Machine Learning I

Lecture 12: Modern Neural Network Architecture

Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

Types of Artificial Neural Networks

Genre of Artificial Neural Networks

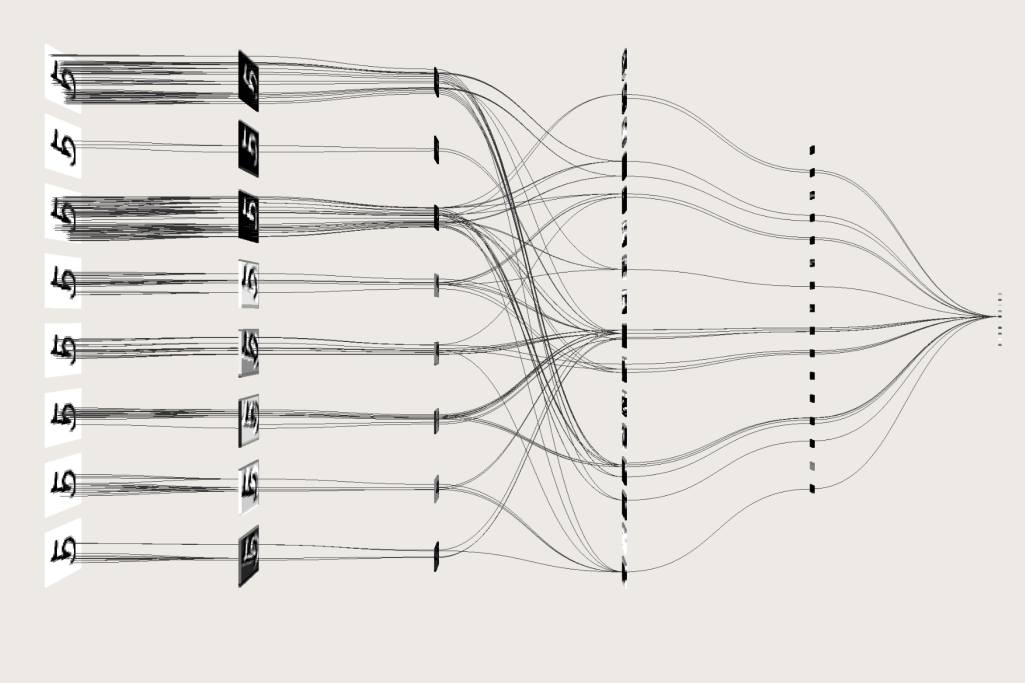
Modern artificial neural networks can be sorted into three broad categories based on their structure:

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.

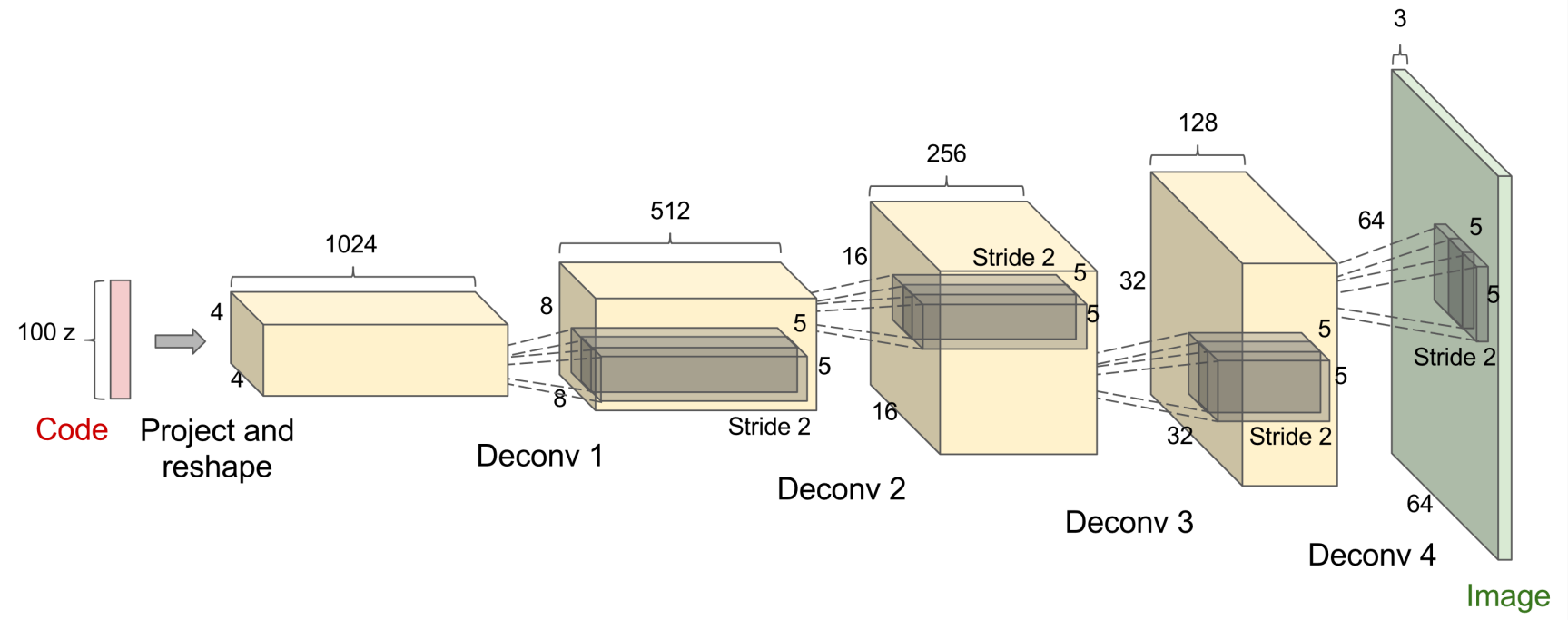
Feed Forward Networks



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 β. They could result in a classifier $\hat{y} = f_\beta(X)$, or a loss function $\ell(X, z) = f_\beta(X)$.

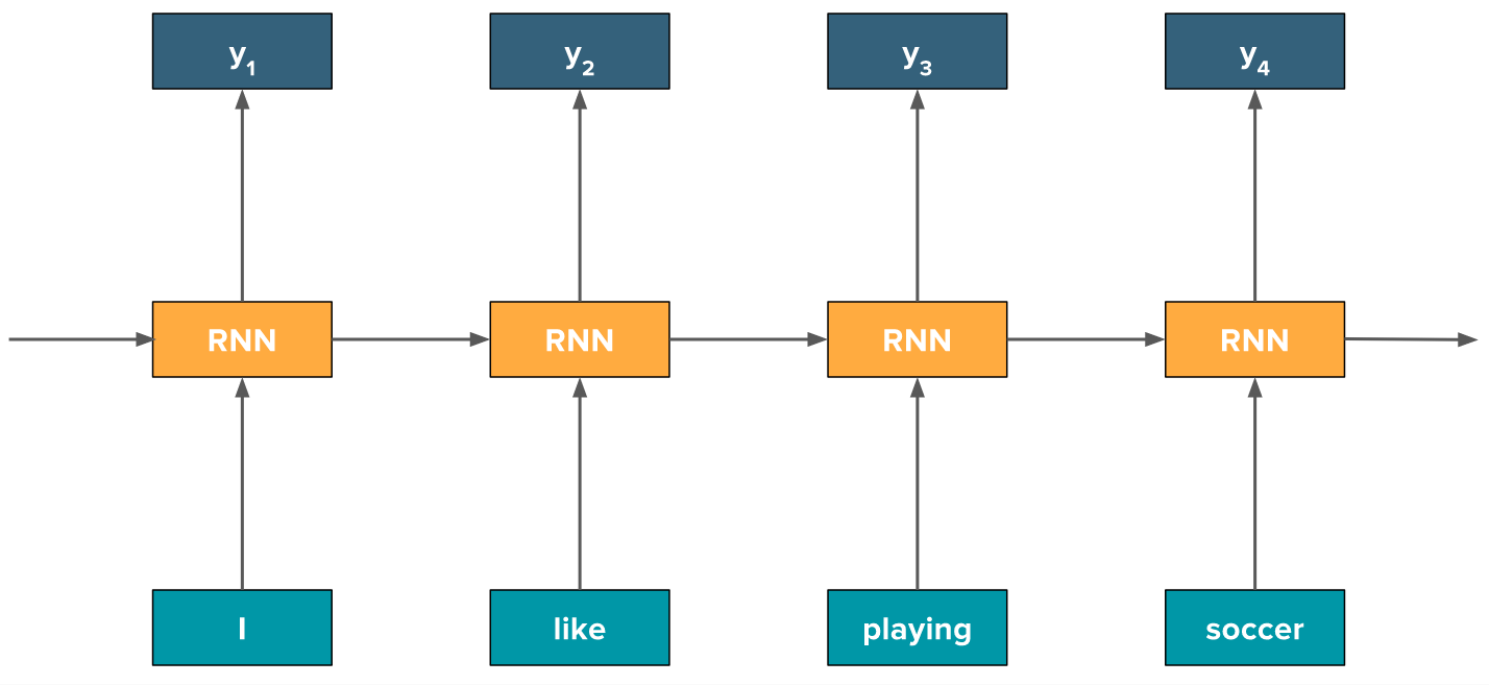
Feed Forward Networks



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 β are **fixed** for all prediction.

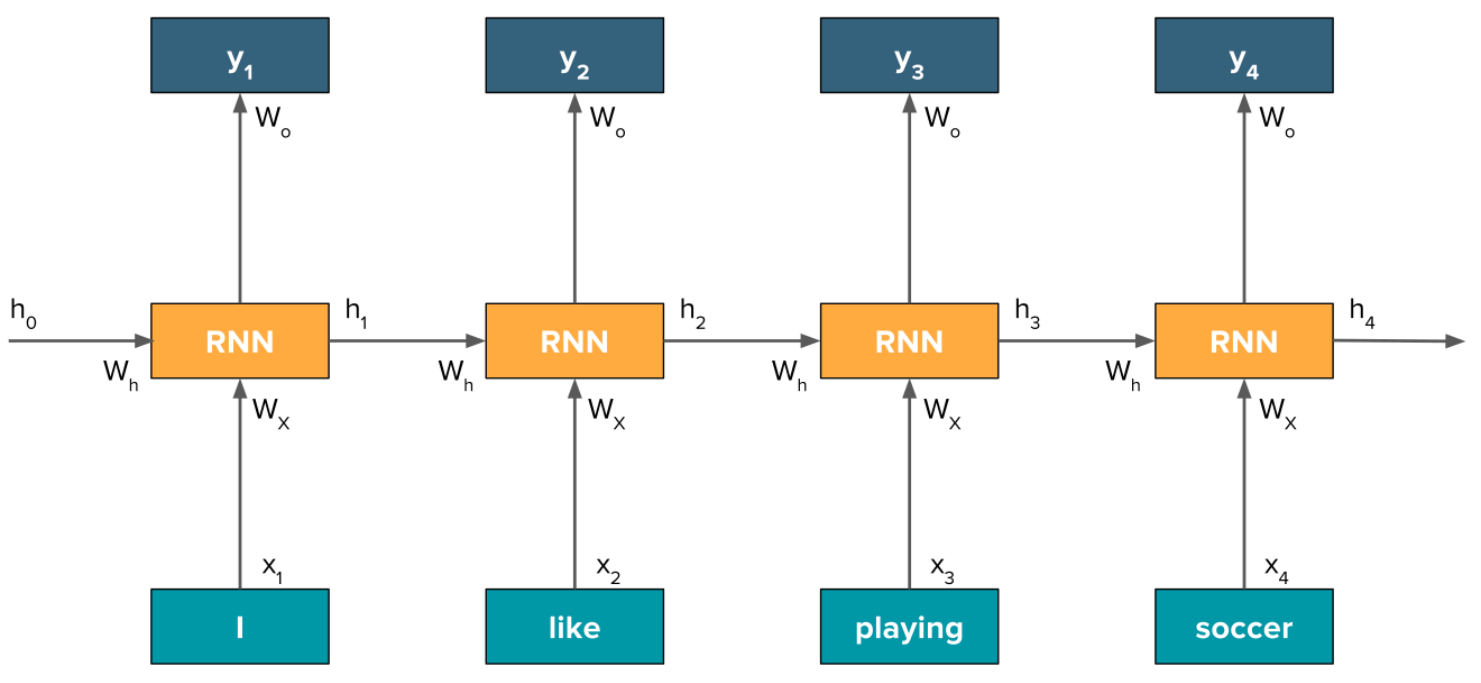
Recurrent Networks



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.

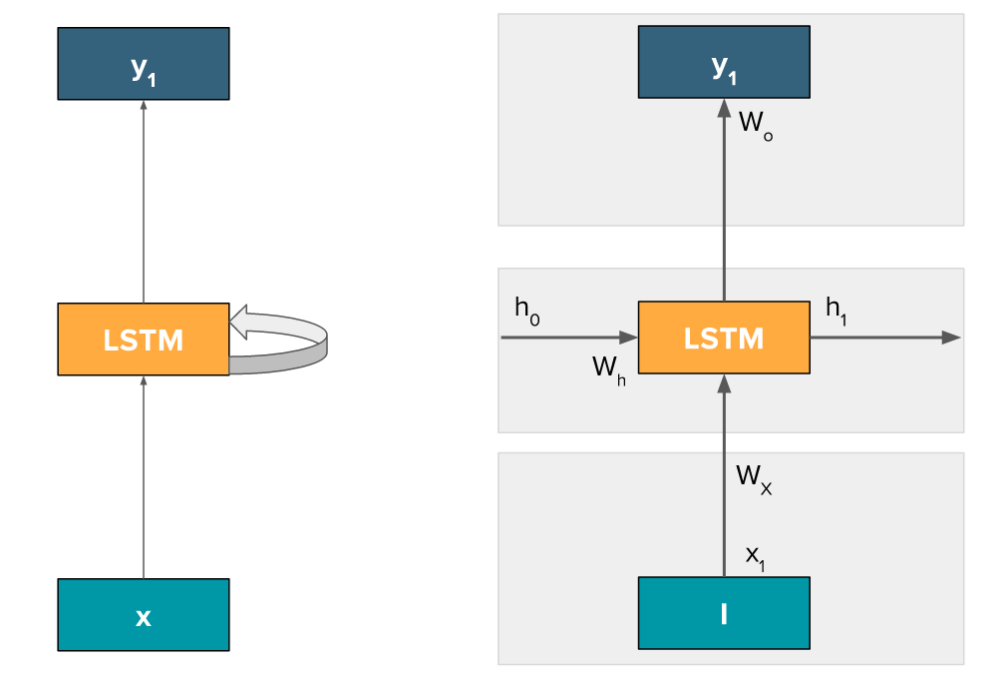
Recurrent Networks



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

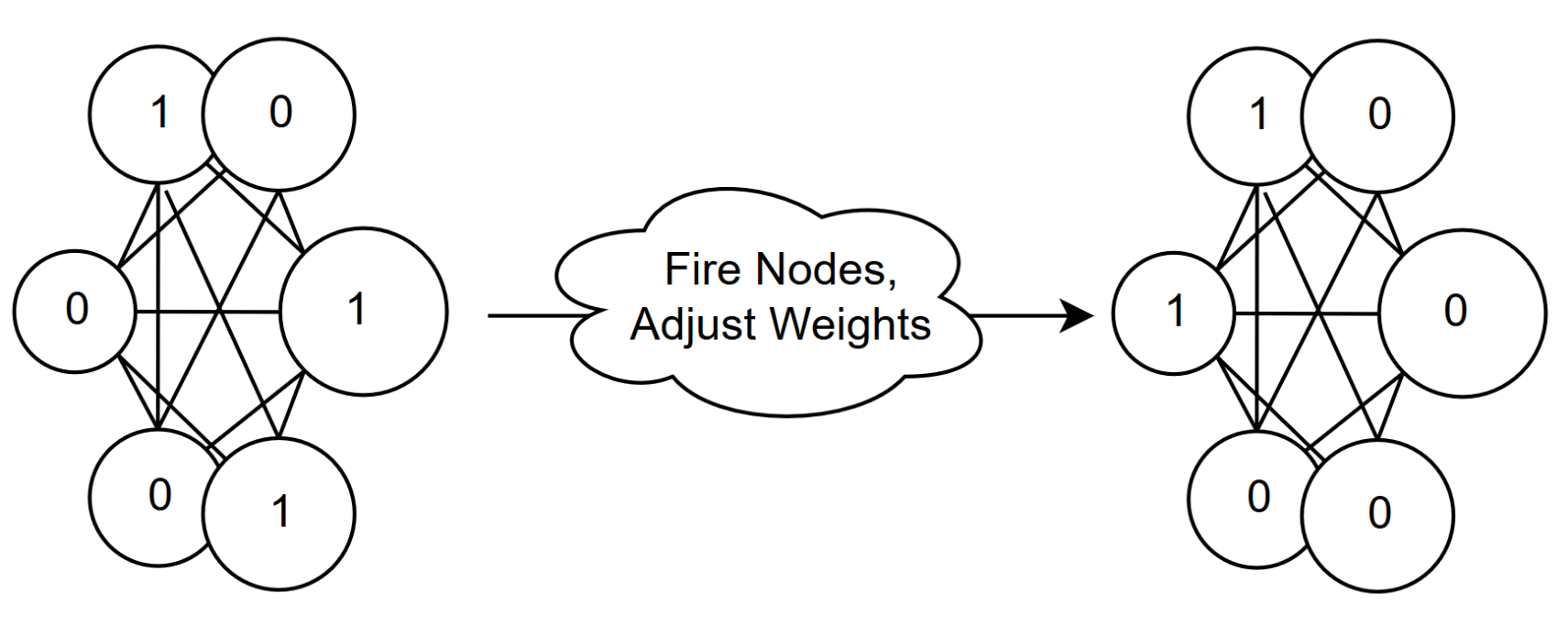
For example, in text prediction an RNN may predict one word at a time while "remembering" its previous predictions.

