# Machine Learning I

## Lecture 10: Modern Neural Network Architecture: RNNs

Nathaniel Bade

Northeastern University Department of Mathematics

## Table of contents

## Types of Artificial Neural Networks

## Genre of Artificial Neural Networks

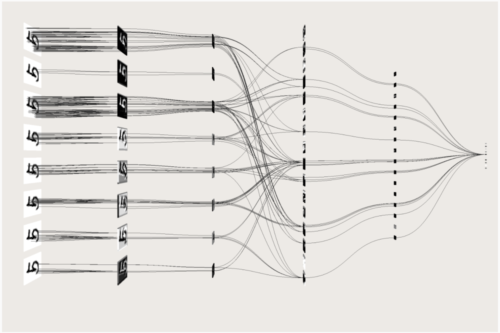
Modern artificial neural networks can be sorted into three broad categories based on their structure:

Feed forward networks: A trained feed forward network acts like a function, taking in a set of data at one end and returning a new set of data at the other.

Recurrent networks: Trained recurrent networks are stateful. That is, RNN's take data and return an output but the remember the last $M$ pieces of data sent through in an internal **state**.

Symmetrically Connected Networks: A trained SCN is a densely connected network with an update rule. For any initial value of the nodes, the function "updates," moving at each step towards a "lower energy state". The result is achieved when updating no longer changes the sate.

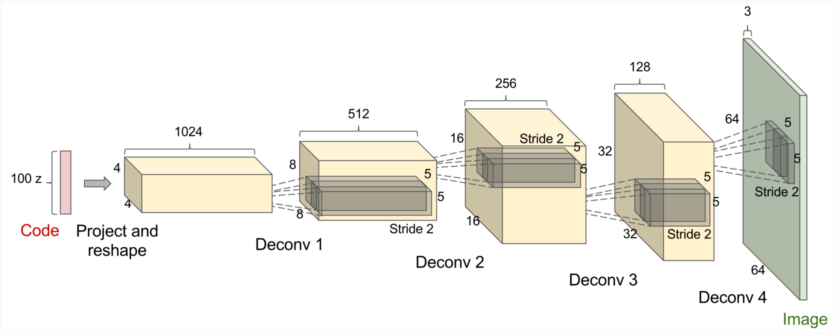
## Feed Forward Networks



Feed forward networks are the most common type, they take an input, process it through a series of operation, and return some output.

Mathematically they are just functions $f_\beta(X)$ depending on some trainable parameters $\beta$. They could result in a classifier $\hat{y} = f_\beta(X)$, or a loss function $\ell(X, z) = f_\beta(X)$.

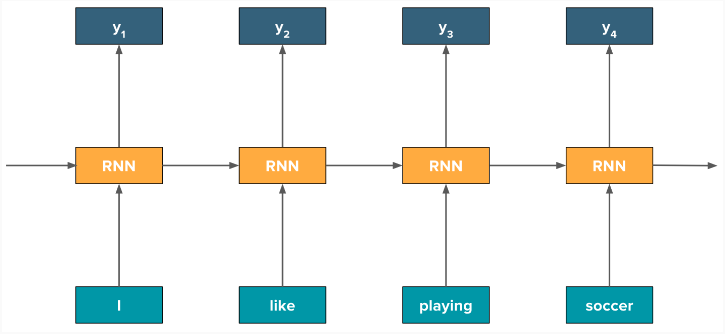
## Feed Forward Networks



For example feed forward networks can be labeling or regression, but they can also be generative networks (returning more data) or unsupervised, returning a dimensional reduction, clustering or other description of the data.

However, once trained the weights $\beta$ are **fixed** for all prediction.

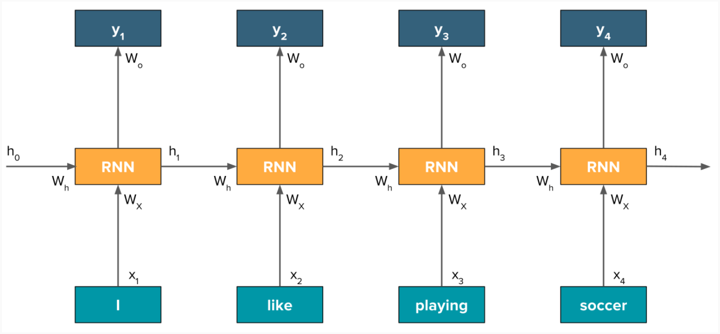
## Recurrent Networks



A recurrent neural network has state variables that can be changed at runtime and that persist between prediction runs.

For example, in text prediction an RNN may predict one word at a time while "remembering" its previous predictions.

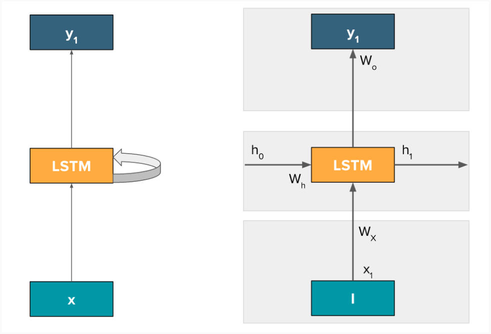
## Recurrent Networks



A recurrent neural network on the other hand has state variables that can be changed at runtime and that persist between prediction runs.

For example, in text prediction an RNN may predict one word at a time while "remembering" its previous predictions.

## Recurrent Networks



A recurrent node is often represented in “wrapped” form as above. RNN’s are used extensively in time series prediction and natural language processing, although there success is still under scrutiny.

