COMP 3225

Natural Language Processing

RNN

Stuart E. Middleton

sem03@soton.ac.uk

University of Southampton

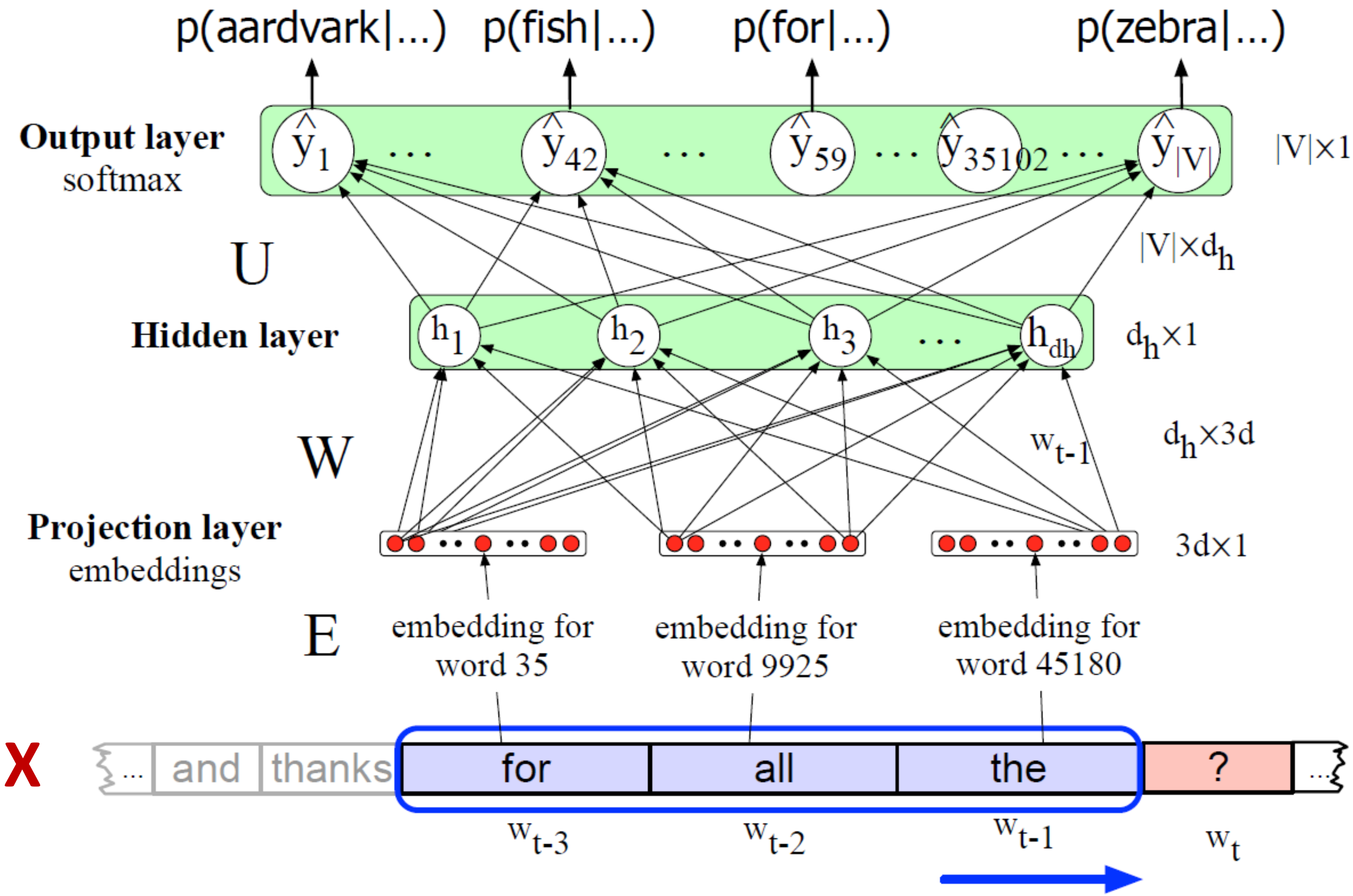
Overview

- Simple Recurrent Neural Networks
- <break - discussion point>
- Applications of RNNs
- Stacked RNNs