Recurrent Networks



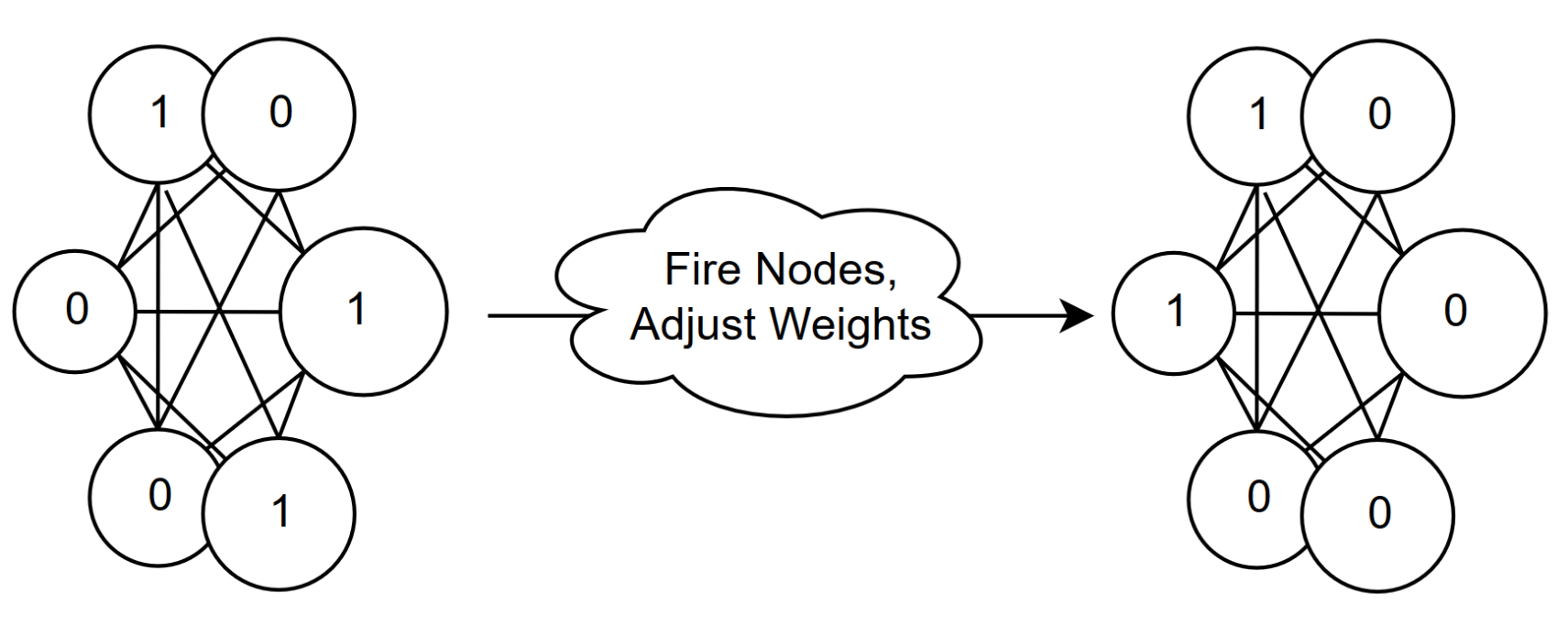
A recurrent node is often represented in "wrapped" form as above. RNN's are used extensively in time series prediction and natural language processing, although there success is still under scrutiny.

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 θ set by training.

Symmetrically Connected Networks

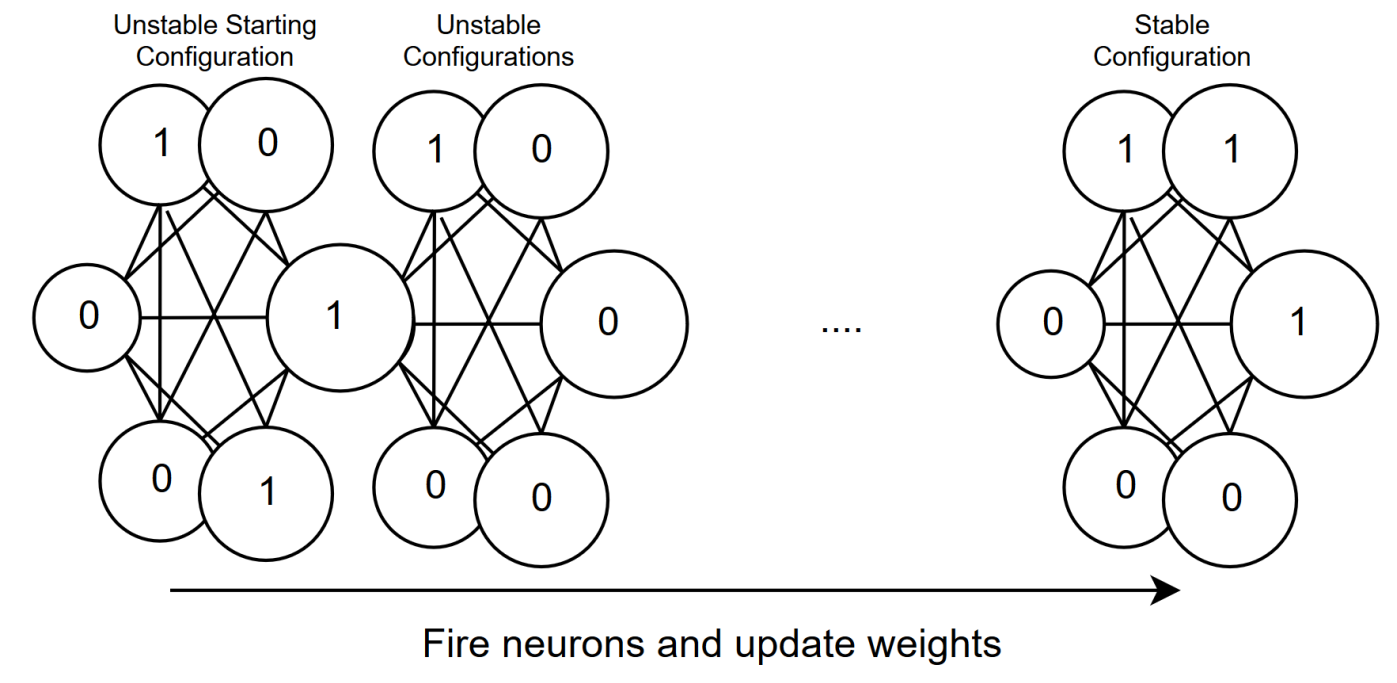


Given an initial assignment of 1's and 0's to each node, the network reconfigures itself using Hebbs rule:

For each node i, with value n_i and threshold θ_i, set

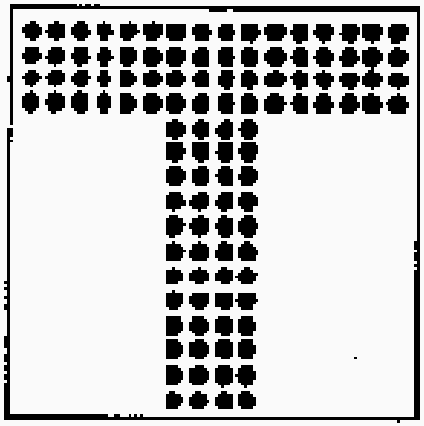
$$n_i = \begin{cases} 1 & \theta_i \leq \sum_j w_{ij} n_j \,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Symmetrically Connected Networks

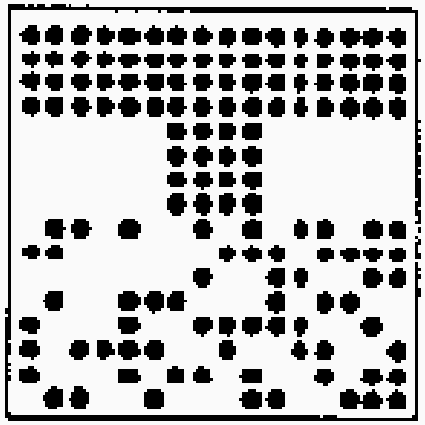


After many such steps the network will fall into a stable configuration. This configuration is the final result. Practically, this means that given an initial configuration, the network "finds" a nearby final configuration.

Symmetrically Connected Networks



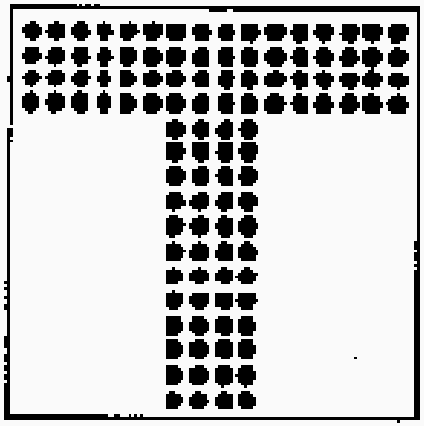
Original 'T'



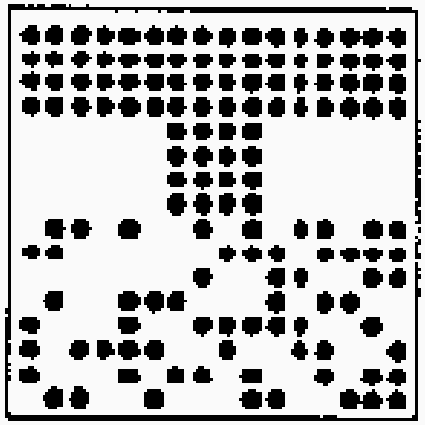
half of image corrupted by noise

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



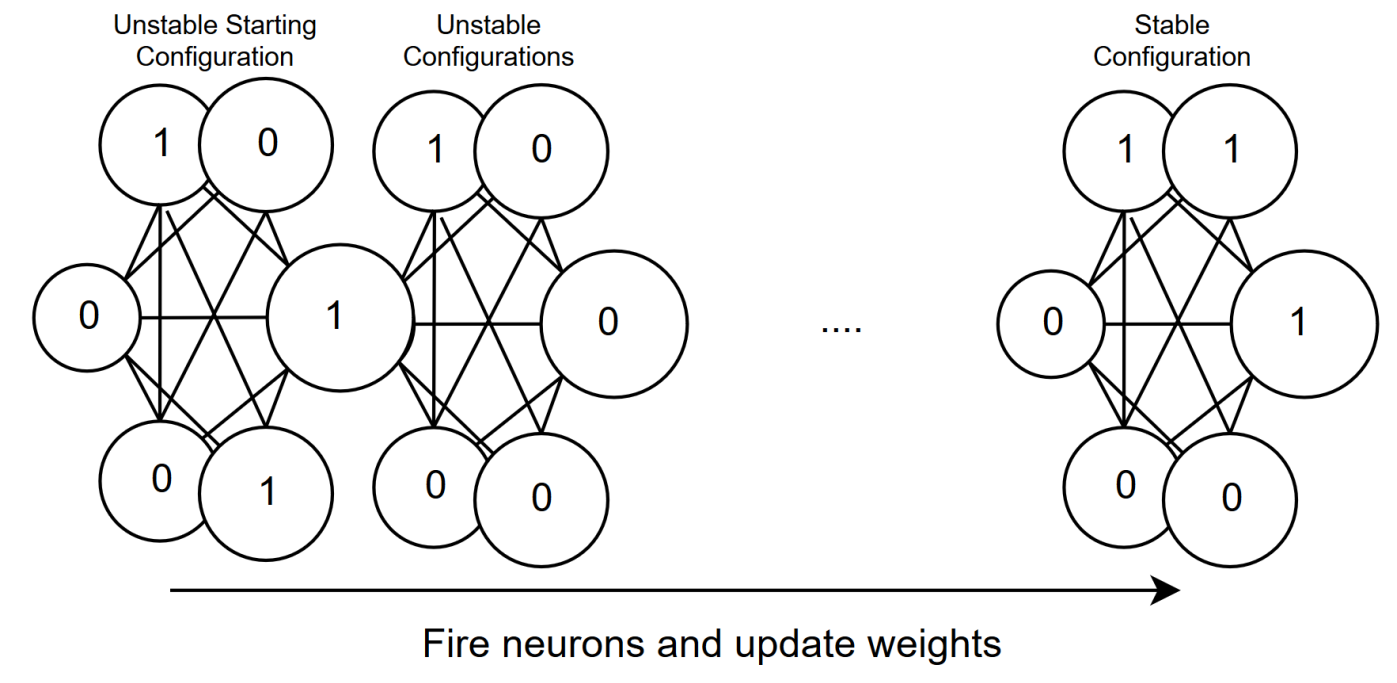
Original ‘T’



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.

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.

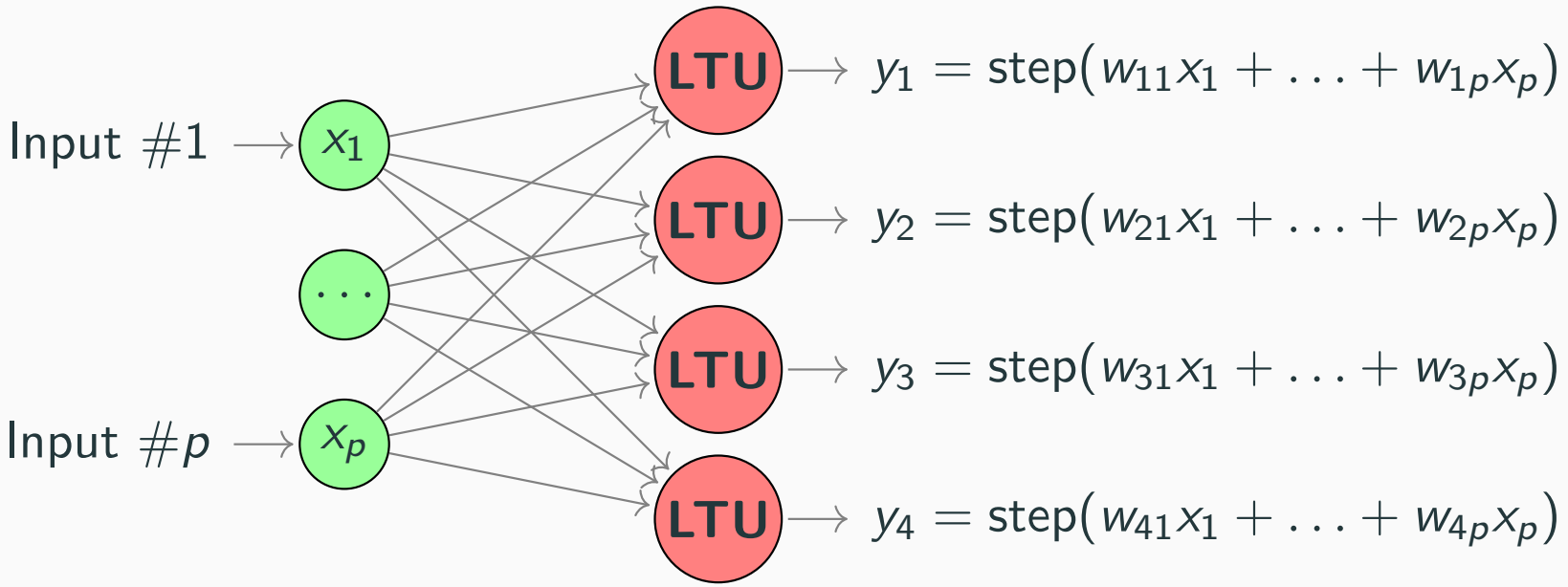
In the third part of the lecture, we will discuss **long term short term** (**LSTM**) networks as examples of recurrent networks. Both of these architectures are implemented in Tensorflow and Keras, and are still not completely understood mathematically.

