

# Process Sequences

# 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 (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>). Many of the images and animations were made by Adam Prügel-Bennett.

# Recurrent Neural Networks - Motivation

$x$ : Jon and Antonia gave deep learning lectures

$y$ : 1 0 1 0 0 0 0

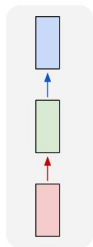
# Recurrent Neural Networks - Motivation

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

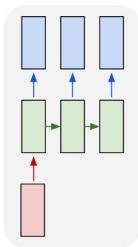
In this example,  $T_x = T_y = 7$  but  $T_x$  and  $T_y$  can be different.

# Recurrent Neural Networks

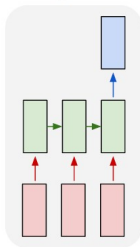
one to one



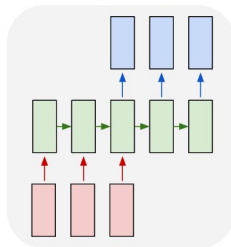
one to many



many to one



many to many



many to many

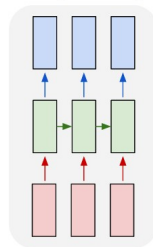


Image from <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

## Aside: One Hot Encoding

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

"a"	"abbreviations"		"zoology"	"zoom"
1	0		0	0
0	1		0	1
0	0		0	0
.	.	...	.	.
.	.		.	.
.	.		.	.
0	0		0	0
0	0		1	0
0	0		0	1

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

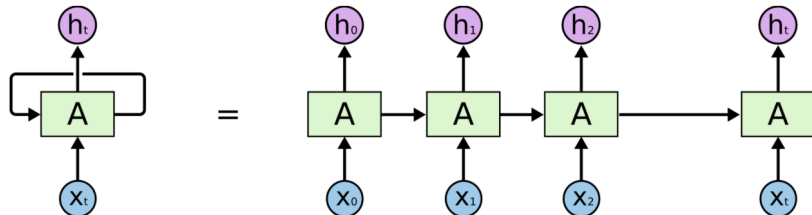
# Why Not a Standard Feed Forward Network?

- 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

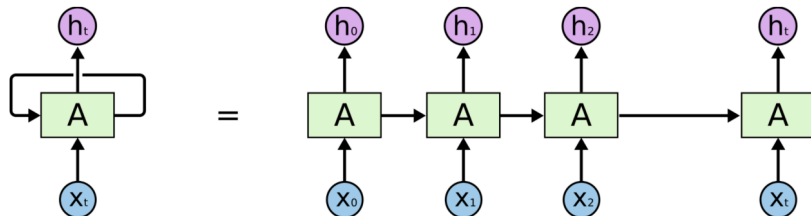
# Recurrent Neural Networks



- RNNs are a family of ANNs for processing sequential data

Image taken from <https://towardsdatascience.com>

# Recurrent Neural Networks



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

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<sup>1</sup>.

---

<sup>1</sup>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<sup>2</sup>.

---

<sup>2</sup>Don't read too much into this - like universal approximation theory, just because they can doesn't mean its necessarily learnable!



# Recurrent Neural Networks

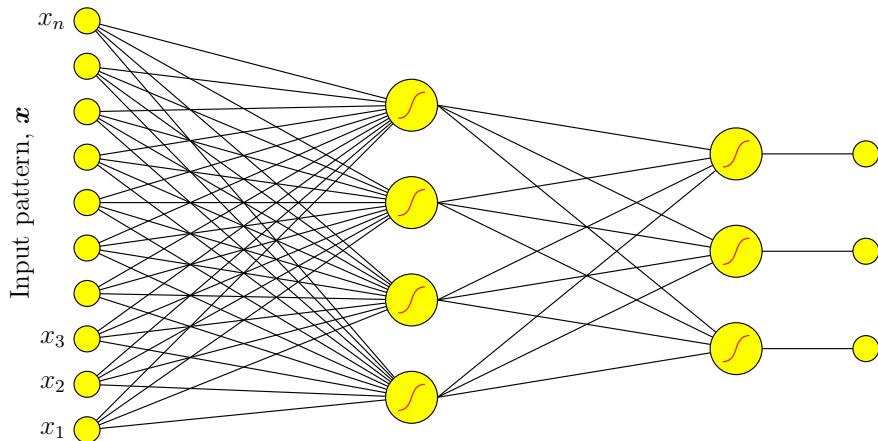
RNNs are Turing complete in the sense they can simulate arbitrary programs<sup>2</sup>.

