



**QUEEN'S
UNIVERSITY
BELFAST**

CSC4007 Advanced Machine Learning

Lesson 08: Recurrent Neural Networks

by Vien Ngo
EEECS / ECIT / DSSC


Outline

- Neural network basics and representation
- Perceptron learning, multi-layer perceptron
- Neural network training: Backpropagation
- Modern neural network architecture (a.k.a Deep learning):
 - Convolutional neural network (CNN)
 - **Recurrent neural network (RNN), long-short term memory network (LSTM)**

Language modelling: predicting the next word in a sequence of words

We need a way to model the word-by-word “surprisal”.

More specifically, we need to estimate a probability distribution over possible continuation words, at each point in the sentence.

This probability may depend on **the full history of previous words** 

“The ...

“The horse ...

“The horse raced ...

“The horse raced past ...

“The horse raced past the ...

“The horse raced past the barn ...

“The horse raced past the barn fell .”

“The horse raced past the barn **fell**”

vs

“The horse shot in the barn **fell**”

Examples of sequence modelling / prediction problem domains

Machine translation

Speech Recognition

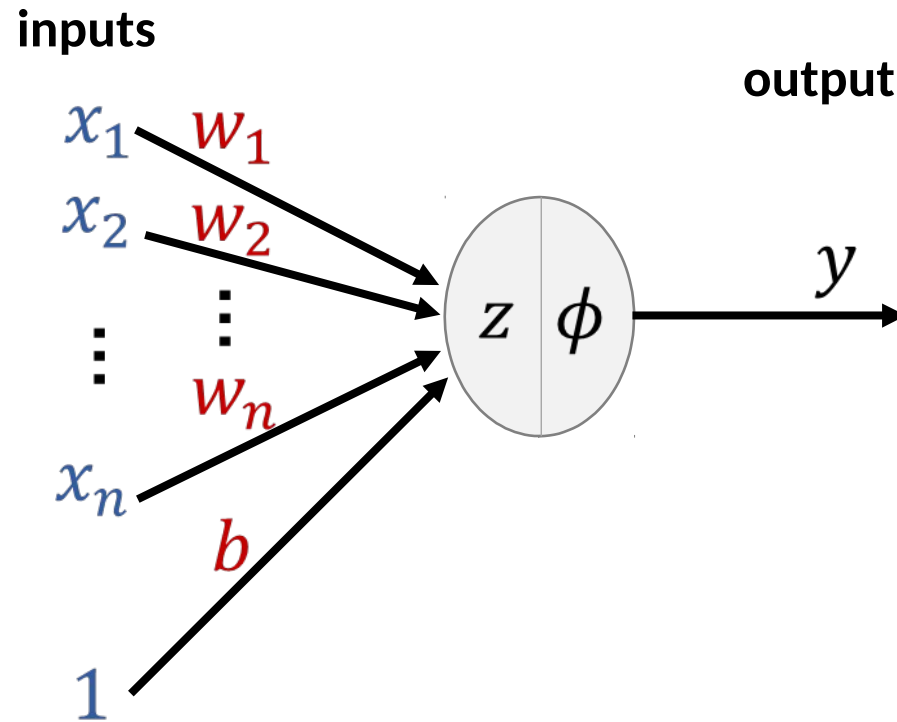
Grammar learning

(Online) handwriting recognition

Time-series prediction (e.g. stock-market prices)

Sequence modelling with Recurrent Neural Networks

Recall our model of a neuron:



Activation flows in 1 direction: inputs -> outputs

Calculating the input, z :

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

$b = \text{bias}$

$w = \text{weights}$

Calculating the output, y :

$$y = \phi(z) = \phi \left(\sum_{i=1}^n w_i \cdot x_i + b \right)$$

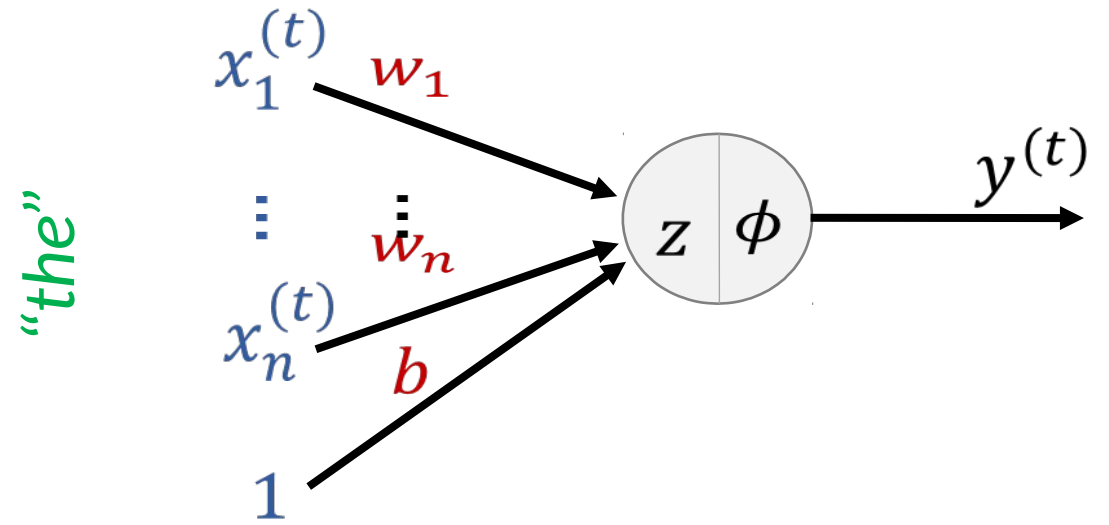
where ϕ is a non-linear activation function
(e.g. sigmoid, ReLU)