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

Convolutional Neural Networks

Perceptron

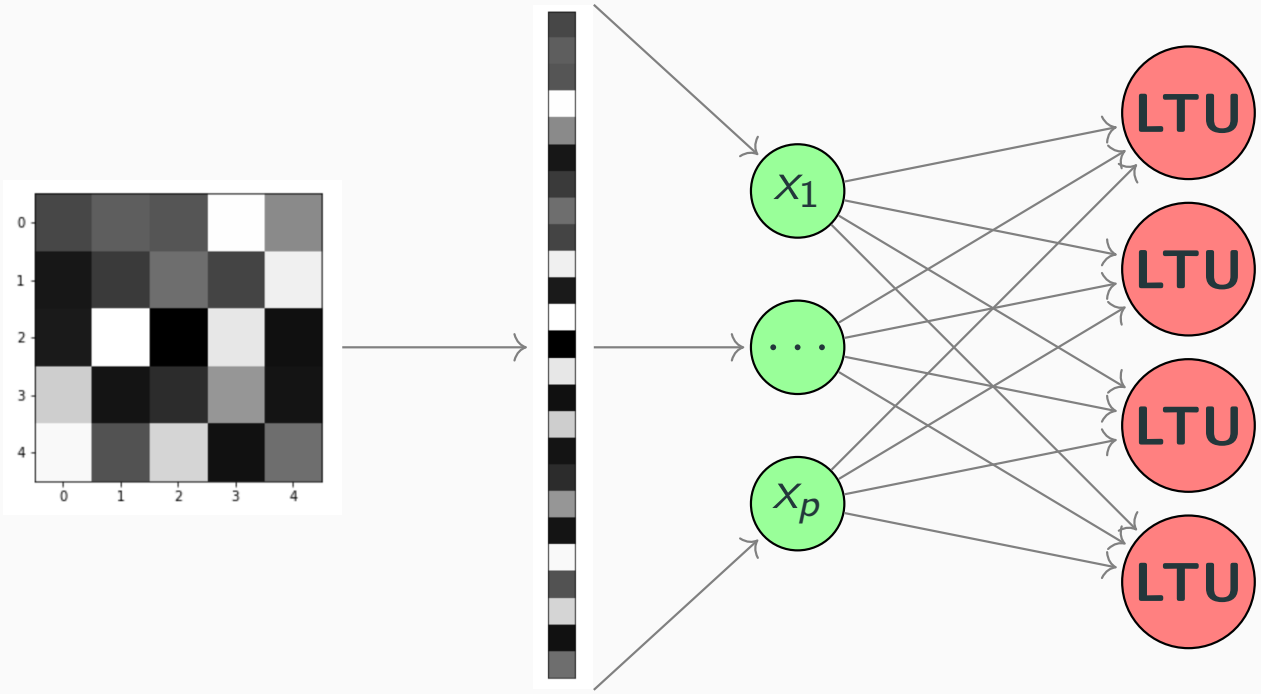
Multilabel Perceptron



Recall that the multilayer perceptron treats all of it's input variables as being on exactly equal footing. During training, the weights will be reconfigured to preference certain combinations of inputs over others. Any structure between the input features must be discovered by the network.

Perceptron

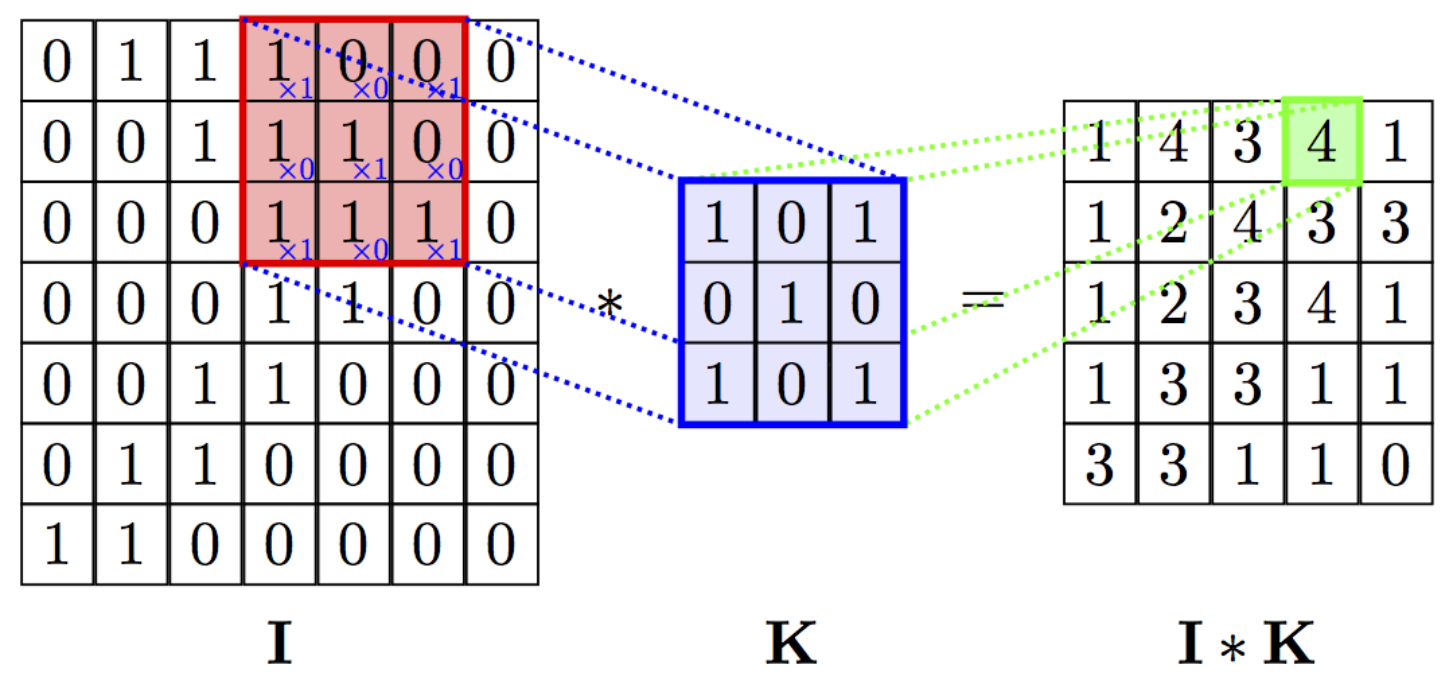
Multilabel Perceptron



For example, when processing a picture the first step is to unspool the 2d image into a 1d vector. This forgets about the spacial nature of the training data.

Is there a way we can capture the spacial nature of the data in the structure network? Can we train a network to search for certain spacial features and then record which features it depends on?

Convolutions

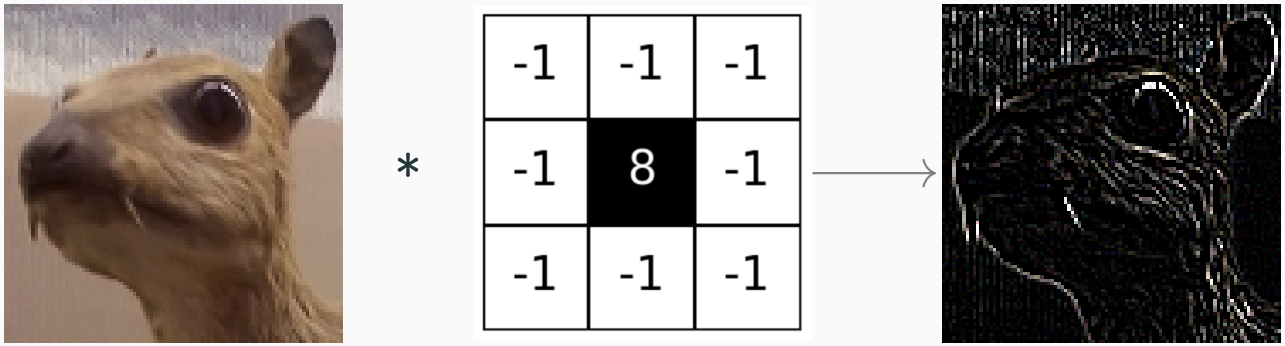


One answer is to use **convolutions**. A convolution combines two matrices to make a third, by taking the dot product of the smaller matrix with every block of the larger. Formally, for a $m \times n$ matrix K and a $(i+m) \times (j+n)$ matrix K, the convolution matrix is

$$(I * K)_{i,j} = \sum_{s=0}^{m-1} \sum_{r=0}^{n-1} I_{i+s,j+r} K_{s,r}$$

Convolutions

Edge Detection

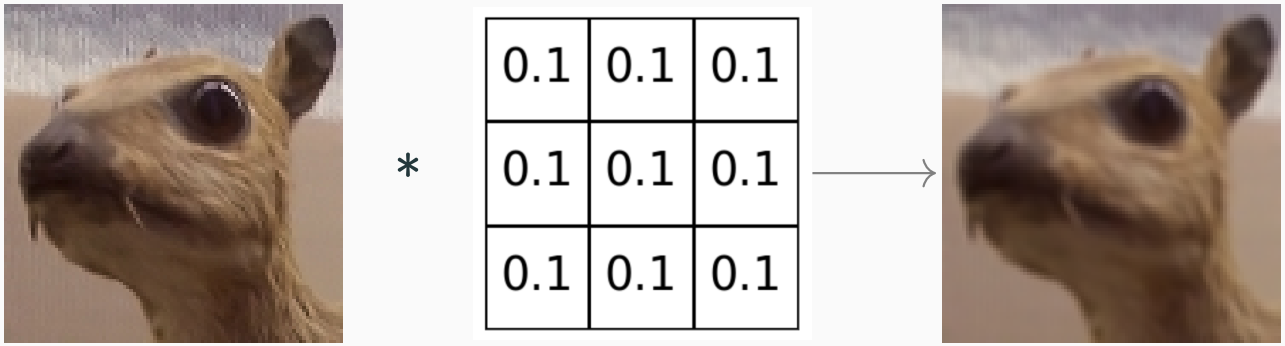


In image processing, this is used to create effects and extract information from images, including

- Finding edges

Convolutions

Edge Detection

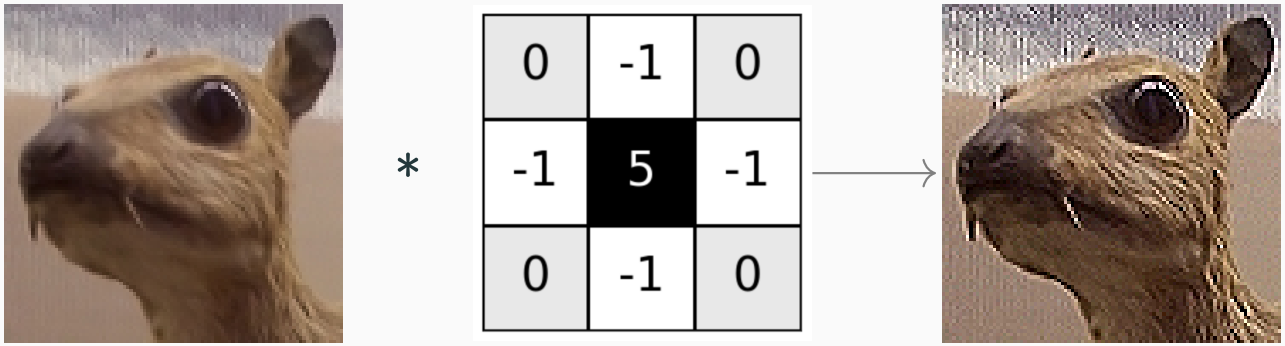


In image processing, this is used to create effects and extract information from images, including

- Finding edges
- Blurring

Convolutions

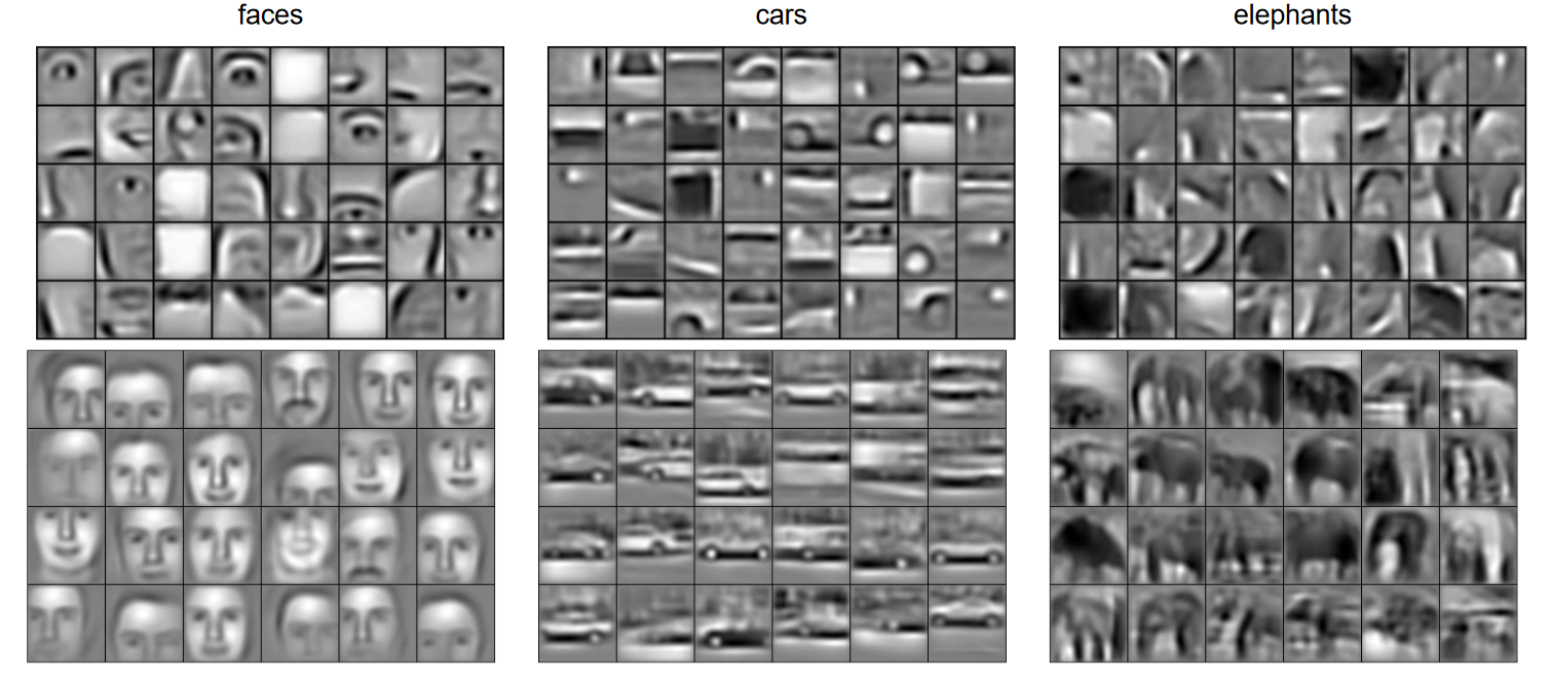
Edge Detection



In image processing, this is used to create effects and extract information from images, including