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

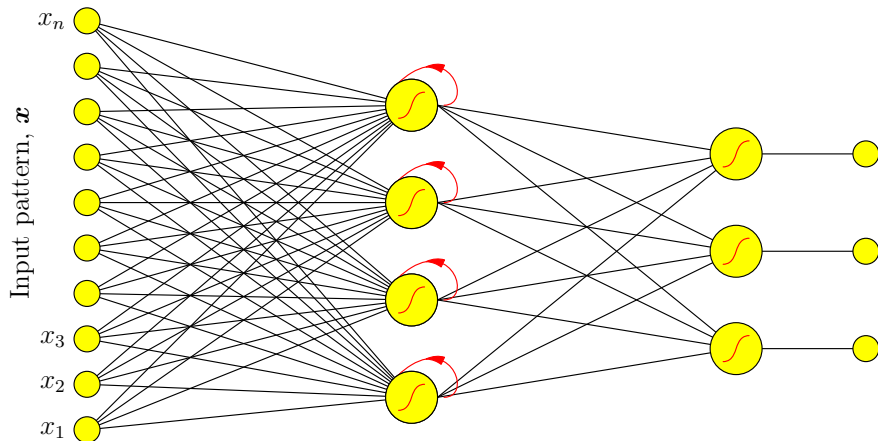
---

<sup>2</sup>Don't read too much into this - like universal approximation theory, just because they can doesn't mean its necessarily learnable!

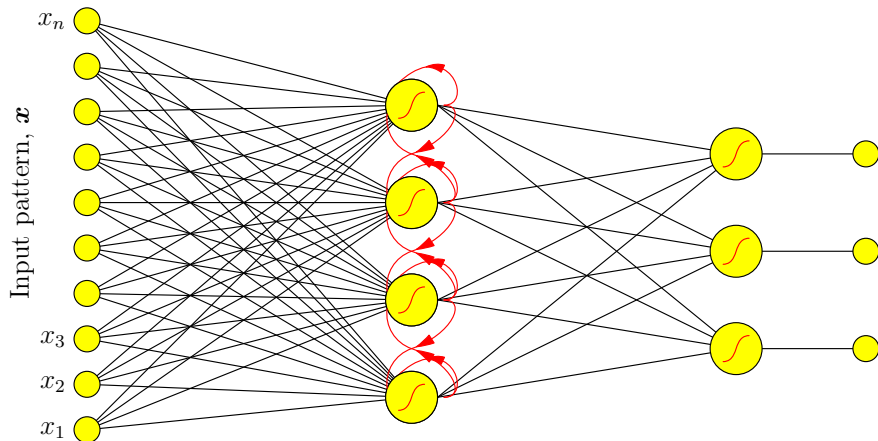
# Recurrent Network



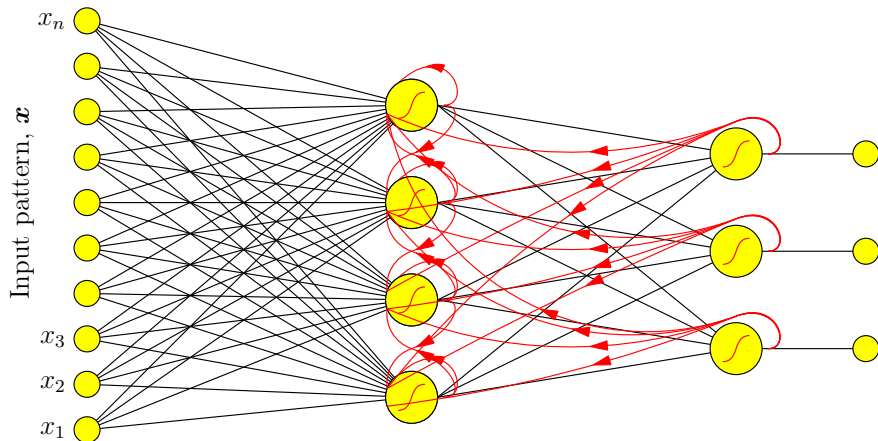
# Recurrent Network



# Recurrent Network



# Recurrent Network

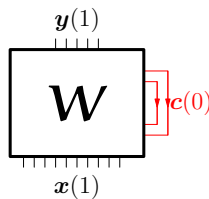


# Training Recurrent Networks

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

# Training Recurrent Networks

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

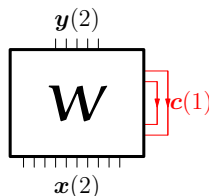


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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



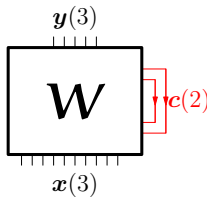
- Minimise an error (here MSE, but your choice):

