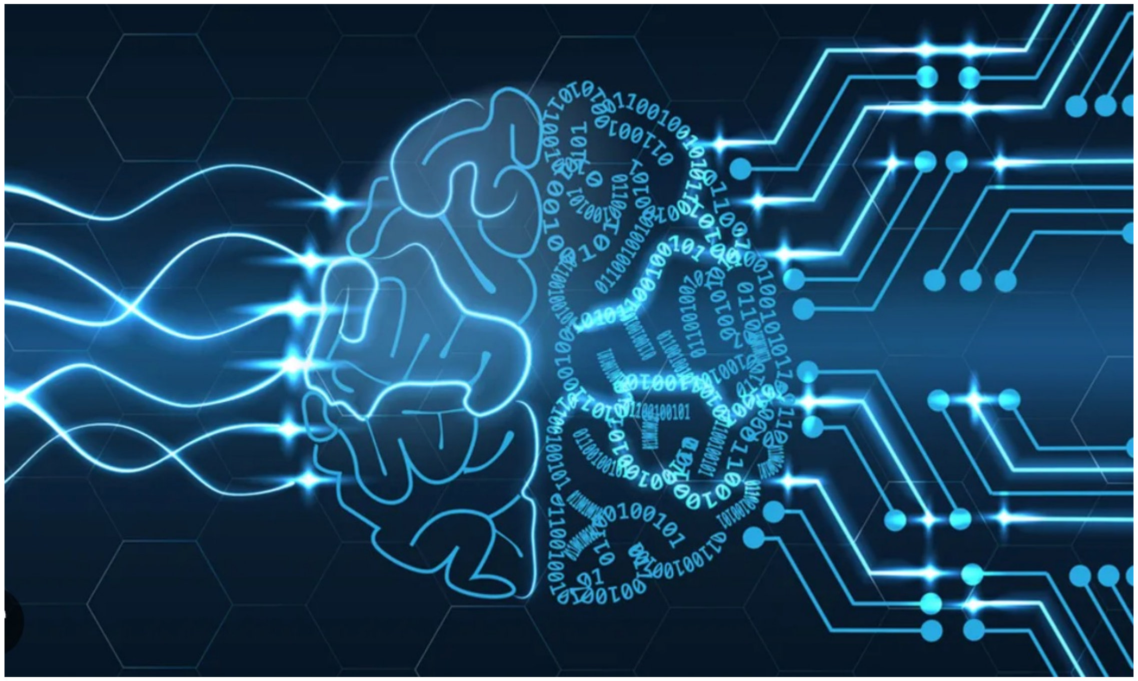
CS 4100: Introduction to AI

Wayne Snyder
Northeastern University

Lecture 22 -- Deep Learning -- Language Models, Sequence Data, and Recurrent NNs.



Plan for this week

Plan for this Lecture:

- Sequence data: Time series, audio, text
- Language Models: N-Gram Models
- Adding memory to NNs: Recurrent NNs

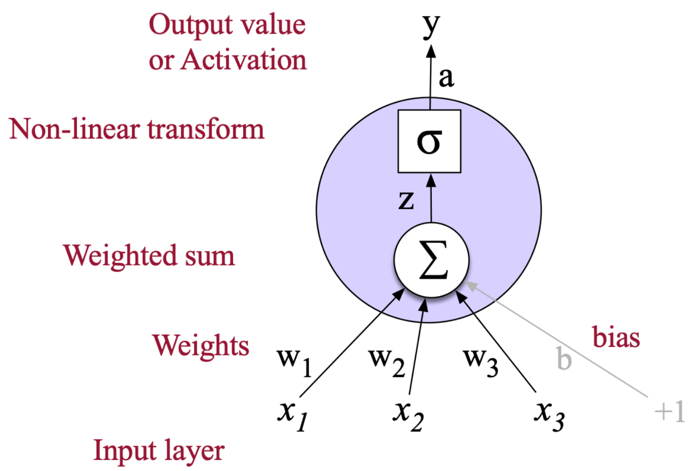
Wednesday's Lecture:

- RNNs, GRUs, LSTMs
- Adding Atttention: BERT, GPT

Recall: Feedforward Neural Networks

(With a little bit of Linear Algebra in Python thrown in......

Recall that a single neuron does logistic regression using the dot product of a vector of weights and a vector of input values, sends the sum through a non-linear activation function f, and outputs the result to the next layer:

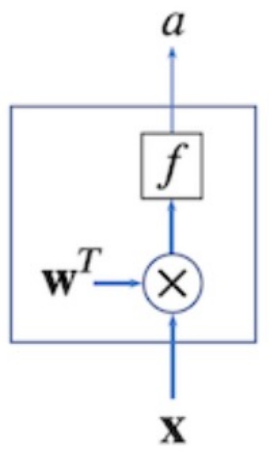


$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

$$\mathbf{x} = (x_1, x_2, \dots x_k).$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

$$\mathbf{w} = (w_1, w_2, \dots w_k).$$

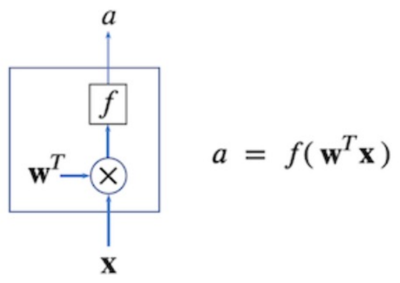


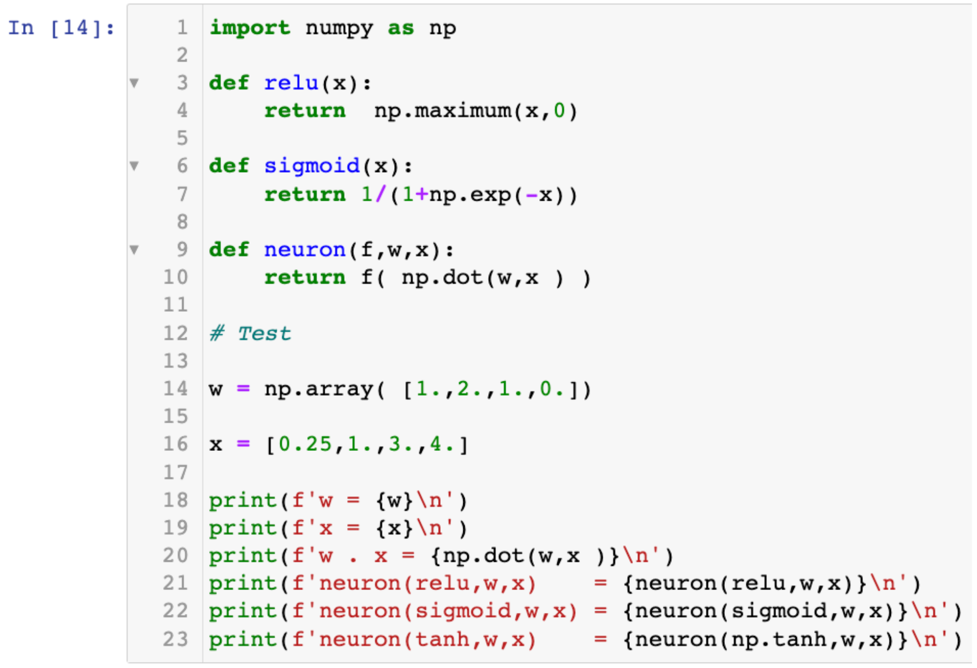
$$a = f(\mathbf{w}^T\mathbf{x})$$

Recall: Feedforward Neural Networks

(With a little bit of Linear Algebra in Python thrown in......

This is nearly trivial to implement in Python





```
In [14]:
import numpy as np

def relu(x):
    return  np.maximum(x,0)

def sigmoid(x):
    return 1/(1+np.exp(-x))

def neuron(f,w,x):
    return f( np.dot(w,x ) )

# Test

w = np.array( [1.,2.,1.,0.])

x = [0.25,1.,3.,4.]

print(f'w = {w}\n')
print(f'x = {x}\n')
print(f'w . x = {np.dot(w,x )}\n')
print(f'neuron(relu,w,x)    = {neuron(relu,w,x)}\n')
print(f'neuron(sigmoid,w,x) = {neuron(sigmoid,w,x)}\n')
print(f'neuron(tanh,w,x)    = {neuron(np.tanh,w,x)}\n')
```

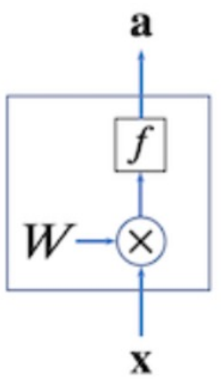
```
w = [1. 2. 1. 0.]

x = [0.25, 1.0, 3.0, 4.0]

w . x = 5.25

neuron(relu,w,x)    = 5.25

neuron(sigmoid,w,x) = 0.9947798743064417

neuron(tanh,w,x)    = 0.9999449286177708
```

Recall: Feedforward Neural Networks

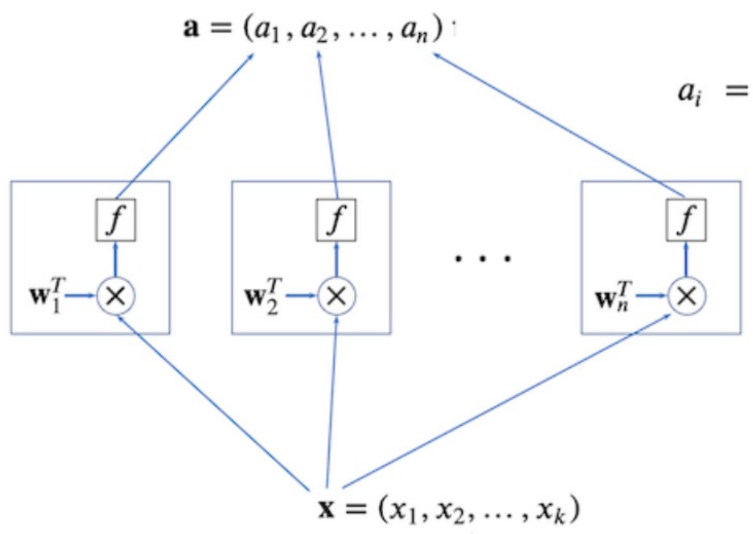
(With a little bit of Linear Algebra thrown in......

Feed-Forward Layer just applies this dot-product-then-activation-function in parallel, using matrix multiplication, and activation functions applied elementwise:

$$W\,\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,k} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \vdots \\ \mathbf{w}_n^T\mathbf{x} \end{bmatrix}.$$

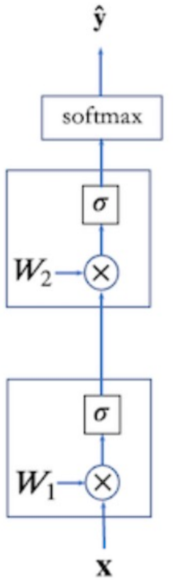


$$\mathbf{a} \;=\; f(\,W\mathbf{x}\,)$$



$\mathbf{a} = (a_1, a_2, \ldots, a_n)$
f
$\mathbf{w}_1^T$
$\mathbf{w}_2^T$
$\mathbf{w}_n^T$
. . .
$\mathbf{x} = (x_1, x_2, \ldots, x_k)$

$$a_i \;=\; f(\,\mathbf{w}_i^T\mathbf{x}\,)$$

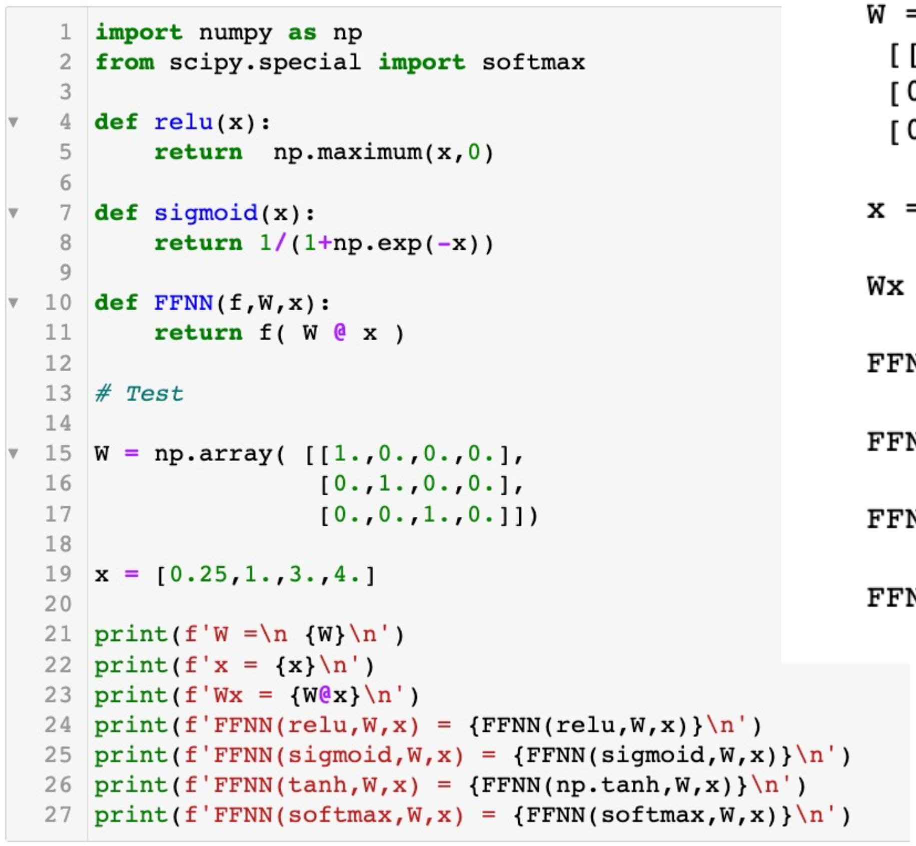


All the matrices are parameters that need to be learned during training.

Sequence Data: A problem for FF NNs

(With a little bit of Linear Algebra thrown in......)

If you know a little linear algebra in Python, this is again trivial to implement:



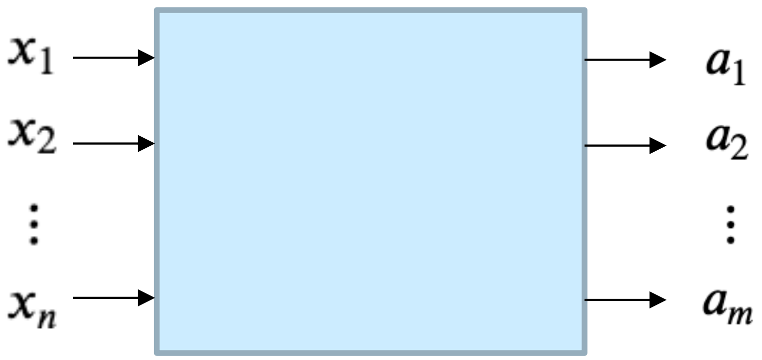
```python
import numpy as np
from scipy.special import softmax

def relu(x):
    return  np.maximum(x,0)

def sigmoid(x):
    return 1/(1+np.exp(-x))

def FFNN(f,W,x):
    return f( W @ x )

# Test

W = np.array( [[1.,0.,0.,0.],
               [0.,1.,0.,0.],
               [0.,0.,1.,0.]])

x = [0.25,1.,3.,4.]

print(f'W =\n {W}\n')
print(f'x = {x}\n')
print(f'Wx = {W@x}\n')
print(f'FFNN(relu,W,x) = {FFNN(relu,W,x)}\n')
print(f'FFNN(sigmoid,W,x) = {FFNN(sigmoid,W,x)}\n')
print(f'FFNN(tanh,W,x) = {FFNN(np.tanh,W,x)}\n')
print(f'FFNN(softmax,W,x) = {FFNN(softmax,W,x)}\n')
```

```
W =
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]

x = [0.25, 1.0, 3.0, 4.0]

Wx = [0.25 1.   3.  ]

FFNN(relu,W,x) = [0.25 1.   3.  ]

FFNN(sigmoid,W,x) = [0.5621765  0.73105858 0.95257413]

FFNN(tanh,W,x) = [0.24491866 0.76159416 0.99505475]

FFNN(softmax,W,x) = [0.05330595 0.1128487  0.83384535]
```

W =

Sequence Data: A problem for FF NNs

A Feed-Forward Neural Network learns a function from vectors to vectors:



Input vectors may be one-hot word vectors, pixel densities, etc. etc.

Note: The order of the inputs does not matter!

Sequence Data: A problem for FF NNs

HOWEVER, many kinds of data are inherently sequential, often as a time series:

Words in a sentence:

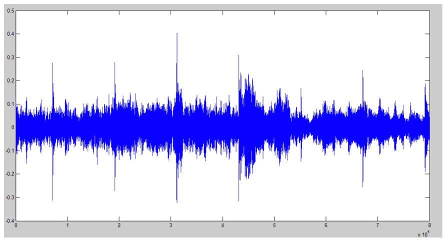
The matters order

Order the matters

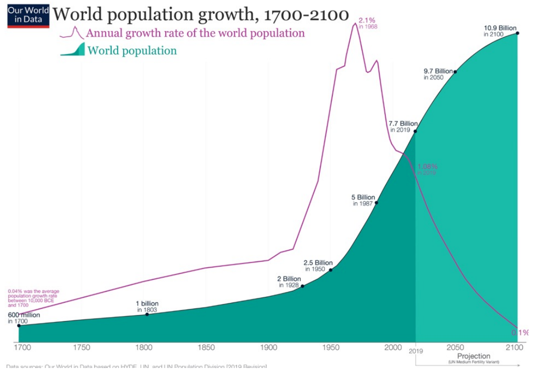
Matters order the

The order matters

Audio:



Population:



Stock prices:

Stock Market Indexes
S&P 500 (Right Axis)
Dow Jones Industrial Average (Left Axis)

We will first consider a simple way of dealing with sequence information, using N-Grams model in NLP; then we shall consider a more essential fix to the notion of a layer in a network.

