

# COSC343: Artificial Intelligence

## Lecture 11 : Recurrent neural networks

Lech Szymanski

Dept. of Computer Science, University of Otago

Lech Szymanski (Otago)

COSC343 Lecture 11

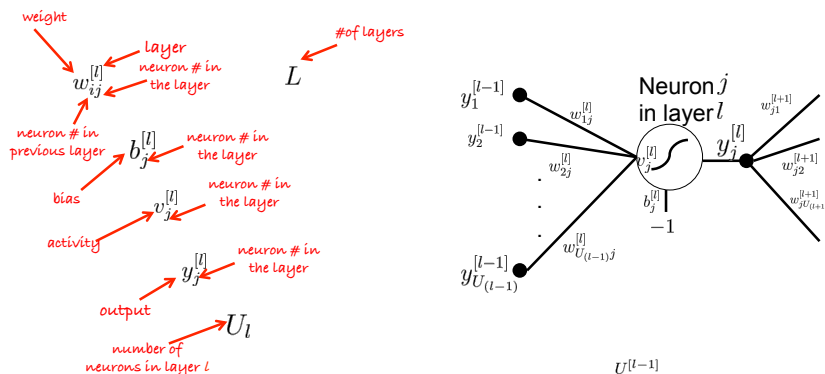
## In today's lecture

- Examples of neural networks in action
- Softmax function – network output as a probability distribution
- Simple Recurrent Network (SRN)
- Teaching SRN to talk

Lech Szymanski (Otago)

COSC343 Lecture 11

## Recap: Notation



- Activity is the weighted sum of inputs from the previous layer minus the bias

$$v_j^{[l]} = \sum_{i=1}^{U^{[l-1]}} w_{ij}^{[l]} y_i^{[l-1]} - b_j^{[l]}$$

- Output is a function of activity

$$y_j^{[l]} = f_{\text{logsig}}(v_j^{[l]}) = \frac{1}{1 + \exp^{-v_j^{[l]}}}$$

where  $y_i^{[0]} = x_i$  and  $U_0 = M$

Lech Szymanski (Otago)

COSC343 Lecture 10

## Recap: Backpropagation

Let's assume that we are minimising some cost function that depends on the network output and the desired output

$$J(y_j^{[L]}, \tilde{y}_j)$$

Steepest gradient descent update for weight connecting input  $i$  with neuron  $j$  in layer  $l$  is:

$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \Delta w_{ij}^{[l]}$$

Steepest gradient descent update for bias on neuron  $j$  in layer  $l$  is:

$$b_j^{[l]} = b_j^{[l]} - \alpha \Delta b_j^{[l]}$$

where...

The change in the weight connecting input  $i$  with neuron  $j$  in layer  $l$  is:

$$\Delta w_{ij}^{[l]} = y_i^{[l-1]} \frac{dy_j^{[l]}}{dv_j^{[l]}} \delta J_j^{[l]}$$

The change in the bias for neuron  $j$  in layer  $l$  is:

$$\Delta b_j^{[l]} = - \frac{dy_j^{[l]}}{dv_j^{[l]}} \delta J_j^{[l]}$$

where...

The cost blame for neuron  $i$  in layer  $l-1$  is:

$$\delta J_i^{[l-1]} = \sum_{j=1}^{U_l} w_{ij}^{[l]} \frac{dy_j^{[l]}}{dv_j^{[l]}} \delta J_j^{[l]}$$

The cost blame for neuron  $j$  in the output layer is a derivative of the overall cost with respect to the neuron's output:

