



DEEP NEURAL NETWORK

MODULE # 1 : FUNDAMENTALS OF NEURAL NETWORK

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

WHAT WE LEARN...

- ① Fundamentals of Neural Network and Multilayer Perceptron
- ② Deep Feed-forward Neural Network
- ③ Improve its performance by Optimization
- ④ Improve its performance by Regularization
- ⑤ Convolutional Neural Networks
- ⑥ Sequence Models
- ⑦ Attention Mechanism
- ⑧ Neural Network search
- ⑨ Time series Modelling and Forecasting
- ⑩ Other Learning Techniques

TEXT AND REFERENCE BOOKS

TEXT BOOKS

- T1 Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola.
https://d2l.ai/chapter_introduction/index.html
- R1 Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>

EVALUATION SCHEDULE

No	Name	Type	Duration	Weight	Day, Date, Time
EC1	Quiz I	Online	0.5 hr	5 %	Check Canvas
	Quiz II	Online	0.5 hr	5 %	Check Canvas
	Assignment I	Online	4 weeks	15 %	Check Canvas
	Assignment II	Online	4 weeks	15 %	Check Canvas
EC2	Mid-sem Regular	Closed book	2 hrs	30%	Check Calendar
EC3	Compre-sem Regular	Open book	2.5 hrs	30%	Check Calendar

LAB SESSIONS

① Python Libraries – Keras and Tensorflow

L1 Introduction to Tensorflow and Keras

L2 Deep Neural Network with Back-propagation and optimization

L3 CNN

L4 RNN

L5 LSTM

L6 Auto-encoders

LMS

Refer Canvas for the following

- Handout
- Schedule for Webinars
- Schedule of Quiz, and Assignments.
- Evaluation scheme
- Session Slide Deck
- Demo Lab Sheets
- Quiz-I, Quiz-II
- Assignment-I, Assignment-II
- Sample QPs

The Lecture Recordings are made available on Microsoft Teams.

HONOR CODE

All submissions for graded components must be the result of your original effort. It is strictly prohibited to copy and paste verbatim from any sources, whether online or from your peers. The use of unauthorized sources or materials, as well as collusion or unauthorized collaboration to gain an unfair advantage, is also strictly prohibited. Please note that we will not distinguish between the person sharing their resources and the one receiving them for plagiarism, and the consequences will apply to both parties equally.

In cases where suspicious circumstances arise, such as identical verbatim answers or a significant overlap of unreasonable similarities in a set of submissions, will be investigated, and severe punishments will be imposed on all those found guilty of plagiarism.

IN CASE OF QUERIES REGARDING THE COURSE ...

STEP 1 : Post in the discussion forum.

Read through the existing posts and if you find any topic similar to your concern, add on to the existing discussion.

Avoid duplication of queries or issues. This is highly appreciated.

STEP 2 : Email to the IC at seetha.p@pilani.bits-pilani.ac.in if the query or issue is not resolved within one weeks' time. Turn around for response to the email is 48hrs.

In the subject please mention the phrase "**DNN**" clearly.

Use BITS email id for correspondence. Emails from personal emails will be ignored without any reply.

PATIENCE is highly APPRECIATED :)

TABLE OF CONTENTS

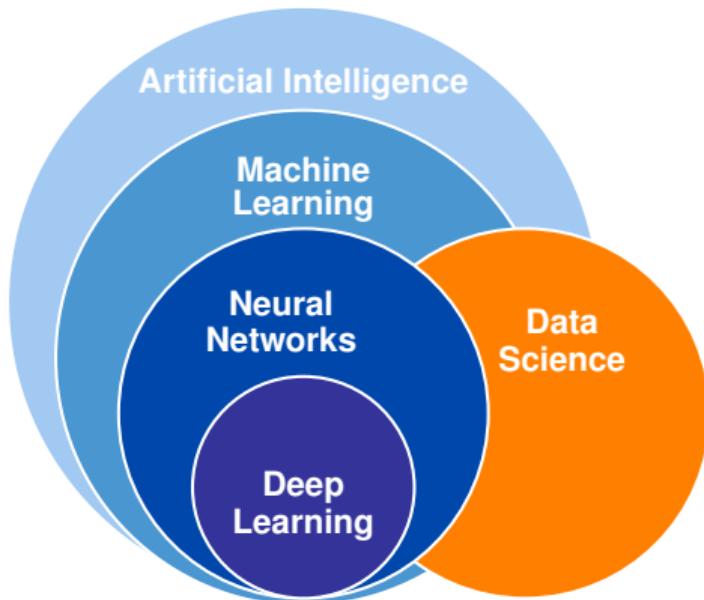
- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

WHAT IS DEEP LEARNING?

- Deep Learning is a type of **machine learning** based on **artificial neural networks** in which multiple layers of processing are used to extract progressively higher level features from data.
- Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is **inspired by the human brain**.
- Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: **learn by example**.
- Deep learning is a subset of machine learning, which is essentially a neural network with **three or more layers**.
- Deep Learning gets its name from the fact that we add more **Layers** to learn from the data.

WHERE IN AI SITS DL?

- AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don't involve any learning.



AI – ML – DL

AI : Artificial intelligence is the **science** of making things smart. The aim is make machines perform human tasks. Eg: Robot cleaning a room.

ML : Machine learning is an **approach** to AI. The machine learns or perform tasks through learning by experience.

DL : Deep Learning is a **technique** for implementing machine learning to recognise patterns.

DEEP (MACHINE) LEARNING

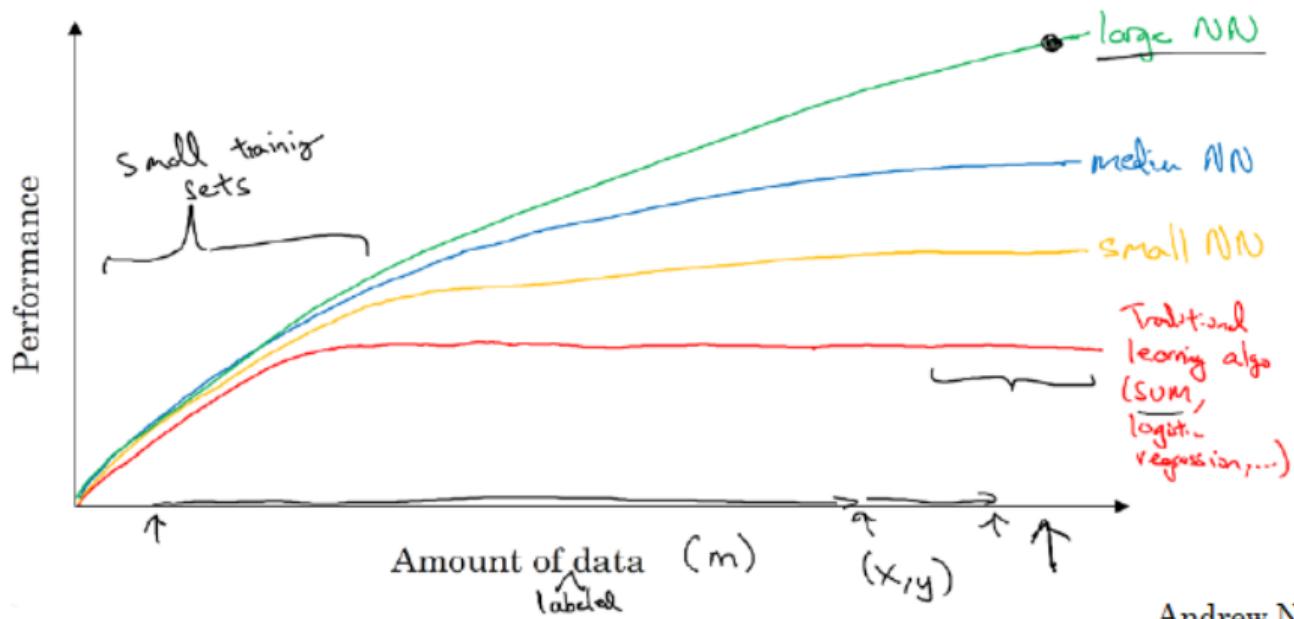
- Deep learning is a specific subfield of machine learning.
- Learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations.
- The **deep** in deep learning stands for this idea of successive layers of representations.
- The number of layers that contribute to model the data is called the **depth** of the model.
- In deep learning, the layered representations are learned via models called **neural networks**, structured in literal layers stacked on top of each other.

WHY DEEP LEARNING?

- Large amounts of data
- Lots and lots of unstructured data like images, text, audio, video
- Cheap, high-quality sensors
- Cheap computation - CPU, GPU, Distributed clusters
- Cheap data storage
- Learn by examples
- Automated feature generation
- Better learning capabilities
- Scalability
- Advance analytics can be applied

WHY DEEP LEARNING?

Scale drives deep learning progress



DEEP LEARNING: TIMELINE

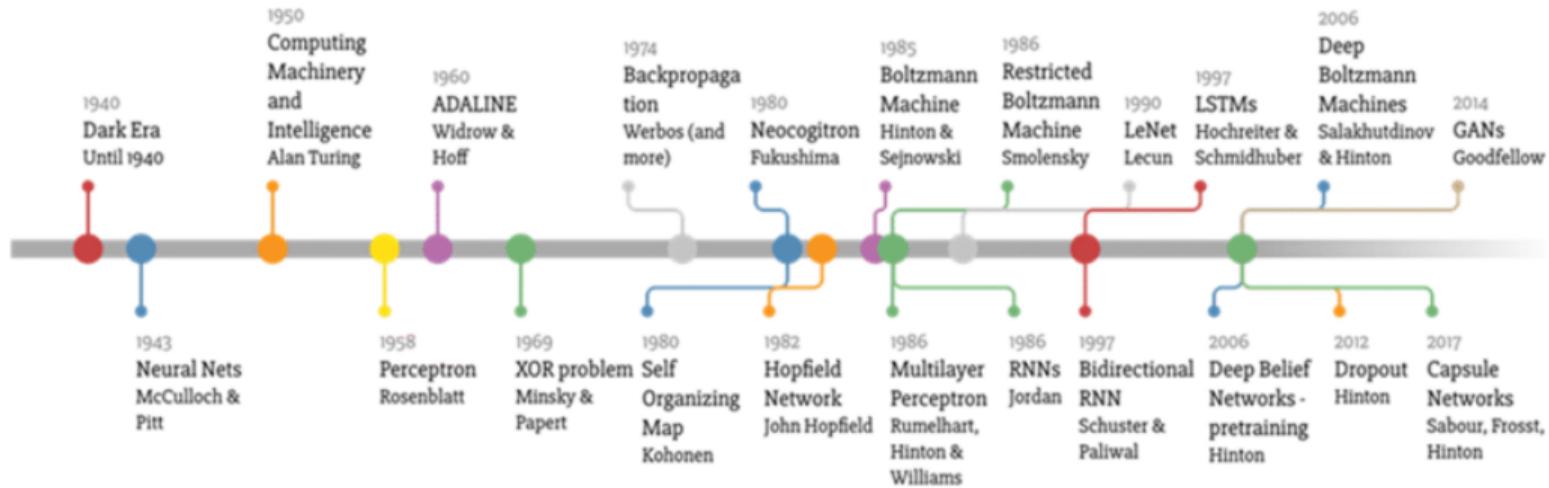


TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

BREAKTHROUGHS WITH NEURAL NETWORKS

TECHNEWSWORLD

EMERGING TECH

SEARCH



Computing

Internet

IT

Mobile Tech

Reviews

Security

Technology

Tech Blog

Reader Service

Microsoft AI Beats Humans at Speech Recognition

By Richard Adhikari

Oct 20, 2016 11:40 AM PT

Print

Email



Most Popular Newsletters News Alert

How do you feel about Black Friday and Cyber Monday?

- They're great -- I get a lot of bargains!
- The deals are too spread out -- I'd prefer just one day.
- They're a fun way to kick off the holiday season.
- I don't like the commercialization of Thanksgiving Day.
- They're crucial for the retail industry and the economy.
- The deals typically aren't that good.

[Vote to See Results](#)



Image: Adobe Stock

Microsoft's Artificial Intelligence and Research Unit earlier this week reported that its speech recognition technology had surpassed the performance of human transcriptionists.

E-Commerce Times

[Black Friday Shoppers Hungry for New Experiences, New Tech](#)

[Pay TV's Newest Innovation: Giving Users Control](#)

[Apple Celebrates Itself in \\$300 Coffee Table Tome](#)

[AWS Enjoys Top Perch in IaaS, PaaS Markets](#)

[US Comptroller Gears Up for Blockchain and](#)

BREAKTHROUGHS WITH NEURAL NETWORKS



The Keyword Latest Stories Product News Topics SEARCH MORE

TRANSLATE NOV 15, 2016

Found in translation: More accurate, fluent sentences in Google Translate

Barak Turovsky
PRODUCT LEAD, GOOGLE TRANSLATE

In 10 years, Google Translate has gone from supporting just a few languages to 103, connecting strangers, reaching across language barriers and even helping

IMAGE SEGMENTATION AND RECOGNITION

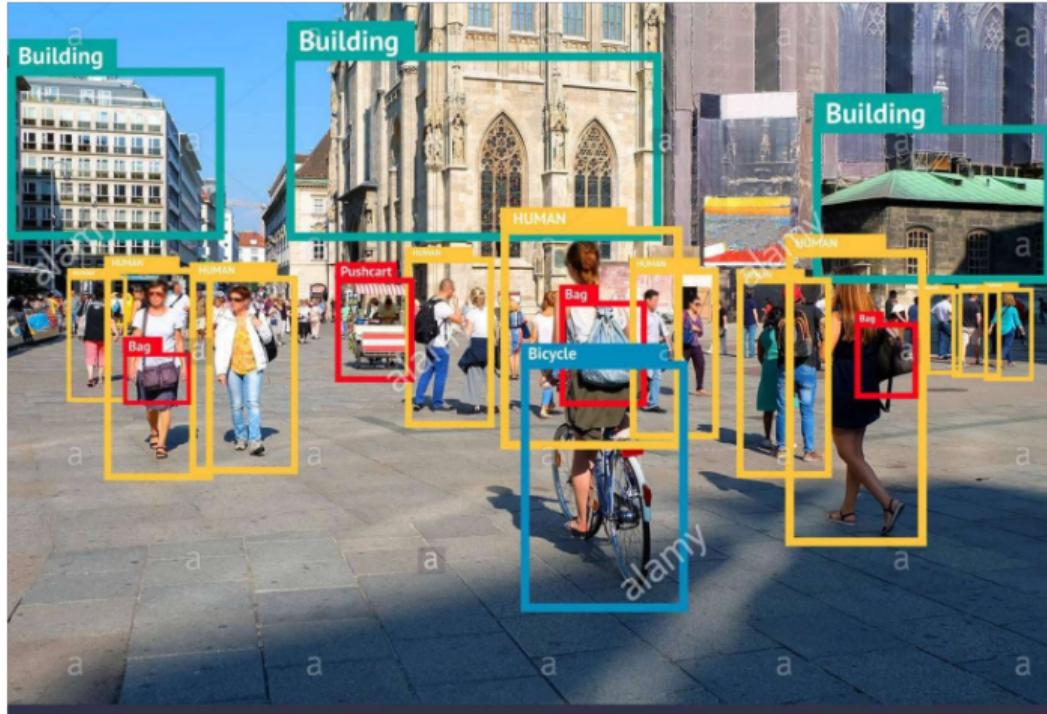
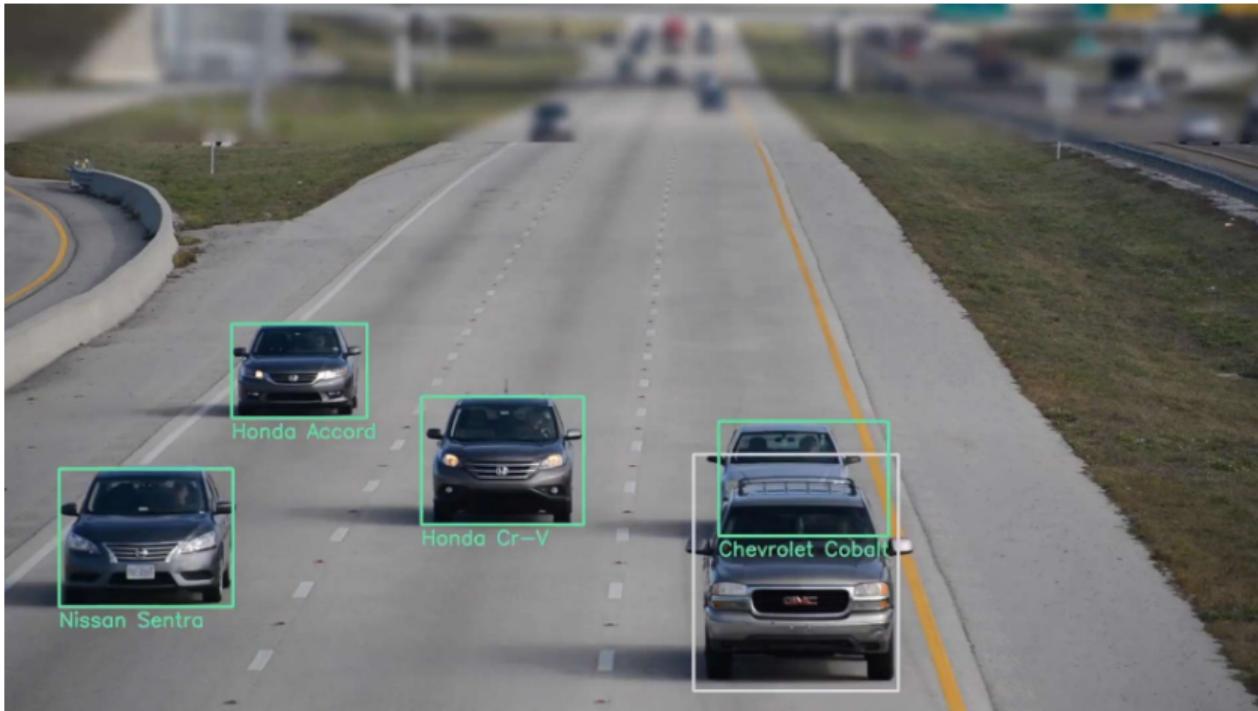


IMAGE RECOGNITION



BREAKTHROUGHS WITH NEURAL NETWORKS

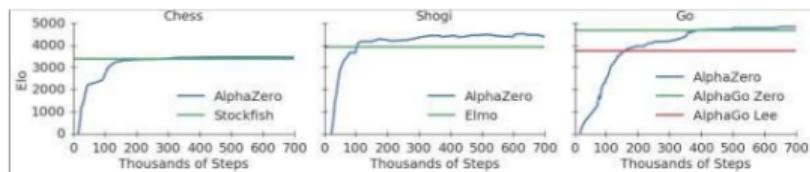
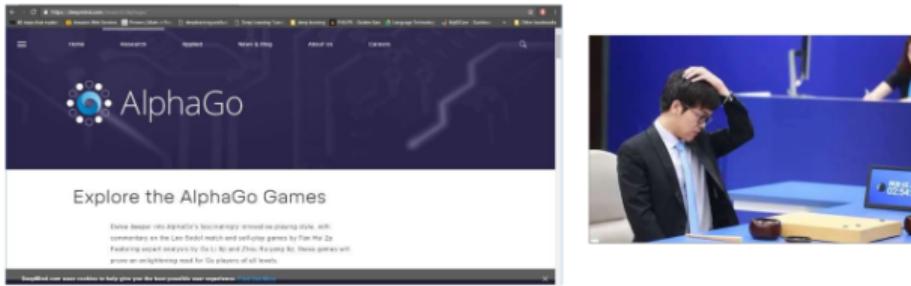


Figure 1: Training *AlphaZero* for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. **a** Performance of *AlphaZero* in chess, compared to 2016 TCEC world-champion program *Stockfish*. **b** Performance of *AlphaZero* in shogi, compared to 2017 CSA world-champion program *Elmo*. **c** Performance of *AlphaZero* in Go, compared to *AlphaGo Lee* and *AlphaGo Zero* (20 block / 3 day) (29).

BREAKTHROUGHS WITH NEURAL NETWORKS

Introducing ChatGPT

We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.

[Try ChatGPT ↗](#)

[Read about ChatGPT Plus](#)



APPLICATIONS OF DEEP LEARNING

Application	Input	Output	Neural Network
Real Estate	House features	House Price	Std NN
Photo Tagging	Image	Text	CNN
Object detection	Image	Bounding box	CNN
Speech Recognition	Audio	Text transcript	RNN
Translation	English Text	French Text	RNN
Autonomous driving	Image, Sensors Radars	Position of other cars, objects, signals	Hybrid NN

MANY MORE APPLICATIONS...

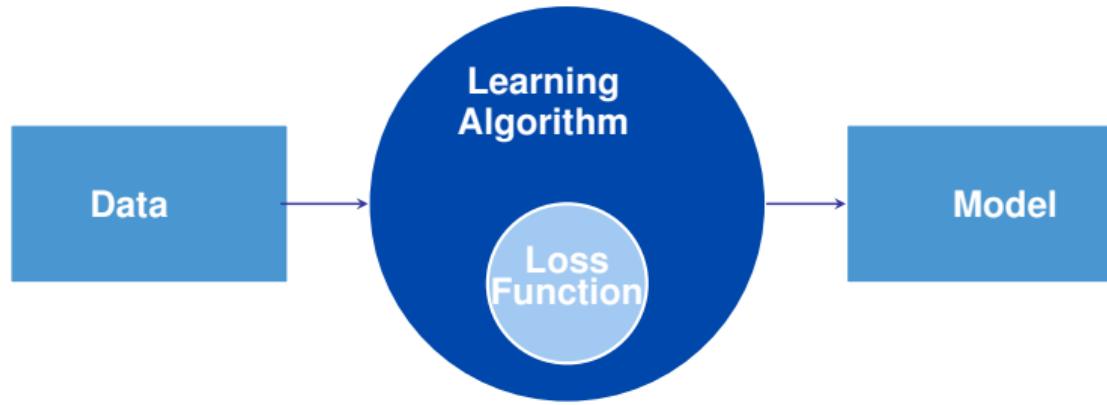
- a program that predicts tomorrow's weather given geographic information, satellite images, and a trailing window of past weather.
- a program that takes in a question, expressed in free-form text, and answers it correctly.
- a program that given an image can identify all the people it contains, drawing outlines around each.
- a program that presents users with products that they are likely to enjoy but unlikely, in the natural course of browsing, to encounter.

TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

CORE COMPONENTS OF DL PROBLEM

- The **data** that we can learn from.
- A **model** of how to transform the data.
- An **objective function** that quantifies how well (or badly) the model is doing.
- An **algorithm** to adjust the model's parameters to optimize the objective function.



1. DATA

- Collection of examples.
- Data has to be converted to an useful and a suitable numerical **representation**.
- Each example (or data point, data instance, sample) typically consists of a set of attributes called **features** (or covariates), from which the model must make its predictions.
- In the supervised learning problems, the attribute to be predicted is designated as the **label** (or target).
- Mathematically, a set of m examples,

$$Data = \mathcal{D} = \{X, t\}$$

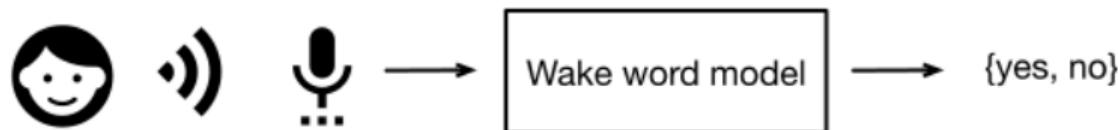
- We need right data.

1. DATA

- Dimensionality of data
 - ▶ Each example has the same number of numerical values. This data consist of fixed-length vectors. Eg: Image
 - ▶ The constant length of the vectors as the dimensionality of the data.
 - ▶ Text data has varying-length data.

2. MODEL

- Model denotes the computational machinery for ingesting data of one type, and spitting out predictions of a possibly different type.
- Deep learning models consist of many successive transformations of the data that are chained together top to bottom, thus the name deep learning.



3. OBJECTIVE FUNCTION

- **Learning means improving at some task over time.**
- A formal mathematical system of learning machines is defined using formal measures of how good (or bad) the models are. These formal measures are called as **objective functions**.
- By convention, objective functions are defined so that lower is better. Because lower is better, these functions are sometimes called **loss functions**.

3. LOSS FUNCTIONS

- To predict numerical values (regression), the most common loss function is squared error.
- For classification, the most common objective is to minimize error rate, i.e., the fraction of examples on which our predictions disagree with the ground truth.

3. LOSS FUNCTIONS

- Loss function is defined
 - ▶ with respect to the model's parameters
 - ▶ depends upon the dataset.
- We learn the best values of our model's parameters by **minimizing the loss** incurred on a set consisting of some number of examples collected for training. However, doing well on the training data does not guarantee that we will do well on unseen data. i.e **Model has to generalize better**.
- When a model performs well on the training set but fails to generalize to unseen data, we say that it is **overfitting**.

4. OPTIMIZATION ALGORITHMS

- Optimization Algorithm is an algorithm capable of **searching for the best possible parameters for minimizing the loss function.**
- Popular optimization algorithms for deep learning are based on an approach called **gradient descent**.

EXAMPLE OF THE FRAMEWORK

- We have to tell a computer explicitly how to map from inputs to outputs.
- We have to define the problem precisely, pinning down the exact nature of the inputs and outputs, and choosing an appropriate model family.
- Collect a huge dataset containing examples of audio and label those that do and that do not contain the wake word.



EXAMPLE OF THE FRAMEWORK

- Create a Model
 - ▶ Define a flexible program whose behavior is determined by a number of parameters.
 - ▶ To determine the best possible set of parameters, use the data. The parameters should improve the performance of the program with respect to some measure of performance on the task of interest.
 - ▶ After fixing the parameters, we call the program a model.
Eg: The model receives a snippet of audio as input, and the model generates a selection among yes, no as output.
 - ▶ The set of all distinct programs (input-output mappings) that we can produce just by manipulating the parameters is called a family of models.
Eg: We expect that the same model family should be suitable for "Alexa" recognition and "Hey Siri" recognition because they seem, intuitively, to be similar tasks.

EXAMPLE OF THE FRAMEWORK

- The meta-program that uses our dataset to choose the parameters is called a learning algorithm.
- In machine learning, the learning is the process by which we discover the right setting of the parameter coercing the desired behavior from our model.
- **Train the model with data.**

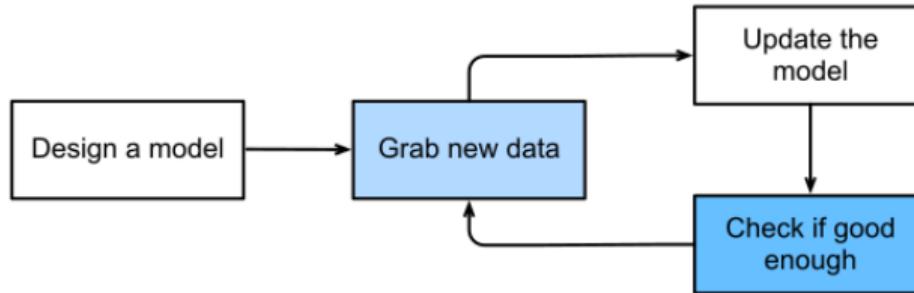
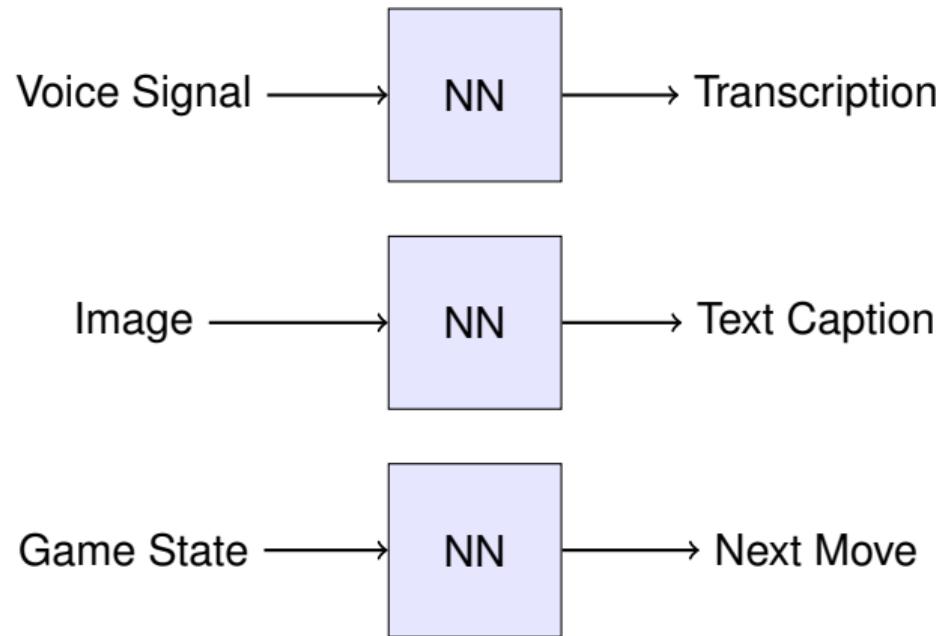


Fig. 1.1.2: A typical training process.

TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

WHAT ARE NEURAL NETWORKS??



WHAT ARE NEURAL NETWORKS???

It begins with brain.



- Humans learn, solve problems, recognize patterns, create, think deeply about something, meditate and many many more.....
- Humans learn through association. [Refer to Associationism for more details.]

OBSERVATION: THE BRAIN

- The brain is a mass of interconnected neurons.
- Number of neurons is approximately 10^{10} .
- Connections per neuron is approximately $10^{(4 \text{ to } 5)}$.
- Neuron switching time is approximately 0.001 second.
- Scene recognition time is 1 second.
- 100 inference steps doesn't seem like enough. Lot of parallel computation.

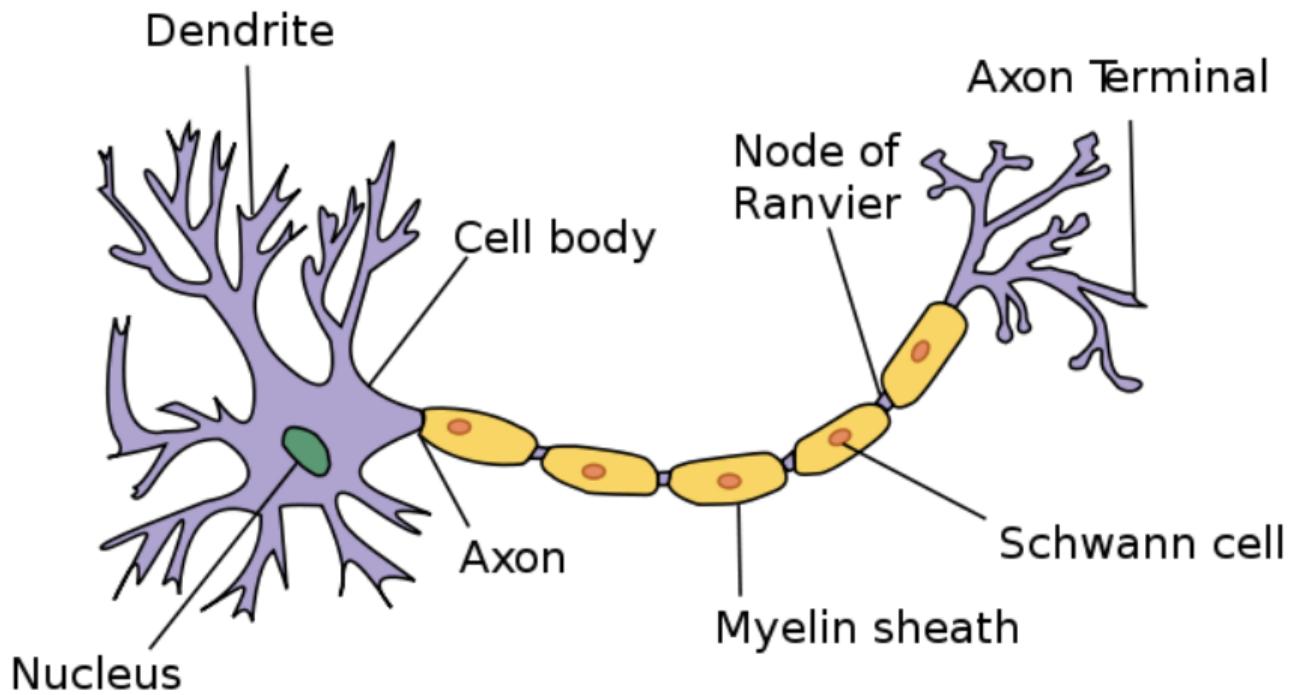


BRAIN: INTERCONNECTED NEURONS

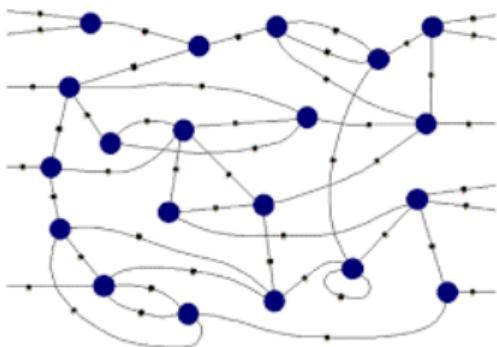


- Many neurons connect **in** to each neuron.
- Each neuron connects **out** to many neurons.

BIOLOGICAL NEURON



CONNECTIONIST MACHINES



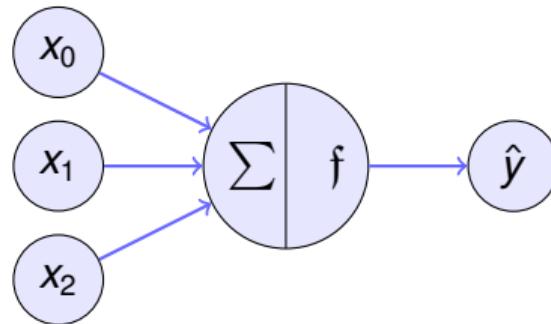
1

- Network of processing elements, called artificial neural unit.
- The neurons are interconnected to form a network.
- All world knowledge is stored in the connections between these elements.
- Neural networks are connectionist machines.

¹alanturing.net

ARTIFICIAL NEURON

- Neuron is a processing element inspired by how the brain works.
- Similar to biological neuron, each artificial neuron will do some computation. Each neuron is interconnected to other neurons.
- Similar to brain, the interconnections between neurons store the knowledge it learns. The knowledge is stored as parameters.



PROPERTIES OF ARTIFICIAL NEURAL NETS (ANNs)

- Many neuron-like threshold switching units.
- Many weighted interconnections among units.
- Highly parallel, distributed process.
- Emphasis on tuning parameters or weights automatically.

WHEN TO CONSIDER NEURAL NETWORKS?

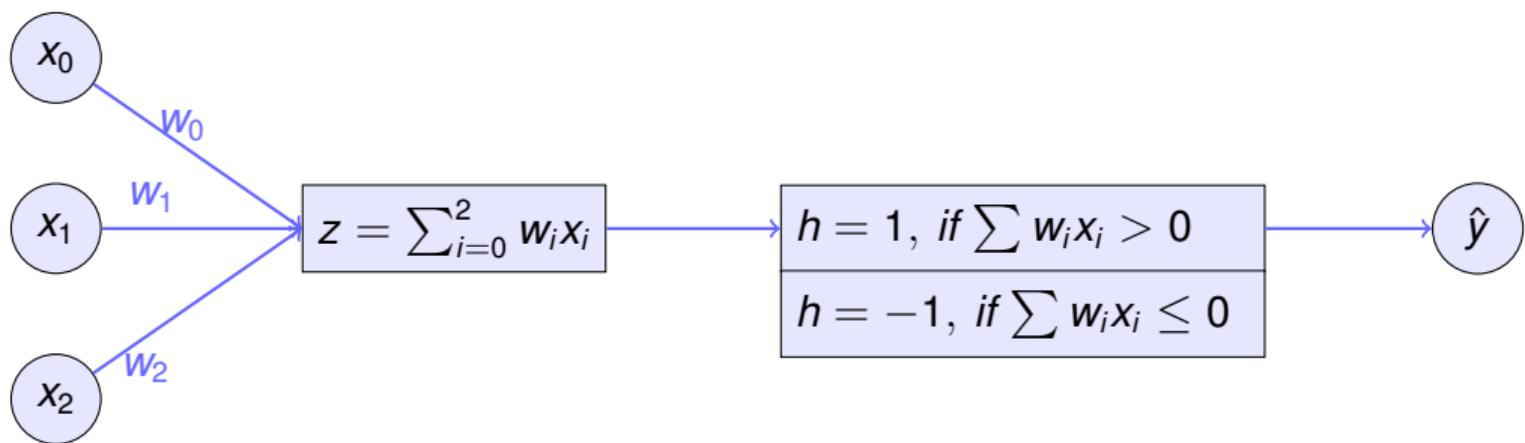
- Input is high-dimensional discrete or real-valued (e.g. raw sensor input).
- Possibly noisy data. Data has lots of errors.
- Output is discrete or real valued or a vector of values.
- Form of target function is unknown.
- Human readability, in other words, explainability, of result is unimportant.
- Examples:
 - ▶ Speech phoneme recognition
 - ▶ Image classification
 - ▶ Financial prediction

TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

PERCEPTRON

- One type of ANN system is based on a unit called a perceptron.
- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.**



REPRESENTING LOGIC GATES USING PERCEPTRON

- Any linear decision can be represented by either a linear regression equation or a Boolean equation.
- The quest is to find out how to represent the above using Perceptron.
- For this we test whether each of the logic gates can be represented by a Perceptron.

NOT GATE

Question:

- How to represent NOT gate using a Perceptron?
- What are the parameters for the NOT Perceptron?
- Data is given below.



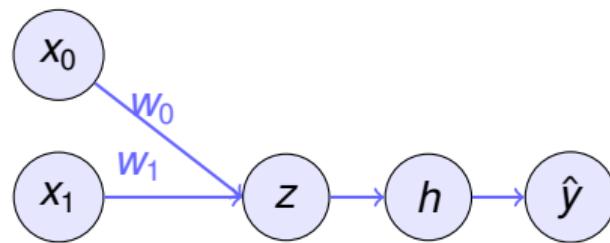
A	B
0	1
1	0

Rewrite as

x_1	t
-1	1
1	-1

PERCEPTRON FOR NOT GATE

- Perceptron equation is
 $\hat{y} = w_0x_0 + w_1x_1.$
- $x_0 = 1$ always.
- $h > 0$ for output to be 1.
- For each row of truth table, the equations are given.
- One solution is $w_0 = 1$ and $w_1 = -1$. (Intuitive solution)
- This give a beautiful linear decision boundary.

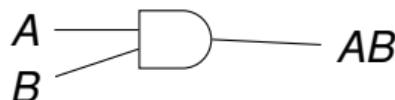


x_1	t_1	h	h
-1	1	$w_0x_0 + w_1(-1) > 0$	$w_0 - w_1 > 0$
1	-1	$w_0x_0 + w_1(1) < 0$	$w_0 + w_1 < 0$

AND LOGIC GATE

Question:

- How to represent AND gate using a Perceptron?
- What are the parameters for the AND Perceptron?
- Data is given below.



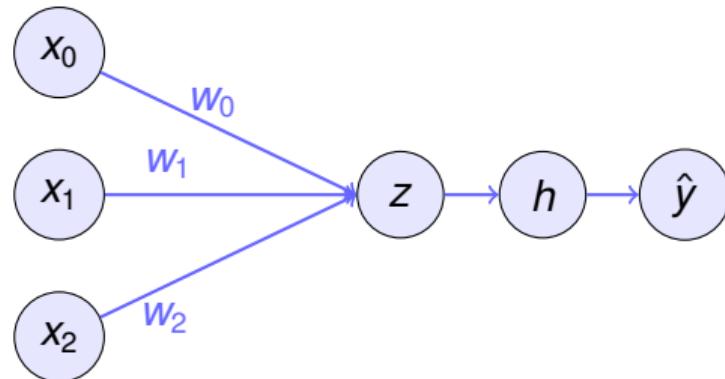
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	t
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Rewrite as

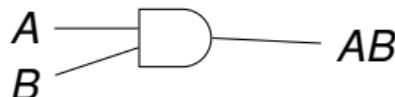
PERCEPTRON FOR AND GATE

- Perceptron equation is
 $\hat{y} = w_0x_0 + w_1x_1 + w_2x_2$.
- $h > 0$ for output to be 1.
- For each row of the truth table, the equations are given.
- Solve for the inequalities.
- One solution is
 $w_1 = w_2 = 2, w_0 = (-1)$.
- This give a beautiful linear decision boundary.

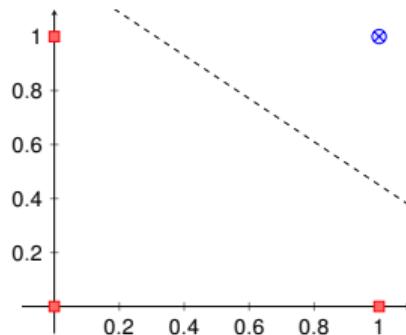
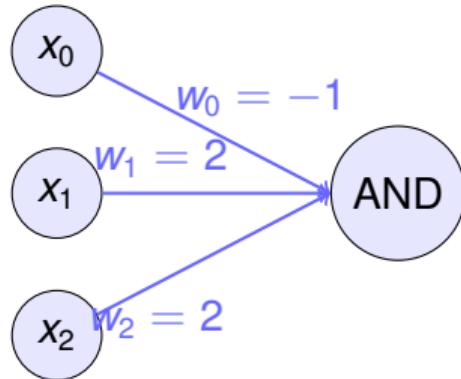


x_1	x_2	t	h
-1	-1	-1	$w_0 + w_1(-1) + w_2(-1) < 0$
-1	1	-1	$w_0 + w_1(-1) + w_2(1) < 0$
1	-1	-1	$w_0 + w_1(1) + w_2(-1) < 0$
1	1	1	$w_0 + w_1(1) + w_2(1) > 0$

PERCEPTRON FOR AND GATE



A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1



Perceptron and Decision boundary for AND gate

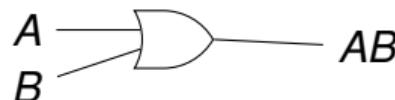
EXERCISES

- Represent OR gate using Perceptron. Compute the parameters of the Perceptron.
- Represent NOR gate using Perceptron. Compute the parameters of the Perceptron.
- Represent NAND gate using Perceptron. Compute the parameters of the Perceptron.

OR LOGIC GATE

Question:

- How to represent OR gate using a Perceptron?
- What are the parameters for the OR Perceptron?
- Data is given below.



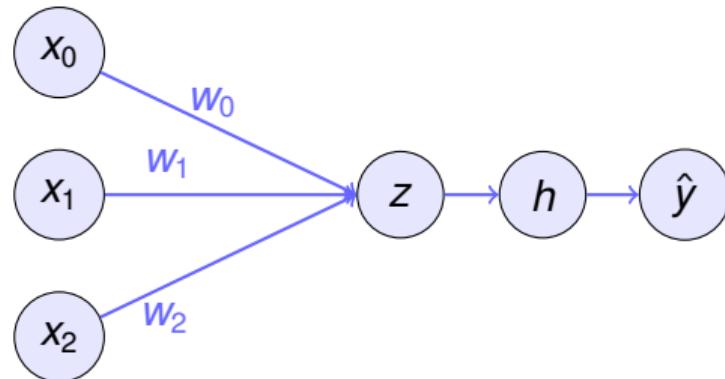
A	B	AB
0	0	0
0	1	1
1	0	1
1	1	1

Rewrite as

x_1	x_2	t
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

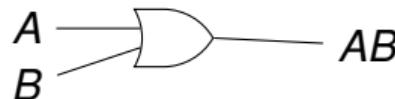
PERCEPTRON FOR OR GATE

- Perceptron equation is
 $\hat{y} = w_0x_0 + w_1x_1 + w_2x_2.$
- $h > 0$ for output to be 1.
- For each row of the truth table, the equations are given.
- Solve for the inequalities.
- One solution is
 $w_0 = w_1 = w_2 = 2.$
- This give a beautiful linear decision boundary.

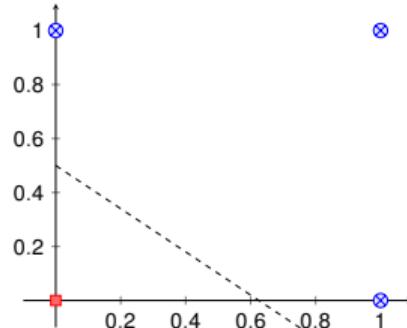
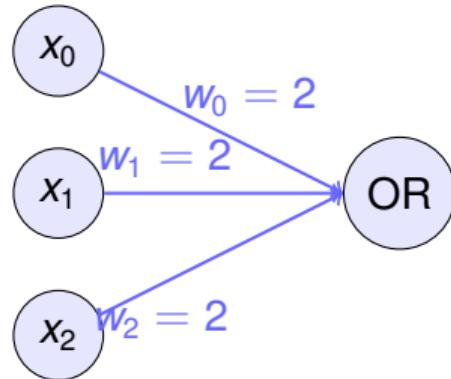


x_1	x_2	t	h
-1	-1	-1	$w_0 + w_1(-1) + w_2(-1) < 0$
-1	1	-1	$w_0 + w_1(-1) + w_2(1) > 0$
1	-1	-1	$w_0 + w_1(1) + w_2(-1) > 0$
1	1	1	$w_0 + w_1(1) + w_2(1) > 0$

PERCEPTRON FOR OR GATE



A	B	AB
0	0	0
0	1	1
1	0	1
1	1	1



Perceptron and Decision boundary for OR gate

PERCEPTRON LEARNING ALGORITHM

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - \hat{y})x_i$$

where :

t = target value

\hat{y} = perceptron output

η = learning rate

CONVERGENCE OF PERCEPTRON LEARNING ALGORITHM

It can be proved that the algorithm will converge

- If training data is linearly separable.
- Learning rate is sufficiently small.
 - ▶ The role of the learning rate is to moderate the degree to which weights are changed at each step.
 - ▶ It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

PERCEPTRON LEARNING FOR NOT GATE

- Equations are

Hypothesis $z = w_0 + w_1 x_1$

Activation $h = \text{sign}(z)$

Compute output $\hat{y} = h = \text{sign}(w_0 + w_1 x_1)$

Compute $\Delta w = \eta(t - \hat{y})x$

Update $w_{new} \leftarrow w_{old} + \Delta w$

PERCEPTRON LEARNING FOR NOT GATE

- Assume $w_0 = w_1 = 0$. Let the learning rate = $\eta = 1$.

- Epoch 1**

x_1	t	w_1	w_0	z	h	$isequal(t, h)$	Δw	New w
0	1	0	0	$0 + 0 = 0$	-1	Not equal	$\Delta w_1 = 1(1 - (-1))0 = 0$ $\Delta w_0 = 1(1 - (-1)) = 2$	$w_1 \leftarrow 0 + 0 = 0$ $w_0 \leftarrow 0 + 2 = 2$
1	-1	0	2	$2 + 0 = 2$	1	Not equal	$\Delta w_1 = 1(-1 - 1)1 = -2$ $\Delta w_0 = 1(-1 - 1) = -2$	$w_1 \leftarrow 0 + -2 = -2$ $w_0 \leftarrow 2 + -2 = 0$

- Epoch 2**

x_1	t	w_1	w_0	z	h	$isequal(t, h)$	Δw	New w
0	1	-2	0	$0 + 0 = 0$	-1	Not equal	$\Delta w_1 = 1(1 - (-1))0 = 0$ $\Delta w_0 = 1(1 - (-1)) = 2$	$w_1 \leftarrow -2 + 0 = -2$ $w_0 \leftarrow 0 + 2 = 2$
1	-1	-2	2	$2 + -2 = 0$	-1	Equal		

- Algorithm Converges.

DEMO PYTHON CODE

- <https://colab.research.google.com/drive/1DUVcOoUIWhl8GQKc6AWR1wi0LaMeNkgD?usp=sharing>

Student pl note:

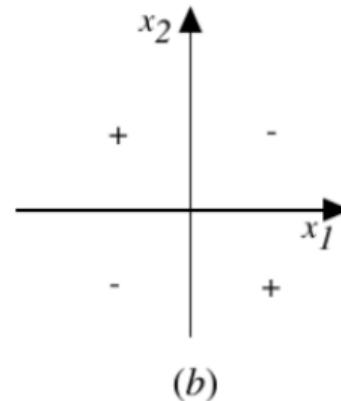
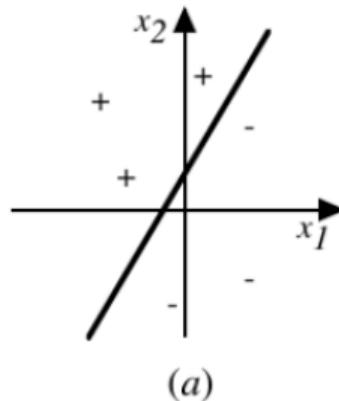
The Python notebook is shared for anyone who has access to the link and the access is restricted to use **BITS email id**. So please do not access from non-BITS email id and send requests for access. Access for non-BITS email id will not be granted.

EXERCISE

- Represent OR gate using Perceptron. Compute the parameters of the Perceptron using Perceptron learning algorithm.
- Represent AND gate using Perceptron. Compute the parameters of the Perceptron using Perceptron learning algorithm.

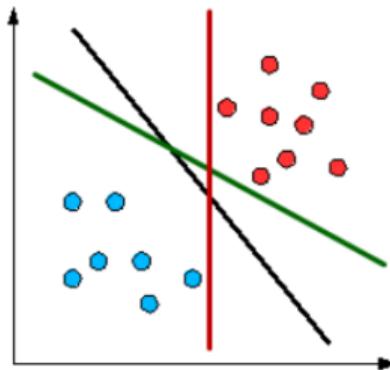
REPRESENTATIONAL POWER OF PERCEPTRON

- A Perceptron represents a hyperplane decision surface in the n-dimensional space of examples.
- The Perceptron outputs a 1 for examples lying on one side of the hyperplane and outputs a -1 for examples lying on the other side.

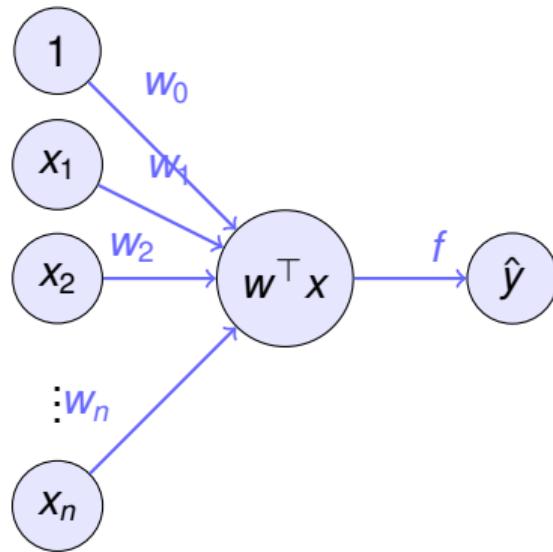


LINEARLY SEPARABLE DATA

- Two sets of data points in a two dimensional space are said to be linearly separable when they can be completely separable by a single straight line.
- A straight line can be drawn to separate all the data examples belonging to class +1 from all the examples belonging to the class -1. Then the two-dimensional data are clearly linearly separable.
- An infinite number of straight lines can be drawn to separate the class +1 from the class -1.



PERCEPTRON FOR LINEARLY SEPARABLE DATA



$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$\begin{aligned} \sum &= w_0 + w_1 x_1 + \cdots + w_n x_n \\ &= w^\top x \end{aligned}$$

$$\hat{y} = \begin{cases} 1 & \text{if } w^\top x > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Challenge: How to learn these n parameters $w_0 \dots w_n$?
 Solution: Use Perceptron learning algorithm.

PERCEPTRON TRAINING

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Perceptron training rule will converge

- If training data is linearly separable
- η sufficiently small

PERCEPTRON AND ITS LEARNING - REVIEW

- Data
 - ▶ Truth tables or set of examples
- Model
 - ▶ Perceptron
- Objective Function
 - ▶ Deviation of desired output t and the computed output \hat{y}
- Learning Algorithm
 - ▶ Perceptron Learning algorithm

TABLE OF CONTENTS

- 1 COURSE LOGISTICS
- 2 INTRODUCTION TO DEEP LEARNING
- 3 APPLICATIONS OF DEEP LEARNING
- 4 KEY COMPONENTS OF DL PROBLEM
- 5 ARTIFICIAL NEURAL NETWORK
- 6 PERCEPTRON
- 7 SINGLE PERCEPTRON FOR LINEAR REGRESSION
- 8 MULTI LAYER PERCEPTRON (MLP)

LINEAR REGRESSION EXAMPLE

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):

$$price = w_{area} * area + w_{age} * age + b$$

w_{area} and w_{age} are called weights, and b is called a bias.

- The weights determine the influence of each feature on our prediction and the bias just says what value the predicted price should take when all of the features take value 0.

DATA

- The dataset is called a **training dataset or training set**.
- Each row is called an **example** (or data point, data instance, sample).
- The thing we are trying to predict is called a **label** (or target).
- The independent variables upon which the predictions are based are called **features** (or covariates).

m – number of training examples

i – index of i^{th} example

$x^{(i)}$ – features of i^{th} example

$= [x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}]$

$y^{(i)}$ – label corresponding to i^{th} example

$= [y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(m)}]$

AFFINE TRANSFORMATIONS AND LINEAR MODELS

- Affine transformations are

$$\begin{aligned}\hat{y} &= w_1x_1 + w_2x_2 + \dots + w_dx_d + b \\ &= w^T x + b \\ \hat{y} &= W X + b\end{aligned}$$

- $\hat{y} = W X + b$ is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.
- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen **weights (w)** and **bias (b)**.

LOSS FUNCTION

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the squared error.

SQUARED ERROR LOSS FUNCTION

- The most popular loss function in regression problems is the squared error.
- For each example,

$$\text{loss}^{(i)}(w, b) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

- For the entire dataset of m examples average (or equivalently, sum) the losses

$$L(w, b) = \frac{1}{m} \sum_{i=1}^m \text{loss}^{(i)}(w, b) = \frac{1}{m} \sum_{i=1}^m \left[\frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2 \right]$$

- When training the model, find parameters (w_{opt}, b_{opt}) that minimize the total loss across all training examples.

$$w_{opt}, b_{opt} = \operatorname{argmin}_{w,b} L(w, b)$$

SINGLE-LAYER NEURAL NETWORK

Linear regression is a single-layer neural network.

- Number of inputs (or feature dimensionality) in the input layer is d . The inputs are x_1, \dots, x_d .
- Number of outputs in the output layer is 1. The output is \hat{y}_1 .
- Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.

SINGLE-LAYER NEURAL NETWORK

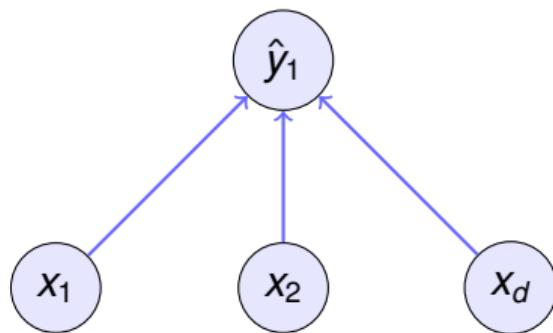
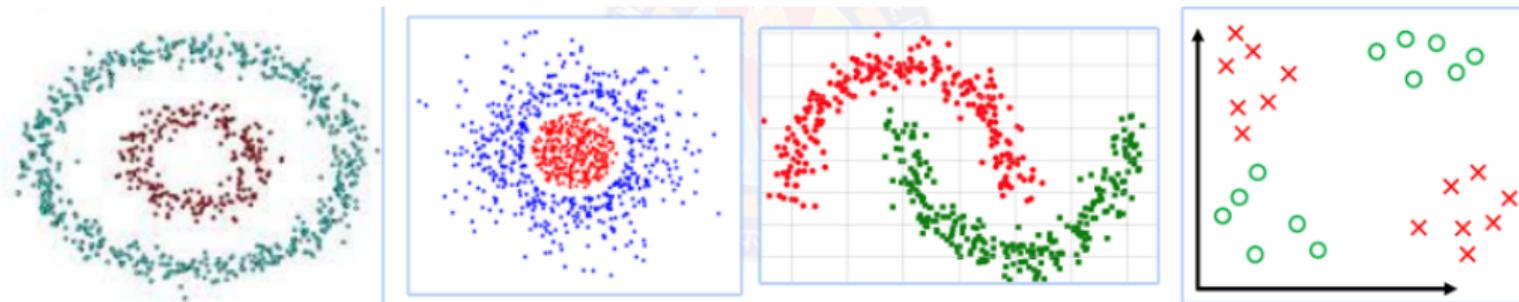


TABLE OF CONTENTS

- ① COURSE LOGISTICS
- ② INTRODUCTION TO DEEP LEARNING
- ③ APPLICATIONS OF DEEP LEARNING
- ④ KEY COMPONENTS OF DL PROBLEM
- ⑤ ARTIFICIAL NEURAL NETWORK
- ⑥ PERCEPTRON
- ⑦ SINGLE PERCEPTRON FOR LINEAR REGRESSION
- ⑧ MULTI LAYER PERCEPTRON (MLP)

NON-LINEARLY SEPARABLE DATA

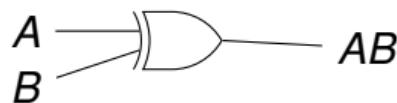
- Two groups of data points are non-linearly separable in a 2-dimensional space if they cannot be easily separated with a linear line.



XOR LOGIC GATE

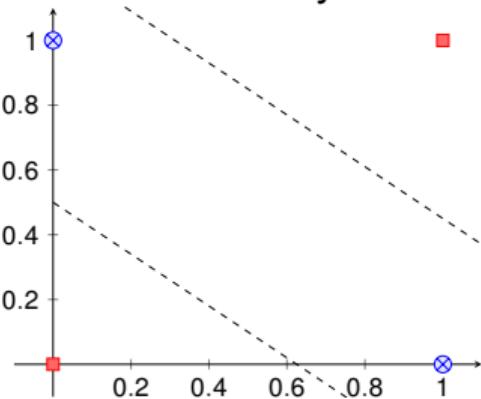
Question:

- How to represent XOR gate using a Perceptron?
- What are the parameters for the XOR Perceptron?
- Data is given below.



A	B	AB
0	0	0
0	1	1
1	0	1
1	1	0

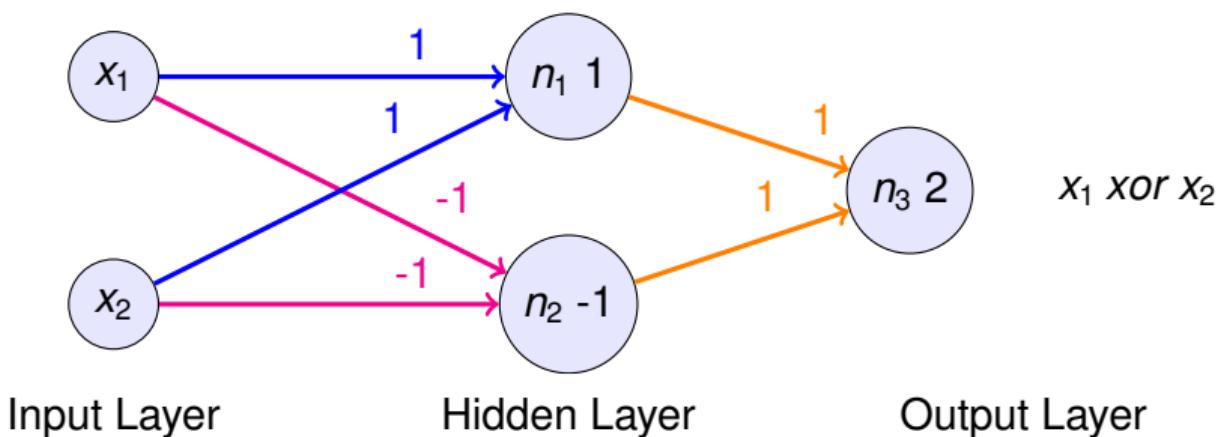
Challenge: Data is non-linearly separable.
Decision boundary for XOR



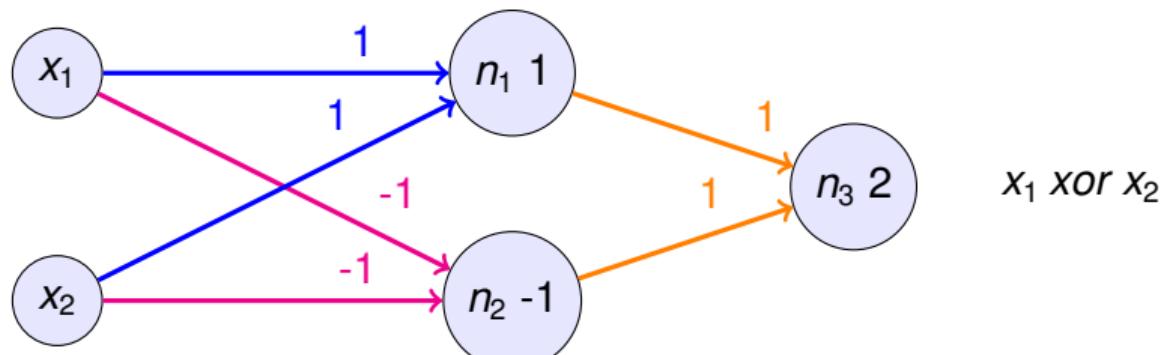
MLP FOR XOR GATE

Qn: How to represent XOR gate using a Perceptron?

- Use Multilayer Perceptron (MLP)
- Introduce another layer in between the input and output.
- This in-between layer is called hidden layer.



MLP FOR XOR GATE



x_1	x_2	n_1	n_2	n_3
0	0	$0 \cdot 1 + 0 \cdot 1 = 0 > th$ $n_1 = 0$	$0 \cdot (-1) + 0 \cdot (-1) = 0 > th$ $n_2 = 1$	$0 \cdot 1 + 1 \cdot 1 = 1 > th$ $n_3 = 0$
0	1	$0 \cdot 1 + 1 \cdot 1 = 1 \geq th$ $n_1 = 1$	$0 \cdot (-1) + 1 \cdot (-1) = -1 \geq th$ $n_2 = 1$	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$ $n_3 = 1$
1	0	$1 \cdot 1 + 0 \cdot 1 = 1 \geq th$ $n_1 = 1$	$1 \cdot (-1) + 0 \cdot (-1) = -1 \geq th$ $n_2 = 1$	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$ $n_3 = 1$
1	1	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$	$1 \cdot (-1) + 1 \cdot (-1) = -2 > th$	$1 \cdot 1 + 0 \cdot 1 = 1 > th$

SOLUTION OF XOR DATA

- Data
 - ▶ Truth table
- Model
 - ▶ Multi-layered Perceptron
- Challenge
 - ▶ How to learn the parameters and threshold?
- Solution for learning
 - ▶ Use gradient descent algorithm

GRADIENT DESCENT

To understand, consider simpler *linear unit*, where

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - \hat{y}_d)^2$$

where D is set of training examples.

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}] \quad \text{i.e.,} \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

GRADIENT DESCENT

$$\begin{aligned}
 \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - \hat{y}_d)^2 \\
 &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - \hat{y}_d)^2 \\
 &= \frac{1}{2} \sum_d 2(t_d - \hat{y}_d) \frac{\partial}{\partial w_i} (t_d - \hat{y}_d) \\
 &= \sum_d (t_d - \hat{y}_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
 \frac{\partial E}{\partial w_i} &= \sum_d (t_d - \hat{y}_d)(-x_{i,d})
 \end{aligned}$$

GRADIENT DESCENT ALGORITHM

Gradient-Descent(*training examples* $\langle \vec{x}, t \rangle$, learning rate η)

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - ▶ Initialize each Δw_i to zero.
 - ▶ For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - ★ Input the instance \vec{x} to the unit and compute the output \hat{y}
 - ★ For each linear unit weight w_i , Do

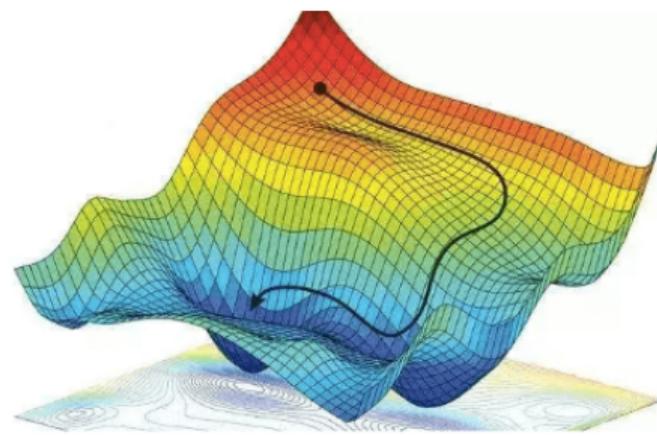
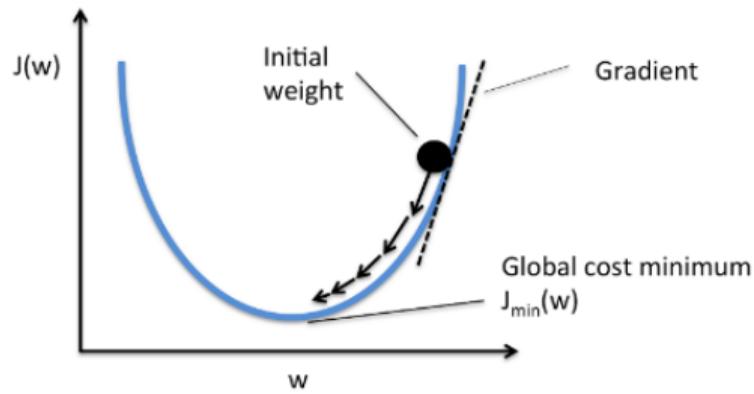
$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - \hat{y}_d)(-x_{i,d})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- ▶ For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i$$

GRADIENT DESCENT ALGORITHM



INCREMENTAL GRADIENT DESCENT ALGORITHM

- Do until satisfied
 - ▶ For each training example d in D
 - ★ Compute the gradient $\nabla E_d[\vec{w}]$
 - ★ $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_d[\vec{w}] \equiv \frac{1}{2}(t_d - \hat{y}_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough.

MINIBATCH STOCHASTIC GRADIENT DESCENT (SGD)

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.
- In each iteration,
 - ① Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.
 - ② Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
 - ③ Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$w \leftarrow w - \frac{\eta}{|B|} \sum_{i \in B} \partial_w \text{loss}^{(i)}(w, b) = w - \frac{\eta}{|B|} \sum_{i \in B} x^{(i)} \left(w^T x^{(i)} + b - y^{(i)} \right)$$

$$b \leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b \text{loss}^{(i)}(w, b) = b - \frac{\eta}{|B|} \sum_{i \in B} \left(w^T x^{(i)} + b - y^{(i)} \right)$$

TRAINING SGD

- Initialize parameters (w, b)
- Repeat until done
 - ▶ compute gradient

$$g \leftarrow \partial_{(w,b)} \frac{1}{|B|} \sum_{i \in B} (w^T x^{(i)} + b - y^{(i)})$$

- ▶ update parameters

$$(w, b) \leftarrow (w, b) - \eta g$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

NUMERICAL EXAMPLE OF SGD

- Equation is $y = (x + 5)^2$.
- When will it be minimum?
- Use gradient descent algorithm .
- Assume starting point as 3 and Learning rate as 0.01.
- Equations:

$$\frac{dy}{dx} = \frac{d}{dx}((x + 5)^2) = 2(x + 5)$$

$$x \leftarrow x - \eta \cdot \frac{dy}{dx}$$

- Epoch 1:

$$x \leftarrow 3 - 0.01 * 2(3 + 5) = 2.84$$

- Epoch 2:

$$x \leftarrow 2.84 - 0.01 * 2(2.84 + 5) = 2.68$$

Further Reading

- ① Chapter 1 of Dive into Deep Learning (T1)
- ② Chapter 2 for Python Prelims, Linear Algebra, Calculus, Probability (T1)
- ③ Chapter 4 of Book: Machine Learning by Tom M. Mitchell

Thank You!



DEEP LEARNING MODULE # 1 : MLP AS UNIVERSAL APPROXIMATORS

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

TABLE OF CONTENTS

1 MULTI LAYER PERCEPTRON (MLP)

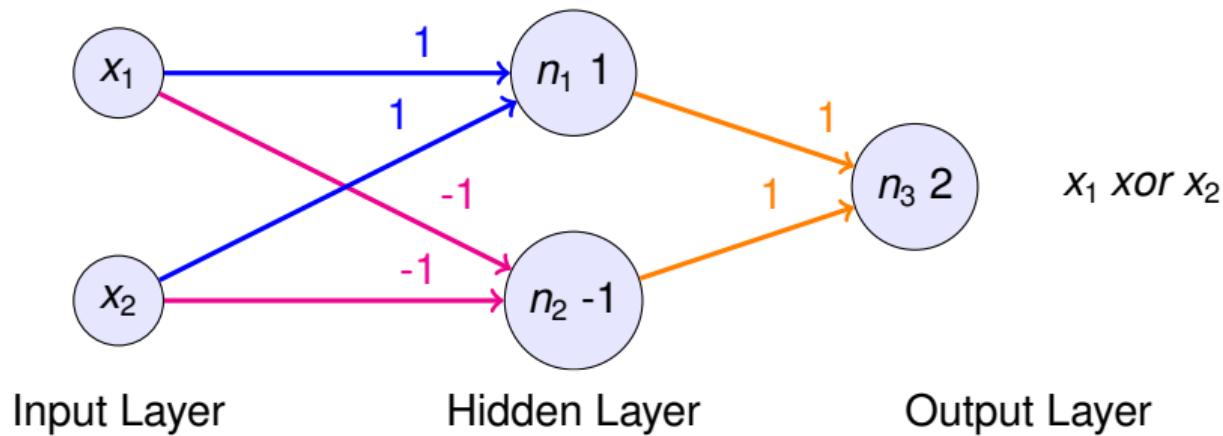
POWER OF MLP

MLP can be used for

- classification
 - ▶ binary
 - ▶ multiclass
- regression
 - ▶ real output
- representing complex decision boundaries
- representing boolean functions

MLP is called as Universal Approximator for the above reasons.

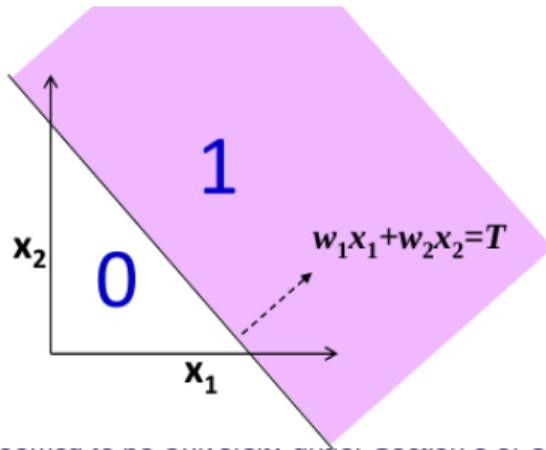
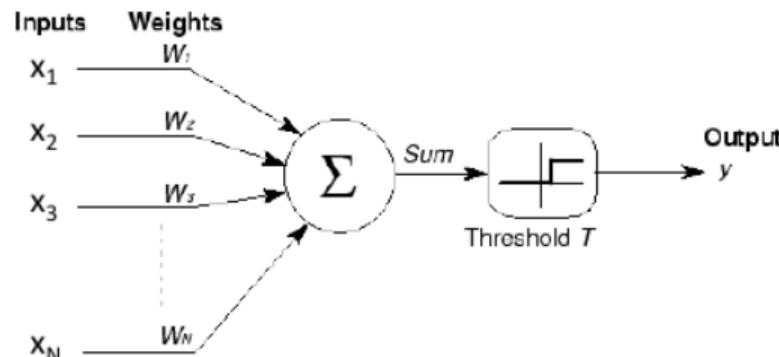
MLP FOR XOR GATE



PERCEPTRON WITH REAL INPUTS

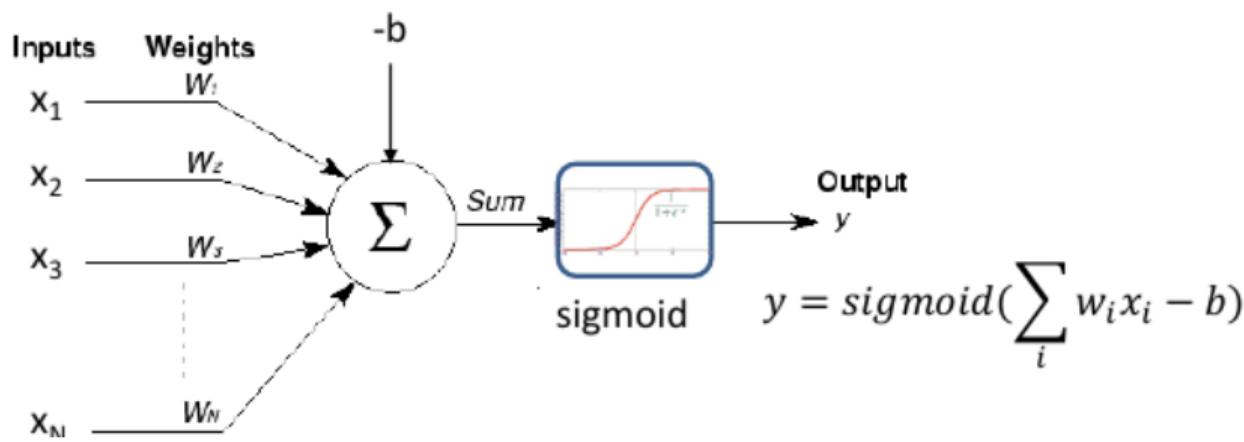
- x_1, \dots, x_N are real valued.
- W_1, \dots, W_N are real valued.
- Unit "fires" if weighted input exceeds a threshold.

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{otherwise} \end{cases}$$



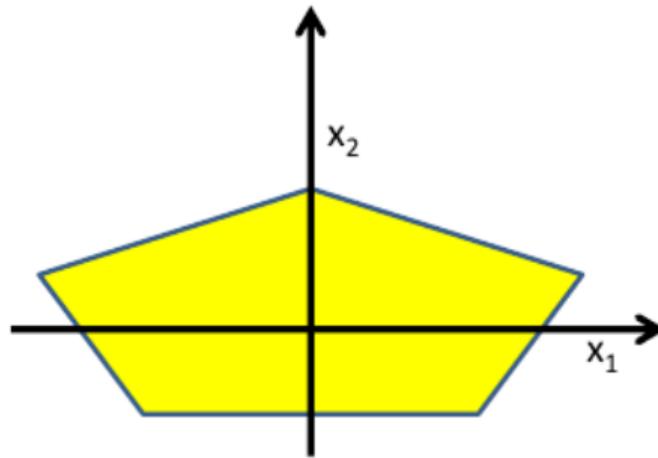
PERCEPTRON WITH REAL OUTPUTS

- x_1, \dots, x_N are real valued.
- W_1, \dots, W_N are real valued.
- The output can also be real valued. – Sometimes viewed as the "probability" of firing.



MLP FOR COMPLICATED DECISION BOUNDARIES

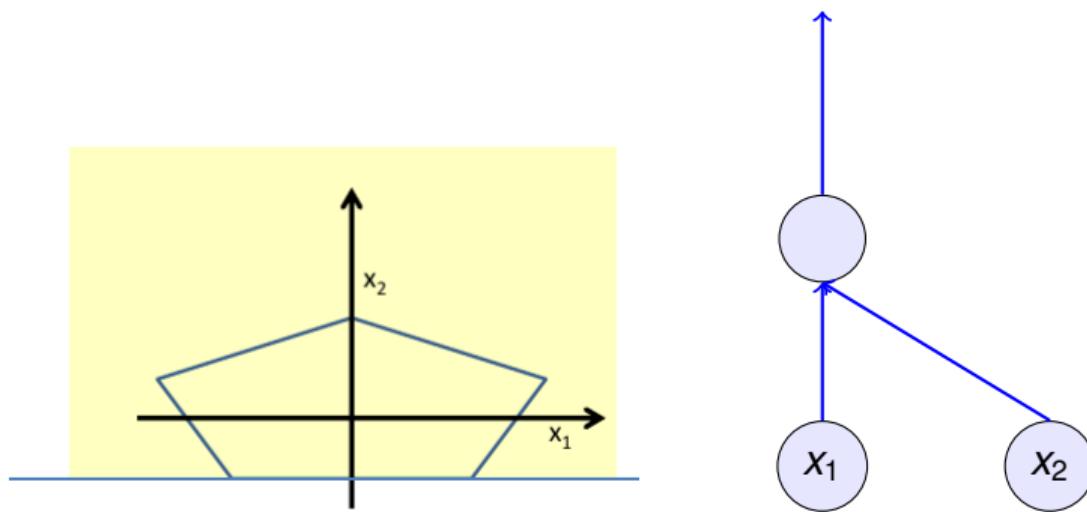
- Build a network of units with a single output that fires if the input is in the coloured area.



Can now be composed into
“networks” to compute arbitrary
classification “boundaries”

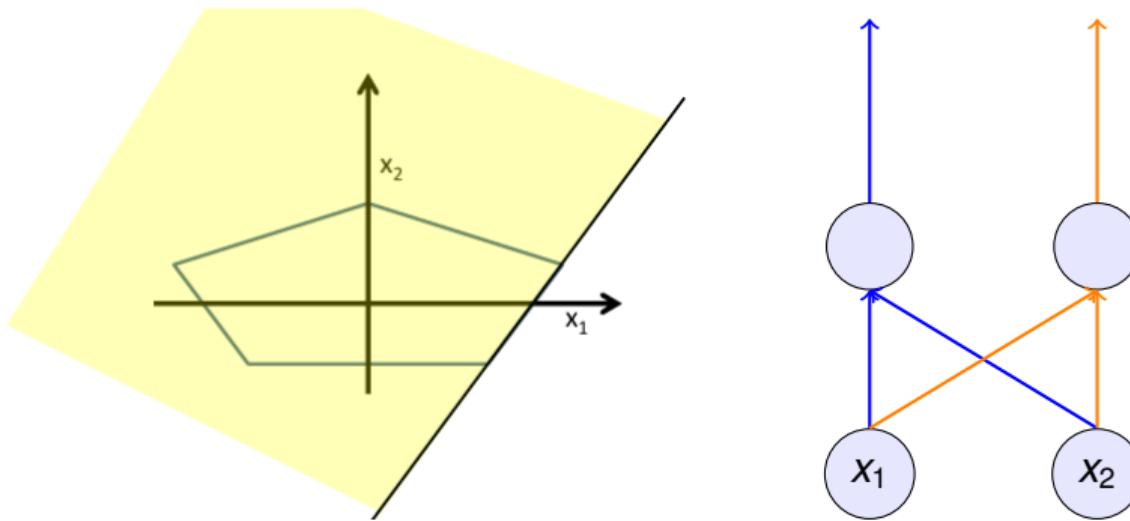
MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.



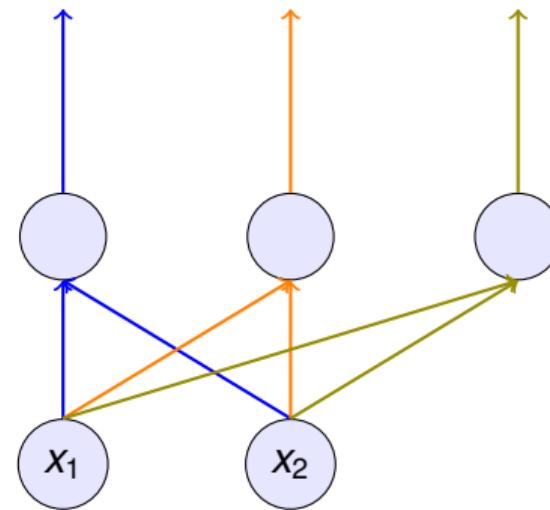
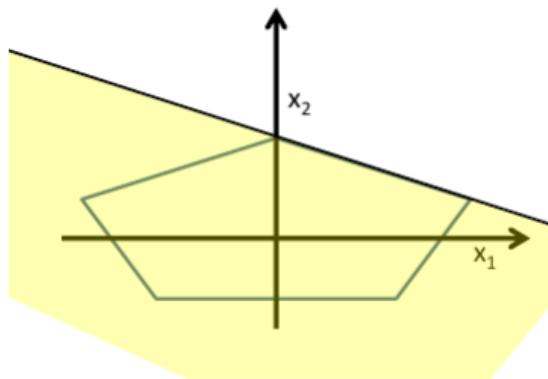
MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.



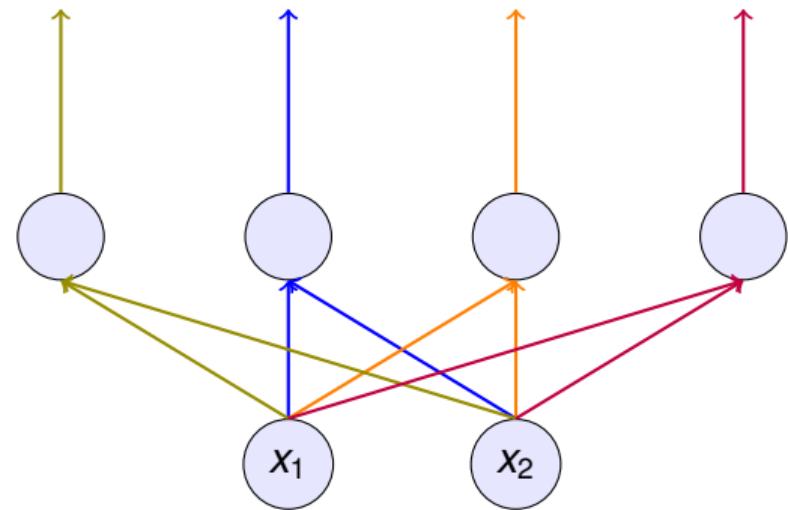
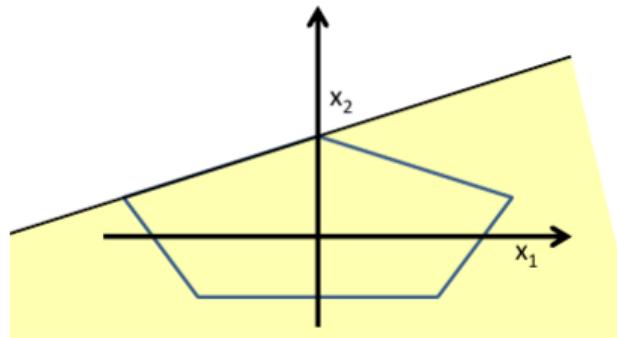
MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.



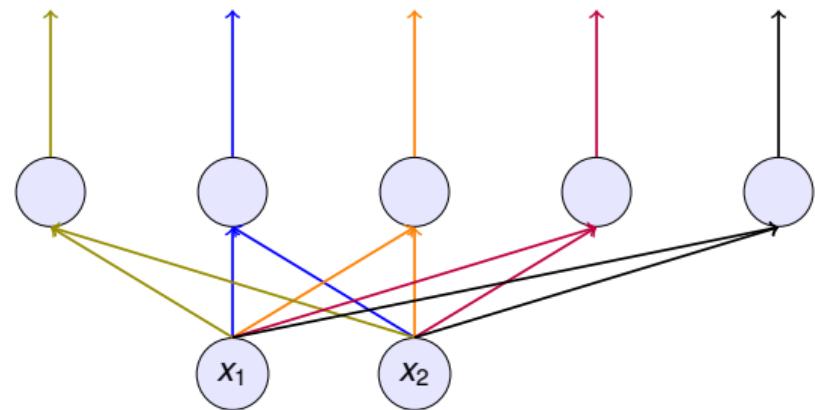
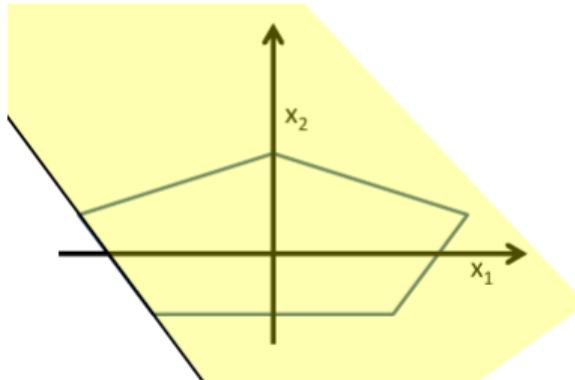
MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.



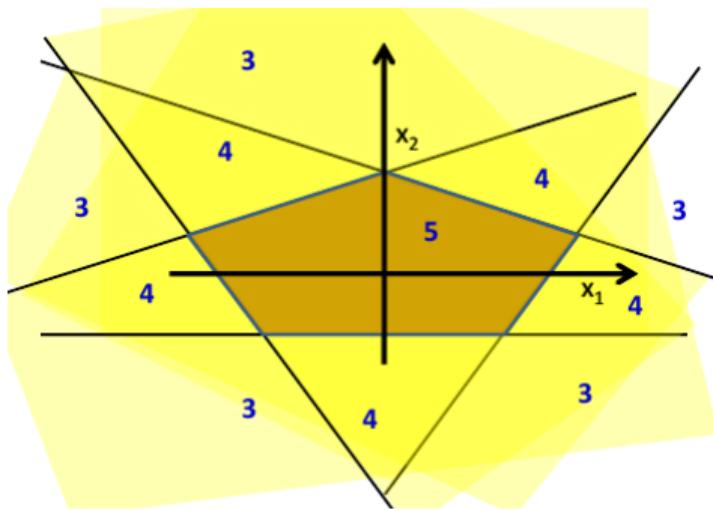
MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.

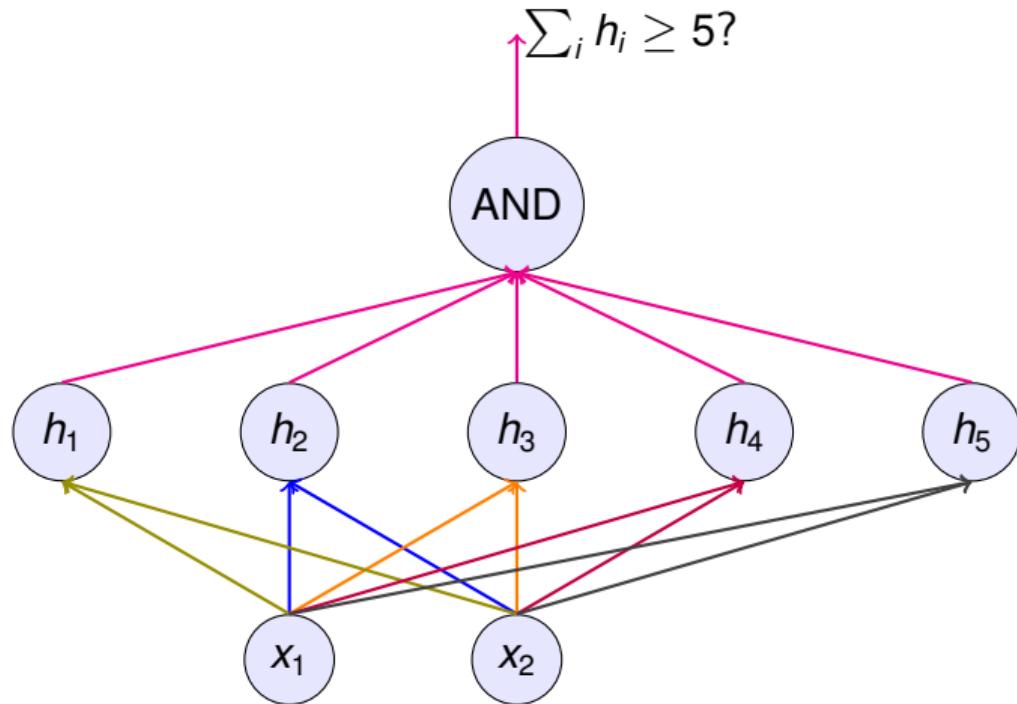


MLP FOR COMPLICATED DECISION BOUNDARIES

- The network must fire if the input is in the coloured area.

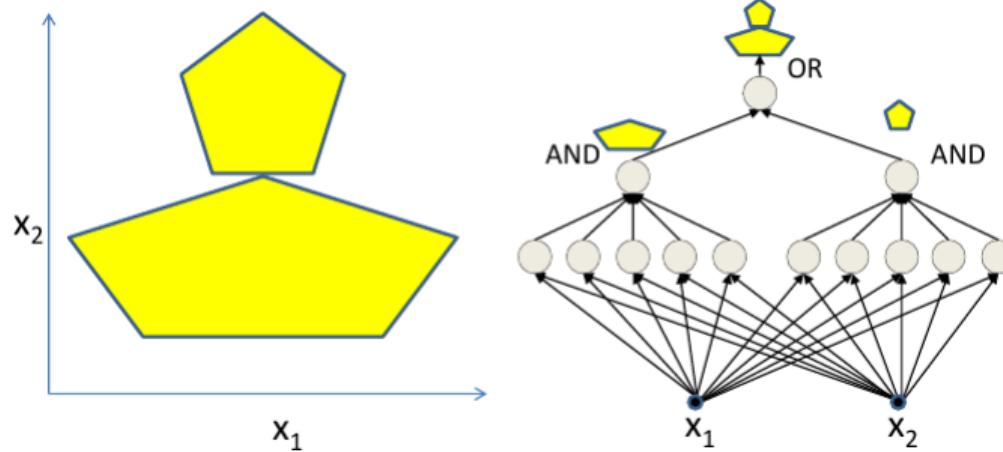


MLP FOR COMPLICATED DECISION BOUNDARIES



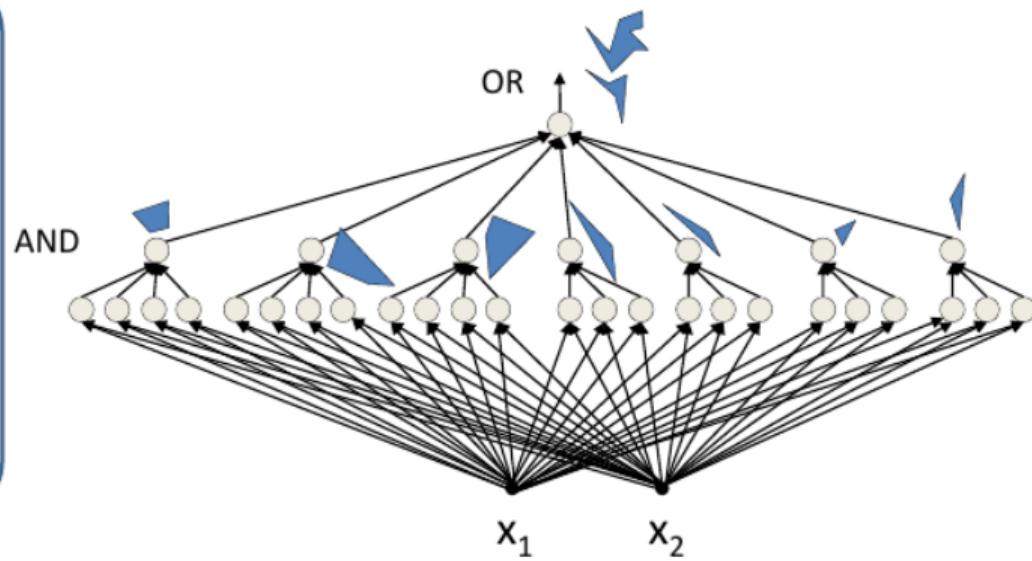
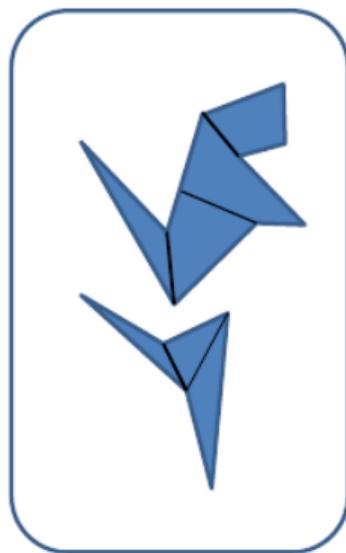
MLP FOR COMPLICATED DECISION BOUNDARIES

- Network to fire if the input is in the yellow area.
 - ▶ "OR" two polygons
 - ▶ A third layer is required



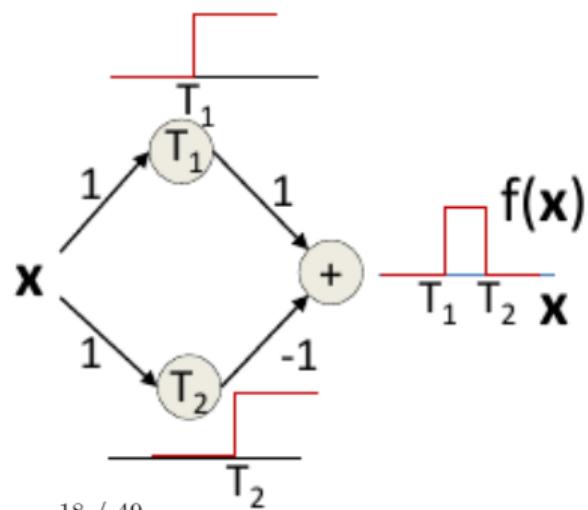
MLPs FOR COMPLEX DECISION BOUNDARIES

- MLPs can compose arbitrarily complex decision boundaries with only one hidden layer.



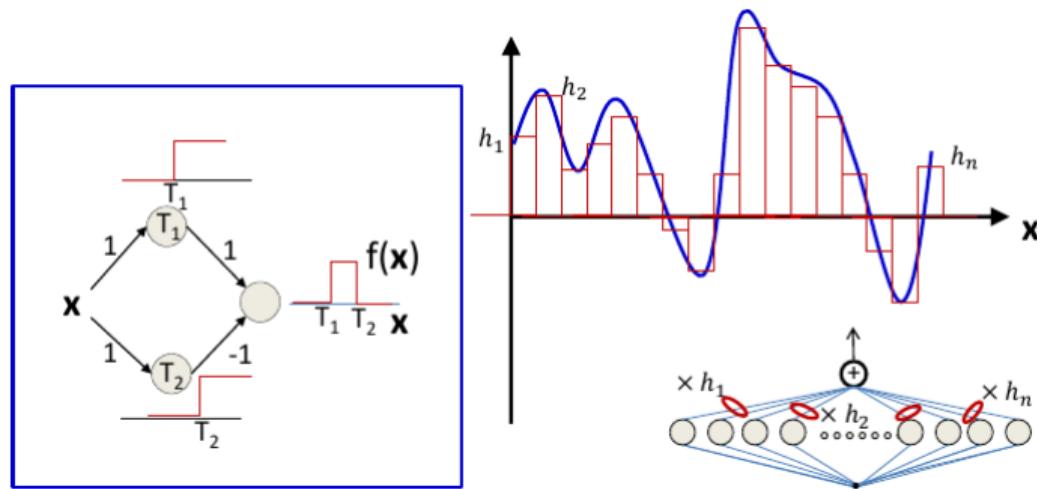
MLP AS CONTINUOUS-VALUED REGRESSION

- A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input.
 - ▶ Output is 1 only if the input lies between T_1 and T_2 .
 - ▶ T_1 and T_2 can be arbitrarily specified



MLP AS CONTINUOUS-VALUED REGRESSION

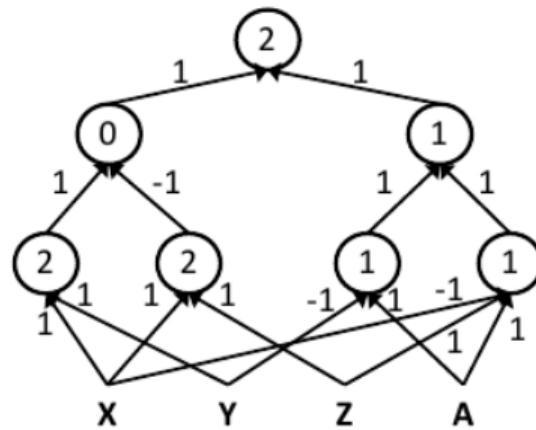
- A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input.
- An MLP with many units can model an arbitrary function over an input.
- This generalizes to functions of any number of inputs.



MLPs ARE UNIVERSAL BOOLEAN FUNCTIONS

- MLPs can compute more complex Boolean functions.
- MLPs can compute any Boolean function; since they can emulate individual gates.

$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth Table

Truth table shows all input combinations for which output is 1

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations
for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + \\ X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

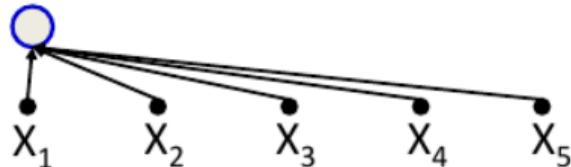
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



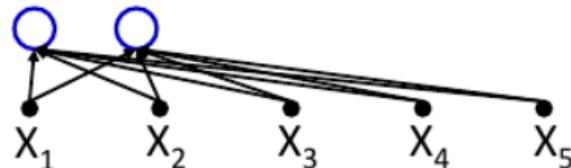
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \textcircled{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1X_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



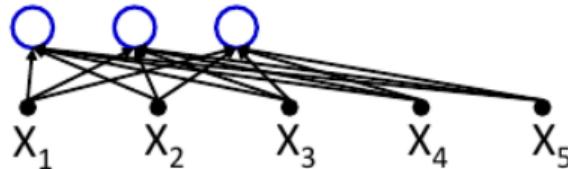
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations
for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \textcircled{\bar{X}_1X_2X_3\bar{X}_4\bar{X}_5} + \\ X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3X_4X_5$$



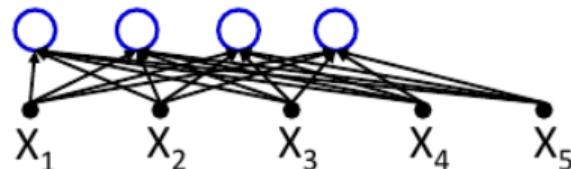
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations
for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + \\ \textcircled{X}_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



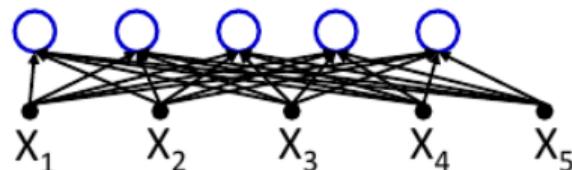
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations
for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + \\ X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + \textcircled{X}_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



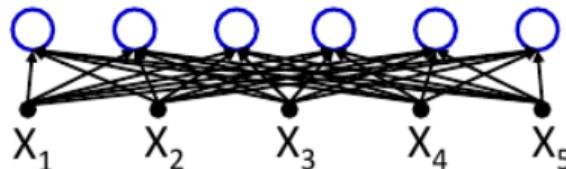
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + \\ X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + \textcircled{X}_1X_2\bar{X}_3\bar{X}_4X_5$$



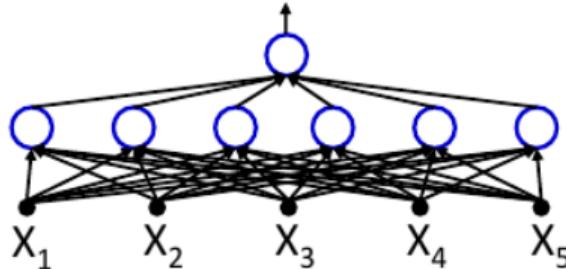
HOW MANY LAYERS FOR A BOOLEAN MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows all input combinations for which output is 1

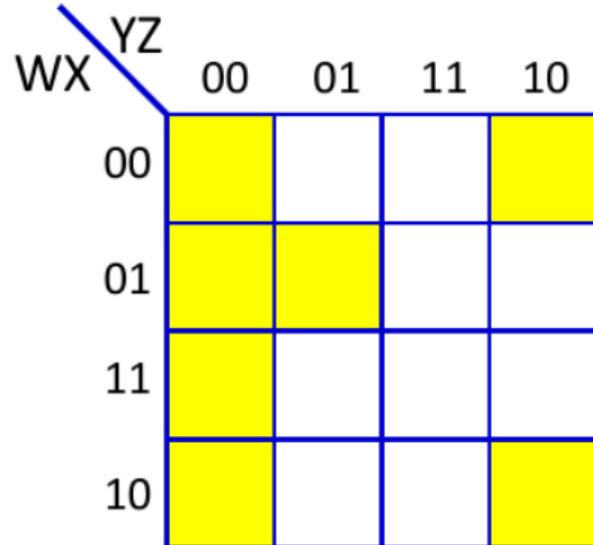
Truth Table					
X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



- Any truth table can be expressed in this manner.
- A one-hidden-layer MLP is a Universal Boolean Function.

