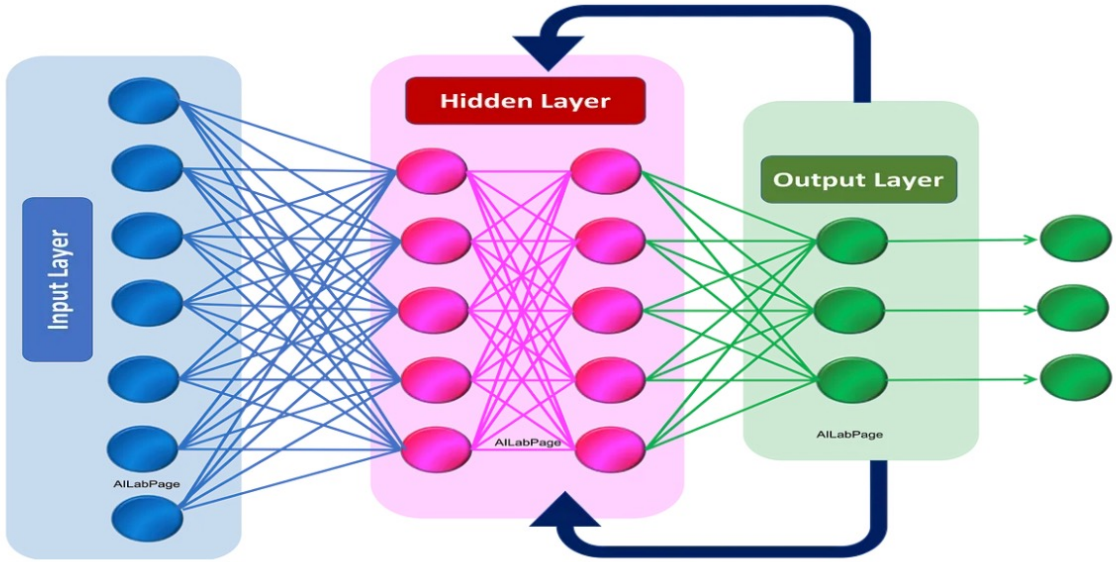
CS 505: Introduction to Natural Language Processing

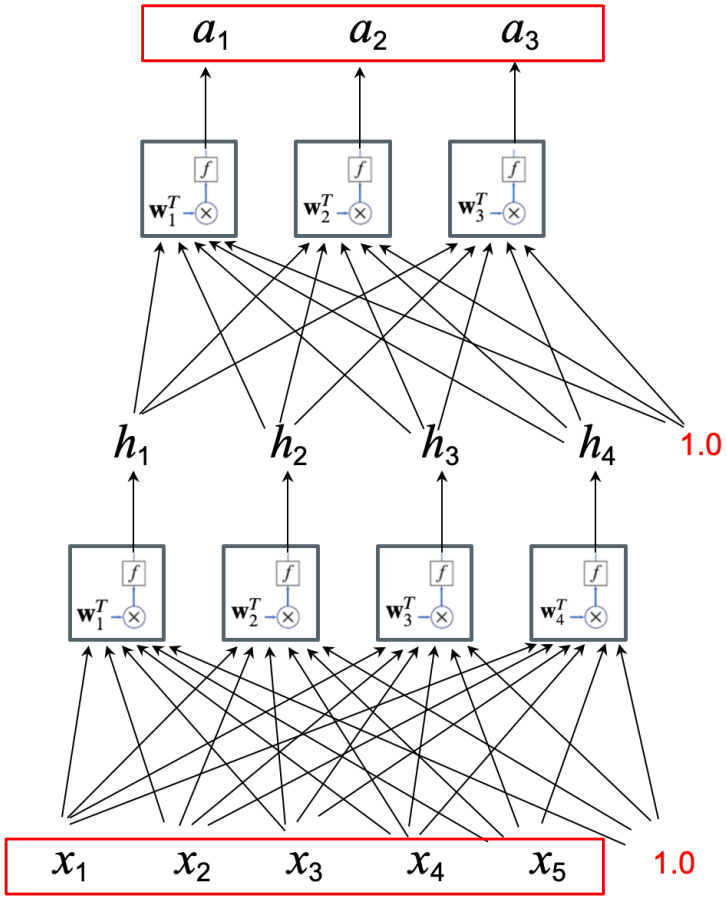
Wayne Snyder
Boston University

Lecture 12 -- Neural Networks for Sequence Data: RNNs, GRUs, LSTMs; Deep RNNs



Sequence Data: A problem for FFNNs

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



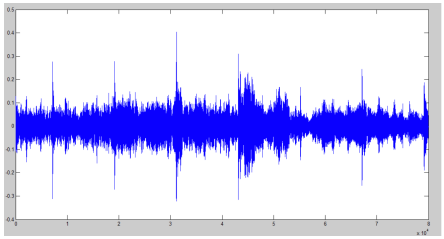
Note: The order of the inputs does not matter at all!

Sequence Data: A problem for FFNNs

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

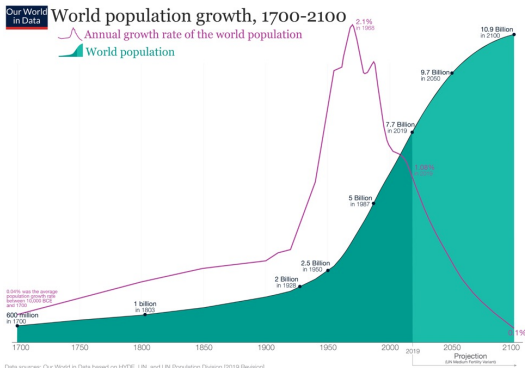
Words in a sentence: The matters order
Order the matters
Matters order the
The order matters

Audio:



Stock prices:

Population:



We have already tried one way to deal with sequences, using the Markov Assumption: N-grams = short subsequences of words..... Can we do better?

Sequence Data: A problem for FFNNs in NLP

So far, our representations for words and documents have either

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

But (modern) natural languages are strongly sequential, and exhibit long-range dependencies:

The students in CS 505 love NLP. vs. The student in CS 505 loves NLP.

The students in Professor Snyder's CS 505 class love NLP.

The students in Professor Snyder's CS 505 (Natural Language Processing) class — at least when an assignment is not due that night — love NLP.

Sequence Data: A problem for FFNNs in NLP

So far, our representations for words and documents have either

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

But (modern) natural languages are strongly sequential, and exhibit long-range dependencies.

The students in CS 505 love NLP. The student in CS 505 loves NLP.

The students in Professor Snyder's CS 505 class love NLP.

The students in Professor Snyder's CS 505 (Natural Language Processing) class — at least when an assignment is not due that night — love NLP.

Or just look at first sentence above!

All verbs are consistently in past tense. ...have ... ignored .... Accounted ....

Consistent use of commas in list ... , ... , ... , ...

Matching parentheses: ... (.....) ...

Miscellaneous: ... either .... or for very short sequences

Clearly we need better methods to process long sequences!

Sequence Data: A problem for FFNNs in NLP

And this is not just a problem with individual sentences.

With an extended discourse, many different things have to be remembered:

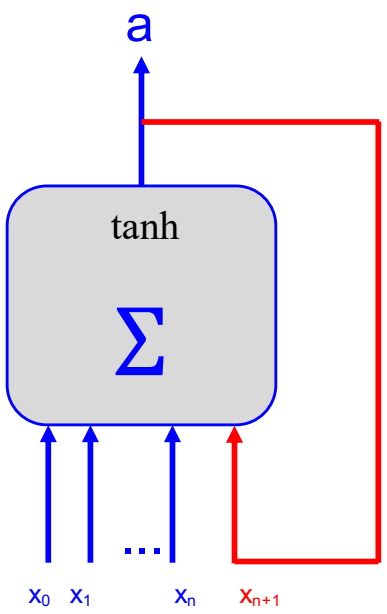
I grew up in France. I learn to cycle when I was very young but only learned to swim as an adult. I also love to cook and bake. I can make a mean cake. I speak several languages but because of where I grew up, I am most fluent in _____.

Clearly we need better methods to process long sequences!

Recurrent Neural Networks

The solution is: recursion! (Though we call it “recurrence” in NNs.)

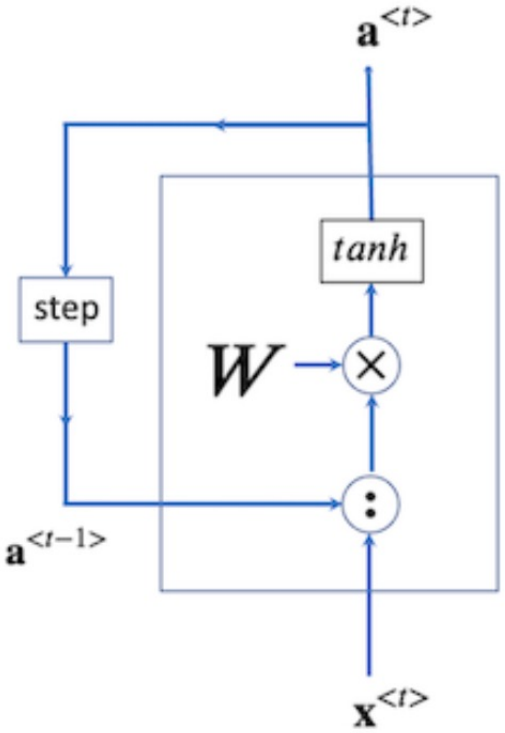
A basic recurrent neuron is the same with one important change: the outputs are fed back into the input layer:



Another, less significant change, is that tanh is generally used instead of sigmoid or relu.