$$\delta J_j^{[L]} = \frac{dJ(y_j^{[L]}, \tilde{y}_j)}{dy_j^{[L]}}$$

Sigmoid function has a derivative

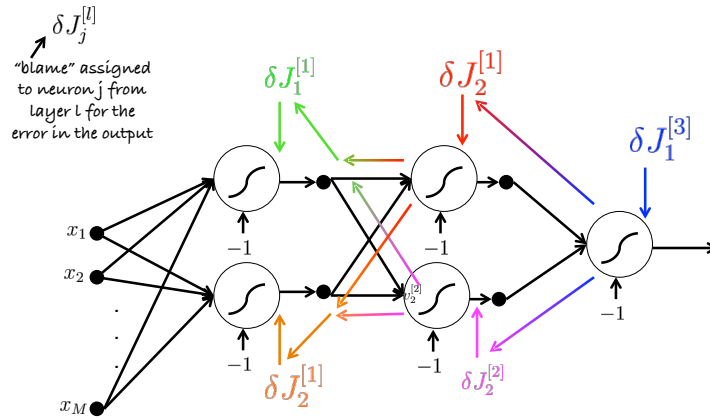
$$y_j^{[l]} = f_{\text{logsig}}(v_j^{[l]}) = \frac{1}{1 + \exp^{-v_j^{[l]}}}$$

$$\frac{\partial y_j^{[l]}}{\partial v_j^{[l]}} = y_j^{[l]}(1 - y_j^{[l]})$$

Lech Szymanski (Otago)

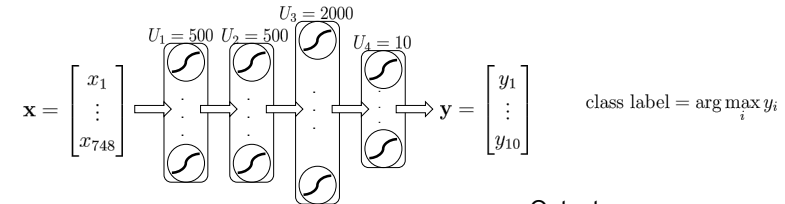
COSC343 Lecture 10

## Recap: Backpropagation



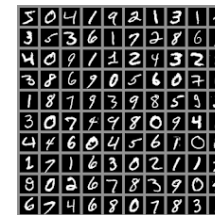
## An example: an artificial neural network for classification

748-500-500-2000-10 neural network:



Input:

- Each image is 28x28 pixels
- Digits are normalised for size, position and orientation.



Output:

- One-zero vector code for digit identified in the image

Mean squared error

Training with backpropagation of MSE:

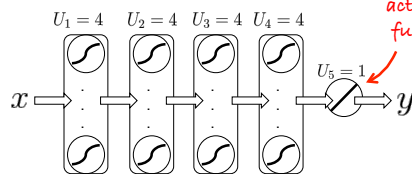
- 60000 images
- 2.5% classification error

Testing:

- 10000 images
- 3% training error

## An example: an artificial neural network for regression

1-4-4-4-1 neural network:

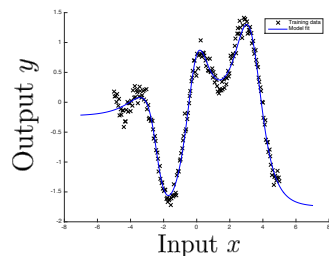


Input:

- Value between -5 and 5

Output:

- Inferred function of input x



Training with backpropagation of MSE:

- 100 samples
- RMSE=0.72

Root mean squared error

Testing:

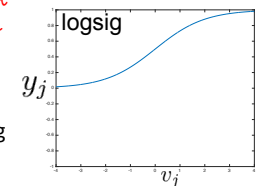
- 100 samples
- RMSE=0.77

## Selection of activation functions

- Logistic sigmoid

$$y_j = \frac{1}{1 + e^{-v_j}}, \quad \frac{dy_j}{dv_j} = y_j(1 - y_j)$$

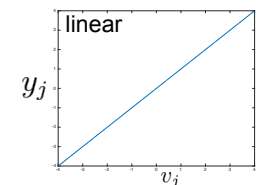
- Output bounded between 0 and 1
- Useful for classification, hidden units, and interpreting output as a probability



- Linear

$$y_j = v_j, \quad \frac{dy_j}{dv_j} = 1$$

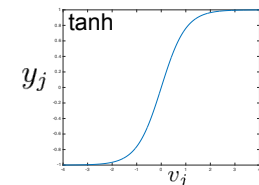
- Output not bounded
- Useful for output neurons in regression tasks



