Deep Learning V

CS771: Introduction to Machine Learning

Purushottam Kar

Announcements

We have received a quiz 4 copy with no identification marks on it

- *Once grades are released, the concerned student may claim the copy*
- *Attendance records, handwriting checks will be used to verify*
- *A token penalty of 1 mark would be awarded to the student for this quiz*

The above scheme worked only because only one unmarked copy

- *Two or more unmarked copies put this process in jeopardy*
- *Students are advised to write their name, roll number and department very legibly on each sheet of the answer sheet in the space provided*

Recap of Last Lecture

Neural networks can be adapted to exploit structure in data

- *Faster/more accurate than simple feedforward networks*
- *May learn more informative features with less training data*
- *Key idea: ask initial hidden layers to learn low-level features that only combine a few raw features "locally" e.g. pixels that are near to each other*
- *Higher level hidden layers could use these to learn more abstract features*

Convolutional Neural Networks

- *Use the idea of convolutions to implement a local feature extractor*
- *Notion of convolutions can be extended to 1D, 2D, 3D, … nD data*
- *Pooling operations popular in reducing total number of features*
- *CNNs very popular for image/video processing (even text processing)*

Pooling and Strides – an example

Pooling and Strides – an example

1	2	3	4	5
9	8	7	6	0
0	2	4	6	8
2	3	4	5	1
6	7	8	9	0

Pooling and Strides – an example

1	2	3	4	5
9	8	7	6	0
0	2	4	6	8
2	3	4	5	1
6	7	8	9	0
0	0	0	0	0

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	$\boxed{3}$	$\boxed{4}$	5	0
9	8	$\boxed{7}$	$\boxed{6}$	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Padding

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

Pooling and Strides – an example

1	2	3	4	5	0
9	8	7	6	0	0
0	2	4	6	8	0
2	3	4	5	1	0
6	7	8	9	0	0
0	0	0	0	0	0

Padding

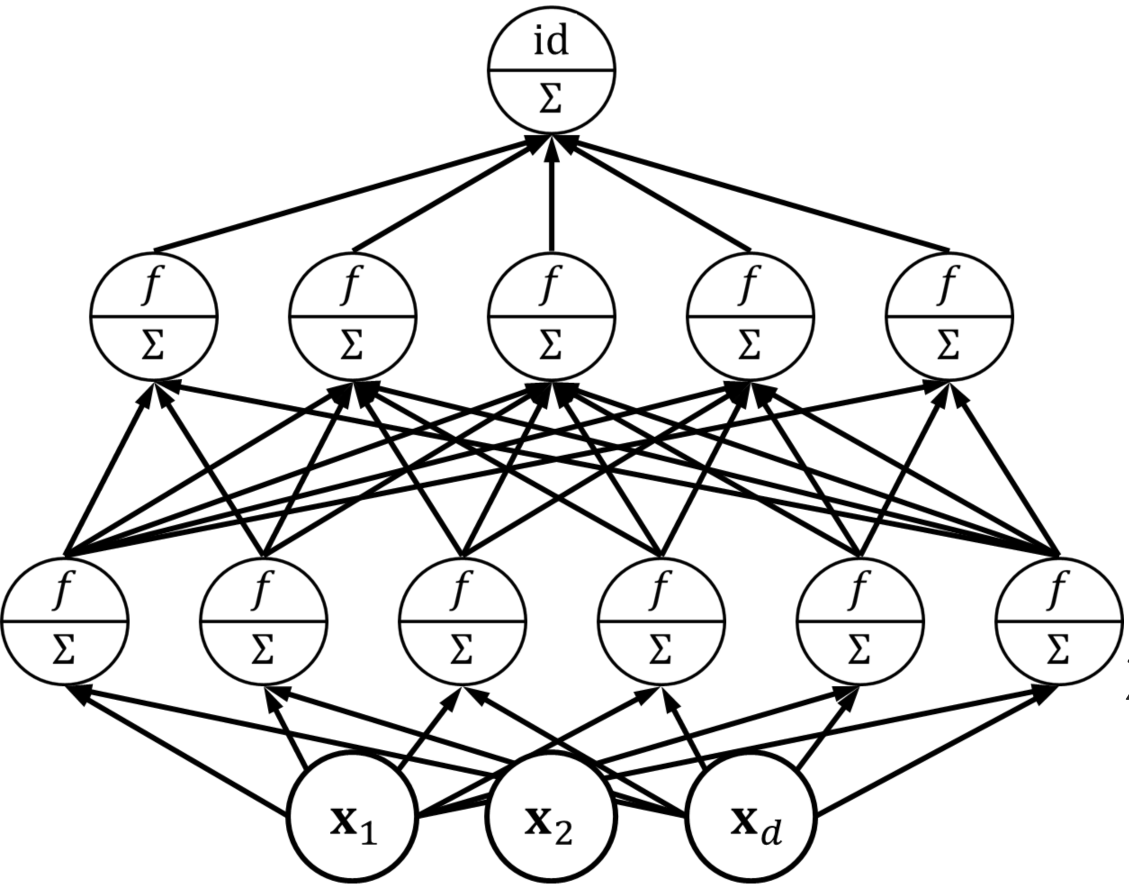
2 x 2 max pool
2 x 1 stride

9	8	7	6	5
3	4	6	8	8
7	8	9	9	0

2 x 2 max pool
1 x 2 stride

9	7	5
9	7	8
3	6	8
7	9	1
7	9	1

Feedforward Networks can be massive



Fully connected layers are powerful

Allow all possible combinations of input dims to create new features which are functions of any subset of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$

New features of the form $f(\mathbf{w}^\top \mathbf{x})$

Also very unnecessary for apps where input has lots of structure

Make networks very bulky – e.g. the AE 784 →1000→500→250→30→250→500→1000→784 needs 2.8 million edge weights to be learnt

From only 60 thousand data points ☹

Also require tons of data to train so many edge weights otherwise NN may overfit

Structured Data needs Local Features

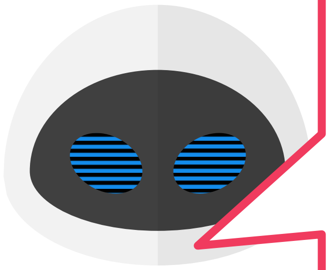
The quick brown fox jumps over the lazy dog

DET ADJ ADJ NN VV PP DET ADJ NN

NP VV PP NP

NP VP

S



CNNs may be (and are) applied to such data as well since even here, we are looking for local features. However, special NN are designed for sequence data e.g. text, DNA

Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Note that "fox" and "dog" interact only at the "sentence" level

More importantly in a context free grammar, same rules apply to, e.g., detect a noun phrase no matter where in sentence we are looking!

Learning with Sequence Data

Data such as text, stock values, DNA best represented as sequences

- *Bag-of-"tokens" can be (and are indeed) used as feature representation*
  - In text, words are tokens, in DNA, 'A''T''G''C' are tokens, in stocks, price values are tokens
  - Fast and convenient but may offer poor results for some apps, good results for others
  - If we apply bag-of-words to DNA, all we will get is a 4-dim representation – poor!
  - Bag of token approach may not work if tokens are continuous values e.g. prices
- *Alternate solution – bag of n-grams*
  - E.g. in 2-gram, we count how many AT, AG, GT, GC etc … in the sequence – 16 dim feature
  - If we count n-grams, we will get a 4^n dimensional representation – expensive!!
  - N-gram approach also does not work if tokens are continuous
- *Another solution – use convolutions (CNNs)*
  - Convolutions would capture features that account for a bunch of neighboring tokens
  - Higher order convolutions would capture higher order features (or so we hope)

Learning with Sequence Data

A classical way to model sequence data is the *time series* model

Suppose we have a sequence $x_1, x_2, \dots, x_T$ *(assume* $x_t \in \mathbb{R}$ *for now)*

Taking $x_t \in \mathbb{N}, \{0,1\}^d$ *can allow us to encode text, DNA as a time series as well*

The linear autoregressive (AR) model postulates that x_t *depends linearly only on previous few, say* k *tokens i.e.* $x_t = a_1 \cdot x_{t-1} + a_2 \cdot x_{t-2} + \cdots + a_k x_{t-k}$

Works decently if, say the time series does not jump around too much

Consider a *sequence prediction* task now

Input is $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ *and we wish output* $y_1, y_2, \dots, y_T$ *i.e. "tag" each token*

Given a sentence, tag each word with its correct part of speech (noun, verb etc)

Given stock data, predict at each time whether price would now go ↑ or ↓

The previous model can be modified to include features and tags as well

$$y_t = \mathbf{w} \cdot \mathbf{x}_t + (a_1 \cdot y_{t-1} + a_2 \cdot y_{t-2} + \cdots + a_k y_{t-k})$$

A class of neural networks takes this intuition to non-linear models

Recurrent Neural Networks

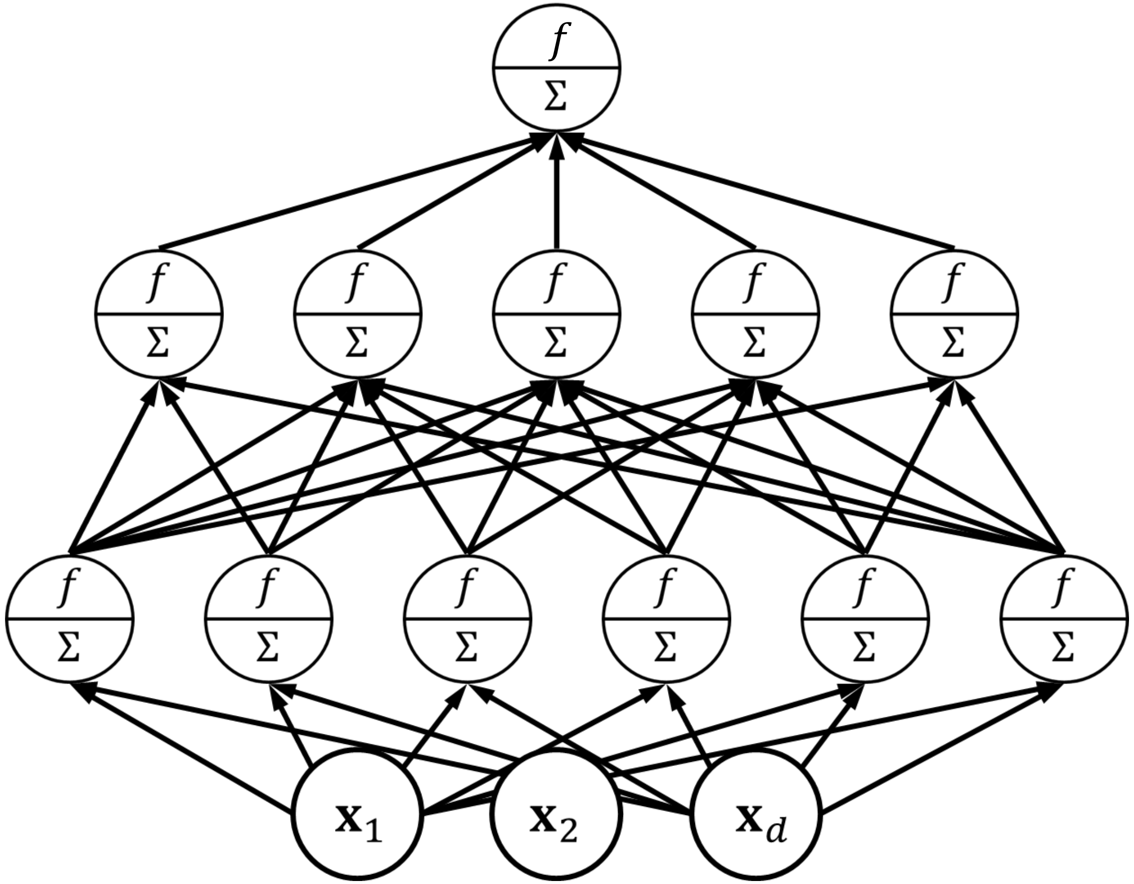
For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information

Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

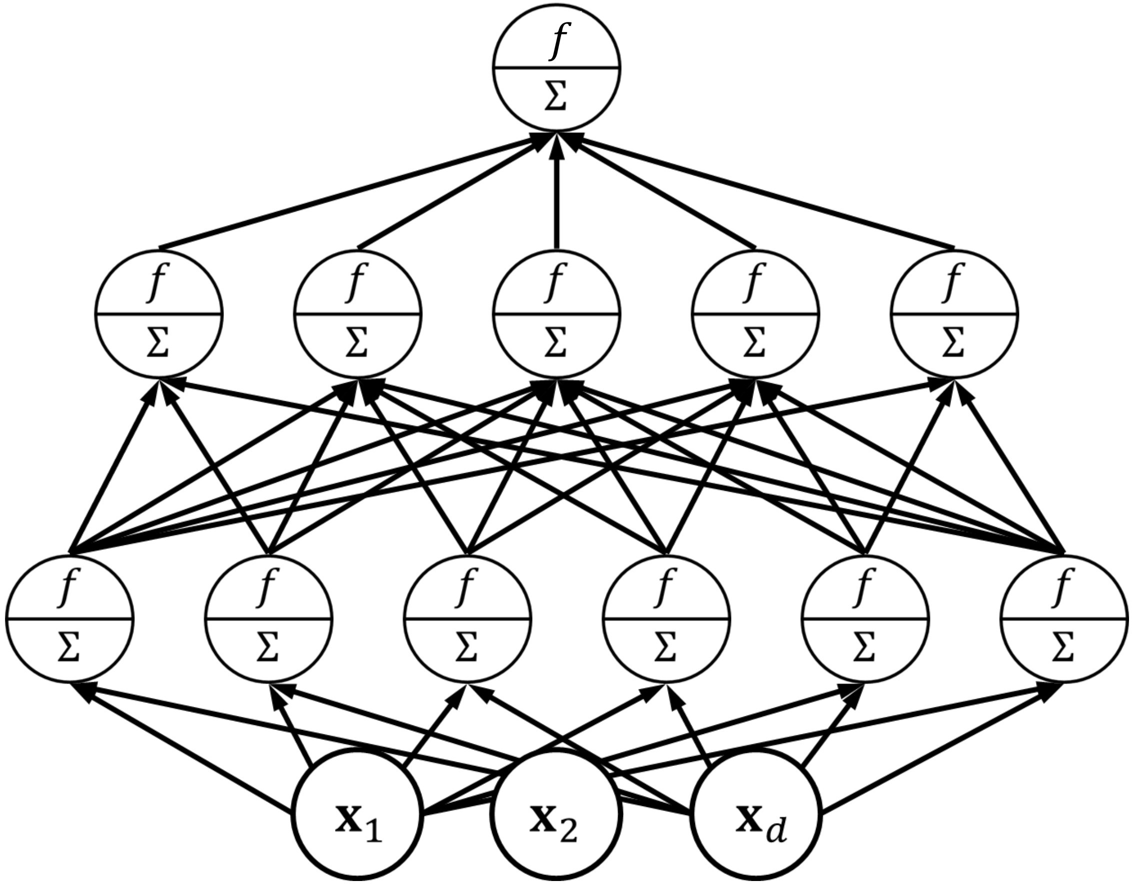
Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information

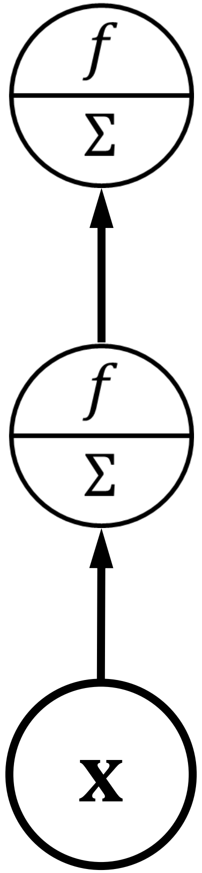


Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information

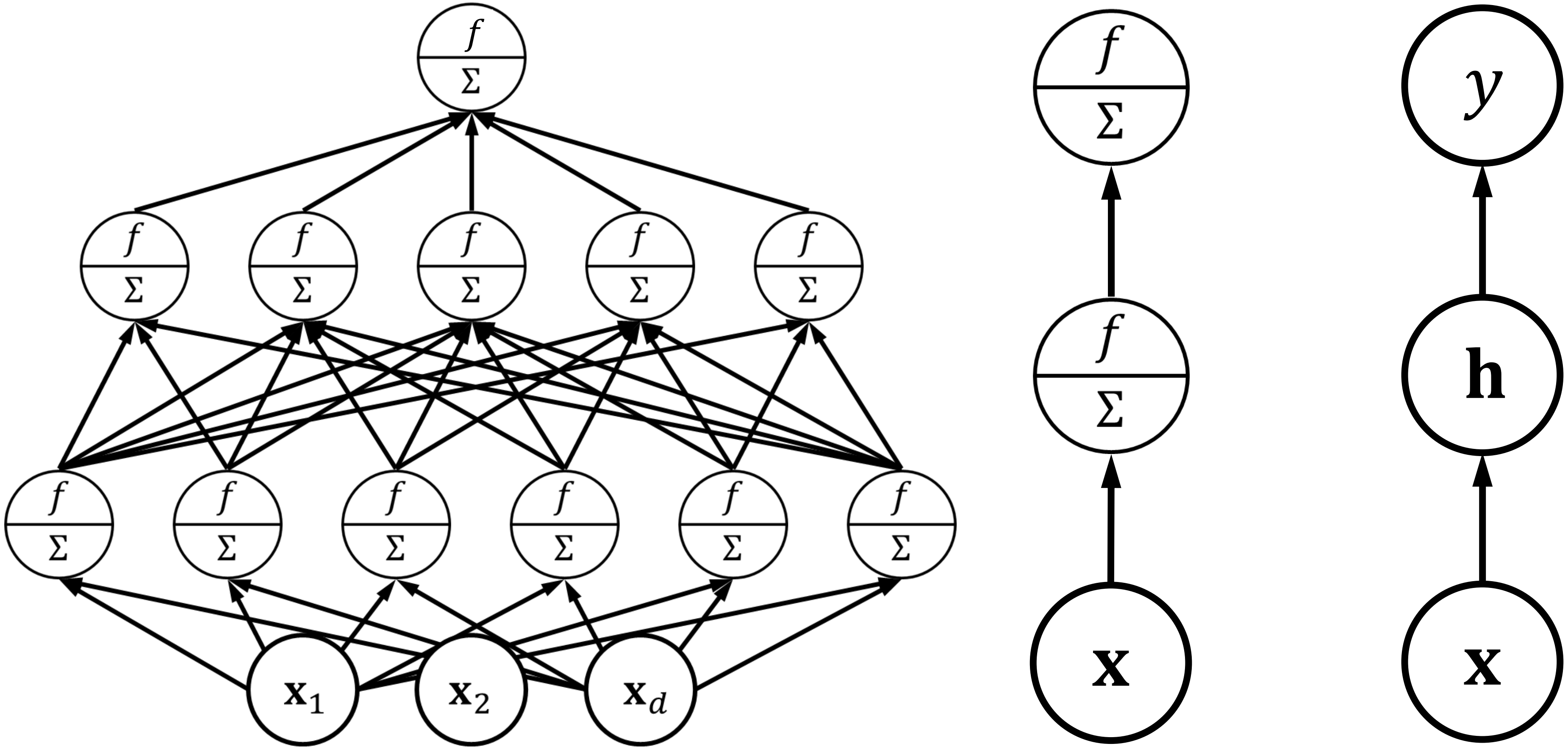




Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

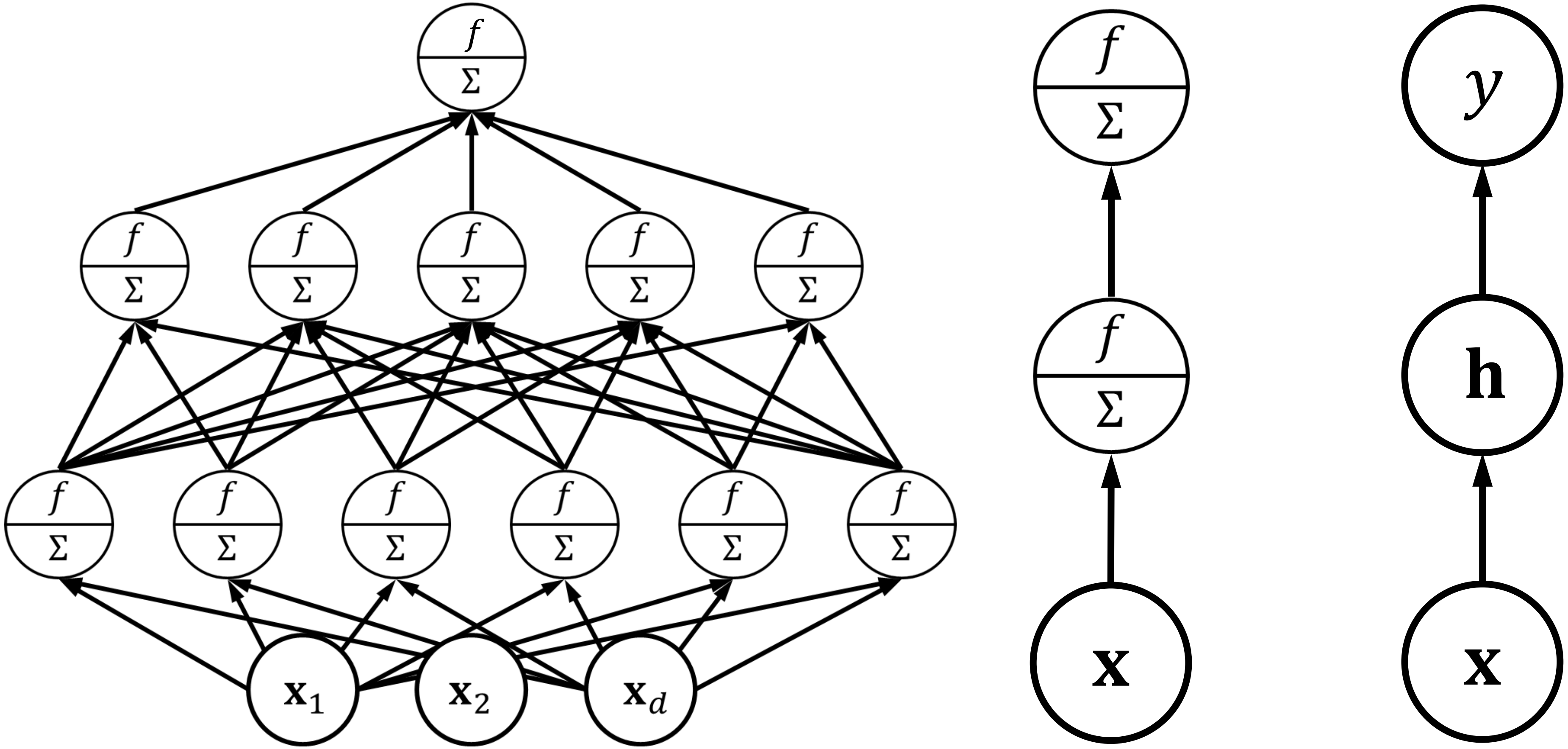
Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information



Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information

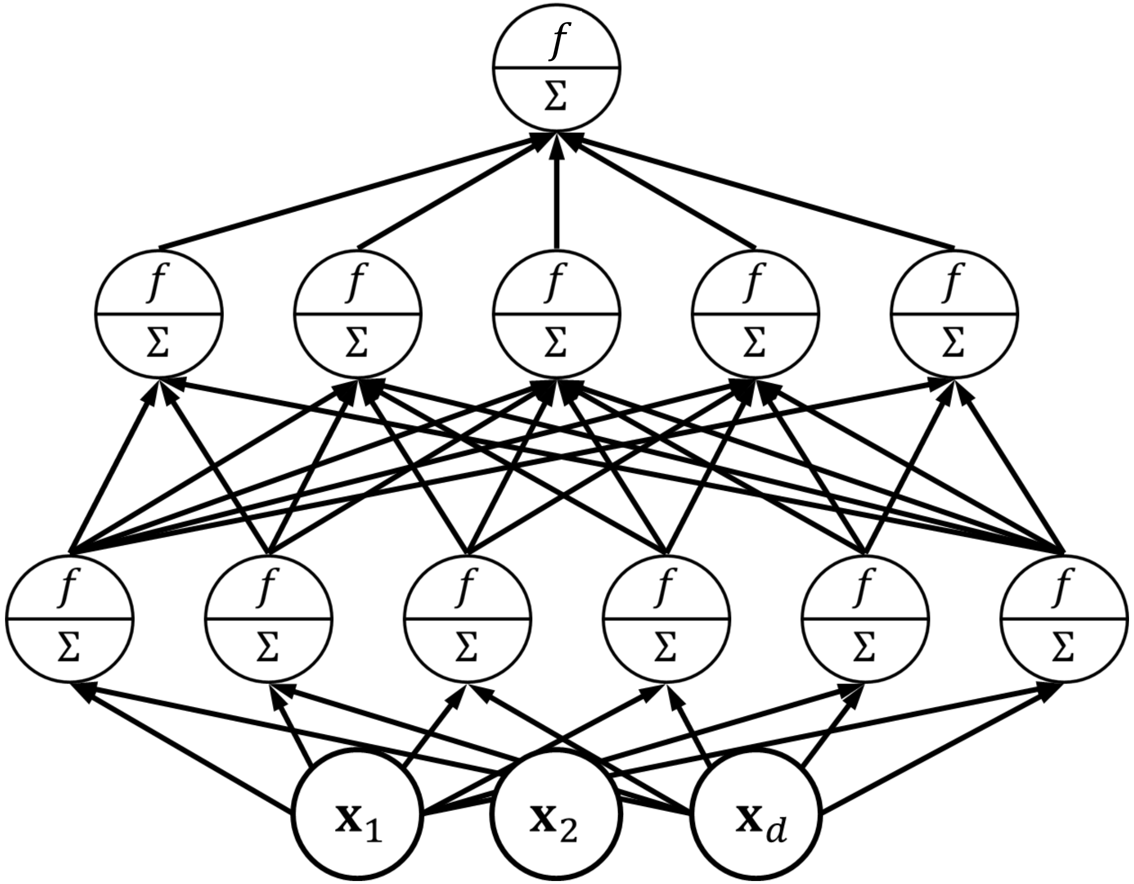


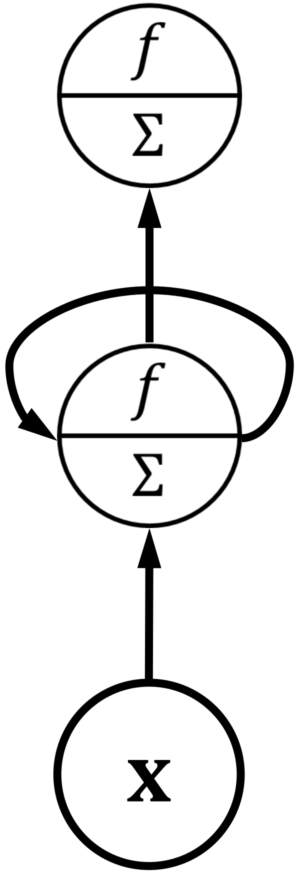
$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x})$$

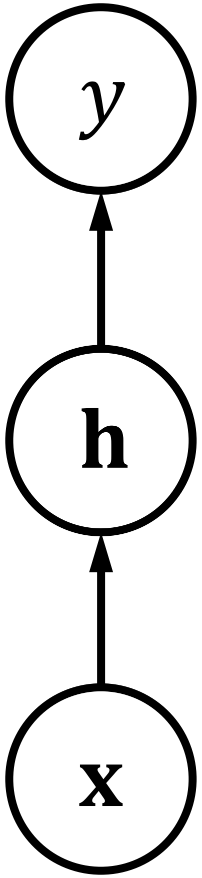
Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information





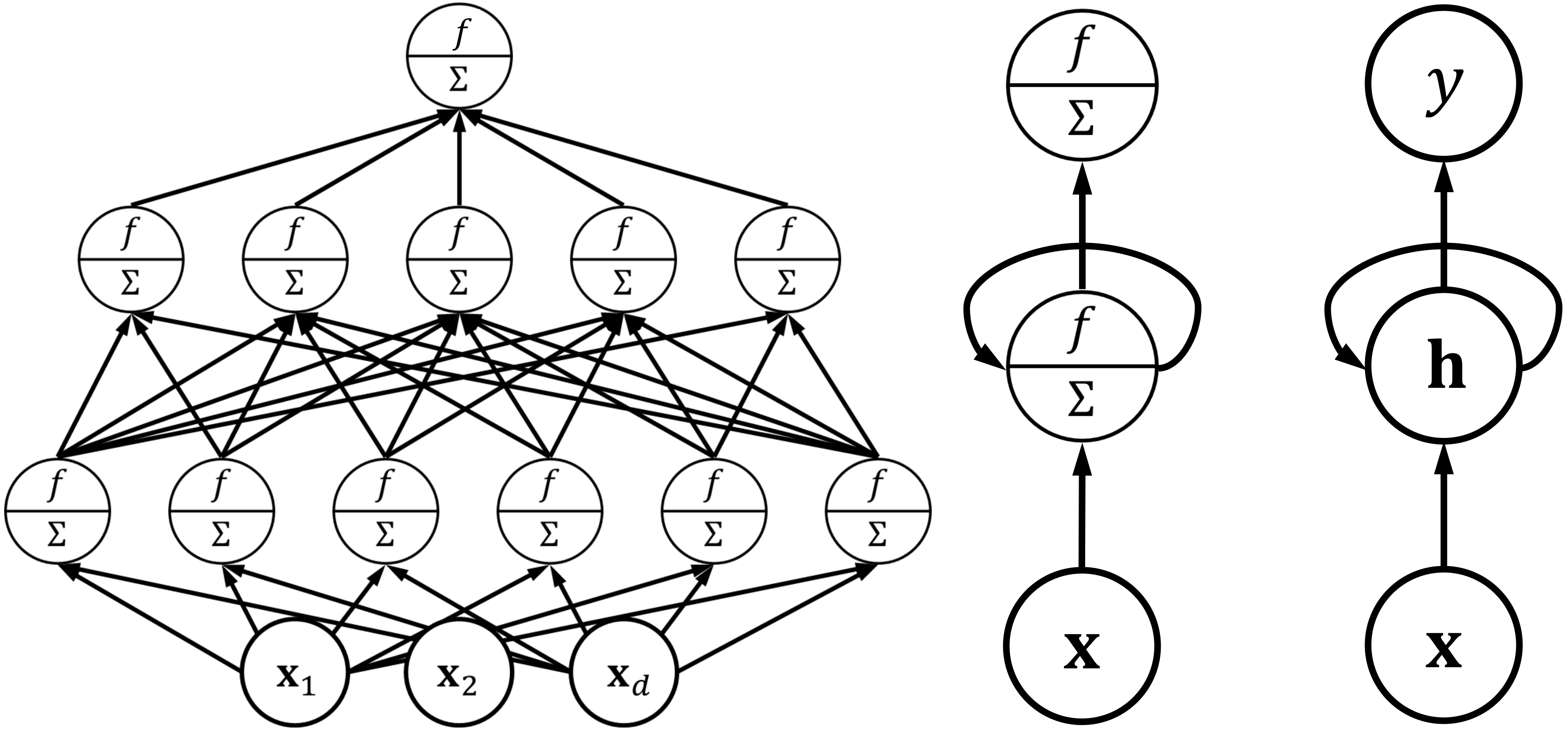


$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x})$$

Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information

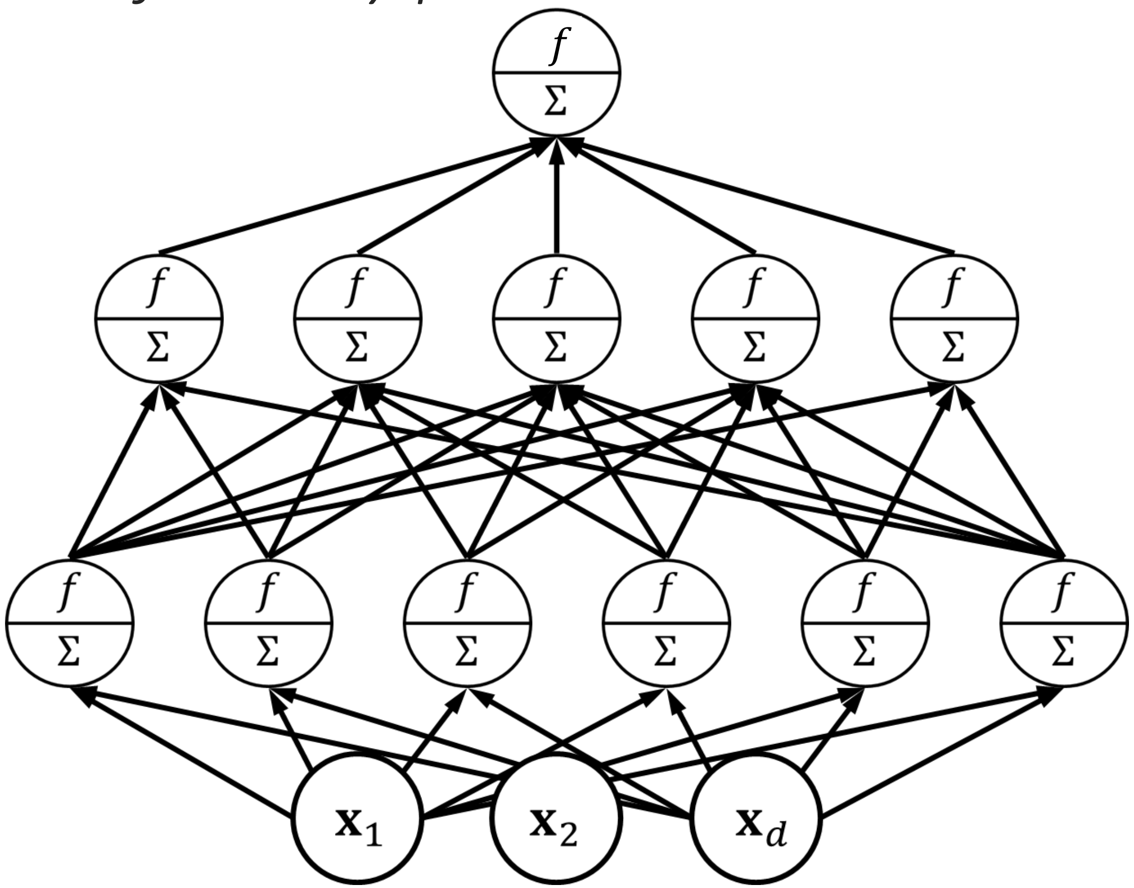


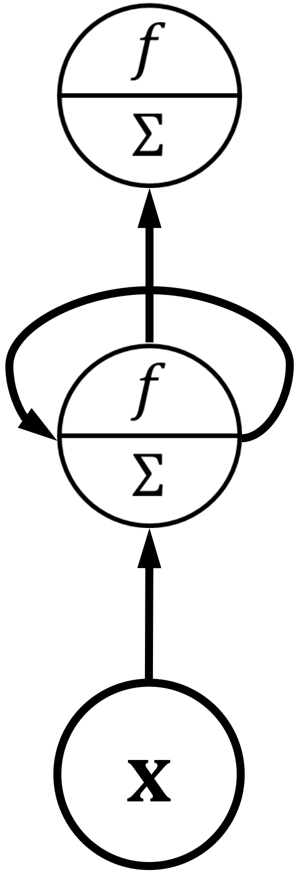
$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x})$$