## Types of Feed Forward Networks

The following are a list of common network types and their uses

**Multilayer perceptron networks** - Stacks of almost linear classifiers for labeling.

**Radial basis function networks** - Perceptron algorithms tweaked to find cluster centers.

**Convolutional Networks** - Spatially aware classifiers.

**Autoencoders** - Dimensional reduction networks.

**Generative adversarial network** - Example generation networks.

Fjodor Van Veen has compiled a zoo of common architectures:
`https://www.asimovinstitute.org/author/fjodorvanveen/`

## Summary

We will now go into the details of convolutional neural networks. Convolutional networks differ from perceptrons by building spacial reasoning directly into their architecture.

In the third part of the lecture, we will discuss **long term short term** (**LSTM**) networks as examples of recurrent networks. Both of these architectures are implemented in Tensorflow and Keras, and are still not completely understood mathematically.

Almost all modern networks are built out variations of perceptrons, CNNs and RNNs.

## Recurrent Networks

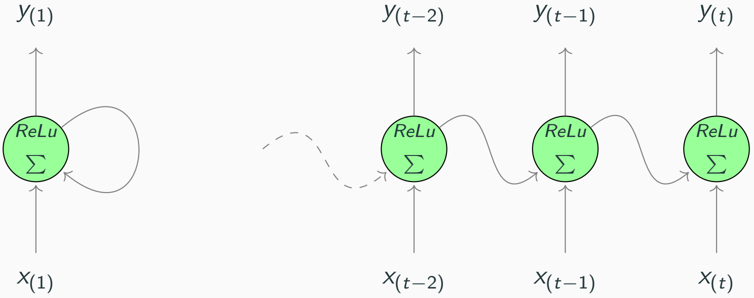
## Recurrent Networks

Recurrent networks are designed to predict not just one, but a whole series of events while incorporating their previous predictions into future ones. They can analyze time series data like stock prices, network traffic, or team performance and produce an arbitrary amount of new data. RNN's can work locally on large sequences, and so can take in a much wider variety of data.

In addition, being statefull, they can interact with humans: You can ask them to predict the 10 most likely next words in a sentence (or notes in a song) and have a human pick the best one over and over. By training the network on different genres, new works in old styles can be co-composed.

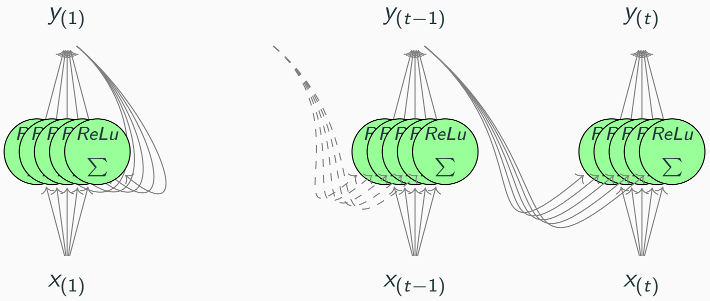
`https://botnik.org/content/harry-potter.html`

## Recurrent Nodes



Unlike feed forward networks, recurrent networks contain nodes that connect back to themselves. Looking at the simplest (one neuron) network, at each **time step** or **frame** $t$, the RNN receives inputs from $x_{(t)}$ and the previous time-step $y_{(t-1)}$.

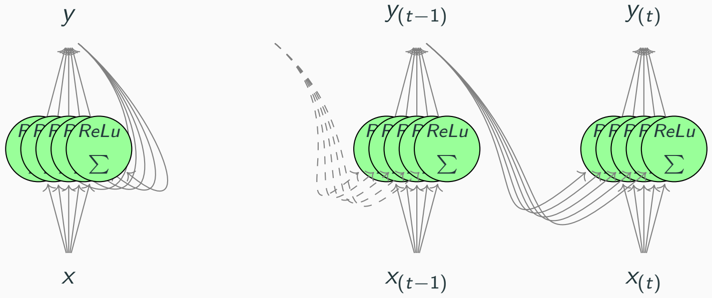
## Recurrent Nodes



Multiple recurrent neurons can be connected together like a perceptron layer. Then each neuron has two sets of weights: $\mathbf{w}_x^i$ for the inputs $x_{(t)}$ and $\mathbf{w}_y^i$ for the outputs $y_{(t-1)}$. We can collect these into matrices $\mathbf{W}_x$ and $\mathbf{W}_y$. Then, for activation $\sigma$ and bias $b$,

$$\mathbf{Y}_{(t)} = \sigma\left(\mathbf{W}_x^T \mathbf{X}_{(t)} + \mathbf{W}_y^T \mathbf{Y}_{(t-1)} + b\right)$$

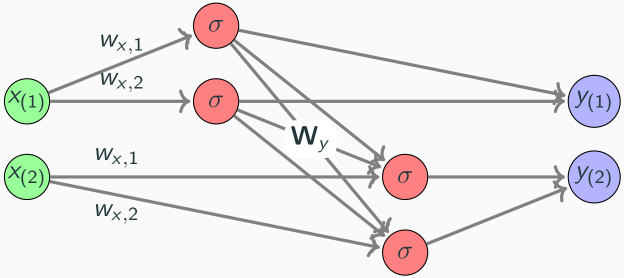
## Memory Cells



Any cell that passes information to the next frame is called a **memory cell**. In the above we pass the output of the previous frame to produce **short term memory**, but we could of course pass another state vector as well to produce **long term memory**.

## Recurrent Nodes

RNN with 2 Recreant Nodes as a Feed Forward Network



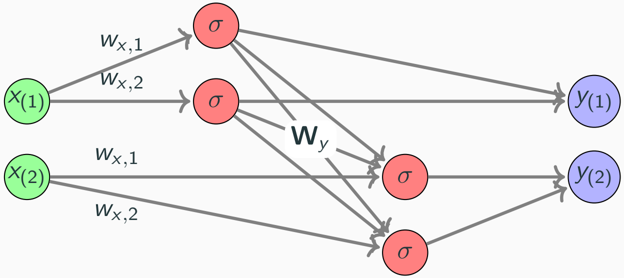
We can represent the node function

$$\mathbf{Y}_{(t)} = \sigma\big(\mathbf{W}_x^T \mathbf{X}_{(t)} + \mathbf{W}_y^T \mathbf{Y}_{(t-1)} + b\big)$$

as a feed forward network. However, we immediately see that this network will get very deep fast. In addition, since $\mathbf{Y}_{(t)}$ depends on $\mathbf{Y}_{(t-1)}$, this will be a highly nonlinear function of $\mathbf{W}_y^T$.

