

Deep Learning

Objectives

After completing this module, you should be able to:

Understand Deep Learning concepts

Differentiate between Deep Learning and Machine Learning

Build a CNN model

Build a RNN model

What is Deep Learning?

Deep learning is a type of machine learning in which a model learns to perform classification tasks directly from images, text, or sound.

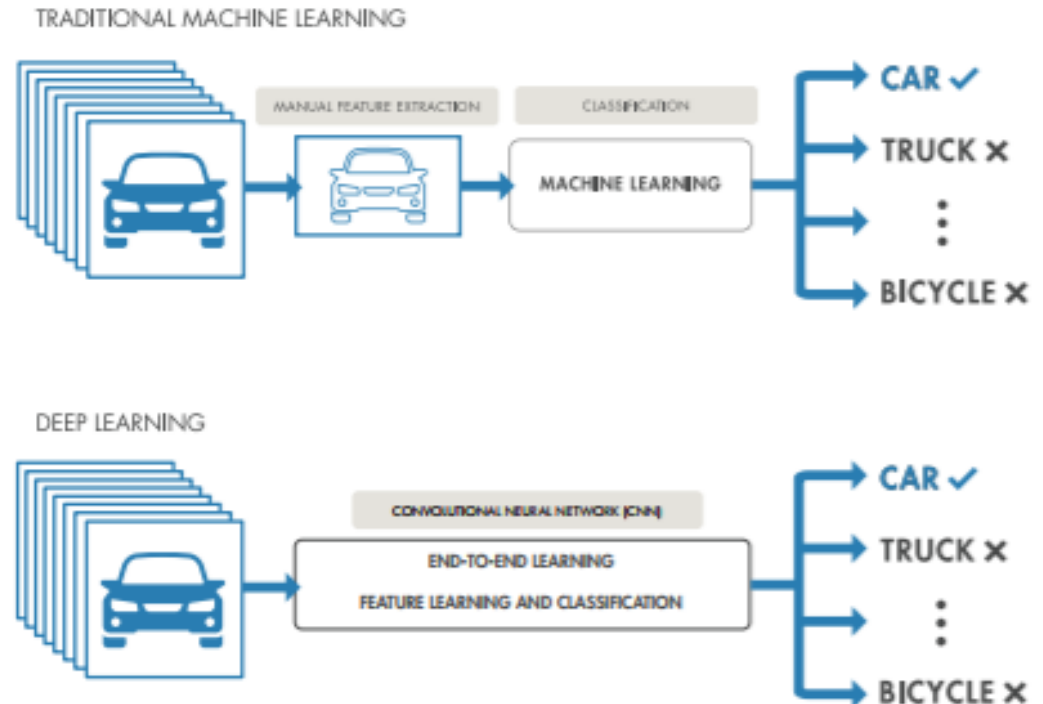
Deep learning is usually implemented using a neural network architecture.

The term “deep” refers to the number of layers in the network—the more layers, the deeper the network. Traditional neural networks contain only 2 or 3 layers, while deep networks can have hundreds.

What is the Difference Between Deep Learning and Machine Learning?

Deep learning is a subtype of machine learning. With machine learning, you manually extract the relevant features of an image.

With deep learning, you feed the raw images directly into a deep neural network that learns the features automatically.



What is the Difference Between Deep Learning and Machine Learning?

Machine Learning	Deep Learning
+ Good results with small datasets	- Requires very large data sets
+ Quick to train a model	- Computationally Intensive
- Need to try different features and classifiers to achieve best results.	+ Learns features and classifiers automatically
- Accuracy plateaus	+ Accuracy is unlimited

Deep Learning Applications

Deep learning is especially well-suited to identification applications such as face recognition, text translation, voice recognition, and advanced driver assistance systems, including, lane classification and traffic sign recognition.

Here are just a few examples of deep learning at work:

- A self-driving vehicle slows down as it approaches a pedestrian crosswalk.
- An ATM rejects a counterfeit bank note.
- A smartphone app gives an instant translation of a foreign street sign.

Image Captioning



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"construction worker in orange safety vest is working on road."



"man in black shirt is playing guitar."

Computational Resources for Deep Learning

Training a deep learning model can take hours, days, or weeks, depending on the size of the data and the amount of processing power you have available.

Currently, there are three computation options: CPU-based, GPUbased, and cloud-based.

- CPU-based computation is the simplest and most readily available option. Using CPU-based computation only for simple examples using a pretrained network.
- Using a GPU reduces network training time from days to hours.
- Cloud-based GPU computation means that you don't have to buy and set up the hardware yourself.














A deep neural network combines multiple nonlinear processing layers, using simple elements operating in parallel and inspired by biological nervous systems.

It consists of an input layer, several hidden layers, and an output layer. The layers are interconnected via nodes, or neurons, with each hidden layer using the output of the previous layer as its input.