Language Models

A Language Model is a simplified representation of a language which facilitates an NLP task, where

- A language (potentially infinite) is approximated by a (finite) data set; and
- The model is a set of (simplified) assumptions about the language, embodied by the algorithms and data structures of your program. It may be

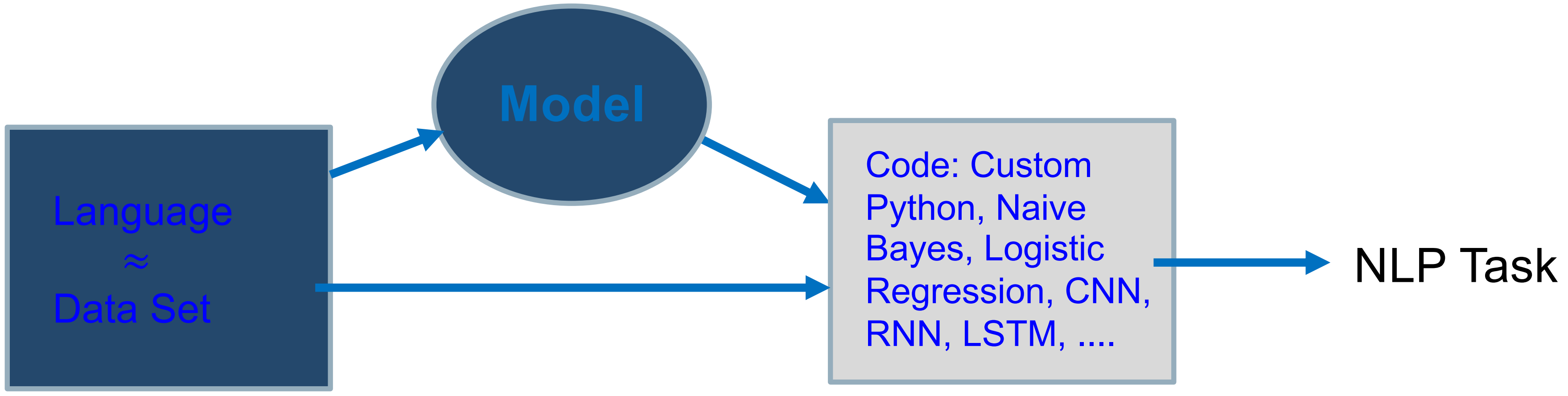

Model
Language
≈
Data Set
Code: Custom Python, Naive Bayes, Logistic Regression, CNN, RNN, LSTM, ...
NLP Task

Image source: medium.com

Language Models: BOWs

The **Bag of Words (BOW)** model represents a text (sentence, sequence of words, entire corpus) by a multiset (bag) of all words in the text, i.e, just the **vocabulary**, no information about order of words! Sometimes BOW also refers to simply sets of words (without the multiplicity).

```
(1) John likes to watch movies. Mary likes movies too.
```

```
(2) Mary also likes to watch football games.
```

```
"John","likes","to","watch","movies","Mary","likes","movies","too"

"Mary","also","likes","to","watch","football","games"
```

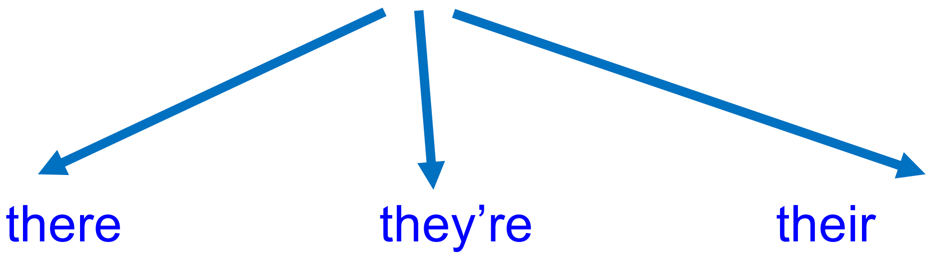
BoW1 =
{"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mar
y":1,"too":1};
BoW2 =
{"Mary":1,"also":1,"likes":1,"to":1,"watch":1,"footbal
l":1,"games":1};

Language Models: Probabilistic Models

Probabilistic models: Assign a probability to text components (letters, words, sentences, ...)   E.g., ambiguity problems can be recast as

"*given N choices for some ambiguous input, choose the most probable one*"

"I went to they're house on Sunday."

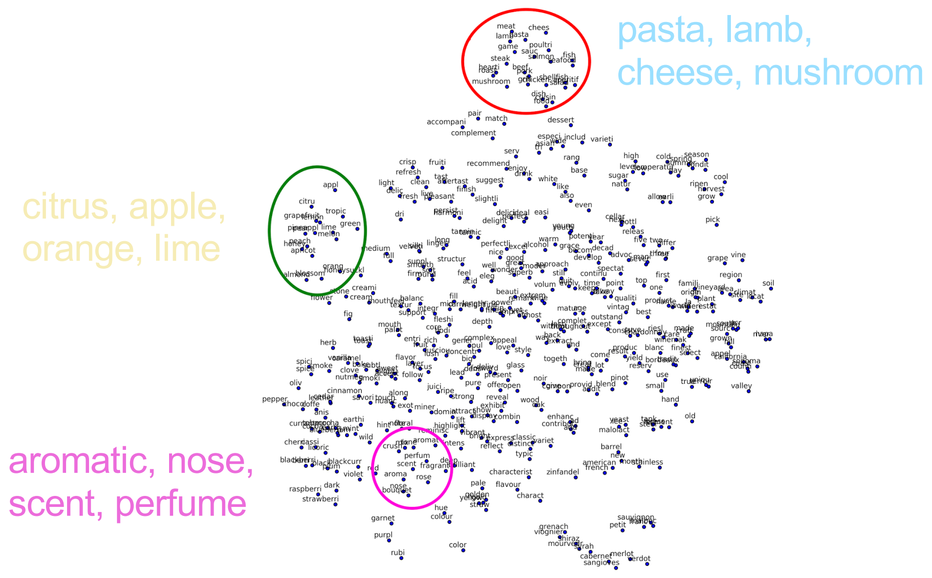


"I went to their house on Sunday.
the bus.
holiday.
a very rainy day.

Which is most likely?

Language Models: Vector Models

- **Vector space models capture word meanings "you shall know a word by the company it keeps (Firth, J. R. 1957:11)"**



http://methodmatters.blogspot.com/2017/11/using-word2vec-to-analyze-word.html

Probabilistic Language Modeling

- **Why is this useful?**
  - Machine Translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - Text Generation: "I", "I saw", "I saw a", "I saw a van."
  - + Summarization, question-answering, etc., etc.!!

Probabilistic Language Modeling

- **Goal: compute the probability of a sentence or sequence of words:**

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

- **Related task: probability of an upcoming word:**

  $P(w_5|w_1,w_2,w_3,w_4)$

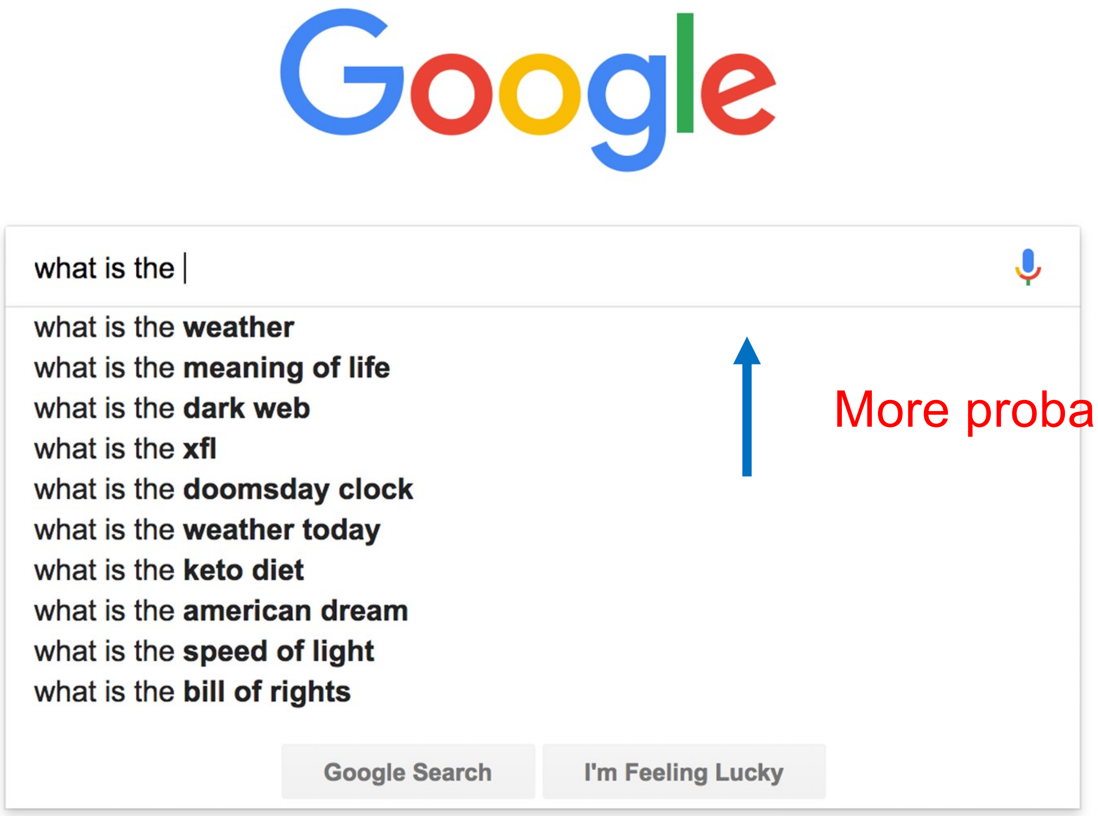
- **A model that computes either of these:**

  $P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$

  is called a **language model** (with **probabilistic** assumed).

Probabilistic Language Modeling

- You have seen these before!



what is the |

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

More probable.

Probabilistic Language Modeling

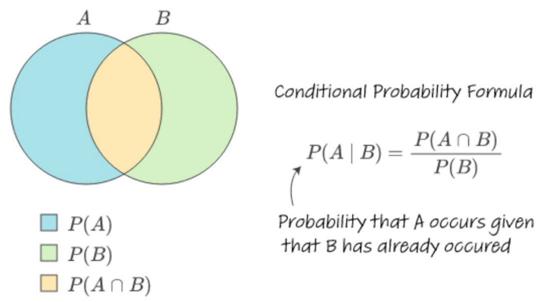
Remark:

It is possible to apply this technique to any sequence, e.g.,

- Letters in a word; *
- Pitches in a melody;
- Phonemes in a voice signal;
- Topics in a discourse.

Probabilistic Language Modeling

- How to compute the joint probability of a sentence:
  - P(I went to their house on Sunday)


A
B
Conditional Probability Formula
$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$
Probability that A occurs given that B has already occured
$P(A)$
$P(B)$
$P(A \cap B)$

- Recall the definition of conditional probabilities
  - p(B|A) = P(A and B) / P(A)   Rewriting: P(A and B) = P(A) $*$ P(B|A)
- More variables:
  - P(A and B and C and D) = P(A) $\times$ P(B|A) $\times$ P(C|A and B) $\times$ P(D|A and B and C)
- Etc.

Probabilistic Language Modeling

The Chain Rule applied to compute joint probability of words in sentence, say “I went to their house on Sunday.”

$$P(w_1 w_2 \ldots w_n) = \prod_{1 \le i \le n} P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P(I went to their house on Sunday.)=

P(I) × P(went|I) × P(to|I went) × P(their|I went to)

× P(house|I went to their) × P(on|I went to their house)

x P(Sunday|I went to their house on)

Probabilistic Language Modeling

Could we just count and divide?

$$P(\text{Sunday} \mid \text{I went to their house on}) = \frac{\text{Count}(\text{I went to their house on Sunday})}{\text{Count}(\text{I went to their house on})}$$

Probabilistic Language Modeling

Can we just count? Not realistic:

In an infinite set of sentences, the probability of any distinct sequence of words is 0. So a data set is a small sample which hopefully represents the essential features of the language.

But realistic data sets never have enough sample sequences, and sequences might be very long or simply not exist in your data.

"I have been here before," I said; I had been there before; first with Sebastian more than twenty years ago on a cloudless day in June, when the ditches were white with fools' parsley and meadowsweet and the air heavy with all the scents of summer; it was a day of peculiar splendor, such as our climate affords once or twice a year, when leaf and flower and bird and sun-lit stone and shadow seem all to proclaim the glory of God; and though I had been there so often, in so many moods, it was to that first visit that my heart returned on this, my latest." Evelyn Waugh: *Brideshead Revisited, first sentence*.

Probabilistic Language Modeling

Solution: Use a finite memory of the last N words (cf. the Markov Property, where N = 0)

Terminology: An N-Gram is a sequence of N contiguous words from the data set.

unigram = 1-gram, bigram = 2-gram, trigram = 3-gram, etc.

So, if N = 2

I went to their house on Sunday

- I went
- went to
- to their
- their house
- house on
- on Sunday

If N = 3

I went to their house on Sunday

- I went to
- went to their
- to their house
- their house on
- house on Sunday

Probabilistic Language Modeling

Markov Assumption: Only consider N-1 words of left context.

Thus, for a sequence of length M,

$$P(w_1 w_2 \ldots w_M) \approx \prod_{N \le i \le M-N} P(w_i | w_{i-N+1} \ldots w_{i-1})$$

For small N, it is reasonable to count the number of N-Grams. Typical values are $1 \le N \le 5$. It is usual to add a beginning <s> and ending token </s> to sentences.

Probabilistic Language Modeling

Bigram Example (N = 2)

P(<s> I went to their house on Sunday </s>) =

P(I |<s>) × P(went | I) × P(to | went) × P(their | to)

× P(house | their) × P(on | house)

x P(Sunday | on) x P(</s> | Sunday)

$$P(\text{I} \mid \text{<s>}) \approx \frac{C(\text{<s> I})}{C(\text{<s>})}$$

$$P(\text{went} \mid \text{I}) \approx \frac{C(\text{I went})}{C(\text{I})}$$

Note that this calculation involves finding the number of occurrences of an N-gram and of an (N-1)-gram (the prefix)!

Probabilistic Language Modeling

Trigram Example (N = 3)

P(<s> I went to their house on Sunday </s>)=

P(I |<s>) × P(went | I) × P(to | went) × P(their | to)

× P(house | their) × P(on | house)

x P(Sunday | on) x P(</s> | Sunday)

$$P(\text{ went} \mid \text{<s> I }) = \frac{C(\text{ <s> I went })}{C(\text{ <s> I })}$$

$$P(\text{ to} \mid \text{I went }) = \frac{C(\text{ I went to })}{C(\text{ I went })}$$

Probabilistic Language Modeling

A bigram example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

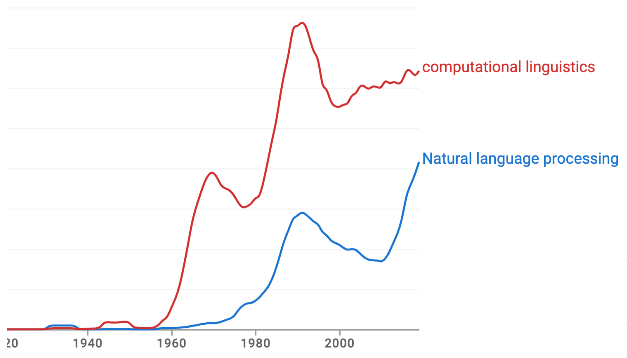
Probabilistic Language Modeling

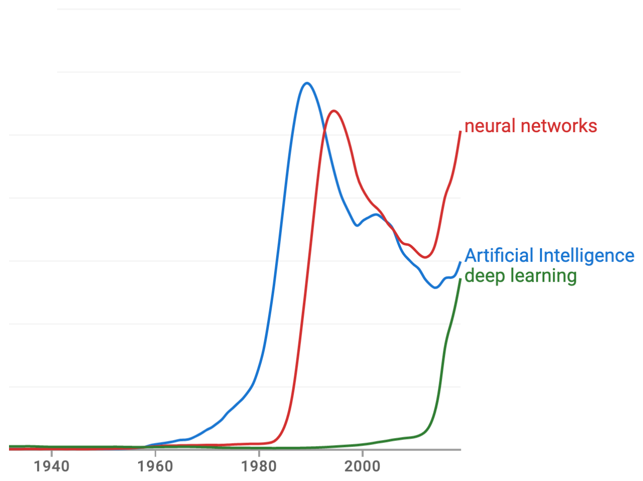
Remarks:

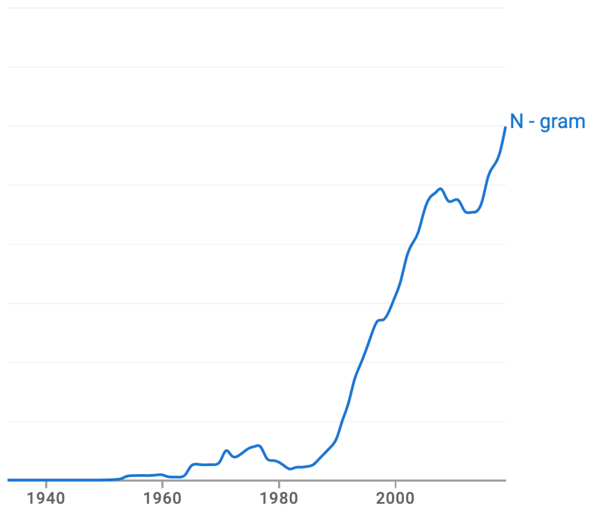
This is almost trivial to code after you have separated your text into words and sentences.