REDUCING A BOOLEAN FUNCTION



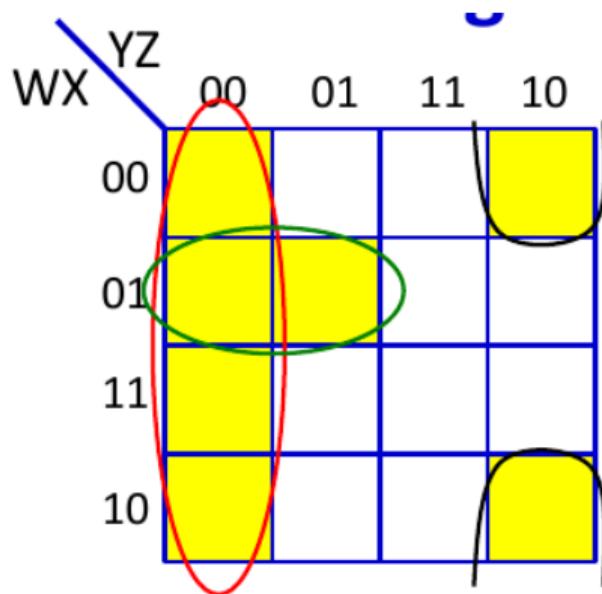
This is a "Karnaugh Map"

It represents a truth table as a grid
Filled boxes represent input combinations
for which output is 1; blank boxes have
output 0

Adjacent boxes can be "grouped" to
reduce the complexity of the DNF formula
for the table

REDUCING A BOOLEAN FUNCTION

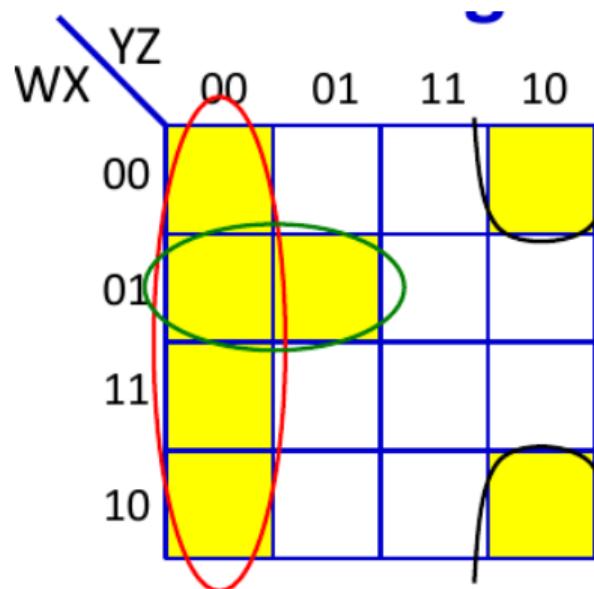
- Use K-map to reduce the DNF to find groups and express as reduced DNF.



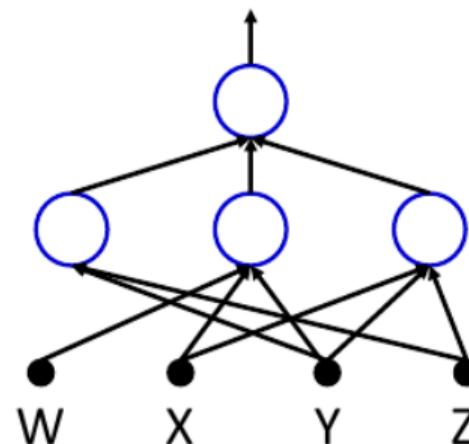
$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

REDUCING A BOOLEAN FUNCTION

- Use K-map to reduce the DNF to find groups and express as reduced DNF.



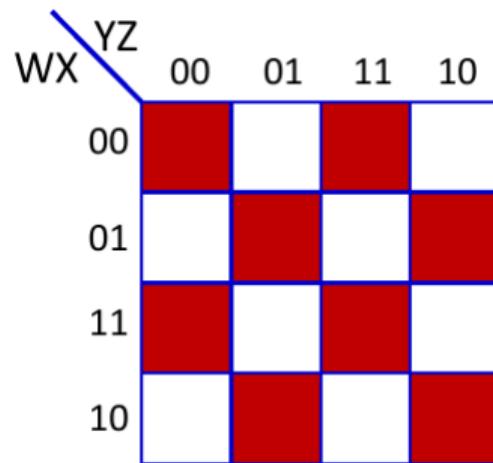
$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$



- *Reduced DNF form:*

LARGEST IRREDUCIBLE DNF?

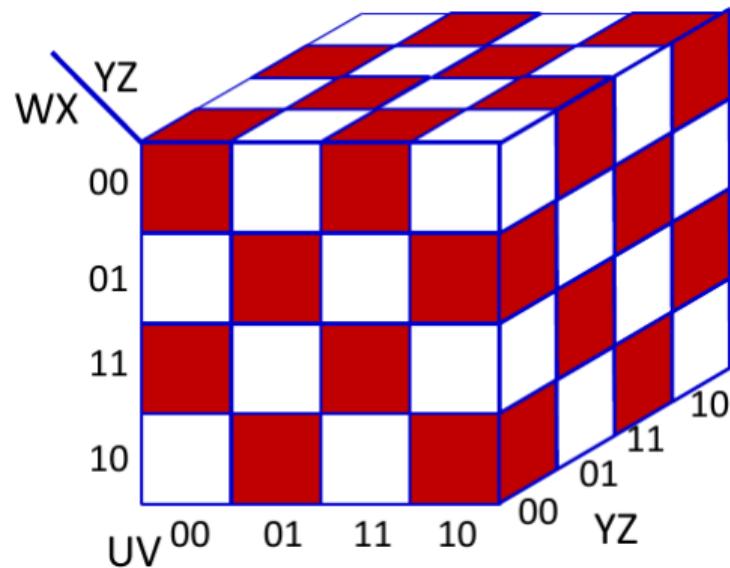
- What arrangement of ones and zeros simply cannot be reduced further?



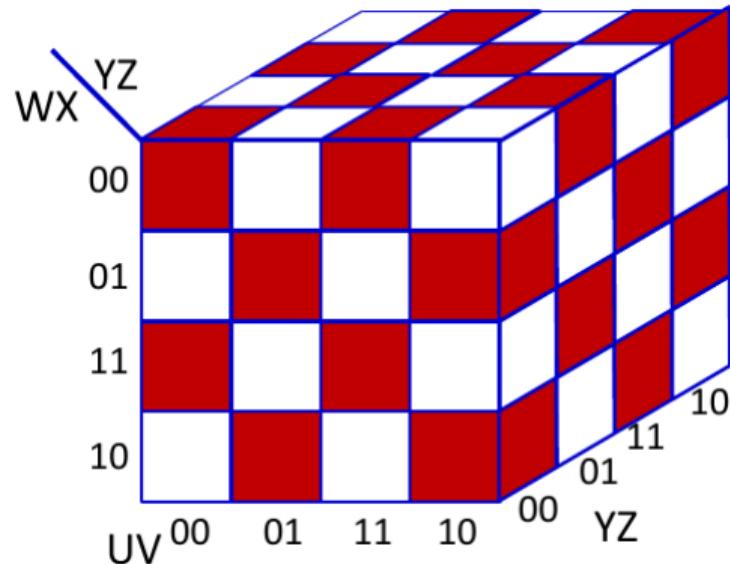
- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function?

WIDTH OF A SINGLE-LAYER BOOLEAN MLP

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

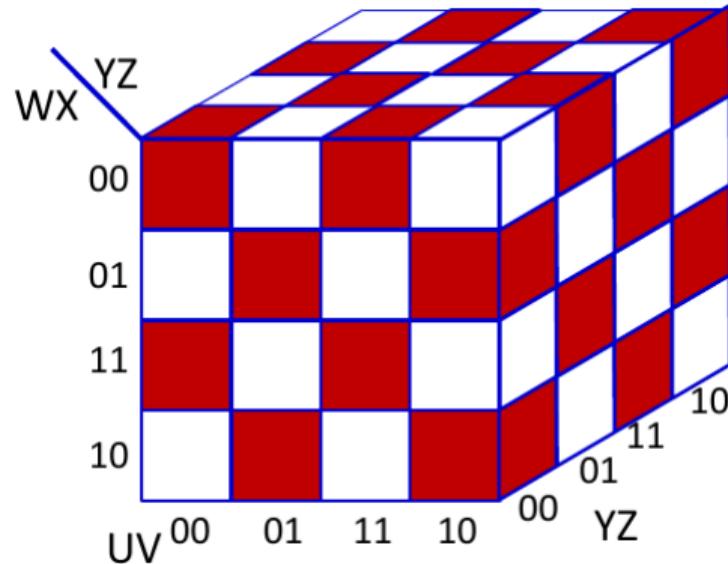


WIDTH OF A SINGLE-LAYER BOOLEAN MLP



- In general, XOR of N variables will require 2^{N-1} perceptrons in single hidden layer. **Exponential in N**

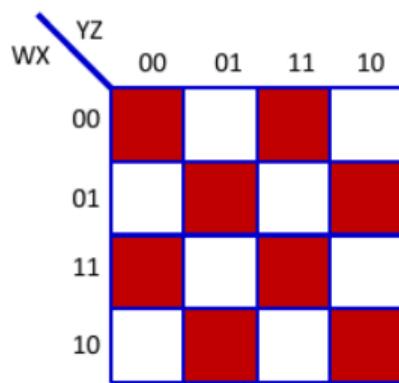
WIDTH OF A SINGLE-LAYER BOOLEAN MLP



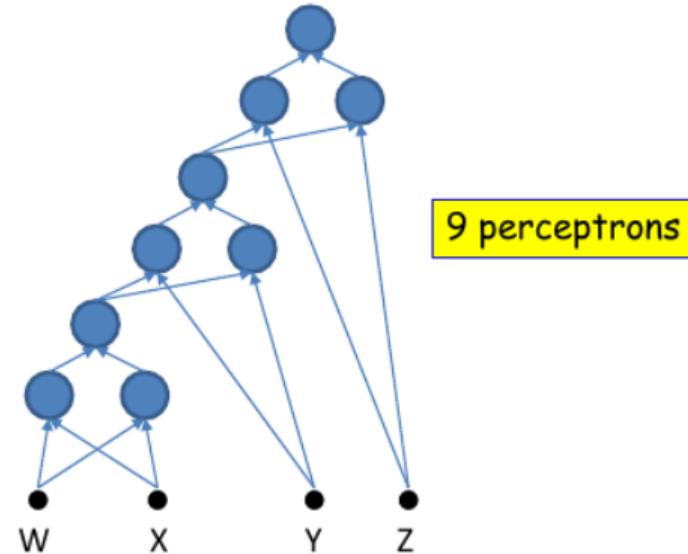
- In general, XOR of N variables will require 2^{N-1} perceptrons in single hidden layer.
- **How many units if we use multiple layers?**

WIDTH OF A DEEP BOOLEAN MLP

- An XOR needs 3 perceptrons.

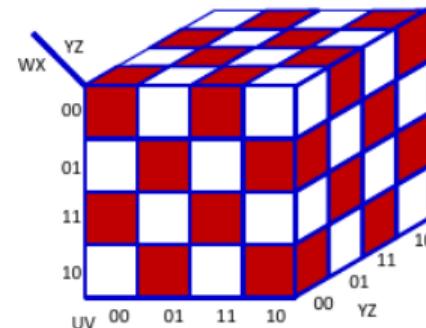
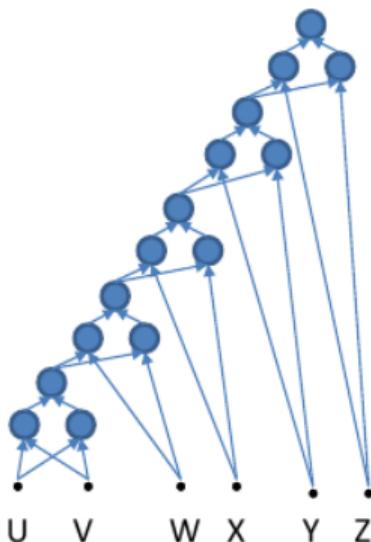


$$O = W \oplus X \oplus Y \oplus Z$$



- This network will require $3 \times 3 = 9$ perceptrons.

WIDTH OF A DEEP BOOLEAN MLP

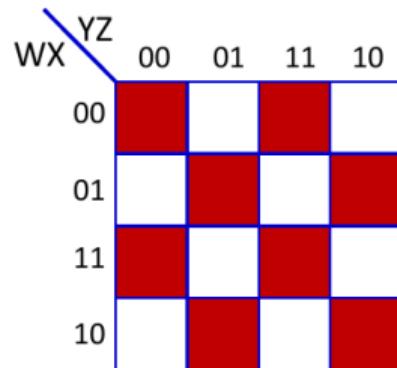


$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

15 perceptrons

- This network will require $3 \times 5 = 9$ perceptrons.
- In general, the XOR of N variables will require $3(N - 1)$ perceptrons.

WIDTH OF A DEEP BOOLEAN MLP



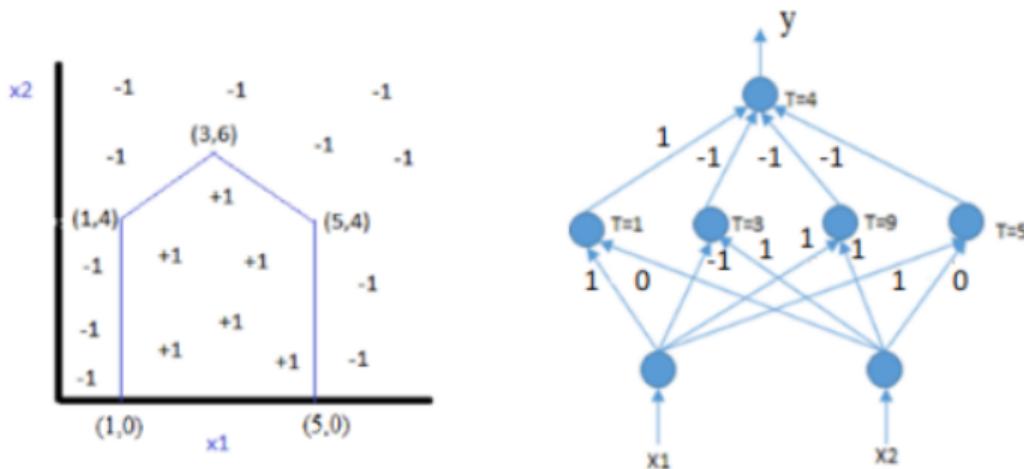
- Single hidden layer: Will require $2^{N-1} + 1$ perceptrons in all (including output unit). Exponential in N.
- Deep network: Will require $3(N - 1)$ perceptrons in a deep network. Linear in N.

This can be arranged in only $2\log_2(N)$ layers.

SUMMARY OF MLP

- MLPs are connectionist computational models,
 - ▶ Individual perceptrons are computational equivalent of neurons.
 - ▶ The MLP is a layered composition of many perceptrons.
- MLP can model universal Boolean functions
 - ▶ Individual perceptrons can act as Boolean gates.
 - ▶ Networks of perceptrons are Boolean functions.
- MLPs are Boolean machines.
 - ▶ They represent Boolean functions over linear boundaries.
 - ▶ They can represent arbitrary decision boundaries.
 - ▶ They can be used to classify data.
- MLP can model continuous valued functions.
- MLPs as universal classifiers.
- MLPs as universal approximators.

EXERCISE



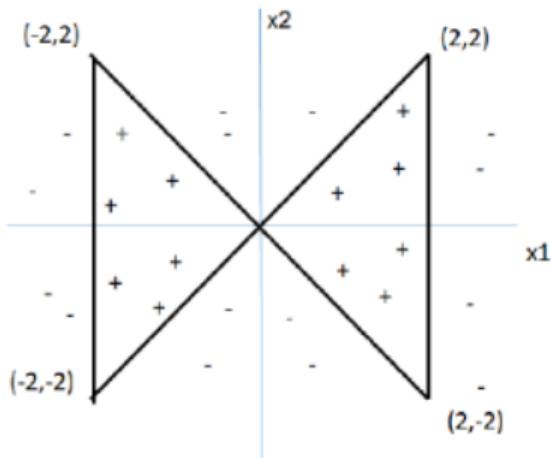
- Note: Different Choices of weights and bias are possible.
- Left hidden node implements $x_1=1$ line
- Right hidden node implements $x_1=5$ line
- 2nd node from left implements $x_2=x_1+3$ and 3rd from left implements $x_1+x_2=9$
- For + class, output of left hidden node = $+1$, for other nodes output = -1

EXERCISE

(x_1, x_2) are input features and target classes are either +1 or -1 as shown in the figure.

A. What is the minimum number of hidden layers and hidden nodes required to classify the following dataset with 100% accuracy using a fully connected multilayer perceptron network? Step activation functions are used at all nodes, i.e., output = +1 if total weighted input \geq bias b at a node, else output = -1.

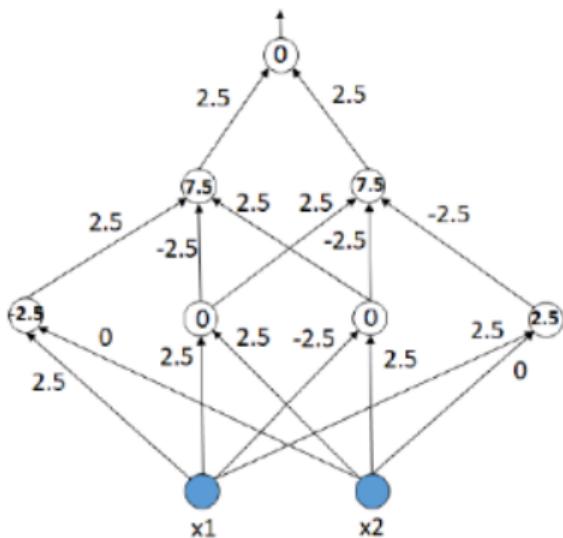
B. Show the minimal network architecture by organizing the nodes in each layer horizontally. Show the node representing x_1 at the left on the input layer. Organize the hidden nodes in ascending order of bias at that node. Specify all weights and bias values at all nodes. Weights can be only -2.5, 2.5 or 0, and bias +ve/-ve multiples of 2.5.



EXERCISE - SOLUTION

A. 2 hidden layers, 4 nodes in first hidden layer and 2 nodes in second hidden layer needed.

B.

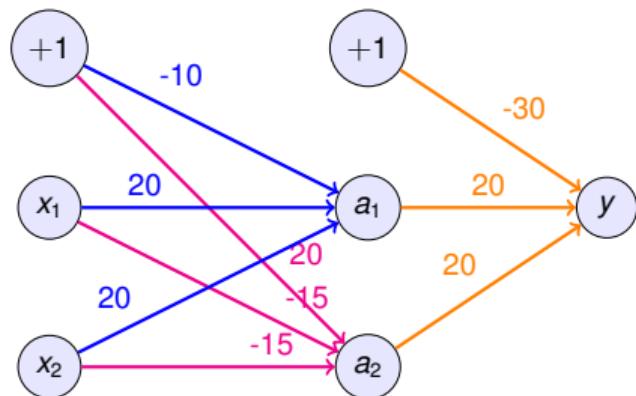


EXERCISE

An XOR cannot be represented using a single perceptron. If this statement is true, demonstrate the alternative way to represent XOR using an ANN.

EXERCISE - SOLUTION

An XOR cannot be represented using a single perceptron as the XOR has non-linear decision boundary. To represent non-linear decision boundary we need a multi-layer perceptron with one hidden layers.



$$a_k = \text{sign}(w_{k1}x_1 + w_{k2}x_2 + b)$$

x_1	x_2	a_1	a_2	y
0	0	-10 ~ 0	20 ~ 1	-10 ~ 0
0	1	10 ~ 1	5 ~ 1	10 ~ 1
1	0	10 ~ 1	5 ~ 1	10 ~ 1
1	1	30 ~ 1	-30 ~ 0	-10 ~ 0

EXERCISE

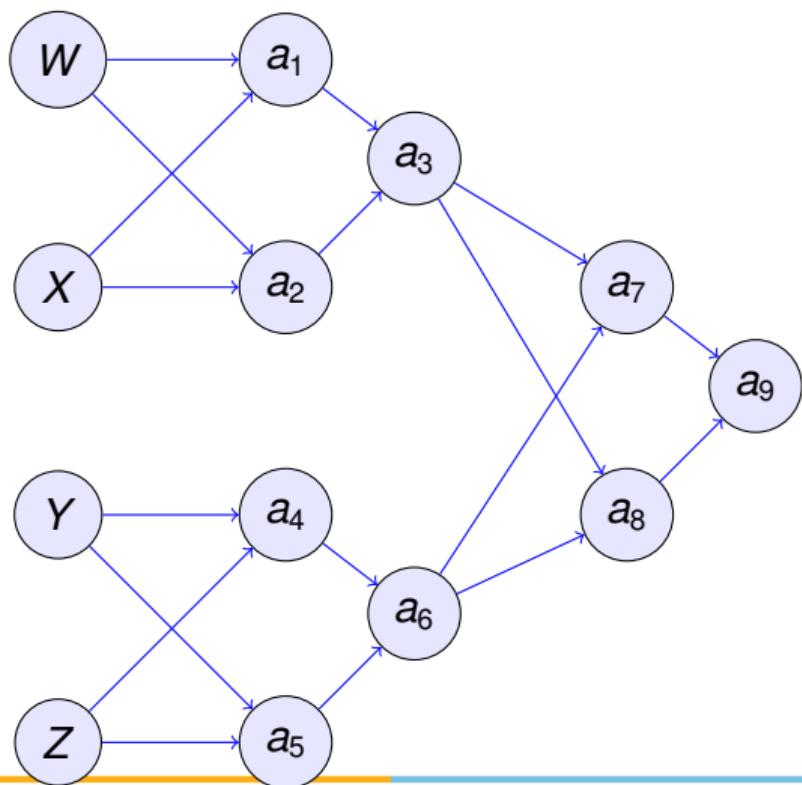
How many perceptrons are required to represent $W \oplus X \oplus Y \oplus Z$?

EXERCISE - SOLUTION

An XOR cannot be represented using a single perceptron as the XOR has non-linear decision boundary. To represent non-linear decision boundary we need a multi-layer perceptron with one hidden layers.

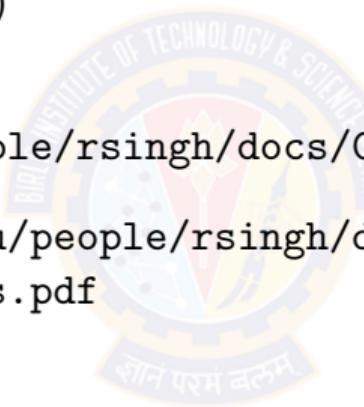
- Traditional method:
 - ▶ Number of variables $n = 4$
 - ▶ Number of perceptrons $= 2^n - 1 = 2^4 - 1 = 15$
- Alternate method:
 - ▶ Layer 1: $A = W \oplus X$
 - ▶ Layer 2: $B = Y \oplus Z$
 - ▶ Layer 3: $C = A \oplus B$
 - ▶ Number of perceptrons $= 3(n - 1) = 3 * (4 - 1) = 9$
 - ▶ Number of layers = Width $= 2 \log_2 n = 2 \log_2 4 = 4$

EXERCISE - SOLUTION



Further Reading

- ① Dive into Deep Learning (T1)
- ② http://mlsp.cs.cmu.edu/people/rsingh/docs/Chapter1_Introduction.pdf
- ③ http://mlsp.cs.cmu.edu/people/rsingh/docs/Chapter2_UniversalApproximators.pdf



Thank You!



DEEP LEARNING MODULE # 2 : DEEP FEEDFORWARD NEURAL NETWORK



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

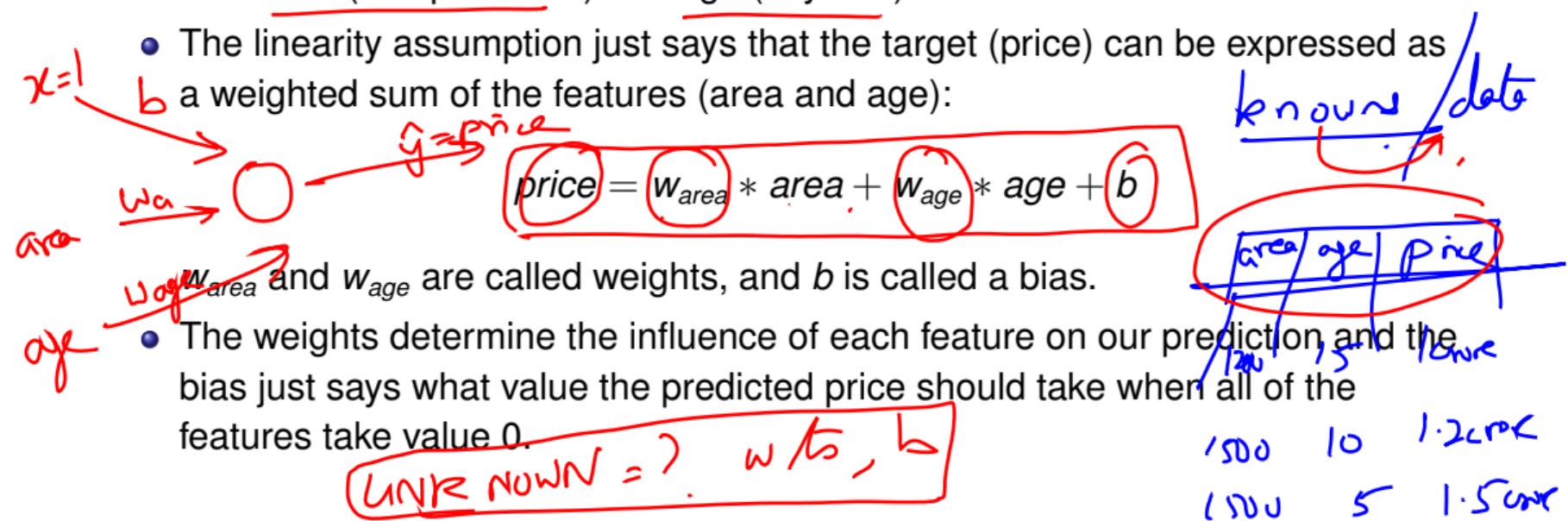
The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

LINEAR REGRESSION EXAMPLE

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):



DATA

x_1 area	x_2 age	y
1200	15	1 or 1.5 ✓
1500	10	1.5 ✓

- The dataset comprises of training dataset and testing dataset. The split of DL is generally 99:1, as we are dealing with millions of examples.
- Each row is called an example (or **data point**, **data instance**, **sample**).
- The thing we are trying to predict is called a **label** (or **target**).
- The independent variables upon which the predictions are based are called features (or **covariates**).

 m exs m — number of training examples d feature i — i^{th} example $x_1 \ x_2 \ x_3 \dots x_d$  x, t $x^{(i)} = [x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(m)}]$ — features of i^{th} example $y^{(i)} = [y_1^{(1)}, y_1^{(2)}, \dots, y_1^{(m)}]$ — label of i^{th} example

AFFINE TRANSFORMATIONS AND LINEAR MODELS

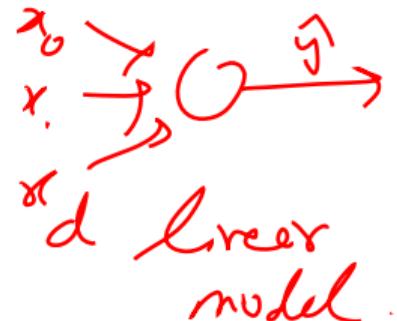
- The equation of the form

1.2
1.5

$$\hat{y} = \underbrace{w_1 x_1 + \dots + w_d x_d}_\text{wtd sum} + b$$

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b \quad \mathbf{x} \in \mathbb{R}^d \quad \mathbf{w} \in \mathbb{R}^d$$

$$\hat{y} = \mathbf{X}\mathbf{w} + b \quad \mathbf{X} \in \mathbb{R}^{m \times d} \quad \hat{y} \in \mathbb{R}^m$$



is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.

- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen weights (w) and bias (b).

LOSS FUNCTION

given data (area, age, price)
innovate achieve lead
compute ~~ptrs~~ ~~w_{age}, w_{area}, b~~

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the squared error.

$$\text{loss}^{(i)} = (y^{(i)} - \hat{y}^{(i)})^2$$

SQUARED ERROR LOSS FUNCTION

- The most popular loss function in regression problems is the squared error.
- For each example,

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

1000

- For the entire dataset of m examples, average (or equivalently, sum) the losses

$$\text{cost} = \mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m l^{(i)}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

- When training the model, find parameters $(\mathbf{w}_{opt}, b_{opt})$ that minimize the total loss across all training examples.

$$(\mathbf{w}_{opt}, b_{opt}) = \arg \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b)$$

MINIBATCH STOCHASTIC GRADIENT DESCENT (SGD)

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.
 - In each iteration, $\# \text{ batches} \approx 1000$ $1 \text{ million} = 2^{20} \text{ eg}$
- ▶ Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.
 ▶ Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
 ▶ Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|B|} \sum_{i \in B} \partial_w \text{loss}^{(i)}(\mathbf{w}, b)$$

$$b \leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b \text{loss}^{(i)}(\mathbf{w}, b)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$$

$$\mathbf{g} = \frac{\partial \text{loss}}{\partial \mathbf{w}}$$

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

w_{opt} , b_{opt}

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

PREDICTION

goal : estimate price w.r.t NN
 (linearreg)
 learn $\theta(\text{ptre})$

- Estimating targets given features is commonly called **prediction or inference**.
- Given the learned model, values of target can be predicted, for any set of features.

loss fn: Sq error
 learning algo: Sgd.

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

BINARY CLASSIFICATION – TRAINING EXAMPLES

area, age

Single training example = $\{(x, y)\}$ where $x \in \mathcal{R}^d$ and $y \in \{0, 1\}$

training examples = m

m training examples = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$

$$X = x_1, x_2$$

$$\begin{bmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \\ x_d^{(1)} & x_d^{(2)} \end{bmatrix}$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \text{where } X \in \mathcal{R}^{d \times m}$$

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}] \quad \text{where } Y \in \mathcal{R}^{1 \times m}$$

FORWARD PROPAGATION

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Given X find $\hat{y} = P(y = 1 | X)$

Input $X \in \mathbb{R}^{d \times m}$

Parameters $w \in \mathbb{R}^d$

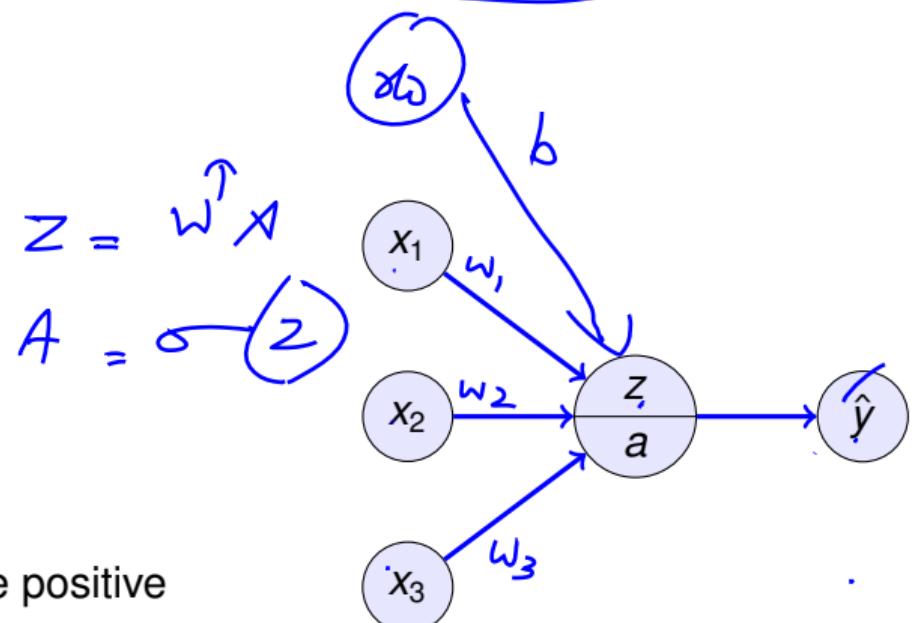
$b \in \mathbb{R}$

Activation $z = w^\top x + b$

Activation function $\sigma(z) = \frac{1}{1 + e^{-z}}$

$$= \begin{cases} 1 & \text{if } z \text{ is large positive} \\ 0 & \text{if } z \text{ is large negative} \end{cases}$$

Output $\hat{y} = \sigma(w^\top x + b) \quad 0 \leq \hat{y} \leq 1$



COST FUNCTION FOR BINARY CLASSIFICATION

Loss function $\text{loss}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

If $y = 1$ $\text{loss}(\hat{y}, y) = -\log \hat{y}$

$\text{loss} \approx 0 \Rightarrow \log \hat{y} \approx \text{large} \Rightarrow \hat{y} \approx \text{large}$

If $y = 0$ $\text{loss}(\hat{y}, y) = -\log(1 - \hat{y})$

$\text{loss} \approx 0 \Rightarrow \log(1 - \hat{y}) \approx \text{small} \Rightarrow \hat{y} \approx \text{small}$

$$y \log \hat{y}$$

$$1 \log 0 = \infty$$

$$(1-y) \log(1-\hat{y})$$

$$(1-0) \log(1-1) = \infty$$

Cost function $\mathcal{L}(w, b) = \frac{1}{m} \sum_{i=1}^m \text{loss}(\hat{y}^{(i)}, y^{(i)})$

$$= \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

HOW TO LEARN PARAMETERS?

To Learn parameters use Gradient Descent Algorithm

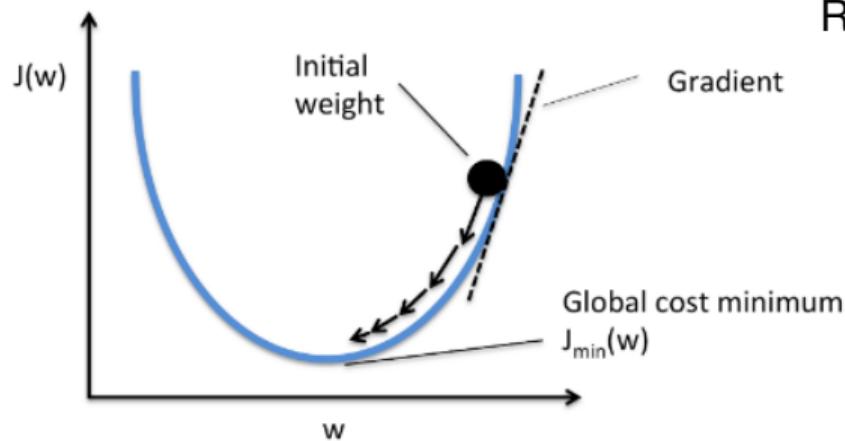
$$\hat{y} = \sigma(w^T x + b) \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(w, b) = \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Find w_{opt}, b_{opt} that minimize $\mathcal{L}(w, b)$.

GRADIENT DESCENT

Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$



Repeat {

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial w}$$

$$b \leftarrow b - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b}$$

}

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

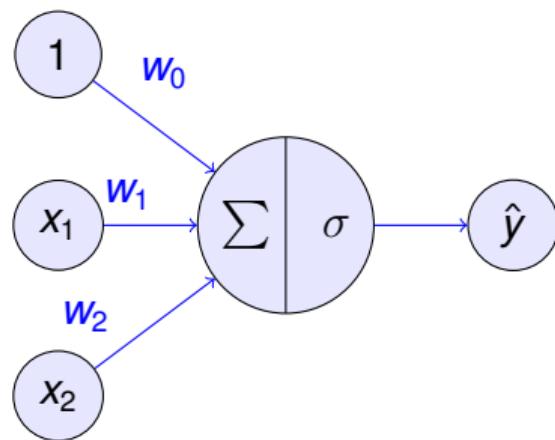
- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

EXERCISE - MSE LOSS

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, mean squared loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.



EXERCISE - MSE LOSS SOLUTION

$$\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$MSE = \frac{1}{2m} \sum_{i=1}^m (y - \hat{y})^2 = \frac{1}{2}(1 - 0.7858)^2 = \frac{0.04588}{2} = 0.02294$$

$$\mathbf{g} = \frac{2}{2m} \sum_{i=1}^m (y - \hat{y}) = (1 - 0.7858) = 0.2142$$

new w \leftarrow *old w* $- \eta * \mathbf{g}$

$$w_0 = 0.5 - 0.01 * 0.2142 = 0.4978$$

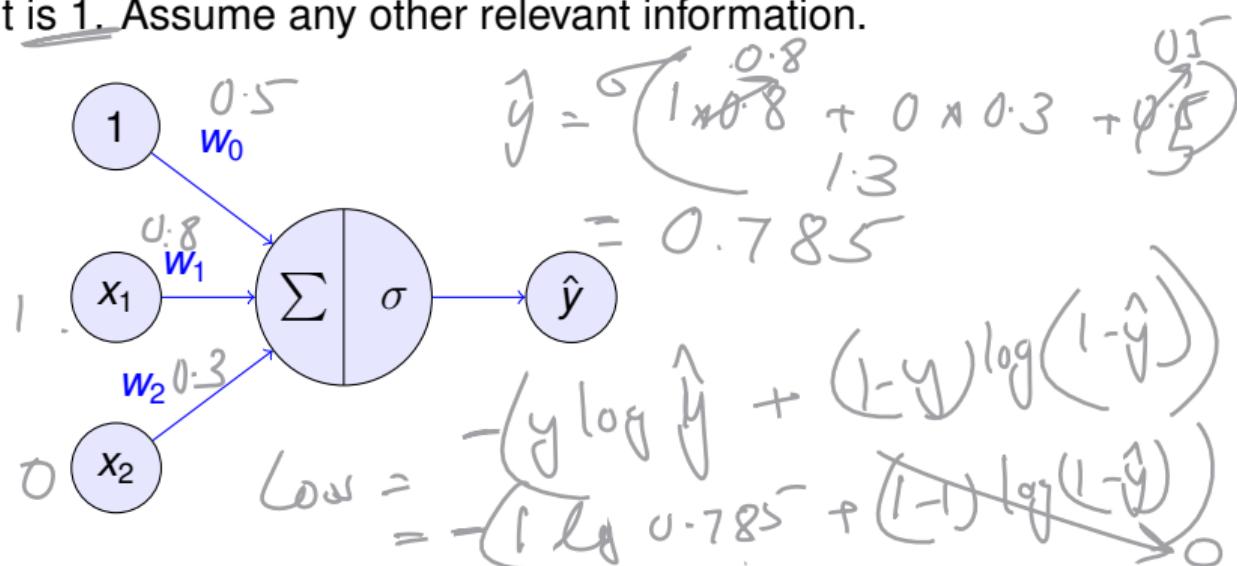
$$w_1 = 0.8 - 0.01 * 0.2142 = 0.7978$$

$$w_2 = 0.3 - 0.01 * 0.2142 = 0.2978$$

EXERCISE - BCE

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, binary cross entropy loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.

$$g = \frac{\partial \text{Loss}}{\partial w} \\ = (y - \hat{y}) \cdot \dots$$



EXERCISE - BCE SOLUTION

$$\hat{y} = \sigma(w_0 + w_1 x_1 + w_2 x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$\begin{aligned} \text{Loss} &= \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \\ &= -[1 \ln 0.7858 + (1 - 1) \ln(1 - 0.7858)] = 0.24 \end{aligned}$$

$$\mathbf{g} = \sum_{i=1}^m (\underline{y} - \hat{\underline{y}}) \underline{z} = (1 - 0.7858) * \underline{1.3} = \underline{0.278}$$

$$w \leftarrow w - \eta * \mathbf{g}$$

$$w_0 = 0.5 - 0.01 * 0.278 = 0.497$$

$$w_1 = 0.8 - 0.01 * 0.278 = 0.797$$

$$w_2 = 0.3 - 0.01 * 0.278 = 0.297$$

SINGLE-LAYER NEURAL NETWORK

- Linear regression is a single-layer neural network.
 - Number of inputs (or feature dimensionality) in the input layer is d . The inputs are x_1, \dots, x_d .
 - Number of outputs in the output layer is 1. The output is \hat{y} .
 - Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
 - Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.

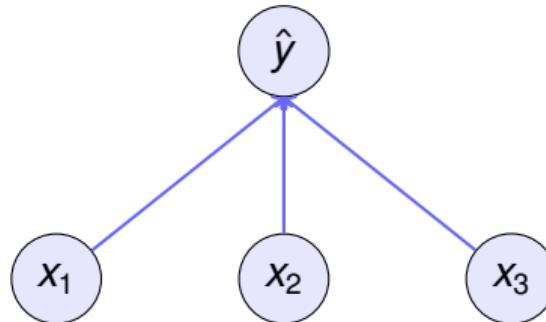


TABLE OF CONTENTS

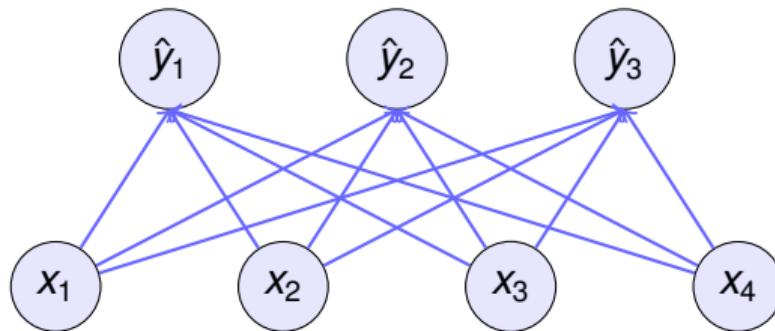
- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

MULTI-CLASS CLASSIFICATION EXAMPLE

- Each input consists of a 2×2 grayscale image.
- Represent each pixel value with a single scalar, giving four features x_1, x_2, x_3, x_4 .
- Assume that each image belongs to one among the categories "square", "triangle", and "circle".
- How to represent the labels?
 - ▶ Use label encoding.
 $y \in \{1, 2, 3\}$, where the integers represent *circle*, *square*, *triangle* respectively.
 - ▶ Use one-hot encoding.
 $y \in \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. y would be a three-dimensional vector, with $(1, 0, 0)$ corresponding to "circle", $(0, 1, 0)$ to "square", and $(0, 0, 1)$ to "triangle".

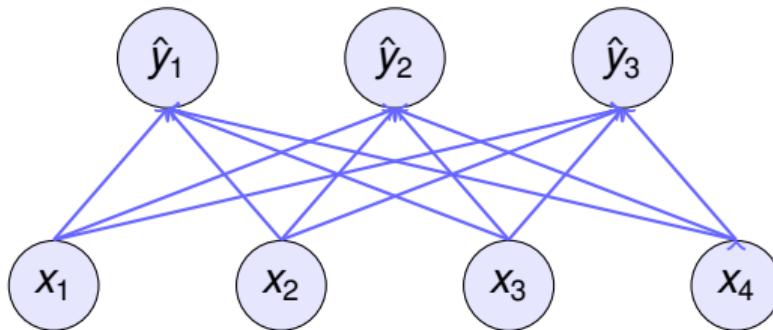
SINGLE-LAYER NEURAL NETWORK

- A model with multiple outputs, one per class. Each output will correspond to its own affine function.
- 4 features and 3 possible output categories
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.



ARCHITECTURE

- 12 scalars to represent the weights and 3 scalars to represent the biases .
- Compute three logits, \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 , for each input.
- Weights is a 3×4 matrix and bias is 1×3 matrix.



$$z_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$

$$\hat{y}_1 = \text{softmax}(z_1)$$

$$z_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$

$$\hat{y}_2 = \text{softmax}(z_2)$$

$$z_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

$$\hat{y}_3 = \text{softmax}(z_3)$$

SOFTMAX OPERATION

- Interpret the outputs of our model as probabilities.
- Any output \hat{y}_j is interpreted as the probability that a given item belongs to class j . Then choose the class with the largest output value as our prediction $\text{argmax}_j \hat{y}_j$.
- If \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 are 0.1, 0.8, and 0.1, respectively, then predict category 2.
- The softmax function transforms the outputs such that they become non-negative and sum to 1, while requiring that the model remains differentiable.

$$\hat{y} = \text{softmax}(Z) \quad \text{where} \quad \hat{y}_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}$$

- First exponentiate each logit (ensuring non-negativity) and then divide by their sum (ensuring that they sum to 1).
- Softmax is a non-linear function.

LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

- The softmax function gives a vector \hat{y} , which can be interpreted as estimated conditional probabilities of each class given any input x ,

$$\hat{y}_j = P(y = \text{class}_j \mid x)$$

- Compare the estimates with reality by checking how probable the actual classes are according to our model, given the features:

$$P(Y \mid X) = \prod_{i=1}^m P(y^{(i)} \mid x^{(i)})$$

LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

- Maximize $P(Y | X)$ = Minimize the negative log-likelihood

$$-\log P(Y | X) = \sum_{i=1}^m -\log P(y^{(i)} | x^{(i)}) = \sum_{i=1}^m loss P(y^{(i)}, \hat{y}^{(i)})$$

$$loss P(y^{(i)}, \hat{y}^{(i)}) = - \sum_{j=1}^m y_j \log \hat{y}_j$$

SOFTMAX EXAMPLE

- z_1 = Un-normalized log probabilities
- e^{z_1} = Un-normalized probabilities
- $\hat{y}_i = \text{softmax}(z_1)$ = normalized probabilities
- L_i = loss = $y_i \log \hat{y}_i$

class	z_1	e^{z_1}	$\text{softmax}(z_1)$	L_1
cat	3.2	24.5	0.13	0.89
car	5.1	164.0	0.87	0.06
dog	-1.7	0.18	0.00	∞

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

DEEP FEEDFORWARD NEURAL NETWORK

- A neural network with more number of hidden layers is considered as a **deep neural network**. There are no feedback connections.
- Convolutional networks are examples of feedforward neural networks.

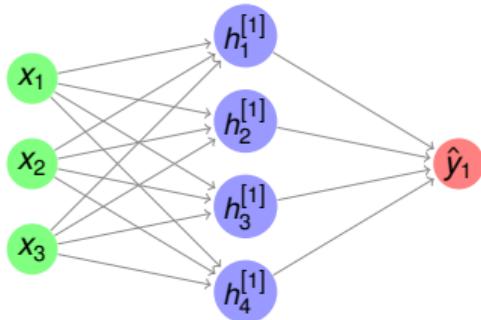


FIGURE: Shallow Neural Network

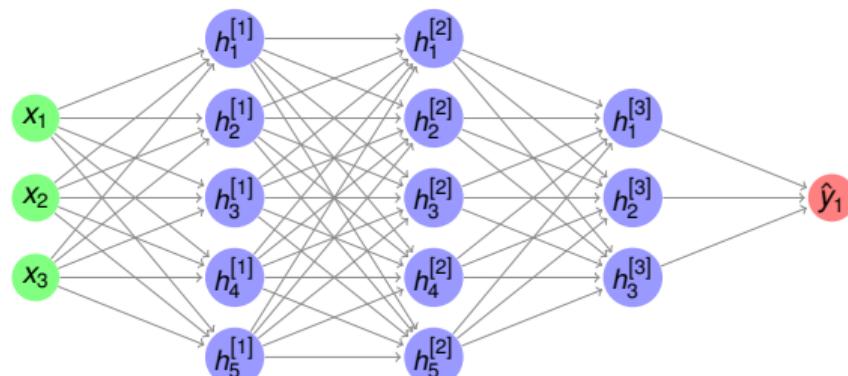
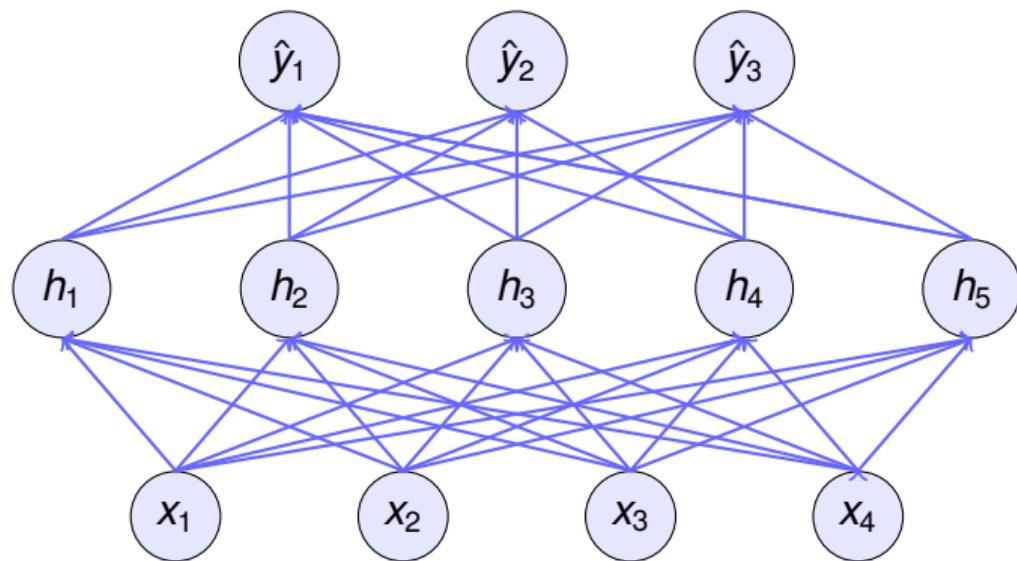


FIGURE: Deep Neural Network

DEEP FEEDFORWARD NEURAL NETWORK (DNN)

- With deep neural networks, use the data to jointly learn both a representation via hidden layers and a linear predictor that acts upon that representation.
- Add many hidden layers by stacking many fully-connected layers on top of each other. Each layer feeds into the layer above it, until we generate outputs.
- The first $(L - 1)$ layers learn the representation and the final layer is the linear predictor.

DNN ARCHITECTURE



- DNN has 4 inputs, 3 outputs, and its hidden layer contains 5 hidden units.
- Number of layers in this DNN is 2.

DNN ARCHITECTURE

- The layers are fully connected.
- Every input influences every neuron in the hidden layer.
- Each of the hidden neurons in turn influences every neuron in the output layer.
- The outputs of the hidden layer are called as **hidden representations** or hidden-layer variable or a hidden variable.

DNN ARCHITECTURE

HIDDEN LAYERS – intermediate layers, desired output for each of these layers are not shown.

DEPTH – number of layers.

WIDTH – dimensionality of the hidden layers

DNN – GENERAL STRATEGY

- Design a DNN architecture of the network. Architecture depends on the problem / application / complexity of the decision boundary.
- Choose the activation functions that will be used to compute the hidden layer activations. The activation function is the same for all neurons in a layer. Activation function can change between layers.
- Choose the cost function. It depends on whether regression, binary classification or multi-class classification.
- Choose the optimizer algorithm, depends on the complexity and variance of the input data.
- Train the feedforward network. Learning in deep neural networks requires computing the gradients using the back-propagation algorithm.
- Evaluate the performance of the network.

NON-LINEARITY IN DNN

- Input is $X \in \mathcal{R}^{m \times d}$ with m examples where each example has d features.
- Hidden layer has h hidden units $H \in \mathcal{R}^{m \times h}$.
- Hidden layer weights $W^{(1)} \in \mathcal{R}^{d \times h}$ and biases $b^{(1)} \in \mathcal{R}^{1 \times h}$
- Output layer weights $W^{(2)} \in \mathcal{R}^{d \times q}$ and biases $b^{(2)} \in \mathcal{R}^{1 \times q}$
- Output is $\hat{Y} \in \mathcal{R}^{m \times q}$
- A non-linear activation function σ has to be applied to each hidden unit following the affine transformation. The outputs of activation functions are called **activations**.

$$H = \sigma(XW^{(1)} + b^{(1)})$$

$$\hat{Y} = \text{softmax}(XW^{(2)} + b^{(2)})$$

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

ACTIVATION FUNCTIONS

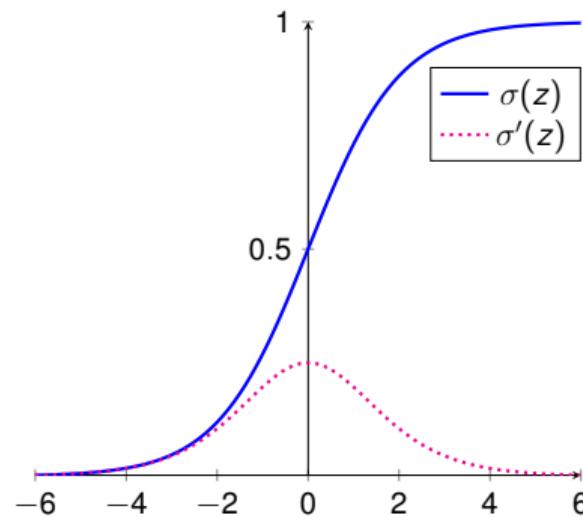
- Activation function of a neuron defines the output of that neuron given an input or set of inputs.
- Introduces non-linearity to a neuron.
- A non-activated neuron will act as a linear regression with limited learning.
- Multilayered deep neural networks learn meaningful features from data.
- Artificial neural networks are designed as universal function approximators, they must have the ability to calculate and learn any non-linear function.

SIGMOID (LOGISTIC) ACTIVATION FUNCTION

$$\text{Function: } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\text{Derivative: } \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Range : $(0, 1)$



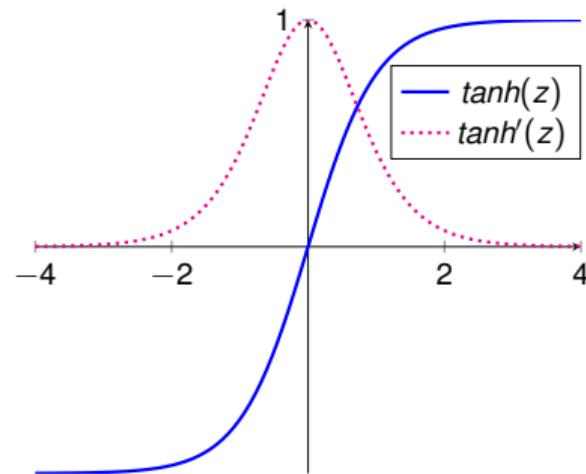
- Non-Linear function
- Small changes in z will be large in $f(z)$. This means it is a good classifier.
- Leads to vanishing gradient. So the learning is minimal.

TANH ACTIVATION FUNCTION

$$\text{Function: } \tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$$

$$\text{Derivative: } \tanh'(z) = (1 - \tanh^2(z))$$

Range : $(-1, 1)$



TANH ACTIVATION FUNCTIONS

- More efficient because it has a wider range for faster learning and grading.
- The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- Issues with tanh
 - ▶ computationally expensive
 - ▶ lead to vanishing gradients

If we initialize the weights to relative large values, this will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

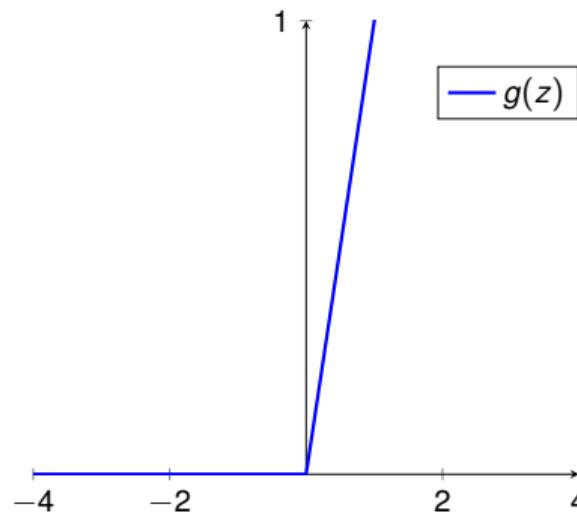
RELU ACTIVATION FUNCTION

Rectified Linear Unit

Function: $g(z) = \max(0, z)$

Derivative: $g'(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0 \end{cases}$

Range : $[0, \infty]$



RELU ACTIVATION FUNCTIONS

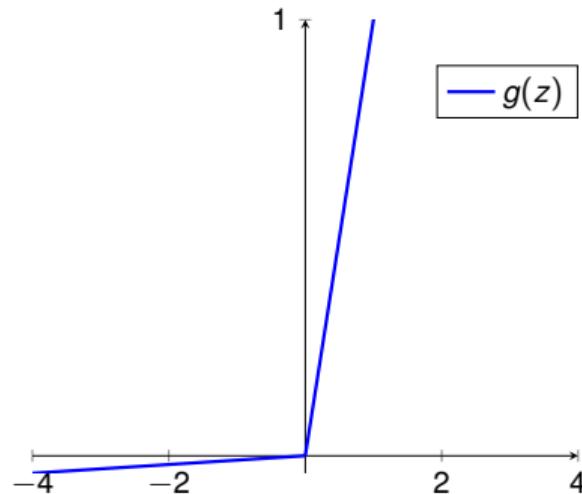
- ReLU activation function is not differentiable at origin.
- If we combine two ReLU units, we can recover a piece-wise linear approximation of the Sigmoid function.
- ReLU activation can lead to exploding gradient.
- In zero value region, learning is not happening.
- Fast Learning
- Fewer vanishing gradient problems
- Sparse activation
- Efficient computation
- Scale invariant (max operation)
- Non Zero centered
- Non differentiable at Zero

LEAKY RELU ACTIVATION FUNCTION

Function: $g(z) = \max(0.01z, z)$

Derivative: $g'(z) = \begin{cases} 0.01 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$



PARAMETRIC RELU (PRELU) ACTIVATION FUNCTION

Function: $g(z) = \max(az, z)$

Derivative:
$$g'(z) = \begin{cases} a & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

Range : $[-\infty, \infty]$

ELU (EXPONENTIAL LINEAR UNITS)

Function:
$$g(z) = \begin{cases} z & \text{for } z < 0 \\ a(e^z - 1) & \text{for } z \geq 0 \end{cases}$$

Range : $[-\infty, \infty]$

COMPARING ACTIVATION FUNCTIONS

Activation Function	Sigmoid	Tanh	ReLU
Linearity	Non Linear	Non Linear	Non Linear
Activation Function	$\sigma(x) = \frac{1}{1+e^{-x}}$	$tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$	$g(x) = \max(0, z)$
Derivative	$\sigma(z)(1 - \sigma(z))$	$(1 - tanh^2(z))$	$1 \text{ if } z > 0 \text{ else } 0$
Symmetric Function	No	Yes	No
Range	$[0, 1]$	$[-1, 1]$	$[0, \infty]$
Vanishing Gradient	Yes	Yes	No
Exploding Gradient	No	No	Yes
Zero Centered	No	Yes	No

COMPARING ACTIVATION FUNCTIONS

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

ACTIVATION FUNCTIONS PLOTS

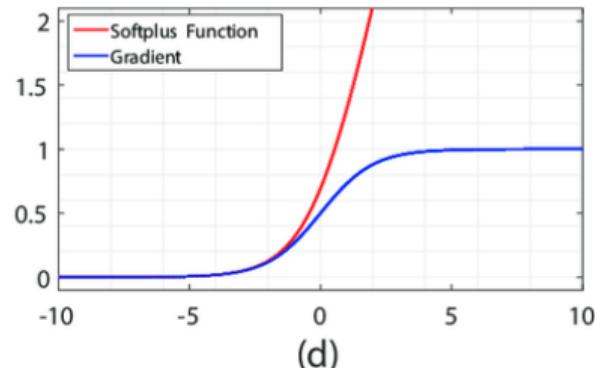
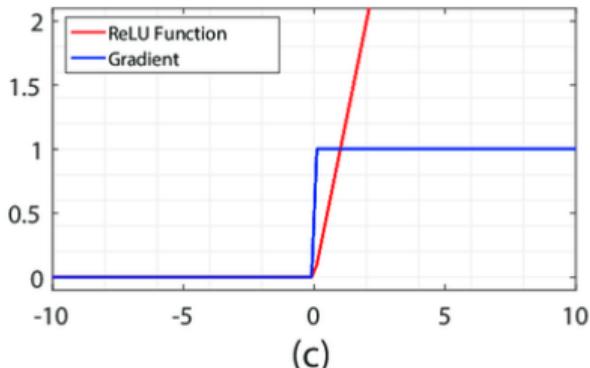
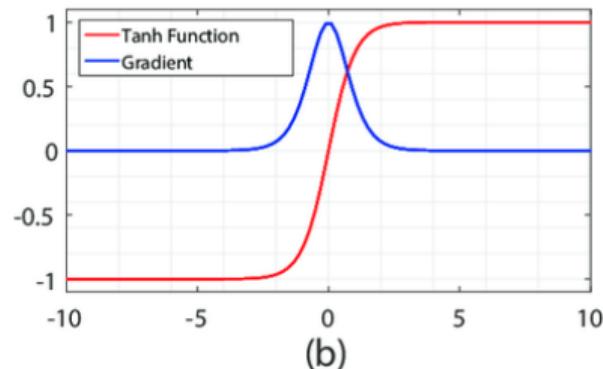
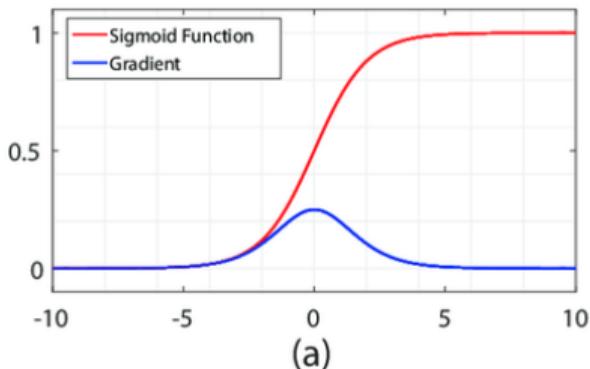


TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

COMPUTATION GRAPHS

- Computations of a neural network are organized in terms of
 - ▶ a forward pass or a forward propagation step, in which we compute the output of the neural network,
 - ▶ followed by a backward pass or back propagation step, which we use to compute gradients or derivatives.
- The computation graph organizes a computation with this blue arrow, left-to-right computation.
- The backward red arrow, right-to-left shows computation of the derivatives.

COMPUTATION GRAPHS

- Each node in the graph indicate a variable.
- An operation is a simple function of one or more variable.
- An operation is defined such that it returns only a single output variable.
- If a variable y is computed by applying an operation to a variable x , then we draw a directed edge from x to y .

COMPUTATION GRAPHS EXAMPLE

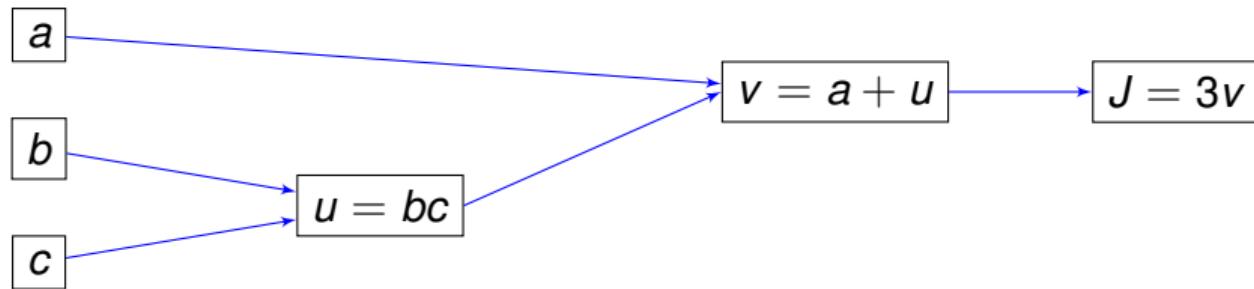
Forward Pass:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$



COMPUTATION GRAPHS EXAMPLE

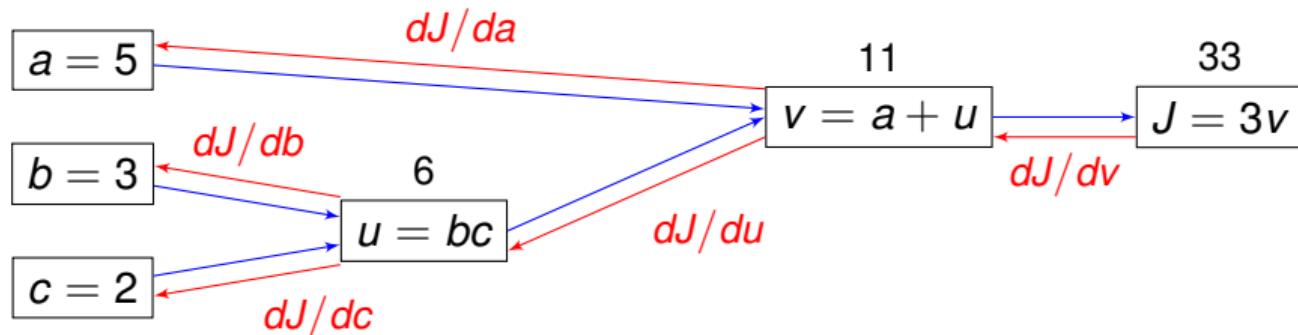
Backward Pass / Back propagation of gradients:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$

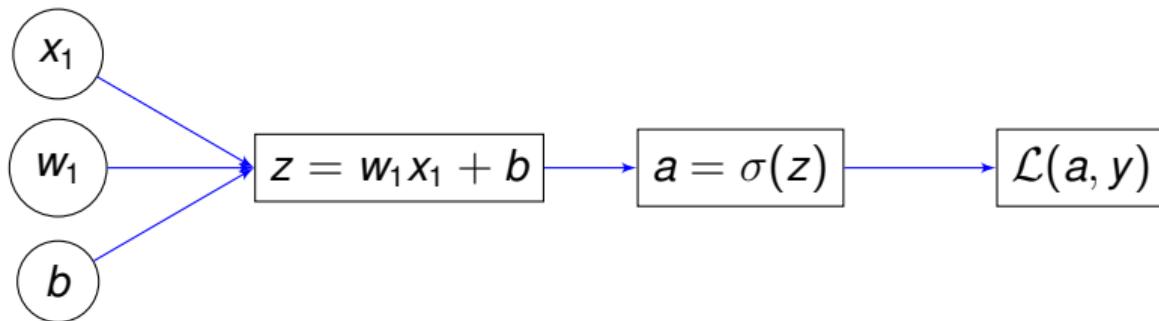


COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^\top X + b$$

$$a = \hat{y} = \sigma(z)$$

$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$



COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^\top X + b$$

$$a = \hat{y} = \sigma(z)$$

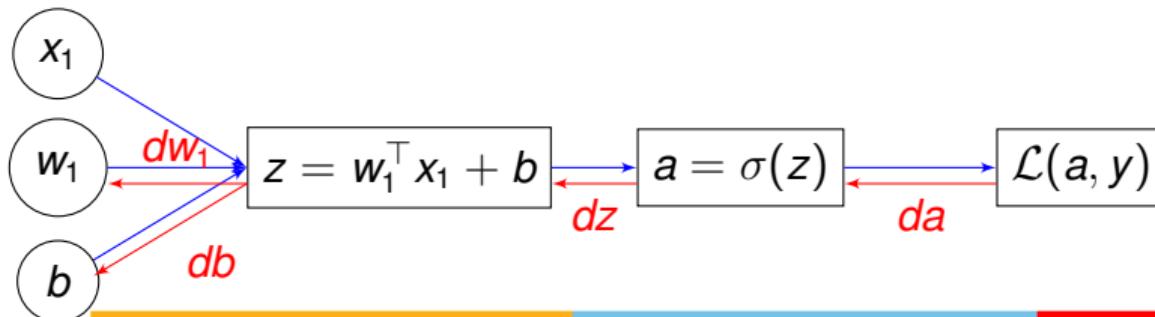
$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$

$$da = \frac{d\mathcal{L}(a, y)}{da} = \frac{-y}{a} + \frac{1+y}{1+a}$$

$$dz = \frac{d\mathcal{L}(a, y)}{dz} = a - y$$

$$dw_1 = \frac{d\mathcal{L}(a, y)}{dw_1} = x_1 dz$$

$$db = dz$$



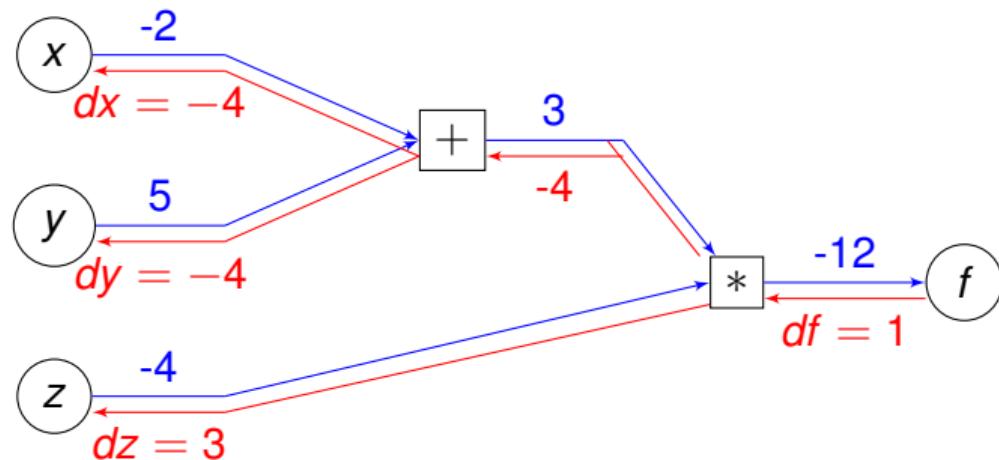
EXERCISE

Draw the computational graph for the equation

$$f(x, y, z) = (x + y)z$$

Assume that $x = -2$, $y = 5$ and $z = -4$. Using the computation graph show the computation of the gradients also.

EXERCISE - SOLUTION



EXERCISE

Draw the computational graph for the equation

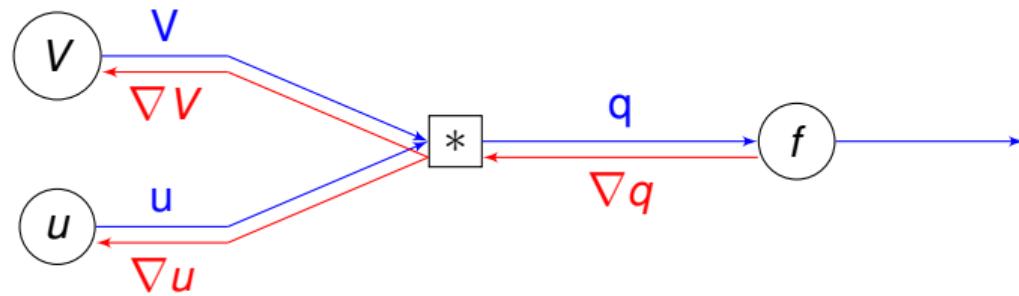
$$f(u, V) = \| V \cdot u \|^2 = \sum_{i=1}^n (V \cdot u)_i^2$$

Using the computation graph show the computation of the gradients also. Assume that

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

EXERCISE - SOLUTION



EXERCISE - SOLUTION

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$q = V \cdot u = \begin{bmatrix} 0.1 * 0.2 + 0.5 * 0.4 \\ -0.3 * 0.2 + 0.8 * 0.4 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$f = \sum_{i=1} q_i^2 = 0.22^2 + 0.26^2 = 0.116$$

EXERCISE - SOLUTION

$$\nabla q = \nabla_q f = 2q = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

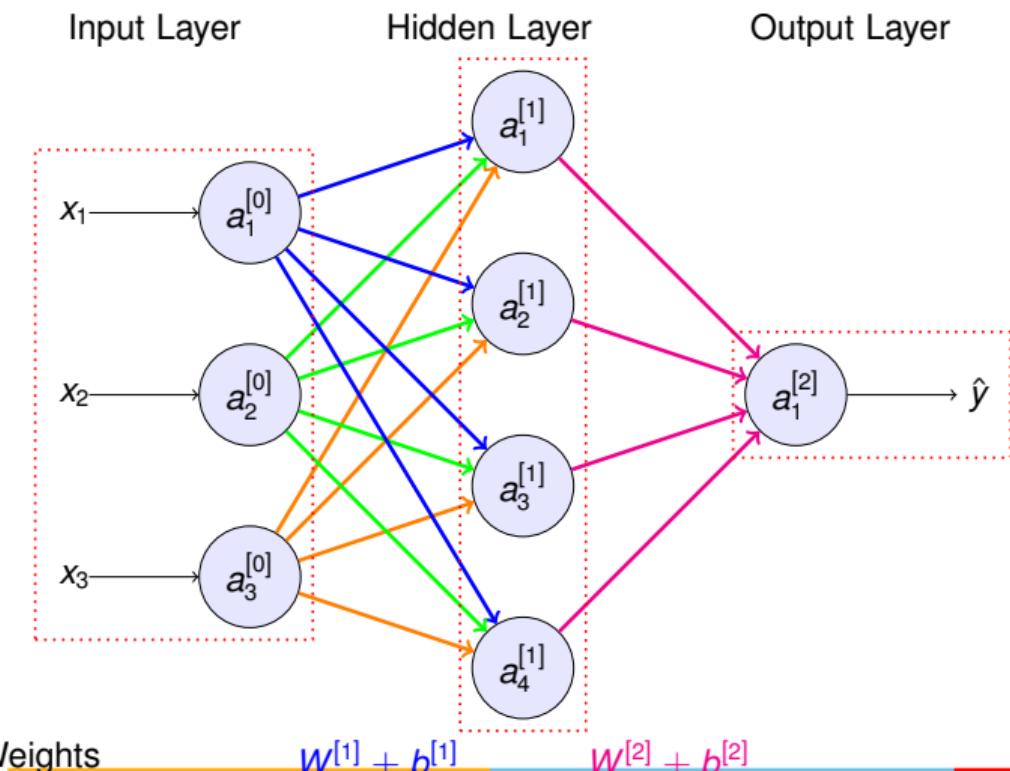
$$\nabla V = \nabla_V f = 2q \cdot u^\top = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.088 & 0.176 \\ 0.105 & 0.208 \end{bmatrix}$$

$$\nabla u = \nabla_u f = 2V^\top \cdot q = 2 \begin{bmatrix} 0.1 & -0.3 \\ 0.5 & 0.8 \end{bmatrix} \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

TWO LAYER NEURAL NETWORK ARCHITECTURE



COMPUTING THE ACTIVATIONS

For Hidden layer

$$z_1^{[1]} = w_1^{[1]} a^{[0]} + b^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]} a^{[0]} + b^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]} a^{[0]} + b^{[1]}$$

$$a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]} a^{[0]} + b^{[1]}$$

$$a_4^{[1]} = \sigma(z_4^{[1]})$$

In the matrix form

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} \dots w_1^{[1]\top} \dots \\ \dots w_2^{[1]\top} \dots \\ \dots w_3^{[1]\top} \dots \\ \dots w_4^{[1]\top} \dots \end{bmatrix}}_{4 \times 3} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{4 \times 1}$$

$$z^{[1]} = \begin{bmatrix} w_1^{[1]\top} a^{[0]} + b_1^{[1]} \\ w_2^{[1]\top} a^{[0]} + b_2^{[1]} \\ w_3^{[1]\top} a^{[0]} + b_3^{[1]} \\ w_4^{[1]\top} a^{[0]} + b_4^{[1]} \end{bmatrix} \leftarrow 4 \times 1 \text{ matrix}$$

$$a^{[1]} = \sigma(z^{[1]}) \leftarrow 4 \times 1 \text{ matrix}$$

ACTIVATIONS FOR TRAINING EXAMPLES

Vectorizing for one training example

$$a^{[0]} = X$$

$$Z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(Z^{[2]})$$

$$\hat{y} = a^{[2]}$$

Vectorizing for all training examples

$$\text{Example 1 : } X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$\text{Example 2 : } X^{(2)} \longrightarrow a^{2} = \hat{y}^{(2)}$$

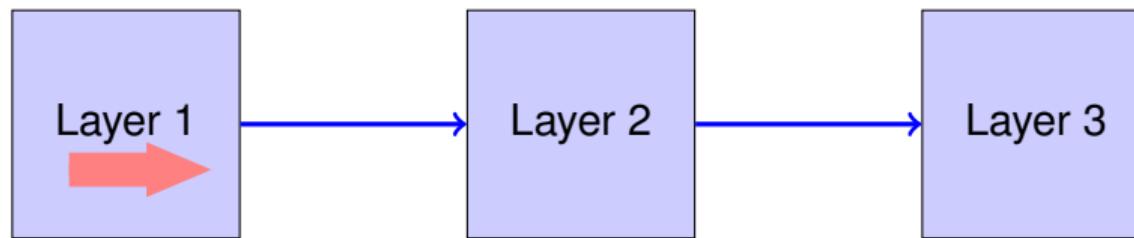
$$\text{Example 3 : } X^{(2)} \longrightarrow a^{[2](3)} = \hat{y}^{(3)}$$

 \vdots
 \vdots

$$\text{Example m : } X^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

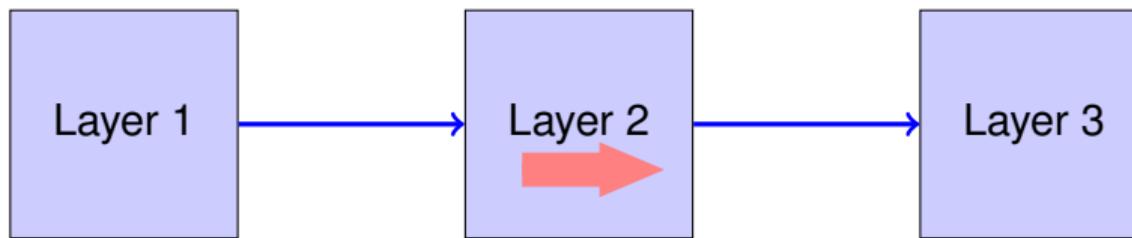
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



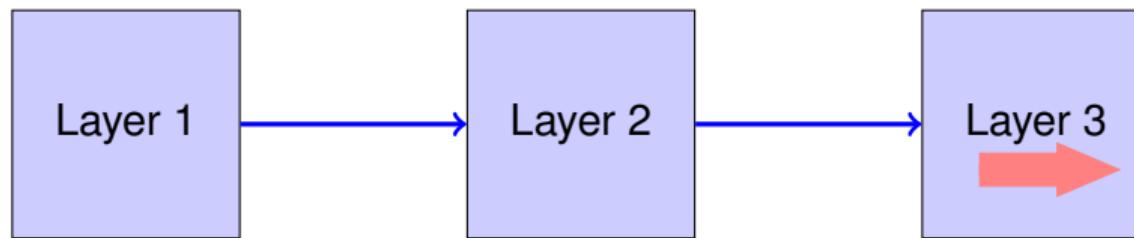
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



FORWARD PROPAGATION ALGORITHM

Algorithm 1: FORWARD PROPAGATION

1 Initialize the weights and bias randomly.

2 $a^{[0]} = X^{(i)}$

3 $Z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$

4 $a^{[1]} = \sigma(Z^{[1]})$

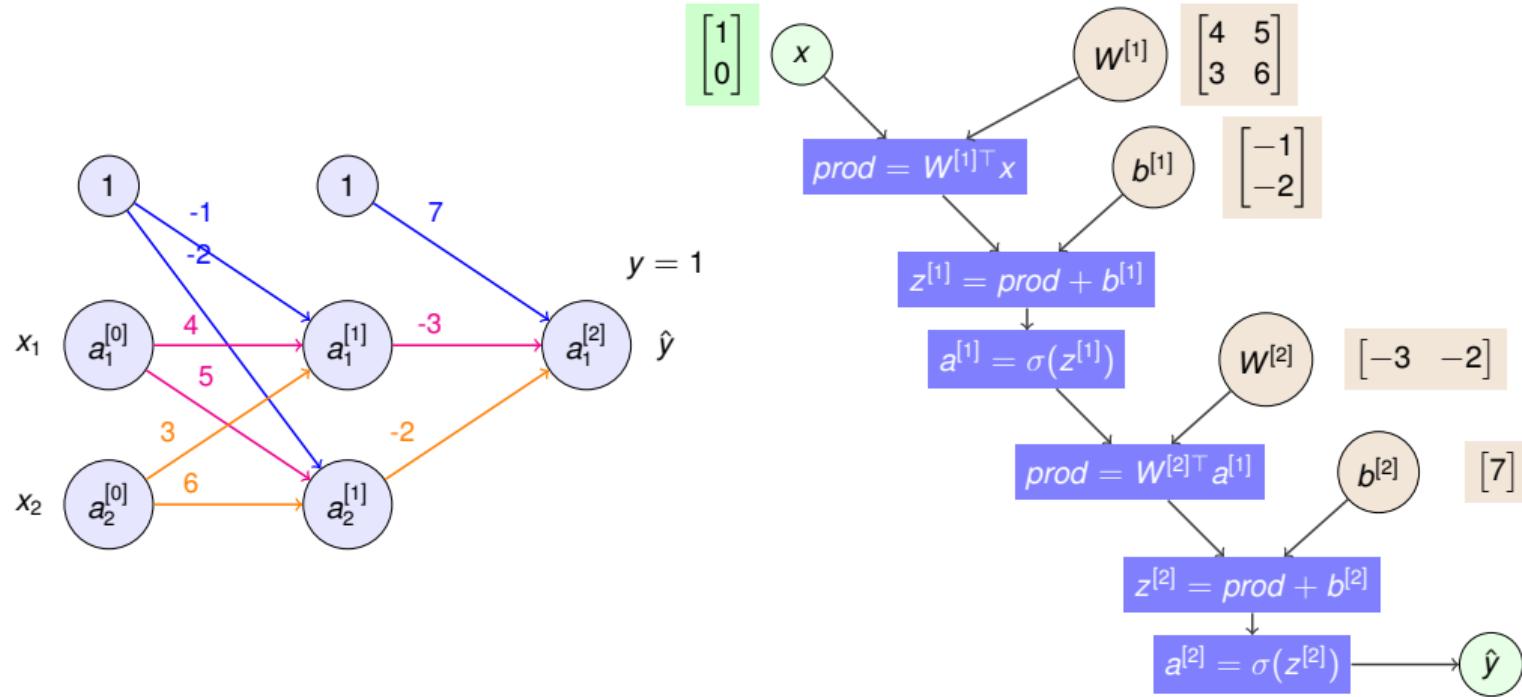
5 $Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

6 $a^{[2]} = \sigma(Z^{[2]})$

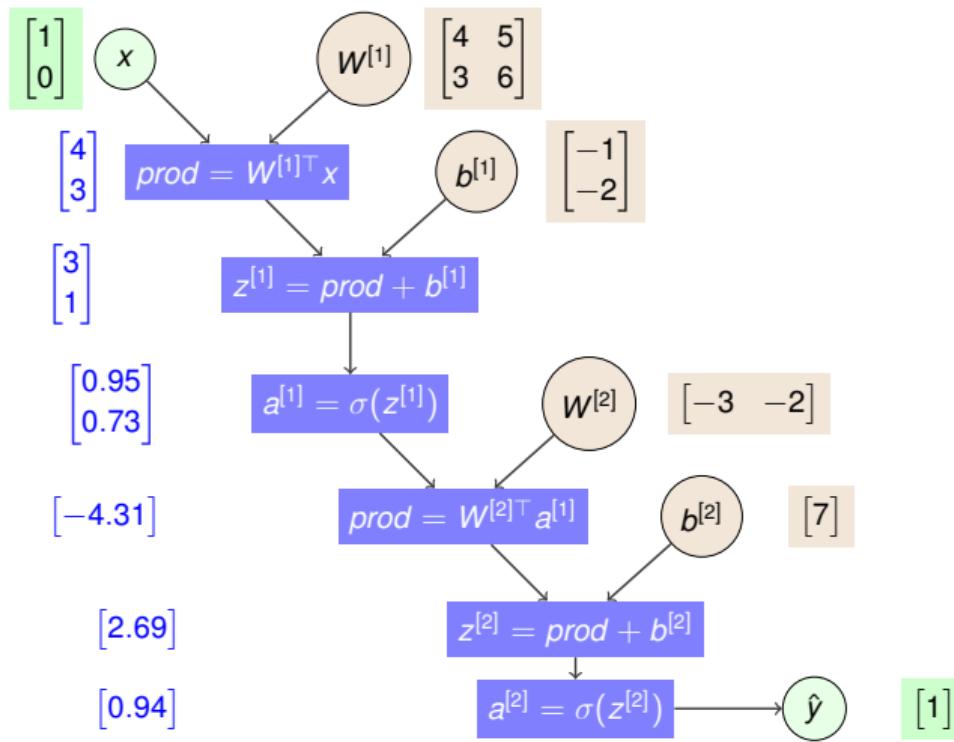
7 $\hat{y} = \begin{cases} 1 & \text{if } a^{[2]} > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Note: All training examples are considered in the vectorized form.

EXAMPLE NEURAL NETWORK



COMPUTATION GRAPH FOR FORWARD PASS



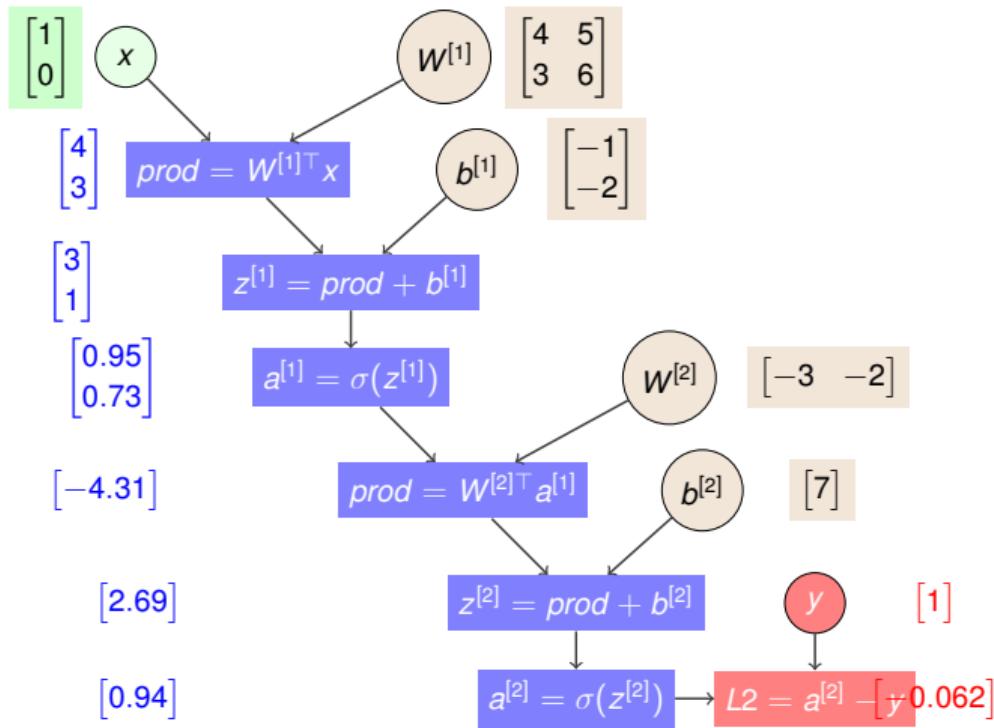
COST FUNCTION

- The difference between the actual observation y and the computed activation \hat{y} gives the error or the cost function.
- For binary classification,

$$\mathcal{L} = \frac{1}{m} \sum_{i=0}^m loss(\hat{y}^{(i)}, y^{(i)})$$

$$\mathcal{L} = -\frac{1}{m} \sum_{i=0}^m (y^{(i)} \log(a^{[2](i)}) + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$$

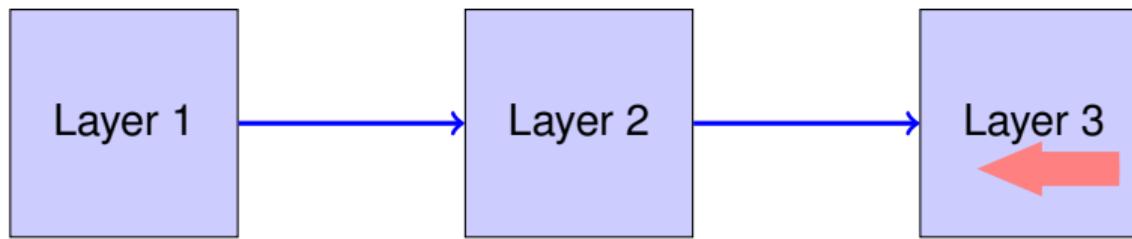
COMPUTATION GRAPH FOR COST FUNCTION



NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

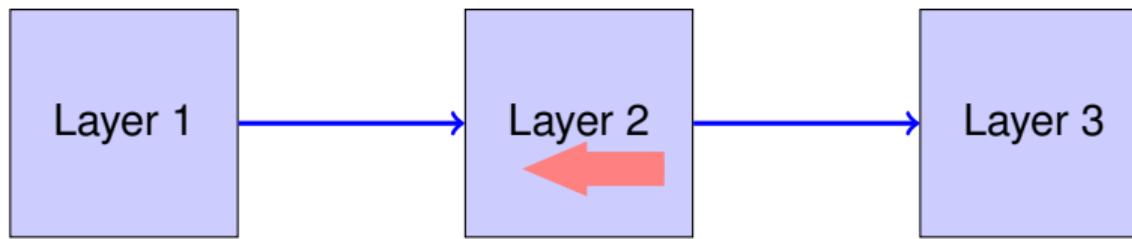
Step 2: Compute gradients wrt parameters [Backward pass]



NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

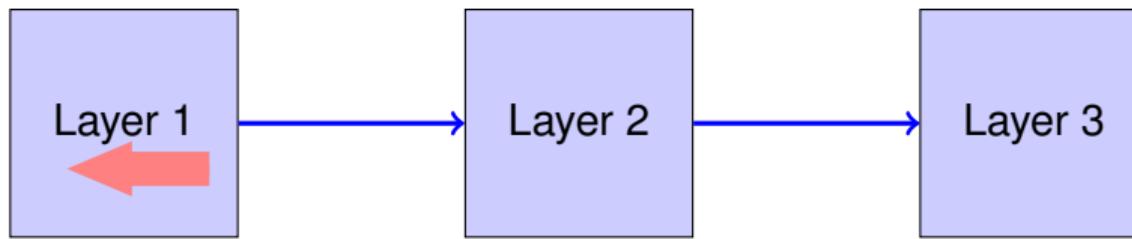
Step 2: Compute gradients wrt parameters [Backward pass]



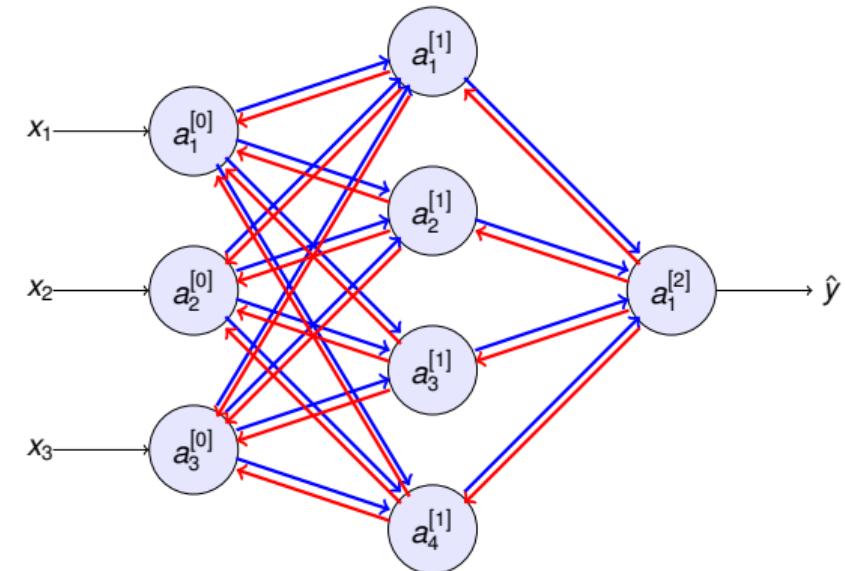
NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]



BACK PROPAGATION OF COST FUNCTION



Weights

$W^{[1]} + b^{[1]}$

$W^{[2]} + b^{[2]}$

Gradients

$da^{[1]} + dz^{[1]}$

$da^{[2]} + dz^{[2]}$

COMPUTING THE GRADIENTS

For all training examples

For one training example

$$dz^{[2]} = a^{[2]} - y$$

$$dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = (w^{[2]T} dz^{[2]}) * \sigma'(z^{[1]})$$

$$dw^{[1]} = dz^{[1]} \cdot x^T$$

$$db^{[1]} = dz_1$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T})$$

$$db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$dZ^{[1]} = (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} (dZ^{[1]} \cdot X^T)$$

$$db^{[1]} = \frac{1}{m} \sum dZ_1$$

BACKWARD PROPAGATION ALGORITHM

Algorithm 2: BACKWARD PROPAGATION

$$1 \quad dZ^{[2]} = A^{[2]} - Y$$

$$2 \quad dW^{[2]} = \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T})$$

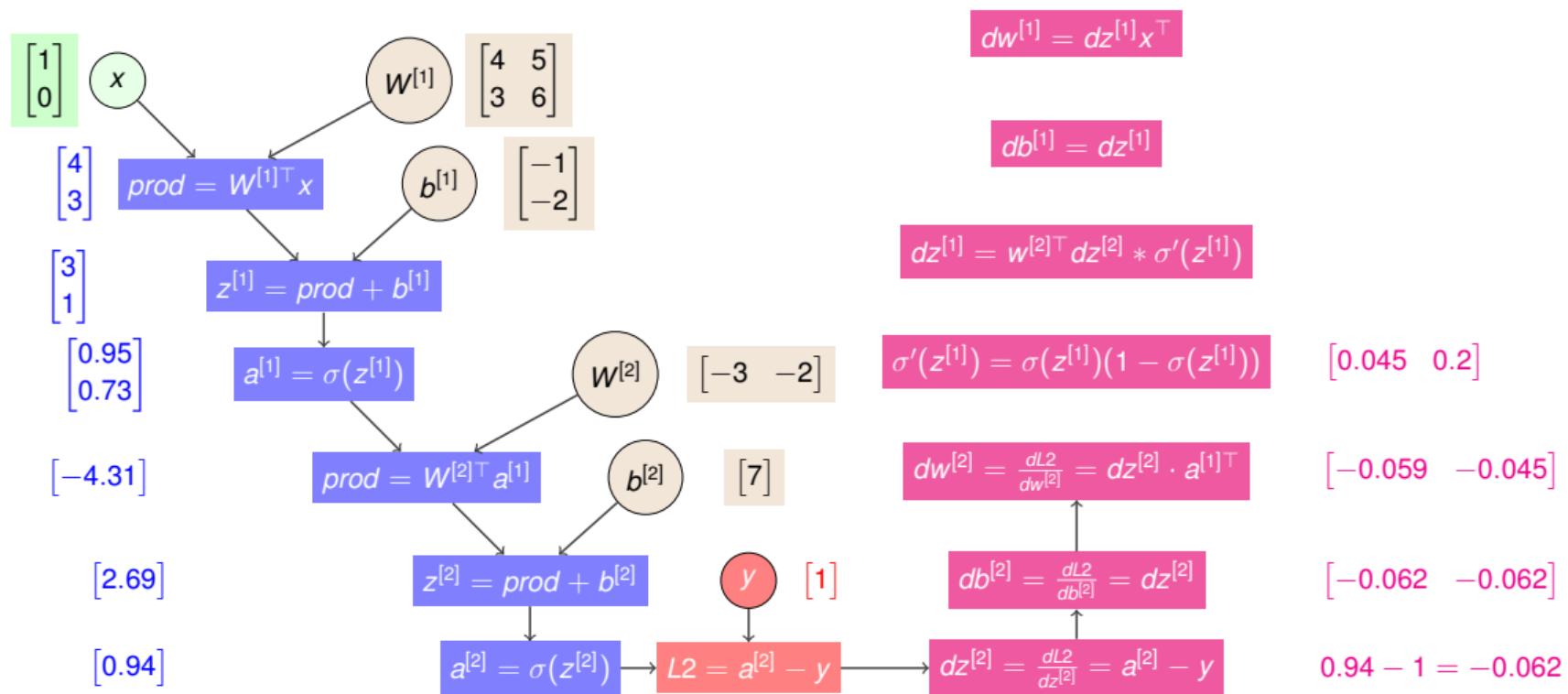
$$3 \quad db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$4 \quad dZ^{[1]} = (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]})$$

$$5 \quad dW^{[1]} = \frac{1}{m} (dZ^{[1]} \cdot X^T)$$

$$6 \quad db^{[1]} = \frac{1}{m} \sum dZ^{[1]}$$

COMPUTATION GRAPH FOR BACKWARD PASS

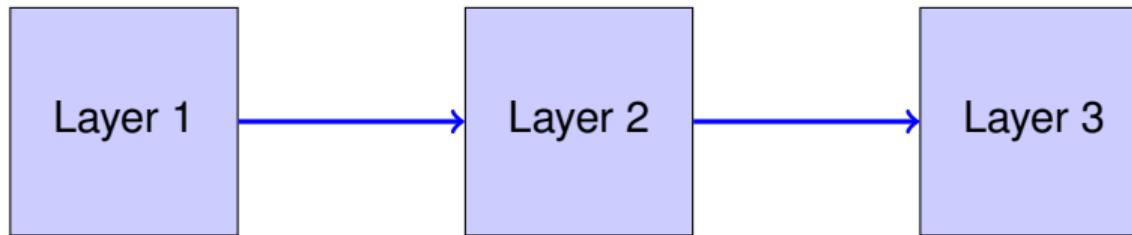


NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]

Step 3: Use gradients to update parameters



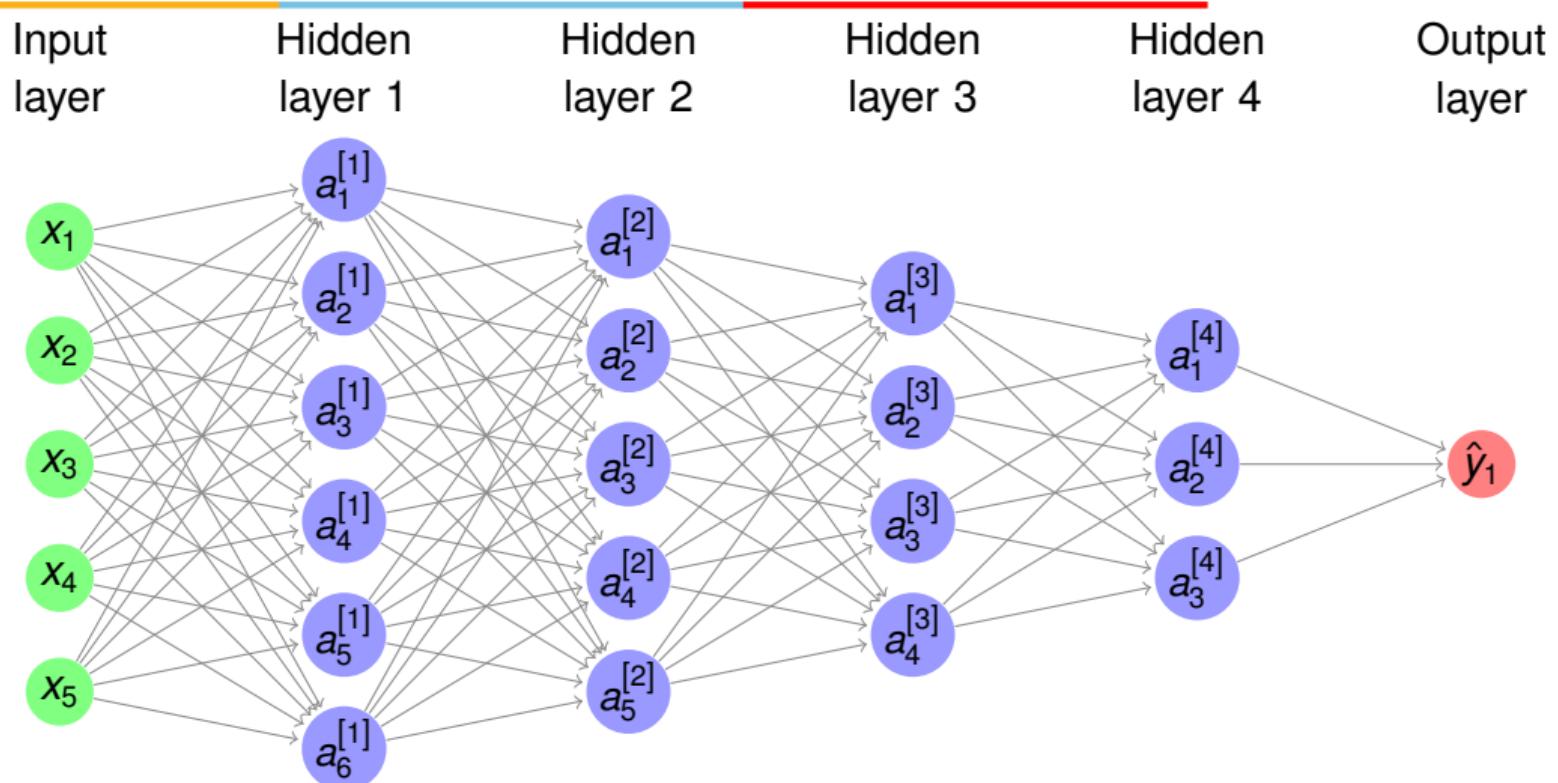
$$\theta \leftarrow \theta - \eta \frac{d\mathcal{L}}{d\theta}$$

UPDATE THE WEIGHTS AND BIAS

Algorithm 3: WEIGHT UPDATION

- 1 $W^{[1]} = W^{[1]} - \text{learning rate} * dW^{[1]}$
 - 2 $b^{[1]} = b^{[1]} - \text{learning rate} * db^{[1]}$
 - 3 $W^{[2]} = W^{[2]} - \text{learning rate} * dW^{[2]}$
 - 4 $b^{[2]} = b^{[2]} - \text{learning rate} * db^{[2]}$
-

DNN ARCHITECTURE



TRAINING DNN

Requires

- ① **Forward Pass** through each layer to compute the output.
- ② Compute the deviation or error between the desired output and computed output in the forward pass (first step). This morphs into **objective function**, as we want to minimize this deviation or error.
- ③ The deviation has to be send back through each layer to compute the delta or change in the parameter values. This is achieved using **back propagation algorithm**.
- ④ **Update** the parameters.

