Time Series and NLP

Time Series

Handling Variable Length Vectors

Sentences or documents are generally not of a fixed length.

Standard neural nets require the input to be of a fixed length.

That’s a problem.

How to deal with Variable Length

Some of the common practices to handle variable length vectors are:

- **Bag-of-wording:** ignore the sequential nature and just count words
- **Embedding:** map the sequence to a vector
- **Truncation and padding:** chop off long vectors and pad short ones
- **Convolution:** use a fixed-length filter and aggregate the results
- **Recurrence**

Bag of words

original	a b c d e f … q r t
a b c a c a	3 1 2 0 0 0…0 0 0
c d d	0 0 1 2 0 0…0 0 0
b c c q r t	0 1 2 0 0 0…1 1 1
f f q	0 0 0 0 0 2…1 0 0

Truncation and Padding

original	**fixed**
a b c a c a	a b c a
c d d	c d d 0
b c c q r t	b c c q
f f q	f f q 0

Embedding

original

a b c a c a

assume embeddings:

a = [1.1 0 2.5 -2]

b = [0 -1 -2 .5]

c = [2 1.3 0 0]

embed sequence as 3 a + b + c =

[5.3 0.3 5.5 -5.5]

Next week, we'll see fancier embedding

1-D convolution

a b c a c a

kernel size = 3

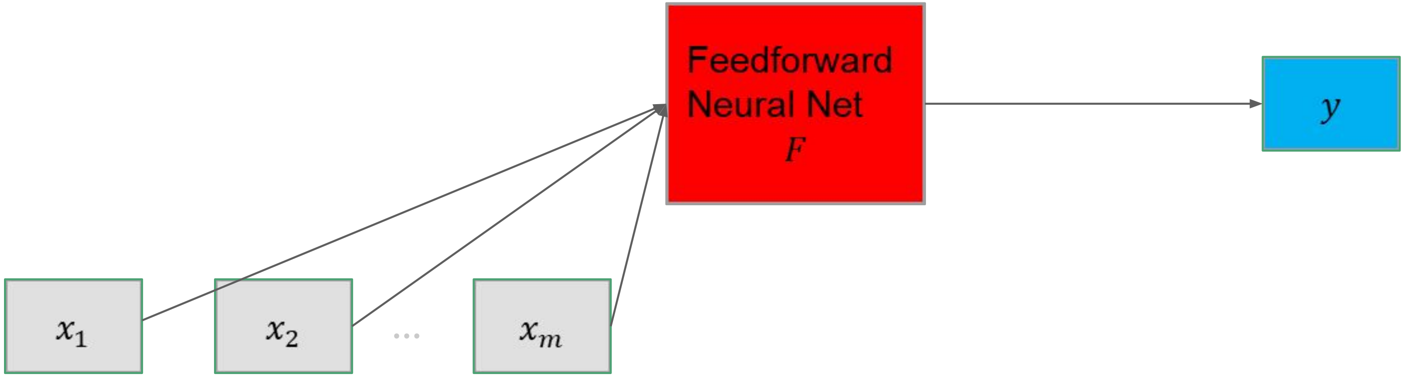
stride = 1

Output	y_1	y_2	y_3	
Hidden layer	h_1	h_2	h_3	h_4
Input	a b c	b c a	c a c	a c a

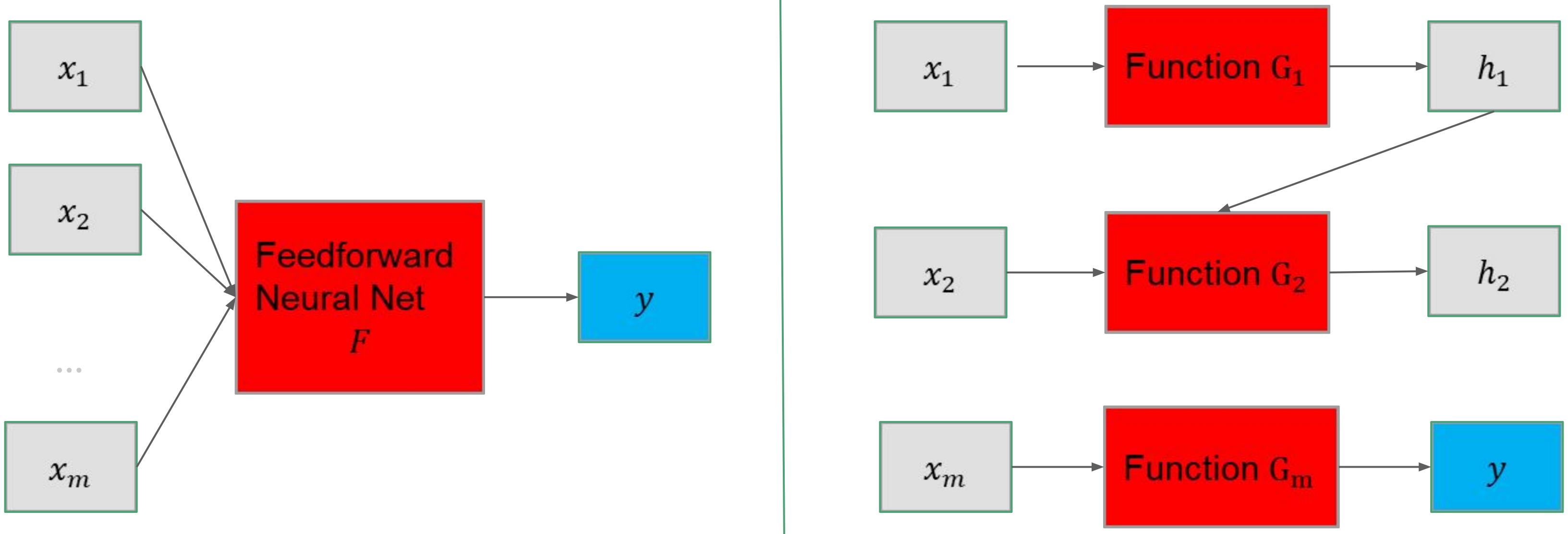
prediction y = *sum*(y_i) or *max*(y_i)

Recurrent Neural Nets (RNNs)

The architecture for feedforward nets is too restrictive for time series.

