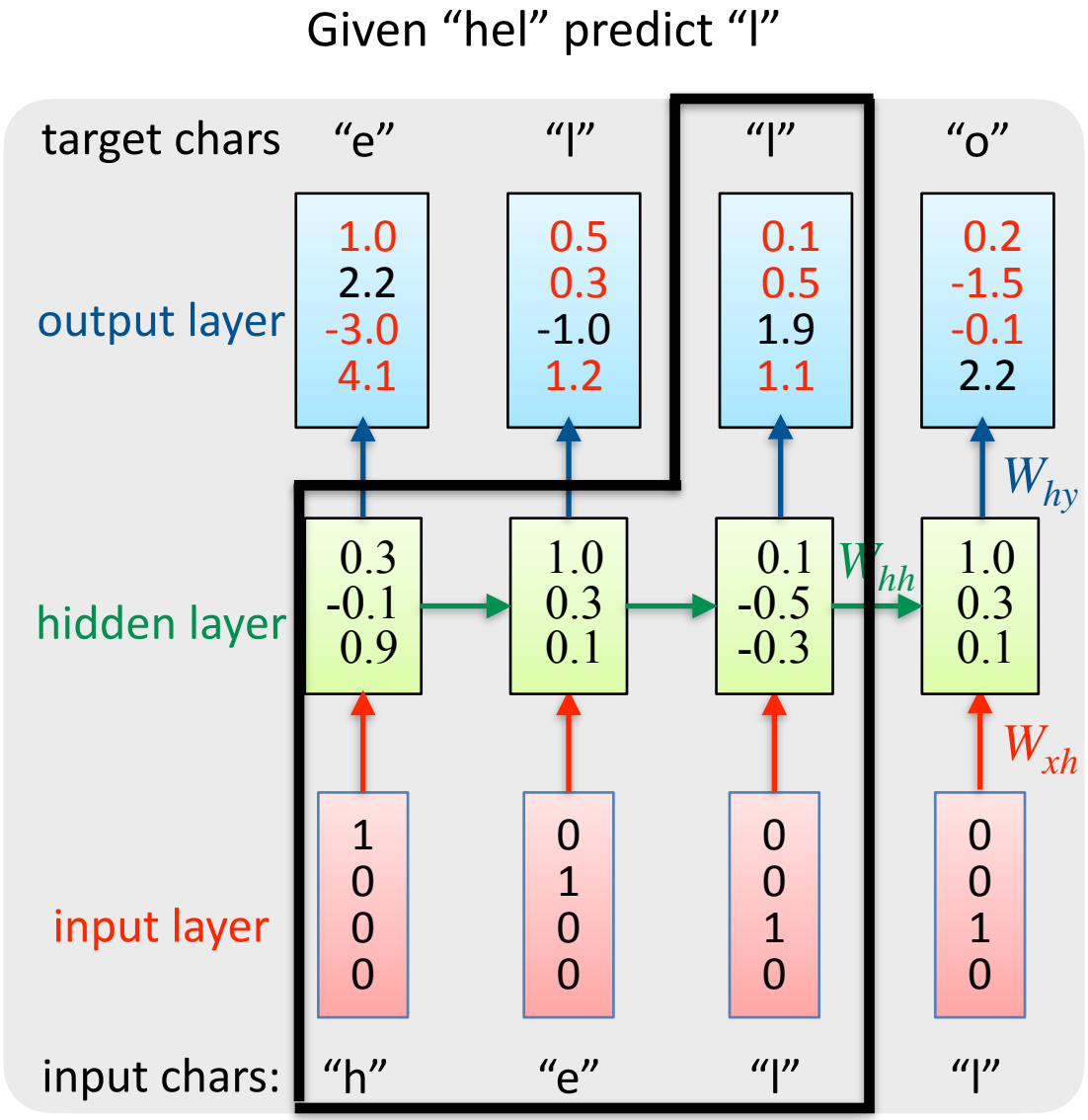
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