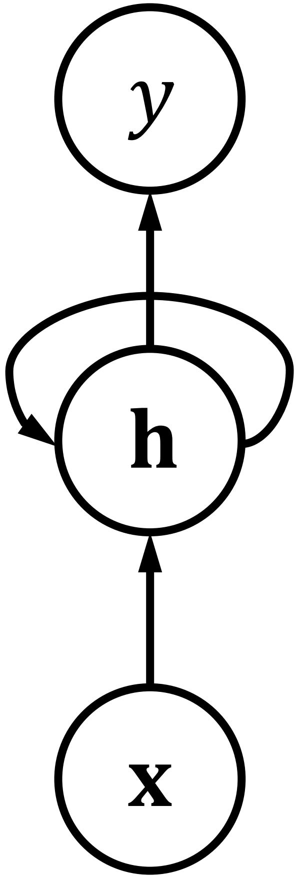
Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information





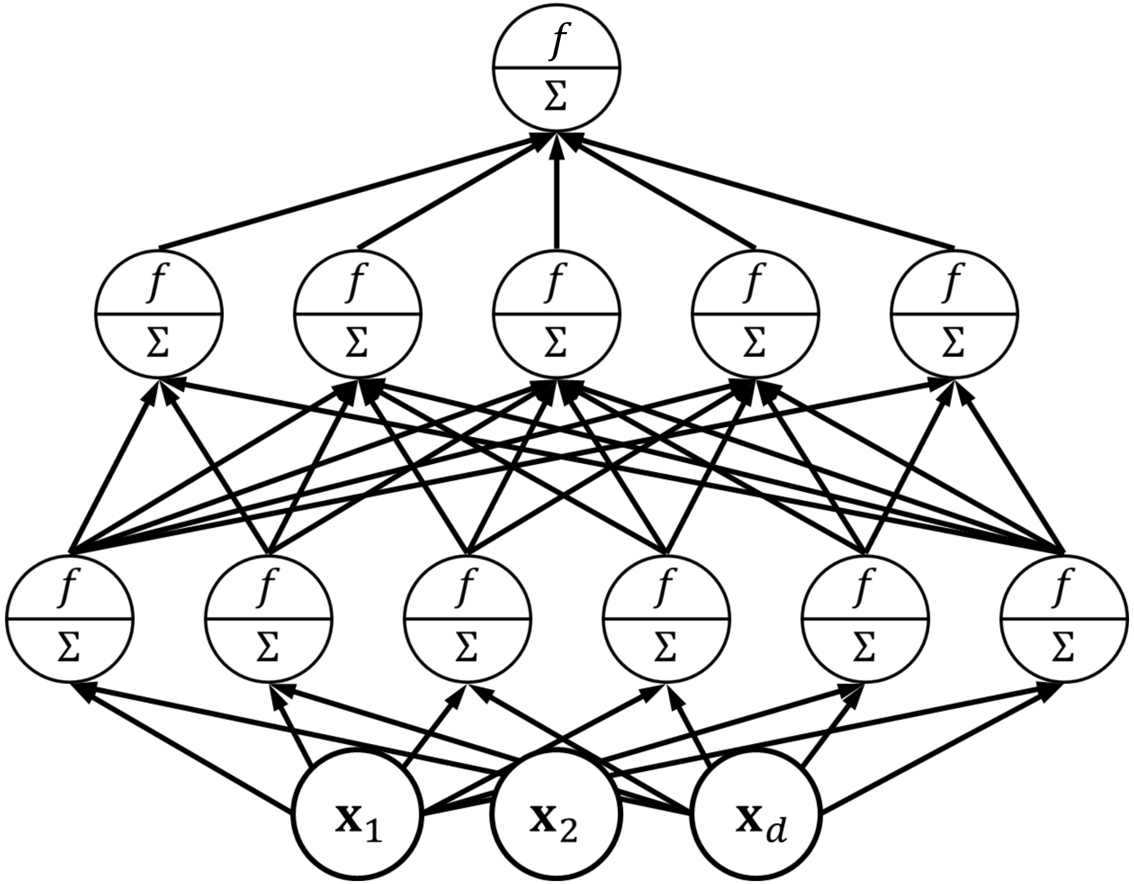


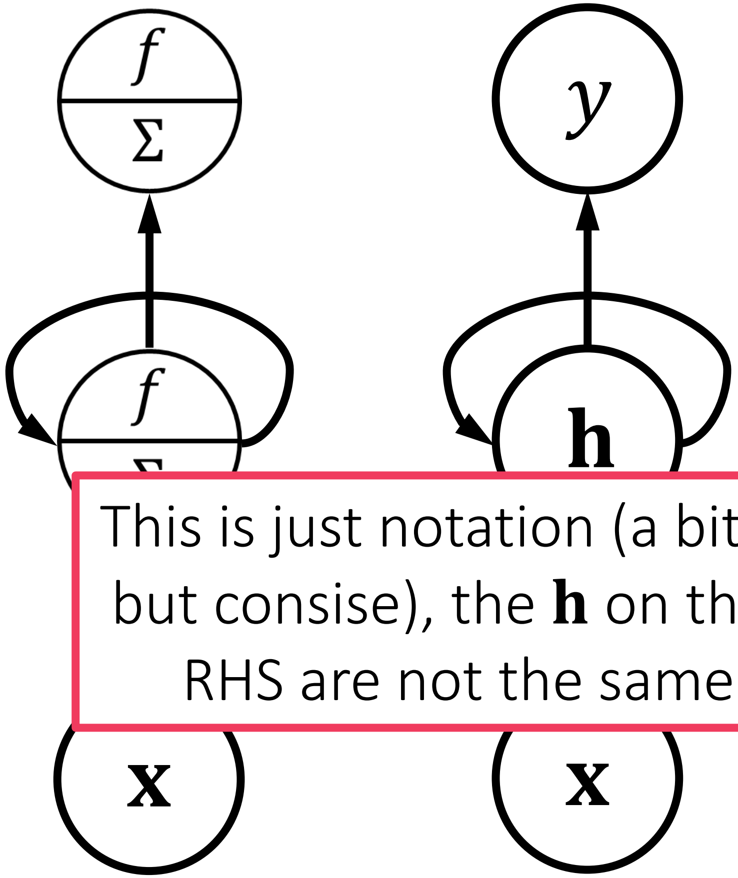
$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x} + U\mathbf{h})$$

Recurrent Neural Networks

For sake of notational clarity, we now represent all hidden layers, all nodes in those hidden layers, using a single hidden layer node

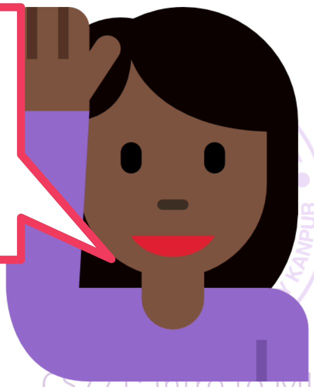
Recurrent neural networks in some sense, create a virtual copy of this network for every point in the time series and allow these copies to share information





$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x} + U\mathbf{h})$$

This is just notation (a bit confusing but consise), the **h** on the LHS and RHS are not the same vector.



Recurrent Neural Networks

$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well

RNNs allow lot of freedom but get harder to train

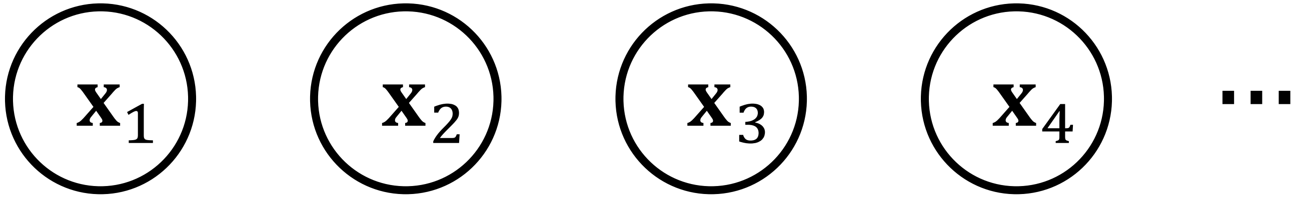
Recurrent Neural Networks

$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

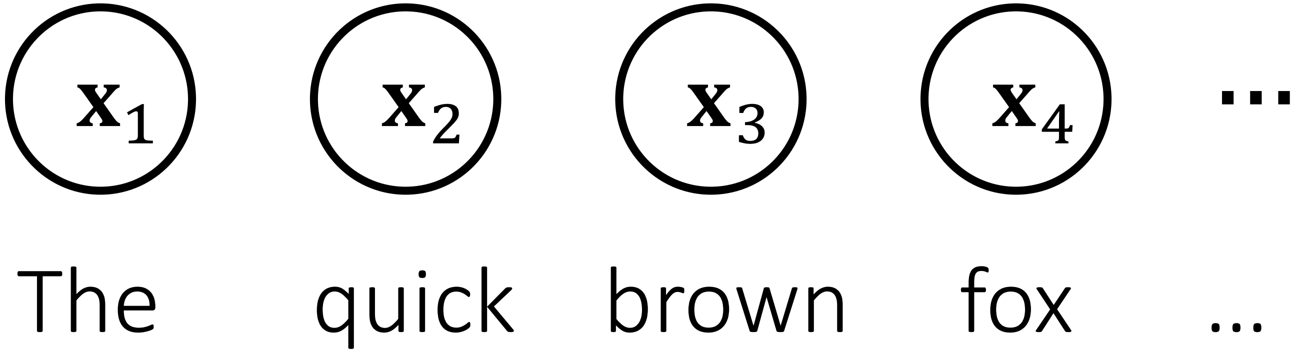
$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

- *$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$*
- *Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well*
- *RNNs allow lot of freedom but get harder to train*