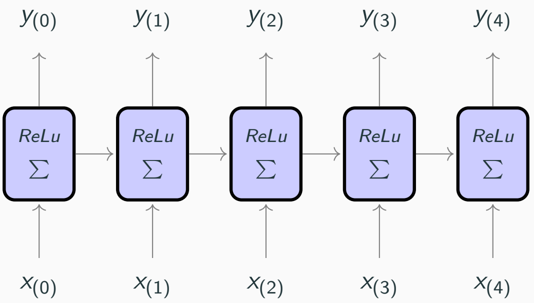
## Recurrent Nodes

RNN with 2 Recreant Nodes as a Feed Forward Network



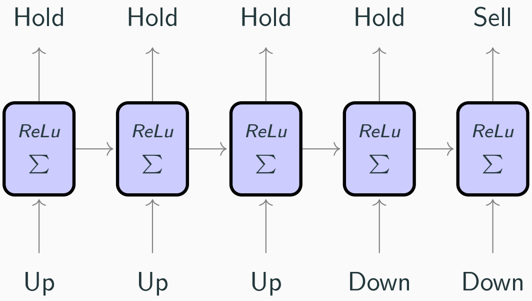
However this also gives a straightforward way to train the network: we simply unroll it into a feed forward network and train it with back propagation. Notice as well that since the weights are shared between recurrent nodes we can unroll network to any depth we want. In principle, we could train on short sequences and then use the network to predict long sequences. Practically, this means the same RNN can be used for many different tasks.

## Basic RNN Structures



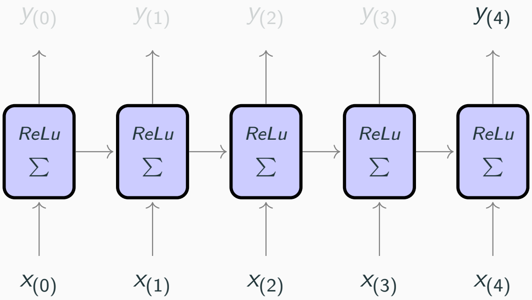
A RNN can take in a series of inputs and produce a series of outputs, as in predicting time series data like stock prices or traffic across a network.

## Basic RNN Structures



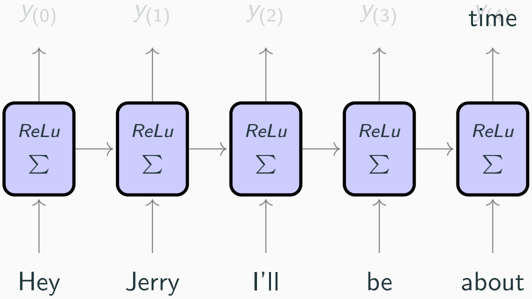
A RNN can take in a series of inputs and produce a series of outputs, as in predicting time series data like stock prices or traffic across a network.

## Basic RNN Structures



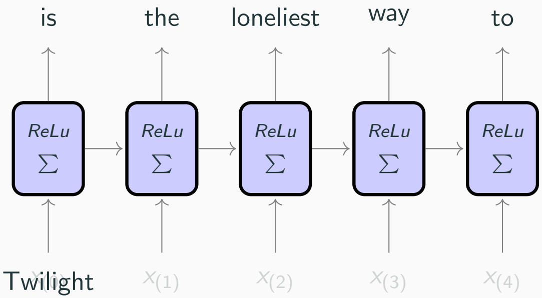
Alternatively, you can feed in many inputs but only record the last output, like in text prediction.

## Basic RNN Structures



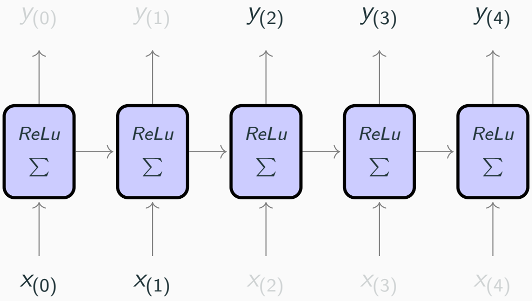
Alternatively, you can feed in many inputs but only record the last output, like in text prediction.

## Basic RNN Structures



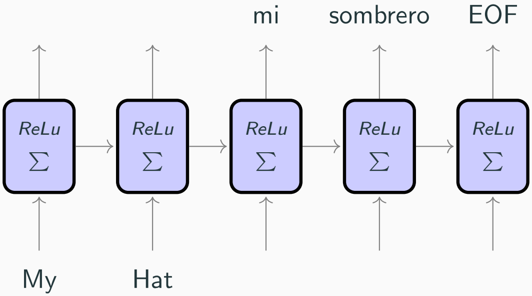
We can also use a trained RNN to predict a sequence of data by only priming on a single (or a small number) of time steps and then outputting a series data based on that input. This is one way to produce generative networks.

