COMP9444: Neural Networks and Deep Learning

Week 5a. Recurrent Neural Networks (RNNs)

Sonit Singh

School of Computer Science and Engineering

June 25, 2024

Outline

- Processing Temporal Sequences
- Sliding Window
- Recurrent Network Architectures
- Hidden Unit Dynamics
- Long Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

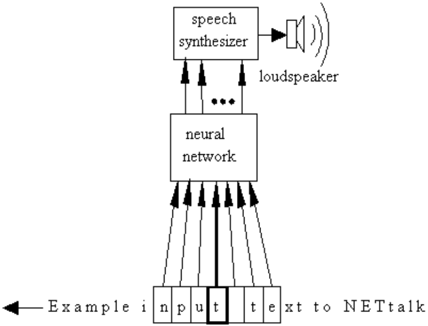
Processing Temporal Sequences

There are many tasks which require a sequence of inputs to be processed rather than a single input.

- speech recognition
- time series prediction
- machine translation
- handwriting recognition

How can neural network models be adapted for these tasks?

Sliding Window



The simplest way to feed temporal input to a neural network is the "sliding window" approach, first used in the NetTalk system (Sejnowski & Rosenberg, 1987).

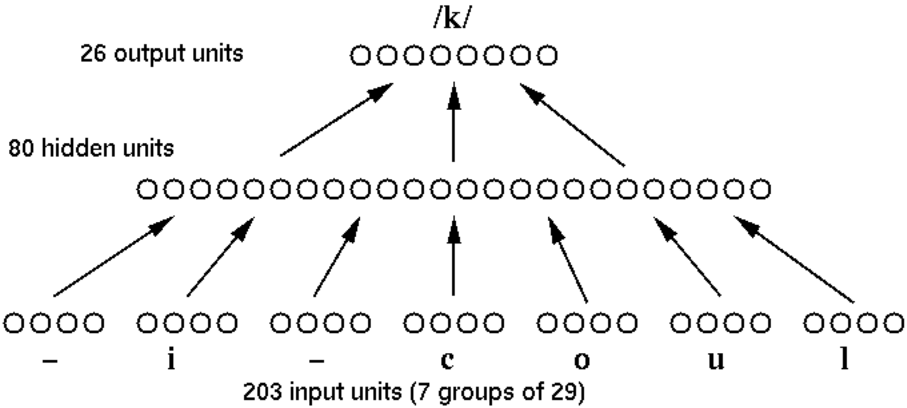
NetTalk Task

Given a sequence of 7 characters, predict the phonetic pronunciation of the middle character.
For this task, we need to know the characters on both sides.
For example, how are the vowels in these words pronounced?

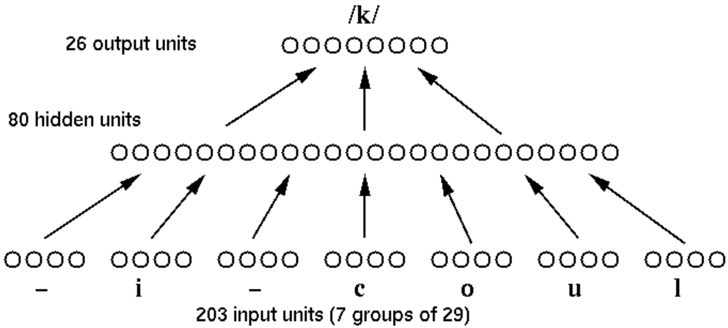
pa pat pate paternal

mo mod mode modern

NetTalk Architecture



NetTalk Test

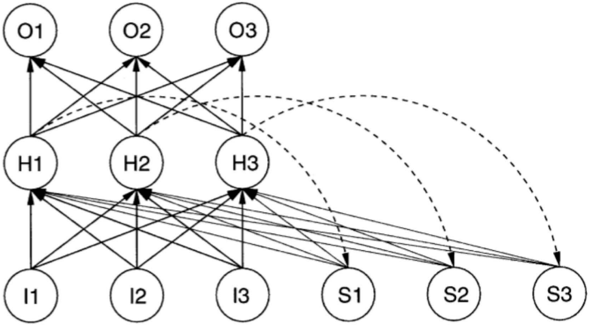


https://www.youtube.com/watch?v=gakJlr3GecE

NetTalk

- NETtalk gained a lot of media attention at the time.
- Hooking it up to a speech synthesizer was very cute. In the early stages of training, it sounded like a babbling baby. When fully trained, it pronounced the words mostly correctly (but sounded somewhat robotic).
- Later studies on similar tasks have often found that a decision tree could produce equally good or better accuracy.
- This kind of approach can only learn short term dependencies, not the medium or long term dependencies that are required for some tasks.