Sequence modelling with Recurrent Neural Networks

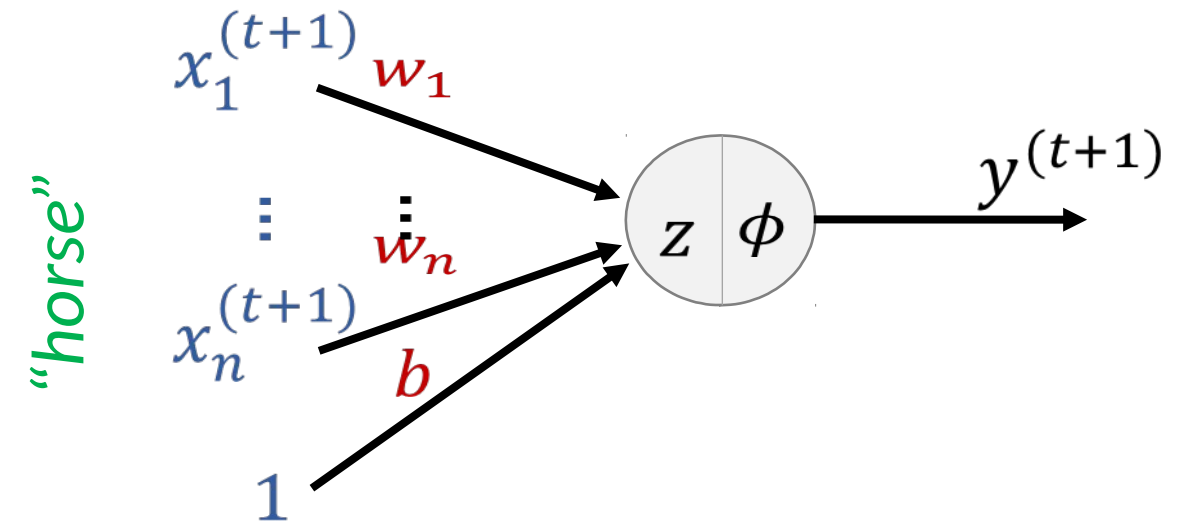
- Suppose we have a sequence of inputs – i.e. different input for each timepoint, t
- We will keep the weights the same for every timepoint, and compute a new output for each input.
- *But we want the output at time $t + 1$ to also depend on the previous input. How might we model this?*



Time t :



Time $t + 1$:



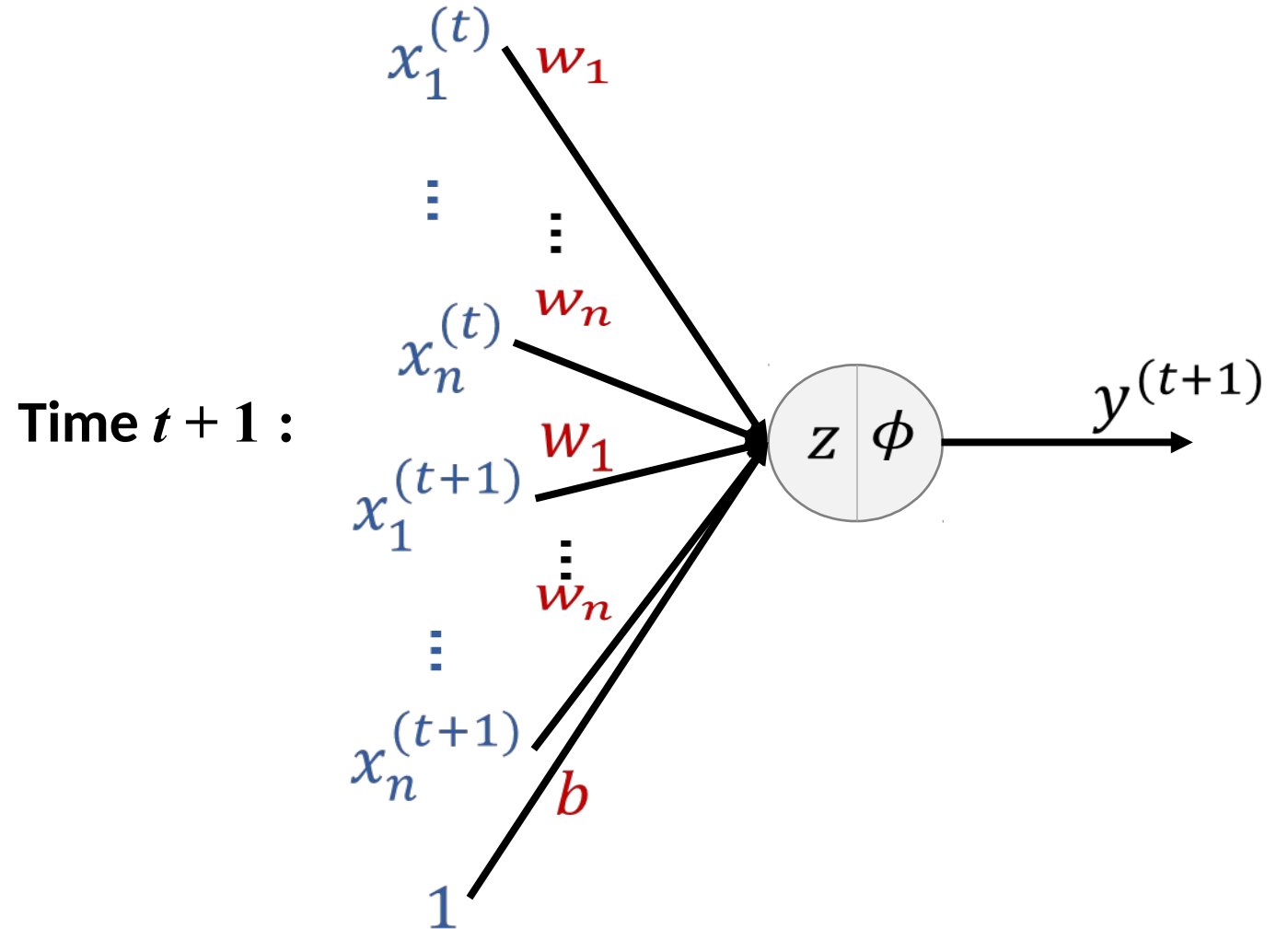
Sequence modelling with Recurrent Neural Networks