## Basic RNN Structures



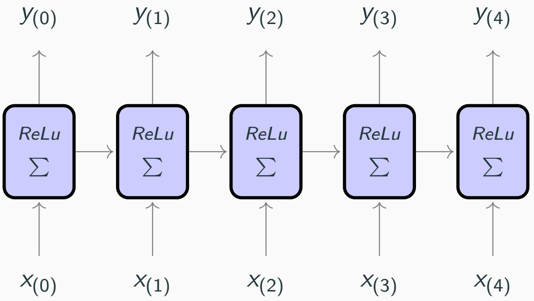
Finally, we can consider encoder/decoder networks where we feed in data encoding it, use the RNN to process it, and then return a new sequence. For example in a translation RNN you would feed *n* words into the network and return a sentence of an arbitrary length. In such a situation, you should train an "end of sentence" character to signify when the network should stop.

## Basic RNN Structures



Finally, we can consider encoder/decoder networks where we feed in data encoding it, use the RNN to process it, and then return a new sequence. For example in a translation RNN you would feed *n* words into the network and return a sentence of an arbitrary length. In such a situation, you should train an "end of sentence" character to signify when the network should stop.

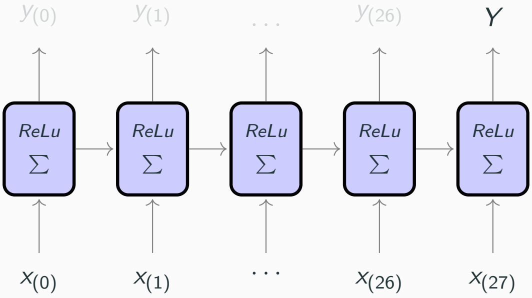
## Training: Backpropagation



To train an RNN, we unroll it to the the number of steps we require to match out input data shape and then perform standard autodiff backpropagation with input vector $\mathbf{X} = (x_{(0)}, x_{(1)}, \ldots, x_{(N)})$ and output $\mathbf{Y} = (y_{(0)}, y_{(1)}, \ldots, y_{(M)})$.

Note that we must define an appropriate cost function on $\mathbf{Y}$.

## Example: MNIST



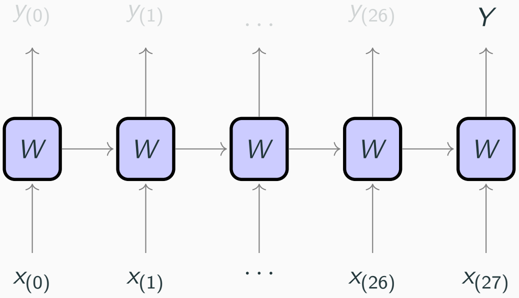
For example, we could classify $28 \times 28$ MNIST using an RNN:

Feed each row in as a sequential vector.

Output $Y$ into a 10 node dense layer

Output to a softmax layer with 0-1 loss.

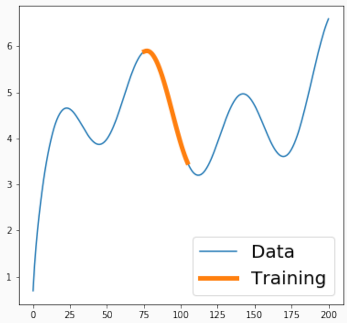
## Example: MNIST

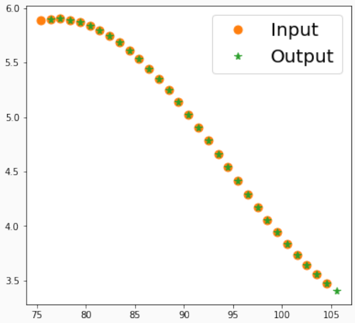


It should be noted that unrolling can be statically or dynamically implemented. Since each sequential layer in the unrolling *contains a copy of the same weights* we can train and test on data with variable lengths.

For example, if we feed a sentence in word by word, we can unroll the RNN to be as long as we need for each sentence, since we only have a single set of weights **W** that is shared across all unrolled nodes.

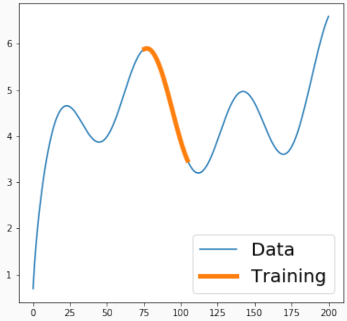
## Example: Time Series

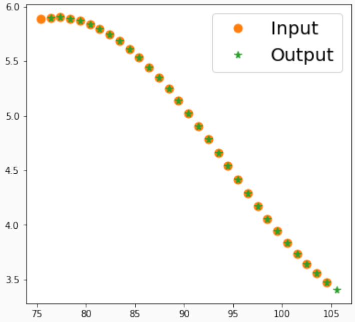




For a standard example, assume we want to predict a time series. To train, we take sequences of 40 data points from the training data and return the data shifted by 1, with a single prediction at the end.

## Example: Time Series





As an RNN then, we define a recurrence layer:

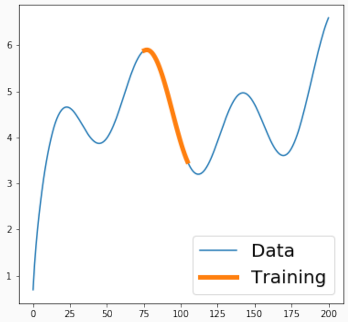
1 input node

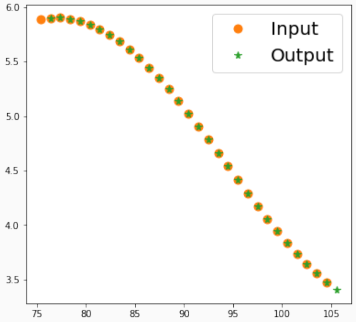
100 node dense layer

1 output node.

and unroll it over 40 (number of points in our training instance) steps.

## Example: Time Series





However, the advantage comes to the use time: Even though we have trained on sequences of 40 points, we can use the network to predict using any length of sequence. Of course, the mantra of machine learning is "Always train on the data you will be testing on." For this reason, RNN's can also be constructed to require input of a specific length.

