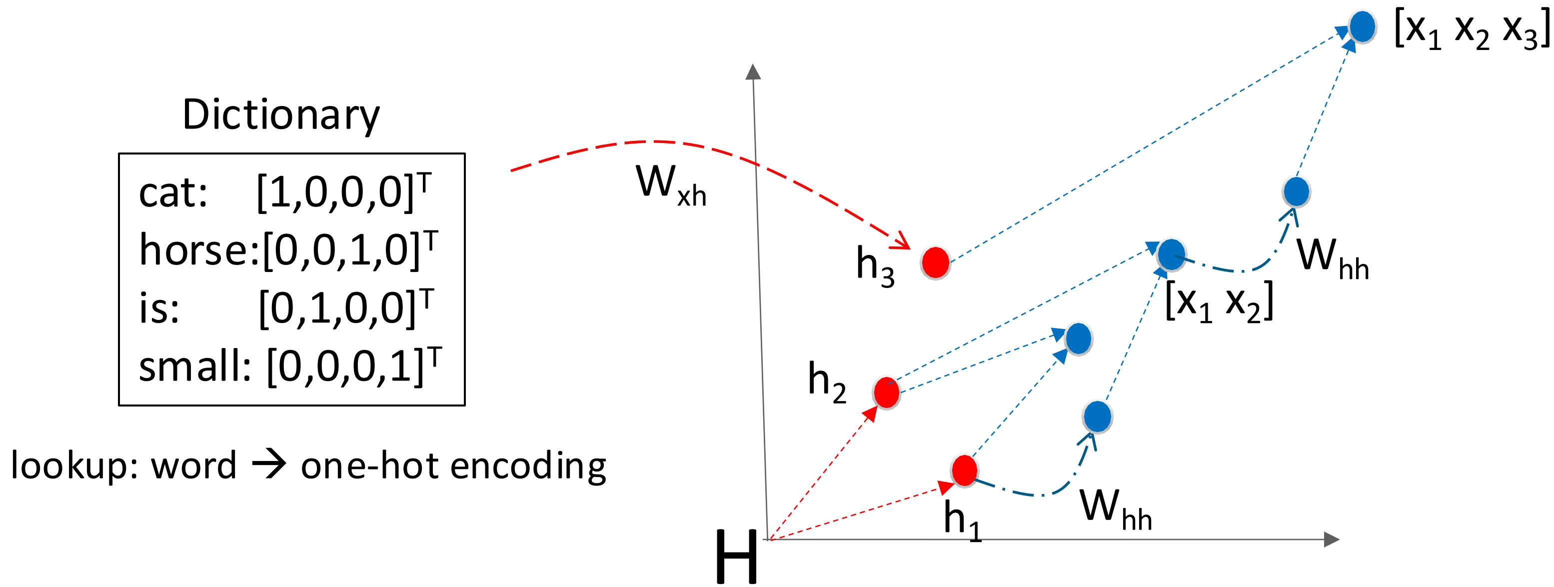
Recurrent Neural Networks (RNNs)