One approach: give the neuron inputs from both timepoints.

Although you will sometimes see models like this, they cannot look back very far: this one only looks back one or few timestep.

We have seen that we sometimes have to look back very far:

The horse left in the meadow jumped but the horse raced passed the barn fell



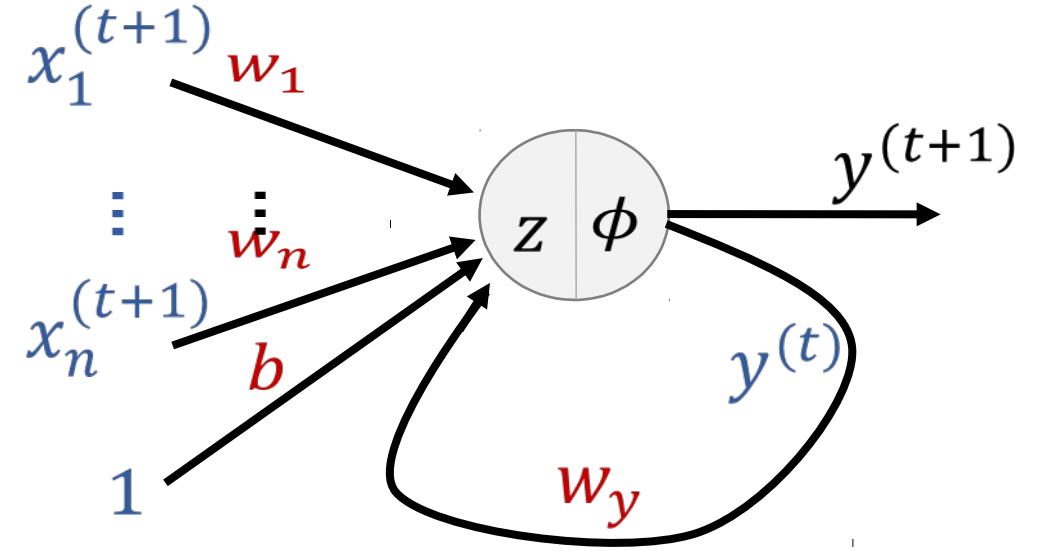
Sequence modelling with Recurrent Neural Networks

A better approach: use the output from time t as an input to time $t+1$.

Now, the output at time t will depend on the output at every previous timestep, $t-1$, $t-2$, etc.

*We have already seen something similar with deep networks: the **input** to a neuron in a layer comes from the **output** of the neurons in the previous layer.*