## Recurrence Nodes

## LSTM Cell

The main challenge with RNN's is that training is highly susceptible to gradient explosion and vanishing. This is thought to be because recurrent nodes lead to highly nonlinear networks, which in tern lead to gradient volatility.

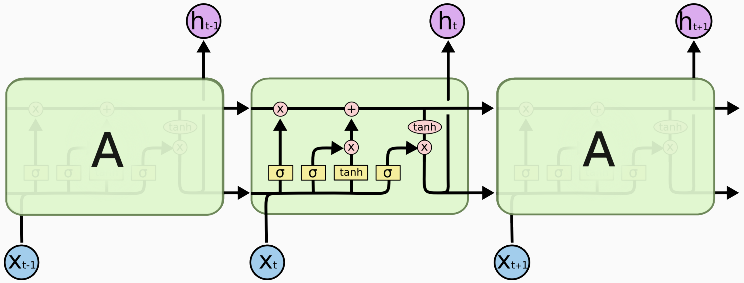
The other challenge is training long term memory vs short term memory. If our time series network only knows about the last 40 data points it might miss long terms moves in the data.

One solution is to build new recurrent nodes by replace the function

$$\mathbf{Y}_{(t)} = \sigma\left(\mathbf{W}_x^T \mathbf{X}_{(t)} + \mathbf{W}_y^T \mathbf{Y}_{(t-1)} + b\right)$$

with a new function tries to solve both these problems. We will finish this lecture by taking a quick look at two such example.

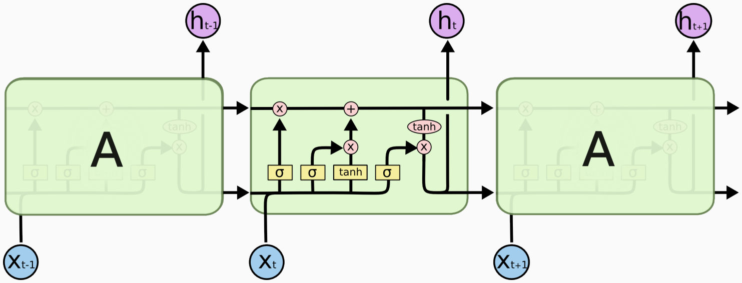
## LSTM Cell



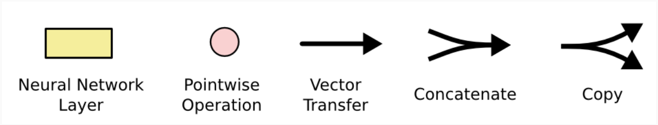
The first example is a **long short-term memory** (**LSTM**) cell. At its boundary, the LSTM cell looks exactly like the recurrence cell from before, except that it sends both its output $y_{(t)}$ and a state vector $C_{(t)}$ to the next training frame.

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

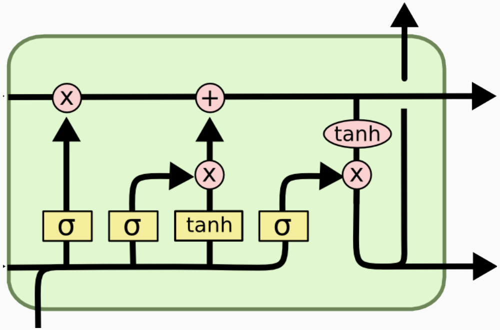
## LSTM Cell



In the diagram above, the product and sum are the component wise product and sum, so at the product node $[1, 2] \otimes [3, 4] = [3, 8]$.



## LSTM Cell

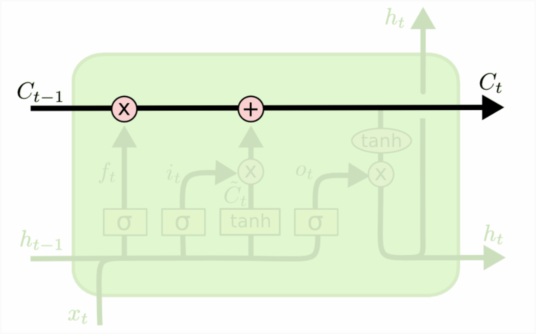


In the LSTM node, each box represents a different LTU with the given activation function:

$$\sigma(\mathbf{W}_x^T\mathbf{X}_t + \mathbf{W}_y^T\mathbf{h}_{t-1} + b)\,, \text{ or } \tanh(\mathbf{W}_x^T\mathbf{X}_t + \mathbf{W}_y^T\mathbf{h}_{t-1} + b)\,.$$

Here $\sigma$ is the sigmoid function, so the output is a vector where each element is between 0 and 1. This means that each $\sigma\otimes$ layer acts as a multiplicative mask of the connecting vector.

## LSTM Cell



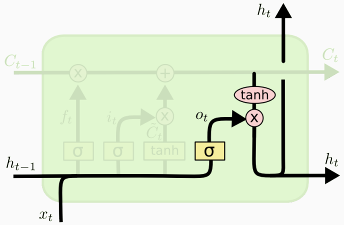
The long term state $C_t$ passes through the network only interacting three times:

The multiplication gate $\otimes$ “forgets” data from $C_{t-1}$ by making it with $\sigma(\mathbf{W}_x^T \mathbf{X}_t + \mathbf{W}_y^T \mathbf{h}_{t-1} + b)$.

It then adds a screened tanh combination of $x_t$ and $h_{t-1}$.

Finally, $C_t$ is composed with tanh and used to screen a sigmoid linear combination $x_t$ and $h_{t-1}$.

