Process Sequences

VLC = ∫∫∫ Vision Learning Control dVdLdC

Recurrent Neural Networks

Jonathon Hare

Vision, Learning and Control
University of Southampton

A lot of the ideas in this lecture come from Andrej Karpathy's blog post on the Unreasonable Effectiveness of RNNs (`https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b`). Many of the images and animations and were made by Adam Prügel-Bennett.

Recurrent Neural Networks - Motivation

x:	Jon	and	Ethan	gave	deep	learning	lectures
y:	1	0	1	0	0	0	0

Recurrent Neural Networks - Motivation

x:	$x^{(1)}$	...	$x^{(t)}$	...	$x^{(T_x)}$
x:	Jon	...	Ethan	...	lectures
y:	$y^{(1)}$	...	$y^{(t)}$	...	$y^{(T_y)}$
y:	1	...	1	...	0

In this example, $T_x = T_y = 7$ but T_x and T_y can be different.

Recurrent Neural Networks

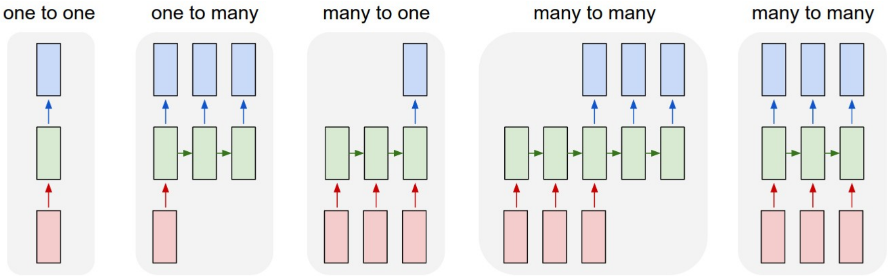


Image from `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Aside: One Hot Encoding

How can we represent individual words (or other discrete tokens)?

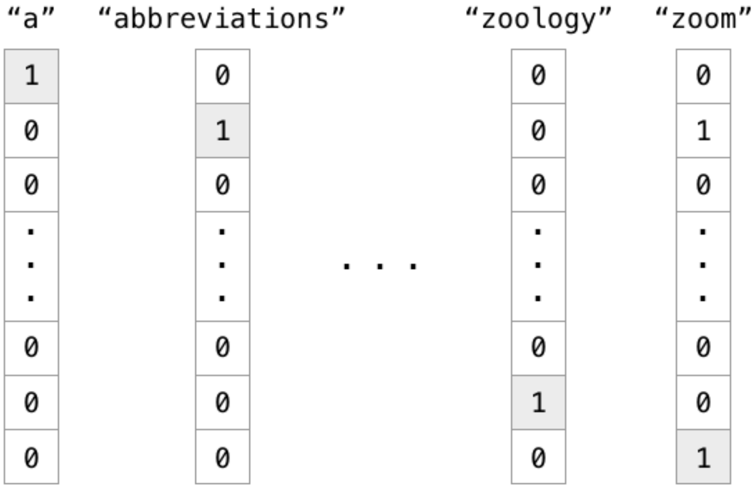


Image from https://ayearofai.com

Why Not a Standard Feed Forward Network?

- For a task such as “Named Entity Recognition” a MLP would have several disadvantages

Why Not a Standard Feed Forward Network?

- For a task such as "Named Entity Recognition" a MLP would have several disadvantages
  - The inputs and outputs may have varying lengths

Why Not a Standard Feed Forward Network?

- For a task such as "Named Entity Recognition" a MLP would have several disadvantages
    - The inputs and outputs may have varying lengths
    - The features wouldn't be shared across different temporal positions in the network

Why Not a Standard Feed Forward Network?

- For a task such as "Named Entity Recognition" a MLP would have several disadvantages
  - The inputs and outputs may have varying lengths
  - The features wouldn't be shared across different temporal positions in the network
    - Note that 1-D convolutions can be (and are) used to address this, in addition to RNNs - more on this in a later lecture

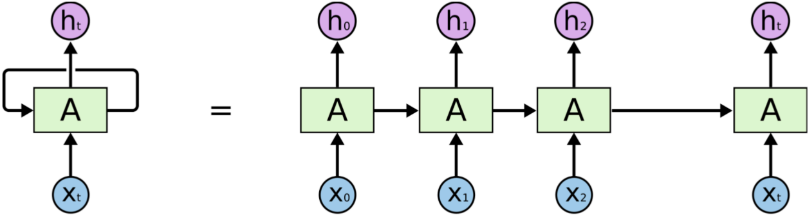
Why Not a Standard Feed Forward Network?