- Hyperbolic tangent

$$y_j = \tanh(v_j), \quad \frac{dy_j}{dv_j} = (1 - y_j^2)$$

- Output bounded between -1 and 1
- Sometimes gives richer hidden layer representation than logistic sigmoid



## Network output as a probability distribution

A neural network with  $K$  outputs can represent probability distribution of a discrete random variable with  $K$  possible outcomes

- Most commonly used for classification, where each output represents probability of classifying the input a belonging to one of  $K$  possible labels:

$$y_j^{[L]} = p(\text{label} = c_j), \text{ where } \text{label} \in \{c_1, \dots, c_K\}.$$

- Probability distribution at the output is computed with the softmax function

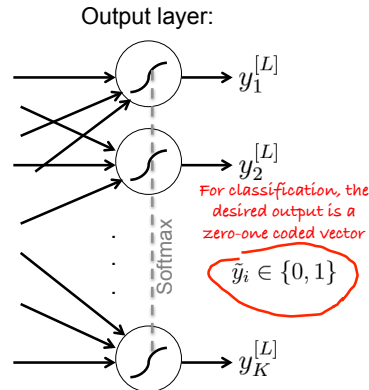
$$y_j^{[L]} = \frac{e^{v_j}}{\sum_{k=1}^K e^{v_k}} \text{ for which the derivative}$$

$$\text{is } \frac{dy_j^{[L]}}{dv_j^{[L]}} = y_j^{[L]}(1 - y_j^{[L]})$$

This is all you need to backpropagate the softmax function

- The training error at the output is  $\delta J_j^{[L]} = \tilde{y}_j(y_j^{[L]} - 1) + (1 - \tilde{y}_j)y_j^{[L]}$ , which

corresponds to minimisation of the following cost:  $J = -\sum_j (\tilde{y}_j \ln y_j^{[L]} + (1 - \tilde{y}_j) \ln(1 - y_j^{[L]}))$



## An example: Classification with softmax

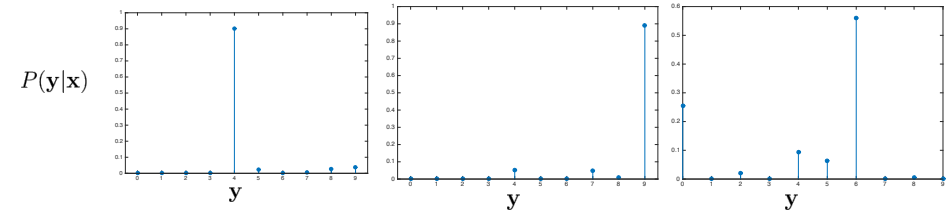
A 784-500-500-2000-10 neural network trained with a the softmax function to recognise digit in an image.



For the following test inputs,







the network produces the following probability distributions over 10 possible labels:



## What about temporal patterns?

So far we have only considered learning tasks where the order of inputs (during training and testing) was of no consequence

- E.g. In digit recognition, input  should be recognised as a "2" regardless whether the previously processed input was a , , or .

What about a language related task?

- E.g. If we wanted to train a neural network to predict the next word in a sentence....the prediction depends on the context:

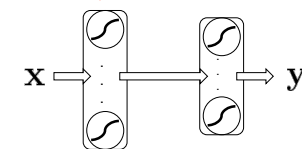
"Dogs are ..."

"Cars are ..."

## Feed forward and recurrent neural networks

A **feed forward neural network** is a directed acyclic graph:

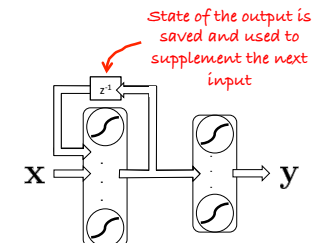
- The internal state of the network depends only on its current input, which allows the model to perform static input-output mapping.