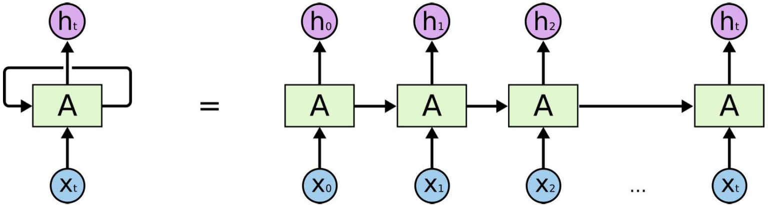
Simple Recurrent Network (Elman, 1990)



- at each time step, hidden layer activations are copied to “context” layer
- hidden layer receives connections from input and context layers
- the inputs are fed one at a time to the network, it uses the context layer to “remember” whatever information is required for it to produce the correct output

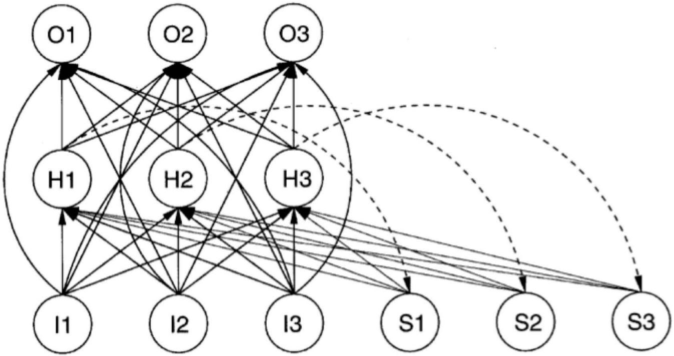
UNSW SYDNEY

Back Propagation Through Time



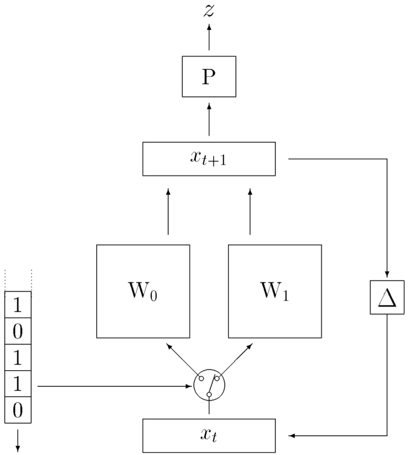
- we can "unroll" a recurrent architecture into an equivalent feedforward architecture, with shared weights
- applying backpropagation to the unrolled architecture is reffered to as "backpropagation through time"
- we can backpropagate just one timestep, or a fixed number of timesteps, or all the way back to beginning of the sequence

Other Recurrent Neural Architectures



- it is sometimes beneficial to add "shortcut" connections directly from input to output
- connections from output back to hidden have also been explored (sometimes called "Jordan Networks")

Second Order (or Gated) Networks



$$x_t^j = \tanh(\mathrm{W}_{\sigma_t}^{j0} + \sum_{k=1}^{d} \mathrm{W}_{\sigma_t}^{jk} x_{t-1}^k)$$

$$z = \tanh(\mathrm{P}_0 + \sum_{j=1}^{d} \mathrm{P}_j \, x_n^j)$$

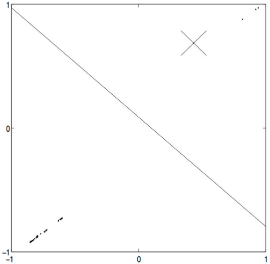
Task: Formal Language Recognition

Accept	Reject
1	0
1 1	1 0
1 1 1	0 1
1 1 1 1	0 0
1 1 1 1 1	0 1 1
1 1 1 1 1 1	1 1 0
1 1 1 1 1 1 1	1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1	1 0 1 1 1 1 1 1

Scan a sequence of characters one at a time,

then classify the sequence as Accept or Reject.

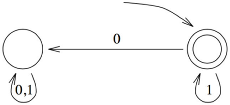
Dynamic Recognizers



$$W_0 = \begin{bmatrix} -0.89 & -0.09 & -0.14 \\ -1.13 & -0.09 & -0.14 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.20 & 0.68 & 0.96 \\ 0.20 & 0.81 & 1.19 \end{bmatrix}$$

$$P = \begin{bmatrix} -0.07 & 0.66 & 0.75 \end{bmatrix}$$



- gated network trained by BPTT
- emulates exactly the behaviour of Finite State Automaton

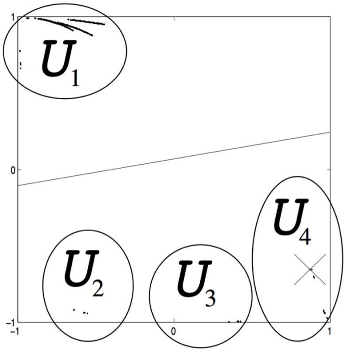
Task: Formal Language Recognition

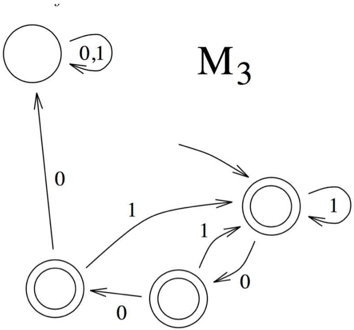
Accept	Reject
1	0 0 0
0	1 1 0 0 0
1 0	0 0 0 1
0 1	0 0 0 0 0 0 0 0 0
0 0	1 1 1 1 1 0 0 0 0 1 1
1 0 0 1 0 0	1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1
0 0 1 1 1 1 1 1 0 1 0 0	1 0 1 0 0 1 0 0 0 1
0 1 0 0 1 0 0 1 0 0	0 0 0 0
1 1 1 0 0	0 0 0 0 0
0 0 1 0	

Scan a sequence of characters one at a time,

then classify the sequence as Accept or Reject.