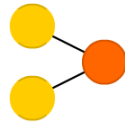
Inside a Deep Neural Network

A mostly complete chart of Neural Networks

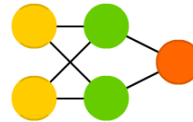
©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probablistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool

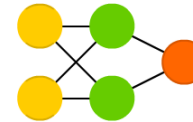
Perceptron (P)



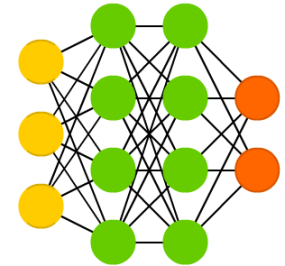
Feed Forward (FF)



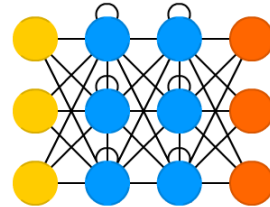
Radial Basis Network (RBF)



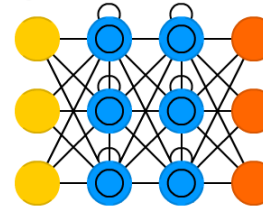
Deep Feed Forward (DFF)



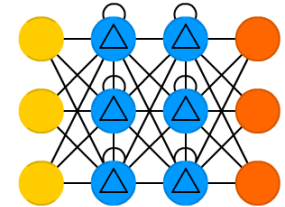
Recurrent Neural Network (RNN)



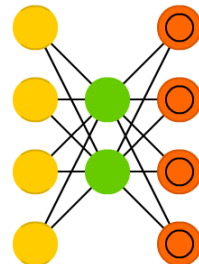
Long / Short Term Memory (LSTM)



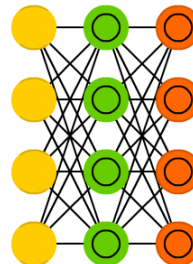
Gated Recurrent Unit (GRU)



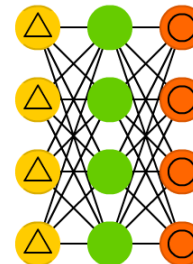
Auto Encoder (AE)



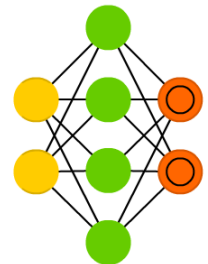
Variational AE (VAE)



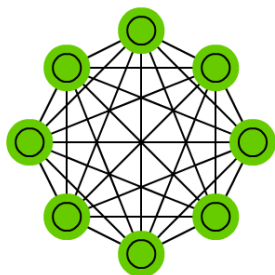
Denoising AE (DAE)



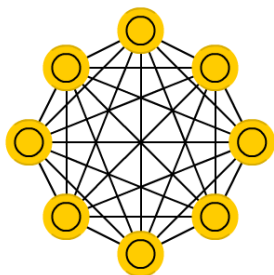
Sparse AE (SAE)



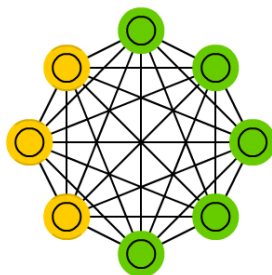
Markov Chain (MC)



Hopfield Network (HN)



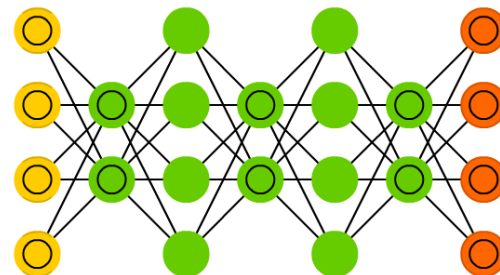
Boltzmann Machine (BM)



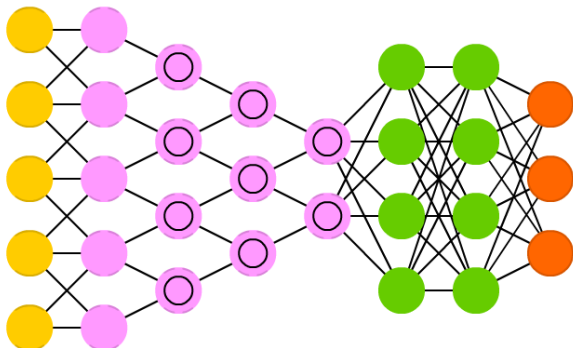
Restricted BM (RBM)



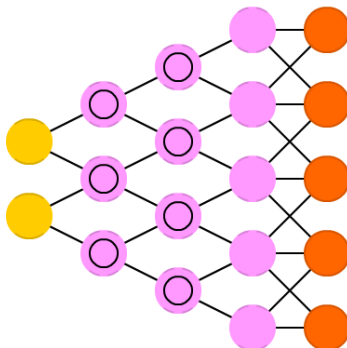
Deep Belief Network (DBN)