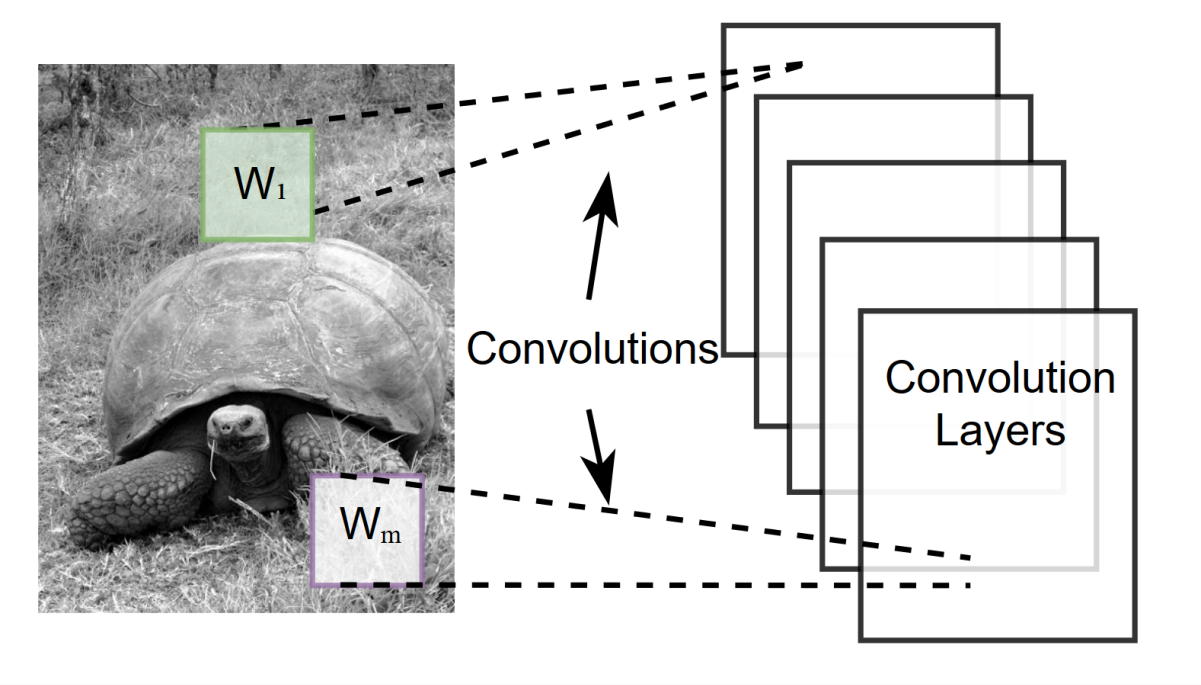
- Finding edges
- Blurring
- Sharpening

Convolution In Neural Networks



In the world of **convolutional neural networks**, the convolution matrices play a slightly different role, giving patterns that detect features emblematic of the classification.

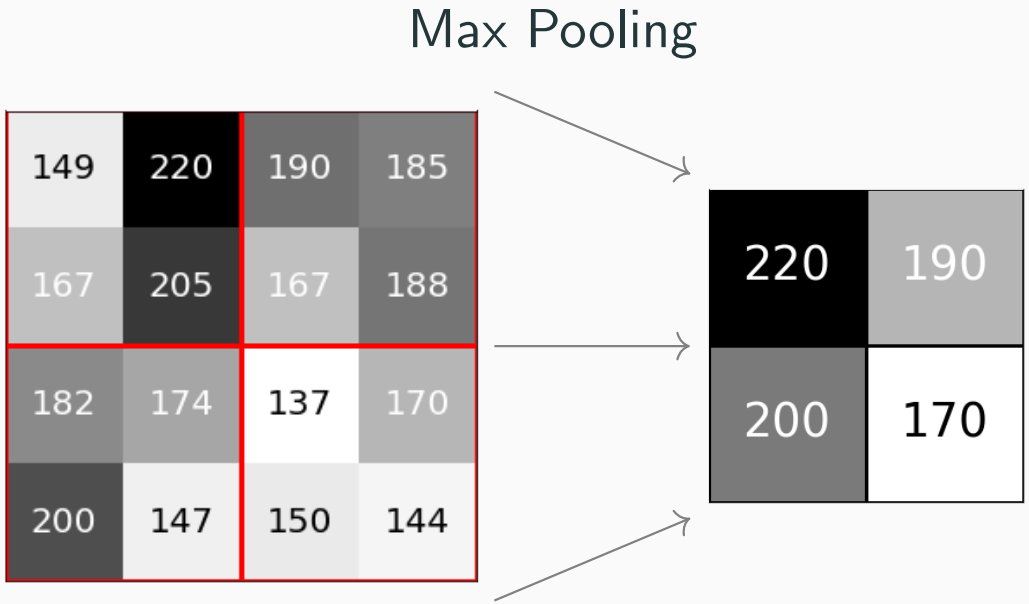
Convolution In Neural Networks



Of course, we are interested in many features so at each convolution step in operation tree we perform m convolutions with weight matrices W_i. For each convolution we fit a matrix of weights

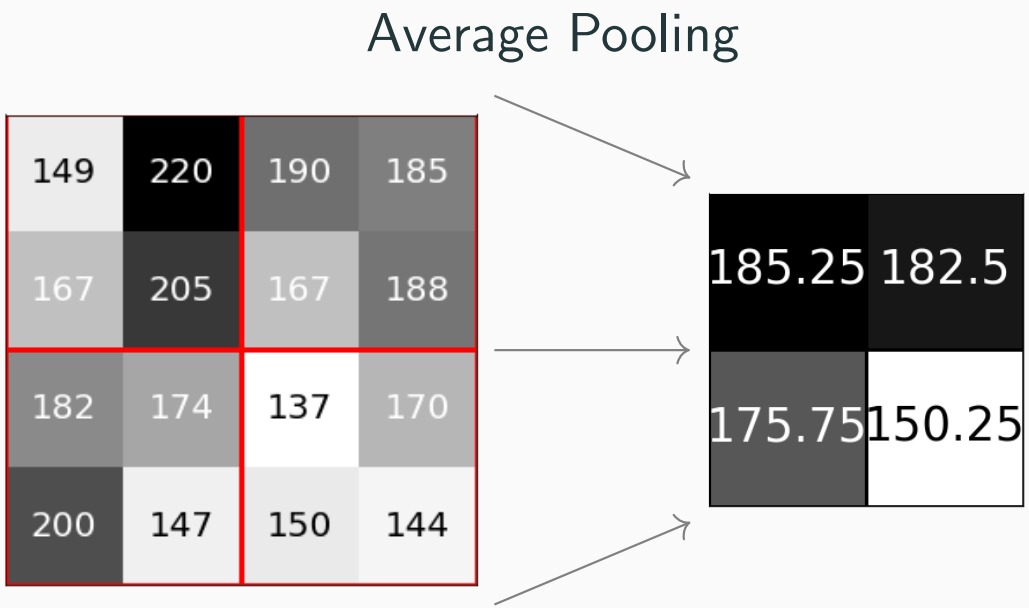
$$W_i = \begin{pmatrix} w_{11}^{(i)} & \dots & w_{1k}^{(i)} \\ \vdots & \ddots & \vdots \\ w_{j1}^{(i)} & \dots & w_{jk}^{(i)} \end{pmatrix}$$

Pooling Layers



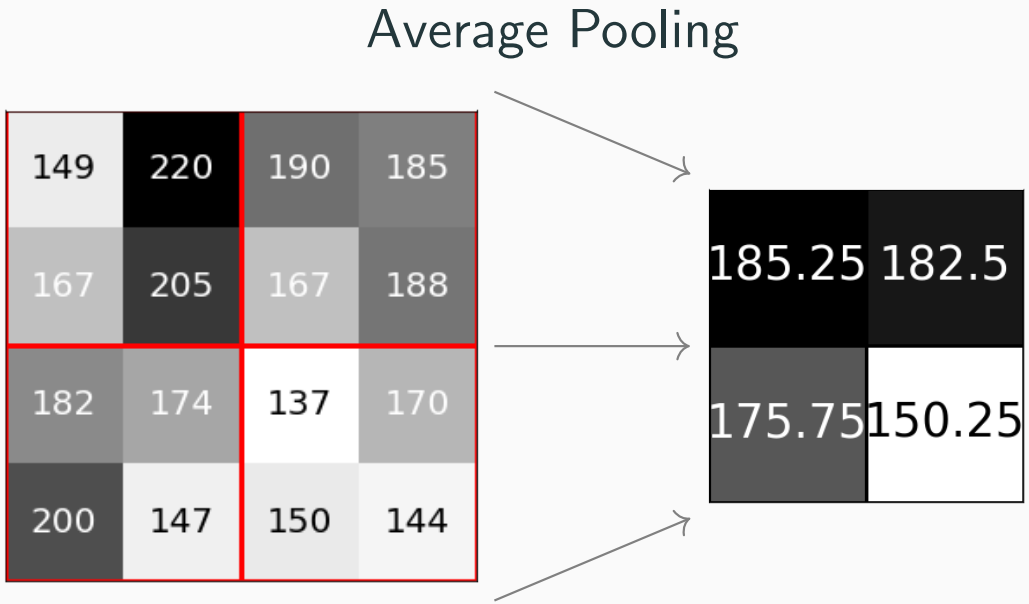
Successive convolution layers will find higher order features (collections of lower order features) but to group them efficiently it's common to start including pooling layers. A pooling layer down samples an image by taking the max, average, or min of a $n \times m$ set of pixels.

Pooling Layers



Successive convolution layers will find higher order features (collections of lower order features) but to group them efficiently it's common to start including pooling layers. A pooling layer down samples an image by taking the max, average, or min of a $n \times m$ set of pixels.

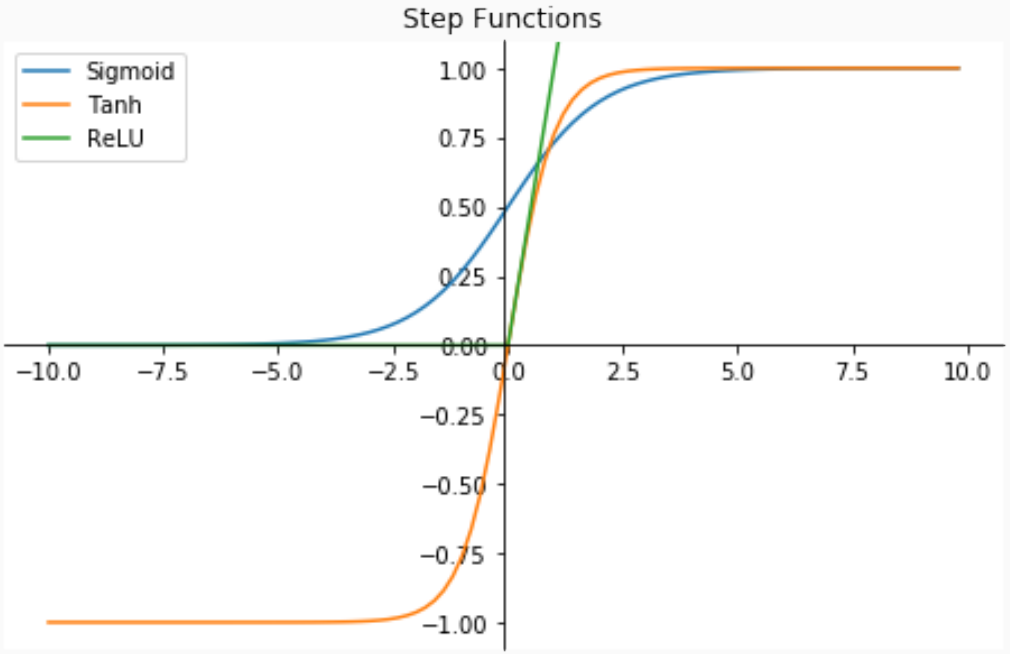
Pooling Layers



As a heuristic, max (min) pooling checks if a low level feature is present and reports "yes" or "no" for each region. Average pooling checks to what degree a low level feature is matched.

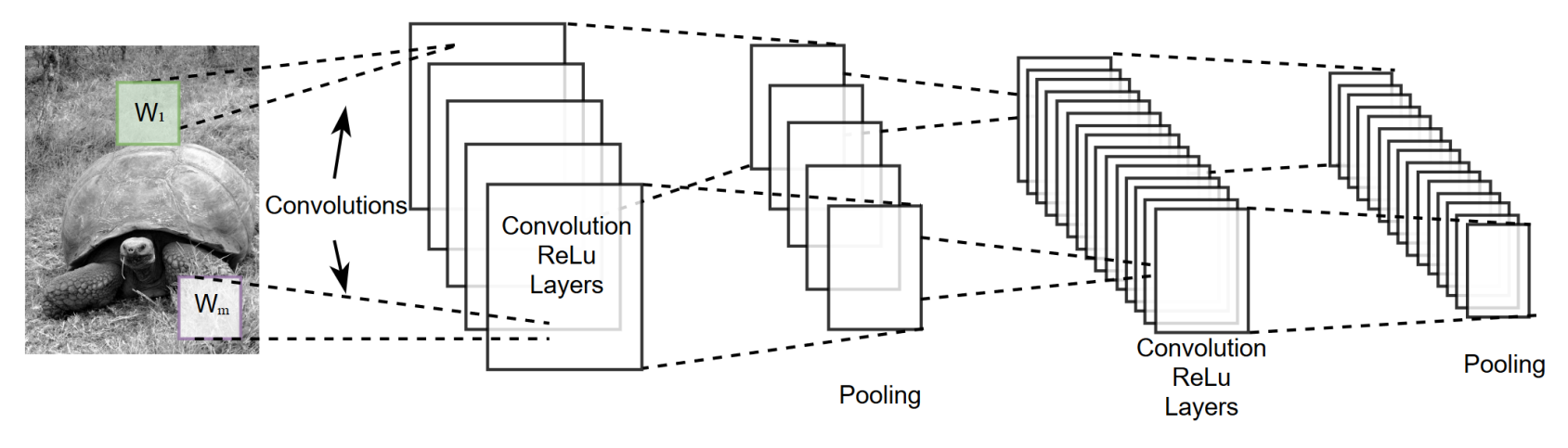
Note that all pooling layers are just convolution layers with a larger **stride.**

Activation Function



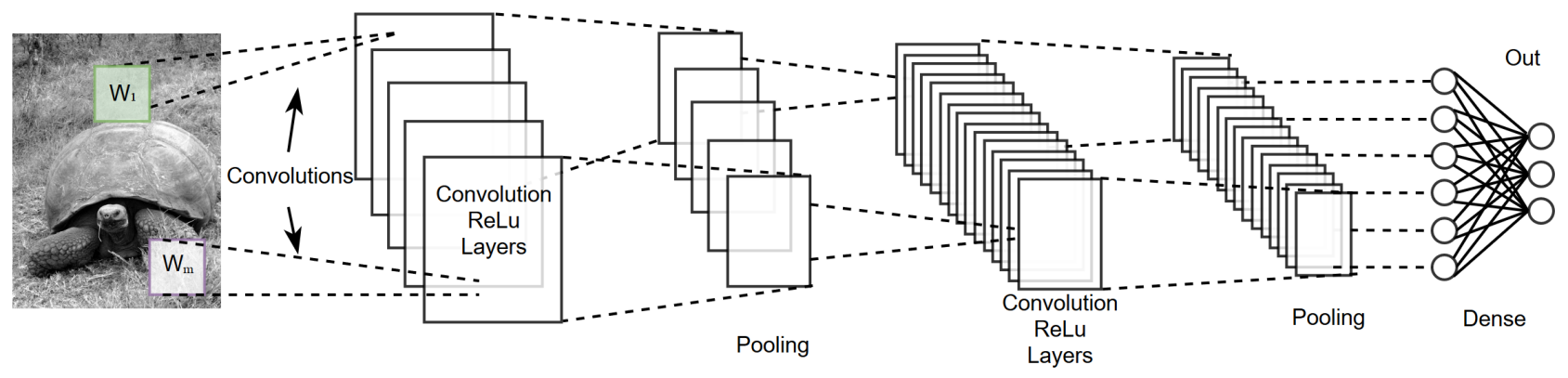
Finally, it is typical to add a ReLu layer after the convolution layer. The ReLu layer has been found to greatly increase training speed without adding the computational cost more complicated activation functions. Of course, ReLu nodes can irreversibly die during training if the weight is knocked far enough into the negative that it cannot recover.