Networks we were looking at up to this point

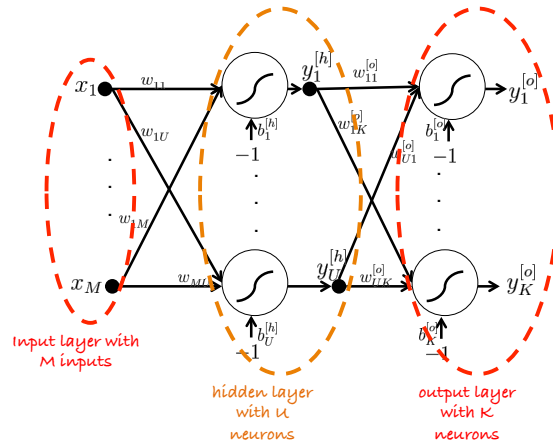
A **recurrent neural network** is a directed graph with cycles (loops):

- The internal state of the network depends on its previous state, which allows the model to exhibit a dynamic temporal behaviour.



## Simple Recurrent Network (SRN)

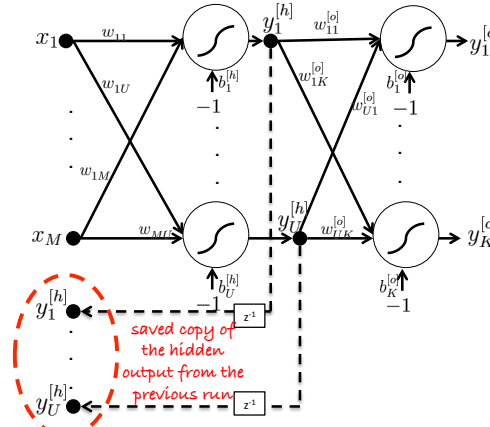
- Also referred to as *Elman network* for its inventor Jeff Elman



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs  $U$  - # of hidden neurons  $K$  - # of outputs

## Simple Recurrent Network (SRN)

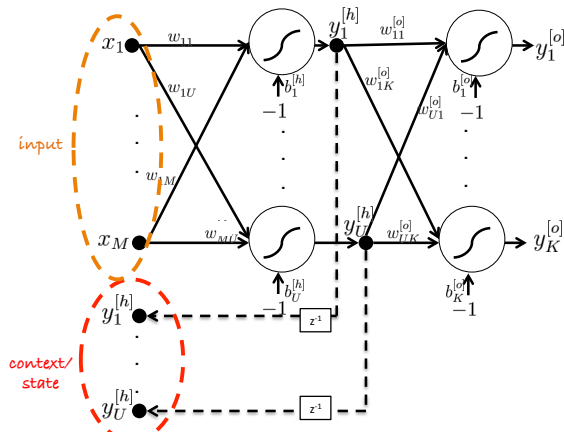
- Output of the hidden layer feeds into the output layer as well as the hidden layer.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs  $U$  - # of hidden neurons  $K$  - # of outputs

## Simple Recurrent Network (SRN)

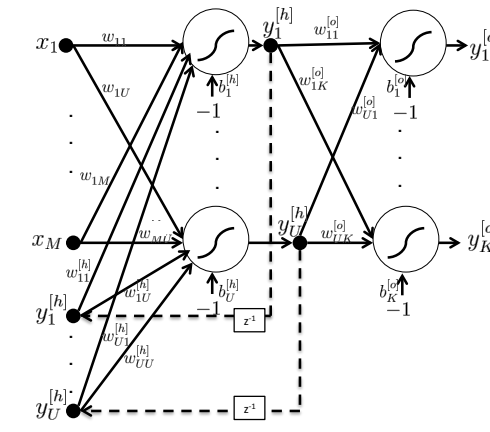
- Output of the hidden layer feeds into the output layer as well as the hidden layer.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs  $U$  - # of hidden neurons  $K$  - # of outputs