Keshav Pingali
The University of Texas at Austin

Introduction

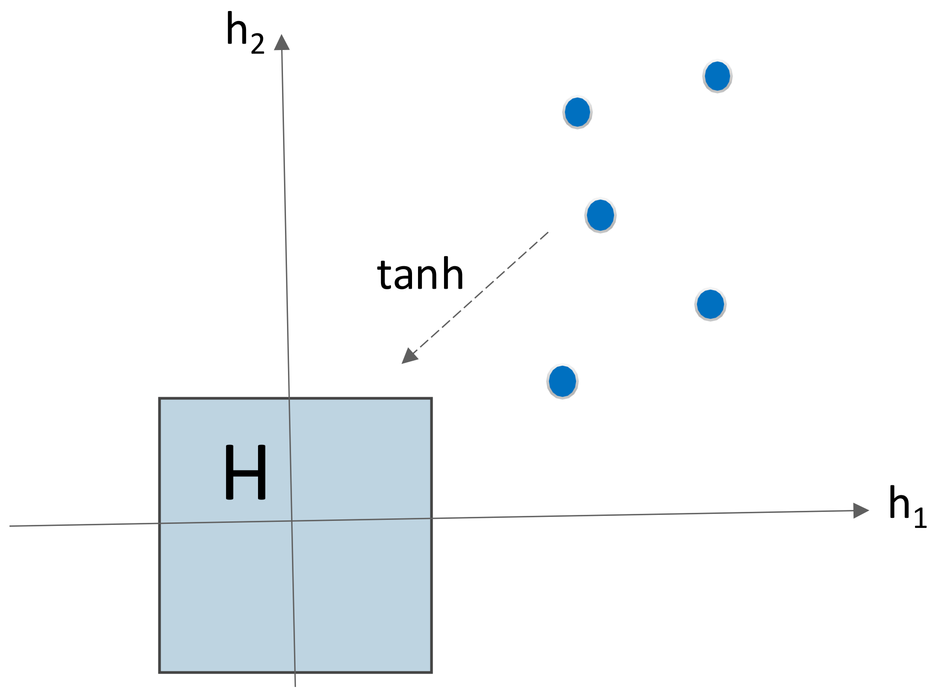
- RNNs: input is sequence, output can be a scalar or sequence
- Sentiment analysis:
  - Input: sentence or paragraph reviewing something
  - Output: positive or negative review?
- Machine translation
  - Input: sentence (sequence of words) in English
  - Output: sentence (sequence of words) in French
- Training data
  - Set of sentence pairs: (sentence in English, sentence in French)
- Abstractly we want a function of this type
  - F: $[x_1,x_2,\ldots,x_m] \rightarrow [y_1,y_2,\ldots,y_n]$ (m,n can be different for different sentences)
  - Input sequence can be of arbitrary length
  - Assume m=n for simplicity
- Questions
  - How do we encode (represent) words?
  - How do we encode sequences of words?
  - How do we handle arbitrarily long sequences?
  - How is output produced?

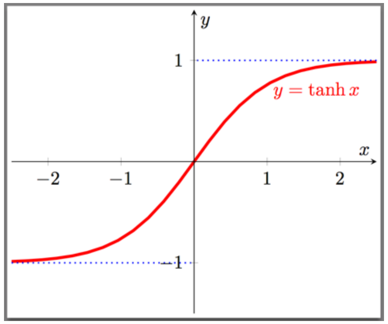
Encoding words and sequences of words



- **Vector space model**: words and sequences of words **embedded** as points in H = $\Re^m$
- Embedding of word x: h = W_{xh} * lookup(x)
  - lookup() uses dictionary to map word x to its one-hot encoding
  - W_{xh} is learned: column m is embedding of word with 1 in the m^{th} position of one-hot encoding
- Embedding of sequence (e.g.) [x_1 x_2]
  - One possibility: add embeddings of x_1 and x_2
    - Drawback: [small cat] will have same embedding as [cat small]
  - Better idea: W_{hh} * h_1 + h_2 (where W_{hh} is learned)
- In general, H: sequence of words → $\Re^m$
  - H([ ]) = $\underline{0}$
  - H([x_1 x_2.... x_{i-1} x_i]) = W_{hh}*H([x_1 x_2 … x_{i-1}]) + W_{xh}*lookup(x_i)