- For a task such as "Named Entity Recognition" a MLP would have several disadvantages
  - The inputs and outputs may have varying lengths
  - The features wouldn't be shared across different temporal positions in the network
    - Note that 1-D convolutions can be (and are) used to address this, in addition to RNNs - more on this in a later lecture
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past

Why Not a Standard Feed Forward Network?

- For a task such as "Named Entity Recognition" a MLP would have several disadvantages
  - The inputs and outputs may have varying lengths
  - The features wouldn't be shared across different temporal positions in the network
    - Note that 1-D convolutions can be (and are) used to address this, in addition to RNNs - more on this in a later lecture
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time

Recurrent Neural Networks

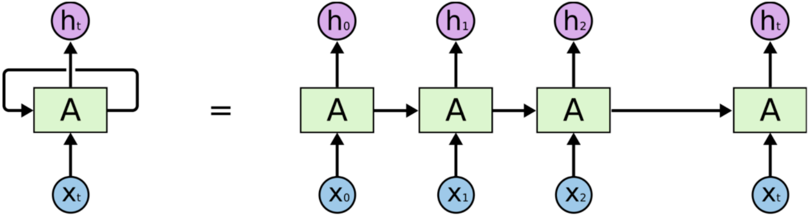


[1]

- RNNs are a family of ANNs for processing sequential data

[1] Image taken from `https://towardsdatascience.com`

Recurrent Neural Networks



[1]

- RNNs are a family of ANNs for processing sequential data
- RNNs have directed cycles in their computational graphs

[1] Image taken from `https://towardsdatascience.com`

Recurrent Neural Networks

RNNs combine two properties which make them very powerful.

1. Distributed hidden state that allows them to store a lot of information about the past efficiently. This is because several different units can be active at once, allowing them to remember several things at once.
2. Non-linear dynamics that allows them to update their hidden state in complicated ways[2].

[2] Often said to be difficult to train, but this is not necessarily true - dropout can help with overfitting for example

Recurrent Neural Networks

RNNs are Turing complete in the sense they can simulate arbitrary programs[3].

[3]Don't read too much into this - like universal approximation theory, just because they can doesn't mean its necessarily learnable!

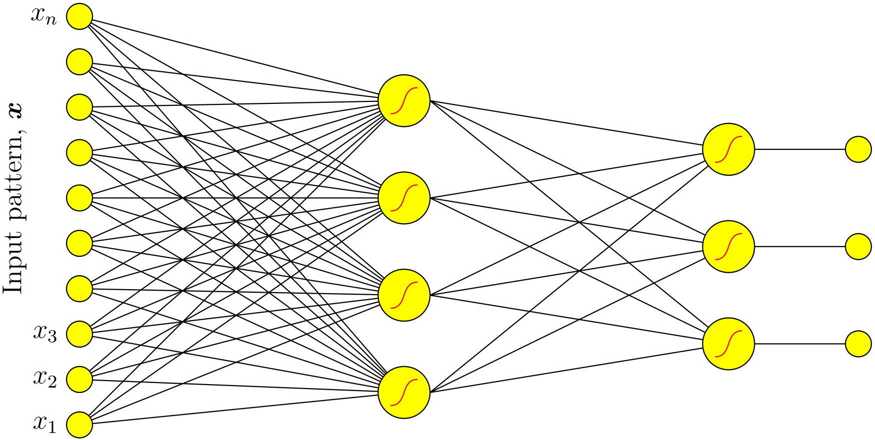
Recurrent Neural Networks

RNNs are Turing complete in the sense they can simulate arbitrary programs[3].

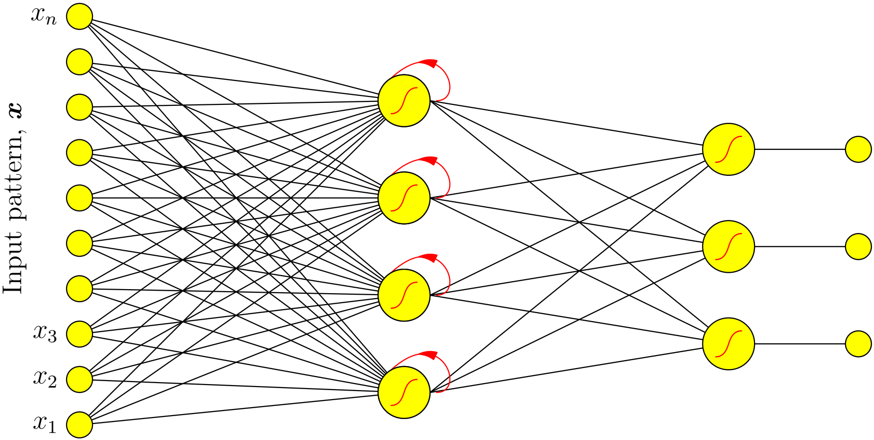
If training vanilla neural nets is optimisation over functions, training recurrent nets is optimisation over programs.