For small N, it will be reasonably efficient. Check out the Google N-Gram viewer:

https://books.google.com/ngrams/







Probabilistic Language Modeling

Sentence generation using N-Grams

A clever feature of this model is that it can easily generate sentences.

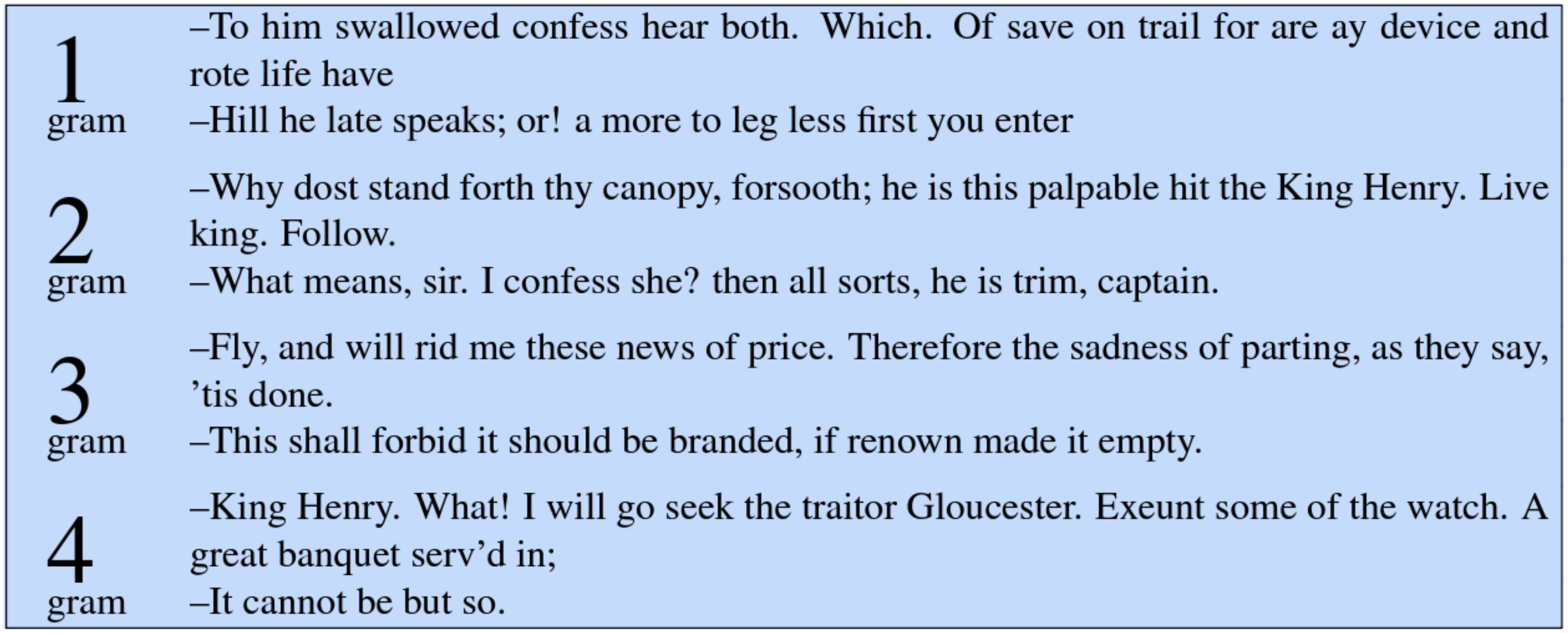
For bigrams, after calculating the probability of all bigrams appearing in the data.

Pick a probable* bigram <s> w_1
Pick a probable bigram w_1 w_2
.... etc. ...
End when you generate a bigram w_k </s>

* You may not want to always choose the most likely, or you will not be able to generate many sentences!

Probabilistic Language Modeling

Textbook examples of sentence generation from Shakespeare:

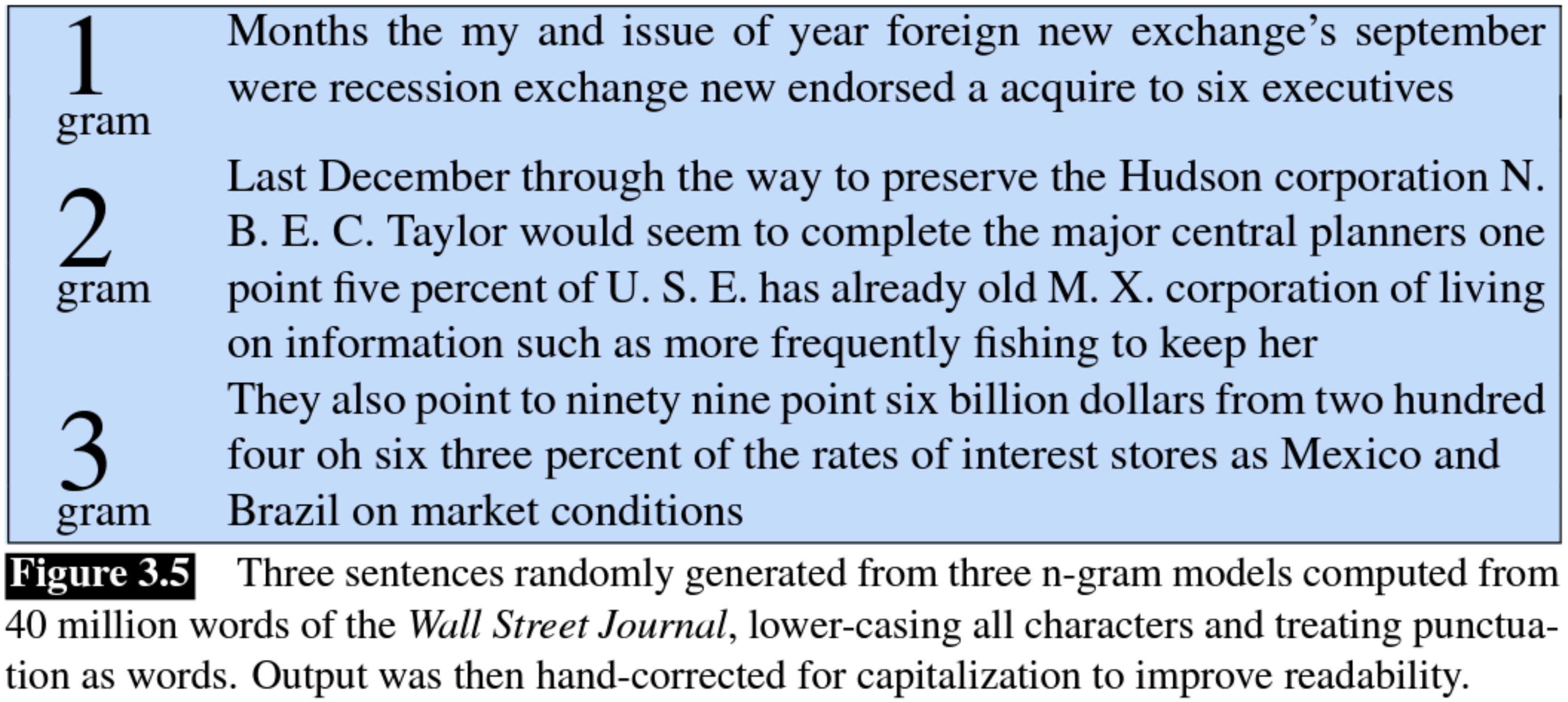


1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Figure 3.4 Eight sentences randomly generated from four n-grams computed from Shakespeare’s works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

Probabilistic Language Modeling

Textbook examples of sentence generation from the Wall Street Journal:



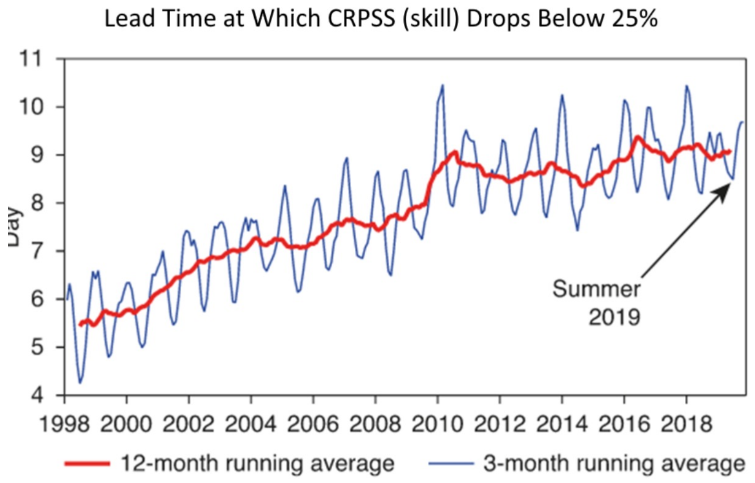
1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

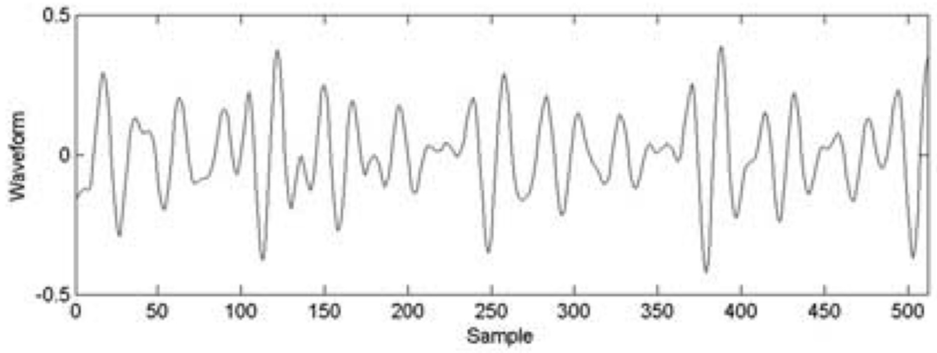
Figure 3.5 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

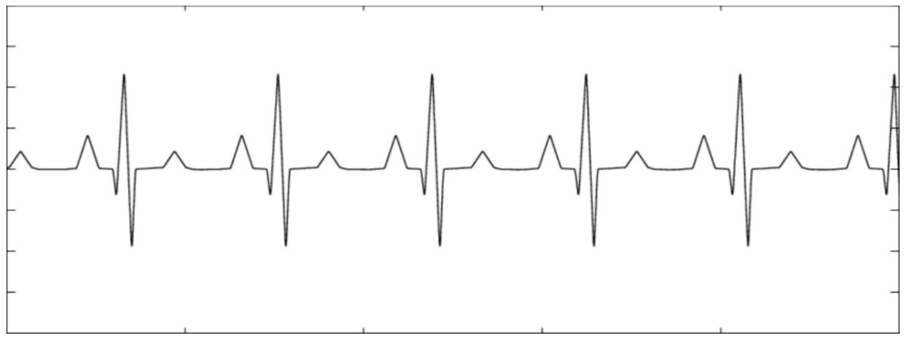
Probabilistic Language Modeling

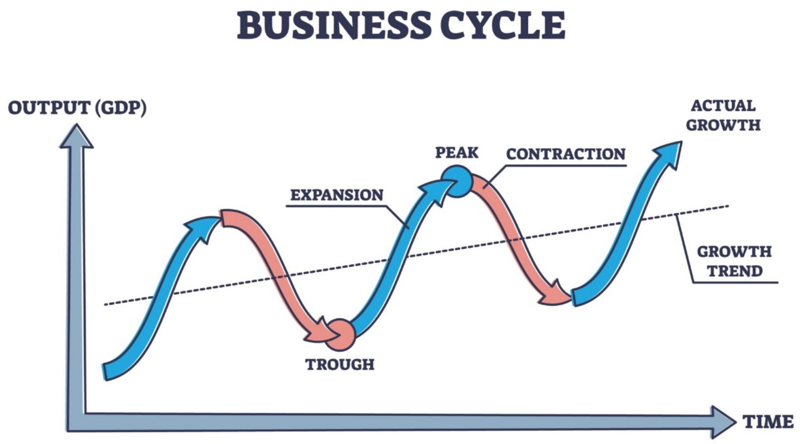
The problem with N-Gram models, and with FF NNs in general: long-range dependencies that are not captured by an N-gram for small N:

Many time series have periodic components that are not captured by small samples of size N:






Maternal heartbeat signal.



Probabilistic Language Modeling

The problem with N-Gram models, and with FF NNs in general: long-range dependencies that are not captured by an N-gram for small N:

Natural language is strongly sequential, and has long-range dependencies:

I don’t think this movie is very good.

This movie, although I thought the prequel was excellent (and I was looking forward to the next installment in the series), is not very good.

'I have been here before,' I said; I had been there before; first with Sebastian more than twenty years ago on a cloudless day in June, when the ditches were creamy with meadowsweet and the air heavy with all the scents of summer; it was a day of peculiar splendour, and though I had been there so often, in so many moods, it was to that first visit that my heart returned on this, my latest. -Brideshead Revisited, Evelyn Waugh

Probabilistic Language Modeling

The problem with feedforward NNs

So far, our representations for texts have either

- Completely ignored sequencing (BOW, TFIDF, mean of embeddings, cosine similarity); or
- Accounted for very short

  (N-Grams):

Clearly we need better methods

to deal with sequenced data!

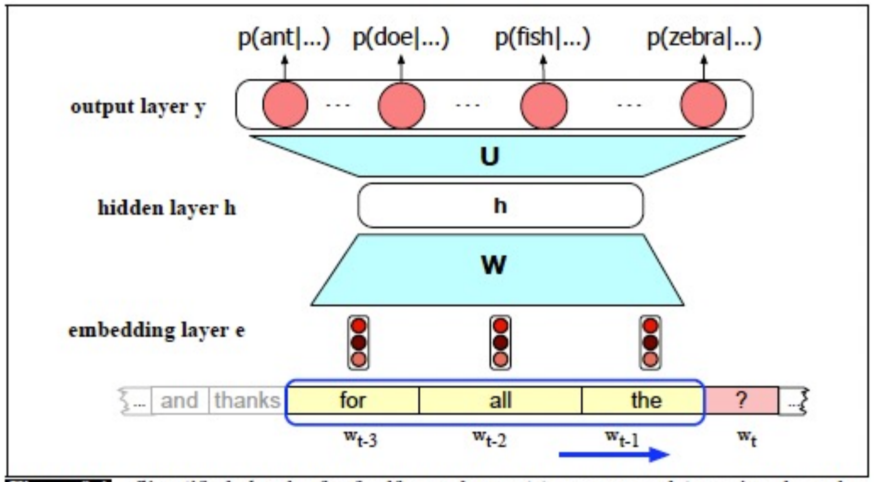
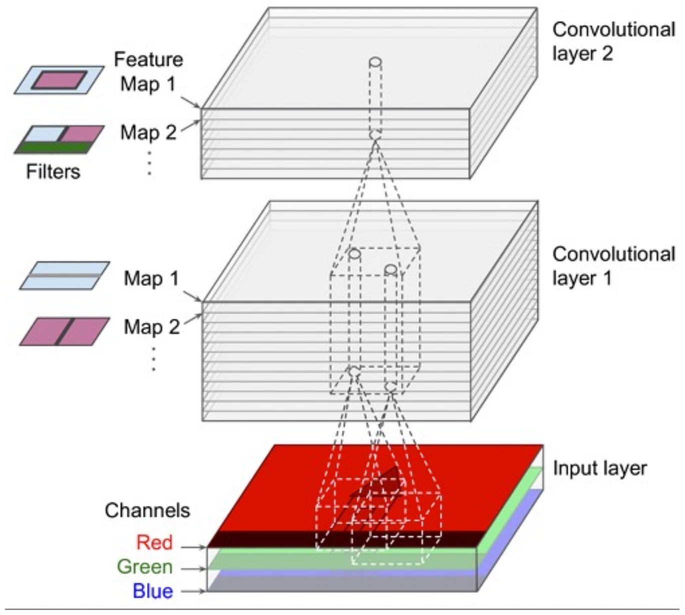


Figure 9.1 Simplified sketch of a feedforward neural language model moving through a text. At each time step t the network converts N context words, each to a d-dimensional embedding, and concatenates the N embeddings together to get the $Nd \times 1$ unit input vector **x** for the network. The output of the network is a probability distribution over the vocabulary representing the model's belief with respect to each word being the next possible word.