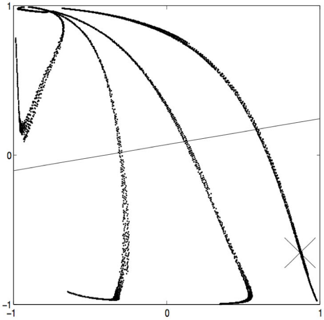
Dynamical Recognizers

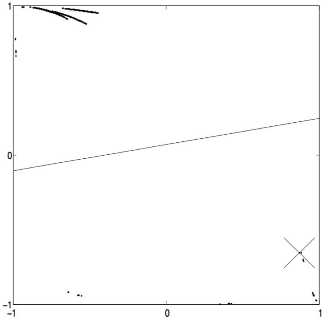




- trained network emulates the behaviour of Finite State Automaton
- training set must include short, medium and long examples

Phase Transition





$$W_0 = \begin{bmatrix} -0.567 & 1.761 & 0.815 \\ -0.219 & -2.591 & 0.446 \end{bmatrix}$$
$$W_1 = \begin{bmatrix} 0.752 & 0.548 & -1.071 \\ 0.074 & -0.813 & 1.502 \end{bmatrix}$$
$$P = \begin{bmatrix} 0.069 & 0.172 & -0.985 \end{bmatrix}$$

$$W_0 = \begin{bmatrix} -0.567 & 1.763 & 0.816 \\ -0.219 & -2.593 & 0.446 \end{bmatrix}$$
$$W_1 = \begin{bmatrix} 0.751 & 0.549 & -1.073 \\ 0.075 & -0.813 & 1.502 \end{bmatrix}$$
$$P = \begin{bmatrix} 0.069 & 0.173 & -0.985 \end{bmatrix}$$

Chomsky Hierarchy

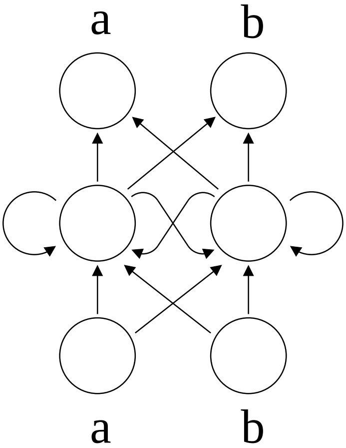
Language	Machine	Example
Regular	Finite State Automaton	a^n (n odd)
Context Free	Push Down Automaton	$a^n b^n$
Context Sensitive	Linear Bounded Automaton	$a^n b^n c^n$
Recursively Enumerable	Turing Machine	true QBF

Task: Formal Language Prediction

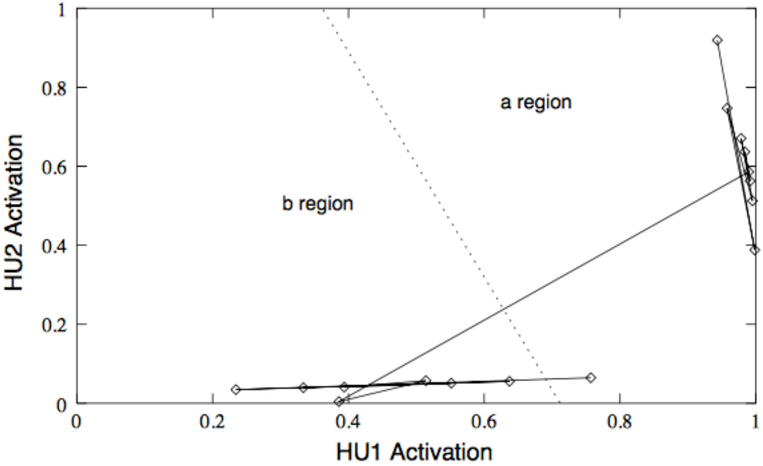
abaabbabaaabbbaaaabbbbabaabbaaaaabbbbb...

- Scan a sequence of characters one at a time, and try at each step to predict the next character in the sequence.
- In some cases, the prediction is probabilistic.
- For the $a^n b^n$ task, the first *b* is not predictable, but subsequent *b*'s and the initial *a* in the next subsequence are predictable.