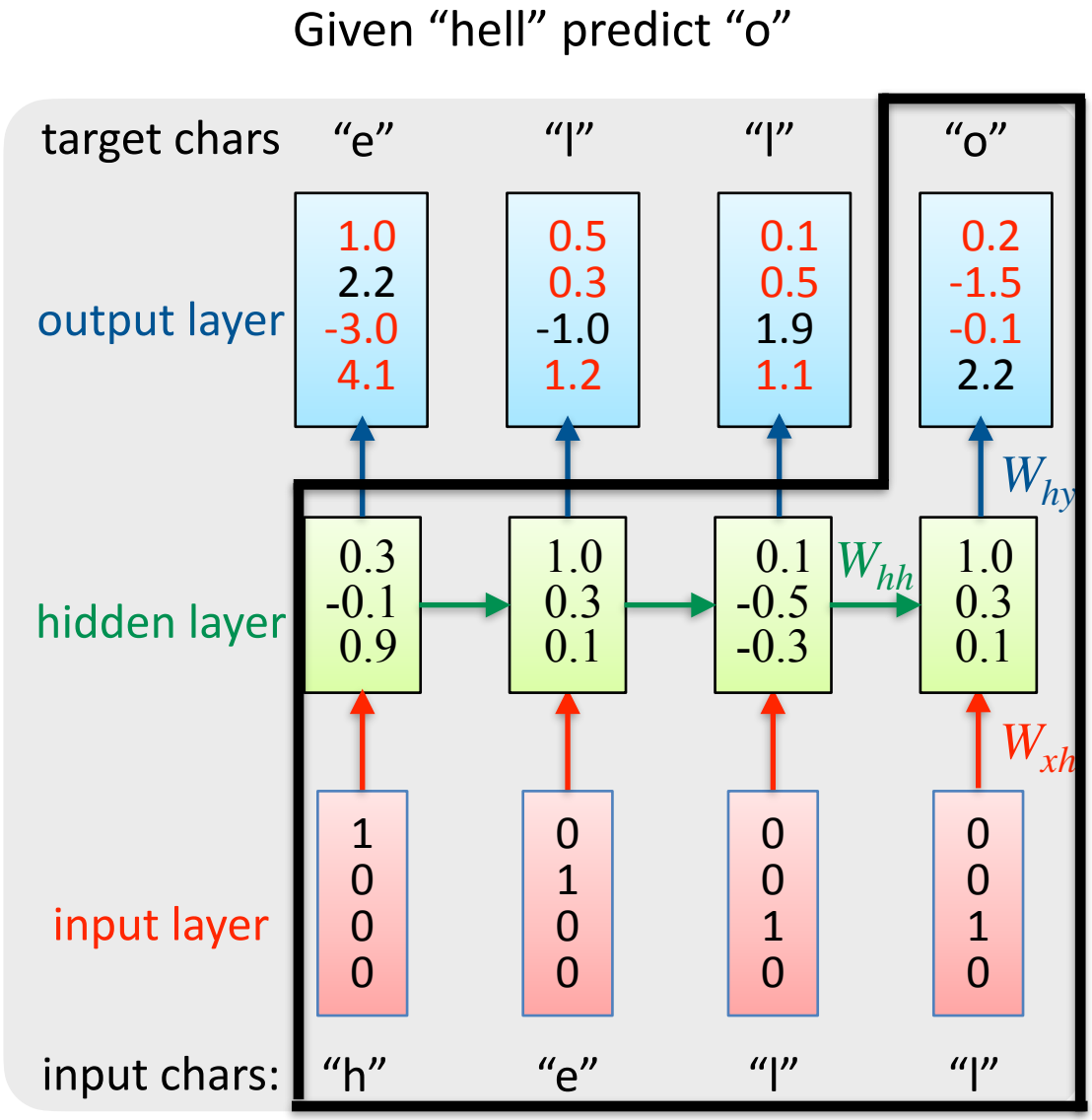
Example: language modeling

Given characters 1, 2, …, t,
model predicts character t

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

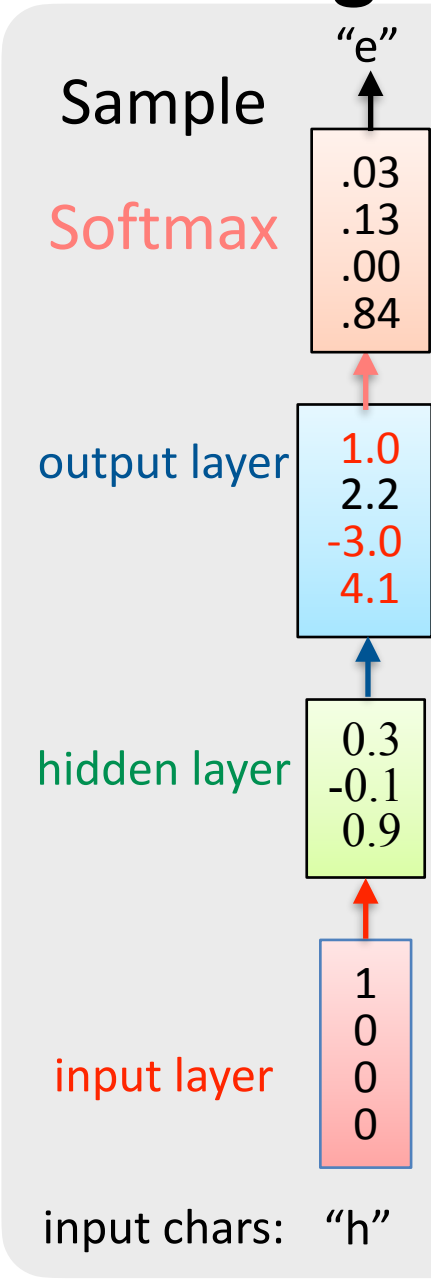
Training sequence: “hello”

Vocabulary: [h, e, l, o]



Example: language modeling

At test time, generate new text: sample characters one at a time, feed back to model