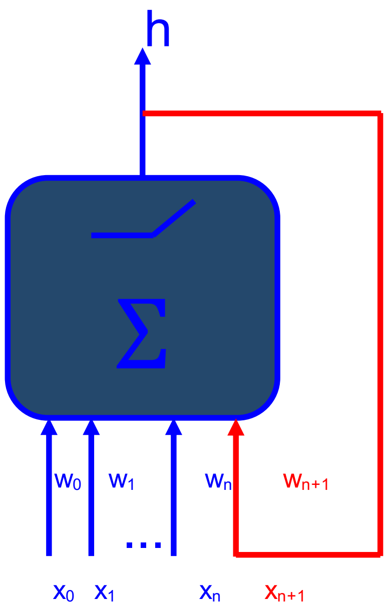
Recurrent Neural Network

One (partial) solution was CNNs



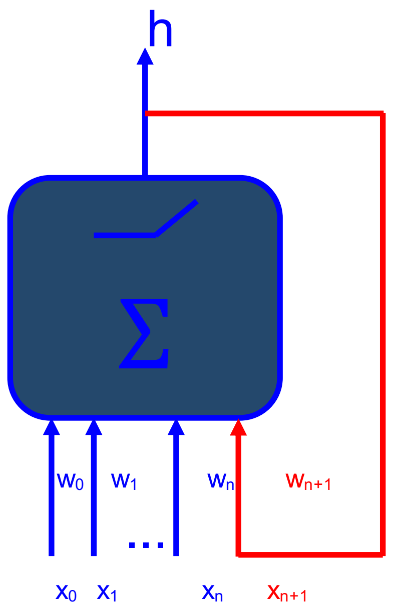
Recurrent Neural Network

- Basic unit is same with one important change: the output is fed back into the inputs:

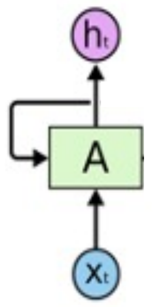


Recurrent Neural Network

- Basic unit is same with one important change: the output is fed back into the inputs:



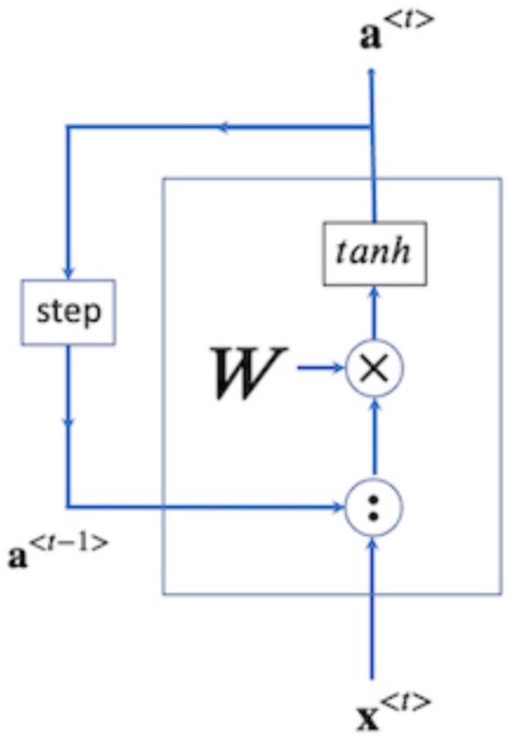
You will see lots of less precise diagrams used which are equivalent:



(Just be sure to note the direction of the arrows, that all outputs have the same value, and that each input has a weight $\mathbf{w_i}$ attached to it!)

Recurrent Neural Network

In Linear Algebra:



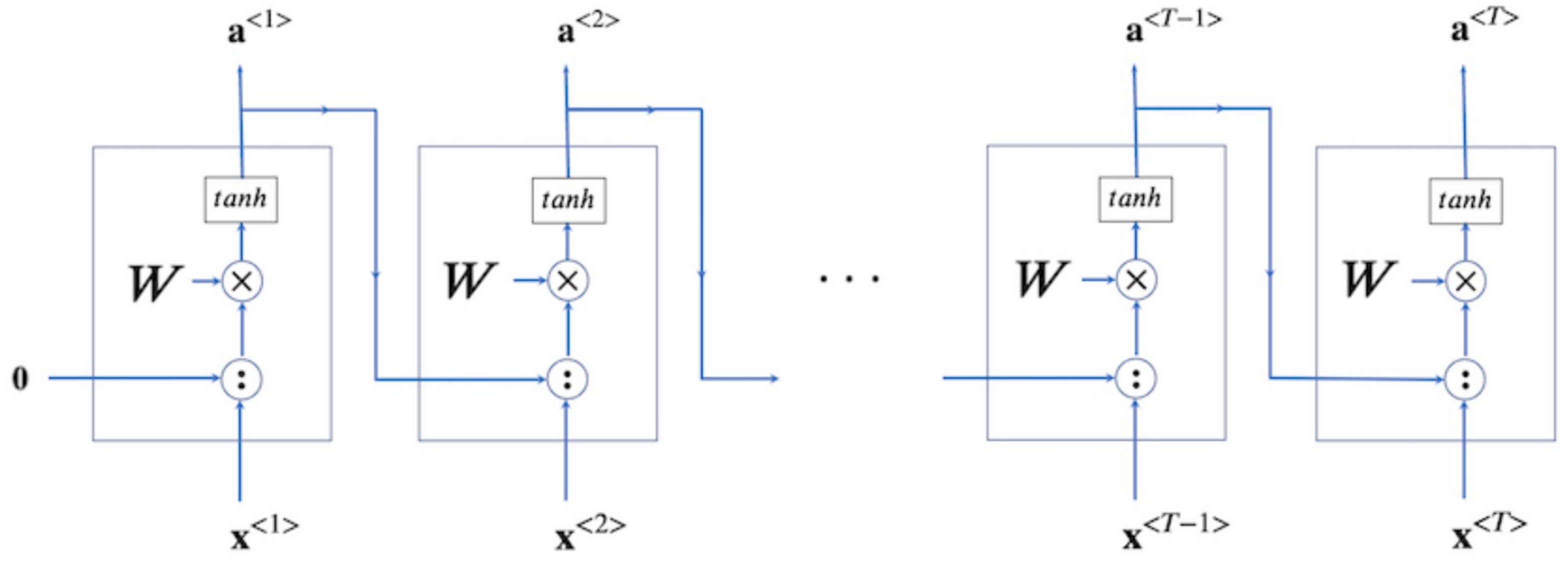
```
$\mathbf{a}^{<0>} = \mathbf{0}$
for t = 1,2,... ,T:
    $\mathbf{a}^{<t>} = tanh(W(\mathbf{a}^{<t-1>} : \mathbf{x}^{<t>}))$
```

$$W = \begin{bmatrix} & \\ \mathbf{w}'_2 & \mathbf{w}_2 \\ \vdots & \\ \mathbf{w}'_n & \mathbf{w_n} \end{bmatrix} = \begin{bmatrix} & & & w'_{1,n} & w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w'_{2,1} & w'_{2,2} & \cdots & w'_{2,n} & w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w'_{n,1} & w'_{n,2} & \cdots & w'_{n,n} & w_{n,1} & w_{n,2} & \cdots & w_{n,k} \end{bmatrix}$$

$$\mathbf{x}^{<1>}, \mathbf{x}^{<2>}, \ldots, \mathbf{x}^{<T>},$$

Recurrent Neural Network

Unrolled through time:

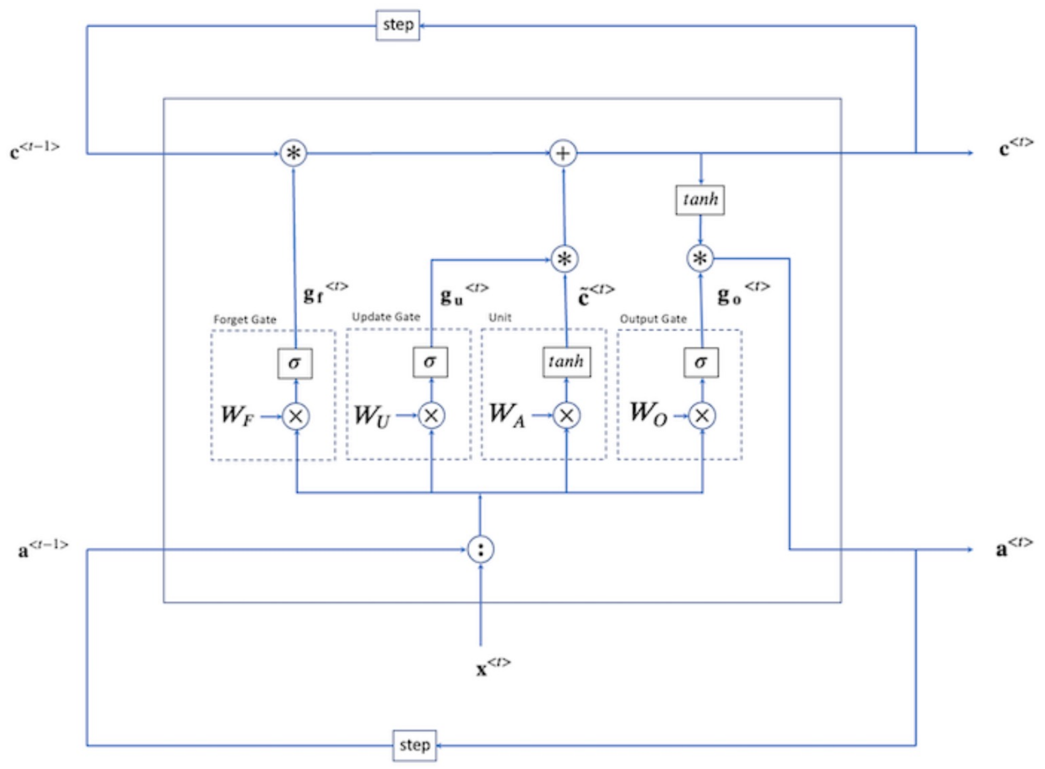


$\mathbf{x}^{<1>}, \mathbf{x}^{<2>}, \ldots, \mathbf{x}^{<T>},$

```
$\mathbf{a}^{<0>} = \mathbf{0}$
for t = 1,2,... ,T:
    $\mathbf{a}^{<t>} = tanh( W(\mathbf{a}^{<t-1>} : \mathbf{x}^{<t>}))$
```

Recurrent Neural Network

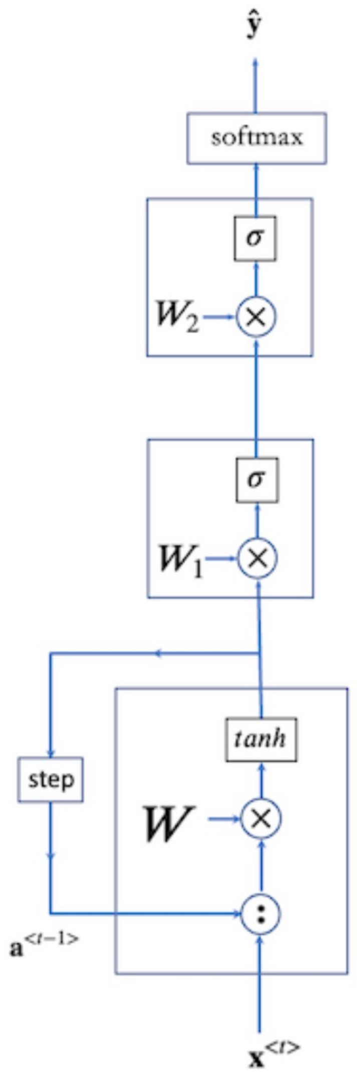
The LSTM (Long Short-Term Memory) is another, earlier design, still very much used:



```
$c^{<0>} = 0$
$a^{<0>} = 0$
for t = 1,2,... ,T:
    $g_f^{<t>} = \sigma(W_F(a^{<t-1>} : x^{<t>}))$
    $g_o^{<t>} = \sigma(W_O(a^{<t-1>} : x^{<t>}))$
    $g_u^{<t>} = \sigma(W_U(a^{<t-1>} : x^{<t>}))$
    $\tilde{c}^{<t>} = tanh(W_A(a^{<t-1>} : x^{<t>}))$
    $c^{<t>} = g_u^{<t>} * \tilde{c}^{<t>} + g_f^{<t>} * c^{<t-1>}$
    $a^{<t>} = g_o^{<t>} * tanh(c^{<t>})$
```

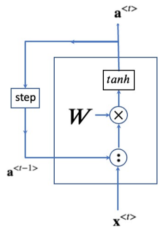
Recurrent Neural Network

A typical design uses down-stream NN networks:



RNN Review: GRUs and LSTMs for NLP

A layer with N activation units and input/feature vector of length K has the following size:

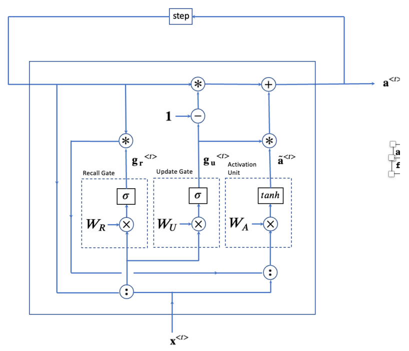


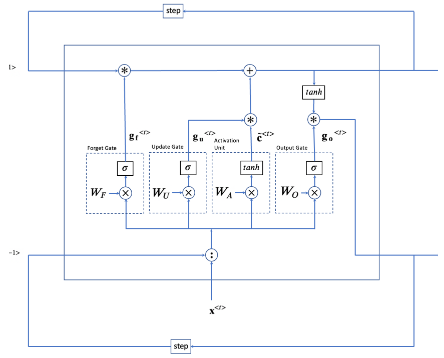
$$W = \begin{bmatrix} \mathbf{w}'_1 & \mathbf{w}_1 \\ \mathbf{w}'_2 & \mathbf{w}_2 \\ \vdots & \\ \mathbf{w}'_n & \mathbf{w}_n \end{bmatrix} = \begin{bmatrix} w'_{1,1} & w'_{1,2} & \cdots & w'_{1,n} & w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w'_{2,1} & w'_{2,2} & \cdots & w'_{2,n} & w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w'_{n,1} & w'_{n,2} & \cdots & w'_{n,n} & w_{n,1} & w_{n,2} & \cdots & w_{n,k} \end{bmatrix}$$

FFNN: N*K

GRU: 3 * (N^2 + N*K)

LSTM: 4 * (N^2 + N*K)



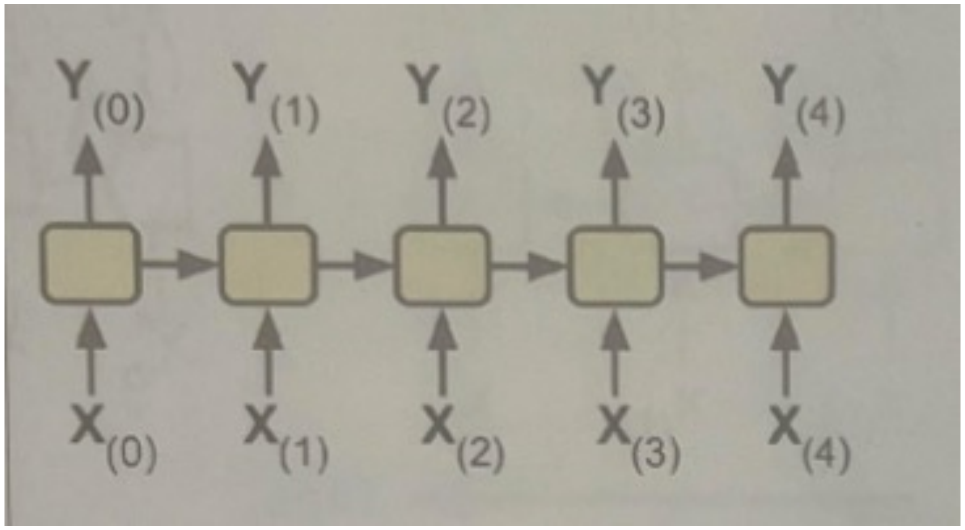


Summary:

- Most designs use embeddings as squence input to the first RNN layer
- Adding gates improves performance, but takes longer to train;
- Generally GRUs are preferred for large networks and large data sets; LSTMs for smaller.
- “No Free Lunch Theorem”: It depends on the application!
- Many heuristics (e.g., picking certain initializations for weight instead of random) seem to help in many cases.