Elman Network for predicting $a^n b^n$



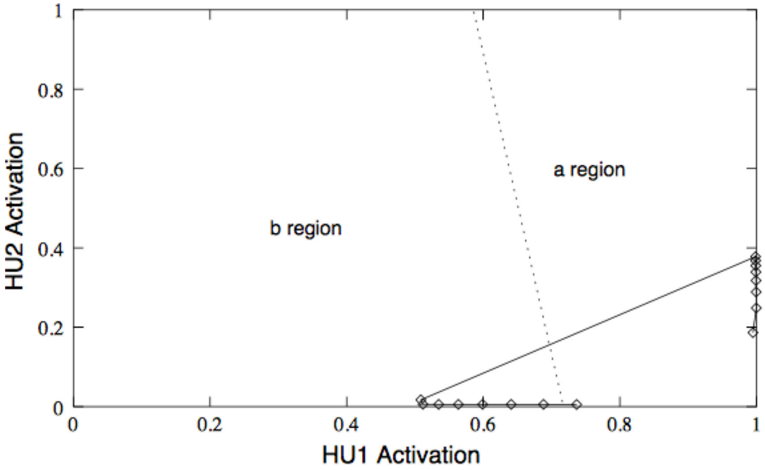
Oscillating Solution for $a^n b^n$



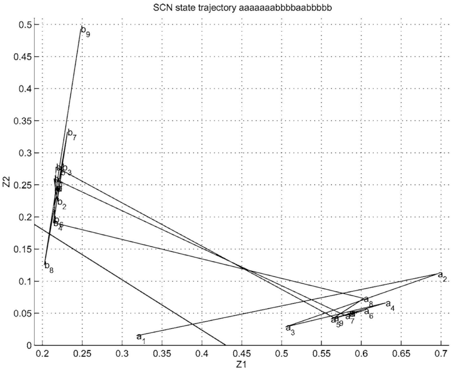
Learning to Predict $a^n b^n$

- the network does not implement a Finite State Automaton but instead uses two fixed points in activation space – one attracting, the other repelling (Wiles & Elman, 1995)
- networks trained only up to $a^{10}b^{10}$ could generalize up to $a^{12}b^{12}$
- training the weights by evolution is more stable than by backpropagation
- networks trained by evolution were sometimes monotonic rather than oscillating

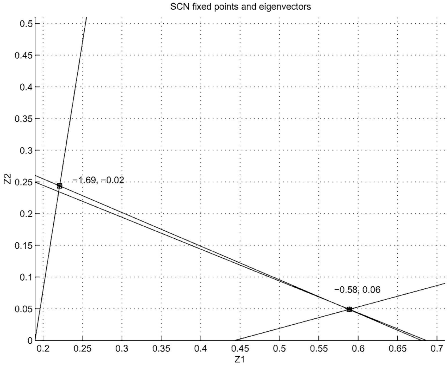
Monotonic Solution for $a^n b^n$



Hidden Unit Analysis for $a^n b^n$

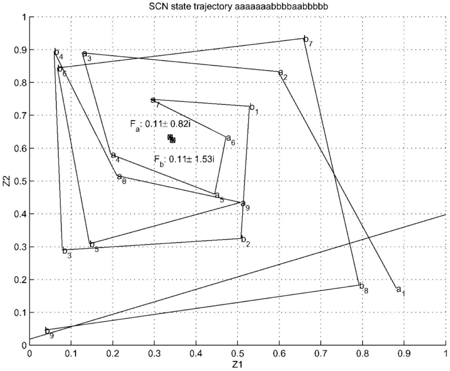


hidden unit trajectory



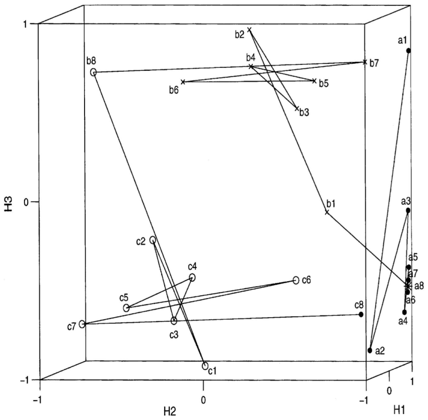
fixed points and eigenvectors

Counting by Spiralling



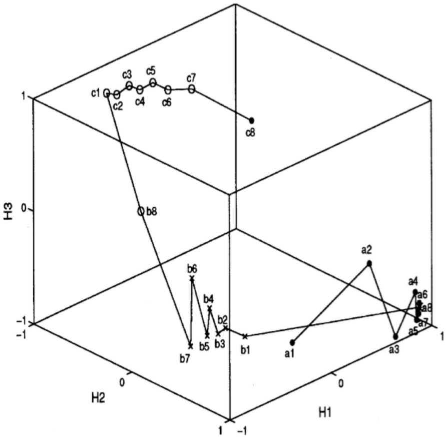
- for this task, sequence is accepted if the number of *a*'s and *b*'s are equal
- network counts up by spiralling inwards, down by spiralling outwards