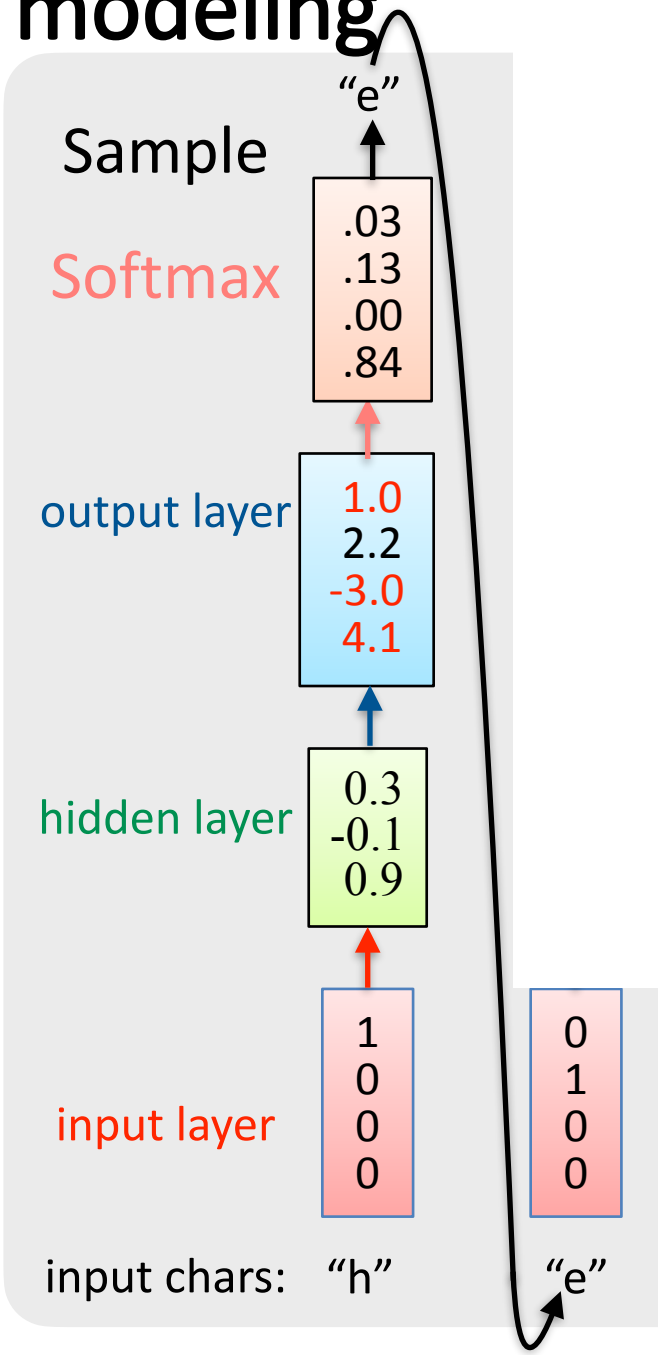


Training sequence: "hello"

Vocabulary: [h, e, l, o]

Example: language modeling

At test time, generate new text: sample characters one at a time, feed back to model