Recurrent Neural Networks



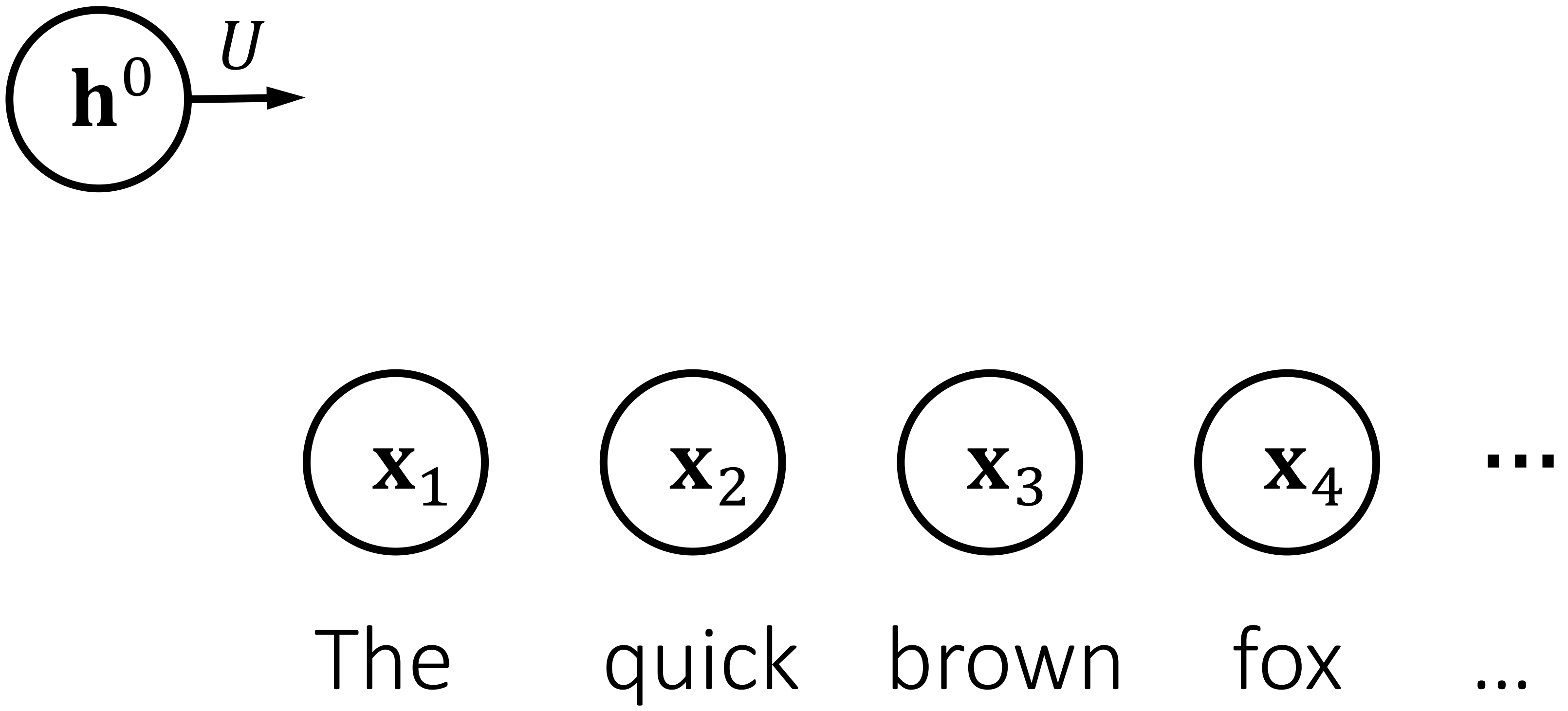
$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

- *$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$*
- *Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well*
- *RNNs allow lot of freedom but get harder to train*

Recurrent Neural Networks



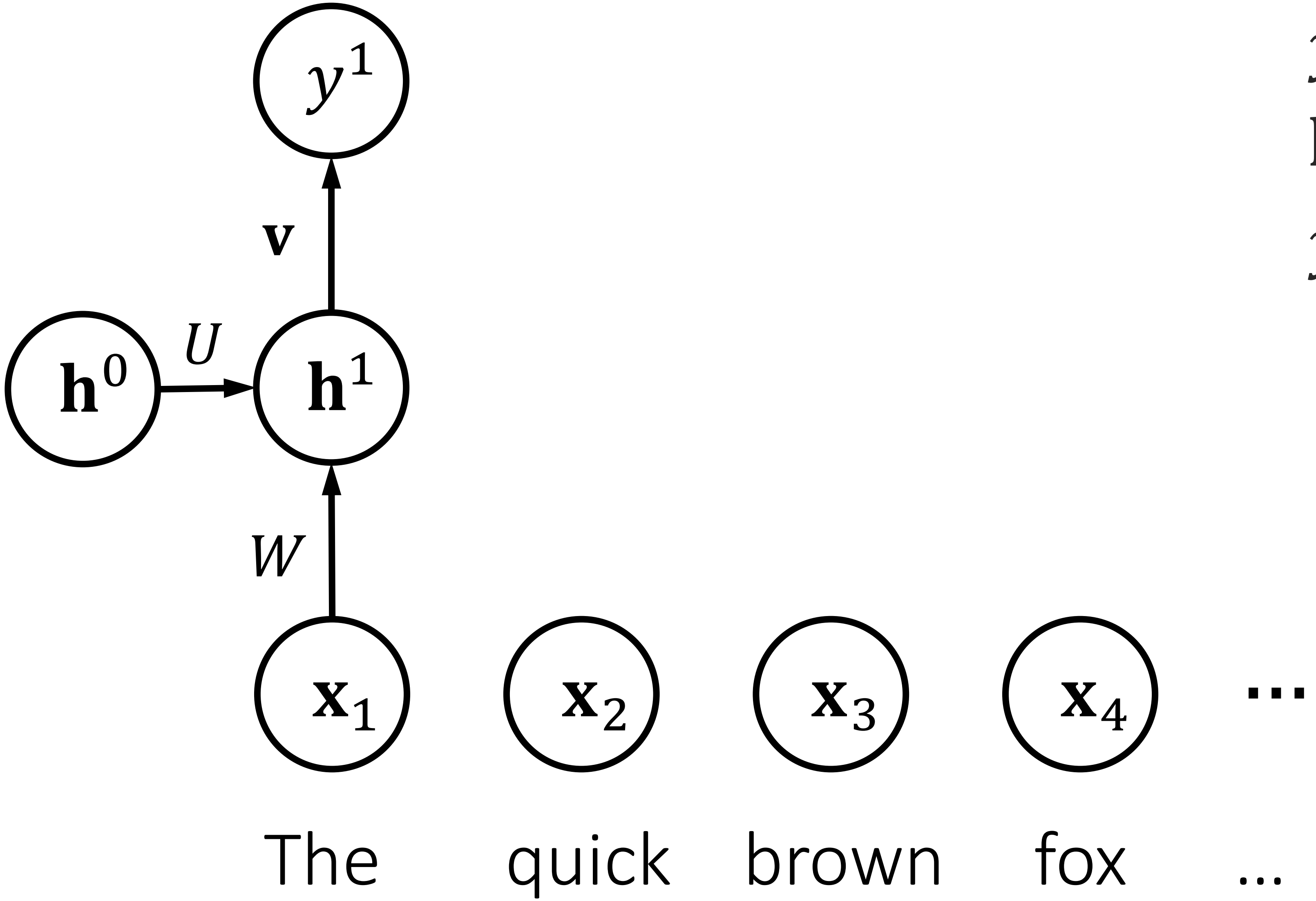
$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

- *$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$*
- *Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well*
- *RNNs allow lot of freedom but get harder to train*

Recurrent Neural Networks



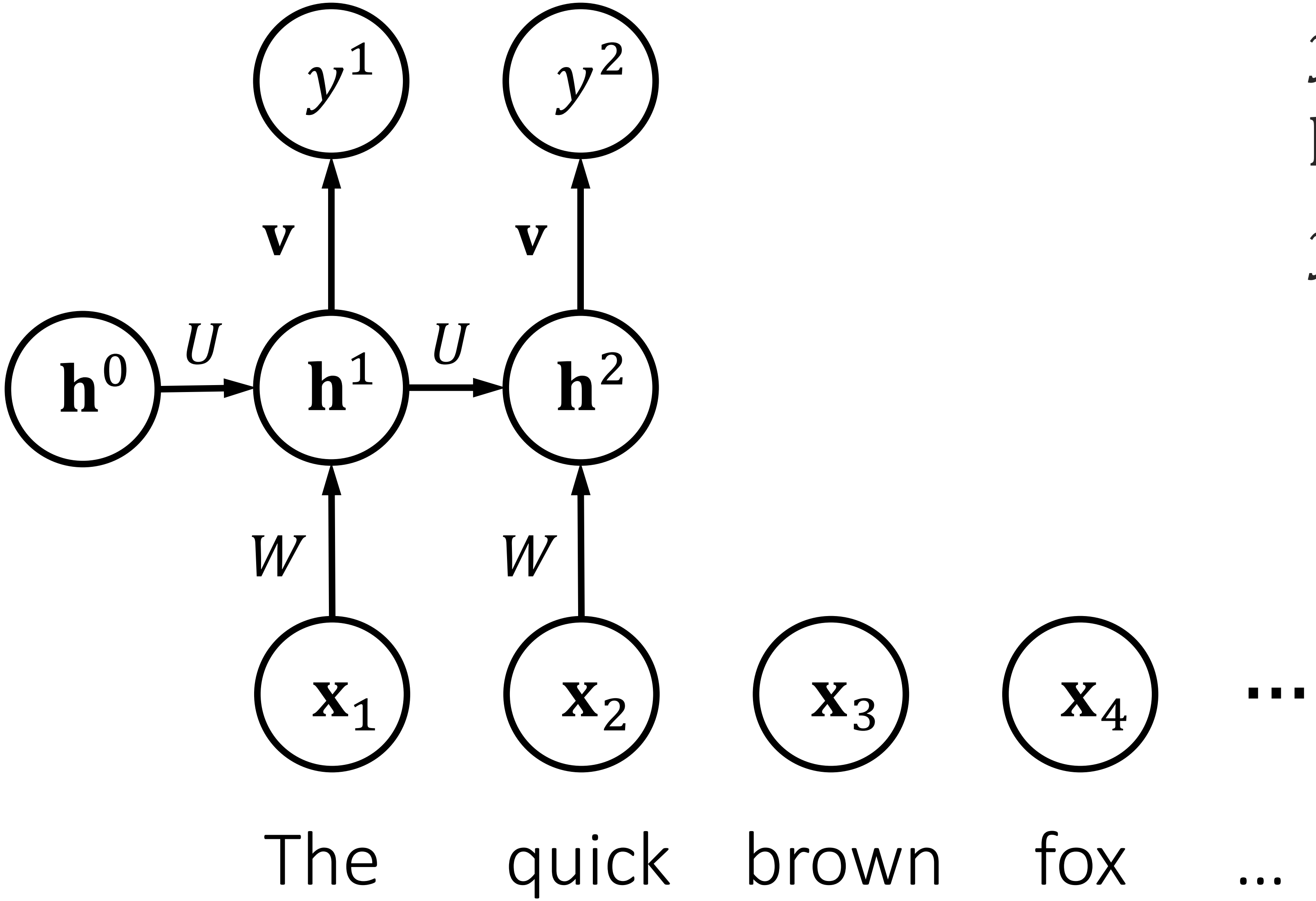
$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

- *$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$*
- *Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well*
- *RNNs allow lot of freedom but get harder to train*

Recurrent Neural Networks



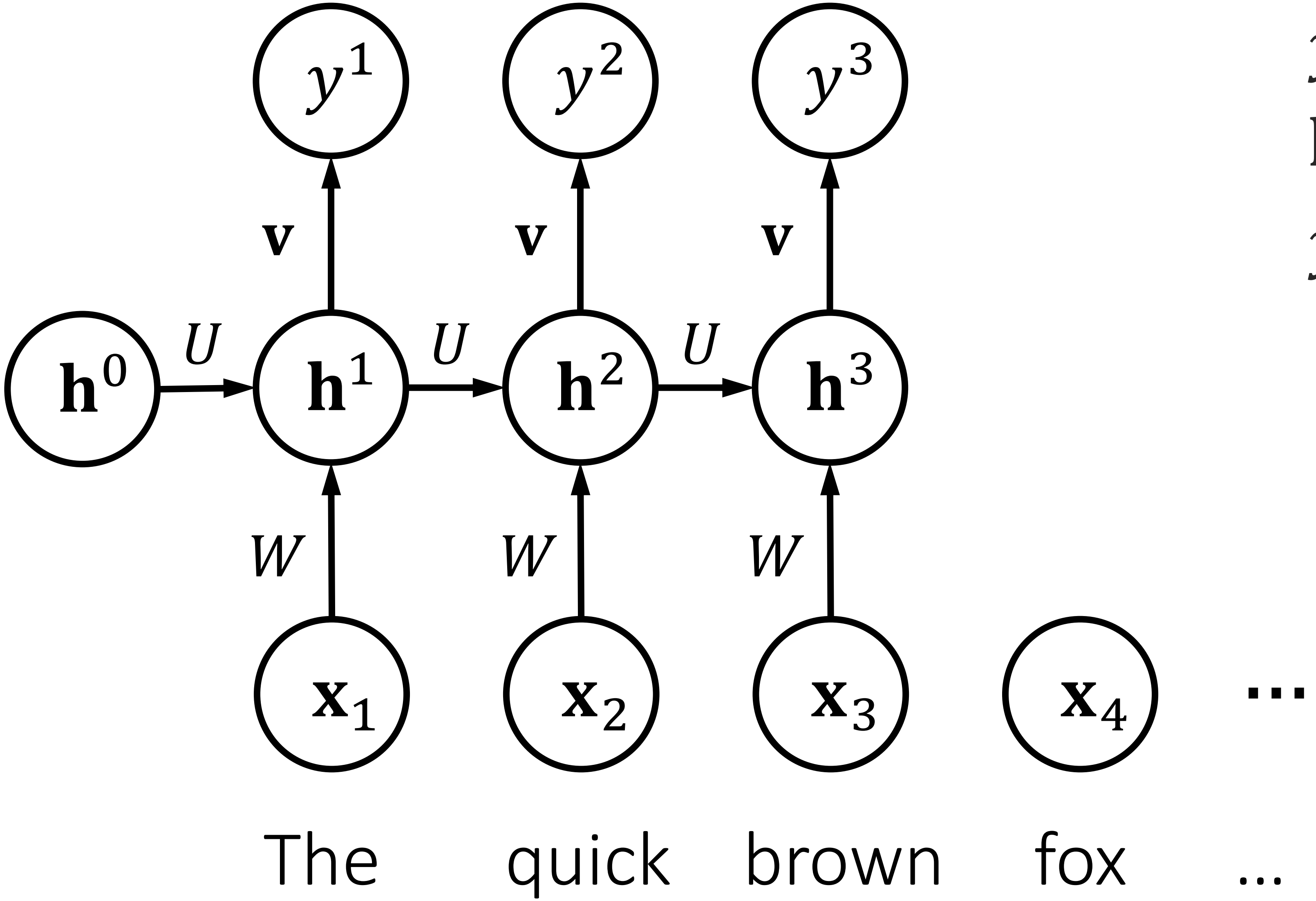
$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$