Convolutional Networks



Putting it all together, a typical CNN is s series of convolutions/ReLu layers followed by max pooling stacked on top of each other.

Convolutions Networks

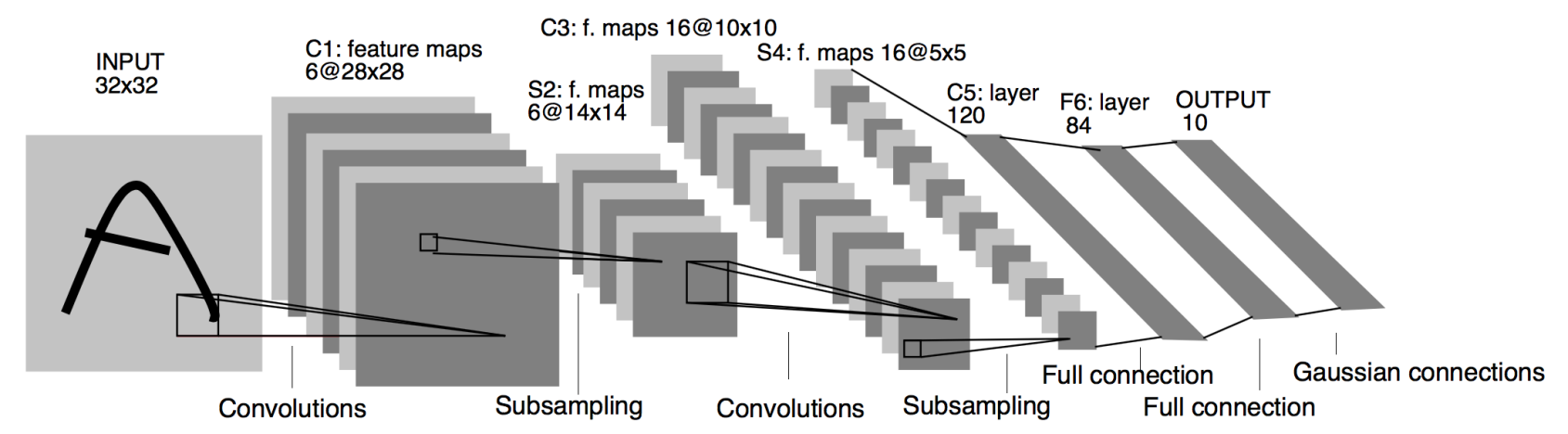


Finally, we flatten the result and feed it into a dense layer for classification. Convolutional networks have enjoyed a lot of attention over the last decade for good reason. They are relatively easy to construct once you understand the layers, but have consistently out performed other models in image classification tasks.

`http://scs.ryerson.ca/~aharley/vis/conv/`

History of CNNs

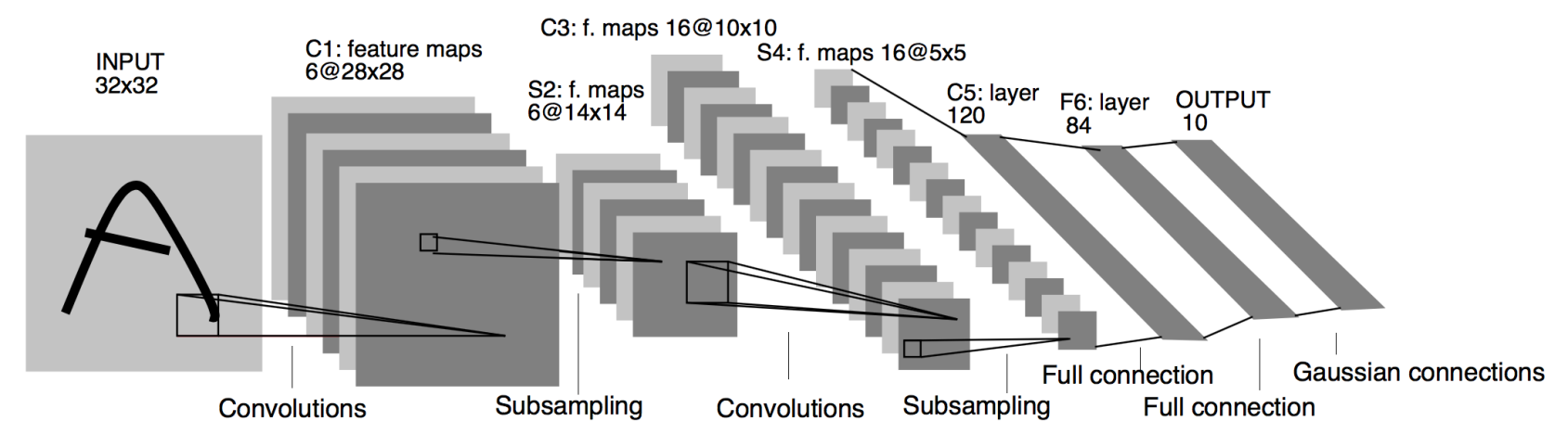
LeNet5



The most widely known CNN, LeNet5, was created by Yann LeCun in 1998 and tested on the MNIST dataset. The 28×28 MNIST data is padded with 0s to make each image 32×32.

Yann's home page describes LeNet5 with several demonstrations `http://yann.lecun.com/exdb/lenet/index.html`.

LeNet5



Layer	Type	Size	Kernel	Stride	Activation	Maps
In	Input	32x32				1
C1	Conv	28x28	5x5	1	tanh	6
S2	Ave	14x14	2x2	2	tanh	6
C3	Conv	10x10	5x5	1	tanh	16
S4	Ave	5x5	2x2	2	tanh	16
C5	Conv	1x1	5x5	1	tanh	120
F6	Dense	84			tanh	
Out	Dense	10				

LeNet5

LeNet5 performed a robust classification of the MNIST dataset. It now serves as the benchmark by which all other CNN architectures are compared.

Layer	Type	Size	Kernel	Stride	Activation	Maps
In	Input	32x32				1
C1	Conv	28x28	5x5	1	tanh	6
S2	Ave	14x14	2x2	2	tanh	6
C3	Conv	10x10	5x5	1	tanh	16
S4	Ave	5x5	2x2	2	tanh	16
C5	Conv	1x1	5x5	1	tanh	120
F6	Dense	84			tanh	
Out	Dense	10				

AlexNet

Synset: floor, level, storey, story has bounding box
Definition: a structure consisting of a room or set of rooms at a single position along a vertica
the office on?".

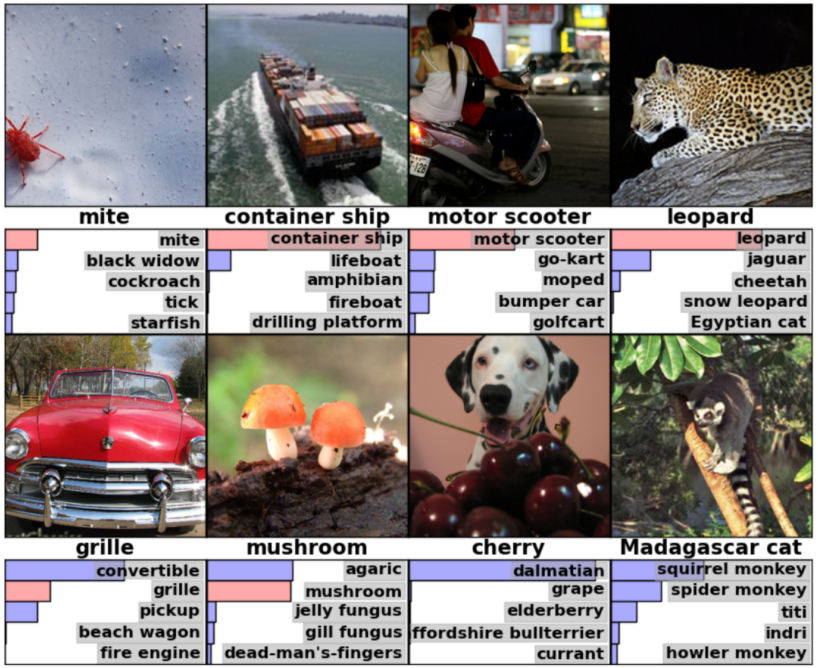
Synset: paper has bounding box
Definition: a material made of cellulose pulp derived mainly from wood or rags or certain gras

Synset: dwelling, home, domicile, abode, habitation, dwelling house has bounding box
Definition: housing that someone is living in; "he built a modest dwelling near the pond"; "they
provide homes for the homeless".

In the mid 2000's, ImageNet was started as a project of Fei-Fei Li as Stanford and comprises 14 million hand annotated images for algorithms to train and test again. Since 2010, the ImageNet project has run a benchmark test for computer vision.

In 2012, AlexNet became the first CNN to win, with 17% top 5 error rate while the second place winner had 26%.

AlexNet



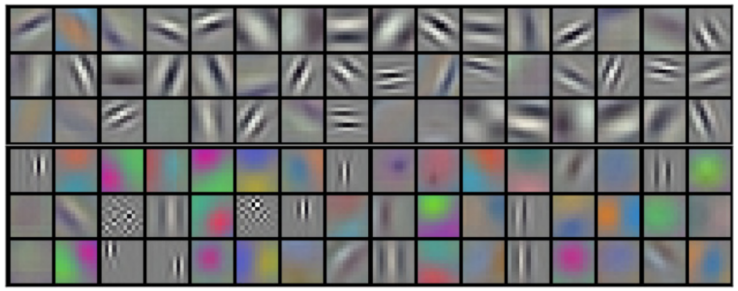


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

AlexNet was much deeper than LeNet5, with two fully connected layers of 4,096 nodes. It also used ReLu as opposed to tanh for activation functions.

Finally, to increase training AlexNet included **dropout layers**, which turn off neurons at random forcing the graph to learn the same concept in a redundant way.

GoogLeNet and ResNet

Over the next few years, CNN drove the top 5 error rate for image net below 4% with major gains trending to come from finding ways to deepen the network in computationally efficient ways.

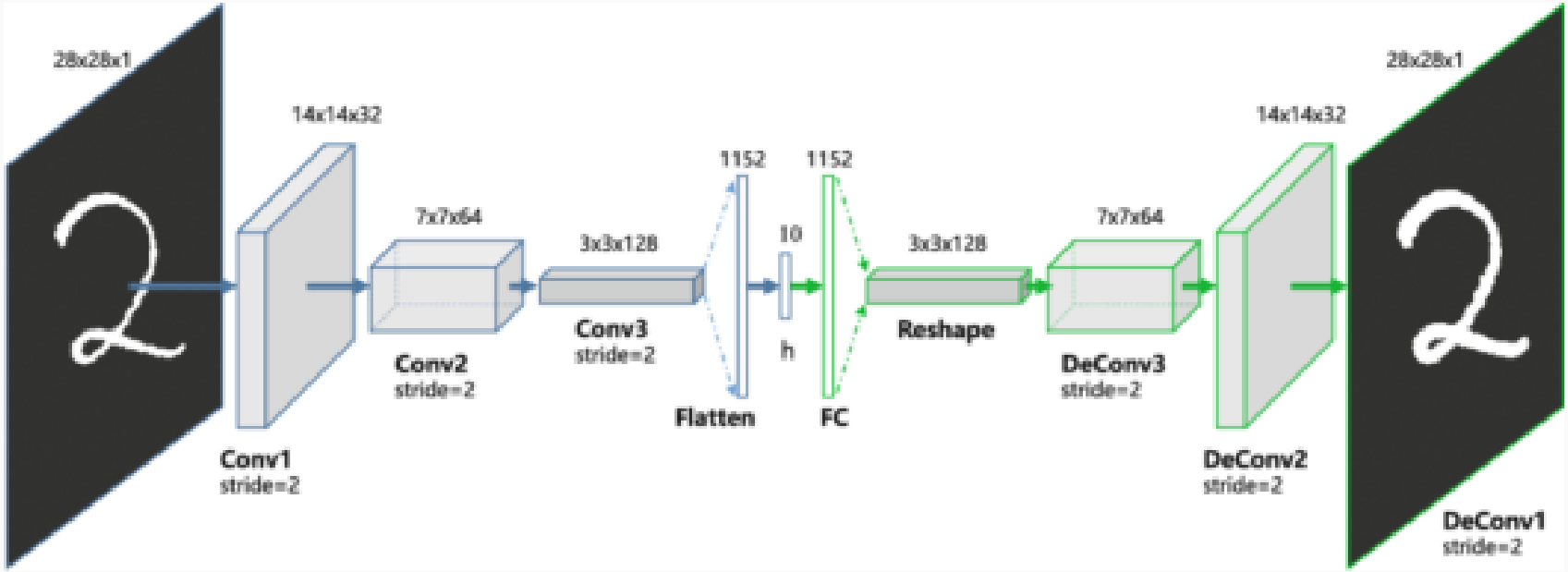
Both GoogLeNet won in 2014 by creating new dense modules that allowed them to deepen their network while using a 10'th as many nodes as AlexNet.

The 2015 winner ResNet used skip layers to jump across dense layer, allowing the network to train a rough structure before the fine structure was trained.

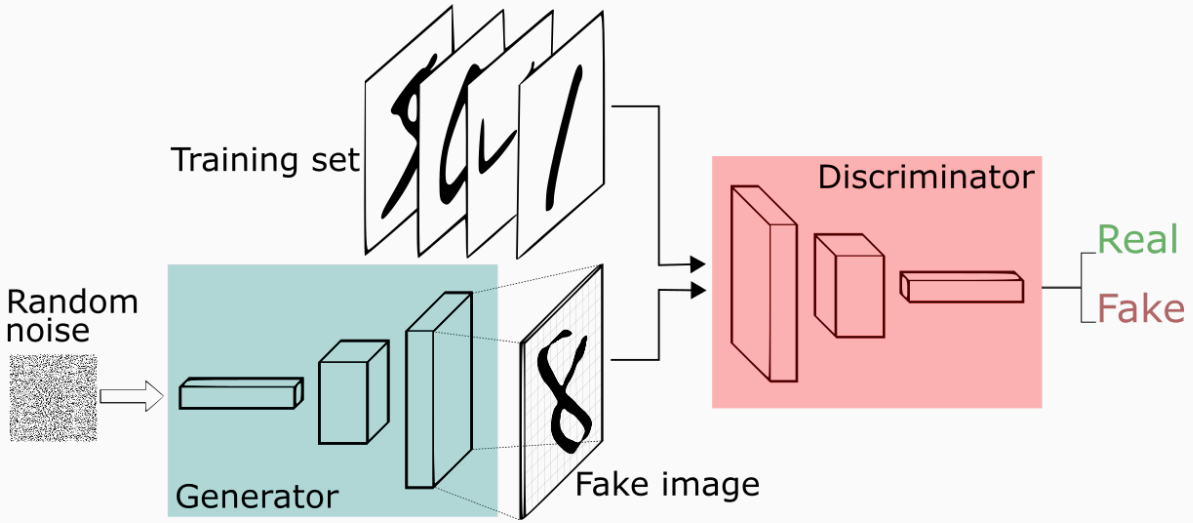
From computer vision to machine translation, CNN's are an active and rich area of research. Furthermore, with libraries like Keras spinning up a CNN for image classification is as easy as defining the order and size of your layers.