## Simple Recurrent Network (SRN)

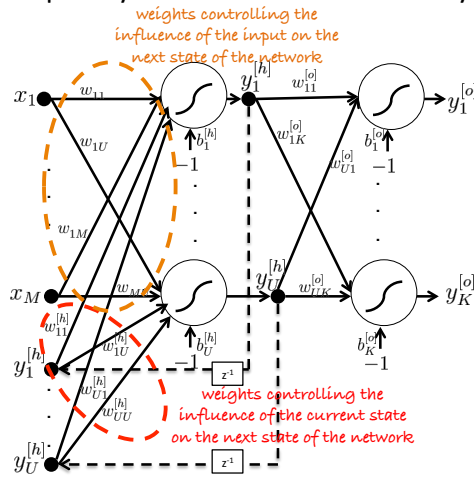
- Output of the hidden layer feeds into the output layer as well as the hidden layer.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[h]}$  - weight connecting saved output of neuron  $i$  from the hidden layer to neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs  $U$  - # of hidden neurons  $K$  - # of outputs

## Simple Recurrent Network (SRN)

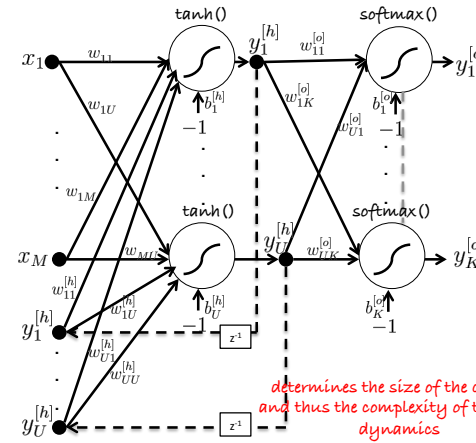
- Output of the hidden layer feeds into the output layer as well as the hidden layer.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[h]}$  - weight connecting saved output of neuron  $i$  from the hidden layer to neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs     $U$  - # of hidden neurons     $K$  - # of outputs

## Simple Recurrent Network (SRN)

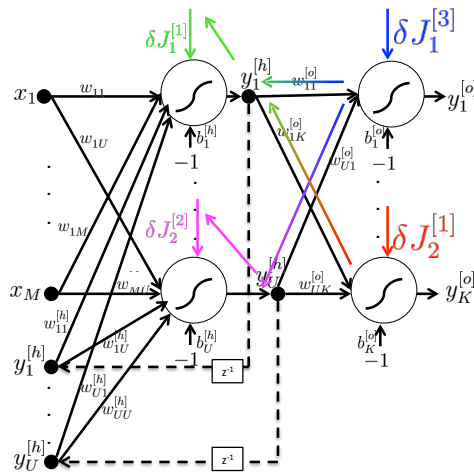
- Output of the hidden layer feeds into the output layer as well as the hidden layer.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[h]}$  - weight connecting saved output of neuron  $i$  from the hidden layer to neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs     $U$  - # of hidden neurons     $K$  - # of outputs

## Simple Recurrent Network (SRN)

- Backpropagation works in SNR.



$w_{ij}$  - weight connecting input  $i$  to neuron  $j$  in the hidden layer  
 $b_j^{[h]}$  - bias of neuron  $j$  in the hidden layer  
 $w_{ij}^{[h]}$  - weight connecting saved output of neuron  $i$  from the hidden layer to neuron  $j$  in the hidden layer  
 $w_{ij}^{[o]}$  - weight connecting neuron  $i$  from the hidden layer to neuron  $j$  in the output layer  
 $b_j^{[o]}$  - bias of neuron  $j$  in the output layer  
 $x_i$  - input  $i$  to the hidden layer  
 $y_i^{[h]}$  - output of neuron  $i$  in the hidden layer  
 $y_i^{[o]}$  - output of neuron  $i$  in the output layer  
 $M$  - # of inputs     $U$  - # of hidden neurons     $K$  - # of outputs