Hidden Unit Dynamics for $a^n b^n c^n$

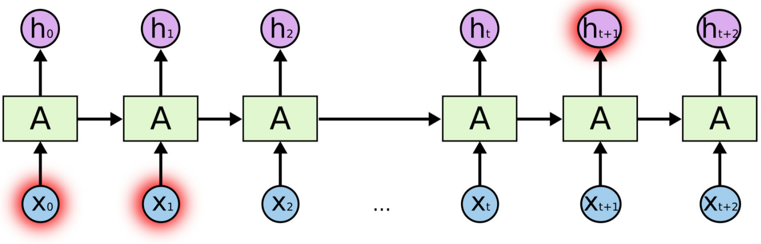


- SRN with 3 hidden units can learn to predict $a^n b^n c^n$ by counting up and down simultaneously in different directions, thus producing a star shape.

Partly Monotonic Solution for $a^n b^n c^n$



Long Range Dependencies



- Simple Recurrent Networks (SRNs) can learn medium-range dependencies but have difficulty learning long range dependencies
- Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) can learn long range dependencies better than SRN

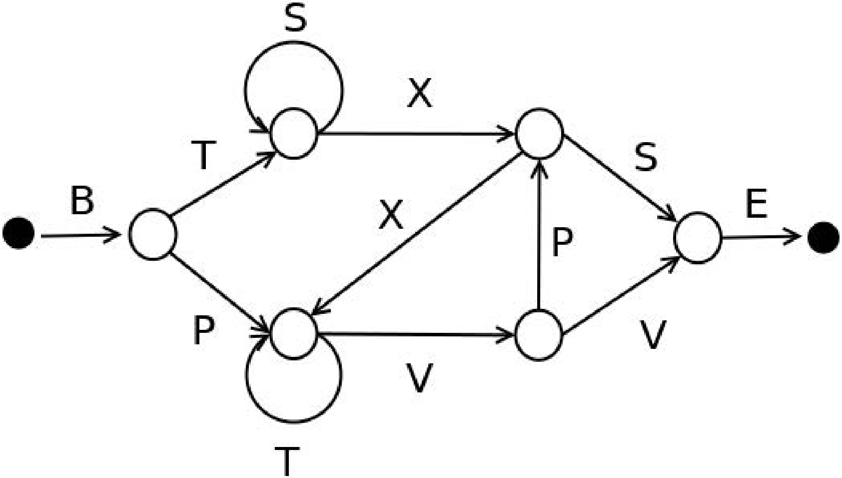
Long Short Term Memory

Two excellent Web resources for LSTM:

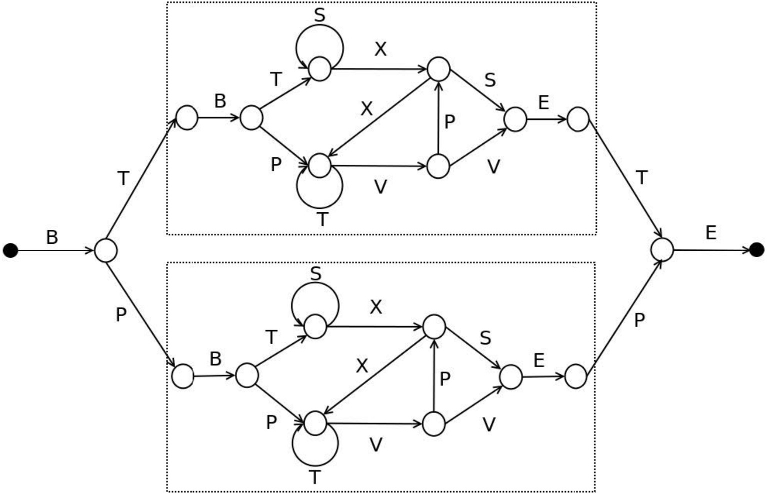
`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

`christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php`

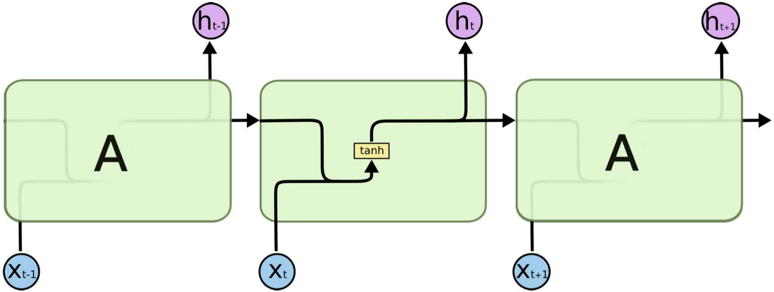
Reber Grammar



Embedded Reber Grammar

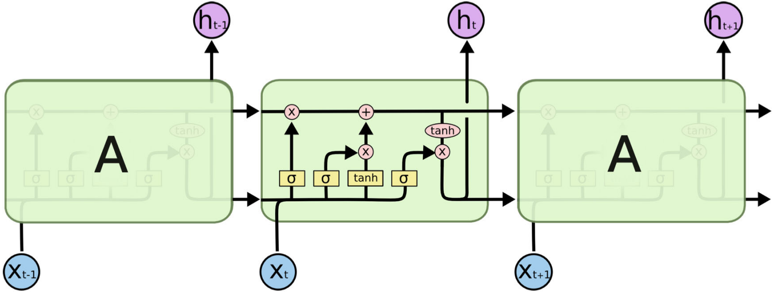


Simple Recurrent Network



- SRN – context layer is combined directly with the input to produce the next hidden layer.
- SRN can learn Reber Grammar, but not Embedded Reber Grammar.

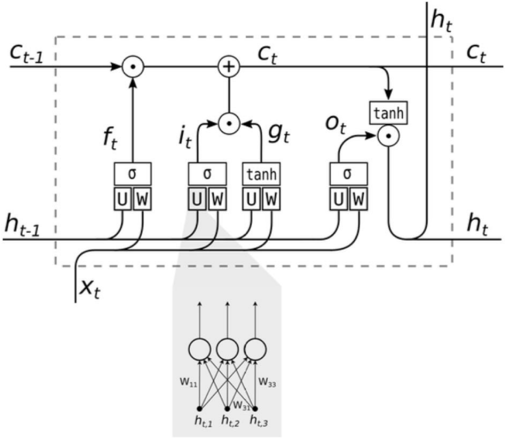
Long Short Term Memory



- LSTM – context layer is modulated by three gating mechanisms: forget gate, input gate and output gate.

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

Long Short Term Memory



Gates:

$$\mathbf{f}_t = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f\right)$$
$$\mathbf{i}_t = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i\right)$$
$$\mathbf{g}_t = \tanh\left(W_g \mathbf{x}_t + U_g \mathbf{h}_{t-1} + \mathbf{b}_g\right)$$
$$\mathbf{o}_t = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o\right)$$

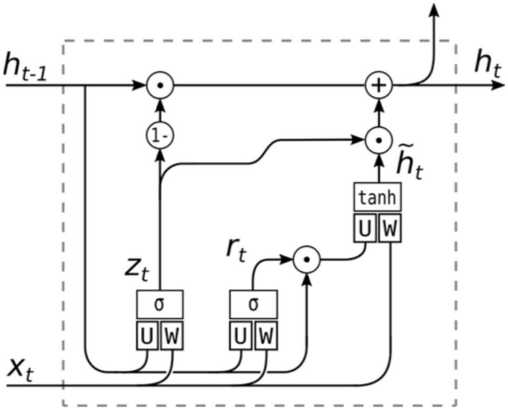
State:

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \mathbf{g}_t$$

Output:

$$\mathbf{h}_t = \tanh \mathbf{c}_t \odot \mathbf{o}_t$$

Gated Recurrent Unit



Gates:

$$\mathbf{z}_t = \sigma(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1} + \mathbf{b}_z)$$

$$\mathbf{r}_t = \sigma(W_r\mathbf{x}_t + U_r\mathbf{h}_{t-1} + \mathbf{b}_r)$$

Candidate Activation:

$$\tilde{\mathbf{h}}_t = \tanh\left(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h\right)$$

Output:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

GRU is similar to LSTM but has only two gates instead of three.

End-to-End Text Classification

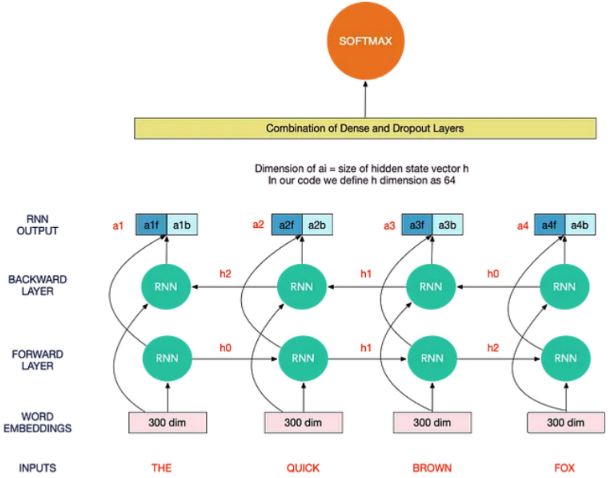
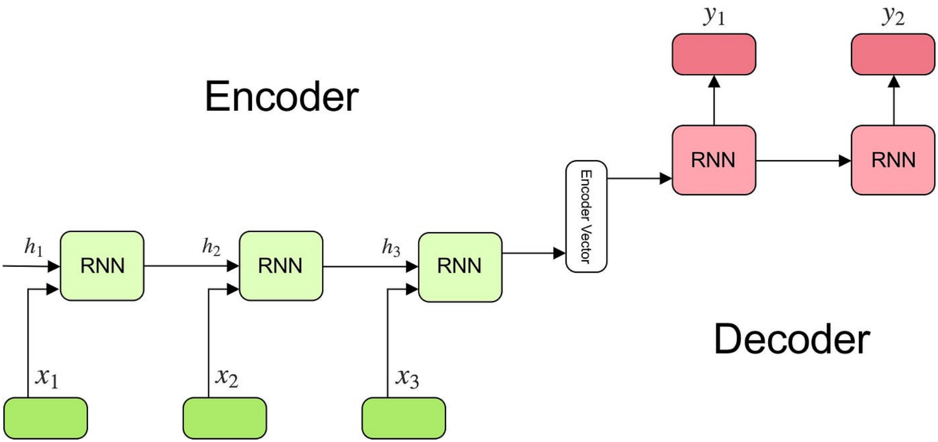