## LSTM Cell



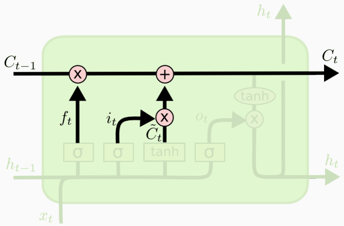
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

This may look complicated but it is a natural extension of the basic recurrent cell: Instead of returning

$$\sigma(\mathbf{W}_x^T \mathbf{X}_t + \mathbf{W}_y^T \mathbf{h}_{t-1} + b)$$

it returns a version screened with the long term memory $C_t$. The long term memory is a vector whose length is the same as the output.

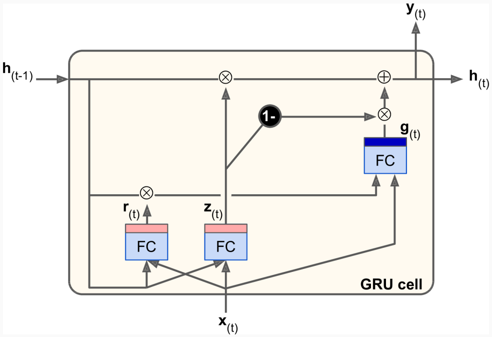
## LSTM Cell



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

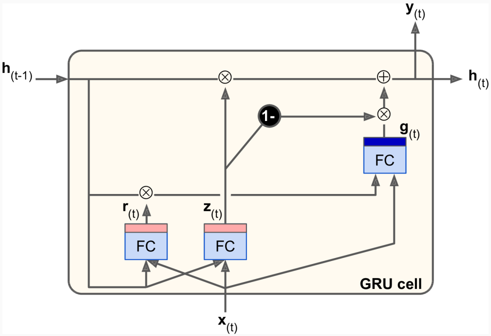
We only need to stipulate how to update the long term memory $C_t$. We allow the long term memory to "forget" by making at the first multiplication, and to then store new information in the memory, by adding on a masked (non-liner) term dependent on the input $x_t$, and the previous output $h_{t-1}$.

## GRU Cell



In 2014, a much simplified version of the LSTM called the Gated Recurrent Unit (GRU) was developed. Structurally, the main differences come from the combining of the output and state vector into a single $\mathbf{h}_{(t)}$ and the fact that the black 1 indicates that the forget gate is only activated where new memories will be stored. That is, you can only overwrite, not forget outright.

## GRU Cell



The accuracy of GRU cells test as comparable to LSTM cells on polyphonic music modeling and speech signal modeling, but are far more trainable relying on many fewer parameters.

However, the it was shown in 2018 that the LSTM is strictly stronger than the GRU and it has been demonstrated that the GRU cannot learn some simple language models easily learned by LSTMs.

## Applying RNN's to Natural Language Processing

## Natural Language Processing

Since RNN's are designed to process sequences they are used particularly in natural language processing. Consider a typical natural language processing machine learning problem:

Detecting fake reviews on a website.

Autocompleting commands and sentences.

Detecting stock market movements from headlines in 100 languages.

Generating images based on text.

Processing voice/text commands to their most likely intent.

In each case, a string of text is taken as input, processed, and some data is outputed. The output may be a class, a number, an image or another sequence.

## Encoding Text

The first step in any natural language processing project is to encode text into a form that is readable by a classifier. The first step in any natural language processing project is to encode text into a form that is readable by a classifier. There are many ways to do this in general, usually by summarizing information about the sentence such as

Frequency and number of words unordered.

Sentence character/word length.

Word adjacency graph.

Ratio of common to uncommon words.

Number of words from parts of language (Nouns, verbs, etc).

These are just to name a few. However, for RNN's we want to preserve both the word structure and sequence, so we will need to use much more direct encodings.

## Encoding Text

**Character Encoding:** The simplest way is to encode each segment of text as a categorical string of characters. Practically, we need to choose a vocabulary, for example a simple English encoding might be

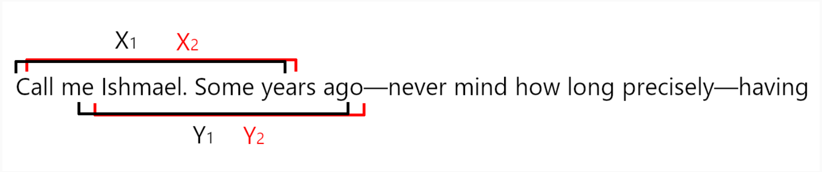
| | | | | |
|---|---|---|---|---|
| a = 1 | b=2 | c=3 | d=4 | ... |
| A = 27 | B=28 | C=29 | D=20 | ... |
| $\langle$space$\rangle$ = 53 | $\langle$padding$\rangle$=54 | ,=55 | .=56 | ... |

Then a sentence can be encoded as

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Text**: | T | h | e | | b | e | s | t | ... |
| **Encoding**: | 46 | 8 | 5 | 27 | 2 | 5 | 19 | 20 | .... |

Practically, we then one-hot-encode each of the encoded vectors. If our vocabulary has size $m$ and each sequence is padded until it have length $L$, the input $X$ is a sparse $L \times m$ matrix, where each row has exactly one nonzero element.

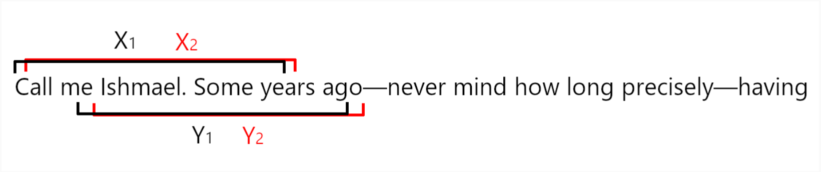
## Building a Text Generator


X1
X2
Call me Ishmael. Some years ago—never mind how long precisely—having
Y1
Y2

**Building a Text Generator:** We can build a simple RNN text generator with such an encoding. Taking a volume of text (say, Moby Dick), we fix a sequence length $L$ and generate training data $X$ by taking sequences of length $L$, possibly offest by some step $S$. The target values $Y_i$ are then the next characters in the text.