$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$

$\hat{y}^t$ can do POS tagging etc

- *$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$*
- *Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well*
- *RNNs allow lot of freedom but get harder to train*

Recurrent Neural Networks



$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$

$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$

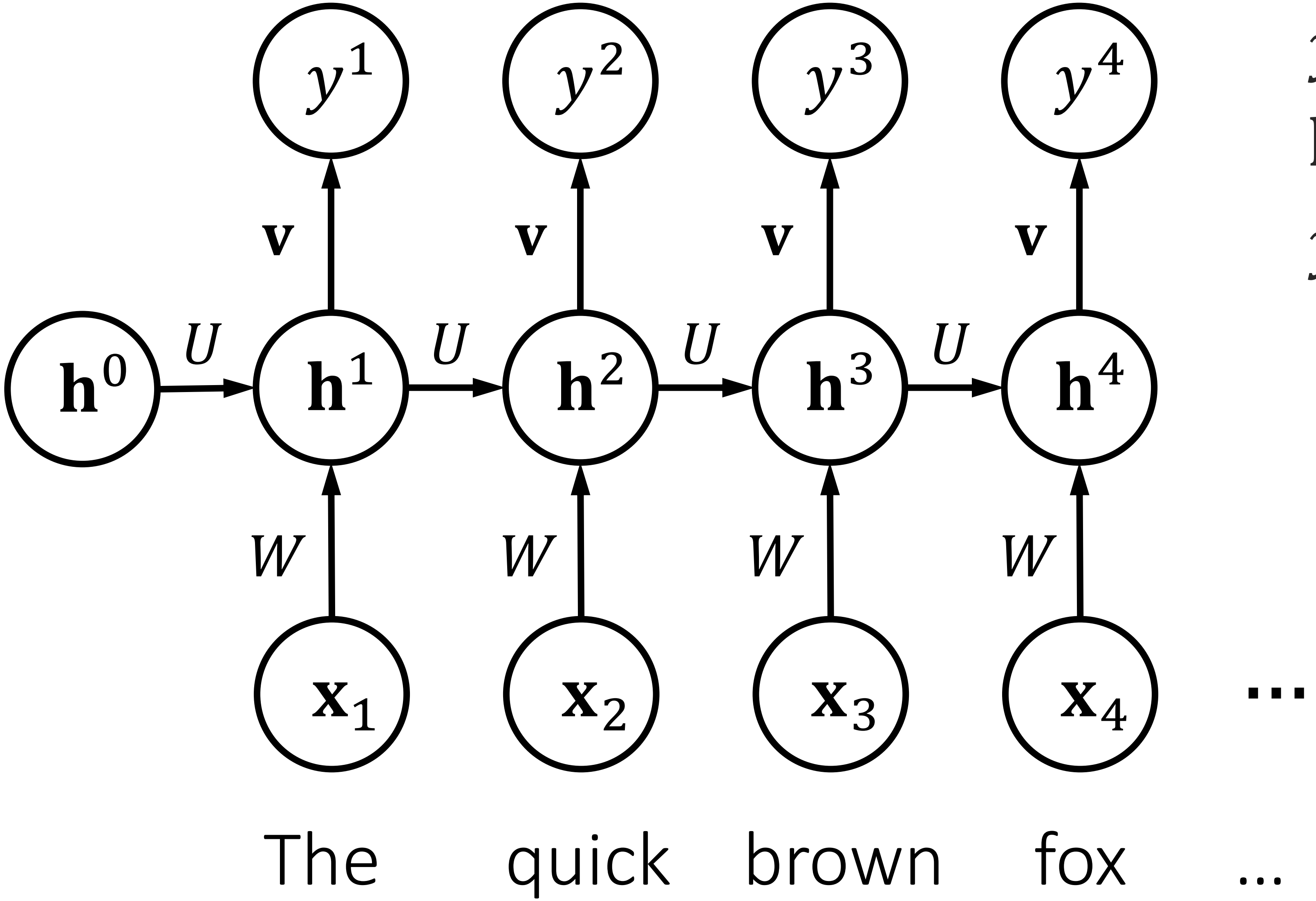
$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well

RNNs allow lot of freedom but get harder to train

Recurrent Neural Networks



$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$
$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

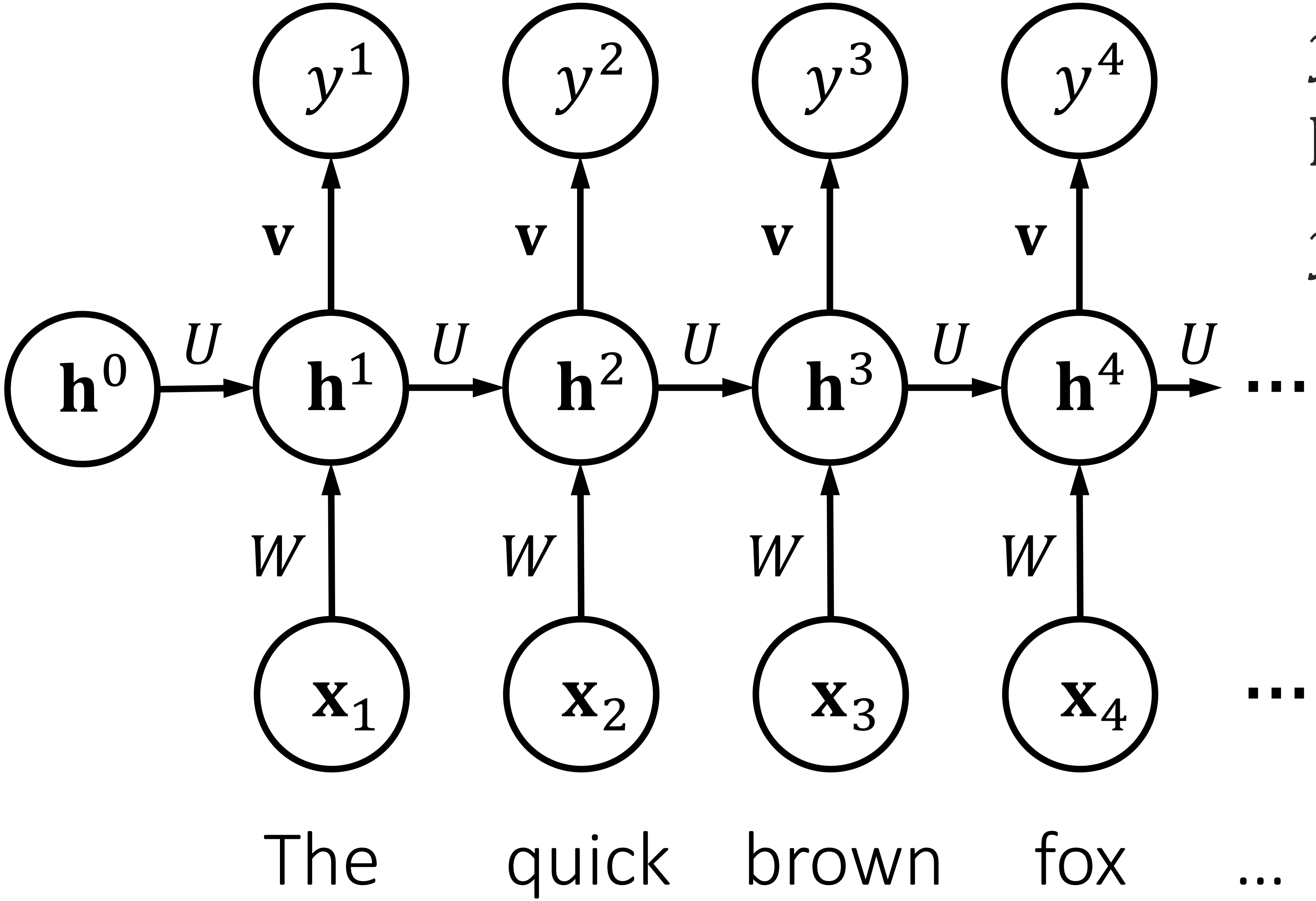
$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well

RNNs allow lot of freedom but get harder to train

Recurrent Neural Networks



$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$
$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

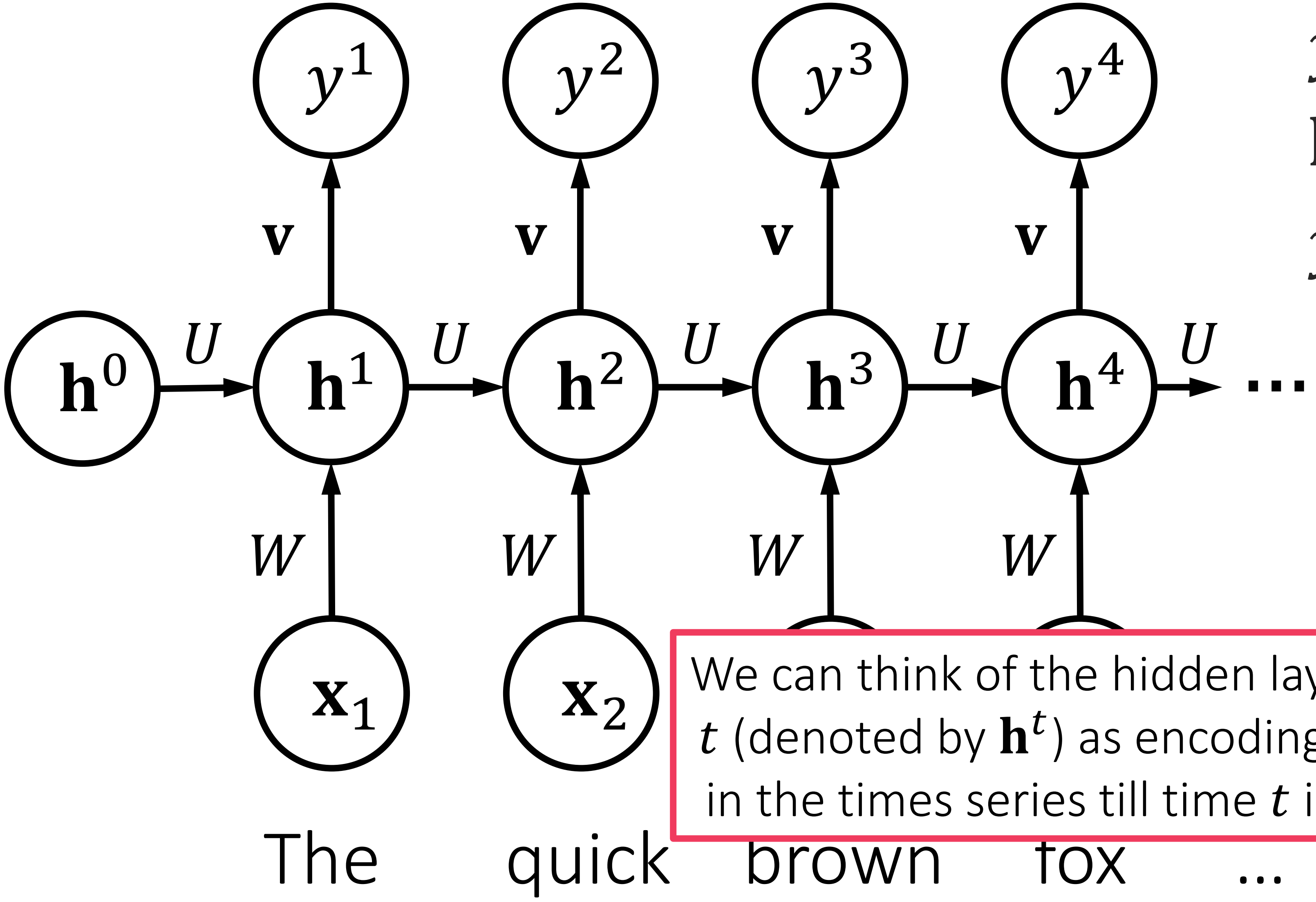
$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous time steps, others can be non recurrent as well