Image Credit: https://towardsdatascience.com/using-deep-learning-for-end-to-end-multiclass-text-classification-39b46aecac81

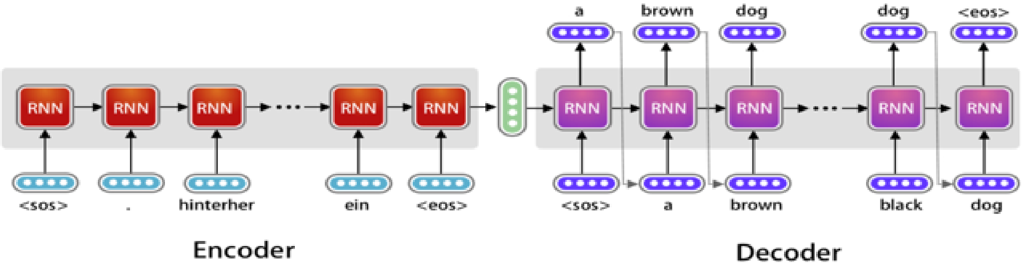
Seq2Seq model

Encoder-Decoder Framework



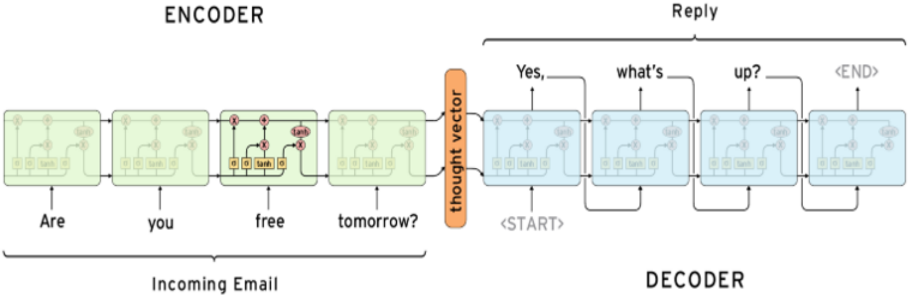
Seq2Seq model applications

Machine Translation



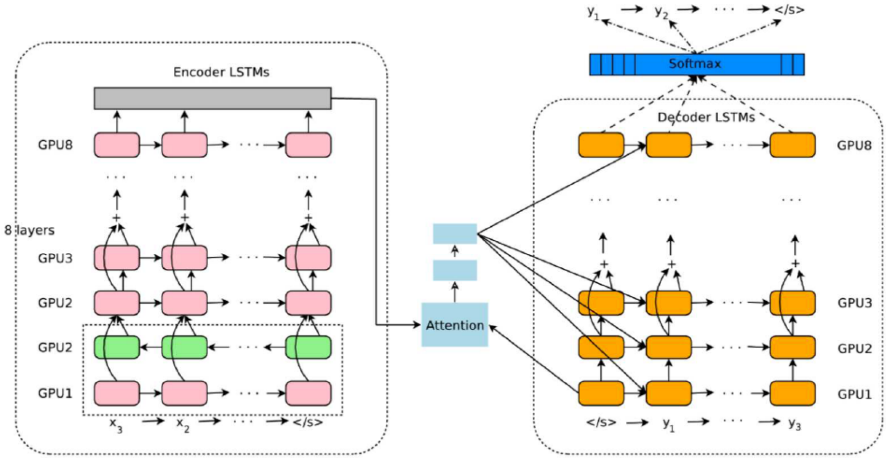
Seq2Seq model applications

Automatic Email Reply



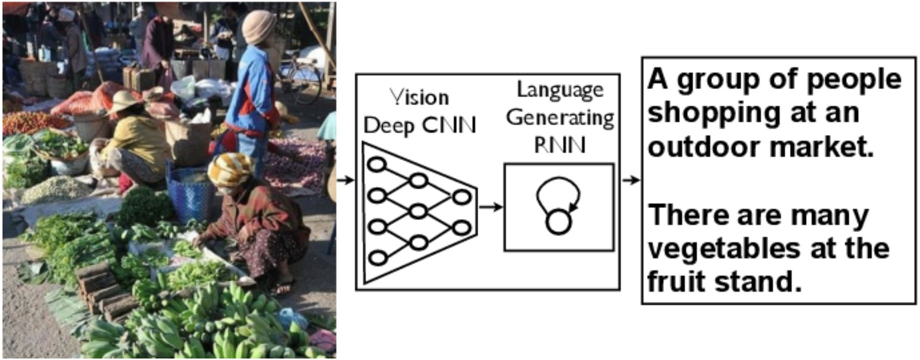
Seq2Seq model applications

Google's Neural Machine Translation



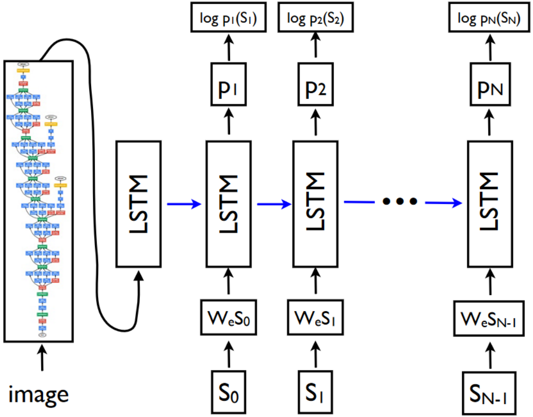
CNNs + LSTM : Image Captioning

Show and Tell: Neural Image Caption Generator (Vinyals et al. 2015)



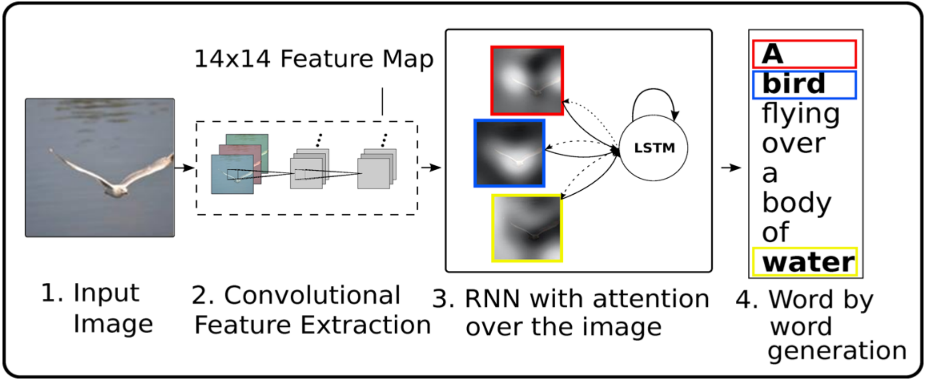
CNNs + LSTM : Image Captioning

Show and Tell: Neural Image Caption Generator (Vinyals et al. 2015)



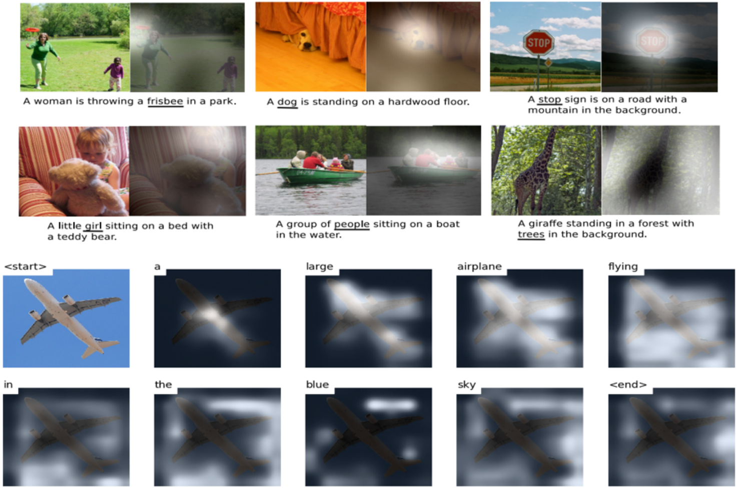
CNNs + LSTM : Image Captioning

Show, Attend and Tell (Xu et al. 2015)



CNNs + LSTM : Image Captioning

Show, Attend and Tell (Xu et al. 2015) Examples



Summary

- Recurrent Neural Networks (RNNs) are specialised neural networks suitable for modelling sequential or time-series data.
- RNNs have a looping mechanism that acts as a highway to allow information to flow from one step to the next. This information is the hidden state, which is a representation of previous inputs.
- Simple RNNs suffers from vanishing gradient problem
  - As the RNNs processes more steps, it has troubles retaining information from previous steps.
  - Due to back-propagation, the earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients.
  - Does not learn the long-range dependencies across time steps
- LSTMs and GRUs are two special RNNs, capable of learning long-term dependencies using mechanisms called **gates**.
- These gates are different tensor operations that can learn what information to add or remove to the hidden state.