## SNR training

- Backpropagation works in SNR:

$$\text{error "blame" on output neurons } \delta J_j^{[o]} = \tilde{y}_j(y_{jt}^{[o]} - 1) + (1 - \tilde{y}_j)y_{jt}^{[o]}$$

$$\text{error "blame" on hidden neurons } \delta J_i^{[h]} = \sum_{j=1}^K w_{ij}^{[o]}(y_{jt}^{[o]}(1 - y_{jt}^{[o]}))\delta J_j^{[o]}$$

$$\text{updates to hidden-output weights } \Delta w_{ij}^{[o]} = y_{it}^{[h]}(y_{jt}^{[o]}(1 - y_{jt}^{[o]}))\delta J_j^{[o]}$$

$$\text{updates to output layer biases } \Delta b_j^{[o]} = -(y_{jt}^{[o]}(1 - y_{jt}^{[o]}))\delta J_j^{[o]}$$

$$\text{updates to hidden-hidden weights } \Delta w_{ij}^{[h]} = y_{i(t-1)}^{[h]}(1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]})\delta J_j^{[h]}$$

$$\text{updates to hidden layer biases } \Delta b_j^{[h]} = -(1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]})\delta J_j^{[h]}$$

$$\text{updates to input-hidden weights } \Delta w_{ij} = x_i(1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]})\delta J_j^{[h]}$$

## SNR training

Backpropagation works in SNR:

error "blame" on output neurons  $\delta J^{[o]} = \sum_U (\tilde{y}_j(y_{jt}^{[o]} - 1) + (1 - \tilde{y}_j)y_{jt}^{[o]})$  derivative of softmax with respect to network output

error "blame" on hidden neurons  $\delta J_i^{[h]} = \sum_{j=1} w_{ij}^o (y_{jt}^{[o]}(1 - y_{jt}^{[o]})) \delta J_j^{[o]}$  derivative of softmax with respect to output neuron's activity

updates to hidden-output weights  $\Delta w_{ij}^{[o]} = y_{it}^{[h]} (y_{jt}^{[o]}(1 - y_{jt}^{[o]})) \delta J_j^{[o]}$  output of the hidden neuron i due to previous input (indexed t-1)

updates to output layer biases  $\Delta b_j^{[o]} = - (y_{jt}^{[o]}(1 - y_{jt}^{[o]})) \delta J_j^{[o]}$  output of the hidden neuron j due to current input (indexed t)

updates to hidden-hidden weights  $\Delta w_{ij}^{[h]} = y_{it}^{[h]} (1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]}) \delta J_j^{[h]}$  derivative of tanh with respect to hidden neuron's activity

updates to hidden layer biases  $\Delta b_j^{[h]} = - (1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]}) \delta J_j^{[h]}$

updates to input-hidden weights  $\Delta w_{ij} = x_i (1 + y_{jt}^{[h]})(1 - y_{jt}^{[h]}) \delta J_j^{[h]}$

## An example: sentence generation

- A dictionary of 6 strings encoded as a 6-dim zero-one vectors (M=6)

$$x \in \begin{Bmatrix} \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} \end{Bmatrix}$$

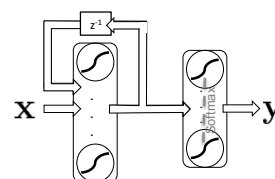
Cars   Dogs   are   furry   red

- Output is a 6-value probability distribution over all possible strings (K=6)

$$p(\text{string}) \quad y = \begin{Bmatrix} y_1^{[o]} & P(\text{string} = \cdot) \\ y_2^{[o]} & P(\text{string} = \text{Cars}) \\ y_3^{[o]} & P(\text{string} = \text{Dogs}) \\ y_4^{[o]} & P(\text{string} = \text{are}) \\ y_5^{[o]} & P(\text{string} = \text{furry}) \\ y_6^{[o]} & P(\text{string} = \text{red}) \end{Bmatrix}$$

SRN:

- 16 hidden neurons (U=16)
- Tanh activity function in the hidden layer
- Softmax function on the output



Training:

- Initial state  $y_h = [y_1^{[h]} \dots y_U^{[h]}]^T$  set to zeros.