Recurrent Neural Networks

In a recurrent layer, all the output activations are fed back and concatenated with the inputs.



Now the input is a sequence of data vectors

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

processed in a loop for time steps t = 1, 2, ..., T:

$$\mathbf{a}^{<0>} = \mathbf{0}$$

```
for t = 1,2,... ,T:
```

$$\mathbf{a}^{<t>} = tanh(W(\mathbf{a}^{<t-1>} : \mathbf{x}^{<t>}))$$

concatenation

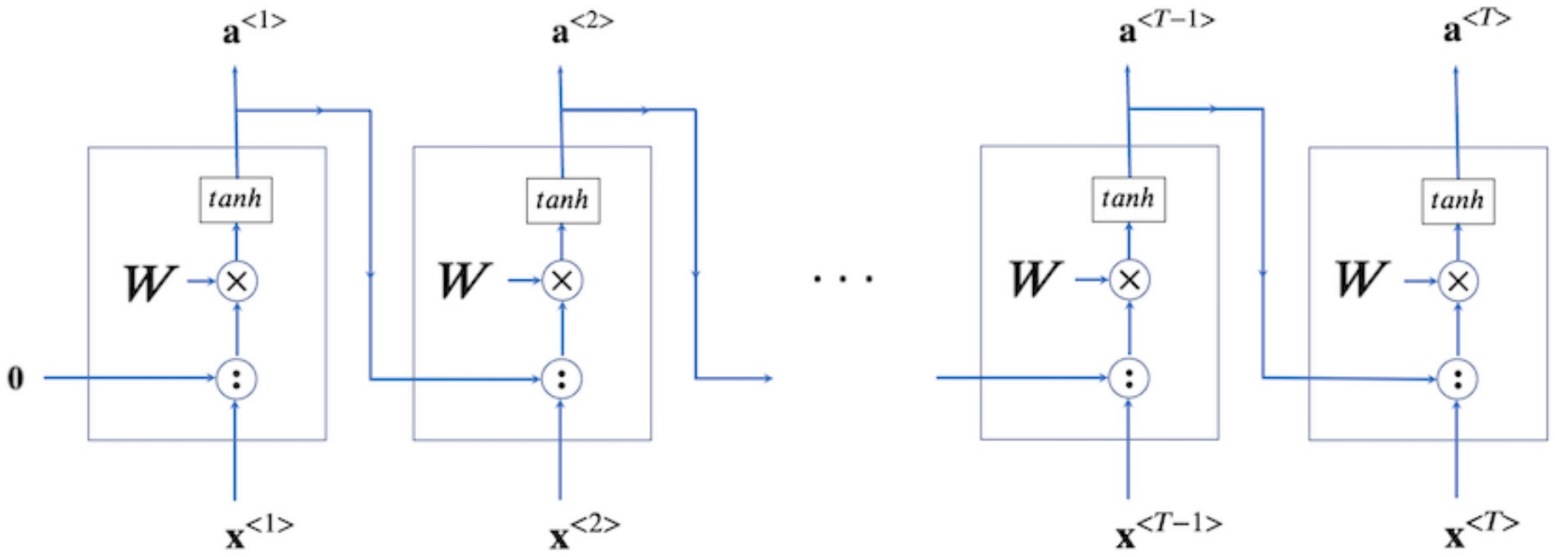
Weights for recurrent activations

Weights for data inputs

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

Recurrent Neural Networks

You will commonly see diagrams unrolled through time:



But make sure you understand that this is one layer, with one set of weights W.

$$\mathbf{a}^{<0>} = \mathbf{0}$$

```
for t = 1,2,... ,T:
```

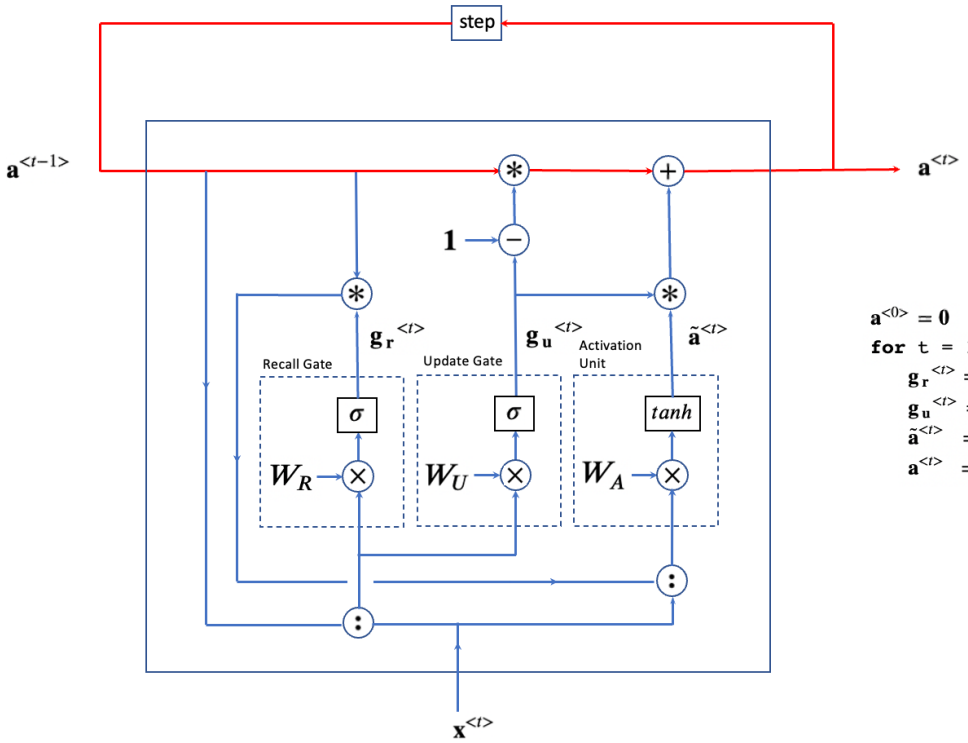
$$\mathbf{a}^{<t>} = tanh(W(\mathbf{a}^{<t-1>} : \mathbf{x}^{<t>}))$$

Recurrent Neural Networks: GRUs

A Gated Recurrence Unit (GRU) adds a recurrent activation path which acts as a memory. Two "sub-neurons" have been added:

Recall Gate: learns how to read from memory;

Update Gate: learns how to write to memory



$\mathbf{a}^{<0>} = \mathbf{0}$

for t = 1,2,... ,T:

$\mathbf{g_r}^{<t>} = \sigma(W_R(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$

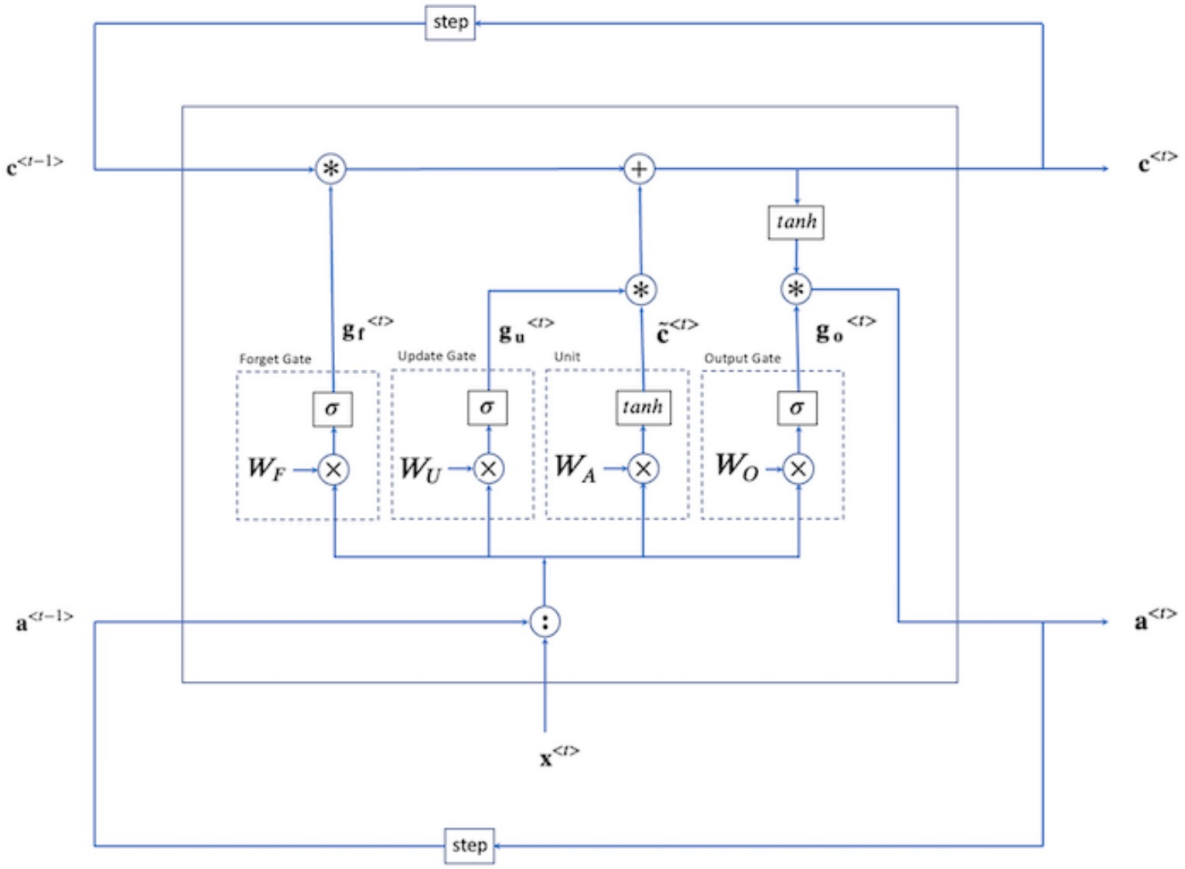
$\mathbf{g_u}^{<t>} = \sigma(W_U(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$