Simple Recurrent Neural Networks

- Language is a sequence that unfolds over time
- Hidden Markov Models input one words at a time
  - Viterbi algorithm for POS tagging
  - CRF model for NER
  - State based >> Given previous state (word sequence) the model predicts the next state (next word)
  - N-gram models use context windows to reduce model complexity
- Supervised Machine Learning take fixed sequence inputs
  - Feedforward Neural Networks called Multi-layer Perceptron (MLP)
  - Sentences do not have a fixed length
  - Workaround >> Sliding window of words
  - Decisions made in one window have no impact on subsequent decisions
- Problem
  - Hard to learn semantic patterns with these techniques such as constituency due to long range dependencies

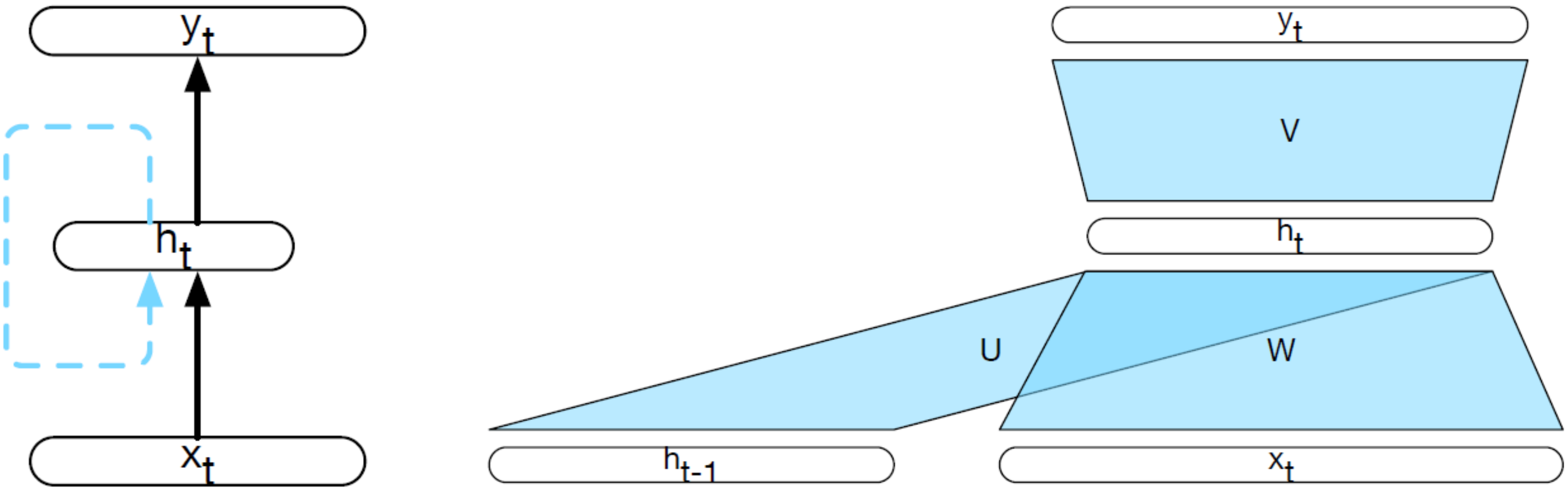
Simple Recurrent Neural Networks



- MLP (not RNN) with sliding window 3 words and a hidden layer with dim d
  Separate patterns learnt for 'thanks for all', 'for all the' ...

Simple Recurrent Neural Networks

- Elman Networks - simple Recurrent Neural Network



- X = input vector
- H = hidden layer W = weights to input U = weights to previous input
- Y = output vector V = weights to output
- Input is augmented with the value of the hidden layer from the previous step
  - This is a type of memory or context
  - Adding this temporal dimension allows patterns to remember long-distant dependencies
- Training via backpropagation

Cite: Elman, J. L. (1990). Finding structure in time. Cognitive science 14(2), 179–211.