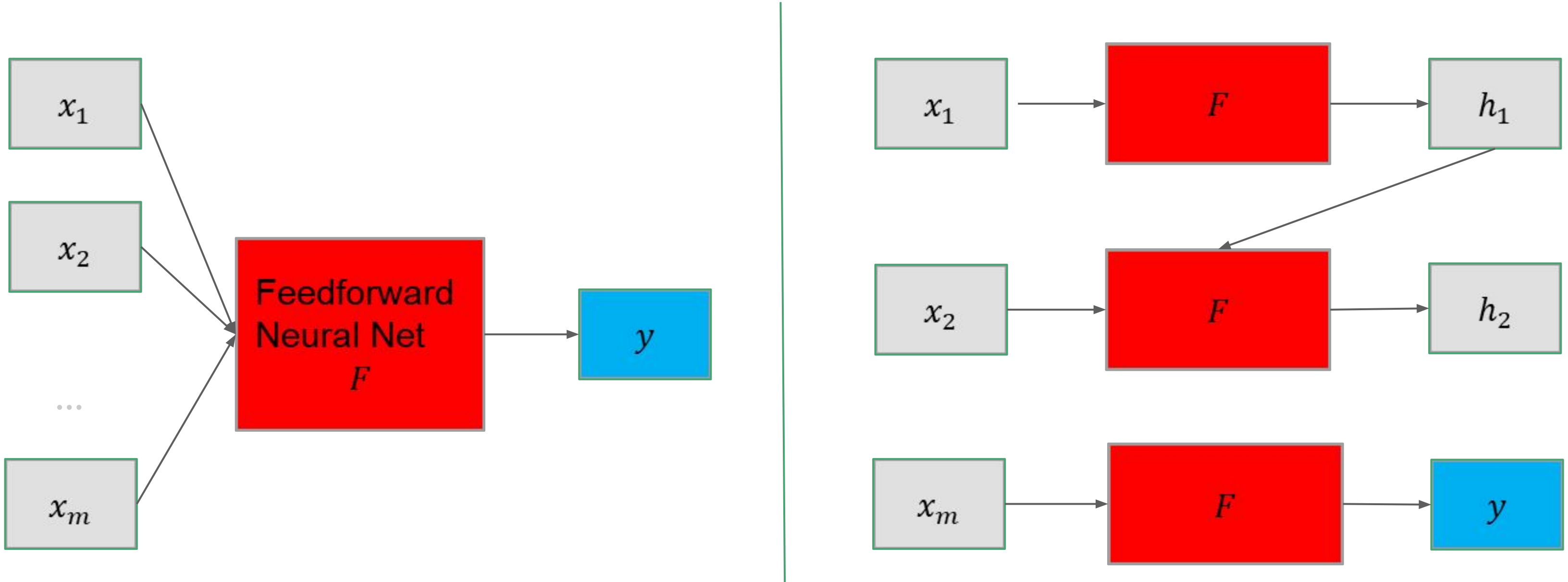


RNNs: Hidden state

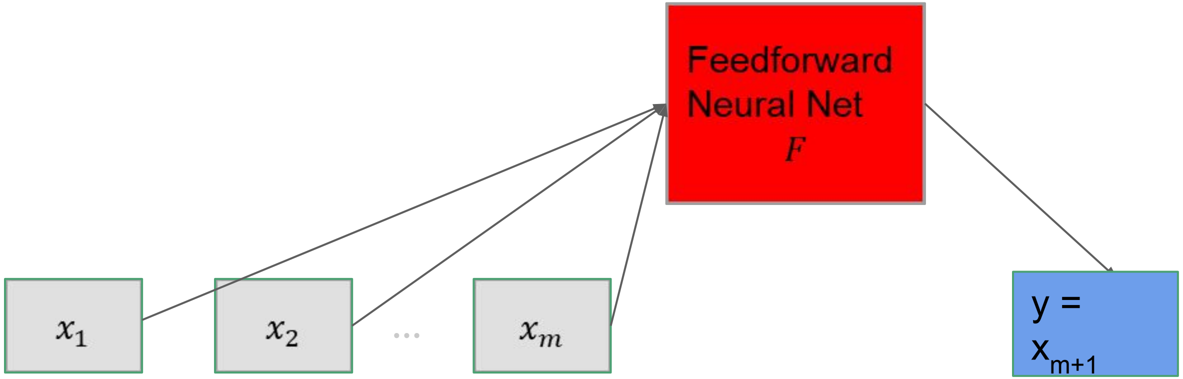


We can't learn a new function for each timestep!
How do we simplify our architecture?
The way we think doesn't change from moment to moment.
Assume stationarity!!!

We share weights across time.



We often make RNNs semi-supervised by predicting the next input.