DNN ARCHITECTURE

Notations

- L : Number of layers
- $n^{[l]}$: Number of units in layer l
- $w^{[l]}$: Weights for layer l
- $b^{[l]}$: Bias for layer l
- $z^{[l]}$: Hypothesis for layer l
- $g^{[l]}$: Activation Function used for layer l
- $a^{[l]}$: Activation for layer l

Matrix Dimensions

- $W^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $b^{[l]}$: $n^{[l]} \times 1$
- $z^{[l]}$: $n^{[l]} \times 1$
- $a^{[l]}$: $n^{[l]} \times 1$
- $dW^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $db^{[l]}$: $n^{[l]} \times 1$
- $dz^{[l]}$: $n^{[l]} \times 1$
- $da^{[l]}$: $n^{[l]} \times 1$

FORWARD PROPAGATION

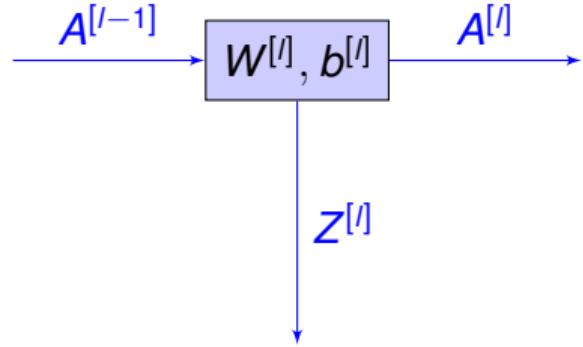
- The inputs \mathbf{X} provide the initial information that then propagates up to the hidden units at each layer and finally produces \hat{y} . This is called forward propagation.
- Information flows forward through the network.
- During training, it produces a scalar cost $\mathcal{L}(\theta)$.

FORWARD PROPAGATION ALGORITHM

Algorithm 4: Forward Propagation

```

1 for  $l$  in range ( $1, L$ ) do
2    $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$ 
3    $A^{[l]} = g^{[l]}(Z^{[l]})$ 
4    $\hat{y} = A^{[L]}$ 
5    $J = L(\hat{y}, y) + \lambda\Omega(\theta)$ 
  
```



BACKWARD PROPAGATION

- Back-propagation algorithm or backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient $\nabla_{\theta} J(\theta)$.
- Back-propagation refers only to the method for computing the gradient.

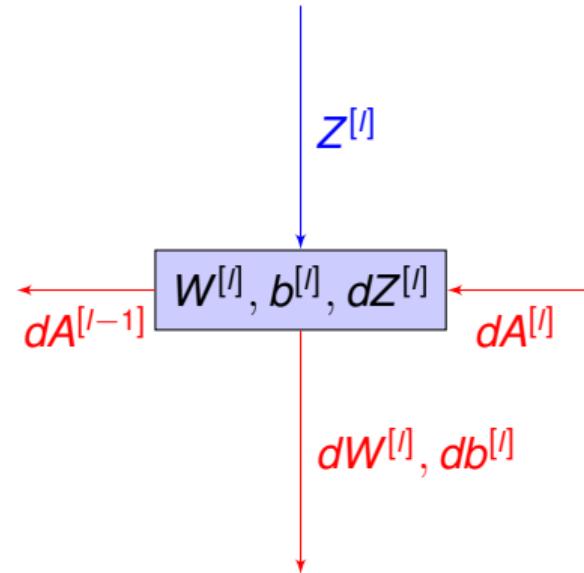
BACKWARD PROPAGATION ALGORITHM

Algorithm 5: Backward Propagation

```

1  $da^{[L]} = \frac{-y}{a} + \frac{1-y}{1-a}$ 
2 for  $l$  in range ( $1, L$ ) do
3    $dZ^{[l]} = dA^{[l]} * g'^{[l]}(Z^{[l]})$ 
4    $dW^{[l]} = dZ^{[l]} \cdot A^{[l-1]T}$ 
5    $db^{[l]} = dZ^{[l]}$ 
6    $dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$ 

```



UPDATE THE WEIGHTS AND BIAS

Algorithm 6: Weight Updation

```
1 for  $l$  in range ( $1, L$ ) do
2      $W^{[l]} = W^{[l]} - \eta * dW^{[l]}$ 
3      $b^{[l]} = b^{[l]} - \eta * db^{[l]}$ 
```

OUTPUT NEURONS

- The choice of how to represent the output determines the form of the cross-entropy function.
- The feedforward network provides a set of hidden features $h = f(x; \theta)$.
- The role of the output layer is to provide some additional transformation.
- Types of output transformation
 - ① Linear
 - ② Sigmoid
 - ③ Softmax

LOSS FUNCTIONS

- Loss function compares the actual output from data set, in case of supervised learning, to the predicted output of the output layer.
- Based on the task, the loss functions differ.

For Regression	For Classification
Mean Squared Error (MSE)	Binary Cross Entropy (Sigmoid)
Mean Absolute Error (MAE)	Categorical Cross Entropy (Softmax)
Huber loss (Smooth MAE)	KL Divergence (Relative Entropy)
Log Hyperbolic Cosine (log cosh)	Exponential Loss
Quantile Loss	Hinge Loss
Cosine Similarity	

Further Reading

- ➊ Dive into Deep Learning (T1)



Thank You!



DEEP LEARNING MODULE # 2 : DEEP FEEDFORWARD NEURAL NETWORK



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

LINEAR REGRESSION EXAMPLE

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):

$$price = w_{area} * area + w_{age} * age + b$$

w_{area} and w_{age} are called weights, and b is called a bias.

- The weights determine the influence of each feature on our prediction and the bias just says what value the predicted price should take when all of the features take value 0.

DATA

- The dataset comprises of training dataset and testing dataset. The split of DL is generally 99:1, as we are dealing with millions of examples.
- Each row is called an **example (or data point, data instance, sample)**.
- The thing we are trying to predict is called a **label (or target)**.
- The independent variables upon which the predictions are based are called **features (or covariates)**.

m – number of training examples

i – i^{th} example

$x^{(i)} = [x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(m)}]$ – features of i^{th} example

$y^{(i)} = [y_1^{(1)}, y_1^{(2)}, \dots, y_1^{(m)}]$ – label of i^{th} example

AFFINE TRANSFORMATIONS AND LINEAR MODELS

- The equation of the form

$$\hat{y} = w_1x_1 + \dots w_dx_d + b$$

$$\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{x} + b \quad \mathbf{x} \in \mathcal{R}^d \quad \mathbf{w} \in \mathcal{R}^d$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b \quad \mathbf{X} \in \mathcal{R}^{m \times d} \quad \hat{\mathbf{y}} \in \mathcal{R}^m$$

is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.

- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen weights (w) and bias (b).

LOSS FUNCTION

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the **squared error**.

SQUARED ERROR LOSS FUNCTION

- The most popular loss function in regression problems is the squared error.
- For each example,

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

- For the entire dataset of m examples, average (or equivalently, sum) the losses

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m l^{(i)}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

- When training the model, find parameters $(\mathbf{w}_{opt}, b_{opt})$ that minimize the total loss across all training examples.

$$(\mathbf{w}_{opt}, b_{opt}) = \arg \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b)$$

MINIBATCH STOCHASTIC GRADIENT DESCENT (SGD)

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.
- In each iteration,
 - ▶ Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.
 - ▶ Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
 - ▶ Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|B|} \sum_{i \in B} \partial_w \text{loss}^{(i)}(\mathbf{w}, b)$$

$$b \leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b \text{loss}^{(i)}(\mathbf{w}, b)$$

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

PREDICTION

- Estimating targets given features is commonly called **prediction or inference**.
- Given the learned model, values of target can be predicted, for any set of features.

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

BINARY CLASSIFICATION – TRAINING EXAMPLES

Single training example = $\{(x, y)\}$ where $x \in \mathcal{R}^d$ and $y \in \{0, 1\}$

training examples = m

m training examples = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \text{where } \mathbf{X} \in \mathcal{R}^{d \times m}$$

$$\mathbf{Y} = [y^{(1)} \quad y^{(2)} \quad \dots \quad y^{(m)}] \quad \text{where } \mathbf{Y} \in \mathcal{R}^{1 \times m}$$

FORWARD PROPAGATION

Given X find $\hat{y} = P(y = 1 | X)$

Input $X \in \mathcal{R}^{d \times m}$

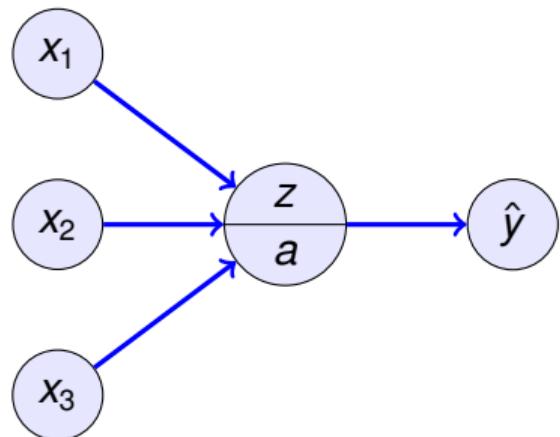
Parameters $w \in \mathcal{R}^d$

$b \in \mathcal{R}$

Activation $z = w^\top x + b$

Activation function $\sigma(z) = \frac{1}{1 + e^{-z}}$
 $= \begin{cases} 1 & \text{if } z \text{ is large positive} \\ 0 & \text{if } z \text{ is large negative} \end{cases}$

Output $\hat{y} = \sigma(w^\top x + b) \quad 0 \leq \hat{y} \leq 1$



COST FUNCTION FOR BINARY CLASSIFICATION

Loss function $\text{loss}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

If $y = 1$ $\text{loss}(\hat{y}, y) = -\log \hat{y}$

$\text{loss} \approx 0 \implies \log \hat{y} \approx \text{large} \implies \hat{y} \approx \text{large}$

If $y = 0$ $\text{loss}(\hat{y}, y) = -\log(1 - \hat{y})$

$\text{loss} \approx 0 \implies \log(1 - \hat{y}) \approx \text{small} \implies \hat{y} \approx \text{small}$

$$\begin{aligned}\text{Cost function } \mathcal{L}(w, b) &= \frac{1}{m} \sum_{i=1}^m \text{loss}(\hat{y}^{(i)}, y^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]\end{aligned}$$

HOW TO LEARN PARAMETERS?

To Learn parameters use Gradient Descent Algorithm

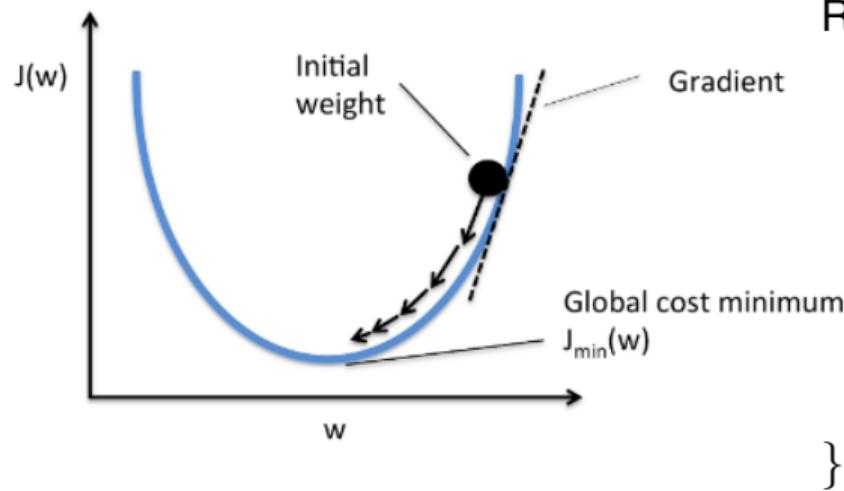
$$\hat{y} = \sigma(w^\top x + b) \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(w, b) = \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Find w_{opt}, b_{opt} that minimize $\mathcal{L}(w, b)$.

GRADIENT DESCENT

Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$



Repeat {

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}}$$

$$b \leftarrow b - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b}$$

}

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

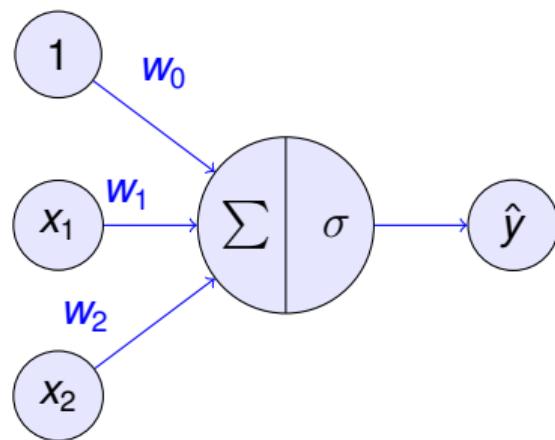
- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

EXERCISE - MSE LOSS

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, mean squared loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.



EXERCISE - MSE LOSS SOLUTION

$$\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$MSE = \frac{1}{2m} \sum_{i=1}^m (y - \hat{y})^2 = \frac{1}{2}(1 - 0.7858)^2 = \frac{0.04588}{2} = 0.02294$$

$$\mathbf{g} = \frac{2}{2m} \sum_{i=1}^m (y - \hat{y}) = (1 - 0.7858) = 0.2142$$

new w \leftarrow *old w* $- \eta * \mathbf{g}$

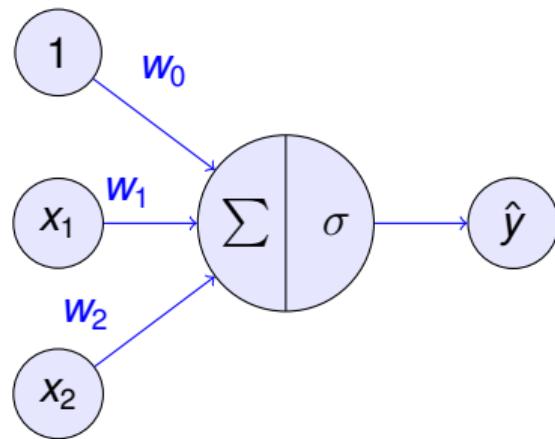
$$w_0 = 0.5 - 0.01 * 0.2142 = 0.4978$$

$$w_1 = 0.8 - 0.01 * 0.2142 = 0.7978$$

$$w_2 = 0.3 - 0.01 * 0.2142 = 0.2978$$

EXERCISE - BCE

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, binary cross entropy loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.



EXERCISE - BCE SOLUTION

$$\hat{y} = \sigma(w_0 + w_1 x_1 + w_2 x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$\begin{aligned} Loss &= \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \\ &= -[1 \ln 0.7858 + (1 - 1) \ln(1 - 0.7858)] = 0.24 \end{aligned}$$

$$\mathbf{g} = \sum_{i=1}^m (y - \hat{y}) z = (1 - 0.7858) * 1.3 = 0.278$$

$$w \leftarrow w - \eta * \mathbf{g}$$

$$w_0 = 0.5 - 0.01 * 0.278 = 0.497$$

$$w_1 = 0.8 - 0.01 * 0.278 = 0.797$$

$$w_2 = 0.3 - 0.01 * 0.278 = 0.297$$

SINGLE-LAYER NEURAL NETWORK

- Linear regression is a single-layer neural network.
 - Number of inputs (or feature dimensionality) in the input layer is d . The inputs are x_1, \dots, x_d .
 - Number of outputs in the output layer is 1. The output is \hat{y} .
 - Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
 - Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.

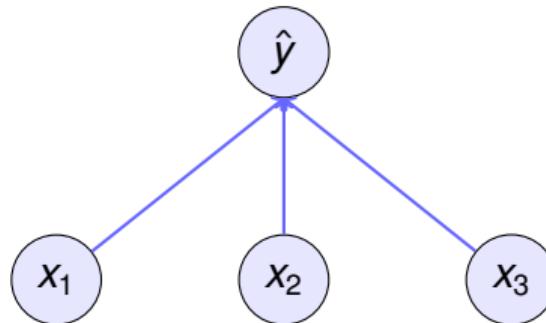


TABLE OF CONTENTS

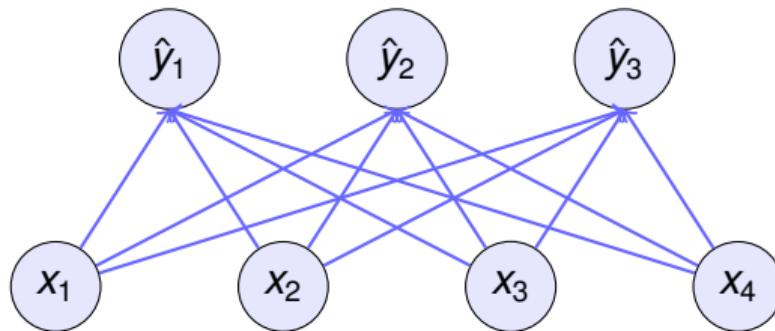
- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

MULTI-CLASS CLASSIFICATION EXAMPLE

- Each input consists of a 2×2 grayscale image.
- Represent each pixel value with a single scalar, giving four features x_1, x_2, x_3, x_4 .
- Assume that each image belongs to one among the categories "square", "triangle", and "circle".
- How to represent the labels?
 - ▶ Use label encoding.
 $y \in \{1, 2, 3\}$, where the integers represent *circle*, *square*, *triangle* respectively.
 - ▶ Use one-hot encoding.
 $y \in \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. y would be a three-dimensional vector, with $(1, 0, 0)$ corresponding to "circle", $(0, 1, 0)$ to "square", and $(0, 0, 1)$ to "triangle".

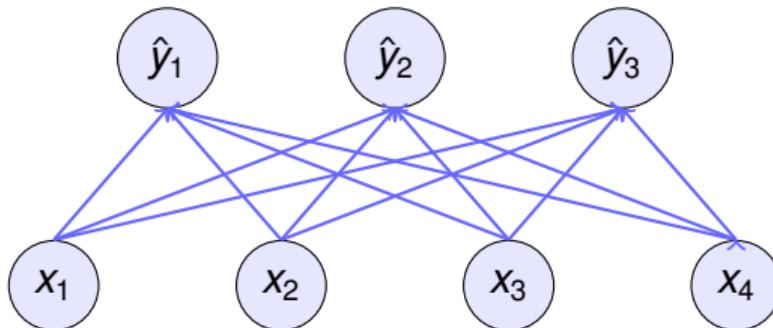
SINGLE-LAYER NEURAL NETWORK

- A model with multiple outputs, one per class. Each output will correspond to its own affine function.
- 4 features and 3 possible output categories
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.



ARCHITECTURE

- 12 scalars to represent the weights and 3 scalars to represent the biases .
- Compute three logits, \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 , for each input.
- Weights is a 3×4 matrix and bias is 1×3 matrix.



$$z_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1$$

$$\hat{y}_1 = \text{softmax}(z_1)$$

$$z_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2$$

$$\hat{y}_2 = \text{softmax}(z_2)$$

$$z_3 = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3$$

$$\hat{y}_3 = \text{softmax}(z_3)$$

SOFTMAX OPERATION

- Interpret the outputs of our model as probabilities.
- Any output \hat{y}_j is interpreted as the probability that a given item belongs to class j . Then choose the class with the largest output value as our prediction $\text{argmax}_j \hat{y}_j$.
- If \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 are 0.1, 0.8, and 0.1, respectively, then predict category 2.
- The softmax function transforms the outputs such that they become non-negative and sum to 1, while requiring that the model remains differentiable.

$$\hat{y} = \text{softmax}(Z) \quad \text{where} \quad \hat{y}_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}$$

- First exponentiate each logit (ensuring non-negativity) and then divide by their sum (ensuring that they sum to 1).
- Softmax is a non-linear function.

LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

- The softmax function gives a vector \hat{y} , which can be interpreted as estimated conditional probabilities of each class given any input x ,

$$\hat{y}_j = P(y = \text{class}_j \mid x)$$

- Compare the estimates with reality by checking how probable the actual classes are according to our model, given the features:

$$P(Y \mid X) = \prod_{i=1}^m P(y^{(i)} \mid x^{(i)})$$

LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

- Maximize $P(Y | X)$ = Minimize the negative log-likelihood

$$-\log P(Y | X) = \sum_{i=1}^m -\log P(y^{(i)} | x^{(i)}) = \sum_{i=1}^m loss P(y^{(i)}, \hat{y}^{(i)})$$

$$loss P(y^{(i)}, \hat{y}^{(i)}) = - \sum_{j=1}^m y_j \log \hat{y}_j$$

SOFTMAX EXAMPLE

- z_1 = Un-normalized log probabilities
- e^{z_1} = Un-normalized probabilities
- $\hat{y}_i = \text{softmax}(z_1)$ = normalized probabilities
- L_i = loss = $y_i \log \hat{y}_i$

class	z_1	e^{z_1}	$\text{softmax}(z_1)$	L_1
cat	3.2	24.5	0.13	0.89
car	5.1	164.0	0.87	0.06
dog	-1.7	0.18	0.00	∞

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

DEEP FEEDFORWARD NEURAL NETWORK

- A neural network with more number of hidden layers is considered as a **deep neural network**. There are no feedback connections.
- Convolutional networks are examples of feedforward neural networks.

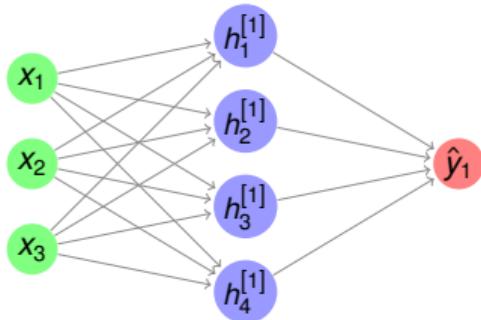


FIGURE: Shallow Neural Network

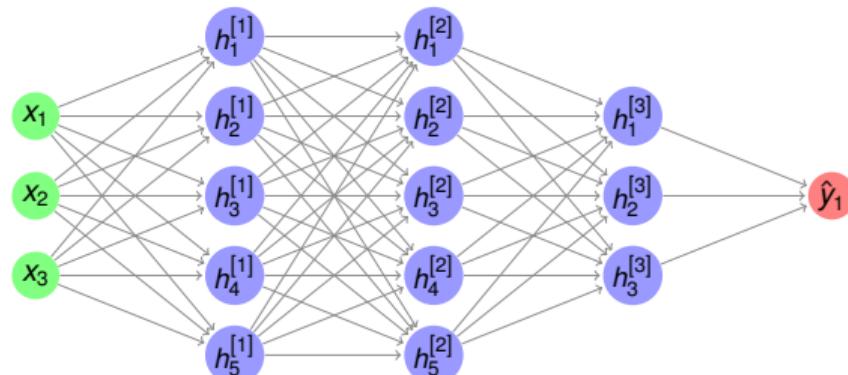
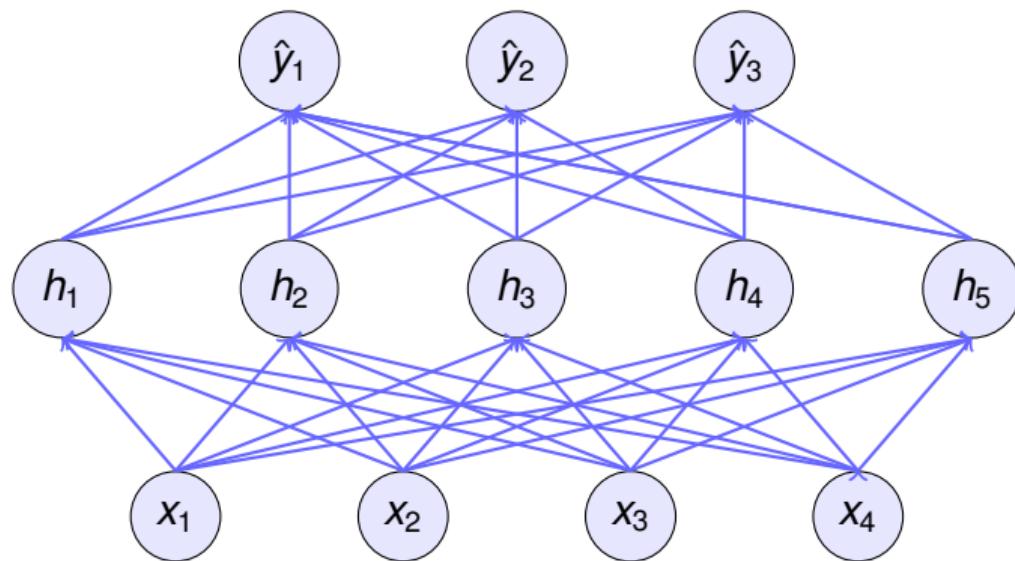


FIGURE: Deep Neural Network

DEEP FEEDFORWARD NEURAL NETWORK (DNN)

- With deep neural networks, use the data to jointly learn both a representation via hidden layers and a linear predictor that acts upon that representation.
- Add many hidden layers by stacking many fully-connected layers on top of each other. Each layer feeds into the layer above it, until we generate outputs.
- The first $(L - 1)$ layers learns the representation and the final layer is the linear predictor.

DNN ARCHITECTURE



- DNN has 4 inputs, 3 outputs, and its hidden layer contains 5 hidden units.
- Number of layers in this DNN is 2.

DNN ARCHITECTURE

- The layers are fully connected.
- Every input influences every neuron in the hidden layer.
- Each of the hidden neurons in turn influences every neuron in the output layer.
- The outputs of the hidden layer are called as **hidden representations** or hidden-layer variable or a hidden variable.

DNN ARCHITECTURE

HIDDEN LAYERS – intermediate layers, desired output for each of these layers are not shown.

DEPTH – number of layers.

WIDTH – dimensionality of the hidden layers

DNN – GENERAL STRATEGY

- Design a DNN architecture of the network. Architecture depends on the problem / application / complexity of the decision boundary.
- Choose the activation functions that will be used to compute the hidden layer activations. The activation function is the same for all neurons in a layer. Activation function can change between layers.
- Choose the cost function. It depends on whether regression, binary classification or multi-class classification.
- Choose the optimizer algorithm, depends on the complexity and variance of the input data.
- Train the feedforward network. Learning in deep neural networks requires computing the gradients using the back-propagation algorithm.
- Evaluate the performance of the network.

NON-LINEARITY IN DNN

- Input is $X \in \mathcal{R}^{m \times d}$ with m examples where each example has d features.
- Hidden layer has h hidden units $H \in \mathcal{R}^{m \times h}$.
- Hidden layer weights $W^{(1)} \in \mathcal{R}^{d \times h}$ and biases $b^{(1)} \in \mathcal{R}^{1 \times h}$
- Output layer weights $W^{(2)} \in \mathcal{R}^{d \times q}$ and biases $b^{(2)} \in \mathcal{R}^{1 \times q}$
- Output is $\hat{Y} \in \mathcal{R}^{m \times q}$
- A non-linear activation function σ has to be applied to each hidden unit following the affine transformation. The outputs of activation functions are called **activations**.

$$H = \sigma(XW^{(1)} + b^{(1)})$$

$$\hat{Y} = \text{softmax}(XW^{(2)} + b^{(2)})$$

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

ACTIVATION FUNCTIONS

- Activation function of a neuron defines the output of that neuron given an input or set of inputs.
- Introduces non-linearity to a neuron.
- A non-activated neuron will act as a linear regression with limited learning.
- Multilayered deep neural networks learn meaningful features from data.
- Artificial neural networks are designed as universal function approximators, they must have the ability to calculate and learn any non-linear function.

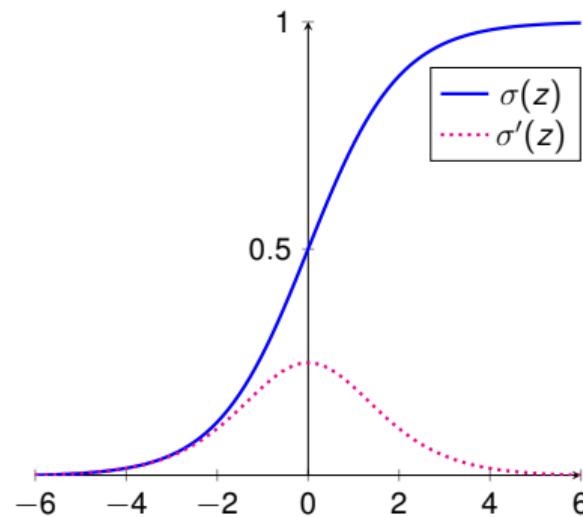
SIGMOID (LOGISTIC) ACTIVATION FUNCTION

$$\text{Function: } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\text{Derivative: } \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Range : $(0, 1)$

- Non-Linear function
- Small changes in z will be large in $f(z)$. This means it is a good classifier.
- Leads to vanishing gradient. So the learning is minimal.

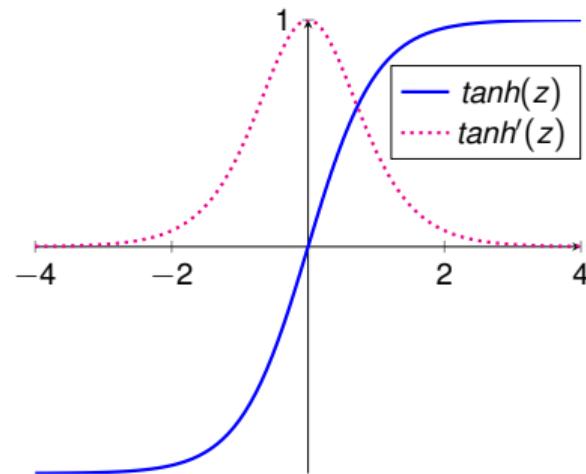


TANH ACTIVATION FUNCTION

$$\text{Function: } \tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$$

$$\text{Derivative: } \tanh'(z) = (1 - \tanh^2(z))$$

Range : $(-1, 1)$



TANH ACTIVATION FUNCTIONS

- More efficient because it has a wider range for faster learning and grading.
- The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- Issues with tanh
 - ▶ computationally expensive
 - ▶ lead to vanishing gradients

If we initialize the weights to relative large values, this will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

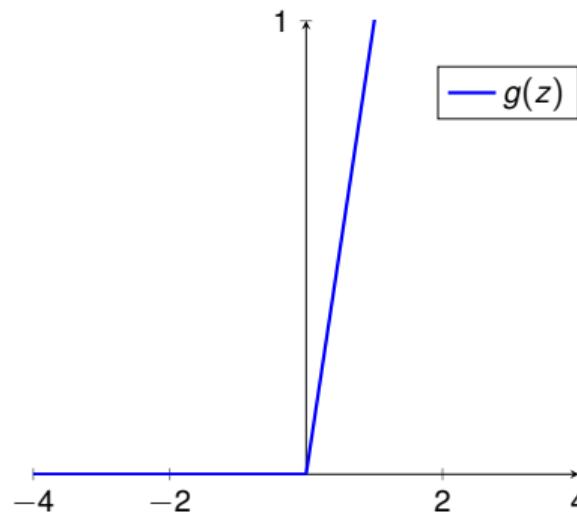
RELU ACTIVATION FUNCTION

Rectified Linear Unit

Function: $g(z) = \max(0, z)$

Derivative: $g'(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0 \end{cases}$

Range : $[0, \infty]$



RELU ACTIVATION FUNCTIONS

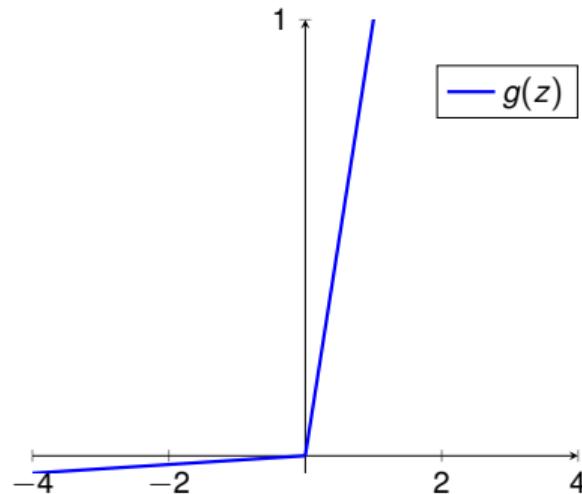
- ReLU activation function is not differentiable at origin.
- If we combine two ReLU units, we can recover a piece-wise linear approximation of the Sigmoid function.
- ReLU activation can lead to exploding gradient.
- In zero value region, learning is not happening.
- Fast Learning
- Fewer vanishing gradient problems
- Sparse activation
- Efficient computation
- Scale invariant (max operation)
- Non Zero centered
- Non differentiable at Zero

LEAKY RELU ACTIVATION FUNCTION

Function: $g(z) = \max(0.01z, z)$

Derivative: $g'(z) = \begin{cases} 0.01 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$



PARAMETRIC RELU (PRELU) ACTIVATION FUNCTION

Function: $g(z) = \max(az, z)$

Derivative: $g'(z) = \begin{cases} a & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$

ELU (EXPONENTIAL LINEAR UNITS)

Function:
$$g(z) = \begin{cases} z & \text{for } z < 0 \\ a(e^z - 1) & \text{for } z \geq 0 \end{cases}$$

Range : $[-\infty, \infty]$

COMPARING ACTIVATION FUNCTIONS

Activation Function	Sigmoid	Tanh	ReLU
Linearity	Non Linear	Non Linear	Non Linear
Activation Function	$\sigma(x) = \frac{1}{1+e^{-x}}$	$tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$	$g(x) = \max(0, z)$
Derivative	$\sigma(z)(1 - \sigma(z))$	$(1 - tanh^2(z))$	$1 \text{ if } z > 0 \text{ else } 0$
Symmetric Function	No	Yes	No
Range	$[0, 1]$	$[-1, 1]$	$[0, \infty]$
Vanishing Gradient	Yes	Yes	No
Exploding Gradient	No	No	Yes
Zero Centered	No	Yes	No

COMPARING ACTIVATION FUNCTIONS

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

ACTIVATION FUNCTIONS PLOTS

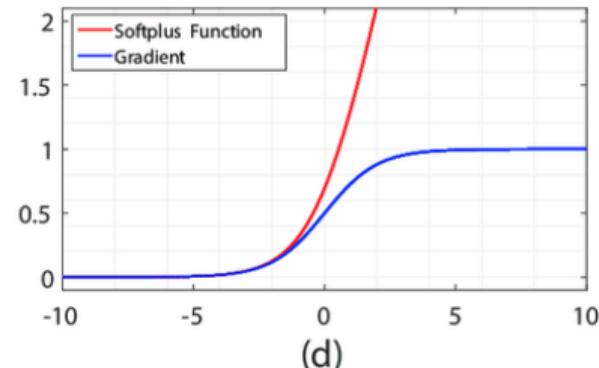
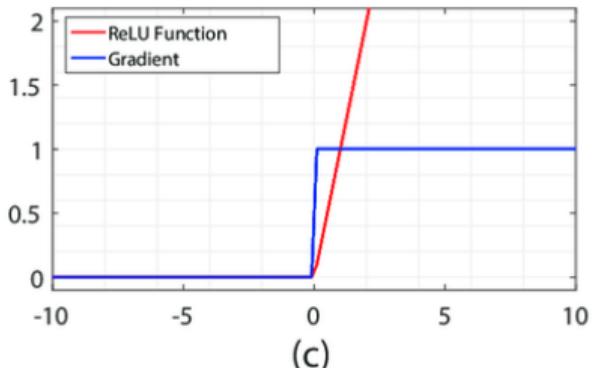
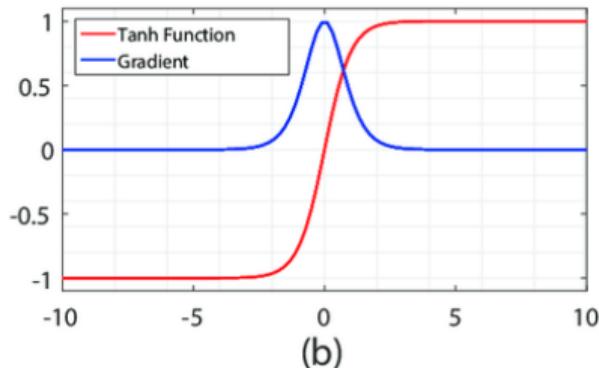
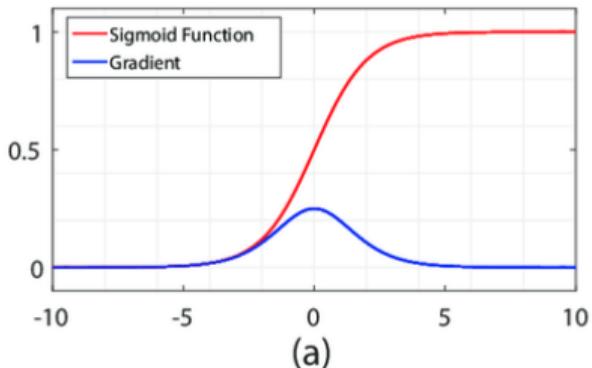


TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

COMPUTATION GRAPHS

- Computations of a neural network are organized in terms of
 - ▶ a forward pass or a forward propagation step, in which we compute the output of the neural network,
 - ▶ followed by a backward pass or back propagation step, which we use to compute gradients or derivatives.
- The computation graph organizes a computation with this blue arrow, left-to-right computation.
- The backward red arrow, right-to-left shows computation of the derivatives.

COMPUTATION GRAPHS

- Each node in the graph indicate a variable.
- An operation is a simple function of one or more variable.
- An operation is defined such that it returns only a single output variable.
- If a variable y is computed by applying an operation to a variable x , then we draw a directed edge from x to y .

COMPUTATION GRAPHS EXAMPLE

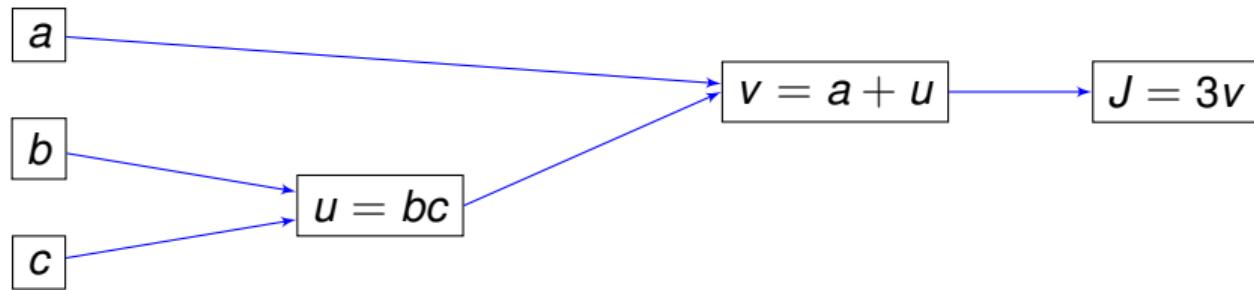
Forward Pass:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$



COMPUTATION GRAPHS EXAMPLE

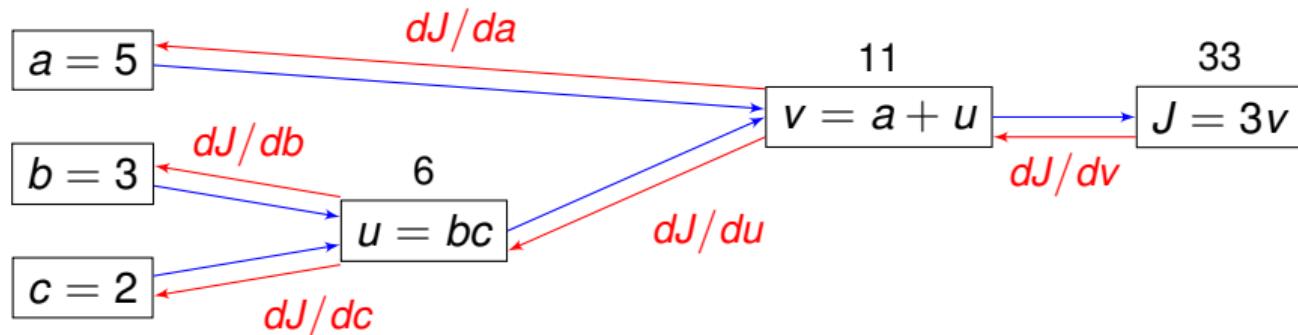
Backward Pass / Back propagation of gradients:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$

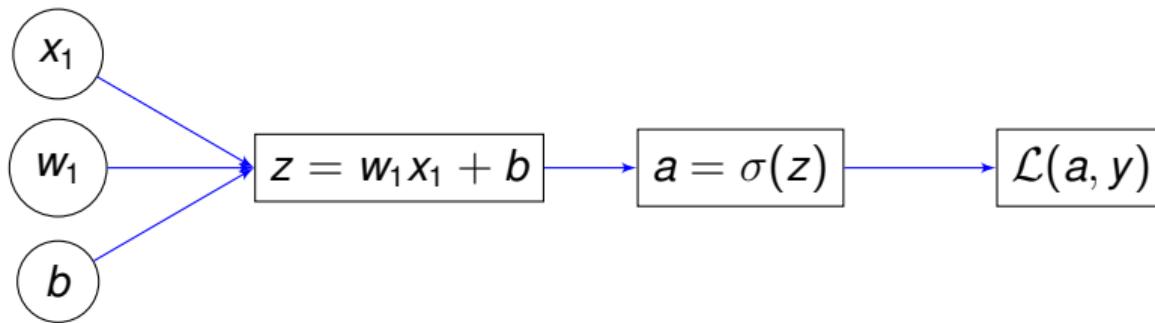


COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^\top X + b$$

$$a = \hat{y} = \sigma(z)$$

$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$



COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^\top X + b$$

$$a = \hat{y} = \sigma(z)$$

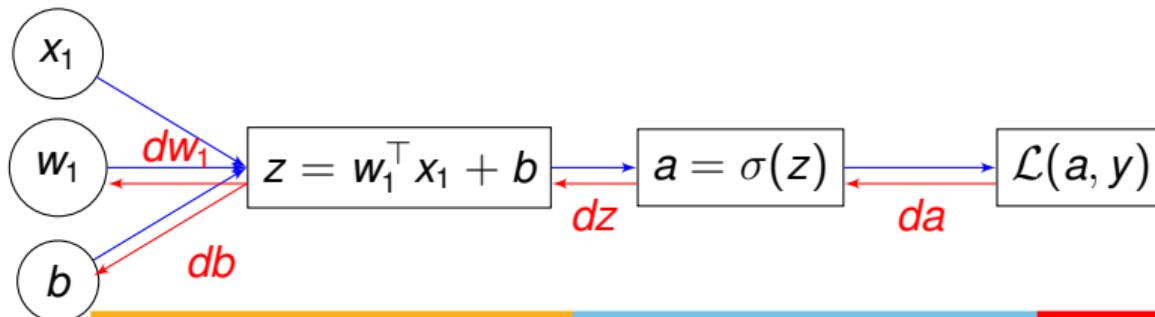
$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$

$$da = \frac{d\mathcal{L}(a, y)}{da} = \frac{-y}{a} + \frac{1+y}{1+a}$$

$$dz = \frac{d\mathcal{L}(a, y)}{dz} = a - y$$

$$dw_1 = \frac{d\mathcal{L}(a, y)}{dw_1} = x_1 dz$$

$$db = dz$$



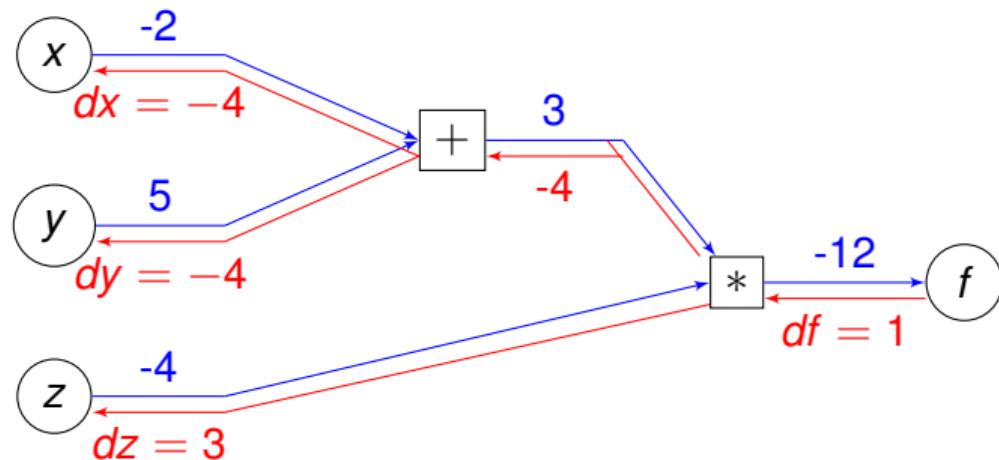
EXERCISE

Draw the computational graph for the equation

$$f(x, y, z) = (x + y)z$$

Assume that $x = -2$, $y = 5$ and $z = -4$. Using the computation graph show the computation of the gradients also.

EXERCISE - SOLUTION



EXERCISE

Draw the computational graph for the equation

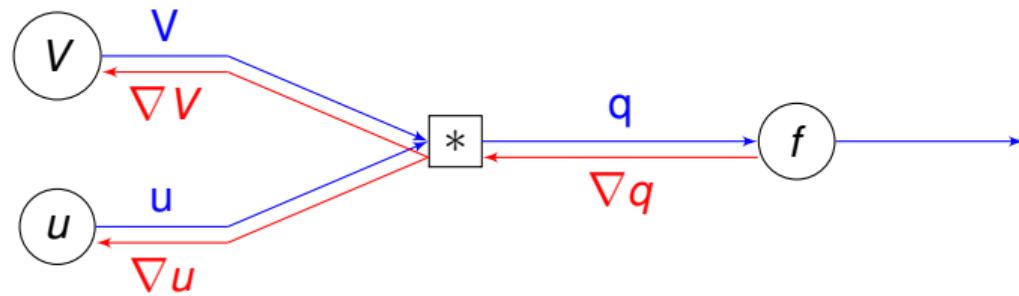
$$f(u, V) = \| V \cdot u \|^2 = \sum_{i=1}^n (V \cdot u)_i^2$$

Using the computation graph show the computation of the gradients also. Assume that

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

EXERCISE - SOLUTION



EXERCISE - SOLUTION

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$q = V \cdot u = \begin{bmatrix} 0.1 * 0.2 + 0.5 * 0.4 \\ -0.3 * 0.2 + 0.8 * 0.4 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$f = \sum_{i=1} q_i^2 = 0.22^2 + 0.26^2 = 0.116$$

EXERCISE - SOLUTION

$$\nabla q = \nabla_q f = 2q = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

$$\nabla V = \nabla_V f = 2q \cdot u^\top = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.088 & 0.176 \\ 0.105 & 0.208 \end{bmatrix}$$

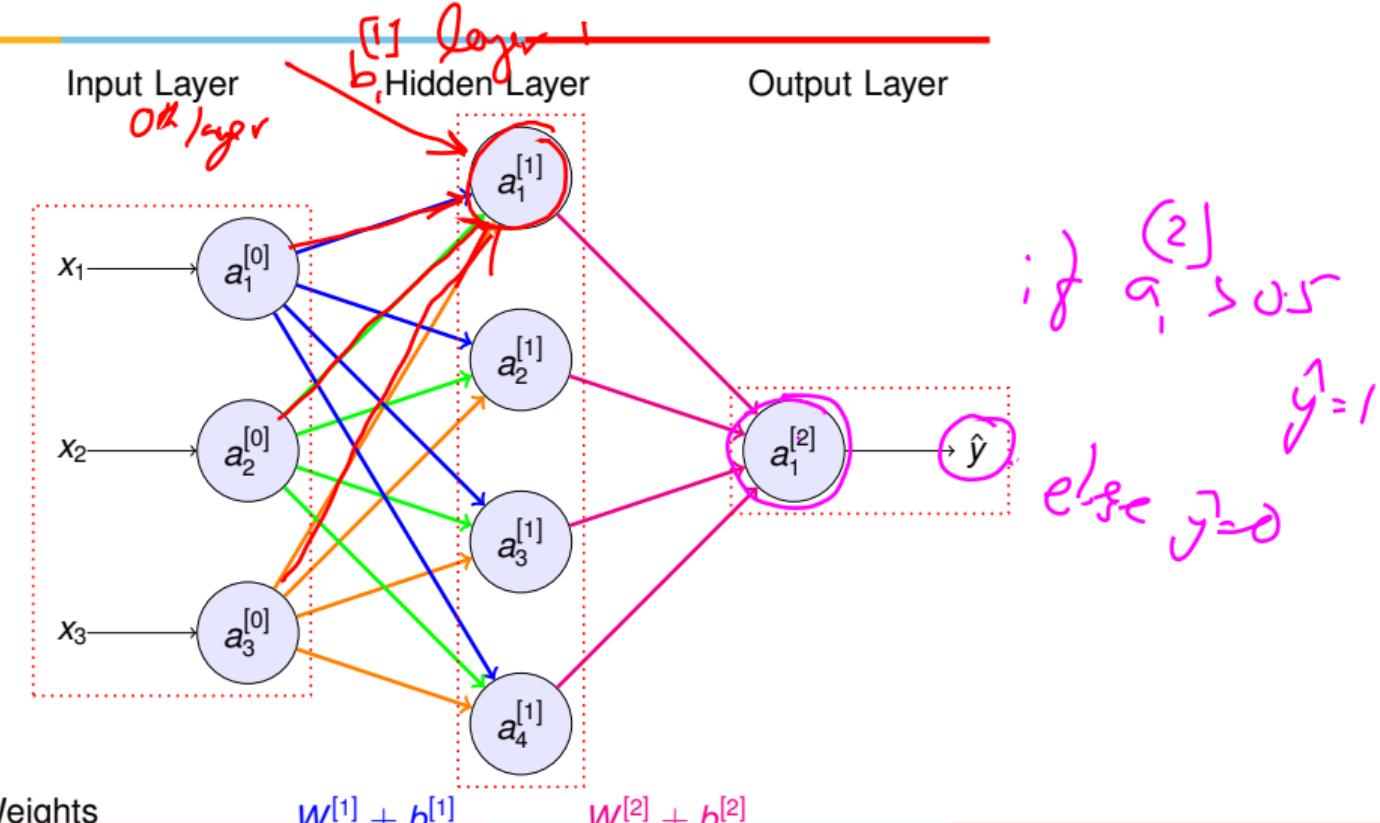
$$\nabla u = \nabla_u f = 2V^\top \cdot q = 2 \begin{bmatrix} 0.1 & -0.3 \\ 0.5 & 0.8 \end{bmatrix} \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

TWO LAYER NEURAL NETWORK ARCHITECTURE

$a_1^{[1]} =$
 $\beta_1 =$



COMPUTING THE ACTIVATIONS

For Hidden layer

$$z_1^{[1]} = w_1^{[1]} a^{[0]} + b^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]} a^{[0]} + b^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]} a^{[0]} + b^{[1]}$$

$$a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]} a^{[0]} + b^{[1]}$$

$$a_4^{[1]} = \sigma(z_4^{[1]})$$

~~$s_k = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$~~

In the matrix form

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} \dots w_1^{[1]\top} \dots \\ \dots w_2^{[1]\top} \dots \\ \dots w_3^{[1]\top} \dots \\ \dots w_4^{[1]\top} \dots \end{bmatrix}}_{4 \times 3} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{3 \times 1} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{4 \times 1}$$

$$z^{[1]} = \begin{bmatrix} w_1^{[1]\top} a^{[0]} + b_1^{[1]} \\ w_2^{[1]\top} a^{[0]} + b_2^{[1]} \\ w_3^{[1]\top} a^{[0]} + b_3^{[1]} \\ w_4^{[1]\top} a^{[0]} + b_4^{[1]} \end{bmatrix} \leftarrow 4 \times 1 \text{ matrix}$$

$$a^{[1]} = \sigma(z^{[1]}) \leftarrow 4 \times 1 \text{ matrix}$$

ACTIVATIONS FOR TRAINING EXAMPLES

Vectorizing for one training example

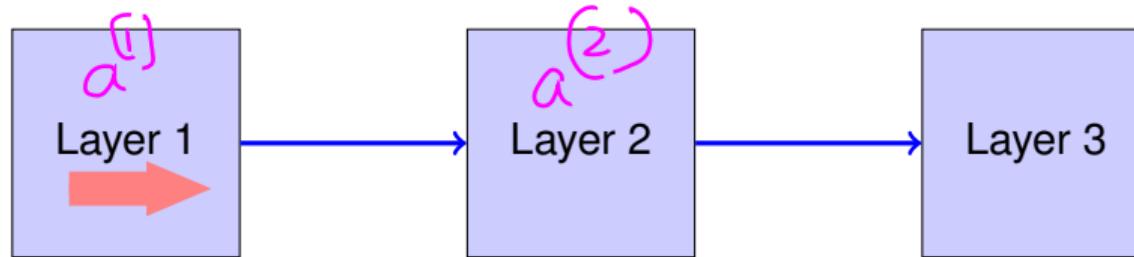
$$\begin{aligned}
 & \checkmark a^{[0]} = X \\
 & \left. \begin{aligned} Z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\ a^{[1]} &= \sigma(Z^{[1]}) \end{aligned} \right\} \text{1st hidden layer} \\
 & \left. \begin{aligned} Z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(Z^{[2]}) \end{aligned} \right\} \text{o/p} \\
 & \hat{y} = a^{[2]}
 \end{aligned}$$

Vectorizing for all training examples

$$\begin{aligned}
 \text{Example 1} : X^{(1)} &\longrightarrow a^{[2](1)} = \hat{y}^{(1)} \\
 \text{Example 2} : X^{(2)} &\longrightarrow a^{2} = \hat{y}^{(2)} \\
 \text{Example 3} : X^{(2)} &\longrightarrow a^{[2](3)} = \hat{y}^{(3)} \\
 &\vdots &&\vdots \\
 \text{Example m} : X^{(\underline{m})} &\longrightarrow a^{[2](m)} = \hat{y}^{(m)}
 \end{aligned}$$

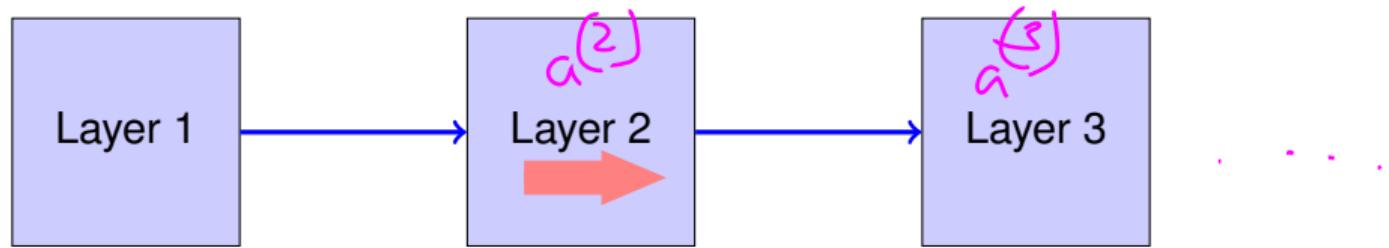
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



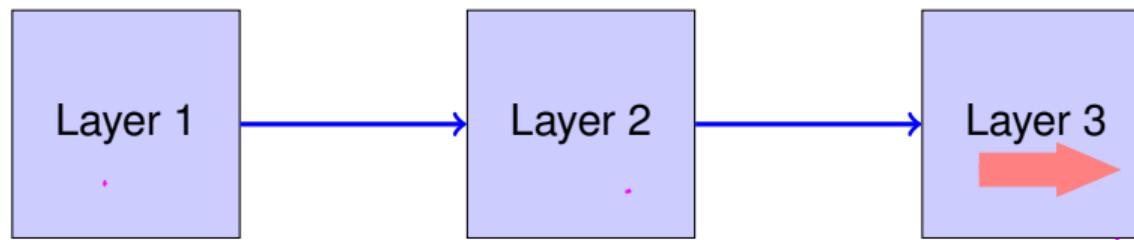
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



FORWARD PROPAGATION ALGORITHM

Algorithm 1: FORWARD PROPAGATION

1 Initialize the weights and bias randomly.

2 $a^{[0]} = X^{(i)}$

3 $Z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$

4 $a^{[1]} = \sigma(Z^{[1]})$

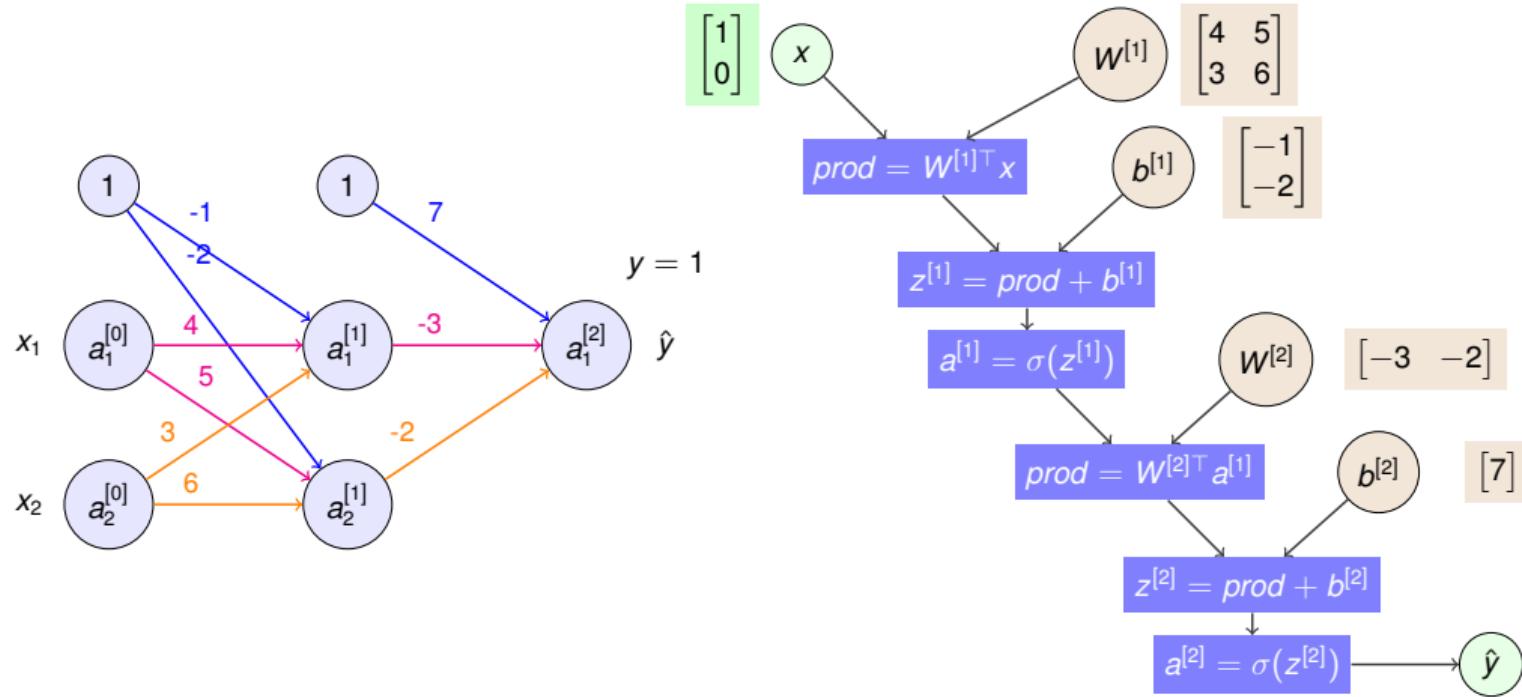
5 $Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

6 $a^{[2]} = \sigma(Z^{[2]})$

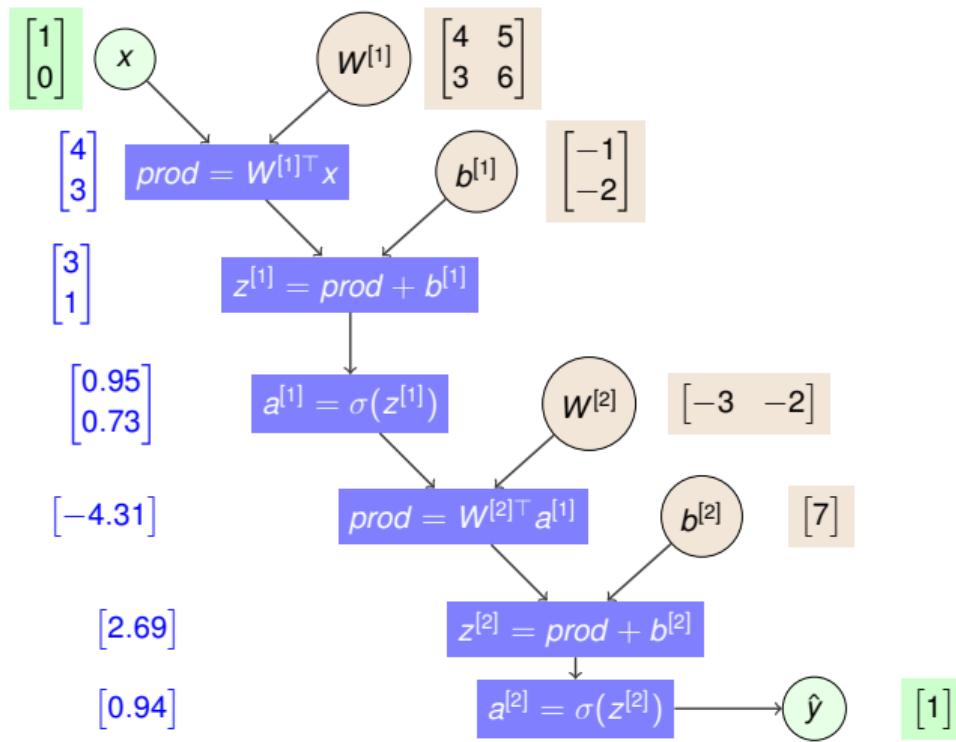
7 $\hat{y} = \begin{cases} 1 & \text{if } a^{[2]} > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Note: All training examples are considered in the vectorized form.

EXAMPLE NEURAL NETWORK



COMPUTATION GRAPH FOR FORWARD PASS



COST FUNCTION

- The difference between the actual observation y and the computed activation \hat{y} gives the error or the cost function.
- For binary classification,

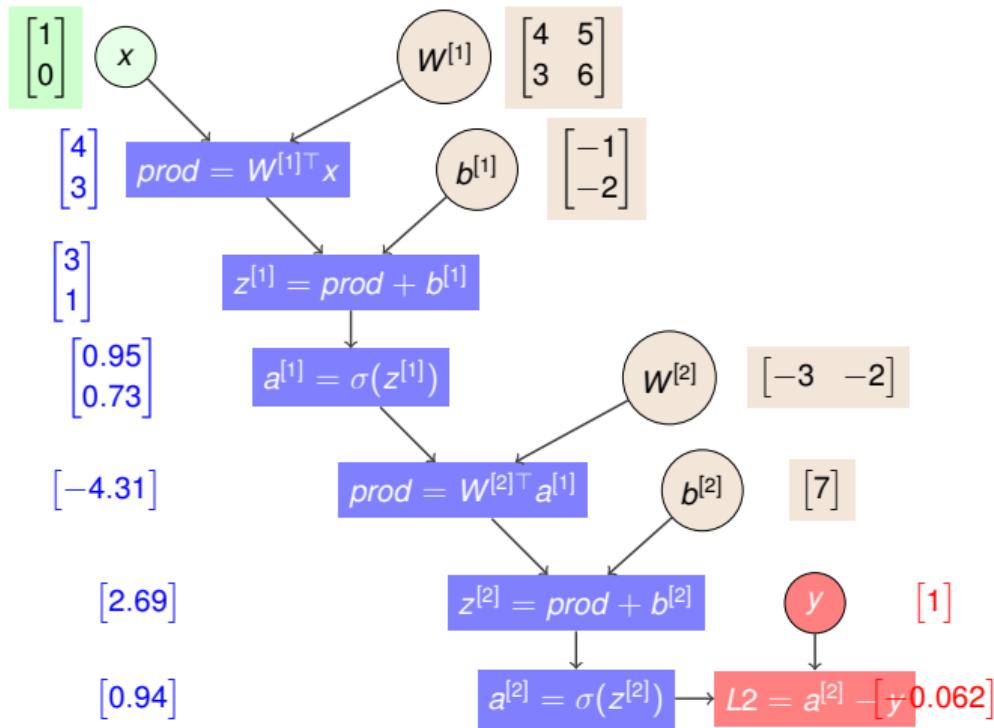
y , a

$$\mathcal{L} = \frac{1}{m} \sum_{i=0}^m \text{loss}(\hat{y}^{(i)}, y^{(i)})$$

$$\mathcal{L} = -\frac{1}{m} \sum_{i=0}^m (y^{(i)} \log(a^{[2](i)}) + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$$

$$\omega \leftarrow \omega - \eta \frac{\partial \mathcal{L}}{\partial \omega}$$

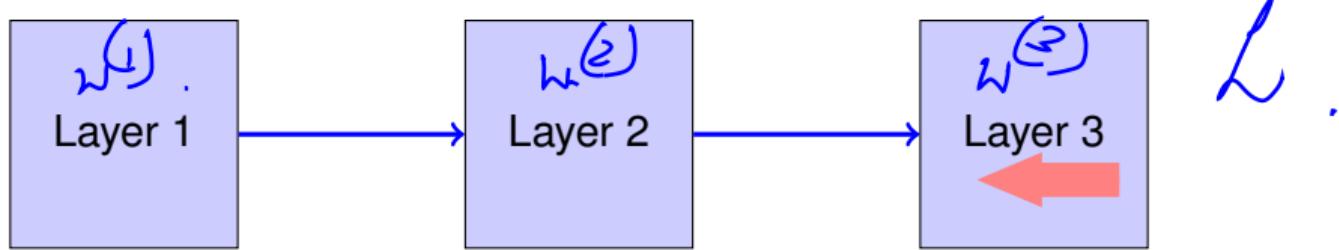
COMPUTATION GRAPH FOR COST FUNCTION



NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]



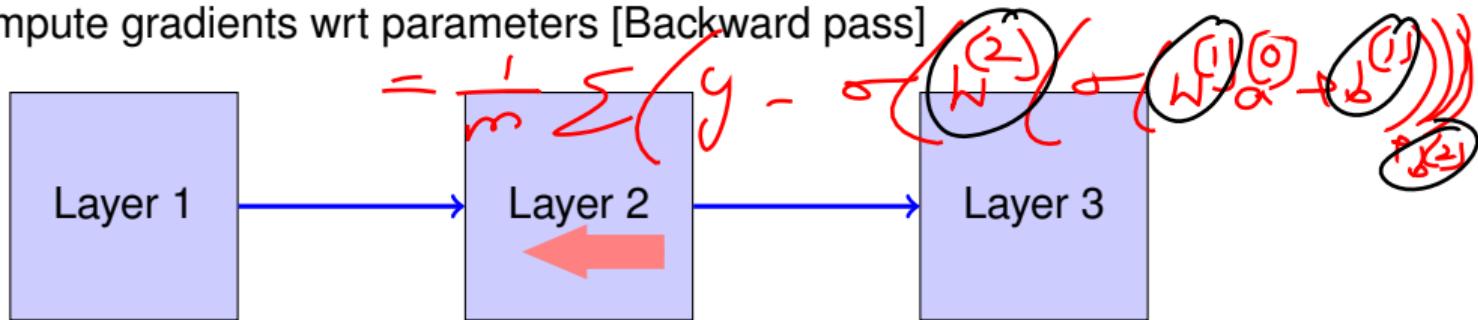
$$L = \frac{1}{m} \sum (y - \hat{a})^2$$

NEURAL NETWORK TRAINING - BACKWARD PASS

$$= \frac{1}{m} \sum (y - \sigma(W^{(2)}_a x^{(1)} + b^{(2)}))^2$$

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]



$$\frac{\partial L}{\partial W^{(2)}}$$

$$\frac{\partial L}{\partial b^{(2)}}$$

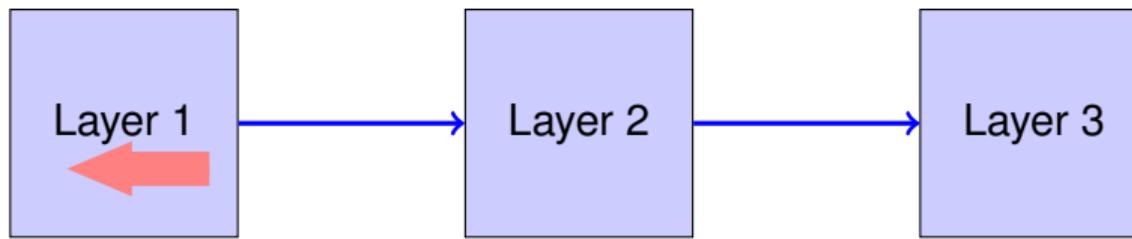
$$\frac{\partial L}{\partial W^{(1)}}$$

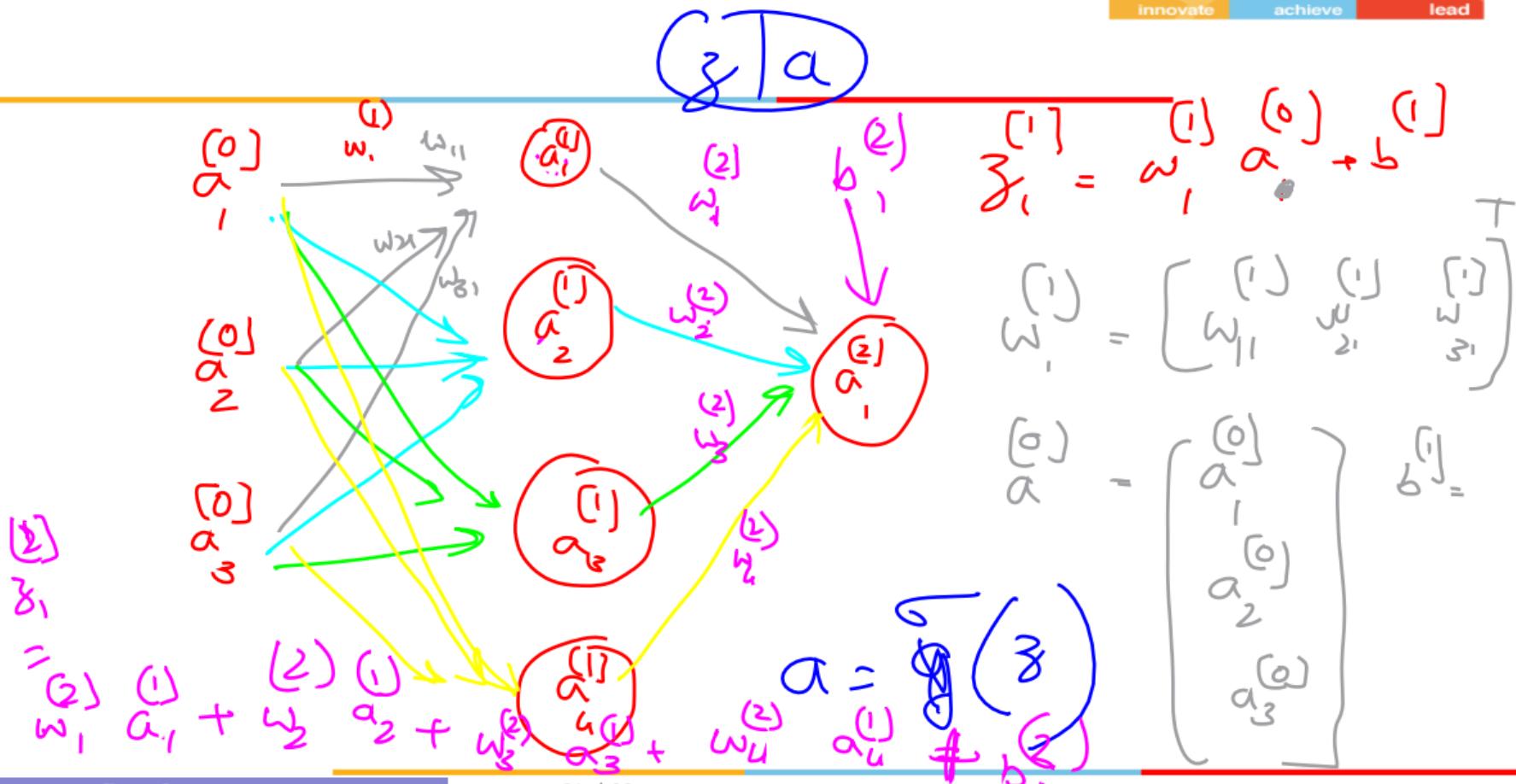
$$\frac{\partial L}{\partial b^{(1)}}$$

NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]





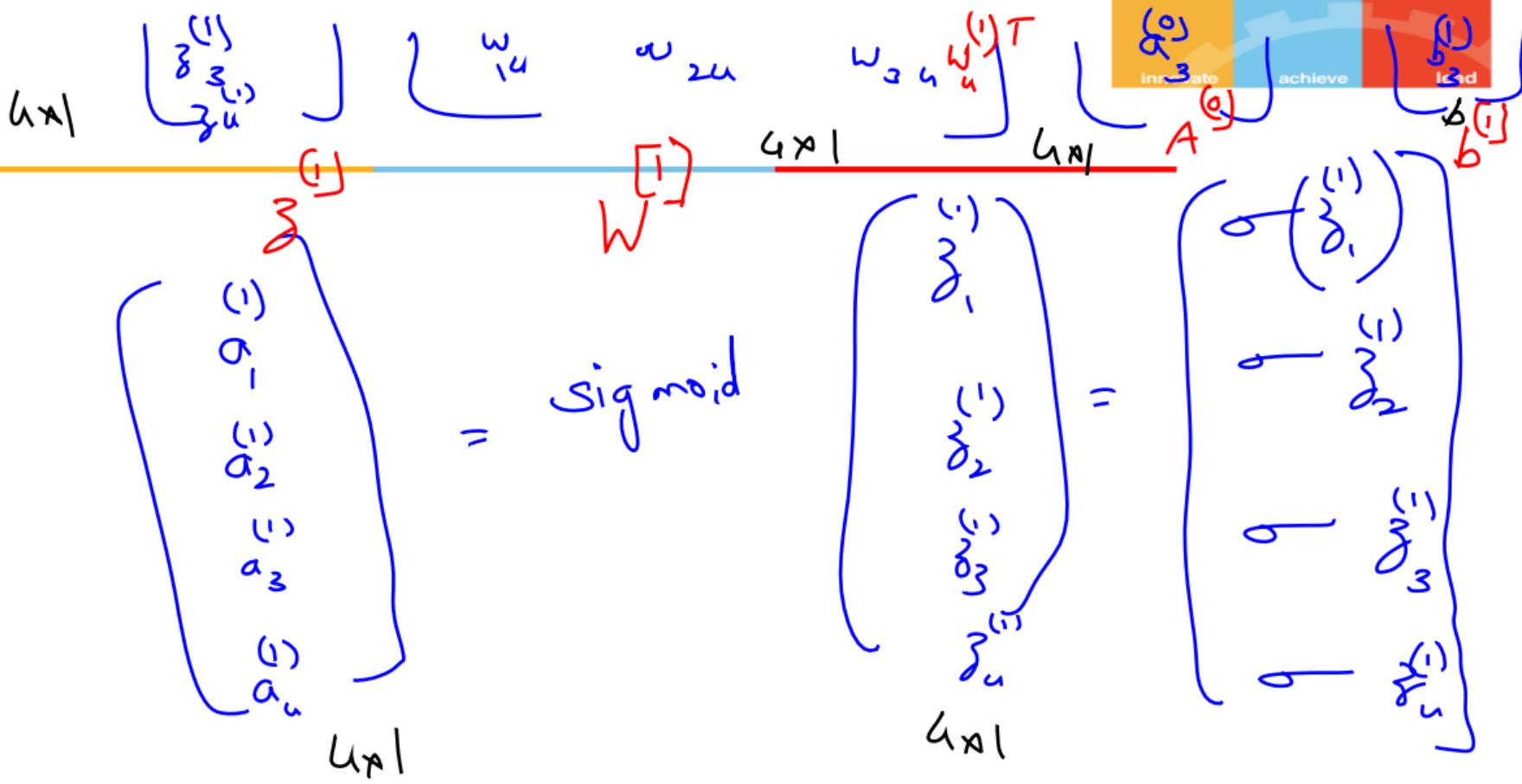
$$\hat{y}_1^{(1)} = w_{11}^{(1)} a_1^{(0)} + w_{21}^{(1)} a_2^{(0)} + w_{31}^{(1)} a_3^{(0)} + b_1^{(1)}$$

$$\hat{y}_2^{(1)} = w_{12}^{(1)} a_1^{(0)} + w_{22}^{(1)} a_2^{(0)} + w_{32}^{(1)} a_3^{(0)} + b_2^{(1)}$$

$$\hat{y}_3^{(1)} = w_{13}^{(1)} a_1^{(0)} + w_{23}^{(1)} a_2^{(0)} + w_{33}^{(1)} a_3^{(0)} + b_3^{(1)}$$

$$\hat{y}_4^{(1)} = w_{14}^{(1)} a_1^{(0)} + w_{24}^{(1)} a_2^{(0)} + w_{34}^{(1)} a_3^{(0)} + b_4^{(1)}$$

$$\begin{bmatrix} \hat{y}_1^{(1)} \\ \hat{y}_2^{(1)} \\ \hat{y}_3^{(1)} \\ \hat{y}_4^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}$$



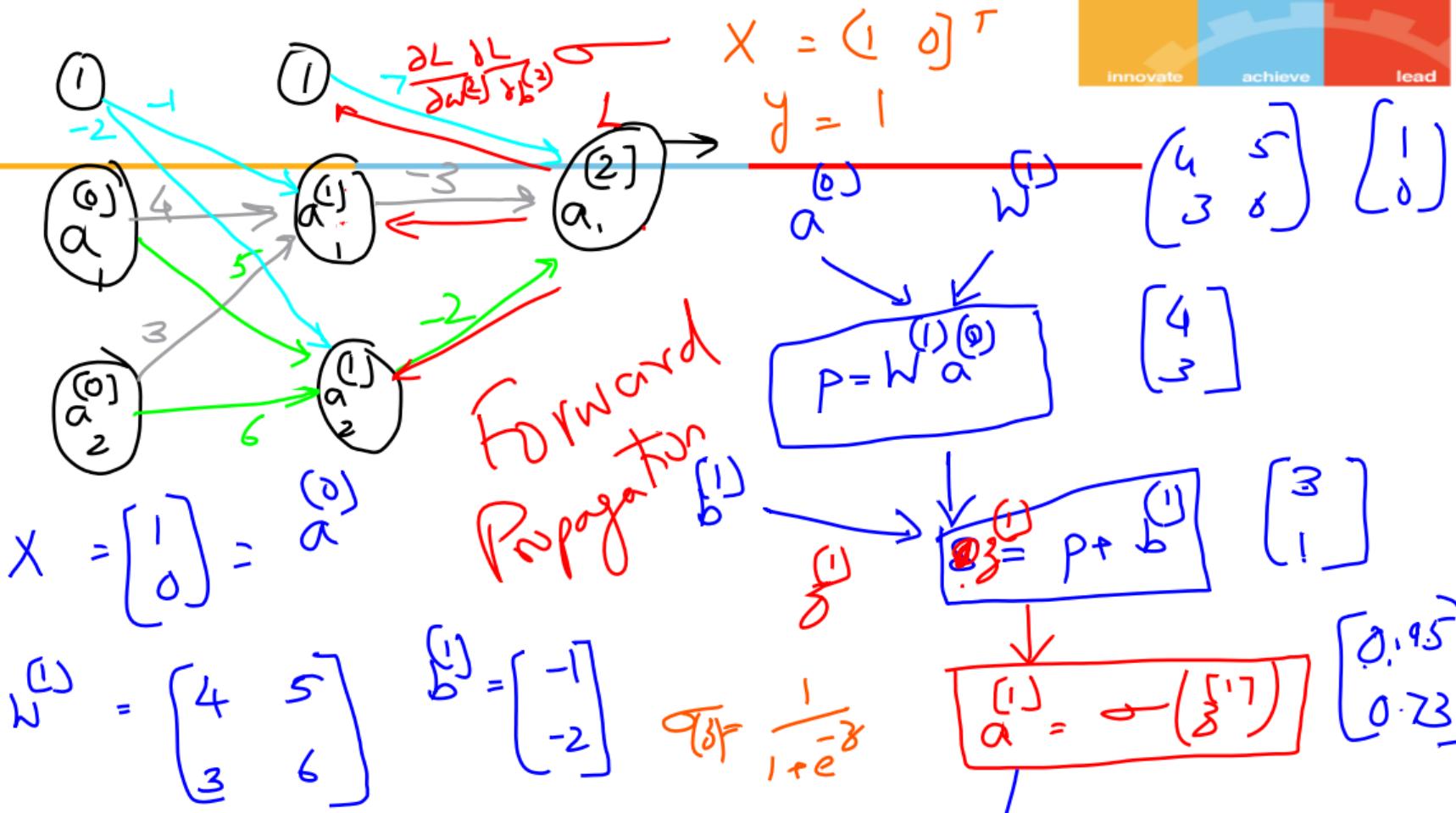
$$4 \times 1 \quad 4 \times 3 \quad 3 \times 1 \quad 4 \times 1$$

$$\vec{z}^{(1)} = W^{[1]} \vec{a}^{(0)} + \vec{b}^{(1)}$$

$$4 \times 1 \quad \vec{a}^{[1]} = \sigma(\vec{z}^{(1)})$$

$$\vec{z}^{(2)} = W^{(2)} \vec{a}^{[1]} + \vec{b}^{(2)}$$

$$\vec{a}^{(2)} = \sigma(\vec{z}^{(2)})$$



$$w^{(2)} = \begin{pmatrix} 0.2 \\ 0.1 \end{pmatrix}$$

$$b^{(2)} = 7$$

$$\begin{pmatrix} -3 \\ -2 \end{pmatrix}$$

$$w^T x + b = 2.87$$

$$\frac{\partial L}{\partial w} = (a - y)x.$$

$$\frac{\partial L}{\partial z} = a - y$$

$$w^{(2)} \downarrow \boxed{P = w^{(2)} a^{(1)}}$$

$$b^{(2)} \downarrow \boxed{z^{(2)} = P + b}$$

$$\alpha^{(2)} = \sigma(z^{(2)})$$

$$[-4.31]$$

$$\begin{pmatrix} 2.69 \\ \cancel{2.87} \end{pmatrix}$$

$$[0.93]$$



$$y = []$$

Compute loss.

$$L = (-1) \log_e(0.93) + (1-y) \log_e(1-0.93)$$

$$\frac{\partial L}{\partial \alpha^{(2)}} = a - y = 0.93 - 1 = -0.07$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial z^{(2)}} = -0.07$$

↓

$$\frac{\partial L}{\partial w^{(2)}} = (a-y)^{(1)} = (-0.07) \begin{bmatrix} 0.95 \\ 0.73 \end{bmatrix}$$

$$= \begin{bmatrix} -0.06 \\ -0.05 \end{bmatrix}$$

$$FP: \hat{y}^{(1)} = w^{(1)} a^{(0)} + b^{(1)}$$

$$a^{(1)} = \sigma(\hat{y}^{(1)})$$

$$\hat{y}^{(2)} = w^{(2)} a^{(1)} + b^{(2)}$$

$$\frac{\partial \hat{y}^{(2)}}{\partial a^{(1)}} = w^{(2)}$$

$$a^{(2)} = \sigma(\hat{y}^{(2)})$$

Loss:

$$L = \alpha e$$

gradient:

$$\frac{\partial L}{\partial \hat{y}^{(2)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial \hat{y}^{(2)}} = \frac{(2) - y}{a^{(2)}} = \frac{\partial L}{\partial b^{(2)}}$$

$$\frac{\partial L}{\partial w^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}} = (a^{(2)} - y) a^{(1)}$$

$$\frac{\partial L}{\partial a^{(1)}} = \left(\frac{\partial L}{\partial z^{(2)}} \right) \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} - (a^{(2)} - y) w^{(2)}$$

$$\begin{aligned} \frac{\partial L}{\partial z^{(1)}} &= \frac{\partial L}{\partial a^{(1)}} \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}} \right) \\ &= (a^{(2)} - y) w^{(2)} \cdot \sigma'(z^{(1)}) \\ &= (a^{(2)} - y) w^{(2)} \cdot \sigma(z^{(1)}) (1 - \sigma(z^{(1)})) \\ &= dz^{(1)} \end{aligned}$$

$$\frac{\partial L}{\partial w^{(0)}} = \frac{\partial L}{\partial z^{(0)}} \cdot \frac{\partial z^{(0)}}{\partial w^{(0)}} = dz^{(0)} \cdot a^{(0)}$$

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial b^{(1)}} = dz^{(1)} \cdot$$

$$a^{(2)} - y = -0.07$$

$$\frac{\partial L}{\partial z^{(1)}} = (-0.07) \begin{bmatrix} -3 \\ -2 \end{bmatrix}$$

$$\begin{pmatrix} 0.95 \\ 0.73 \end{pmatrix} \left(1 - \begin{pmatrix} 0.95 \\ 0.73 \end{pmatrix} \right)$$

$$= \begin{pmatrix} 0.27 \\ 0.14 \end{pmatrix} \begin{pmatrix} 0.95 \\ 0.73 \end{pmatrix} \begin{pmatrix} 0.05 \\ 0.27 \end{pmatrix}$$

$$\eta = 0.01 \Rightarrow \begin{pmatrix} 0.01 \\ 0.03 \end{pmatrix} = \frac{\partial L}{\partial b^{(1)}}_{\text{new}} = \begin{pmatrix} -1 \\ -2 \end{pmatrix} - 0.01 \begin{pmatrix} 0.01 \\ 0.03 \end{pmatrix}$$

$$\frac{\partial L}{\partial w} = \begin{pmatrix} 0.01 \\ 0.03 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 0.01 \\ 0.03 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \\ 3 \\ 6 \end{pmatrix}$$

$$W \leftarrow \begin{pmatrix} 4 & 5 \\ 3 & 6 \end{pmatrix} - 0.01 \begin{pmatrix} 0.01 & 0 \\ 0.03 & 0 \end{pmatrix}$$

BACK PROPAGATION OF COST FUNCTION

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$\frac{d(\log x)}{dx} = \frac{1}{x}$$

$$1-a$$

Weights

$$W^{[1]} + b^{[1]}$$

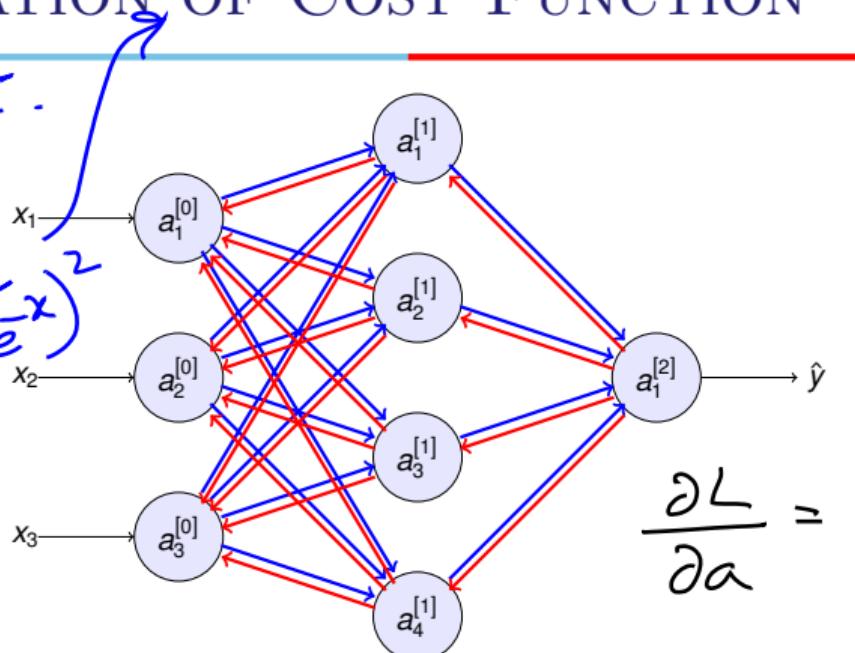
$$W^{[2]} + b^{[2]}$$

Gradients

$$da^{[1]} + dz^{[1]}$$

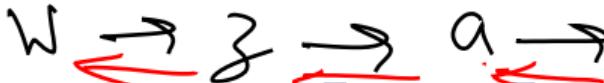
$$da^{[2]} + dz^{[2]}$$

$$\sigma(x)(1 - \sigma(x))$$



$$\frac{\partial L}{\partial a} = \frac{-y}{a} - \frac{(1-y)(-1)}{(1-a)}$$

$$= \frac{-y}{a} + \left(\frac{1-y}{1-a} \right)$$

~~w, a~~ 

achieve

lead

DERIVATION OF GRADIENTS IN LAST LAYER

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$z = w^T x + b$$

$$a = \sigma(z) \quad \frac{da}{dy} =$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z}$$

$$= \left(-\frac{y}{a} + \frac{1-y}{1-a} \right) a(1-a)$$

$$= (-y)(1-a) + (1-y)a$$

$$= -y + a - ay + a - ay = a - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w}$$

$$dz = (a-y)x$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} = \left(-\frac{y}{a} + \frac{1-y}{1-a} \right) a(1-a) = a - y$$

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w} = (a-y)x$$

$$\frac{\partial L}{\partial b} = (a-y)^{+1}$$

$$= a - y$$

COMPUTING THE GRADIENTS

For all training examples

For one training example

$$\left\{ \begin{array}{l} dz^{[2]} = a^{[2]} - y \\ dw^{[2]} = dz^{[2]} \cdot a^{[1]T} \\ db^{[2]} = dz^{[2]} \\ \\ dz^{[1]} = (w^{[2]T} dz^{[2]}) * \sigma'(z^{[1]}) \\ dw^{[1]} = dz^{[1]} \cdot x^T \\ db^{[1]} = dz^{[1]} \end{array} \right.$$

$$\begin{aligned} dZ^{[2]} &= A^{[2]} - Y \\ dW^{[2]} &= \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T}) \\ db^{[2]} &= \frac{1}{m} \sum dZ^{[2]} \\ dZ^{[1]} &= (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]}) \\ dW^{[1]} &= \frac{1}{m} (dZ^{[1]} \cdot X^T) \\ db^{[1]} &= \frac{1}{m} \sum dZ^{[1]} \end{aligned}$$

BACKWARD PROPAGATION ALGORITHM

Algorithm 2: BACKWARD PROPAGATION

$$1 \quad dZ^{[2]} = A^{[2]} - Y$$

$$2 \quad dW^{[2]} = \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T})$$

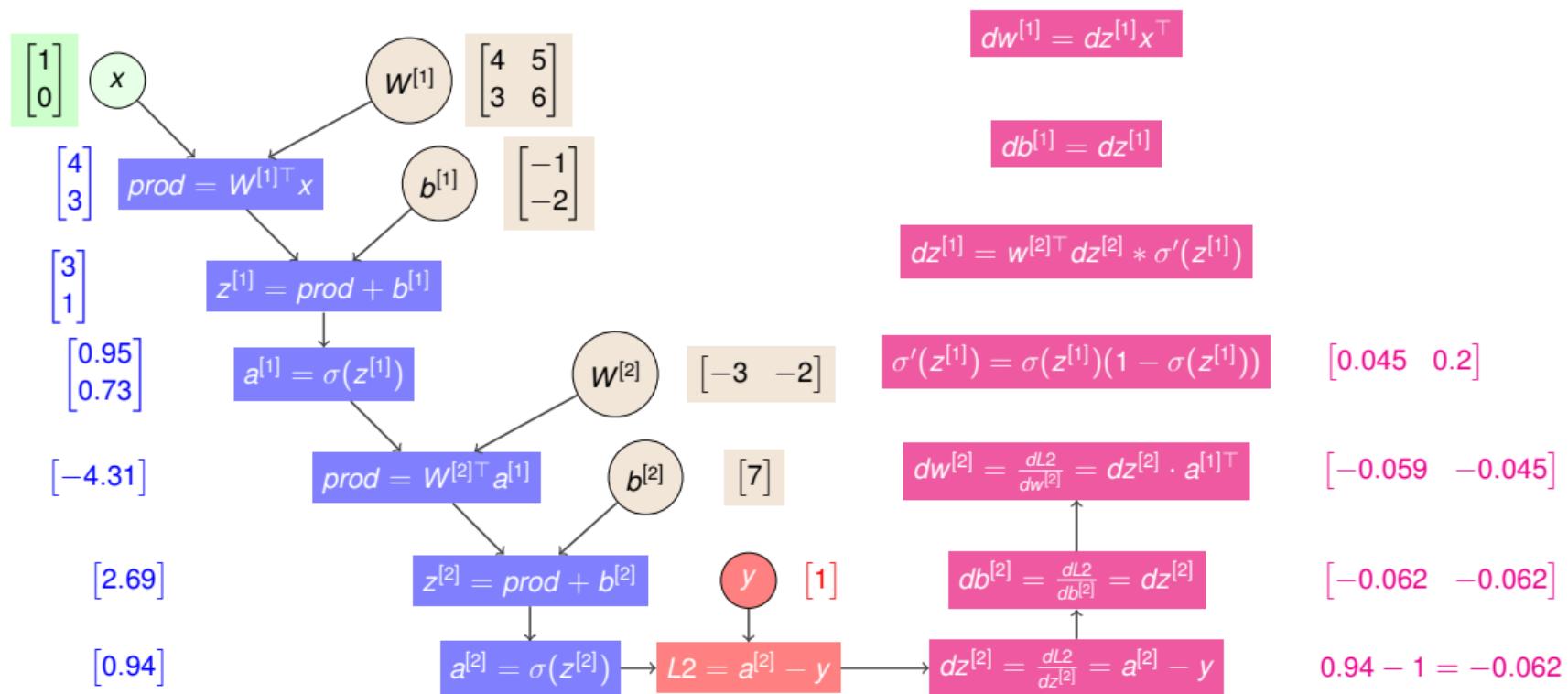
$$3 \quad db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$4 \quad dZ^{[1]} = (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]})$$

$$5 \quad dW^{[1]} = \frac{1}{m} (dZ^{[1]} \cdot X^T)$$

$$6 \quad db^{[1]} = \frac{1}{m} \sum dZ^{[1]}$$

COMPUTATION GRAPH FOR BACKWARD PASS

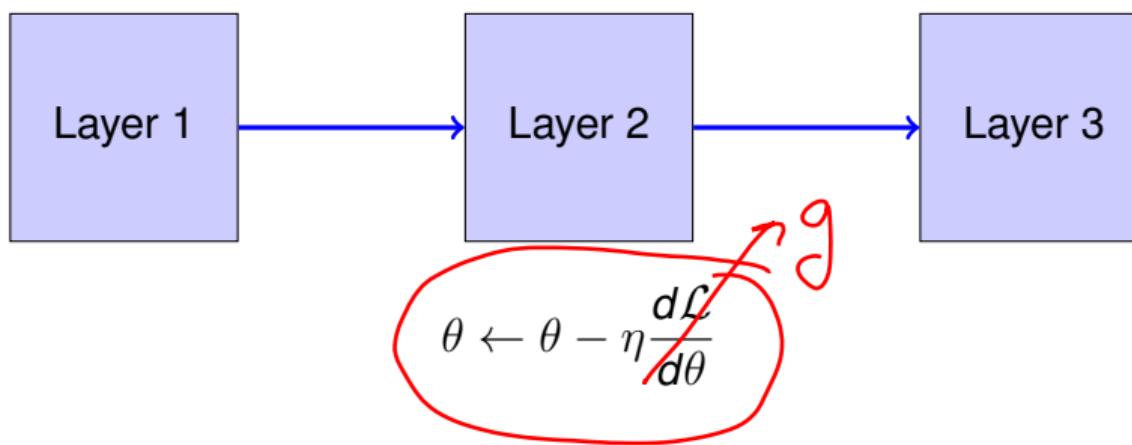


NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]

Step 3: Use gradients to update parameters

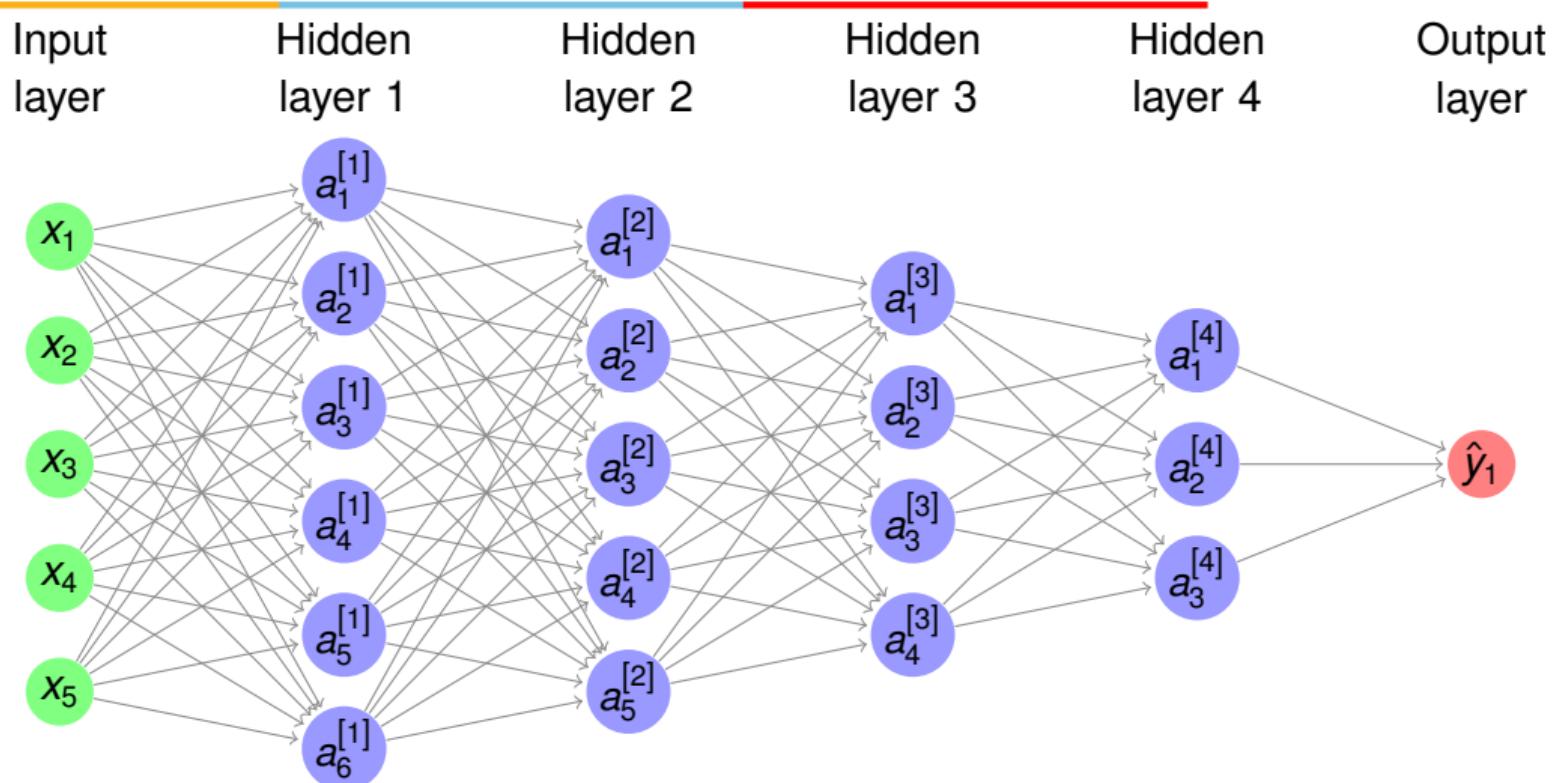


UPDATE THE WEIGHTS AND BIAS

Algorithm 3: WEIGHT UPDATION

- 1 $W^{[1]} = W^{[1]} - \text{learning rate} * dW^{[1]}$
 - 2 $b^{[1]} = b^{[1]} - \text{learning rate} * db^{[1]}$
 - 3 $W^{[2]} = W^{[2]} - \text{learning rate} * dW^{[2]}$
 - 4 $b^{[2]} = b^{[2]} - \text{learning rate} * db^{[2]}$
-

DNN ARCHITECTURE



TRAINING DNN

Requires

- ① **Forward Pass** through each layer to compute the output.
- ② Compute the deviation or error between the desired output and computed output in the forward pass (first step). This morphs into **objective function**, as we want to minimize this deviation or error. *Loss*
- ③ The deviation has to be send back through each layer to compute the delta or change in the parameter values. This is achieved using **back propagation algorithm**.
- ④ **Update** the parameters.

$$\frac{\partial L}{\partial \theta} = g$$

θ

DNN ARCHITECTURE

Notations

- L : Number of layers
- $n^{[l]}$: Number of units in layer l
- $w^{[l]}$: Weights for layer l
- $b^{[l]}$: Bias for layer l
- $z^{[l]}$: Hypothesis for layer l
- $g^{[l]}$: Activation Function used for layer l
- $a^{[l]}$: Activation for layer l

Matrix Dimensions

- $W^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $b^{[l]}$: $n^{[l]} \times 1$
- $z^{[l]}$: $n^{[l]} \times 1$
- $a^{[l]}$: $n^{[l]} \times 1$
- $dW^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $db^{[l]}$: $n^{[l]} \times 1$
- $dz^{[l]}$: $n^{[l]} \times 1$
- $da^{[l]}$: $n^{[l]} \times 1$

FORWARD PROPAGATION

- The inputs \mathbf{X} provide the initial information that then propagates up to the hidden units at each layer and finally produces \hat{y} . This is called forward propagation.
- Information flows forward through the network.
- During training, it produces a scalar cost $\mathcal{L}(\theta)$.