Simple Recurrent Neural Networks

- Activation function g computes hidden layer values h_t
- Output vector is computed using a function f (often softmax)

$$
\begin{aligned}
h_t &= g(Uh_{t-1} + Wx_t) \\
y_t &= f(Vh_t) \\
y_t &= \text{softmax}(Vh_t)
\end{aligned}
$$

softmax maps vector of values to a probability distribution

- Weight matrix (current W, previous U) and input vector are multiplied together
- Weight matrix V (size = vocab) is multiplied with hidden layer values

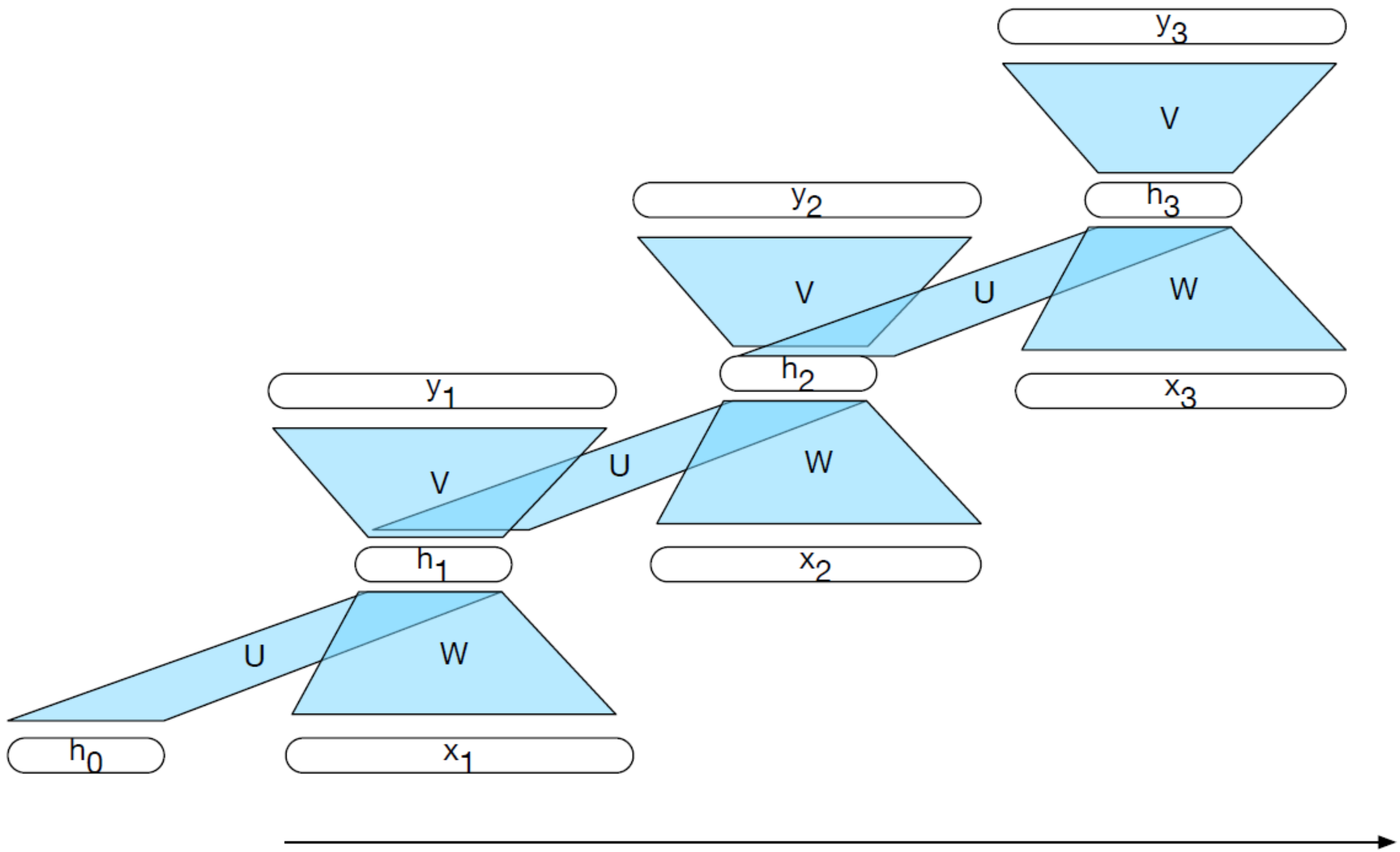
function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
  $y_i \leftarrow f(V\ h_i)$
return y