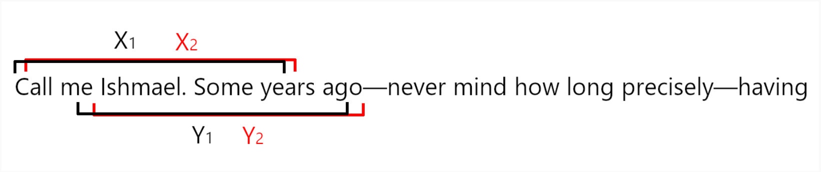
A good sized book can provide hundreds of thousands of training examples.

## Building a Text Generator


X1
X2
Call me Ishmael. Some years ago—never mind how long precisely—having
Y1
Y2

Training a sequence-to-sequence RNN and this data, we can then generate new data by inserting a seed sentence $X_{seed}$, generating a new sentence $G_1 =$RNN$(X_{seed})$, and then repeating $G_2 =$RNN$(G_1)$, etc. For example:

## Building a Text Generator


X1
X2
Call me Ishmael. Some years ago—never mind how long precisely—having
Y1
Y2

For example: For a network with 128 LSTM nodes trained for 15 epochs on sequences from Moby Dick of length 60, the seed $X_{seed}$ =" his leg were off at the hip, now, it would be a different t" generates

*his leg were off at the hip, now, it would be a different the bown was by the blackness and such the bleating make and little face the whole fleepan with a common flame in the descend of same hands and heards, and the ray his sea,*

## Word Level Encoding

**Word Encoding:** The RNN has done a shockingly good job learning the words from the character encoding, but in the simple case above that also seems to be almost all it is doing. An alternative is to encode whole words as numbers, forming a vocabulary of all of the words in the we may encounter. The word level vocabulary will be much larger the character level vocabulary, possibly including tens of thousands of words. On the other hand, we deal with images that size frequently, and indeed for a computer it can be tractable.

## Word Level Encoding

**Word Encoding:** Form a vocabulary, either from the data at hand or from a pretrained vocabulary:

| | | | | |
|---|---|---|---|---|
| "a" = 1 | "of"=2 | "to"=3 | "and"=4 | ... |
| "quotient" = 996 | "teeth"=997 | "shell"=998 | "neck"=999 | ... |

Then a sentence can be encoded as

| | | | | |
|---|---|---|---|---|
| **Text**: | Call | me | Ishmeal | ... |
| **Encoding**: | 456 | 34 | 888 | .... |

Again, we then one-hot-encode each of the encoded vectors. If our vocabulary has size $m$ and each sequence is padded until it has $L$ words, the input $X$ is a sparse $L \times m$ matrix, where each row has exactly one nonzero element.

## Word Level Encoding

**Word Encoding:** Form a vocabulary, either from the data at hand or from a pretrained vocabulary:

| | | | | |
|---|---|---|---|---|
| "a" = 1 | "of"=2 | "to"=3 | "and"=4 | ... |
| "quotient" = 996 | "teeth"=997 | "shell"=998 | "neck"=999 | ... |

Then a sentence can be encoded as

| | | | | |
|---|---|---|---|---|
| **Text**: | Call | me | Ishmeal | ... |
| **Encoding**: | 456 | 34 | 888 | .... |

Again, we then one-hot-encode each of the encoded vectors. If our vocabulary has size $m$ and each sequence is padded until it has $L$ words, the input $X$ is a sparse $L \times m$ matrix, where each row has exactly one nonzero element.

## Word Level Encoding

We can train a word level text generator just as we did for the character level. After one epoch, a network with 128 LSTM nodes trained on sequences of length 20 words with the seed

*”so fast and furious . it is furnished with a small rope called a warp , of considerable length , by which it can be hauled back to the hand”*

generates

*, and the whale . the whale , and the whale , and the whale . the whale whale . “ “ , he , he was , and he was the whale . the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale , and the whale . “ i , i have not to the whale . ” “ i , ” “ “ , ”*

## Word Level Encoding

However, after 10 epochs, the seed

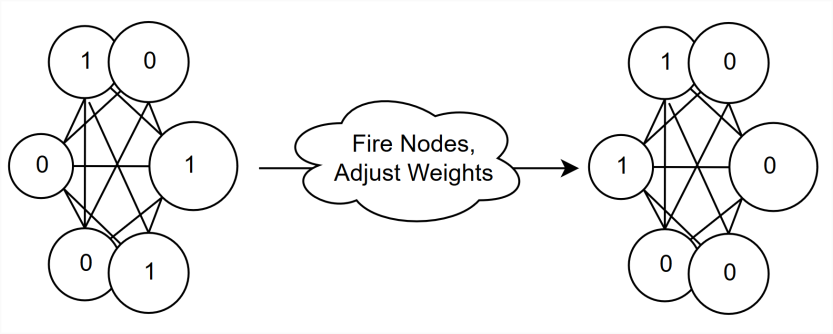
*“ now , three to three , ye stand . sunset . the cabin ; by the stern windows ; ahab sitting alone , and gazing out . ’tis iron”*

generates

*that i know not next to the two boats were , in the mast heads . but this had not the same thing . it was a little part of the cabin . it was the first man in the best ; and so it was by this own hunted . by man this is the least . the sperm whale . it is the first man in the pequod . “ queequeg , ” said i . “ no , no ; we must go down this way . ” “ no , no , no ; no*