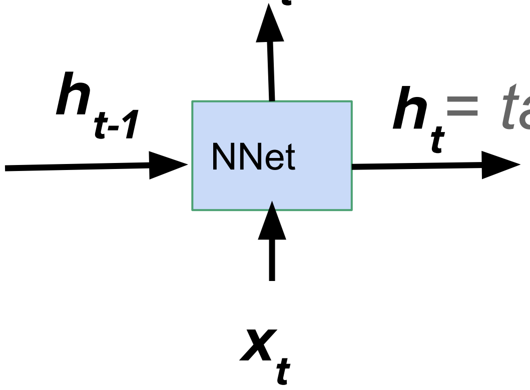


RNNs: Basic Architecture

Recurrent neural network (RNN)

- $\boldsymbol{x}_t$ **= input** (e.g. a word embedding)
- $\boldsymbol{h}_t$ **= hidden state**
- $\boldsymbol{o}_t$ **= output,** estimating the true value $\boldsymbol{y}_t$ (e.g. $\boldsymbol{x}_{t+1}$)

$\boldsymbol{o}_t = softmax(\boldsymbol{h}_t)$

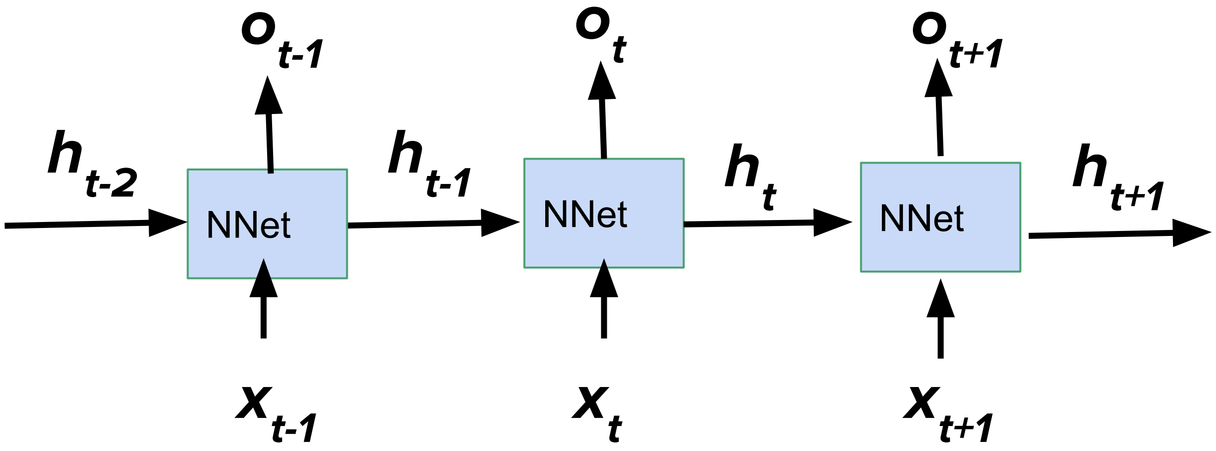


$\boldsymbol{h}_t = tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$

RNN “unrolled”

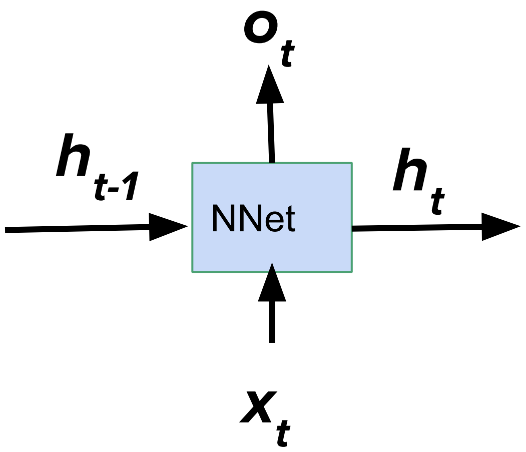
Assumes “stationarity”: the same weights at all time steps

Gradient descent requires “backpropagation through time”



RNNs can be deep

- $\boldsymbol{x}_t$ = **input**
- $\boldsymbol{h}_t$ = **hidden state** = $f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$
- $\boldsymbol{o}_t$ = **output** = $softmax(\boldsymbol{h}_t)$



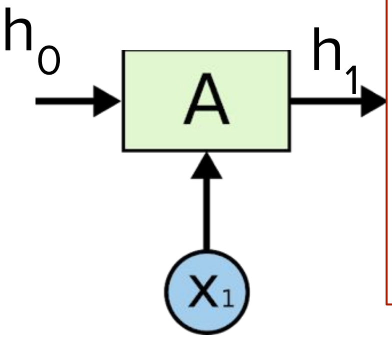
RNN Architectures for NLP

and Language Models

RNN architectures

	Input		Output
language models	subsequence	➜	next item in sequence
sequence labeling	sequence	➜	label of entire sequence
tagging	subsequence	➜	label of current item
seq2seq	sequence	➜	other sequence