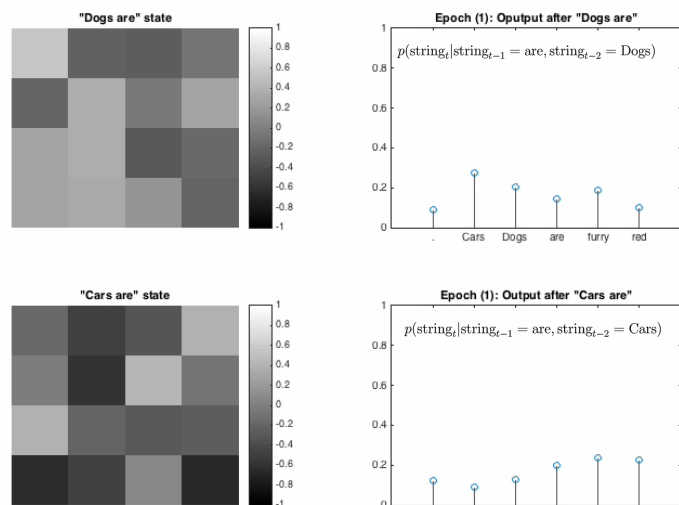
Input sequence 1:  $\begin{Bmatrix} \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix} \end{Bmatrix}$  Dogs   are   furry

Desired output 1:  $\begin{Bmatrix} \begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \end{Bmatrix}$  are   furry

Input sequence 2:  $\begin{Bmatrix} \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix} \end{Bmatrix}$  Cars   are   red

Desired output 2:  $\begin{Bmatrix} \begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \end{Bmatrix}$  are   red

## An example: sentence generation



## An example: more complicated text generation

- A dictionary of 1038 words, zero-one coded into a 1038-dimensional input
- SRN with 64 hidden units and 1038-dimensional output
- Trained with softmax on the first 5,000 words of "Pride and Prejudice" to predict the next word in a sentence.
- Used to generate text by priming the network with a sentence fragment...
- ...then taking the predicted most likely word to follow and feeding it as the next input...repeating this over and over.

## An example: more complicated text generation

- Sample of the training data:

*"It is a truth universally acknowledged, that a single man in possession of a good fortune must be in want of a wife. However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is so well fixed in the minds of the surrounding families, that he is considered as the rightful property of some one or other of their daughters."*

- Sentences generated after the starting sequence **"It is"**:

- After 1 epoch,  $J=4.331e+04$

*"It is rode seminaries no respectable enduring one little your does fortune bad everywhere please sensible information your views too"*

- After 300 epochs,  $J=2.742e+04$

*"It is barefaced truth the I. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley. Bingley."*

- After 3500 epochs,  $J=7.595e+03$

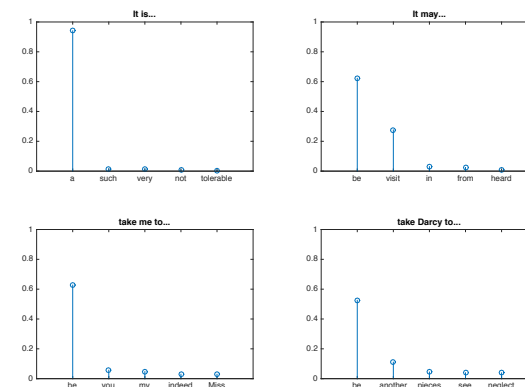
*"It is a truth universally acknowledged, and no two for your husband day, and the two sixth of joy was a lively and it till the room"*

- After 9900 epochs,  $J=2.310e+03$

*"It is a truth universally acknowledged, that a single man ought to be above his company, and above his father, my dear. I have a high respect for your nerves."*

## An example: more complicated text generation

- Probabilities of the top 5 most likely words after a certain word sequence, as given by the "Pride and Prejudice"-trained SRN after 10000 training epochs:



## Summary

- Feed forward neural networks can do classification and regression
- Softmax function gives network output as a probability distribution
- Recurrent neural networks find temporal patterns in the sequences of input data
  - Simple Recurrent Network can be trained using the backpropagation algorithm

Reading for the lecture: Andrej Karpahty, "The Unreasonable Effectiveness of Recurrent Neural Networks", <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Reading for next lecture: AIMA Chapter 18.9