$\tilde{\mathbf{a}}^{<t>} = tanh(W_A((\mathbf{g_r}^{<t>} * \mathbf{a}^{<t-1>}):\mathbf{x}^{<t>}))$

$\mathbf{a}^{<t>} = \mathbf{g_u}^{<t>} * \tilde{\mathbf{a}}^{<t>} + (\mathbf{1} - \mathbf{g_u}^{<t>}) * \mathbf{a}^{<t-1>}$

Recurrent Neural Network: LSTM

A Long Short-Term Memory (LSTM) is another, earlier design, still very much used: The memory cycle is separate from the activation, and an Output gate determines how memory is used to form the activation.



$$0^{<>} = \mathbf{0}$$

$$0^{<>} = \mathbf{0}$$

for t = 1,2,... ,T:

$$\mathbf{g_f}^{<t>} = \sigma(W_F(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$$

$$\mathbf{g_o}^{<t>} = \sigma(W_O(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$$

$$\mathbf{g_u}^{<t>} = \sigma(W_U(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$$

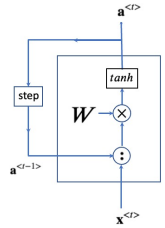
$$\tilde{\mathbf{c}}^{<t>} = tanh(W_A(\mathbf{a}^{<t-1>}:\mathbf{x}^{<t>}))$$

$$\mathbf{c}^{<t>} = \mathbf{g_u}^{<t>} * \tilde{\mathbf{c}}^{<t>} + \mathbf{g_f}^{<t>} * \mathbf{c}^{<t-1>}$$

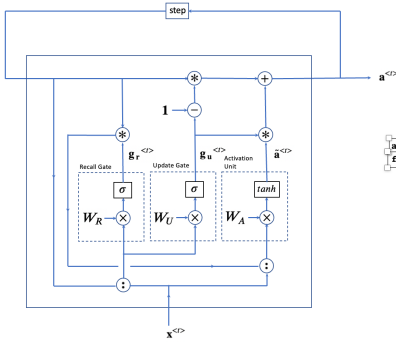
$$\mathbf{a}^{<t>} = \mathbf{g_o}^{<t>} * tanh(\mathbf{c}^{<t>})$$

RNNs: Which one is best for NLP?

Historically, the LSTM design was first, and the GRU was designed as a simplified version of the LSTM. The very basic RNN is also used.

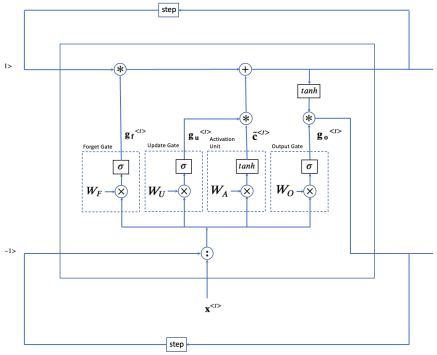


However, the LSTM has 4 weight matrices, compared with 3 for the GRU, 2 for the Simple GRU, and one for the FFNN.



Adding more gates in general improves performance, but has an obvious impact on the complexity of training.

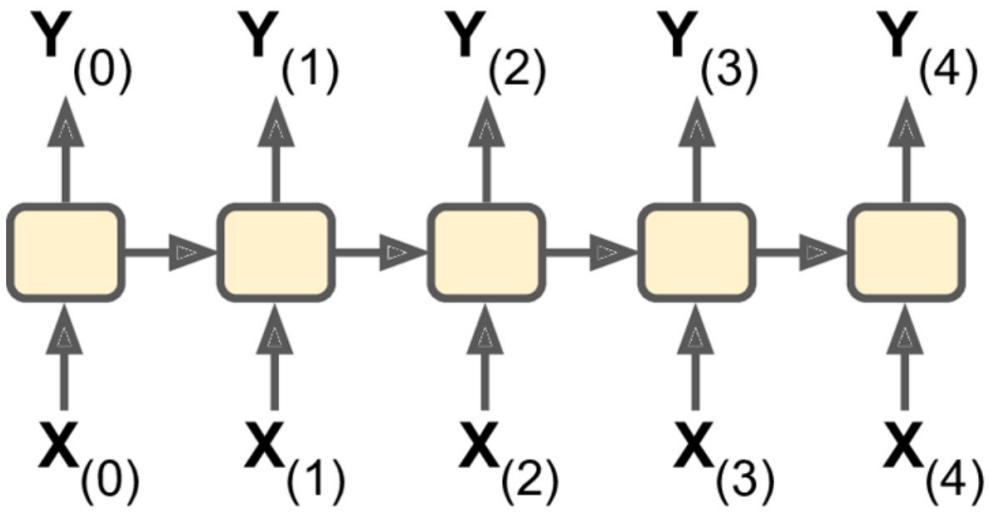
For very large networks with large data sets, the GRU is generally used, and the LSTM seems to work better for smaller projects such as you would do in an academic course.



My experience has been that the LSTM works best for the size and type of projects we'll do in CS 505.

RNN Network Architectures

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



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.)

RNN Network Architectures

Example 1: Part of speech tagging

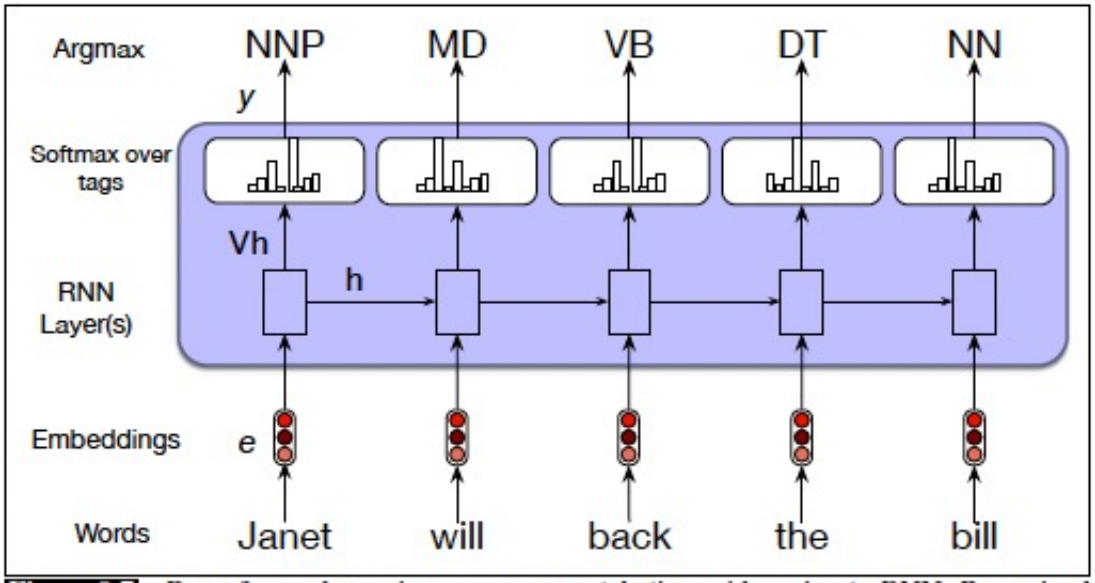


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

RNN Network Architectures

Example 2: A sentence generator using a trained RNN can generate sentences by picking the most likely next word in each step, until it generates the end-of-sentence token </s>:

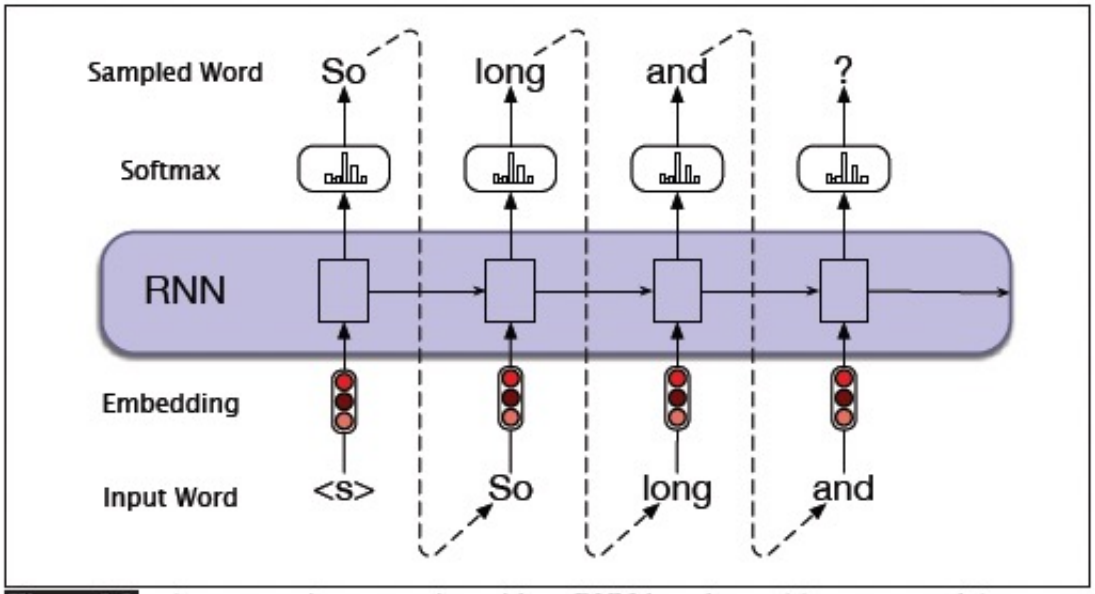
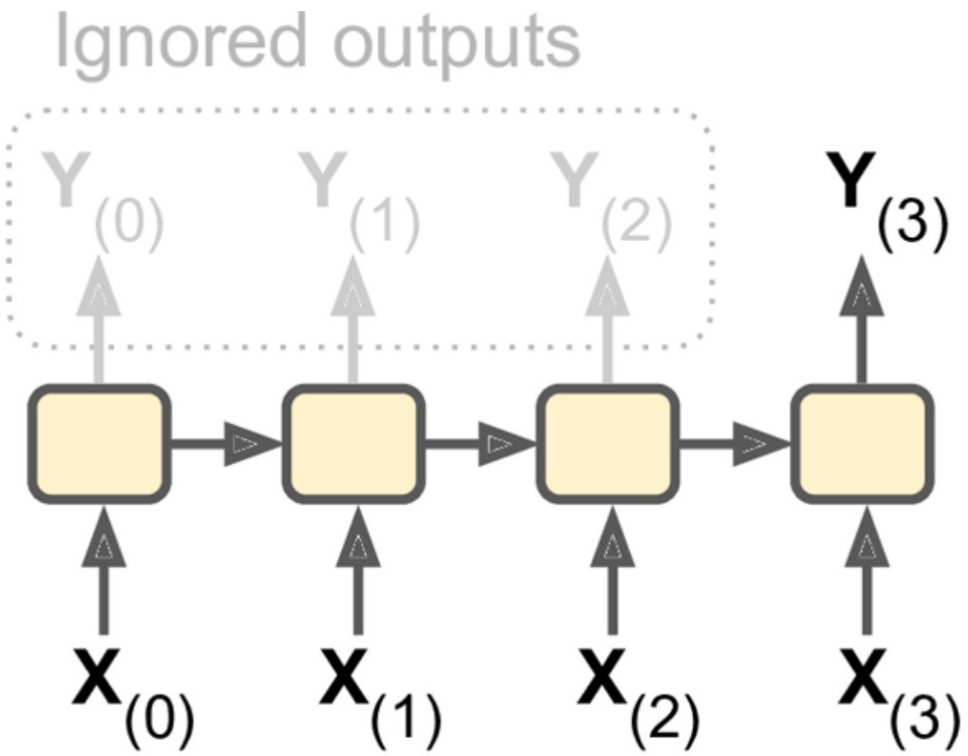


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

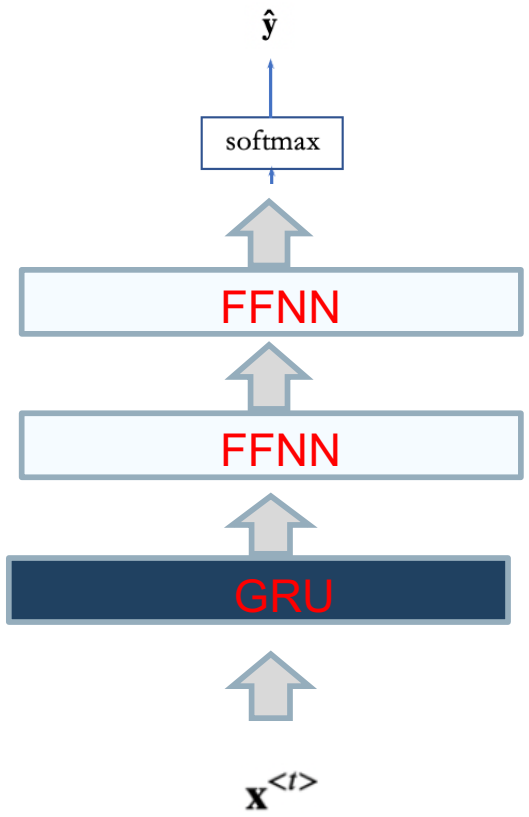
RNN Network Architectures

Another configuration is sequence-to-vector:



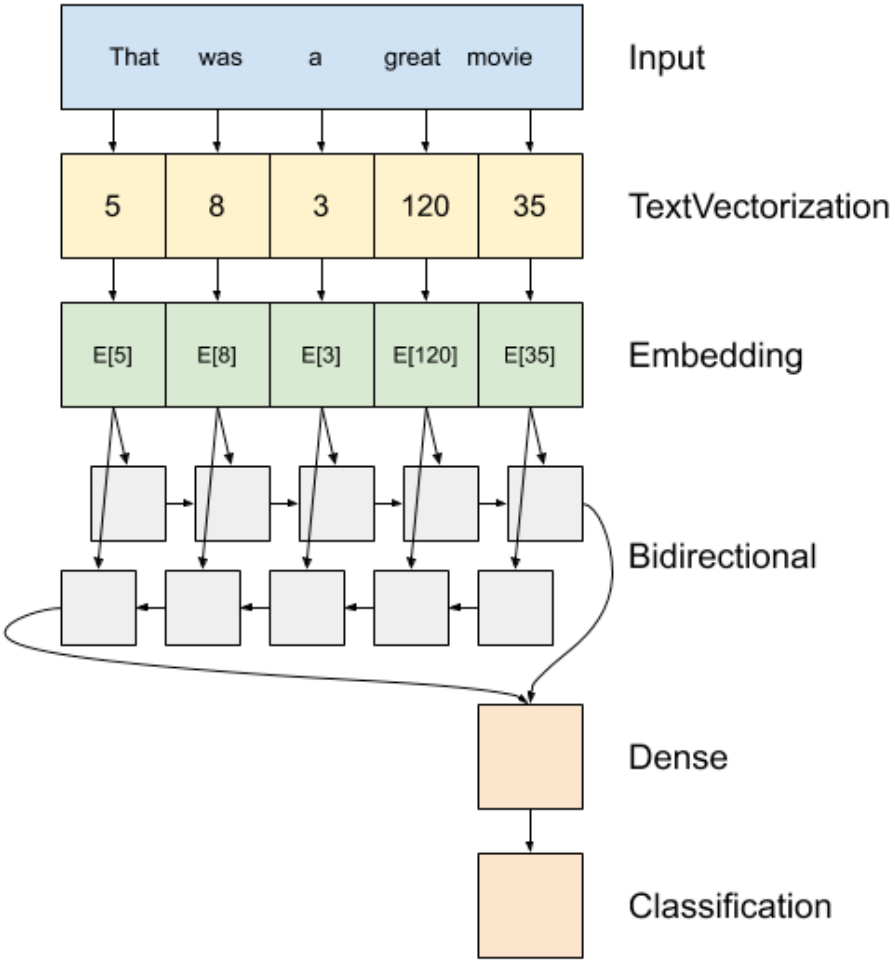
RNN Network Architectures

With sequence-to-vector layers, it is very common to add FF layers downstream, especially for classification:



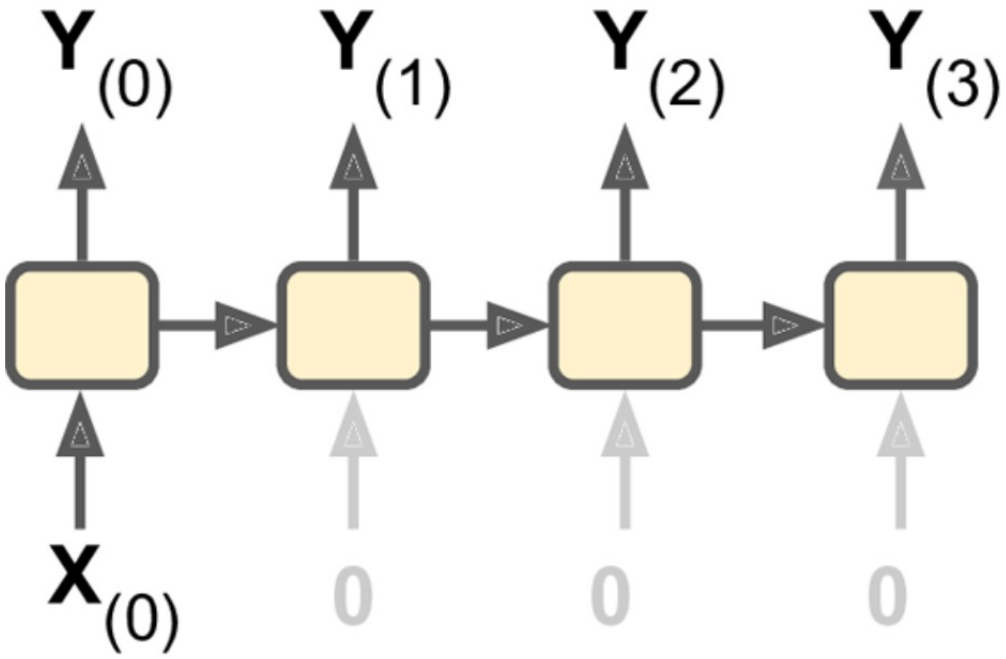
RNN Network Architectures

Example: Text Classification



RNN Network Architectures

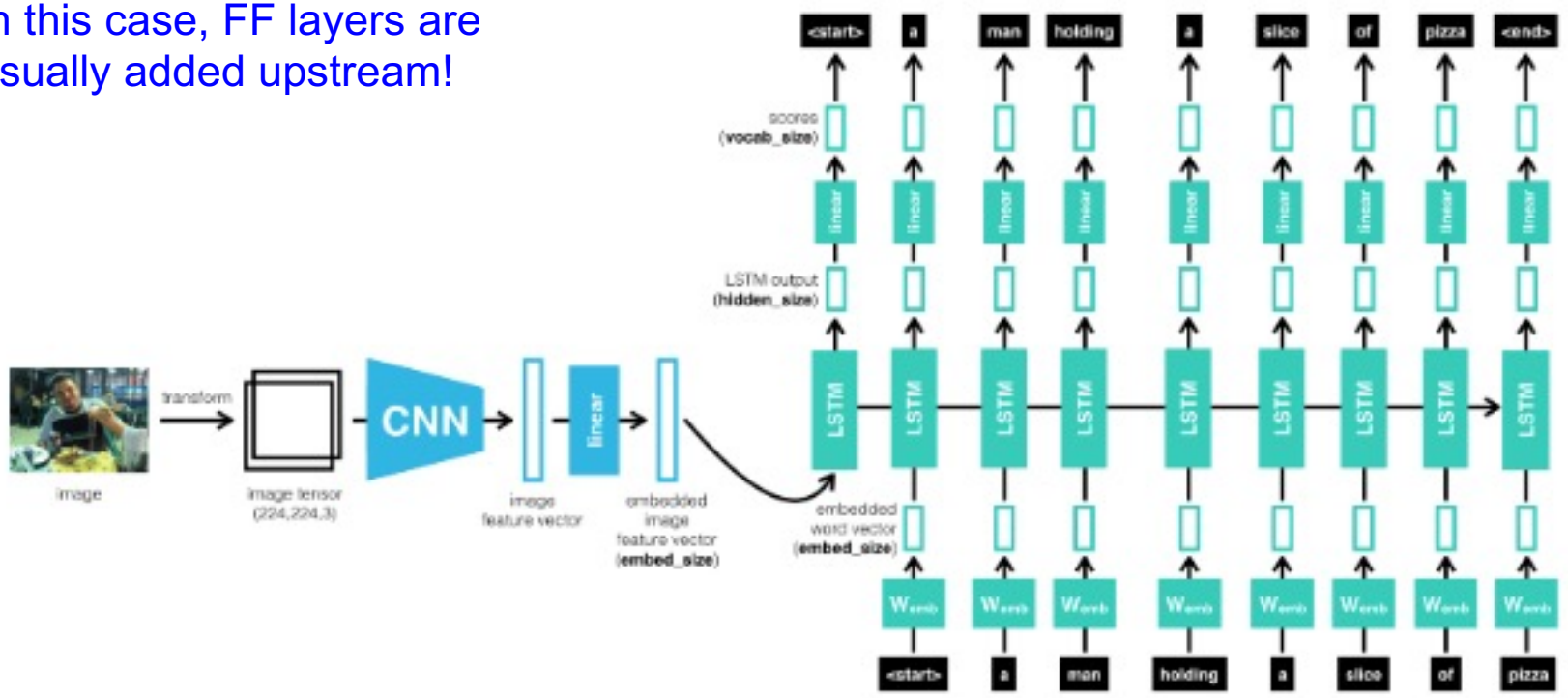
A third possibility is vector-to-sequence:



RNN Network Architectures

Example: Image Captioning

In this case, FF layers are usually added upstream!



RNN Network Architectures

Example: Image Captioning (hopefully accurate)