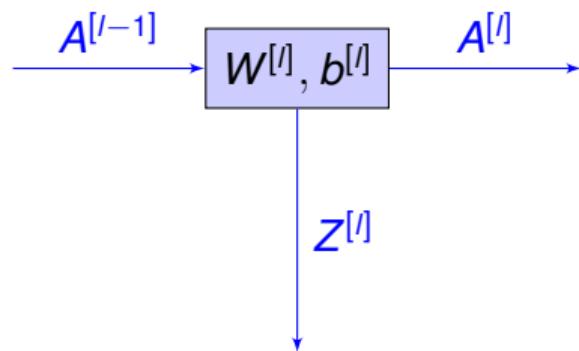
FORWARD PROPAGATION ALGORITHM

Algorithm 4: Forward Propagation

```

1 for  $l$  in range ( $1, L$ ) do
2      $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$ 
3      $A^{[l]} = g^{[l]}(Z^{[l]})$ 
4      $\hat{y} = A^{[L]}$ 
5      $J = L(\hat{y}, y) + \lambda \Omega(\theta)$ 

```



BACKWARD PROPAGATION

- Back-propagation algorithm or backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient $\nabla_{\theta} J(\theta)$.
- Back-propagation refers only to the method for computing the gradient.

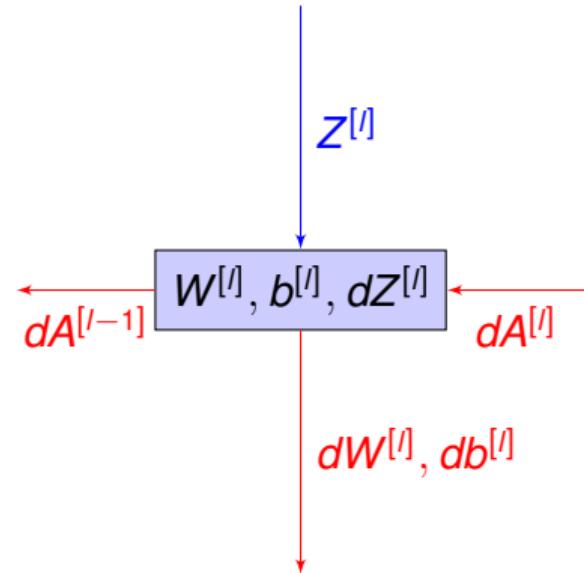
BACKWARD PROPAGATION ALGORITHM

Algorithm 5: Backward Propagation

```

1  $da^{[L]} = \frac{-y}{a} + \frac{1-y}{1-a}$ 
2 for  $i$  in range  $(1, L)$  do
3    $dZ^{[i]} = dA^{[i]} * g'^{[i]}(Z^{[i]})$ 
4    $dW^{[i]} = dZ^{[i]} \cdot A^{[i-1]T}$ 
5    $db^{[i]} = dZ^{[i]}$ 
6    $dA^{[i-1]} = W^{[i]T} \cdot dZ^{[i]}$ 

```



UPDATE THE WEIGHTS AND BIAS

Algorithm 6: Weight Updation

```
1 for  $l$  in range ( $1, L$ ) do
2      $W^{[l]} = W^{[l]} - \eta * dW^{[l]}$ 
3      $b^{[l]} = b^{[l]} - \eta * db^{[l]}$ 
```

OUTPUT NEURONS

- The choice of how to represent the output determines the form of the cross-entropy function.
- The feedforward network provides a set of hidden features $h = f(x; \theta)$.
- The role of the output layer is to provide some additional transformation.
- Types of output transformation
 - ① Linear
 - ② Sigmoid
 - ③ Softmax

LOSS FUNCTIONS

- Loss function compares the actual output from data set, in case of supervised learning, to the predicted output of the output layer.
- Based on the task, the loss functions differ.

For Regression	For Classification
Mean Squared Error (MSE)	Binary Cross Entropy (Sigmoid)
Mean Absolute Error (MAE)	Categorical Cross Entropy (Softmax)
Huber loss (Smooth MAE)	KL Divergence (Relative Entropy)
Log Hyperbolic Cosine (log cosh)	Exponential Loss
Quantile Loss	Hinge Loss
Cosine Similarity	

Further Reading

- ➊ Dive into Deep Learning (T1)



Thank You!



DEEP LEARNING MODULE # 3 : OPTIMIZATION

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.

WHAT WE LEARN ...

- ① 3.1 CHALLENGES IN NEURAL NETWORK OPTIMIZATION - SADDLE POINTS AND PLATEAU
- ② 3.2 NON-CONVEX OPTIMIZATION INTUITION
- ③ 3.3 OVERVIEW OF OPTIMIZATION ALGORITHMS
- ④ 3.4 MOMENTUM BASED ALGORITHMS
- ⑤ 3.5 ALGORITHMS WITH ADAPTIVE LEARNING RATES

VARIANCE VS BIAS

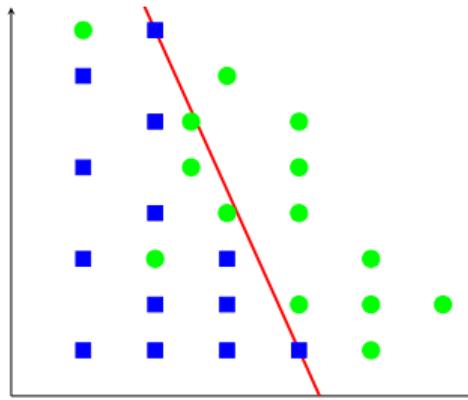


FIGURE: Underfitting; High Bias

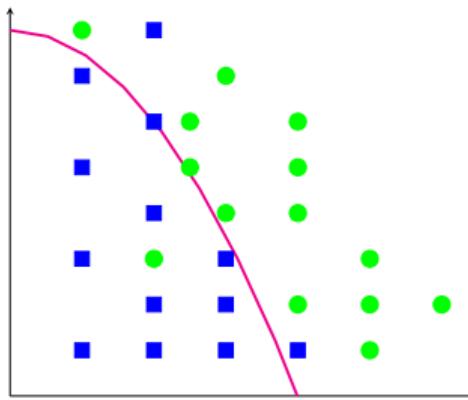


FIGURE: Low variance and bias

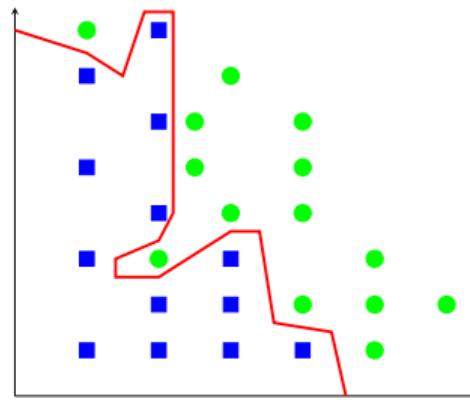
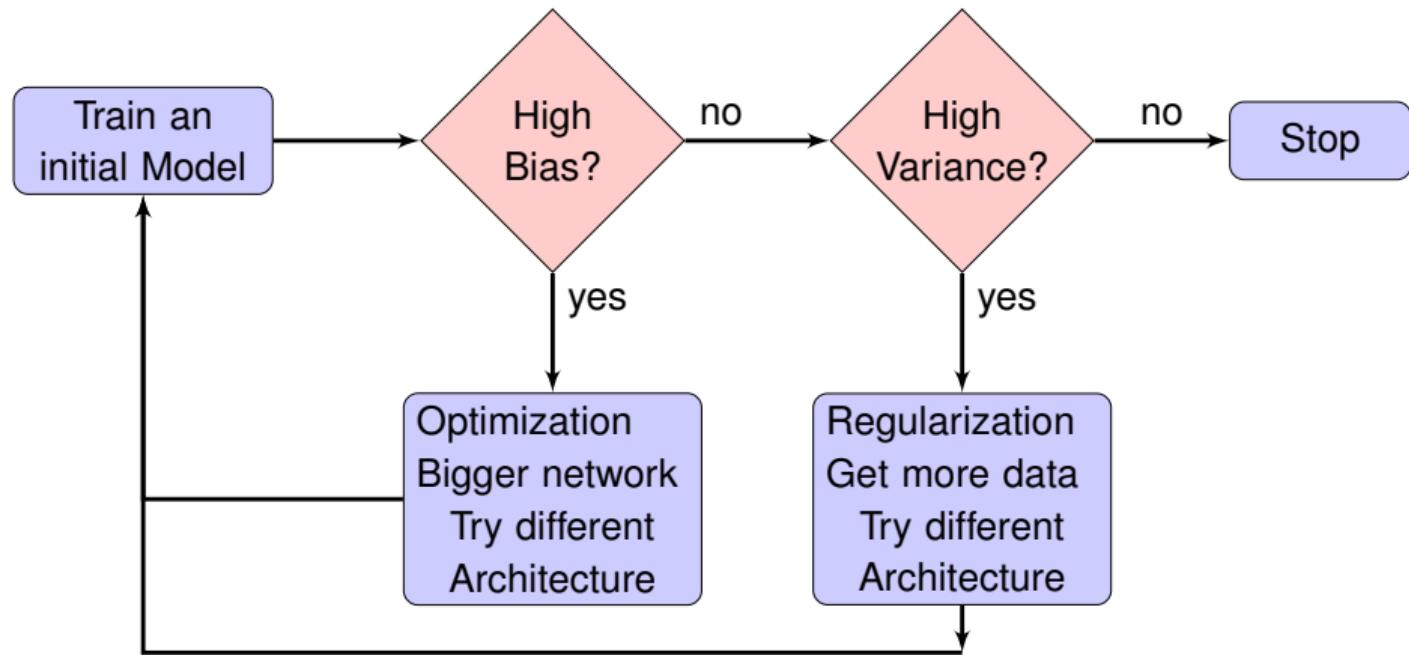


FIGURE: Overfitting; High variance

- Our aim is to achieve low variance and bias.

DEEP LEARNING RECIPE (ANDREW NG)



IN THIS SEGMENT

1 OPTIMIZATION

2 CHALLENGES IN OPTIMIZATION

3 OPTIMIZATION ALGORITHMS

- Gradient Descent
- Effect of Learning Rate on Gradient Descent
- Stochastic Gradient Descent
- Dynamic Learning Rate
- Minibatch Stochastic Gradient Descent
- Momentum
- Adagrad
- RMSProp
- ADAM

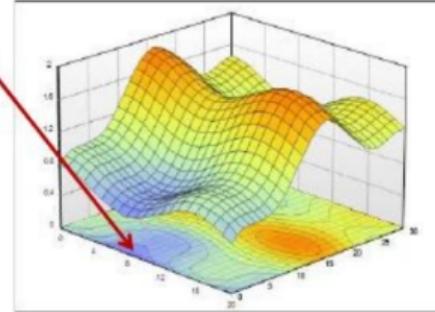
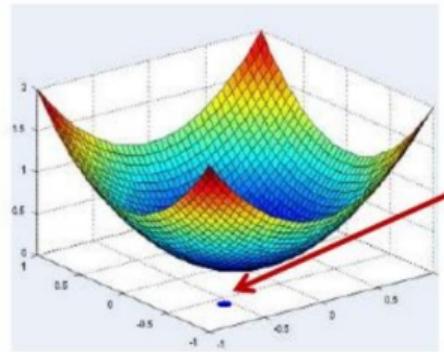
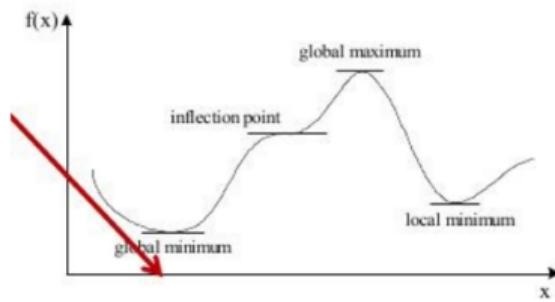
OPTIMIZATION

- Training of DNN is an iterative process.
- We may be training the DNN on a very large dataset or big data.
- The aim of DNN is improve the performance measure evaluated on the training set, i.e. **reduce the bias**.
- The performance measure P is optimized indirectly. Train the DNN to find the parameters θ that significantly reduce the cost function $J(\theta)$, so that P improves.

OPTIMIZATION ALGORITHMS

- Optimization algorithms train deep learning models.
- Optimization algorithms are the tools that allow
 - ▶ continue updating model parameters
 - ▶ to minimize the value of the loss function, as evaluated on the training set.
- In optimization, a loss function is often referred to as the **objective function** of the optimization problem.
- By tradition and convention most optimization algorithms are concerned with **minimization**.

OPTIMIZATION



- General problem of optimization: find the value of x where $f(x)$ is minimum

WHY OPTIMIZATION ALGORITHMS?

- The **performance** of the optimization algorithm directly affects the model's training efficiency.
- Understanding the principles of different optimization algorithms and the role of their hyperparameters will enable us to **tune the hyperparameters in a targeted manner** to improve the performance of deep learning models.
- **The goal of optimization is to reduce the training error.** The goal of deep learning is to reduce the generalization error, this requires reduction in overfitting also.

IN THIS SEGMENT

1 OPTIMIZATION

2 CHALLENGES IN OPTIMIZATION

3 OPTIMIZATION ALGORITHMS

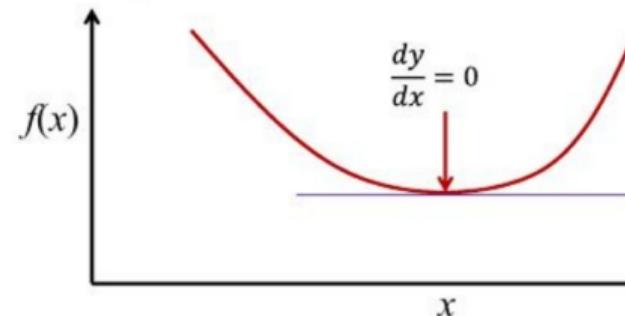
- Gradient Descent
- Effect of Learning Rate on Gradient Descent
- Stochastic Gradient Descent
- Dynamic Learning Rate
- Minibatch Stochastic Gradient Descent
- Momentum
- Adagrad
- RMSProp
- ADAM

CHALLENGES IN OPTIMIZATION

- Local minima
- Saddle points
- Vanishing gradients

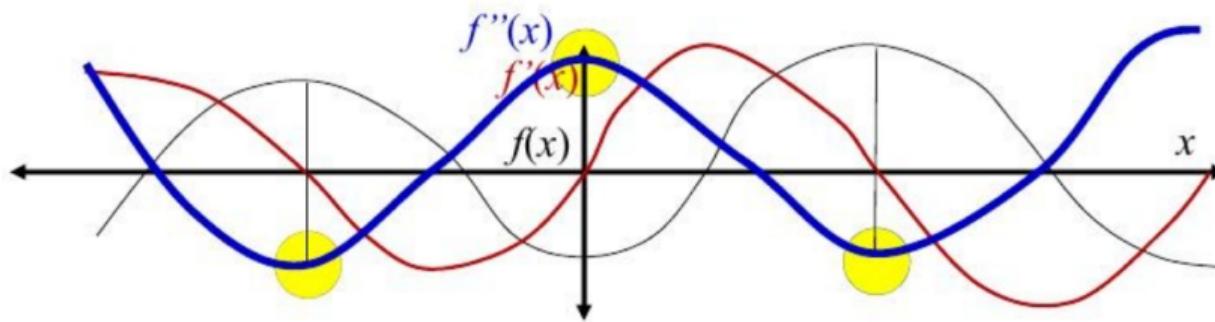
FINDING MINIMUM OF A FUNCTION

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution is at a turning point.
- A turning point is a value where the derivatives turn from positive to negative or vice versa.
- Turning points occur at minima, maxima or inflection (saddle) points.

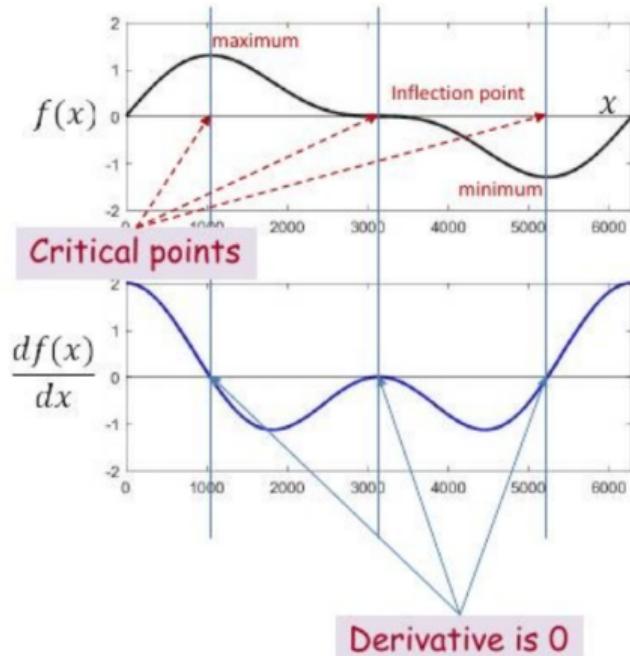


DERIVATIVES OF A FUNCTION

- Both maxima and minima are turning points.
- Both maxima and minima have zero derivatives.
- The second derivative $f''(x)$ is negative at maxima and positive at minima.

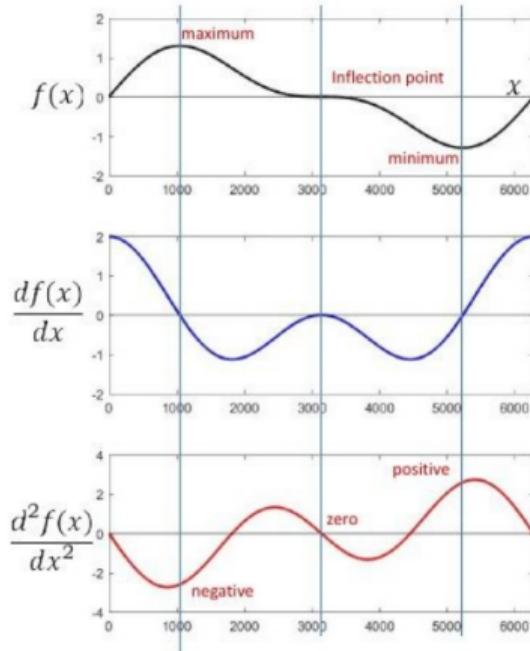


DERIVATIVES OF A FUNCTION



- All locations with zero derivative are **critical points**.
 - ▶ These can be local minima, local maxima or saddle points.

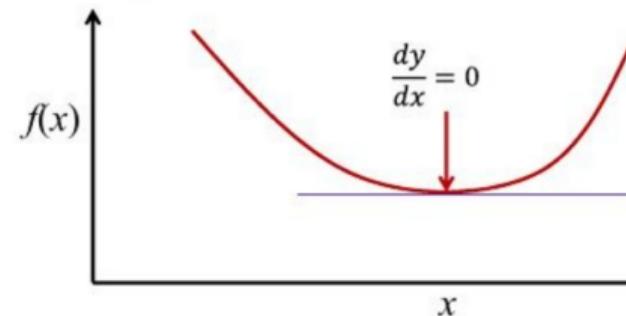
DERIVATIVES OF A FUNCTION



- All locations with zero derivative are **critical points**.
 - ▶ These can be local minima, local maxima or saddle points.
- The second derivative is
 - ▶ ≥ 0 at minima
 - ▶ ≤ 0 at maxima
 - ▶ $= 0$ at saddle point
- It is more complicated for functions of multiple variables.

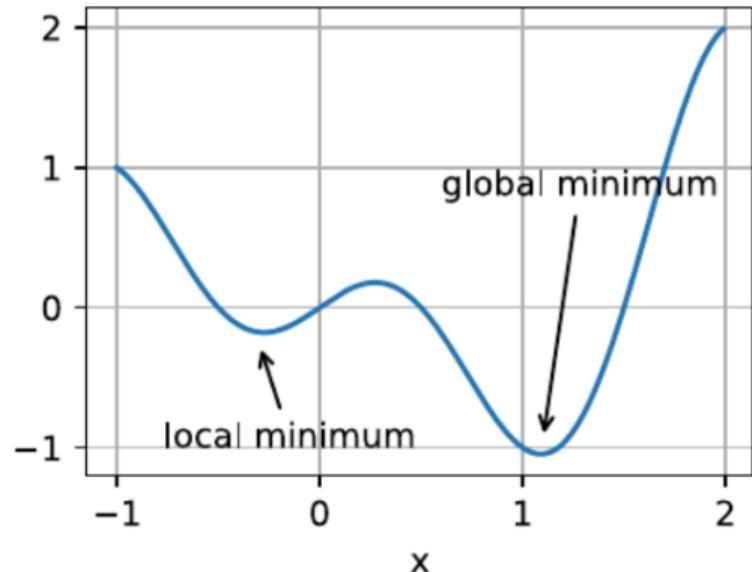
FINDING MINIMUM OF A FUNCTION

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution x_{soln} is at a turning point.
- Check the double derivative $f''(x_{soln})$ i.e $f''(x_{soln}) \frac{df'(x)}{dx}$.
- If $f''(x_{soln})$ is positive, x_{soln} is minimum.
- If $f''(x_{soln})$ is negative, x_{soln} is maximum.
- If $f''(x_{soln})$ is zero, x_{soln} is saddle point.



LOCAL MINIMA

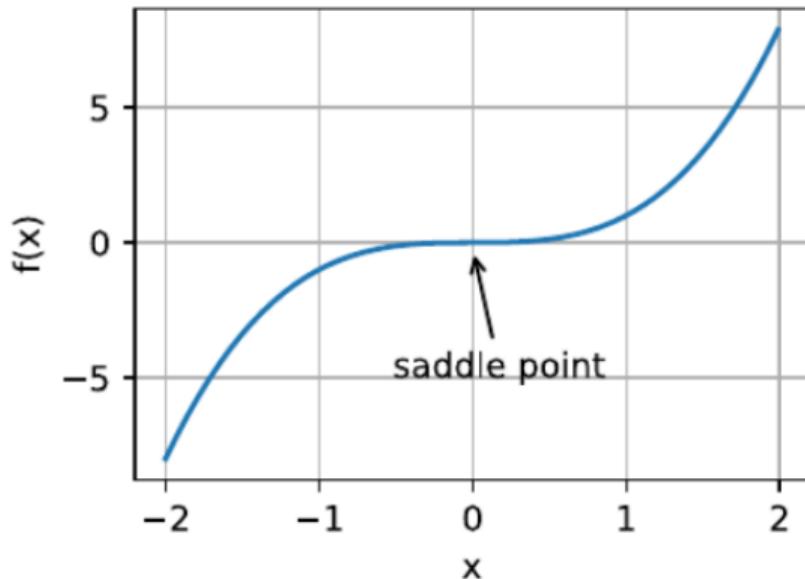
- For any objective function $f(x)$, if the value of $f(x)$ at x is smaller than the values of $f(x)$ at any other points in the vicinity of x , then $f(x)$ could be a local minimum.
- If the value of $f(x)$ at x is the minimum of the objective function over the entire domain, then $f(x)$ is the global minimum.
- In minibatch stochastic gradient descent, the natural variation of gradients over minibatches is able to dislodge the parameters from local minima.



SADDLE POINTS

- In high dimensional space, the local points with gradients closer to zero are known as saddle points.
- Saddle points tend to slow down the learning process.
- A saddle point is any location where all gradients of a function vanish but which is neither a global nor a local minimum.

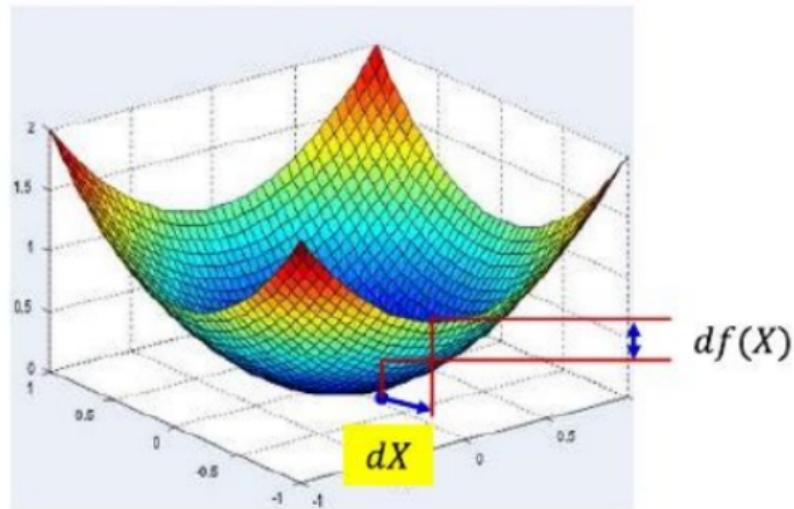
Eg: $f(x, y) = x^2 - y^2$
saddle point at $(0, 0)$; maximum wrt y and minimum wrt x



GRADIENT OF A SCALAR FUNCTION WITH MULTIPLE VARIABLES

- The gradient $\nabla f(X)$ of a multi-variate input X is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in X .

$$df(X) = \nabla f(X)d(X)$$



GRADIENT OF A SCALAR FUNCTION WITH MULTIPLE VARIABLES

- Consider $f(X) = f(x_1, x_2, \dots, x_n)$
- The gradient is given by

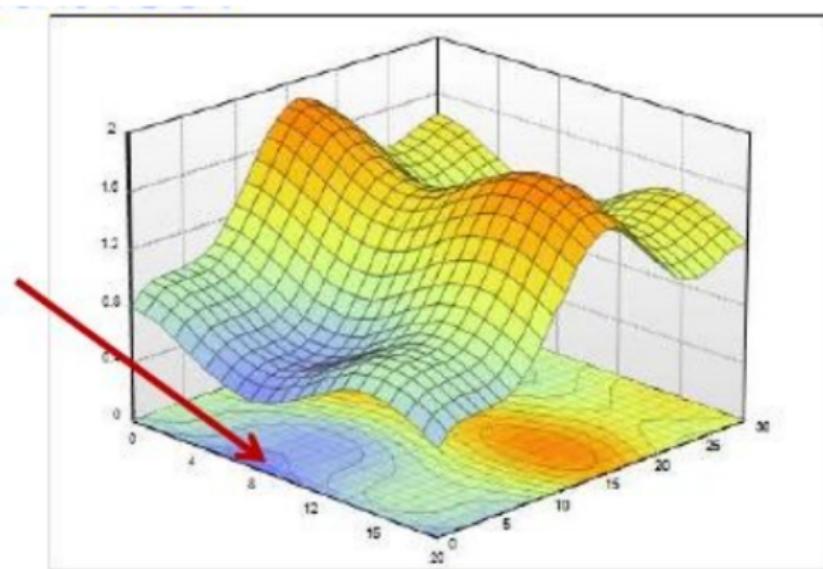
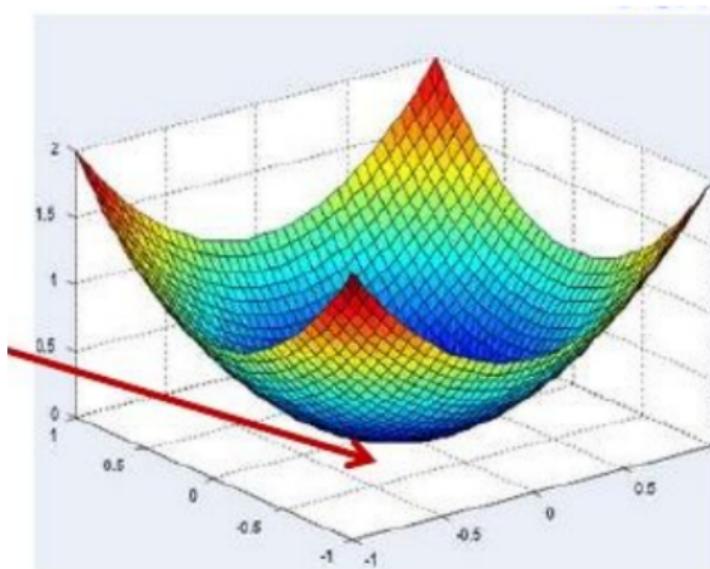
$$\nabla f(X) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \dots, \frac{\partial f(X)}{\partial x_n} \right]$$

- The second derivative of the function is given by the Hessian.

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

GRADIENT OF A SCALAR FUNCTION WITH MULTIPLE VARIABLES

- The optimum point is the turning point. The gradient is zero.



SOLUTION OF UNCONSTRAINED MINIMIZATION

- Solve for X where the gradient equation equal to zero.

$$\nabla f(X) = 0$$

- Compute the Hessian matrix $\nabla^2 f(X)$ at the candidate solution and verify that
 - Local Minimum
 - ▶ Eigenvalues of Hessian matrix are all positive.
 - Local Maximum
 - ▶ Eigenvalues of Hessian matrix are all negative.
 - Saddle Point
 - ▶ Eigenvalues of Hessian matrix at the zero-gradient position are negative and positive.

EXAMPLE

- Minimize

$$f(x_1, x_2, x_3) = x_1^2 + x_1(1 - x_2) + x_2^2 - x_2x_3 + x_3^2 + x_3$$

- Gradient

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

EXAMPLE

- Set Gradient = 0

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Solve the 3 equations in 3 unknowns.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

EXAMPLE

- Compute the Hessian

$$\nabla^2 f(X) = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

- Evaluate the eigenvalues of the Hessian matrix

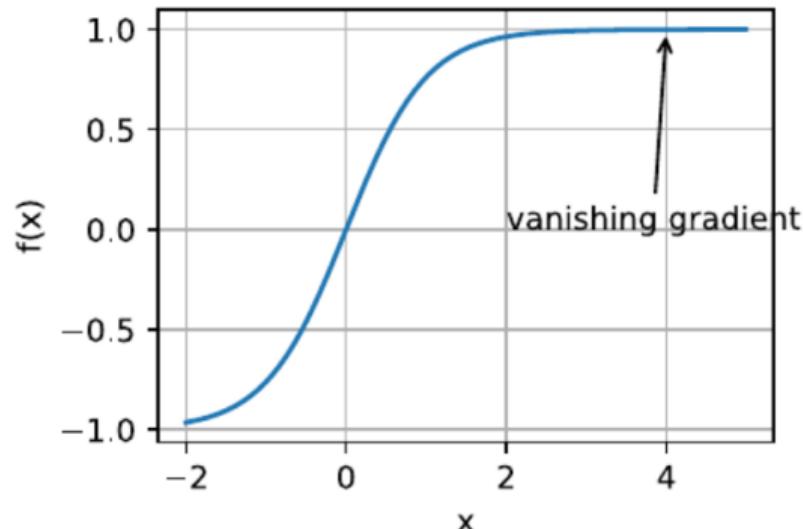
$$\lambda_1 = 3.414 \quad \lambda_2 = 0.586 \quad \lambda_3 = 2$$

- All Eigen vectors are positive.
- The obtained solution is minimum.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

VANISHING GRADIENTS

- Function $f(x) = \tanh(x)$
- $f'(x) = 1 - \tanh^2(x)$
- $f'(4) = 0.0013$.
- Gradient of f is close to nil.
- Vanishing gradients can cause optimization to stall.
- Reparameterization of the problem helps.
- Good initialization of the parameters.



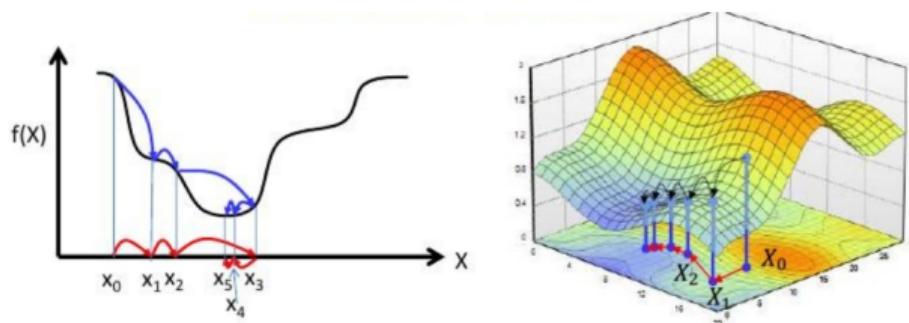
IN THIS SEGMENT

- 1 OPTIMIZATION
- 2 CHALLENGES IN OPTIMIZATION

3 OPTIMIZATION ALGORITHMS

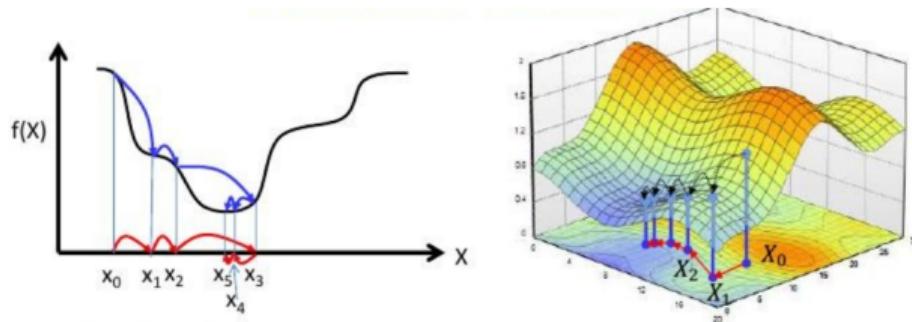
- Gradient Descent
- Effect of Learning Rate on Gradient Descent
- Stochastic Gradient Descent
- Dynamic Learning Rate
- Minibatch Stochastic Gradient Descent
- Momentum
- Adagrad
- RMSProp
- ADAM

HOW TO FIND GLOBAL MINIMA?



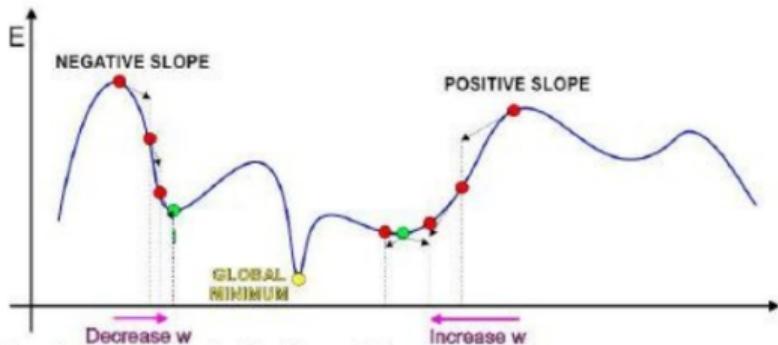
- It is not possible to solve $\nabla f(X) = 0$.
 - ▶ The function to minimize / maximize may have an intractable form
- In these situations, iterative solutions are used.
 - ▶ Begin with a guess for the optimal X and refine it iteratively until the correct value is obtained.

FIND GLOBAL MINIMA ITERATIVELY



- Iterative solutions
 - ▶ Start with an initial guess X_0 .
 - ▶ Update the guess towards a better value of $f(X)$.
 - ▶ Stop when $F(X)$ no longer decreases.
- Challenges
 - ▶ Which direction to step in?
 - ▶ How big the steps should be?

GRADIENT DESCENT



● Iterative Solution

- ▶ Start at some initial guess.
- ▶ Find the direction to shift this point to decrease error.
 - ★ This can be found from the derivative of the function.
 - ★ A positive derivative implies, move left to decrease error.
 - ★ A negative derivative implies, move right to decrease error.
- ▶ Shift the point in this direction.

GRADIENT DESCENT



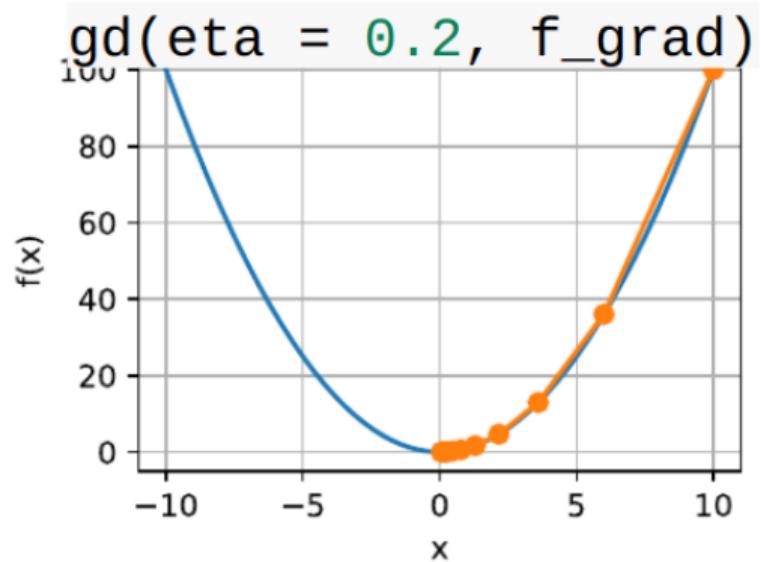
Algorithm 1: Gradient Descent Algorithm

- 1 Initialize x_0
 - 2 **while** $f'(x^k) \neq 0$ **do**
 - 3 | $x^{k+1} = x^k - \text{step_size}f'(x^k)$
-

GRADIENT DESCENT

- First order Gradient Descent algos consider the first order derivatives to get the magnitude and direction of update.

```
def gd(eta, f_grad):  
    x = 10.0  
    results = [x]  
    for i in range(10):  
        x -= eta * f_grad(x)  
        results.append(float(x))  
    return results
```



NUMERICAL EXAMPLE

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2) . Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t) . Assume $\alpha = 1.5$, $\beta = 0.6$, $\eta = 0.3$.

NUMERICAL EXAMPLE

Compute the value that minimizes (w_1, w_2) . Compute the minimum possible value of error.

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6} = 0$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6} = 0$$

Solving $w_1 = 3$

$$w_2 = 4$$

Substituting $E = 0.05$

NUMERICAL EXAMPLE

Compute the value of (w_1, w_2) at time $(t + 1)$ if standard gradient descent is used.

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6}$$

at $(t + 1)$

$$w_{1(t+1)} = w_{1(t)} - \eta \frac{\partial E}{\partial w_1} = 1.5 - 0.3 * \left[\frac{(1.5 - 3)}{2} - \frac{(2 - 4)}{6} \right] = 1.625$$

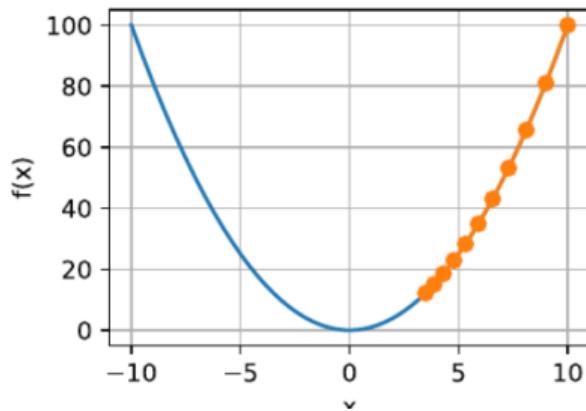
$$w_{2(t+1)} = w_{2(t)} - \eta \frac{\partial E}{\partial w_2} = 2 - 0.3 * \left[\frac{2(2 - 4)}{9} - \frac{(1.5 - 3)}{6} \right] = 2.05$$

LEARNING RATE

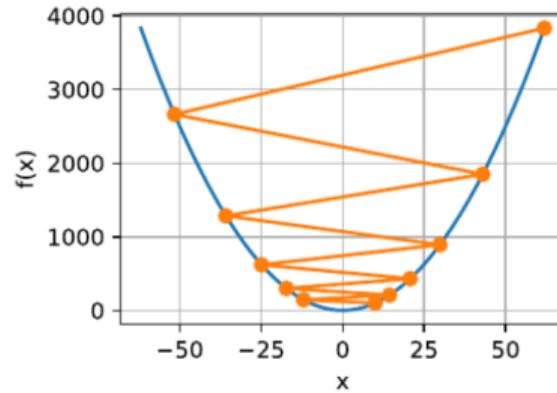
- The role of the learning rate is to moderate the degree to which weights are changed at each step.
- Learning rate η is set by the algorithm designer.
- If the learning rate that is too small, it will cause parameters to update very slowly, requiring more iterations to get a better solution.
- If the learning rate that is too large, the solution oscillates and in the worst case it might even diverge.

GRADIENT DESCENT

gd(eta = 0.05, f_grad)



gd(eta = 1.1, f_grad)



- Small learning rate
 - ▶ Slow Learning
 - ▶ Definitely converge
- Large learning rate
 - ▶ Oscillations
 - ▶ Worst case it might diverge

EXAMPLE

Error surface is given by $E(x, y, z) = 3x^2 + 2y^2 + 4z^2 + 6$. Assume gradient descent is used to find the minimum of this error surface. What is the optimal learning rate that leads to fastest convergence to the global minimum?

$$E(x, y, z) = 3x^2 + 2y^2 + 4z^2 + 6$$

$$\eta_x = 1/6$$

$$\eta_y = 1/4$$

$$\eta_z = 1/8$$

Optimal learning rate for convergence $\eta_{opt} = \min[\eta_x, \eta_y, \eta_z] = 1/8$

Largest learning rate for convergence $= \min[2\eta_x, 2\eta_y, 2\eta_z] = 0.33$

Smallest Learning rate for divergence $> 2\eta_{opt} = 2 * 1/8 = 0.25$

STOCHASTIC GRADIENT DESCENT

- In deep learning, the objective function is the average of the loss functions for each example in the training dataset.
- Given a training dataset of m examples, let $f_i(\theta)$ is the loss function with respect to the training example of index i , where θ is the parameter vector.
- Computational cost of each iteration is $O(1)$.

STOCHASTIC GRADIENT DESCENT ALGORITHM

Algorithm 2: STOCHASTIC GRADIENT DESCENT ALGORITHM

Data: Learning Rate η

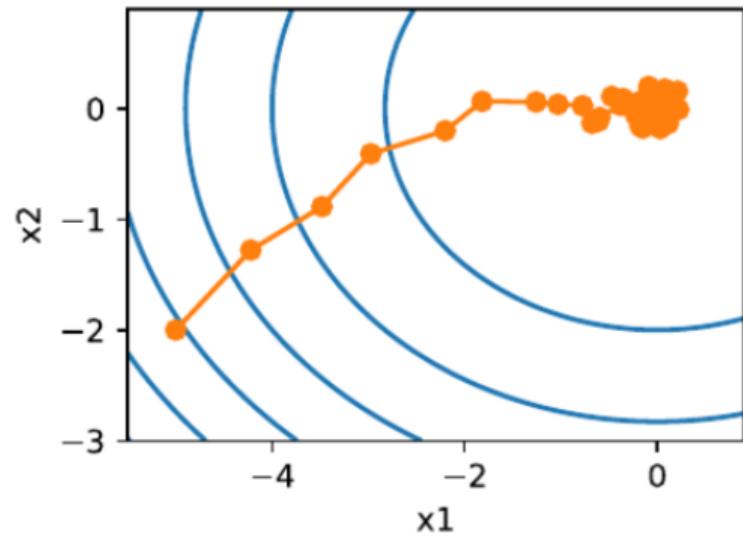
Data: Initial Parameters θ

```
1 while stopping criterion not met do
2   for each example  $i$  do
3     Perform Forward prop on  $X^{\{i\}}$  to compute  $\hat{y}$  and cost  $J(\theta)$ 
4     Perform Backprop on  $(X^{\{i\}}, Y^{\{i\}})$  to compute gradient  $\mathbf{g}$ 
5     Update parameters as  $\theta \leftarrow \theta - \eta \mathbf{g}$ 
```

STOCHASTIC GRADIENT DESCENT

- Trajectory of the variables in the stochastic gradient descent is much more noisy. This is due to the stochastic nature of the gradient. Even near the minimum, uncertainty is injected by the instantaneous gradients.

```
def sgd(x1, x2, s1, s2, f_grad):  
    g1, g2 = f_grad(x1, x2)  
    # Simulate noisy gradient  
    g1 += np.random.normal(0.0, 1, (1,))  
    g2 += np.random.normal(0.0, 1, (1,))  
    eta_t = eta * lr()  
    return (x1-eta_t*g1, x2-eta_t*g2, 0, 0)
```



DYNAMIC LEARNING RATE

- Replace η with a time-dependent learning rate $\eta(t)$ adds to the complexity of controlling convergence of an optimization algorithm.
- A few basic strategies that adjust η over time.
 - Piecewise constant
 - decrease the learning rate, e.g., whenever progress in optimization stalls.
 - This is a common strategy for training deep networks

$$\eta(t) = \eta_i \quad \text{if } t_i \leq t \leq t_{i+1}$$

- Exponential decay
 - Leads to premature stopping before the algorithm has converged.

$$\eta(t) = \eta_0 e^{-\lambda t}$$

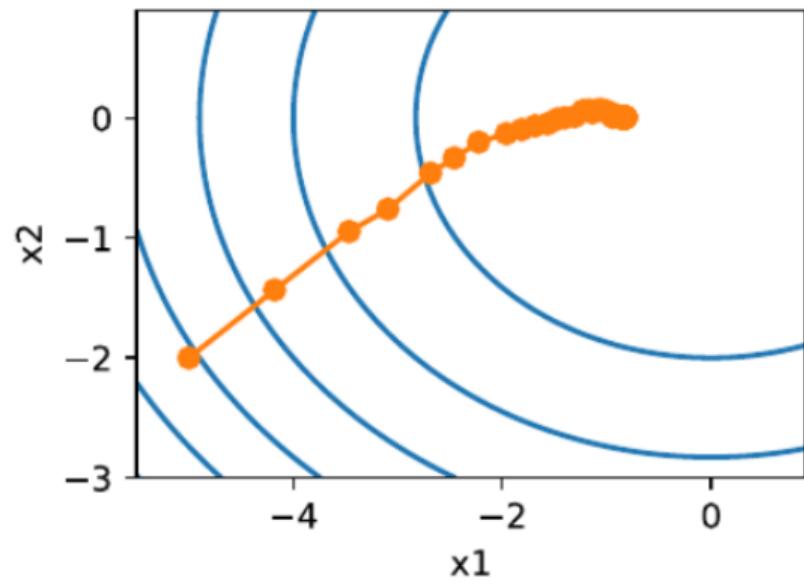
- Polynomial decay

$$\eta(t) = \eta_0 (\beta t + 1)^{-\alpha}$$

EXPONENTIAL DECAY

- Variance in the parameters is significantly reduced.
- The algorithm fails to converge at all.

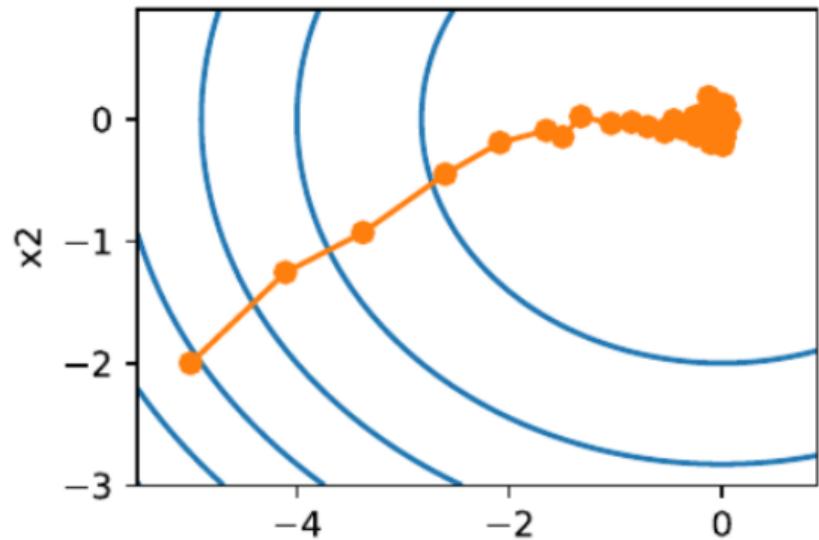
```
def exponential_lr():
    global t
    t += 1
    return math.exp(-0.1 * t)
```



POLYNOMIAL DECAY

- Learning rate decays with the inverse square root of the number of steps.
- Convergence gets better after only 50 steps.

```
def polynomial_lr():
    global t
    t += 1
    return (1 + 0.1 * t) ** (-0.5)
```



GAP

- Gradient descent
 - ▶ Uses the full dataset to compute gradients and to update parameters, one pass at a time.
 - ▶ Gradient Descent is not particularly data efficient whenever data is very similar.
- Stochastic Gradient descent
 - ▶ Processes one observation at a time to make progress.
 - ▶ Stochastic Gradient Descent is not particularly computationally efficient since CPUs and GPUs cannot exploit the full power of vectorization.
 - ▶ For noisy gradients, choice of the learning rate is critical.
 - ▶ If we decrease it too rapidly, convergence stalls.
 - ▶ If we are too lenient, we fail to converge to a good enough solution since noise keeps on driving us away from optimality.
- Minibatch SGD
 - ▶ Accelerate computation, or better or computational efficiency.

MINIBATCH STOCHASTIC GRADIENT DESCENT

- Suppose we have 4 million examples in the training set.

$$X = [X^{(1)}, X^{(2)}, \dots, X^{(m)}]$$

$$Y = [Y^{(1)}, Y^{(2)}, \dots, Y^{(m)}]$$

- The gradient descent algorithm processes the entire dataset of $m = 4$ million examples before proceeding to the next step. This is cumbersome and time consuming.
- Hence split the training dataset.

MINIBATCH STOCHASTIC GRADIENT DESCENT

- Let the mini-batch size be 1000.
- Then we have 4000 mini-batches, each having 1000 examples.

$$\begin{aligned}
 X &= [X^{(1)}, X^{(2)}, \dots, X^{(m)}] \\
 &= [X^{(1)}, X^{(2)}, \dots, X^{(1000)}, X^{(1001)}, \dots, X^{(2000)}, \dots, X^{(m)}] \\
 &= X^{\{1\}}, X^{\{2\}}, \dots, X^{\{m/b\}} \\
 Y &= [Y^{(1)}, Y^{(2)}, \dots, Y^{(m)}] \\
 &= [Y^{(1)}, Y^{(2)}, \dots, Y^{(1000)}, Y^{(1001)}, \dots, Y^{(2000)}, \dots, Y^{(m)}] \\
 &= Y^{\{1\}}, Y^{\{2\}}, \dots, Y^{\{m/b\}}
 \end{aligned}$$

- The minibatch t is denoted as $X^{\{t\}}, Y^{\{t\}}$
- The Gradient descent algorithm will be repeated for t batches.

MINIBATCH STOCHASTIC GRADIENT DESCENT ALGORITHM

- In each iteration, we first randomly sample a minibatch B consisting of a fixed number of training examples.
- We then compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
- Finally, we multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$\theta \leftarrow \frac{\eta}{|B|} \sum_{i \in B} \partial_{\theta} \text{loss}^{(t)}(\theta)$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and η denotes the learning rate.

MINIBATCH STOCHASTIC GRADIENT DESCENT ALGORITHM

- Gradients at time t is calculated as

$$g_{t,t-1} = \partial_{\theta} f \left(\frac{1}{|B|} \sum_{i \in B_t} f(x_i, \theta_{t-1}) \right) = \frac{1}{|B|} \sum_{i \in B_t} h_{i,t-1}$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and η denotes the learning rate.
- $h_{i,t-1} = \partial_{\theta} f(x_i, \theta_{t-1})$ is the stochastic gradient descent for sample i using the weights updated at time $t - 1$.

MINIBATCH STOCHASTIC GRADIENT DESCENT ALGORITHM

Algorithm 3: MINIBATCH STOCHASTIC GRADIENT DESCENT ALGORITHM

Data: Learning Rate η

Data: Initial Parameters θ

```
1 while stopping criterion not met do
2   for t in range (1, m/batch_size) do
3     Perform Forward prop on  $X^{\{t\}}$  to compute  $\hat{y}$  and cost  $J(\theta)$ 
4     Perform Backprop on  $(X^{\{t\}}, Y^{\{t\}})$  to compute gradient  $\mathbf{g}$ 
5     Update parameters as  $\theta \leftarrow \theta - \eta \mathbf{g}$ 
```

MINIBATCH SIZE

- Typical mini batch size is taken as some power of 2; say 32, 64, 128, 256.
- Better utilization of multicore architecture.
- Amount of memory scales with batch size.

LEAKY AVERAGE IN MINIBATCH SGD

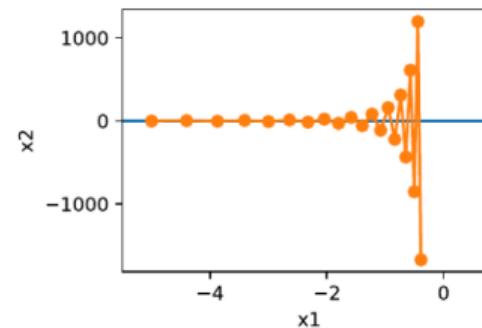
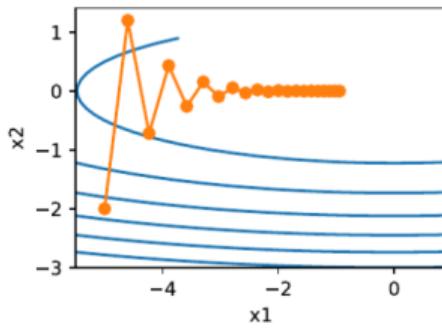
- Replace the gradient computation by a "leaky average" for better variance reduction.

$$\beta \in (0, 1)$$

- This effectively replaces the instantaneous gradient by one that's been averaged over multiple past gradients.
- v is called momentum.
- Momentum accumulates past gradients.**
- Large β amounts to a long-range average and small β amounts to only a slight correction relative to a gradient method.
- The new gradient replacement no longer points into the direction of steepest descent on a particular instance any longer but rather in the direction of a weighted average of past gradients.

MOMENTUM METHOD EXAMPLE

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- f has its minimum at $(0, 0)$. This function is very flat in x_1 direction.

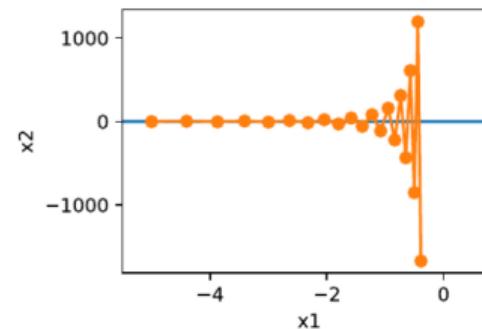
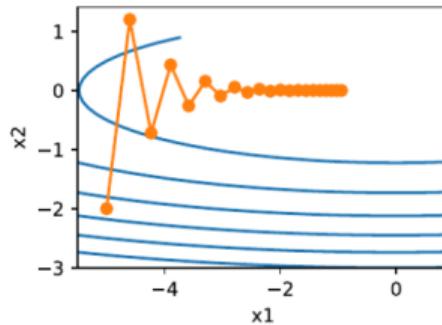


- For $\eta = 0.4$ Without momentum
- The gradient in the x_2 direction oscillates than in the horizontal x_1 direction.

- For $\eta = 0.6$ Without momentum
- Convergence in the x_1 direction improves but the overall solution quality is diverging.

MOMENTUM METHOD EXAMPLE

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- Apply momentum for $\eta = 0.6$



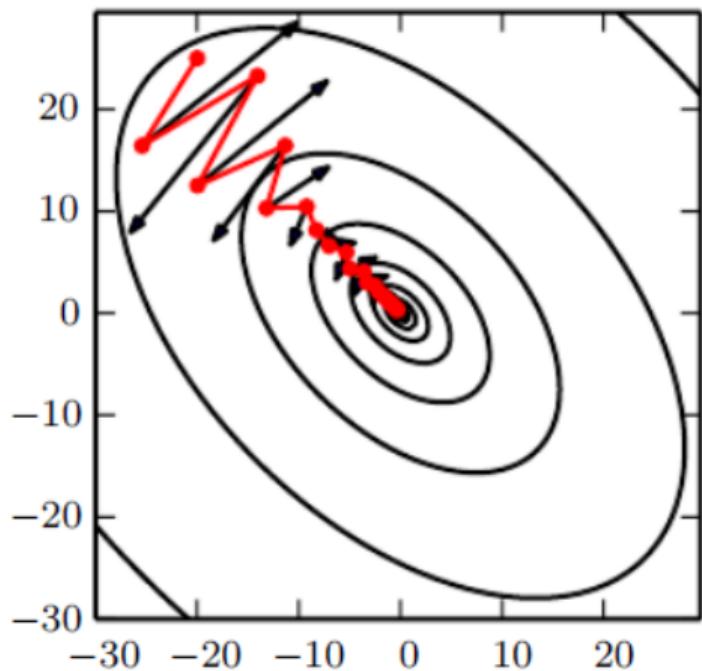
- For $\beta = 0.5$
- Converges well. Lesser oscillations. Larger steps in x_1 direction.

- For $\beta = 0.25$
- Reduced convergence. More oscillations. Larger magnitude of oscillations.

MOMENTUM

- Compute an exponentially weighted average of the gradients and use it to update the parameters.
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- The momentum introduces a velocity v that denotes the direction and speed at which the parameters change.
- The velocity is set to an exponentially decaying average of the negative gradient.
- The Momentum parameter $\beta \in [0, 1)$ is a hyperparameter and typically takes a value of 0.9.

SGD WITH MOMENTUM



SGD WITH MOMENTUM ALGORITHM

Algorithm 4: STOCHASTIC GRADIENT DESCENT WITH MOMENTUM

Data: Learning Rate η , Momentum Parameter β

Data: Initial Parameters θ , Initial velocities v

```
1 while stopping criterion not met do
2   for t in range (1, m/batch_size) do
3     Perform Forward prop on  $X^{\{t\}}$  to compute  $\hat{y}$  and cost  $J(\theta)$ 
4     Perform Backprop on  $(X^{\{t\}}, Y^{\{t\}})$  to compute gradient  $\mathbf{g}$ 
5     Update velocities as  $v = \beta v + (1 - \beta)\mathbf{g}$ 
6     Update parameters as  $\theta \leftarrow \theta - \eta v$ 
```

MOMENTUM METHOD: SUMMARY

- Momentum replaces gradients with a leaky average over past gradients. This accelerates convergence significantly.
- Momentum prevents stalling of the optimization process that is much more likely to occur for stochastic gradient descent.
- The effective number of gradients is given by $1/(1 - \beta)$ due to exponentiated down weighting of past data.
- Implementation is quite straightforward but it requires us to store an additional state vector (momentum v).

NUMERICAL EXAMPLE

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2) . Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t) . Assume $\alpha = 1.5, \beta = 0.6, \eta = 0.3$. Compute the value of (w_1, w_2) at time $(t + 1)$ if momentum is used.

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6}$$

NUMERICAL EXAMPLE

$$\begin{aligned}
 \text{at } (t+1) \quad w_{1(t+1)} &= w_{1(t)} - \eta \frac{\partial E}{\partial w_1} + \beta \Delta w_1 \\
 &= 1.5 - 0.3 * \left[\frac{(1.5 - 3)}{2} - \frac{(2 - 4)}{6} \right] + 0.9 * (1.5 - 1) \\
 &= 2.075
 \end{aligned}$$

$$\begin{aligned}
 w_{2(t+1)} &= w_{2(t)} - \eta \frac{\partial E}{\partial w_2} + \beta \Delta w_1 \\
 &= 2 - 0.3 * \left[\frac{2(2 - 4)}{9} - \frac{(1.5 - 3)}{6} \right] + 0.9 * (2 - 1) \\
 &= 2.958
 \end{aligned}$$

ADAGRAD

- Used for features that occur infrequently (sparse features).
- Adagrad uses aggregate of the squares of previously observed gradients.

ADAGRAD ALGORITHM

- Variable s_t to accumulate past gradient variance.
- Operations are applied coordinate wise. $\sqrt{1/v}$ has entries $\sqrt{1/v_i}$ and $u \cdot v$ has entries $u_i v_i$.
- η is the learning rate and ϵ is an additive constant that ensures that we do not divide by 0.
- Initialize $s_0 = 0$.

$$g_t = \partial_w \text{loss}(y_t, f(x_t, w))$$

$$s_t = s_{t-1} g_t^2$$

$$w_t = w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \cdot g_t$$

ADAGRAD: SUMMARY

- Adagrad decreases the learning rate dynamically on a per-coordinate basis.
- It uses the magnitude of the gradient as a means of adjusting how quickly progress is achieved - coordinates with large gradients are compensated with a smaller learning rate.
- If the optimization problem has a rather uneven structure Adagrad can help mitigate the distortion.
- Adagrad is particularly effective for sparse features where the learning rate needs to decrease more slowly for infrequently occurring terms.
- On deep learning problems Adagrad can sometimes be too aggressive in reducing learning rates.

RMSPROP (ROOT MEAN SQUARE PROP)

- Adapts the learning rates of all model parameters by scaling the gradients into an exponentially weighted moving average.
- Performs better with non-convex setting.
- Adagrad use learning rate that decreases at a predefined schedule of effectively $O(t^{-1/2})$.
- RMSProp algorithm decouples rate scheduling from coordinate-adaptive learning rates. This is essential for non-convex optimization.

RMSProp ALGORITHM

- Use leaky average to accumulate past gradient variance.
- Parameter $\gamma > 0$.
- The constant ϵ is set to 10^{-6} to ensure that we do not suffer from division by zero or overly large step sizes.

$$s_t = \gamma s_{t-1} + (1 - \gamma) g_t^2$$
$$w_t = w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t$$

RMSPROP ALGORITHM

Algorithm 5: RMSPROP

Data: Learning Rate η , Decay Rate γ

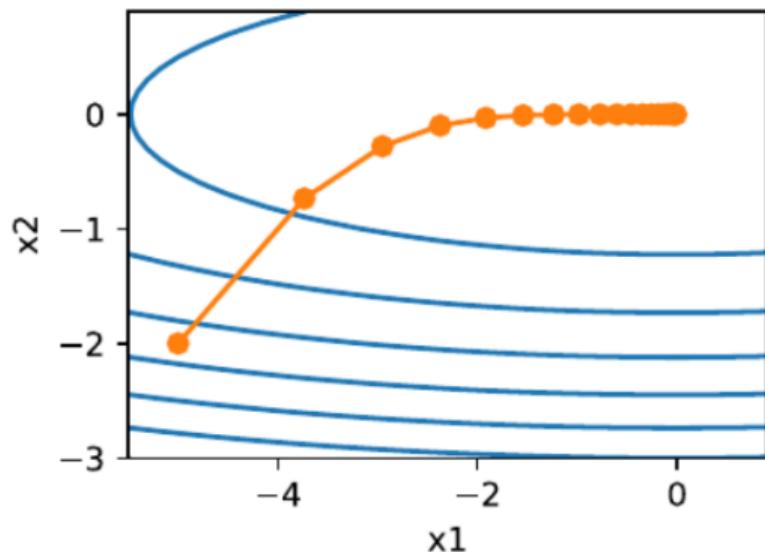
Data: Small constant $\epsilon = 10^{-7}$ to stabilize division

Data: Initial Parameters θ

- 1 Initialize accumulated gradient $s = 0$
 - 2 **while** stopping criterion not met **do**
 - 3 **for** t in range $(1, m/\text{batch_size})$ **do**
 - 4 Perform Forward prop on $X^{\{t\}}$ to compute \hat{y} and cost $J(\theta)$
 - 5 Perform Backprop on $(X^{\{t\}}, Y^{\{t\}})$ to compute gradient \mathbf{g}
 - 6 Compute accumulated squared gradient $s = \gamma s + (1 - \gamma)\mathbf{g}^2$
 - 7 Update parameters as $\theta \leftarrow \theta - \frac{\eta}{\sqrt{s+\epsilon}} \odot \mathbf{g}$
-

RMSProp EXAMPLE

```
def rmsprop_2d(x1, x2, s1, s2):
    g1, g2, eps = 0.2*x1, 4*x2, 1e-6
    s1 = gamma*s1 +(1-gamma)*g1**2
    s2 = gamma*s2 +(1-gamma)*g2**2
    x1 == eta/math.sqrt(s1 + eps)*g1
    x2 == eta/math.sqrt(s2 + eps)*g2
    return x1, x2, s1, s2
```



RMSProp: SUMMARY

- RMSProp is very similar to Adagrad as both use the square of the gradient to scale coefficients.
- RMSProp shares with momentum the leaky averaging. However, RMSProp uses the technique to adjust the coefficient-wise preconditioner.
- The learning rate needs to be scheduled by the experimenter in practice.
- The coefficient γ determines how long the history is when adjusting the per-coordinate scale.

REVIEW OF ALGORITHMS LEARNED SO FAR

① Stochastic gradient descent

- ▶ more effective than Gradient Descent when solving optimization problems, e.g., due to its inherent resilience to redundant data.

② Minibatch Stochastic gradient descent

- ▶ affords significant additional efficiency arising from vectorization, using larger sets of observations in one minibatch. This is the key to efficient multi-machine, multi-GPU and overall parallel processing.

③ Momentum

- ▶ added a mechanism for aggregating a history of past gradients to accelerate convergence.

REVIEW OF ALGORITHMS LEARNED SO FAR

① Adagrad

- ▶ used per-coordinate scaling to allow for a computationally efficient preconditioner.

② RMSProp

- ▶ decoupled per-coordinate scaling from a learning rate adjustment.

ADAM (ADAPTIVE MOMENTS)

- Adam combines all these techniques into one efficient learning algorithm.
- Very effective and robust algorithm
- Adam can diverge due to poor variance control. (disadvantage)
- Adam uses exponential weighted moving averages (also known as leaky averaging) to obtain an estimate of both the momentum and also the second moment of the gradient.

ADAM ALGORITHM

- State variables

$$v_t \leftarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t$$

$$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2) g_t^2$$

- β_1 and β_2 are nonnegative weighting parameters. Common choices for them are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. That is, the variance estimate moves much more slowly than the momentum term.
- Initialize $v_0 = s_0 = 0$.

ADAM ALGORITHM

- Normalize the state variables

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$

- Rescale the gradient

$$\hat{g}_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

- Compute updates

$$w_t \leftarrow w_{t-1} - \frac{\eta \hat{v}}{\sqrt{\hat{s} + \epsilon}}$$

ADAM ALGORITHM

Algorithm 6: ADAM

Data: Learning Rate $\eta = 0.001$, Momentum parameter $\beta_1 = 0.9$, Decay Rate $\beta_2 = 0.999$, Small constant $\epsilon = 10^{-7}$ to stabilize division

Data: Initial Parameters θ ; Initialize first and second moments $v = 0, s = 0$

- 1 **while** stopping criterion not met **do**
- 2 **for** t in range $(1, m/batch_size)$ **do**
- 3 Perform Forward prop on $X^{\{t\}}$ to compute \hat{y} and cost $J(\theta)$
- 4 Perform Backprop on $(X^{\{t\}}, Y^{\{t\}})$ to compute gradient \mathbf{g}
- 5 Update first moment as $v \leftarrow \beta_1 v + (1 - \beta_1)\mathbf{g}$
- 6 Update second moment as $s = \beta_2 s + (1 - \beta_2)\mathbf{g}^2$
- 7 Perform bias correction as $\hat{v} = \frac{v}{1-\beta_1^t}$ and $\hat{s} = \frac{s}{1-\beta_2^t}$
- 8 Update parameters as $\theta \leftarrow \theta - \frac{\eta \hat{v}}{\sqrt{\hat{s}} + \epsilon}$

ADAM: SUMMARY

- Adam combines features of many optimization algorithms into a fairly robust update rule.
- Adam uses bias correction to adjust for a slow startup when estimating momentum and a second moment.
- For gradients with significant variance we may encounter issues with convergence. They can be amended by using larger minibatches or by switching to an improved estimate for state variables.
- Yogi algorithm offers such an alternative.

NUMERICAL EXAMPLE

The training data $(x_1, y_1) = (3.5, 0.5)$ for a single Sigmoid neuron and initial values of $w = -2.0, b = -2.0, \eta = 0.10, \beta_1 = 0.90, \beta_2 = 0.99, \epsilon = 1e-8, s_w = 0$ and $s_B = 0$ are provided. Showing all the calculations, find out the values of w, b, s_w, s_B, r_w, r_B after one iteration of Adam. Use BCE as loss function. Apply bias-correction.

NUMERICAL EXAMPLE

- Forward Pass and Calculate Loss

$$\hat{y} = \sigma(wx + b) = \sigma(-2.0 \times 3.5 - 2.0) \approx 0.0180$$

$$L = -[y \times \log(\hat{y}) + (1 - y) \times \log(1 - \hat{y})]$$

$$L = -[0.5 \times \log(0.0180) + (1 - 0.5) \times \log(1 - 0.0180)] \approx 4.0076$$

- Calculate Gradients

$$\nabla_w L = (\hat{y} - y) \times x = (0.0180 - 0.5) \times 3.5 \approx -0.493$$

$$\nabla_b L = \hat{y} - y = 0.0180 - 0.5 \approx -0.482$$

- Update First Moments

$$s_w = \beta_1 \times s_w + (1 - \beta_1) \times \nabla_w L = 0.90 \times 0 - 0.10 \times (-0.493) \approx 0.04437$$

$$s_b = \beta_1 \times s_b + (1 - \beta_1) \times \nabla_b L = 0.90 \times 0 - 0.10 \times (-0.482) \approx 0.04382$$

NUMERICAL EXAMPLE

- Update Second Moments

$$\begin{aligned}
 r_W &= \beta_2 \times r_W + (1 - \beta_2) \times (\nabla_w L)^2 \\
 &= 0.999 \times 0 - 0.001 \times (-0.493)^2 \approx 0.0001216 \\
 r_B &= \beta_2 \times r_B + (1 - \beta_2) \times (\nabla_b L)^2 \\
 &= 0.999 \times 0 - 0.001 \times (-0.482)^2 \approx 0.0001168
 \end{aligned}$$

- Corrected First Moments

$$\begin{aligned}
 \hat{s}_W &= \frac{s_W}{1 - \beta_1^1} \approx \frac{0.04437}{1 - 0.90^1} \approx 0.4437 \\
 \hat{s}_B &= \frac{s_B}{1 - \beta_1^1} \approx \frac{0.04382}{1 - 0.90^1} \approx 0.4382
 \end{aligned}$$

NUMERICAL EXAMPLE

- Corrected Second Moments

$$\hat{r}_W = \frac{r_W}{1 - \beta_2^1} \approx \frac{0.0001216}{1 - 0.999^1} \approx 0.0608$$

$$\hat{r}_B = \frac{r_B}{1 - \beta_2^1} \approx \frac{0.0001168}{1 - 0.999^1} \approx 0.0584$$

- Update Weights and Biases

$$w = w - \frac{\eta}{\sqrt{\hat{r}_W} + \epsilon} \times \hat{s}_W = -2.0 - \frac{0.10}{\sqrt{0.0608} + 1e-8} \times 0.4437 \approx -2.081$$

$$b = b - \frac{\eta}{\sqrt{\hat{r}_B} + \epsilon} \times \hat{s}_B = -2.0 - \frac{0.10}{\sqrt{0.0584} + 1e-8} \times 0.4382 \approx -2.079$$

References

- ① Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola
https://d2l.ai/chapter_introduction/index.html
- ② Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>

Thank You!



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

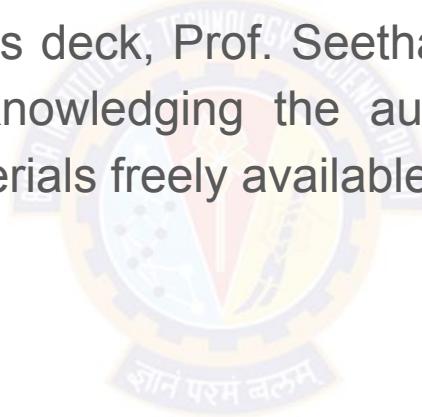
Deep Learning

DSE Module 4

Seetha Parameswaran

BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors who made
their course materials freely available online.



Regularization Techniques



What we Learn....

4.1 Model Selection

4.2 Underfitting, and Overfitting

4.3 L1 and L2 Regularization

4.4 Dropout

4.5 Challenges - Vanishing and Exploding Gradients, Covariance shift

4.6 Parameter Initialization

4.7 Batch Normalization



Generalization in DNN



Generalization

- Goal is to discover patterns that generalize.
 - The goal is to discover patterns that capture regularities in the underlying population from which our training set was drawn.
 - Models are trained on a sample of data.
 - When working with finite samples, we run the risk that we might discover apparent associations that turn out not to hold up when we collect more data or on newer samples.
- The trained model should predict for newer or unseen data. This problem is called generalization.

Training Error and Generalization Error

- **Training error** is the error of our model as calculated on the training dataset.
 - Obtained while training the model.
- **Generalization error** is the expectation of our model's error, if an infinite stream of additional data examples drawn from the same underlying data distribution as the original sample were applied on the model.
 - Cannot be computed, but estimated.
 - Estimate the generalization error by applying the model to an independent test set, constituted of a random selection of data examples that were withheld from the training set.

Model Complexity

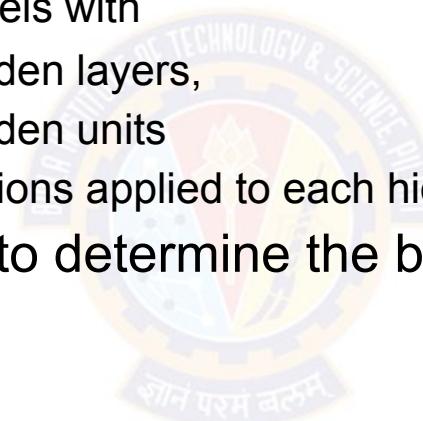
- Simple models and abundant data
 - Expect the generalization error to resemble the training error.
- More complex models and fewer examples
 - Expect the training error to go down but the generalization gap to grow.
- Model complexity
 - A model with more parameters might be considered more complex.
 - A model whose parameters can take a wider range of values might be more complex.
 - A neural network model that takes more training iterations are more complex, and one subject to early stopping (fewer training iterations) are less complex.
-

Factors that influence the generalizability of a model

1. The number of tunable parameters.
 - When the number of tunable parameters, called the degrees of freedom, is large, models tend to be more susceptible to overfitting.
2. The values taken by the parameters.
 - When weights can take a wider range of values, models can be more susceptible to overfitting.
3. The number of training examples.
 - It is trivially easy to overfit a dataset containing only one or two examples even if your model is simple. But overfitting a dataset with millions of examples requires an extremely flexible model.
- 4.

Model Selection

- Model selection is the process of selecting the final model after evaluating several candidate models.
- With MLPs, compare models with
 - different numbers of hidden layers,
 - different numbers of hidden units
 - different activation functions applied to each hidden layer.
- Use Validation dataset to determine the best among our candidate models.

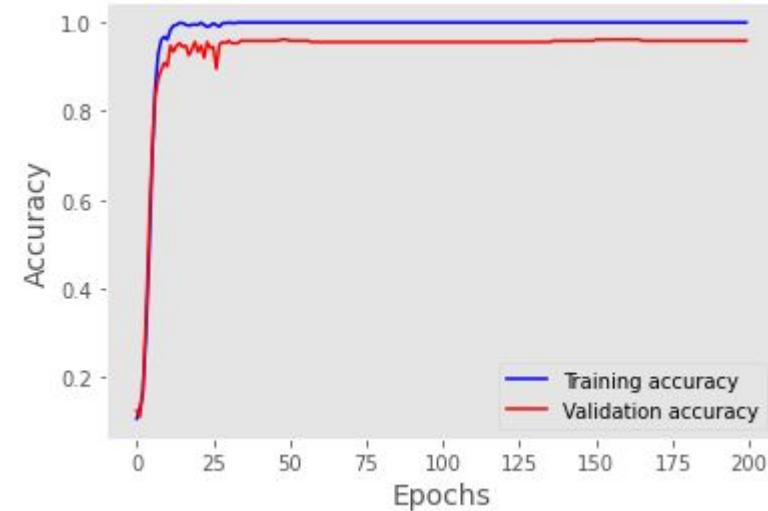
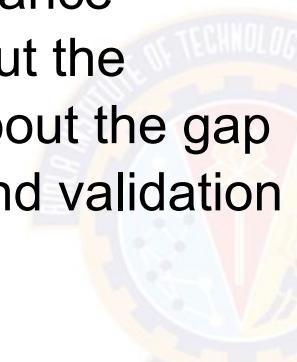


Validation dataset

- Never rely on the test data for model selection.
 - Risk of overfit the test data
- Do not rely solely on the training data for model selection
 - We cannot estimate the generalization error on the very data that we use to train the model.
- Split the data three ways, incorporating a validation dataset (or validation set) in addition to the training and test datasets.
- In deep learning, with millions of data available, the split is generally
 - Training = 98-99 % of the original dataset
 - Validation = 1-2 % of training dataset
 - Testing = 1-2 % of the original dataset

Just Right Model

- High Training accuracy
- High Validation accuracy
- Low Bias and Low Variance
- Usually care more about the validation error than about the gap between the training and validation errors.



```
844/844 [=====] - 3s 3ms/step - loss: 0.0848 - accuracy: 0.9750 - val_loss: 0.0714 - val_accuracy: 0.9790
Epoch 2/25
844/844 [=====] - 2s 2ms/step - loss: 0.0821 - accuracy: 0.9761 - val_loss: 0.0730 - val_accuracy: 0.9802
Epoch 3/25
844/844 [=====] - 2s 2ms/step - loss: 0.0793 - accuracy: 0.9769 - val_loss: 0.0706 - val_accuracy: 0.9798
Epoch 4/25
844/844 [=====] - 2s 3ms/step - loss: 0.0769 - accuracy: 0.9771 - val_loss: 0.0753 - val_accuracy: 0.9795
```

Underfitting

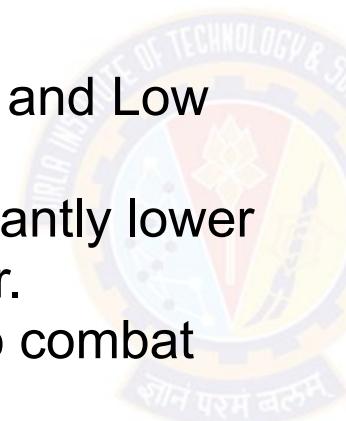
- Low Training accuracy and Low Validation accuracy.

```
Train on 2594 samples, validate on 495 samples
Epoch 1/200
2594/2594 [=====] - 6s 2ms/step - loss: 2.8738 - acc: 0.1191 - val_loss: 2.2041 - val_a
cc: 0.1232
Epoch 2/200
2594/2594 [=====] - 5s 2ms/step - loss: 2.1731 - acc: 0.1415 - val_loss: 2.1776 - val_a
cc: 0.1192
Epoch 3/200
2594/2594 [=====] - 5s 2ms/step - loss: 2.0862 - acc: 0.1974 - val_loss: 2.1261 - val_a
cc: 0.1596
Epoch 4/200
2594/2594 [=====] - 5s 2ms/step - loss: 1.9268 - acc: 0.2787 - val_loss: 2.0487 - val_a
cc: 0.2525
```

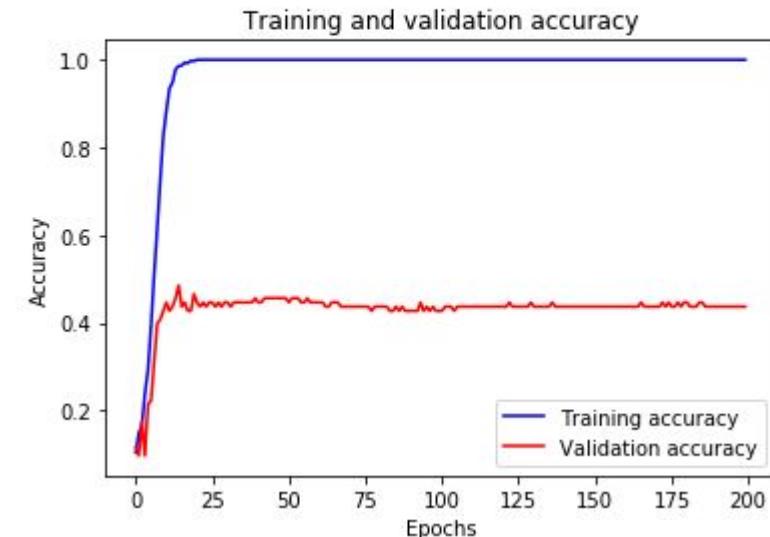
- Training error and validation error are both substantial but there is a little gap between them.
- The model is too simple (insufficiently expressive) to capture the pattern that we are trying to model.
- If generalization gap between our training and validation errors is small, a more complex model may be better.

Overfitting

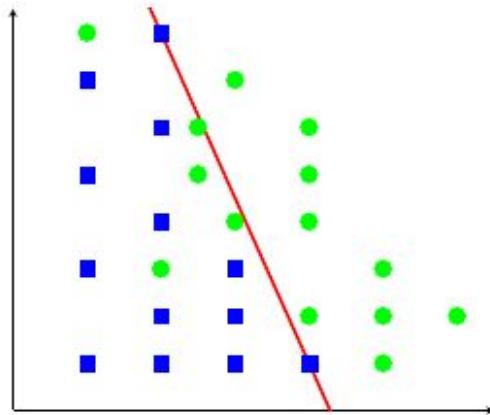
- The phenomenon of fitting the training data more closely than fit the underlying distribution is called **overfitting**.
- High Training accuracy and Low Validation accuracy
- Training error is significantly lower than the validation error.
- The techniques used to combat overfitting are called **regularization**.



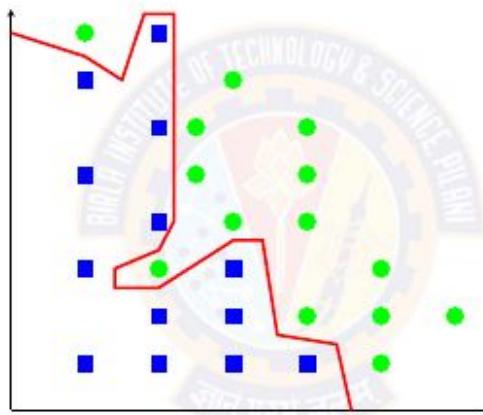
Training accuracy = 1.0
Validation accuracy = 0.43689320475152393
Testing accuracy = 0.5614035087719298



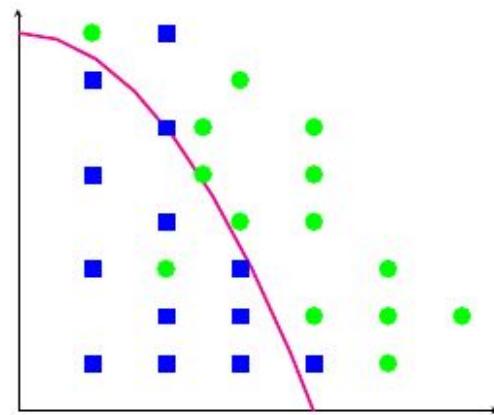
Underfitting or Overfitting?



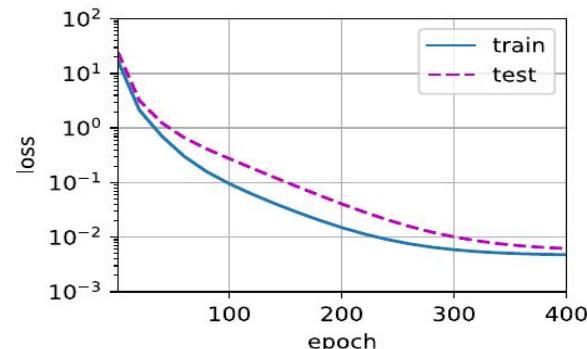
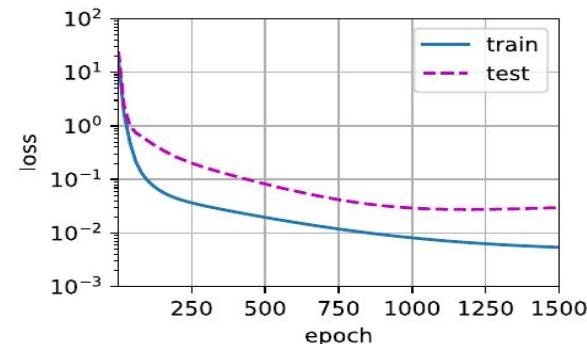
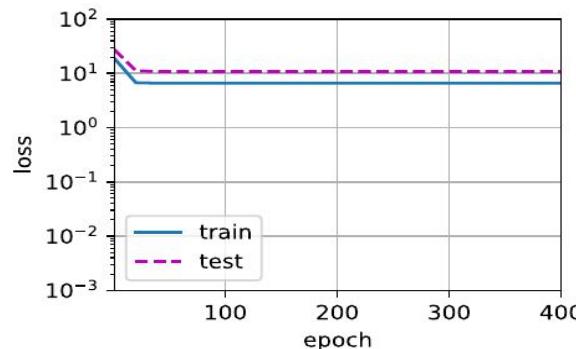
Simple Model
Underfitting



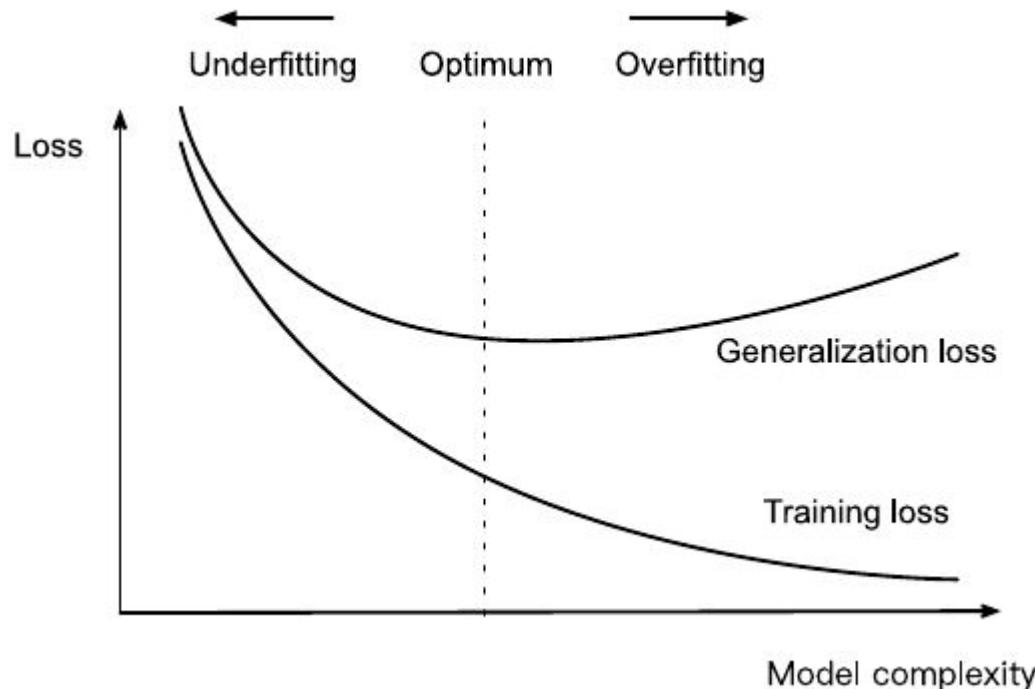
Complex Model
Overfitting



Just right model

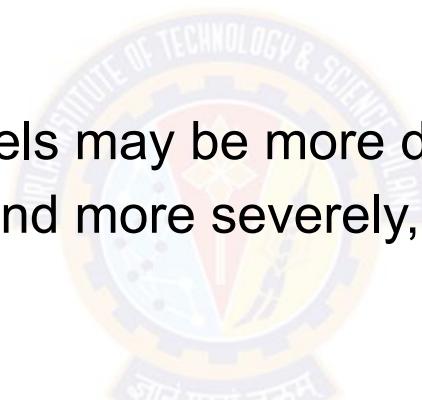


Polynomial degree and underfitting vs. overfitting



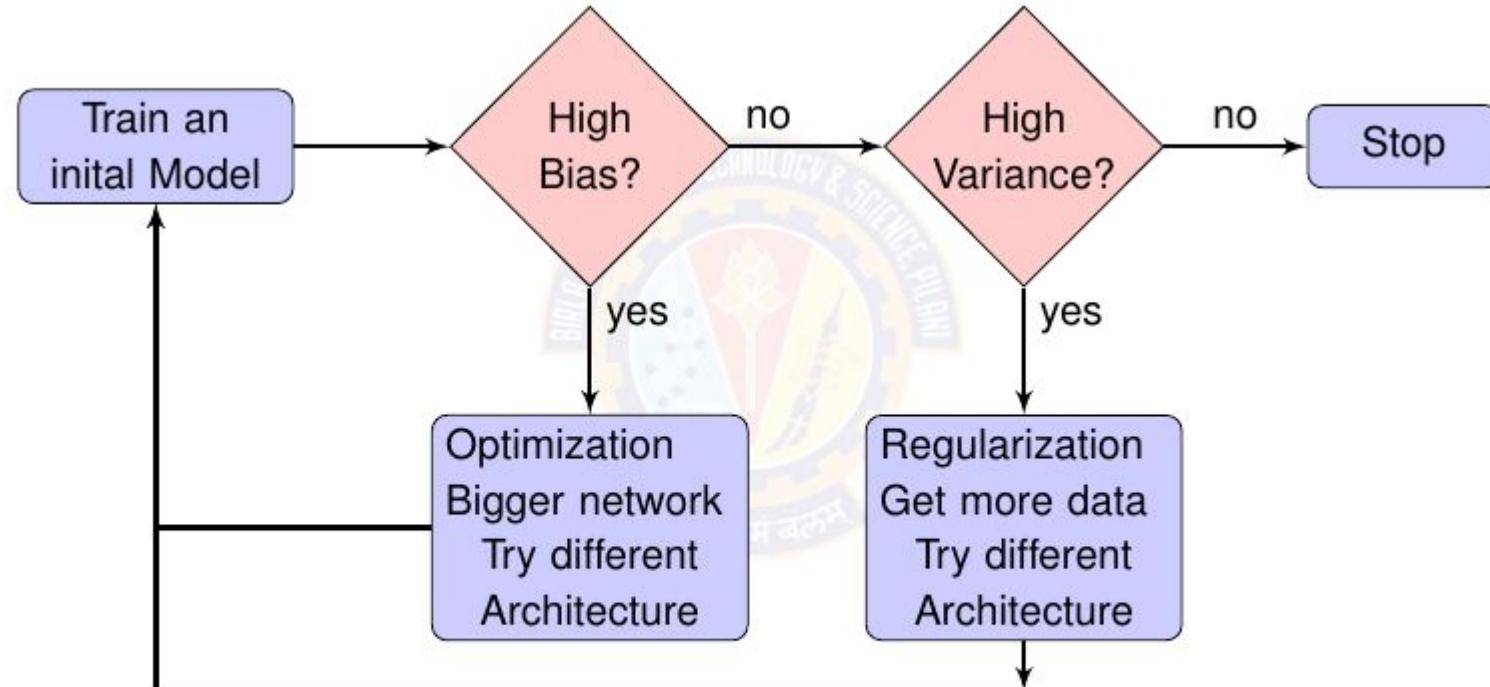
Model complexity and dataset size

- More data, fit a more complex model.
 - More data, the generalization error typically decreases.
-
- Less data, simpler models may be more difficult to beat.
 - Less data, more likely and more severely, models may over-fit.



The current success of deep learning owes to the current abundance of massive datasets due to Internet companies, cheap storage, connected devices, and the broad digitization of the economy.

Deep Learning Model Selection



Credit: Andrew Ng

Regularization



Regularization Techniques

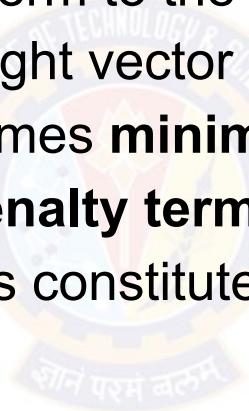
- Weight Decay (L2 regularization)
- Dropout
- Early Stopping



Weight Decay

L2 Regularization

- Measure the complexity of a linear function $f(x) = w^T x$ by some norm of its weight vector, e.g., $\|w\|^2$.
- Add the norm as a penalty term to the problem of minimizing the loss. This will ensure that the weight vector is small.
- The objective function becomes **minimizing the sum of the prediction loss and the penalty term.**
- L2-regularized linear models constitute the ridge regression algorithm.



L2 Regularization

- The trade off between standard loss and the additive penalty is given by regularization constant λ , a non-negative hyperparameter.

$$\text{Objective function} = \mathcal{L}(w, b) + \frac{\lambda}{2} \| w \|^2$$

$$\text{Weight updation} \quad w \leftarrow (1 - \eta\lambda)w - \eta g$$

- For $\lambda = 0$, we recover the original loss function.
- For $\lambda > 0$, we restrict the size of $\| w \|$.
- Smaller values of λ correspond to less constrained w , whereas larger values of λ constrain w more considerably.
- By squaring the L2 norm, we remove the square root, leaving the sum of squares of each component of the weight vector.
- The derivative of a quadratic function ensure that the 2 and 1/2 cancel out.

L1 Regularization

- L1-regularized linear regression is known as lasso regression.
- L1 penalties lead to models that concentrate weights on a small set of features by clearing the other weights to zero. This is called feature selection.

$$\text{Objective function} = \mathcal{L}(w, b) + \lambda \sum_i |w_i|$$

$$\text{Weight updation} \quad w \leftarrow (1 - \eta\lambda)w - \eta g$$

L2 Regularization

L1 Regularization

Sum of square of weights

Sum of absolute value of weights

Learn complex data patterns

Generates simple and interpretable models

Estimate mean of data

Estimate median of data

Not robust to outliers

Robust to outliers

Shrink coefficients equally

Shrink coefficients to zero

Non sparse solution

Sparse solution

One solution

Multiple solutions

No feature selection

Selects features

Useful for collinear features

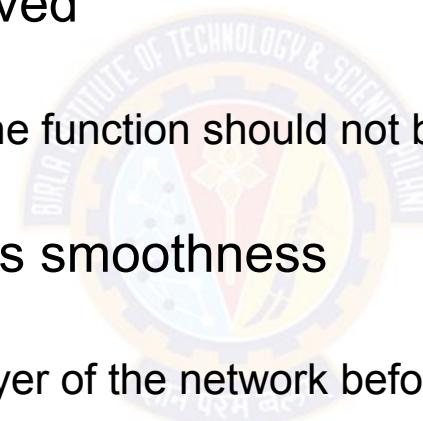
Useful for dimensionality reduction



Dropout

Smoothness

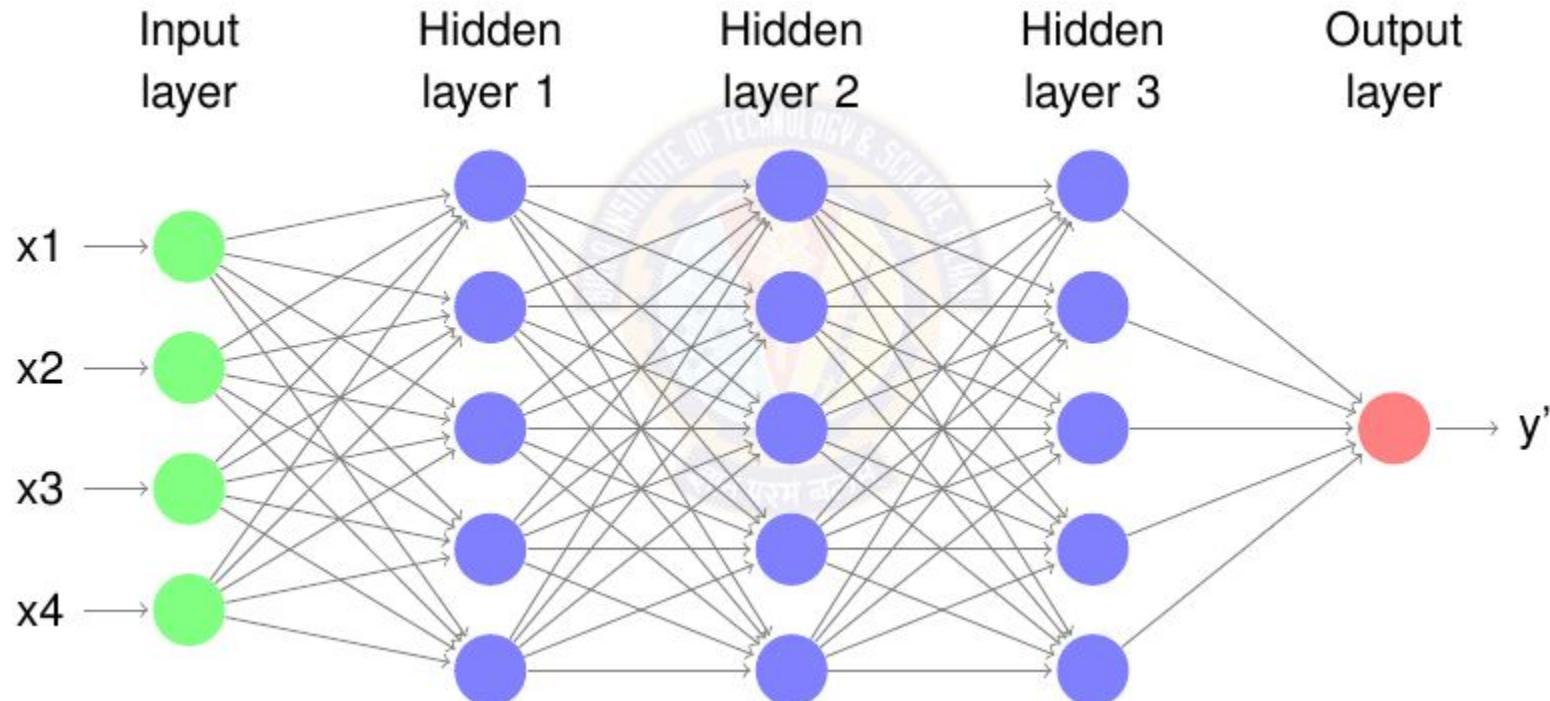
- Classical generalization theory suggests that to close the gap between train and test performance, aim for a simple model.
- Simplicity can be achieved
 - Using weight decay
 - Smoothness, i.e., that the function should not be sensitive to small changes to its inputs.
- Injecting noise enforces smoothness
 - training with input noise
 - inject noise into each layer of the network before calculating the subsequent layer during training.