Recurrent Neural Network

There are many ways to configure an RNN. The simplest is a sequence-to-sequence RNN:



Recurrent Neural Network

Example: Part of speech tagging

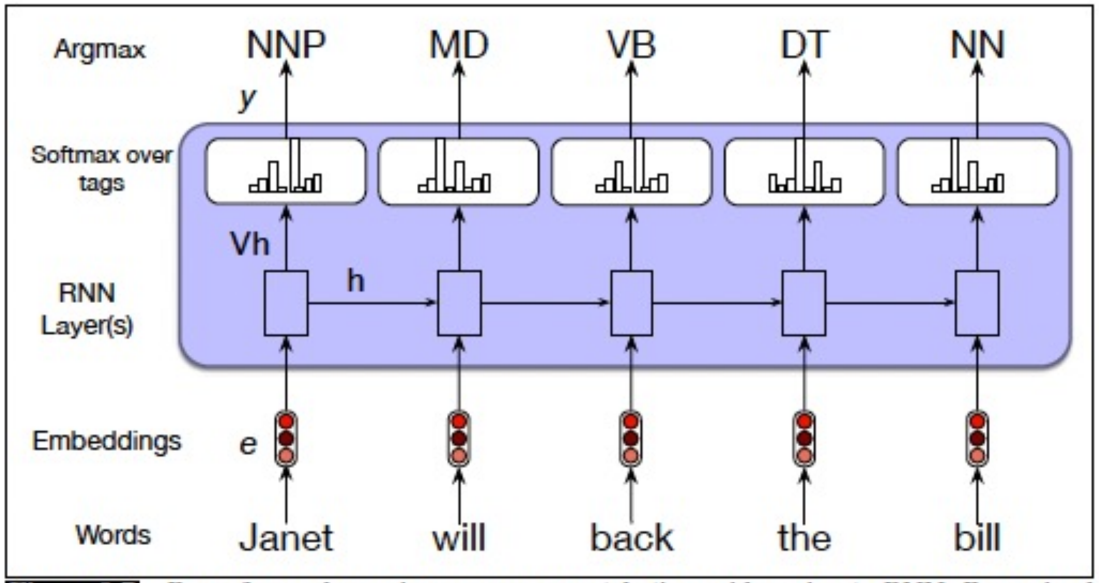
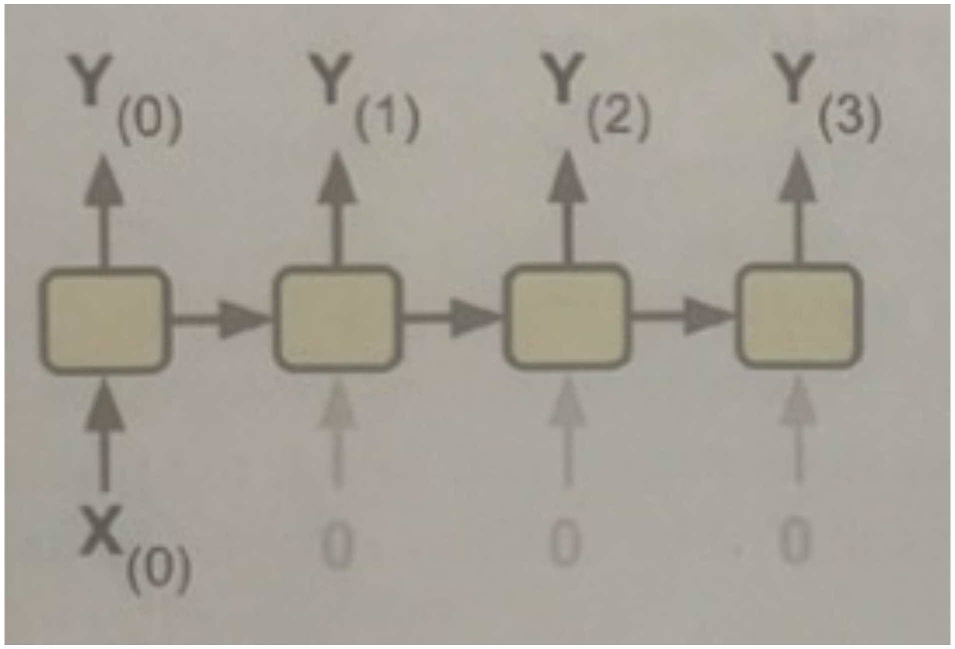


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

Recurrent Neural Network

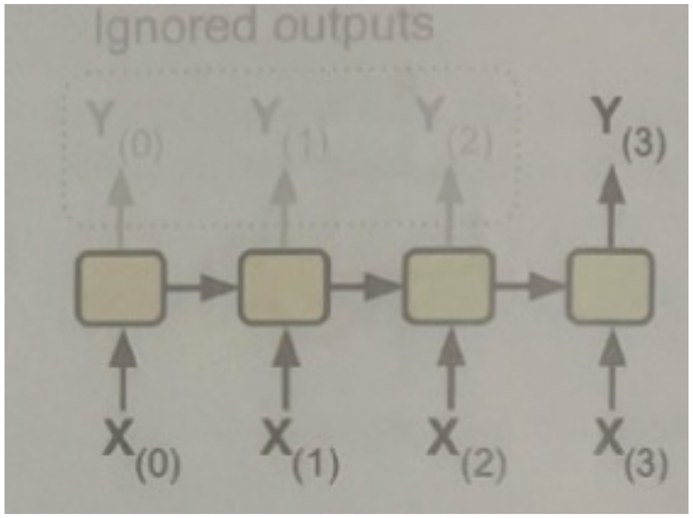
You can also take a single input and recursively process it using a vector-to-sequence RNN:



Example: Input is an image and the output is a text label for that image.

Recurrent Neural Network

In a classification task, only a single output is needed, and by ignoring all but the last output, you have a sequence-to-vector RNN:



Example: Input is a movie review and output is classification into good, bad, neutral.

Recurrent Neural Network

Recurrent layers can feed into a feedforward network....

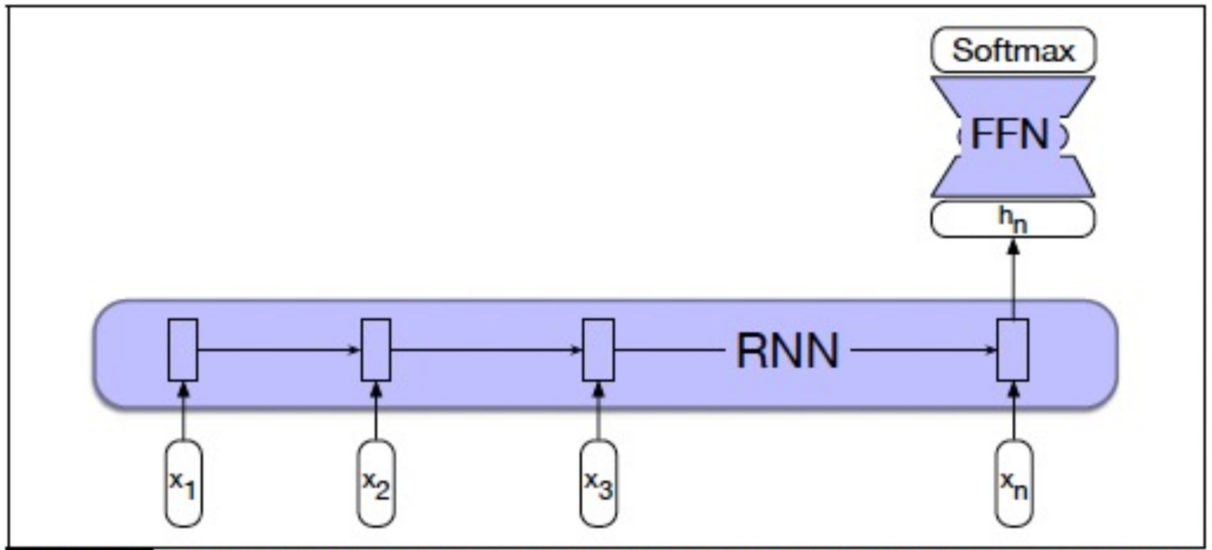


Figure 9.8 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Recurrent Neural Network

And you can stack RNNs....

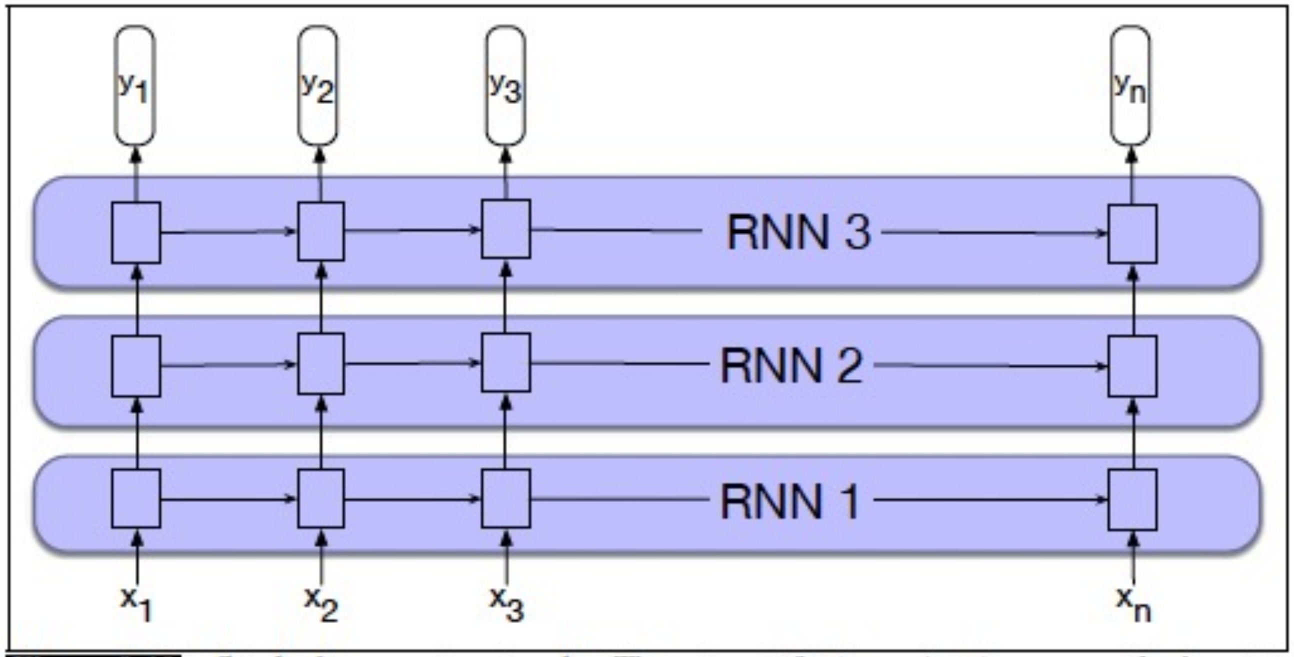
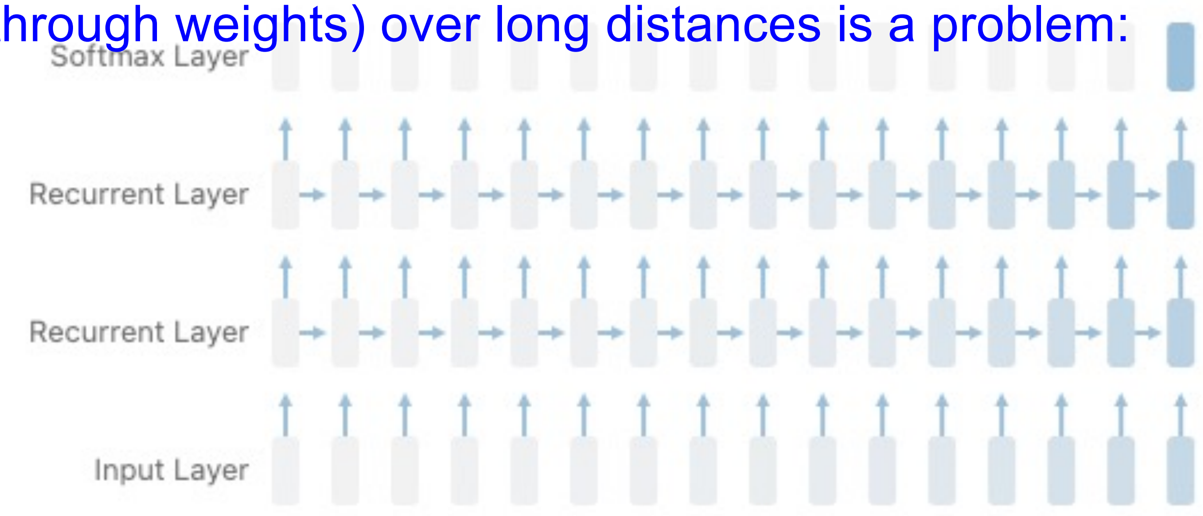


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Recurrent Neural Network

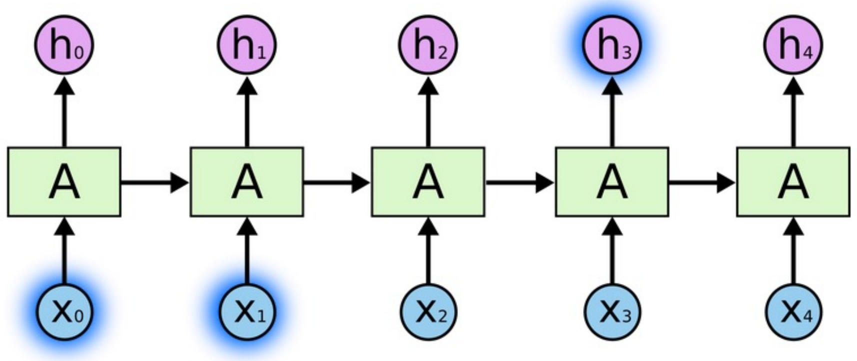
Does this solve all our problems? Unfortunately not, due to the vanishing gradients problem: unrolling through time makes the network very large and preserving information (through weights) over long distances is a problem:



Vanishing Gradient: where the contribution from the earlier steps becomes insignificant in the gradient for the vanilla RNN unit.

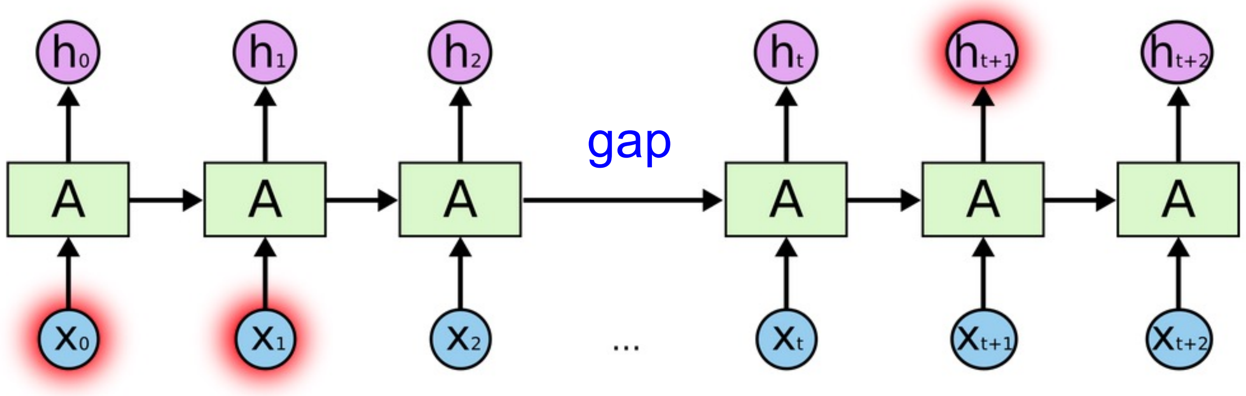
Long-term Dependencies

- **RNN might be able to connect previous information to the present task**
  - When the gap between the relevant information and the place it is needed is small
  - E.g., predict the last word in ”the clouds are in the XXX”



Long-term Dependencies

- **RNN might be able to connect previous information to the present task**
  - But there are cases where we need more context – it is possible that the gap between the relevant information and the point where it's needed to become very large

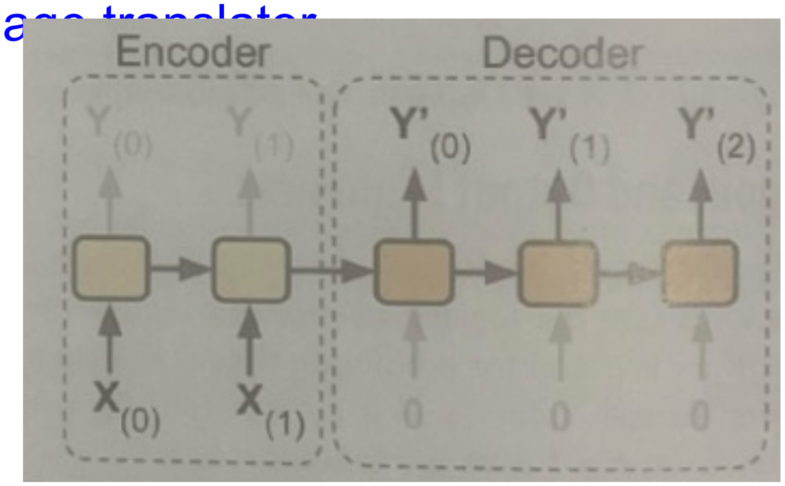


Long-term Dependencies

- **RNN might be able to connect previous information to the present task**
  - As the gap grows, RNNs become unable to learn to connect the information
  - E.g., “I grew up in France. I learn to cycle when I was very young but only learned to swim as an adult. I also love to cook and bake. I can make a mean cake. Since I grew up there, I also speak fluent XXX”

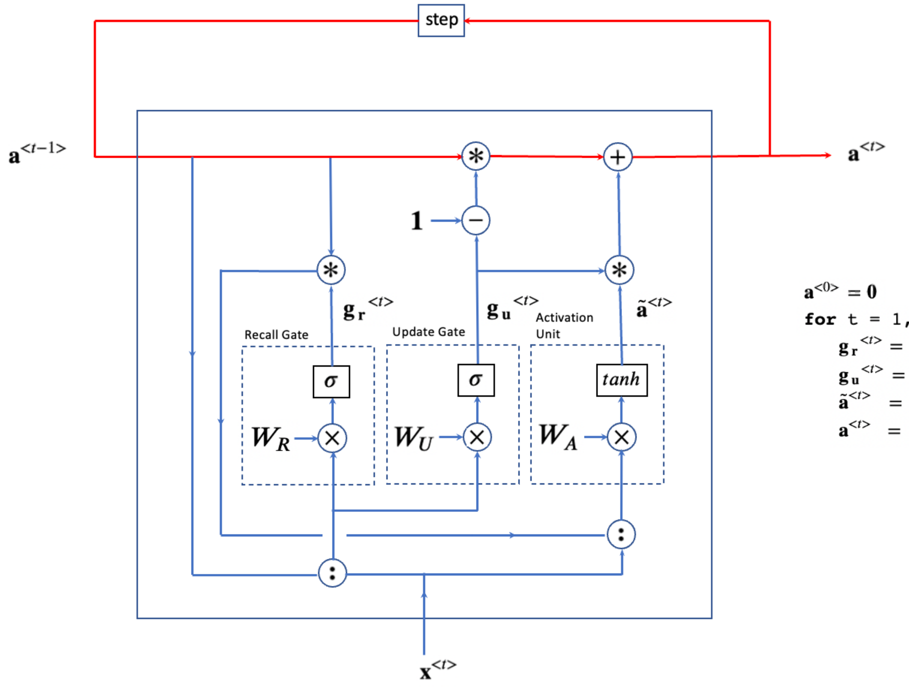
Recurrent Neural Network

A solution to the vanishing gradients problem is to build a sequence-to-sequence RNN in two stages: By combining a sequence-to-vector and a vector-to-sequence RNN, you have a sequence-to-sequence RNN, but with the advantage that the entire sequence is processed into some internal representation (a vector) and then processed into an output sequence. This is a typical organization for a language translator.