RNNs allow lot of freedom but get harder to train

Recurrent Neural Networks



$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

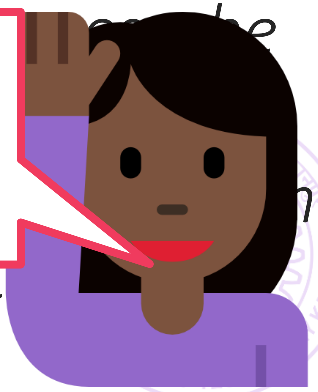
$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

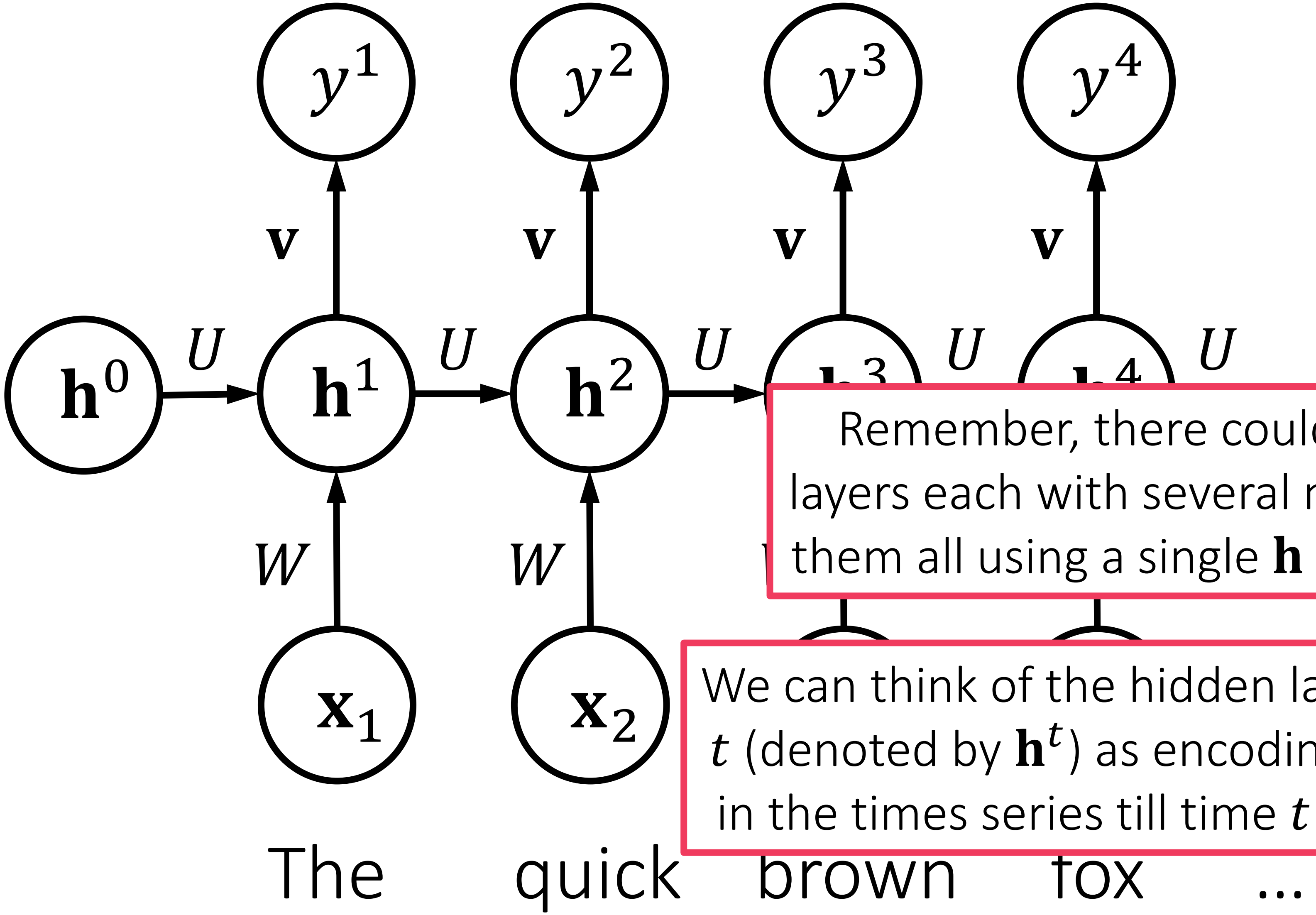
Can have several hidden layers some recurrent i.e. receive input from their counterparts in previous

but get harder t

We can think of the hidden layer activations at time t (denoted by $\mathbf{h}^t$) as encoding whatever happened in the times series till time t i.e. $\{\mathbf{x}^\tau, y^\tau, \hat{y}^\tau\}, \tau < t$



Recurrent Neural Networks



$$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$$

$$\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$$

$\hat{y}^t$ can do POS tagging etc

$\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$

... hidden

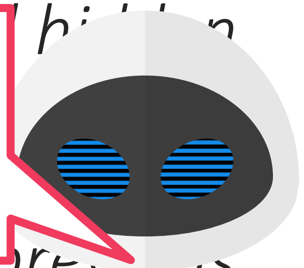
counterparts in previous

...s can be

well

... freedom

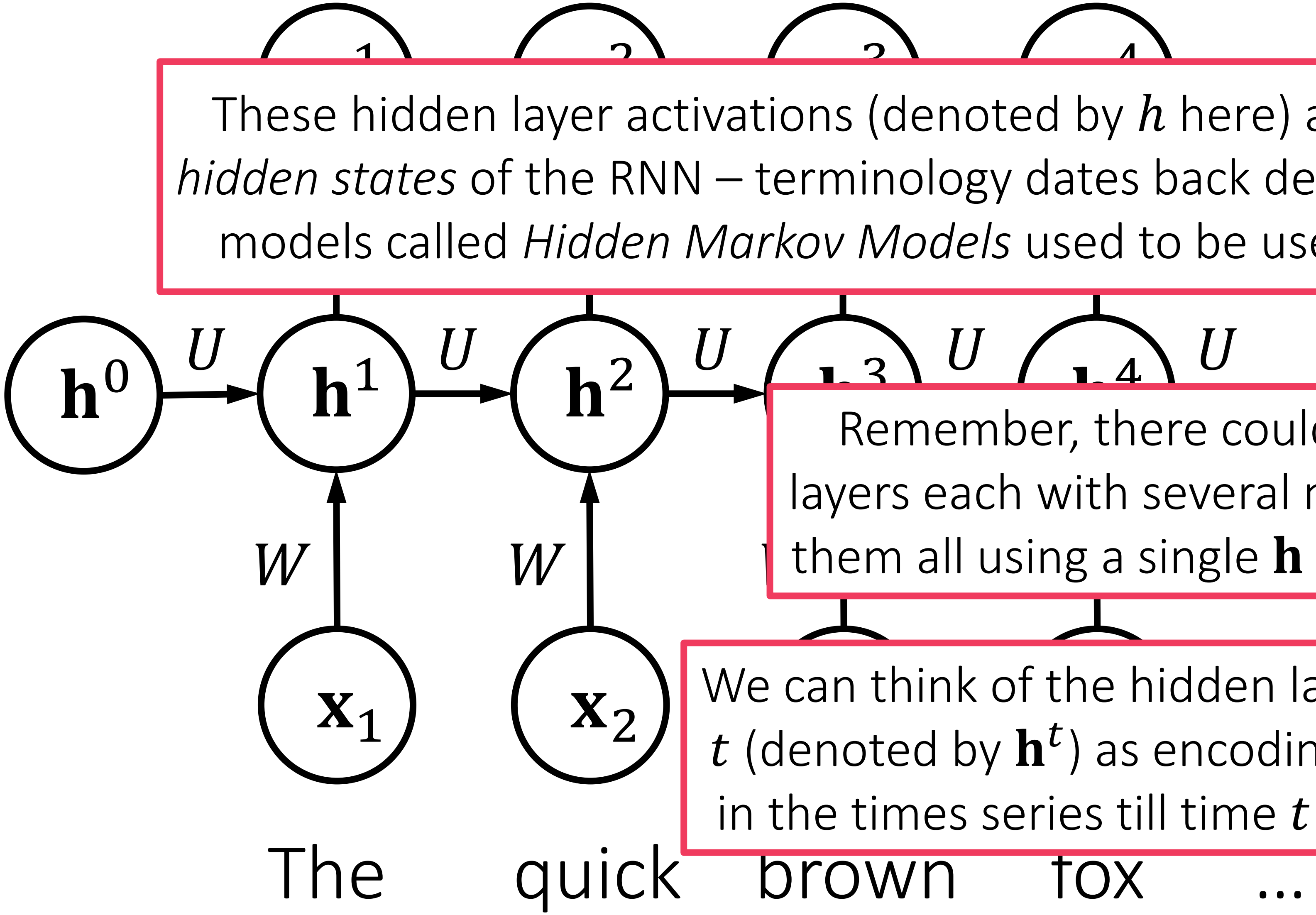
but get harder to train



Remember, there could be multiple hidden layers each with several nodes but are denoting them all using a single $\mathbf{h}$ here for sake of clarity.