One step of the RNN (again)



embed

went

predict next word

y_1 = *softmax*(h_1) =

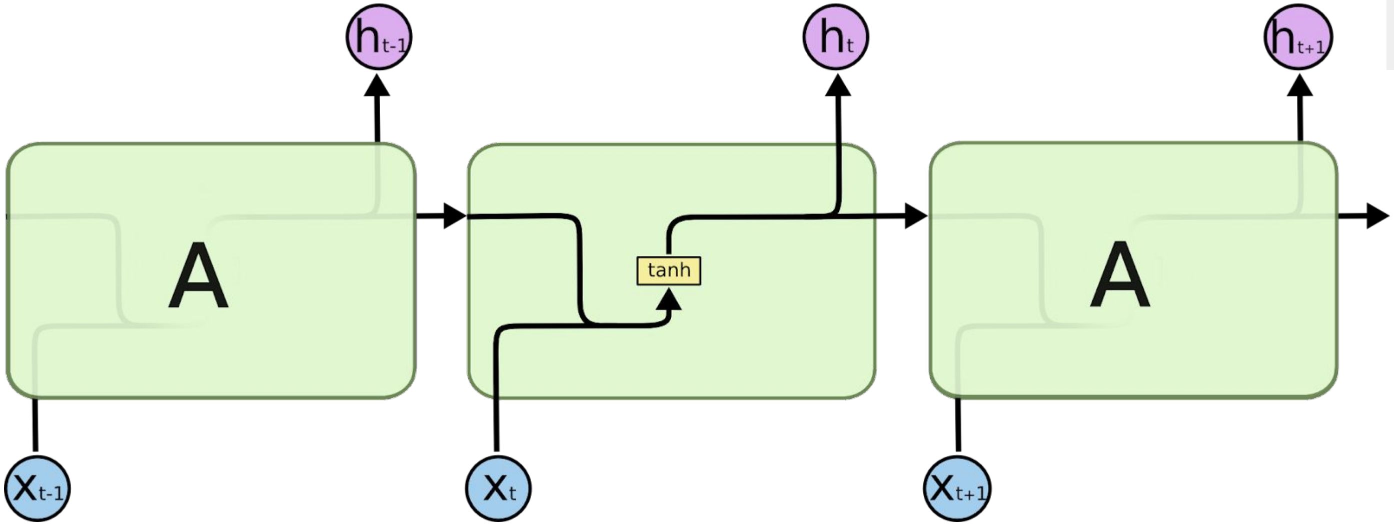
x_2 = *'to'*

Inside the **A** box:

$h_t = f(x_t, h_{t-1})$

$y_t = g(h_t)$

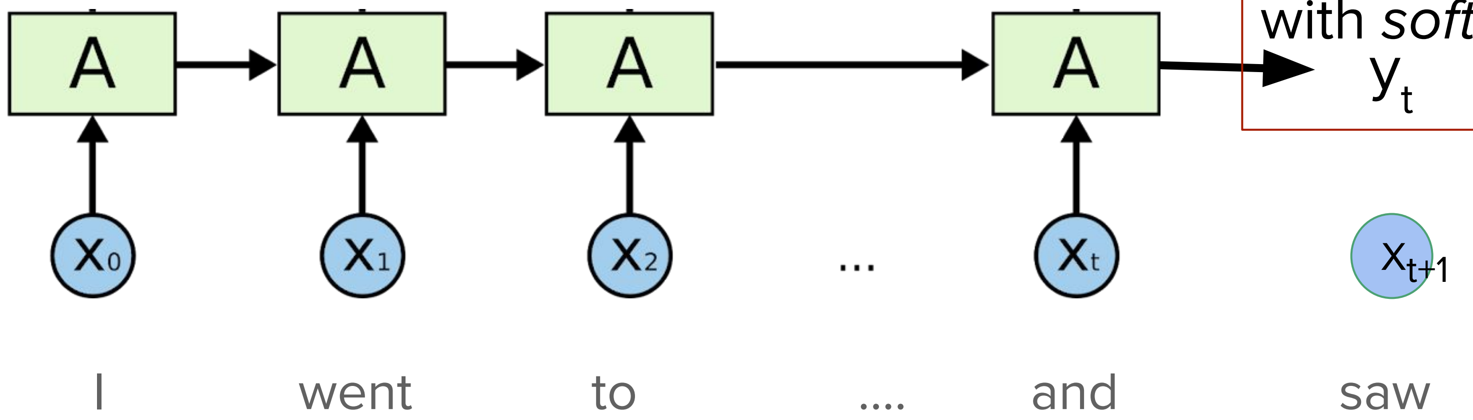
RNN unrolled



input concatenates x_t, h_{t-1}

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Language model



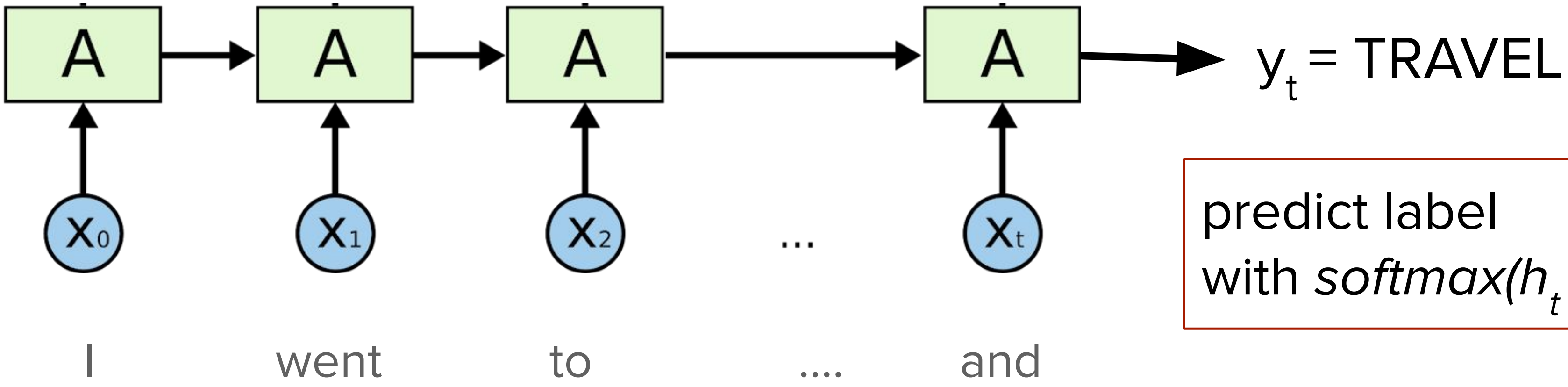
predict next word with *softmax(h_t)*

y_t

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequence Modeling

Sequence labeling

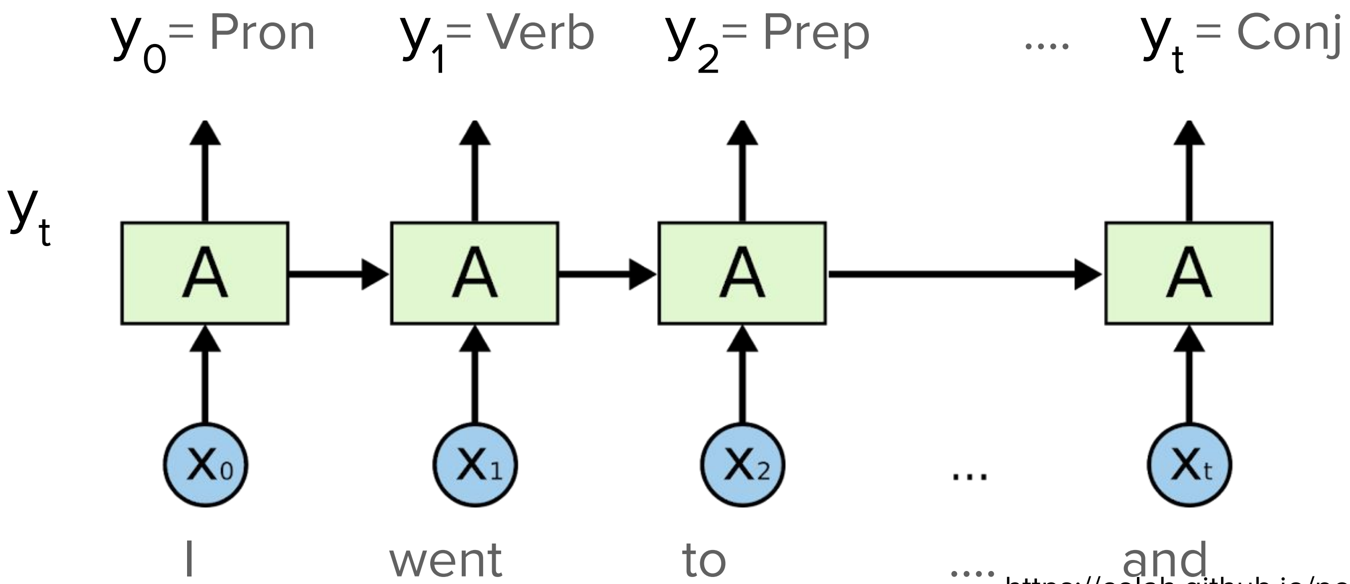


predict label with *softmax(h_t)*

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Tagging

Tagging



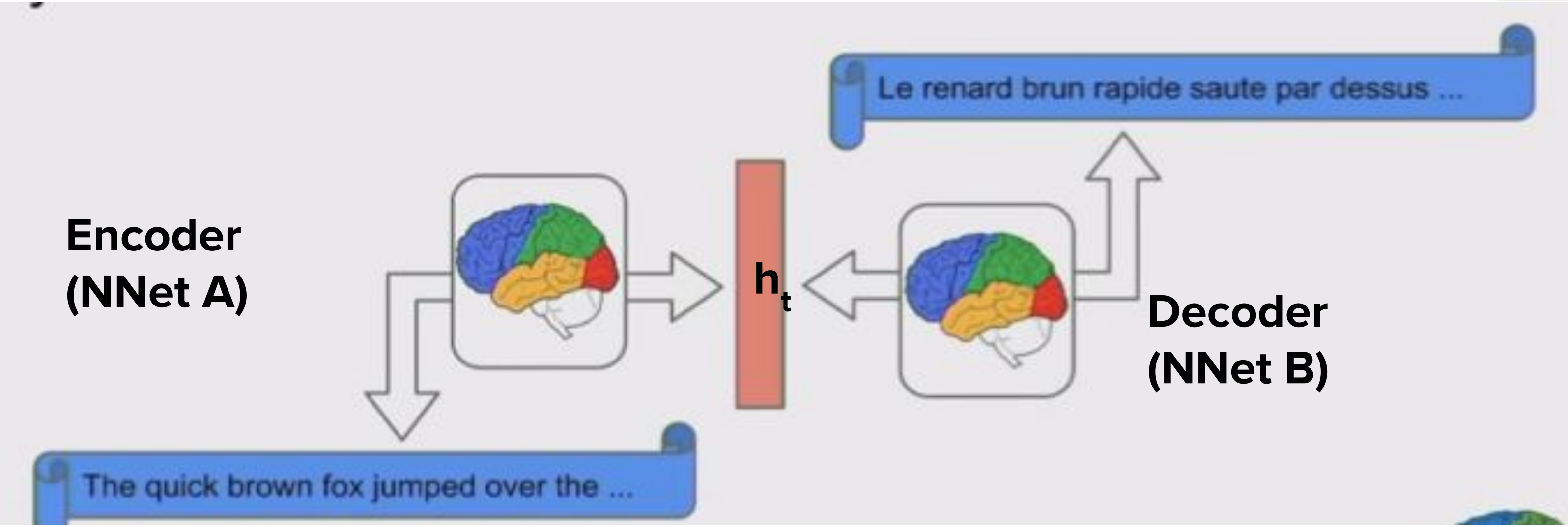
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

predict labels with *softmax*(h_t)