Feature Extraction

Autoencoder



General Advisatial Network



Recurrent Networks

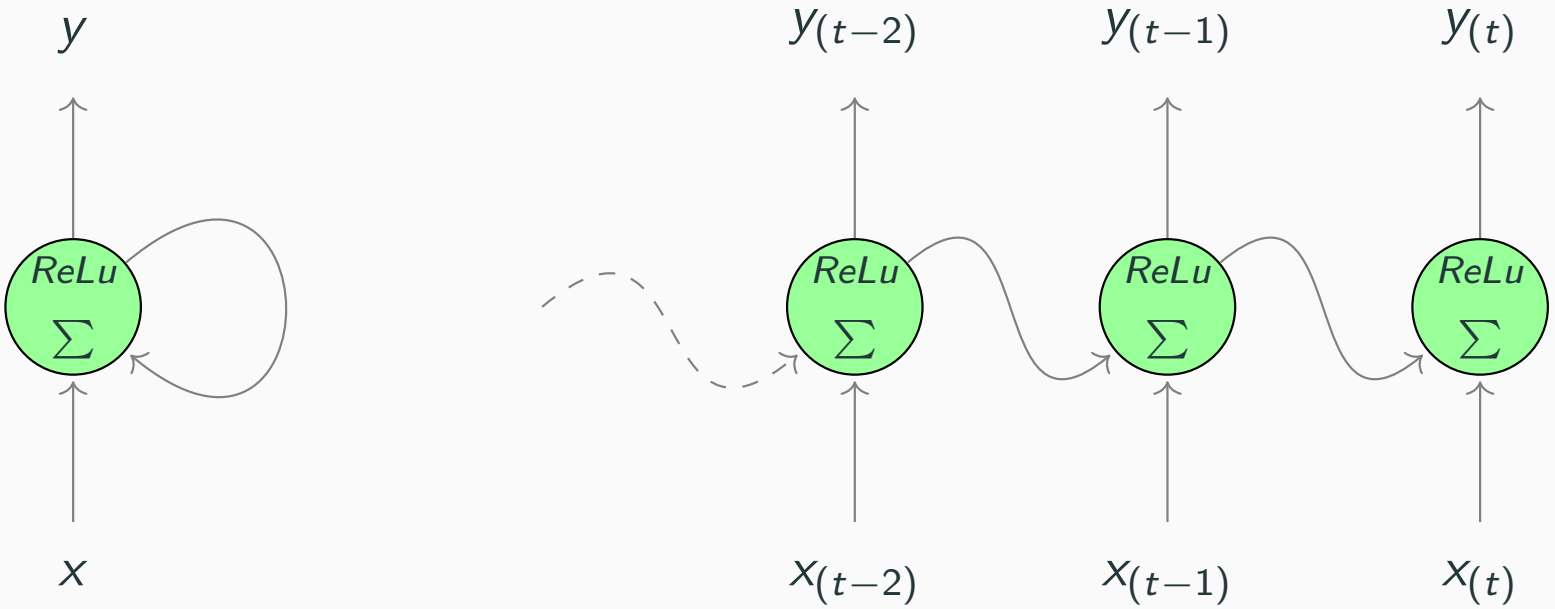
Recurrent Networks

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.

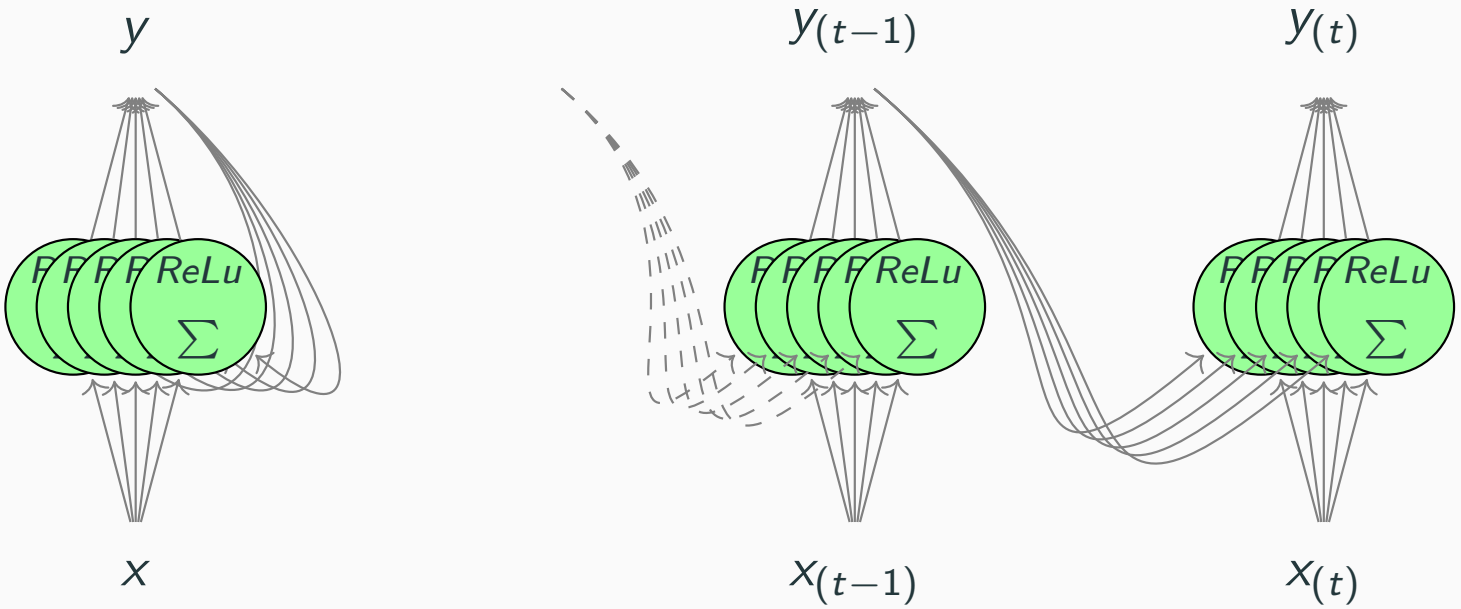
`https://botnik.org/content/harry-potter.html`

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 $y_{(t-1)}$.

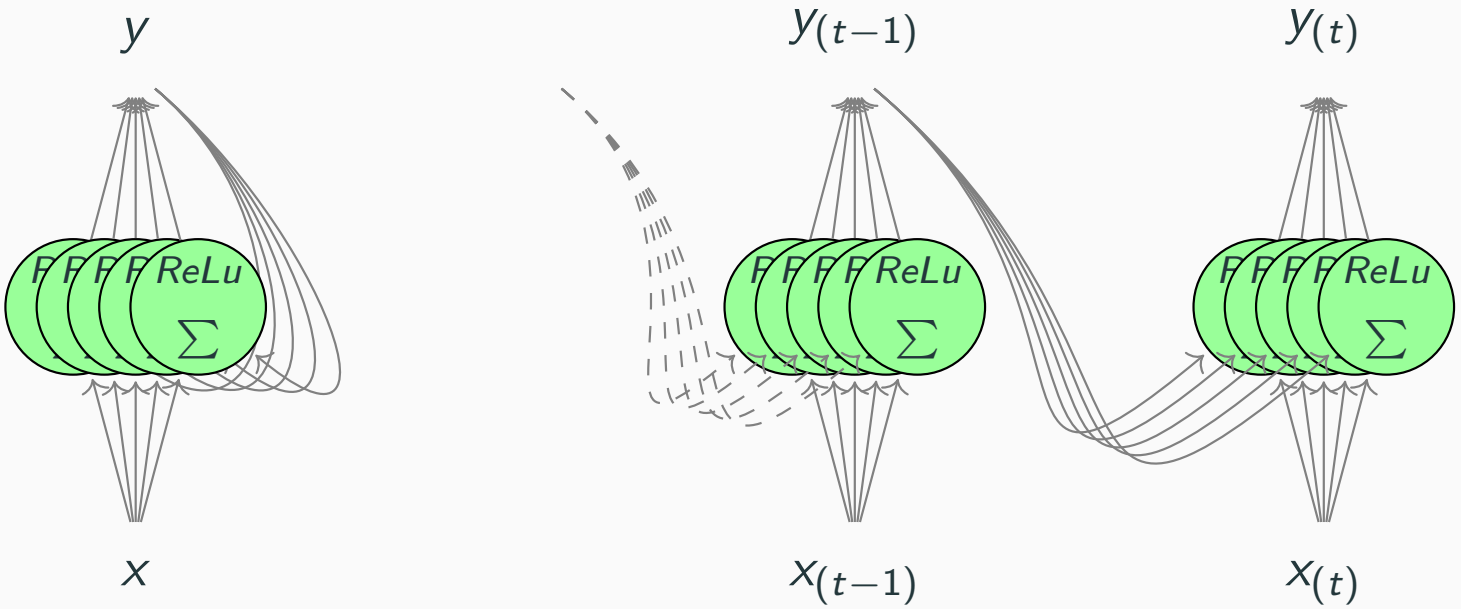
Recurrent Nodes



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 ϕ and bias b,

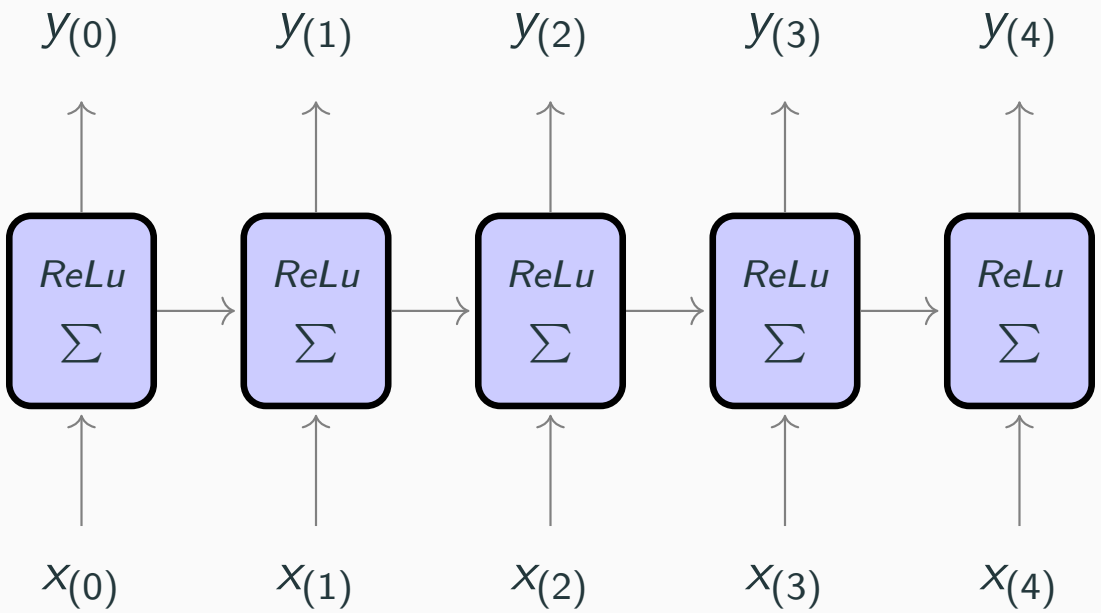
$$\mathbf{Y}_{(t)} = \phi\big(\mathbf{W}_x^T\mathbf{X}_{(t)} + \mathbf{W}_y^T\mathbf{Y}_{(t-1)} + b\big)$$

Memory Cells



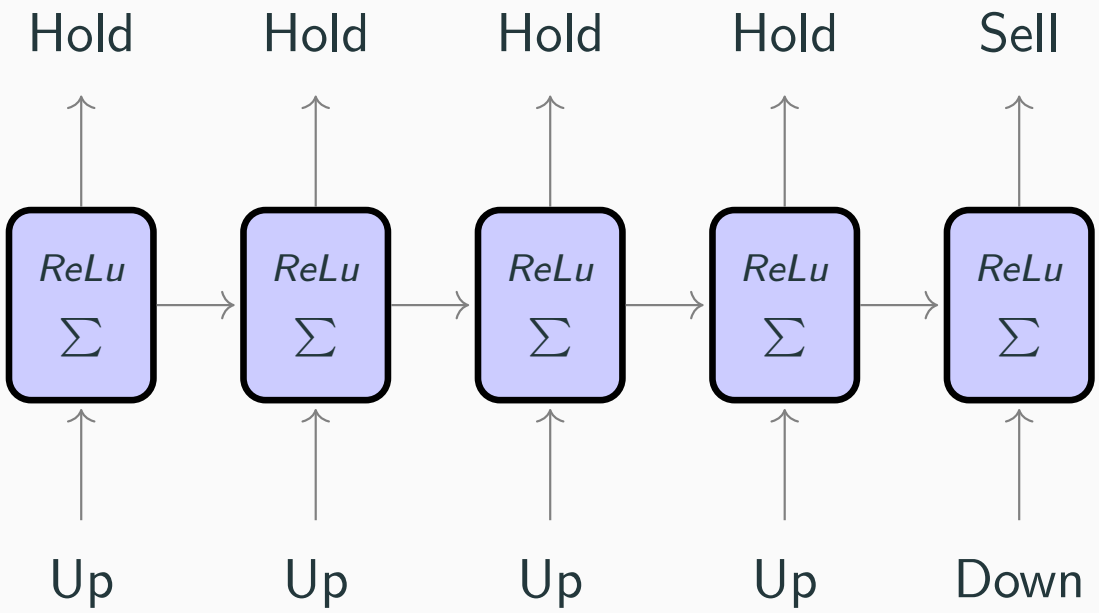
Any cell that passes information to the next frame is called a **memory cell**. In the above we pass the output of the previous frame to produce **short term memory**, but we could of course pass another state vector as well to produce **long term memory**.

Basic RNN Structures



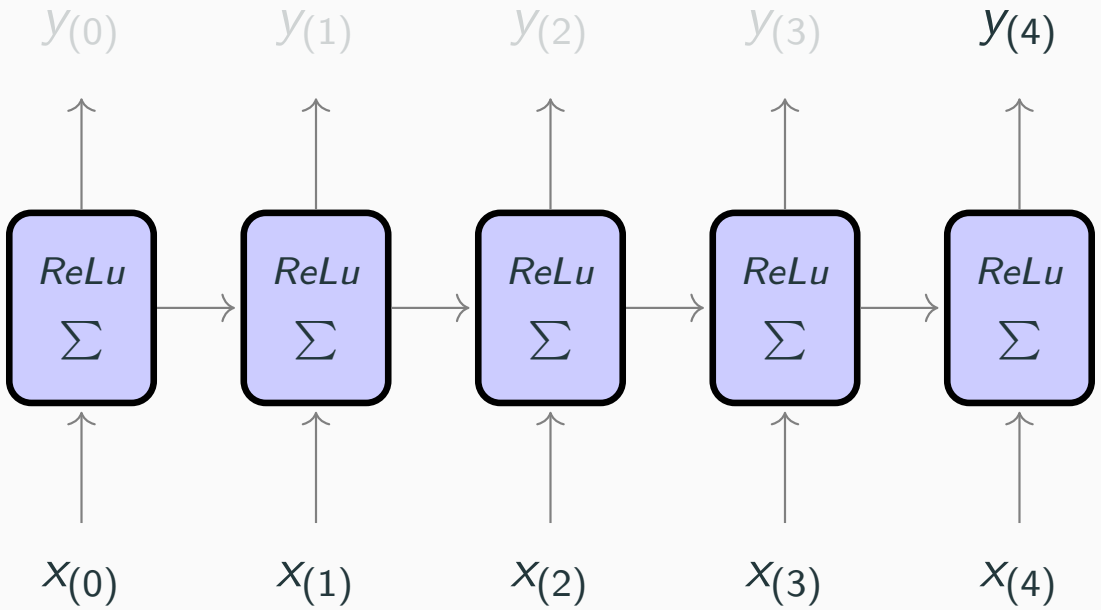
A RNN can take in a series of inputs and produce a series of outputs, as in predicting time series data like stock prices or traffic across a network.