- Weights U, V and W are shared across time

Simple Recurrent Neural Networks

- Loss function needs h_t and h_{t-1}, which in turns needs h_{t-2} ...
- Modern deep learning frameworks will unroll the recurrent networks in time to compute loss



Applications of RNNs

- Language model using RNN (e.g. predicting next word)
- Input X = sequence of L words in vocab V

  = each word x_t is a one-hot vector (dim = size of V)

  = matrix L x V

$$\begin{array}{cccccccccccc} [0 & 0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 & 0] \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots & & \ldots & & |V| \end{array}$$

'1' indicates the word is at index 5 within V

- Output Y = predicted next word in seq = probability dist over V
- E = Word embedding matrix (shape = one-hot dim x hidden dim d)

$$\begin{aligned} e_t &= E^T x_t \\ h_t &= g(Uh_{t-1} + We_t) \\ y_t &= \text{softmax}(Vh_t) \end{aligned}$$

Dot product of matrix E + one-hot vector x_t = hidden layer values e_t
concat this with previous hidden layer values and apply softmax

Applications of RNNs

- Cross-entropy loss function
  - Cross-entropy measures how well a set of estimated class probabilities matches the target class
  - Variants available via Tensorflow or PyTorch for use with RNN models
  - See required reading for more details

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Softmax

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

Cross-entropy

- Teacher forcing during training
  - Use output from prior training steps as input to help model convergence
  - When predicting y_{t+1} use ground truth sequence $x_{1..t}$ not the predicted word values based on $y_{1..t}$

Break

- Panopto Quiz - discussion point

- Why is an RNN able to handle long-distance dependencies when tagging words in a sentence?

Each hidden layer can remember its previous activation
There is a layer for each position in the sequence
The softmax function allows memory of previous activations
It cannot, long-distance dependencies are a weakness of RNNs

Break

- Panopto Quiz - discussion point

- Why is an RNN able to handle long-distance dependencies when tagging words in a sentence?

Each hidden layer can remember its previous activation
>> Yes

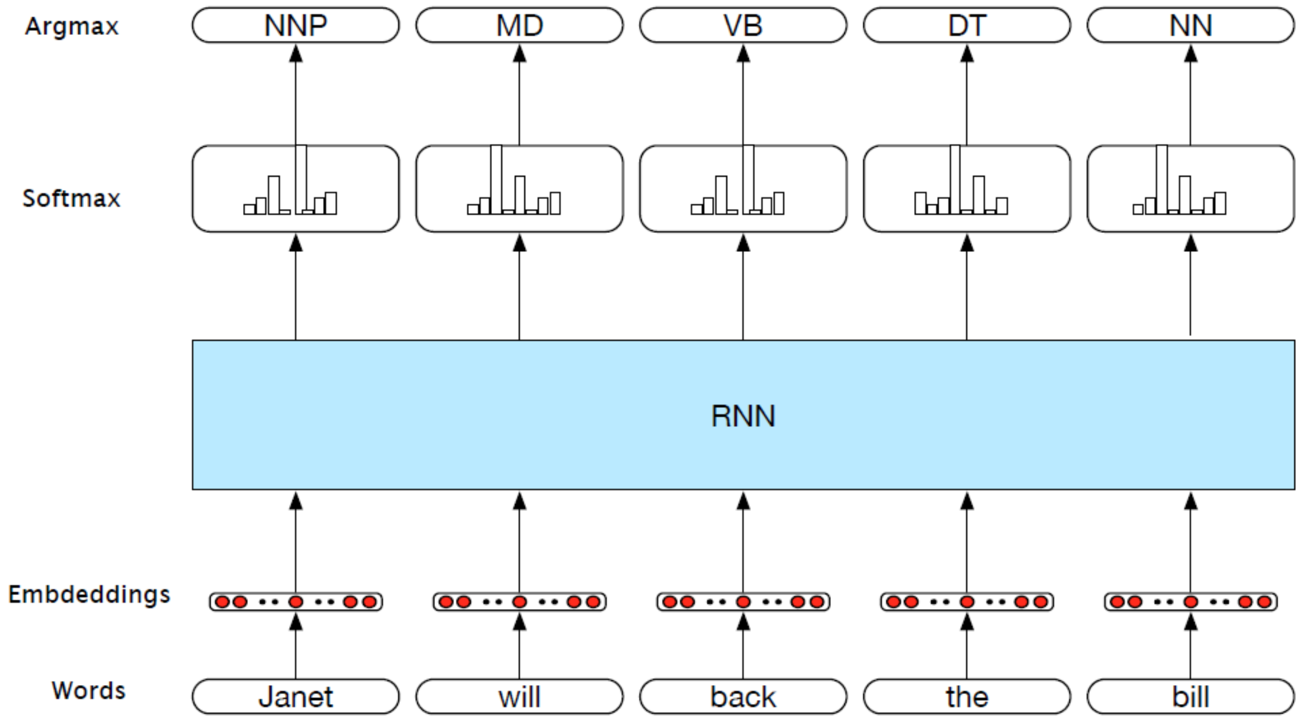
There is a layer for each position in the sequence
>> There is only one layer - multiple layers are stacked RNN's (will discuss later)

