

50.007 Machine Learning

Neural Network, Deep Learning

Berrak Sisman

Assistant Professor, ISTD Pillar, SUTD

berrak_sisman@sutd.edu.sg

Introduction & Content

- Neural Networks and Deep Learning (week 5 and 6)

Instructor: Prof. Berrak Sisman
Email: berrak_sisman@sutd.edu.sg
Feel free to contact me!



History of Neural Networks

What is Neural Network?

Why do we use it?

Can computer beat the human brain now?

- Yes and no.

Can you list a few tasks that the computer can beat the human brain?

- Playing chess---the Deep Blue defeated the world champion Garry Kasparov in 1997.
- Solving equations

A recent famous victory of computer beating the human brain?

- IBM Watson won Jeopardy in 2011!

Are we doomed to lose to the machine?

What is Neural Network?

Why do we use it?

There are certain things that you can do much better than the computer at least for now...

- Pattern recognition such as recognizing one familiar face among a crowd!

Half a century ago, artificial-intelligence pioneer, Marvin Minsky of MIT predicted that computers would exceed human intelligence within a generation.



Marvin Minsky, 1927-2016

What is Neural Network?

Why do we use it?

There are certain things that you can do much better than the computer at least for now...

- Pattern recognition such as recognizing one familiar face among a crowd!

Half a century ago, artificial-intelligence pioneer, Marvin Minsky of MIT predicted that computers would exceed human intelligence within a generation.

Recently he admitted:

“The world’s most powerful computers lack the common sense of a toddler; they cannot even distinguish cats from dogs unless they are explicitly and painstakingly programmed to do so.”



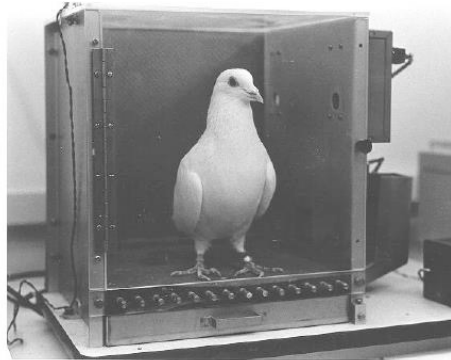
Marvin Minsky, 1927-2016

Brains of other animals

Pigeons are art experts (Watanabe et al., 1995)

Present paintings of two different artists (e.g., Chagall/Van Gogh)

Reward the pigeon for pecking when it is presented a particular artist (e.g., Van Gogh)



Vincent Willem van Gogh (1853-1890)
Dutch Post-Impressionist artist

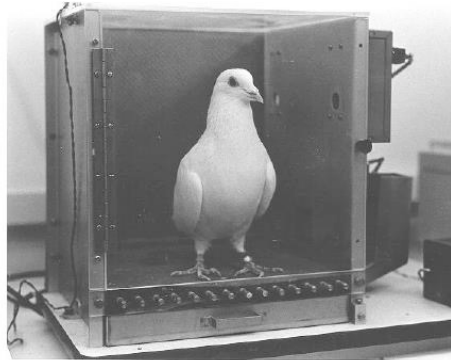
Marc Chagall (1887-1985)
Russian Jewish modernism artist

Brains of other animals

Pigeons are art experts (Watanabe et al., 1995)

Present paintings of two different artists (e.g., Chagall/Van Gogh)

Reward the pigeon for pecking when it is presented a particular artist (e.g., Van Gogh)



Vincent Willem van Gogh (1853-1890)
Dutch Post-Impressionist artist

Marc Chagall (1887-1985)
Russian Jewish modernism artist

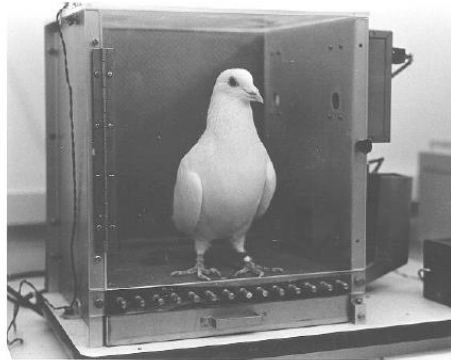
Pigeons could tell the difference between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)!

Brains of other animals

Pigeons are art experts (Watanabe et al., 1995)

Present paintings of two different artists (e.g., Chagall/Van Gogh)

Reward the pigeon for pecking when it is presented a particular artist (e.g., Van Gogh)



Vincent Willem van Gogh (1853-1890)
Dutch Post-Impressionist artist

Marc Chagall (1887-1985)
Russian Jewish modernism artist

Pigeons could tell the difference between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)!

Can pigeon recognize new paintings? The most remarkable thing is that it is still 85% successful!

What is a Neural Network (NN)?

A neural network is a **massively parallel distributed** processor made up of simple processing unit, which has a natural propensity for **storing experiential knowledge** and making it available for use.

massively parallel distributed

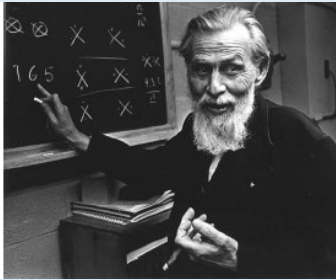
It employs a massive inter-connection of “simple” computing units - neurons. It is capable of organizing its structure consists of many neurons, to perform tasks that are many times faster than the fastest digital computers nowadays.

storing experiential knowledge

Knowledge is obtained from the data/input signals provided.
Knowledge is **learned!**

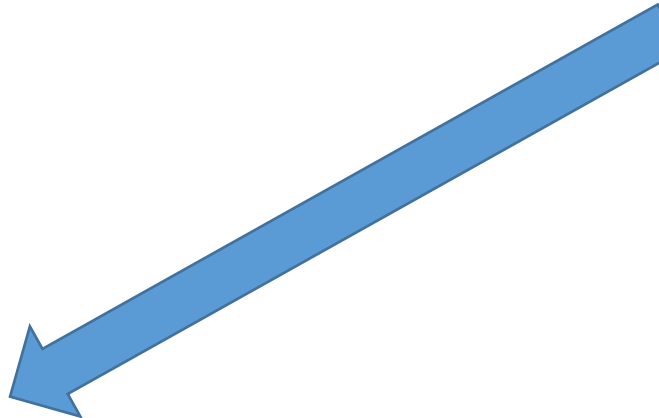
History of Neural Networks

McCulloch-Pitts Neurons, 1943



**Perceptron
1958
(built around the
McCulloch-Pitts model)**

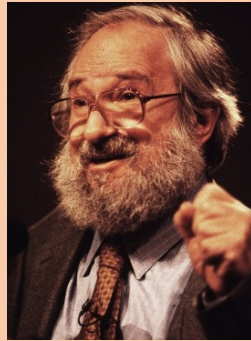
Frank Rosenblatt



Beginning of the Dark Age (1969-1982)...



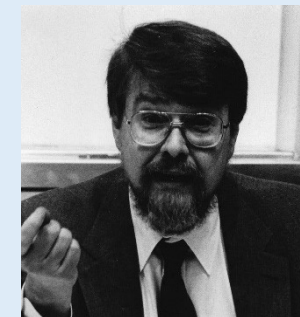
Marvin Minsky



Seymour Papert



Multi-layer Perceptron, Back Propagation Algorithm



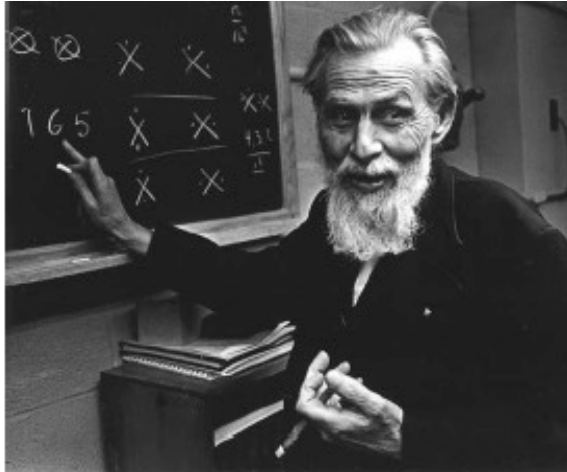
David Rumelhart



Paul Werbos

The beginning of the artificial neural networks

- In 1943 Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, published "A logical calculus of the ideas immanent in nervous activity" in the Bulletin of Mathematical Biophysics.
- In this paper they tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected together.
- These basic brain cells are called **neurons**.



Warren Sturgis McCulloch (1898-1969)

The McCulloch and Pitts model of a neuron has made an important contribution to the development of artificial neural networks -- which model key features of biological neurons



Walter Pitts (right) (1923-1969)

The next major step: Perceptron—single layer neural networks

Frank Rosenblatt, 1958

- Rosenblatt was best known for the Perceptron, an electronic device which was constructed in accordance with biological principles and showed an ability to learn.
- Perceptron was the first computer that could learn new skills by trial and error.
- The weights were initially random. Then it could alter them so that it could be taught to perform certain simple tasks in pattern recognition.
- Rosenblatt proved that for a certain class of problems, it could learn to behave correctly!



1928-1971, died in a boating accident

The dark age of neural networks: 1969-1982

Marvin Minsky and Seymour Papert, 1969

- They proved that the perceptron could not execute simple logic like “exclusive OR” (i.e., apples OR oranges, but not both).

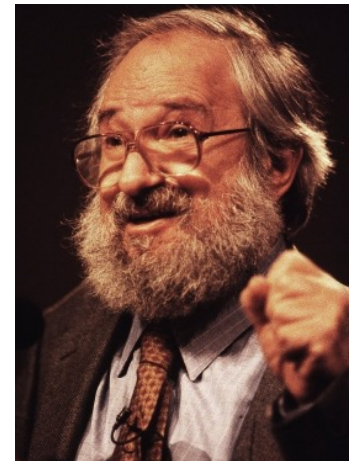
This killed interest in perceptron for over 10 years!

- Rosenblatt died in a boating accident in 1969.

Minsky and Papert would have contributed more if they had produced a solution to this problem rather than beating the perceptron to death.



Marvin Minsky



Seymour Papert

The book that attacks the Perceptron (and neural networks)

It cannot even compute the XOR function!

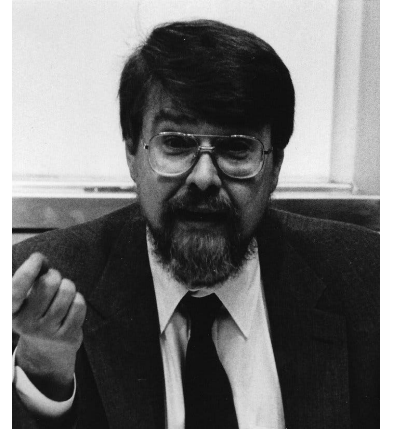


Multilayer Perceptron (MLP) and Back Propagation Algorithm

David Rumelhart and his colleagues, 1986

It was proved later that MLP is a universal approximator, which can approximate any continuous function. It can solve the XOR problem easily.

The Back Propagation Algorithm can be easily implemented to adjust the synaptic weights.

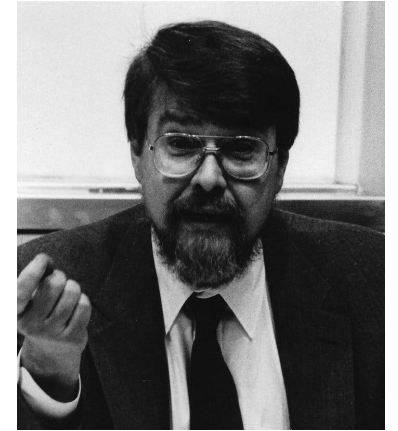


David Rumelhart

Multilayer Perceptron (MLP) and Back Propagation Algorithm

David Rumelhart and his colleagues, 1986

It was proved later that MLP is a universal approximator, which can approximate any continuous function. It can solve the XOR problem easily.



David Rumelhart

The Back Propagation Algorithm can be easily implemented to adjust the synaptic weights.



It turned out that the BP was described in the Ph.D. thesis of **Paul Werbos** in 1974 at Harvard University!

A lot of research since the late 1980's. NNs have now been used successfully in many areas and applications!

Applications of NNs

NNs are mainly used for solving many problems:

- Pattern Recognition (Pattern Classification)
- Regression (Data Fitting, Function Approximation)
- Image synthesis, human voice generation, ...

Can you fit the data without knowing the mathematical form of the models?

- YES! The neural networks can fit the data without any knowledge of the models.

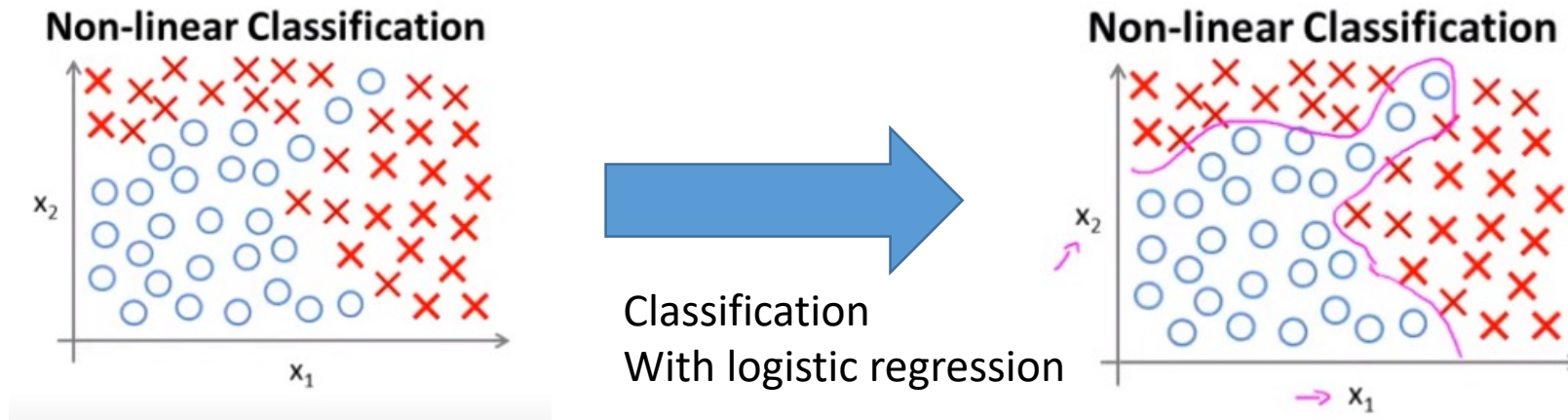
Many real-world examples at <http://www.ics.uci.edu/~mlearn/MLRepository.html>

Why do we need Neural Networks?

From logistic regression to Neural Networks...

Why Neural Networks?

- Why do we need NN? We already have linear and logistic regression...
- Today NN is the state-of-the-art technique for many real-life application.
- Let's motivate the discussion of NN for learning complex non-linear hypothesis.
- Consider the following classification problem:



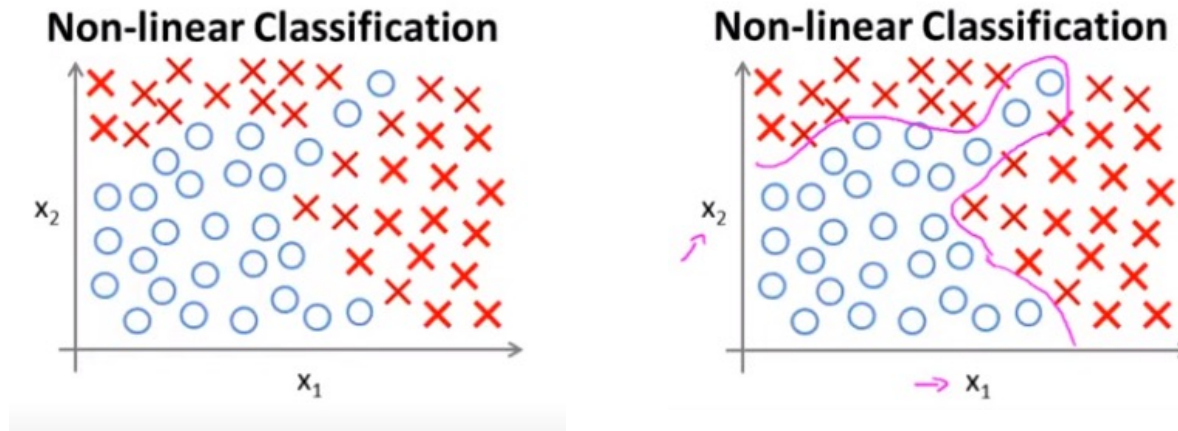
From Logistic Regression to NN

You can apply logistic regression with a lot of nonlinear features like this:

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \dots)$$

where $g()$ is the sigmoid function. If we include lot of polynomial terms, we can maybe find a hypothesis that classifies this data.

This method works well when you have 2 features, because you can then include all those polynomial terms of x_1 and x_2 .



What happens if we have more features, not only x_1 and x_2 ? In many ML problems, we have more features than 2!

From Logistic Regression to NN

For many interesting machine learning problems, we have more than two features.

For example: Housing classification problem

You want to predict the odds of a house would be sold within the 6 months.

We may have hundreds of features:



x_1 = size,
 x_2 = age,
 x_3 = number of floors,
 x_4 = number of rooms,
...
 x_{100}



From Logistic Regression to NN



You want to predict the odds of a house would be sold within the 6 months. We may have hundreds of features:

x_1 = size,
 x_2 = age,
 x_3 = number of floors,
 x_4 = number of rooms,
...
 x_{100}



$x_1^2, x_1x_2, x_1x_3, \dots$
 x_2^2, x_2x_3, \dots
.....
 x_{100}^2, \dots

If you include just the second order polynomial terms, we'll have about 5000 features for $n=100$... You will end up overfitting, and it is computationally expensive to be working with many features.

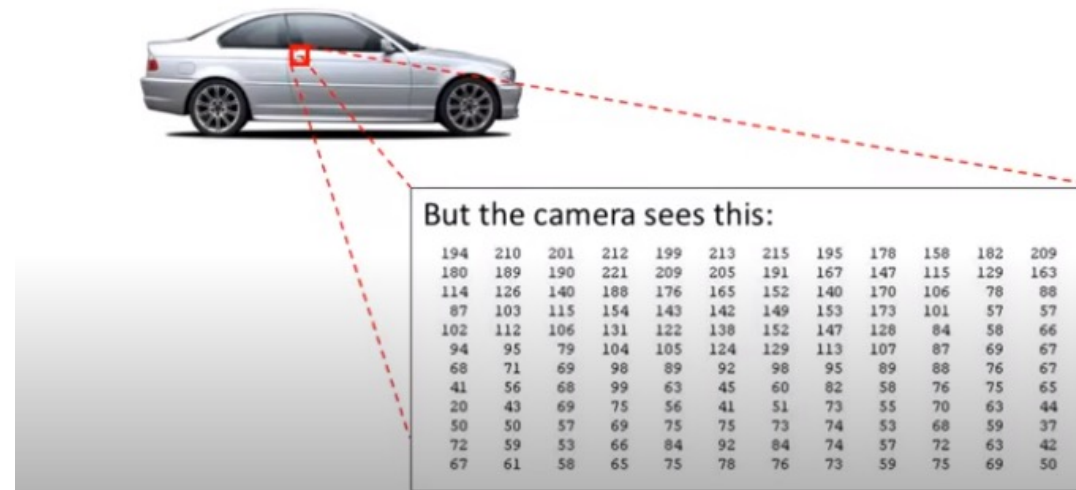
One solution: One way is to include a subset of these features... For example: include only quadratic features ($x_1^2, x_2^2, \dots, x_{100}^2$)... This is not enough features to fit a complex data!

If we also include 3rd order polynomials $x_1x_2x_3, x_4x_5x_6$, we may end up with ~170,000 features. It doesn't seem a good way when n is large...

For many machine learning applications, n will be very large...

Why computer vision (CV) is hard?

You want to use ML to train a classifier to examine the image, and tell us if it is a car.
Let's zoom into the red rectangle. What we see is a car, but what camera sees is...



Matrix: pixel intensity values.

CV problem: looks at the matrix of intensity values, and tell us it is door handle of the car.

An example:

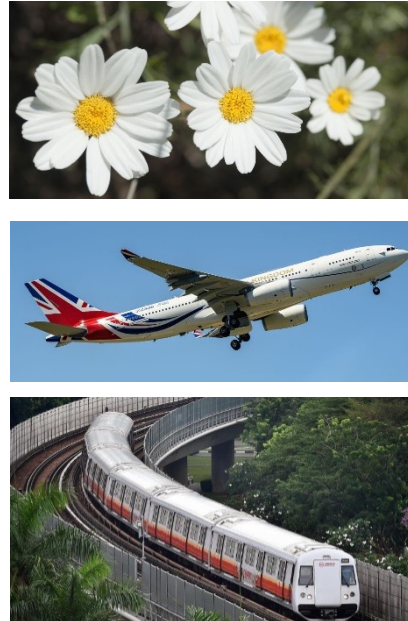
How do we train a network for car detection?

Training

CAR



NOT A CAR



Testing

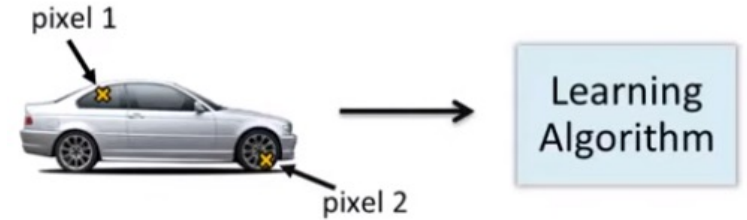


Is this a car?

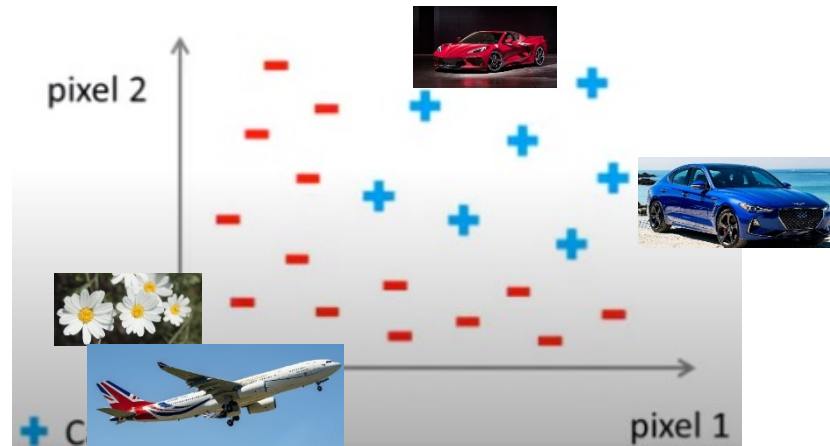
For many machine learning algorithms, n will be very large!

An example – non-linear hypothesis

Training car detection with NN:



Cars and non-cars end up in different places in space.



We need a nonlinear hypothesis that separates 2 classes.

Here we only pick 2 pixel locations. What is the feature dimension for this problem in real-life?

An example – non-linear hypothesis

Training car detection with NN:



What if our images are 50*50 pixel \rightarrow $n=2500$ pixels (for each image, grey scale)

If we want to learn a non-linear hypothesis function with quadratic features such as x_1x_2 , we may end up with **3 million features!**

Too large to be reasonable. So, logistic regression together with adding quadratic and cubic features, is not a good way to learn complex nonlinear hypothesis, **especially when n is large.** Because we end up with too many features.

NN is a much better way to learn complex nonlinear hypothesis even when n is large.

Neural Networks

Neural Networks – Neuron and Brain

Neural Networks are pretty old algorithms and originally motivated by having machines that can mimic brain.

- We can do so many amazing things: we can see, hear, do math. Our brain is the most amazing machine!
- Can we perform everything through programming?

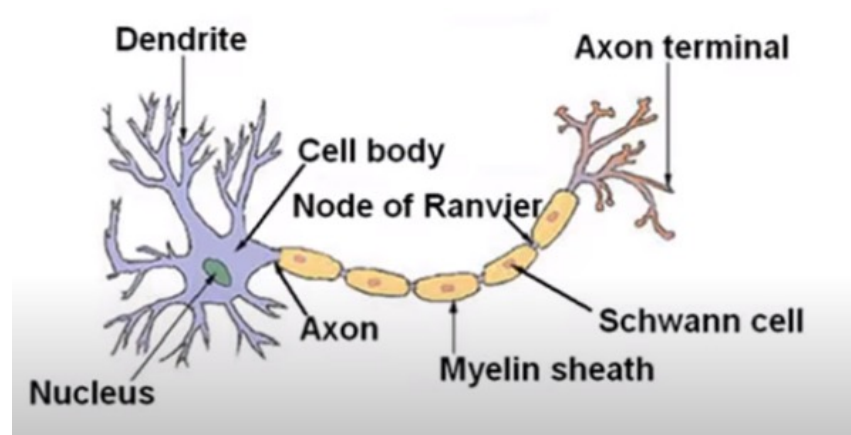
Big AI dream: someday building truly intelligent machines

- NNs are computationally expensive algorithms. Computers recently became fast enough to run large scale NN. This is one the reasons why NN became popular recently.

Neurons in the Brain

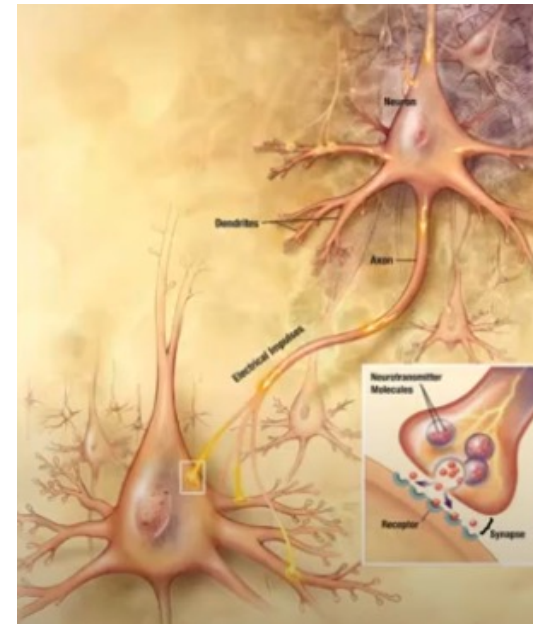
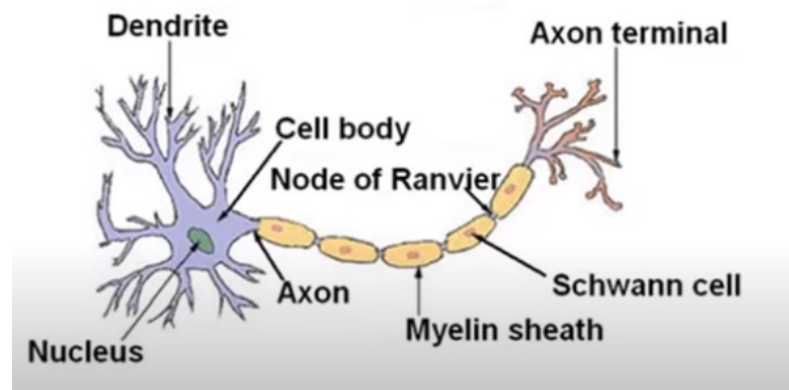
- Neural networks is developed simulating networks of neurons. We will first study how a single neuron looks like.
- Basically neuron is a computational unit that gets a number of inputs, does some computations and send outputs to other neurons through axon.

A single neuron in the brain looks like:



Neurons in the Brain

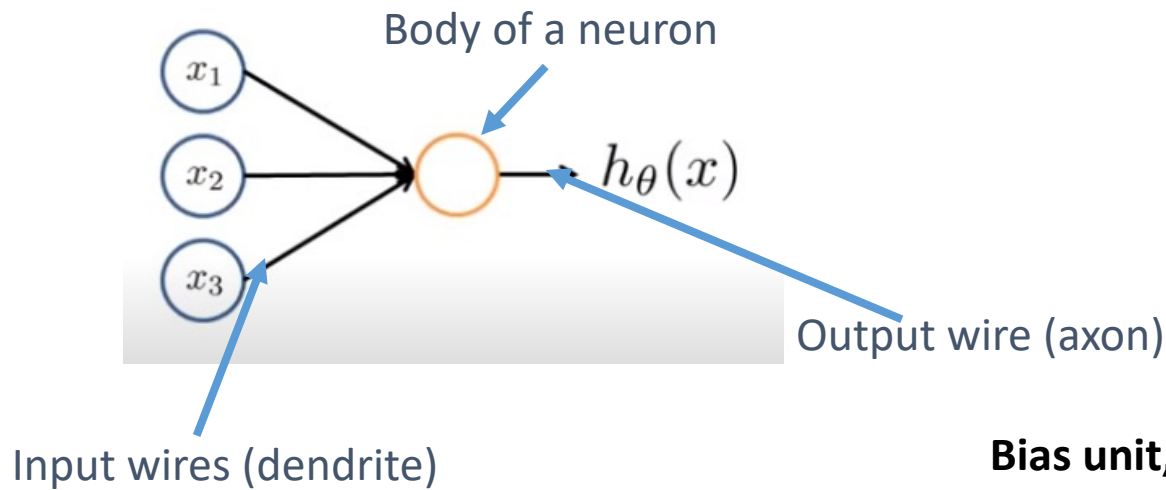
- Communication between neurons is with little pulses of electricity.
- If the neuron wants to send a message, it sends an electricity through its axon.
- The receiver gets it from its dendrite, and perform computation, and send to other neurons by their axon.



Neuron Model

Logistic Unit

We will model a neuron as a logistic unit.



$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Parameters or weights (NN)

$$\theta \in \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix}$$

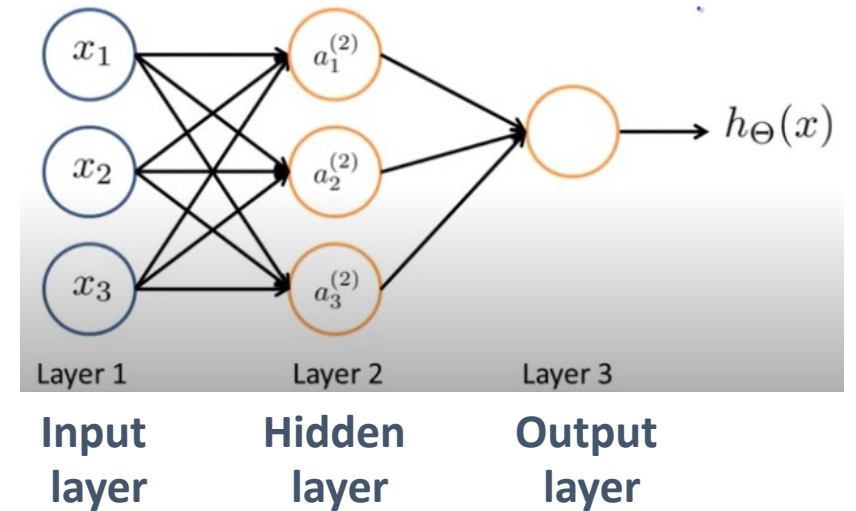
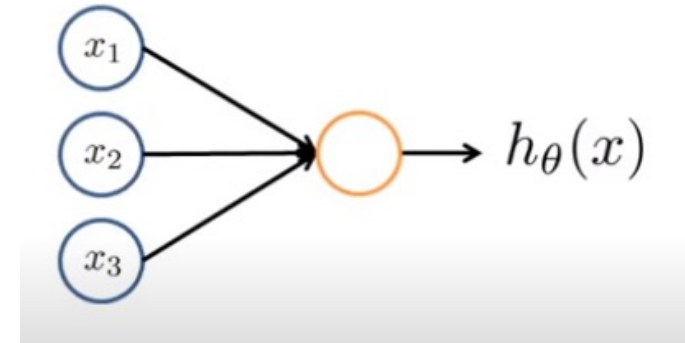
Bias unit, always 1

$$X \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}$$

"This is a very simple artificial neuron with sigmoid/logistic activation function."

Neural Network

- This figure represents a single neuron:
- Many neurons come together \rightarrow NN
 - Input units are x_1, x_2, x_3
 - Neurons in 2nd layer: $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$
 - First layer is called input layer
 - Final layer is called output layer (outputs the final value)



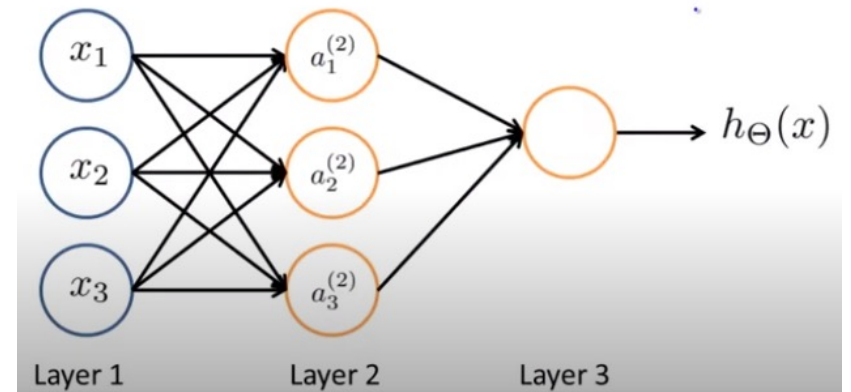
In supervised learning, we don't get to see the parameters of hidden layers. We see only inputs and output. Anything which is not input layer and output layer is considered hidden layer.

Neural Networks – Mathematical Representation

A mathematical representation of how we describe the hypothesis used by neural networks.

$a_i^{(j)}$: “activation” of unit i in layer j where activation means the value calculated as an output of neuron

$\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j + 1$



Neural Networks – Mathematical Representation

A mathematical representation of how we describe the hypothesis used by neural networks.

$a_i^{(j)}$: “activation” of unit i in layer j where activation means the value calculated as an output of neuron

$\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

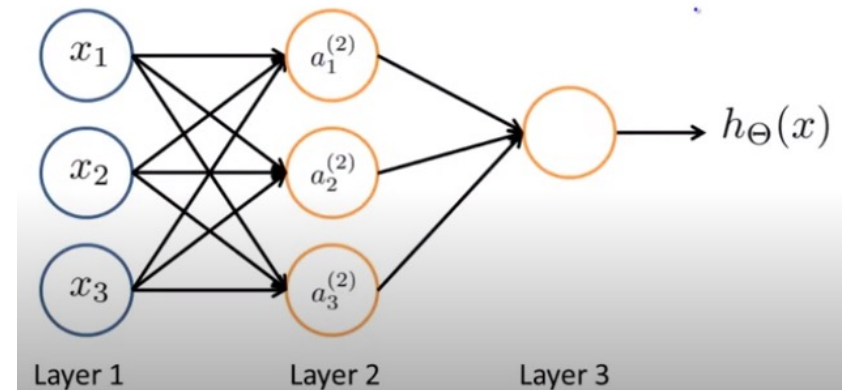
We now know what the 3 hidden unit computes. Let's look at the output layer:

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j , and s_{j+1} units in layer $j + 1$, then the dimension of $\Theta^{(j)}$ will be $s_{j+1} * (s_j + 1)$.

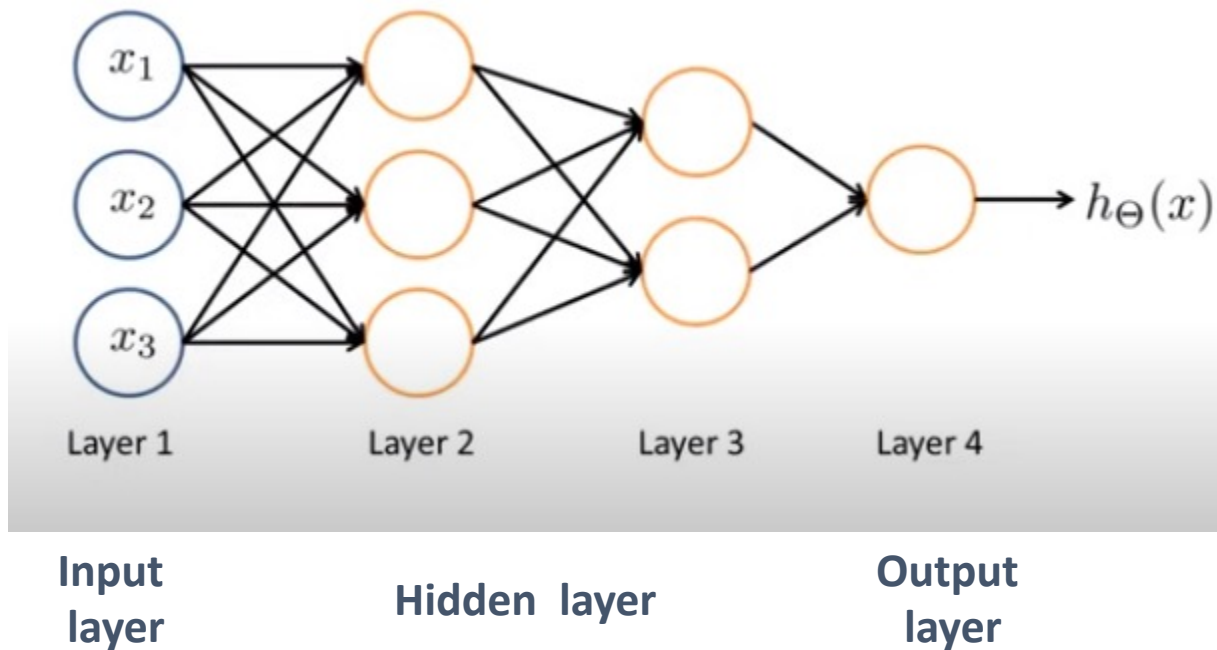
$$\Theta^{(1)} = 3 * 4 \text{ matrix}$$

We'll later study how to perform computation efficiently, and why these NN architectures are good and useful...



Other Network Architectures

Architecture: How the different neurons are connected to each other... An example of a different network architecture is given below:



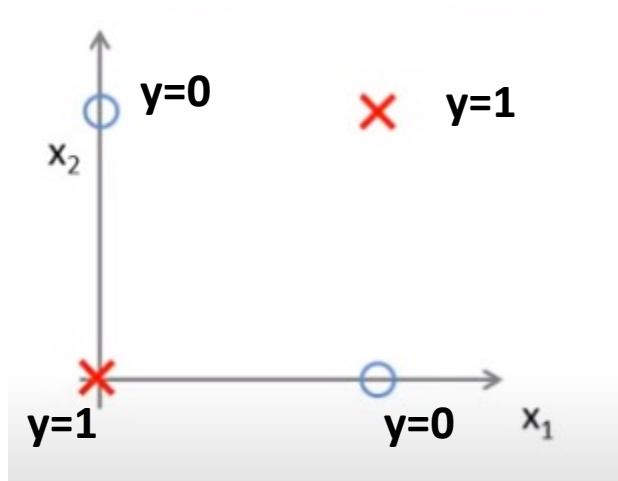
Neural Networks - Examples

We studied the mathematical representation of NNs.

We'll now study how a NN can learn complex nonlinear hypothesis...

Can you design a neural network that can compute the XNOR function?

Non-linear classification



We want to learn a nonlinear decision boundary that separates these 2 classes.

Remember: XNOR is $NOT(x XOR y)$

Before solving this problem, I would like to solve 3 simple problems. 😊

Neural Networks Examples and Intuitions

AND Function: Can we show a network that fits the AND function?

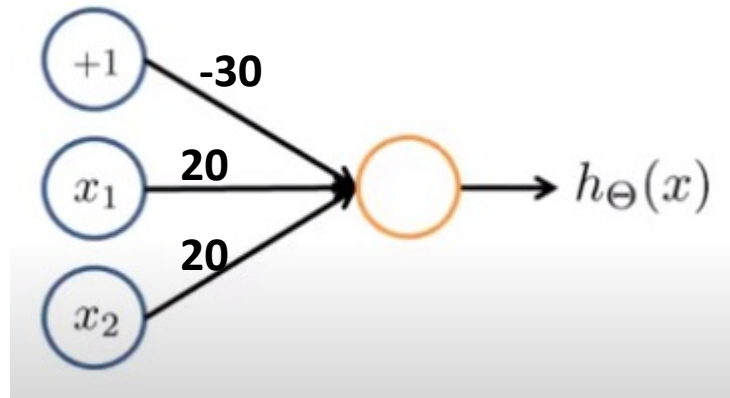
x_1, x_2 are binary: 0 or 1

$$y = x_1 \text{ AND } x_2$$

Let's assume the parameters are: $-30, 20, 20$

$$h(x) = g(-30 + 20x_1 + 20x_2)$$

Can we get one unit network that solves AND?



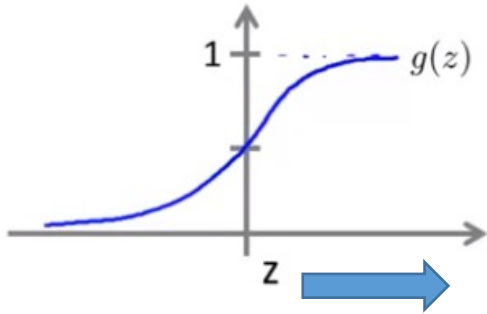
Neural Networks Examples and Intuitions

AND Function :

$$h(x) = g(-30 + 20x_1 + 20x_2)$$

z

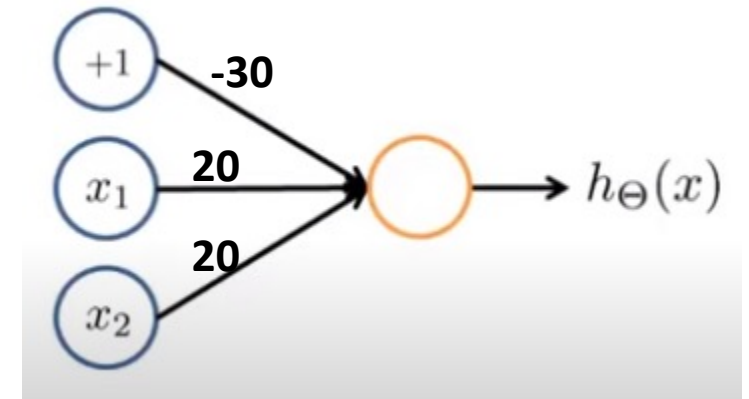
Remember: Logistic Regression



$g(z) \geq 0.5$ predict $y = 1$

$g(z) < 0.5$ predict $y = 0$

Can we get one unit network that solves AND?



x_1	x_2	AND
0	0	$g(-30) = 0$
0	1	$g(-10) = 0$
1	0	$g(-10) = 0$
1	1	$g(10) = 1$

$y = x_1 \text{ AND } x_2$

Neural Networks Examples and Intuitions

OR Function: Can we show a network that fits the OR function?

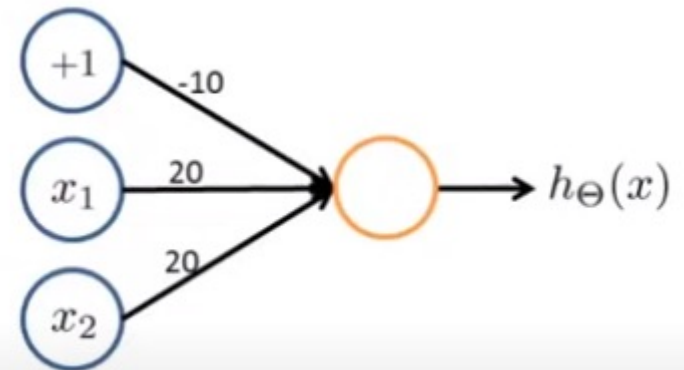
x_1, x_2 are binary, 0 or 1

$$y = x_1 \text{ OR } x_2$$

Let's assume the parameters are: $-10, 20, 20$

$$h(x) = g(-10 + 20x_1 + 20x_2)$$

Can we get one unit network that solves OR?



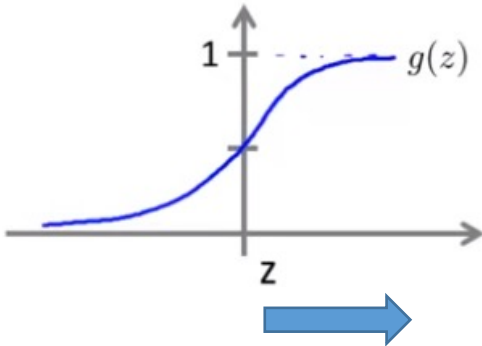
Neural Networks Examples and Intuitions

OR Function:

$$h(x) = g(-10 + 20x_1 + 20x_2)$$

z

Remember: Logistic Regression



$g(z) \geq 0.5$ predict $y = 1$

$g(z) < 0.5$ predict $y = 0$

Can we get one unit network that solves OR?

x_1	x_2	OR
0	0	$g(-10) = 0$
0	1	$g(10) = 1$
1	0	$g(10) = 1$
1	1	$g(30) = 1$

$$y = x_1 \text{ OR } x_2$$

Neural Networks Examples and Intuitions

Negation: (not x_1)

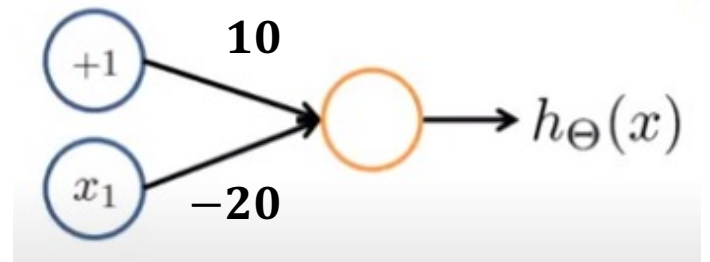
We have only one input: x_1 which is binary, 0 or 1

$$y = \text{NOT } x_1$$

Let's assume the parameters are: 10, -20

$$h(x) = g(10 - 20x_1)$$

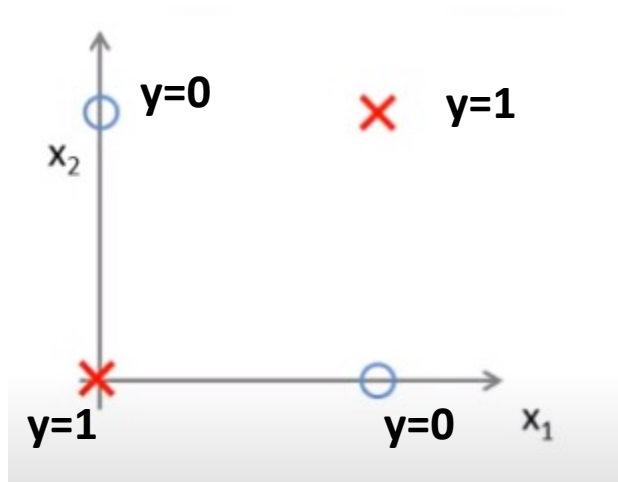
Can we get one unit network that solves negation?



x_1	Negation
0	$g(10) = 1$
1	$g(-10) = 0$

Can you design a neural network that can compute the XNOR function?

Non-linear classification



Remember: XNOR is $NOT(x XOR y)$

Let's put everything together!

Neural Networks Examples and Intuitions

PUTTING TOGETHER FOR XNOR

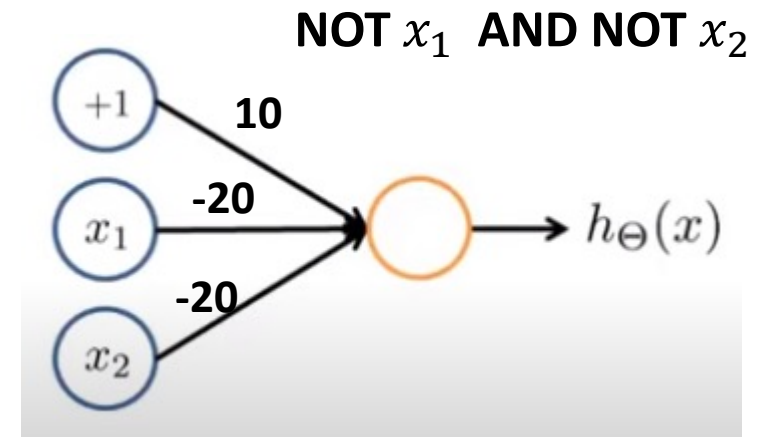
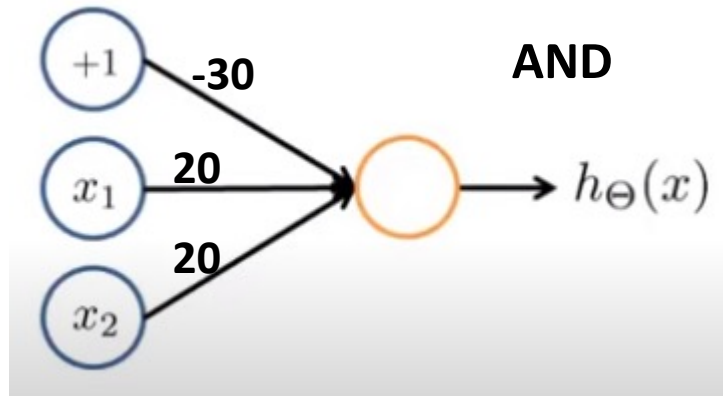
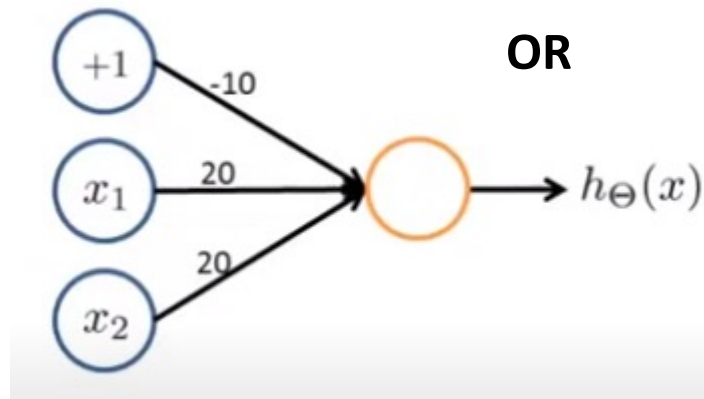
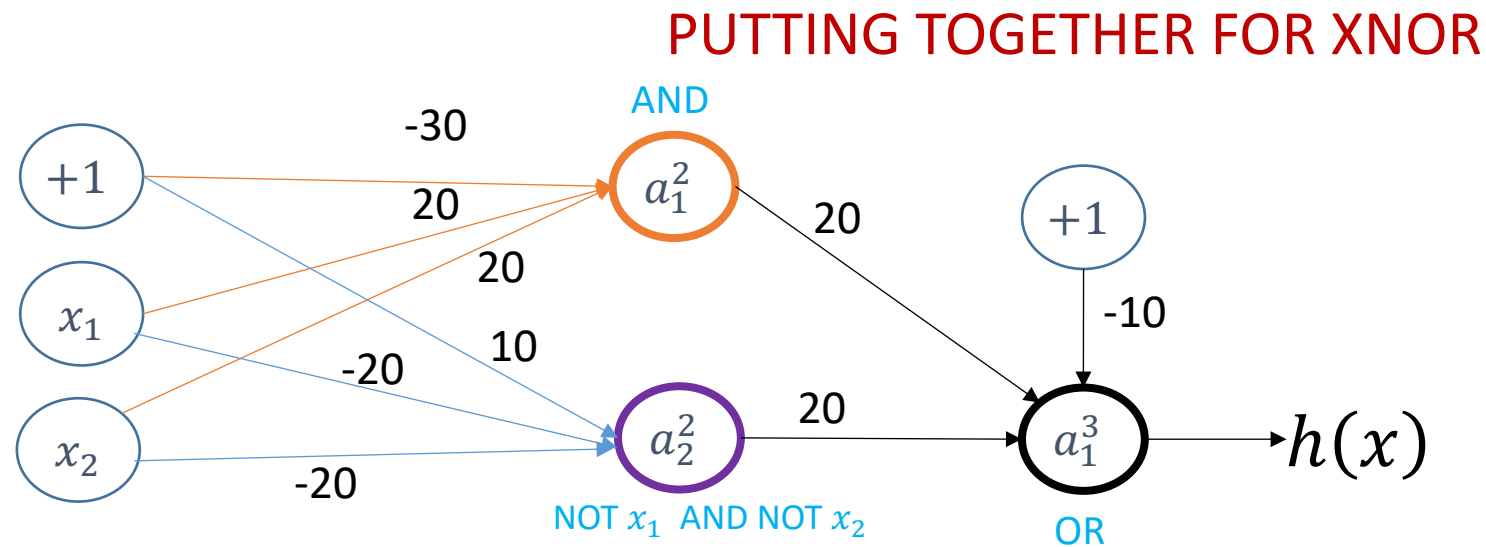


Figure out yourself 😊



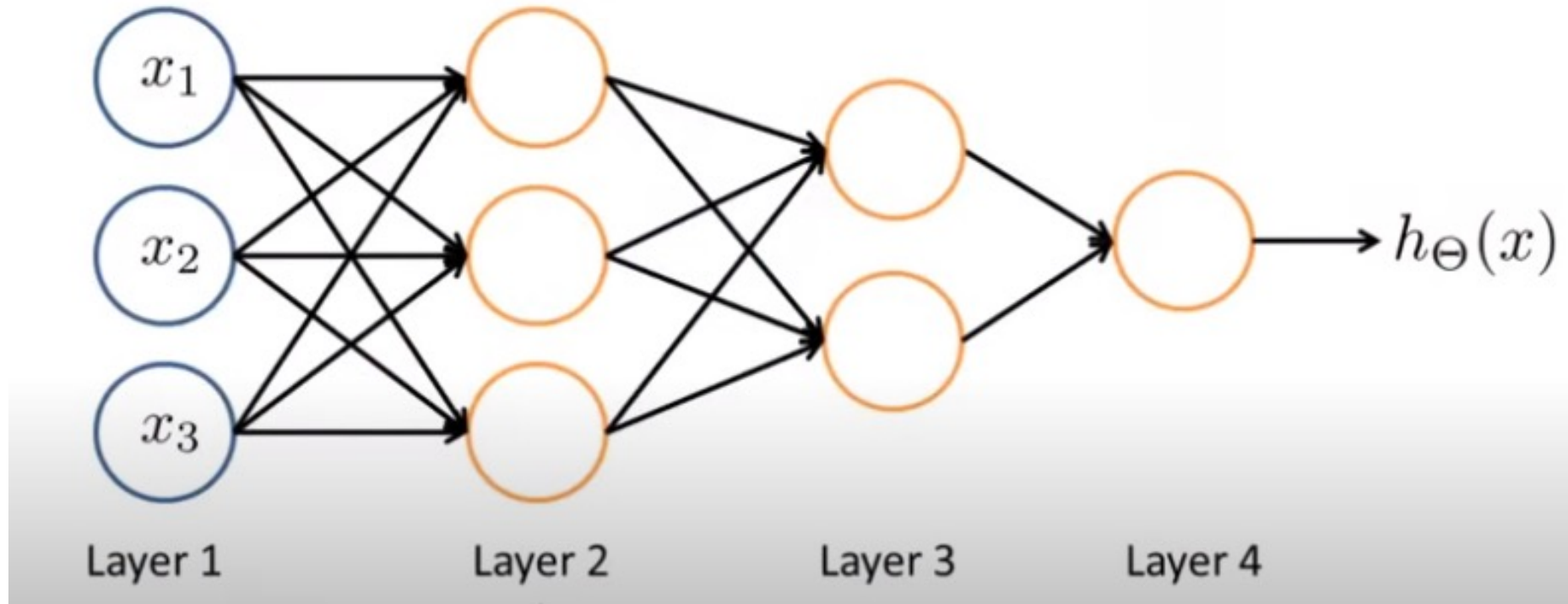
Neural Networks Examples and Intuitions



x_1	x_2	a_1^2	a_2^2	$h(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

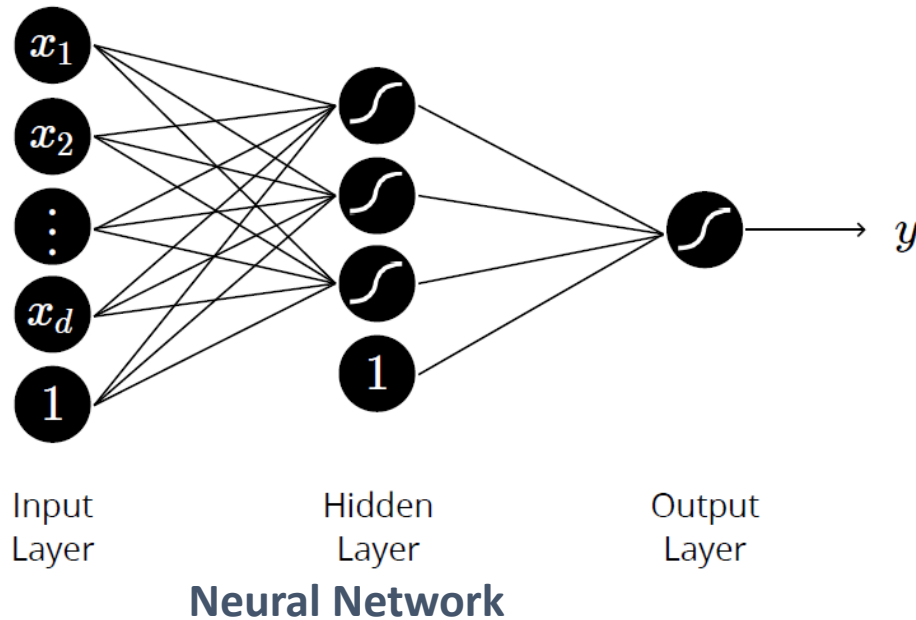
XNOR

Neural Network Intuition

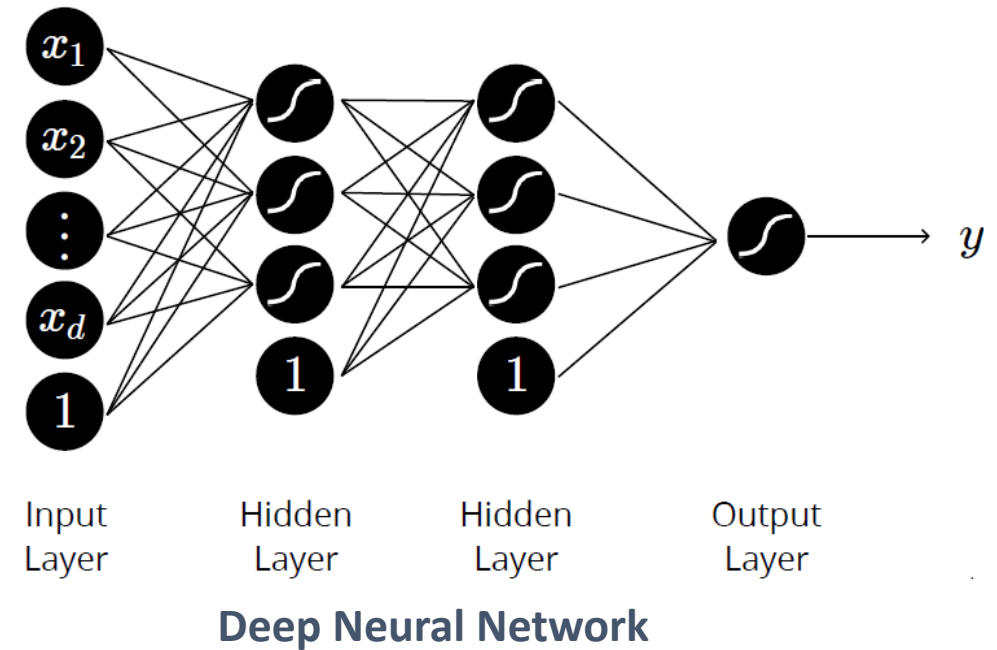


When you have multiple layers, the deeper layers computes more complex functions...

Neural Network vs Deep Neural Networks



A shallow neural network has three layers of neurons that process inputs and generate outputs.



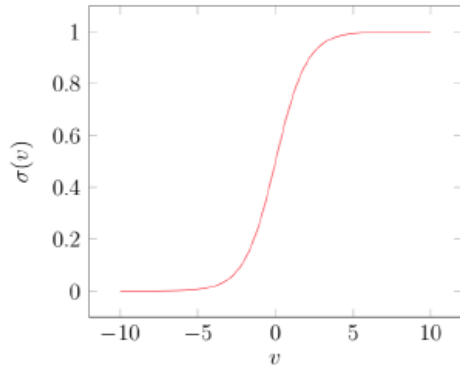
A Deep Neural Network (DNN) has two or more “hidden layers” of neurons that process inputs.

According to Goodfellow, Bengio and Courville, and other experts, while shallow neural networks can tackle equally complex problems, deep learning networks are more accurate and improve in accuracy as more neuron layers are added.

Neural networks can have different activation functions

Logistic/sigmoid function:

$$y = \sigma(\sum_{i=1}^d \theta_i \cdot x_i + \theta_0)$$



$$\sigma(x) = \frac{e^x}{1+e^x}$$



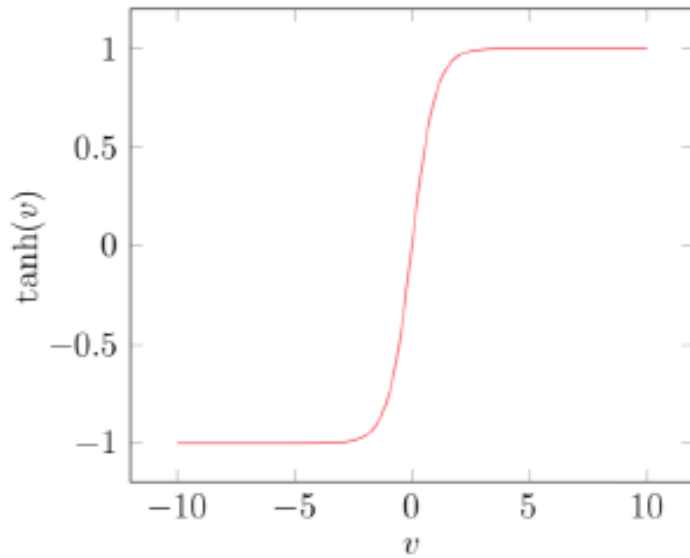
- Smooth gradient, preventing “jumps” in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.



- Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

Neural networks can have different activation functions

TanH / Hyperbolic Tangent function:



$$\tanh(v) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- Otherwise like the Sigmoid function.

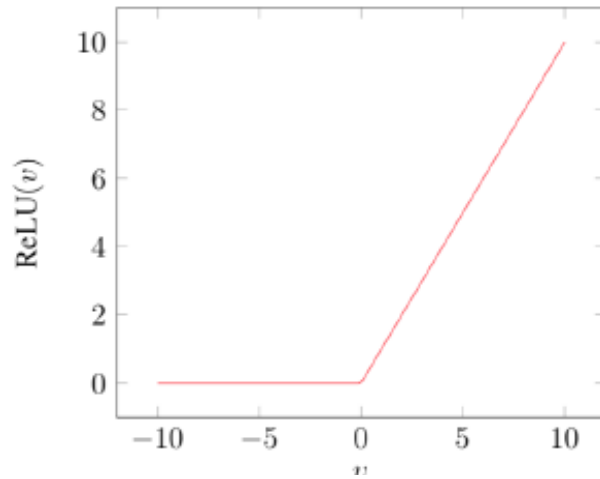


- Like the Sigmoid function

$$y = \tanh\left(\sum_{i=1}^d \theta_i \cdot x_i + \theta_0\right)$$

Neural networks can have different activation functions

ReLU (Rectified Linear Unit) function:



- Computationally efficient—allows the network to converge very quickly!



- The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

$$y = \text{ReLU}\left(\sum_{i=1}^d \theta_i \cdot x_i + \theta_0\right) \quad \text{ReLU}(v) = \max(0, x)$$

ReLu VS TanH neurons

A very famous NN paper published in NIPS 2012!

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca	Ilya Sutskever University of Toronto ilya@cs.utoronto.ca	Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca
--	---	---

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully connected

3.1 ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the

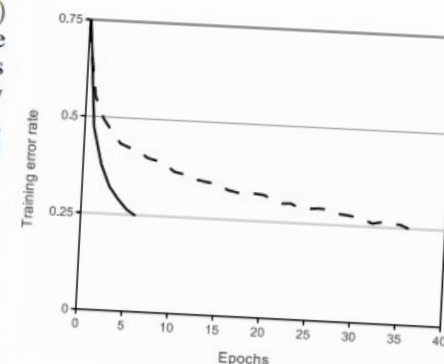
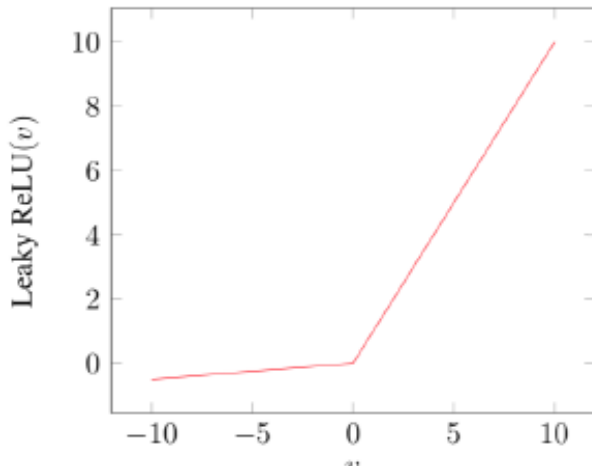


Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Neural networks can have different activation functions

Leaky ReLU function:



- Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- Otherwise like ReLU



- Results not consistent—leaky ReLU does not provide consistent predictions for negative input values.

For RELU-based activation function comparison:

Some interesting papers

Under review as a conference paper at ICLR 2016

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec, luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Empirical Evaluation of Rectified Activations in Convolution Network

Bing Xu
University of Alberta

ANTINUCLEON@GMAIL.COM

Naiyan Wang
Hong Kong University of Science and Technology

WINSTY@GMAIL.COM

Tianqi Chen
University of Washington

TQCHEN@CS.WASHINGTON.EDU

Mu Li
Carnegie Mellon University

MULI@CS.CMU.EDU

Abstract

In this paper we investigate the performance of different types of rectified activation functions in convolutional neural network: standard rectified linear unit (ReLU), leaky rectified linear unit (Leaky ReLU), parametric rectified linear unit (PReLU) and a new ran-

et al., 2014), object detection(Girshick et al., 2014) and tracking(Wang et al., 2015). Despite its depth, one of the key characteristics of modern deep learning system is to use non-saturated activation function (e.g. ReLU) to replace its saturated counterpart (e.g. sigmoid, tanh). The advantage of using non-saturated activation function lies in two aspects: The first is to solve the so called “exploding/vanishing gradient”.

Neural networks can have different activation functions

Other activation functions include:

- Softmax,
- Parametric ReLU,
- Swish,
- Exponential Linear Units (ELU),
-

→ Active research area!

The first paper that describes an effective way of training a deep neural network

A fast learning algorithm for deep belief nets *

Geoffrey E. Hinton and Simon Osindero

Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
{hinton, osindero}@cs.toronto.edu

Yee-Whye Teh

Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg



Abstract

We show how to use “complementary priors” to eliminate the explaining away effects that make inference difficult in densely-connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers

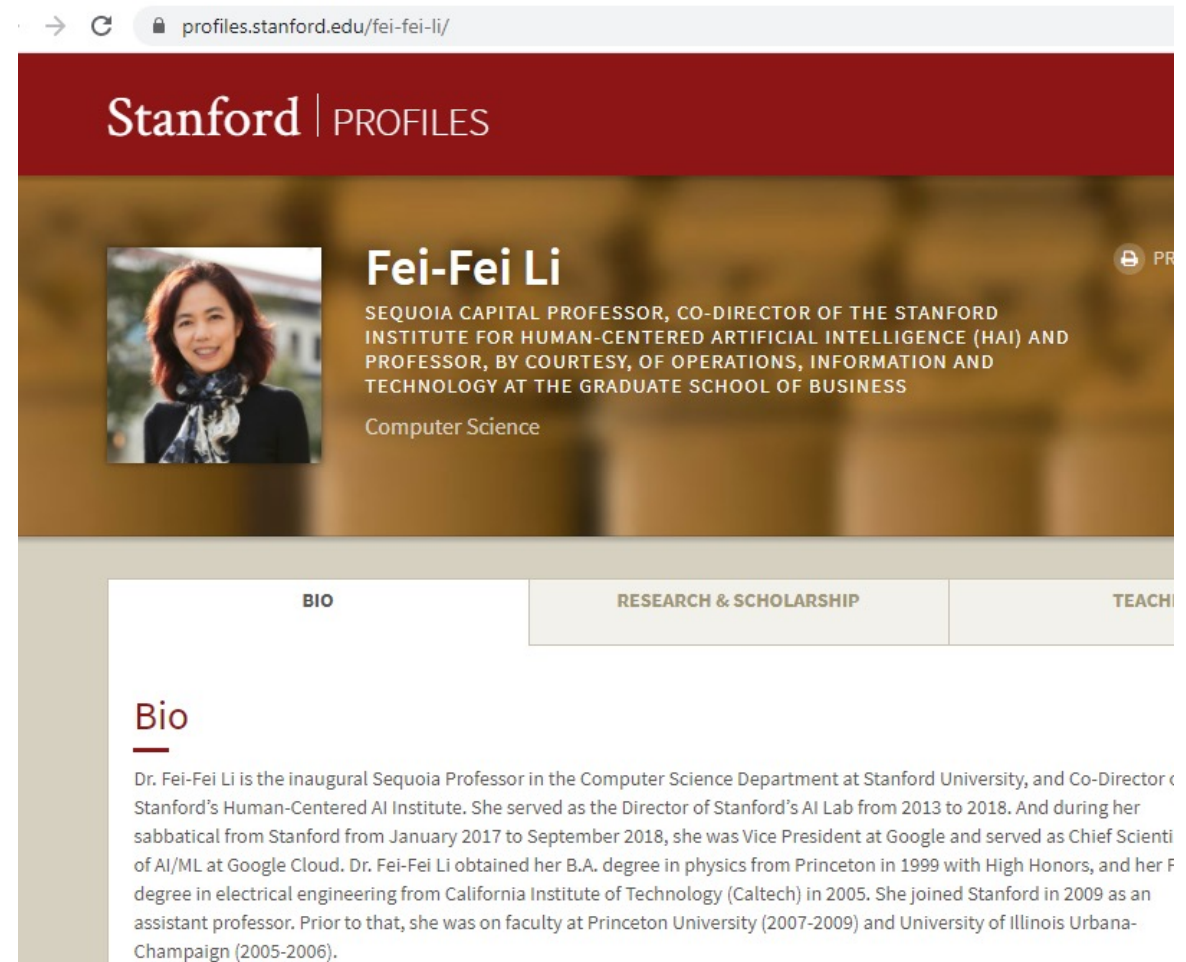
remaining hidden layers form a directed acyclic graph that converts the representations in the associative memory into observable variables such as the pixels of an image. This hybrid model has some attractive features:

1. There is a fast, greedy learning algorithm that can find a fairly good set of parameters quickly, even in deep networks with millions of parameters and many hidden layers.

<https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf>


Computer Vision & Neural Networks

[TED Talk by Prof. Fei-Fei Li](#)



The screenshot shows a web browser window with the URL `profiles.stanford.edu/fei-fei-li/`. The page features a red header with the Stanford logo and the word "PROFILES". Below the header is a large banner image of Prof. Fei-Fei Li. To the right of her photo, her name "Fei-Fei Li" is displayed in large white text, followed by her titles: "SEQUOIA CAPITAL PROFESSOR, CO-DIRECTOR OF THE STANFORD INSTITUTE FOR HUMAN-CENTERED ARTIFICIAL INTELLIGENCE (HAI) AND PROFESSOR, BY COURTESY, OF OPERATIONS, INFORMATION AND TECHNOLOGY AT THE GRADUATE SCHOOL OF BUSINESS". Below this, "Computer Science" is listed. At the bottom of the page, there are three tabs: "BIO", "RESEARCH & SCHOLARSHIP", and "TEACHING". The "BIO" tab is currently selected, showing a section titled "Bio" with a red underline. The text under "Bio" provides a detailed biography of Dr. Fei-Fei Li, mentioning her roles at Stanford, Google, and her educational background at Princeton and Caltech.

Stanford | PROFILES

 **Fei-Fei Li**

SEQUOIA CAPITAL PROFESSOR, CO-DIRECTOR OF THE STANFORD INSTITUTE FOR HUMAN-CENTERED ARTIFICIAL INTELLIGENCE (HAI) AND PROFESSOR, BY COURTESY, OF OPERATIONS, INFORMATION AND TECHNOLOGY AT THE GRADUATE SCHOOL OF BUSINESS

Computer Science

BIO RESEARCH & SCHOLARSHIP TEACHING

Bio

Dr. Fei-Fei Li is the inaugural Sequoia Professor in the Computer Science Department at Stanford University, and Co-Director of Stanford's Human-Centered AI Institute. She served as the Director of Stanford's AI Lab from 2013 to 2018. And during her sabbatical from Stanford from January 2017 to September 2018, she was Vice President at Google and served as Chief Scientist of AI/ML at Google Cloud. Dr. Fei-Fei Li obtained her B.A. degree in physics from Princeton in 1999 with High Honors, and her Ph.D. degree in electrical engineering from California Institute of Technology (Caltech) in 2005. She joined Stanford in 2009 as an assistant professor. Prior to that, she was on faculty at Princeton University (2007-2009) and University of Illinois Urbana-Champaign (2005-2006).

Neural Networks Learning

How to fit the parameters?

Neural Networks – A classification problem

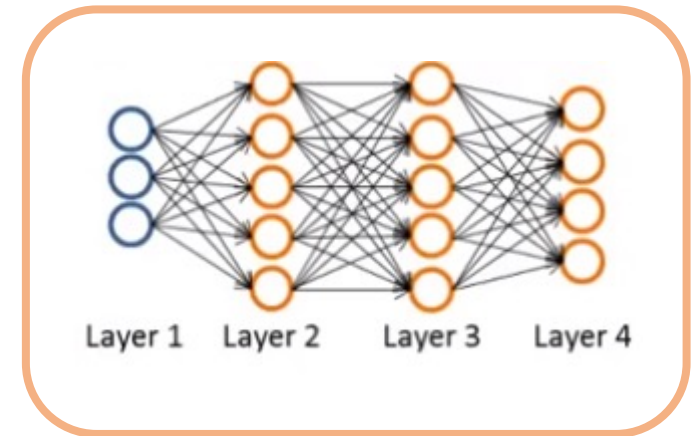
- Let's talk about the cost function for fitting the parameters.

Training data: $\{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$

Total number of layers: $L, L = 4$

s_L : number of units (excluding bias in layer l)

For example: $s_1 = 3, s_2 = 5, s_3 = 5, s_4 = 4$



Binary Classification

$y = 0 \text{ or } 1$
if we have binary
classification, we would
have **1 output unit**.

Multi-class Classification (K classes)

y can be $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$

K output units,
Output is a K dimensional vector

Neural Networks – Cost Function

The cost function will be a generalized version of logistic regression's cost function (*previously studied*).

For 1 logistic regression, we minimize $J(\theta)$

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

Neural Networks – Cost Function

The cost function will be a generalized version of logistic regression's cost function (*previously studied*).

For 1 logistic regression, we minimize $J(\theta)$

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

For NN, instead of having one logistic regression unit, we'll instead have K of them.

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log (h_{\theta}(x^{(i)})_k) + (1 - y_k^{(i)}) \log (1 - h_{\theta}(x^{(i)})_k) \right]$$

If we have 4 output units, then K will be equal to 4.

For binary classification, $K=1$.

Basically logistic unit cost function, but summing the cost function for the last 4 units.

Neural Networks Learning – BIG PICTURE

Now we will try to minimize this cost function.

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - h_{\theta}(x^{(i)}))_k \right]$$

$\min J(\theta)$ w.r.t θ

In order to use either gradient descent (or an advance optimization algorithm), we need to compute:

$$J(\theta),$$
$$\frac{\partial}{\partial (\theta_{ij}^{(l)})} J(\theta) \rightarrow \text{how to compute these partial derivatives?}$$

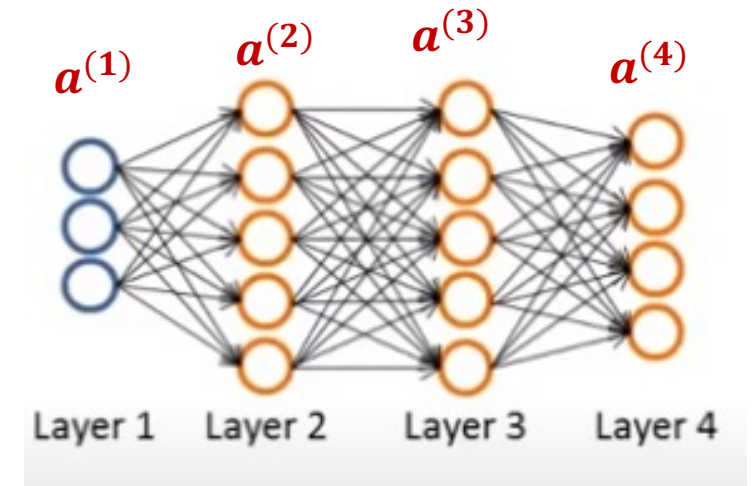
(Forward & backpropagation algorithm)

Neural Networks Learning

Forward propagation

- Let's assume we only have a **pair of training data** (x, y) . We'll first calculate the forward propagation in order to compute what a hypothesis outputs, given this input.
- $a^{(1)}$ is the activation value of the first layer.

$$\begin{aligned}a^{(1)} &= x \\z^{(2)} &= \theta^{(1)} a^{(1)} \\a^{(2)} &= g(z^{(2)}) \\z^{(3)} &= \theta^{(2)} a^{(2)} \\a^{(3)} &= g(z^{(3)}) \\z^{(4)} &= \theta^{(3)} a^{(3)} \\a^{(4)} &= g(z^{(4)}) = h_{\theta}(x)\end{aligned}$$



We can compute all the activation values of the neurons in our network.

In order to compute derivatives, we'll use an algorithm called backpropagation.

Neural Networks Learning - Backpropagation Algorithm

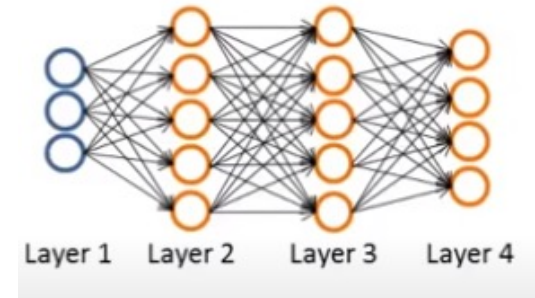
For each node, we'll compute: $\delta_j^{(l)}$ = '*error*' of node j in layer l .

For the 4th layer: $\delta_j^{(4)} = a_j^{(4)} - y_j$ (the difference between what the activation (hypothesis output) is and what the label actually is)

Vectorized implementation:

How to compute the δ terms of earlier layers:

$\delta^{(4)} = a^{(4)} - y$ (dimension is equal to the number of output units in network)



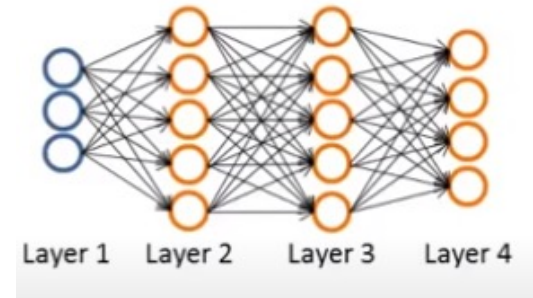
Neural Networks Learning - Backpropagation Algorithm

For each node, we'll compute: $\delta_j^{(l)}$ = '*error*' of node j in layer l .

For the 4th layer: $\delta_j^{(4)} = a_j^{(4)} - y_j$ (the difference between what the activation (hypothesis output) is and what the label actually is)

Vectorized implementation:

How to compute the δ terms of earlier layers:



$\delta^{(4)} = a^{(4)} - y$ (dimension is equal to the number of output units in network)

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

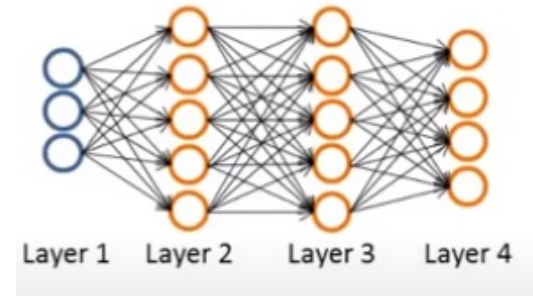
Neural Networks Learning - Backpropagation Algorithm

For each node, we'll compute: $\delta_j^{(l)}$ = '*error*' of node j in layer l .

For the 4th layer: $\delta_j^{(4)} = a_j^{(4)} - y_j$ (the difference between what the activation (hypothesis output) is and what the label actually is)

Vectorized implementation:

How to compute the δ terms of earlier layers:



$\delta^{(4)} = a^{(4)} - y$ (dimension is equal to the number of output units in network)

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

How about $\delta^{(1)}$?

Neural Networks Learning - Backpropagation Algorithm

$$\begin{aligned}\delta^{(4)} &= a^{(4)} - y \\ \delta^{(3)} &= (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \\ \delta^{(2)} &= (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})\end{aligned}$$

There is no $\delta^{(1)}$ term because the first layer is the input layer, so no error is associated there.

Back propagation term: we start calculating $\delta^{(4)}$, then go back and calculate $\delta^{(3)}$, $\delta^{(2)}$. We are **back propagating** the errors from the output layer to hidden layers.

$$\frac{\partial}{\partial(\theta_{ij}^{(l)})} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

By performing backpropagation and updating delta terms, you can easily and quickly compute these partial derivatives. :)

Backpropagation Algorithm Implementation

Suppose that we have a training set of m examples. Training data: $\{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$
 $\Delta_{ij}^{(l)} = 0$ for all (i, j, l) (for computing $\frac{\partial}{\partial(\theta_{ij}^{(l)})} J(\theta)$, they will act as an accumulator)

For $i=1$ to m

$$\mathbf{a}^{(1)} = \mathbf{x}^{(i)}$$

forward propagation to compute all the activations for all layers.

compute all $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}^{(i)}$, and then $(\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)})$ using backpropagation

update $\Delta_{ij}^{(l)} \rightarrow \Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \mathbf{a}_j^{(l)} \delta_i^{(l+1)}$ (to accumulate the partial derivatives terms)

end

$$\mathbf{D}_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \Rightarrow \quad \boxed{\frac{\partial}{\partial(\theta_{ij}^{(l)})} J(\theta) = \mathbf{D}_{ij}^{(l)}}$$

Gradient Descent algorithm for optimization.

BIG PICTURE of NN Learning

To sum up, try to minimize this cost function, and study backpropagation algorithm.

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - h_{\theta}(x^{(i)}))_k \right]$$

$\min J(\theta)$ w.r.t θ

we need to compute (through coding):

$$J(\theta), \quad \frac{\partial}{\partial (\theta_{ij}^{(l)})} J(\theta) \rightarrow \text{BACKPROPAGATION}$$

Gradient Descent algorithm for optimization!!

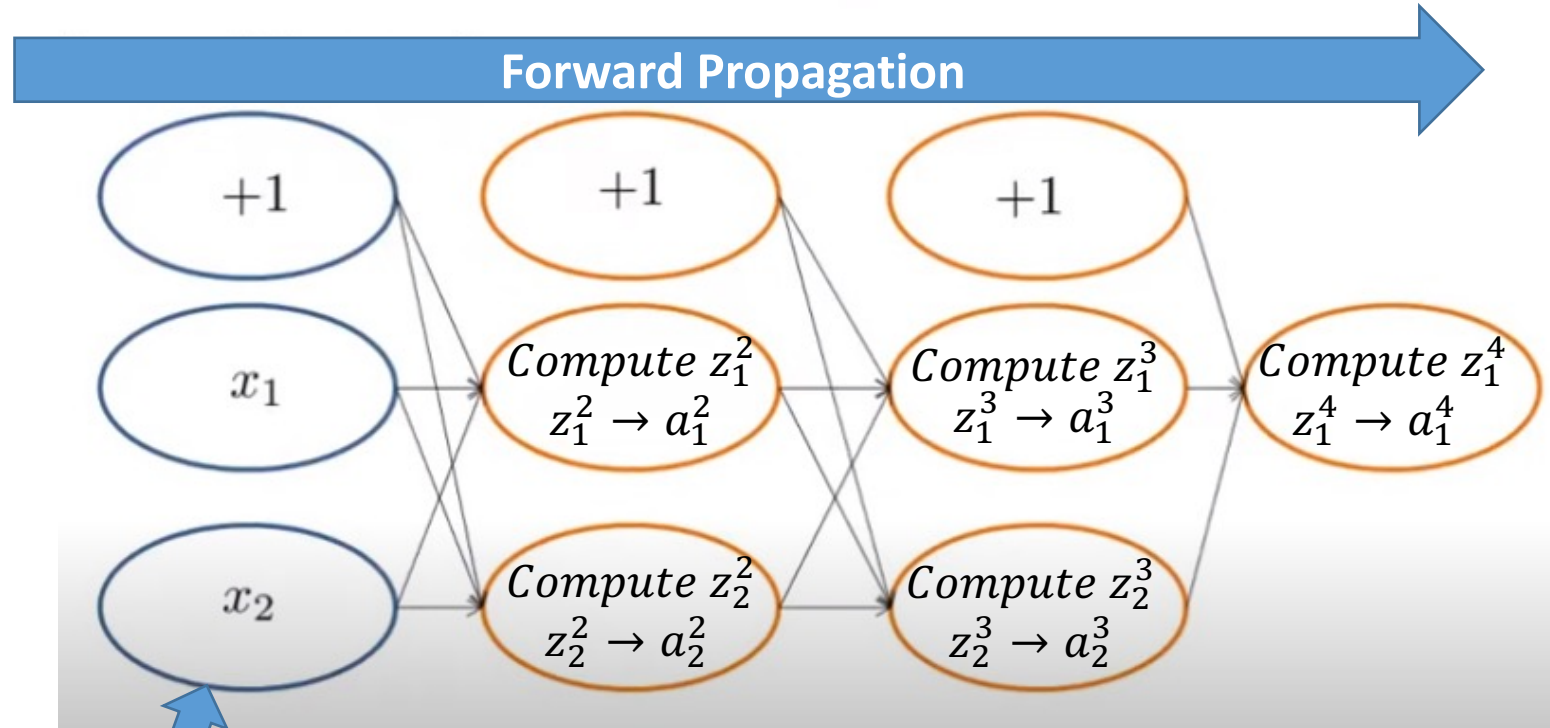
Let's summarize & provide the intuition one more time...

- Less mathematically clean compared to linear/logistic regression.
- Hands on experience will also teach you these steps.
- In order to understand the concept better, let's focus on the following example. 😊

Neural Networks Learning - Forward Propagation Intuition

Let's assume we have a particular example (x_i, y_i) .

(2 input units, 2 hidden units in layer 2, 2 hidden units in layer 3, 1 output unit in layer 4.)

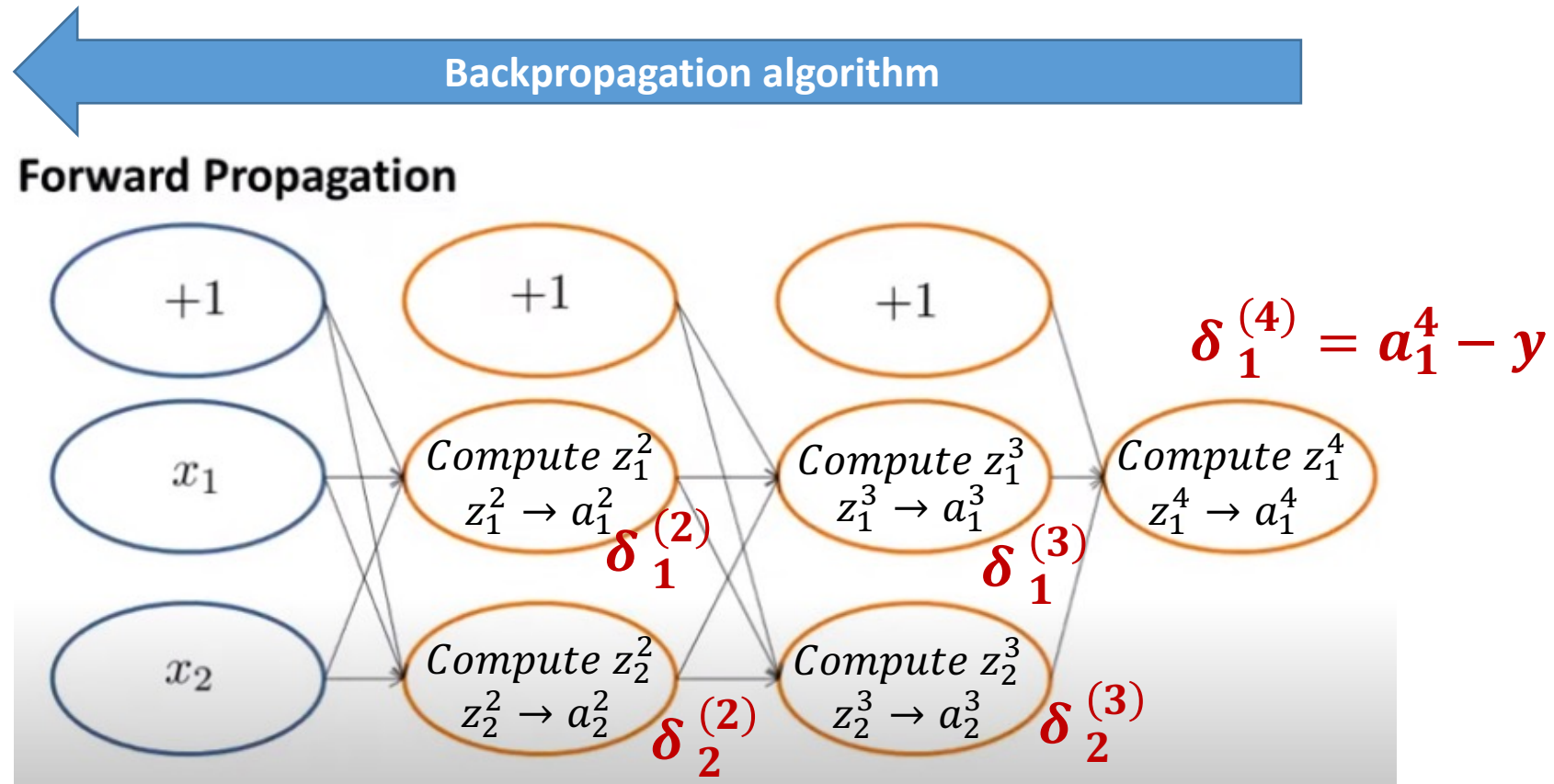


$(x^{(i)}, y^{(i)})$

$$a^{(2)} = g(z^{(2)})$$

Neural Networks Learning – Backpropagation Intuition

- Let's look at what BP is doing:
- For each node, BP computes: $\delta_j^{(l)}$ = 'error' of node j in layer l .



Study page 67 for $\delta_j^{(l)}$ calculation.

Neural Networks Learning

- BP is actually very similar to forward propagation.
- Instead of flowing from left to the right, the computation flows from right to the left.

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - h_{\theta}(x^{(i)}))_k \right]$$

Focusing on single example ($m=1$), case of 1 output unit ($K=1$):

$$cost(i) = y^i \log h_{\theta}(x^{(i)}) + (1 - y^i) \log(1 - h_{\theta}(x^{(i)}))$$

This cost is looking how well is the network doing on example i ? How close is the output to the actual label? (logistic regression idea)

Conclusion

- What is Neural Network? When should we use it?
- History
- Neural Networks vs Deep Neural Networks
- Activation functions
- Mathematical Representation of NN
- Learning: forward and backpropagation

MUST: Please study the lecture notes.

Suggestion: If you're not very clear or want to learn more, please do read:
Neural Network part of "C. Bishop: Pattern Recognition and Machine Learning.
Springer, 2006" (*recommended text book*).

THE END

Acknowledgement

- National University of Singapore – Pattern Recognition Module (EE5907R)
- National University of Singapore – Neural Networks Module (EE5904R)
- University of Edinburgh, United Kingdom – Machine Learning And Pattern Recognition (MLPR, INFR11130)
- Stanford University – Machine Learning (CS229)