Basic RNN Structures



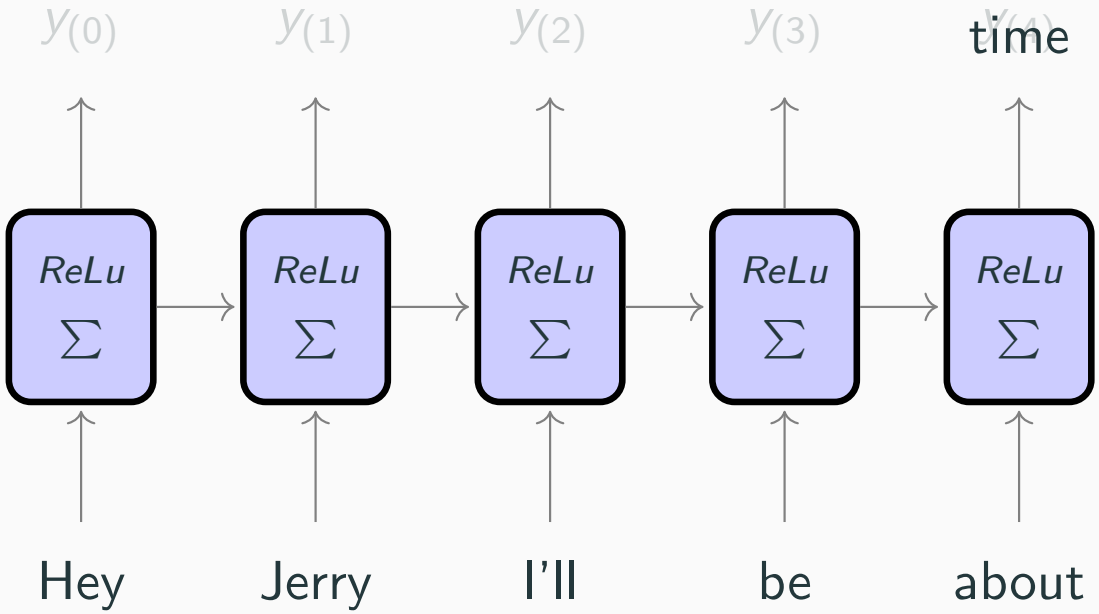
A RNN can take in a series of inputs and produce a series of outputs, as in predicting time series data like stock prices or traffic across a network.

Basic RNN Structures



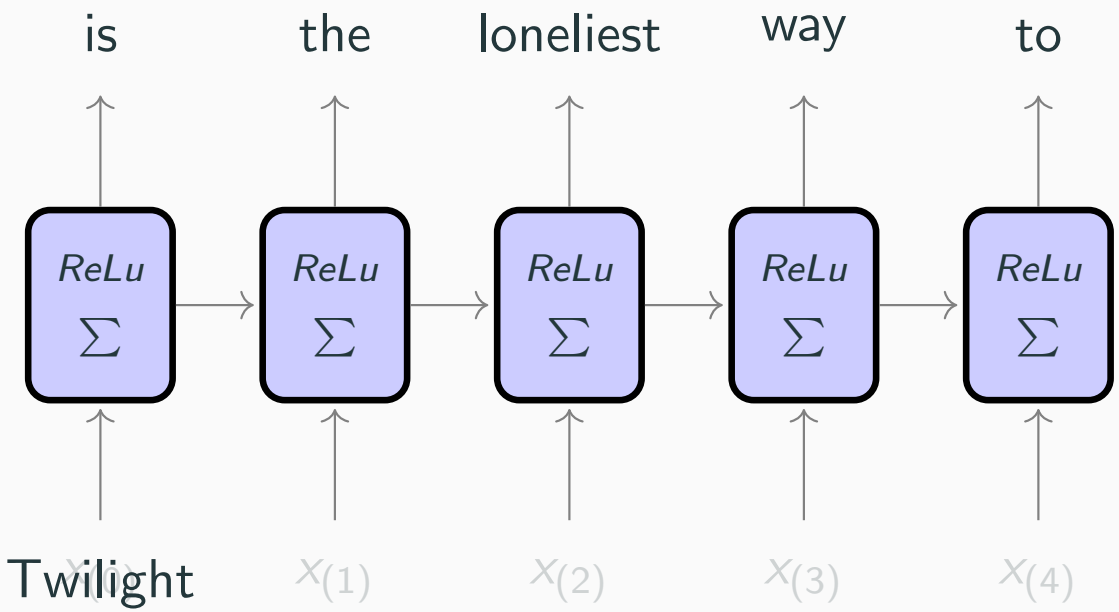
Alternatively, you can feed in many inputs but only record the last output, like in text prediction.

Basic RNN Structures



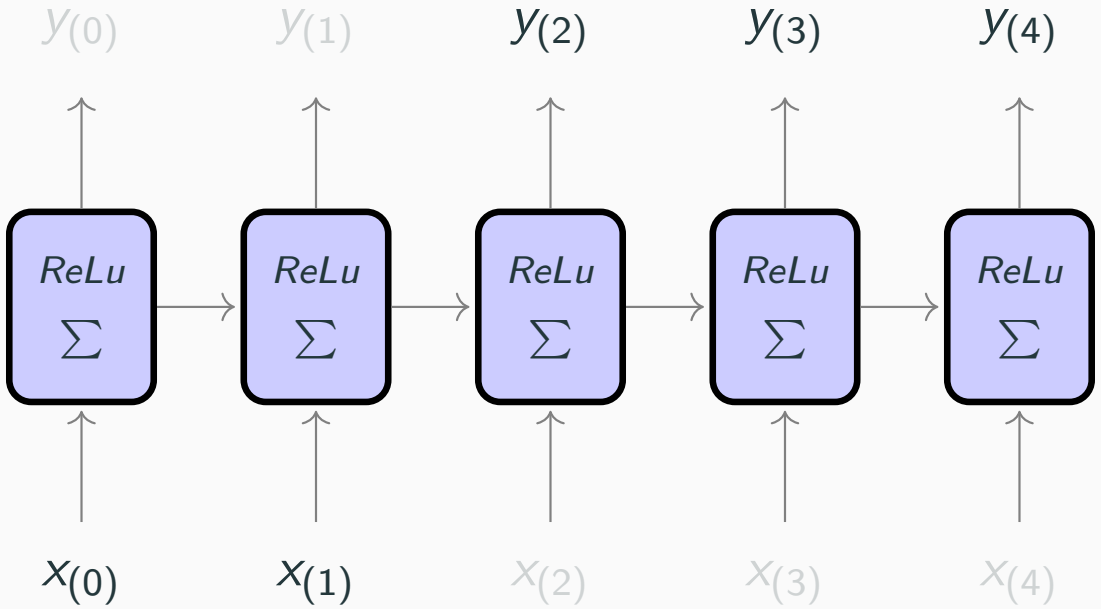
Alternatively, you can feed in many inputs but only record the last output, like in text prediction.

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

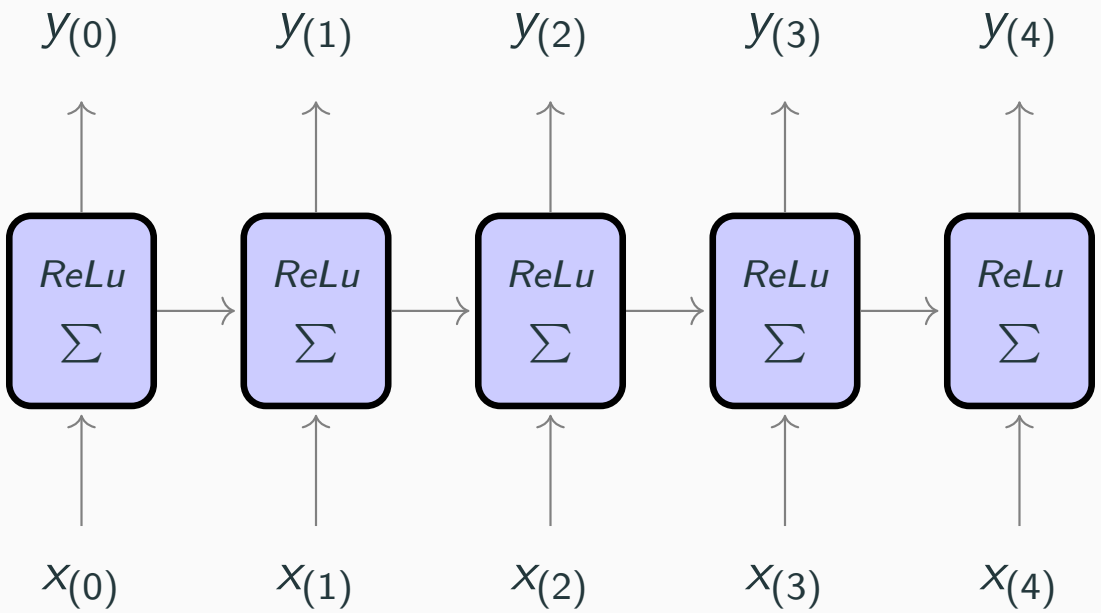


Finally, we can consider encoder/decoder networks where we feed in data encoding it, use the RNN to process it, and then return a new sequence. For example in a translation RNN you would feed n words into the network and return a sentence of an arbitrary length. In such a situation, you should train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

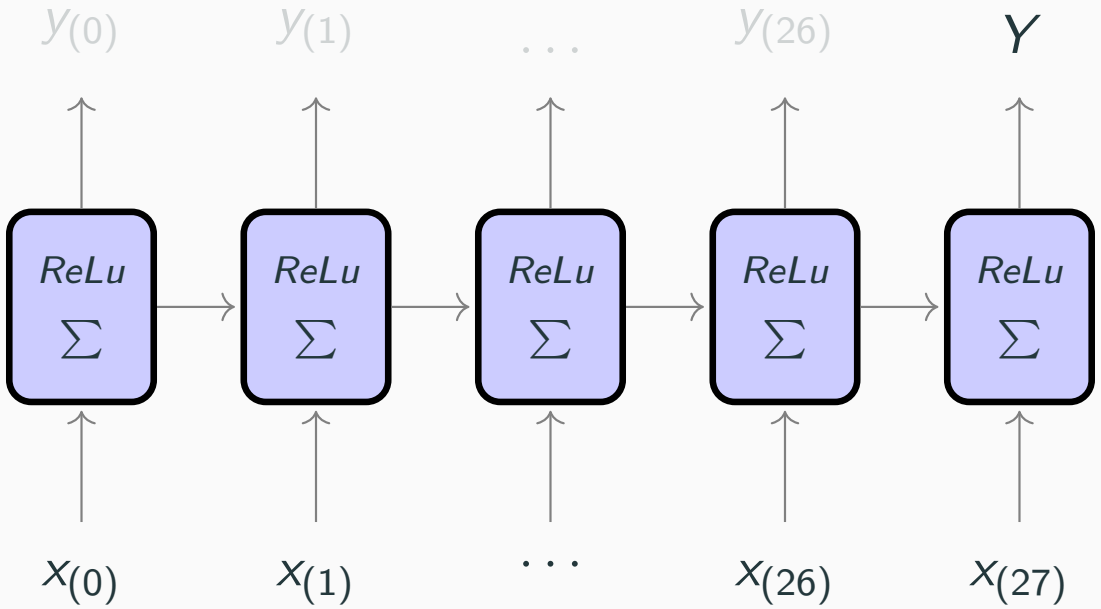
Training: Backpropagation



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 $\mathbf{Y}$.

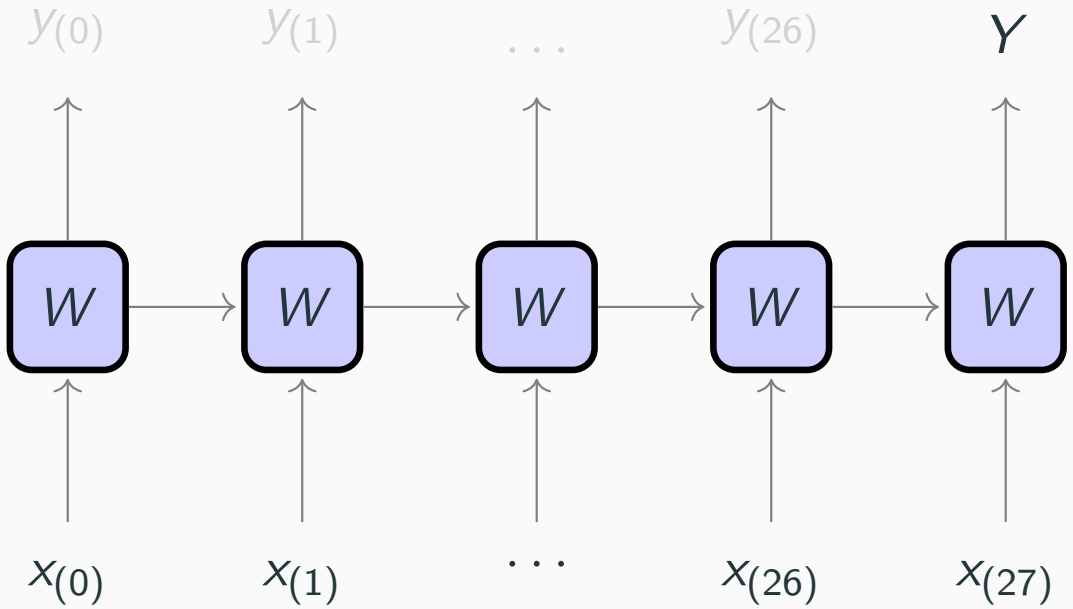
Example: MNIST



For example, we could classify 28×28 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.

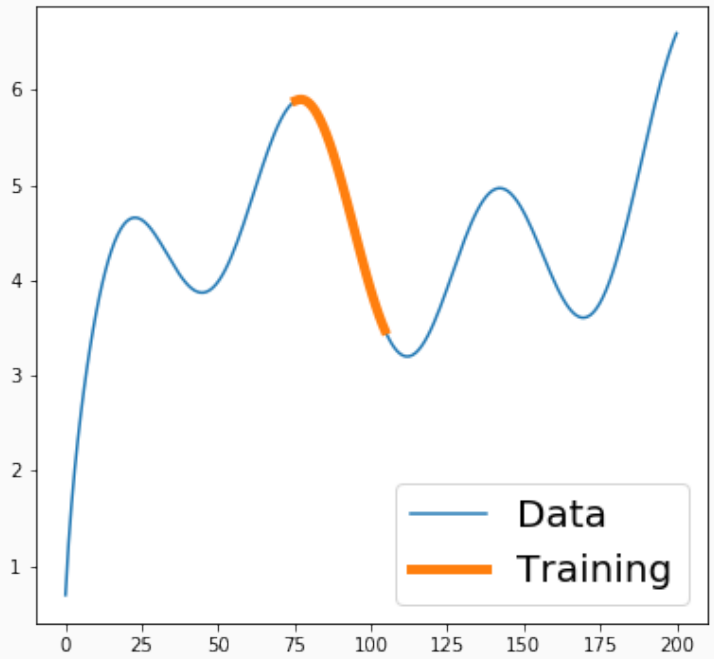
Example: MNIST

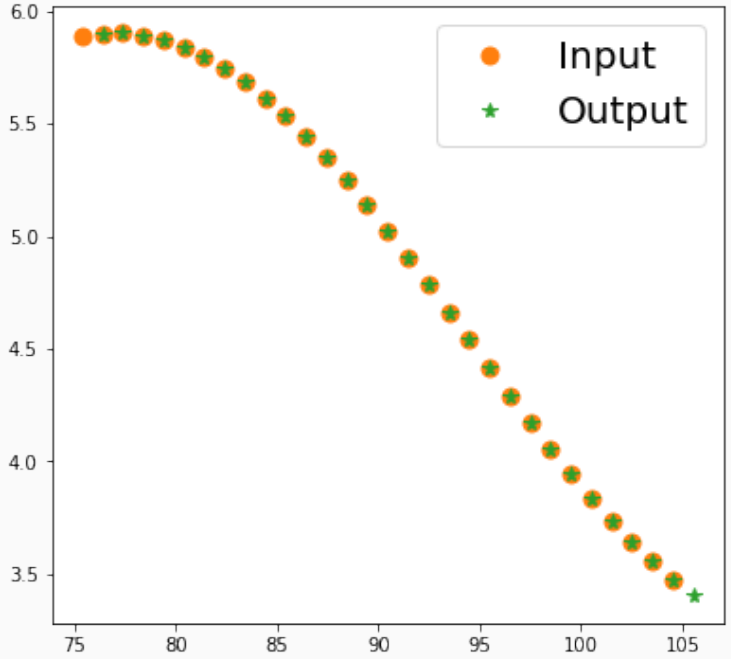


It should be noted that unrolling can be statically or dynamically implemented. Since each sequential layer in the unrolling *contains a copy of the same weights* we can train and test on data with variable lengths.

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.