The softmax function allows memory of previous activations
>> Softmax function create a probability distribution from a set of activation values

It cannot, long-distance dependencies are a weakness of RNNs
>> One of RNN's key benefits is its memory and ability to handle long sequences

Applications of RNNs

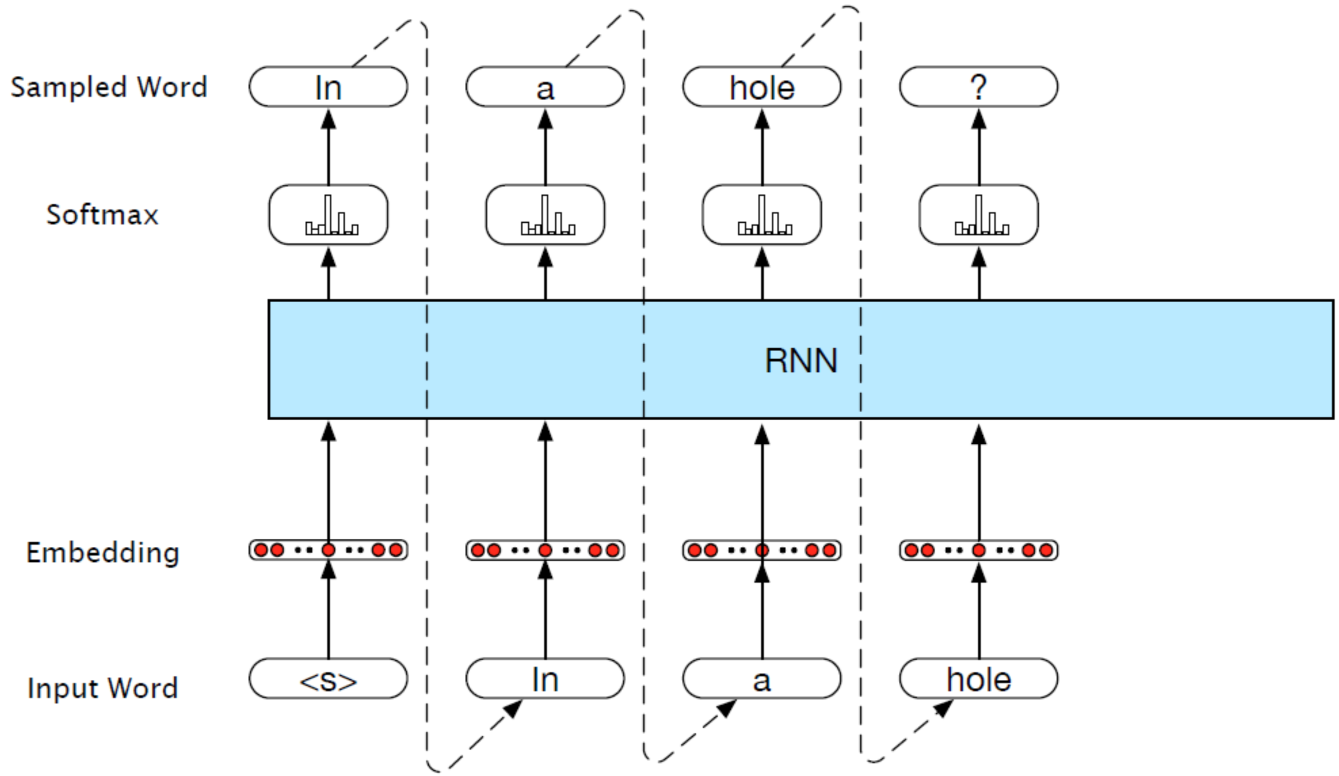
- Sequence labelling using RNN (e.g. POS tagging)