Seq2seq

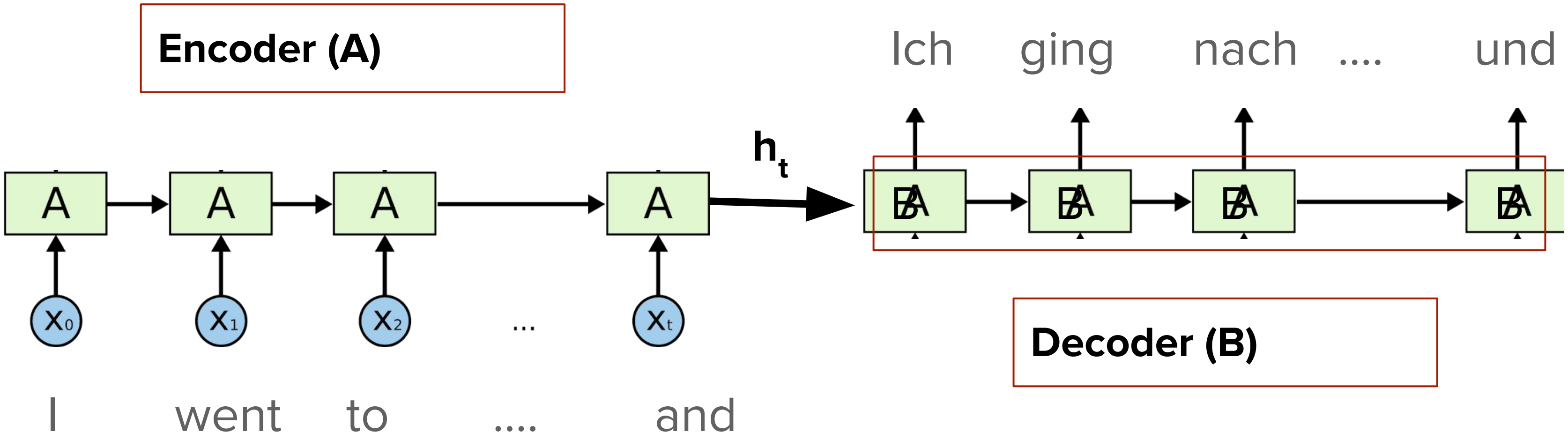
Language generation

Seq2seq



Jeff Dean, google

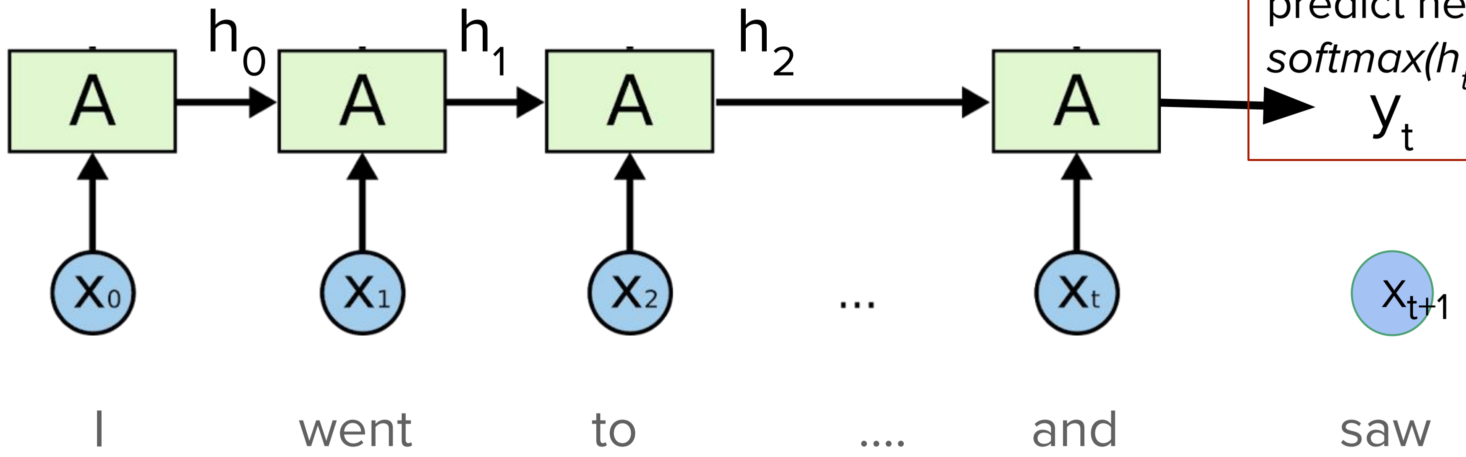
Sequence to Sequence



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Technical Details

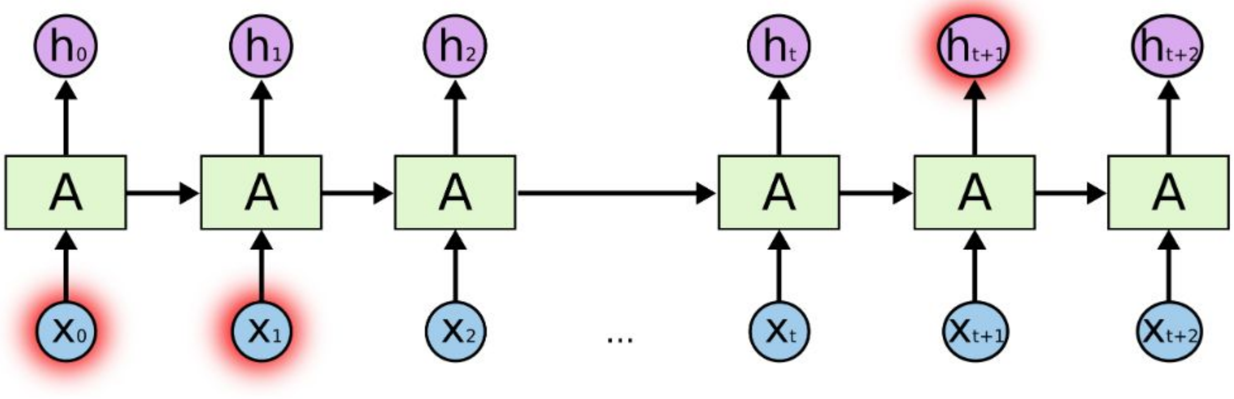
Language model



predict next word with
softmax(h_t)

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs have trouble doing gradient descent



Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Student Member, IEEE*

Gradients explode or vanish

For exploding gradients

use tanh

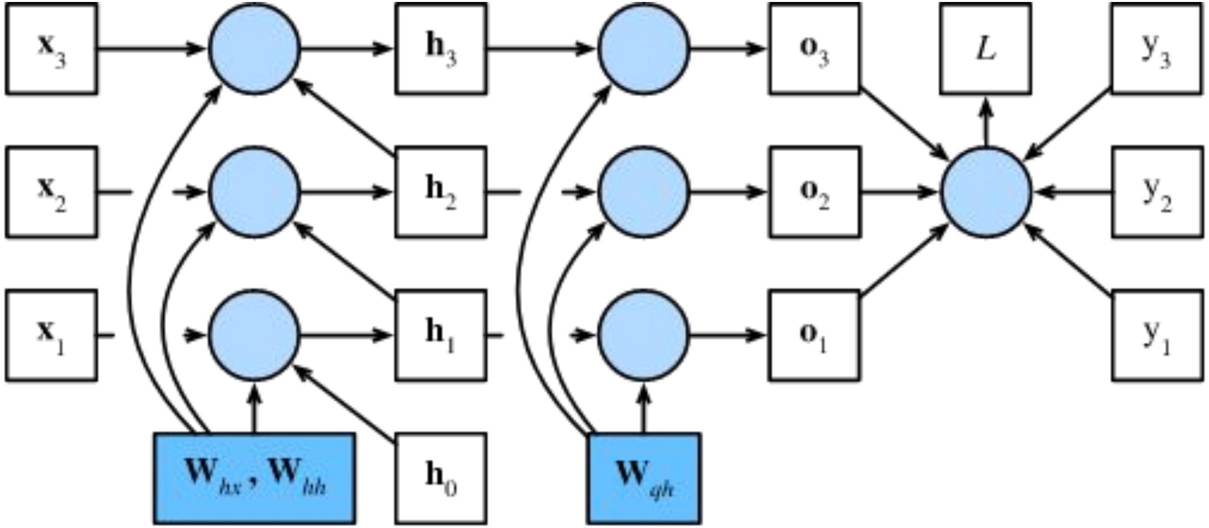
use gradient clipping

For vanishing gradients

use gated networks

train for a very long time

The exact unrolled architecture depends on the learning problem



Computational graph showing dependencies for an RNN model with three time steps. Boxes represent variables (not shaded) or parameters (shaded) and circles represent operators. From d2l.ai

Backpropagation through time (BPTT)

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1},\\ \mathbf{o}_t &= \mathbf{W}_{qh}\mathbf{h}_t,\end{aligned}$$

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{o}_t, y_t).$$

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial l(\mathbf{o}_t, y_t)}{T \cdot \partial \mathbf{o}_t} \in \mathbb{R}^q$$

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{qh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t}\mathbf{h}_t^\top$$

see d2l.ai

Backpropagation through time (BPTT)

$$\frac{\partial L}{\partial \mathbf{h}_T} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_T}, \frac{\partial \mathbf{o}_T}{\partial \mathbf{h}_T}\right) = \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_T}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_{t+1}}, \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right) + \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right) = \mathbf{W}_{hh}^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_t}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} \left(\mathbf{W}_{hh}^\top\right)^{T-i} \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

$$\frac{\partial L}{\partial \mathbf{W}_{hx}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hx}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^\top,$$