In practice



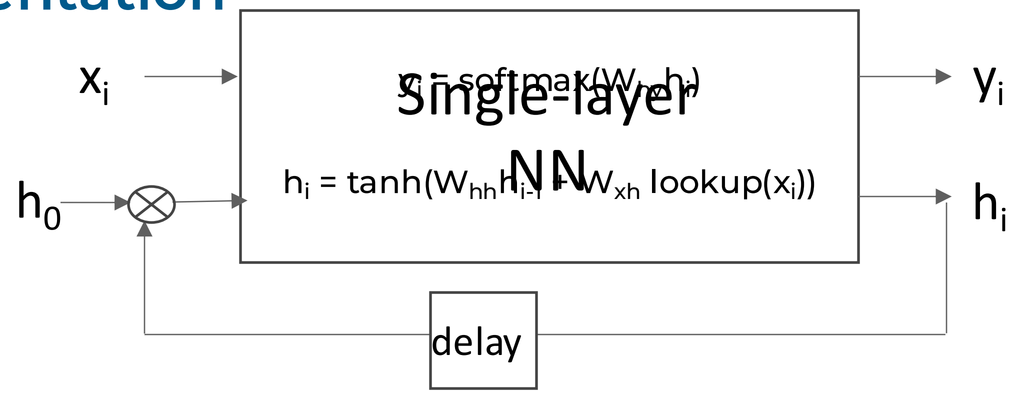


$$\frac{dy}{dx} = 1 - \tanh^2(x)$$

- Use tanh to squash H encodings into unit hypercube to prevent blow-up
  – $H([x_1\ x_2 \ldots.\ x_{i-1}\ x_i]) = \tanh(W_{hh} * H([x_1\ x_2 \ldots\ x_{i-1}]) + W_{xh} * \text{lookup}(x_i))$

- Output for simple RNN produced “online”
  – y_i depends only on $[x_1, \ldots, x_i]$
  – $y_i \sim \text{softmax}(W_{hy} * H([x_1\ x_2 \ldots.\ x_{i-1}\ x_i]))$
    > Output of softmax = probability vector for next output word
    > y_i is sampled from output distribution of softmax

RNN implementation

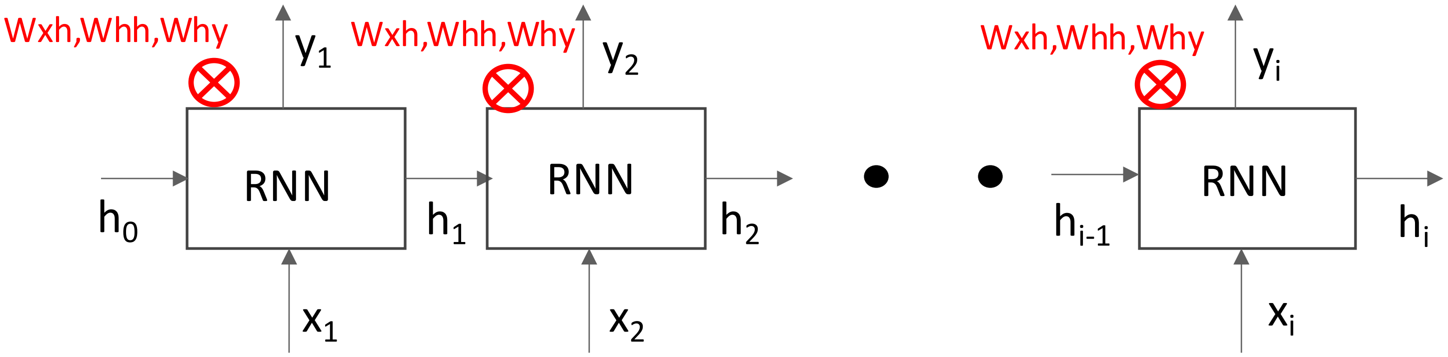


- RNN is single-layer neural network with feedback loop

- $H([x_1\ x_2 \ldots.\ x_{i-1}\ x_i])$ represented as vector h_i
  - Fed back to next iteration

- Details
  - W_{xh} can be initialized to embeddings from Word2Vec
  - RNNs can be chained to form "multi-layer" RNNs