- Input X = sequence of words
- Output Y = POS tag probabilities (argmax chooses most likely)
- Pre-trained word embeddings
- Cross-entropy loss function

Applications of RNNs

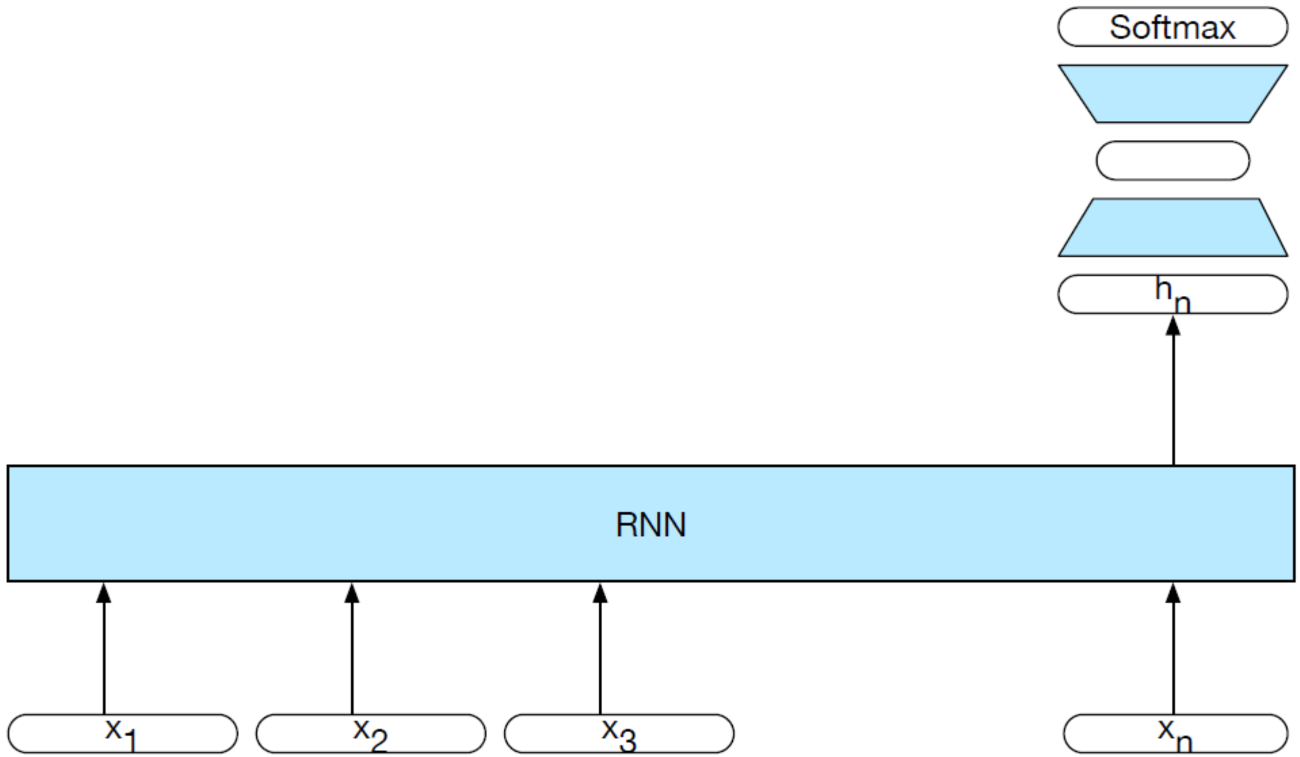
- Autoregressive Generation using RNN (e.g. text generator)



- Input X = sequence of words so far (start with token <s>)
- Output Y = next word to be added to X
- Pre-trained word embeddings; Cross-entropy loss function