We can think of the hidden layer activations at time t (denoted by $\mathbf{h}^t$) as encoding whatever happened in the times series till time t i.e. $\{\mathbf{x}^\tau, y^\tau, \hat{y}^\tau\}, \tau < t$

Recurrent Neural Networks



$\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$

etc

$\hat{y}^t$ can even be a

hidden

counterparts in previous

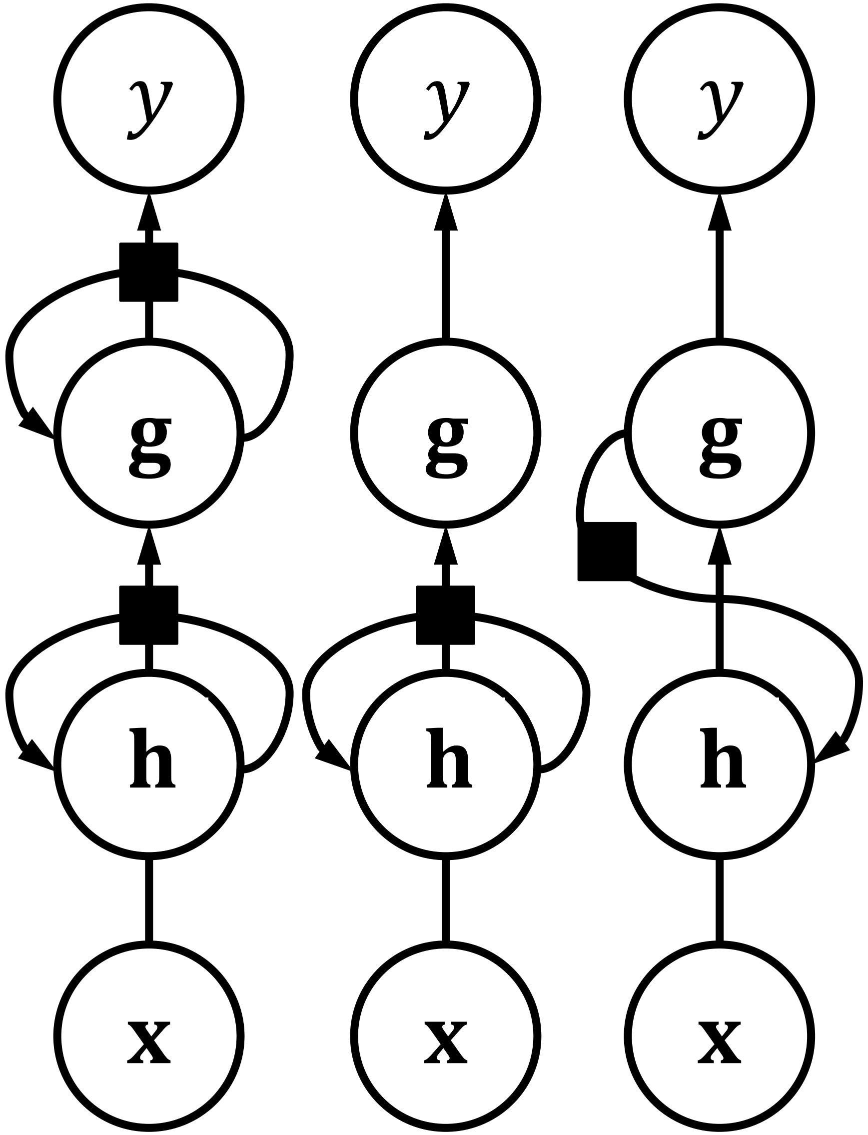
but get harder t

These hidden layer activations (denoted by h here) are often called the *hidden states* of the RNN – terminology dates back decades to when other models called *Hidden Markov Models* used to be used instead of RNNs

Remember, there could be multiple hidden layers each with several nodes but are denoting them all using a single $\mathbf{h}$ here for sake of clarity.

We can think of the hidden layer activations at time t (denoted by $\mathbf{h}^t$) as encoding whatever happened in the times series till time t i.e. $\{\mathbf{x}^\tau, y^\tau, \hat{y}^\tau\}, \tau < t$

RNN Variants



■ indicates a time lag

$\mathbf{g}^t / y^t$ is passed onto $\mathbf{h}^{t+1}$ and not $\mathbf{h}^t$

Often this symbol is omitted and assumed

Notice that RNNs violate the strict rules that feedforward networks obeyed

Nodes can send signals to nodes in lower layers

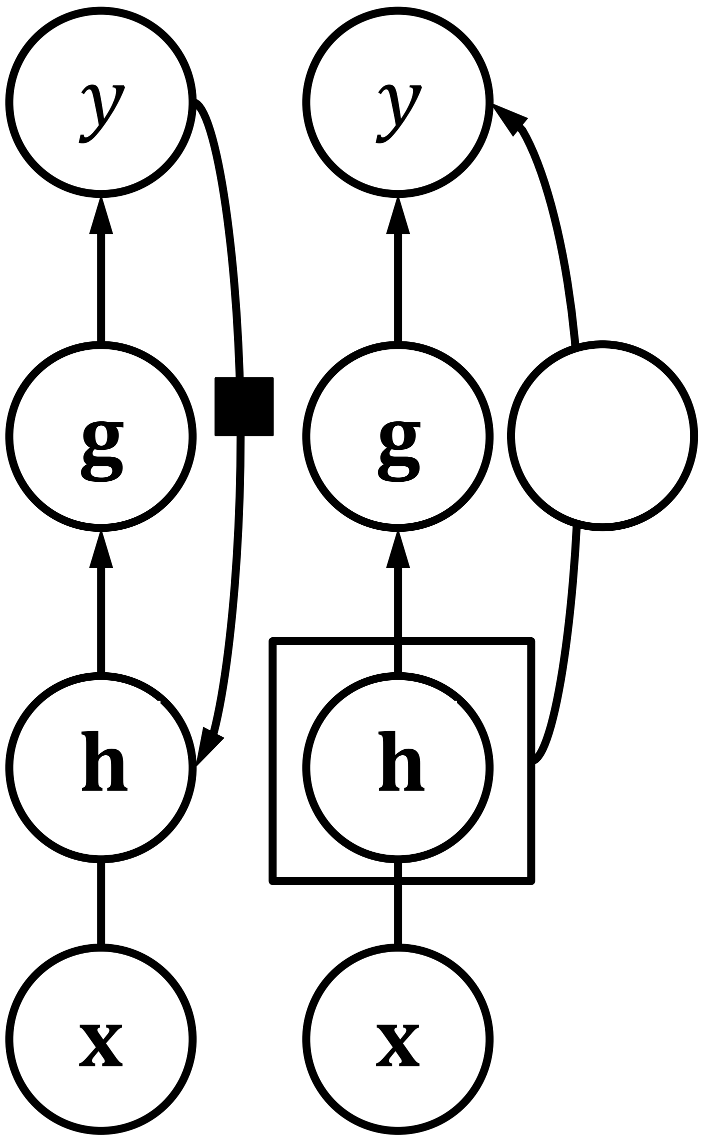
Can even send signals to the future

Can construct RNNs using convolutional layers as well i.e. the hidden layers can be CNN-like

The RNN model can in principle handle sequences of arbitrary length

Practical difficulties arise in training on long sequences – will soon see some of them

RNN Variants



The first variant is called "teacher forcing"

The true label at time t is available to hidden layers at time $t+1$ as an input

At test time since $\mathbf{y}^t$ is not available, $\hat{\mathbf{y}}^t$ passed instead

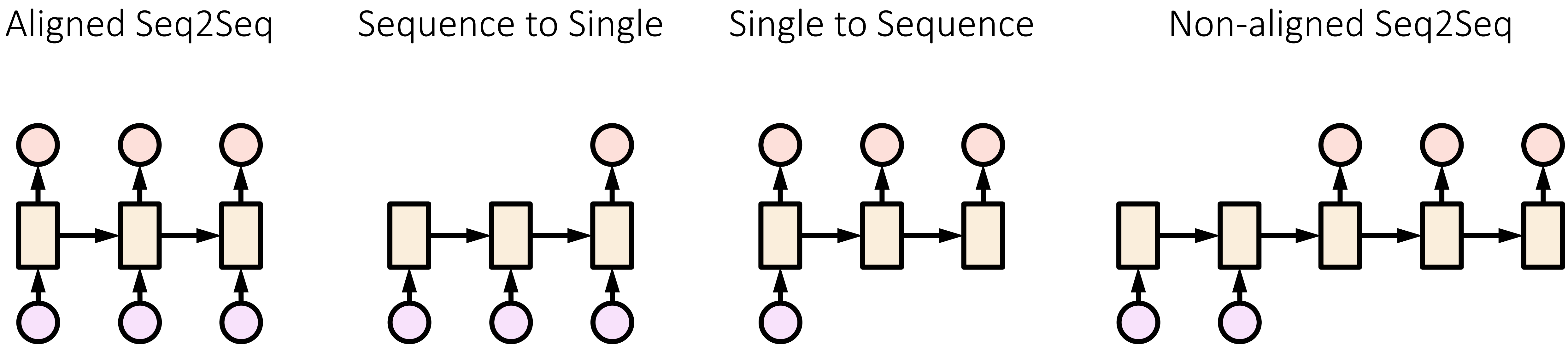
The second variant is called *attention* mechanism

Very powerful and popular. It is "all you need"!

Usually a separate NN used to select a subset $S_t \subseteq [T]$ (T is length of seq) such that hidden states $\mathbf{h}_t, t \in S_t$ are useful in predicting $\mathbf{y}^t$

Idea stems from machine translation where sentences in different languages may reorder words

Applications of RNNs



POS tagging, predicting next word, language model learning, labelling frames of a video

Sentiment analysis, video/document classification

Image captioning

Machine translation, query rewriting, error correction in input seq

Backprop with RNNs

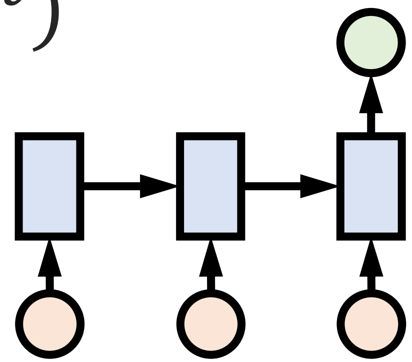
A bit tricky since the network is essentially replicated across time

Hence have to do "Backpropagation Through Time" (BPTT)

Lets look at sequence to single prediction with a single hidden neuron

We have $\hat{y} = \langle \mathbf{v}, \mathbf{h}^T \rangle$, and $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1}) \triangleq f(\mathbf{z}^t)$

Need to be very careful about chain rule now



$$\frac{d\ell}{d\mathbf{v}} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{v}} = \ell'(\hat{y}) \cdot \mathbf{h}^T$$

$$\frac{d\ell}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{h}^T} \cdot \frac{d\mathbf{h}^T}{dW} = \ell'(\hat{y}) \cdot \mathbf{v} \cdot \frac{d\mathbf{h}^T}{dW}$$

$$\frac{d\mathbf{h}^T}{dW} = \frac{d\mathbf{h}^T}{d\mathbf{z}^t} \cdot \frac{d\mathbf{z}^t}{dW} = J^f_{\mathbf{z}^t} \cdot \left(\mathbf{x}^t + U \cdot \frac{d\mathbf{h}^{T-1}}{dW} \right) = \cdots$$

Backprop with RNNs

Notice that

$$\frac{d\mathbf{h}^T}{dW} = J^f_{\mathbf{z}^T} \cdot U \cdot \frac{d\mathbf{h}^{T-1}}{dW} + \text{blah} = J^f_{\mathbf{z}^T} \cdot U \cdot J^f_{\mathbf{z}^{T-1}} \cdot U \cdot \frac{d\mathbf{h}^{T-2}}{dW} + \text{blah}$$

$$= J^f_{\mathbf{z}^T} \cdot U \cdot J^f_{\mathbf{z}^{T-1}} \cdot U \cdot J^f_{\mathbf{z}^{T-2}} \cdot U \cdot \frac{d\mathbf{h}^{T-3}}{dW} + \text{blah}$$

Perfect recipe for gradients to either blow up or vanish entirely

Solution 1*: Unitary RNN (Arjovsky et al ICML 2016)*

Force U to have singular values around 1 so that neither blowup nor vanishing happens

Solution 2*: not allow this chain of $J^f_{\mathbf{z}^T} \cdot U \cdot J^f_{\mathbf{z}^{T-1}} \cdot U \cdots$ to continue for long*

Gated Recurrent Units, LSTMs, echo networks, skip connections, leaky units. Use "gates" to force network to forget data that appeared long ago in the series. LSTM (Hochreiter and Schmidhuber 1997): long-short term memory