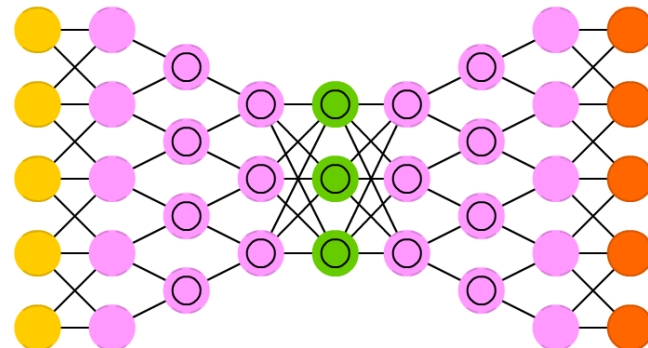
Deep Convolutional Network (DCN)



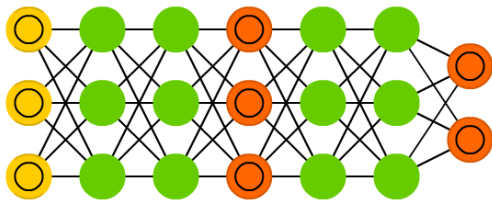
Deconvolutional Network (DN)



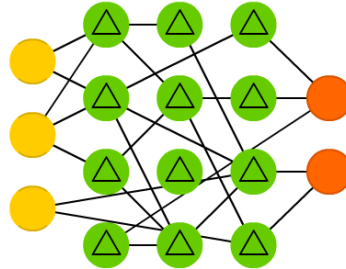
Deep Convolutional Inverse Graphics Network (DCIGN)



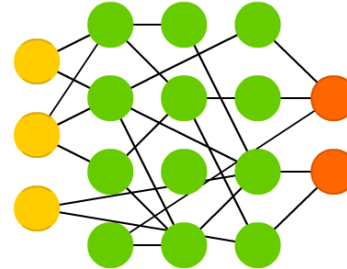
Generative Adversarial Network (GAN)



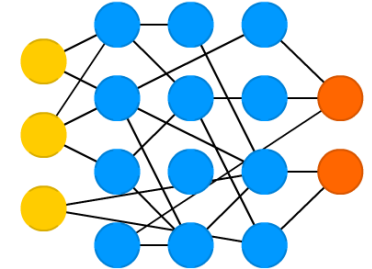
Liquid State Machine (LSM)



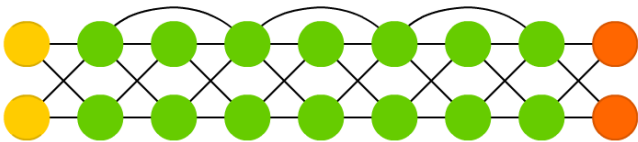
Extreme Learning Machine (ELM)



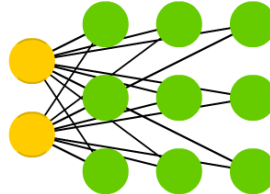
Echo State Network (ESN)



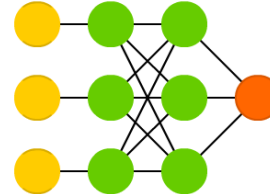
Deep Residual Network (DRN)



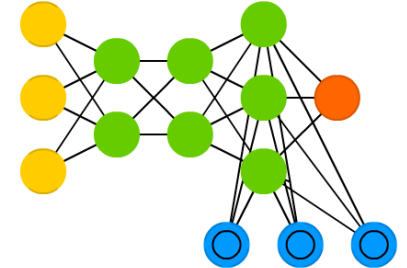
Kohonen Network (KN)



Support Vector Machine (SVM)



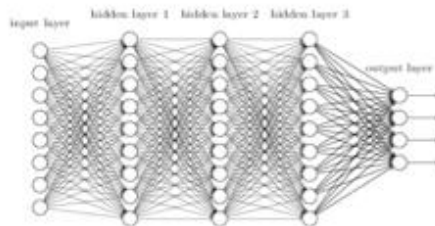
Neural Turing Machine (NTM)



Deep Learning Algorithms

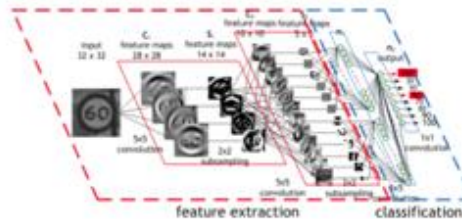
providing lift for
classification and
forecasting models

Deep
Neural
Networks



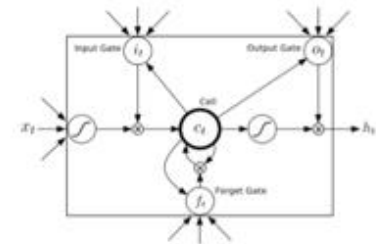
feature extraction
and classification of
images