Applications of RNNs

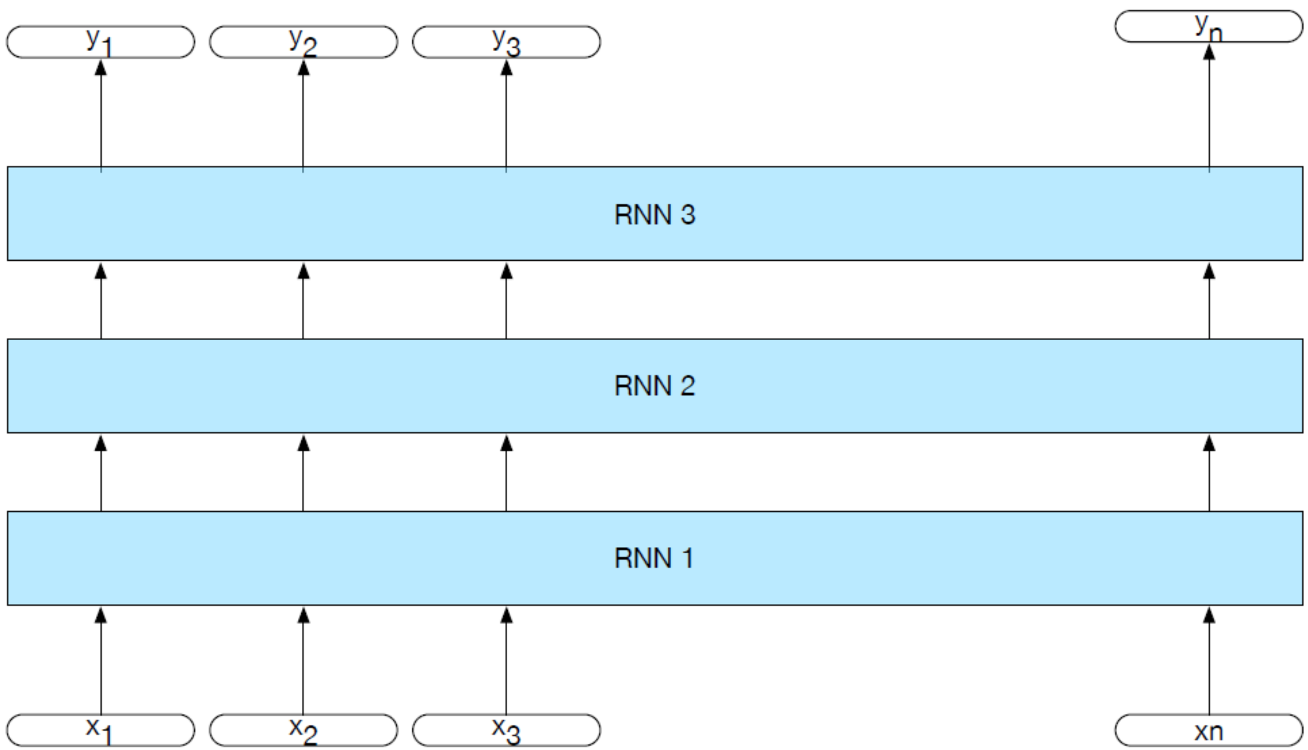
- Sequence classification using RNN (e.g. sent/document classifier)



- Input X = sequence of words in sent/document
- Output Y = class probability
- RNN + MLP, cross-entropy loss based on classification result

Stacked RNNs

- Stacked RNNs >> Deep Learning
- Entire output sequence of one RNN used as input to another
- Adding RNN layers boosts performance at expense of train time
- RNN layers encode different levels of abstract representations, allowing more sophisticated patterns to be encoded



Stacked RNNs

- Bi-Directional RNNs
- RNN hidden layer value h_t is a function of its input seq x from 1..t

$$h_t^f = RNN_{forward}(x_1^t)$$

x[1::t]