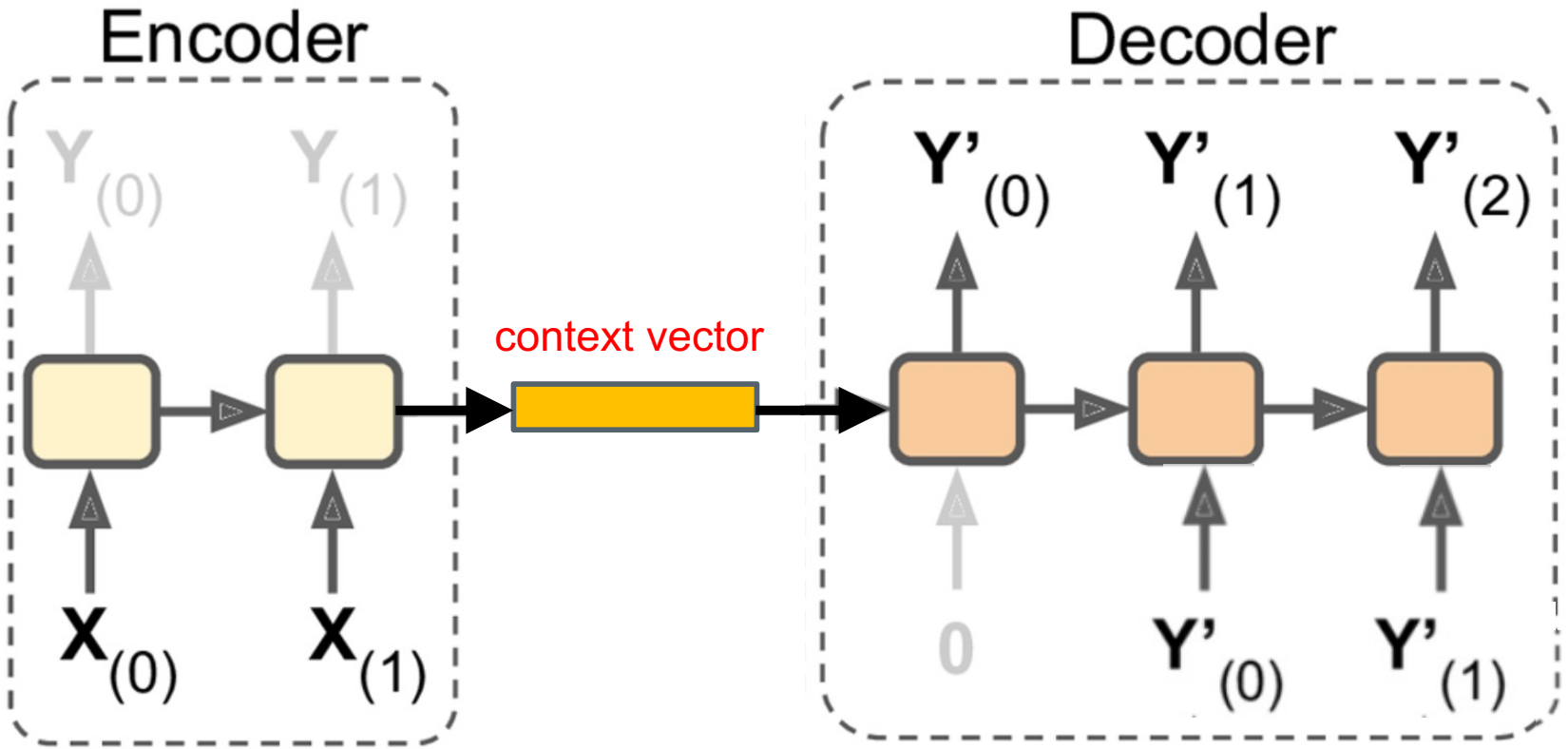
a man in a white shirt playing tennis

RNN Network Architectures

Finally, the SOTA sequence-to-sequence model uses

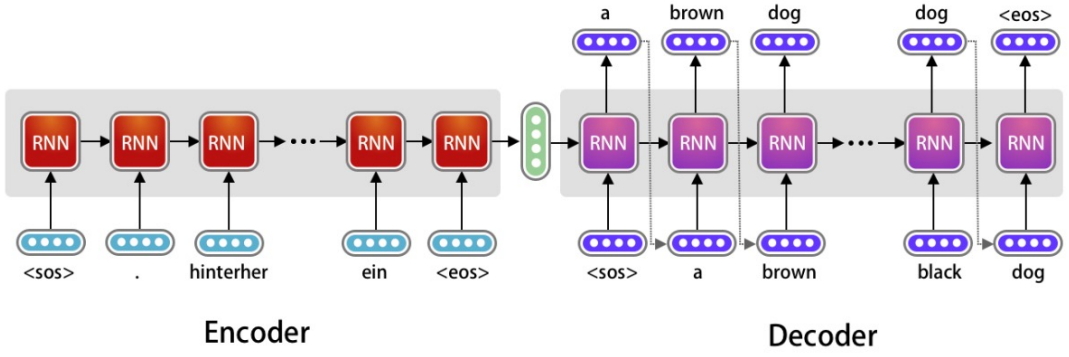
- An Encoder (sequence-to-vector) and
- A Decoder (vector-to-sequence)

and passes a context vector between them:

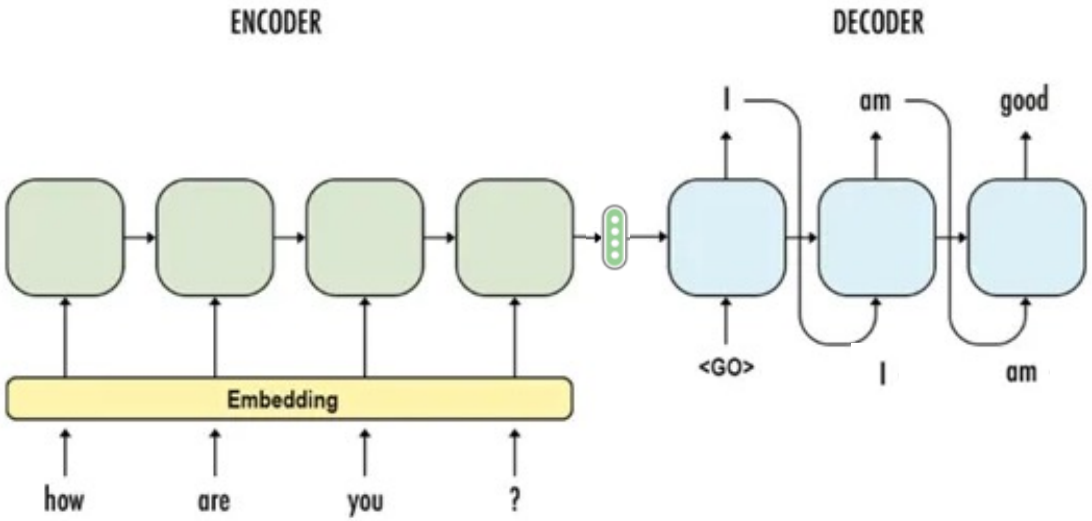


RNN Network Architectures

Examples: Machine Translation, Conversation Agents:



Always keep in mind that these are unrolled in time!



Note that the input to the decoder at each time step is the previous output plus additional activations.

RNN Network Architectures

One more idea: Stacked Recurrent Layers!

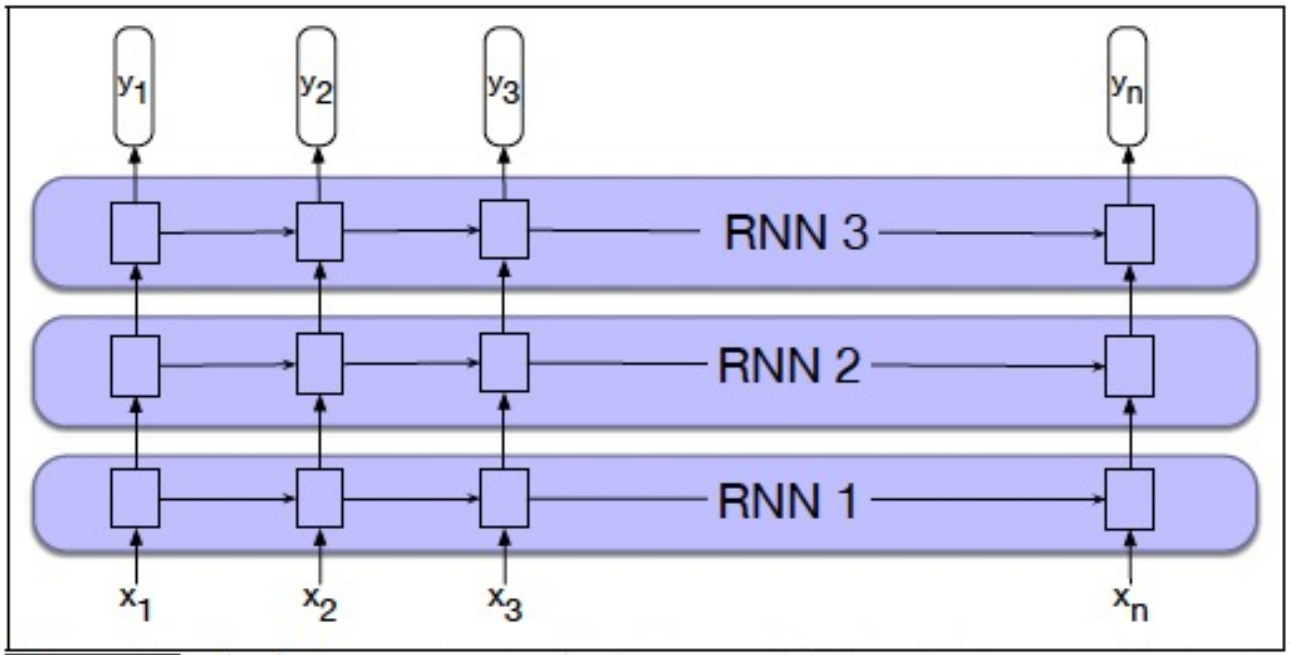
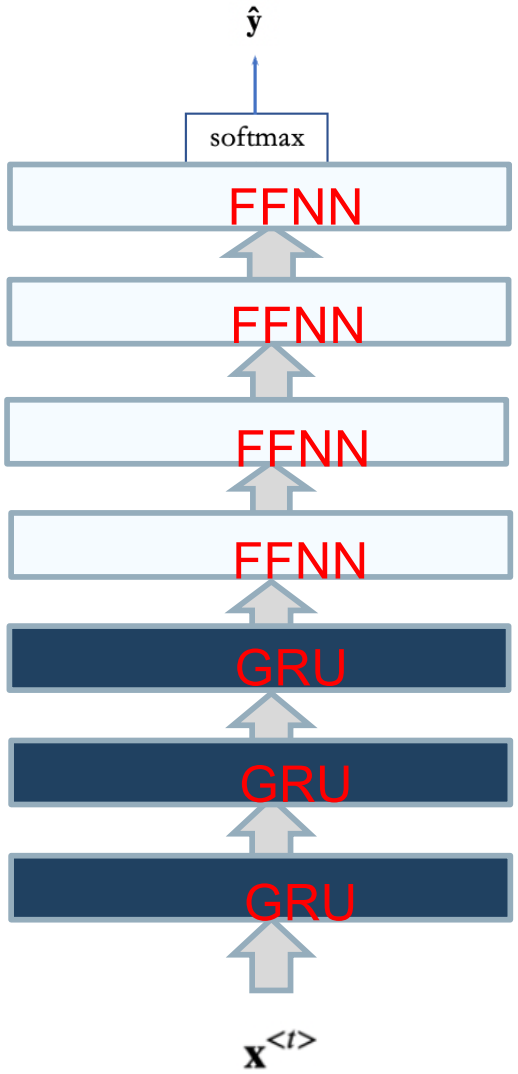