Convolutional
Neural
Networks

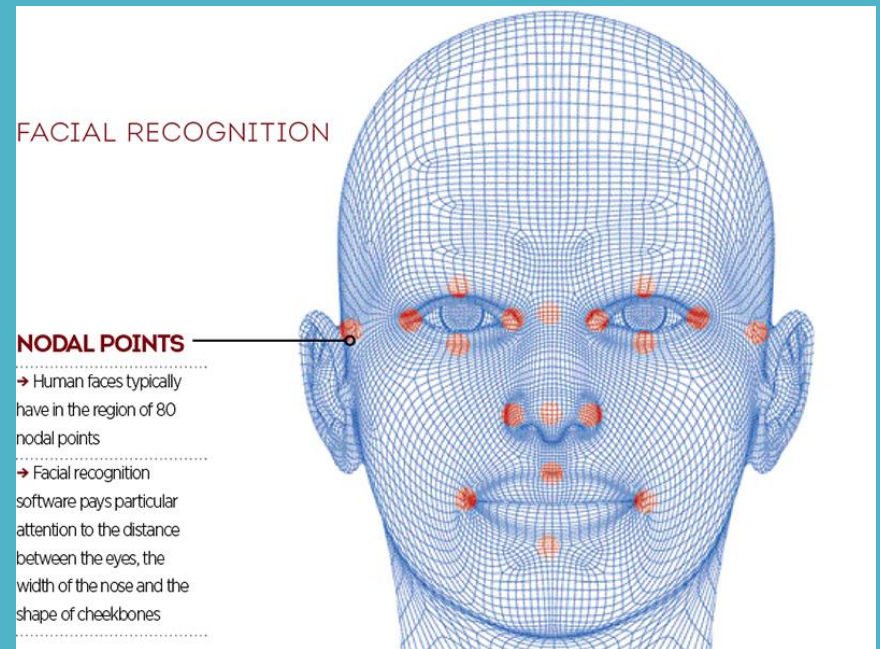


for sequence of events,
language models, time
series, etc.

Recurrent
Neural
Networks

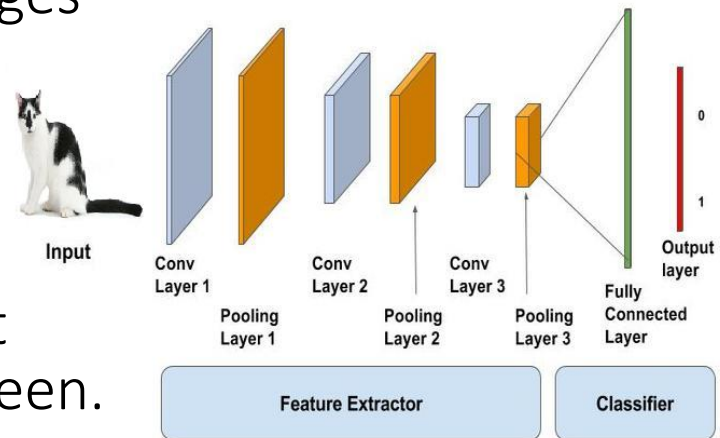


Convolutional Neural Network



A convolutional neural network (CNN, or ConvNet) is one of the most popular algorithms for deep learning with images and video.

Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between.



Convolutional Neural Networks

Feature Detection Layers

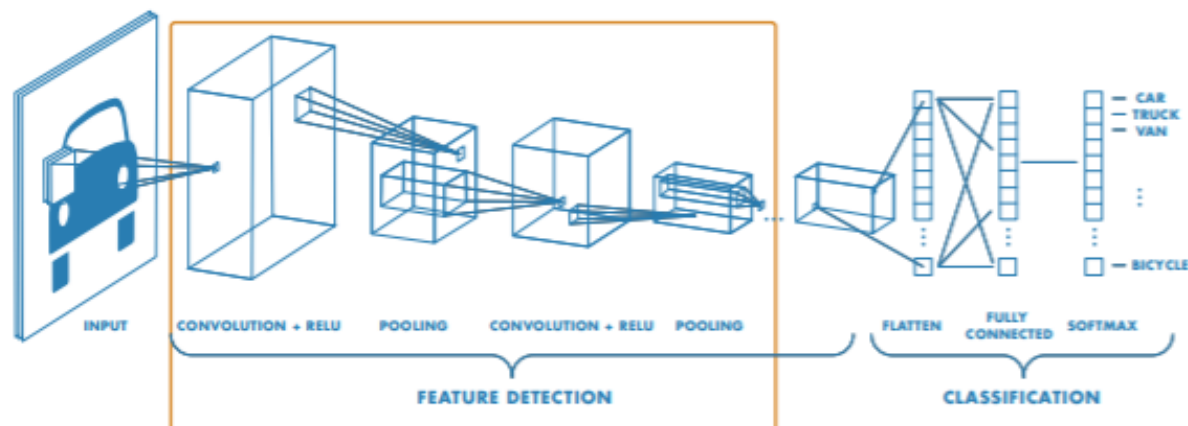
These layers perform one of three types of operations on the data: convolution, pooling, or rectified linear unit (ReLU).

Convolution puts the input images through a set of convolutional filters, each of which activates certain features from the images.

Pooling simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn about.

Rectified linear unit (ReLU) allows for faster and more effective training by mapping negative values to zero and maintaining positive values.

These three operations are repeated over tens or hundreds of layers, with each layer learning to detect different features.

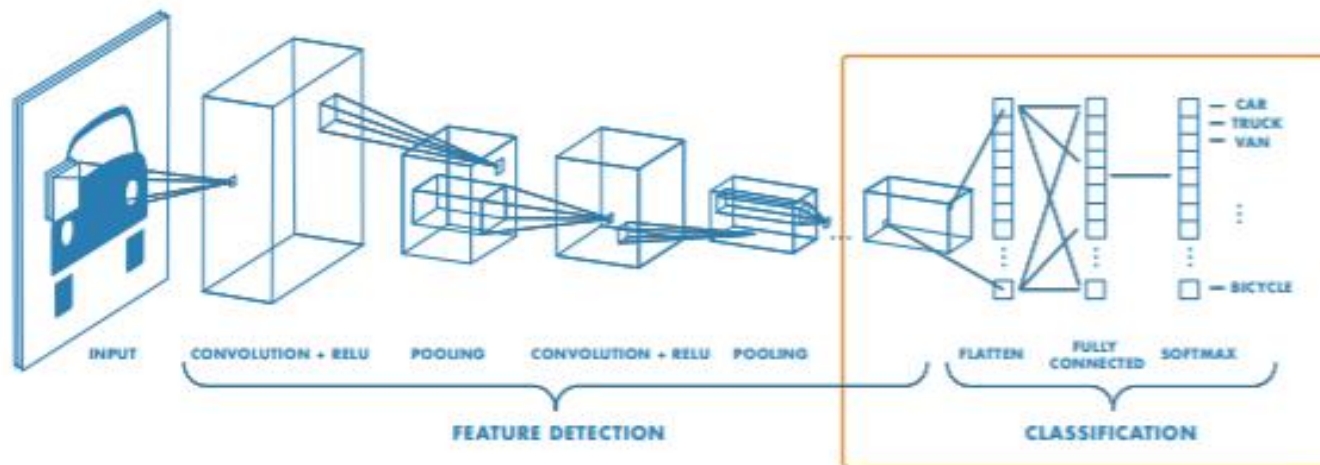


Classification Layers

After feature detection, the architecture of a CNN shifts to classification.

The next-to-last layer is a fully connected layer (FC) that outputs a vector of K dimensions where K is the number of classes that the network will be able to predict. This vector contains the probabilities for each class of any image being classified.

The final layer of the CNN architecture uses a softmax function to provide the classification output.



Convolution layer

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

WEIGHT

1	0	1
0	1	0
1	0	1

429

concept of stride and padding

Stride

INPUT IMAGE						WEIGHT			
18	54	51	239	244	188	1	0	1	429
55	121	75	78	95	88	0	1	0	
35	24	204	113	109	221	1	0	1	
3	154	104	235	25	130				
15	253	225	159	78	233				
68	85	180	214	245	0				

concept of stride and padding

Padding

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

WEIGHT

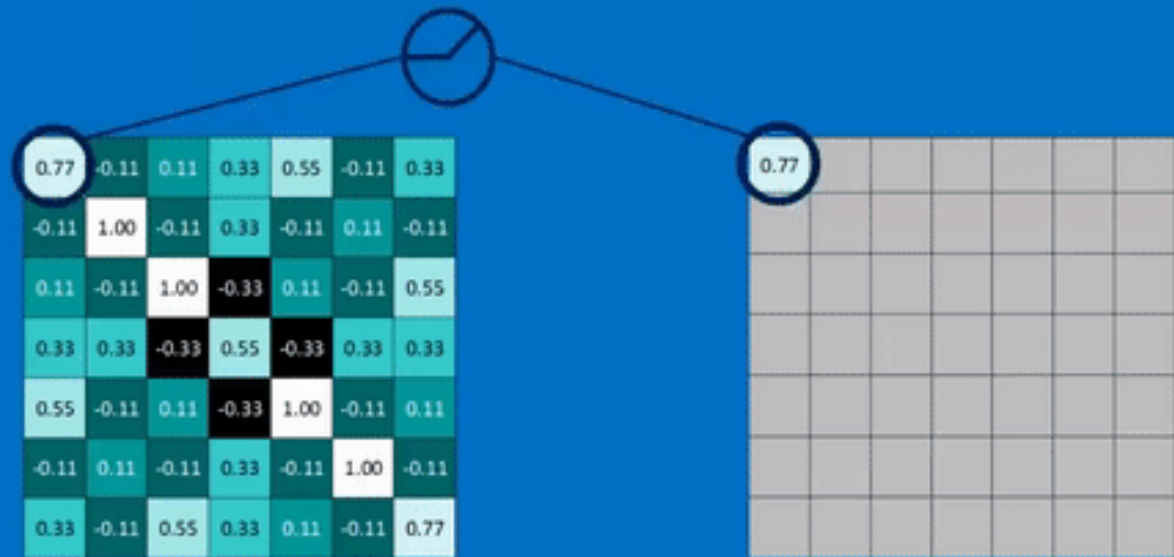
1	0	1
0	1	0
1	0	1

139

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

Relu

Rectified Linear Units (ReLUs)

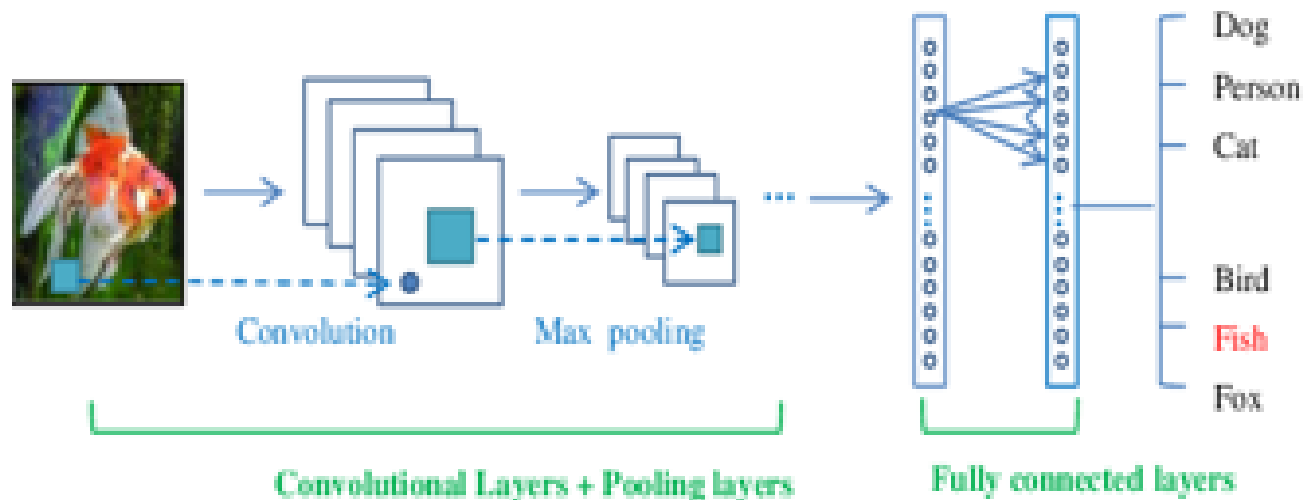


Maxpooling

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

Output Layer or Fully Connected Layer



Design of Convolution Neural Network

Recurrent Neural Network (RNN)

RNN algorithm mostly used in NLP task.

Best quality of RNN algorithm is it has capacity of remember things or past data.

This algorithm is widely used when all inputs are not independent to each other.

Application(Famous)

- Text Translation
- Speech Recognition
- Language Modeling and Generating Text

Recurrent Neural Network (RNN)

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea.

If you want to predict the next word in a sentence you better know which words came before it.

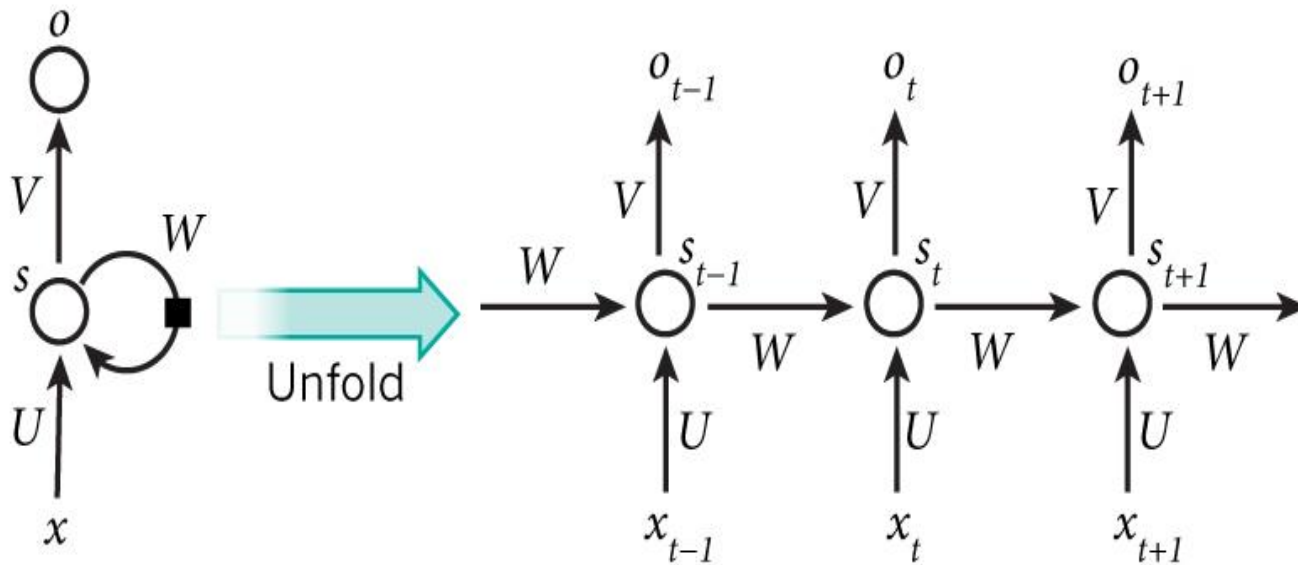
RNN

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.

Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far.

In theory RNNs can make use of information in arbitrarily long sequences.

RNN -Model

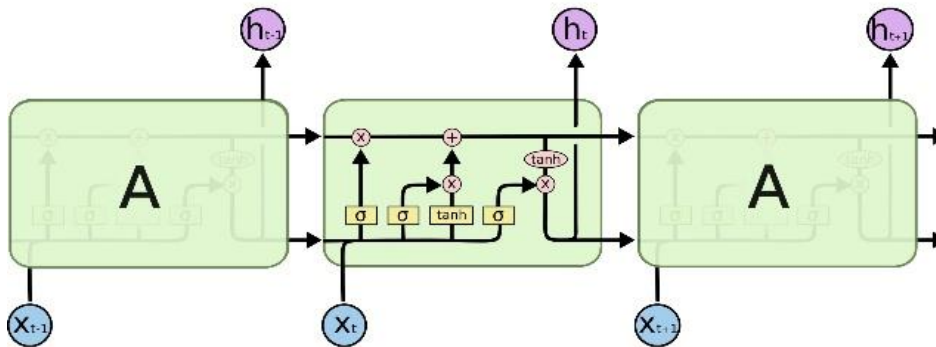


Long Short Term Memory

Long Short Term Memory

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

Long Short-Term Memory (LSTM)



LSTM

LSTMs are explicitly designed to avoid the long-term dependency problem.

Remembering information for long periods of time.

Why LSTM?

LSTM based on

- Long term dependencies
- LSTM fight the Vanishing/Exploding gradient problem
- In LSTM we remember only important things(from sequence data) for long time not everything.

vanishing and exploding gradients

Two major problems torment the RNNs — vanishing and exploding gradients.

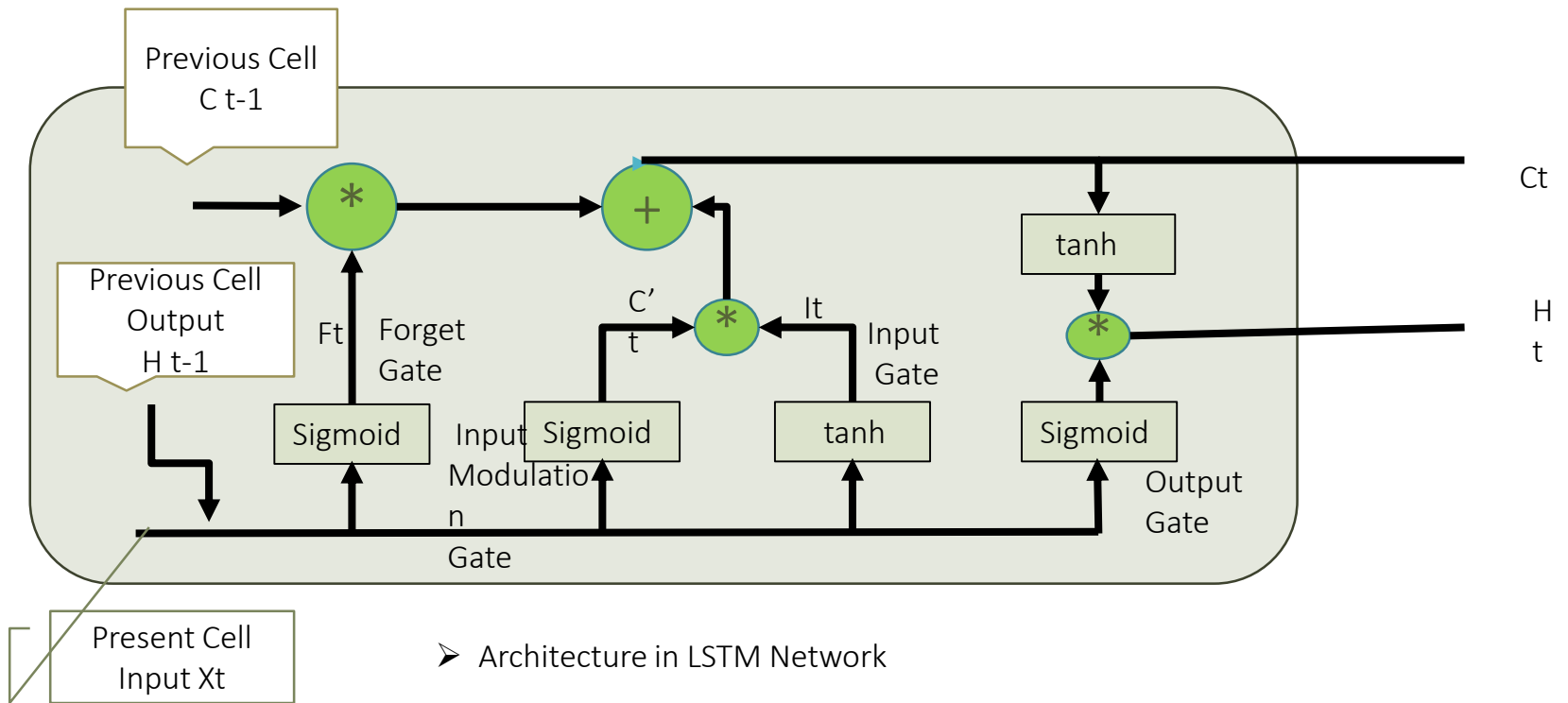
In traditional RNNs the gradient signal can be multiplied a large number of times by the weight matrix.

And we know that magnitude of the weights of the transition matrix can play an important role.

If the weights in the matrix are small, the gradient signal becomes smaller at every training step, thus making learning very slow or completely stops it. This is called vanishing gradient.

vanishing and exploding gradients

And exploding gradient refers to the weights in this matrix being so large that it can cause learning to diverge.



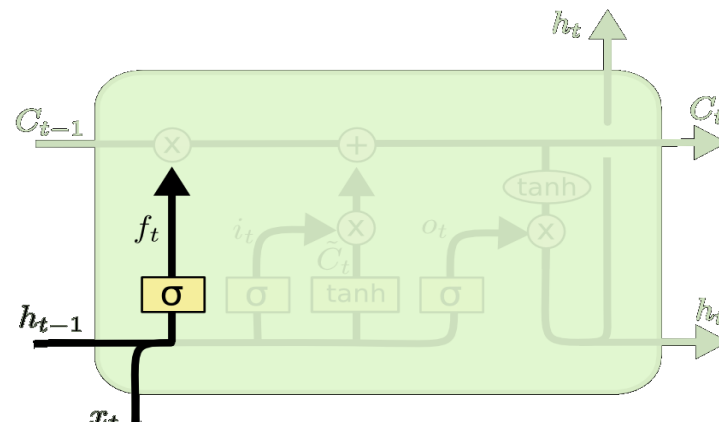
Step-by-Step LSTM Walk Through

In this step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

It looks at h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} .

1 represents "completely keep this" while a 0 represents "completely get rid of this."

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



Step-by-Step LSTM Walk Through

For example of a language model trying to predict the next word based on all the previous ones.

In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used.

When we see a new subject, we want to forget the gender of the old subject.

Step-by-Step LSTM Walk Through

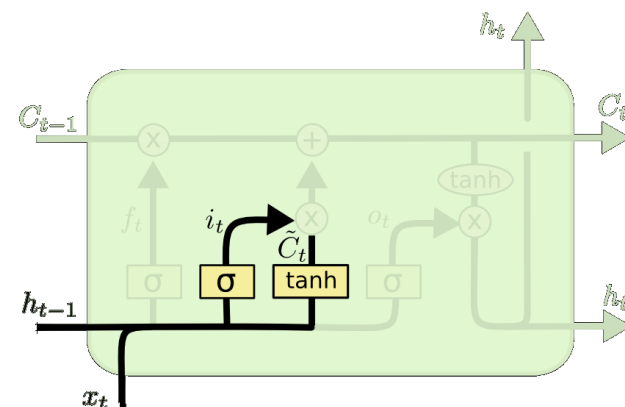
The next step is to decide what new information we're going to store in the cell state.

This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update.

Next, a tanh layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. In the next step, we'll combine these two to create an update to the state

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



Step-by-Step LSTM Walk Through

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

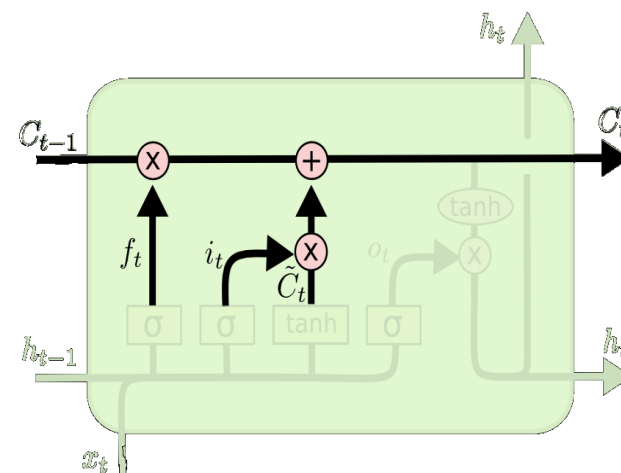
Step-by-Step LSTM Walk Through

Now we will update old cell state, C_{t-1} , into the new cell state C_t .

We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$.

This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

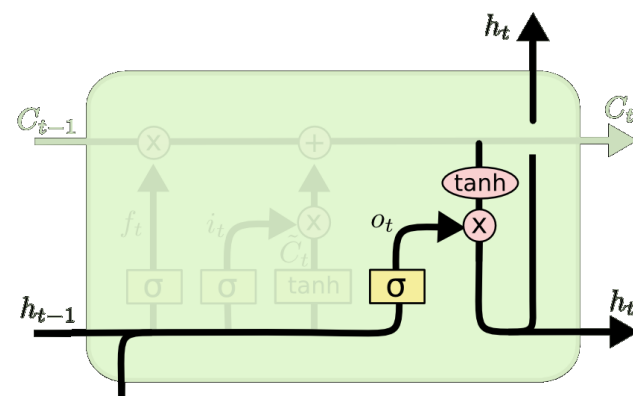


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM Walk Through

Output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Step-by-Step LSTM Walk Through

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next.

For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

Thank you