- In applications where we know the entire input sequence 1..n, we can do the same but work backwards using t..n

$$h_t^b = RNN_{backward}(x_t^n)$$

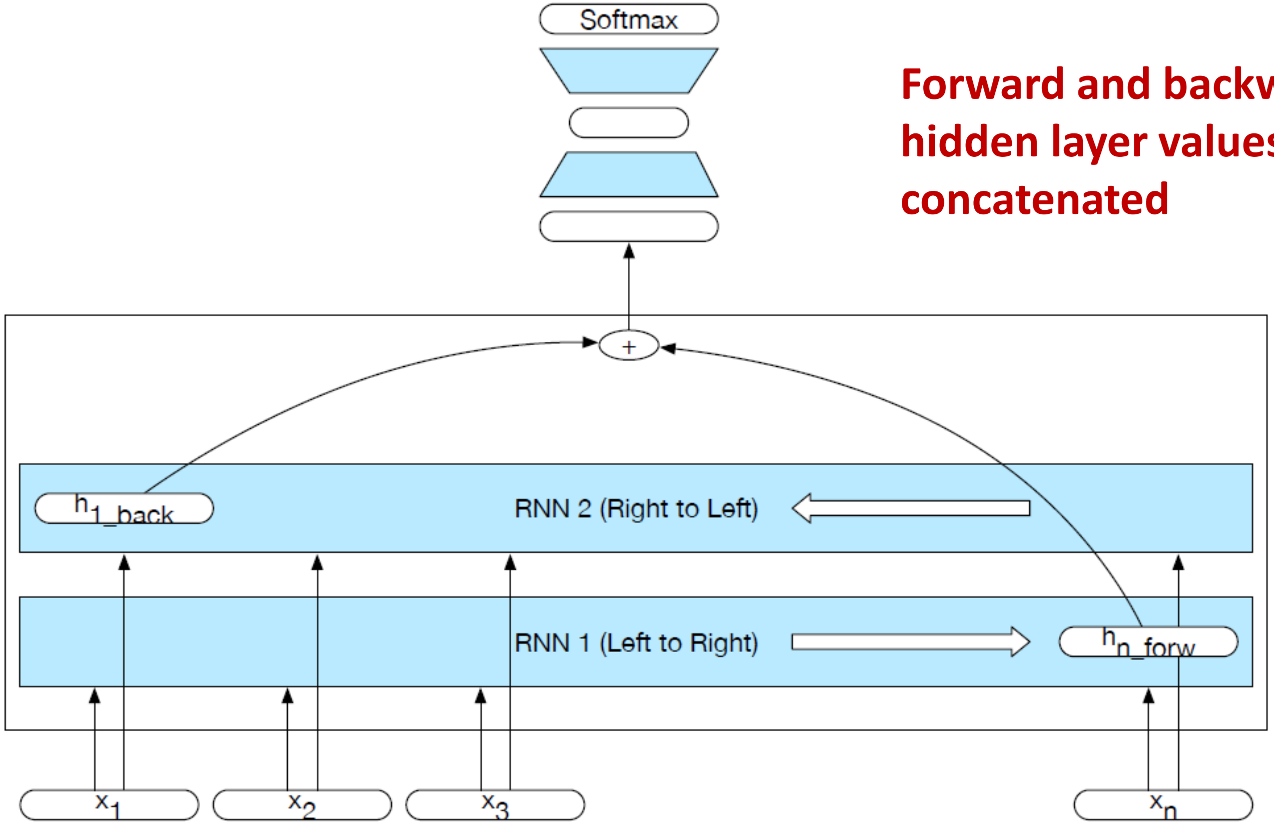
x[t::n]

- Then we concat the hidden layer values from the two layers for each position t in the sequence

$$h_t = h_t^f \oplus h_t^b$$

Stacked RNNs

- Bi-RNN sequence classifier



Forward and backward hidden layer values concatenated

Required Reading

- RNN
  - Jurafsky and Martin, Speech and Language Processing, 3rd edition (online) >> chapter 9
- Softmax, cross-entropy and cross-entropy gradient vectors
  - Aurelien Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly , 2017.
    >> Chapter 4: Training Models: Softmax regression
- Neural networks and MLPs (optional)
  - Jurafsky and Martin, Speech and Language Processing, 3rd edition (online) >> chapter 7

Questions

- Panopto Quiz - 1 minute brainstorm for interactive questions

    Please write down in Panopto quiz in **1 minute** two or three questions that you would like to have answered at the next interactive session.

    Do it **right now** while its fresh.

    Take a screen shot of your questions and **bring them with you** at the interactive session so you have something to ask.