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



# Training Recurrent Networks

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

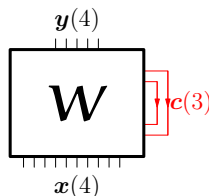


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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

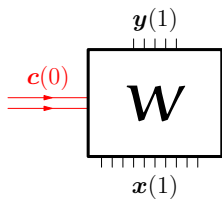


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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

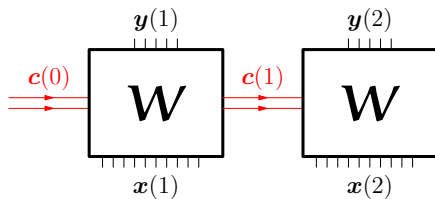


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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

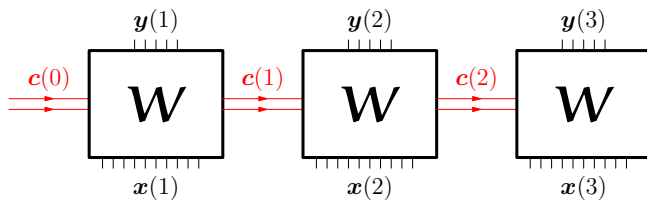


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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

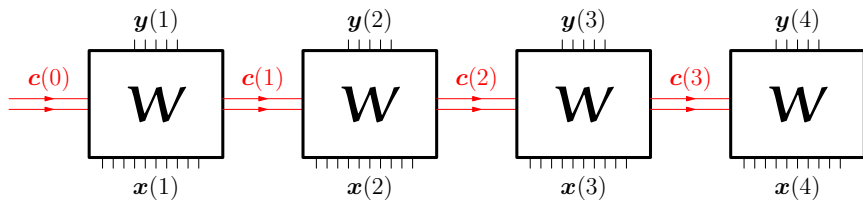


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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

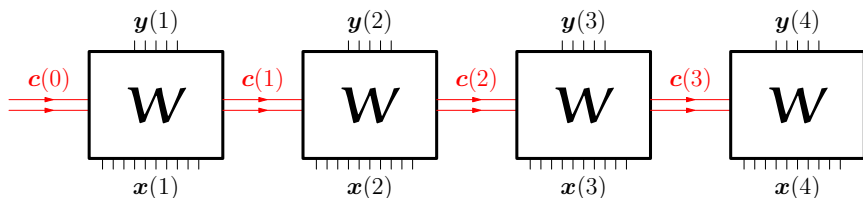


- Minimise an error (here MSE, but your choice):

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

# Training Recurrent Networks

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



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

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

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

# An RNN is just a recursive function invocation

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



# An RNN is just a recursive function invocation

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

# An RNN is just a recursive function invocation

- $\mathbf{y}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{c}(t-1) | \mathbf{W})$
- and the state  $\mathbf{c}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{c}(t-1) | \mathbf{W})$
- If the output  $\mathbf{y}(t)$  depends on the input  $\mathbf{x}(t-2)$ , then prediction will be

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

# An RNN is just a recursive function invocation

- $\mathbf{y}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{c}(t-1) | \mathbf{W})$
- and the state  $\mathbf{c}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{c}(t-1) | \mathbf{W})$
- If the output  $\mathbf{y}(t)$  depends on the input  $\mathbf{x}(t-2)$ , then prediction will be

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

- 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” )

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

# Elman Networks (“Vanilla RNNs”)

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_{ih}\mathbf{x}_t + \mathbf{b}_{ih} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_{hh})$$
$$\mathbf{y}_t = \sigma_y(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y)$$

- $\sigma_h$  is usually tanh
- $\sigma_y$  is usually identity (linear) – the  $y$ ’s could be regressed values or logits
- the state  $\mathbf{h}_t$  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”)

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_{ih}\mathbf{x}_t + \mathbf{b}_{ih} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_{hh})$$
$$\mathbf{y}_t = \sigma_y(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y)$$

- $\sigma_h$  is usually tanh
- $\sigma_y$  is usually identity (linear) – the  $y$ ’s could be regressed values or logits
- the state  $\mathbf{h}_t$  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

- RNNs can be trivially stacked into deeper networks



# Going deep: Stacking RNNs

- RNNs can be trivially stacked into deeper networks
- It's just function composition:

$$\mathbf{y}(t) = \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}(t), \mathbf{c}_2(t-1)|\mathbf{W}_1), \mathbf{c}_2(t-1)|\mathbf{W}_2)$$

# Going deep: Stacking RNNs

- RNNs can be trivially stacked into deeper networks
- It's just function composition:

$$\mathbf{y}(t) = \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}(t), \mathbf{c}_2(t-1)|\mathbf{W}_1), \mathbf{c}_2(t-1)|\mathbf{W}_2)$$