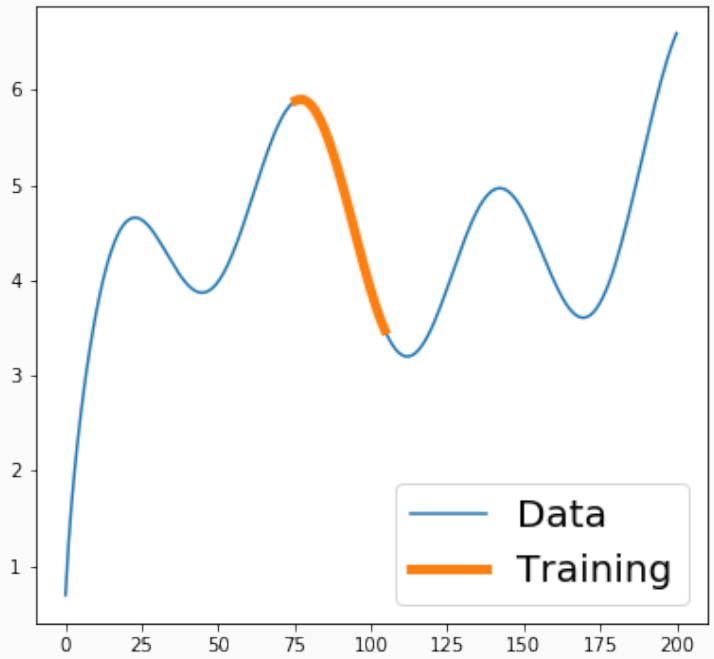
Example: Time Series

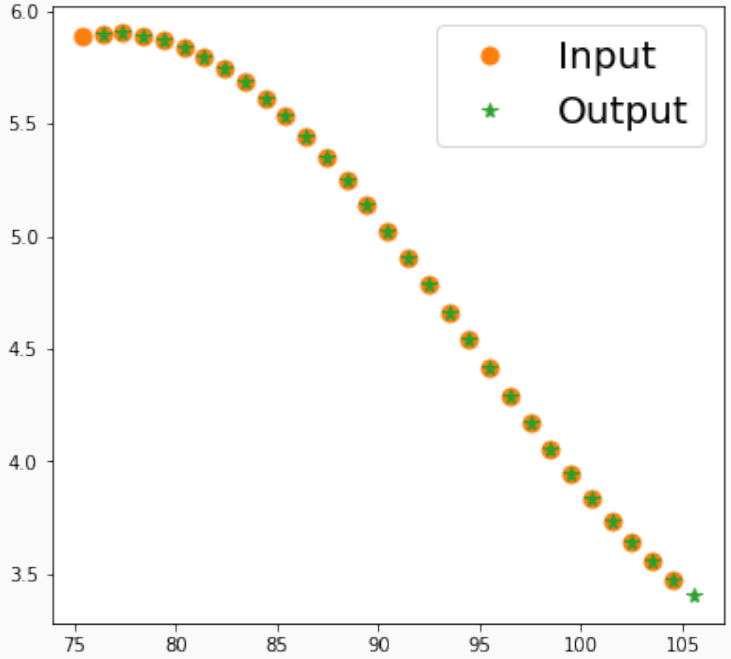




For a standard example, assume we want to predict a time series. To train, we take sequences of 40 data points from the training data and return the data shifted by 1, with a single prediction at the end.

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.

Recurrence Nodes

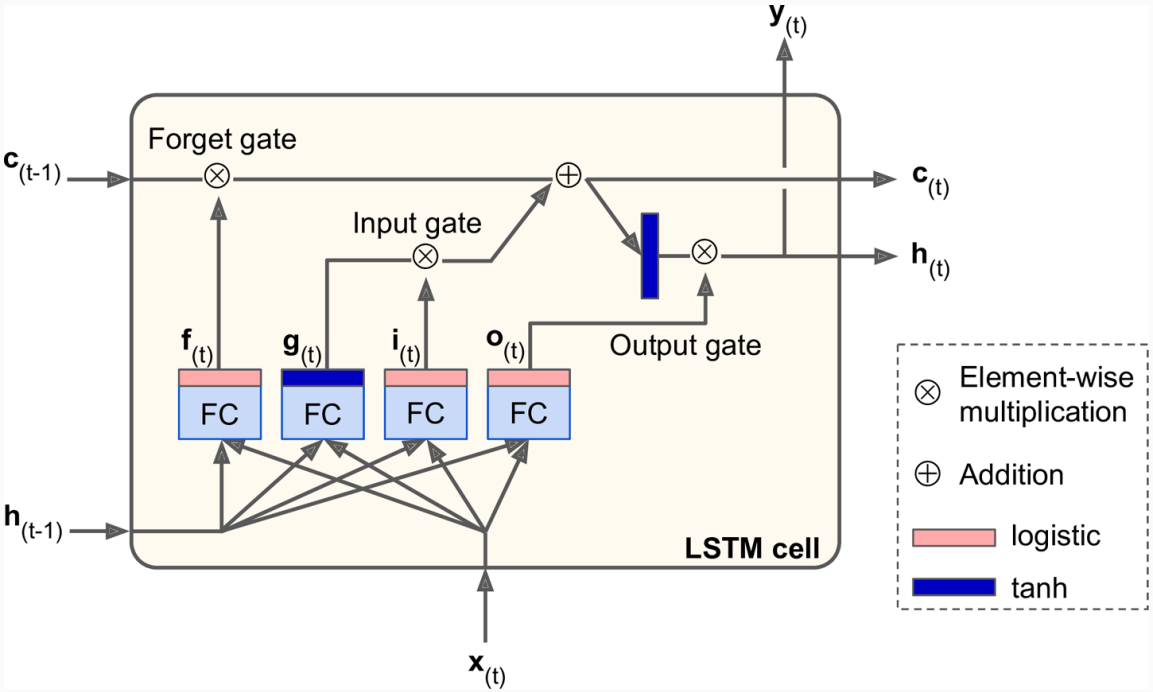
LSTM Cell

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.

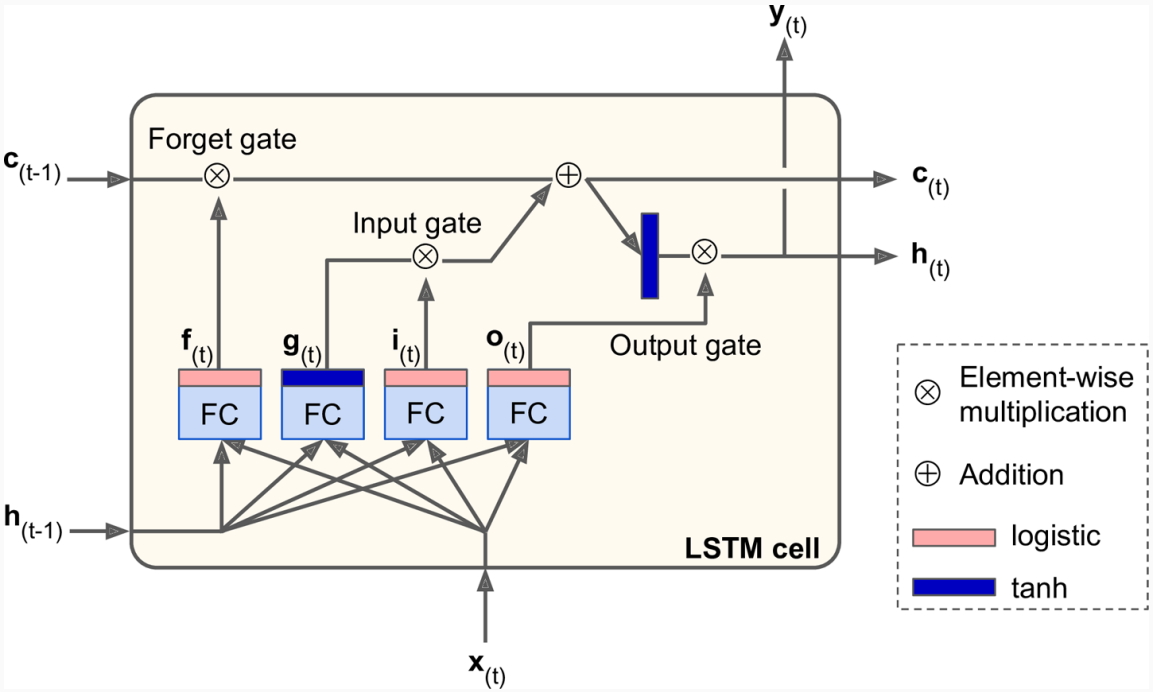
One solution is to build new recurrent nodes that try to solve both these problems. We will finish this lecture by taking a quick look at two such example.

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.

LSTM Cell



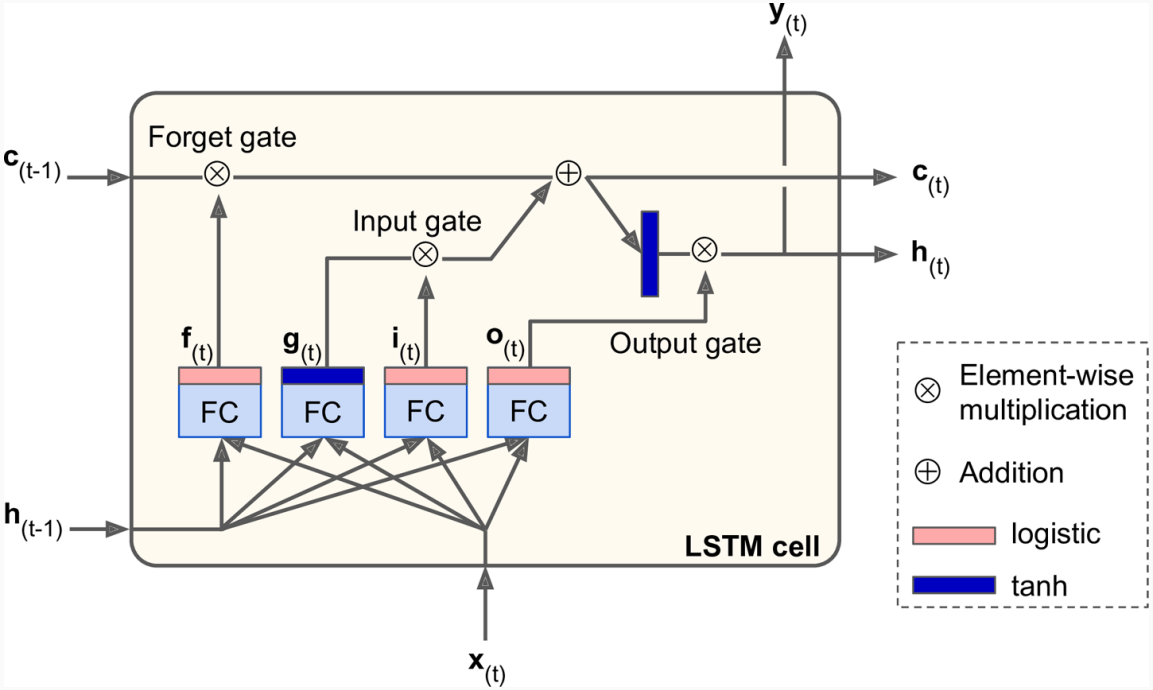
The long term state $c_{(t)}$ passes through the network only interacting three times:

It "forgets" data from $c_{(t-1)}$ depending on the input $x_{(t)}$ and $y_{(t-1)}$.

It then adds a tanh combination of $x_{(t)}$ and $y_{(t-1)}$.

Finally, $c_{(t)}$ is dotted with a sigmoid linear combination $x_{(t)}$ and $y_{(t-1)}$.

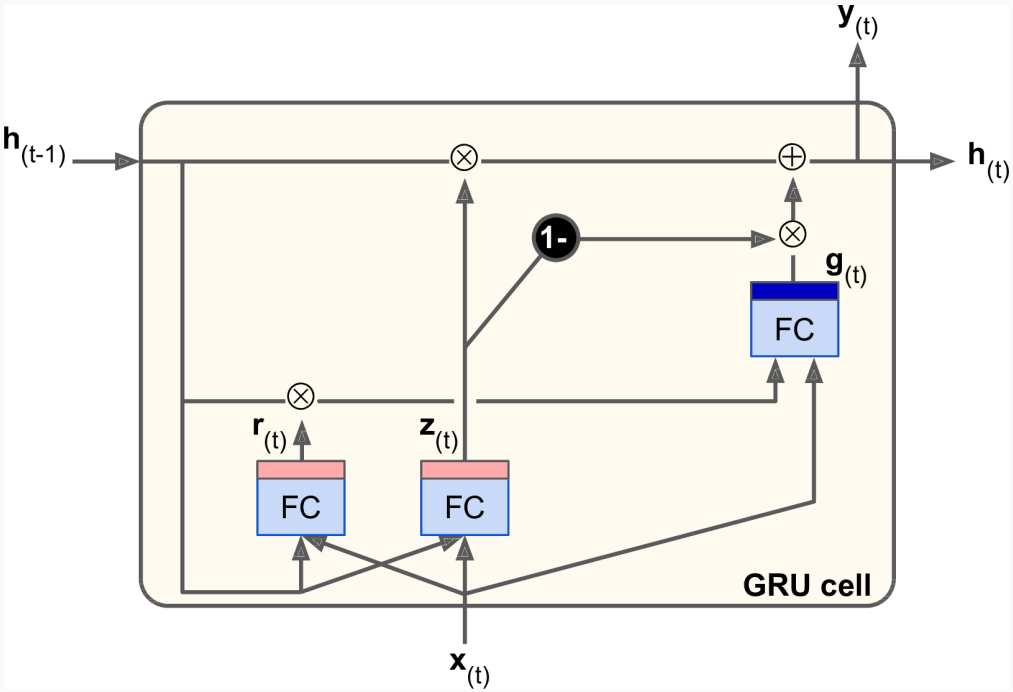
LSTM Cell



Here, each layer is biased linear combination composed with a sigmoid or tanh function. For example,

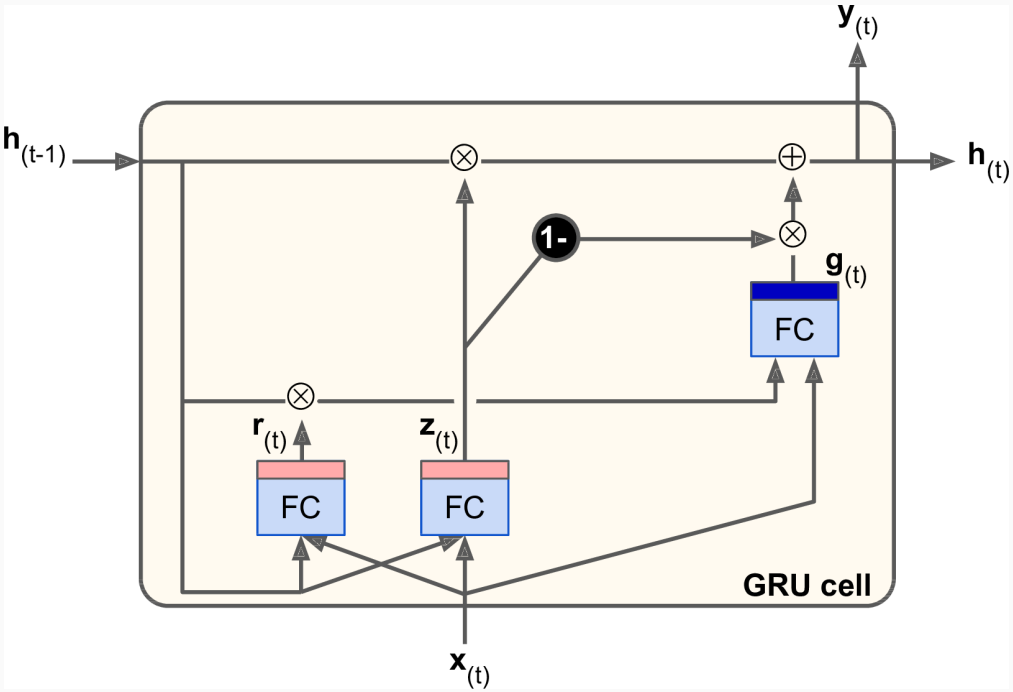
$$\mathbf{i}_{(t)} = \sigma\left(\mathbf{W}_{xi}^T \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \mathbf{h}_{(t-1)} + \mathbf{b}_i\right)$$
$$\mathbf{g}_{(t)} = \sigma\left(\mathbf{W}_{xg}^T \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \mathbf{h}_{(t-1)} + \mathbf{b}_g\right)$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \cdot \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \mathbf{g}_{(t)}$$
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \cdot \tanh\left(\mathbf{c}_{(t)}\right)$$

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.

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