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

RNN Network Architectures

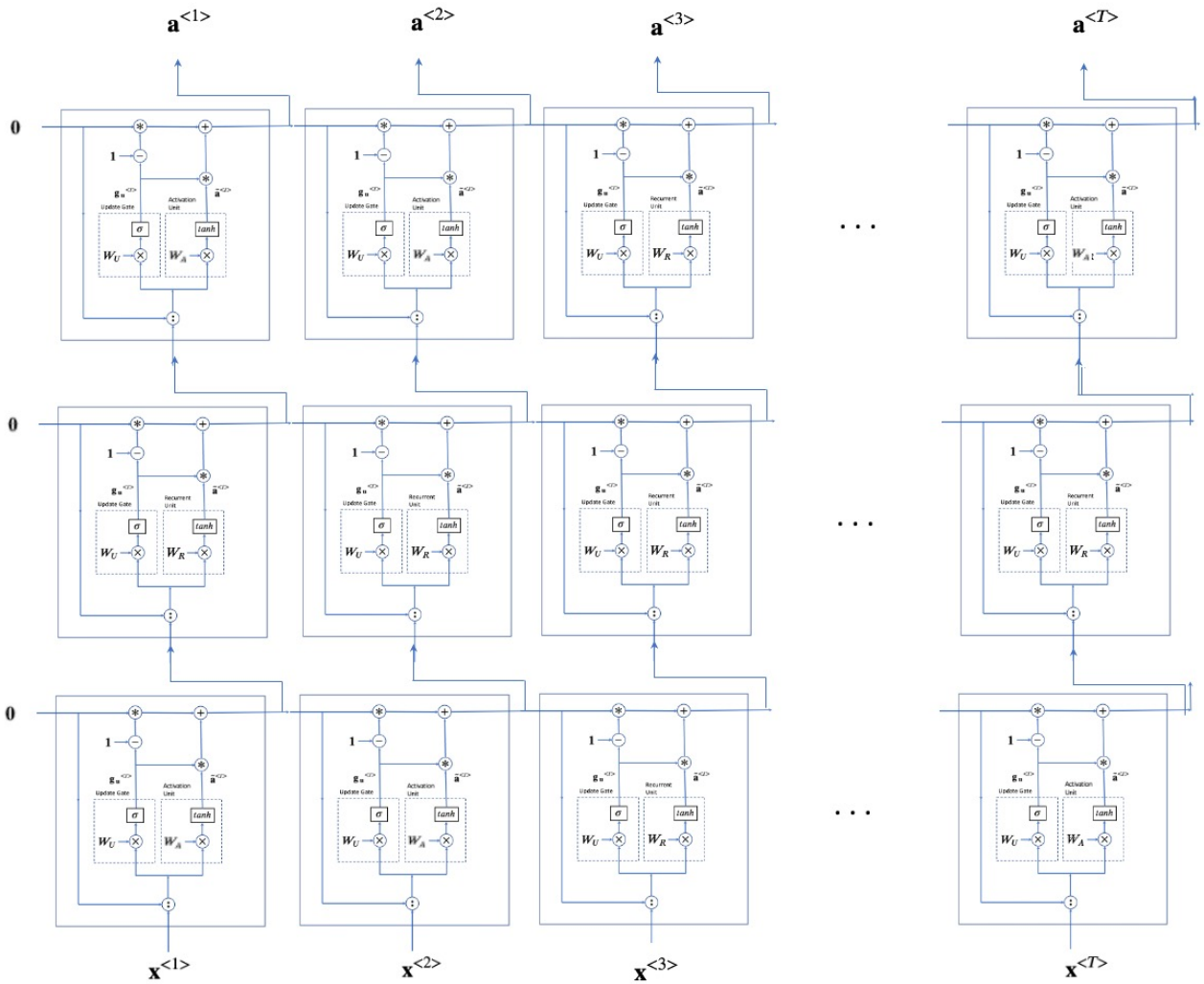
Deep Networks

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.



Recurrent Neural Networks: RNNs, GRUs, LSTMs

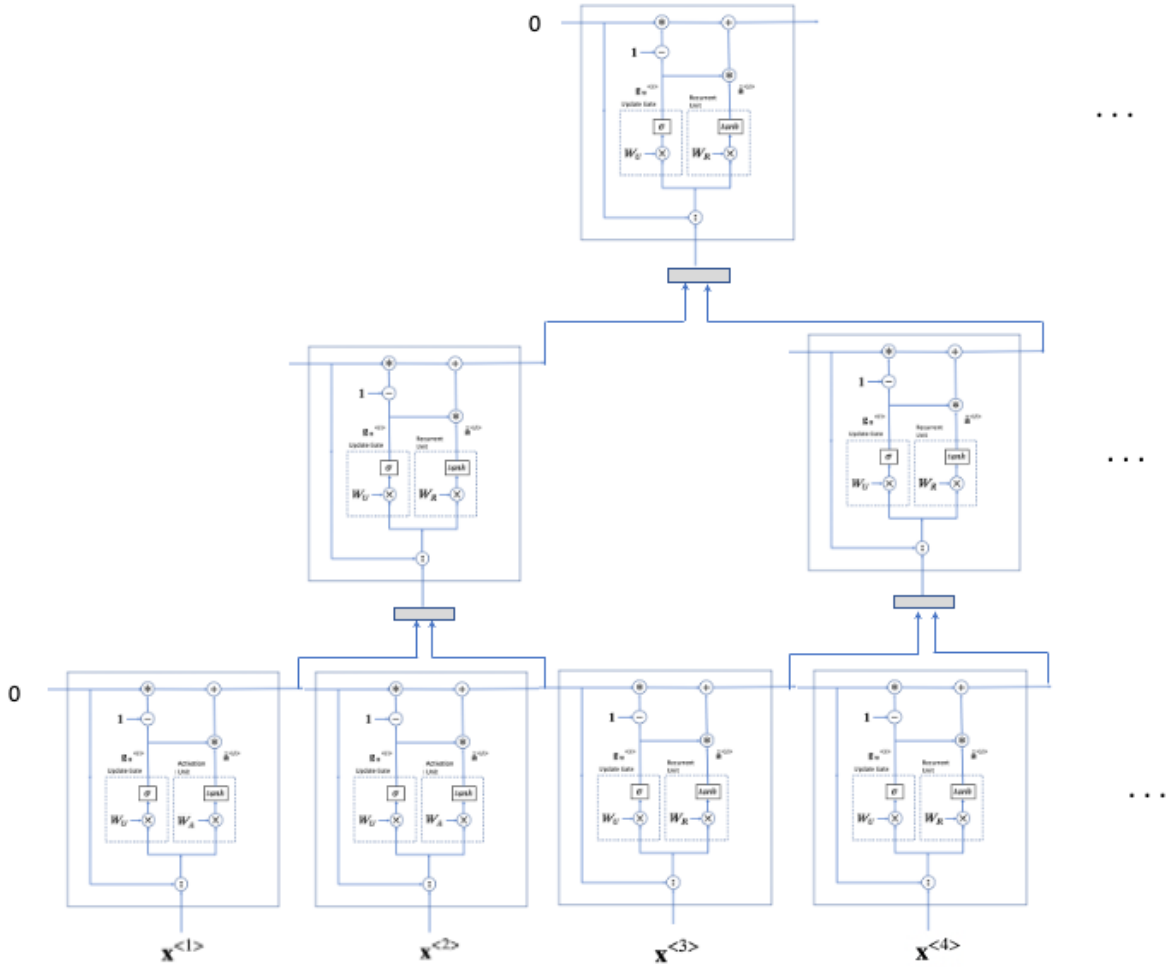
Unrolling a deep RNN network reveals a very complicated design!



Recurrent Neural Networks: RNNs, GRUs, LSTMs

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

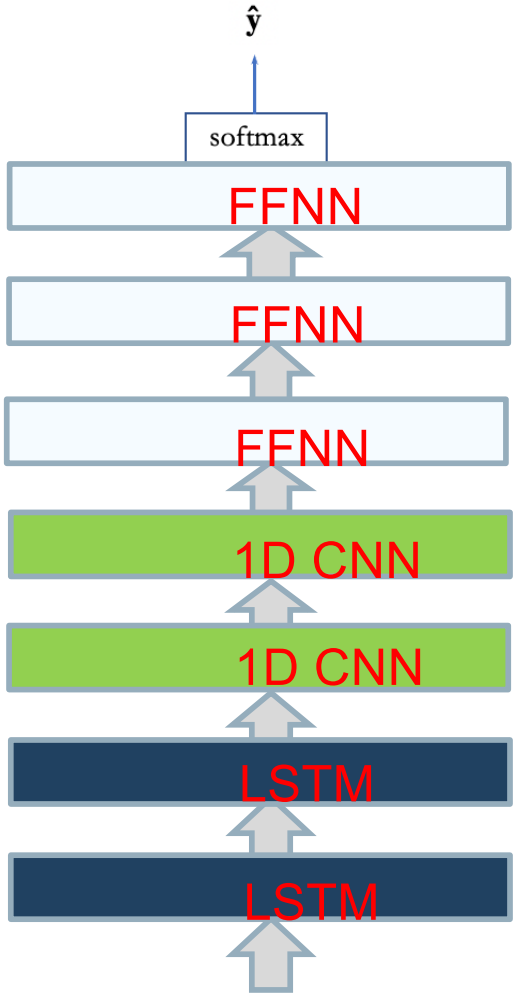
Idea 1: Tree-structured network which combines lower levels using some aggregating function (weighted) sum, perhaps controlled by a gate.