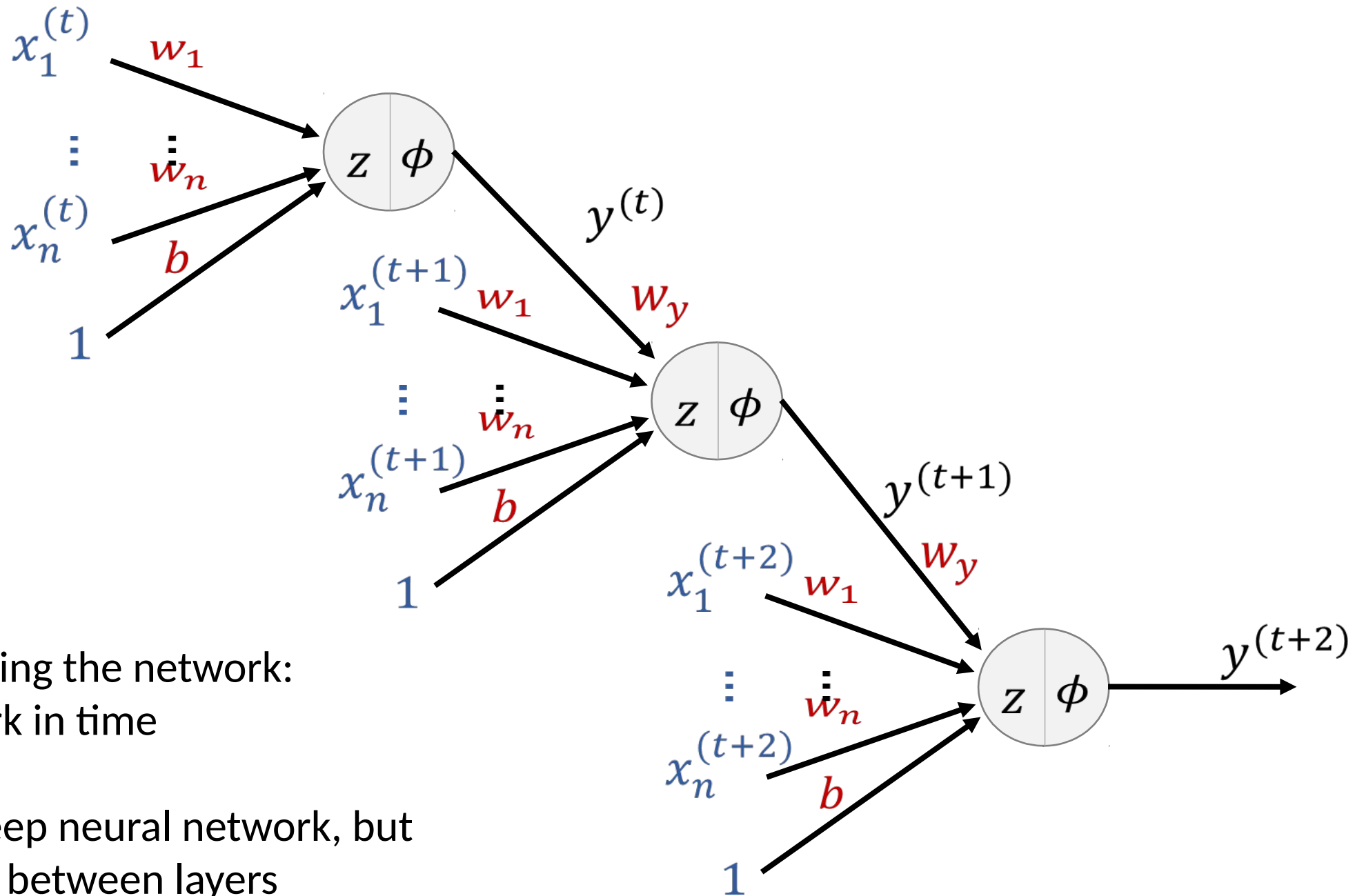
Time $t + 1$:



Calculating the input, z , at each timestep:

$$z^{(t+1)} = w_y y^{(t)} + \sum_{i=1}^n w_i x_i^{(t+1)} + b$$

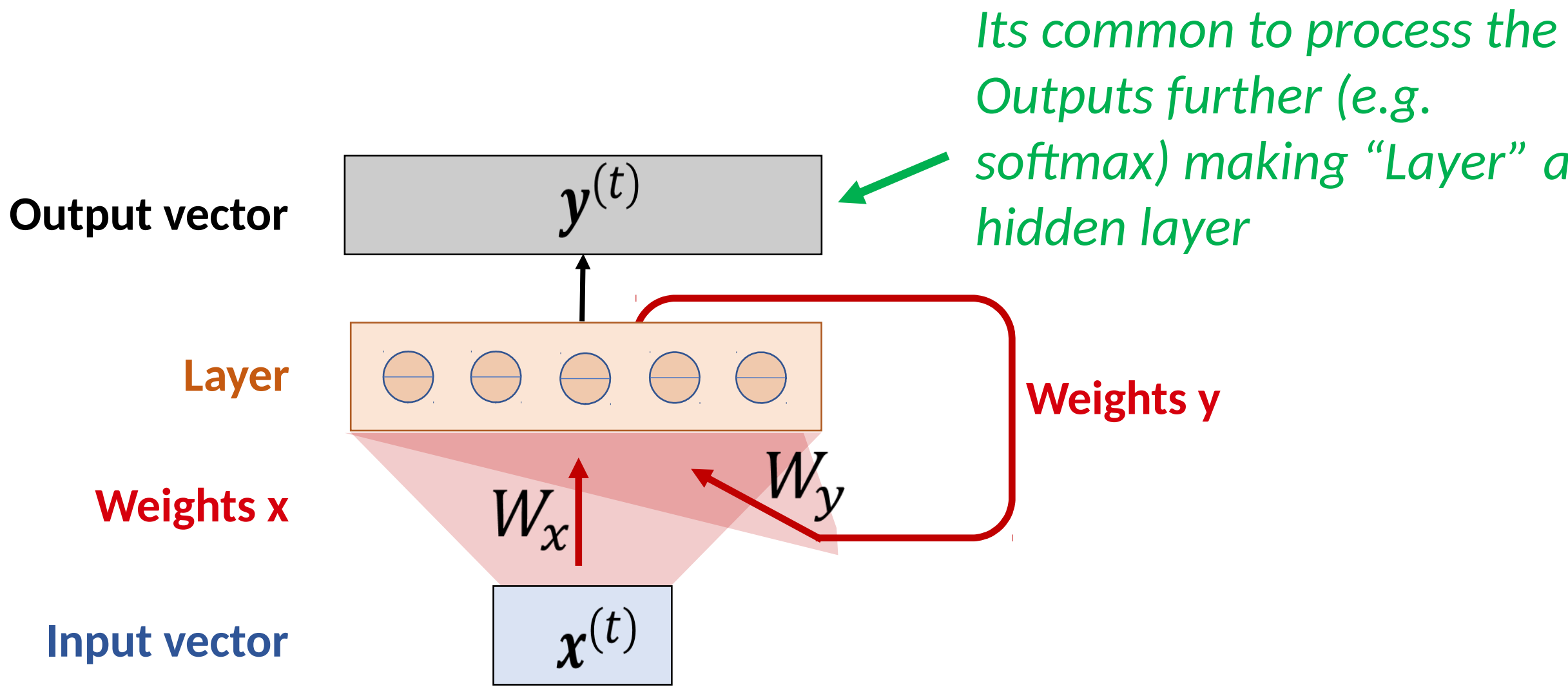
Sequence modelling with Recurrent Neural Networks