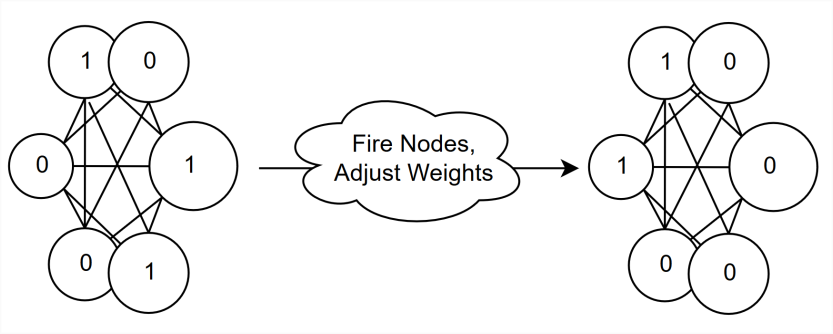
## Extra: Symmetrically Connected Networks

## Symmetrically Connected Networks



Symmetrically connected networks hew most closely to biological neurons. They are dense graphs (each node connects to each other) with binary nodes. Each node has a threshold $\theta$ set by training.

## Symmetrically Connected Networks

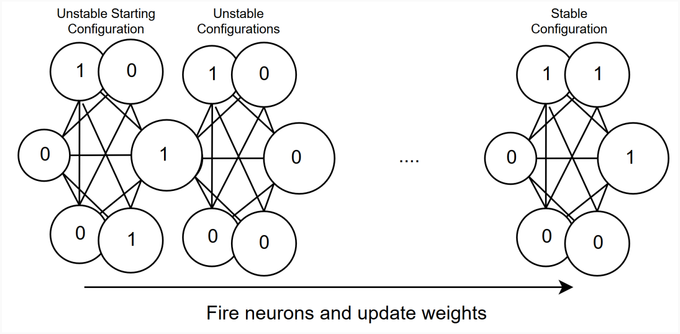


Given an initial assignment of 1's and 0's to each node, the network reconfigures itself using Hebbs rule:

For each node $i$, with value $n_i$ and threshold $\theta_i$, set

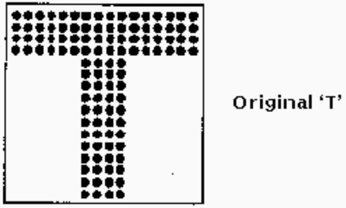
$$n_i = \begin{cases} 1 & \theta_i \leq \sum_j w_{ij} n_j \,, \\ 0 & \text{otherwise} \,. \end{cases}$$

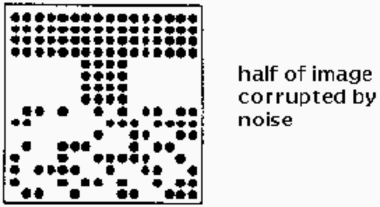
## Symmetrically Connected Networks



After many such steps the network will fall into a stable configuration. This configuration is the final result. Practically, this means that given an initial configuration, the network "finds" a nearby final configuration.

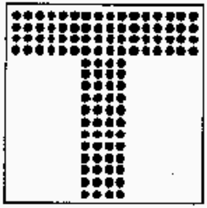
## Symmetrically Connected Networks



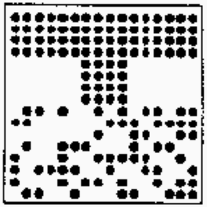


After many such steps the network will fall into a stable configuration. This configuration is the final result. Practically, this means that given an initial configuration, the network "finds" a nearby final configuration. For example, feeding a noisy picture in leads to denoising above.

## Symmetrically Connected Networks



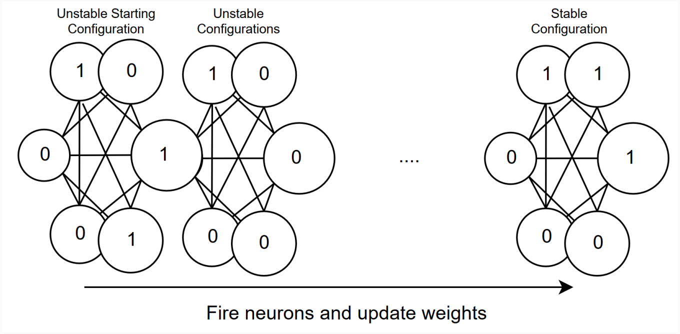
Original ‘T’



half of image corrupted by noise

There is a nontrivial connection between such networks and Hamiltonian mechanics that ensures that a lowest energy configuration can always be found.

## Symmetrically Connected Networks



The ]**Hopfeild network** above is a deterministic SCNs. If we allow the update to be stochastic we have a **Boltzmann Machine.**
Boltzmann Machines important to the study of biological neural networks, and are the first networks that can store a "representation" of a solution within the network structure. Many early AIs were modified Boltzmann machines.

## References

Additional References:

Imagenet visualization https://ai.stanford.edu/~ang/papers/icml09-ConvolutionalDeepBeliefNetworks.pdf

Neural network zoo:
https://www.asimovinstitute.org/author/fjodorvanveen/

Stanford CS 231 CNN's for Visual Recognition:
http://cs231n.github.io/

LeNet5: Gradient Based Learning Applied to Document Recognition
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

## References

Additional References:

AlexNet: ImageNet Classification with Deep Convolutional Neural Networks

`https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-netw pdf`

LSTM's are stronger than GRU's: `https://arxiv.org/abs/1805.04908`

## Images

GRU and LSTM images courtesy of O'Reilly media. Other images taken from

https://www.kdnuggets.com/2018/02/
8-neural-network-architectures-machine-learning-researchers-nee
html

https://blog.openai.com/generative-models/

https://techblog.gumgum.com/articles/
deep-learning-for-natural-language-processing-part-2-rnns

Autoencoder
https://www.edureka.co/blog/autoencoders-tutorial/

GAN
https://skymind.ai/wiki/generative-adversarial-network-gan