Review: GRUs and LSTMs for NLP

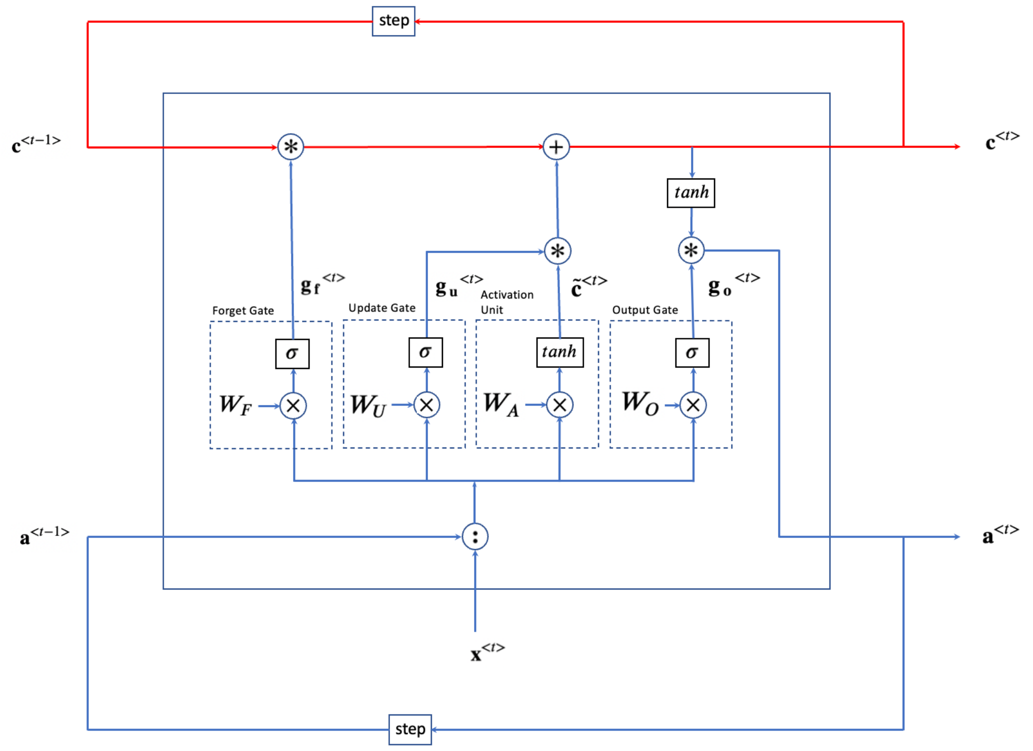
Gated Recurrence Unit Networks add a recurrent activation path which acts as a memory; the gate determines how much of the information is “remembered” at each step



```
$\mathbf{a}^{<0>} = \mathbf{0}$
for t = 1,2,... ,T:
    $\mathbf{g_r}^{<t>} = \sigma(W_R(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$
    $\mathbf{g_u}^{<t>} = \sigma(W_U(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$
    $\tilde{\mathbf{a}}^{<t>} = tanh(W_A((\mathbf{g_r}^{<t>} * \mathbf{a}^{<t-1>}):\mathbf{x}^{<t>}))$
    $\mathbf{a}^{<t>} = \mathbf{g_u}^{<t>} * \tilde{\mathbf{a}}^{<t>} + (\mathbf{1} - \mathbf{g_u}^{<t>}) * \mathbf{a}^{<t-1>}$
```

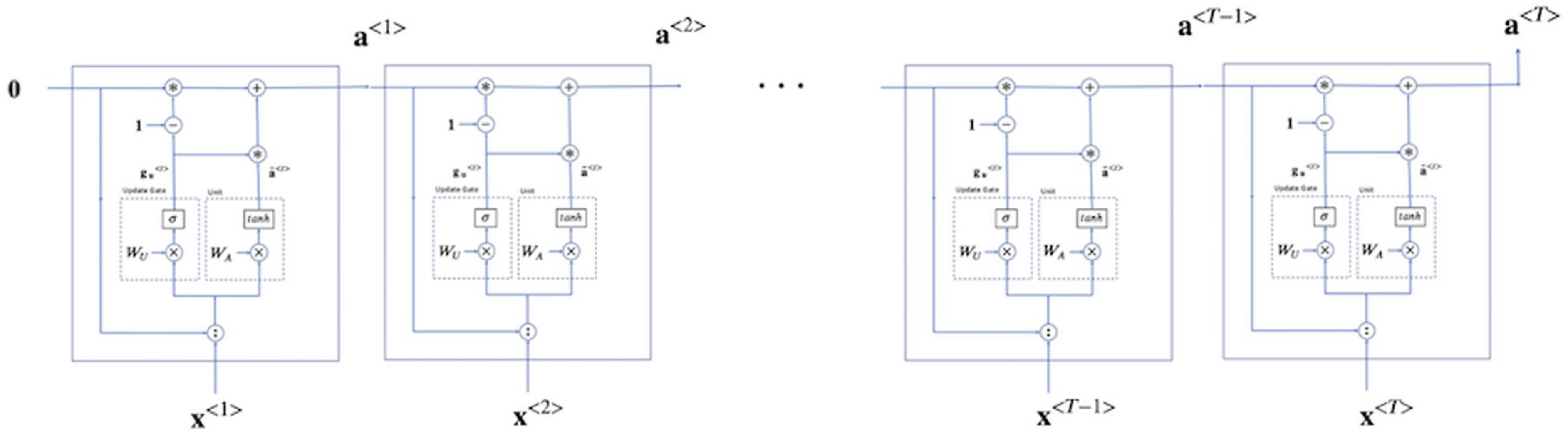
RNN Review: GRUs and LSTMs for NLP

Long Short Term Memory Layers use a separate “carry” path for the memory, 4 gates, and calculate the activation from the memory and the current inputs:



RNN Review: GRUs and LSTMs for NLP

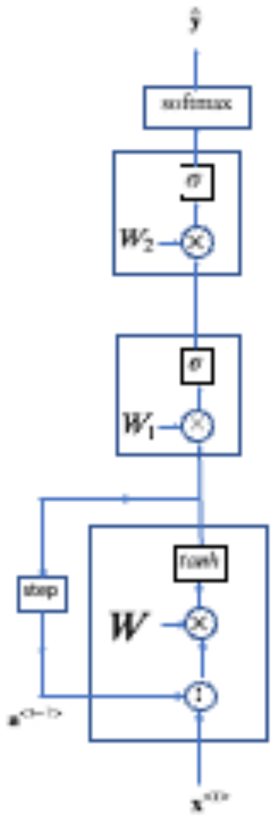
The universal method in the literature is to show these networks “unrolled through time,”
but keep in mind that these are illustrations only:

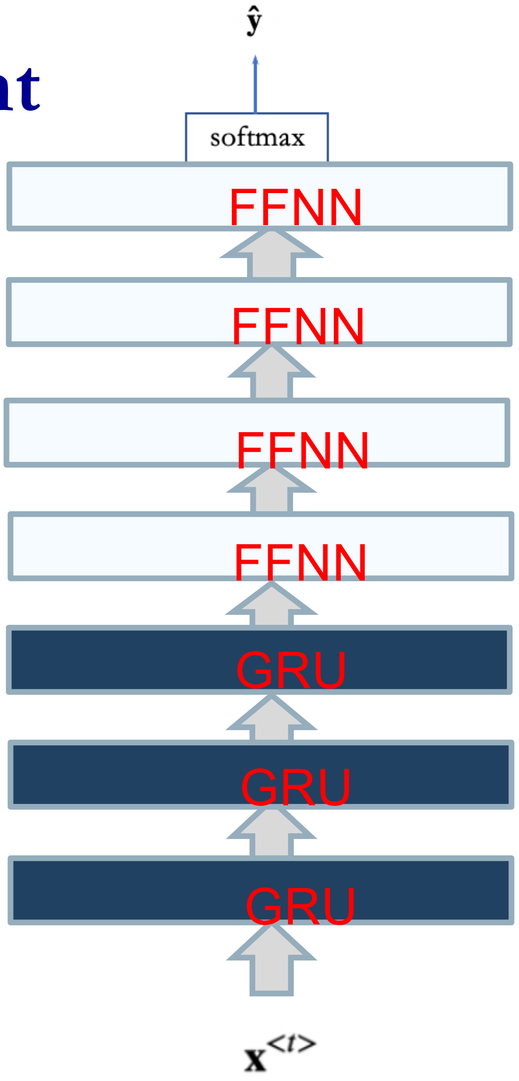


How are Networks with Recurrent Layers Designed?

Deep Networks

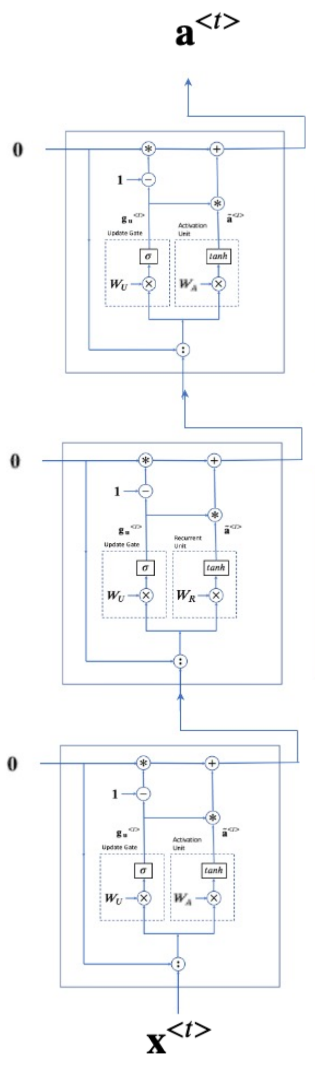
Generally, networks for sequence data such as text have recurrent layers processing the sequence, and feed forward layers interpreting and producing output such as a classification.





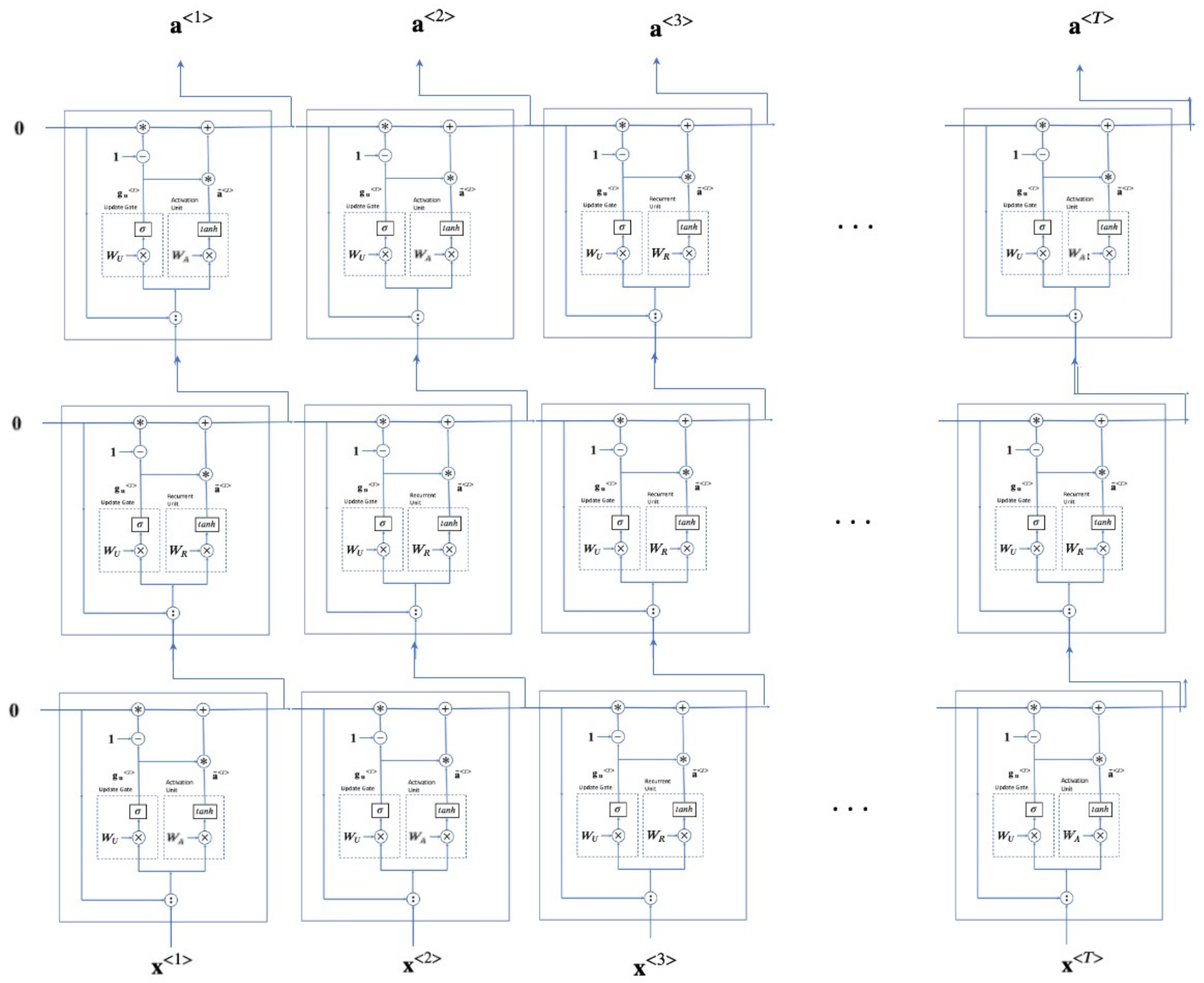
How are Netwo with Recurrent Layers Designe

Unrolling a deep RNN network reveals a very complicated design!