Another way of viewing the network:
unrolling the network in time

This is similar to a deep neural network, but
with **shared weights** between layers

Simple Recurrent Neural Network architecture



Simple Recurrent neural network architecture

Forward pass calculation:

$$Y^{(t)} = \phi \left(\underbrace{X^{(t)} \cdot W_x + b}_{\text{Input from input vector}} + \underbrace{Y^{(t-1)} \cdot W_y}_{\text{Input from previous output}} \right)$$

Input from input vector

Input from previous output

Training a Simple Recurrent neural network

- As we have seen, a recurrent NN is equivalent to a deep feedforward NN.
We just have to “unroll” the recurrent NN in time.
- For training, we backpropagate errors through the graph, exactly like we saw with feedforward NNs, called **BPTT** (**Backpropagation Through Time**).
- The training signal is back-propagated through the network; through several non-linearities, its utility tends to diminish the further back you go. **This is called the vanishing gradient problem.**

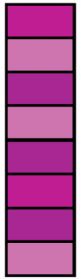
LSTM: Long Short Term Memory cells

- The solution to the vanishing gradient problem is to use **long-short-term memory cells, proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber**
- LSTMs are more complex hidden layer units (i.e. not simple nonlinear functions) that can explicitly keep track of different kinds of information over arbitrarily long periods of time.
- Gated recurrent units (GRUs) is like LSTM, introduced in 2014 by Kyunghyun Cho et al. (with fewer parameters)

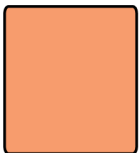
LSTM: Long Short Term Memory cells

However, as well as hidden and input representations, they contain an explicit memory component:

$memory_{i-1}$



There are different slots in this memory, which can represent information about what is in the preceding context.