[3] Don't read too much into this - like universal approximation theory, just because they can doesn't mean its necessarily learnable!

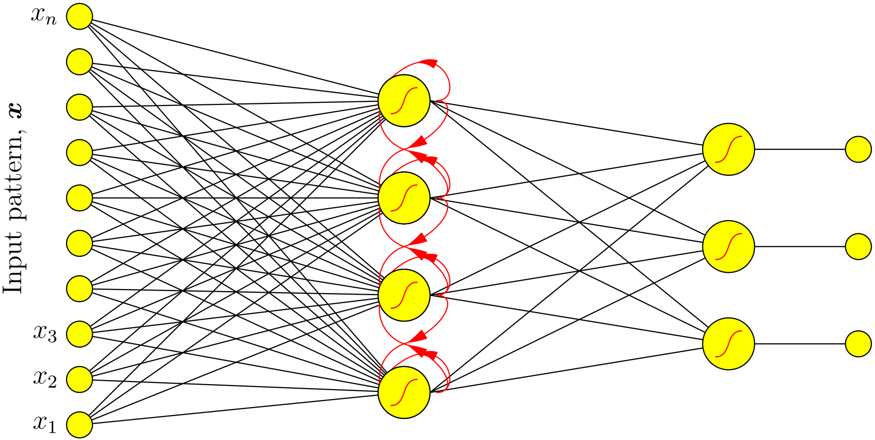
Recurrent Network



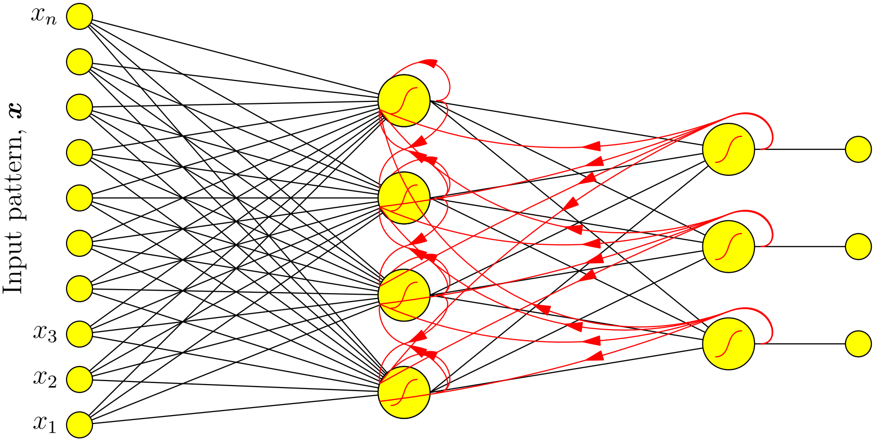
Recurrent Network



Recurrent Network



Recurrent Network

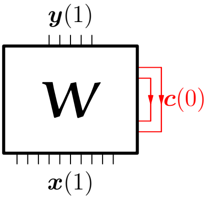


Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \big((\boldsymbol{x}(t), \boldsymbol{y}(t))\big| t = 1, 2, \ldots, T\big)$

Training Recurrent Networks

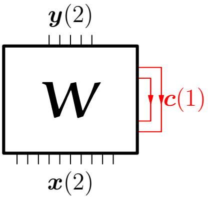
- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t)) \middle| t = 1,\, 2,\, \ldots,\, T\right)$



- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \|\boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})\|^2$$

- Given a set of inputs $\mathcal{D} = \big((\boldsymbol{x}(t), \boldsymbol{y}(t))\big| t = 1, 2, \ldots, T\big)$

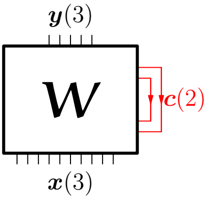


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1) | \boldsymbol{W}) \right\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \big((\boldsymbol{x}(t), \boldsymbol{y}(t))\big| t = 1, 2, \ldots, T\big)$

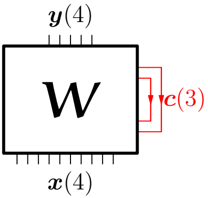


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1) | \boldsymbol{W}) \right\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$

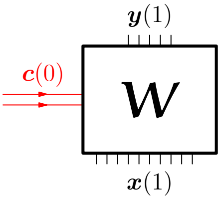


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \|\boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t)) \middle| t = 1,\, 2,\, \ldots,\, T \right)$