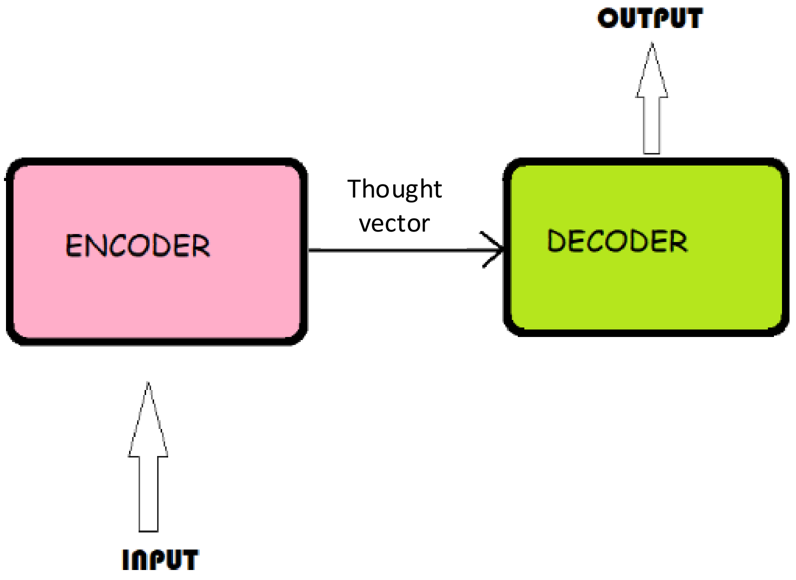
Training RNNs



- Training data: $\{[x_1,x_2,\ldots,x_n] \rightarrow [Y_1,Y_2,\ldots,Y_n]\}$
  - Notation: Y_i is training data, y_i is output produced by RNN during "inference"
- Training: at each step
  - Compute cross-entropy between ground truth Y_i and computed value y_i
    - Strictly speaking, between one-hot encoding of Y_i and output of softmax at step i
  - Back-propagate using weight-sharing to update weights
  - In practice, limit the size of the "look-back" window to 3-4
- Analogy: path-sensitive dataflow analysis

Improving RNNs: Encoder-decoder architectures

- Requiring output to be produced online means
  - No "look-ahead" in input stream is possible when determining how to produce next output word
  - Input and output sequences must have same length
- Solution
  - First encode entire input sequence (encoder)
  - Then produce output one word at a time (decoder)
- Two architectures
  - Baseline encoder-decoder architecture based on RNNs
  - Transformers



Baseline encode-decoder architecture

- We want to learn a function F: $[x_1,x_2,...,x_n,y_1,y_2,..,y_{i-1}] \rightarrow y_i$
- Training input: $[x_1,...,x_n]$, $[Y_1,...,Y_n]$
- Encoder
  - $h_i = f_1(h_{i-1},x_i)$
    - $h_0 = 0$
  - h_n = thought vector (embedding of $[x_1,x_2,...,x_n]$)

$$f_1 = \tanh(W_{hh}*h_{i-1} + W_{xh}*\text{lookup}(x_i))$$

- Decoder (training)
  - $h_{n+i} = f_1(h_{n+i-1},Y_{i-1})$ (embedding of $[x_1,x_2,...,x_n,Y_0,Y_1,...,Y_{i-1}]$)
    - Y_0 = _START
  - $y_i = f_2(h_{n+i})$ (used to compute loss between y_i and Y_i)

$$f_2 \sim \text{softmax}(W_{hy}* h_{n+i})$$

- Decoder (inference)
  - $h_{n+i} = f_1(h_{n+i-1},y_{i-1})$ (embedding of $[x_1,x_2,...,x_n,y_0,y_1,...,y_{i-1}]$)
    - y_0 = _START
  - $y_i = f_2(h_{n+i})$

Remarks on RNNs

- Drawbacks of RNN-based translation

  (1) Encoding and decoding are sequential

  (2) Information loss for long sequences

  - In principle, encoder-decoder RNN architectures allow the decoder to see the entire input sequence before producing any output
  - However, signal from first few words is lost by end of long sequence
  - Experience: RNN-based translation works only for sentences of 4-5 words and if languages are well aligned

- Solution: transformer

  (1) Create encoding of sequence in **parallel**

  (2) **Attention**: pick up important signals for a given word from *anywhere* in input sequence

  - Example: The boy stood on the burning deck whence all but he had fled.