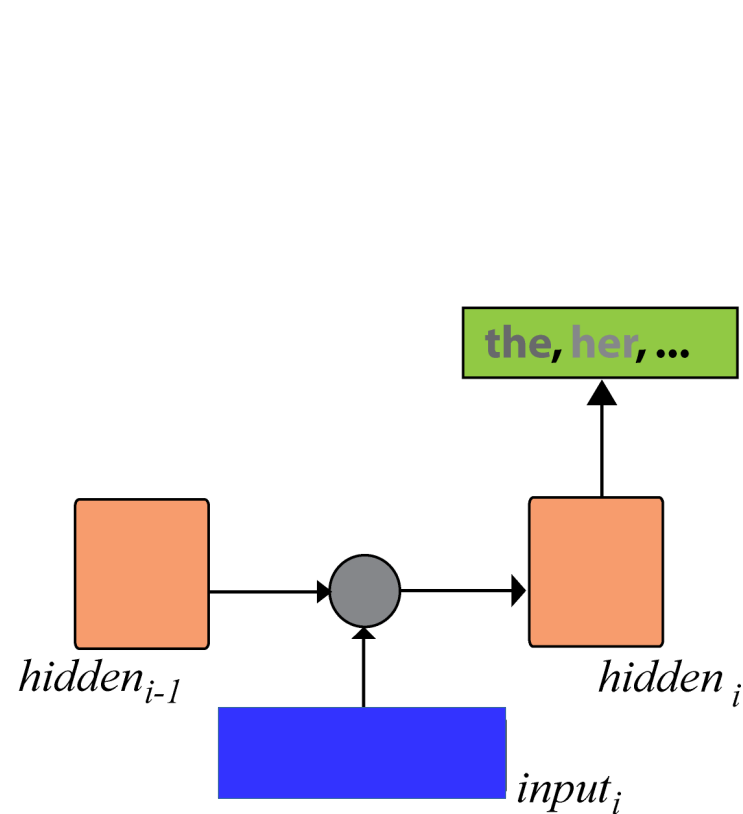
$hidden$



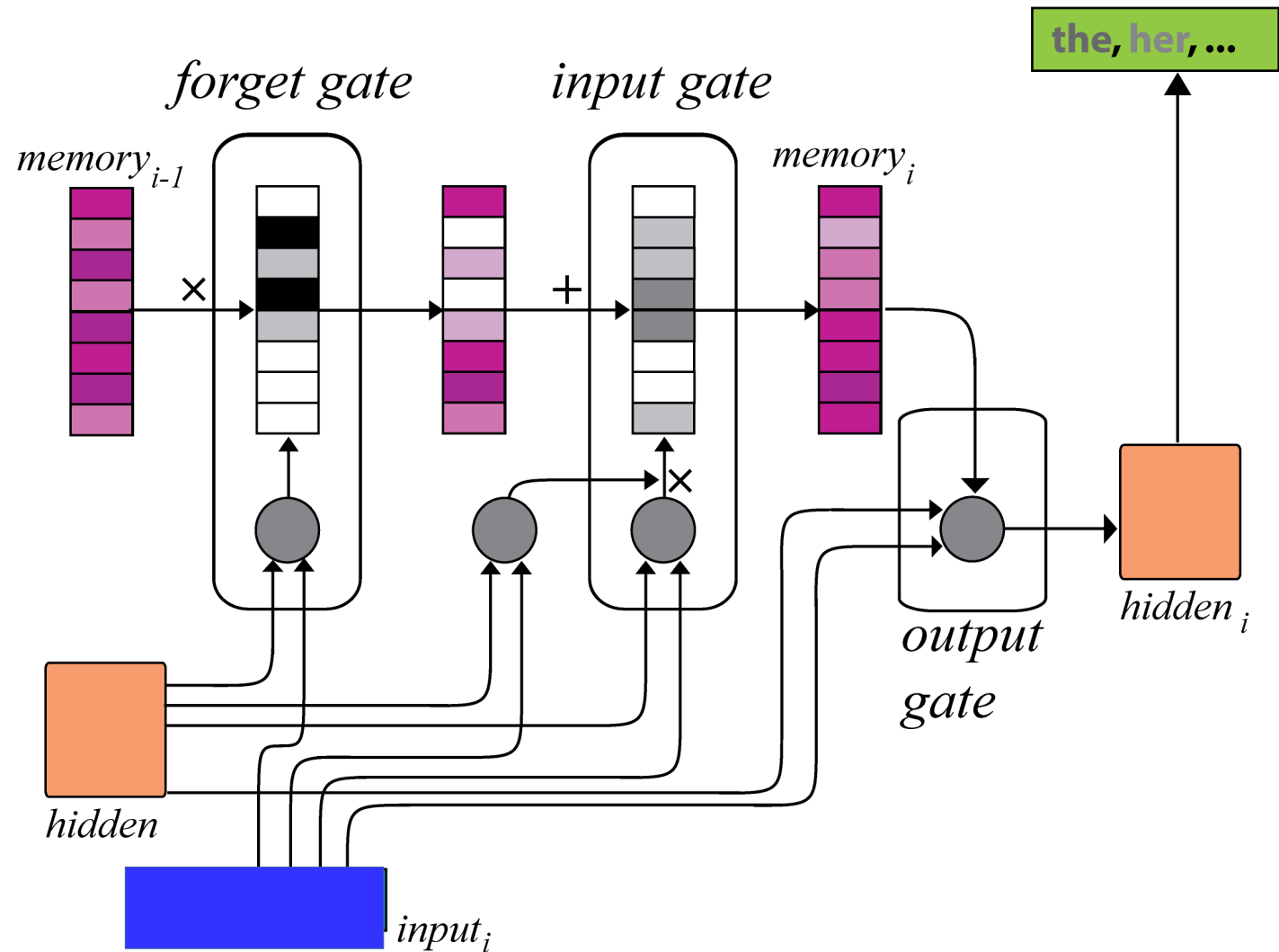
$input_i$

LSTM: Long Short Term Memory cells

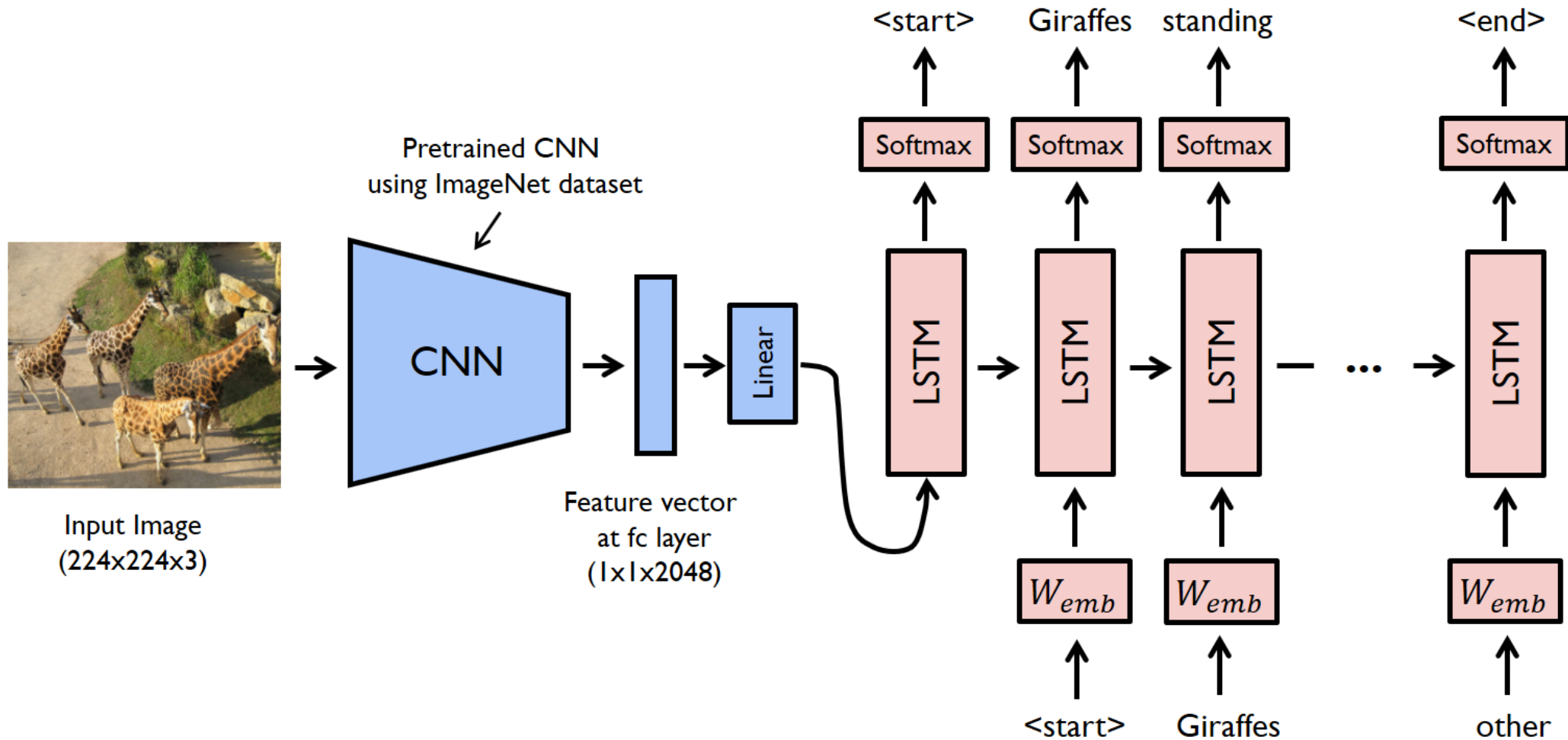
Simple RNN layer



LSTM layer

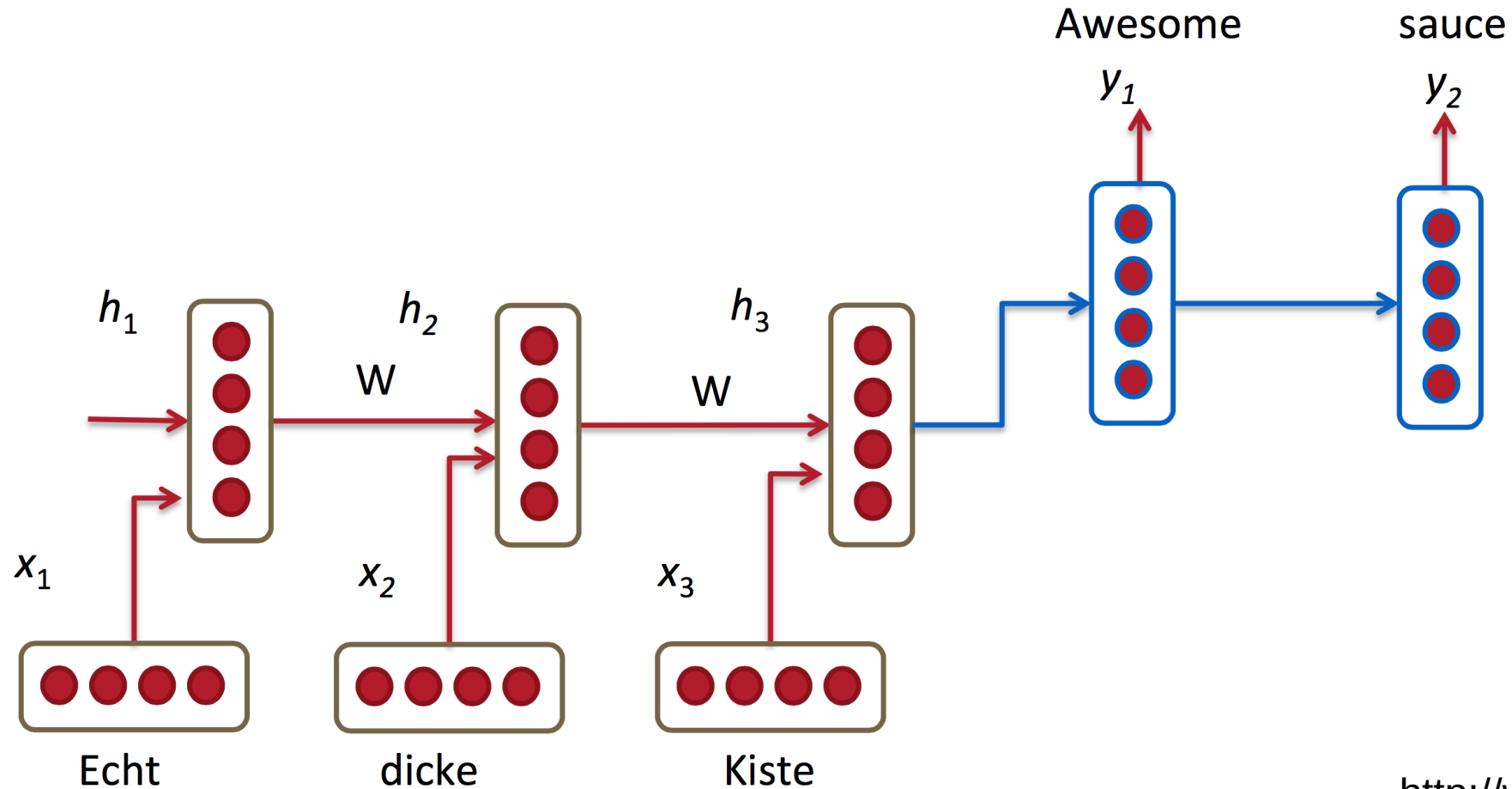