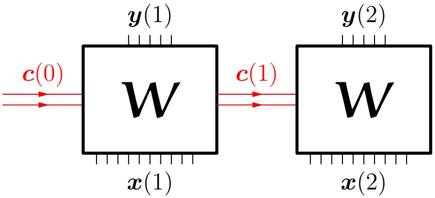


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1) | \boldsymbol{W}) \right\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \big((\boldsymbol{x}(t), \boldsymbol{y}(t))\big| t = 1, 2, \ldots, T\big)$

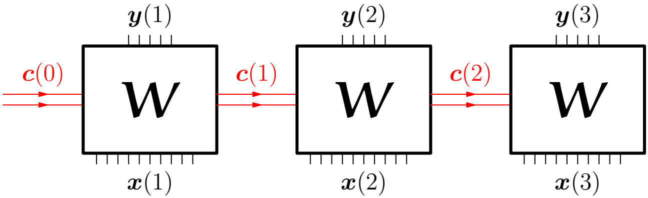


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1) | \boldsymbol{W}) \right\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \big((\boldsymbol{x}(t), \boldsymbol{y}(t))\big| t = 1, 2, \ldots, T\big)$

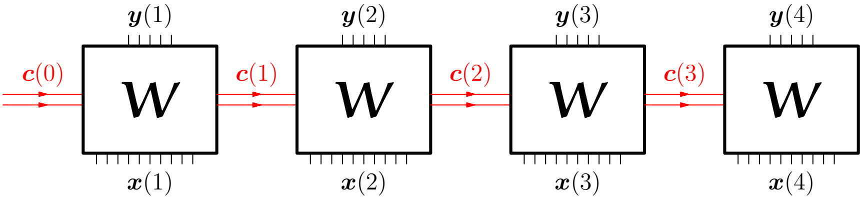


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \|\boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t)) \middle| t = 1, 2, \ldots, T\right)$

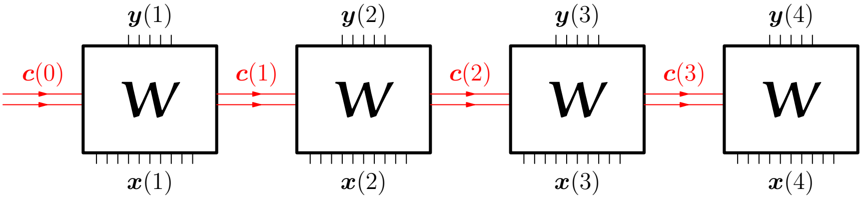


- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \|\boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})\|^2$$

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



- Minimise an error (here MSE, but your choice):

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \|\boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})\|^2$$

- This is known as *back-propagation through time*

An RNN is just a recursive function invocation

- $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})$

An RNN is just a recursive function invocation

- $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- and the state $\boldsymbol{c}(t) = \boldsymbol{g}(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$

An RNN is just a recursive function invocation

- $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- and the state $\boldsymbol{c}(t) = \boldsymbol{g}(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{g}(\boldsymbol{x}(t-1), \boldsymbol{g}(\boldsymbol{x}(t-2), \boldsymbol{g}(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

An RNN is just a recursive function invocation

- $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- and the state $\boldsymbol{c}(t) = \boldsymbol{g}(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{g}(\boldsymbol{x}(t-1), \boldsymbol{g}(\boldsymbol{x}(t-2), \boldsymbol{g}(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- it should be clear that the gradients of this with respect to the weights can be found with the chain rule

What is the state update $g()$?

- It depends on the variant of the RNN!
  - Elman
  - Jordan
  - LSTM
  - GRU

Elman Networks ("Vanilla RNNs")

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{b}_{ih} + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hh}) \\
\boldsymbol{y}_t &= \sigma_y(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y)
\end{aligned}
$$

Elman Networks ("Vanilla RNNs")

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{b}_{ih} + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hh}) \\
\boldsymbol{y}_t &= \sigma_y(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y)
\end{aligned}
$$

- σ_h is usually tanh
- σ_h is usually identity (linear) – the y's could be regressed values or logits
- the state $\boldsymbol{h}_t$ is referred to as the "hidden state"
- the output at time t is a projection of the hidden state at that time

Elman Networks ("Vanilla RNNs")

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{b}_{ih} + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hh}) \\
\boldsymbol{y}_t &= \sigma_y(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y)
\end{aligned}
$$

- σ_h is usually tanh
- σ_h is usually identity (linear) – the y's could be regressed values or logits
- the state $\boldsymbol{h}_t$ is referred to as the "hidden state"
- the output at time t is a projection of the hidden state at that time
- the hidden state at time t is a summation of a projection of the input and a projection of the previous hidden state

Going deep: Stacking RNNs

Example: Character-level language modelling

Sampling the Language Model