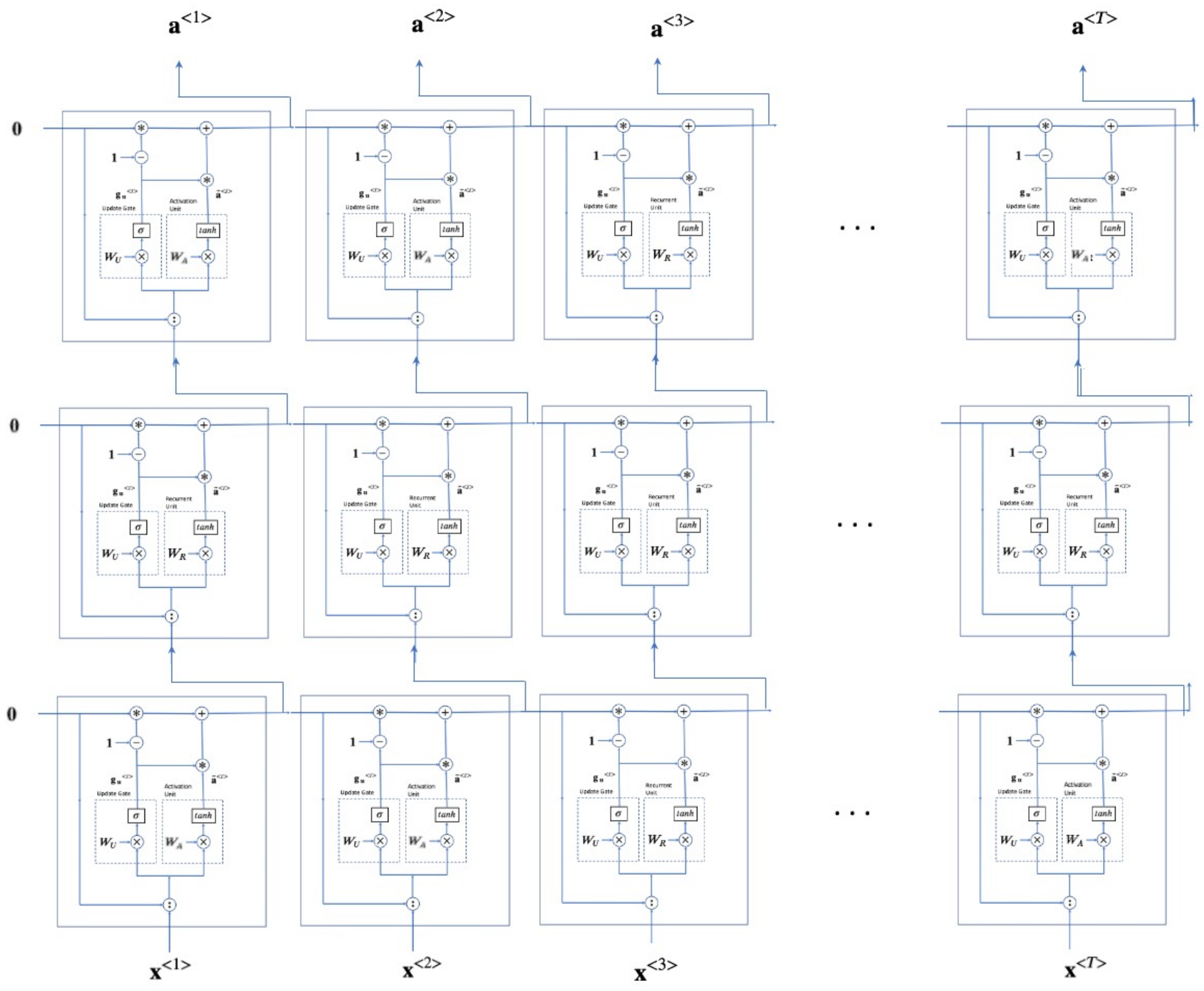
How are Netwo with Recurrent Layers Designe

Unrolling a deep RNN network reveals a very complicated design!



How are Netwo with Recurrent Layers Designe

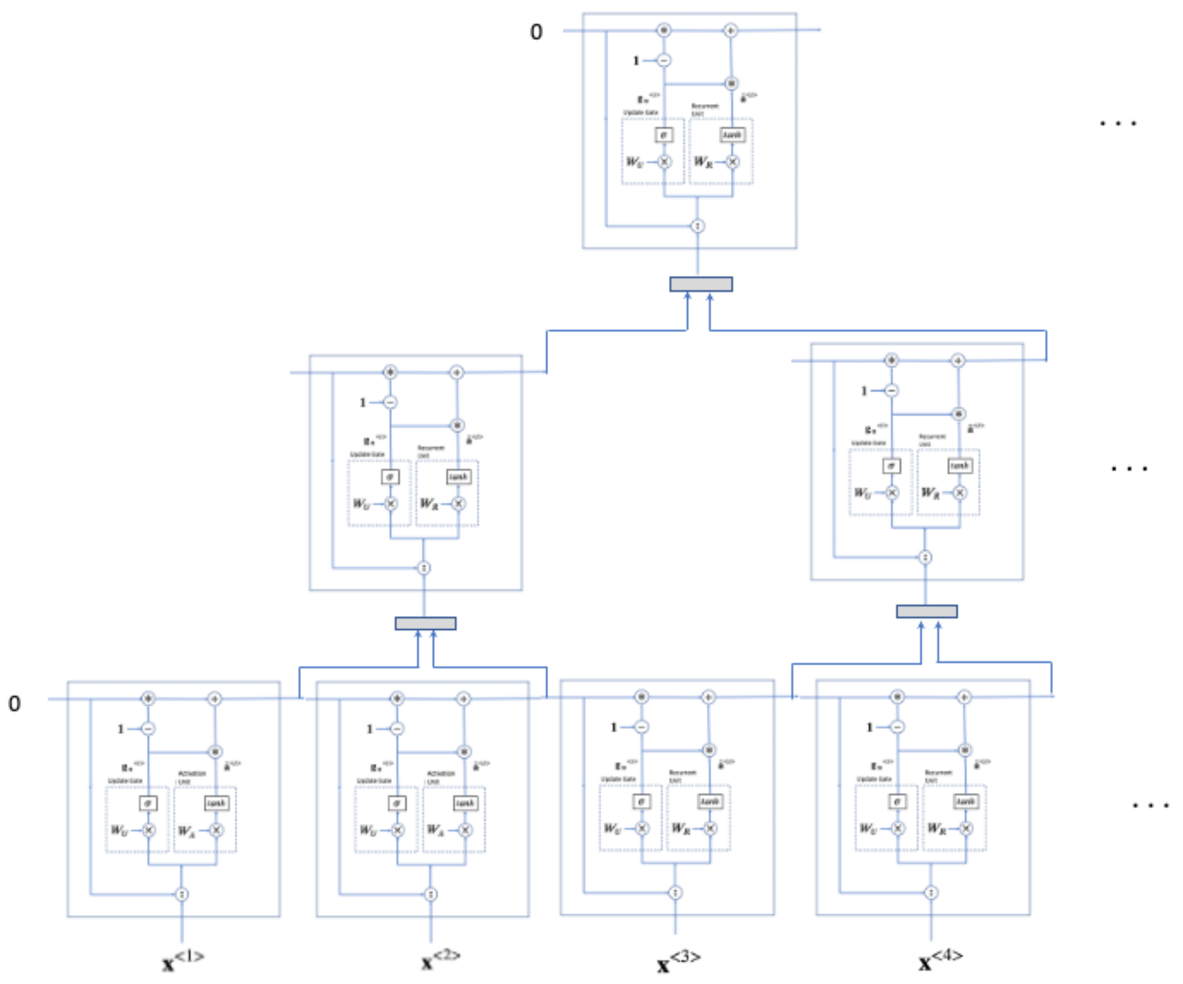
Unrolling a deep RNN network reveals a very complicated design!



How are Networ with Recurrent Layers Designed

MANY different designs have been proposed, with advantages and disadvantages.

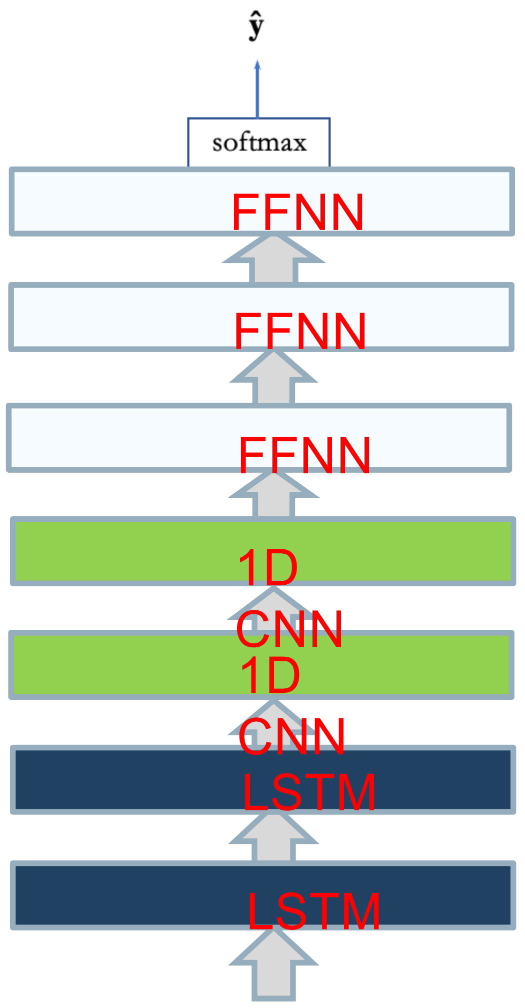
Idea 1: Tree-structured network which combines lower levels using some aggregating function (weighted) sum, perhaps controlled by a gate.



How are Networks with Recurrent Layers Designed?

MANY different designs have been proposed, with advantages and disadvantages.

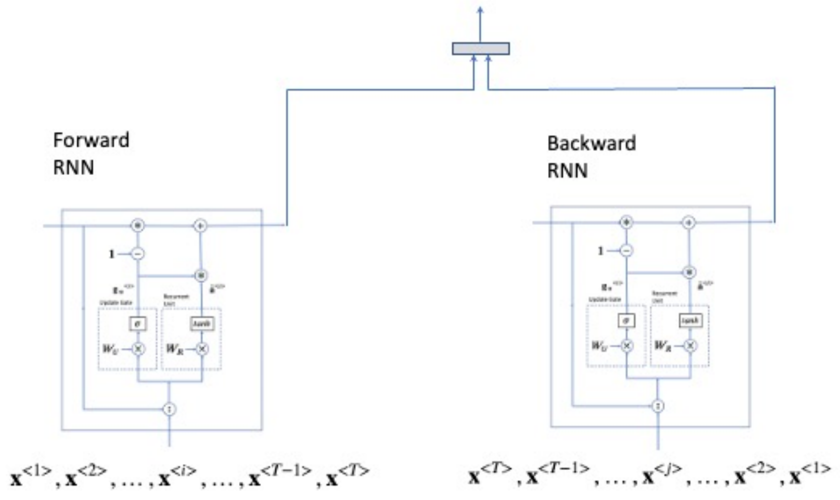
Idea 2: Apply 1D Convolutions to the RNN layers.



How are Networks with Recurrent Layers Designed?

MANY different designs have been proposed, with advantages and disadvantages.

Idea 3: Bidirectional RNN: Combine result of running two RNNs on forward and reverse sequence simultaneously. Results are fed to the next layer, usually by concatenation.


Forward RNN
Backward RNN
$x^{<1>}, x^{<2>}, \ldots, x^{<i>}, \ldots, x^{<T-1>}, x^{<T>}$
$x^{<T>}, x^{<T-1>}, \ldots, x^{<j>}, \ldots, x^{<2>}, x^{<1>}$

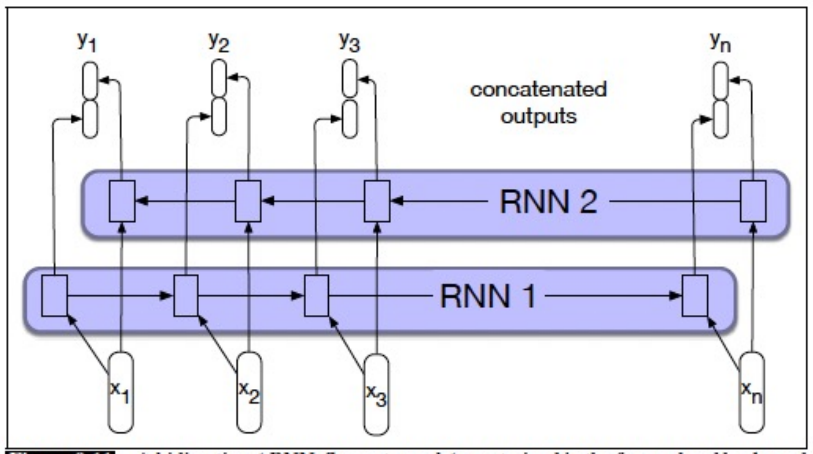


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

Training RNNs with Sequence Data: Classification

Example 1: Sentnece classification

Easy! Just use the last output and proceed as usual!

NOTE: From now on, we'll show every RNN unrolled through time, though you should always remember that there is a for loop controlling the whole process.)

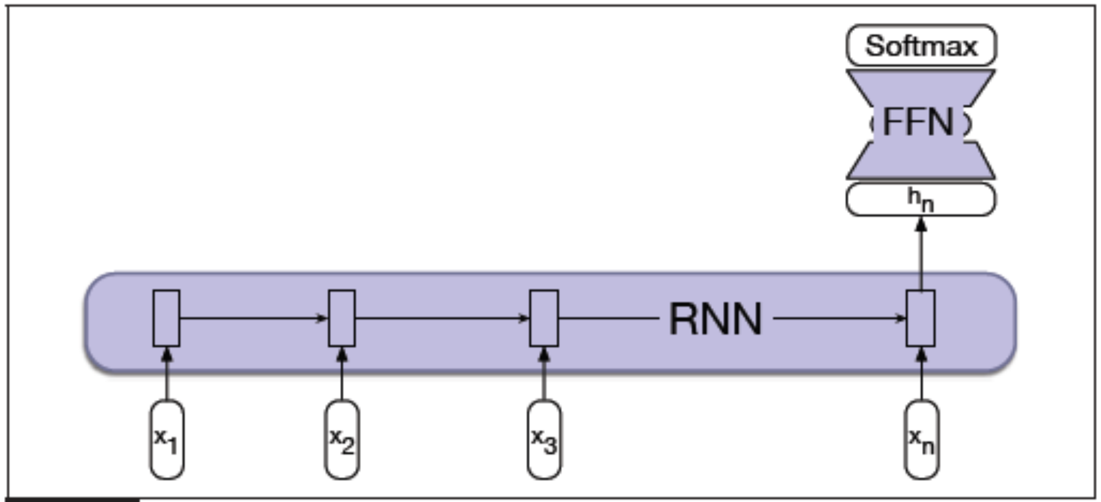


Figure 9.8 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Training RNNs with Sequence Data: POS Tagging

Example 2: Part-of-Speech Tagging

In POS Tagging, the input is a sentence, and the output is a sequence of multinomial classifications into parts of speech:

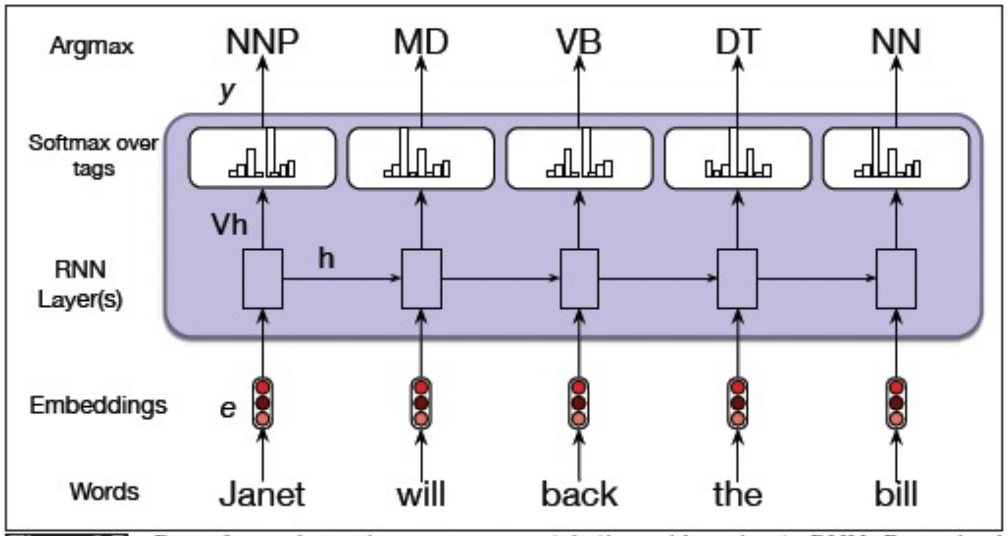


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

NOTE: From now on, we'll show every RNN unrolled through time, though you should always remember that there is a for loop controlling the whole process.)

Training RNNs with Sequence Data: POS Tagging

For each word in the sequence, the estimated tag is compared with the actual tag label, and the log loss is added across the whole sequence; to prevent longer sequences from having lower probabilities, we take the average log loss per token:

Log loss: 0.01 + 0.003 + 0.023 + 0.005 + 0.04

= 0.081 / 5 = 0.0162

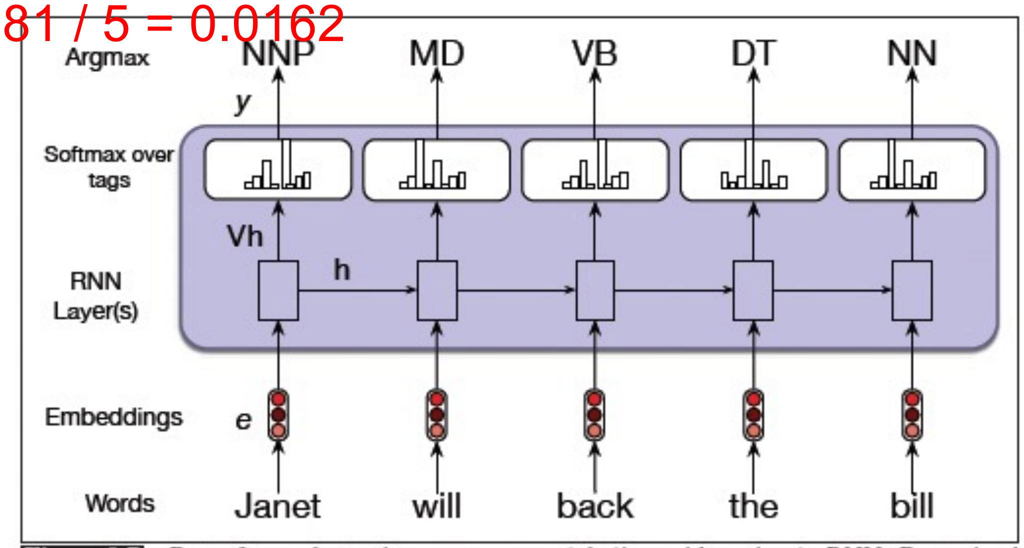


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

Training RNNs with Sequence Data: Generating Sentence using a Language Model

Recall: A Language Model assigns a probability to each sequence of words. To teach an RNN a language model, we can add the log loss of each word generated compared with an N-Gram model (there are more sophisticated approaches).