- The output of the inner RNN at time  $t$  is fed into the input of the outer RNN which produces the prediction  $y$

# Going deep: Stacking RNNs

- RNNs can be trivially stacked into deeper networks
- It's just function composition:

$$\mathbf{y}(t) = \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}(t), \mathbf{c}_2(t-1)|\mathbf{W}_1), \mathbf{c}_2(t-1)|\mathbf{W}_2)$$

- The output of the inner RNN at time  $t$  is fed into the input of the outer RNN which produces the prediction  $y$
- Also note: RNNs are most often not used in isolation - it's quite common to process the inputs and outputs with MLPs (or even convolutions)

## Example: Character-level language modelling

- We'll end with an example: an RNN that learns to 'generate' English text by learning to predict the next character in a sequence

## Example: Character-level language modelling

- We'll end with an example: an RNN that learns to 'generate' English text by learning to predict the next character in a sequence
- This is "Character-level Language Modelling"

# Example: Character-level language modelling

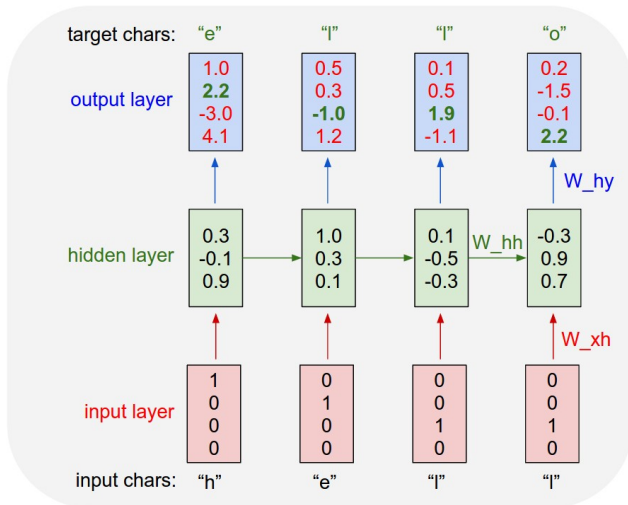


Image from <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Training a Char-RNN

- The training data is just text data (e.g. sequences of characters)
- The task is unsupervised (or rather self-supervised): given the previous characters predict the next one
  - All you need to do is train on a reasonable sized corpus of text

# Training a Char-RNN

- The training data is just text data (e.g. sequences of characters)
- The task is unsupervised (or rather self-supervised): given the previous characters predict the next one
  - All you need to do is train on a reasonable sized corpus of text
  - Overfitting could be a problem: dropout is very useful here



# Sampling the Language Model

- Once the model is trained what can you do with it?
- if you feed it an initial character it will output the logits of the next character

# Sampling the Language Model

- Once the model is trained what can you do with it?
- if you feed it an initial character it will output the logits of the next character
- you can use the logits to select the next character and feed that in as the input character for the next timestep

# Sampling the Language Model

- Once the model is trained what can you do with it?
- if you feed it an initial character it will output the logits of the next character
- you can use the logits to select the next character and feed that in as the input character for the next timestep
- how do you 'sample' a character from the logits?

# Sampling the Language Model

- Once the model is trained what can you do with it?
- if you feed it an initial character it will output the logits of the next character
- you can use the logits to select the next character and feed that in as the input character for the next timestep
- how do you 'sample' a character from the logits?
  - you could pick the most likely (maximum-likelihood solution), but this might lead to generated text with very low variance (it might be boring and repetitive)

# Sampling the Language Model

- Once the model is trained what can you do with it?
- if you feed it an initial character it will output the logits of the next character
- you can use the logits to select the next character and feed that in as the input character for the next timestep
- how do you 'sample' a character from the logits?
  - you could pick the most likely (maximum-likelihood solution), but this might lead to generated text with very low variance (it might be boring and repetitive)
  - you could treat the softmax probabilities defined by the logits as a categorical distribution and sample from them
    - you might increase the 'temperature',  $T$ , of the softmax to make the distribution more diverse (less 'peaky'):  $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$

A lot of the ideas in this lecture on the input  $c(t-1)$ ,  $g(x_i(x(t-2), g(t-1) - W)$

used in the "Monte Carlo Net" to predict the probability

of a word given the previous words in the sentence. This is used in the paper "Learning to Predict the Next Word in a Sentence" by Peter F. Smith, 1994. The next step is to use the state at time

- Sampled from a single layer RNN<sup>3</sup>.

---

<sup>3</sup>LSTM, 128 dim hidden size, with linear input projection to 8-dimensions and output to the number of characters (84). Trained on the text of these slides for 50 epochs.