Recurrent Neural Networks: RNNs, GRUs, LSTMs

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

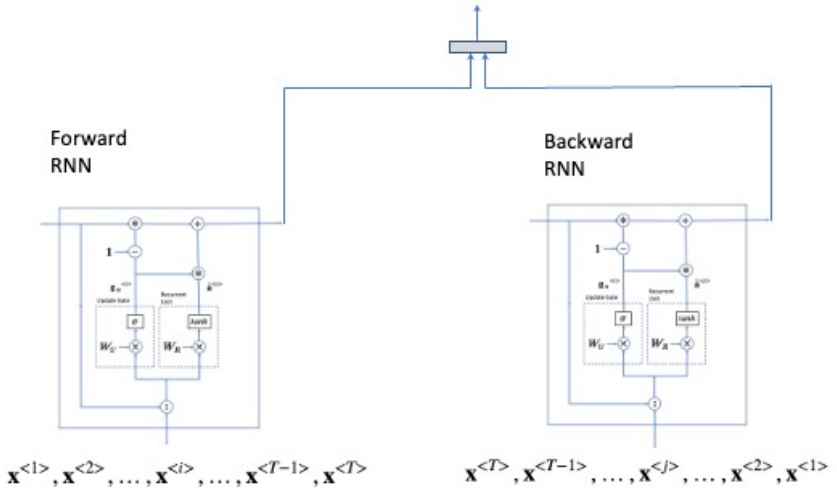
Idea 2: Apply 1D Convolutions to the RNN layers.



Recurrent Neural Networks: RNNs, GRUs, LSTMs

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

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



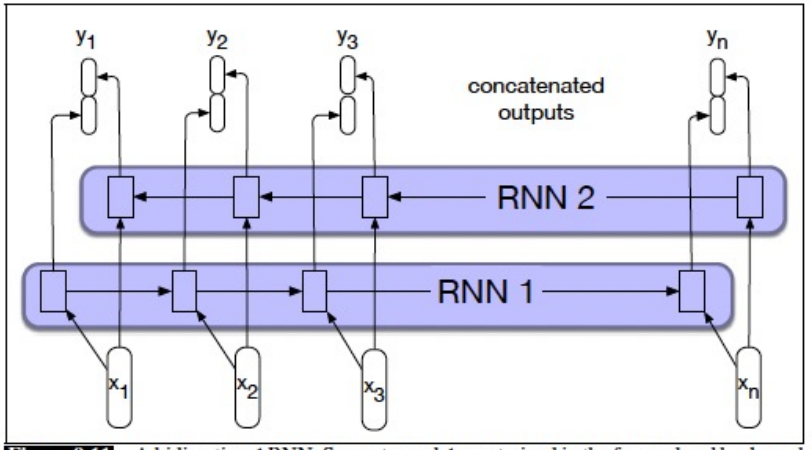


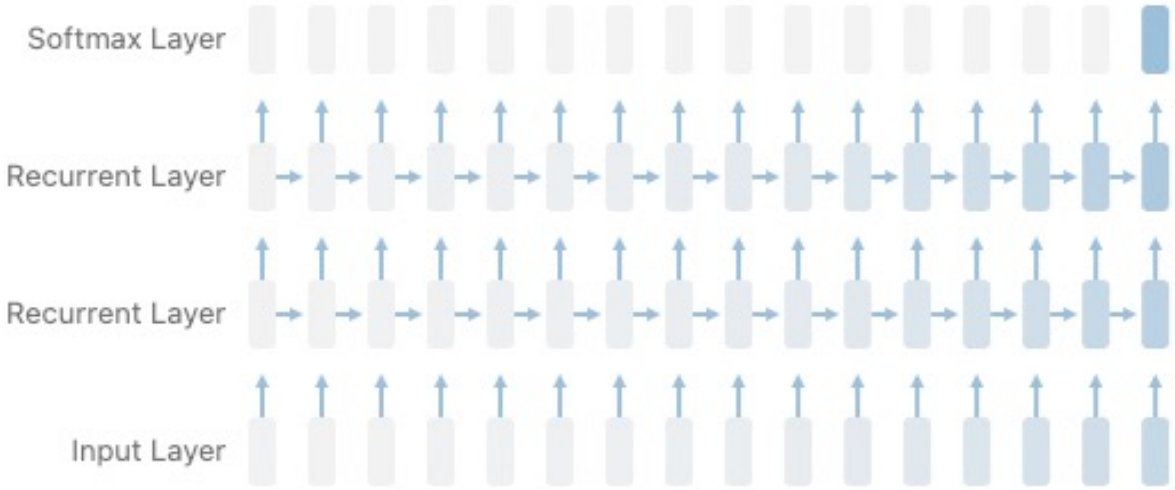
Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

RNN Network Architectures

Unfortunately, all this complexity comes with a cost!

Deep recurrent networks have a large number of parameters and take a long time to train.

Plus, we have 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.

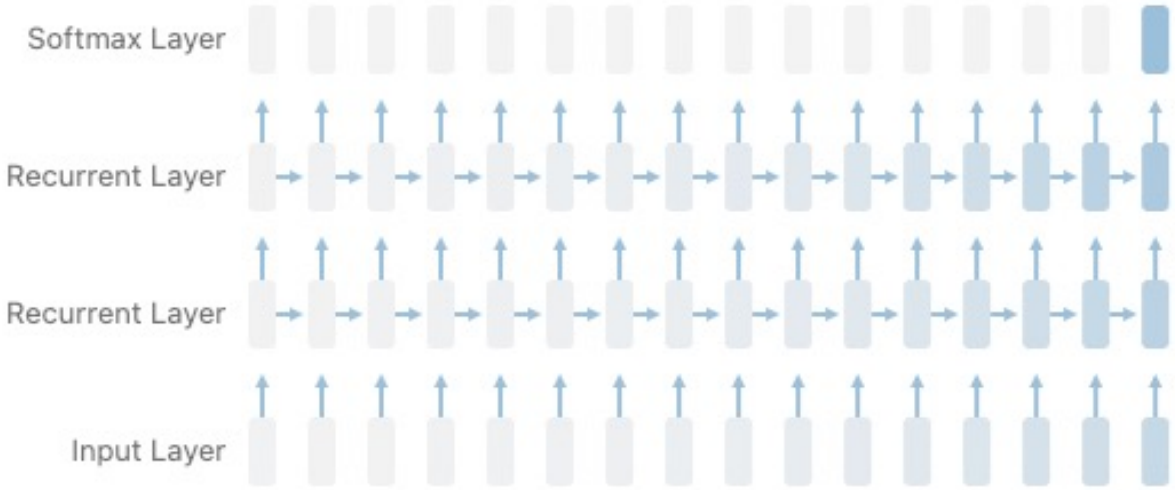
There is a symmetric problem, called the Exploding Gradients Problem!

RNN Network Architectures

Unfortunately, all this complexity comes with a cost!

Deep recurrent networks have a large number of parameters and take a long time to train.

Plus, we have 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.

There is a symmetric problem, called the Exploding Gradients Problem!

Fun with Character-Level RNN Text Generation

From. The Unreasonable Effectiveness of Recurrent Neural Networks

The evolution of samples while training

First, it's fun to look at how the sampled text evolves while the model trains. For example, I trained an LSTM of Leo Tolstoy's War and Peace and then generated samples every 100 iterations of training. At iteration 100 the model samples random jumbles:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

However, notice that at least it is starting to get an idea about words separated by spaces. Except sometimes it inserts two spaces. It also doesn't know that comma is amost always followed by a space. At 300 iterations we see that the model starts to get an idea about quotes and periods:

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

Fun with Character-Level RNN Text Generation

From. The Unreasonable Effectiveness of Recurrent Neural Networks

The words are now also separated with spaces and the model starts to get the idea about periods at the end of a sentence. At iteration 500:

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```

the model has now learned to spell the shortest and most common words such as “we”, “He”, “His”, “Which”, “and”, etc. At iteration 700 we’re starting to see more and more English-like text emerge:

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

Fun with Character-Level RNN Text Generation

From. The Unreasonable Effectiveness of Recurrent Neural Networks

At iteration 1200 we're now seeing use of quotations and question/exclamation marks. Longer words have now been learned as well:

```
"Kite vouch!" he repeated by her
door. "But I would be done and quarts, feeling, then, son is people...."
```

Until at last we start to get properly spelled words, quotations, names, and so on by about iteration 2000:

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

The picture that emerges is that the model first discovers the general word-space structure and then rapidly starts to learn the words; First starting with the short words and then eventually the longer ones. Topics and themes that span multiple words (and in general longer-term dependencies) start to emerge only much later.