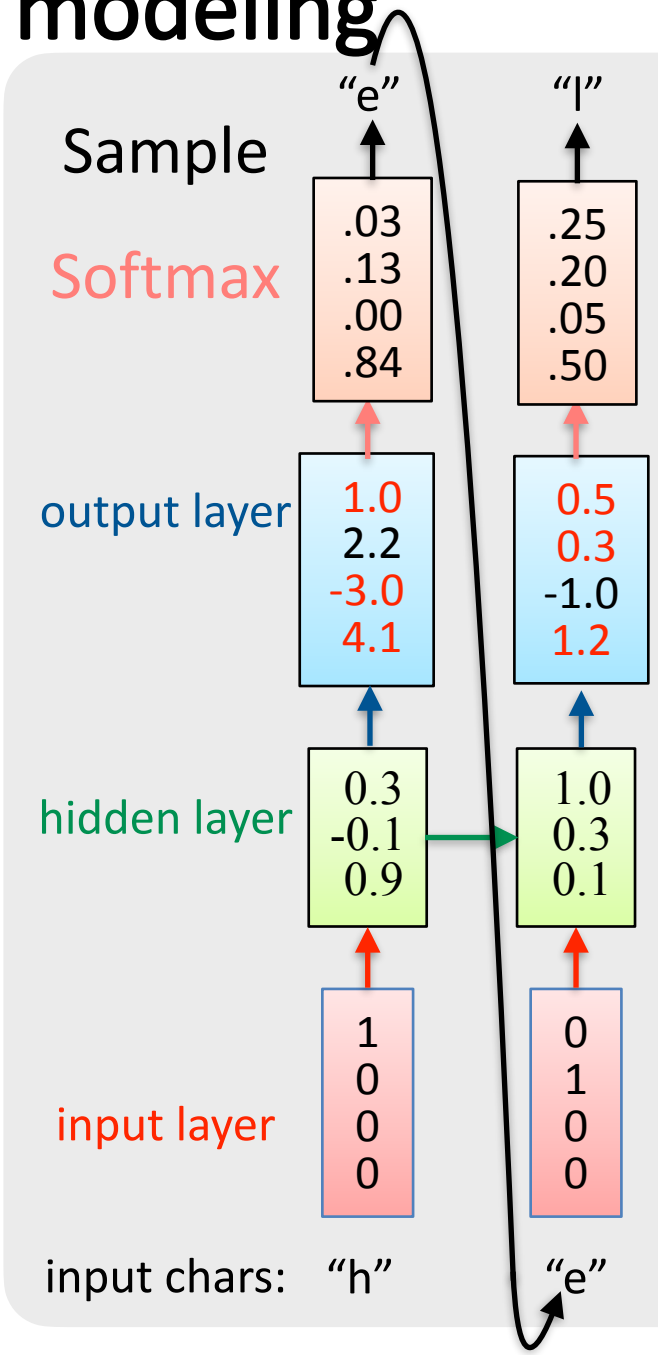


Training sequence: "hello"

Vocabulary: [h, e, l, o]

Example: language modeling

At test time, generate new text: sample characters one at a time, feed back to model