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^\top,$$

GRUs and LSTMs

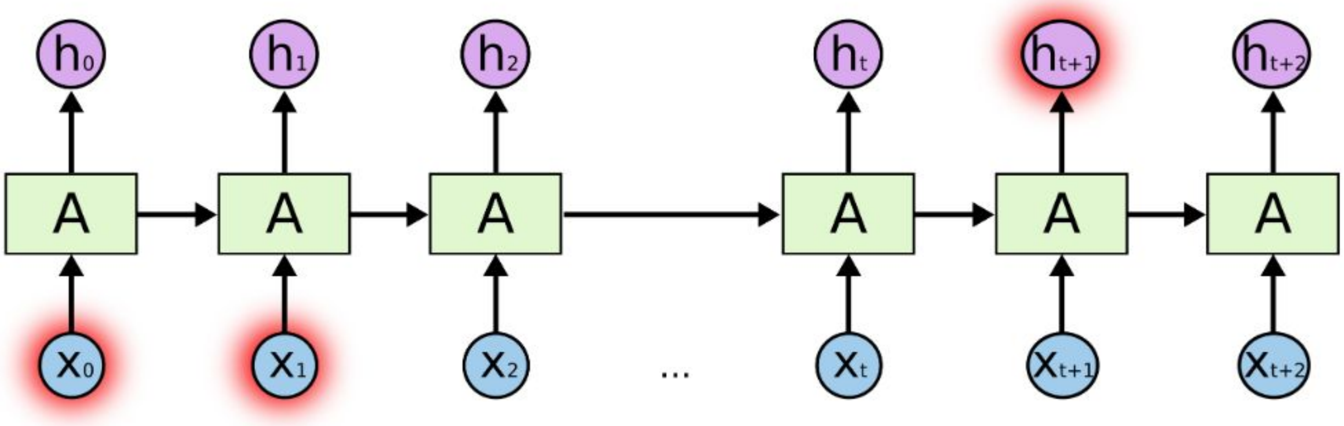
Longer term memory

Gated Recurrent Units

Long Short Term Memory nets

RNNs forget exponentially fast

Like a hidden markov model



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

A ‘simple’ task

A relevant sequence (length L)

Followed by an irrelevant sequence (length T>>L)

Answer at end

solution idea: figure out relevant information. Then remember it for L steps.

Controlling remembering and forgetting

We want the RNN to choose

which hidden states to store

when to store them

when to retrieve them

when to forget them

Note: this feels like discrete logic

we can approximate it with a tanh pushed to +/- 1

and estimate with gradient descent

So use LSTMs or Transformers

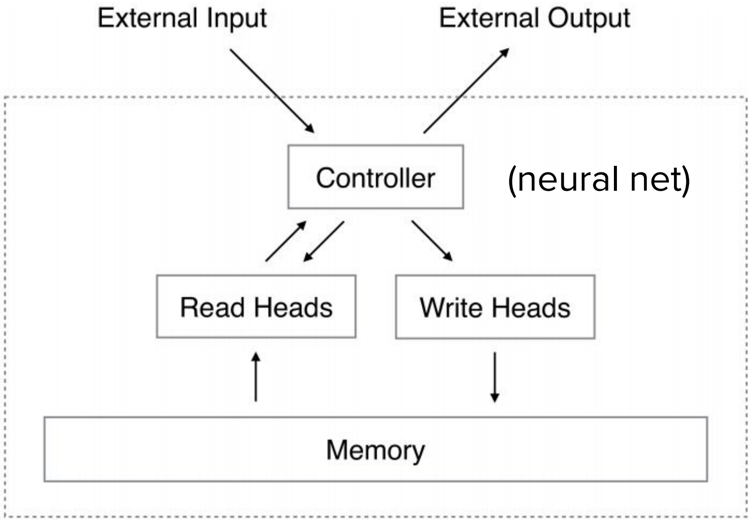
LSTM

Transformers

Next class!

Or external memory: Neural Turing Machines

Write to and read from a separate memory bank



Neural Turing Machines

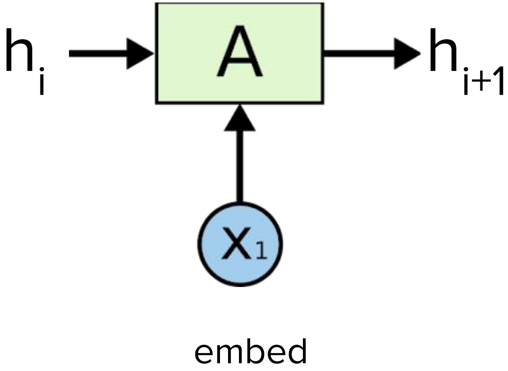
Alex Graves gravesa@google.com
Greg Wayne gregwayne@google.com
Ivo Danihelka danihelka@google.com

Summary

What we learned

Time series: variable-length problems

- Bag of words
  - Average the embeddings of the words
- Truncate or pad
  - Make all inputs be the same size
- Use recurrence
  - Hidden state is a function of current observation and previous hidden state



word

Embeddings capture distributional similarity

- **Context oblivious**
  - Word2vec, FastText, …
  - Can be multilingual or multi-modal
  - “Similar” words are close in embedding space
- **Context sensitive**
  - RNNs and Transformers (BERT, Roberta …)
  - These ake a sequence of context-oblivious embeddings (usually of word-parts) as input
  - Learn “hidden states”, which are context-sensitive embeddings

RNNs

- Learn a mapping from input sequence to hidden state
  - Which is a context-sensitive embedding
- People mostly use GRU/LSTM/biLSTM to reduce forgetting problems
- Language models are the basis for semi-supervised learning

	Input		**Output**
language models	subsequence	➜	next item in sequence
sequence labeling	sequence	➜	label of entire sequence
tagging	subsequence	➜	label of current item
seq2seq	sequence	➜	other sequence

RNNs are often replaced by transformers

- RNNs require backpropagation through time (unrolling)
  - For a memory of n time steps, requires O(n) sequential operations on the GPU
- Transformers (covered next class)
  - Use truncation/padding to learn in a “single step”
    - “Mask” some words and predict them
  - Fast on current GPUs