One-to-Many RNN: Image Captions

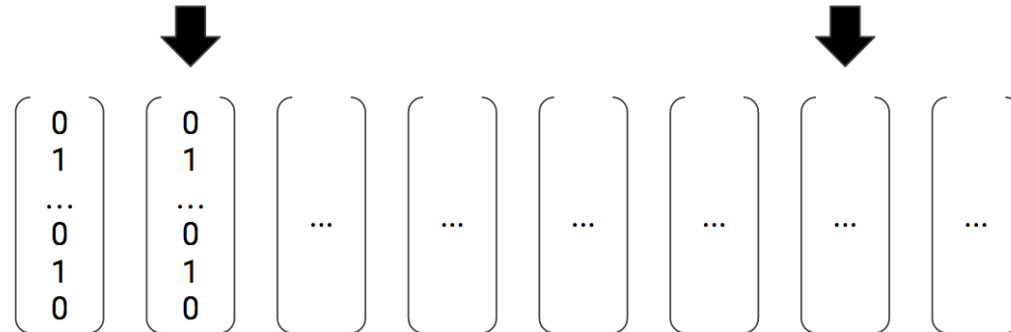
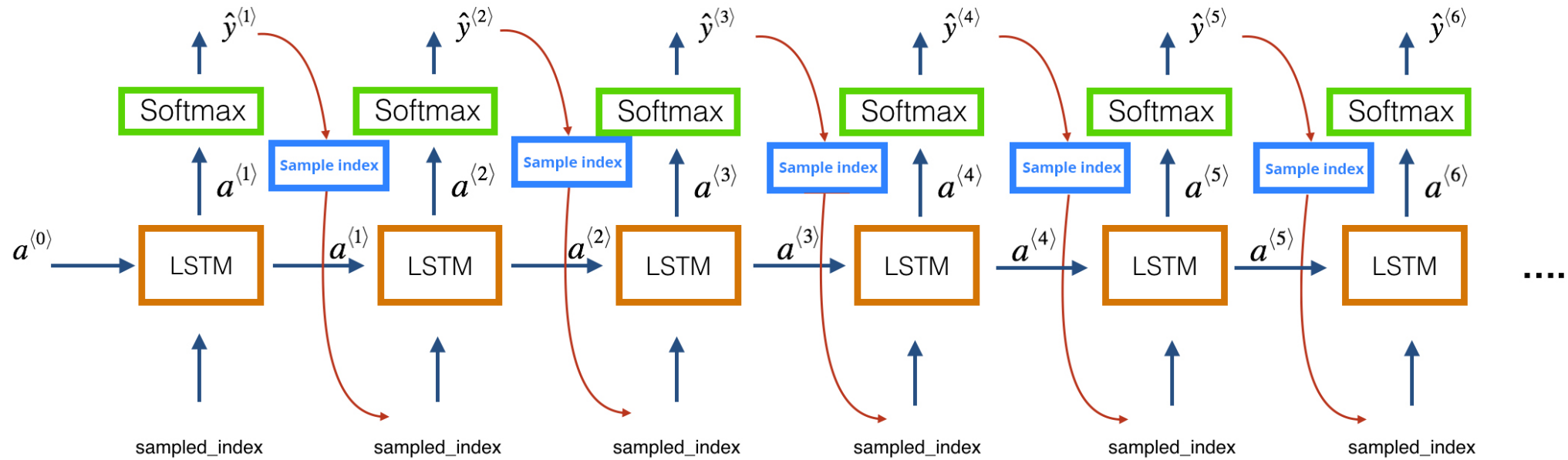


Example: Many-to-Many RNN

E.g: Machine Translation



One-to-Many RNN: Music Generation



RNN: Summary

- Recurrent Neural Networks (RNN) for sequences
- Backpropagation Through Time
- Vanishing and Exploding Gradients and Remedies
- RNNs using Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU)
- Applications of Recurrent Neural Networks