Training sequence: "hello"

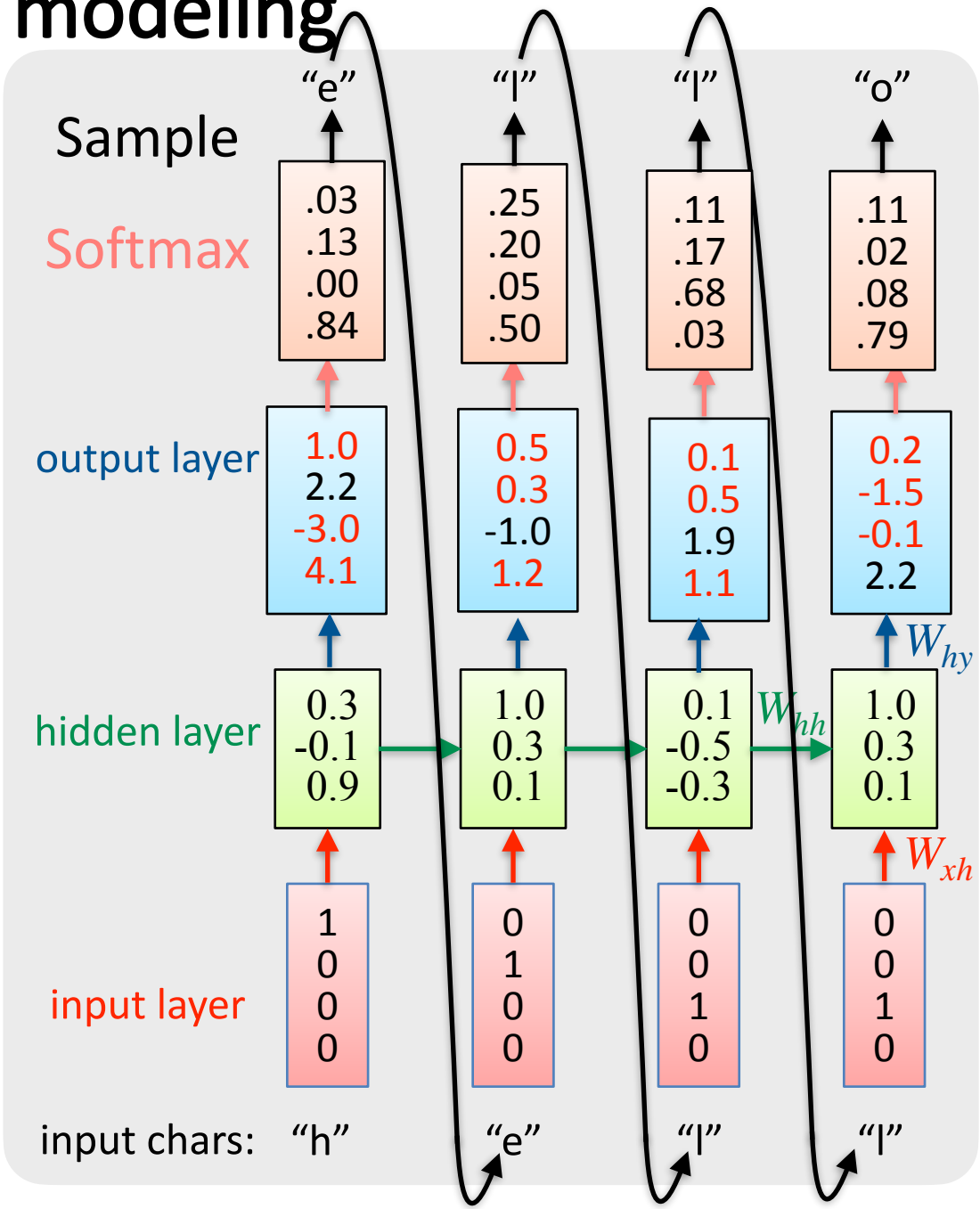
Vocabulary: [h, e, l, o]

Example: language modeling

At test time, generate new text: sample characters one at a time, feed back to model