Log loss: 0.021 + 0.0034 + 0.0023 + ...

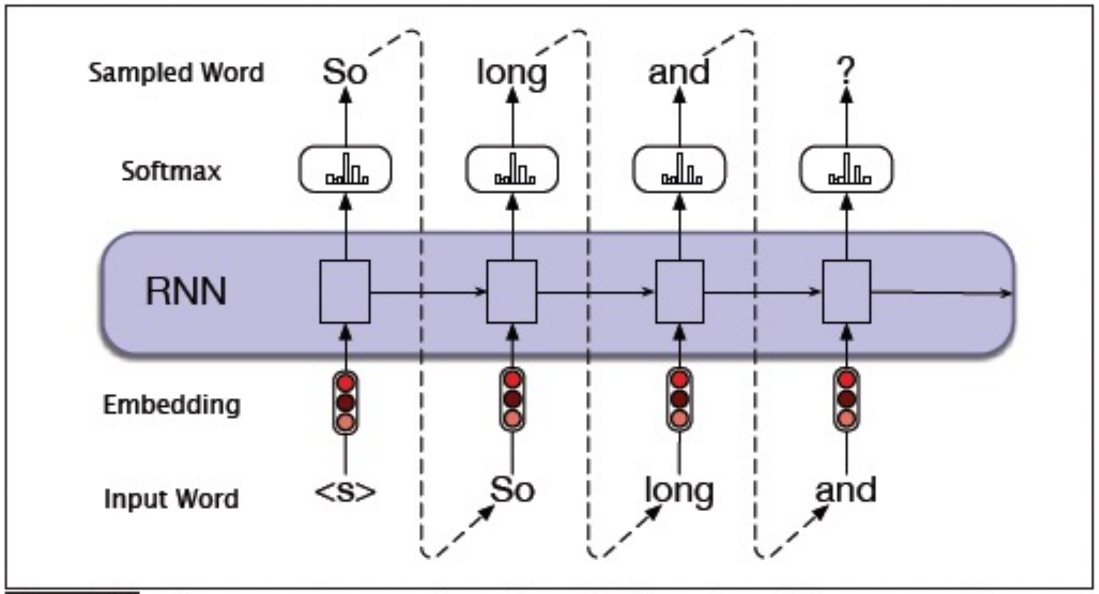


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

Training RNNs with Sequence Data: Generating Sentence using a Language Model

A sentence generator using the trained RNN can generate sentences by picking the most likely next word in each step, until it generates the end-of-sentence token </s>:

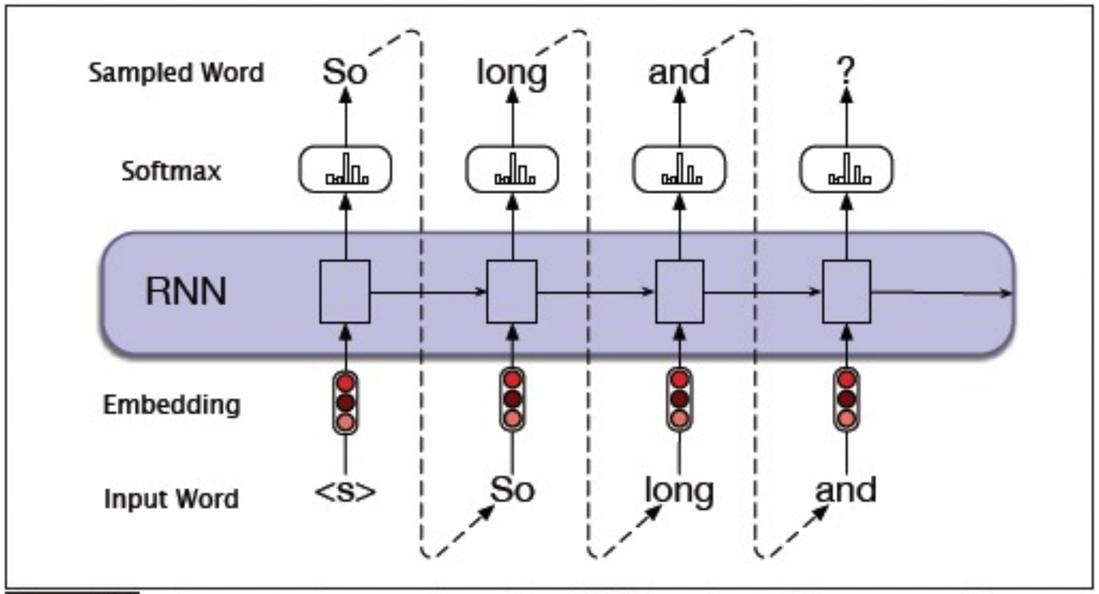


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

Training RNNs with Sequence Data: Generating Sentence using a Language Model from a Context.

An important variation of the autoregressive generator is to give the RNN a context at the beginning; here we give the generator the start-of-sentence token <s> (which says that the next word is one that must follow <s>, duh...).

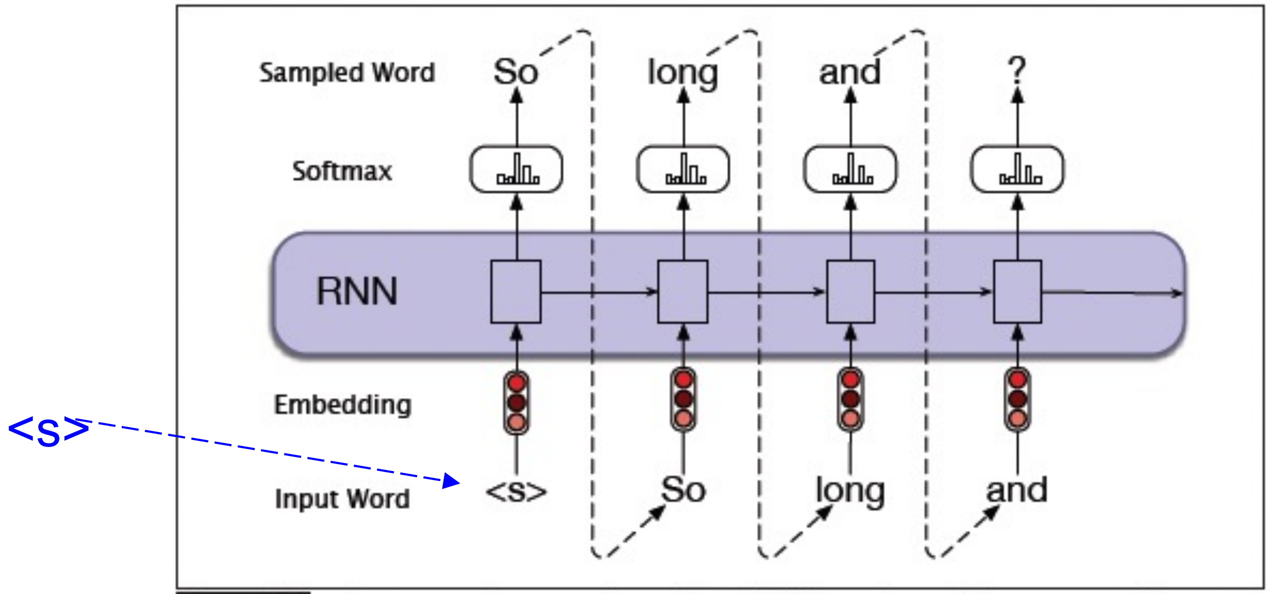
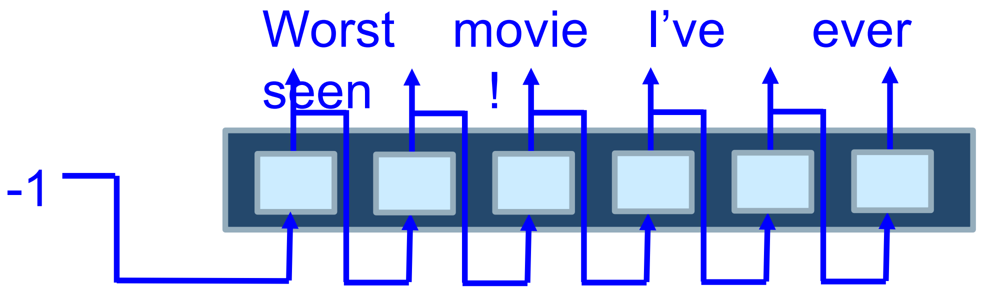


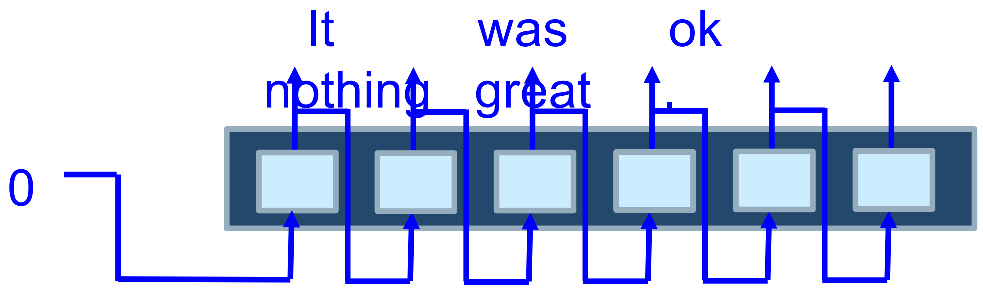
Figure 9.9 Autoregressive generation with an RNN-based neural language model.

Training RNNs with Sequence Data: Generating Sentence using a Language Model from a Context.

But this context could be anything! For example, we could give it an integer:

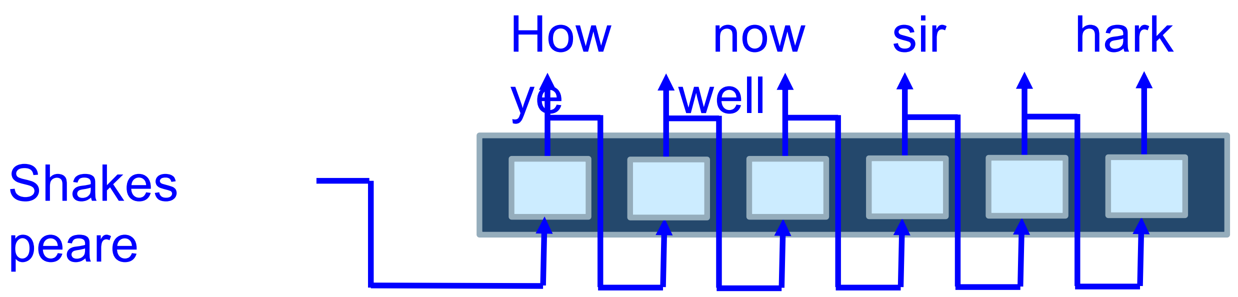
-1	Negative movie review
0	Neutral movie review
1	Positive movie review

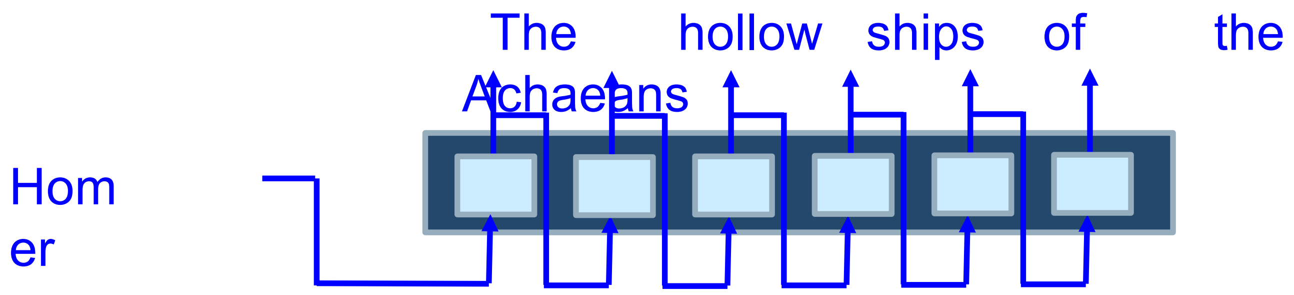




Training RNNs with Sequence Data: Generating Sentence using a Language Model from a Context.

Or it could be an author to imitate:





Training RNNs with Sequence Data: Generating Sentence using a Language Model

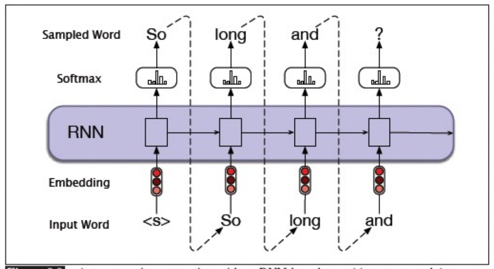


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

One Problem: The RNN makes local decisions about the most likely next word. However, a series of such local decisions will not necessarily find the globally most likely sentence (cf. gradient descent, which has the same problem).

The usual optimization is Beam Search:

1. Pick the "width of the beam" N (at each iteration, we will store the N most likely sequences of words);
2. Generate a list of the N most likely words to start a sentence, and concatenate them with <s>;
3. At each iteration, examine ALL possible next words in the sequence; toss all but the N most likely sequences;
4. Repeat until </s> is generated. Return the most likely sentence.

Note: sentences might be different lengths; stop when sequence ends in

Training RNNs with Sequence Data: Generating Sentence using a Language Model

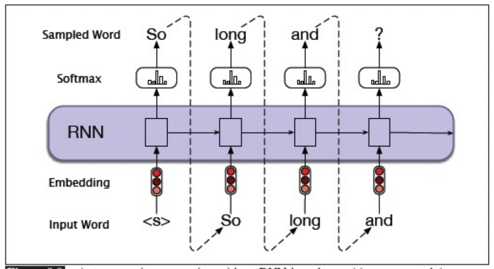
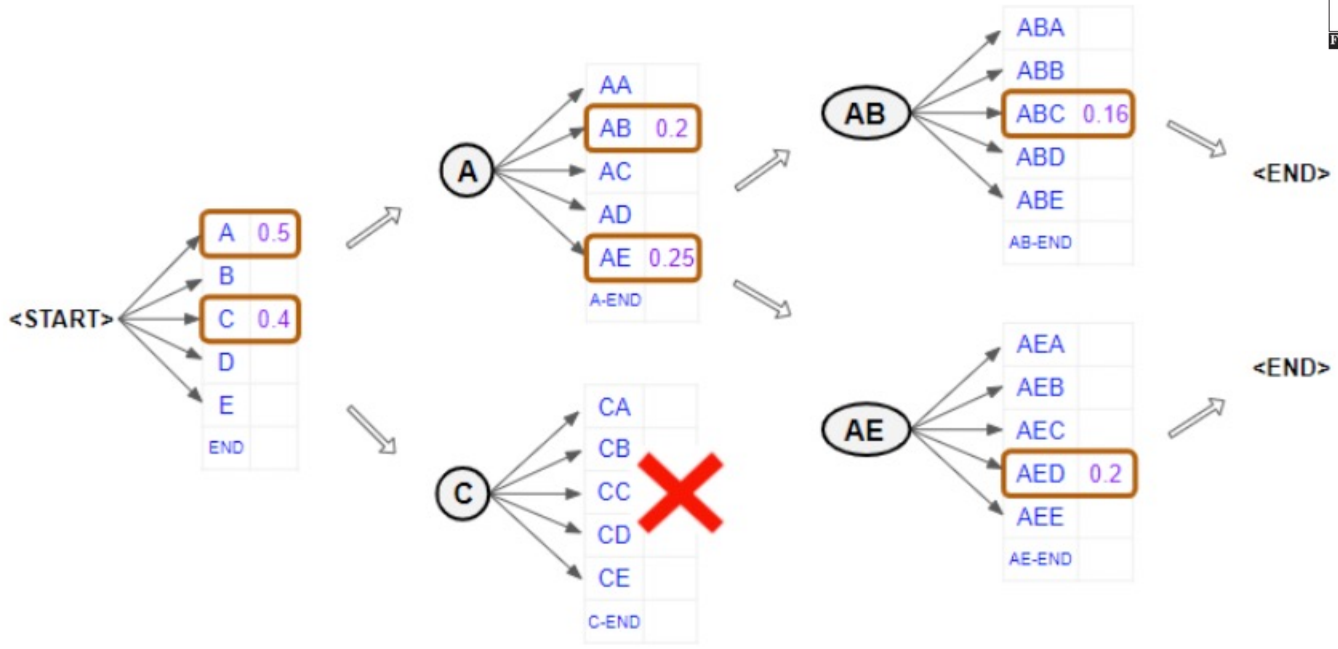


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

Example of Beam Search with N = 2 using letters instead of words:



Result: AED

Punchline: Beam search is not guaranteed to find the optimal sequence, but as a heuristic it works very well. There is an obvious efficiency/performance tradeoff. Common values of N are 10, 100, 1000.