Dropout

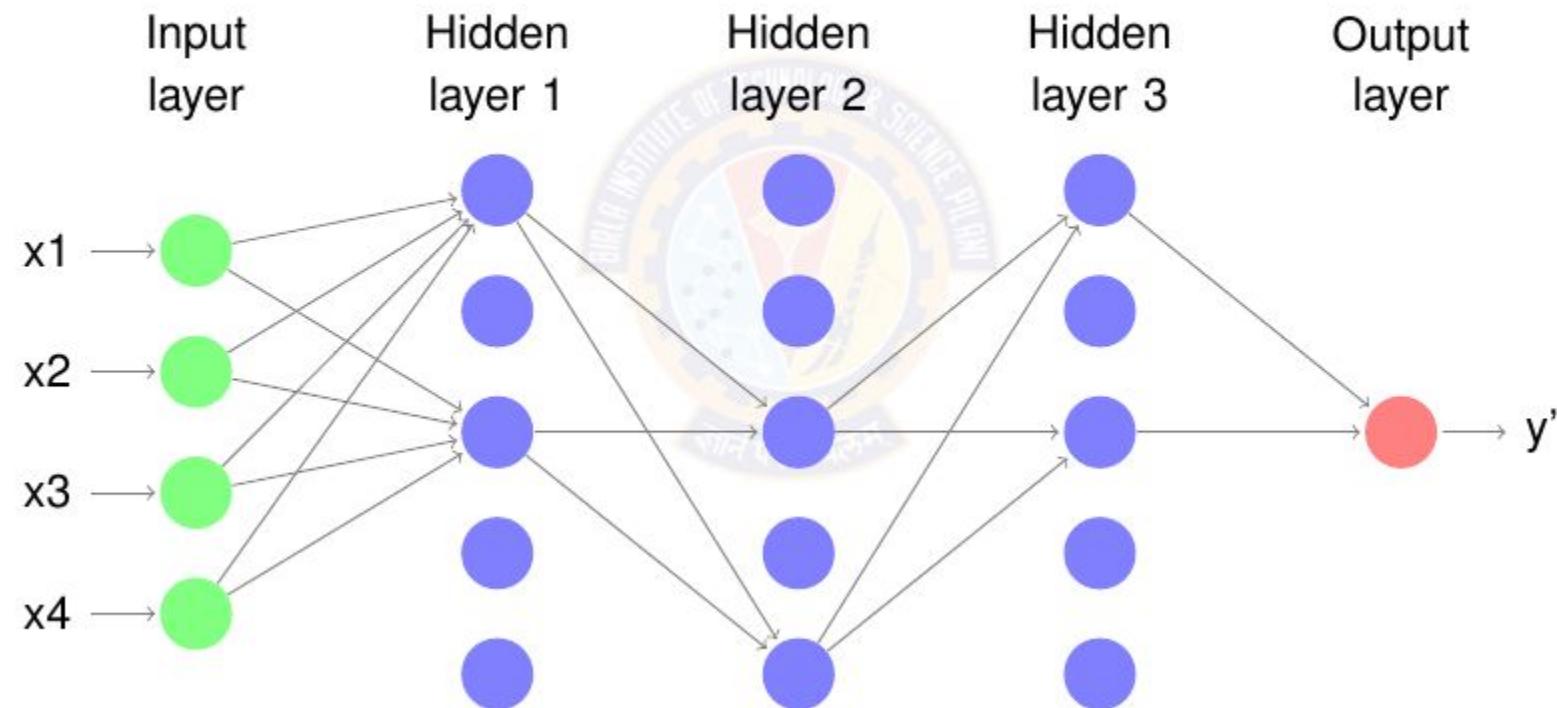
- **Dropout** involves injecting noise while computing each internal layer during forward propagation.
- It has become a **standard** technique for training neural networks.
- The method is called **dropout** because we literally drop out some neurons during training.
- Apply dropout to a hidden layer, zeroing out each hidden unit with probability p .
- The calculation of the outputs no longer depends on dropped out neurons and their respective gradient also vanishes when performing backpropagation. (in that iteration)
- Dropout is disabled at test time.

Without Dropout



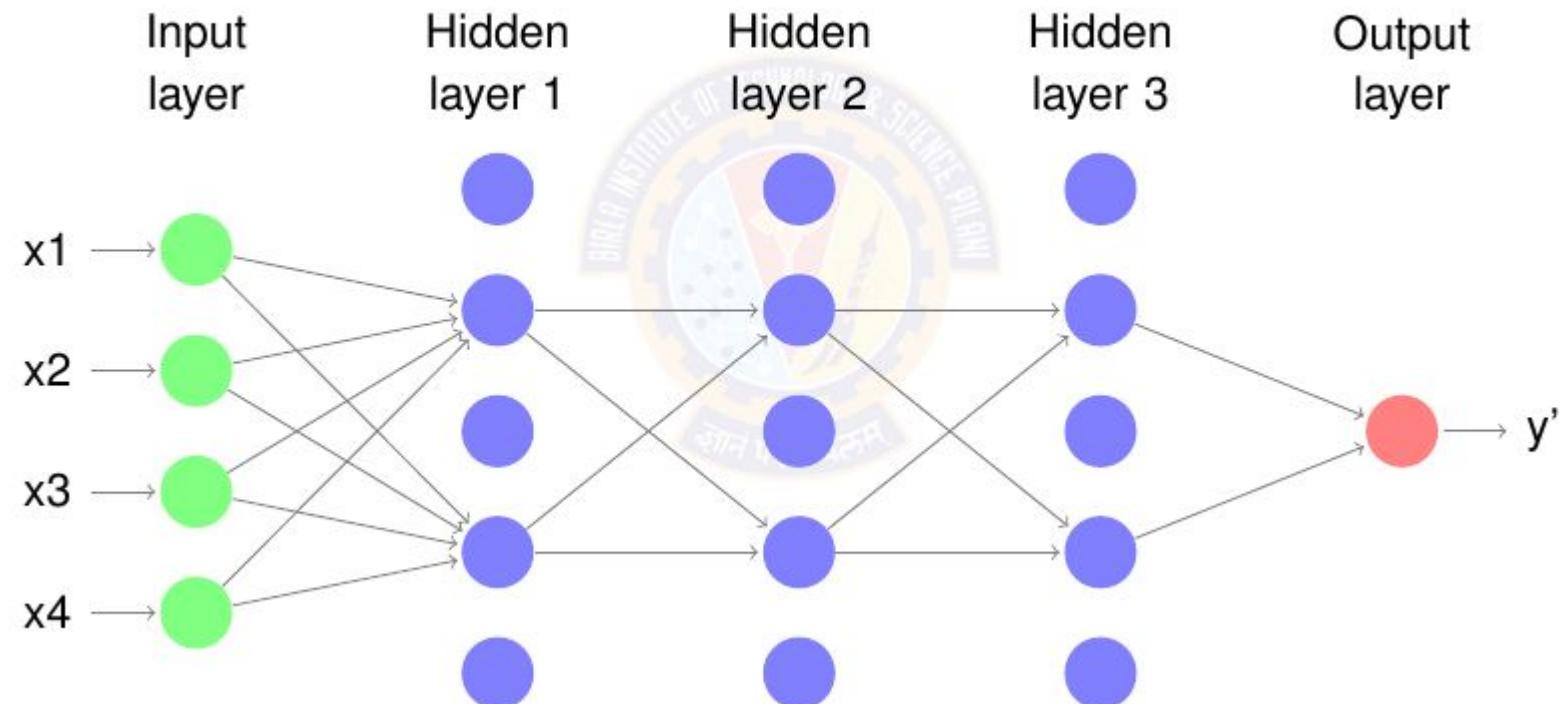
Dropout for first iteration

Keep probability = 0.5



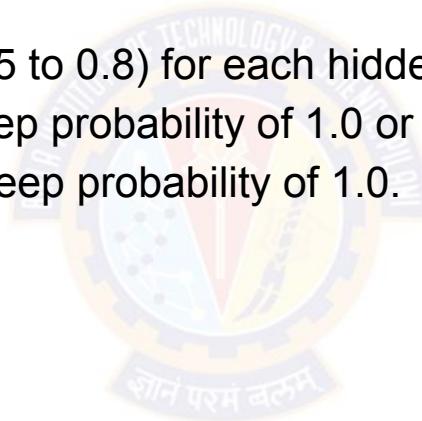
Dropout for second iteration

Keep probability = 0.5



Dropout

- Dropout gives a smaller neural network, giving the effect of regularization.
- In general,
 - Vary keep probability (0.5 to 0.8) for each hidden layer.
 - The input layer has a keep probability of 1.0 or 0.9.
 - The output layer has a keep probability of 1.0.
-

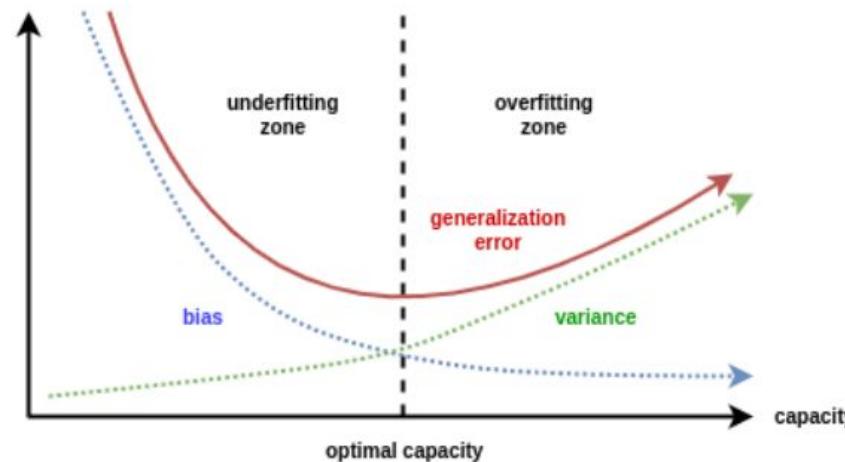


Early Stopping



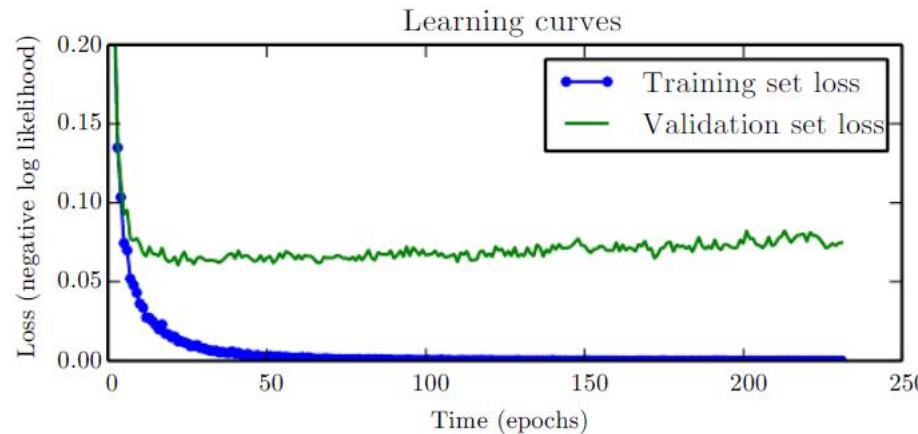
Before Early stopping

- When training large models, training error decreases steadily over time, but validation set error begins to rise again.
- Training objective decreases consistently over time.
- Validation set average loss begins to increase again, forming an asymmetric U-shaped curve.



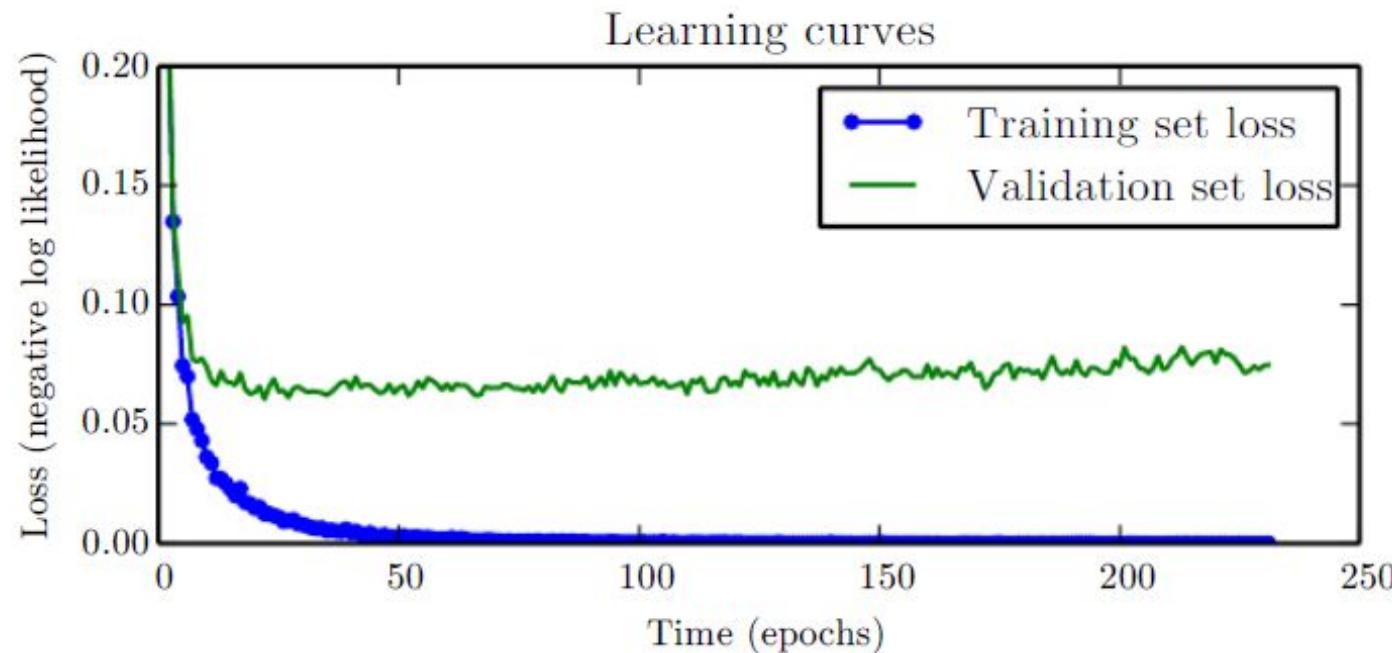
Early stopping

- No longer looking for a local minimum of validation error, while training.
- Train until the validation set error has not improved for some amount of time.
- Every time the error on the validation set improves, store a copy of the model parameters. When the training algorithm terminates, return these parameters.



Early stopping

- Effective and simple form of regularization.
- Trains simpler models



Early Stopping code

```
>>> callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
>>> # This callback will stop the training when there is no improvement in
>>> # the loss for three consecutive epochs.
>>> model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
>>> model.compile(tf.keras.optimizers.SGD(), loss='mse')
>>> history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                      epochs=10, batch_size=1, callbacks=[callback],
...                      verbose=0)
>>> len(history.history['loss']) # Only 4 epochs are run.
```

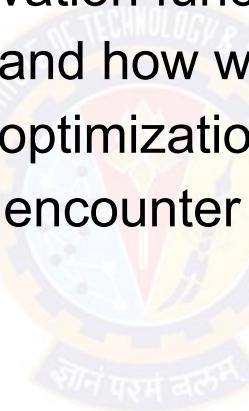
4

Numerical Stability and Initialization



Why Initialization is important?

- The choice of initialization is crucial for maintaining numerical stability.
- The choices of initialization can be tied up in interesting ways with the choice of the nonlinear activation function.
- Which function we choose and how we initialize parameters can determine how quickly our optimization algorithm converges.
- Poor choices can cause to encounter exploding or vanishing gradients while training.



Vanishing and Exploding Gradients

- Consider a deep network with L layers, input \mathbf{x} and output \mathbf{o} . With each layer I defined by a transformation f_I parameterized by weights $\mathbf{W}(I)$, whose hidden variable is $\mathbf{h}(I)$

$$\mathbf{h}^{(l)} = f_l(\mathbf{h}^{(l-1)}) \text{ and thus } \mathbf{o} = f_L \circ \dots \circ f_1(\mathbf{x}).$$

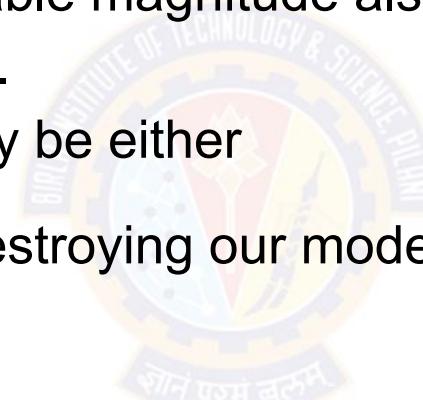
- If all the hidden variables and the input are vectors, then the gradient of \mathbf{o} with respect to any set of parameters $\mathbf{W}(I)$

$$\partial_{\mathbf{W}^{(l)}} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{(L-1)}} \mathbf{h}^{(L)}}_{\mathbf{M}^{(L)} \stackrel{\text{def}}{=} \dots} \cdot \dots \cdot \underbrace{\partial_{\mathbf{h}^{(l)}} \mathbf{h}^{(l+1)}}_{\mathbf{M}^{(l+1)} \stackrel{\text{def}}{=} \dots} \underbrace{\partial_{\mathbf{W}^{(l)}} \mathbf{h}^{(l)}}_{\mathbf{v}^{(l)} \stackrel{\text{def}}{=}}.$$

- Gradient is the product of $L - 1$ matrices $\mathbf{M}(L) \cdot \dots \cdot \mathbf{M}(l+1)$ and the gradient vector $\mathbf{v}(l)$.

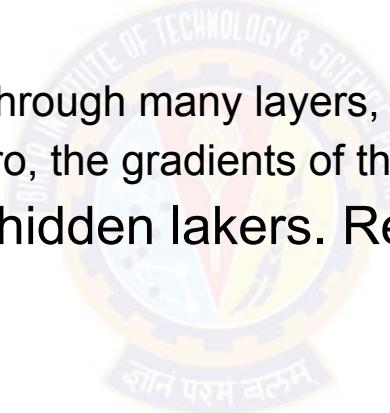
Vanishing and Exploding Gradients

- The matrices $M(I)$ may have a wide variety of eigenvalues. They might be small or large, and their product might be very large or very small.
- Gradients of unpredictable magnitude also threaten the stability of the optimization algorithms.
- Parameter updates may be either
 - (i) excessively large, destroying our model (the **exploding gradient** problem);
 - (ii) excessively small (the **vanishing gradient** problem), rendering learning impossible as parameters hardly move on each update.



Vanishing Gradients

- Activation function sigmoid σ can cause the vanishing gradient problem.
 - The sigmoid's gradient vanishes both when its inputs are large and when they are small.
 - When backpropagating through many layers, where the inputs to many of the sigmoids are close to zero, the gradients of the overall product may vanish.
- Solution: Use ReLU for hidden layers. ReLU is more stable.



Parameter Initialization

1. Default Initialization
 - Used a normal distribution to initialize the values of the parameters.
2. Xavier Initialization
 - samples weights from a Gaussian distribution with zero mean and variance

$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}.$$

- now-standard and practically beneficial

Batch Normalization



Why Batch Normalization?

1. Standardize the input features to each have a mean of zero and variance of one. This standardization puts the parameters a priori at a similar scale. Better optimization.
2. In a MLP or CNN, as we train, the variables in intermediate layers may take values with widely varying magnitudes: both along the layers from the input to the output, across units in the same layer, and over time due to our updates to the model parameters. This drift in the distribution of such variables could hamper the convergence of the network.
3. Deeper networks are complex and easily capable of overfitting. This means that regularization becomes more critical.

Batch Normalization

- Batch normalization is a popular and effective technique that consistently accelerates the convergence of deep networks.
- Batch normalization is applied to individual layers.
- It works as follows:
 - In each training iteration, first normalize the inputs (of batch normalization) by subtracting their mean and dividing by their standard deviation, where both are estimated based on the statistics of the current minibatch.
 - Next, apply a scale coefficient and a scale offset.
- Due to the normalization based on batch statistics that batch normalization derives its name.
- Batch normalization works best for moderate minibatches sizes in the 50 to 100 range.

Batch Normalization

- Denote by $\mathbf{x} \in \mathcal{B}$ an input to batch normalization (BN) that is from a minibatch \mathcal{B} , batch normalization transforms \mathbf{x} as

$$\text{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \hat{\mu}_{\mathcal{B}}}{\hat{\sigma}_{\mathcal{B}}} + \beta.$$

- $\hat{\mu}_{\mathcal{B}}$ is the sample mean and $\hat{\sigma}_{\mathcal{B}}$ is the sample standard deviation of the minibatch \mathcal{B} .
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

- Denote by $\mathbf{x} \in \mathcal{B}$ an input to batch normalization (BN) that is from a minibatch \mathcal{B} , batch normalization transforms \mathbf{x} as

$$\text{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \hat{\mu}_{\mathcal{B}}}{\hat{\sigma}_{\mathcal{B}}} + \beta.$$

- $\hat{\mu}_{\mathcal{B}}$ is the sample mean and $\hat{\sigma}_{\mathcal{B}}$ is the sample standard deviation of the minibatch \mathcal{B} .
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

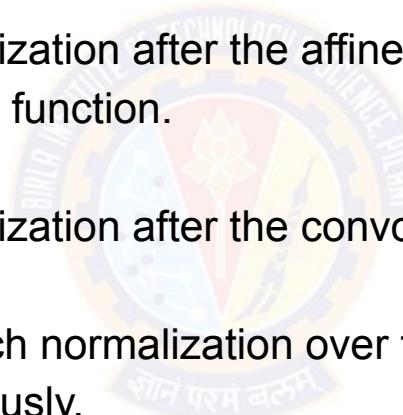
- Elementwise scale parameter γ and shift parameter β that have the same shape as x . γ and β are parameters are learned jointly with the other model parameters.
- Batch normalization actively centers and rescales the inputs to each layer back to a given mean and size.
- Calculate $\hat{\mu}_B$ and $\hat{\sigma}_B^2$

$$\hat{\mu}_B = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x},$$

$$\hat{\sigma}_B^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \hat{\mu}_B)^2 + \epsilon.$$

Batch Normalization Layers

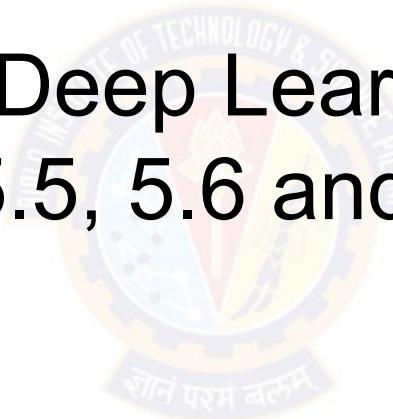
- Batch normalization implementations for fully-connected layers and convolutional layers are slightly different.
 - Fully-Connected Layers
 - Insert batch normalization after the affine transformation and before the nonlinear activation function.
 - Convolutional Layers
 - Apply batch normalization after the convolution and before the nonlinear activation function.
 - Carry out each batch normalization over the $m \cdot p \cdot q$ elements per output channel simultaneously.
- It operates on a full minibatch at a time.



Batch Normalization During Prediction

- After training, use the entire dataset to compute stable estimates of the variable statistics and then fix them at prediction time.



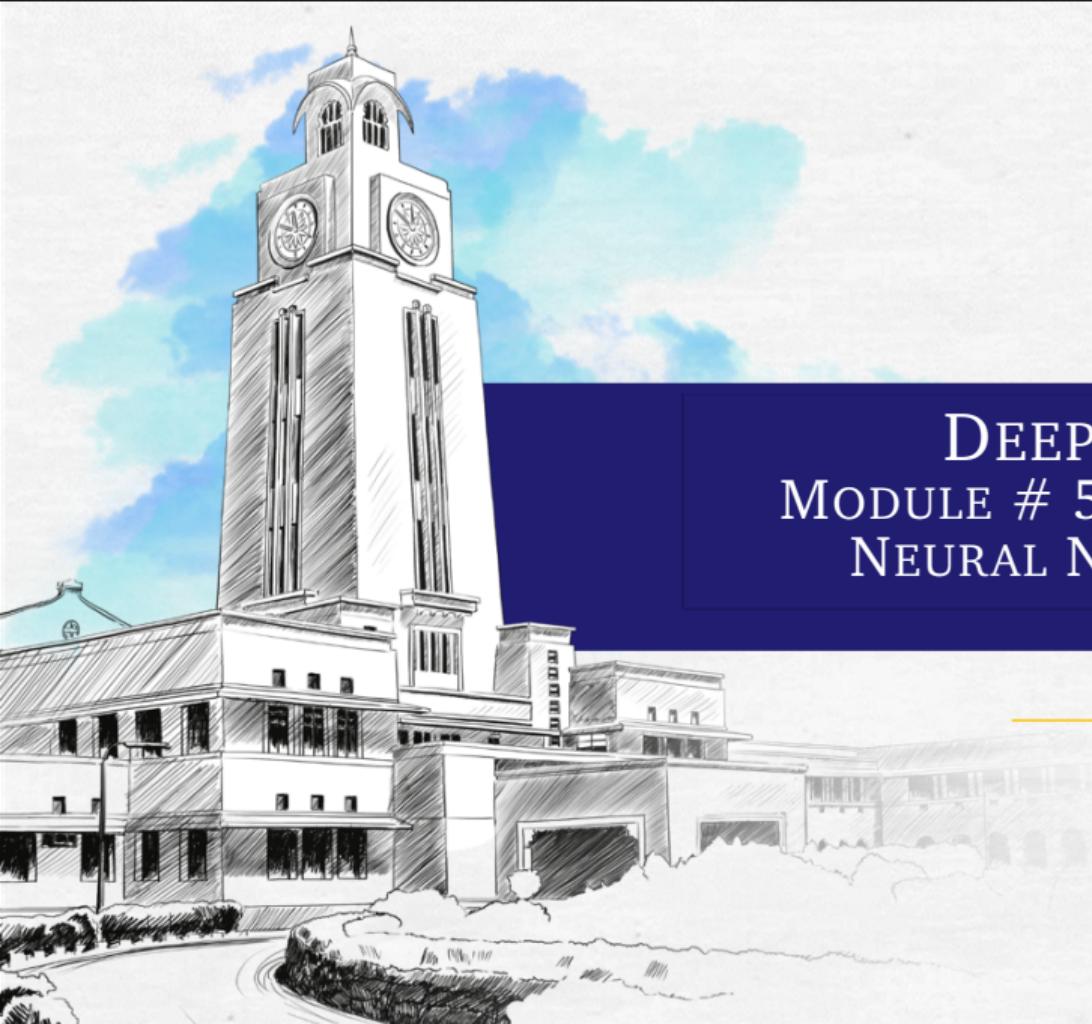


Ref TB Dive into Deep Learning

- Sections 5.4, 5.5, 5.6 and 8,5 (online version)

Next Session: CNN



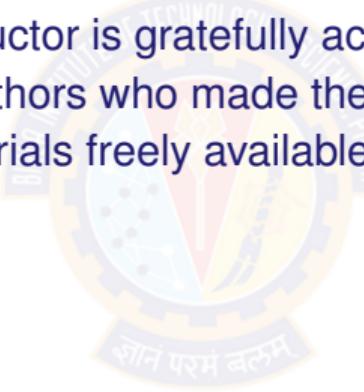


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DEEP LEARNING MODULE # 5 : CONVOLUTIONAL NEURAL NETWORK [CNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.

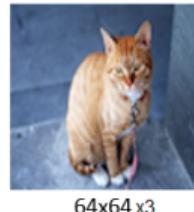


IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



IMAGE CLASSIFICATION



→ Cat? (0/1)



→ Cat? (0/1)



Classify into 3 classes

FACE DETECTION



OBJECT RECOGNITION

"SIFT" & Object Recognition, David Lowe, 1999

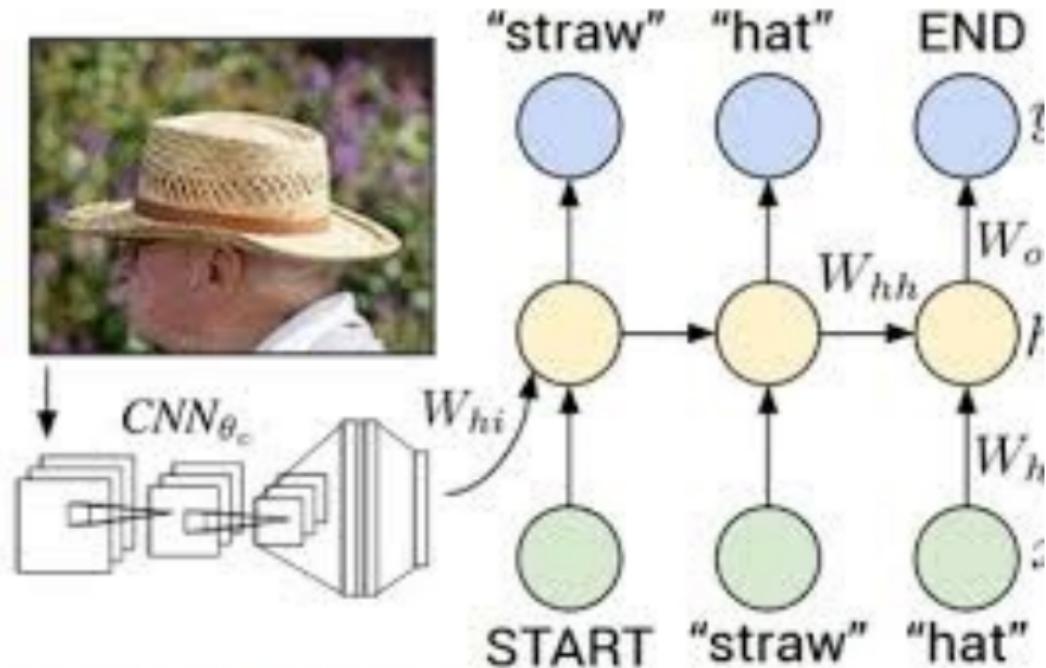


[Image](#) is public domain



[Image](#) is CC BY-SA 2.0

IMAGE CAPTIONING



Andrej Karpathy, Fei-Fei Li

IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



DIGITAL GRayscale IMAGE

Pixel intensities = $I(x, y)$



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0
0	10	16	115	238	255	244	245	243	250	249	255	222	103	10
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124
2	90	255	228	255	251	254	211	141	116	122	215	251	238	255
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137
0	81	252	250	248	215	60	0	1	21	252	255	248	144	6
0	13	111	255	255	245	255	182	181	248	252	242	208	36	0
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1
0	0	4	91	255	255	255	248	252	255	244	255	182	10	0
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125
0	173	255	255	103	9	20	0	13	3	13	182	251	245	61
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0
0	0	23	111	215	255	250	248	255	255	248	248	118	14	12
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4
0	0	5	5	0	0	0	0	0	0	14	1	0	6	6

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	0	111	255	242	255	158	24	0	0	6	39	255	232	230	56
0	0	218	251	250	137	7	11	0	0	0	2	62	255	250	125
0	0	173	255	255	101	9	20	0	13	3	13	182	251	245	61
0	0	107	251	241	255	230	98	55	19	118	217	248	253	255	52
0	0	18	146	250	255	247	245	255	255	249	255	240	255	129	0
0	0	0	0	23	113	215	255	250	248	255	255	248	248	118	14
0	0	0	6	1	0	52	153	233	255	252	147	37	0	0	4
0	0	0	5	5	0	0	0	0	0	14	1	0	6	6	0

DIGITAL COLOR IMAGE

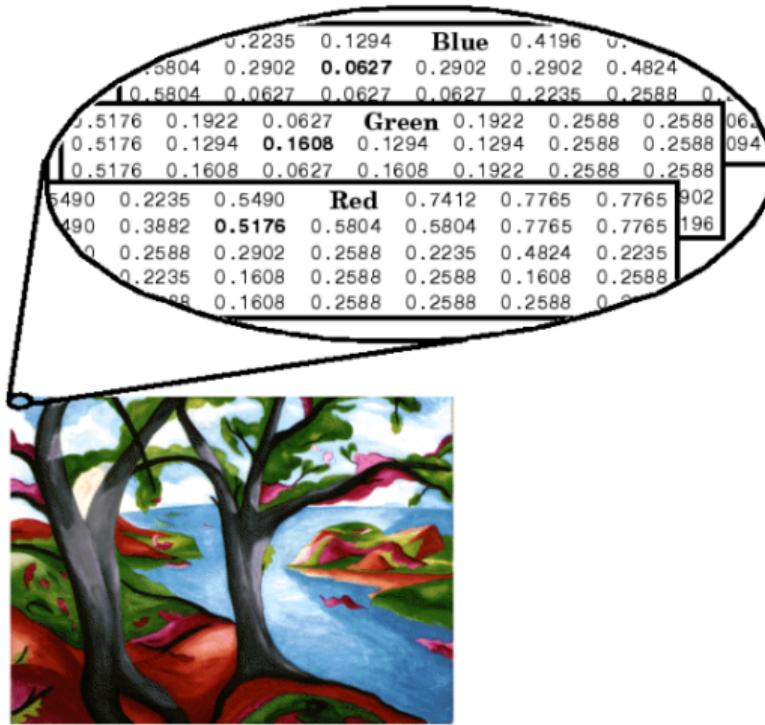


Colour Image

This image is composed of many colors and almost all colors can be generated from the three primary colors- **Red, Green, and Blue**. We can say that each colored image is composed of these three colors or 3 channels- Red, Green, and Blue-



DIGITAL COLOR IMAGE



IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



IMAGE CLASSIFICATION - CHALLENGES

Image Classification: A core task in Computer Vision



This image by Nikita is
licensed under CC-BY 2.0

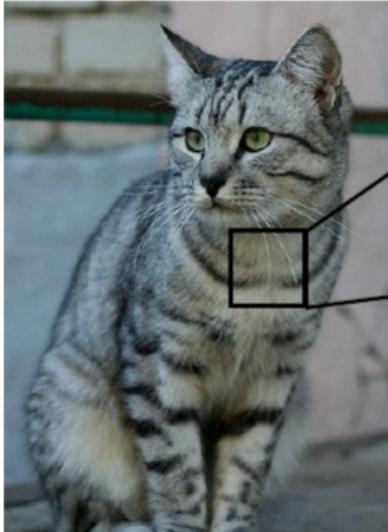
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}



cat

IMAGE CLASSIFICATION - CHALLENGES

The Problem: Semantic Gap



This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

[1] 185 112 106 111 104 99 106 99 96 103 112 119 104 97 93 87]
[1] 91 98 192 106 104 79 98 183 99 105 123 136 118 105 94 85]
[1] 76 85 98 105 128 105 87 96 95 99 115 112 106 103 99 85]
[1] 99 81 81 93 128 131 127 189 95 98 102 99 96 92 181 94]
[1] 106 91 61 64 69 91 88 85 181 187 109 98 75 84 96 95]
[1] 114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[1] 133 137 147 103 65 81 88 65 52 54 74 84 102 93 85 82]
[1] 128 137 144 148 109 95 86 79 62 65 63 63 68 73 88 181]
[1] 125 133 148 137 119 121 117 94 65 79 86 65 54 64 72 98]
[1] 127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[1] 115 114 109 123 158 148 131 118 113 109 106 92 74 65 72 78]
[1] 89 93 98 97 108 147 131 118 113 114 113 109 106 95 77 88]
[1] 63 77 86 81 77 79 102 123 117 115 117 125 125 138 115 87]
[1] 62 65 82 89 78 71 88 101 124 126 119 101 107 114 131 119]
[1] 63 65 75 88 89 71 62 81 120 138 135 105 81 90 118 118]
[1] 87 85 71 87 106 95 69 45 76 130 126 107 92 94 185 112]
[1] 118 97 82 86 117 122 116 66 41 51 95 93 89 95 182 187]
[1] 164 146 112 88 82 128 124 184 76 48 45 66 80 101 182 189]
[1] 157 170 157 128 93 86 114 132 112 97 69 55 78 82 99 94]
[1] 139 128 134 161 139 108 105 118 121 134 114 87 65 53 69 86]
[1] 128 112 96 117 158 144 128 115 184 197 102 93 87 81 72 79]
[1] 123 187 96 86 83 112 153 149 122 189 194 75 89 107 112 99]
[1] 122 121 192 98 82 85 94 117 145 148 153 102 58 78 92 147]
[1] 122 164 148 143 71 56 78 83 93 103 118 139 102 61 69 84]

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Viewpoint variation



All pixels change when
the camera moves!

This image by Nikita is
licensed under [CC-BY 2.0](#)

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Illumination



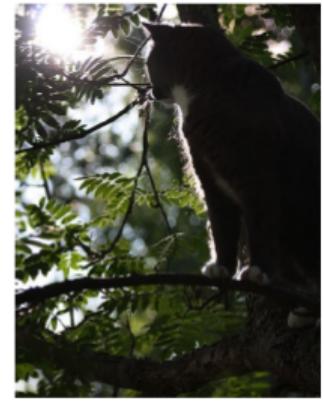
[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



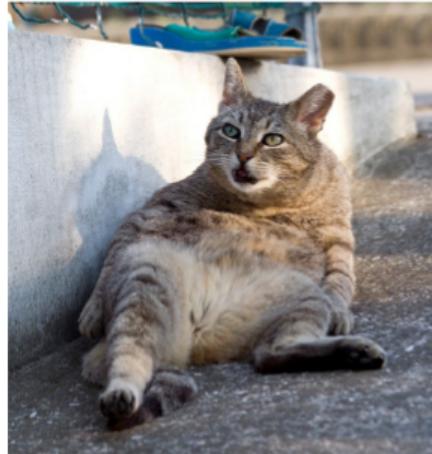
[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Deformation



This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by sare bear is
licensed under CC-BY 2.0



This image by Tom Thai is
licensed under CC-BY 2.0

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Occlusion



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image by jonsson is licensed under CC-BY 2.0](#)

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Background Clutter



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

IMAGE CLASSIFICATION - CHALLENGES

Challenges: Intraclass variation



This image is CC0 1.0 public domain

IMAGE CLASSIFICATION - CHALLENGES

Pixelated Images



Larger Images



- A 50x50 Image requires 2500 input dimensions.
- A HD image 1920x1080 is about 2 million inputs. (2 Mega Pixels)

IMAGE INVARIANCE - CHALLENGES

Object can appear anywhere in the image.

Image A

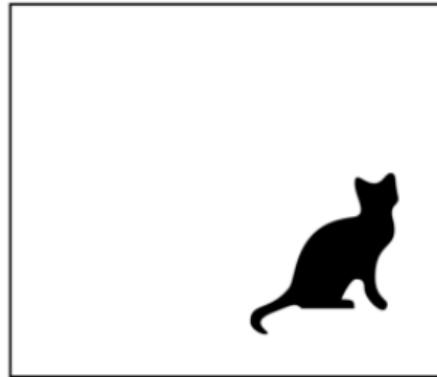
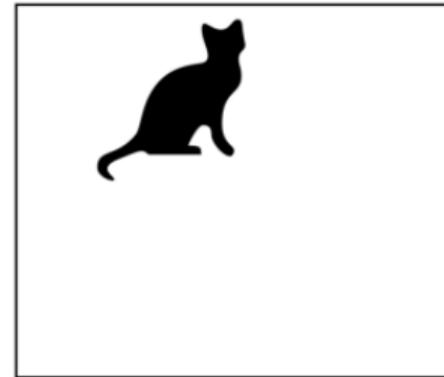


Image B

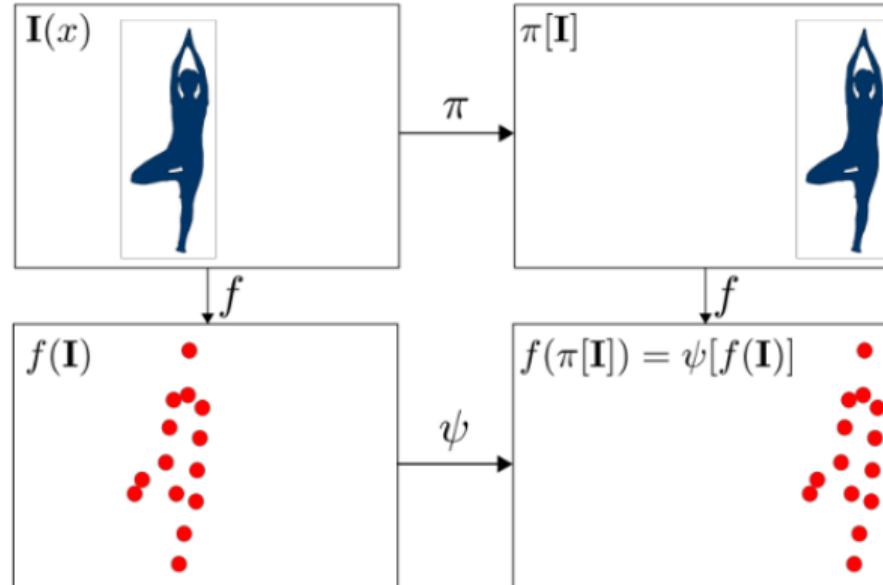


Model predicts “cat” for both images!

Source: <https://penkovsky.com/neural-networks/day5/>

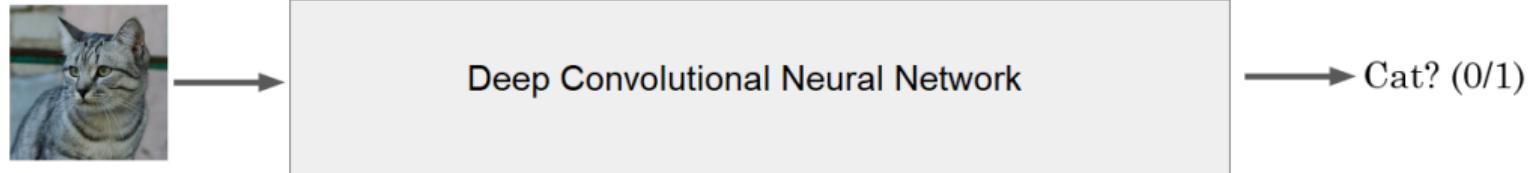
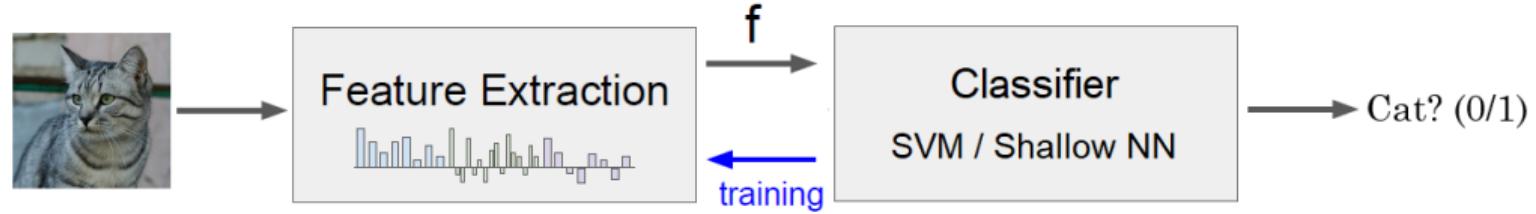
IMAGE EQUIVARIANCE - CHALLENGES

Changes to the input predictably (equivalently) should not affect the output.



Source: <https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fcfa63a>

IMAGE CLASSIFICATION - SOLUTION



IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



CONVOLUTIONAL NEURAL NETWORKS (CNN)

- Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- Special kind of neural network for processing data that has a known, grid-like topology.
- Eg: Time-series data – 1D grid taking samples at regular time intervals.
- Eg: Image data – 2D grid of pixels.
- Network employs a mathematical operation called **convolution**.

EXAMPLE OF CNN – LENET-5

- Handwritten Character recognition
- 10-categories classification problem

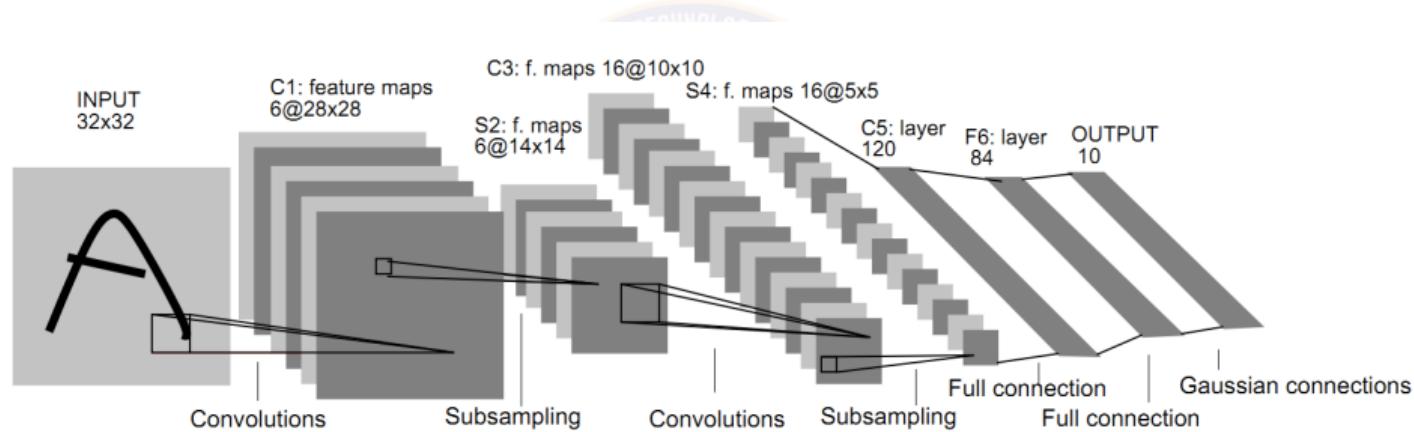


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

WHY CNN FOR IMAGES

- Some patterns are much smaller than the whole image.



WHY CNN FOR IMAGES

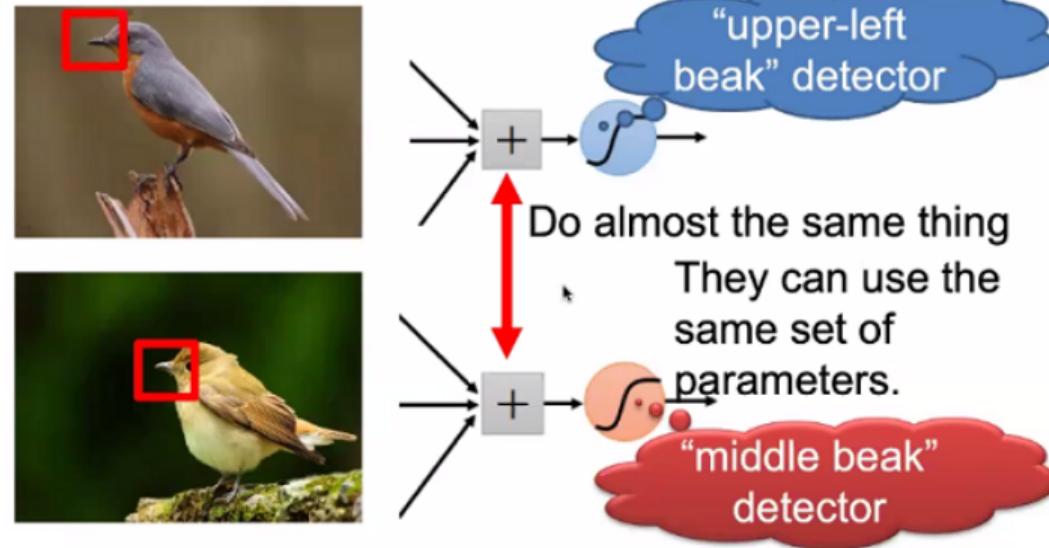
- Some patterns are much smaller than the whole image.

A neuron does not have to see the whole image to discover the pattern.



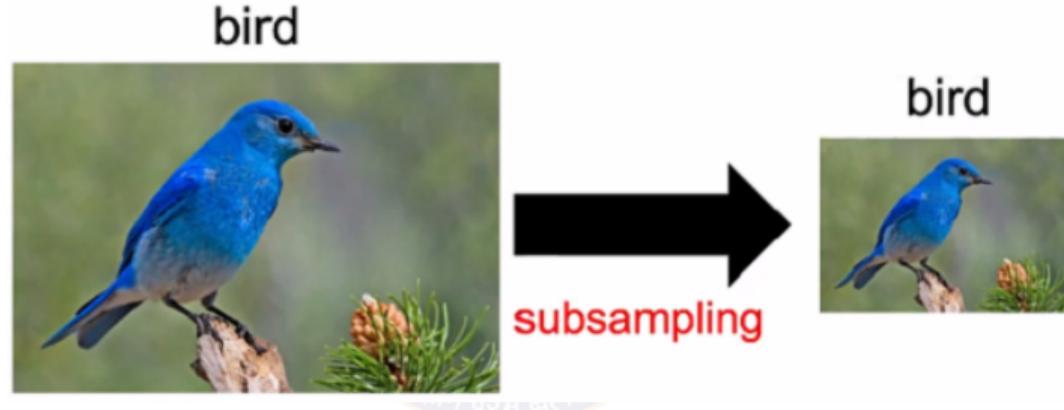
WHY CNN FOR IMAGES

- Same pattern may appear in different regions.



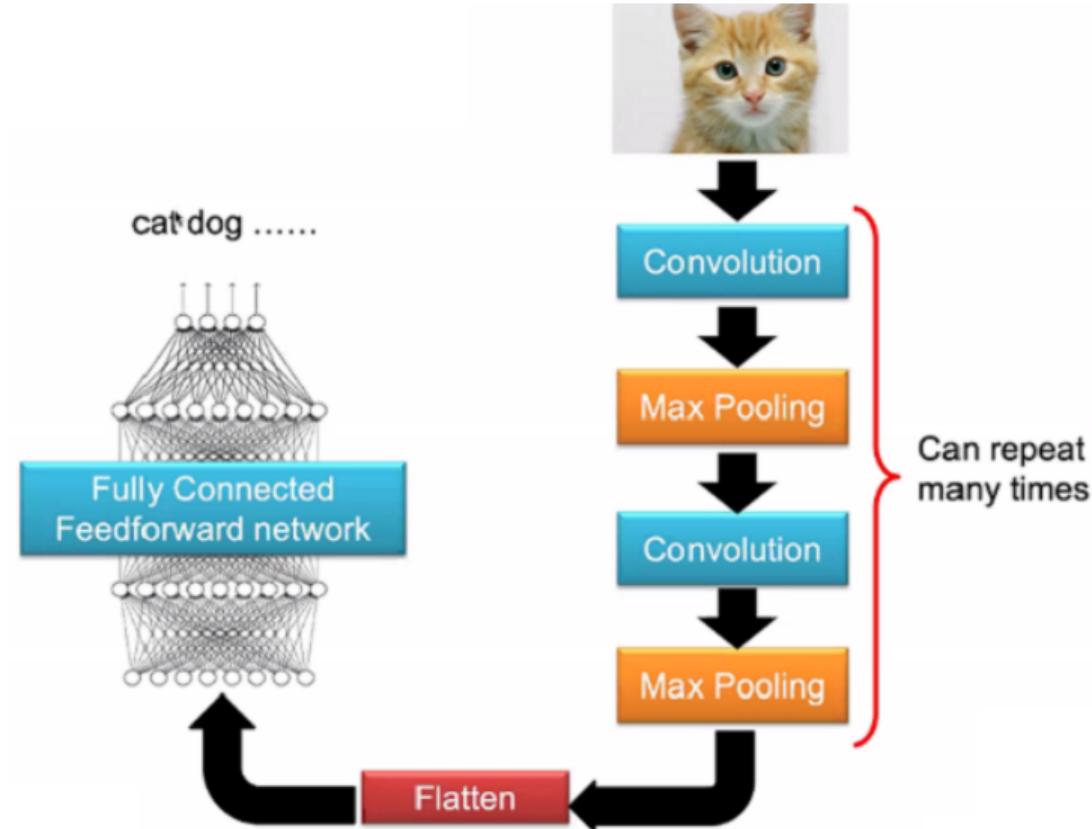
WHY CNN FOR IMAGES

- Sub-sampling the pixels will not change the object.



- We can sub-sample the image to make it smaller.
- Less parameters for the network to process the image.

THE WHOLE CNN



THE WHOLE CNN

Property 1

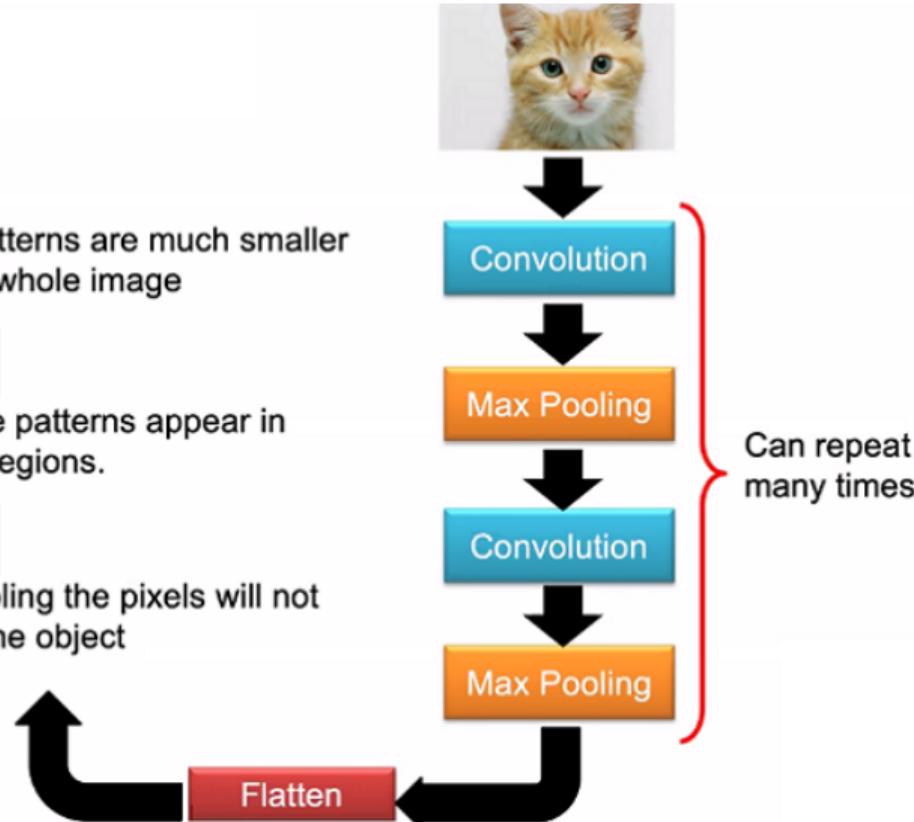
- Some patterns are much smaller than the whole image

Property 2

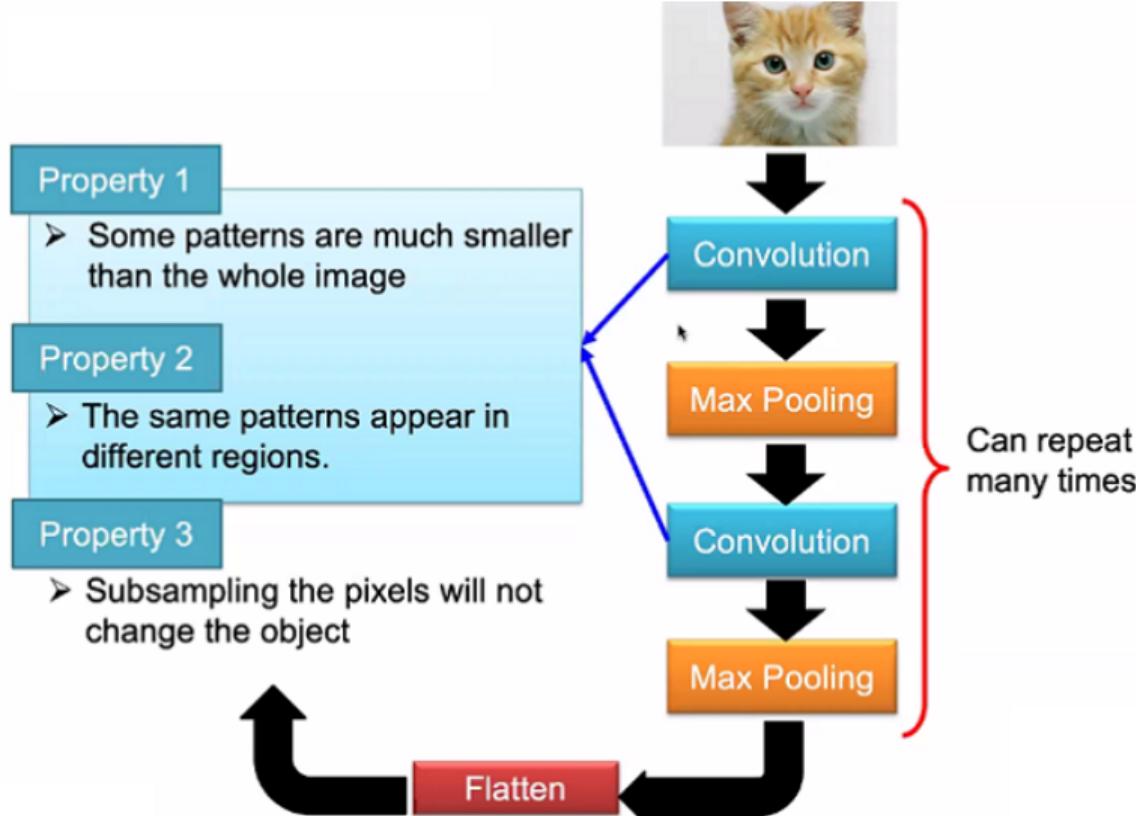
- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object



THE WHOLE CNN



THE WHOLE CNN

The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object



Convolution

Max Pooling

Convolution

Max Pooling



Can repeat
many times

IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



CONVOLUTION LAYER

- Perform Convolution operation.
- In mathematics, convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. [Wikipedia]
- The term convolution refers to both the result function and to the process of computing it. [Wikipedia]
- We will assume that the kernel is square matrix.
- We will assume that the kernel is centered on the pixel of interest. (One reason why kernel is odd matrix.)
- Convolution operation

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

1-D CONVOLUTION

- Suppose we track position of an object using a sensor or GPS device at discrete time intervals.
- Assume that the sensor is noisy.
- To obtain a less noisy estimate, we would like to take a weighted average.
- We would only sum over a small window or a weight array (filter) (w).

$$s_t = \sum_{a=0}^k x_{t-a} w_a$$

- Slide the filter over the input and convolve.
- The input and the kernel are one dimensional.

1-D CONVOLUTION

$$S = x_5w_0 + x_4w_1 + x_3w_2 + x_2w_3 + x_1w_4 + x_0w_5$$

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50
---	------	------	------	------	------	------	------	------	------	------	------

W	0.01	0.02	0.03	0.04	0.4	0.5					
---	------	------	------	------	-----	-----	--	--	--	--	--

S			1.80								
---	--	--	------	--	--	--	--	--	--	--	--

W	0.01	0.02	0.03	0.04	0.4	0.5					
---	------	------	------	------	-----	-----	--	--	--	--	--

S		1.80	1.96								
---	--	------	------	--	--	--	--	--	--	--	--

W	0.01	0.02	0.03	0.04	0.4	0.5					
---	------	------	------	------	-----	-----	--	--	--	--	--

S		1.80	1.96	2.11							
---	--	------	------	------	--	--	--	--	--	--	--

2D CONVOLUTION OPERATION

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

These values inside the matrix are to be learned by the network. We call these as parameters or weights.

1	-1	-1
-1	1	-1
-1	-1	1

Kernel / Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Kernel / Filter 2

Property 1: Each filter detects a small pattern (3×3)

CONVOLUTION OPERATION

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Start at top left corner.

Perform element wise product and add all the values.

Output

3

CONVOLUTION OPERATION

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

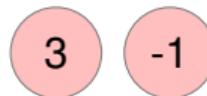
Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right.
Perform element wise product and
add all the values.

Output



CONVOLUTION OPERATION

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

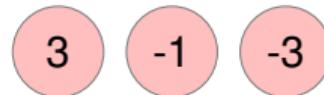
Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right.
Perform element wise product and
add all the values.

Output



CONVOLUTION OPERATION

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

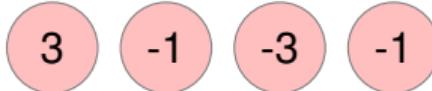
Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right.
Perform element wise product and
add all the values.

Output



CONVOLUTION OPERATION

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

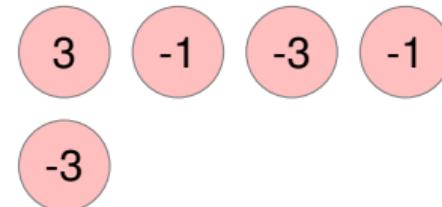
Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Jump to next row, left most pixel.
The filter has to completely
superimpose the pixels of input.
NOT PARTIALLY.

Output



CONVOLUTION OPERATION

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

After sliding through the entire image.

3	-1	-3	-1
-3	-1	0	-3
-3	-2	0	1
3	-2	-2	-1

CONVOLUTION OPERATION

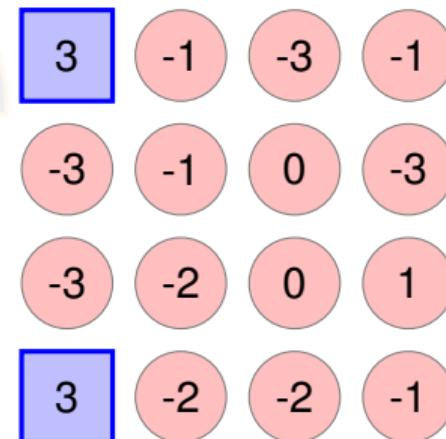
stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



Property 2 is satisfied.

CONVOLUTION OPERATION

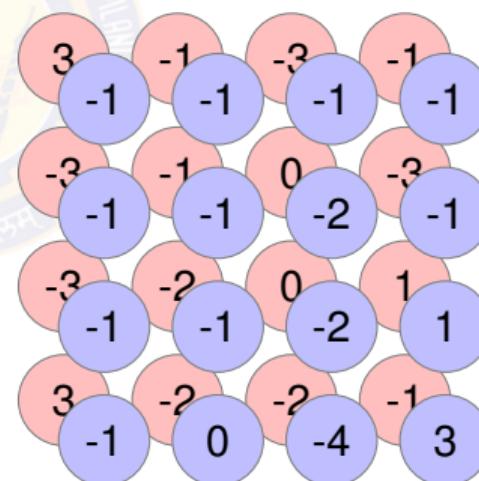
stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6



Filter 1			Filter 2		
1	-1	-1	-1	1	-1
-1	1	-1	-1	1	-1
-1	-1	1	-1	1	-1



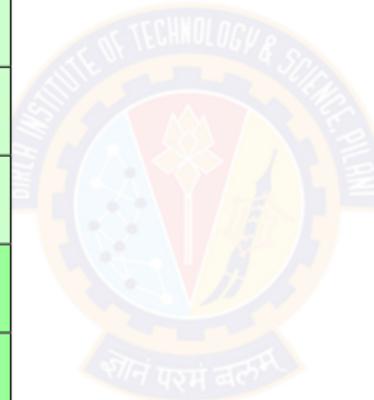
Feature
Maps

CONVOLUTION OPERATION

stride $s = 3$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6



Filter1

1	-1	-1
-1	1	-1
-1	-1	1

Output

3	-1
3	-1

CONVOLUTION WITH STRIDING

- Stride means skip by how many pixels.

Input Size : $n \times n$

Stride : s

Kernel Size : $f \times f$

Output Size : $\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n-f}{s} + 1 \right\rfloor$

- The filter must lie completely inside the input.

PADDING

- When filter is convolved with the input image
 - ▶ Image shrinks – 6×6 image became a 4×4 image.
 - ▶ Pixels in the corners are used only once, when compared the pixels in the middle. The data in the corners are thrown away.
- To resolve, apply padding to the input image before convolution.
- Padding means append zeros around the boundary of the input.

CONVOLUTION WITH PADDING

Input Image with padding 8×8

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

Filter1

$$* \begin{matrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{matrix}$$

PADDING

- Two kinds of padding

- ▶ Valid Padding

- ★ Padding is not applied.

$$p = 0$$

- ★ Input size \neq output size.
 - ★ Output shrinks when compared to the input.

- ▶ Same Padding

- ★ Padding is applied.

$$p = \frac{f - 1}{2}$$

- ★ Input size = output size.
 - ★ We maintain the size of input and output as the same.

PADDING

- How much to pad?

$$\text{Padding : } p = \frac{f - 1}{2}$$

Kernel $3 \times 3 : p = 1$

Kernel $5 \times 5 : p = 2$

Kernel $7 \times 7 : p = 3$

- Padding preserves the input size.

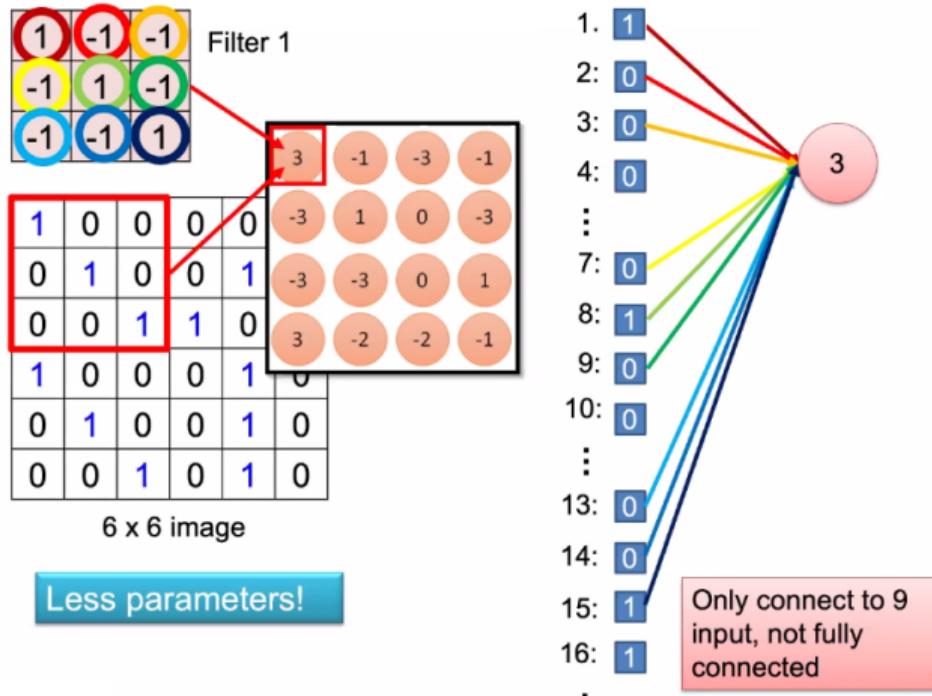
Input Size : $n \times n$

Padding : p

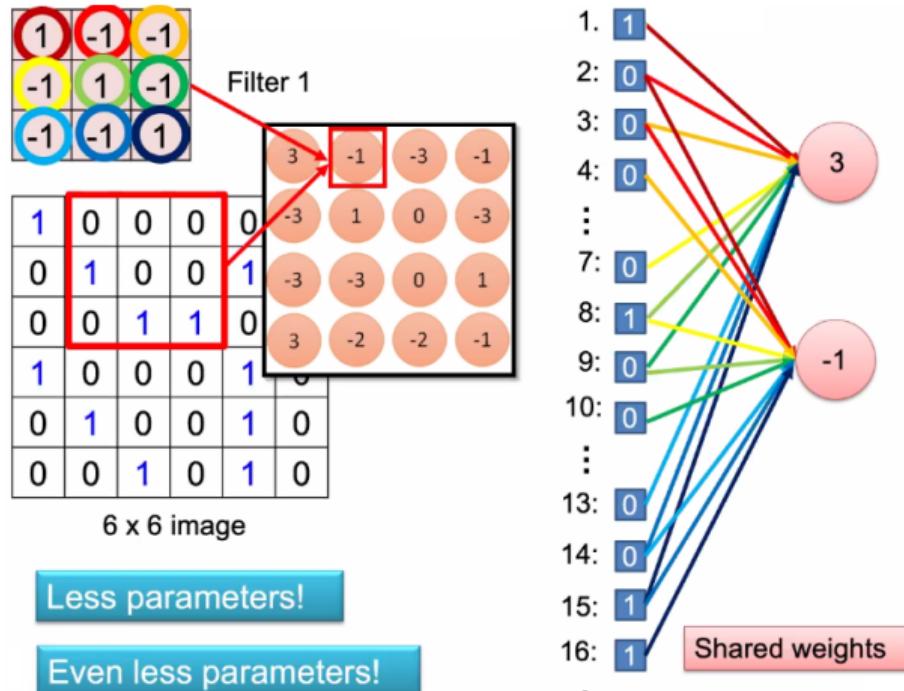
Kernel Size : $f \times f$

Output Size : $(n + 2p - f + 1) \times (n + 2p - f + 1)$

CONVOLUTIONAL LAYER

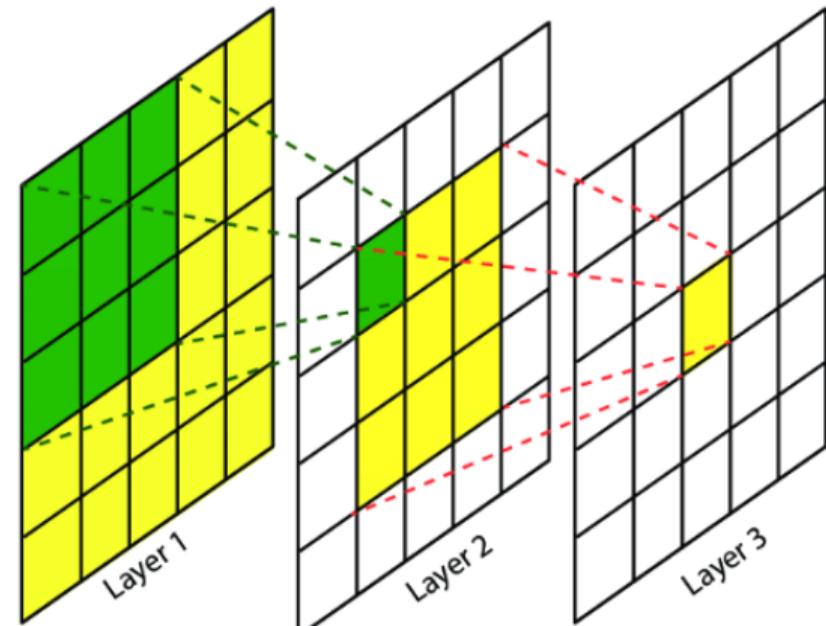


CONVOLUTIONAL LAYER

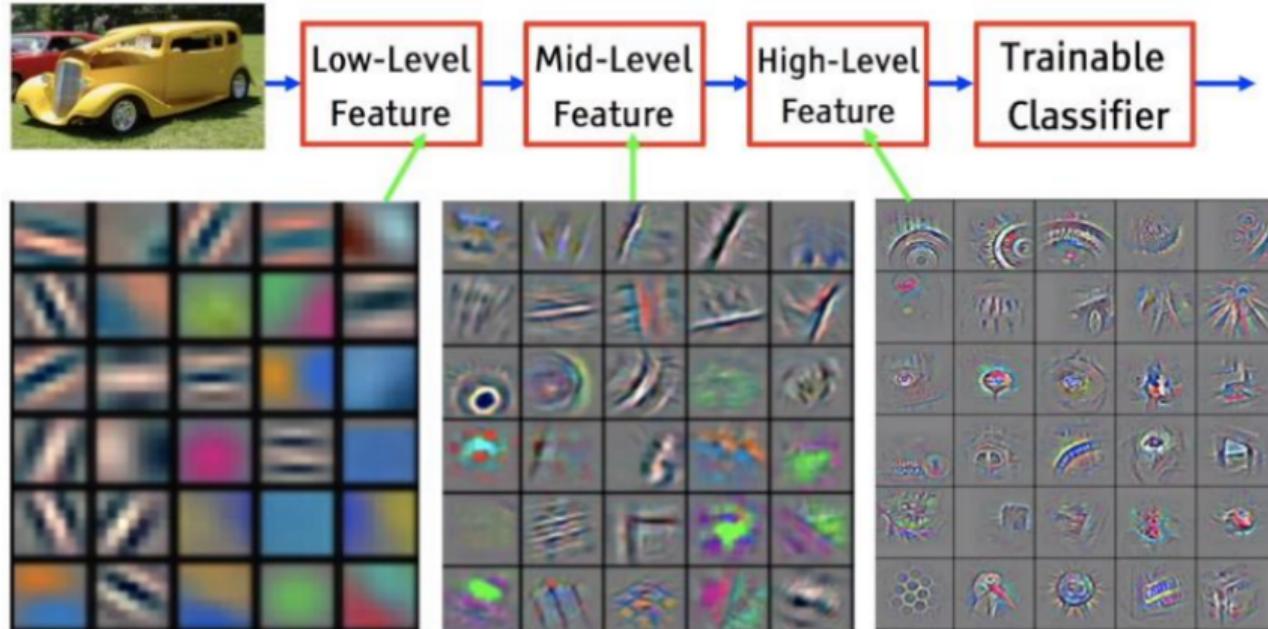


ACTIVATIONS OR RECEPTIVE FIELDS

- Effective area “seen” by a neuron
- What is the effective number of pixels seen at each layer?
 - Input image x
 - $h_1 = \text{Conv2d}(x, k = 3)$
 - $h_2 = \text{Conv2d}(h_1, k = 3)$
 - $h_3 = \text{Conv2d}(h_2, k = 3)$
 - ...
 - $h_n = \text{Conv2d}(h_{n-1}, k = 3)$
- Homework: What happens if the kernel size was 5?



LOCAL TO GLOBAL



PARAMETER SIZE IN CONVOLUTION LAYER

- Input image: $B \times W_{in} \times H_{in} \times C_{in}$
 - ▶ Input data has B samples
 - ▶ Input or Image size is $W_{in} \times H_{in} \times C_{in}$
- Convolution filter
 - ▶ Weight parameters: $C_{in} \times C_{out} \times K_W \times K_H$
 - ▶ Bias parameters: C_{out}
- Output activation: $B \times W_{out} \times H_{out} \times C_{out}$
 - ▶ C_{out} filters of shape $C_{in} \times K_W \times K_H$ operate on each sample

PARAMETER SIZE IN CONVOLUTION LAYER

- Each filter gives one 2D output.
- K filter will give K such 2D outputs.
- The number of parameters depend on the number of kernels and the kernel size.

Number of kernels in layer ($I - 1$) : K_{I-1}

Kernel size : $f \times f$

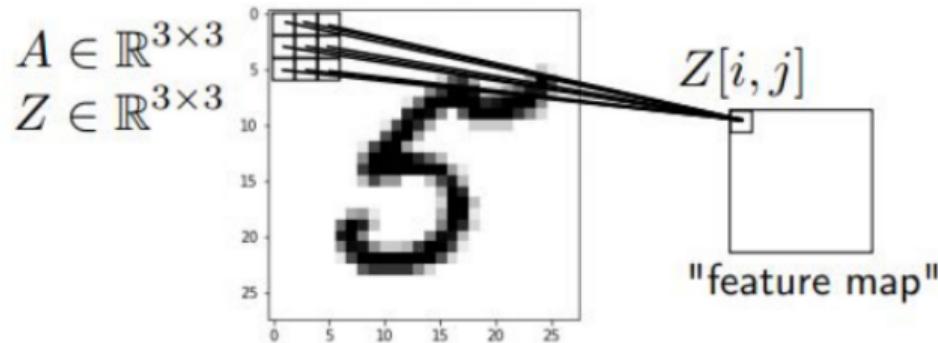
Number of kernels in layer (I) : K_I

Number of Parameters in layer (I) : $K_{I-1} * f * f * K_I$

CROSS-CORRELATION VS CONVOLUTION

- Convolution in Deep Learning is actually cross-correlation.
- Cross-correlation is sliding dot product over the images.
- Cross-correlation

$$Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v] A[i + u, j + v]$$



CROSS-CORRELATION VS CONVOLUTION

Cross-Correlation

$$Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v] A[i+u, j+v]$$

(Note the looping direction given in red.)

1) -1,-1	2) -1,0	3) -1,1
4) 0,-1	5) 0,0	6) 0,1
7) 1,-1	8) 1,0	9) 1,1

Convolution

$$Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v] A[i-u, j-v]$$

(Flipping the kernel.)

9) -1,-1	8) -1,0	7) -1,1
6) 0,-1	5) 0,0	4) 0,1
3) 1,-1	2) 1,0	1) 1,1

CROSS-CORRELATION VS CONVOLUTION

- Convolution in Deep Learning is actually cross-correlation.
- Convolution (as in Signal Processing) has nice association property.

$$(A * B) * C = A * (B * C)$$

- In DL, we usually don't care about that (as opposed to traditional computer vision and signal processing applications.)
- Cross-correlation is easier to implement.
- Maybe the term "convolution" for cross-correlation because popular, because "Cross-correlation Neural Network" sounds weird.

TRANSPOSED CONVOLUTION

Regular Convolution:

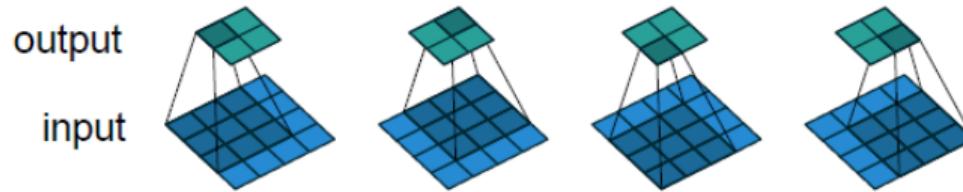
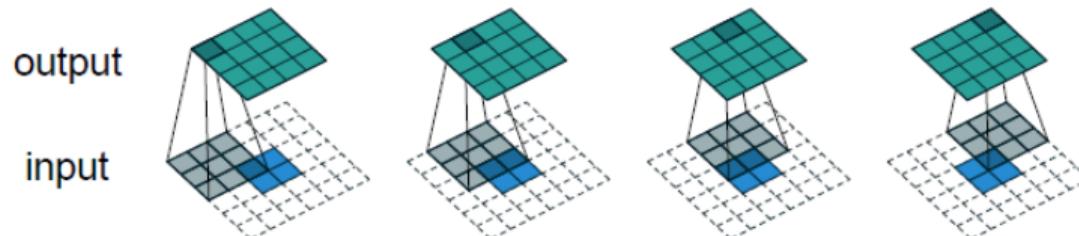


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Transposed Convolution (emulated with direct convolution):



Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

DILATED CONVOLUTION

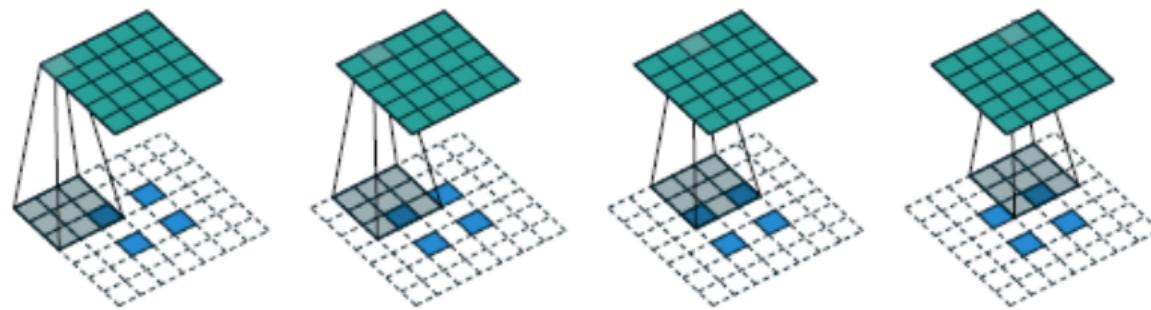


Figure 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5, k = 3, s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides

Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." arXiv preprint arXiv:1603.07285 (2016).

REMEMBER IMAGE INVARIANCE CHALLENGE

Object can appear anywhere in the image.

Image A

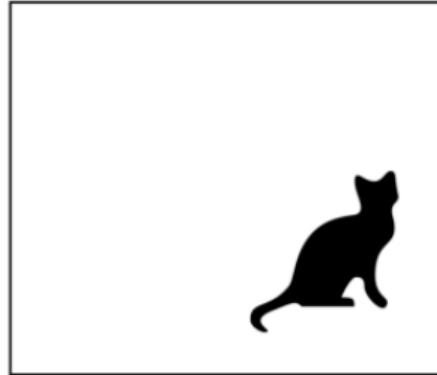
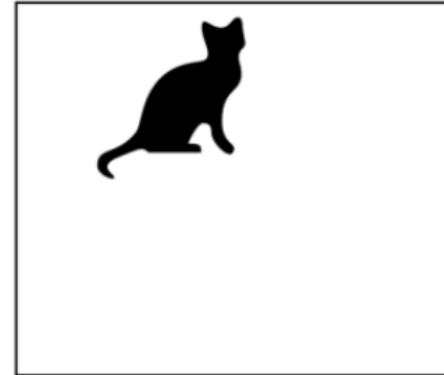


Image B

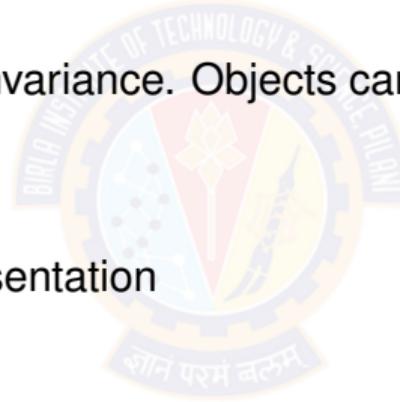


Model predicts “cat” for both images!

Source: <https://penkovsky.com/neural-networks/day5/>

POOLING (POOL) LAYER

- Helps to achieve image invariance. Objects can appear anywhere in the Image and still get detected.
- Used for sub-sampling
- Reduce the size of representation
- Speed up computation



POOLING

- Apply a filter of size f and stride s . Padding is not applied.
- The output dimensions will be

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n-f}{s} + 1 \right\rfloor$$

- Fixed computation
- Gradient descent is not applied as there are not parameters to be learned.
- Pooling is applied on each of the channels.

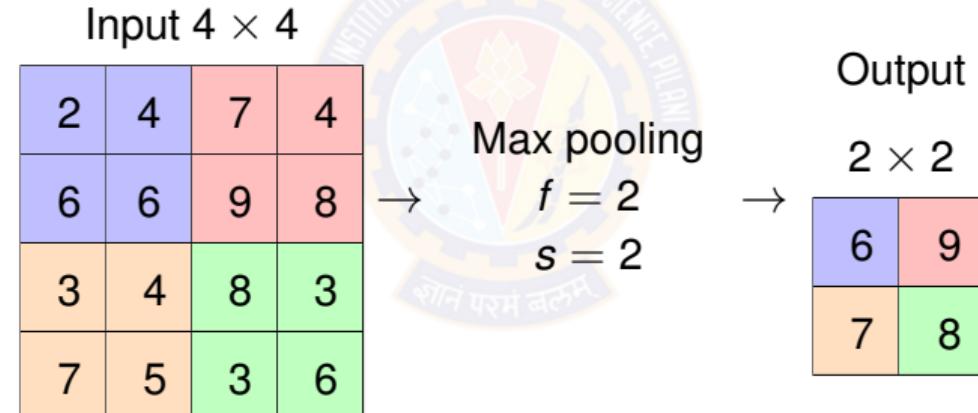
POOLING LAYER

- Two types
 - ① Max pooling
 - ② Average pooling



MAX POOLING

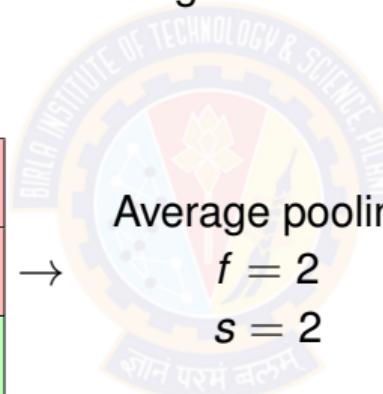
- Take the maximum of the sub region that is considered.



AVERAGE POOLING

- Compute the average of the sub region that is considered.

Input 4×4			
2	4	7	4
6	6	9	8
3	4	8	3
7	5	3	6



Average pooling

$$f = 2$$

$$s = 2$$

Output

$$2 \times 2$$

4.5	7
4.75	5

CONVOLUTION + MAX POOLING

Input Image 6×6

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

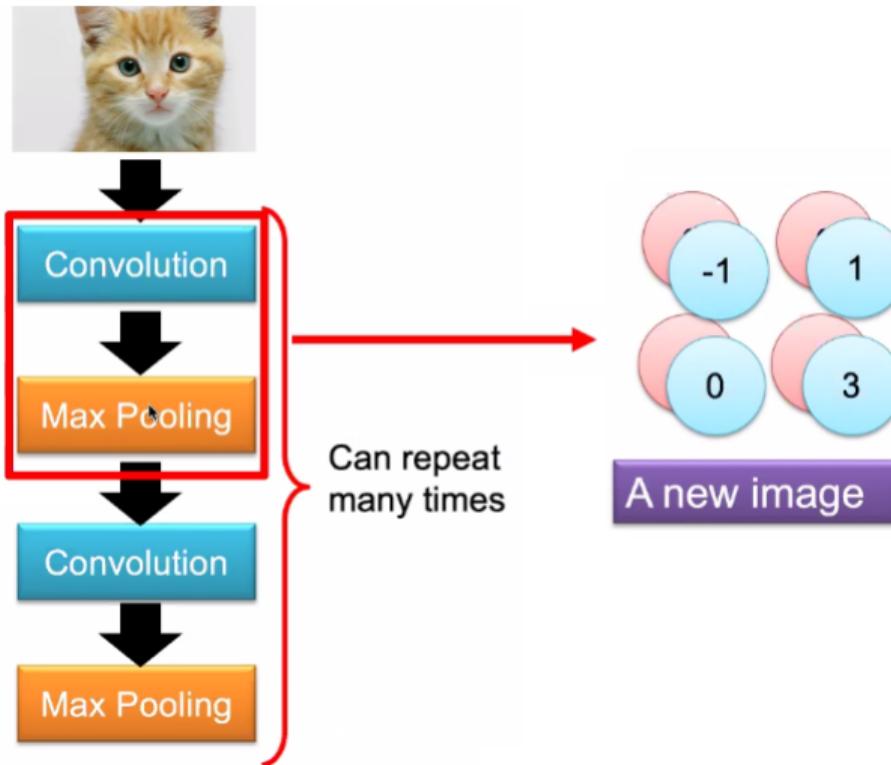
Convolve s=1

3	-1	-3	-1
-3	-1	0	-3
-3	-2	0	1
3	-2	-2	-1
-1	-1	-1	-1
-1	-1	-2	-1
-1	-1	-2	1
-1	0	-4	3

Maxpool f=2, s=2

3	0
3	1
-1	-1
0	3

CONVOLUTION + MAX POOLING



FLATTEN

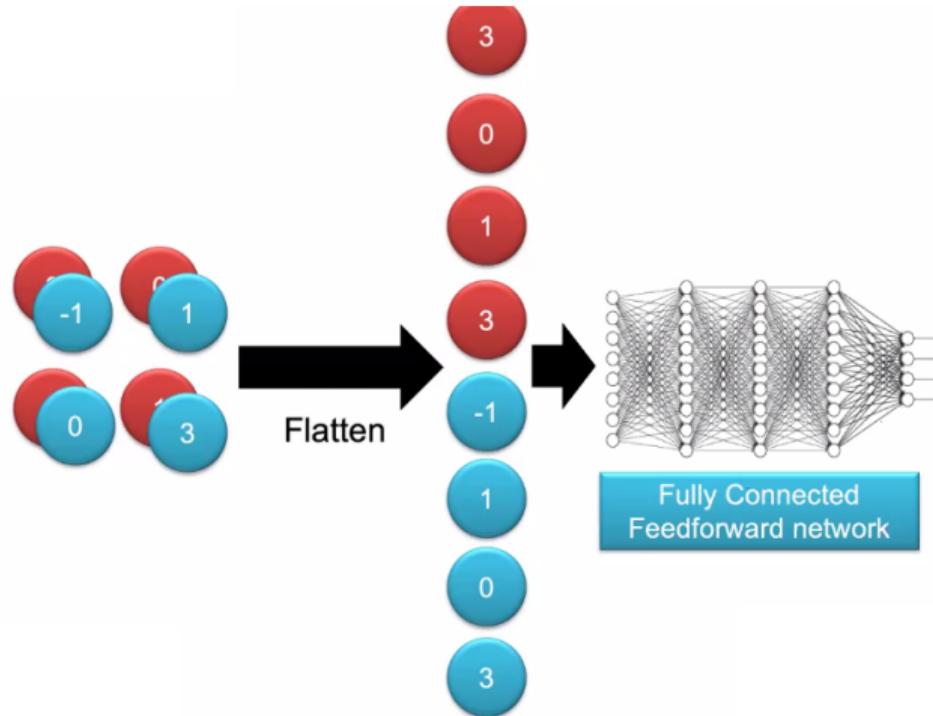


IMAGE IS HIGHLY NON-LINEAR

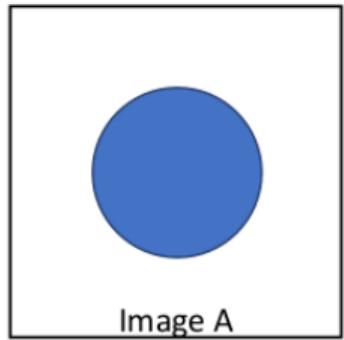


Image A

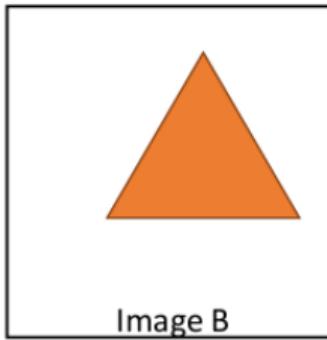
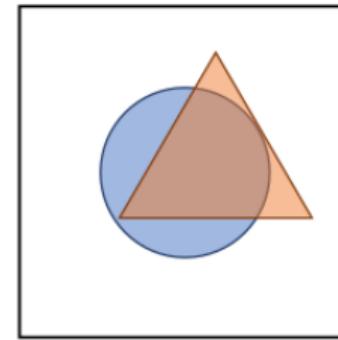
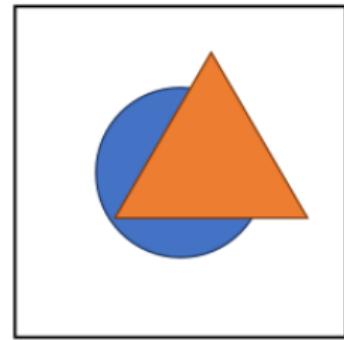


Image B



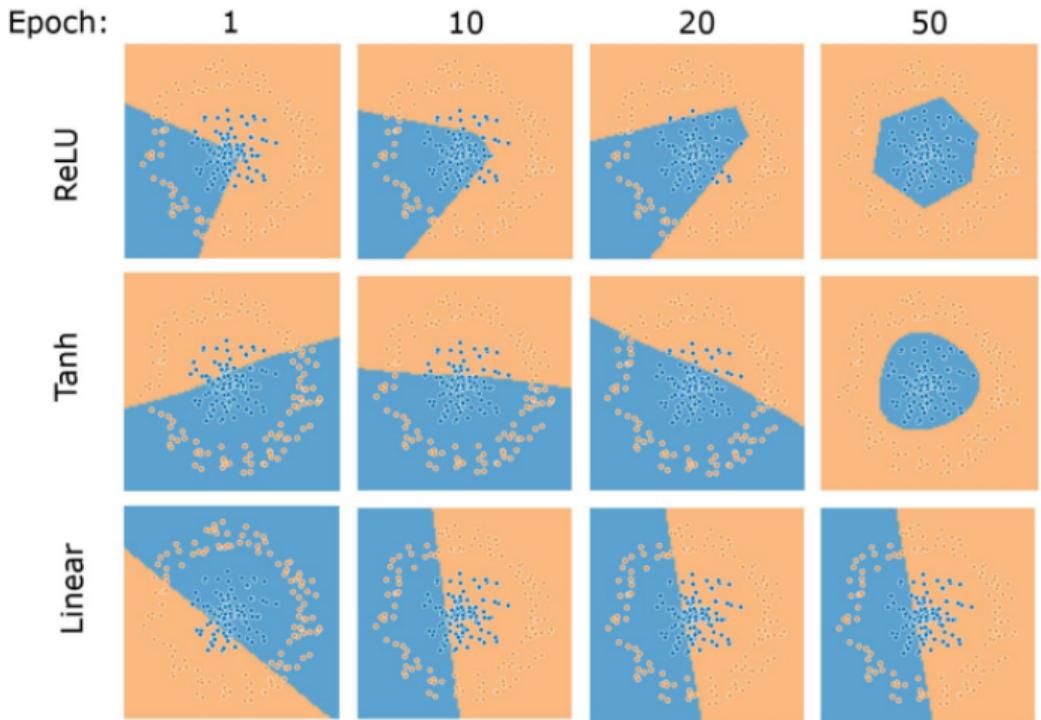
Linear combination
 $A+B$



Actual combination
 $A+B$

RECTIFIED LINEAR UNIT (ReLU)

- Very fast
- Gradients
- Piece-wise linear approximation



RELU IN IMAGES

Ground truth image



Standard fully-connected net

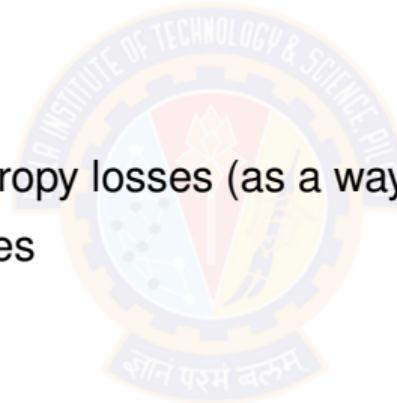


With Positional Encoding



LOSS FOR TRAINING

- Mean Squared Error
- Binary Cross Entropy
- Multi class Cross Entropy
- Class-weighted cross-entropy losses (as a way to counter imbalance)
- L1 (MAE), L2 (MSE) losses
- Focal, Dice, Huber loss
- Margin ranking losses
- Generalized Intersection-over-Union (GloU) loss
- Connectionist Temporal Classification (CTC) loss



REGULARISATION AND OPTIMIZATION

- Drop out
- Batch Normalisation
- RMSProp, Adam or any other optimizer can be used.



CLASSIFICATION

- $p \in [0, 1]$ - use Sigmoid activation function
- k classes - use softmax activation function

IN THIS SEGMENT

- 1 COMPUTER VISION TASKS
- 2 IMAGE BASICS
- 3 IMAGE CLASSIFICATION CHALLENGES
- 4 CONVOLUTIONAL NEURAL NETWORK
- 5 CONVOLUTION
- 6 PUTTING IT TOGETHER



PUTTING EVERYTHING TOGETHER

- ① Identify the task as classification or regression. This helps in identify the output size.
- ② Collect the N images. This helps in identifying input size as $W \times H \times C$.
- ③ Feature extraction
 - ▶ Decide on the number of Convolution layers. This can be derived by looking at how large or small pattern we are interested in.
 - ▶ Decide on the kernel size and number of kernels in each layer. Smaller patterns in images require smaller kernels. Number of kernels can be derived from the non-linearity.
 - ▶ Use ReLU activation function for better approximation.
- ④ Classification
 - ▶ Use Fully connected layers for classification.
 - ▶ The number of (dense) layers depends on the non-linearity.
 - ▶ The last (output) layer activation is decided based on Step 1.
- ⑤ Train and Test on the available dataset.

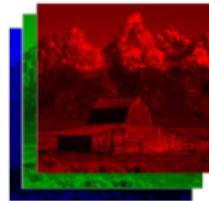
CONVOLUTIONAL NEURAL NETWORK



- Input: 1 or 3 images
 - Grey scale or color
 - Will assume color to be generic

CONVOLUTIONAL NEURAL NETWORK

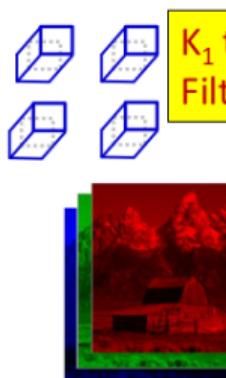
- Input is convolved with a set of K_1 filters or kernels.
 - ▶ Typically, K_1 is a power of 2 e.g 2,4,8,16,32,...
 - ▶ Filters are typically $5 \times 5 \times 3$, $3 \times 3 \times 3$ or $1 \times 1 \times 3$ (better notation).
 - ▶ Typical stride is 1 or 2.



$I \times I$ image

CONVOLUTIONAL NEURAL NETWORK

- Input is convolved with a set of K_1 filters or kernels.
 - ▶ Typically, K_1 is a power of 2 e.g 2,4,8,16,32,...
 - ▶ Filters are typically 5x5, 3x3 or 1x1.
 - ▶ Filters should be small enough to capture fine features, particularly important for scaled down images.



Parameters to choose: K_1 , L and S

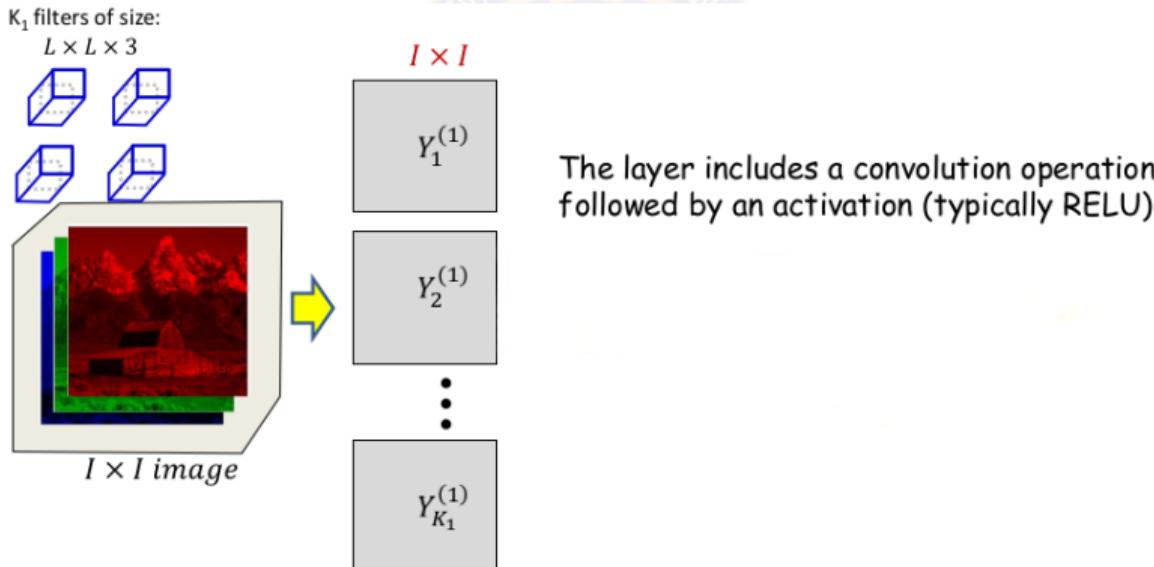
1. Number of filters K_1
2. Size of filters $L \times L \times 3 + \text{bias}$
3. Stride of convolution S

Total number of parameters: $K_1(3L^2 + 1)$

CONVOLUTIONAL NEURAL NETWORK

- First Convolutional layer:

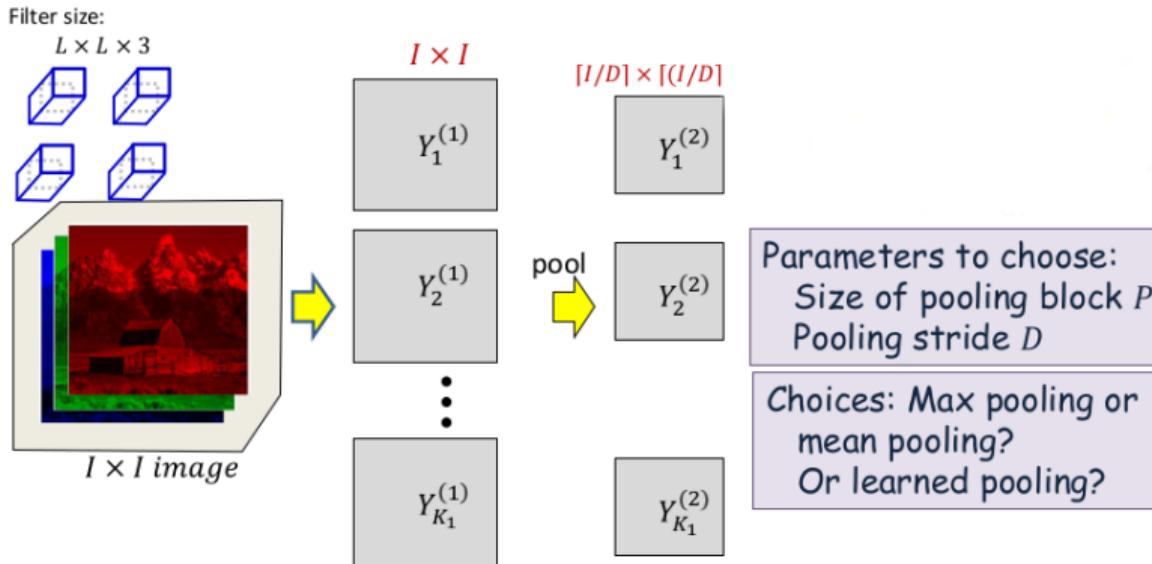
- ▶ Several convolutional filters, which are 3D.
- ▶ Convolution is followed typically by a ReLU activation.
- ▶ Each filter creates a singly 2D feature map.



CONVOLUTIONAL NEURAL NETWORK

- First Pooling layer:

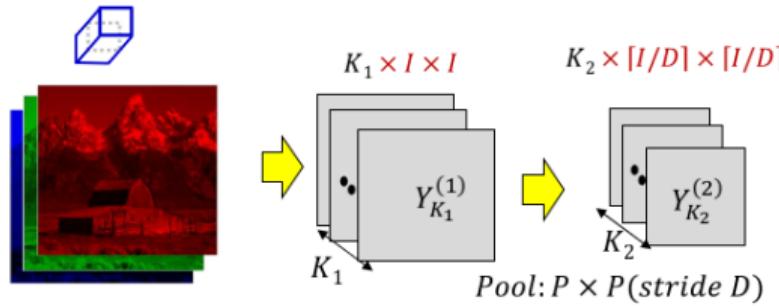
- ▶ For each $P \times P$ block of each map, pool down to a single value.



CONVOLUTIONAL NEURAL NETWORK

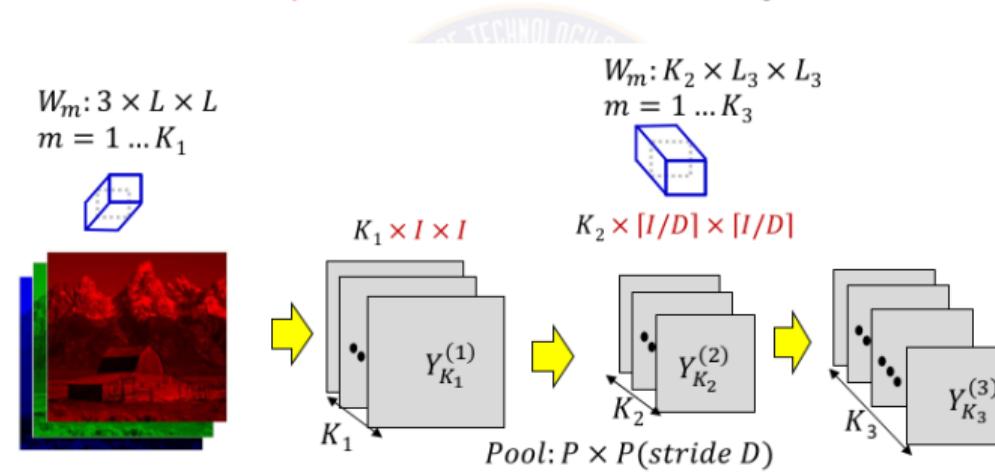
- Better drawing

$$W_m: 3 \times L \times L$$
$$m = 1 \dots K_1$$



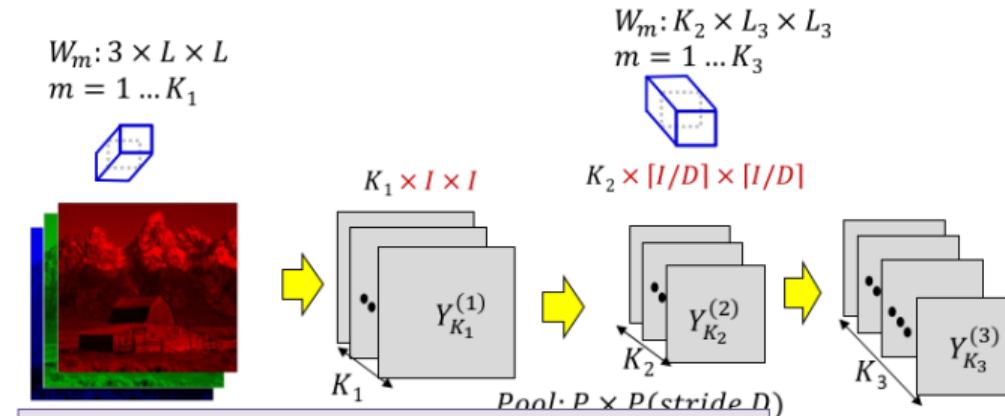
CONVOLUTIONAL NEURAL NETWORK

- Second Convolutional layer" K_2 3D filters resulting in K_2 feature maps.



CONVOLUTIONAL NEURAL NETWORK

- Second Convolutional layer



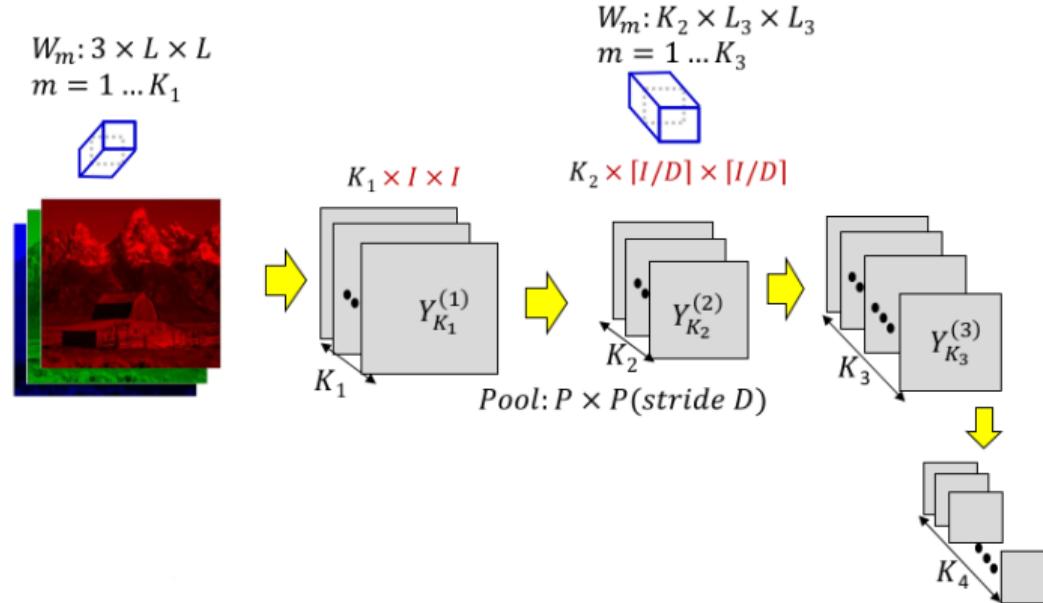
Parameters to choose: K_3, L_3 and S_3

1. Number of filters K_3
2. Size of filters $L_3 \times L_3 \times K_2 + bias$
3. Stride of convolution S_3

Total number of parameters: $K_3(K_2L_3^2 + 1)$
All these parameters must be learned

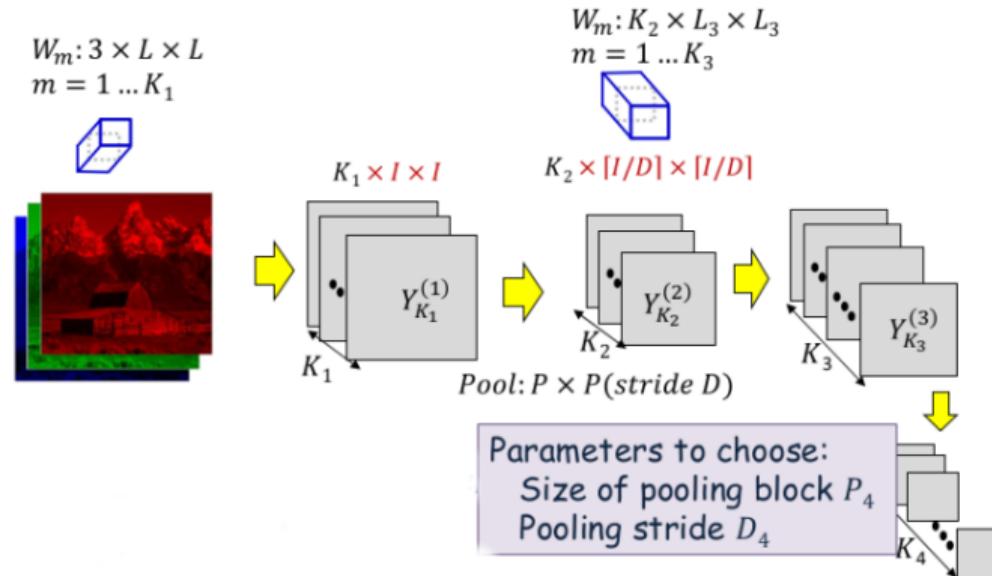
CONVOLUTIONAL NEURAL NETWORK

- **Second Pooling layer:** K_2 pooling operations results in K_2 reduced 2D maps.



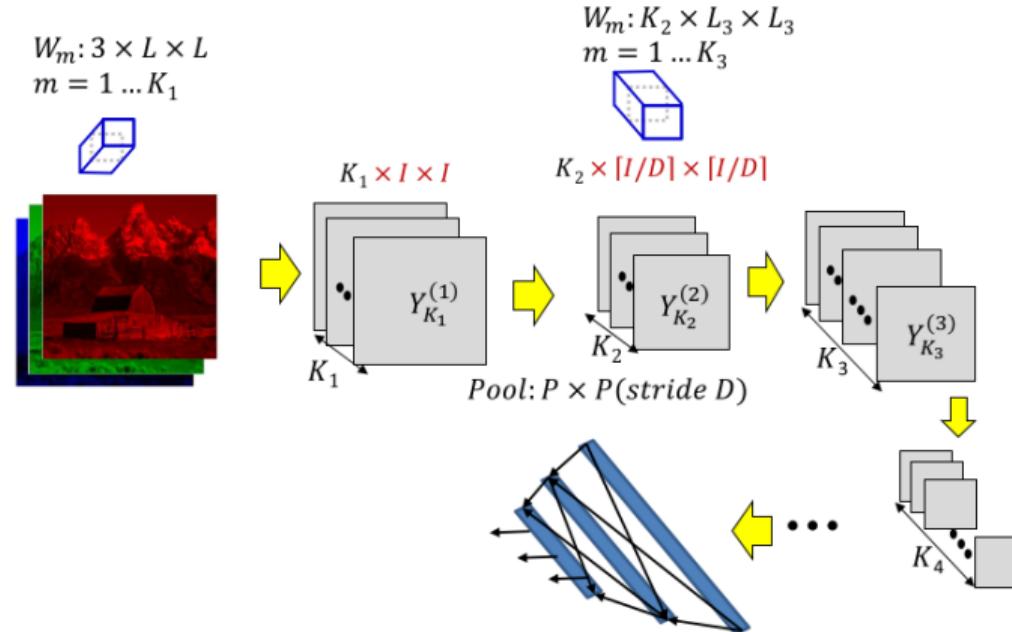
CONVOLUTIONAL NEURAL NETWORK

- Second Pooling layer



CONVOLUTIONAL NEURAL NETWORK

- **Fully connected layer:** This continues for several layers until the final convolved feature maps are fed to MLP.



CONVOLUTIONAL NEURAL NETWORK

Size of Layers

- Each convolution layer with stride 1 typically maintains the size of the image.
 - ▶ With appropriate zero padding.
 - ▶ If performed without zero padding it will decrease the size of the input.
- Each convolution layer will generally increase the number of maps from the previous layer.
 - ▶ Increasing layers reduces the amount of information lost by subsequent downsampling.
- Each pooling layer with stride D decreases the size of the maps by a factor of D.
- Filters within a layer must all be the same size, but sizes may vary with layer. Similarly for pooling D may vary with layer.
- In general the number of convolutional filters increases with layers.

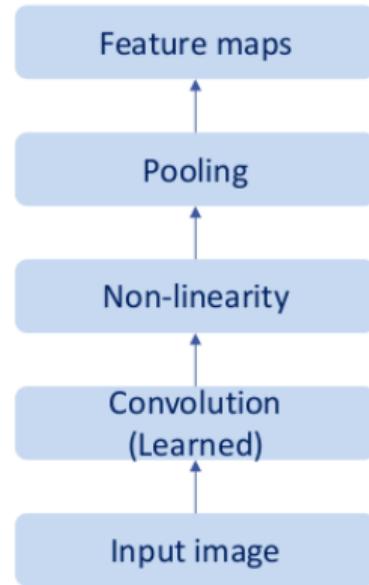
CONVOLUTIONAL NEURAL NETWORK

Parameters to choose (design choices)

- Number of convolutional and downsampling layers
 - ▶ And arrangement (order in which they follow one another)
- For each convolution layer:
 - ▶ Number of filters K_i
 - ▶ Spatial extent of filter $L_i \times L_i$
 - ★ The “depth” of the filter is fixed by the number of filters in the previous layer K_{i-1} .
 - ▶ The stride S_i
- For each downsampling/pooling layer:
 - ▶ Spatial extent of filter $P_i \times P_i$
 - ▶ The stride D_i
- For the final MLP:
 - ▶ Number of layers
 - ▶ Number of neurons in each layer

CNN-Recap

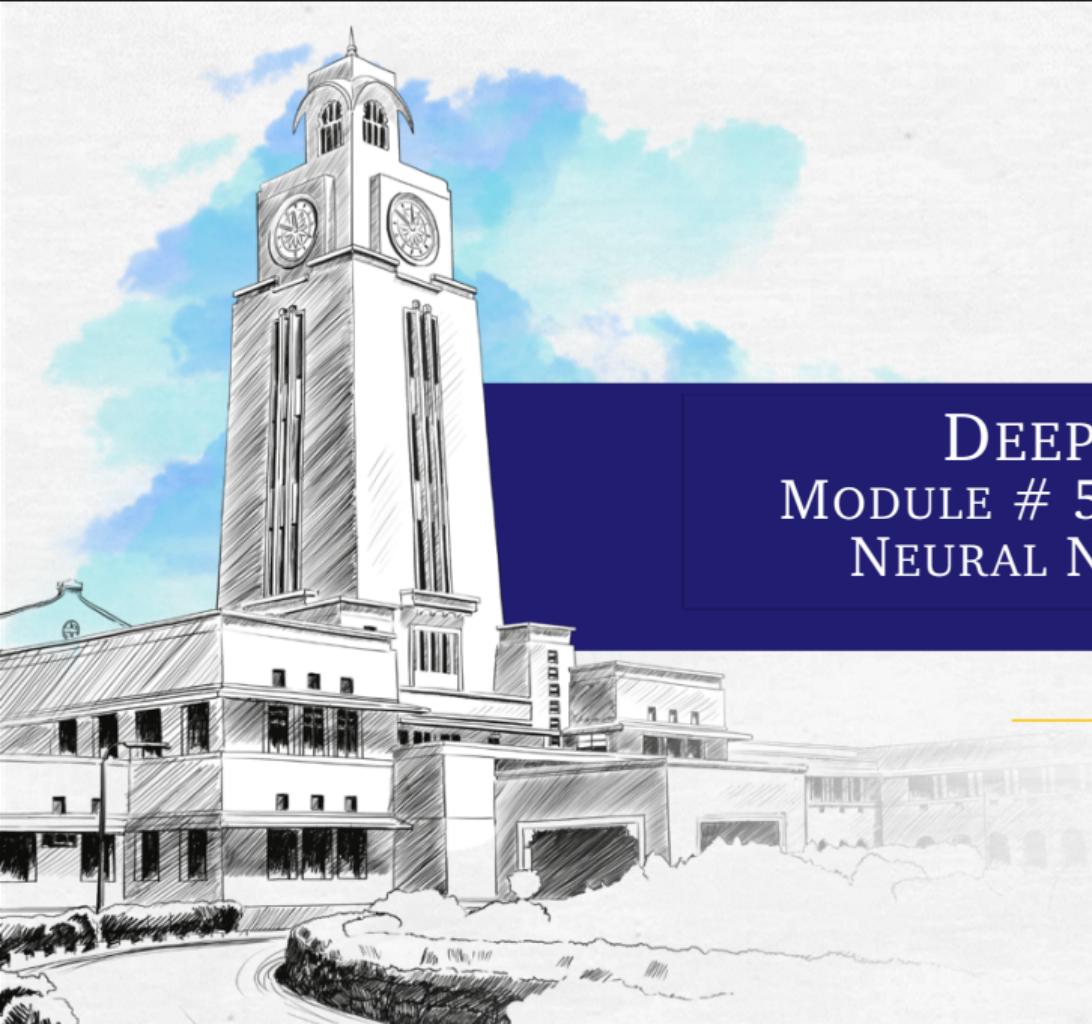
- Neural network with specialized connectivity structure
- Feed-forward:
 - Convolve input
 - Non-linearity (rectified linear)
 - Pooling (local max)
- Supervised training
- Train convolutional filters by back-propagating error
- Convolution over time



References

- ① Deep Learning with Python by Francois Chollet.
<https://livebook.manning.com/book/deep-learning-with-python/>
- ② Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>

Thank You!

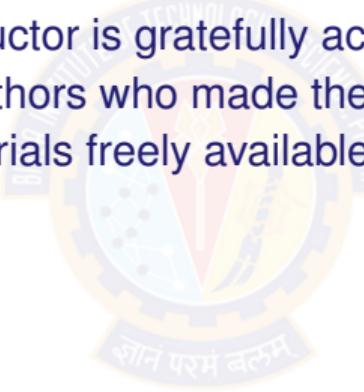


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DEEP LEARNING MODULE # 5 : CONVOLUTIONAL NEURAL NETWORK [CNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.



IN THIS SEGMENT

1 CNN ARCHITECTURES

2 TRANSFER LEARNING

3 FURTHER READING



LENET 5

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions.
- Using convolution to extract spatial features.
- Conv filters were 5×5 , applied at stride 1.
- Subsampling average pooling layer. Subsampling (Pooling) layers were 2×2 applied at stride 2.
- tanh activation function.
- Using MLP as the last classifier.
- Architecture is [CONV-POOL-CONV-POOL-FC-FC]
- Sparse connection between layers to reduce the complexity of computation.

LENET-5

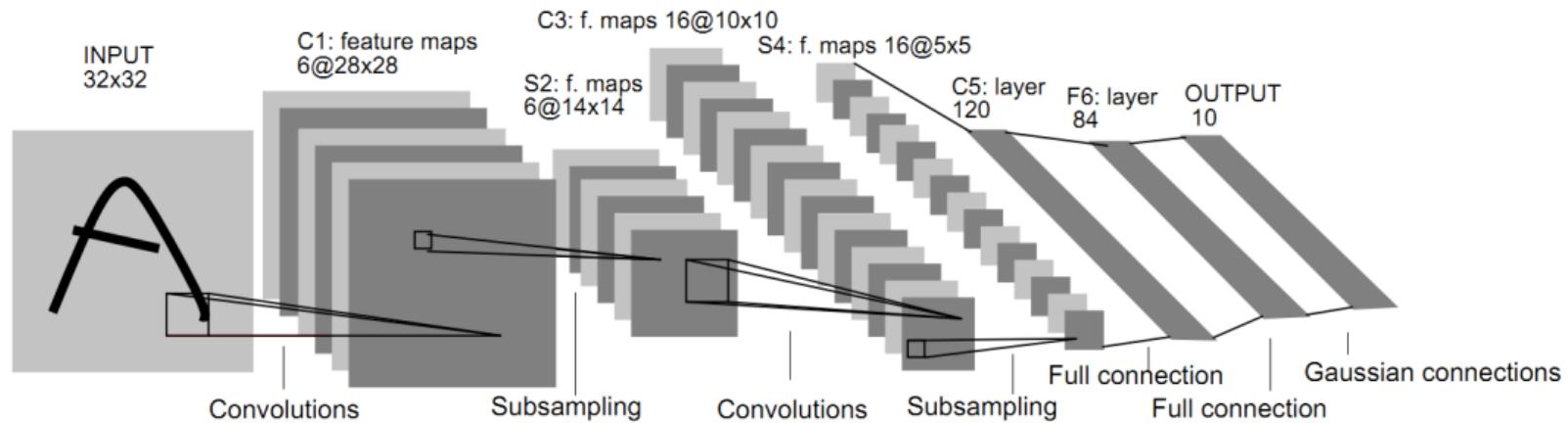
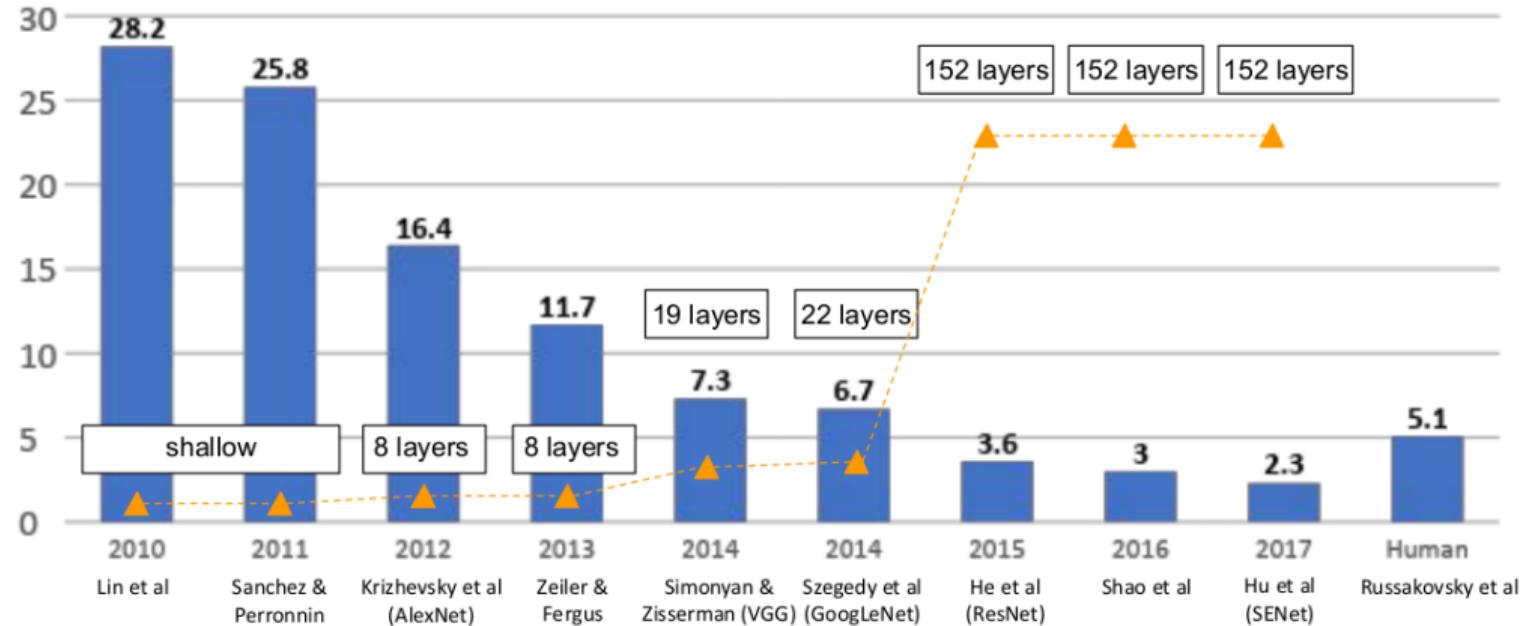


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

IMAGENET LARGE SCALE VISUAL RECOGNITION



IMAGENET DATA

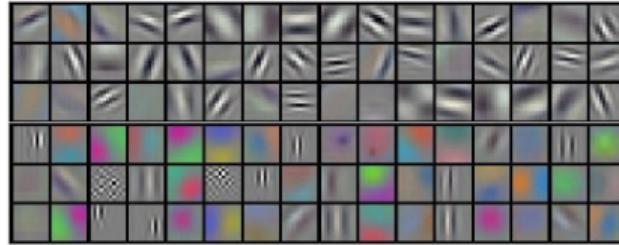


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU

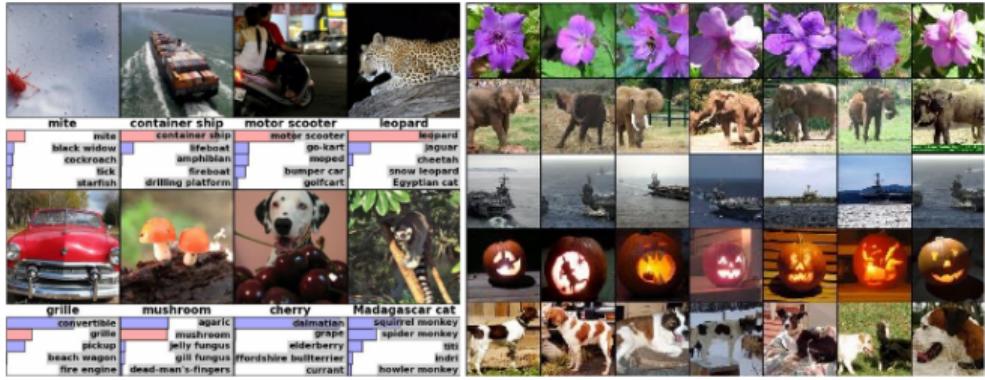


Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

IMAGENET DATA

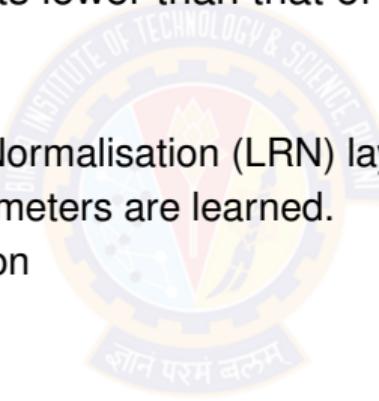


The ImageNet set that was used has ~1.2 million images and 1000 classes

Accuracy is measured as top-5 performance:
Correct prediction if the true label matches one of the top 5 predictions of the model

ALEXNET

- AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012.[3] The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.
- Details/Retrospectives:
 - ▶ first use of ReLU
 - ▶ used Local Response Normalisation (LRN) layers (not common anymore)
 - ▶ Approx. 60 million parameters are learned.
 - ▶ heavy data augmentation
 - ▶ dropout 0.5
 - ▶ batch size 128
 - ▶ SGD Momentum 0.9
 - ▶ Learning rate $1e - 2$, reduced by 10 manually when val accuracy plateaus
 - ▶ L2 weight decay $5e - 4$
 - ▶ 7 CNN ensemble: 18.2% – > 15.4%



ALEXNET ARCHITECTURE

227x227x3 INPUT

55x55x96 CONV1 : 96 11x11 filters at stride 4, pad 0

27x27x96 MAX POOL1 : 3x3 filters at stride 2

27x27x96 NORM1 : Normalization layer

27x27x256 CONV2 : 256 5x5 filters at stride 1, pad 2

13x13x256 MAX POOL2 : 3x3 filters at stride 2

13x13x256 NORM2 : Normalization layer

13x13x384 CONV3 : 384 3x3 filters at stride 1, pad 1

13x13x384 CONV4 : 384 3x3 filters at stride 1, pad 1

13x13x256 CONV5 : 256 3x3 filters at stride 1, pad 1

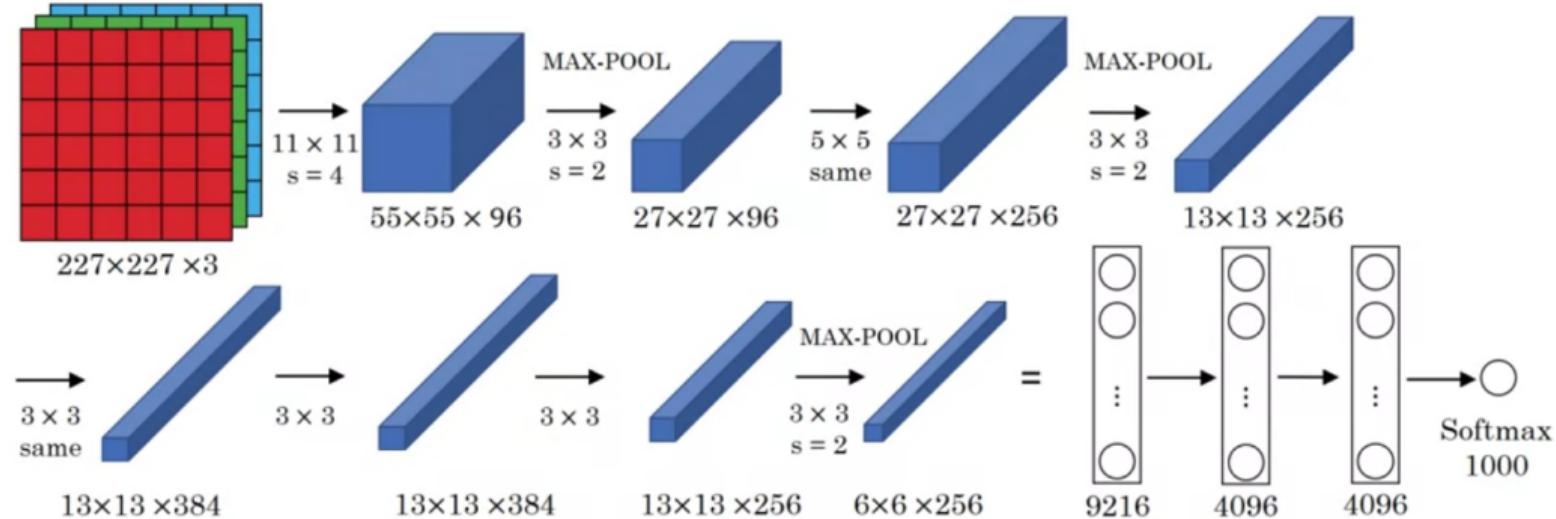
6x6x256 MAX POOL3 : 3x3 filters at stride 2

4096 FC6 : 4096 neurons

4096 FC7 : 4096 neurons

1000 FC8 : 1000 neurons (class scores)

ALEXNET

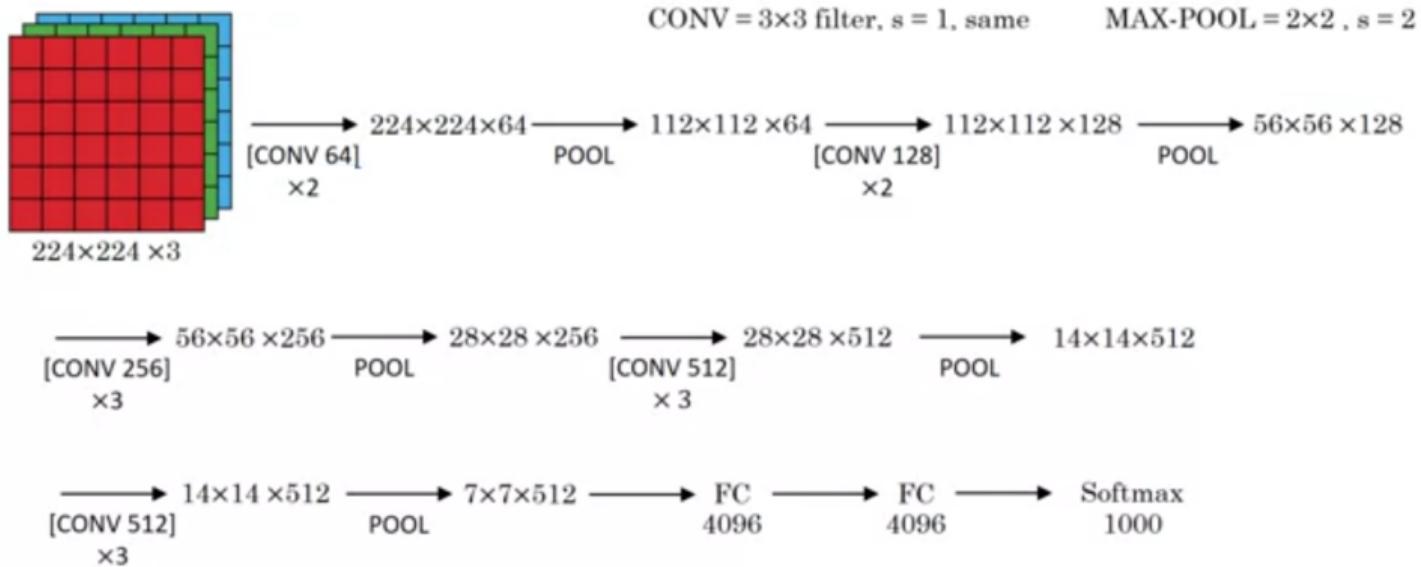


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG 16

- VGG stands for Visual Geometry Group with 16 Layers
- Plug and play in Caffe
- Deeper the better
- Details
 - ▶ ILSVRC'14 2nd in classification, 1st in localization
 - ▶ Similar training procedure as Krizhevsky 2012
 - ▶ Approx. 138 million parameters are learned.
 - ▶ No Local Response Normalisation (LRN)
 - ▶ Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
 - ▶ Use ensembles for best results
 - ▶ All convolutions with a 3×3 kernel
 - ▶ All max-pooling layers with a 2×2 kernel
 - ▶ FC7 features generalize well to other tasks

VGG 16



[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

Andrew Ng

NETWORK IN NETWORK

- A convolution kernel can be thought of as a generalized linear model (GLM).
- Using a sophisticated non-linear function approximator like MLP, enhance the abstraction ability of the local model.
- Replace GLM by **micro network**.
- Replace the MLP micro structure via convolutions.
- Replace fully connected layers in the last layers by global average pooling.

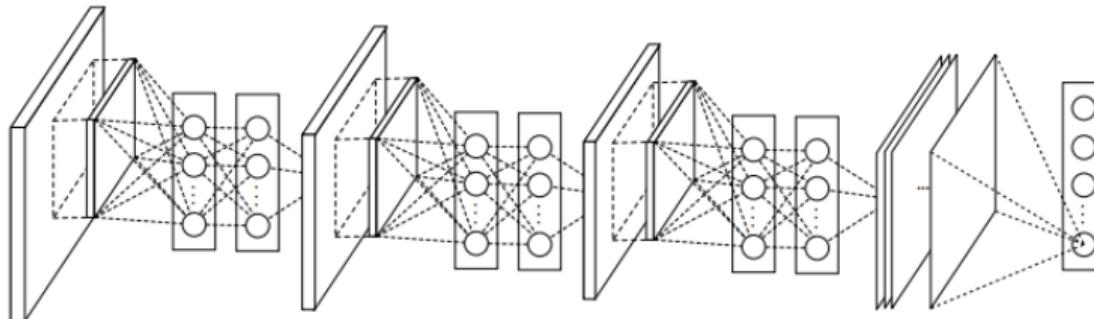
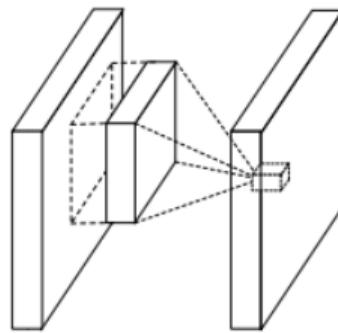


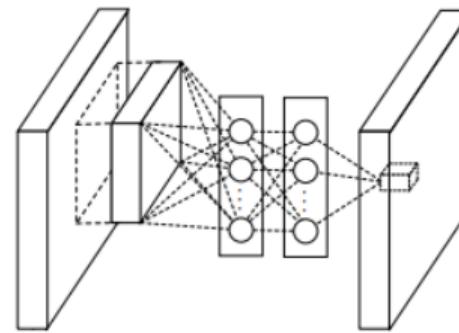
Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

NETWORK IN NETWORK

Using micro-networks allow to extract more sophisticated features and may need fewer extractors and can avoid learning too simple or redundant features.



(a) Linear convolution layer



(b) Mlpconv layer

Figure 1: Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (we choose the multilayer perceptron in this paper). Both layers map the local receptive field to a confidence value of the latent concept.

NETWORK IN NETWORK

Fully connected layers have a lot of learnable parameters and may cause overfitting.
Replace FC by global average pooling for better generalization.

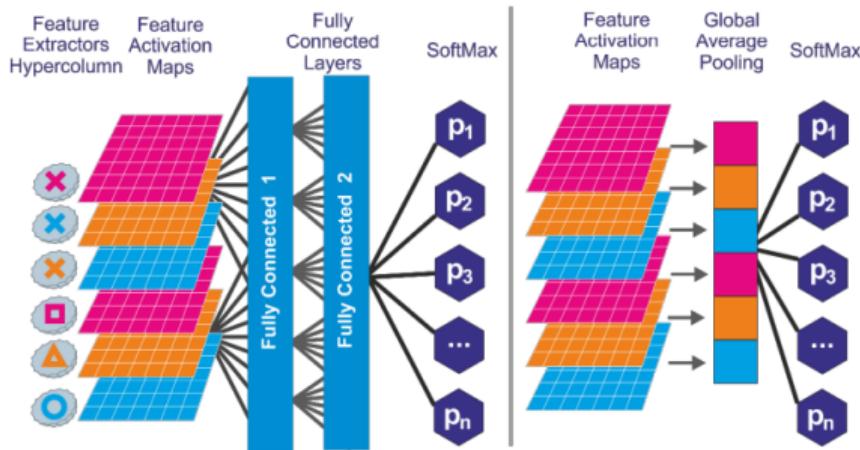


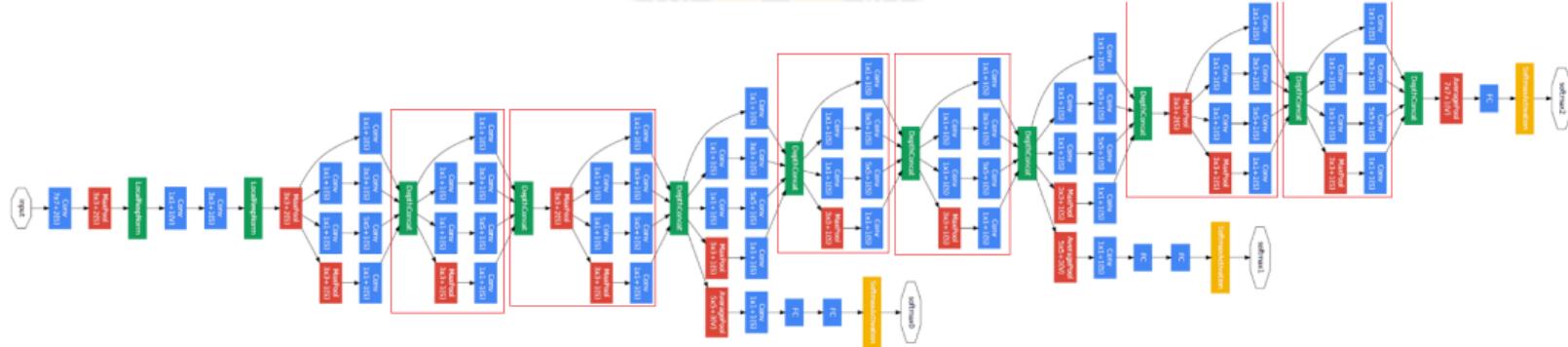
Figure 16: Global average pooling layer replacing the fully connected layers. The output layer implements a Softmax operation with p_1, p_2, \dots, p_n the predicted probabilities for each class.

Figure Source: Singh, Anshuman Vikram. "Content-based image retrieval using deep learning." (2015)

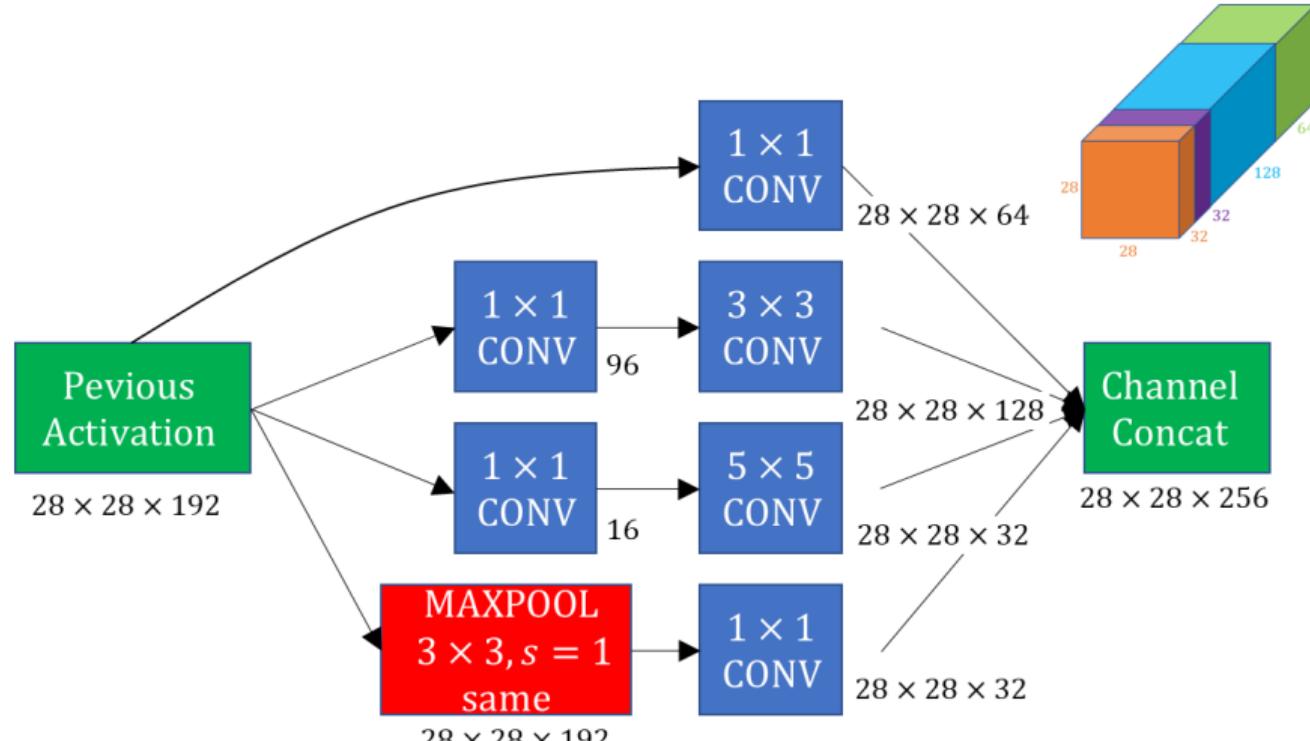
GOOGLENET / INCEPTION NET

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers with weights
- Only 5 million parameters (12x less than AlexNet and 27x less than VGG-16)
- Inception Module - convolutional “blocks” – efficient
 - ▶ Design a good local network topology (network within a network) and then stack these modules on top of each other.
- Linear layers at the end
- Max pooling in between, multiple Conv layers between pooling
- Great ideas for data augmentation
- Deeper networks, with computational efficiency
- No FC layers.
- After the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer.

INCEPTION NET



INCEPTION NET MODULE



32 filters, $1 \times 1 \times 192$

Andrew Ng

INCEPTION NET MODULE

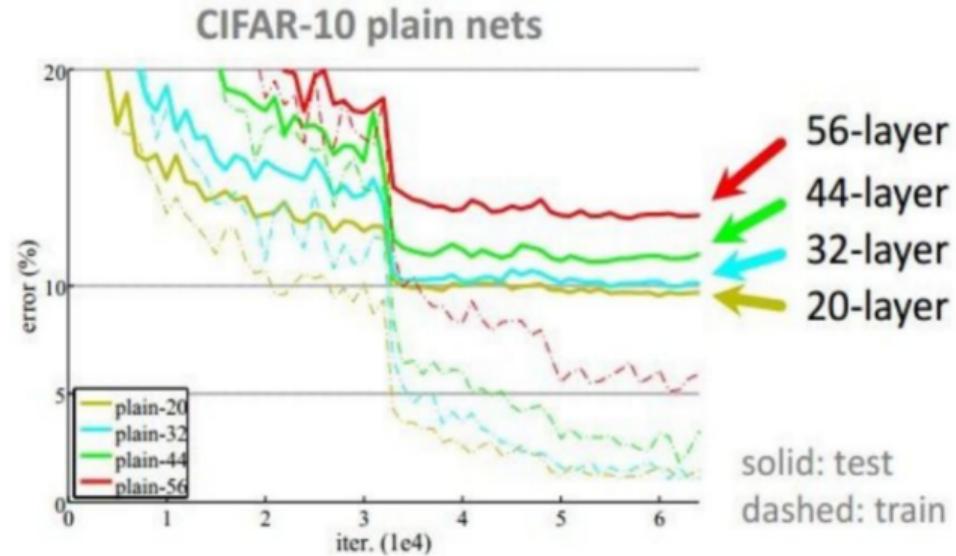
- Apply parallel filter operations on the input from previous layer.
- Multiple receptive field sizes for convolution (1×1 , 3×3 , 5×5) and Pooling operation (3×3).
- Concatenate all filter outputs together channel-wise.
- Very expensive compute
- 1×1 convolutions to reduce feature channel size.
 - ▶ Assume an input of $W \times H \times 64$.
 - ▶ Each filter has size $1 \times 1 \times 64$, and performs a 64-dimensional dot product.
 - ▶ Preserves spatial dimensions, reduces depth.
 - ▶ Projects depth to lower dimension (combination of feature maps).

RESIDUAL NET (RESNET)

- Very deep networks using residual connections
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15.
- Add residual connections between convolution blocks
- Easy gradient flow for deep models
- Enable very deep models (from 20 layers to 150 and even 1000 layers!)
- No linear layers except classifier.
- Solve the problems of vanishing gradients using Residual connections.

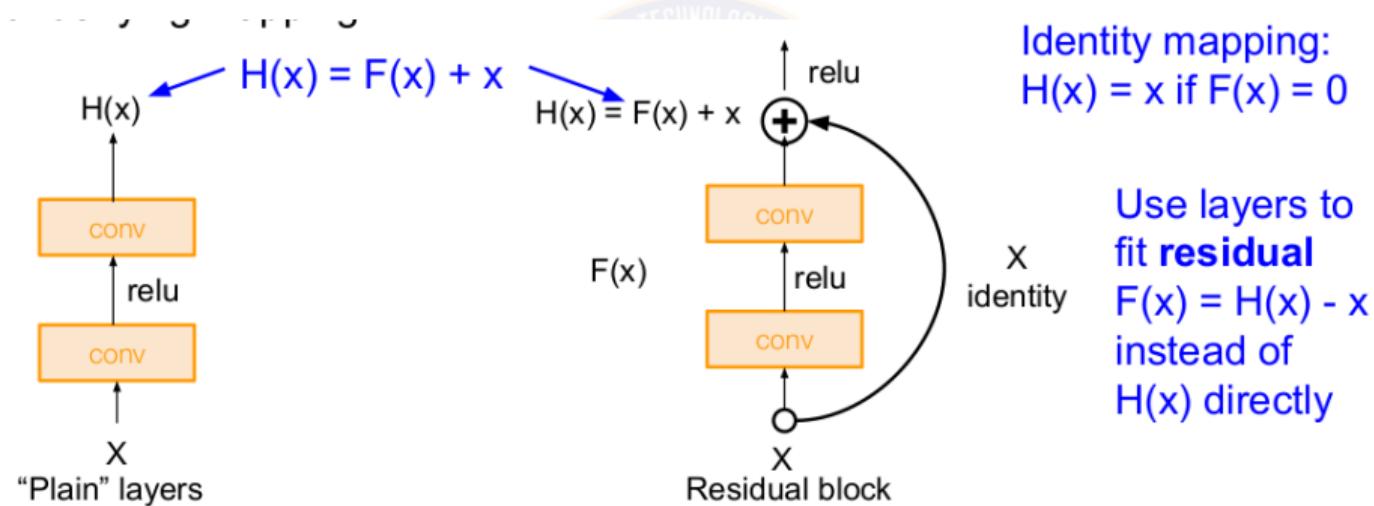
RESNET BLOCK

- What happens when we continue stacking deeper layers on a convolutional neural network?
- The deeper model performs worse, but it's not caused by overfitting.

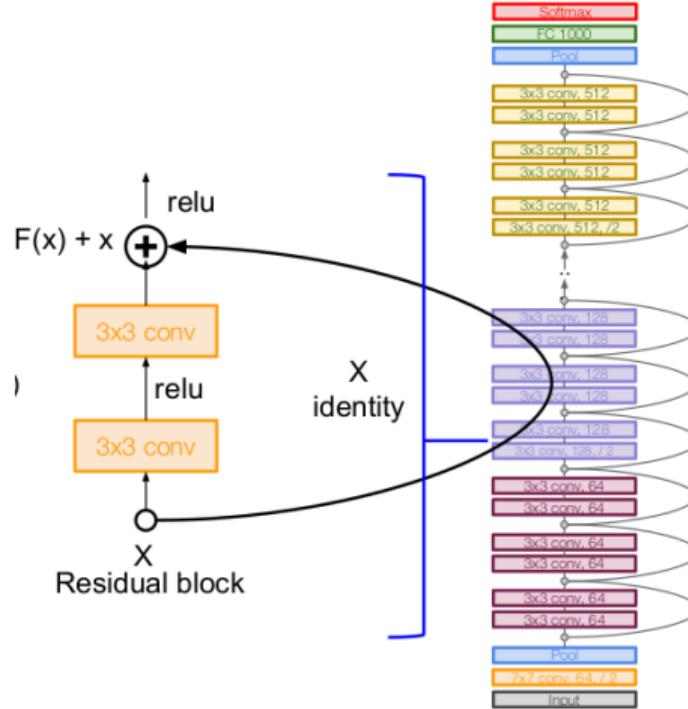


RESNET BLOCK

- Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.



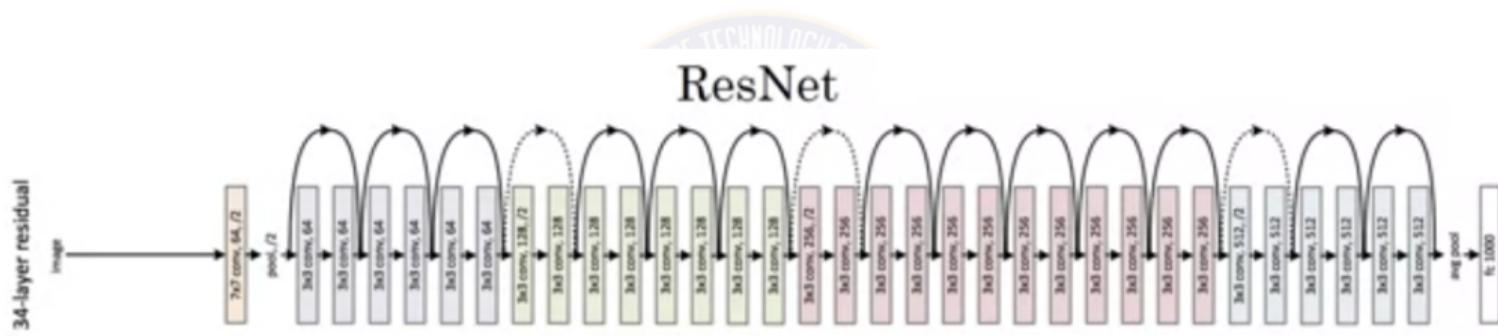
RESNET



RESNET MODULES

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

RESNET



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

RESNET ARCHITECTURE

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2.
Reduce the activation volume by half.
- Additional conv layer at the beginning (stem).
- No FC layers at the end (only FC 1000 to output classes)
- Total depths of 18, 34, 50, 101, or 152 layers for ImageNet.
- Training ResNet in practice:
 - ▶ Batch Normalization after every CONV layer
 - ▶ Xavier initialization from He et al.
 - ▶ SGD + Momentum (0.9)
 - ▶ Learning rate: 0.1, divided by 10 when validation error plateaus
 - ▶ Mini-batch size 256
 - ▶ Weight decay of 1e-5
 - ▶ No dropout used

EFFICACY OF RESNET

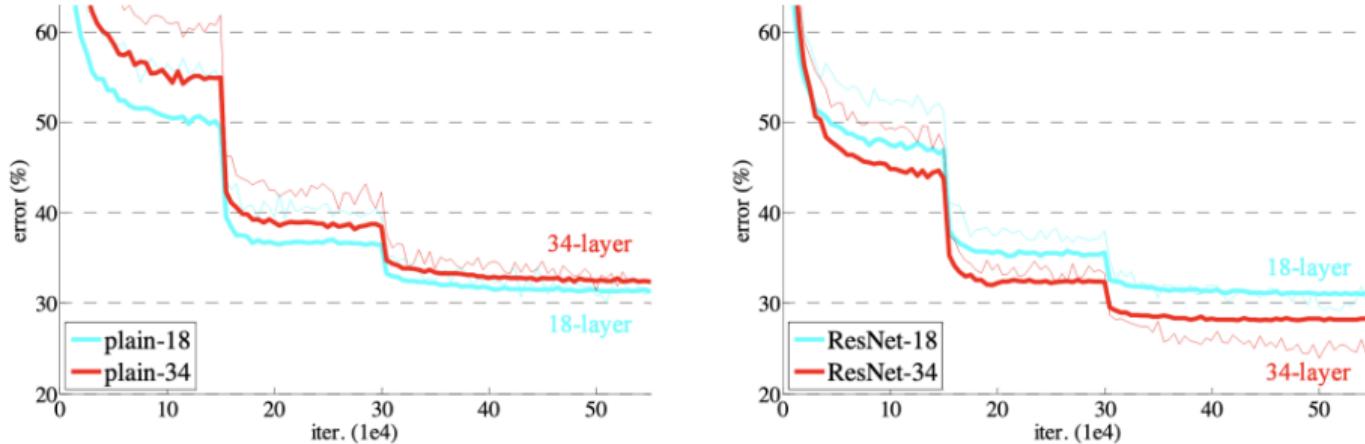
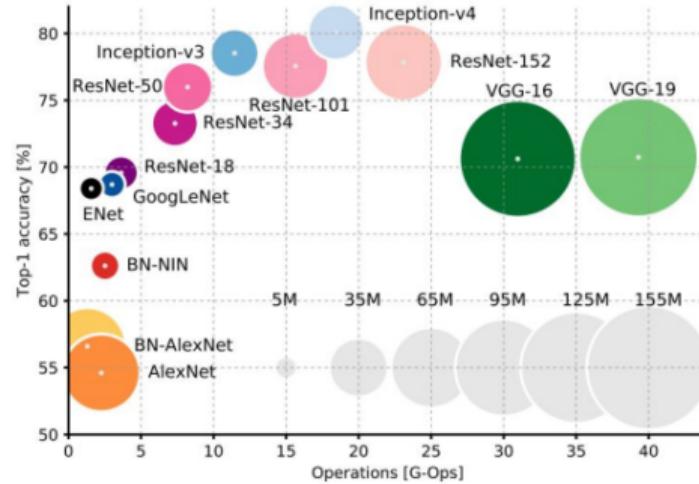
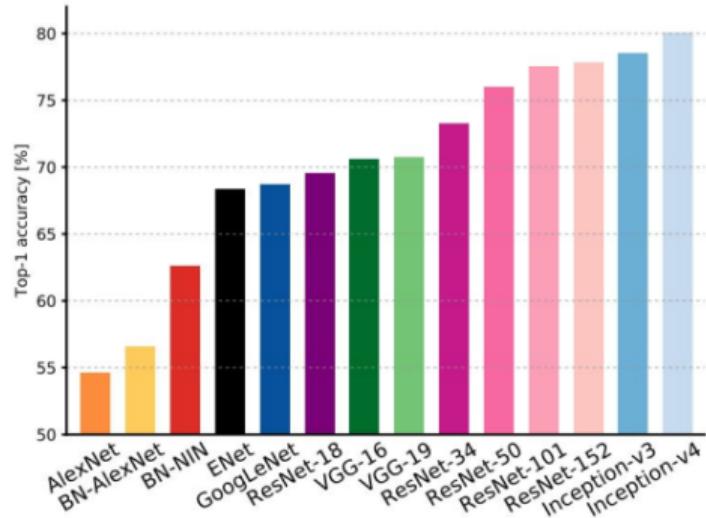


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

COMPARING COMPLEXITY



COMPARING COMPLEXITY

- Inception-v4: Resnet + Inception – more accuracy
- VGG: most parameters, most operations
- GoogLeNet: most efficient
- AlexNet: Smaller compute, still memory heavy, lower accuracy
- ResNet: Moderate efficiency depending on model, highest accuracy

IN THIS SEGMENT

1 CNN ARCHITECTURES

2 TRANSFER LEARNING

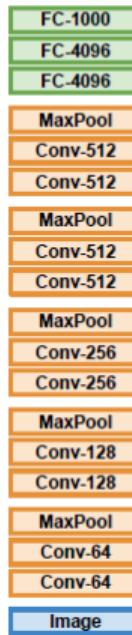
3 FURTHER READING



TRANSFER LEARNING

Transfer Learning with CNNs

1. Train on Imagenet

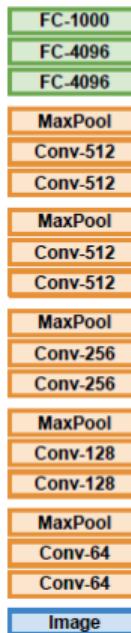


Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

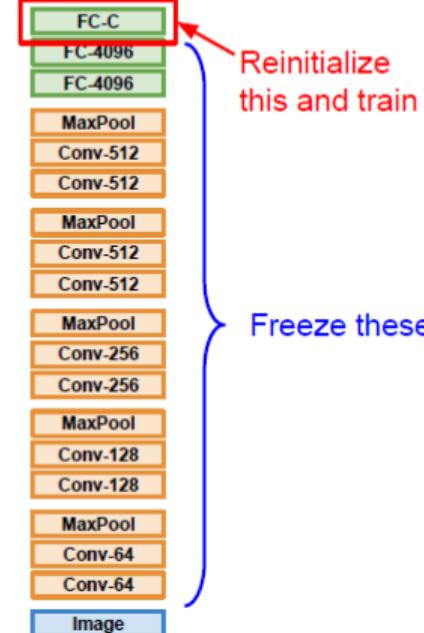
TRANSFER LEARNING

Transfer Learning with CNNs

1. Train on Imagenet



2. Small Dataset (C classes)

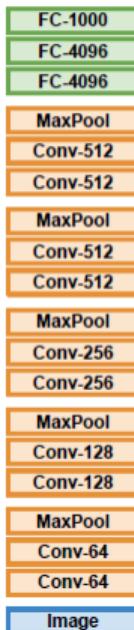


Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

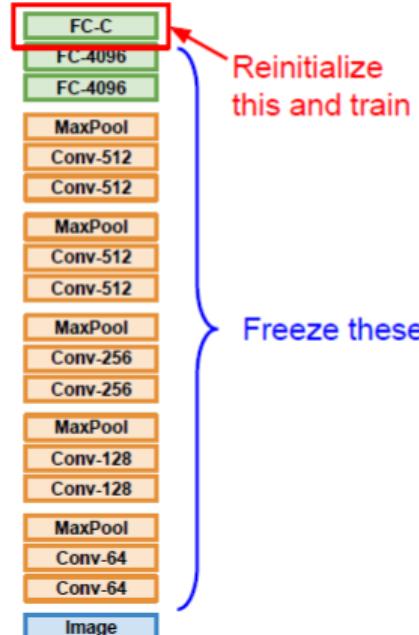
TRANSFER LEARNING

Transfer Learning with CNNs

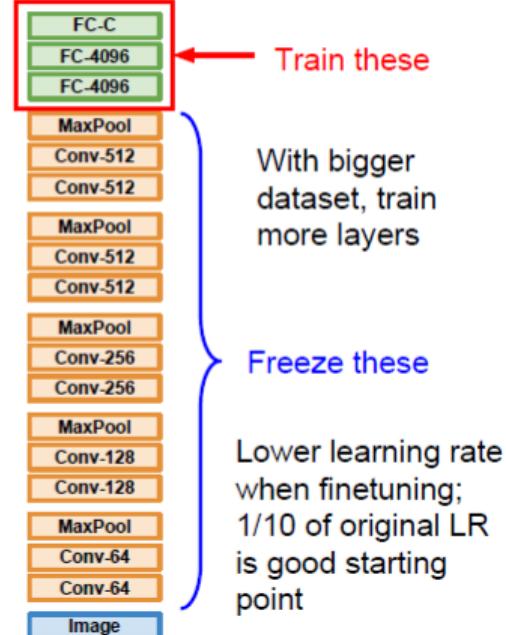
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset



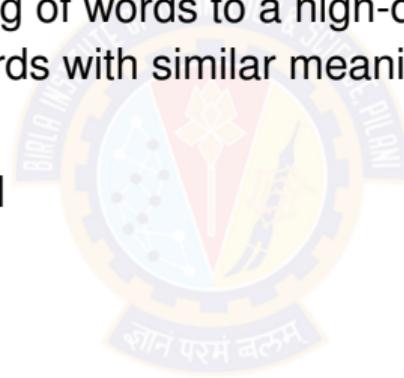
TRANSFER LEARNING FOR IMAGE DATA

- Use a deep learning model that is pre-trained on large dataset like ImageNet or MS Coco.
- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model



TRANSFER LEARNING FOR TEXT DATA

- Embedding is the mapping of words to a high-dimensional continuous vector space where different words with similar meanings have similar vector representations.
- Google's word2vec Model
- Stanford's GloVe Model
- FastText
- Gensim

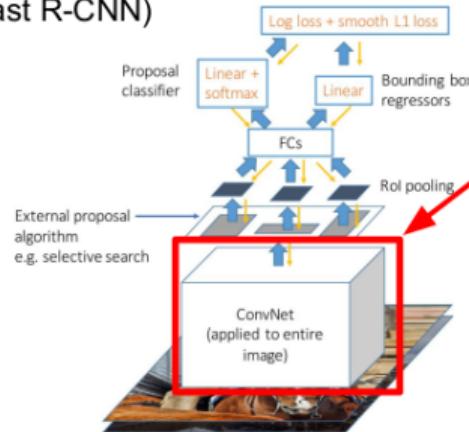


TRANSFER LEARNING - WHEN TO USE?

- You need a lot of data if you want to train/use CNNs / RNNs.
- Task A and Task B have the same type of input. Eg: Input is images for both tasks.
- We have lot of data for training Task A and relatively low data for training in Task B.
- Low level features obtained from Task A could be more helpful for learning Task B.

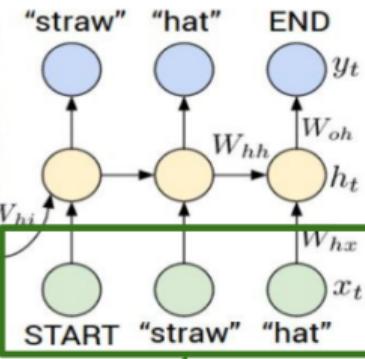
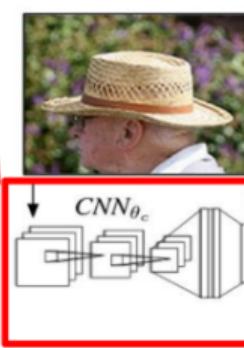
TRANSFER LEARNING EXAMPLE

Object Detection
(Fast R-CNN)



CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



Word vectors pretrained
with word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

IN THIS SEGMENT

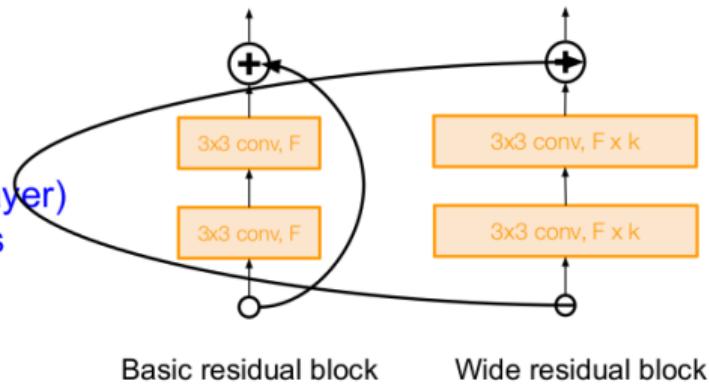
- 1 CNN ARCHITECTURES
- 2 TRANSFER LEARNING
- 3 FURTHER READING



Wide Residual Networks

[Zagoruyko et al. 2016]

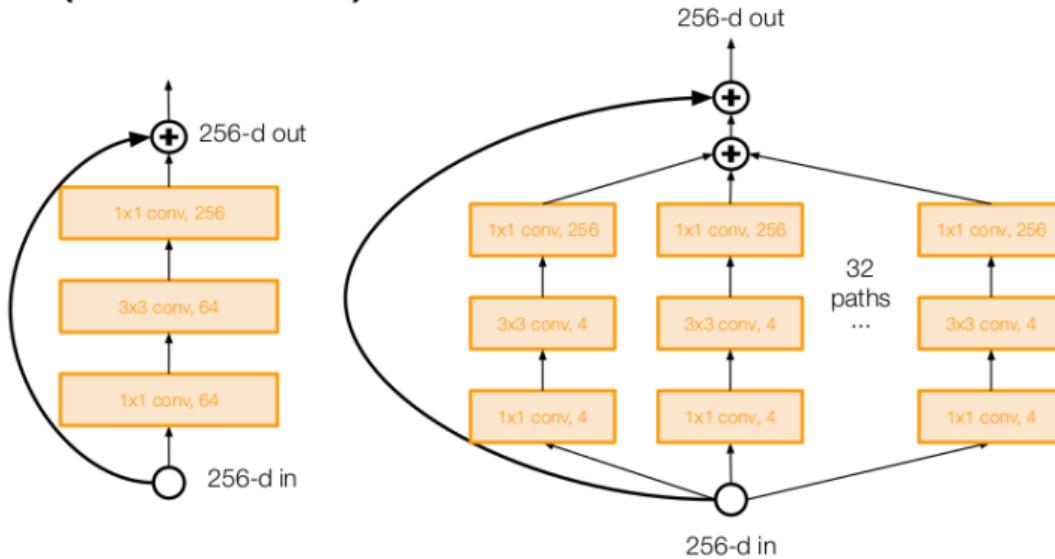
- Argues that residuals are the important factor, not depth
- Use wider residual blocks ($F \times k$ filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

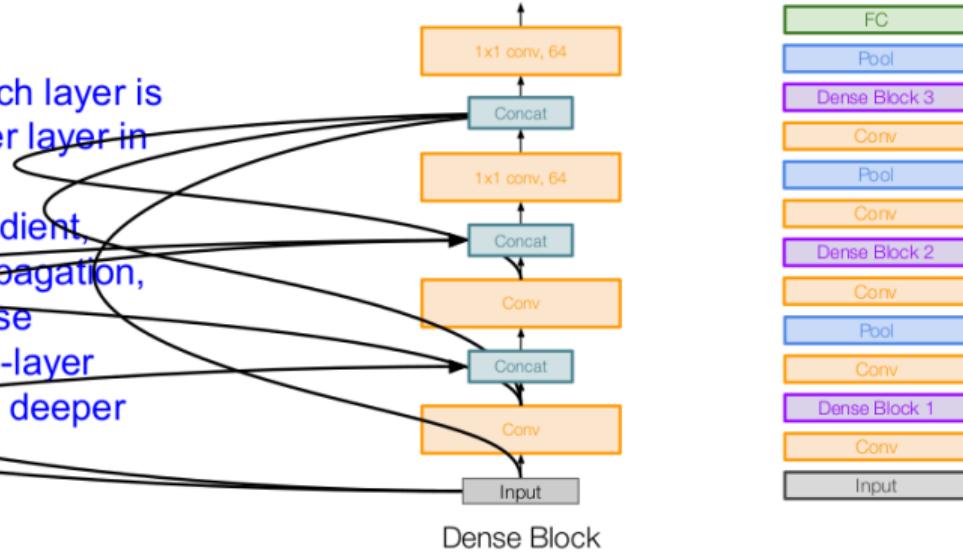
- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

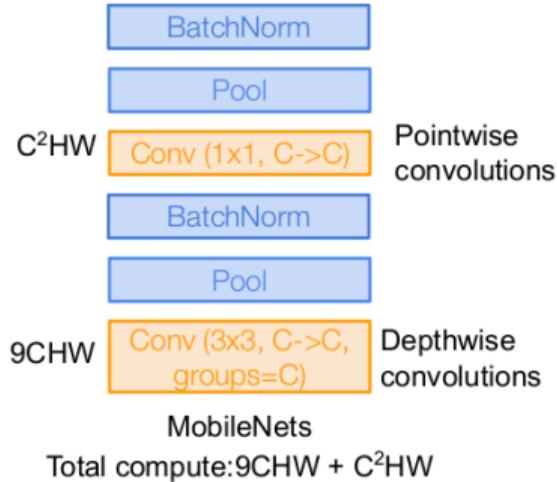
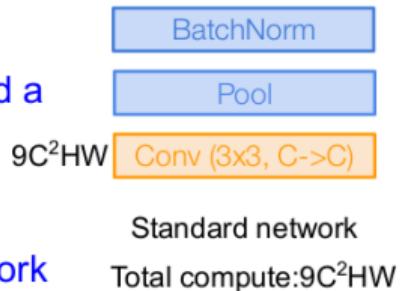
- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

[Howard et al. 2017]

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1×1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al., CVPR 2018



EfficientNet: Smart Compound Scaling

[Tan and Le, 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

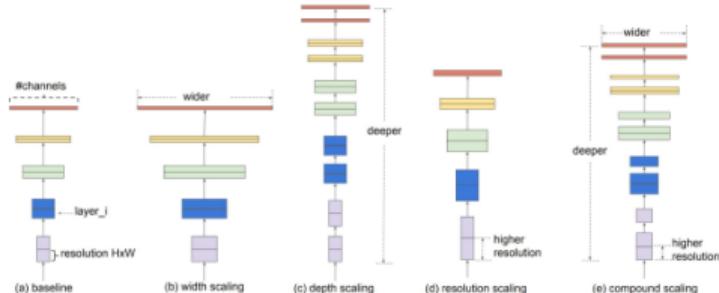
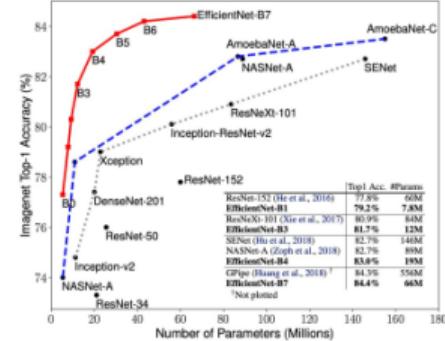
$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



REFERENCES

Refer Chapter 8 of Dive into Deep Learning book.

LENET5 : Yann Lecun and Leon Bottou and Yoshua Bengio and Patrick Haffner,
Gradient-based learning applied to document recognition, IEEE-1998

ALEXNET : Imagenet classification with deep convolutional neural networks,
Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E, In: Advances in
neural information processing systems, NIPS 2012

VGG16 : Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional
networks for large-scale image recognition."

INCEPTION v1 : Going deeper with convolutions, Szegedy, Christian et.al CVPR 2015

RESNET : Deep residual learning for image recognition, He et.al CVPR-2016



Thank You!



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DEEP LEARNING

MODULE # 6 : RECURRENT NEURAL NETWORK [RNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.

ISSUES OF FEEDFORWARD NEURAL NETWORKS

In feedforward and convolutional neural networks

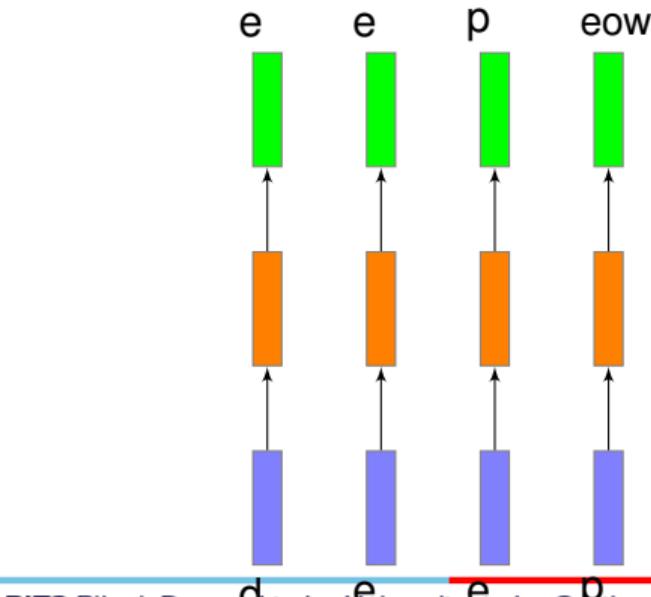
- The size of the input is always fixed.
- Each input to the network is independent of the previous or future inputs.
- The computations, outputs and decisions for two successive inputs / images are completely independent of each other.

ISSUES OF FEEDFORWARD NEURAL NETWORKS

This is not true in many applications.

- The size of the input is not always fixed.
- Successive inputs may not be independent of each other.
- Each network (blue - orange - green structure) is performing the same task – input : character output : character.

Example: Auto-completion.



IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

SEQUENCE LEARNING PROBLEMS

To model a sequence we need

- Process an input or sequence of inputs.
- The inputs may have be dependent.
- We may have to maintain the sequence order.
- Each input corresponds to one time step.
- Keep track of long term dependencies.
- Produce an output or sequence of outputs.
- Supervised Learning.
- Share parameters across the sequences.

SEQUENCE MODEL

Speech recognition



"The quick brown fox jumped over the lazy dog."

Music generation



Sentiment classification

"There is nothing to like
in this movie."



DNA sequence analysis

AGCCCCCTGTGAGGAACTAG



AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec
moi?



Do you want to sing with
me?

Video activity recognition



Running

Name entity recognition

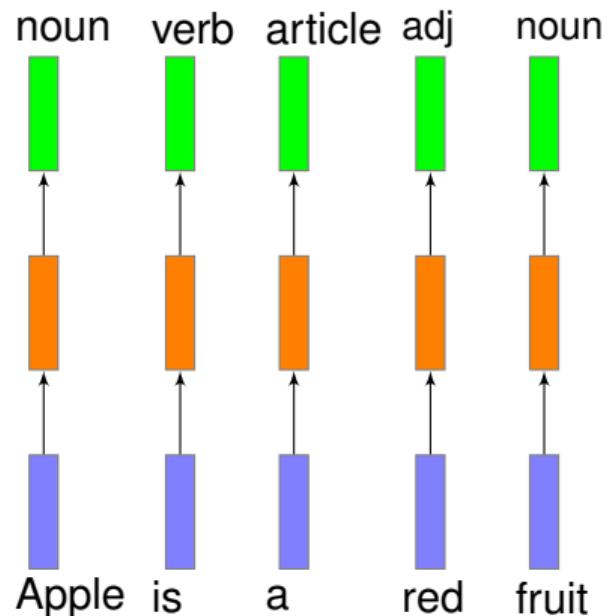
Yesterday, Harry Potter
met Hermione Granger.



Yesterday, Harry Potter
met Hermione Granger.

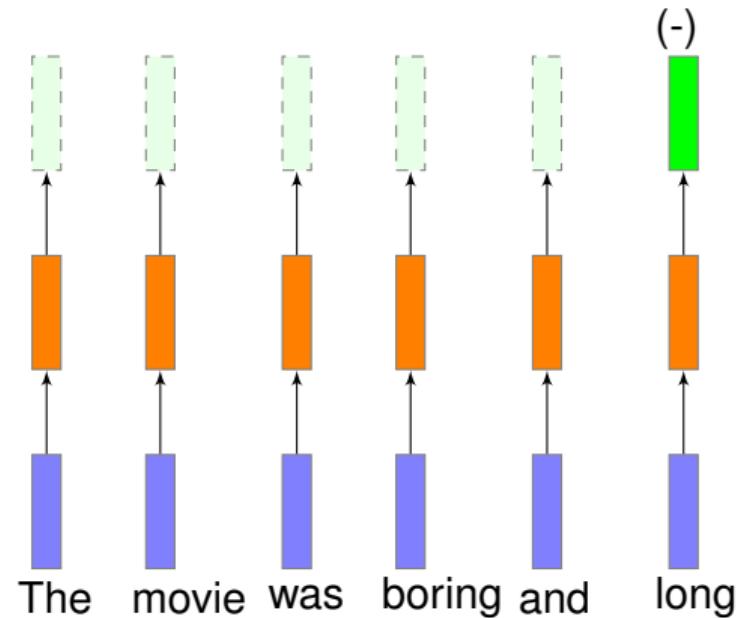
PART OF SPEECH TAGGING

- Task is predicting the part of speech tag (noun, adverb, adjective, verb) of each word in a sentence.
- When we see an adjective we are almost sure, the next word should be a noun.
- The current output depends on the current input as well as the previous input.
- The size of the input is not fixed. Sentences have any number of words.
- An output is produced at end of each time step.
- Each network is performing the same task – input : word, output : tag.



SENTIMENT ANALYSIS

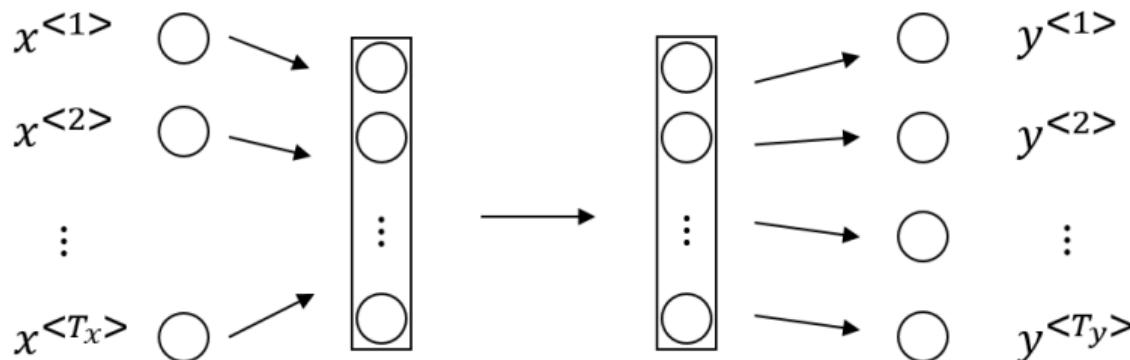
- Task is predicting the sentiment of a whole sentence.
- Input is the entire sequence of inputs.
- An output is **not** produced at end of each time step.
- Each network is performing the same task – input : word, output : polarity +/–.



IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

RECURRENT NEURAL NETWORK (RNN)



Andrew Ng

- Accounts for variable number of inputs.
- Accounts for dependencies between inputs.
- Accounts for variable number of outputs.
- Ensures that the same function executed at each time step.
- The features learned across the inputs at different time step has to be shared.

RNN I

- The function learned at each time step.

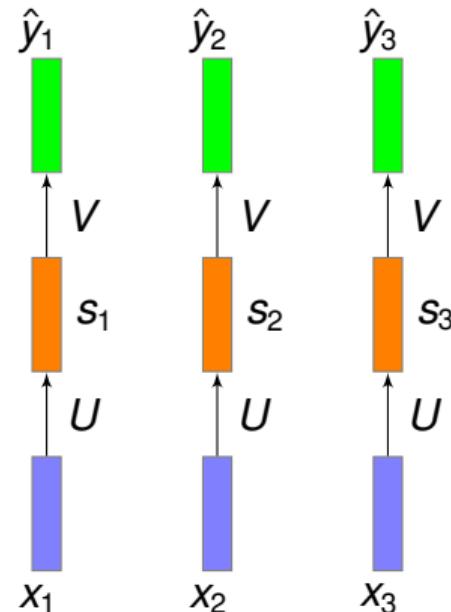
$t = \text{time step}$

$x_t = \text{input at time step } t$

$s_t = \sigma(Ux_t + b)$

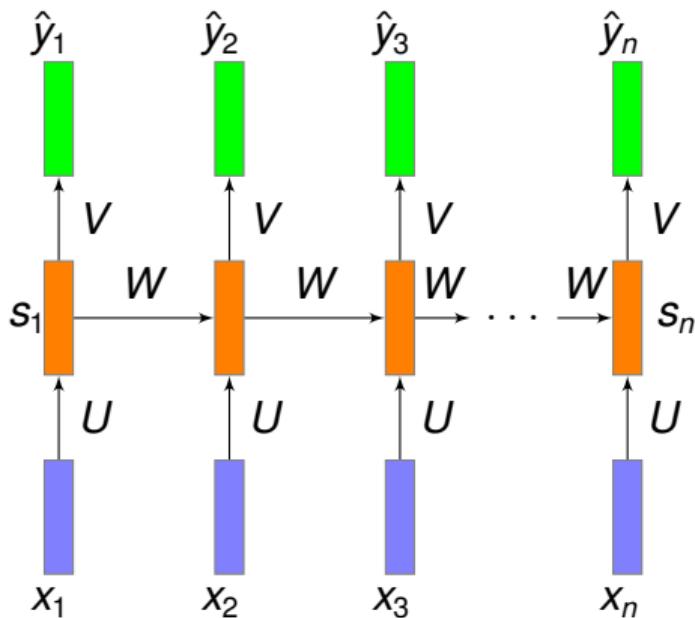
$y_t = g(Vs_t + c)$

- Since the same function has to be executed at each time step we should share the same network i.e., same parameters at each time step.



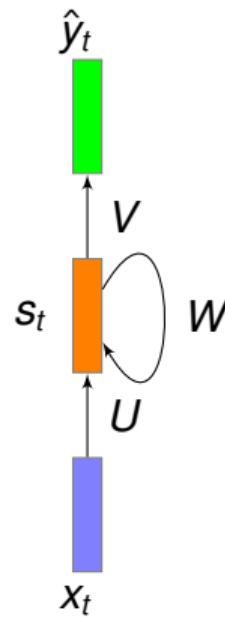
RNN II - UNROLLED NETWORK

- The parameter sharing ensures that
 - ▶ the network becomes invariant to the length of the input.
 - ▶ the number of time steps doesn't matter.
- Create multiple copies of the network and execute them at each timestep.
 - ▶ i.e. create a loop effect.
 - ▶ i.e. add recurrent connection in the network.

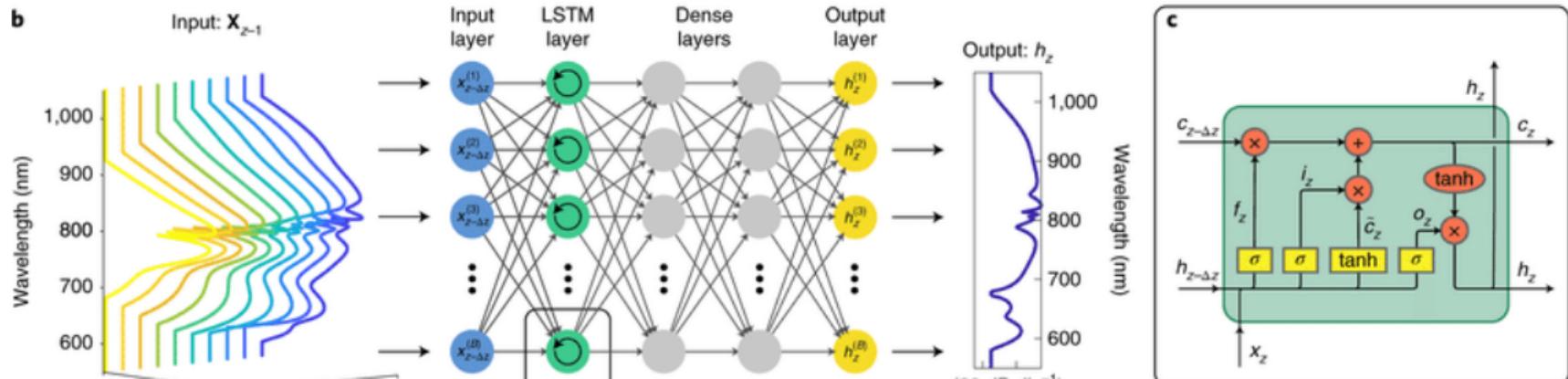
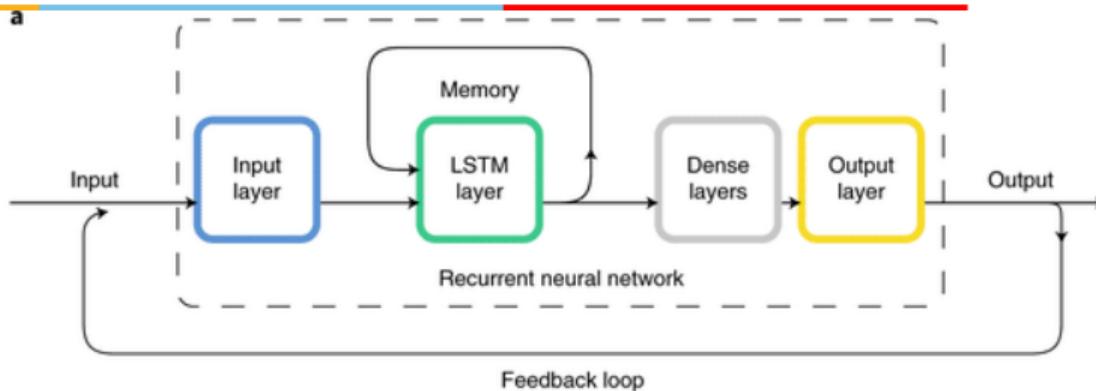


RNN III - ROLLED NETWORK

- Parameter sharing across time steps
- Self loop or recurrent connection



SCHEMATIC OF RNN

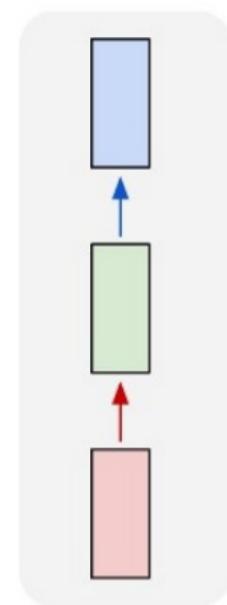


IN THIS SEGMENT

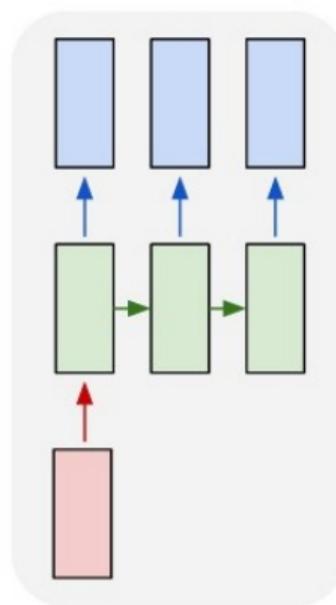
- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

TYPES OF RNN

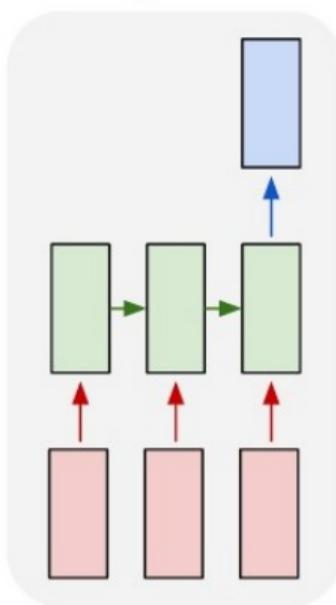
one to one



one to many

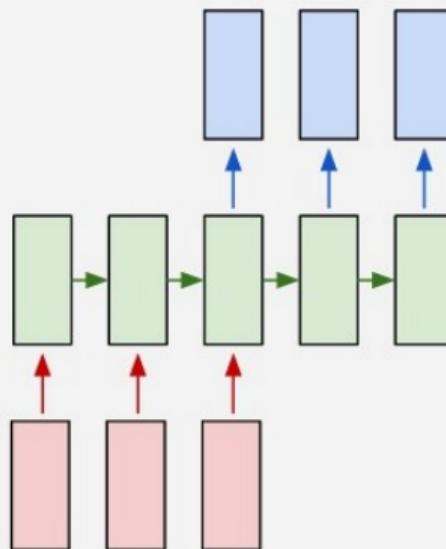


many to one

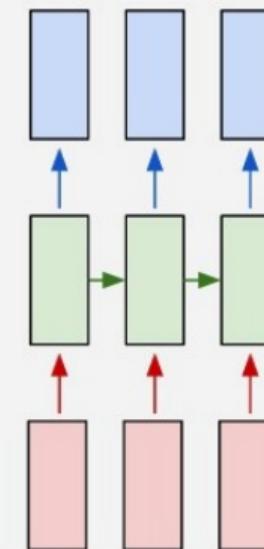


TYPES OF RNN

many to many



many to many



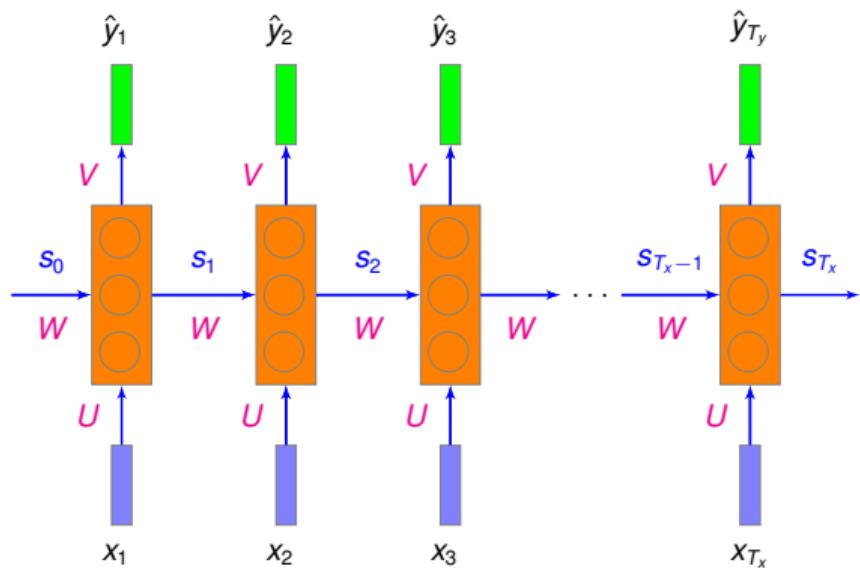
TYPES OF RNN AND APPLICATIONS

- One to one – Generic neural network, Image classification
- One to many – Music generation, Image Captioning
- Many to one – Movie review or Sentiment Analysis
- Many to many – Machine translation
- Synced Many to many – Video classification

IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

FORWARD PROPAGATION IN RNN



- s_t is the **state** of the network at time step t .

$$s_0 = 0$$

$$s_t = \sigma(Ux_t + Ws_{t-1} + b)$$

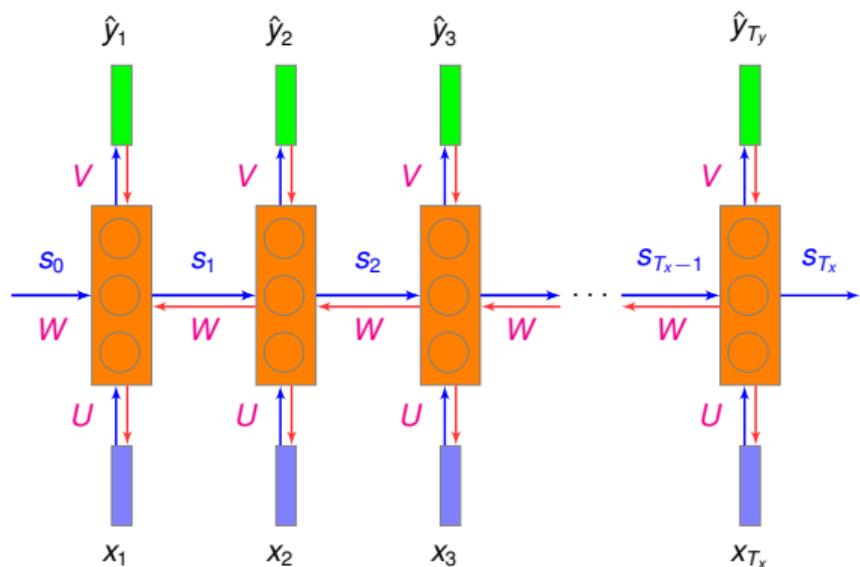
$$\hat{y}_t = g(Vs_t + c)$$

or

$$\hat{y}_t = f(x_t, s_{t-1}, W, U, V, b, c)$$

- The parameters W, U, V, b, c are shared across time steps.

BACK PROPAGATION IN RNN



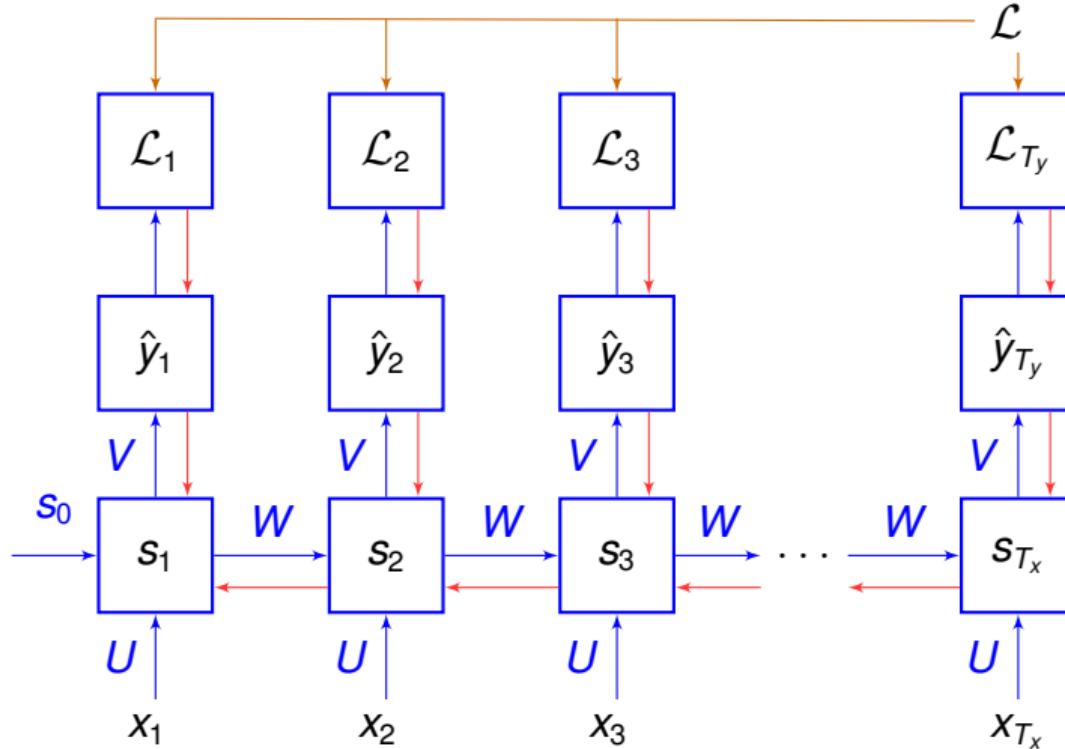
Loss function

$$\mathcal{L}_t(\hat{y}_t, y_t) = \prod_{t=1}^{T_y} P(\hat{y}_t | \hat{y}_{t-1}, \dots, \hat{y}_1)$$

Overall Loss

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}_t(\hat{y}_t, y_t)$$

BACK PROPAGATION THROUGH TIME (BPTT) IN RNN



IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

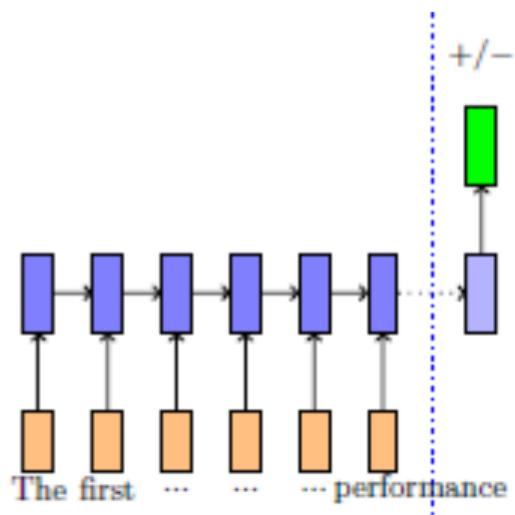
ISSUE OF MAINTAINING STATES

- The old information gets morphed by the current input at each new time step.
- After t steps the information stored at time step $t - k$ (for some $k < t$) gets completely morphed so much that it would be impossible to extract the original information stored at time step $t - k$.
- It is very hard to assign the responsibility of the error caused at time step t to the events that occurred at time step $t - k$.
- Basically depends on the size of memory that is available.

STRATEGY TO MAINTAIN STATES

- Selectively write on the states.
- Selectively read the already written content.
- Selectively forget (erase) some content.

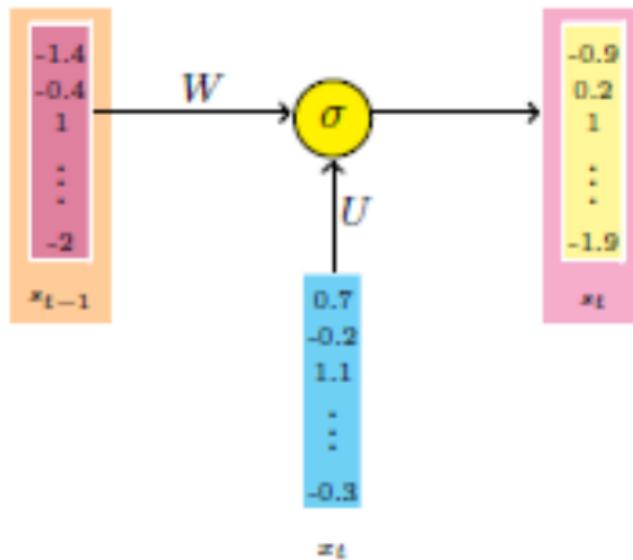
SENTIMENT ANALYSIS



Review: The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

- RNN reads the document from left to right and after every word updates the state.
- By the time we reach the end of the document the information obtained from the first few words is completely lost.
- Ideally we want to
 - ▶ forget the information added by stop words (a, the, etc.).
 - ▶ selectively read the information added by previous sentiment bearing words (awesome, amazing, etc.)
 - ▶ selectively write new information from the current word to the state.

SELECTIVE WRITE



- Recall that in RNNs we use s_{t-1} to compute s_t .

$$s_t = \sigma(Ws_{t-1} + Ux_t + b)$$

SELECTIVE WRITE



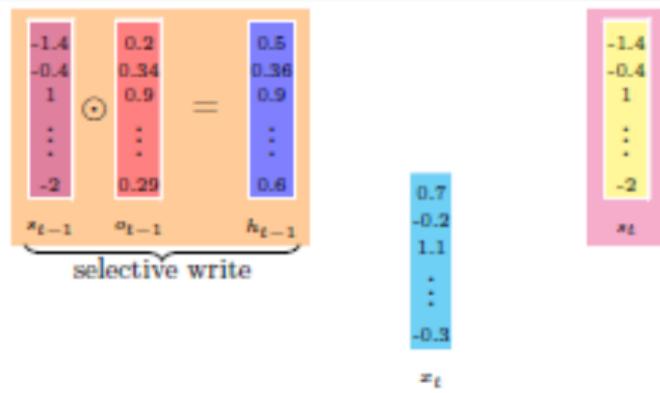
- Introduce a vector o_{t-1} which decides what fraction of each element of s_{t-1} should be passed to the next state.
- Each element of o_{t-1} gets multiplied with the corresponding element of s_{t-1} .
- Each element of o_{t-1} is restricted to be between 0 and 1.
- The RNN has to learn o_{t-1} along with the other parameters (W, U, V).

SELECTIVE WRITE

- Compute o_{t-1} and h_{t-1} as

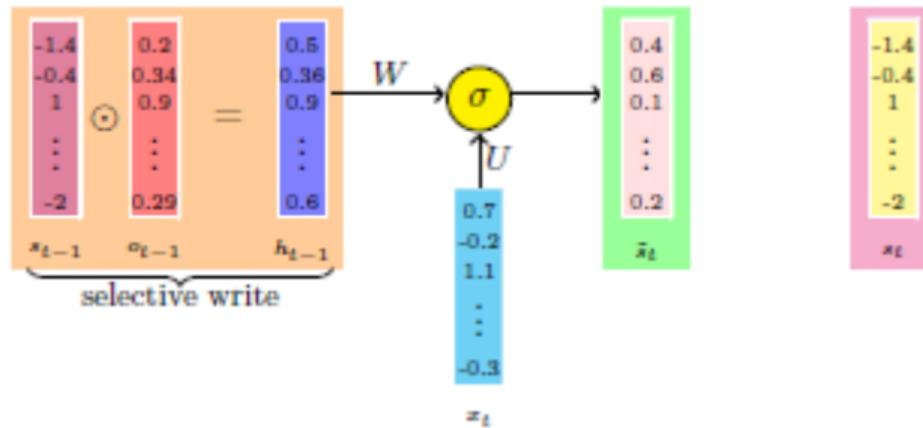
$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$

$$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$$



- The parameters (W_o, U_o, b_o) are learned along with the existing parameters (W, U, V) .
- The sigmoid function ensures that the values are between 0 and 1.
- o_t is called the **output gate** as it decides how much to pass (write) to the next time step.

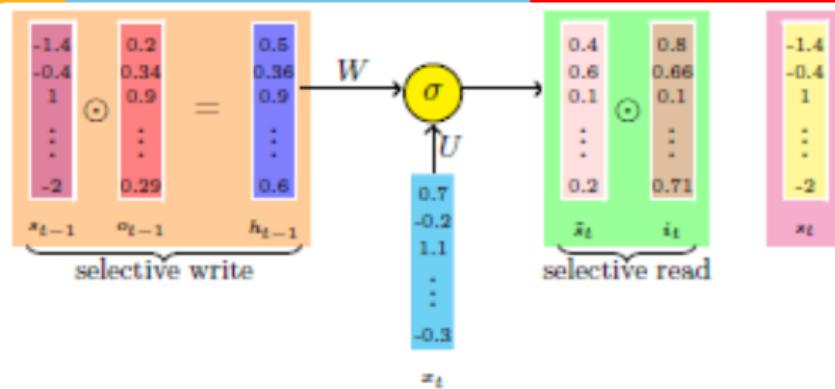
COMPUTE STATE



- h_{t-1} and x_t are used to compute the new state at the next time step.

$$\tilde{s}_t = \sigma(Wh_{t-1} + Ux_t + b)$$

SELECTIVE READ

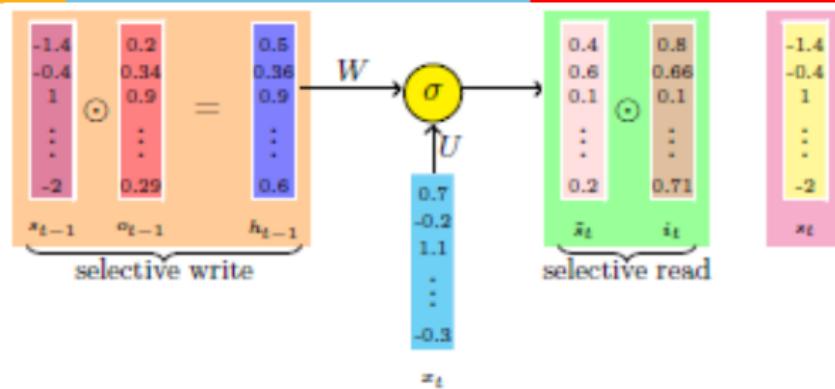


- \tilde{s}_t captures all the information from the previous state h_{t-1} and the current input x_t .
- To do selective read, introduce another gate called the **input gate**.

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$\text{Selectively Read} = i_t \odot \tilde{s}_t$$

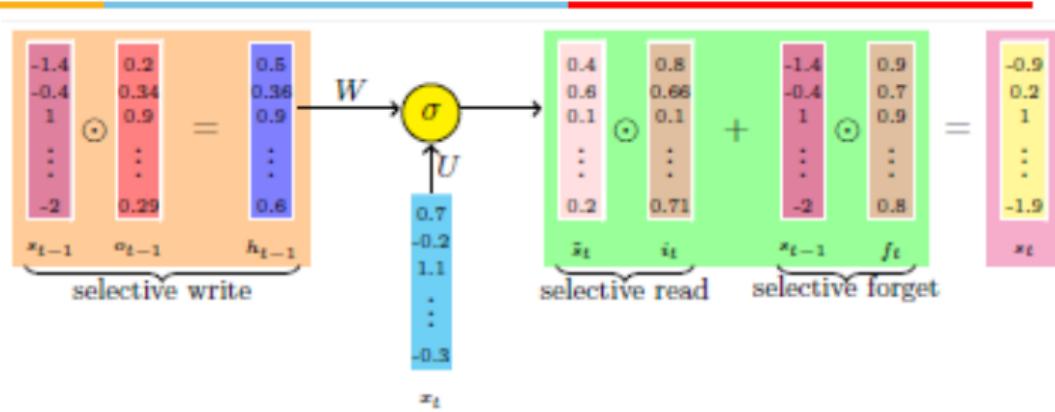
SELECTIVE READ



- \tilde{s}_t captures all the information from the previous state h_{t-1} and the current input x_t .
- To do selective read, introduce another gate called the **input gate**.

$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$

SELECTIVE FORGET

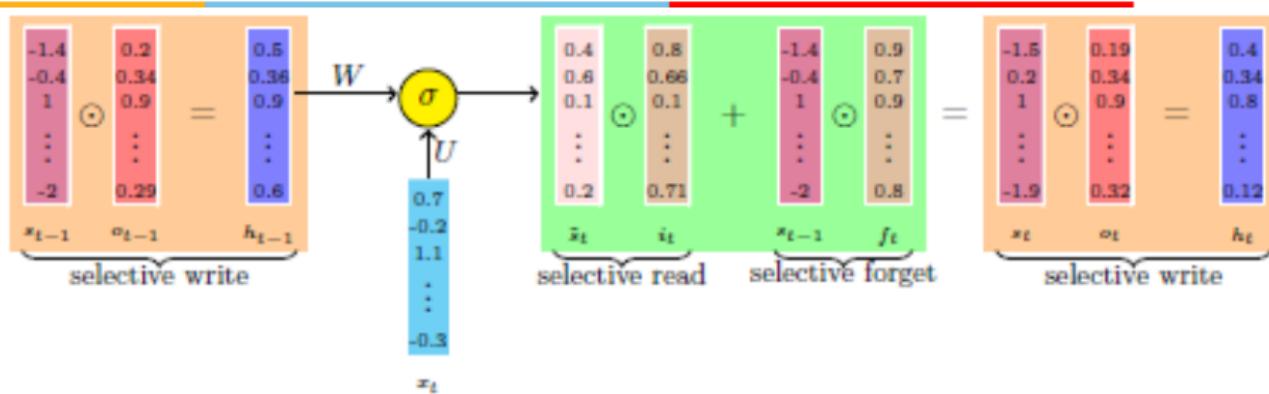


- To do selective forget, introduce another gate called the **forget gate**.

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

FULL LSTM



- 3 gates

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

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

- 3 states

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \sigma(s_t)$$

$$\hat{y}_t = g(V s_t + c)$$

IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

LONG SHORT TERM MEMORY UNIT (LSTM)

- Another representation
- 3 gates are used – Update gate Γ_u , Forget gate Γ_f and Output gate Γ_o .

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u])$$

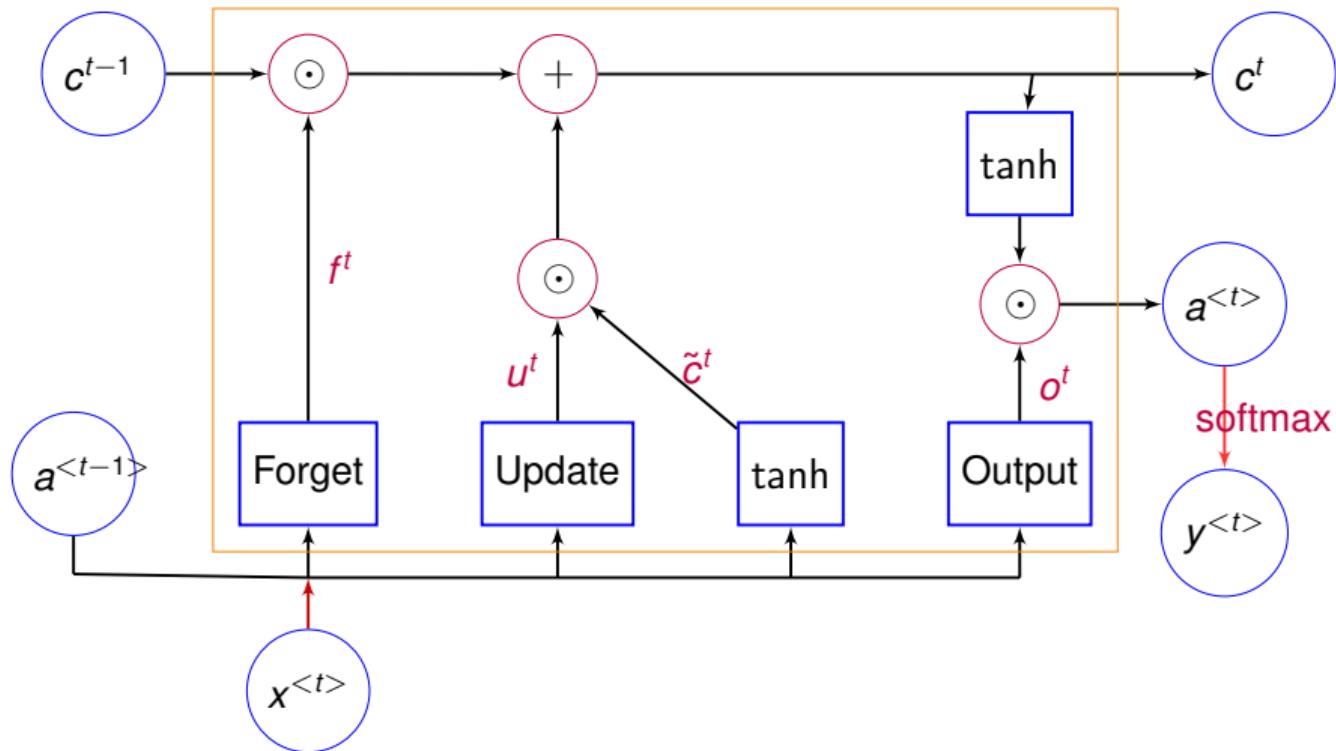
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f])$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o])$$

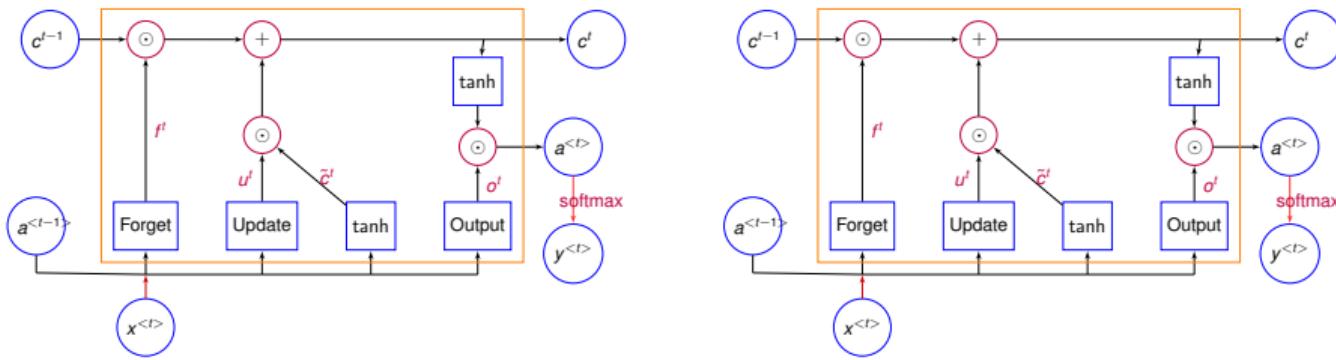
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

LSTM



LSTM



IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

GATED RECURRENT UNIT (GRU)

- Introduce a memory cell $c^{<t>} = a^{<t>}.$
- Candidate for replacing $c^{<t>}$ is given as $\tilde{c}^{<t>}.$
- The decision whether to update $c^{<t>}$ with $\tilde{c}^{<t>}$ is given by the **update gate** $\Gamma_u.$ Γ_u takes the value of 0 or 1.

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

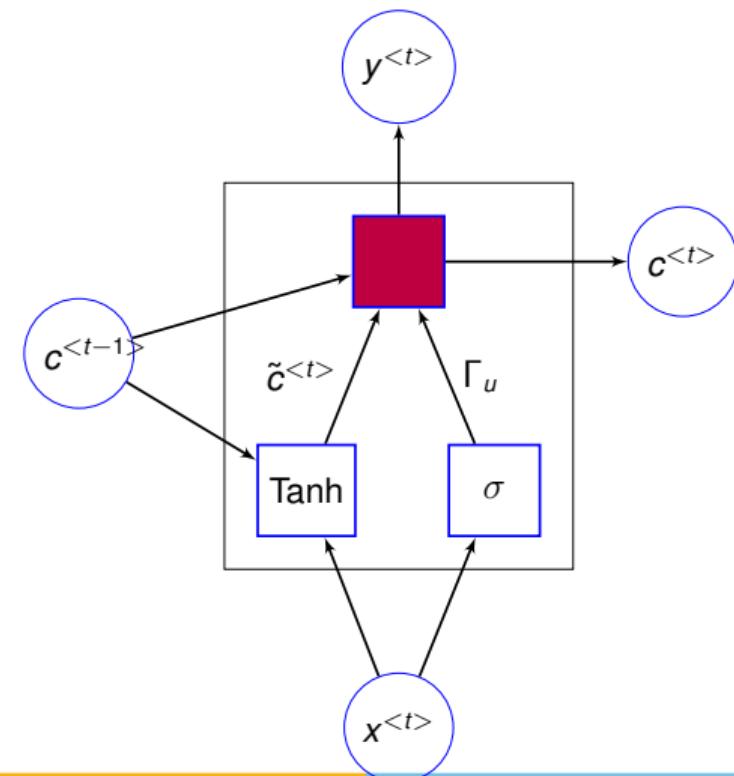
$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>} + b_u])$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>} + b_r])$$

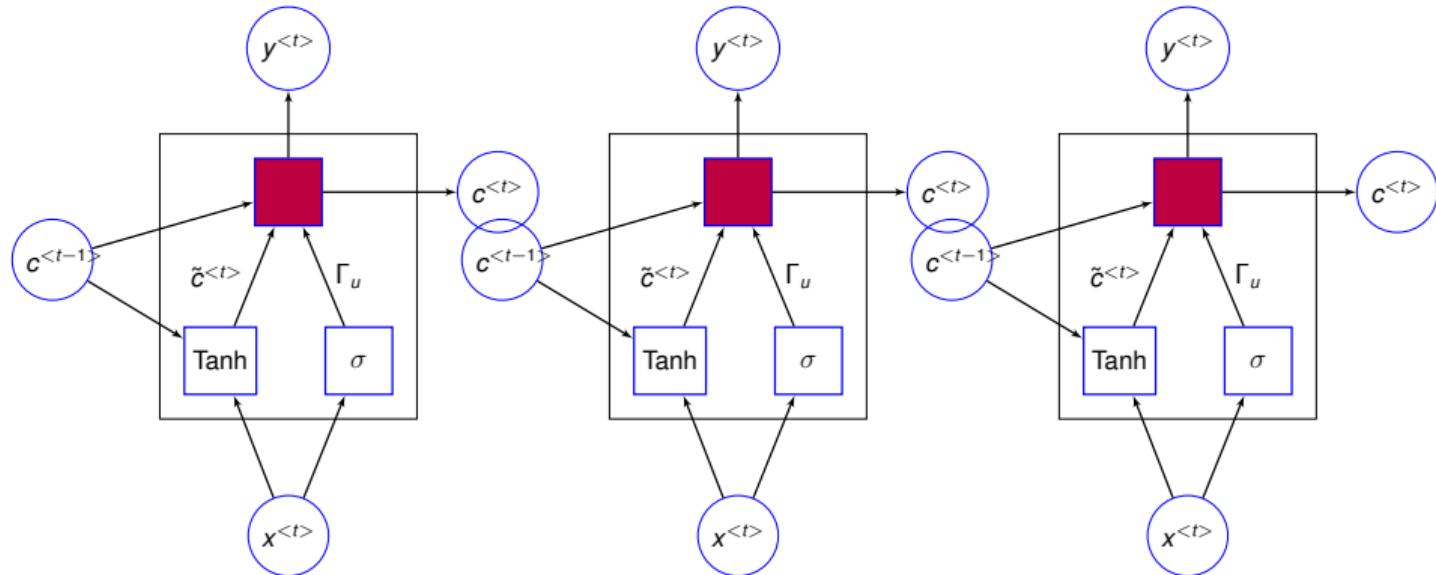
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

GATED RECURRENT UNIT (GRU)



GATED RECURRENT UNIT (GRU)



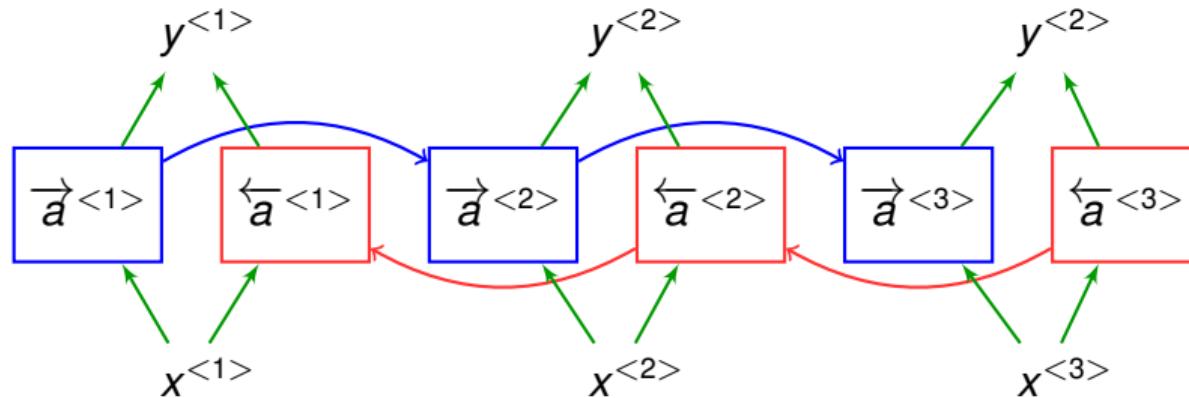
IN THIS SEGMENT

- ① SEQUENCE LEARNING
 - ② RECURRENT NEURAL NETWORK (RNN)
 - ③ TYPES OF RNN
 - ④ LEARNING IN RNN
 - ⑤ ISSUES IN RNN
 - ⑥ LONG SHORT TERM MEMORY UNIT (LSTM)
 - ⑦ GATED RECURRENT UNIT (GRU)
 - ⑧ BIDIRECTIONAL RNN
-

BIDIRECTIONAL RNN (BRNN)

- Forward and backward connections.
- The blocks can be RNN, GRU, LSTM.
- Mostly used in the NLP.
- Acyclic graph
- Example: Name entity recognition
He said “Teddy bear is soft.”
He said “Teddy Roosevelt was a President.”

BRNN ARCHITECTURE



$$\hat{y}^{<t>} = g(W_y [\vec{a}^{<t>} \overleftarrow{a}^{<t>} x^{<t>}] + b_y)$$

SUMMARY

- Use GRU, when dependency is short. Eg: Weather forecasting
- Use LSTM, when dependency is long. Eg: NLP Translation
- Use BRNN, dependency is in both direction. Eg: Stock prediction

References

- ① Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola
https://d2l.ai/chapter_introduction/index.html
- ② Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>

Thank You!



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

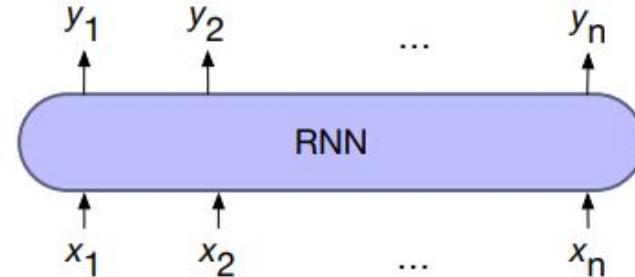
Attention Models & Transformers

S. P. Vimal, CSIS Department (WILP Division)
vimalsp@pilani.bits-pilani.ac.in

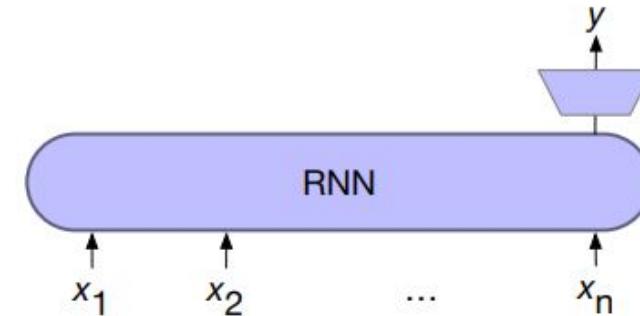
Topics

- ★ Brief review of Sequence Models (RNN, LSTMS)
- ★ Encoder - Decoder Models
- ★ Attention
- ★ Transformers
- ★ Vision Transformers

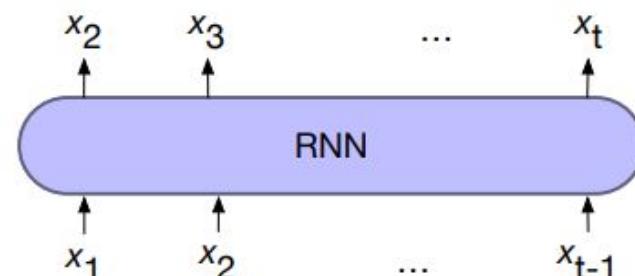
RNN Architectures for NLP Tasks



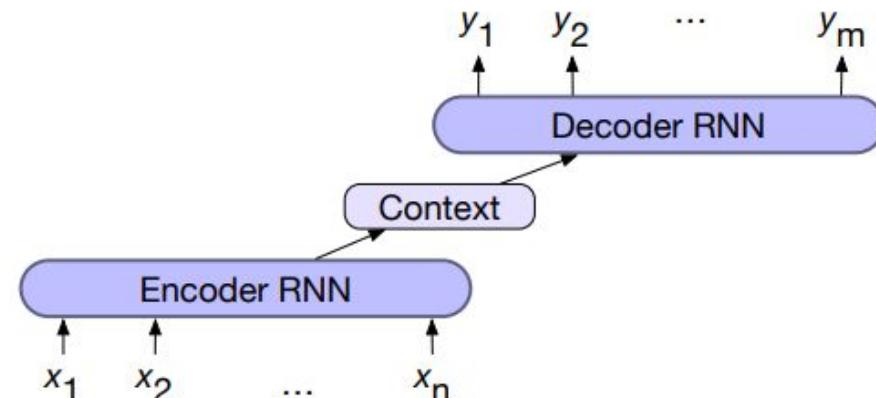
a) sequence labeling
Ex: POS Tagging, Named Entity Tagging



b) sequence classification
Ex: Sentiment Analysis



c) language modeling
Ex: Predict Next Word



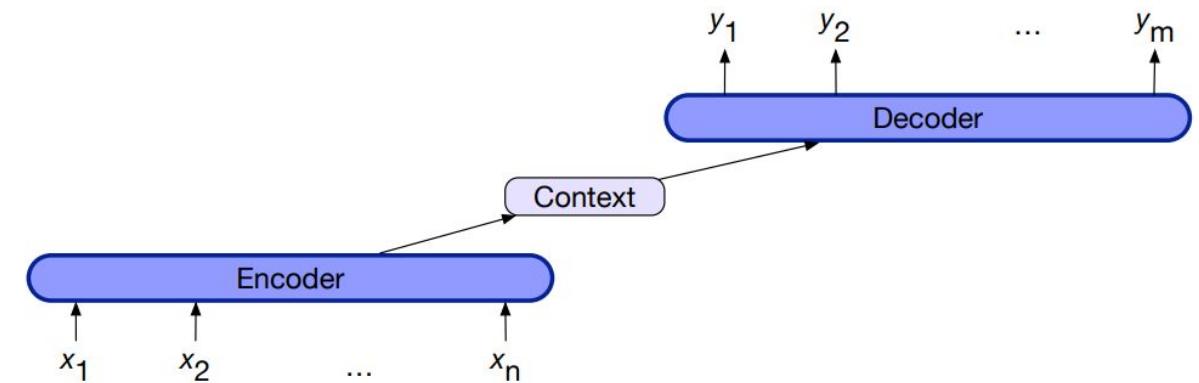
d) encoder-decoder
Ex: Language Translation

Encoder - Decoder

- **Goal:** Develop an architecture capable of generating *contextually appropriate, arbitrary length*, output sequences

- **Applications:**

- Machine translation
- Summarization
- Question answering
- Dialogue modeling.



Encoder - Decoder

Encoder:

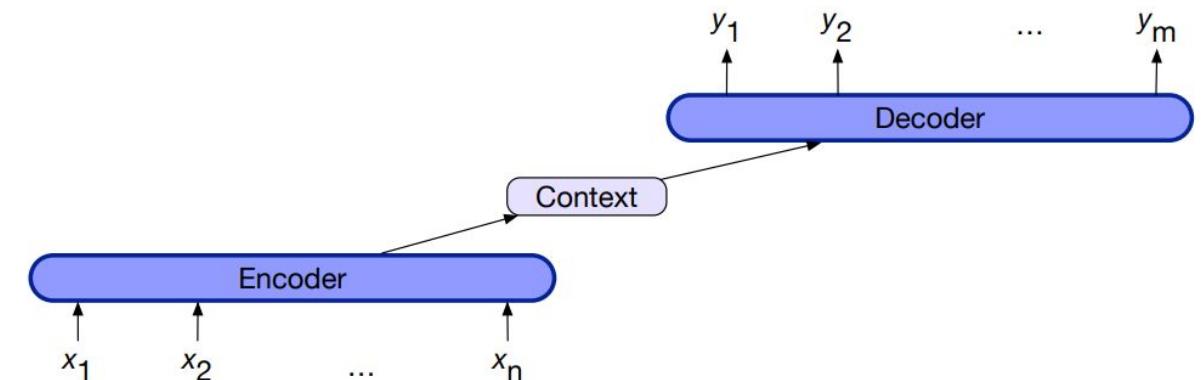
- Input sequence $(x_{1 \dots n}) \rightarrow$ Sequence of contextualized representations, $(h_{1 \dots n})$
- Ex: LSTM, CNN, Transformers etc.

Context:

- c , a function of $(h_{1 \dots n})$

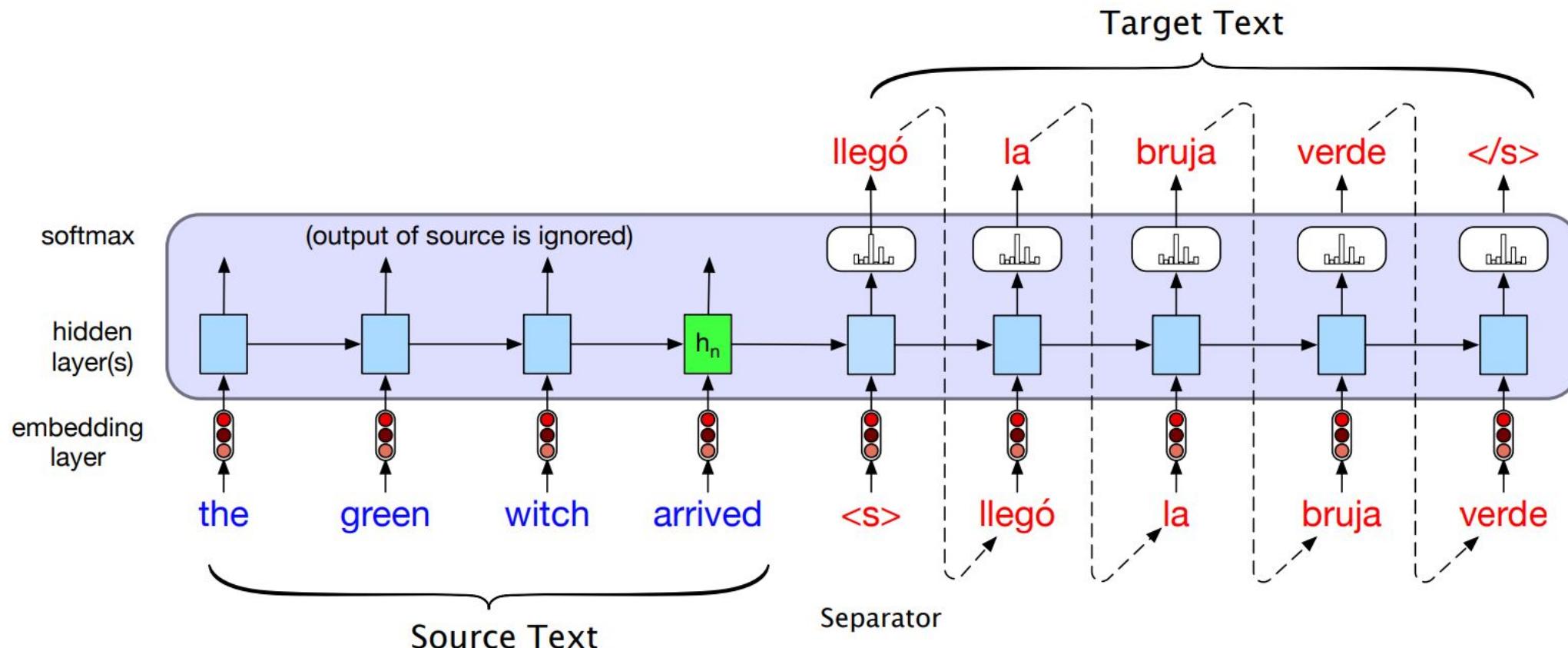
Decoder:

- $c \rightarrow$ arbitrary length sequence of hidden states $(h_{1 \dots m}) \rightarrow$ sequence of output states $(y_{1 \dots m})$



Encoder - Decoder

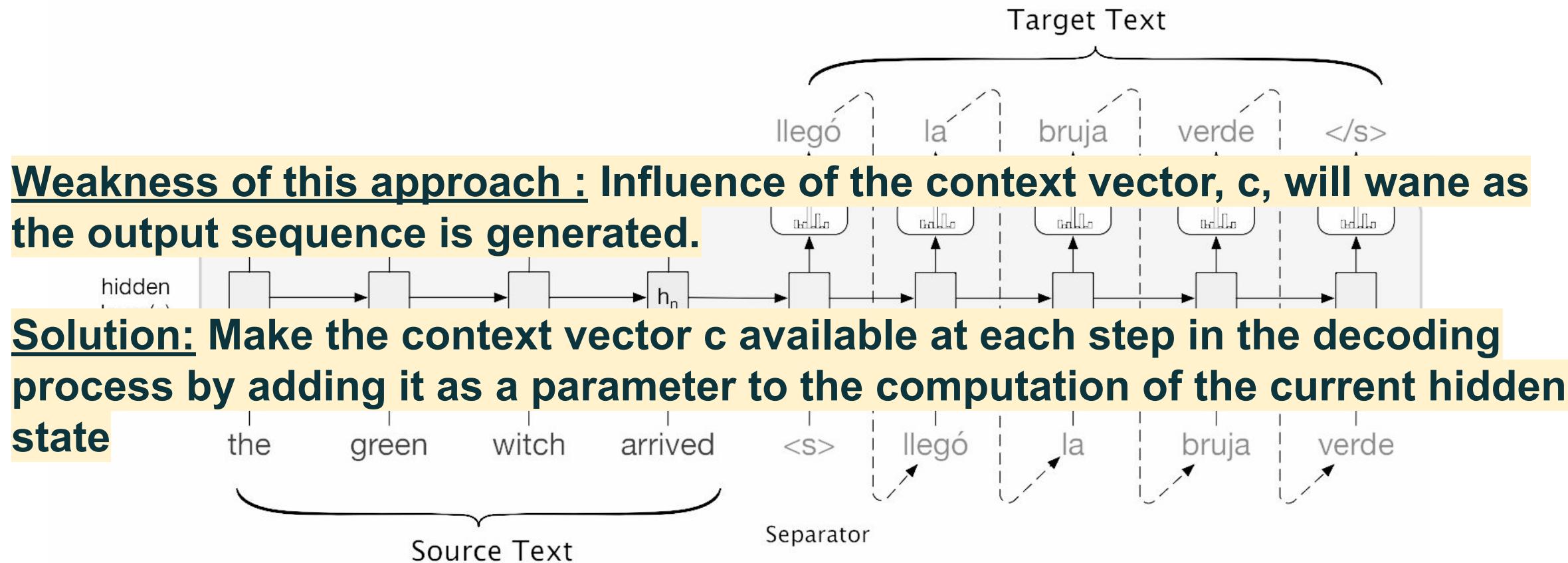
Encoder - Decoder for Language Translation



Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

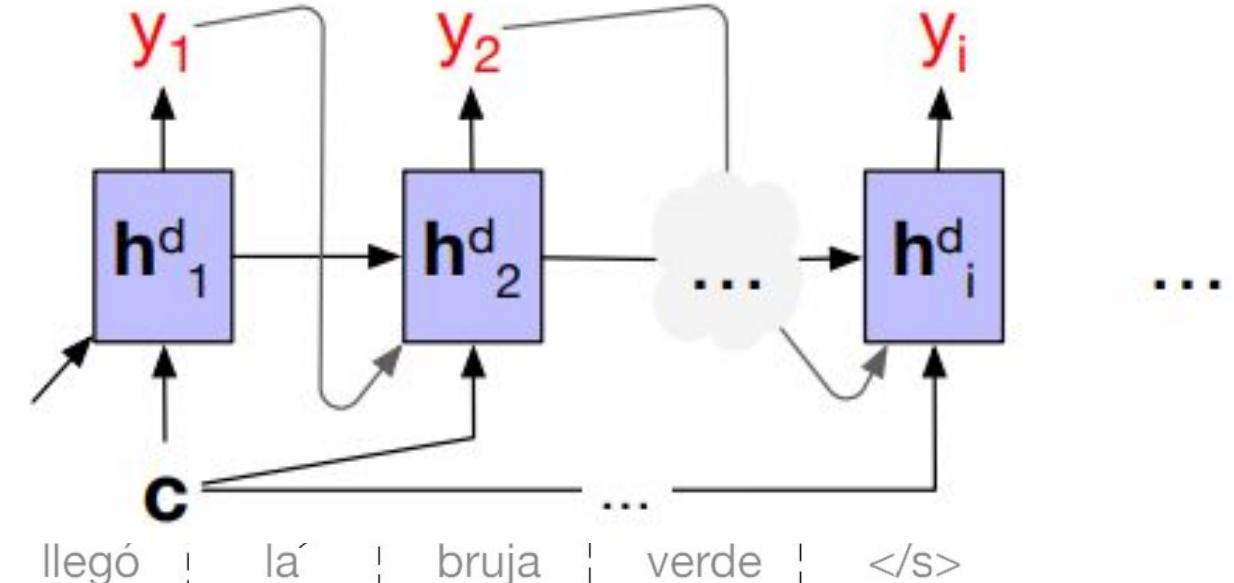
Encoder - Decoder for Language Translation



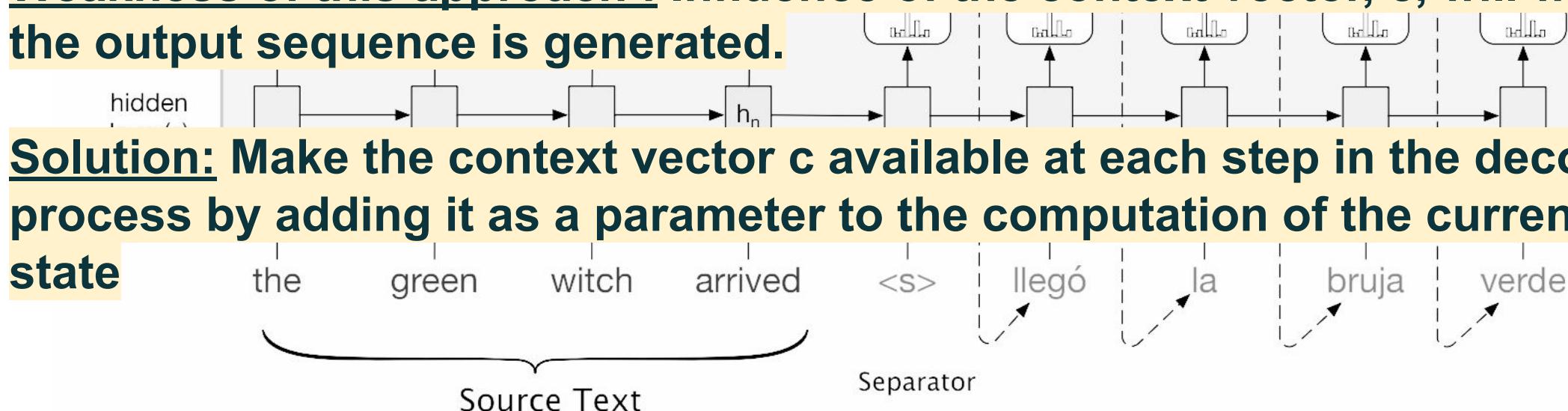
Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation



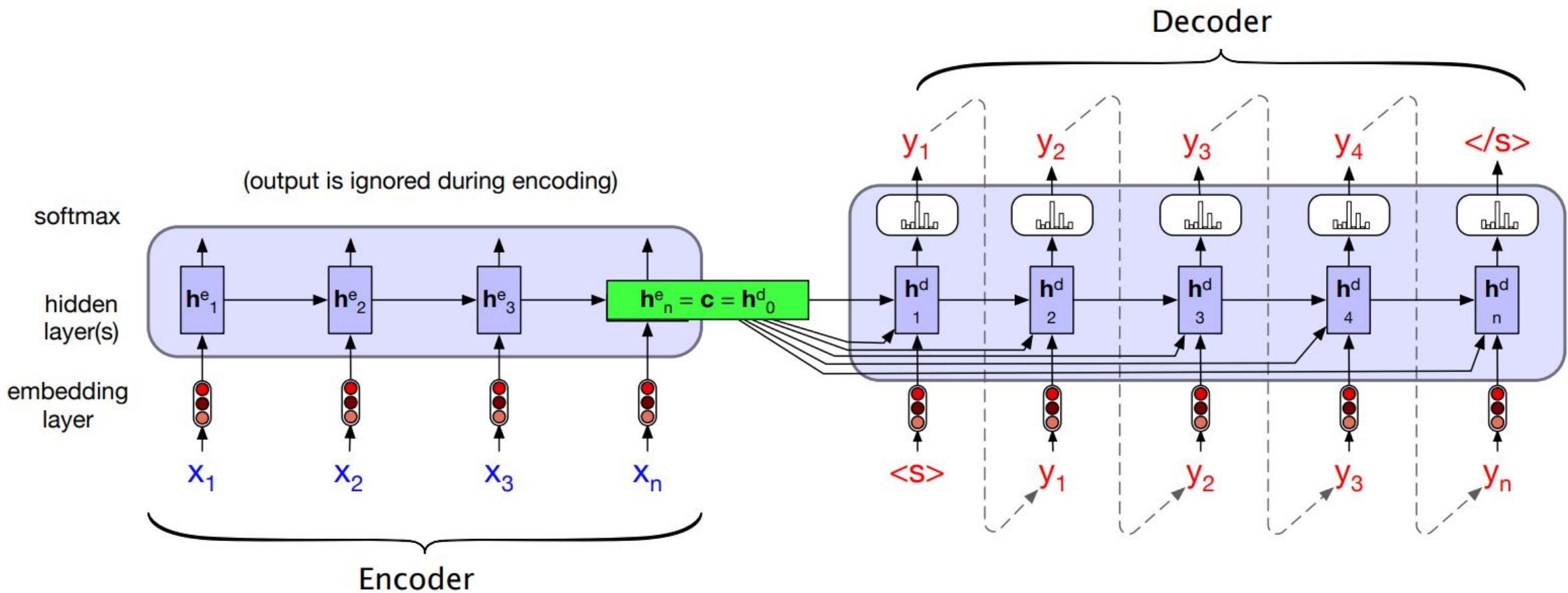
Weakness of this approach : Influence of the context vector, c , will wane as the output sequence is generated.



Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

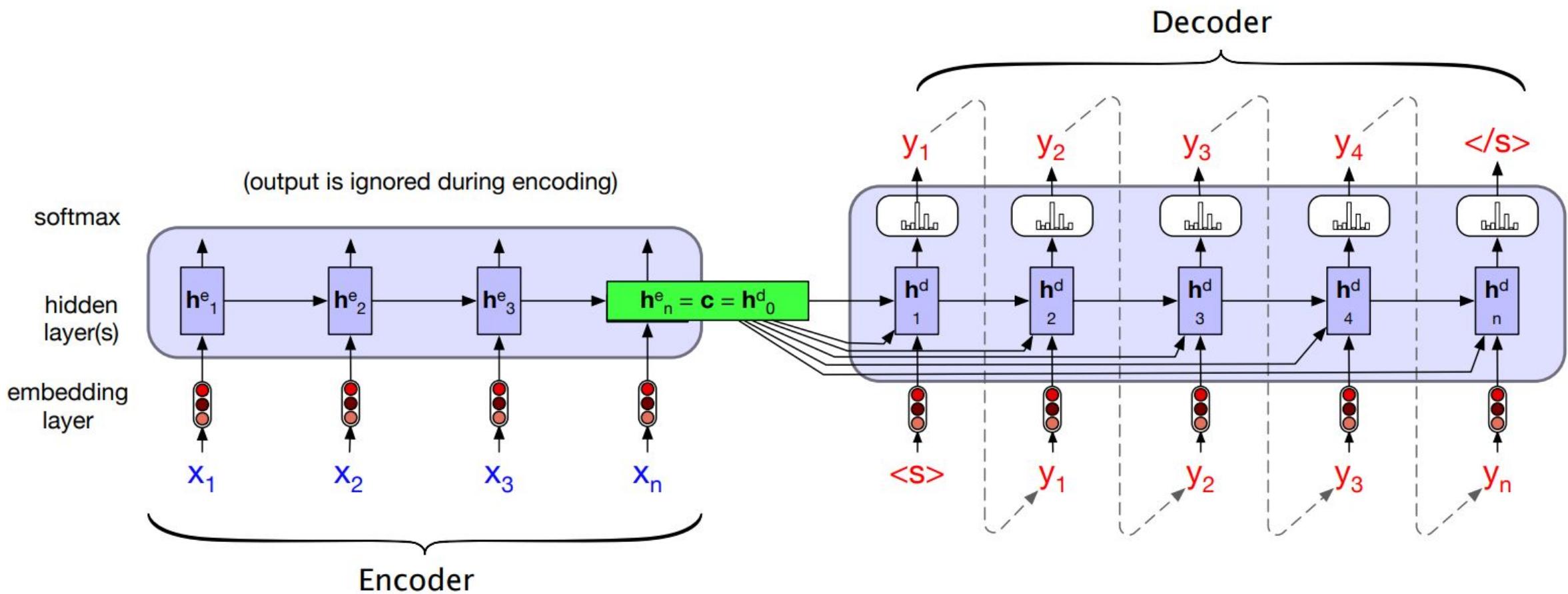
Encoder - Decoder for Language Translation



Encoder - Decoder

Encoder - Decoder for Language Translation

$$\begin{aligned}
 \mathbf{c} &= \mathbf{h}_n^e \\
 \mathbf{h}_0^d &= \mathbf{c} \\
 \mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
 \mathbf{z}_t &= f(\mathbf{h}_t^d) \\
 y_t &= \text{softmax}(\mathbf{z}_t)
 \end{aligned}$$

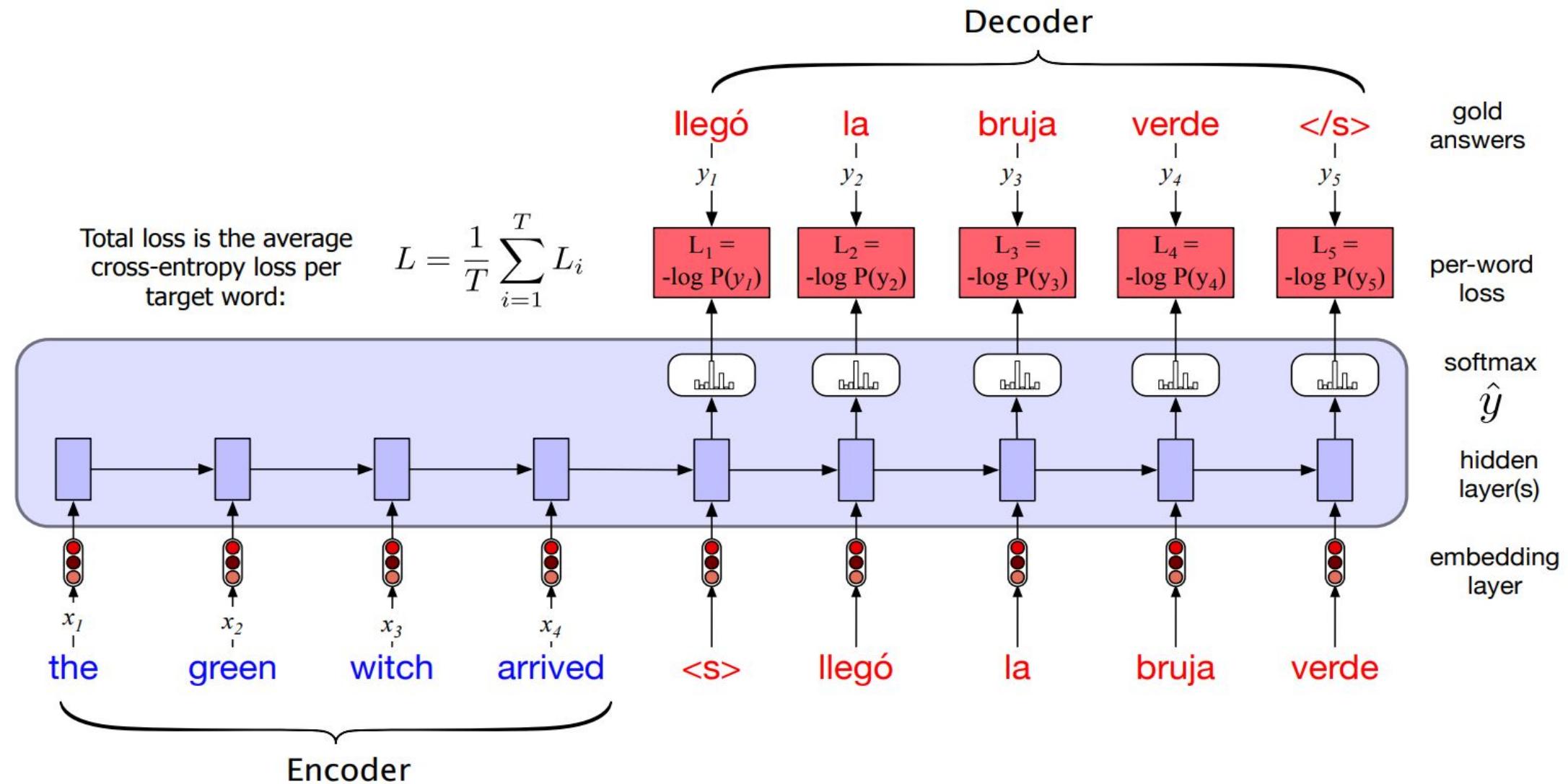


Encoder - Decoder

Training

Total loss is the average cross-entropy loss per target word:

$$L = \frac{1}{T} \sum_{i=1}^T L_i$$



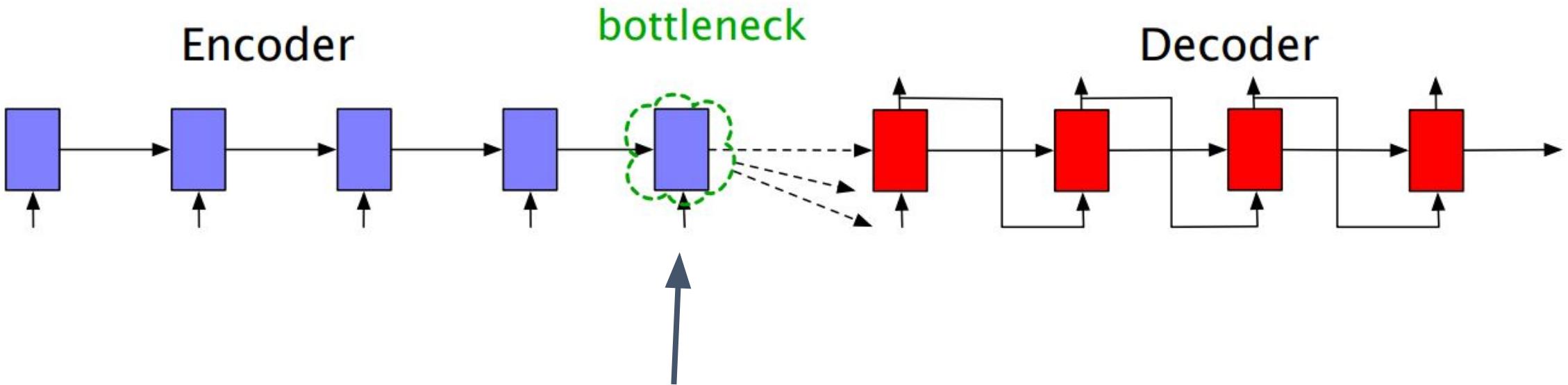


Encoder - Decoder

Teaching Forcing

- Force the system to use the gold target token from training as the next input x_{t+1} , rather than allowing it to rely on the (possibly erroneous) decoder output \hat{y}_t .
- Speeds up training

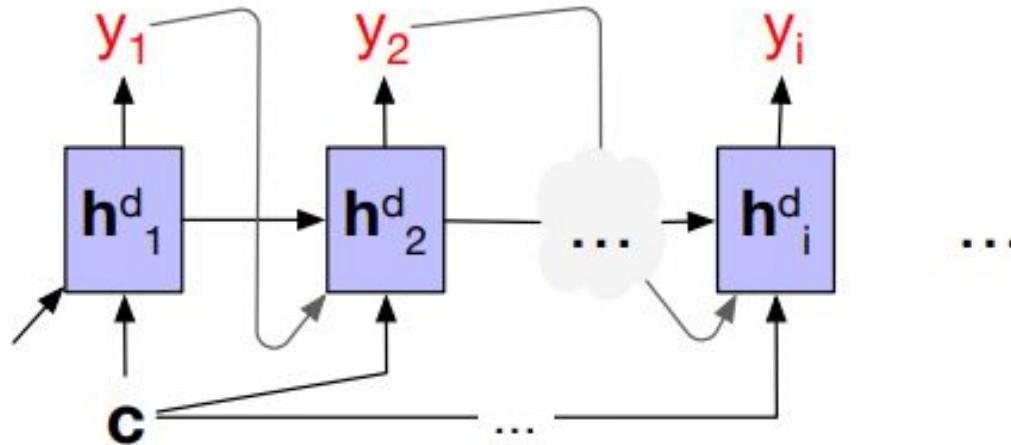
Attention !



In an encoder-decoder arch, the final hidden state acts as a bottleneck:

- It must represent absolutely everything about the meaning of the source text
- The only thing the decoder knows about the source text is what's in this context vector

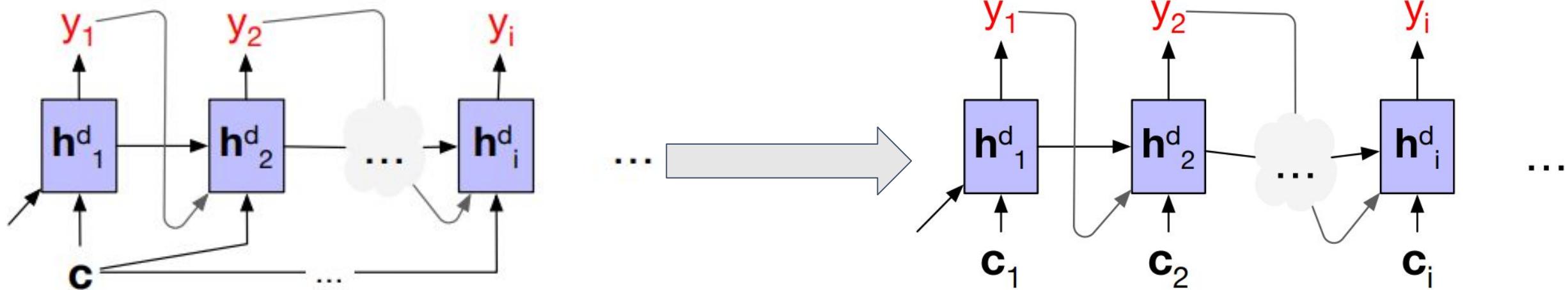
Attention !



Without attention, a decoder sees the same context vector ,
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

Attention !



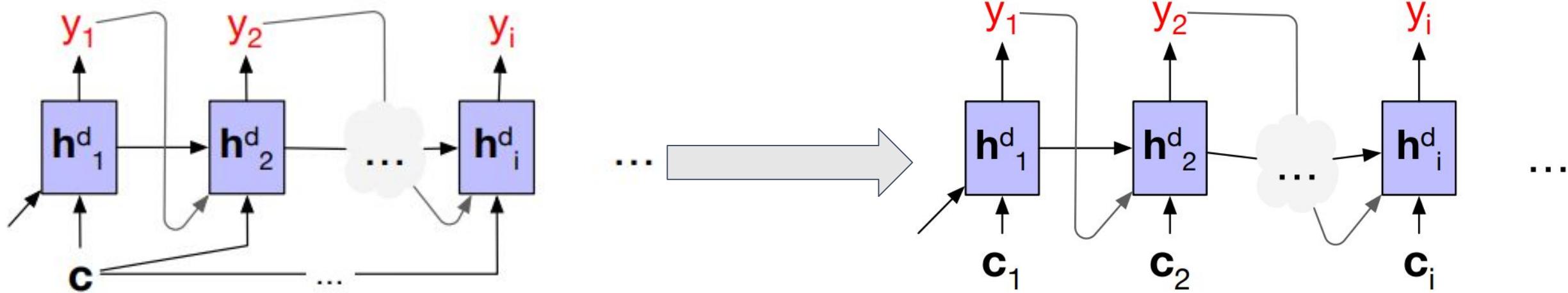
Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

Attention !



Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder

Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

Attention !

Step -1 : Find out how relevant each encoder state is to the present decoder state \mathbf{h}_{i-1}^d

Compute a score of similarity between \mathbf{h}_{i-1}^d and all the encoder states : $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$

Dot Product Attention : $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$

Step -2 : Normalize all the scores with softmax to create a vector of weights, $\alpha_{i,j}$

$\alpha_{i,j}$ indicates the proportional relevance of each encoder hidden state j to the prior hidden decoder state, \mathbf{h}_{i-1}^d

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) \quad \forall j \in e) \\ &= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))}{\sum_k \exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))}\end{aligned}$$



Attention !

Step -3 : Given the distribution in α , compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

Attention !

Step -3 : Given the distribution in α , compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

Plus : In step-1, we can get a more powerful scoring function by parameterizing the score with its own set of weights, \mathbf{W}_s :

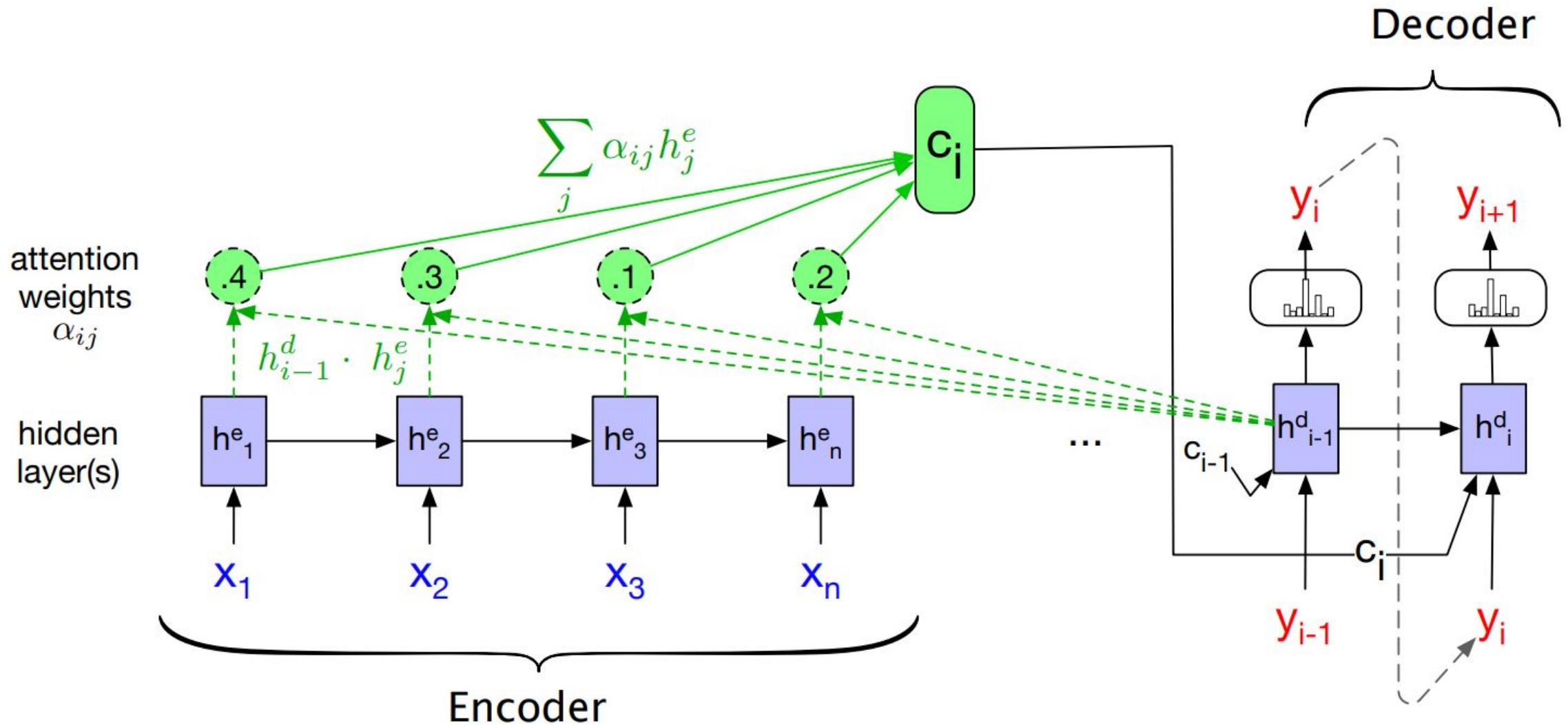
$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

\mathbf{W}_s , is trained during normal end-to-end training,

\mathbf{W}_s , gives the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

Attention !

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$





Transformers

- 2017, NIPS, Vaswani et. al., **Attention Is All You Need !!!**
- Transformers **map sequences** of input vectors (x_1, \dots, x_n) to sequences of output vectors (y_1, \dots, y_n) **of the same length**
- Made up of **transformer blocks** in which the key component is **self-attention** layers

[Self-attention allows a network to directly extract and use information from arbitrarily large contexts directly !!!]

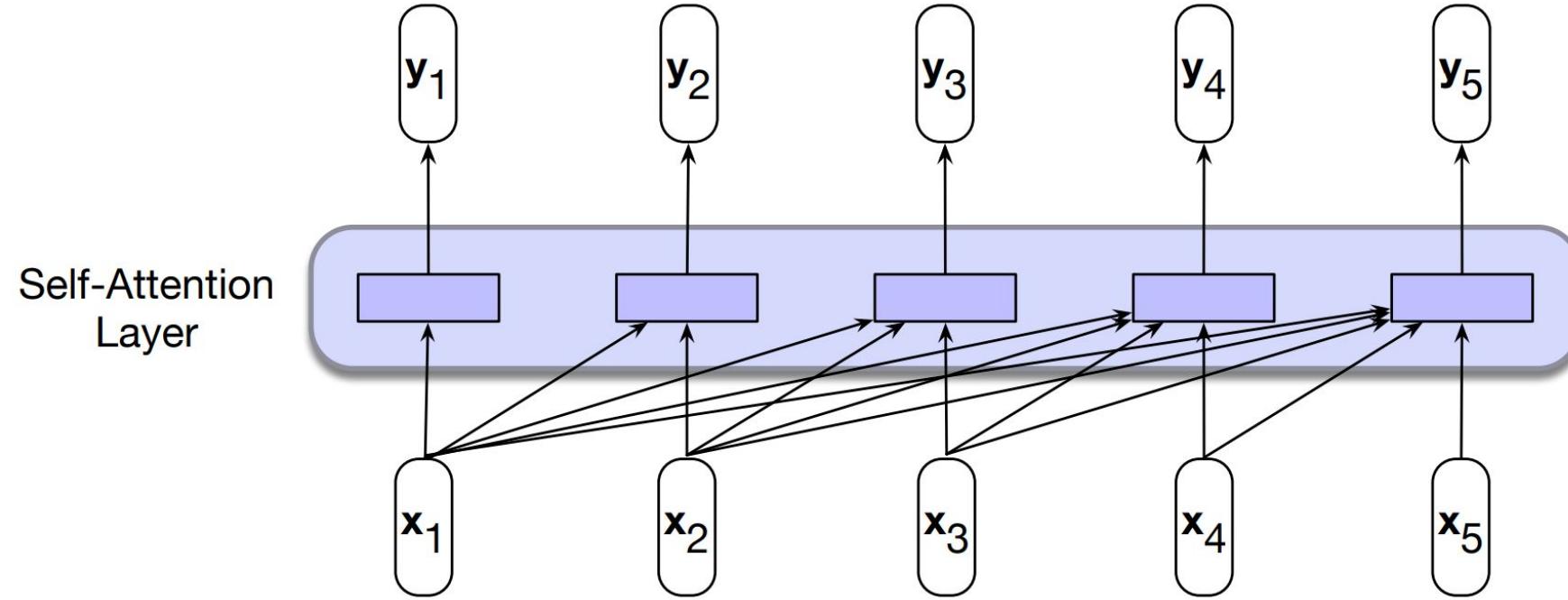
- Transformers are **not based on recurrent connections** \Rightarrow Parallel implementations possible \Rightarrow Efficient to scale (comparing LSTM)



Self-Attention | Transformers

- Attention \Rightarrow Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention \Rightarrow
 - > Set of **comparisons** are to other elements **within a given sequence**
 - > Use these comparisons to compute an output for the current input

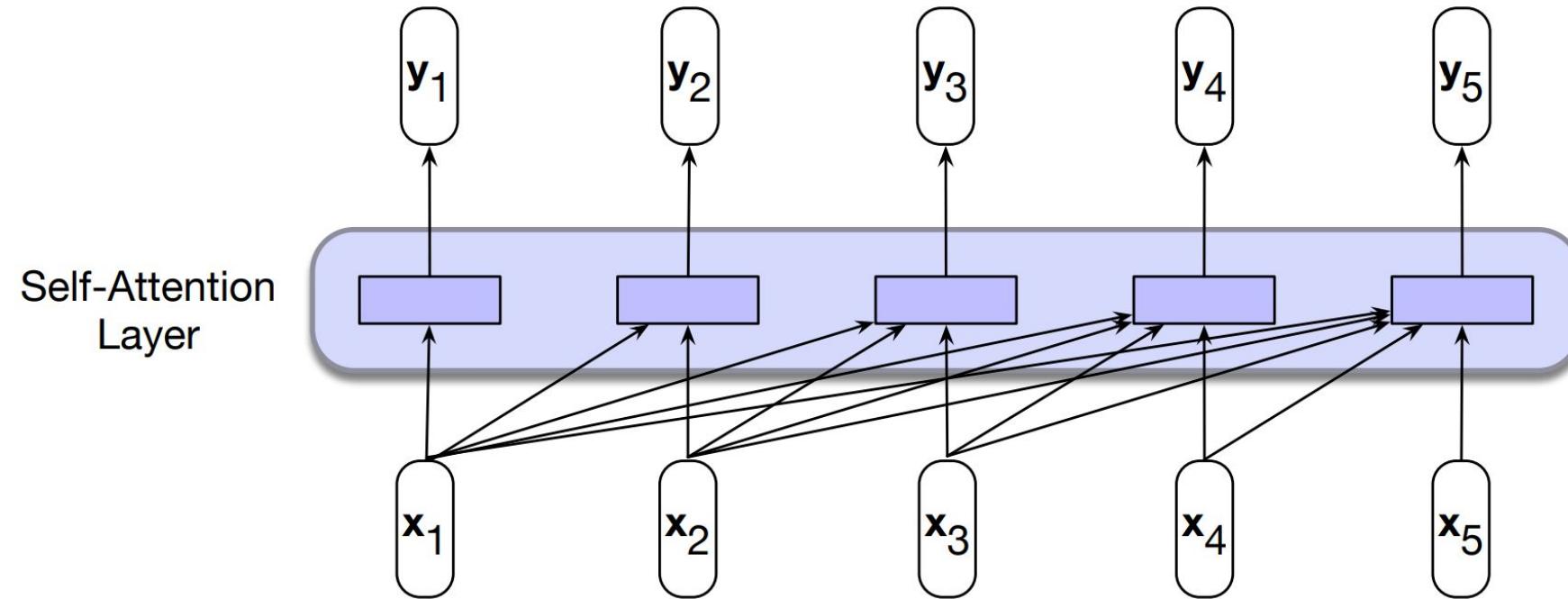
Self-Attention | Transformers



In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.

Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention | Transformers



$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i \end{aligned}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$



Self-Attention | Transformers

- Let us understand how transformers uses self-attention !

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

Query, Q

As the current focus of attention when being compared to all of the other preceding inputs.

Key, K

In its role as a preceding input being compared to the current focus of attention.

Value, V

As a value used to compute the output for the current focus of attention

Three different roles each \mathbf{x}_i (input embedding) , in the computation of self attention

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

The simple dot product can be an arbitrarily large;
scaled dot-product is used in transformers;

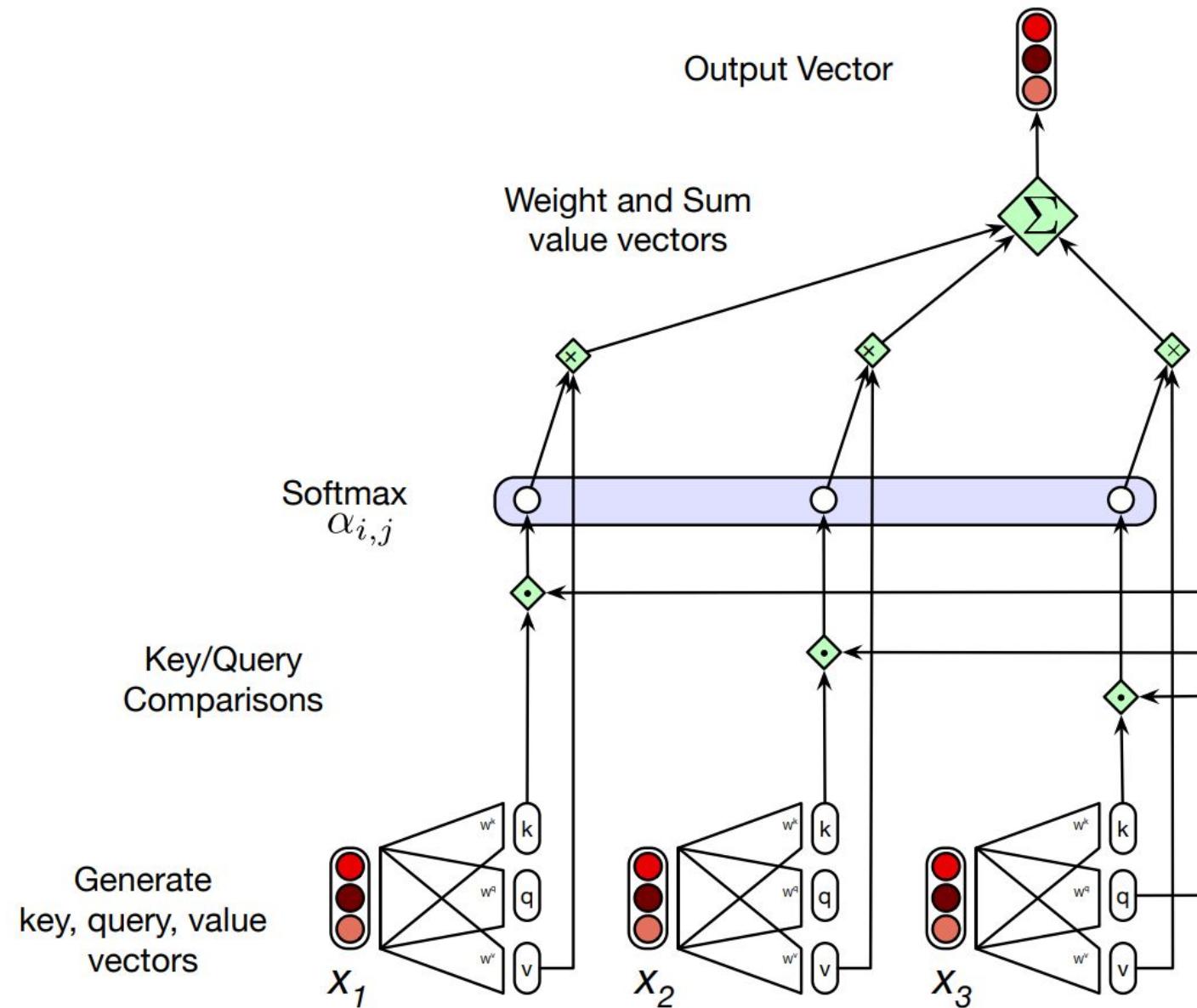
$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i\end{aligned}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Self-Attention | Transformers

- Each output, y_i , is computed independently
- Entire process can be parallelized

Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention



Self-Attention | Transformers

- Pack the input embeddings of the N input tokens into a single matrix

$$\mathbf{X} \in \mathbb{R}^{N \times d}$$

> Each row of X is the embedding of one token of the input

- Multiply X by the key, query, and value ($d \times d$) matrices

$$\mathbf{Q} = \mathbf{XW}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{XW}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{XW}^{\mathbf{V}}$$

Self-Attention | Transformers

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}$$

QK^T Matrix N

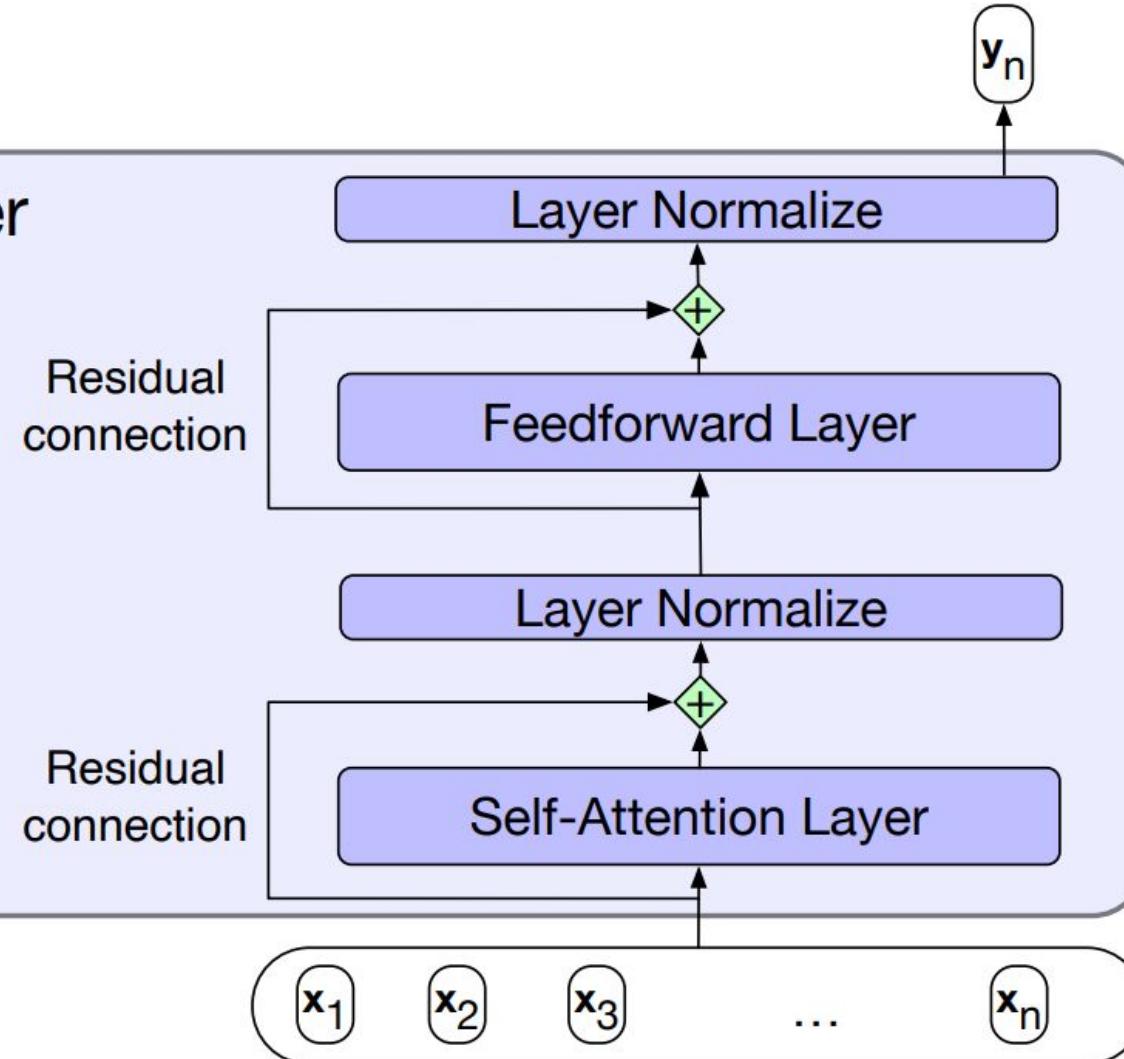
q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

Note: Upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero)

Transformer Blocks | Transformers

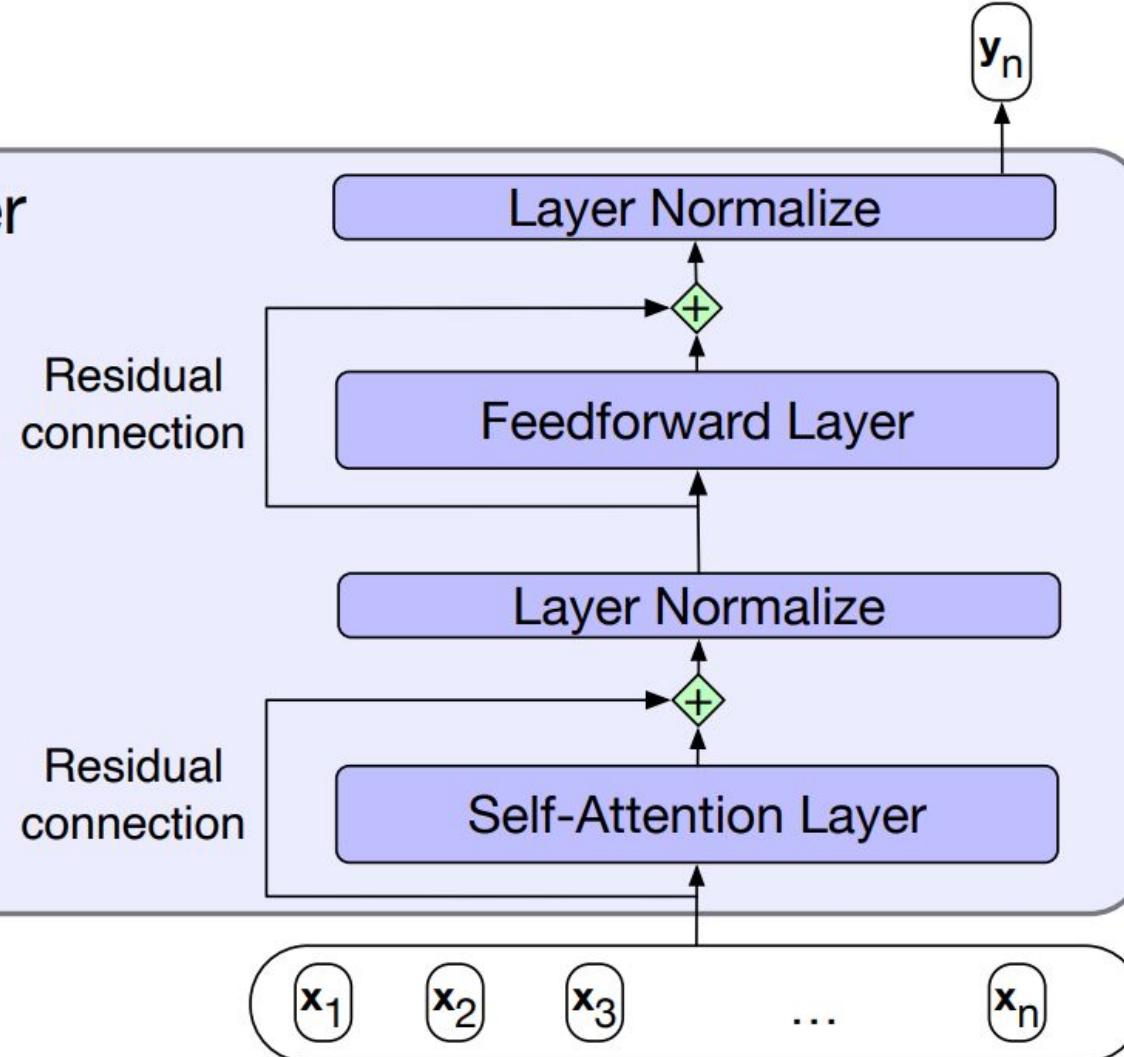
Transformer Block



$$\begin{aligned}\mathbf{z} &= \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x})) \\ \mathbf{y} &= \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))\end{aligned}$$

Transformer Blocks | Transformers

Transformer Block



$$\begin{aligned} \mathbf{z} &= \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x})) \\ \mathbf{y} &= \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z})) \end{aligned}$$

LayerNorm:

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm} = \gamma \hat{\mathbf{x}} + \beta$$



Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously
 - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
 - >> **Heads** ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
 - >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

Multihead-Attention | Transformers

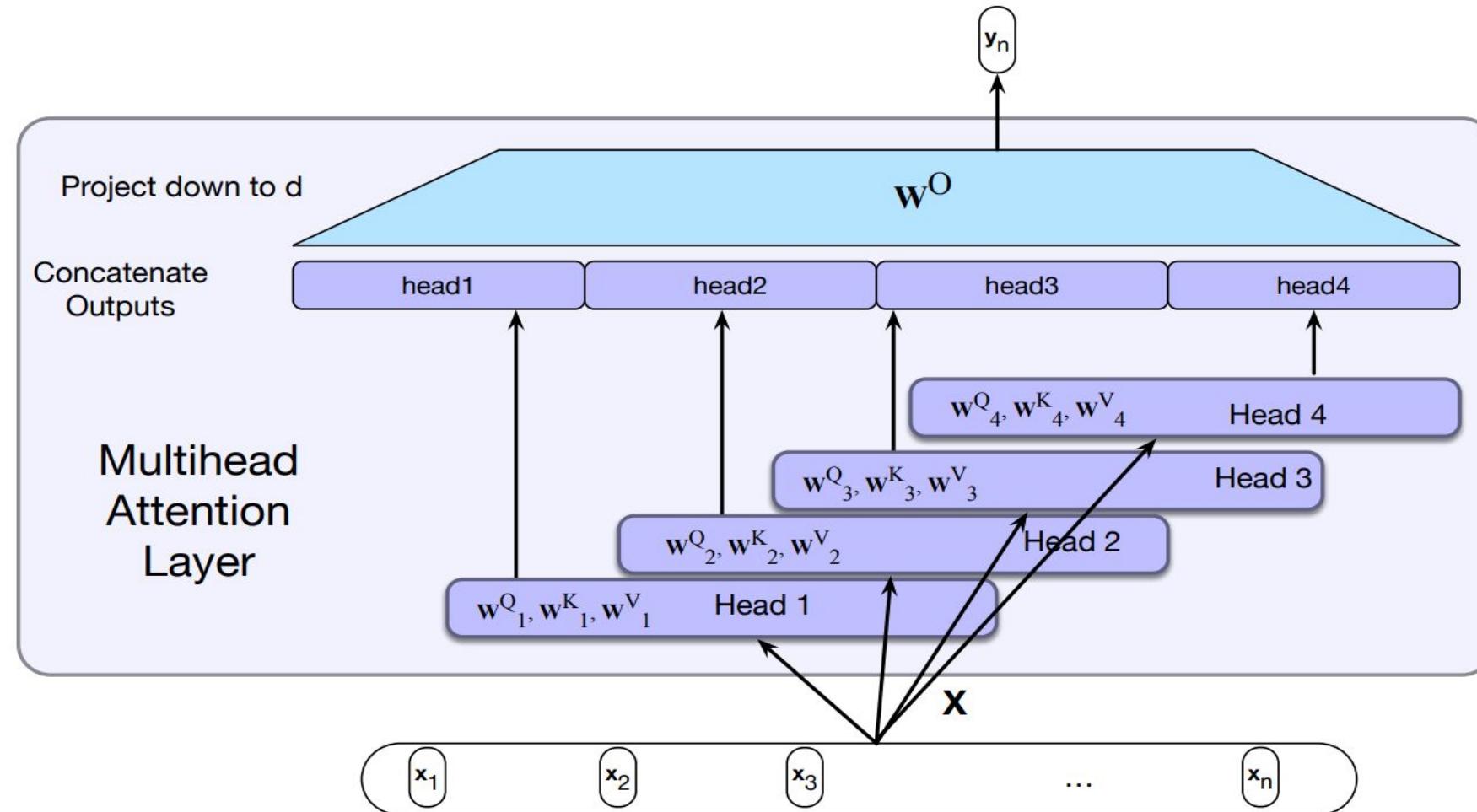
- Different words in a sentence can relate to each other in many different ways simultaneously
 - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
 - >> **Heads** ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
 - >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X} \mathbf{W}_i^Q ; \mathbf{K} = \mathbf{X} \mathbf{W}_i^K ; \mathbf{V} = \mathbf{X} \mathbf{W}_i^V$$

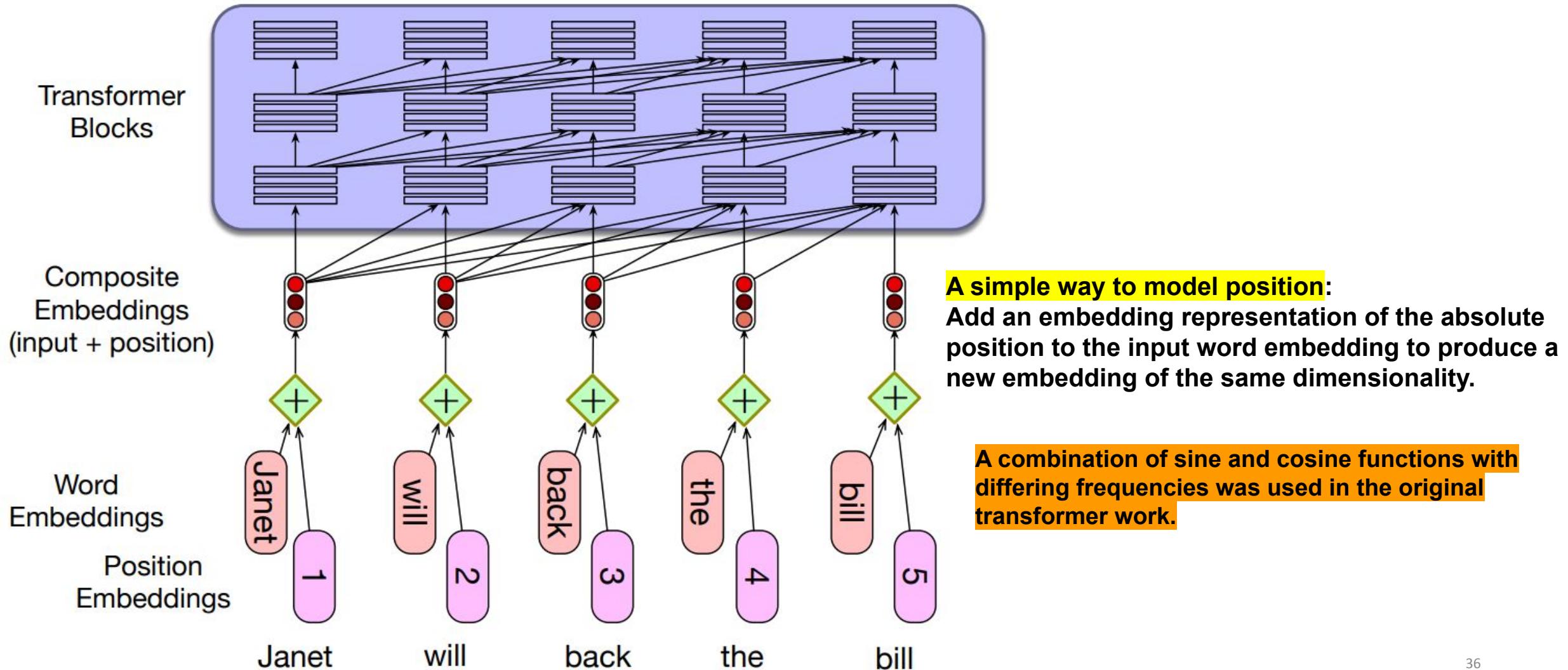
$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

Multihead-Attention | Transformers

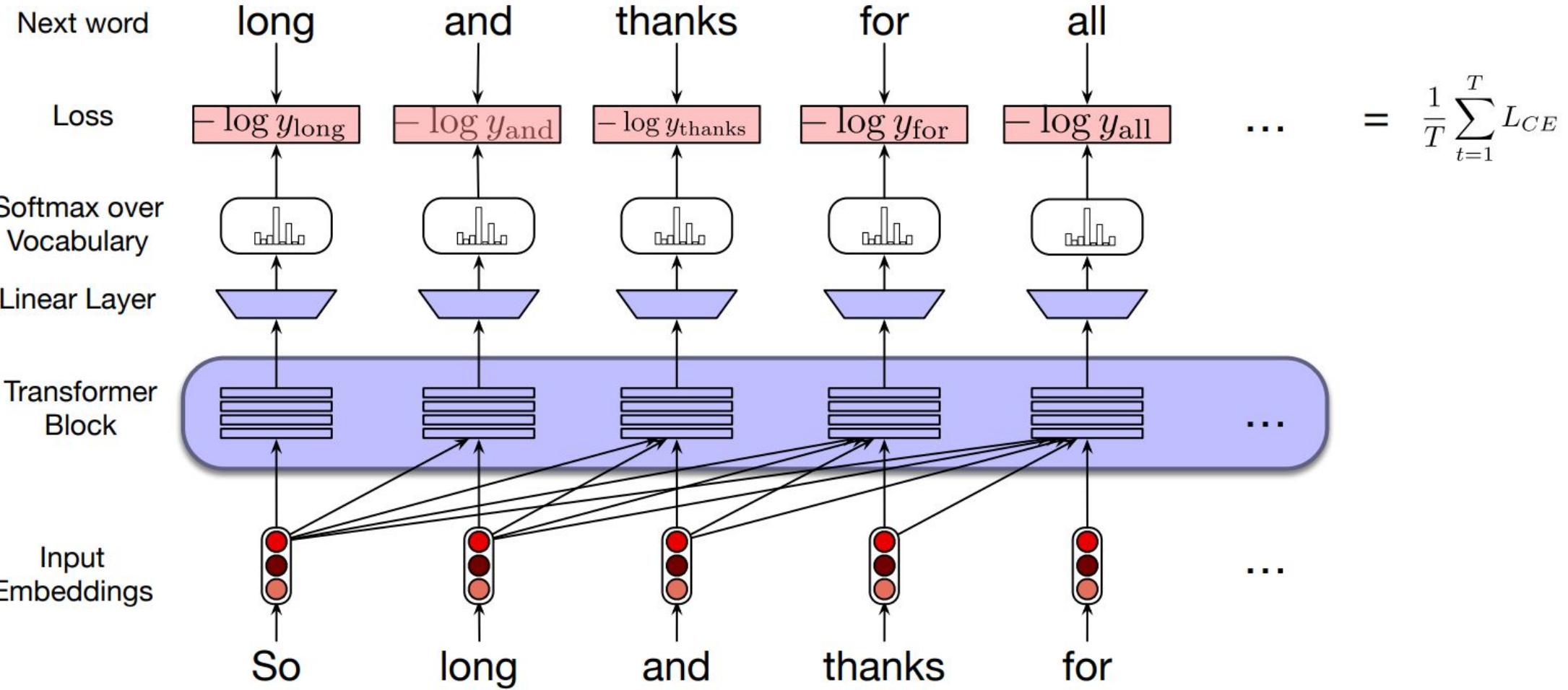


Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d , thus producing an output of the same size as the input so layers can be stacked.

Positional Embeddings | Transformers



Transformers as Language Models





Vision Transformers (ViT)

- Ref: Alexey et. al.(2021) , ICLR, An Image Is Worth 16x16 Words: Transformers For Image Recognition At Scale
-



Readings

1. [Draft Chapter 9](#) and [Draft Chapter 10](#), Speech and Language Processing. Daniel Jurafsky & James H. Martin.
2. Vaswani, Ashish, et al. "Attention is all you need." [Advances in neural information processing systems 30 \(2017\)](#).
3. Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." [arXiv preprint arXiv:2010.11929 \(2020\)](#).



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you

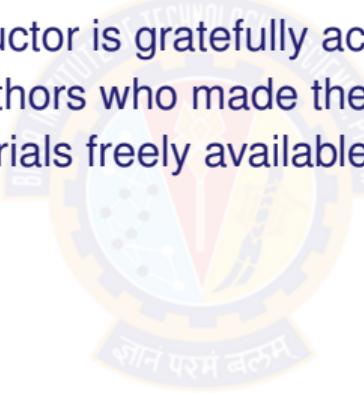


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DEEP LEARNING MODULE 7: AUTOENCODERS

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.



IN THIS SEGMENT

- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS



SUPERVISED LEARNING

DATA: m training examples (X, y)
 X is the data and y is the label.

GOAL: Learn a function to map data to the label. $y = f(X)$

EXAMPLE TASKS: Classification, regression, object detection, semantic segmentation,image captioning,etc.

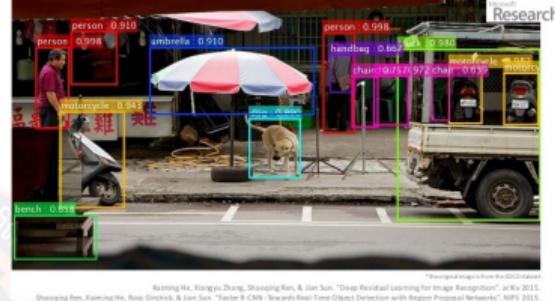


FIGURE: Object recognition: He: Faster R-CNN



FIGURE: OCR: Hui Li, Wang, & Shen

SUPERVISED LEARNING

DATA: m training examples (X, y)
 X is the data and y is the label.

GOAL: Learn a function to map data to the label. $y = f(X)$

EXAMPLE TASKS: Classification, regression, object detection, semantic segmentation,image captioning,etc.

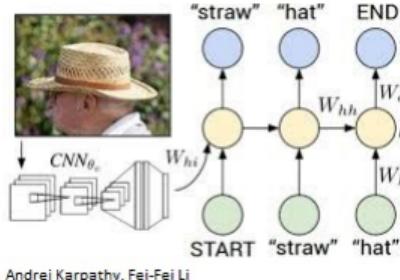
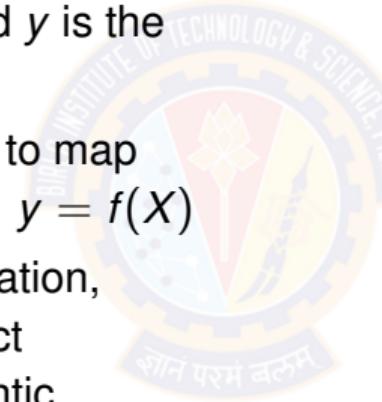


FIGURE: Image Captioning:
Andrej Karpathy, Fei-Fei Li



FIGURE: Semantic Segmentation

UNSUPERVISED LEARNING

DATA: m training examples (X)
 X is the data. No Labels !!!

GOAL: Learn some underlying
hidden structure of the
data.

EXAMPLE TASKS: Clustering,
dimensionality reduction,
feature learning, density
estimation,etc.

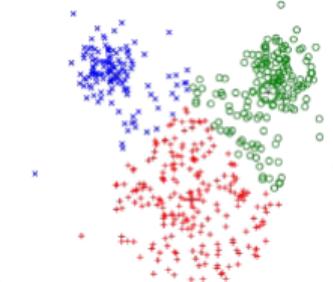


FIGURE: k-means Clustering

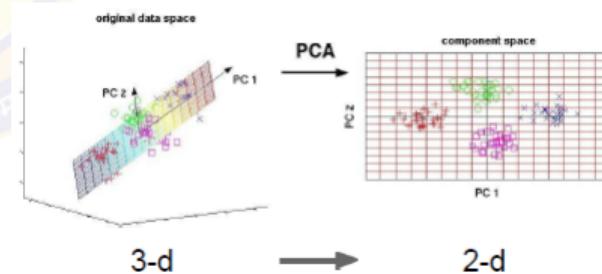


FIGURE: Dimensionality Reduction

UNSUPERVISED LEARNING

DATA: m training examples (X)
 X is the data. No Labels !!!

GOAL: Learn some underlying
hidden structure of the
data.

EXAMPLE TASKS: Clustering,
dimensionality reduction,
feature learning, density
estimation,etc.

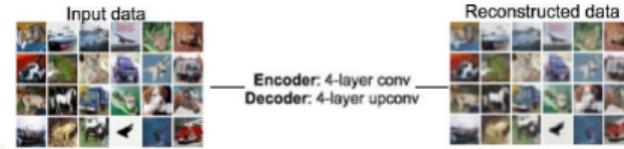


FIGURE: Feature Learning using Autoencoders



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

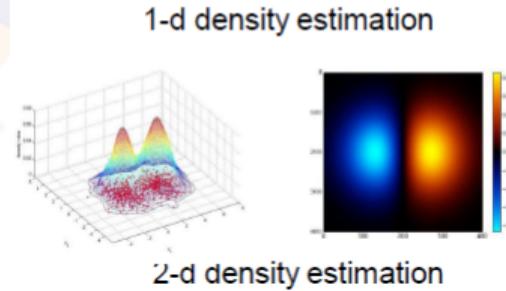


FIGURE: Density Estimation

IN THIS SEGMENT

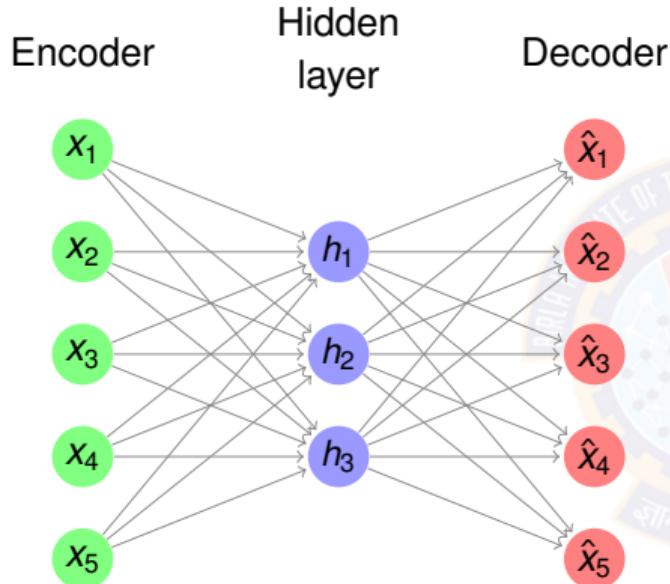
- 1 SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- 2 AUTOENCODERS
- 3 UNDERCOMPLETE AUTOENCODERS
- 4 REGULARIZED AUTOENCODERS
- 5 CONVOLUTIONAL AUTOENCODERS
- 6 DENOISING AUTOENCODERS
- 7 SPARSE AUTOENCODERS
- 8 DEEP AUTOENCODERS



AUTOENCODERS

- **Unsupervised** experience.
- Learns a lower-dimensional feature representation called **code**; from unlabeled training data.
- The code can be used as features also. Autoencoders are efficient **feature detectors**.
- Autoencoders perform **dimensionality reduction**.
- Trained to attempt to copy its input to its output.

AUTOENCODERS ARCHITECTURE

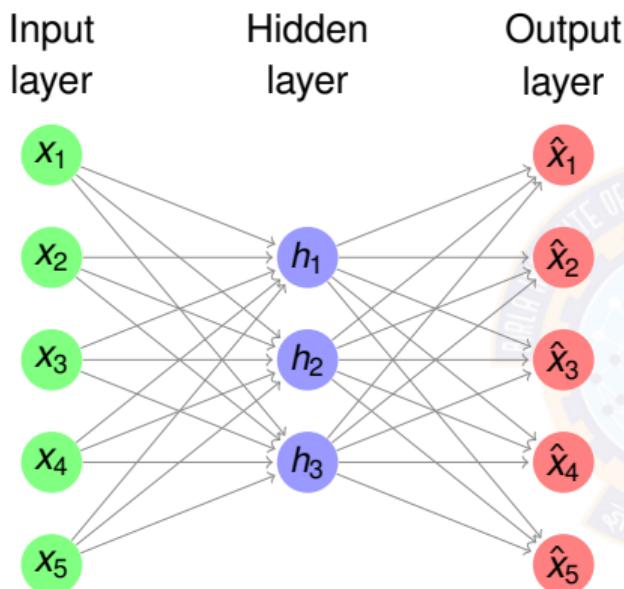


$$h = f(W_e x_i + b)$$

$$\hat{x}_i = g(W_d h + c)$$

- An autoencoder is a special type of feed forward neural network which does the following.
- **Encodes** its input x_i into a hidden representation h .
- **Decodes** the input again from this hidden representation.
- The model is trained to minimize a certain loss function which will ensure that \hat{x}_i is close to x_i .

AUTOENCODERS ARCHITECTURE



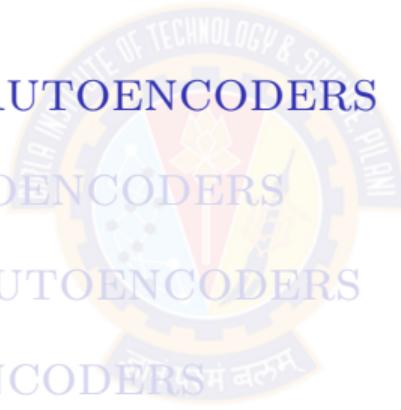
$$h = f(W_e x_i + b)$$

$$\hat{x}_i = g(W_d h + c)$$

- Feedforward deep neural network
- Contains input layer, hidden layer and output layers.
- The number of output neurons is exactly same as the number of input neurons
- The hidden layer describes the code used to represent the input.
- All techniques and optimizations of DNN are applicable.
- Train the autoencoder such that features can be used to reconstruct original data.

IN THIS SEGMENT

- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS

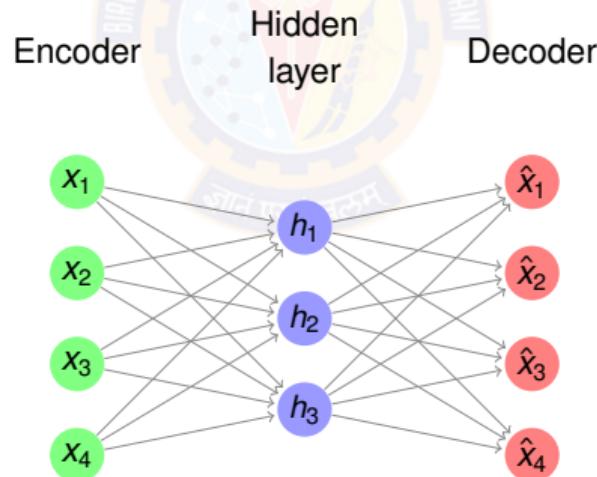


UNDERCOMPLETE AUTOENCODERS

- One hidden layer which has lesser units than input layer.

$$\dim(h) < \dim(x)$$

- Input and output layers have equal number of neurons.
- h is a loss-free encoding of x_i . It captures all important characteristics of x_i .



UNDERCOMPLETE AUTOENCODERS

- Components of Autoencoders

- Encoder (recognition network)
 - converts input to code.

$$h = f(x) = f(W_e x + b)$$

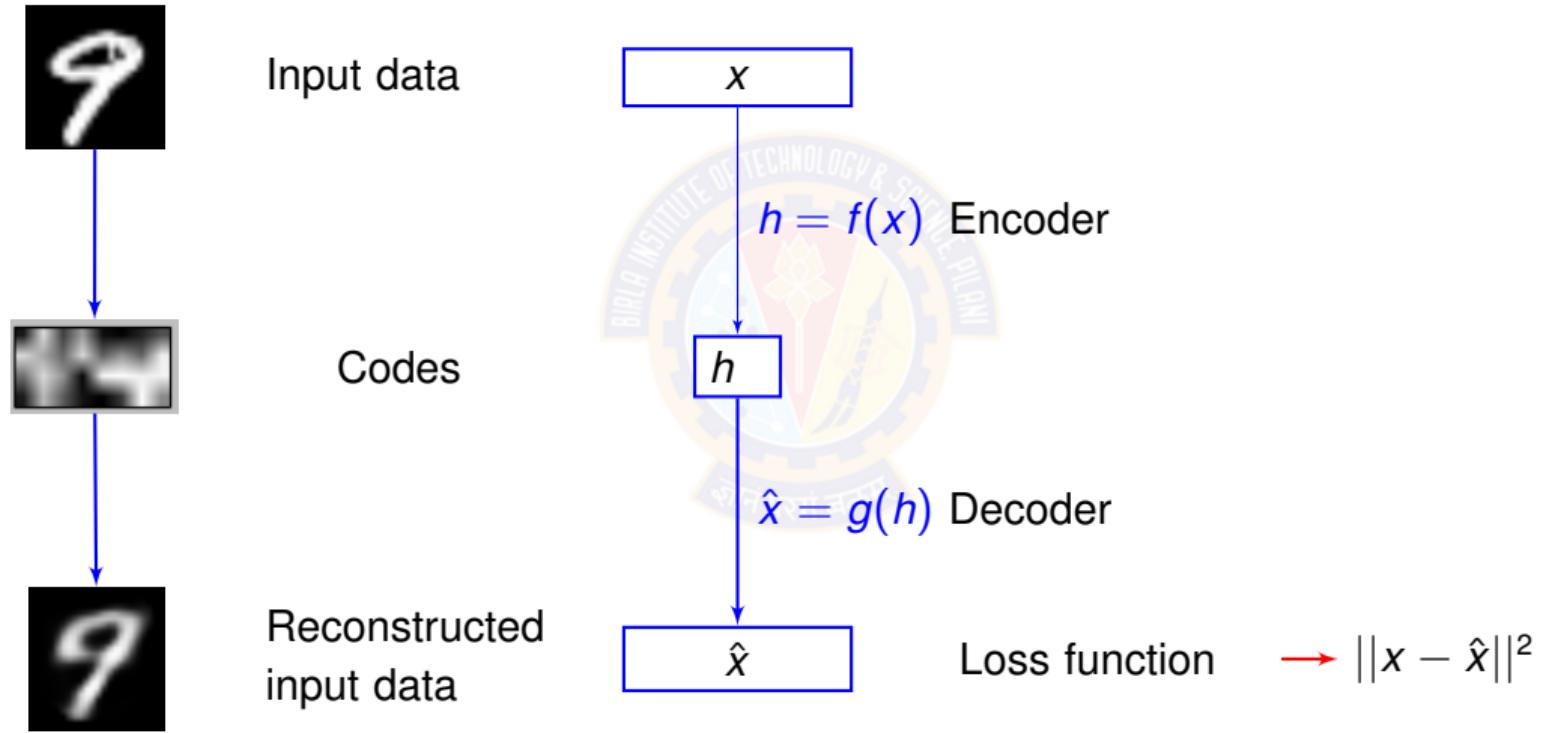
- Code h has a smaller representation than input x .
- Decoder (generative network)
 - produces a reconstruction.

$$\hat{x} = g(h) = g(W_d h + c) = g(f(x))$$

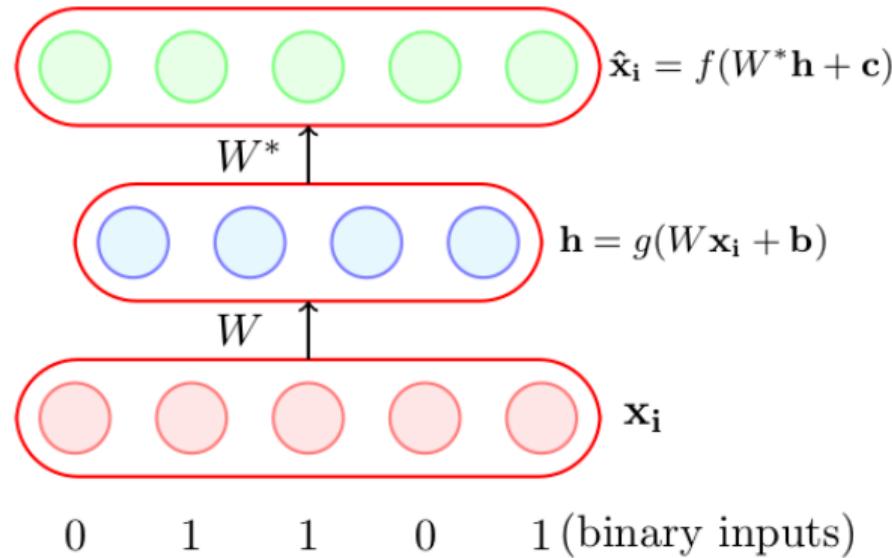
- Loss function
 - reconstruction loss which penalizes when outputs are different from input.
 - Mean squared error

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

UNDERCOMPLETE AUTOENCODERS



AUTOENCODERS – BINARY INPUT



g is typically chosen as the sigmoid function

- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

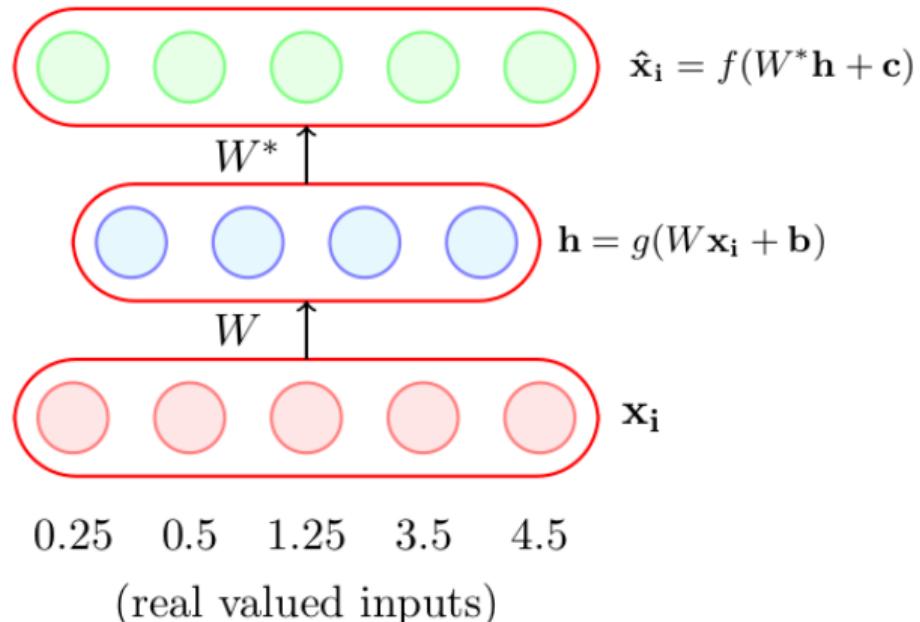
$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- Logistic as it naturally restricts outputs to be between 0 and 1

AUTOENCODERS – REAL INPUT



Again, g is typically chosen as the sigmoid function

- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

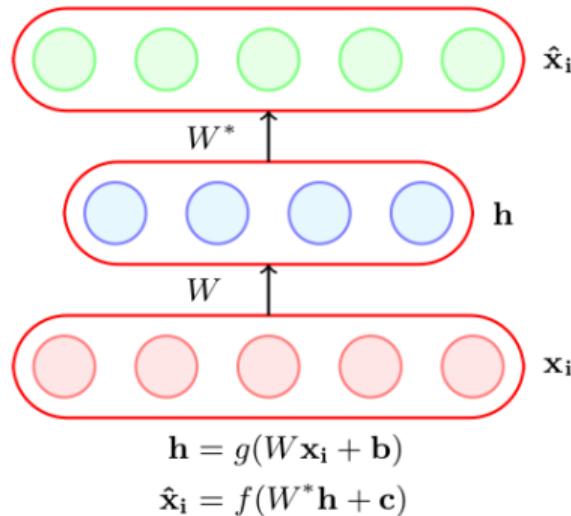
$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?
- They will restrict the reconstructed $\hat{\mathbf{x}}_i$ to lie between [0,1] or [-1,1], whereas we want $\hat{\mathbf{x}}_i \in \mathbb{R}^n$

AUTOENCODERS – LEARNING



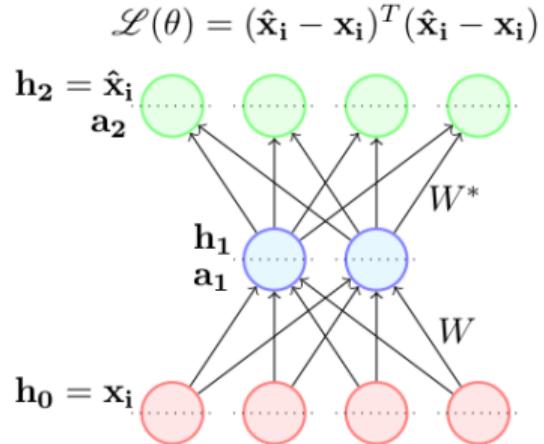
- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible
- This can be formalized using the following objective function:

$$\min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

$$i.e., \min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

- We can then train the autoencoder just like a regular feedforward network using backpropagation
- All we need is a formula for $\frac{\partial \mathcal{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ which we will see now

AUTOENCODERS – LEARNING



- Note that the loss function is shown for only one training example.

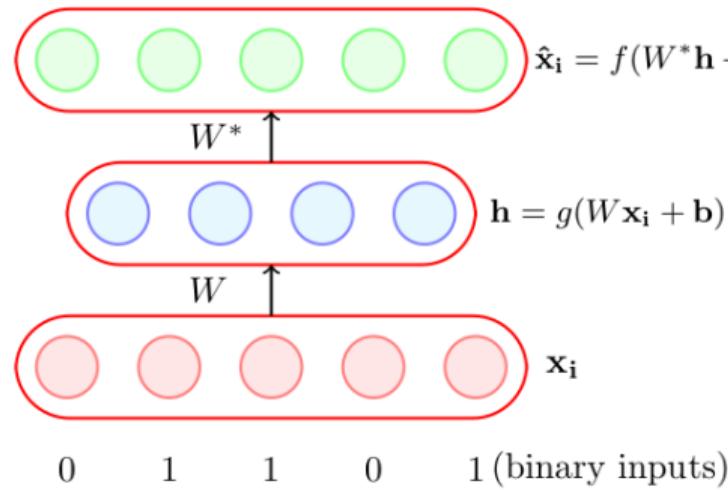
- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$

- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned}\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} &= \frac{\partial \mathcal{L}(\theta)}{\partial \hat{\mathbf{x}}_i} \\ &= \nabla_{\hat{\mathbf{x}}_i} \{(\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)\} \\ &= 2(\hat{\mathbf{x}}_i - \mathbf{x}_i)\end{aligned}$$

AUTOENCODERS – LEARNING



What value of \hat{x}_{ij} will minimize this function?

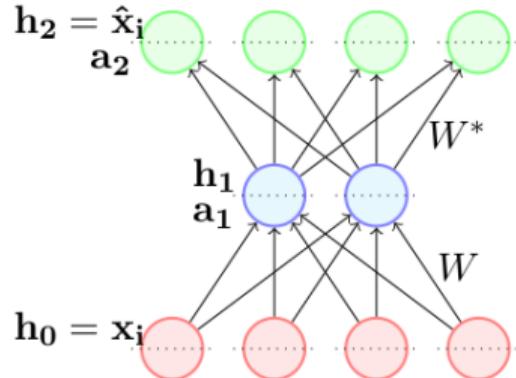
- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

Indeed the above function will be minimized when $\hat{x}_{ij} = x_{ij}$!

- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function
$$\min\left\{-\sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))\right\}$$
- Again we need a formula for $\frac{\partial \mathcal{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ to use backpropagation

AUTOENCODERS – LEARNING

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



$$\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} = \begin{pmatrix} \frac{\partial \mathcal{L}(\theta)}{\partial h_{21}} \\ \frac{\partial \mathcal{L}(\theta)}{\partial h_{22}} \\ \vdots \\ \frac{\partial \mathcal{L}(\theta)}{\partial h_{2n}} \end{pmatrix}$$

- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1}} \boxed{\frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1}} \boxed{\frac{\partial \mathbf{a}_1}{\partial W}}$

- We have already seen how to calculate the expressions in the square boxes when we learnt BP
- The first two terms on RHS can be computed as:

$$\frac{\partial \mathcal{L}(\theta)}{\partial h_{2j}} = -\frac{x_{ij}}{\hat{x}_{ij}} + \frac{1 - x_{ij}}{1 - \hat{x}_{ij}}$$

$$\frac{\partial h_{2j}}{\partial a_{2j}} = \sigma(a_{2j})(1 - \sigma(a_{2j}))$$

AUTOENCODERS AND PCA

- The encoder of a linear autoencoder is equivalent to PCA if we
 - ▶ use a linear encoder
 - ▶ use a linear decoder
 - ▶ use a squared error loss function
 - ▶ normalize the inputs to

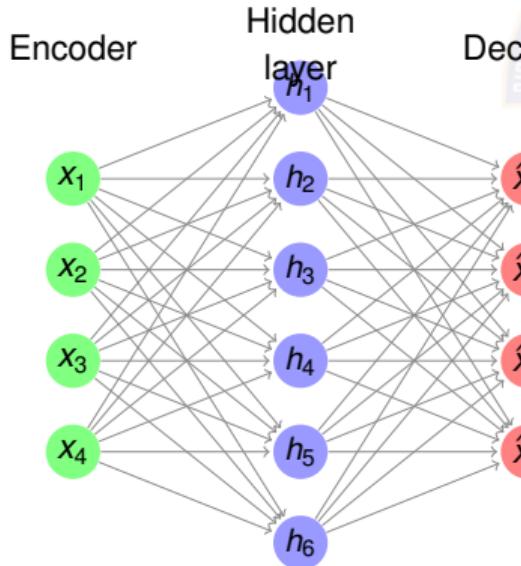
$$r_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj} \right)$$

OVERCOMPLETE AUTOENCODERS

- One hidden layer which has more units than input layer.

$$\dim(h) \geq \dim(x)$$

- Learn a trivial encoding by simply copying x_i into h and then copying h into \hat{x}_i .



$$h = g(W_e x_i + b)$$

$$\hat{x}_i = f(W_d h + c)$$

IN THIS SEGMENT

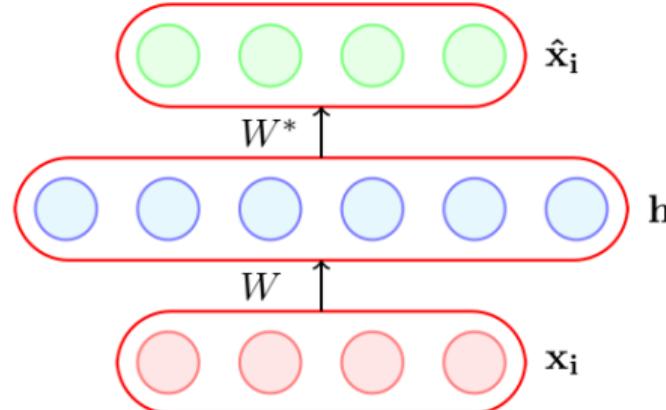
- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS



REGULARIZED AUTOENCODERS

- Autoencoders fail to learn anything useful if the
 - ▶ the encoder and decoder are given too much capacity.
 - ▶ the hidden code is allowed to have dimension equal to the input.
 - ▶ the hidden code has dimension greater than the input. (overcomplete case)
- Use Regularized autoencoders to resolve the above problems.
- To avoid poor generalization, introduce regularization.
- Introduces properties
 - ▶ sparsity of the representation
 - ▶ smallness of the derivative of the representation
 - ▶ robustness to noise or to missing inputs.

REGULARIZED AUTOENCODERS – L2

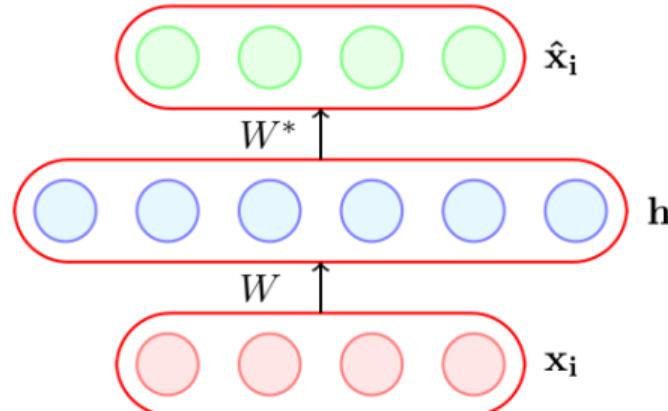


- The simplest solution is to add a L_2 -regularization term to the objective function

$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

- This is very easy to implement and just adds a term λW to the gradient $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ (and similarly for other parameters)

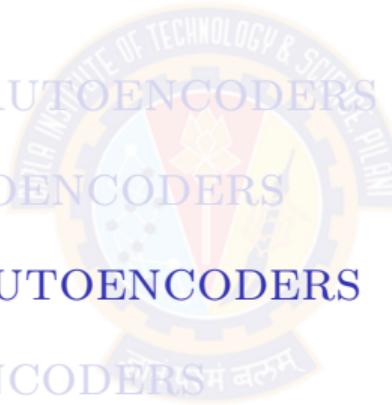
REGULARIZED AUTOENCODERS – WEIGHT SHARING



- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$
- This effectively reduces the capacity of Autoencoder and acts as a regularizer

IN THIS SEGMENT

- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS



CONVOLUTIONAL AUTOENCODERS

- Allows us to increase the size of the output feature map compared to the input feature map.
- Fractionally strided convolution is another term.
- **Strides** In transposed convolutions, we stride over the output; hence, larger strides will result in larger outputs (opposite to regular convolutions).

CONVOLUTIONAL AUTOENCODERS

Regular Convolution:

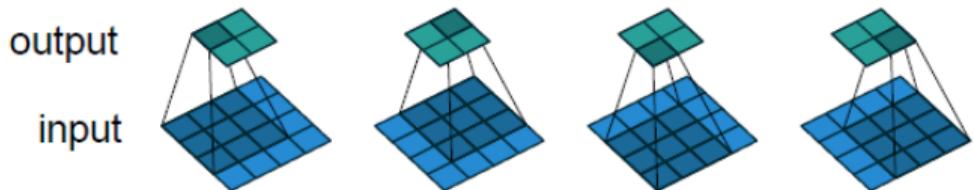
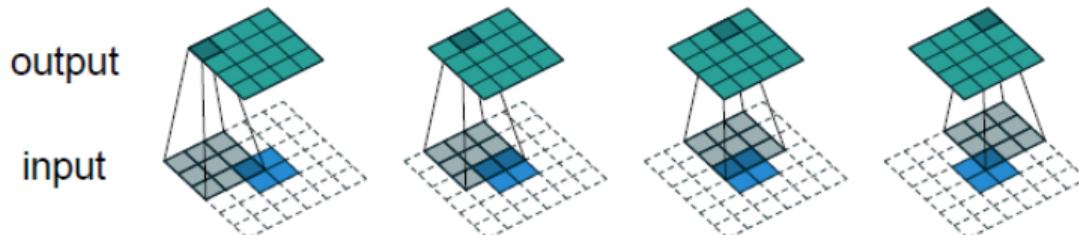


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Transposed Convolution (emulated with direct convolution):



Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

CONVOLUTIONAL AUTOENCODERS

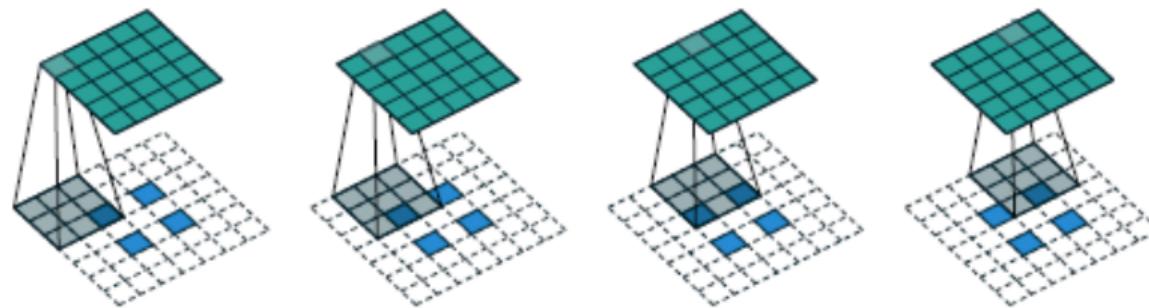
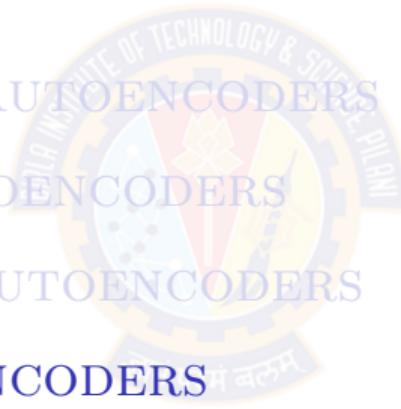


Figure 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5, k = 3, s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides

Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." arXiv preprint arXiv:1603.07285 (2016).

IN THIS SEGMENT

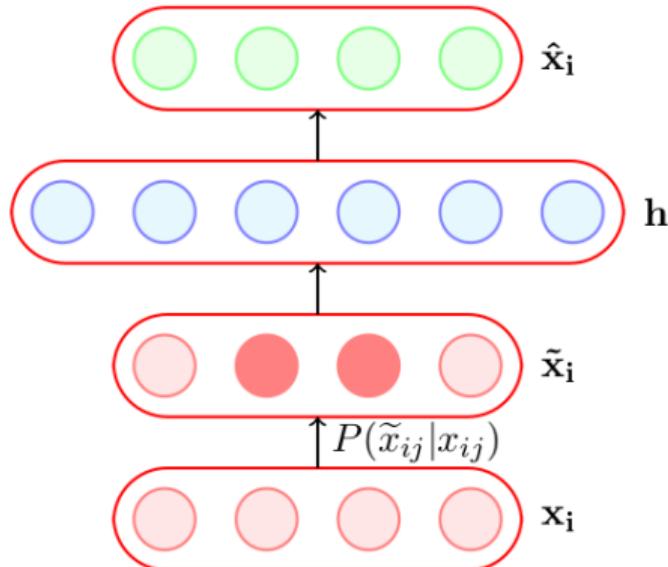
- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS



DENOISING AUTOENCODERS (DAE)

- Receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
- \tilde{x} is a copy of x corrupted by some form of noise.
- The corruption process adds some noise to x according to the conditional distribution $C(\tilde{x}, x)$.
- Denoising autoencoders removes this corruption.

DENOISING AUTOENCODER



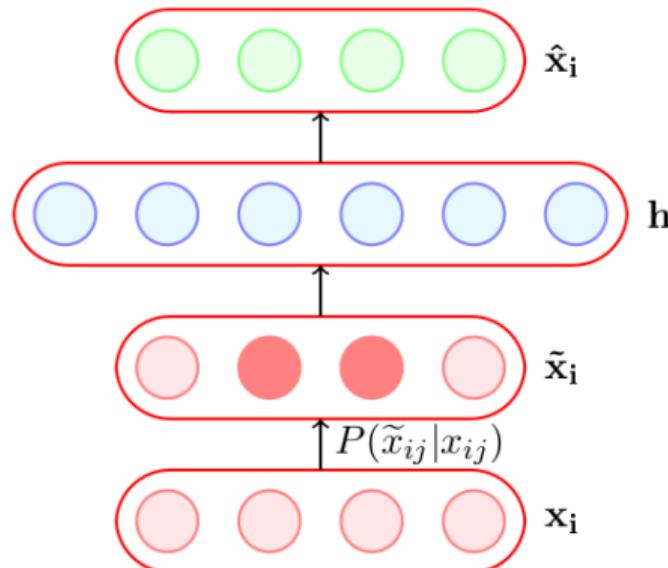
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\tilde{x}_{ij}|x_{ij})$ used in practice is the following

$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$

$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

- In other words, with probability q the input is flipped to 0 and with probability $(1 - q)$ it is retained as it is

DENOISING AUTOENCODER



For example, it will have to learn to reconstruct a corrupted x_{ij} correctly by relying on its interactions with other elements of \mathbf{x}_i .

- How does this help ?
- This helps because the objective is still to reconstruct the original (uncorrupted) \mathbf{x}_i

$$\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted $\tilde{\mathbf{x}}_i$ into $h(\tilde{\mathbf{x}}_i)$ and then into $\hat{\mathbf{x}}_i$ (the objective function will not be minimized by doing so)
- Instead the model will now have to capture the characteristics of the data correctly.

DENOISING AUTOENCODER

Task: Hand-written digit
recognition

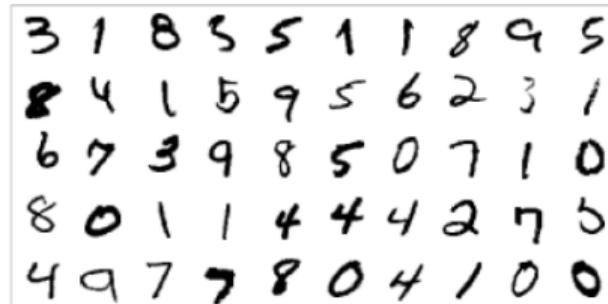


Figure: MNIST Data

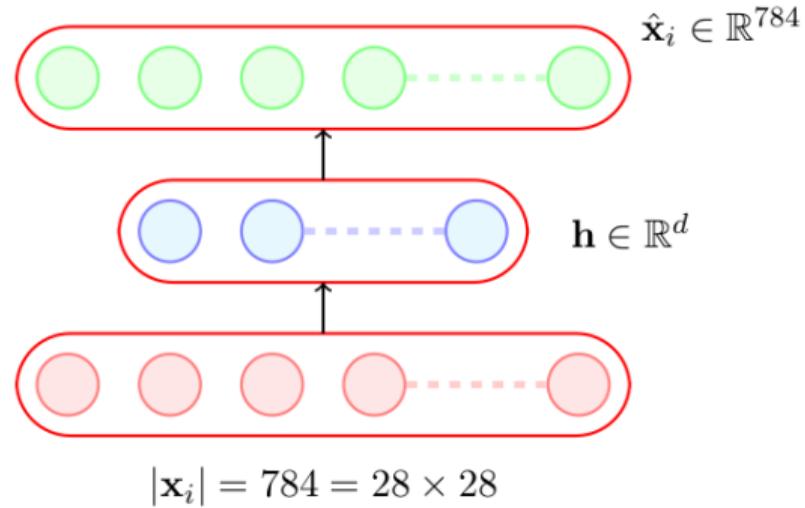


Figure: AE approach (first learn important characteristics of data)

DENOISING AUTOENCODER

Task: Hand-written digit recognition

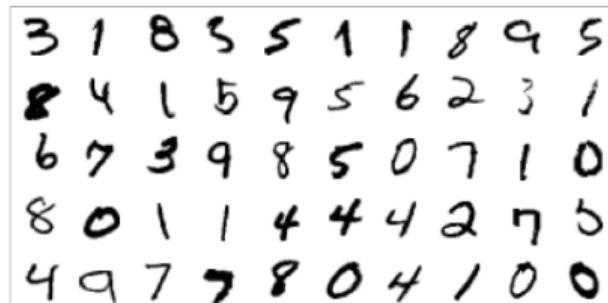


Figure: MNIST Data

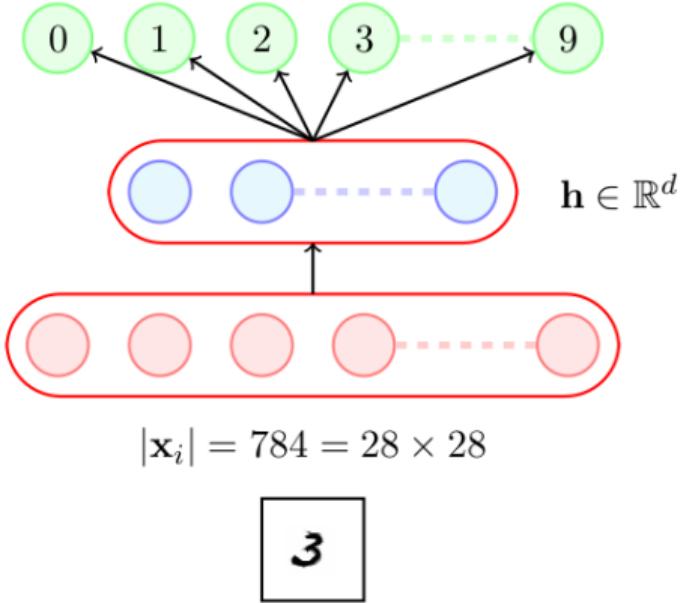


Figure: AE approach (and then train a classifier on top of this hidden representation)

DENOISING AUTOENCODER

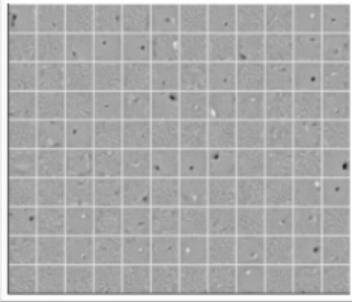


Figure: Vanilla AE
(No noise)

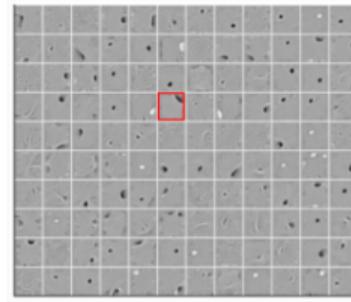


Figure: 25% Denoising
AE ($q=0.25$)

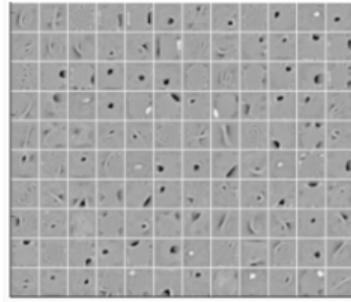
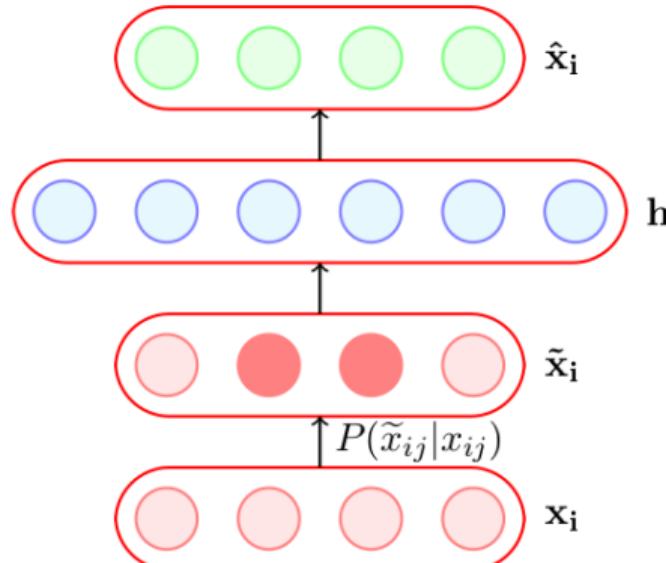


Figure: 50% Denoising
AE ($q=0.5$)

- The vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a '0' or a '2' or a '3' or a '8' or a '9')
- As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke

DENOISING AUTOENCODER



- We saw one form of $P(\tilde{x}_{ij}|x_{ij})$ which flips a fraction q of the inputs to zero
- Another way of corrupting the inputs is to add a Gaussian noise to the input

$$\tilde{x}_{ij} = x_{ij} + \mathcal{N}(0, 1)$$

- We will now use such a denoising AE on a different dataset and see their performance

DENOISING AUTOENCODER

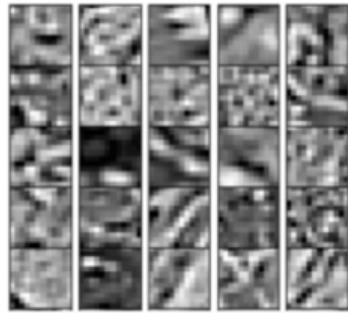


Figure: Data

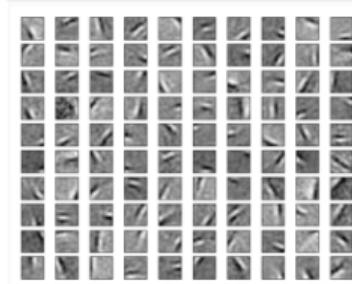


Figure: AE filters

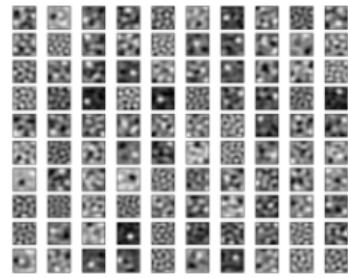


Figure: Weight decay filters

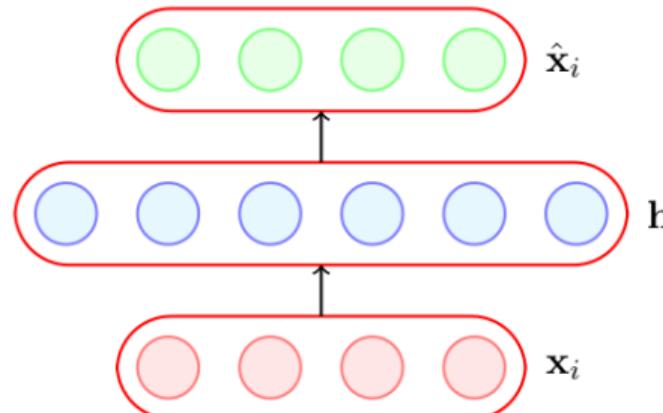
- The hidden neurons essentially behave like edge detectors
- PCA does not give such edge detectors

IN THIS SEGMENT

- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS

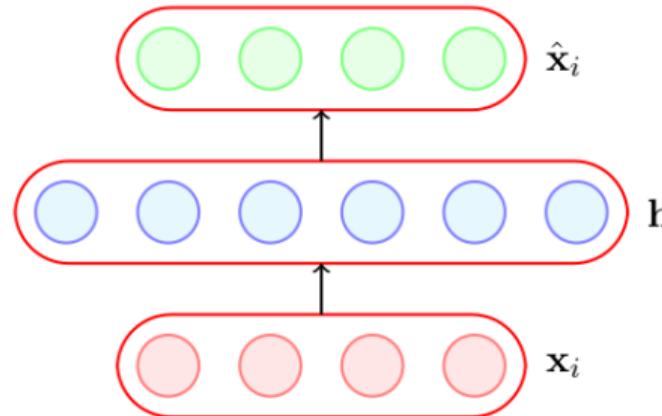


SPARSE AUTOENCODER



- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure the neuron is inactive most of the times.

SPARSE AUTOENCODER



The average value of the activation of a neuron l is given by

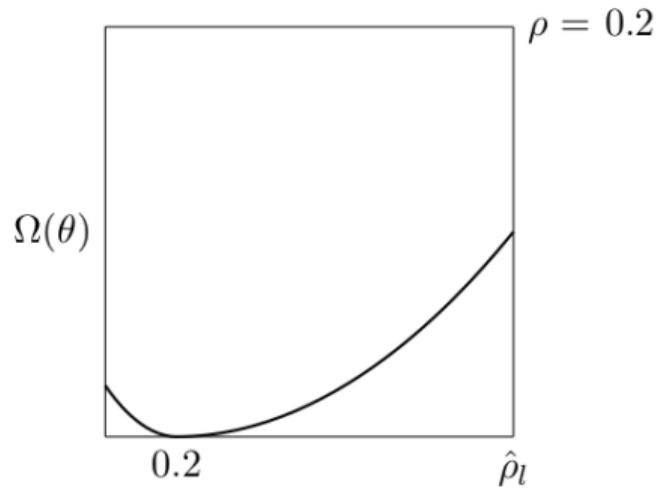
$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$

- When will this term reach its minimum value and what is the minimum value? Let us plot it and check.

SPARSE AUTOENCODER



- The function will reach its minimum value(s) when $\hat{\rho}_l = \rho$.

SPARSE AUTOENCODER

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$

Can be re-written as

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l)$$

By Chain rule:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W}$$

$$\frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[\frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \dots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T$$

For each neuron $l \in 1 \dots k$ in hidden layer, we have

$$\frac{\partial \Omega(\theta)}{\partial \hat{\rho}_l} = -\frac{\rho}{\hat{\rho}_l} + \frac{(1 - \rho)}{1 - \hat{\rho}_l}$$

and $\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$ (see next slide)

- Now,

$$\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta)$$

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- Let us see how to calculate $\frac{\partial \Omega(\theta)}{\partial W}$.
- Finally,

$$\frac{\partial \hat{\mathcal{L}}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

(and we know how to calculate both terms on R.H.S)

IN THIS SEGMENT

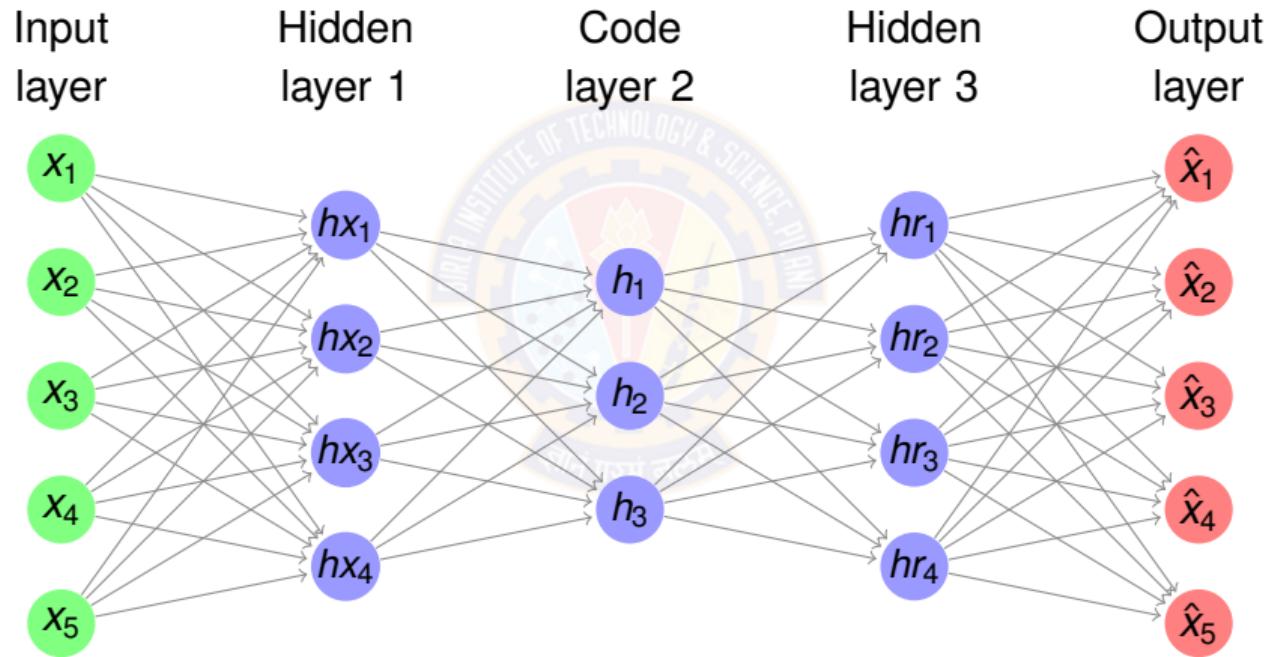
- ① SUPERVISED VS UNSUPERVISED LEARNING EXPERIENCE
- ② AUTOENCODERS
- ③ UNDERCOMPLETE AUTOENCODERS
- ④ REGULARIZED AUTOENCODERS
- ⑤ CONVOLUTIONAL AUTOENCODERS
- ⑥ DENOISING AUTOENCODERS
- ⑦ SPARSE AUTOENCODERS
- ⑧ DEEP AUTOENCODERS



DEEP AUTOENCODERS

- Input and output layers have hidden units.
- Code layer has lesser units than input layer.
- Depth can exponentially reduce the computational cost of representing some functions.
- Depth can exponentially decrease the amount of training data needed to learn some functions.
- Deep autoencoders yield much better compression than corresponding shallow or linear autoencoders .

DEEP AUTOENCODERS



AUTOENCODER APPLICATIONS

- Dimensionality reduction (representation learning)
- Information retrieval/semantic hashing tasks - we can store all database entries in a hash table that maps binary code vectors to entries
- Classification
- Denoising autoencoders
- Useful for segmentation and deep-feature
- Neural inpainting

AUTOENCODER APPLICATIONS

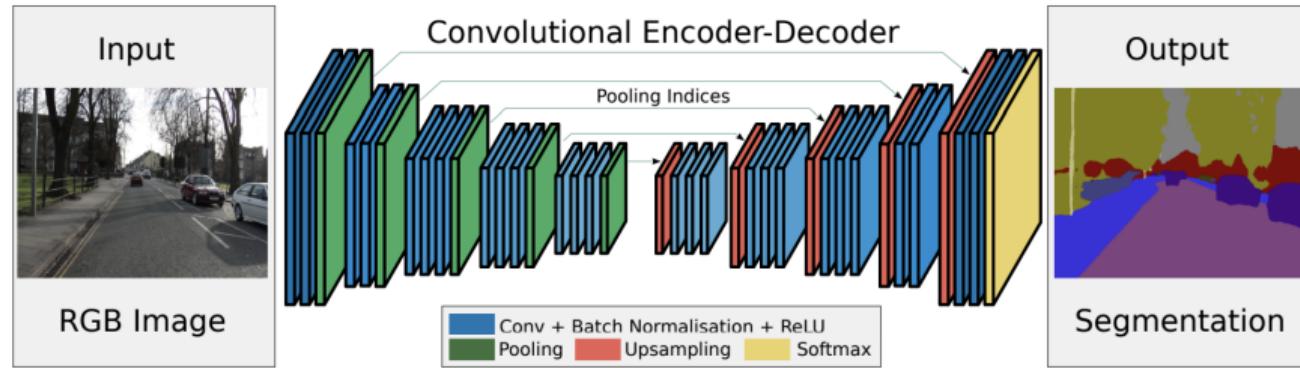


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

AUTOENCODER APPLICATIONS

A comparison of methods for compressing digit images to 30 real numbers.

0	/	2	3	4	5	6	7	8	9	real data
0	/	2	3	4	5	6	7	8	9	30-D deep auto
0	/	2	3	4	5	6	7	8	9	30-D logistic PCA
0	/	2	3	4	5	6	7	8	9	30-D PCA

AUTOENCODER – SUMMARY

- An autoencoder network is actually a pair of two connected networks, an encoder and a decoder.
- An encoder network takes in an input, and converts it into a smaller, dense representation.
- A decoder network can use to convert the dense representation back to the original input.



References

- ① Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>
- ② Deep Learning with Python by Francois Chollet.
<https://livebook.manning.com/book/deep-learning-with-python/>

Thank You!

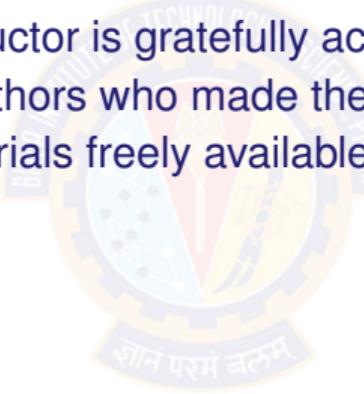


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

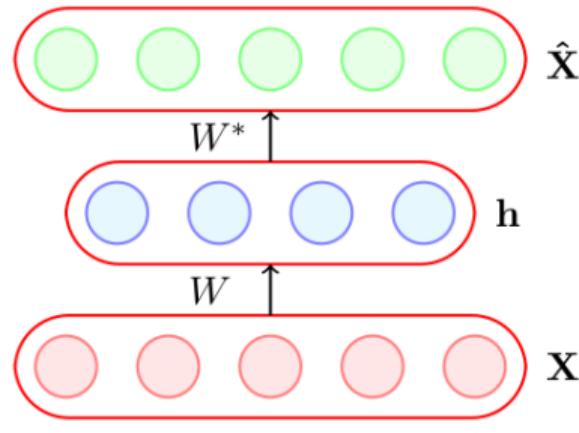
DEEP LEARNING MODULE 7 : VARIATIONAL AUTOENCODERS

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.



AUTOENCODER – SUMMARY



$$h = g(WX + b)$$

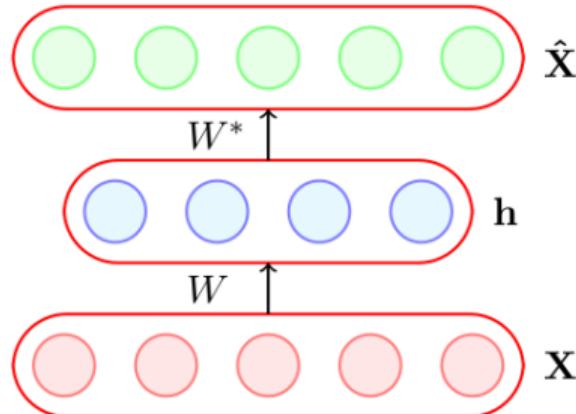
$$\hat{X} = f(W^*h + c)$$

- Before we start talking about VAEs, let us quickly revisit autoencoders
- An autoencoder contains an encoder which takes the input X and maps it to a hidden representation
- The decoder then takes this hidden representation and tries to reconstruct the input from it as \hat{X}
- The training happens using the following objective function

$$\min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- where m is the number of training instances, $\{x_i\}_{i=1}^m$ and each $x_i \in R^n$ (x_{ij} is thus the j -th dimension of the i -th training instance)

AUTOENCODER – SUMMARY

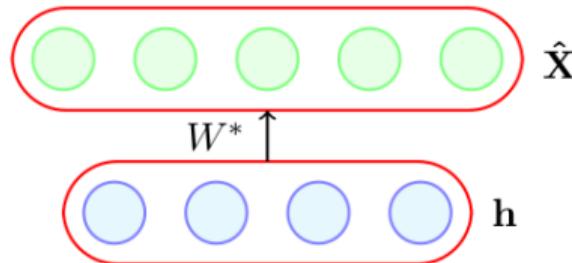


$$\mathbf{h} = g(\mathbf{W}\mathbf{X} + \mathbf{b})$$

$$\hat{\mathbf{X}} = f(\mathbf{W}^*\mathbf{h} + \mathbf{c})$$

- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a \hat{X} from it ?
- In principle, yes! But in practice there is a problem with this approach
- h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input

AUTOENCODER – SUMMARY



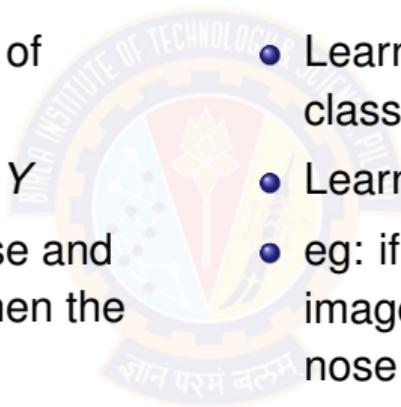
$$\hat{\mathbf{X}} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*
- In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h 's which have a high probability
- But unlike RBMs, autoencoders do not have such a probabilistic interpretation
- They learn a hidden representation h but not a distribution $P(h|X)$
- Similarly the decoder is also deterministic and does not learn a distribution over X (given a h we can get a X but not $P(X|h)$)

DISCRIMINATIVE MODELS VS GENERATIVE MODELS

DISCRIMINATIVE MODELS

- Set of features X and Set of categories Y
- Learn mapping from X to Y
- eg: if features are wet nose and tongue out and no purr; then the per is a dog.
- Model $P(Y | X)$



GENERATIVE MODELS

- Learn realistic representation of a class
- Learn mapping from Y to X
- eg: if we have to generate a dog image, then it should have wet nose and tongue out.
- Model $P(X | Y)$

VARIATIONAL AUTOENCODER

- Generative model
- Variational autoencoder generates a latent representation and then uses this representation to generate new images.
- Data are assumed to be represented by a set of normally-distributed latent factors.
- Encoder generates parameters of these parameters namely μ and σ .
- Images are generated by sampling from these distributions.

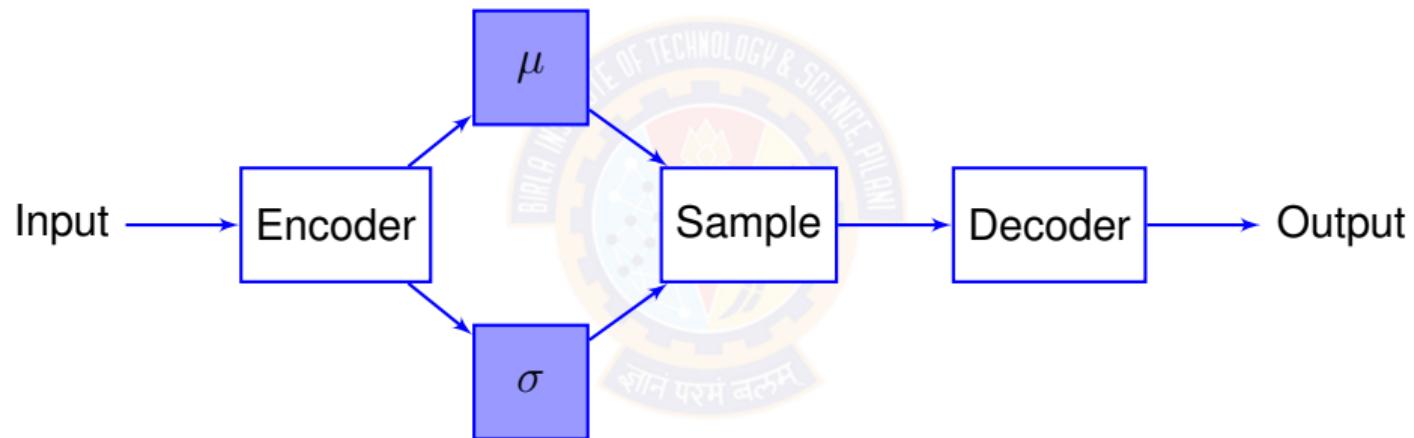
VARIATIONAL AUTOENCODER

- Primary Goal: Generate images using decoder.
- Latent vector: each element drawn from a normal distribution.
- Parameters are learned by the encoder.
- Secondary Goal: Similar images be close together in latent space.



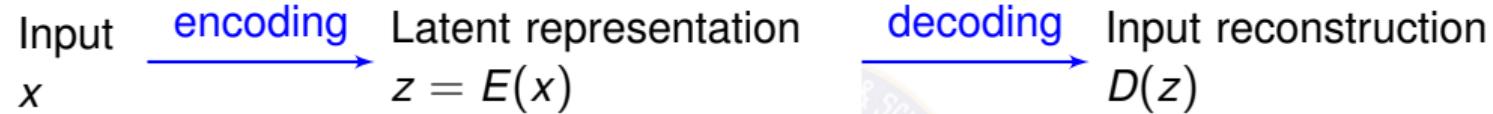
IBM

VARIATIONAL AUTOENCODER



AUTOENCODER VS VARIATIONAL AUTOENCODER

AUTOENCODER

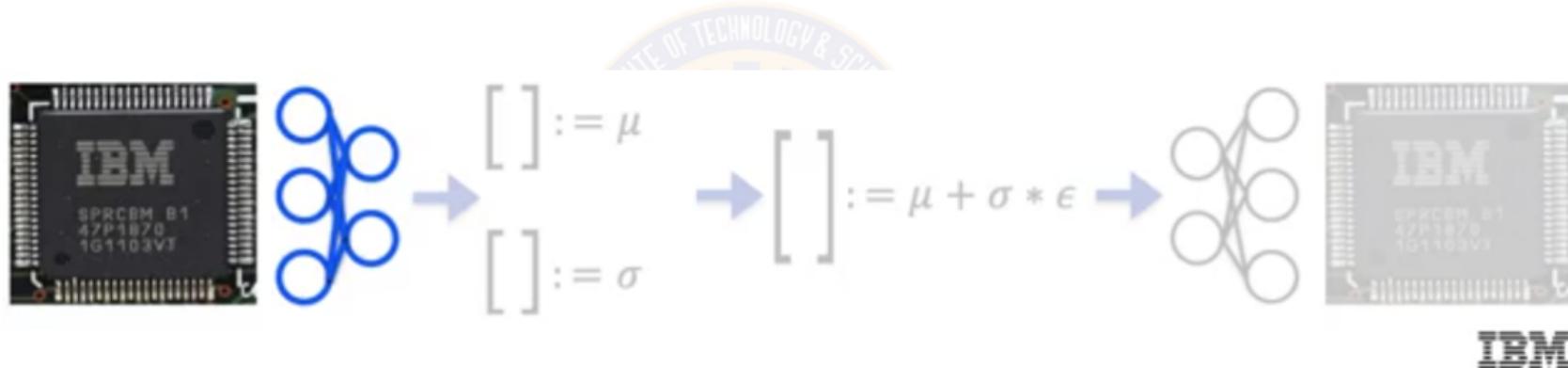


VARIATIONAL AUTOENCODER



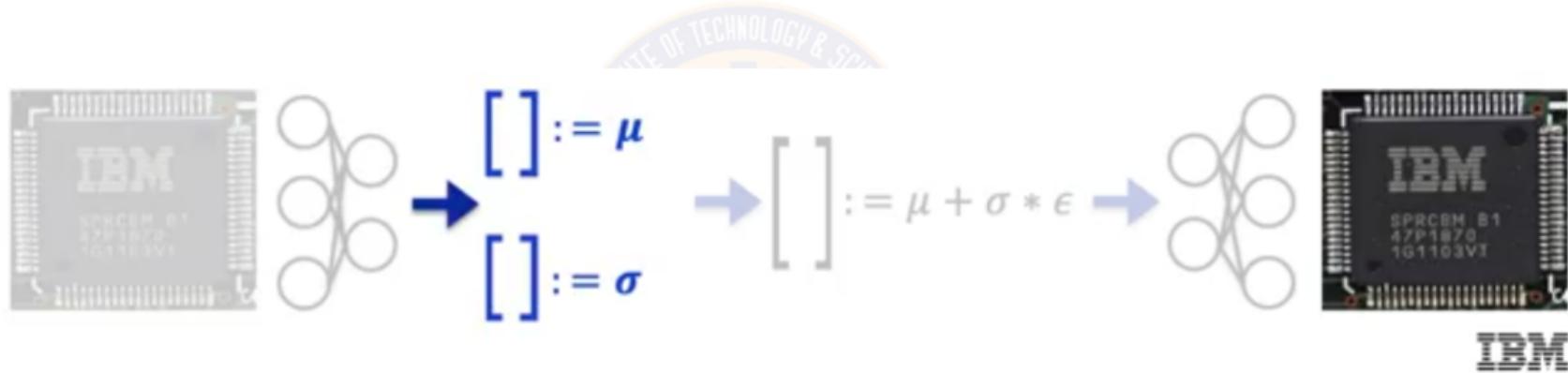
VARIATIONAL AUTOENCODER

- Step 1 : Image is fed through the encoder network.



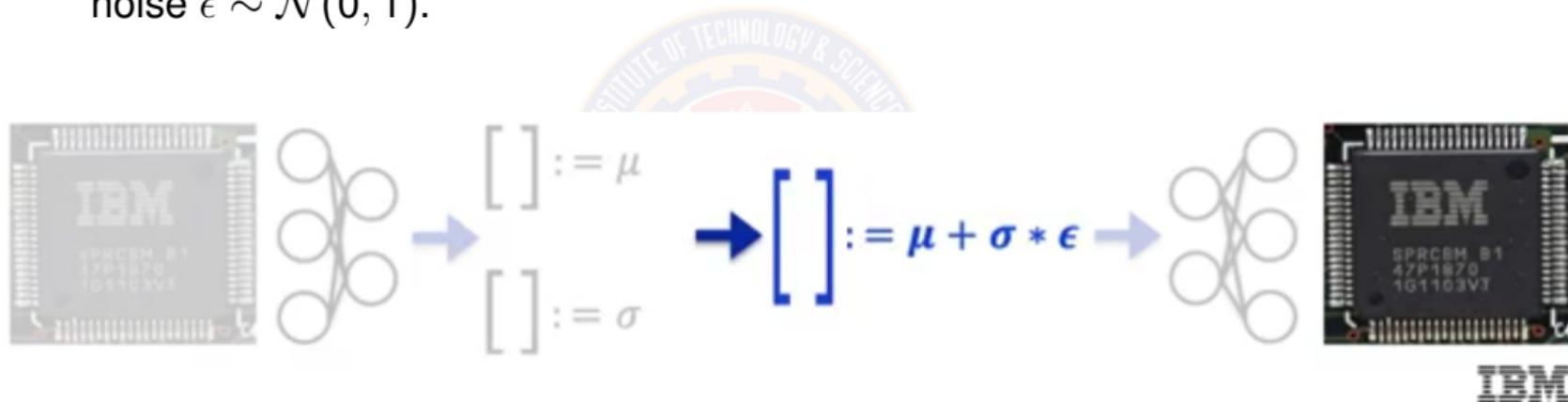
VARIATIONAL AUTOENCODER

- Step 2 : The encoder network learns the parameters μ and σ .



VARIATIONAL AUTOENCODER

- Step 3 : Combine the parameters μ and σ and normally distributed random noise $\epsilon \sim \mathcal{N}(0, 1)$.



VARIATIONAL AUTOENCODER EXAMPLE

Let $\mu = \begin{bmatrix} 0.7 \\ -0.6 \end{bmatrix}$

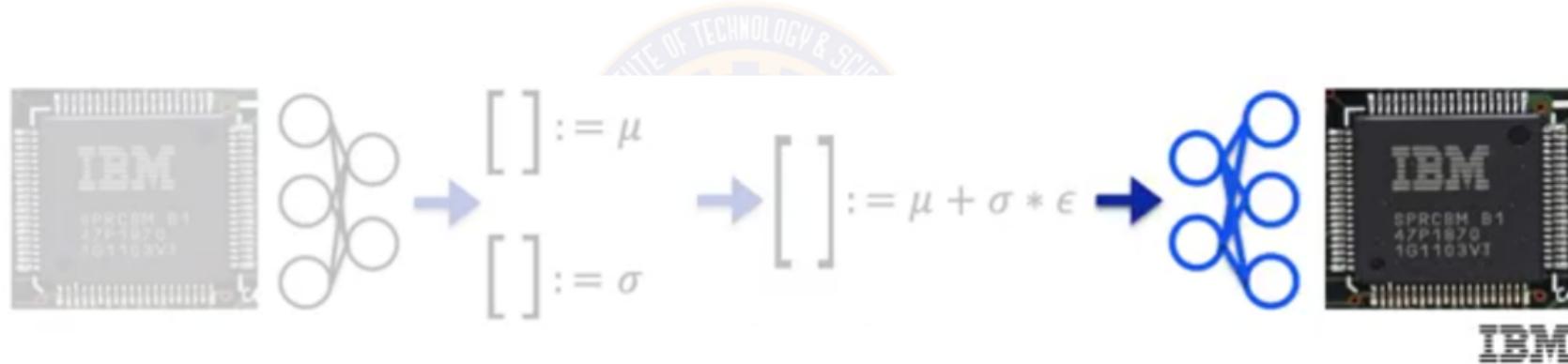
$$\sigma = \begin{bmatrix} 1.0 \\ 0.6 \end{bmatrix}$$

$$\epsilon = \begin{bmatrix} 1.4 \\ 1.9 \end{bmatrix}$$

$$\begin{aligned} Decoder_{in} &= \mu + \sigma * \epsilon \\ &= \begin{bmatrix} 0.7 \\ -0.6 \end{bmatrix} + \begin{bmatrix} 1.0 \\ 0.6 \end{bmatrix} * \begin{bmatrix} 1.4 \\ 1.9 \end{bmatrix} \\ &= \begin{bmatrix} 2.10 \\ 0.54 \end{bmatrix} \end{aligned}$$

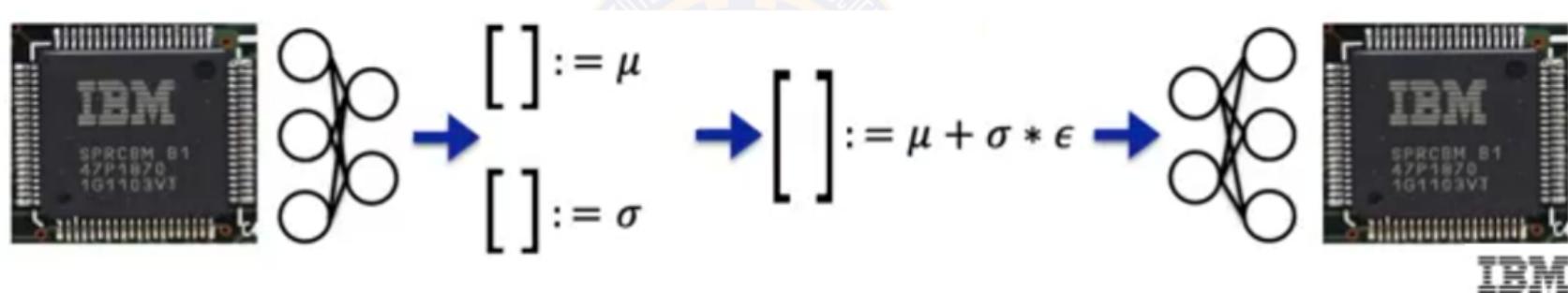
VARIATIONAL AUTOENCODER

- Step 4 : Feed the vector through the decoder.



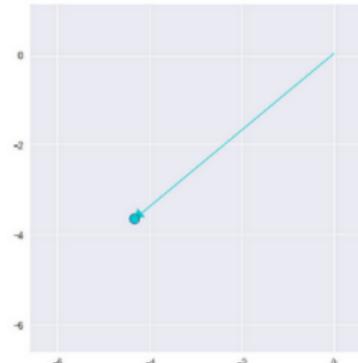
VARIATIONAL AUTOENCODER

- Step 5 : The reconstructed image is generated using the decoder.

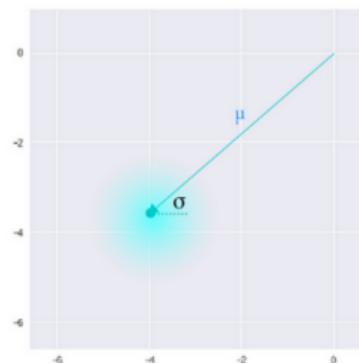


VARIATIONAL AUTOENCODER

- Encoder learns two encodings or outputs two vectors of size n
 - a vector of means μ – controls where the encoding of an input should be centered around.
 - a vector of standard deviations σ – controls (area) how much from the mean the encoding can vary.
- Decoder learns that a single point in latent space refers to a sample of that class. It also learns that all nearby points refer to the same class as well.



Standard Autoencoder
(direct encoding coordinates)



Variational Autoencoder
(μ and σ initialize a probability distribution)

VARIATIONAL AUTOENCODER – LOSS FUNCTION

- VAE should learn to reconstruct the original images.
- The loss function will have 2 components.
 - ① A penalty for NOT reconstructing the image correctly.
 - ★ Pixel-wise difference between the reconstructed image and the original image.
 - ★ Use Mean squared error (MSE)
 - ★ Use binary cross entropy
 - ② A penalty for generating parameter vectors μ and σ that are different than 0 and 1, respectively.
 - ★ The difference between vectors produced by the encoder and the standard normal distribution parameters $\mathcal{N}(0, 1)$.
 - ★ Use Kullback–Leibler divergence (KL divergence).

KULLBACK–LEIBLER DIVERGENCE

- The KL divergence between two probability distributions measures how much they diverge from each other.
- Minimizing the KL divergence means optimizing the probability distribution parameters (μ and σ) to closely resemble that of the target distribution.
- KL loss is equivalent to the sum of all the KL divergences between the component $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ in X , and the standard normal $\mathcal{N}(\mu = 0, \sigma = 1)$

$$KL\ loss = \sum_{i=1}^n \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

KL DIVERGENCE EXAMPLE

Compare $\mu = \begin{bmatrix} 0.7 \\ -0.6 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Compare $\log(\sigma) = \begin{bmatrix} 1.0 \\ 0.6 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

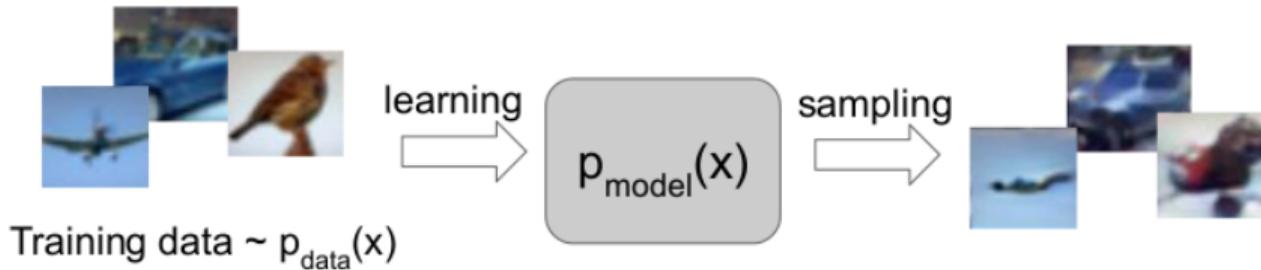
$\log(\sigma)$ compared to 0 since $\log(1) = 0$.

σ is interpreted as $\log(\sigma)$ since variance cannot be negative.

$$\text{KL Loss} = \frac{1}{2} * (e^{\log(\sigma)} - \log(\sigma) - 1 + \mu^2)$$

GENERATIVE MODEL

Given training data, generate new samples from same distribution

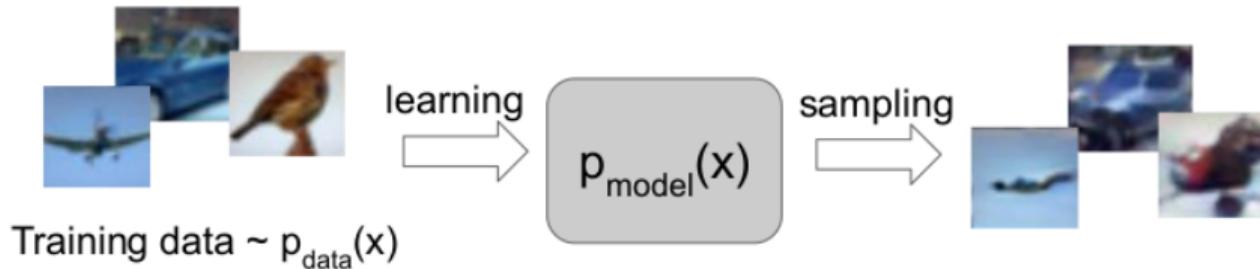


Objectives:

1. Learn $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
2. **Sampling new x from $p_{\text{model}}(x)$**

GENERATIVE MODEL

Given training data, generate new samples from same distribution



Formulate as density estimation problems:

- **Explicit density estimation:** explicitly define and solve for $p_{\text{model}}(x)$
- **Implicit density estimation:** learn model that can sample from $p_{\text{model}}(x)$ **without explicitly defining it.**

GENERATIVE MODEL

Taxonomy of Generative Models

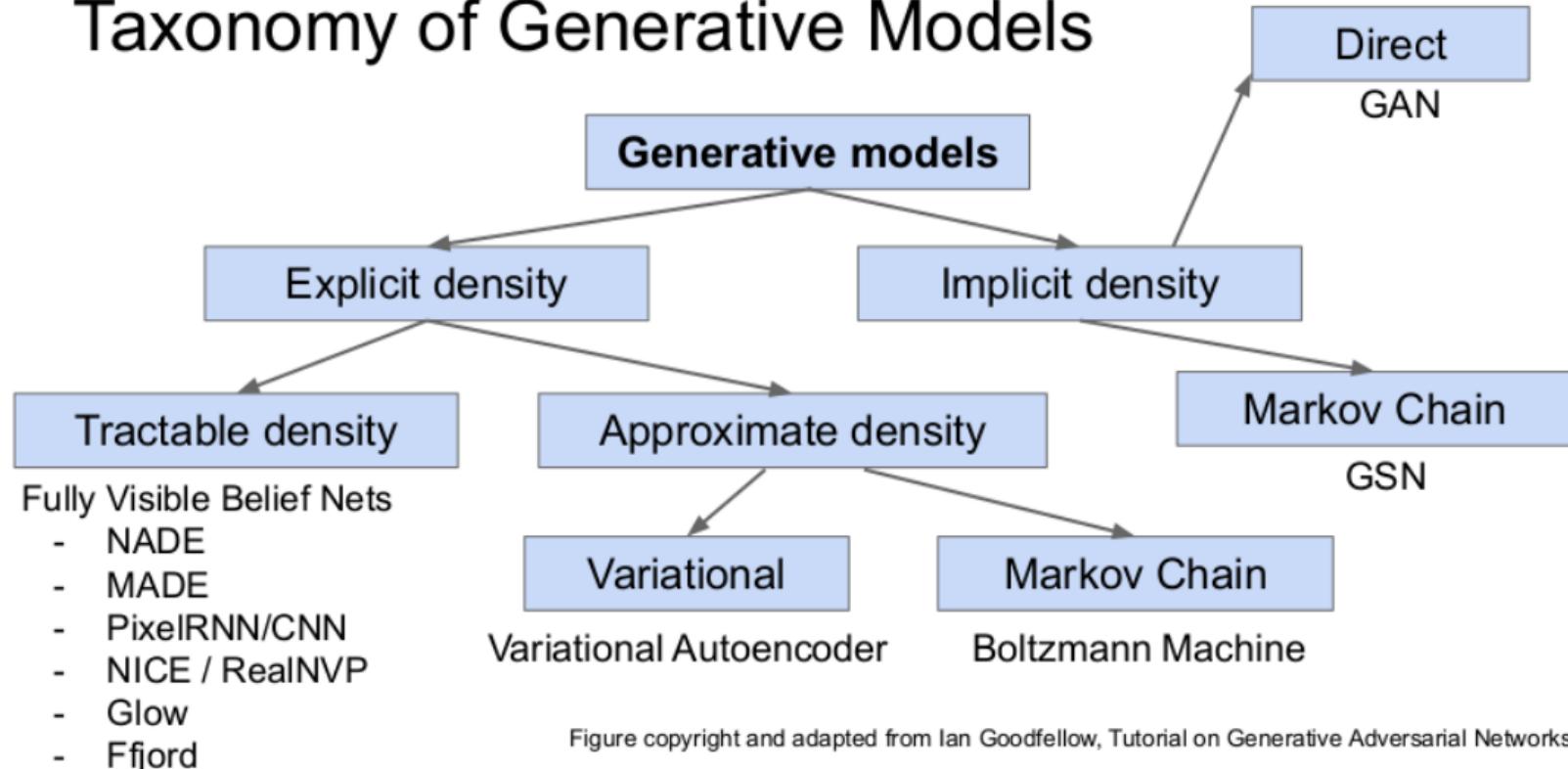


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

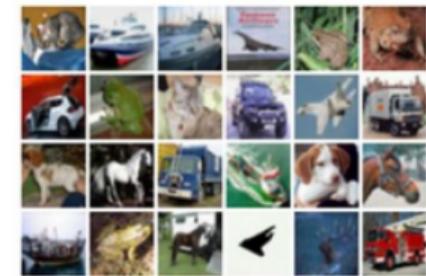
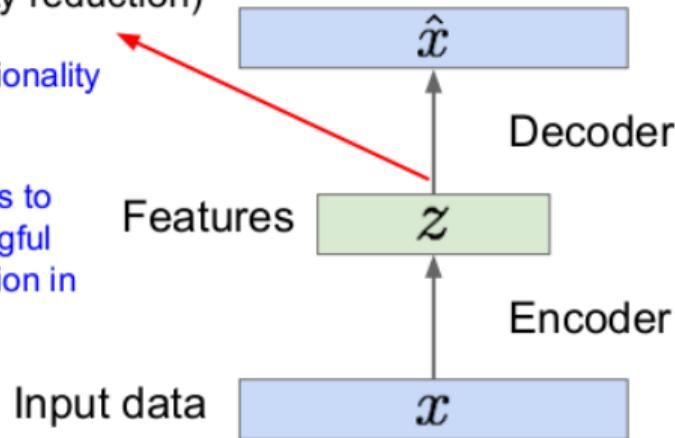
AUTOENCODERS – RECAP

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

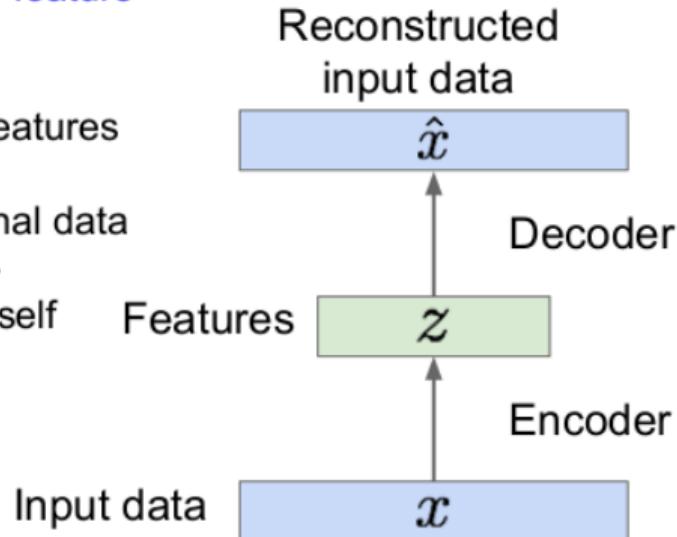
A: Want features to capture meaningful factors of variation in data



AUTOENCODERS – RECAP

How to learn this feature representation?

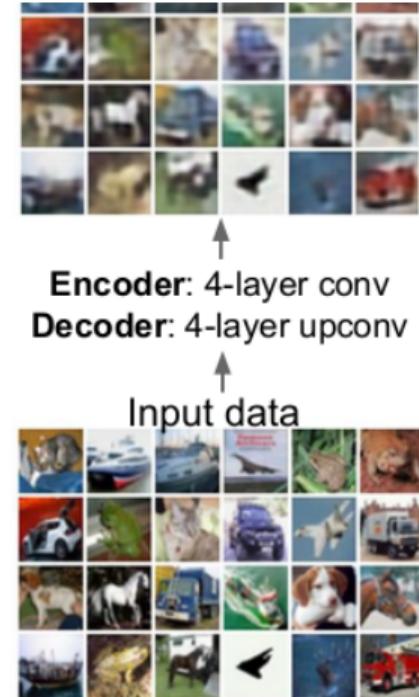
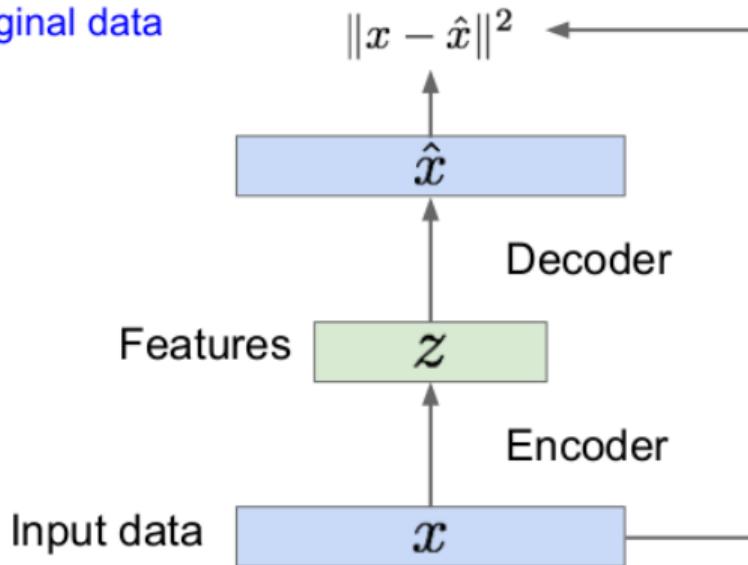
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding input itself



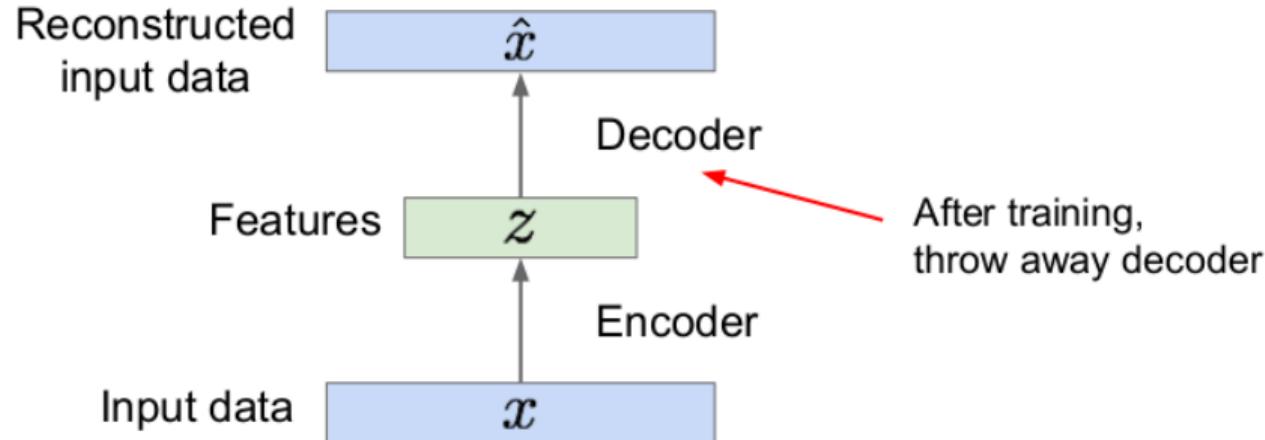
AUTOENCODERS – RECAP

Train such that features can be used to reconstruct original data

L2 Loss function:
Doesn't use labels!



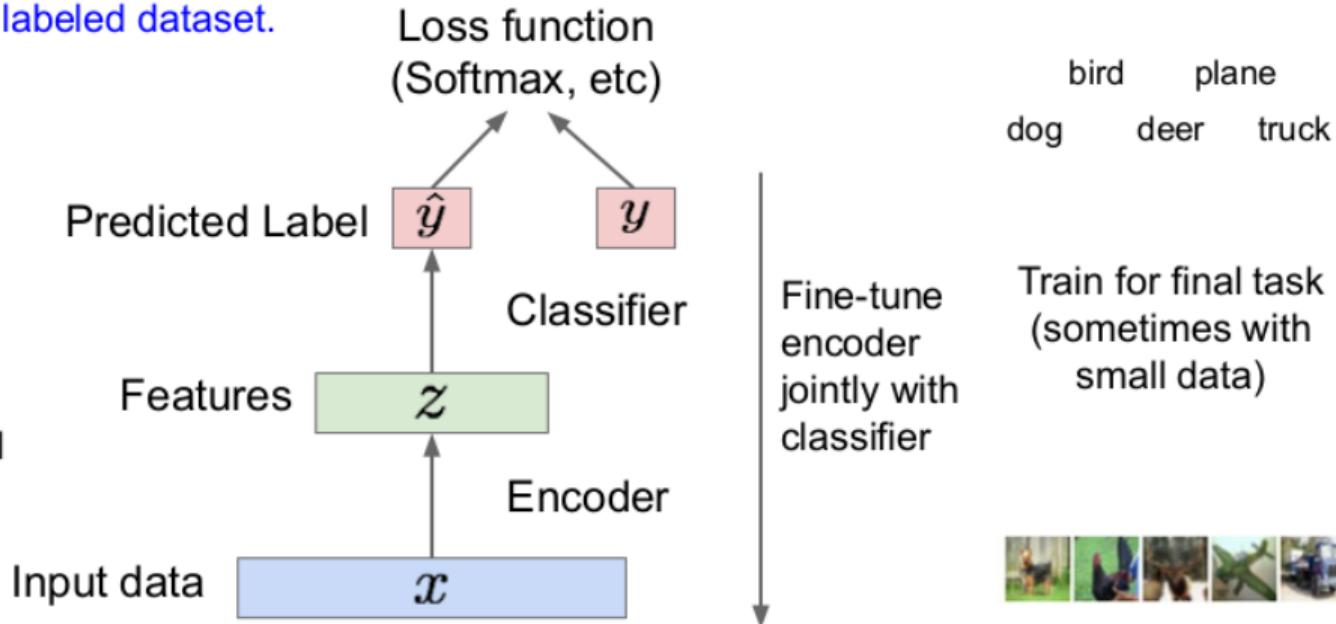
AUTOENCODERS – RECAP



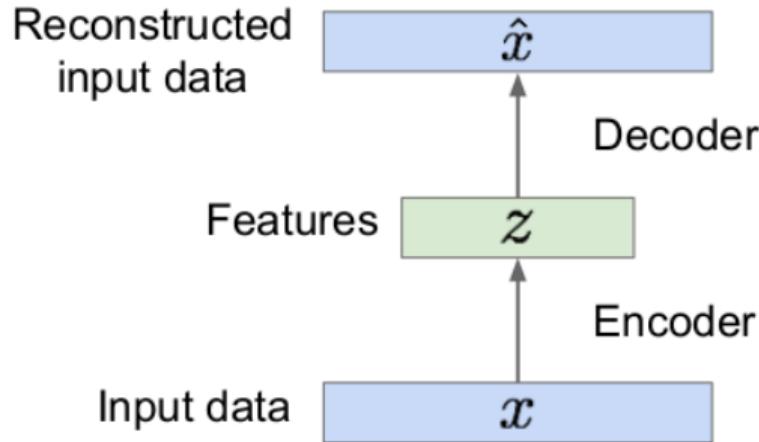
AUTOENCODERS – RECAP

Transfer from large, unlabeled dataset to small, labeled dataset.

Encoder can be used to initialize a **supervised** model



AUTOENCODERS – RECAP



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of z .

How do we make autoencoder a **generative model**?

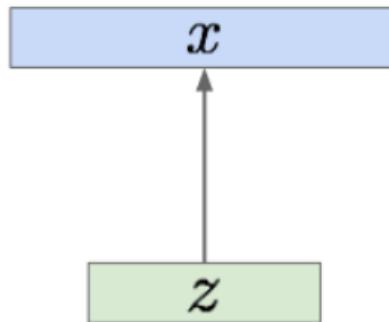
VARIATIONAL AUTOENCODERS

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation z

Sample from
true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$



Sample from
true prior

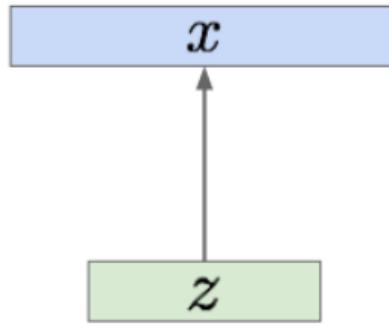
$$z^{(i)} \sim p_{\theta^*}(z)$$

VARIATIONAL AUTOENCODERS

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation \mathbf{z}

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

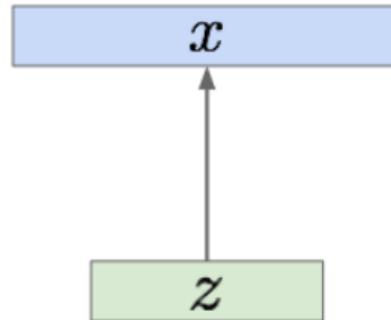
Intuition (remember from autoencoders!):
 \mathbf{x} is an image, \mathbf{z} is latent factors used to
generate \mathbf{x} : attributes, orientation, etc.

VARIATIONAL AUTOENCODERS

We want to estimate the true parameters θ^* of this generative model given training data x .

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

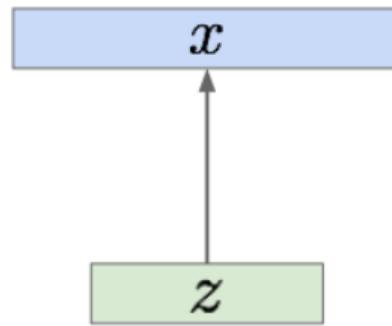


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

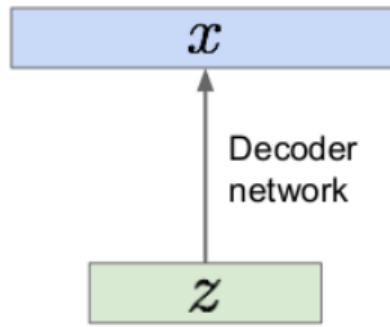
How should we represent this model?

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Learn model parameters to maximize likelihood
of training data

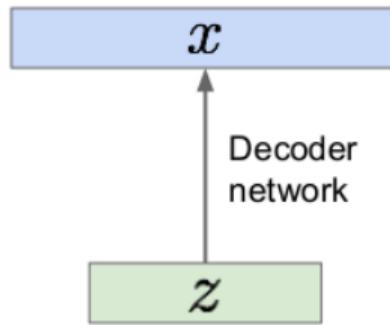
$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Learn model parameters to maximize likelihood
of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Q: What is the problem with this?

Intractable!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS: INTRACTABILITY

Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

↑
Simple Gaussian prior

VARIATIONAL AUTOENCODERS: INTRACTABILITY

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

The diagram illustrates the data likelihood equation for a variational autoencoder. The equation is $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$. Two green checkmarks are placed above the integral sign. A blue arrow points from the text "Decoder neural network" to the term $p_{\theta}(x|z)$, indicating that this term is generated by the decoder neural network.

VARIATIONAL AUTOENCODERS: INTRACTABILITY

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$



Intractable to compute $p(x|z)$ for every z !

VARIATIONAL AUTOENCODERS: INTRACTABILITY

Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Posterior density: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Intractable data likelihood

VARIATIONAL AUTOENCODERS

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density also intractable: $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

Solution: In addition to modeling $p_{\theta}(x|z)$, learn $q_{\phi}(z|x)$ that approximates the true posterior $p_{\theta}(z|x)$.

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data x

VARIATIONAL AUTOENCODERS

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))\end{aligned}$$



Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).



This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!



$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always ≥ 0 .

VARIATIONAL AUTOENCODERS

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)})] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

We want to
maximize the
data
likelihood

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z [\log p_{\theta}(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z [\log p_{\theta}(x^{(i)} | z)] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))$$

Decoder network gives $p_{\theta}(x|z)$, can
compute estimate of this term through
sampling.

This KL term (between
Gaussians for encoder and z
prior) has nice closed-form
solution!

$p_{\theta}(z|x)$ intractable (saw
earlier), can't compute this KL
term :(But we know KL
divergence always ≥ 0 .

VARIATIONAL AUTOENCODERS

We want to maximize the data likelihood

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))}_{\geq 0}\end{aligned}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

VARIATIONAL AUTOENCODERS

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)})] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

Decoder:
reconstruct
the input data

Encoder:
make approximate
posterior distribution
close to prior

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z [\log p_{\theta}(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z [\log p_{\theta}(x^{(i)} | z)] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take
gradient of and optimize! ($p_{\theta}(x|z)$ differentiable,
KL term differentiable)

VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior

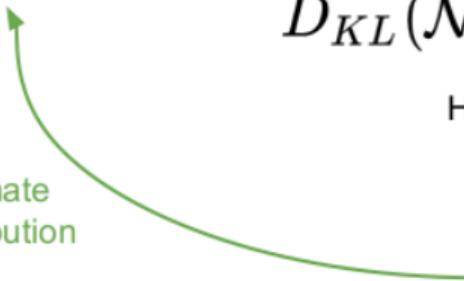
Encoder network

$$q_\phi(z|x)$$

Input Data

$$\mu_{z|x}$$

$$\Sigma_{z|x}$$



VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

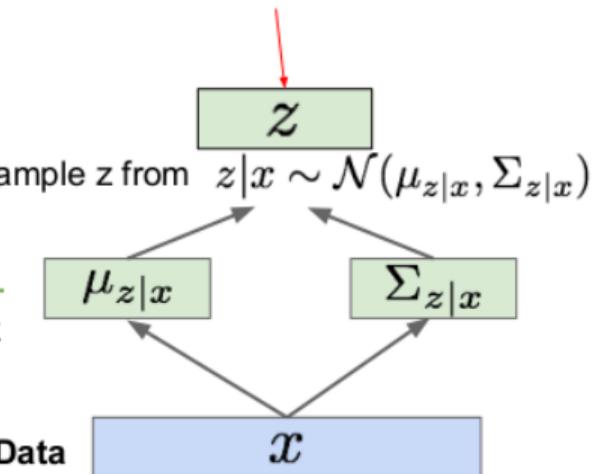
Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data

Not part of the computation graph!



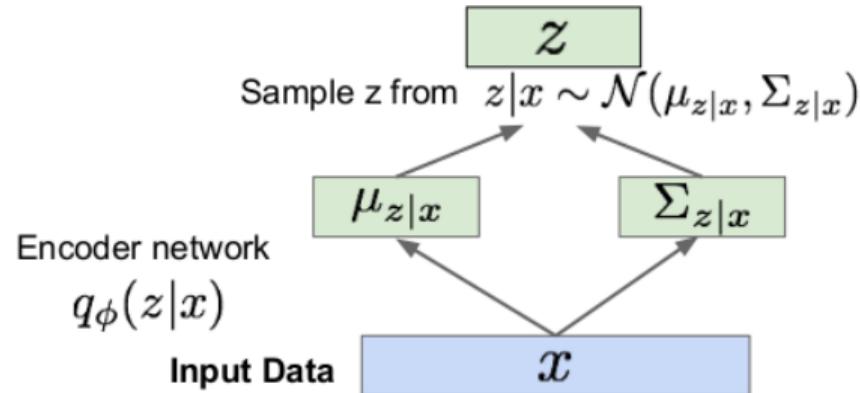
VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

$$\text{Sample } \epsilon \sim \mathcal{N}(0, I)$$

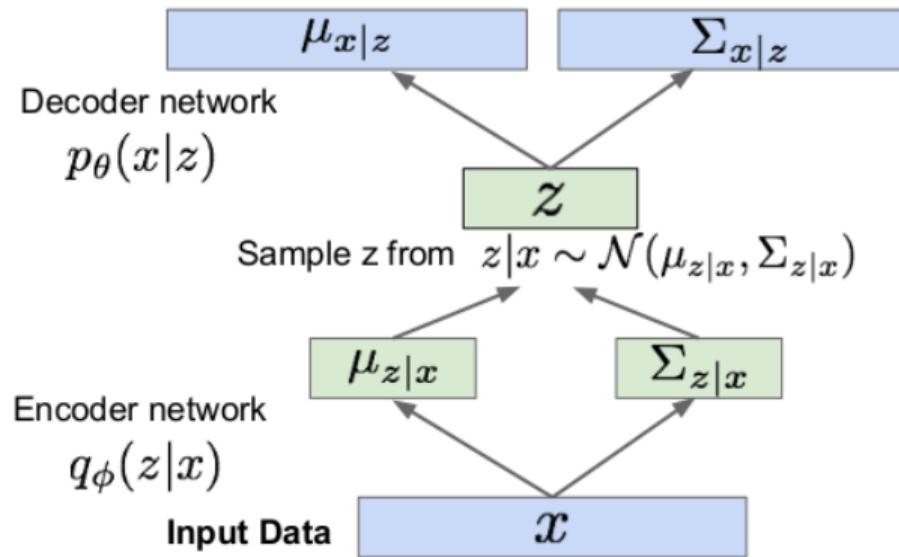
$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$



VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

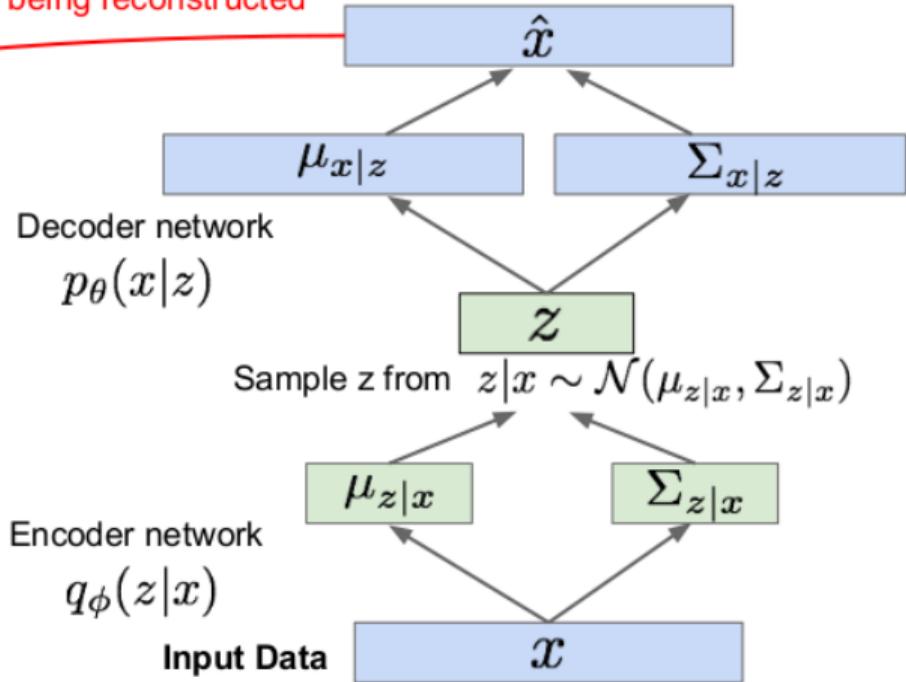


VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Maximize likelihood of original input being reconstructed



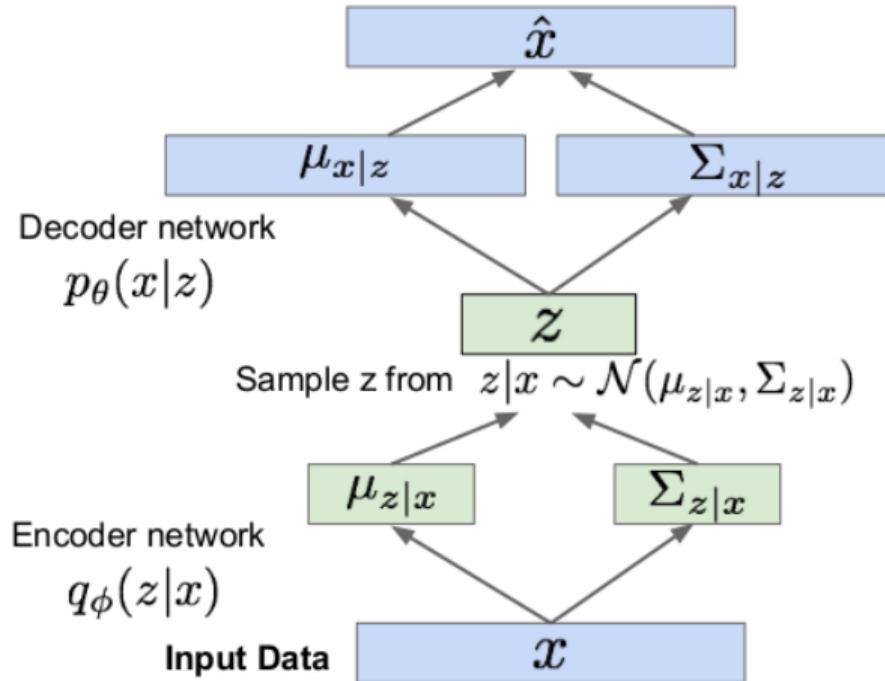
VARIATIONAL AUTOENCODERS

Putting it all together: maximizing the likelihood lower bound

$$\mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

$\mathcal{L}(x^{(i)}, \theta, \phi)$

For every minibatch of input data: compute this forward pass, and then backprop!



VARIATIONAL AUTOENCODERS: GENERATING DATA

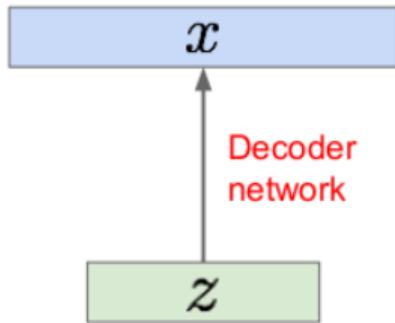
Our assumption about data generation process

Sample from true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



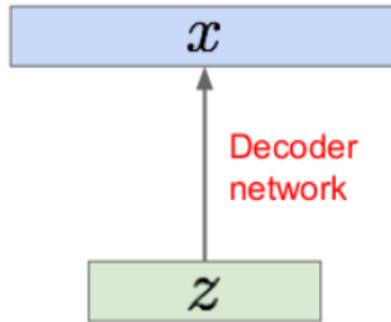
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS: GENERATING DATA

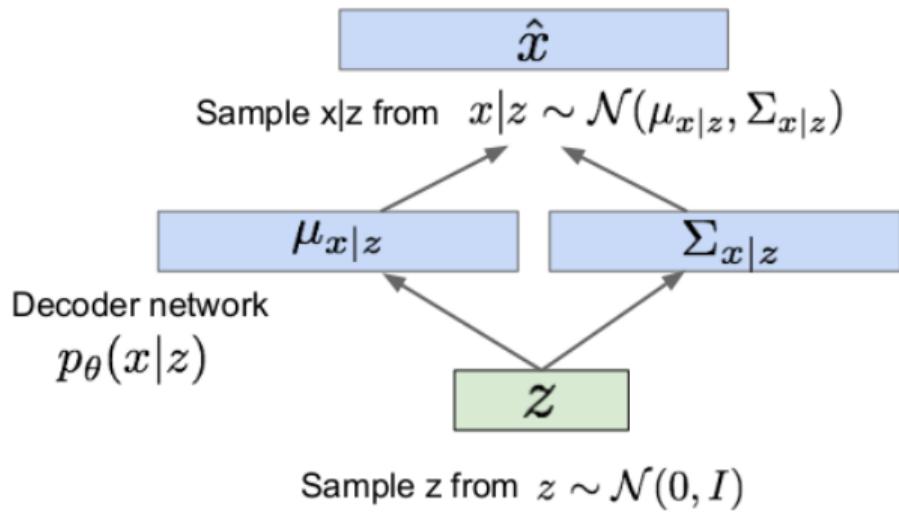
Our assumption about data generation process

Sample from true conditional
 $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



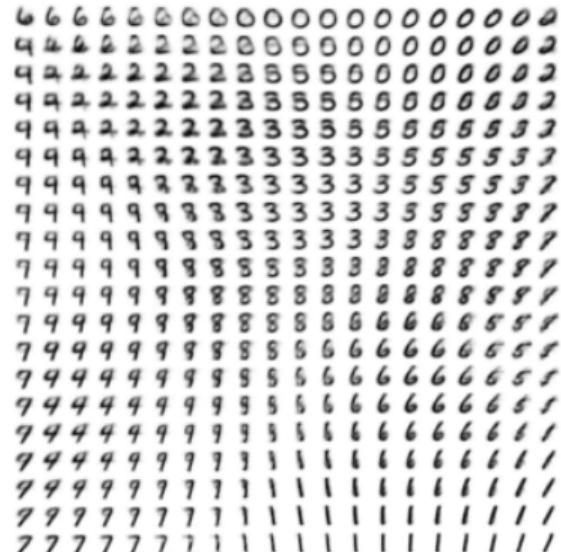
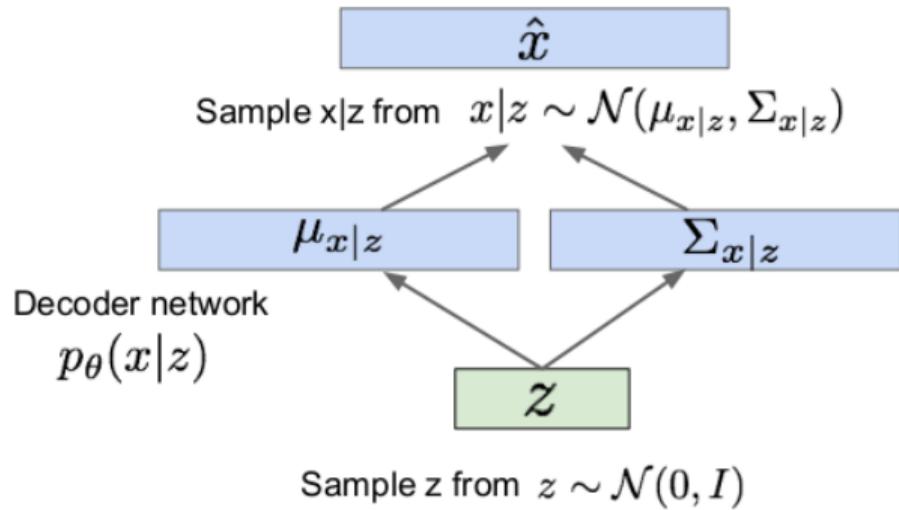
Now given a trained VAE:
use decoder network & sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS: GENERATING DATA

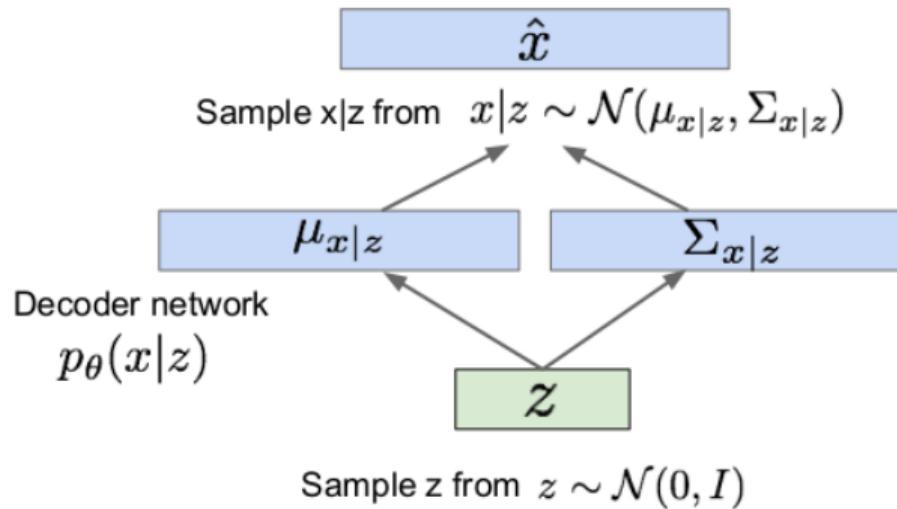
Use decoder network. Now sample z from prior!



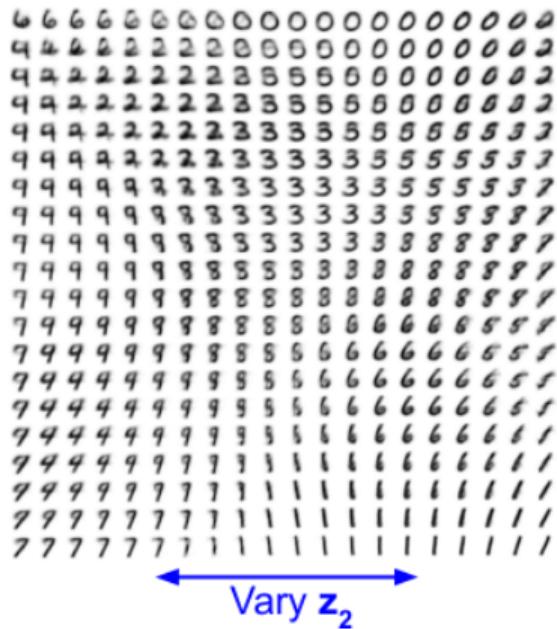
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS: GENERATING DATA

Use decoder network. Now sample z from prior!



Data manifold for 2-d \mathbf{z}



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS: GENERATING DATA

Diagonal prior on \mathbf{z}
=> independent
latent variables

Different
dimensions of \mathbf{z}
encode
interpretable factors
of variation

Degree of smile
Vary \mathbf{z}_1



Vary \mathbf{z}_2 Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VARIATIONAL AUTOENCODERS

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

References

- ① Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
<https://www.deeplearningbook.org/>
- ② Deep Learning with Python by Francois Chollet.
<https://livebook.manning.com/book/deep-learning-with-python/>

Thank You!