Training sequence: “hello”

Vocabulary: [h, e, l, o]

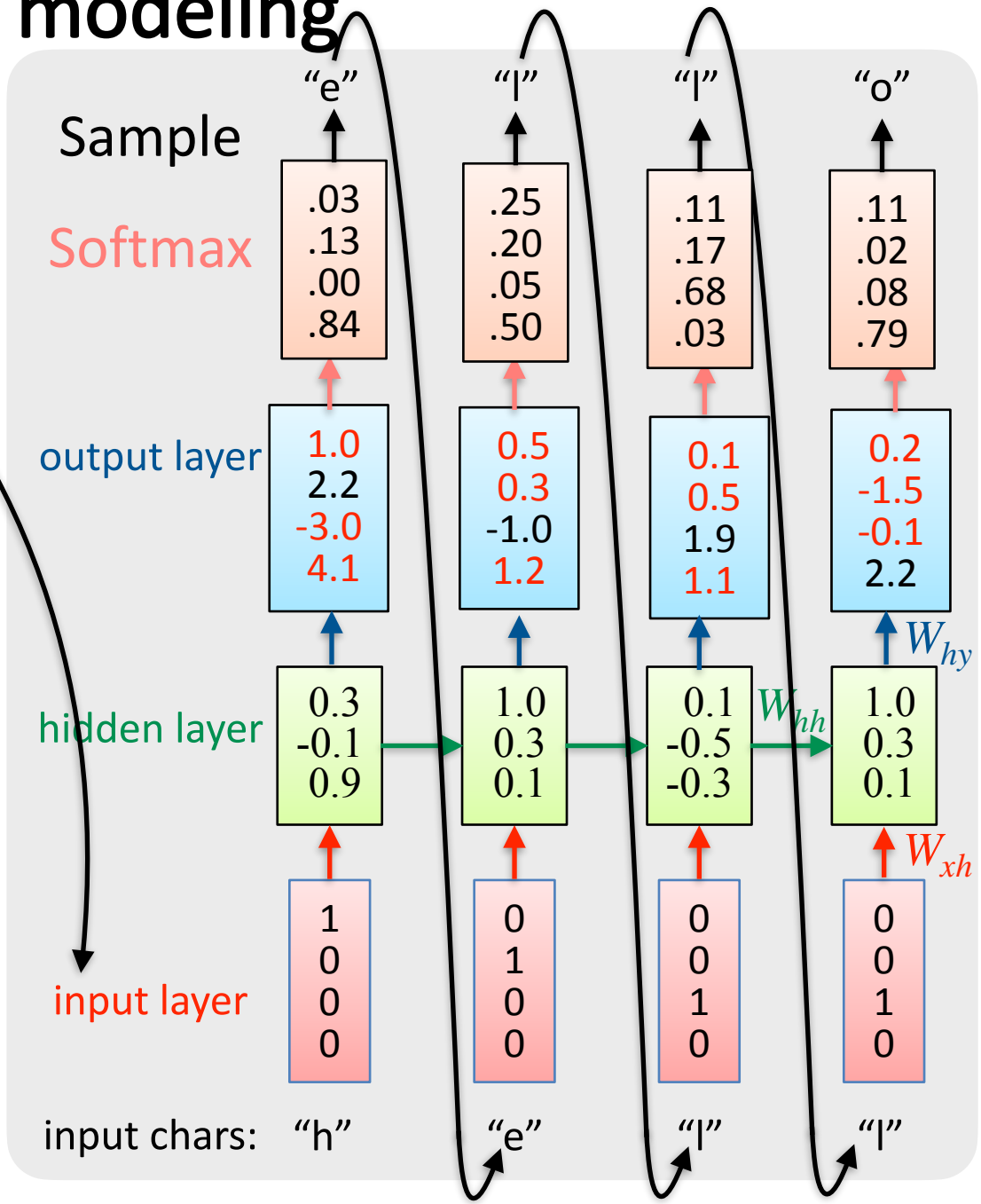


Example: language modeling

So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



Example: language modeling

So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer