Introduction and Machine Learning Basics

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- **Subject overview**
- **What is AI**
- **Brief introduction to cyber security**
- **How AI helps Cyber security**
- **Limitations of AI in Security**
- **Math preliminaries**
- **Linear regression**

Subject information

- **Lecturer**
  - Dr Wei Zong
  - Email: wzong@uow.edu.au
- **7 Lectures (recordings will be available via Moodle)**:
  - Introduction and machine learning basics.
  - Training algorithm to learn.
  - Spam and phishing.
  - Backpropagation and anomaly detection.
  - SVM and Decision Trees.
  - Adversarial machine learning.
  - Network intrusion detection case study.

- **8 Labs**:
  - 10 tasks.
  - guided coding practice.
  - not recorded.

Subject overview

This subject:

- covers the principles, techniques and applications of AI for cybersecurity
- focuses on how to protect systems with data and algorithms
- provides examples of how machine learning can be applied to augment or replace rule-based or heuristic solutions to problems such as **anomaly detection**, **malware classification**, and **network traffic analysis**
- also talks about security issues of machine learning/deep learning models.
  - Adversarial examples & backdoor attacks.

Subject learning outcomes

- Analyze and solve AI for cybersecurity problems by gathering, organising and evaluating data
- Explain core AI concepts and tools in relation to cybersecurity
- Describe principles in cybersecurity and apply machine learning techniques in system protection and cyber threat detection.
- Demonstrate how AI can be applied to solve cybersecurity-related issues.

Syllabus

- Introduction to AI
- Machine learning basics
- Training algorithms to learn
- Spam and phishing detection
- Anomaly detection
- Malware analysis
- Malware threat detection
- Network traffic analysis
- Adversarial machine learning

- **Programming Language and tool:**
- Python
- Jupyter notebook

Prerequisites

Abilities and skills that are expected:

- Familiarity with Python
- Understanding of Math (basic knowledge of probability and statistics)
- Ability to understand algorithms

Assessment

- **CSIT375:**
  - Quiz 1 (5%)
  - Quiz 2 (25%)
  - Assignment (20%):
  - Final exam (50%)

Assessment

- All written assignments must be submitted electronically via Moodle before the due date
- **Penalties apply to all late assessments**
  - 25% mark penalty per day for late submissions
  - Submissions received more than 3 days late will receive a mark of zero
  - Submit wrong files = Late. 1s late => 1 day late
  - e.g. if the assignment is marked on a scale from 0-10 and your mark is 8.5/10
    - 1 day late the mark will be reduced to 6/10
    - 2 days late the mark will be reduced to 3.5/10
    - more than 3 days late the mark will be reduced to 0/10

Attendance requirements

- Satisfactory attendance is deemed by the University, to be attendance at 80% of the allocated contact hours
- Lecture recordings are made available via the subject Moodle site
- Labs will not be recorded

Minimum performance requirements

- To be eligible for a ‘Pass’ in this subject a student must achieve a mark of at least 40% in the final exam (i.e. 20/50)
  - And at least 50/100 final marks.

- Plagiarism is treated seriously. All submissions are checked for plagiarism
- **All assessment must be your own original work**
  - DO NOT: copy from others, copy from the internet, pay someone to do it
- If we suspect any work is not original, all students involved are likely to receive zero for the affected assessment

Important message

- **Make sure you check the subject's Moodle site regularly**
  - Subject materials are made available on Moodle
  - We will post announcements to Moodle from time to time
  - such as changes in assessment requirements / due date

- **Your feedback is welcome**
  - positive aspects of this subject
  - issues of this subject to be addressed

Outline

- Subject overview
- **What is AI**
- Brief introduction to cyber security
- How AI helps Cyber security
- Limitations of AI in Security
- Math preliminaries
- Linear regression

What is AI

- **Artificial Intelligence (AI)**

Artificial intelligence is a popular term that indicates algorithmic solutions to complex problems typically solved by humans.

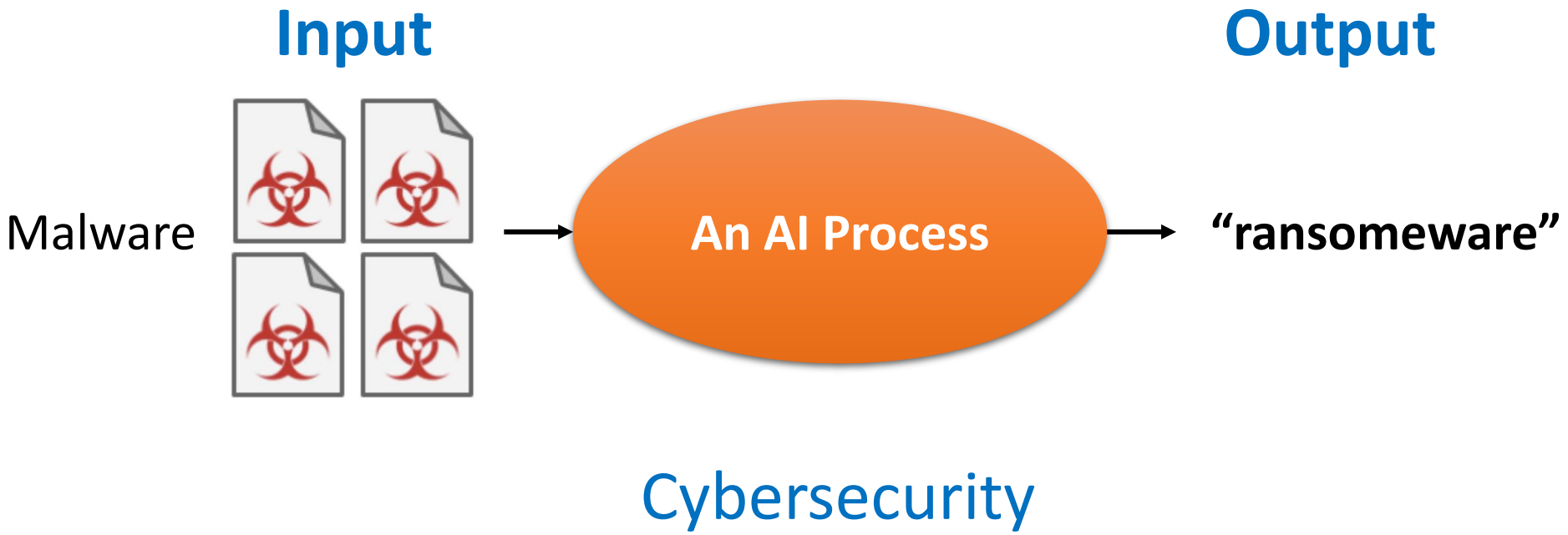
AI systems have been loosely defined to be machine-driven decision engines that can achieve near-human-level intelligence.

What is AI

- **Artificial Intelligence (AI)** has been moving extremely quickly in the last few years, demonstrating a potential to revolutionize every aspect of our lives

What is AI

- **AI can be broadly defined as technology that can *learn* and produce intelligent behavior**



What is AI

- **AI can be broadly defined as technology that can *learn* and produce intelligent behavior**

Input **Output**

Pixels:

An AI Process

"Four kids are playing with a ball"

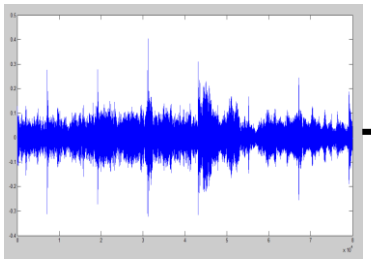
More than just a category about the image!

Computer Vision

What is AI

- **AI can be broadly defined as technology that can *learn* and produce intelligent behavior**

Input **Output**

Audio Clip:  → **An AI Process** → **“set an alarm for 7:00 a.m”**

Speech Recognition

What is AI

- **AI can be broadly defined as technology that can *learn* and produce intelligent behavior**

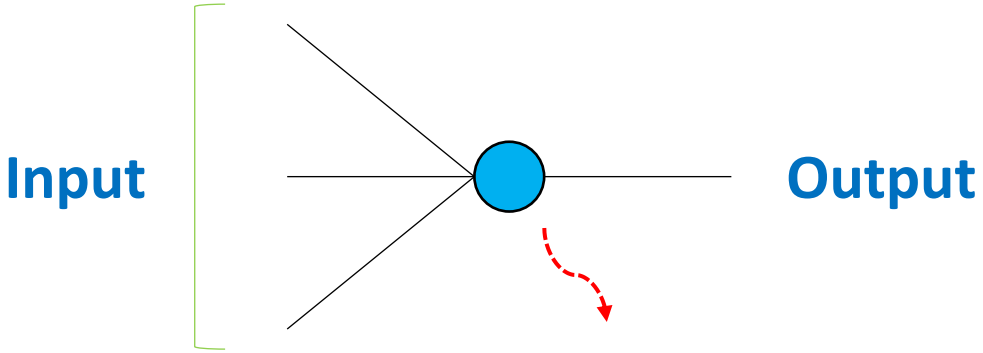
Input **Output**

Text: "Hello, how are you?" → **An AI Process** → **"Bonjour, comment allez-vous"**

Machine Translation

What is AI

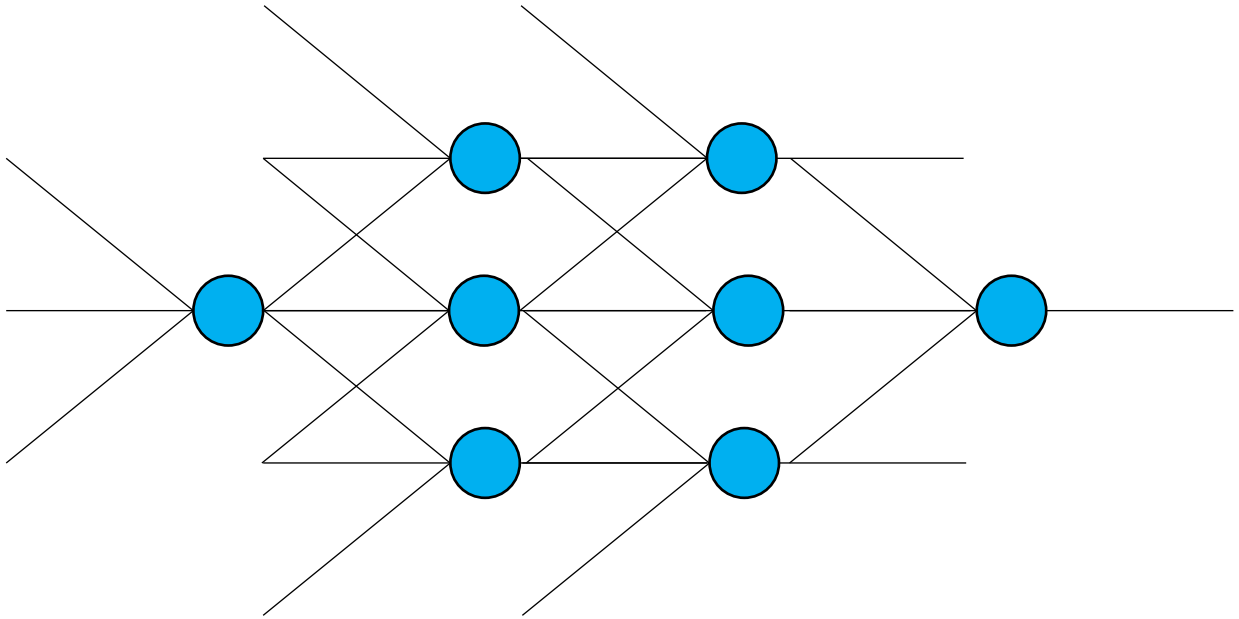
- Think of this as incoming impulses (input) passed from one ***neuron*** (AI process) to the next, if any, and finally generating an output



AI process, which is essentially a
mathematical function-- more on this later.

What is AI

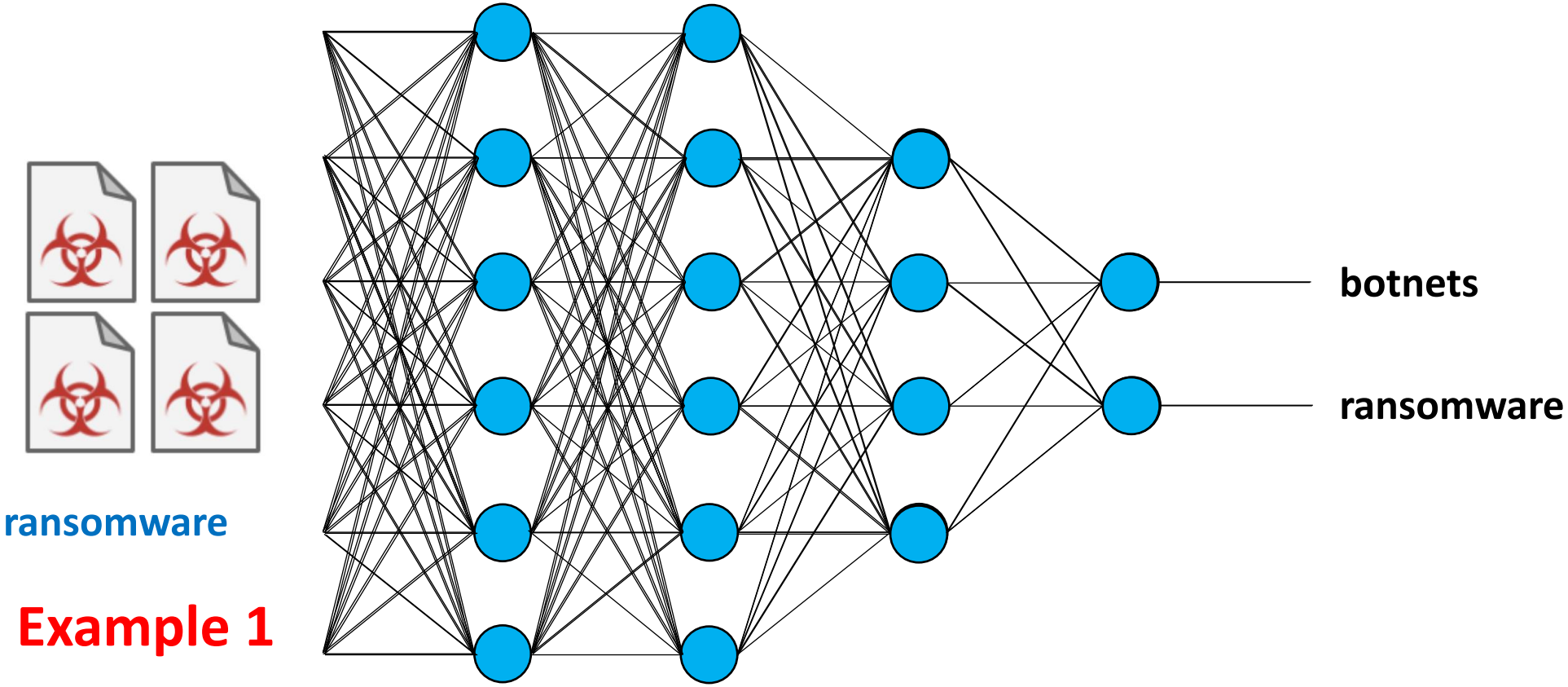
- Connect as many of these neurons as needed, resulting in what is called a ***neural network*** (a branch of AI)



The more layers you add, the deeper it becomes. Deep ones are referred to as ***deep neural networks*** or ***deep learning (DL) models***

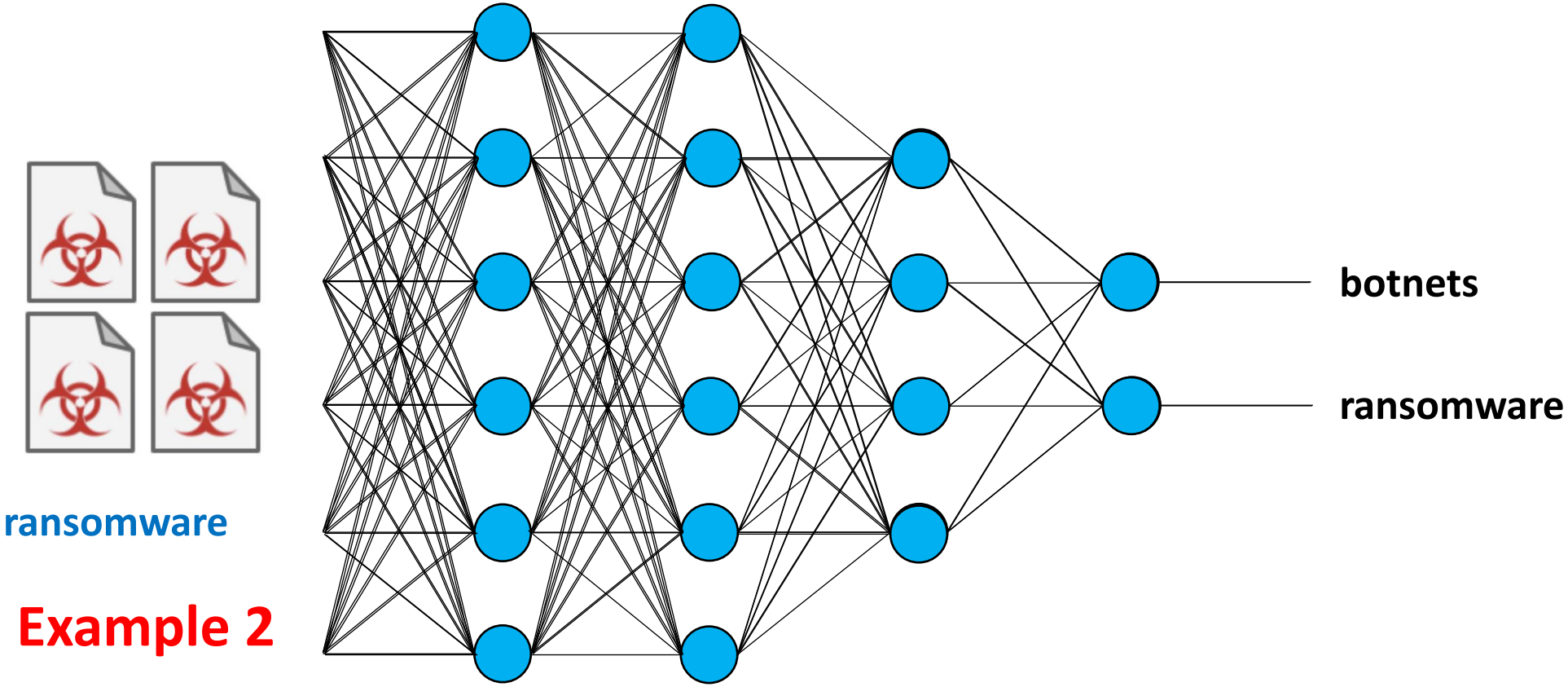
What is AI

- Subsequently, ***train*** the DL model



What is AI

- Again, with a different *known* example

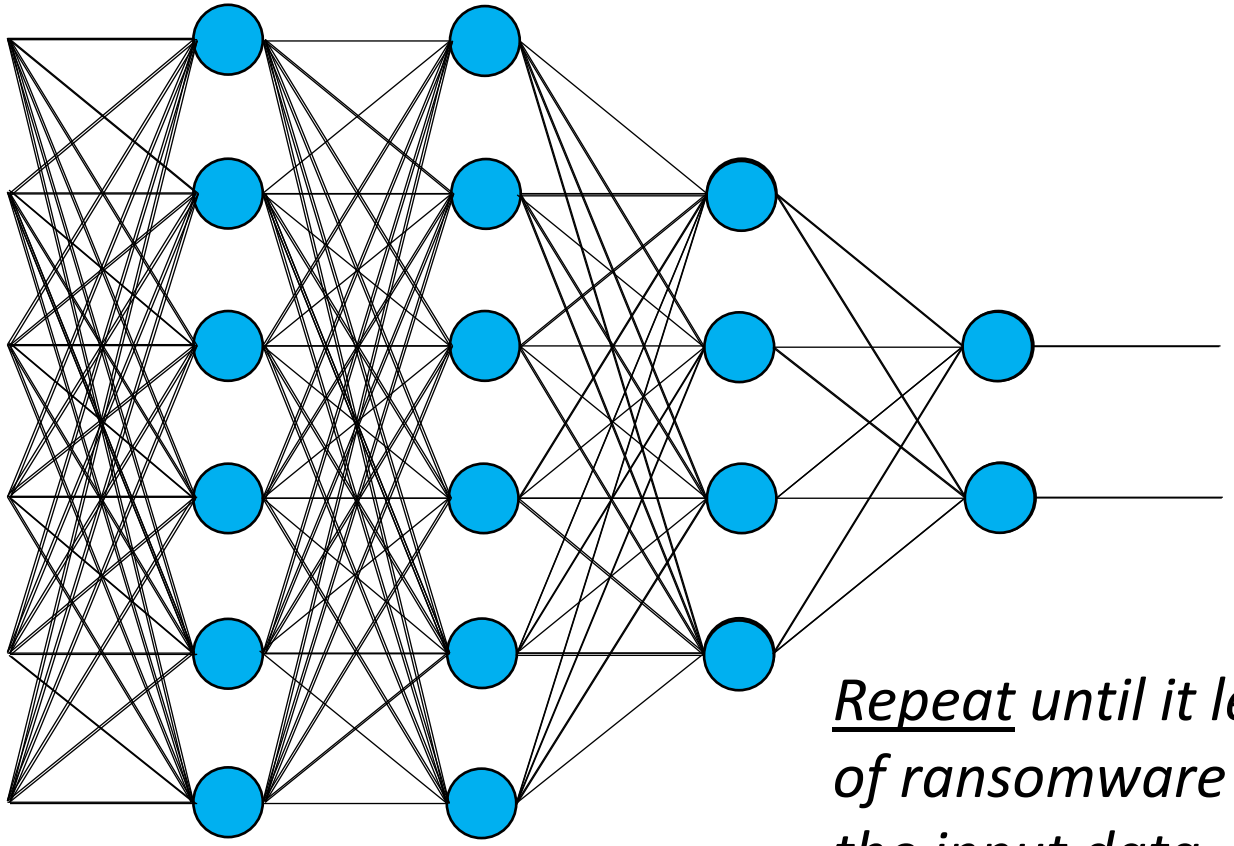


What is AI

- Yet again, with another different *known* example

ransomware

Example 3



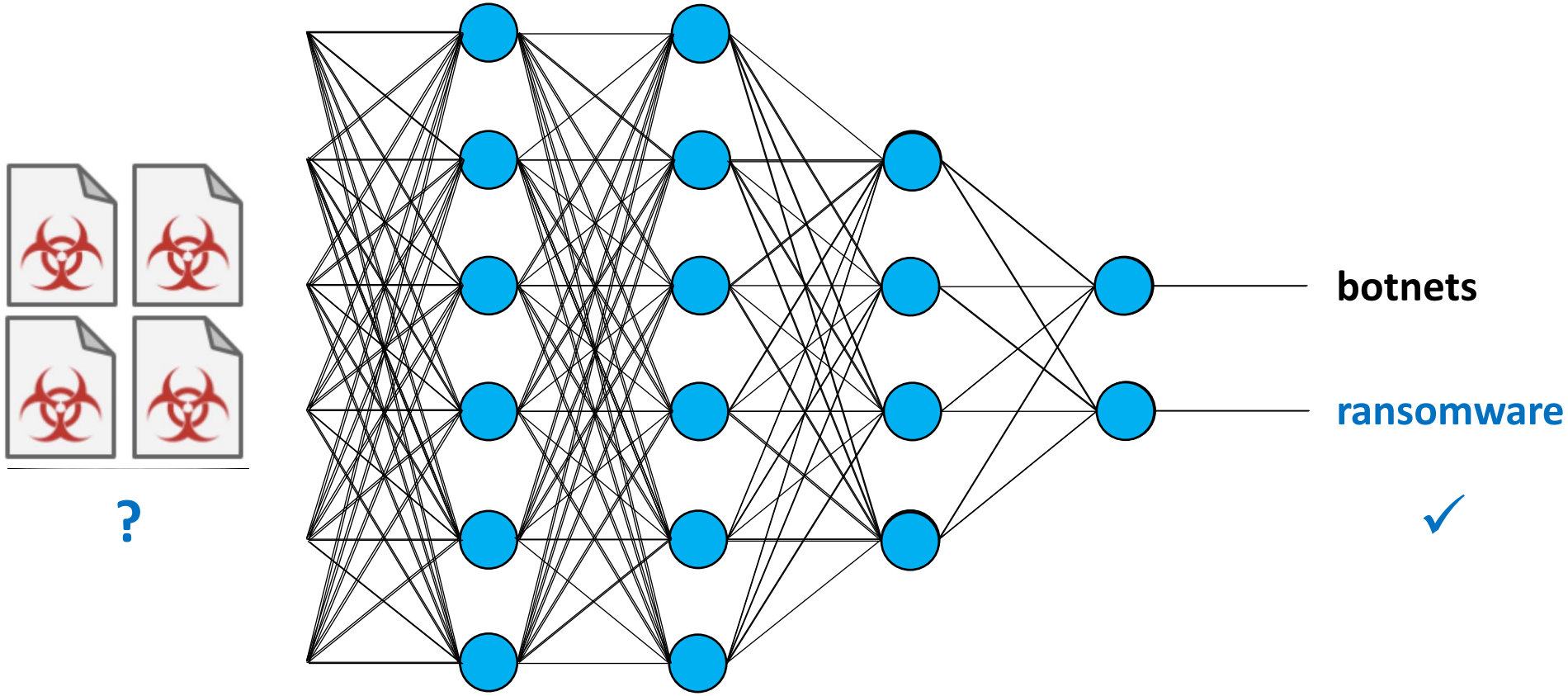
botnets

ransomware

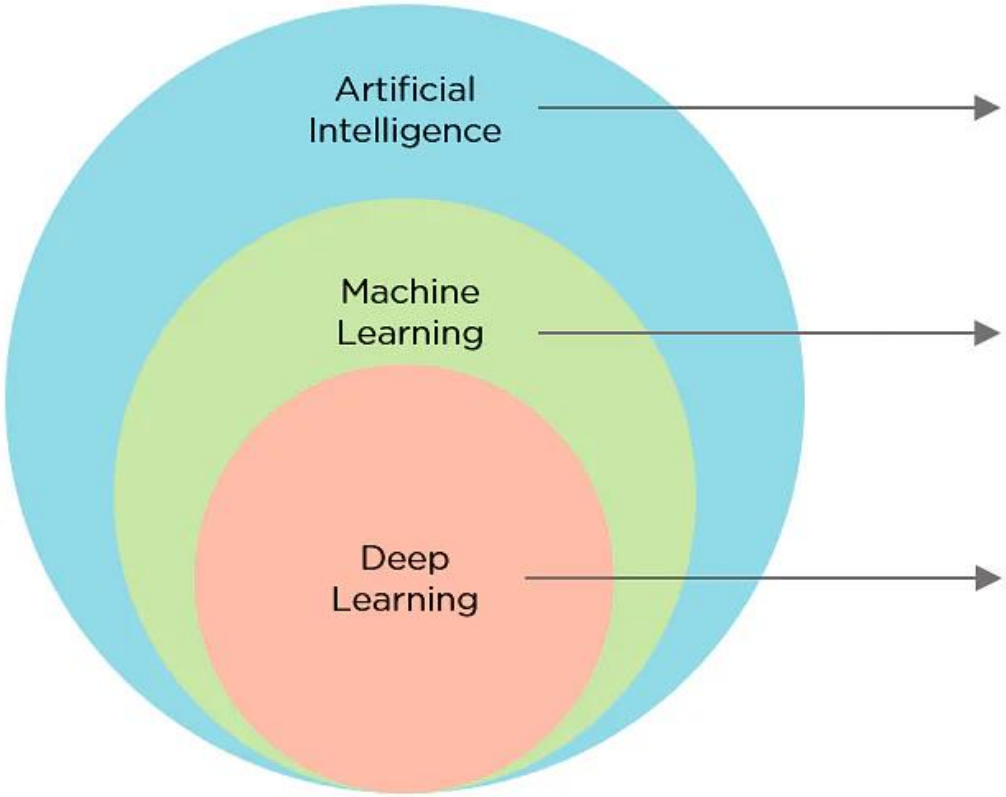
Repeat until it learns the patterns of ransomware and botnets in the input data

What is AI

- After training the DL model, use it to *infer* what an <u>*unknown*</u> malware is



Definitions



AI: is the concept of creating intelligent machines

ML: a branch of AI, helps to build data-driven applications

DL: a branch of machine learning, uses large amount of data and deep neural networks to train a model

Outline

- Subject overview
- What is AI
- **Brief introduction to cyber security**
- How AI helps Cyber security
- Limitations of AI in Security
- Math preliminaries
- Linear regression

Definitions

- **Cyber Security**
  - Cybersecurity is the body of technologies, processes and practices designed to protect networks, computers, programs and data from attack, damage or unauthorized access
- **Goals: C-I-A triad**
  - **Confidentiality** unauthorized disclosure of information
  - **Integrity** unauthorized modification of information
  - **Availability** unauthorized withholding of information

What is cybersecurity all about?

- Protecting information and systems from threats and attacks
- ensuring they remain secure and functional
- In an organization, the people, processes, and technology need to complement one another to create an effective defense from cyber attacks
- **Users**: understand and comply with basic data security principles
- **Processes**: framework for how to deal with cyber attacks
- **Technology**: tools needed to protect from cyber attacks
  - entities need to be protected: endpoint, networks, the cloud
  - firewalls, malware protection, antivirus software, email security solutions…

Common types of cybersecurity threats

- **Phishing**
  - sending fraudulent emails that resemble emails from reputable sources
  - steal sensitive data, e.g. credit card numbers, login information
- **Ransomware**
  - extort money by blocking access to files or the computer systems until the ransom is paid
- **Malware**
  - gain unauthorized access or to cause damage to a computer
- **Social engineering**
  - trick people into revealing sensitive information

Outline

- Subject overview
- What is AI
- Brief introduction to cyber security
- **How AI helps Cyber security**
- Limitations of AI in Security
- Math preliminaries
- Linear regression

How AI helps security

- **AI + ML + Cyber Security**

Machine learning has been quickly adopted in cybersecurity for its potential to automate the detection and prevention of attacks, particularly for next-generation antivirus AI systems

AI has a lot of potential for cybersecurity applications, such as learning from existing cyber incidents, predicting attacker behavior, and taking proactive defensive measures to protect critical infrastructures

How AI helps security

- AI lets computers learn without being explicitly programmed
- Challenges in managing threat information
  - track and correlate massive data
  - not feasible to manage with only people => automate the analysis
- In security, AI continuously learns by analyzing data to find patterns and predict threats in massive data sets
  - detect malware
  - find insider threats
  - keep people safe when browsing
  - uncovering suspicious user behavior
  - …

How AI helps security

- **Find threats on a network**
  - detect threats by constantly monitoring the behavior of the network for anomalies
  - AI engines process massive amount of data in near real time to discover critical incidents
  - detection of insider threats, unknown malware, and policy violations
- **Keep people safe when browsing**
  - predict “bad neighborhoods” online to help prevent people from connecting to malicious websites
  - analyze Internet activity to automatically identify attack infrastructures staged for current and emergent threats

How AI helps security

- **Provide endpoint malware protection**
  - detect unknown malware that is trying to run on endpoints
  - identify new malicious files and activity based on the attributes and behaviors of known malware

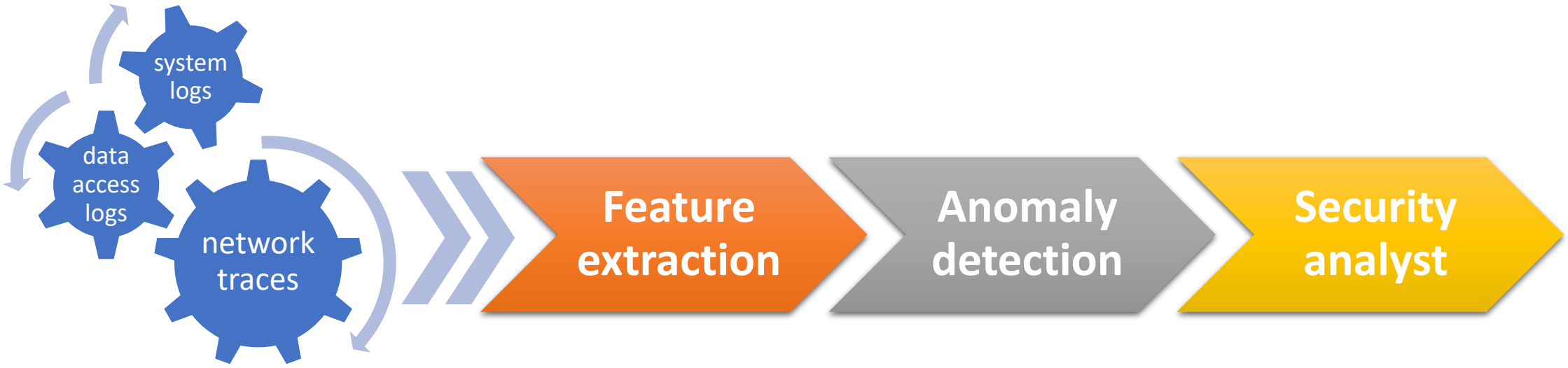
- **Protect data in the cloud**
  - analyzing suspicious cloud app login activity
  - detecting location-based anomalies
  - conducting IP reputation analysis to identify threats and risks in cloud apps and platforms

Applications of AI in Security

- **Monitoring**
  - Automated, continuous, feed data to analysts
  - Things to look for: intrusion, privacy violation, exfiltration, ...

- **Analysis**
  - Not necessarily continuous
  - Often initiated by humans
  - Applications: threat intelligence, incident investigation, vulnerability assessment, ...

High-level view of a monitoring pipeline

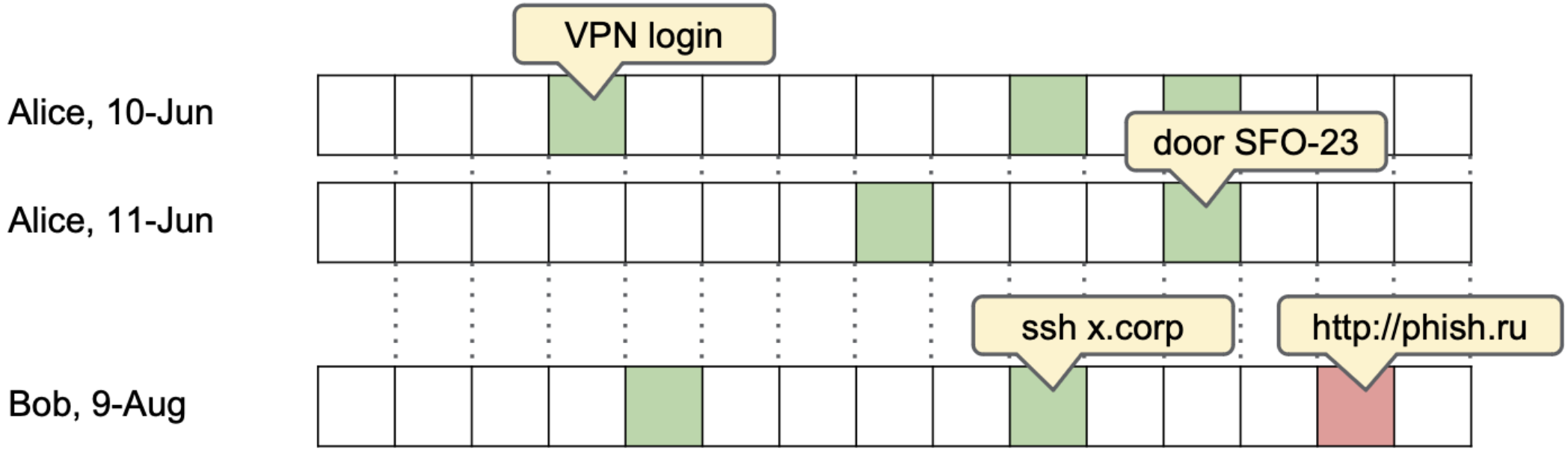


Example: detecting anomalous actor behavior

- Some reasons to care
  - Employee account compromised by malware?
  - Intentional malicious activity?
- Goal: model actor behavior, find anomalies
- What we need to do
  - Identify useful features
  - Model normalcy
  - Find outliers

Feature extraction: modeling actors

- Partition logs by actor and time
- Represent (actor, time) pairs as vectors of binary variables

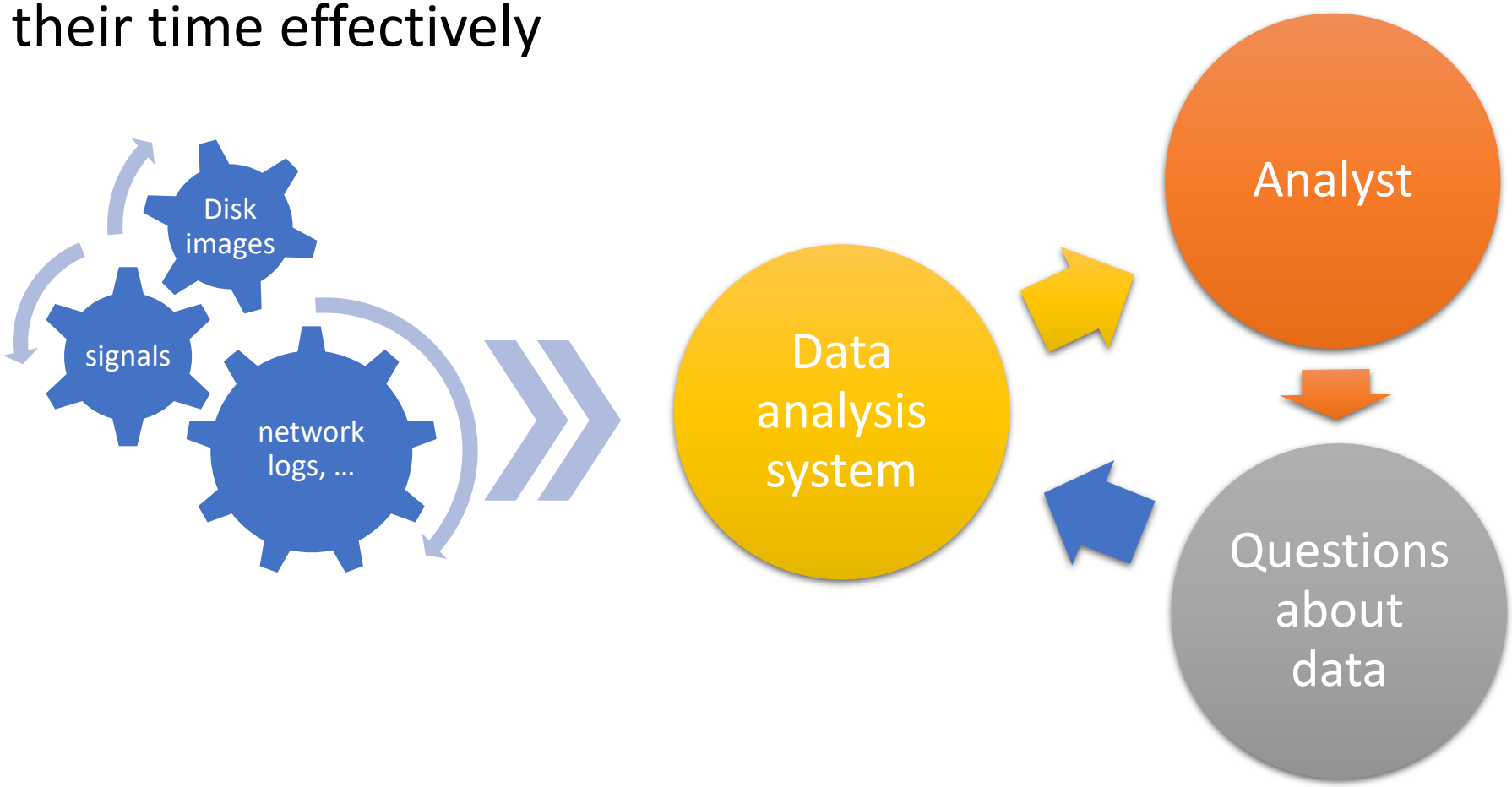


Modeling normalcy and finding outliers

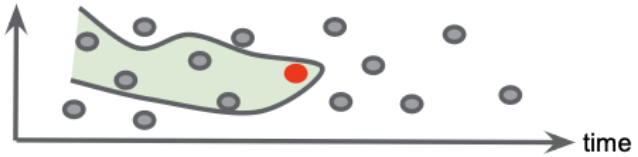
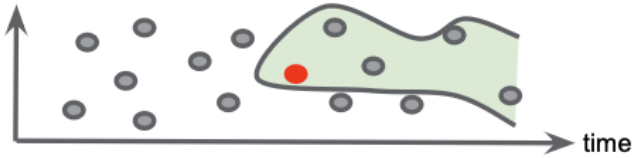
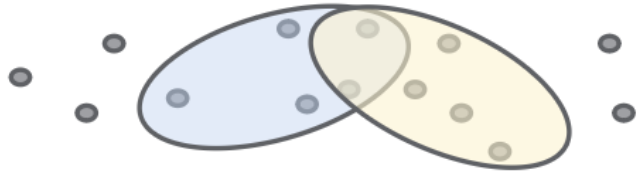
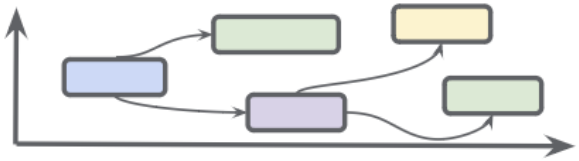
- Need to find low-probability features or combinations of features
- Many possible approaches
- Nearest neighbors ⇒ similarity metric between actors
  - Intuition: find users that are not very similar to any other users
  - Variant: compare a user to her past
    - “Neighbors” are feature vectors in user’s past
    - Identify changes in behavior
- “Strange pairs”⇒ features that rarely appear together
  - Intuition: identify users with pairs of features that occur frequently individually but rarely together
  - E.g., “accessed source code” and “works in HR”

AI for security analysis

- Skilled analysts are a valuable resource: give them the tools to use their time effectively



Questions an analyst might ask

Causation	How did the attacker get root?	
Consequence	What was the effect of running the script?	
Correlation	Which signals fired simultaneously?	
Summarization	What was the user doing last night?	

Broad spectrum of tools

- Looking for causes and effects
  - graph traversal
- Triaging malware
  - classification
- Statistical (as well as graph-based) approaches are effective in this problem domain

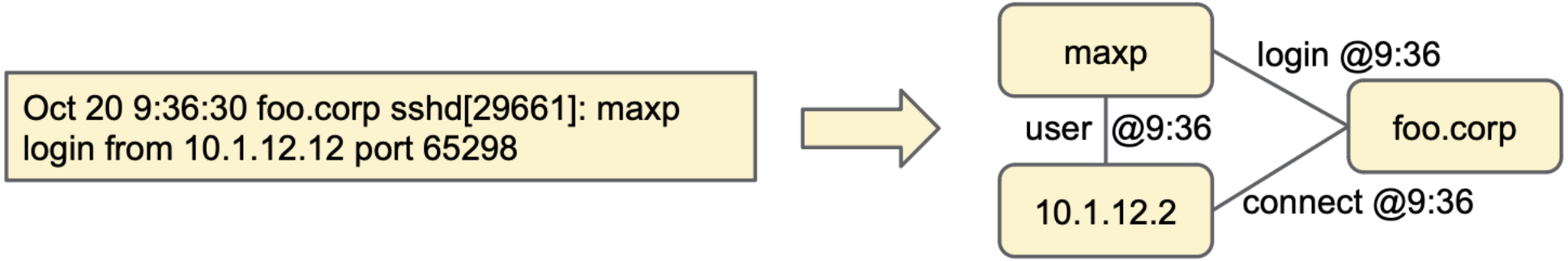
Example 1: graph traversal for incident investigation

- Some questions to answer:
  - Were any machines affected by watering hole attack?
    - A watering hole attack is a form of cyberattack that targets groups of users by infecting websites that they commonly visit.
  - User A downloaded malware. What should be cleaned up?

- Given a graph representation of all relevant logs, can be framed as a large-scale graph search problem

Graph creation

- Log lines induce graph components



- Edges annotated with times and semantics
- Many different log sources in one huge graph

Sample graph query

Given watering hole hostname X ...

→ IPs that it resolved to

→ internal IPs that talked to them

→ machines (assets) those internal IPs belonged to

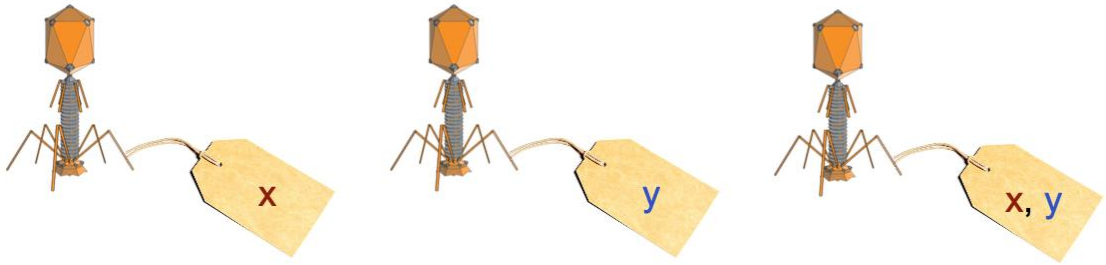
→ users who used those machines

→ other machines those users have logged into

Hours of manual research replaced by a ~10-second query

Example 2: malware classification

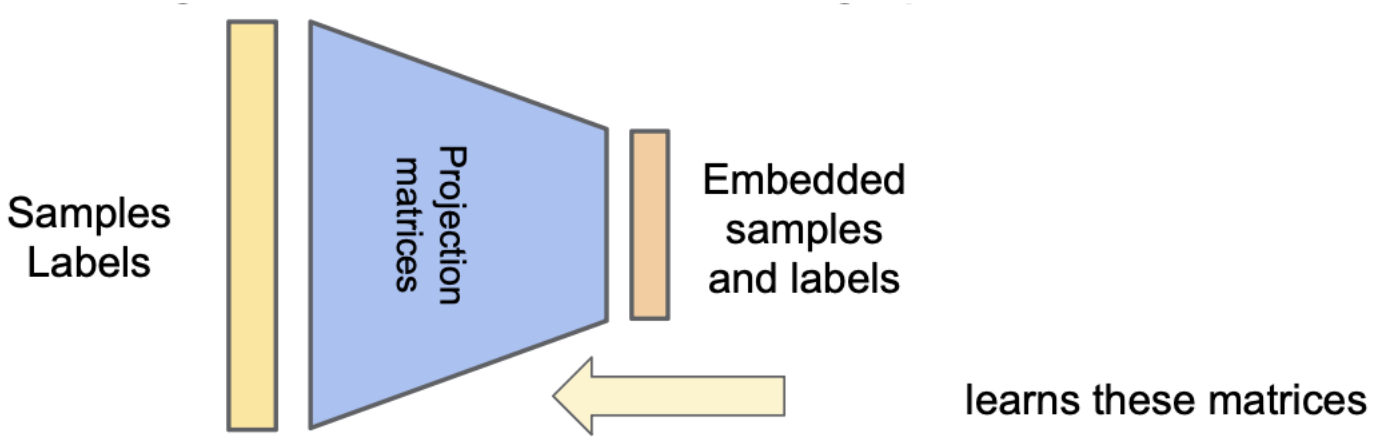
- Given a binary, is it malware? If so, what kind?
- Each sample is an executable. It has indicators (features) from static and dynamic analysis (e.g., basic block structure, registry changes, ...)
- Malware in training corpus also has one or more labels (from manual labeling, signatures, etc.) denoting its families



- Why is this useful?
  - Incident triage – is this malware we should care about?

Modeling the data

- Each sample X is a sparse N-dimensional vector (N ≈ millions)
- Each label is an integer in [1, k] (k ≈ thousands)
- Learns a projection into a low-dimensional embedding space
  - Makes the problem computationally feasible
  - Provides meaningful metric inside embedding space



Using the model

- Once the projection matrices are learned, we can do useful things

- Compare two samples? Project into embedding space, measure distance
- Closest family to a sample? Project sample and all families, find smallest distance
- Approximate nearest sample? Filter samples by closest family

Outline

- Subject overview
- What is AI
- Brief introduction to cyber security
- How AI helps Cyber security
- **Limitations of AI in Security**
- Math preliminaries
- Linear regression

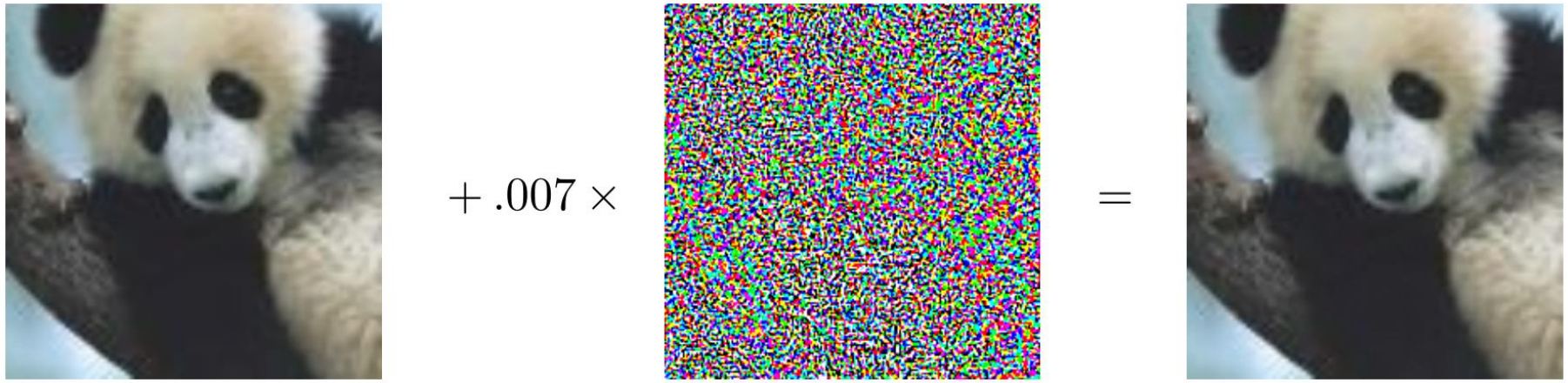
Limitations of AI in Security

- “Security is a process”
- Technology (AI, ML, etc.) is only a tool, not a complete solution
- User education (social engineering is surprisingly successful)
- System hardening (auth, secure engineering, timely patches, ...)
- Operational procedures
  - Adapting to growth (new hires / platforms)
  - Maintaining alertness (in the absence of major incidents)
  - Etc.

Limitations of AI in Security

- **False negatives can be very expensive**
  - Could cause arbitrary damage to our users
  - Network intrusion.
- **False positives can also be expensive**
  - Analyst time is valuable
- **Alerts should make sense to a human**
  - The analyst (security expert) is key
  - False positives + inexplicable results → signal fatigue
    - Inexplicable predictions are useless for the judicial purpose.

AI Models Have Limitations



$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Explaining and Harnessing Adversarial Examples. ICLR (Poster) 2015

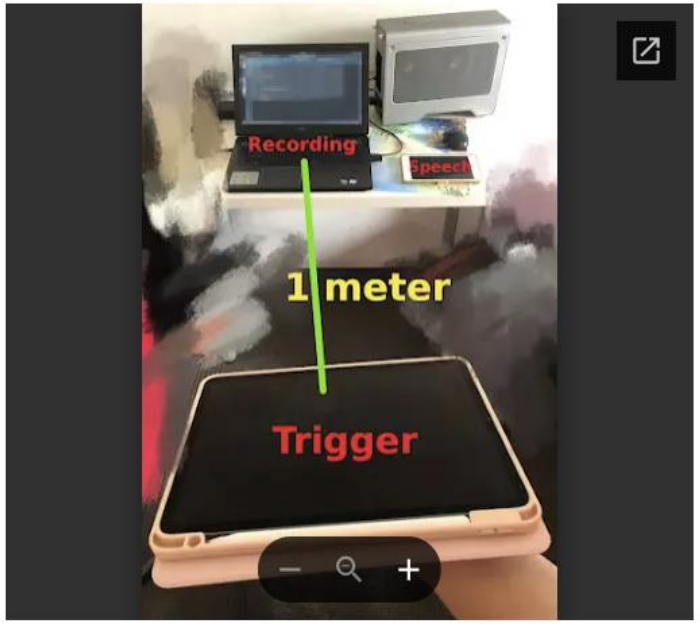
Real-world Attacks against AI Models

Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.

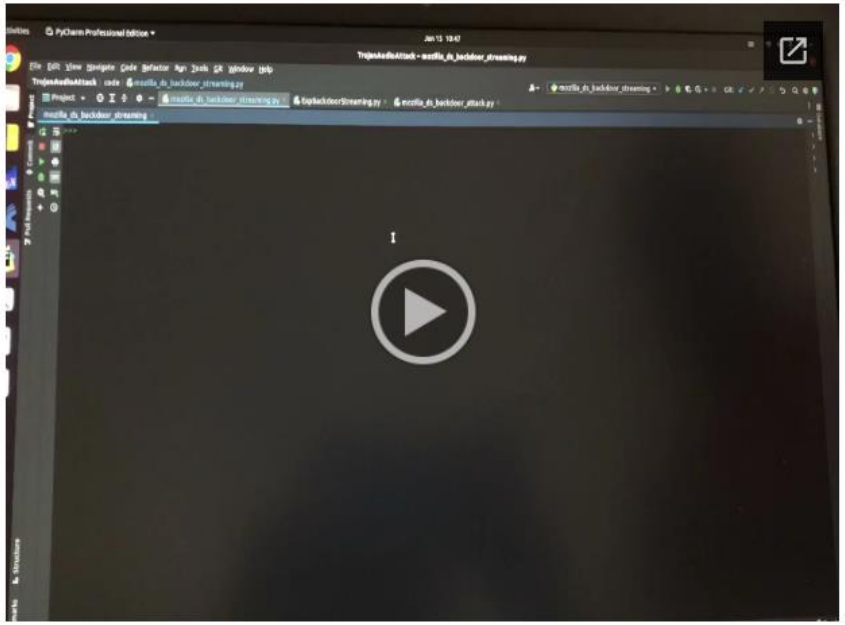
Real-world Attacks against AI Models

Real-world Attack Demo 1

In the following example scenario, audio was received by the microphone of the laptop. The laptop was used for recording and transcribing input audio. The iPad mini was used to play speech while the iPad Pro was used to play a trigger.



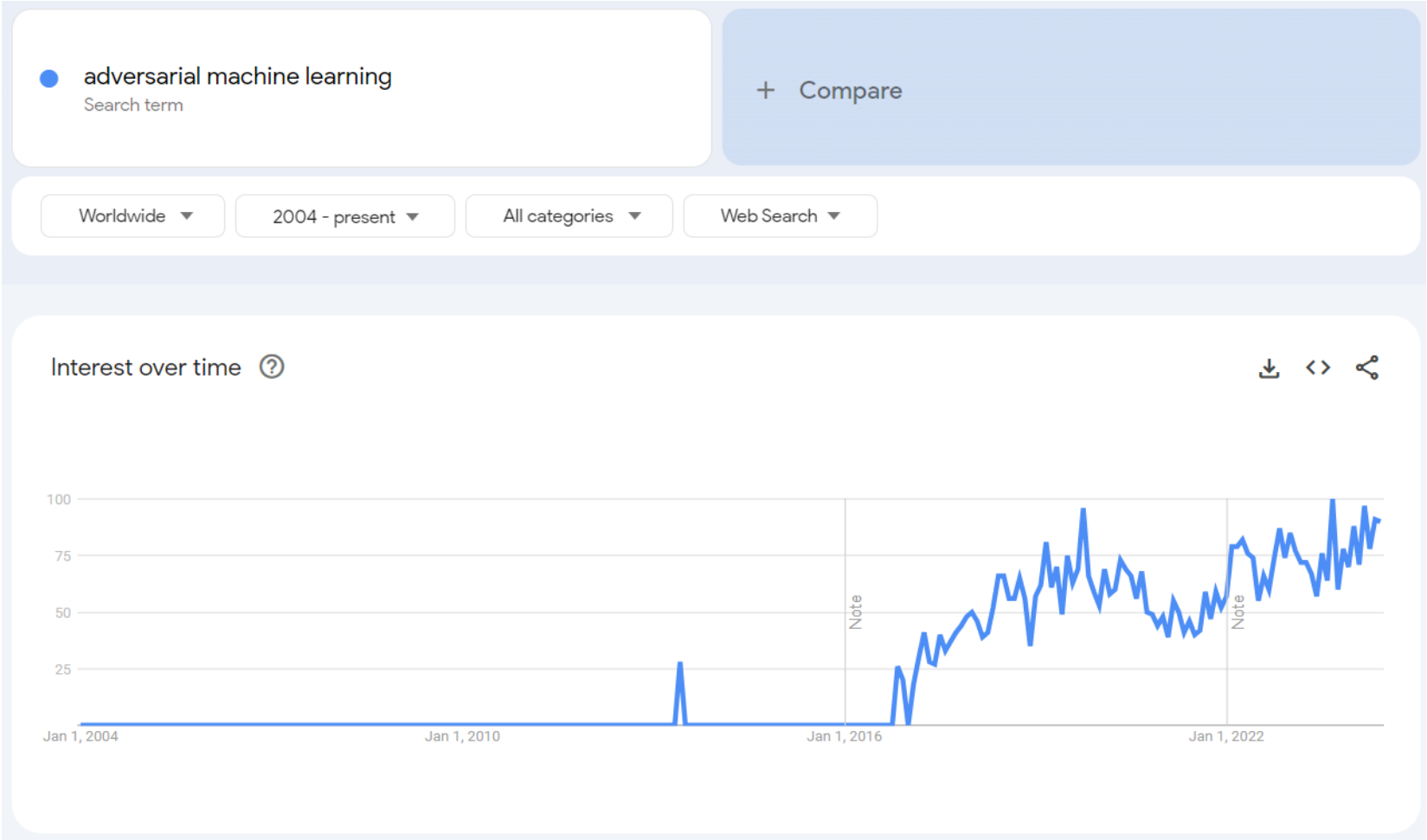
The trigger was played at a distance of 1 meter away from



The benign speech was correctly transcribed.

https://sites.google.com/view/trojan-attacks-asr

Adversarial Machine Learning



https://trends.google.com/

Outline

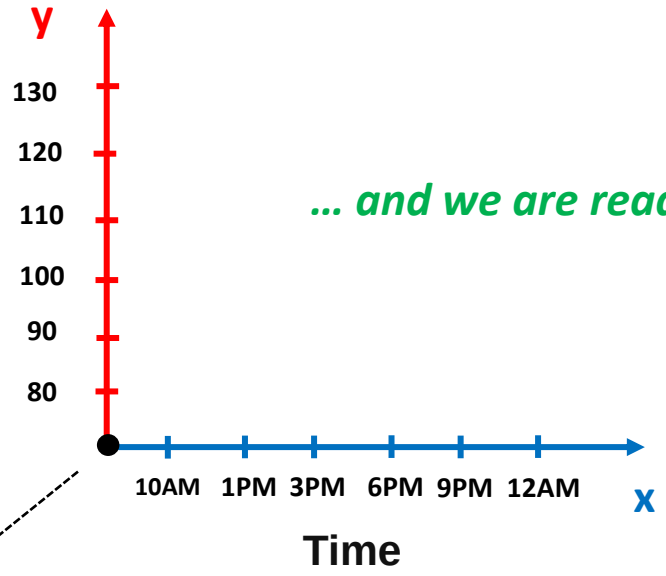
- Subject overview
- What is AI
- Brief introduction to cyber security
- How AI helps Cyber security
- Limitations of AI in Security
- **Math preliminaries**
- Linear regression

Plotting Data in 2D Space

- How to plot values of something (e.g., *network traffic*) over something else (e.g., *time*)?

The horizontal direction is commonly called **x-axis**

The vertical direction is commonly called **y-axis**



… and we are ready to go

Where they cross over is the **origin (0,0)**

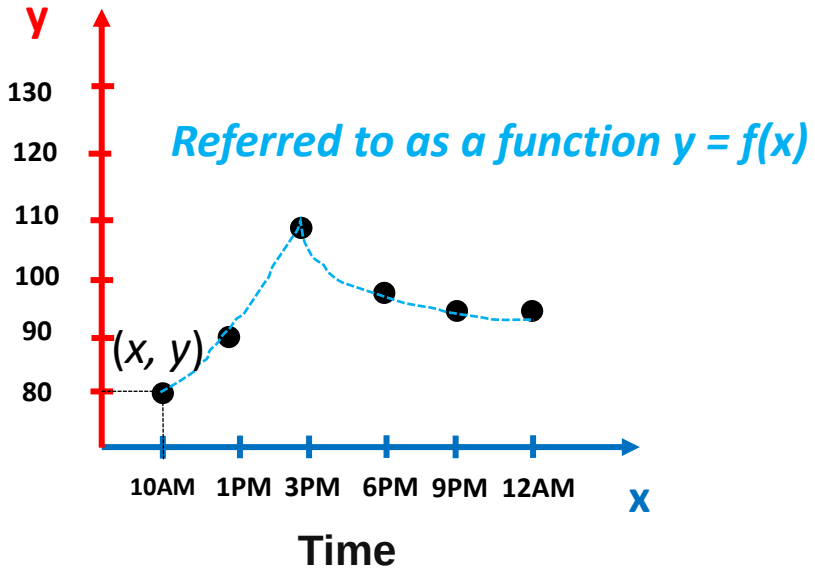
1. **Put them together on a chart (or *graph*)**
2. **Define the *units* on the axes and label them**

Plotting Data in 2D Space

- How to plot values of something (e.g., *network traffic*) over something else (e.g., *time*)?

The horizontal direction is commonly called **x-axis**

The vertical direction is commonly called **y-axis**

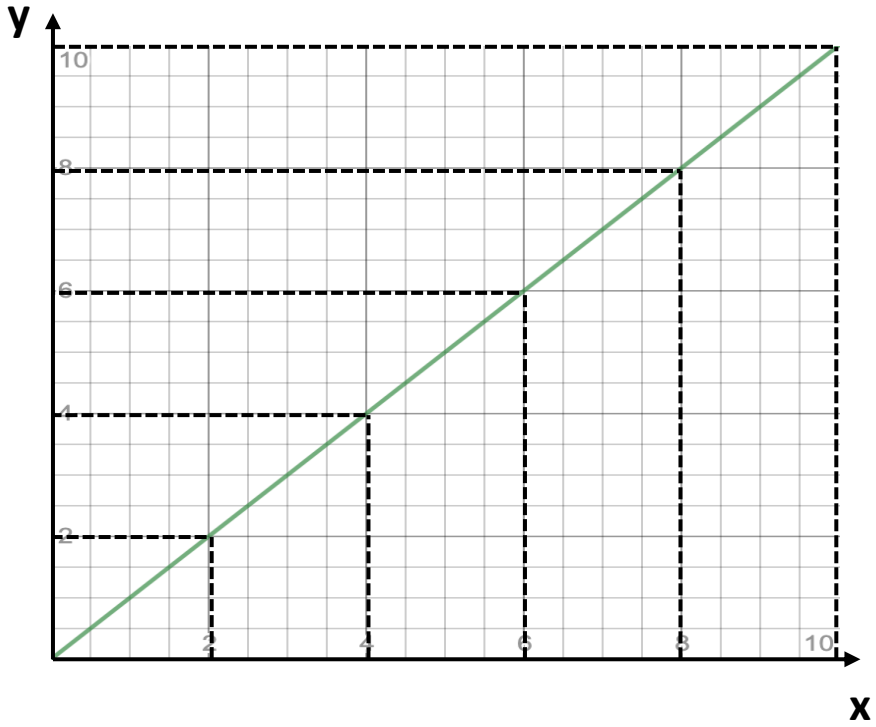


1. Put them together on a chart (or *graph*)

2. Define the *units* on the axes and label them

Mathematical Functions

- Here is a very simple function



x	y = f(x)
2	2
4	4
6	6
8	8
10	10

What is this function?

A line: y = 1x

For any given input (x), the output (y) is *exactly* the same

Mathematical Functions

- Here is another very simple function

x	y = f(x)
2	4
4	8
6	12
8	16
10	20

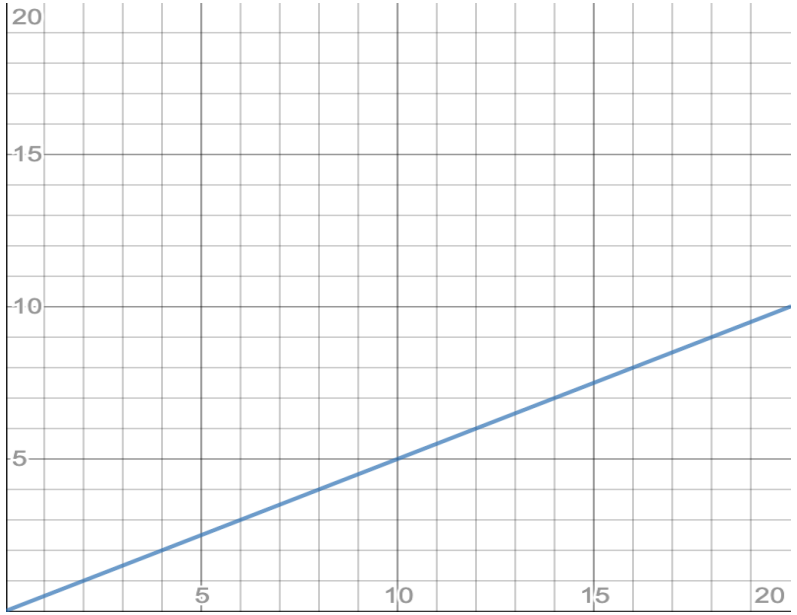
What is this function?

A line: y = 2x

For any given input (x), the output (y) is *twice* the input

Mathematical Functions

- Here is another very simple function



x	y = f(x)
2	1
4	2
6	3
8	4
10	5

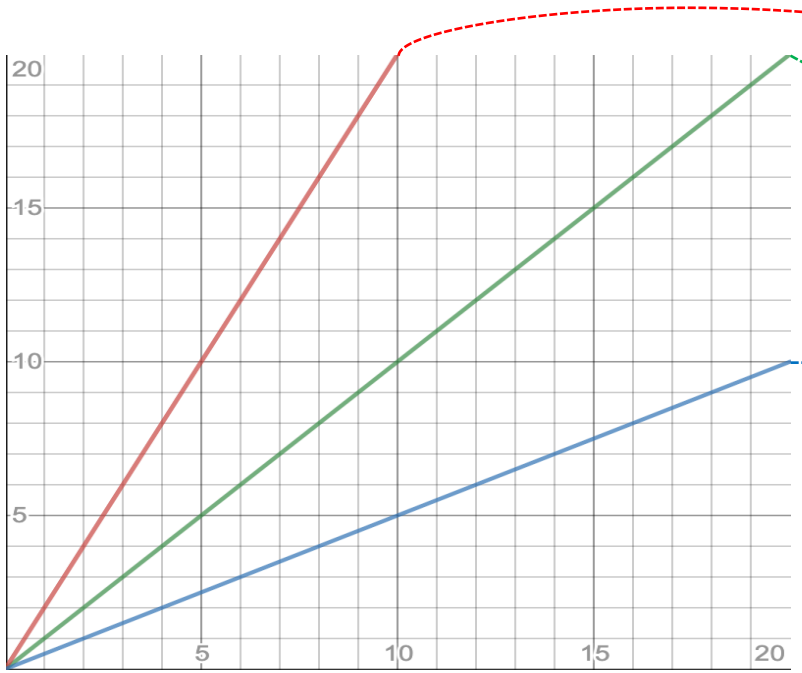
What is this function?

A line: y = 1/2x

For any given input (x), the output (y) is *half* the input

Slope

- Here are the 3 functions together



y=2x

y=1x

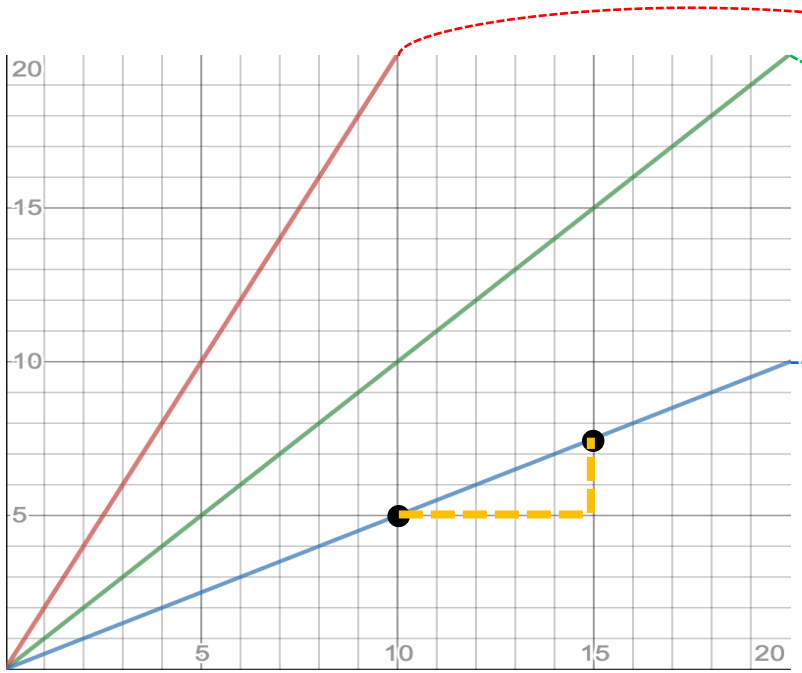
y=1/2x

What is the difference?

The *steepness*
(or the *slope*)

Slope

- Here are the 3 functions together



y=2x

y=1x

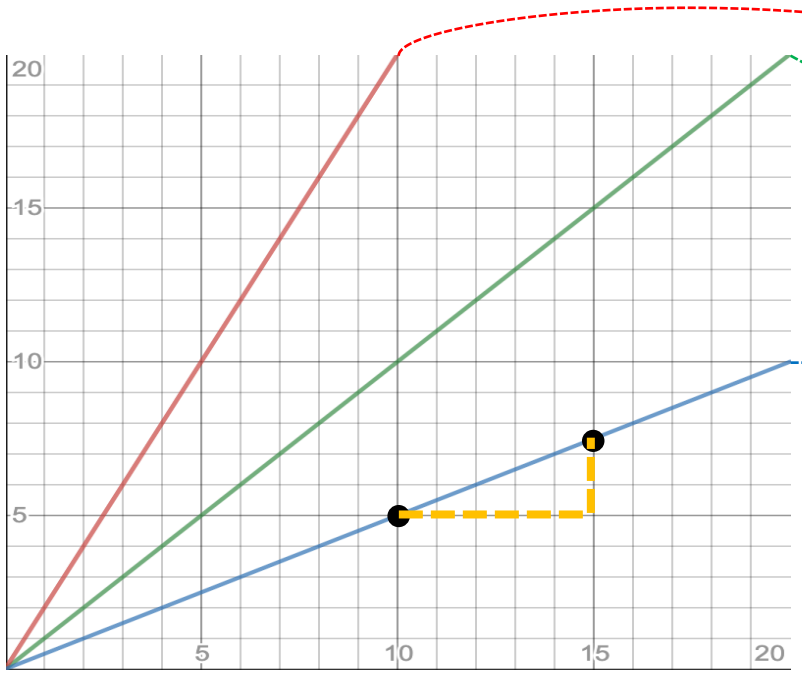
y=1/2x

How to measure the slope?

How much does a line increase in the vertical direction (i.e., y) for a given increase in the horizontal direction (i.e., x)?

Slope

- Here are the 3 functions together



y=2x

y=1x

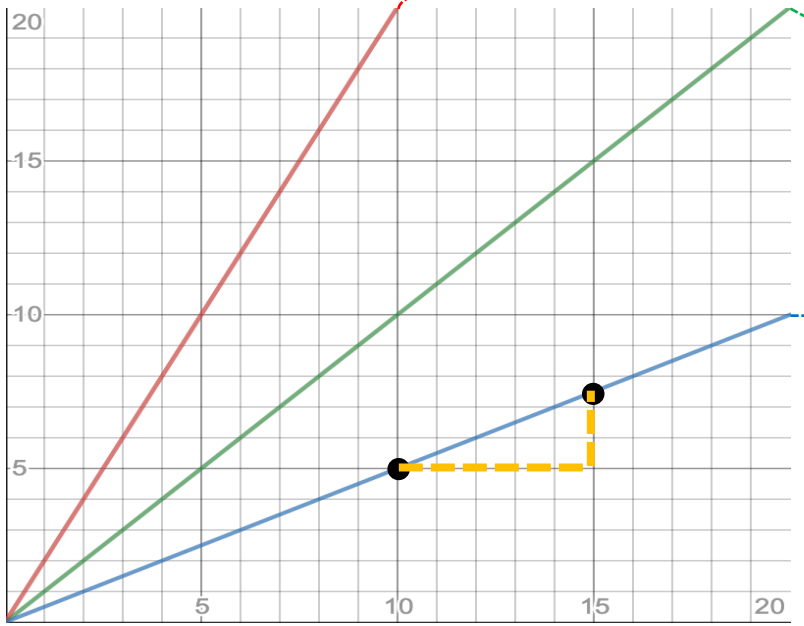
y=1/2x

How to measure the slope?

Slope = Δy/Δx

Slope

- Here are the 3 functions together



y=2x

y=1x

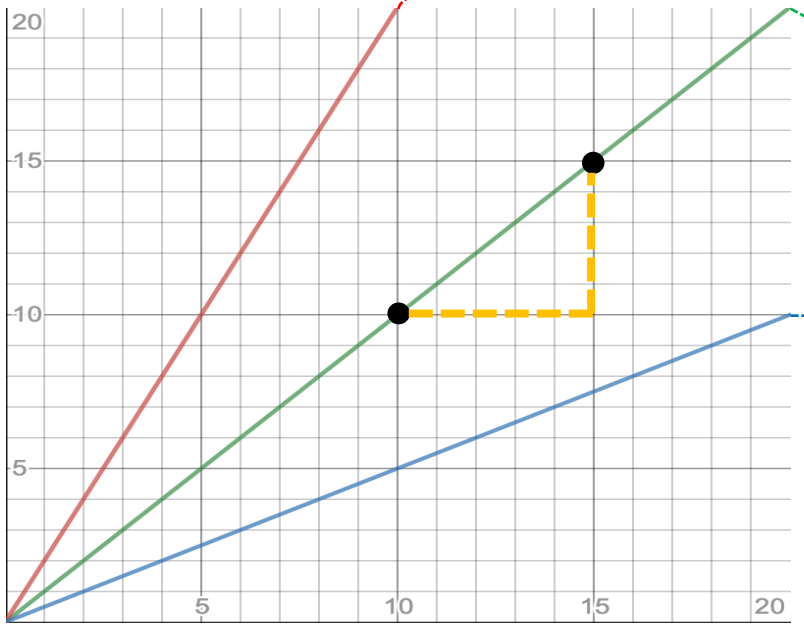
y=1/2x

How to measure the slope?

Slope = Δy/Δx = (7.5-5)/(15-10) = 2.5/5 = 1/2

Slope

- Here are the 3 functions together



y=2x

y=1x

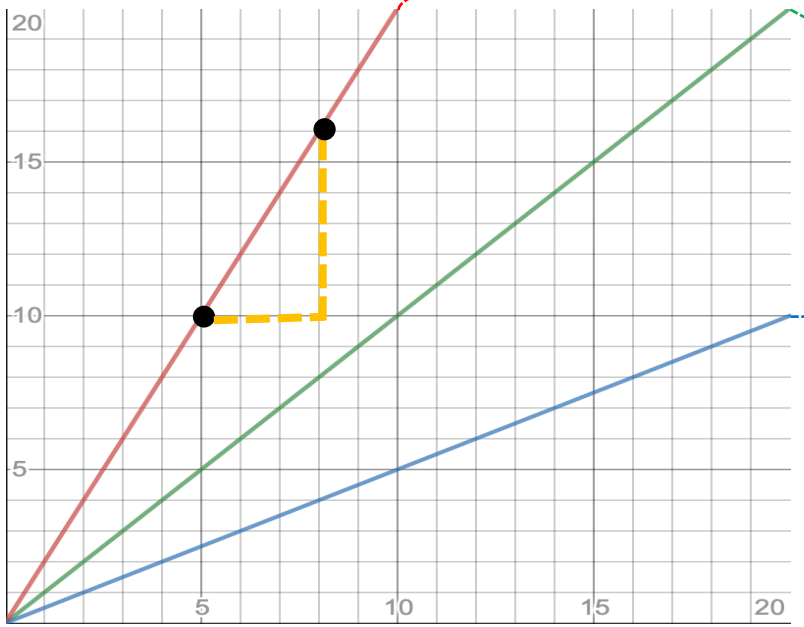
y=1/2x

How to measure the slope?

Slope = Δy/Δx = (15-10)/(15-10) = 5/5 = 1

Slope

- Here are the 3 functions together



y=2x

y=1x

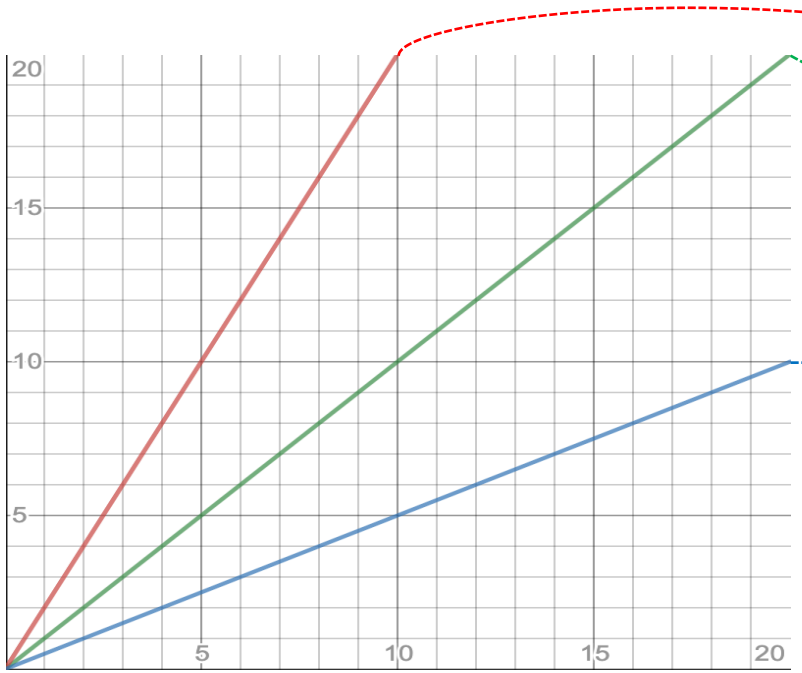
y=1/2x

How to measure the slope?

Slope = Δy/Δx = (16-10)/(8-5) = 6/3 = 2

Slope

- Here are the 3 functions together



y=2x

y=1x

y=1/2x

What is the general formula of the line?

y = mx + b

slope

y-intercept, which describes where the line crosses the y-axis

Outline

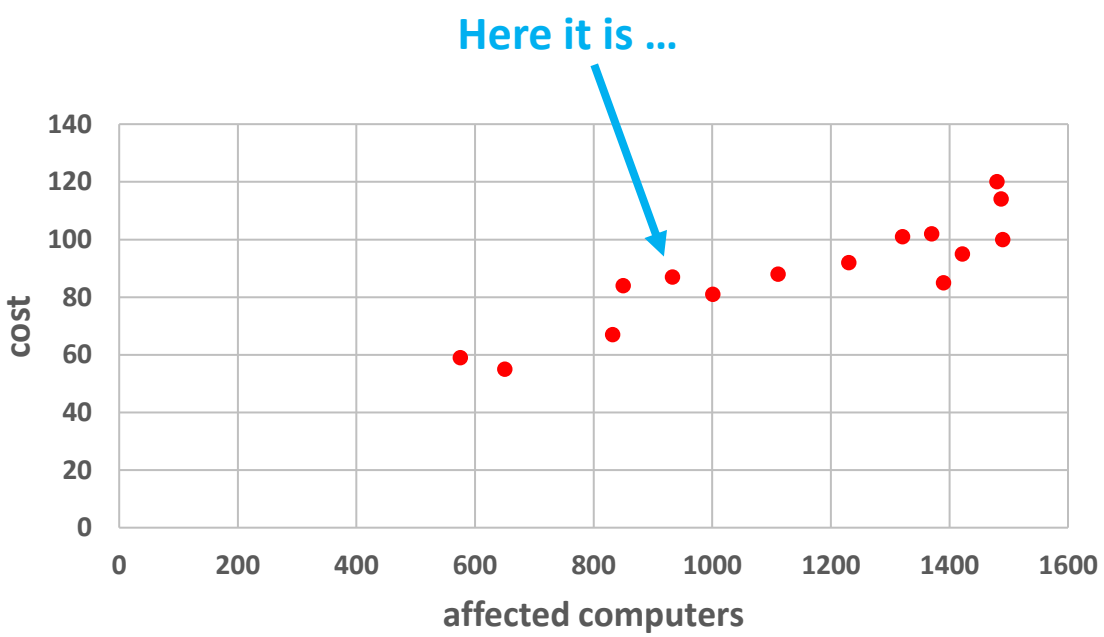
- Subject overview
- What is AI
- Brief introduction to cyber security
- How AI helps Cyber security
- Limitations of AI in Security
- Math preliminaries
- **Linear regression**

Example

- You are given the "right answer" (i.e., cost of security incident) for each example (i.e., #affected computers) in the data

# computer	cost
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Scatter Plot

What is the cost of the security incident where there are **933** computers that have been affected?

Answer: 87

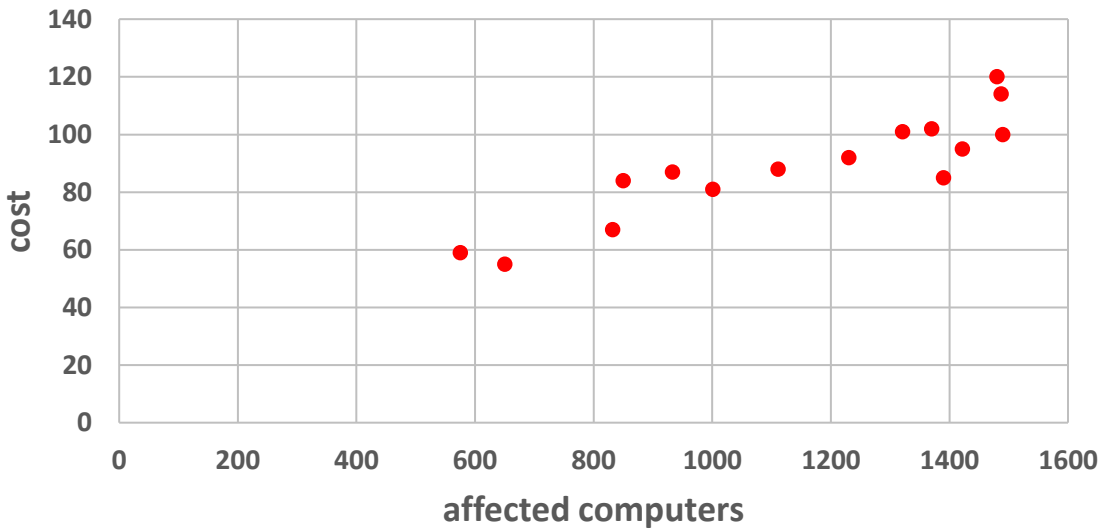
Example

- You are given the "right answer" (i.e., cost of security incident) for each example (i.e., #affected computers) in the data

?

# computer	cost
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Scatter Plot

What is the cost of the security incident where there are **900** computers that have been affected?

Answer: NOT in the dataset!

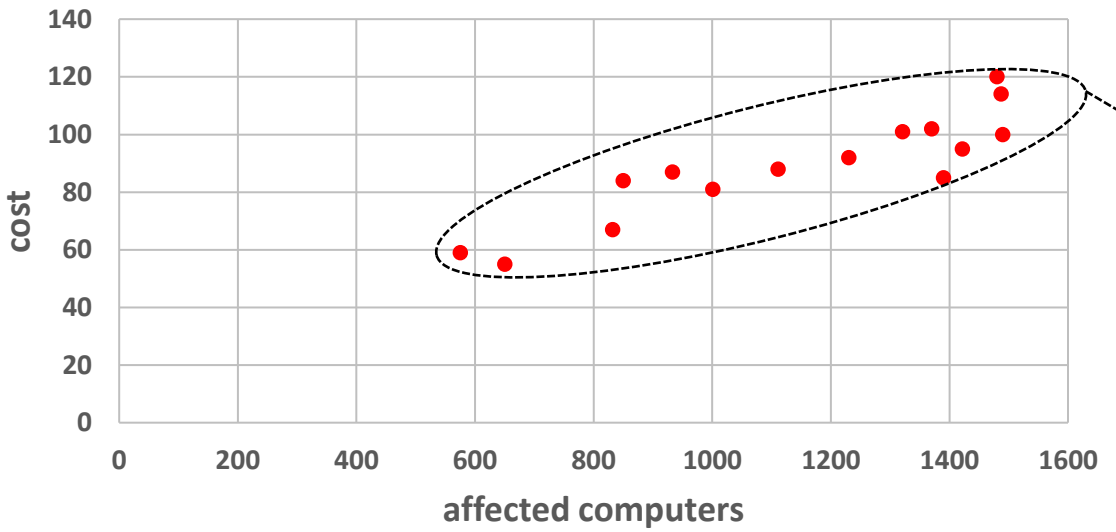
But we can *predict* it using machine learning

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



There seems to be a ***pattern***!

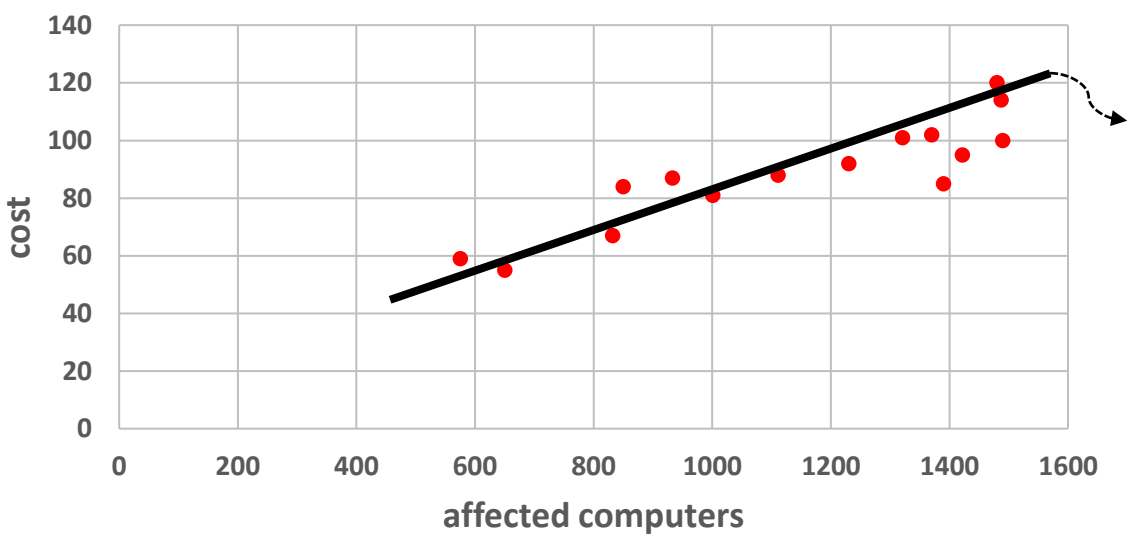
Scatter Plot

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Scatter Plot

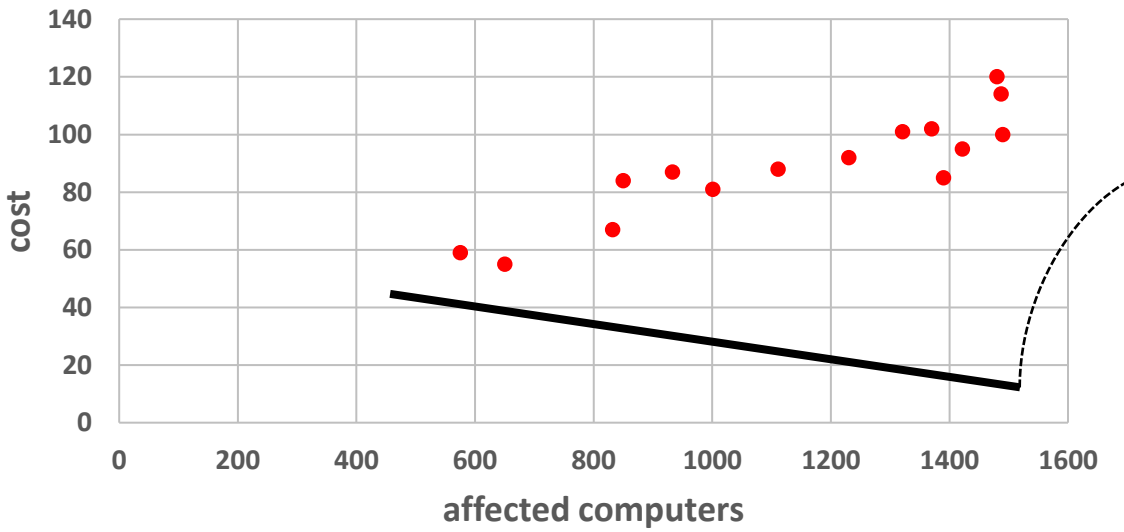
Which could be potentially captured (or fit) through a ***line***

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Scatter Plot

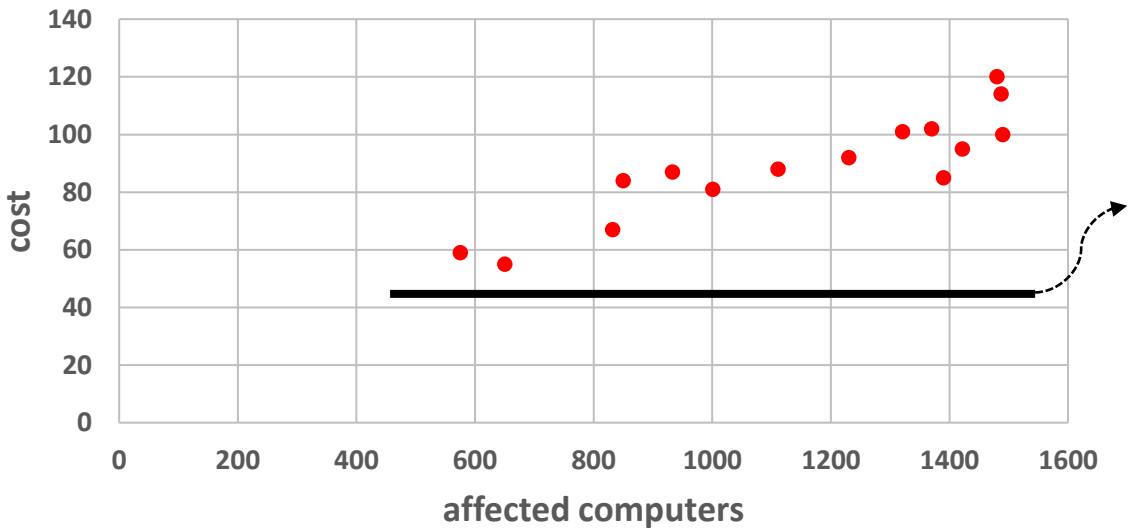
This line is far off and, accordingly, not a good fit!

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



This line is also not a good fit!

Scatter Plot

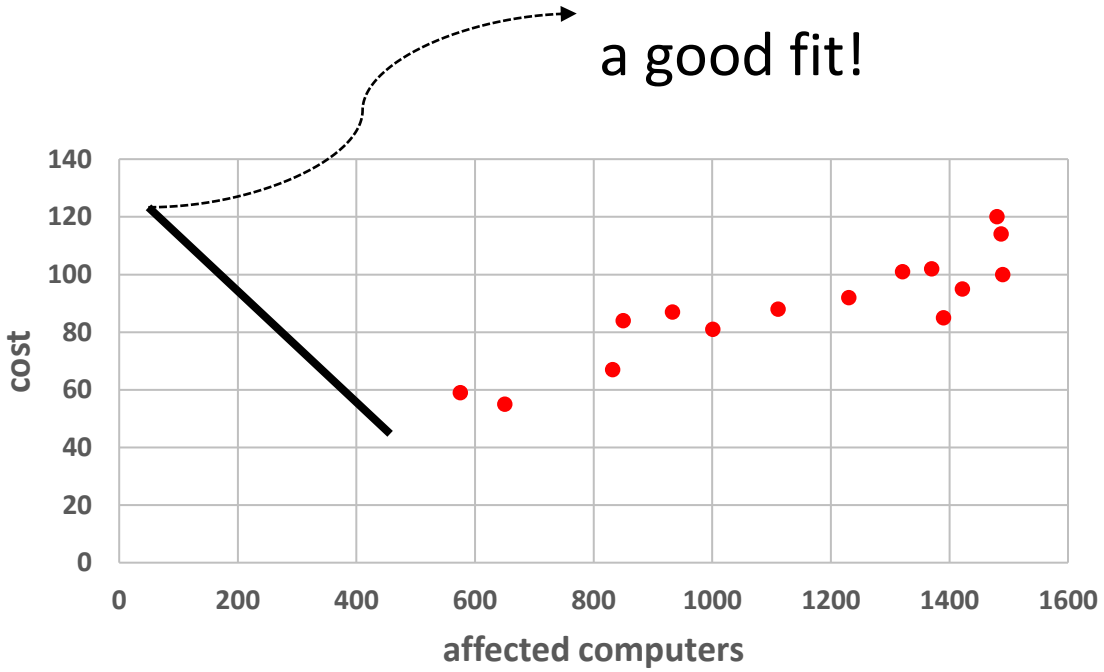
Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset

This line is also not a good fit!



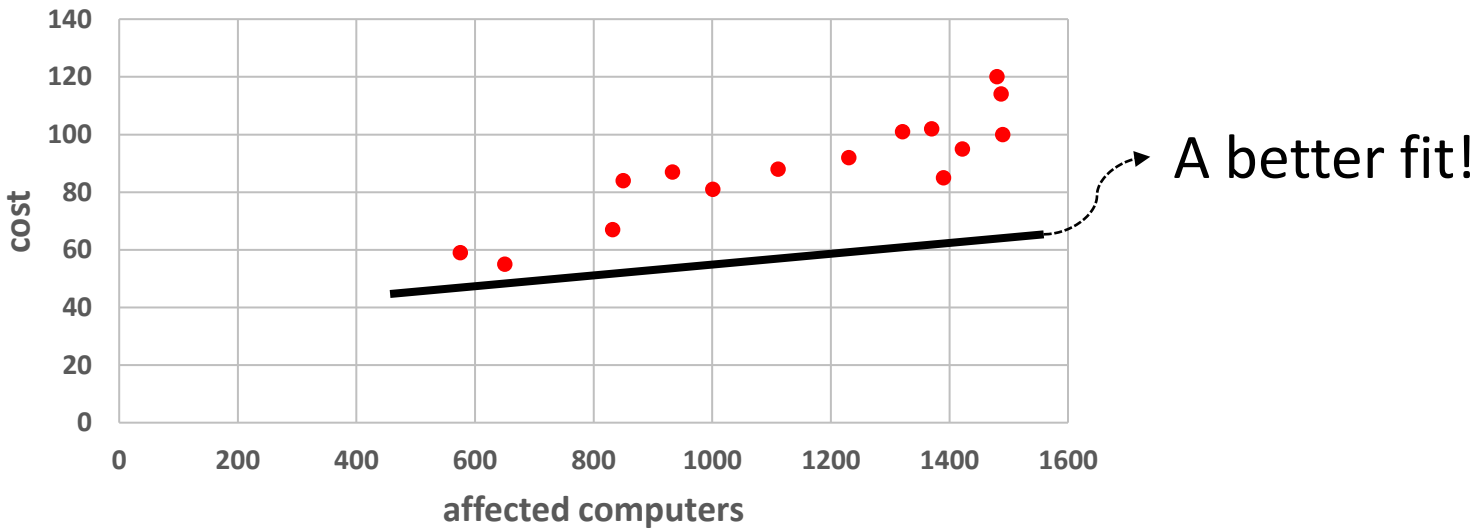
Scatter Plot

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



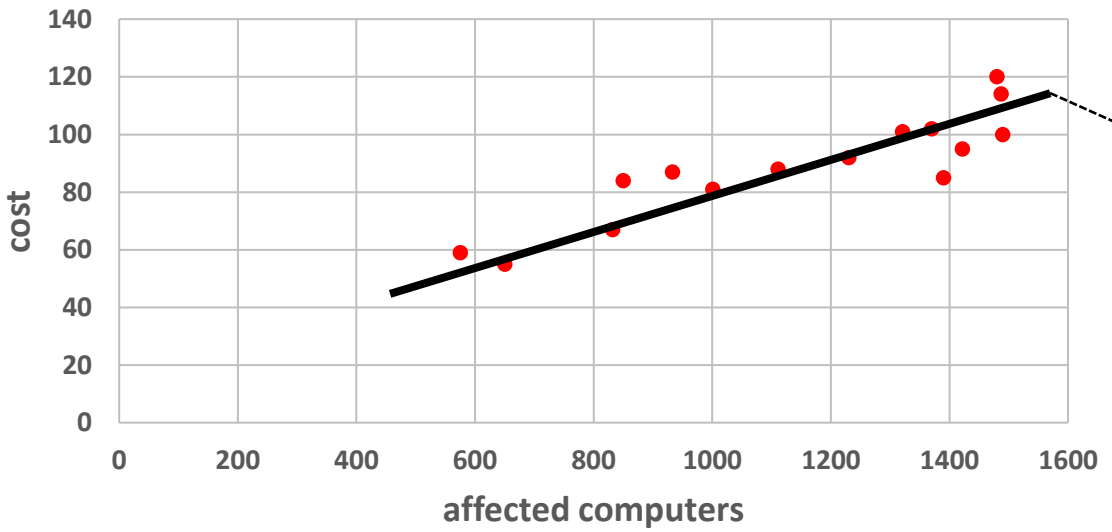
Scatter Plot

Predicting the Unknown

- How to start?
  - Let us visualize the data

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Yet, a much better fit!

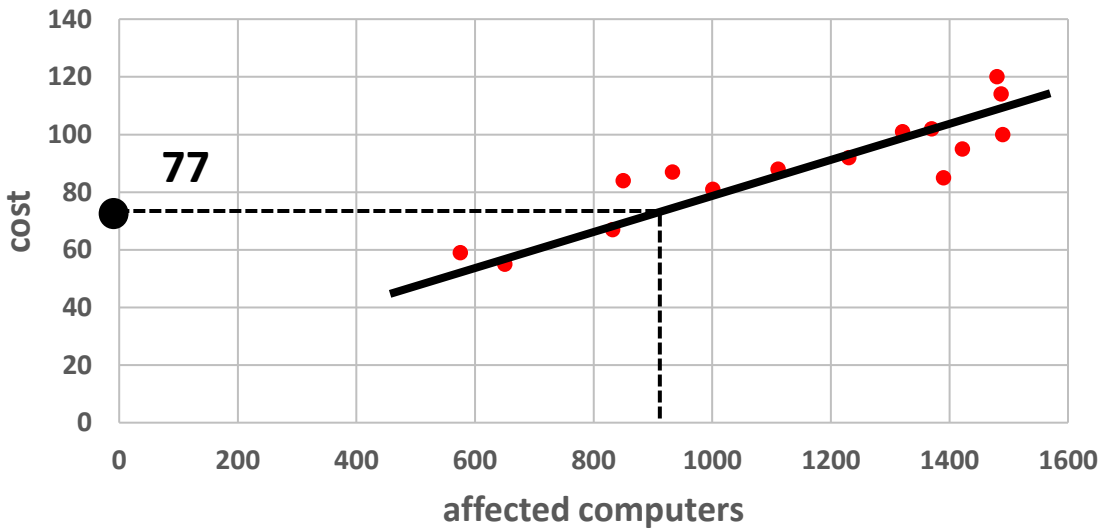
Scatter Plot

Predicting the Unknown

- Assume there is a way for us to find and fit this line around the given dataset, how can we use it to predict an unseen data point?

# computer (*x*)	cost (*y*)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



Scatter Plot

What will be the cost of the security incident where there are **900** computers that have been affected?

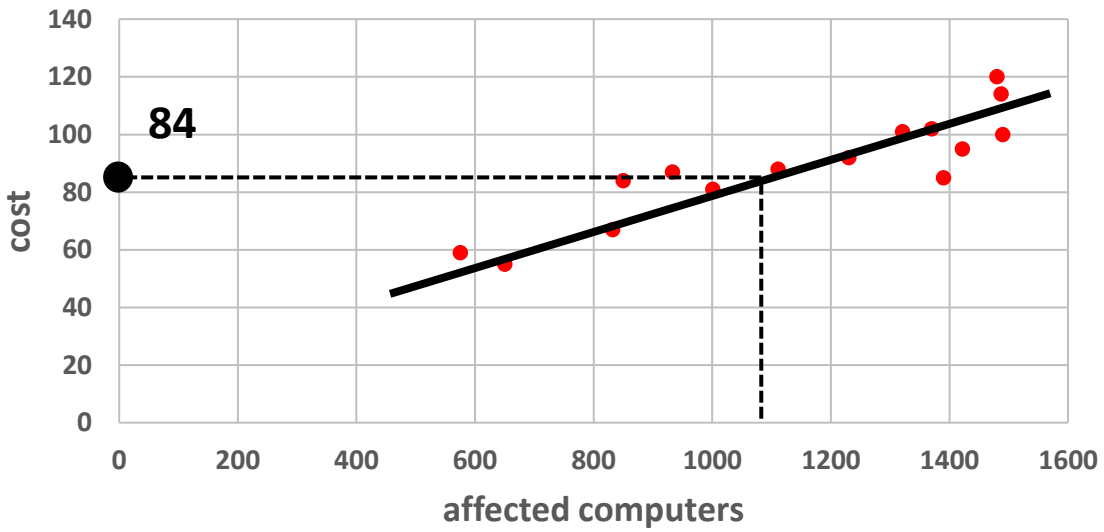
Answer: 77

Predicting the Unknown

- Assume there is a way for us to find and fit this line around the given dataset, how can we use it to predict an unseen data point?

# computer (*x*)	cost (*y*)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



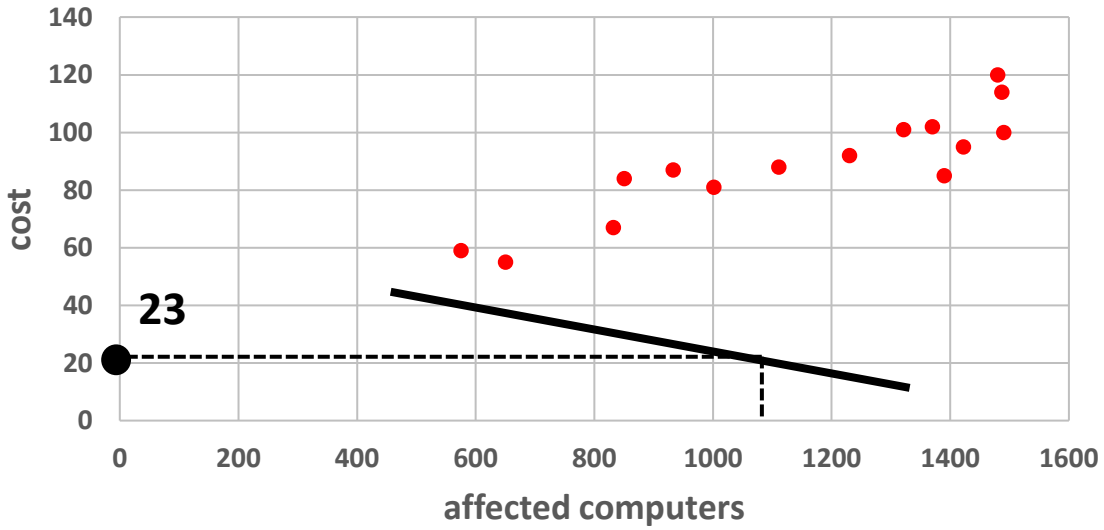
Scatter Plot

What will be the cost of the security incident where there are **1090** computers that have been affected?

Answer: 84

Predicting the Unknown

- The better the fit, the higher the accuracy of prediction

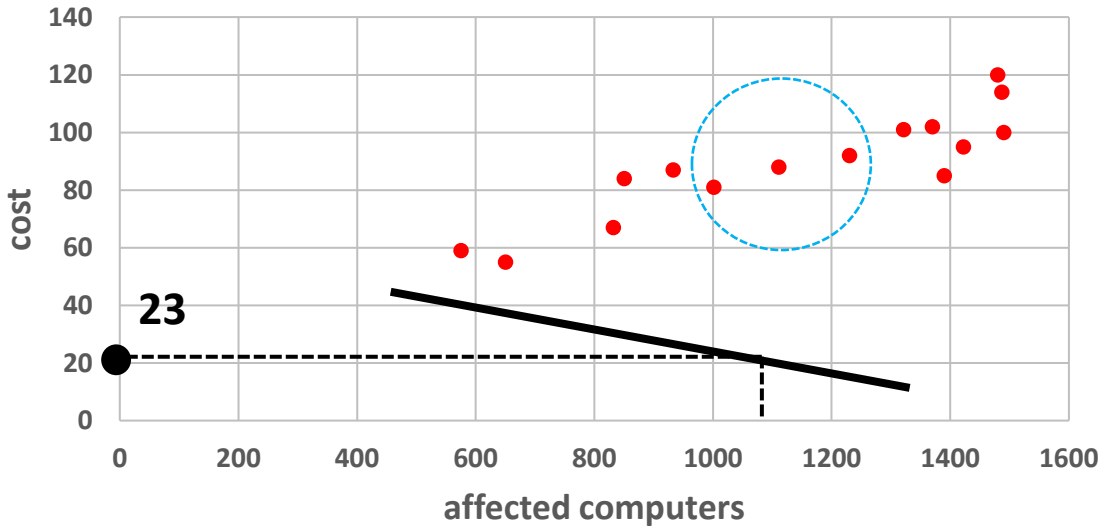


What will be the cost of the security incident where there are **1090** computers that have been affected?

Answer: 23

Predicting the Unknown

- The better the fit, the higher the accuracy of your predictions

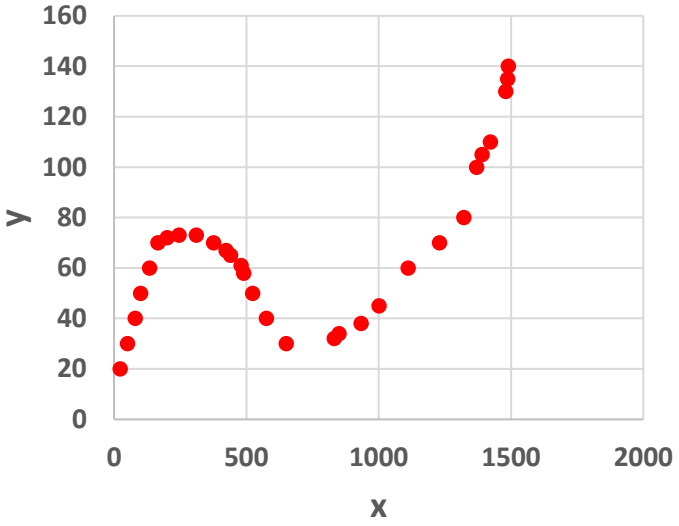


The prediction given by this line is unreasonable

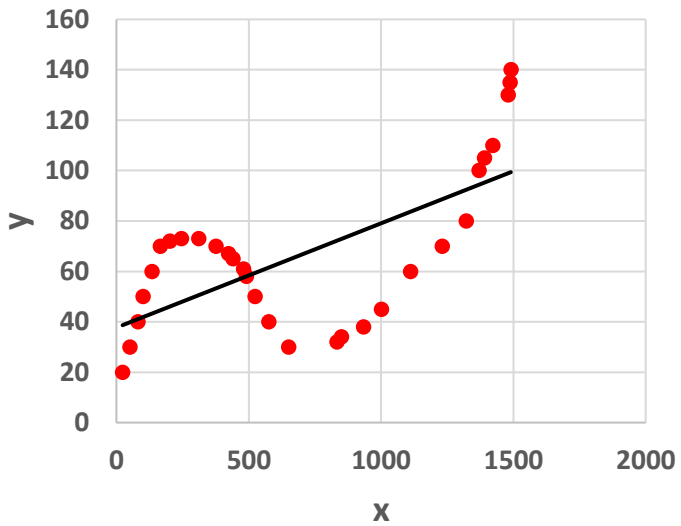
The whole idea becomes, how to fit a good function (here a *line*) which is more generally referred to as a ***model***

Mathematical Models

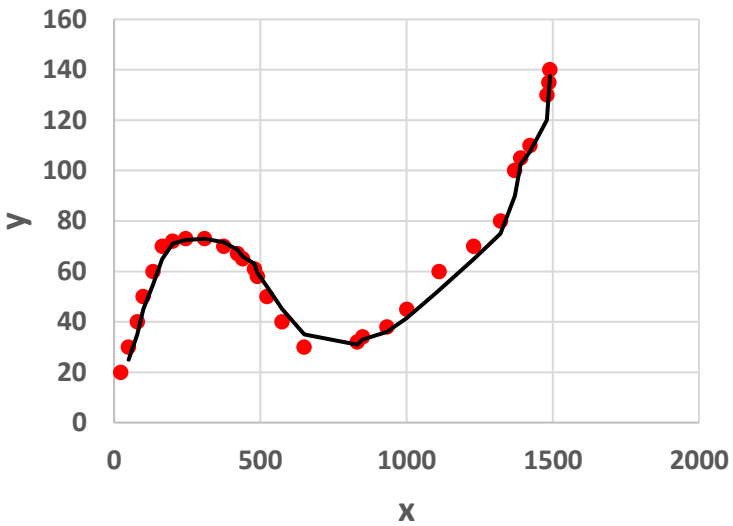
- Of course, a model needs not be a line!



Consider this data



Line is not a good fit



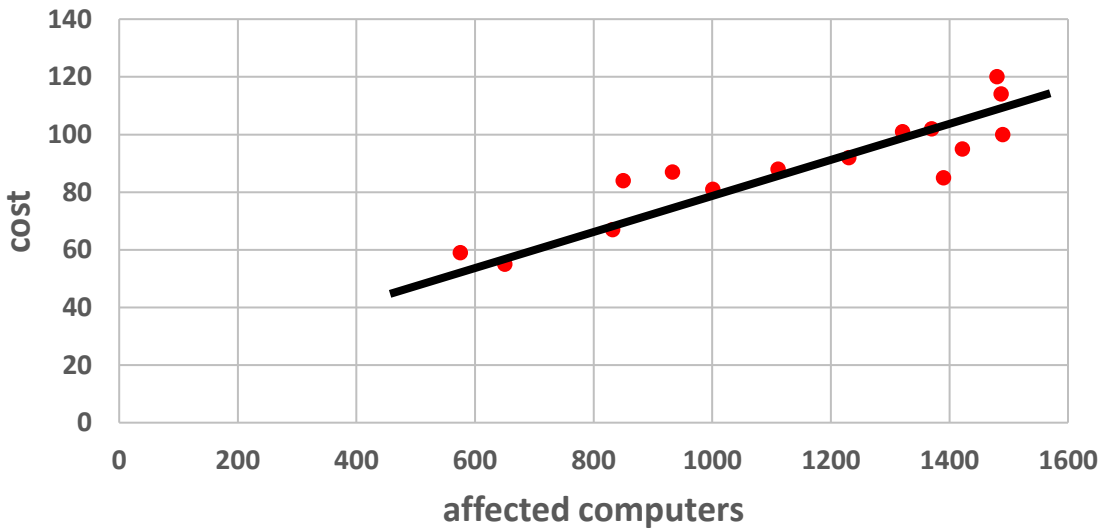
This model is a better fit

Learning a Model

- How to fit a mathematical model, after which you can predict any *target* value (e.g., cost) given a *feature* value (e.g., # computers)?

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



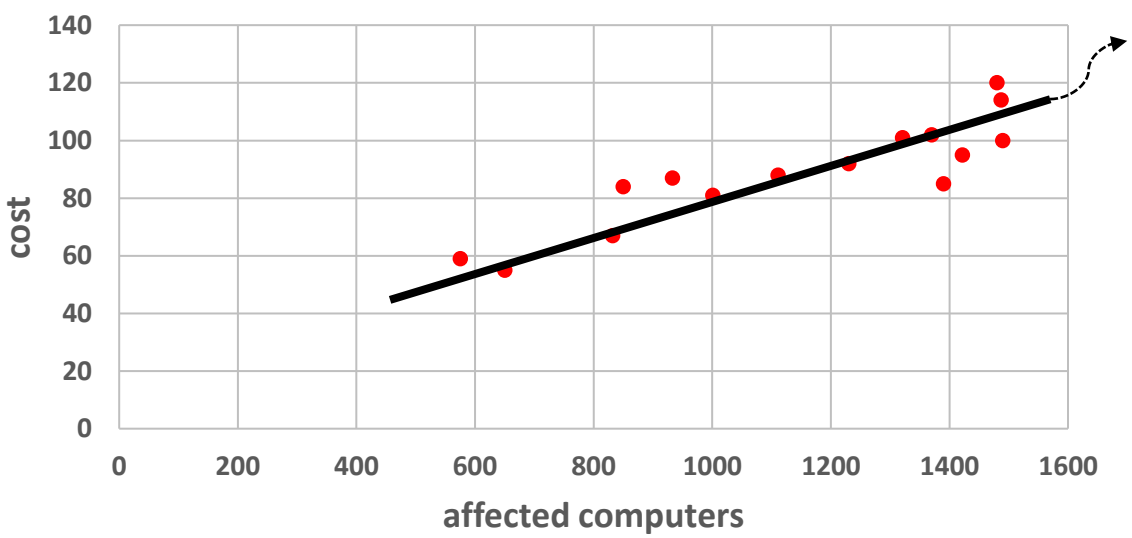
In other words, how to ***learn*** a model (e.g., a line) given a dataset?

Learning a Model

- How do we represent a line in mathematics?
  - $y = mx + b$

# computer (x)	cost (y)
575	59
650	55
832	67
850	84
933	87
1001	81
1111	88
1230	92
1321	101
1370	102
1390	85
1422	95
1480	120
1487	114
1490	100

Dataset



For this best fitting line:

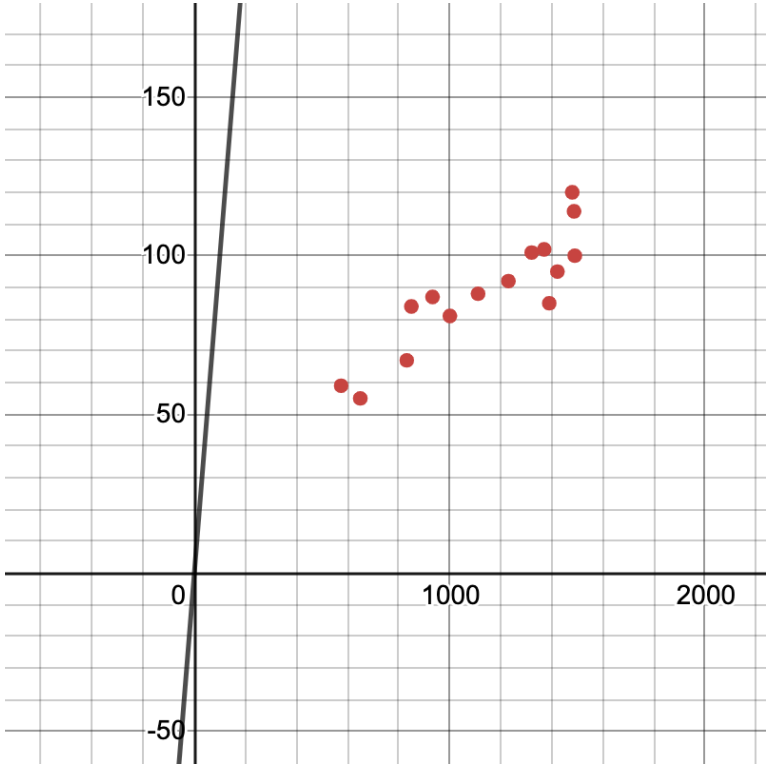
$m = 0.052$

$b = 29.21$

But, how can we find (or *learn*) m and b?

Learning a Model

- Let us try different random values of ***m*** and ***b***:
  - ***m*** = 1 and ***b*** = 0

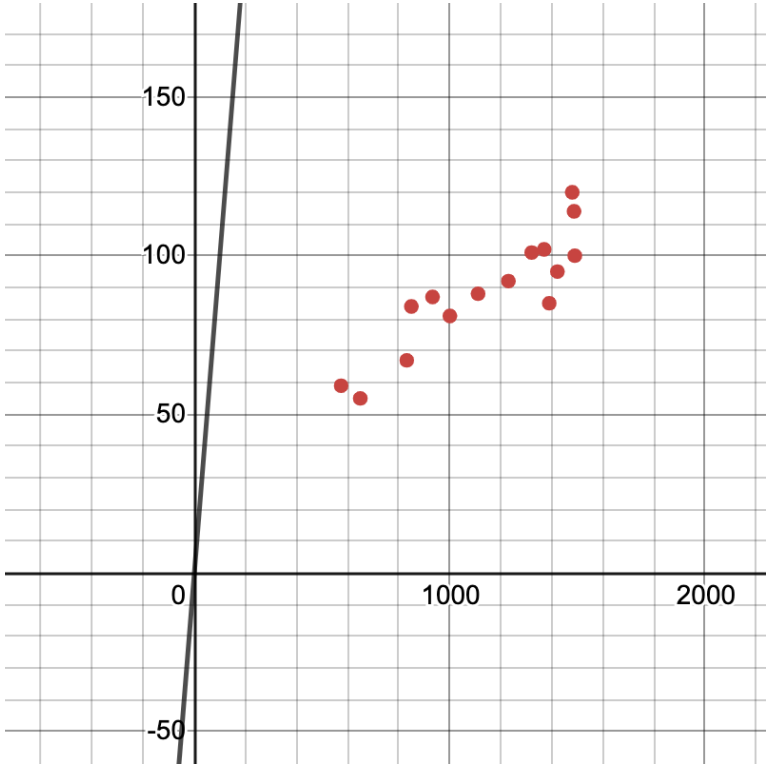


# computer (*x*)	Actual cost (*y*)	Predicted cost (y')
575	59	575
650	55	650
832	67	832
850	84	850
933	87	933
1001	81	1001
1111	88	1111
1230	92	1230
1321	101	1321
1370	102	1370
1390	85	1390
1422	95	1422
1480	120	1480
1487	114	1487
1490	100	1490

$$y' = \boldsymbol{m}x + \boldsymbol{b}$$
$$= 1*575 + 0$$
$$= 575$$

Learning a Model

- Let us try different random values of ***m*** and ***b***:
  - ***m*** = 1 and ***b*** = 0



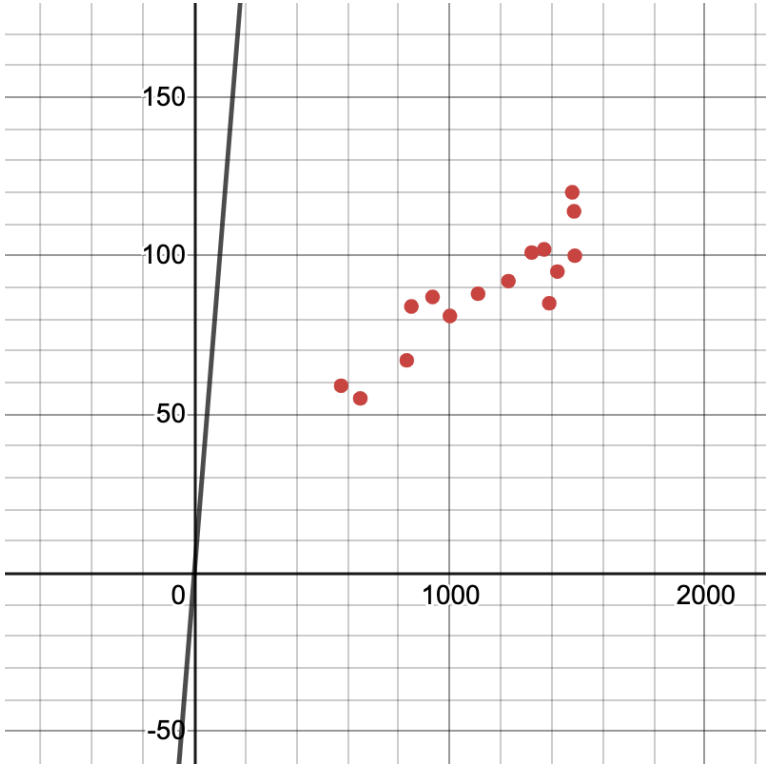
# computer (*x*)	Actual cost (*y*)	Predicted cost (y')
575	59	575
650	55	650
832	67	832
850	84	850
933	87	933
1001	81	1001
1111	88	1111
1230	92	1230
1321	101	1321
1370	102	1370
1390	85	1390
1422	95	1422
1480	120	1480
1487	114	1487
1490	100	1490

$$y' = \boldsymbol{m}x + \boldsymbol{b}$$
$$= 1*1001 + 0$$
$$= 1001$$

How close are these predicted cost to the actual ones?

Learning a Model

- Let us try different random values of ***m*** and ***b***:
  - ***m*** = 1 and ***b*** = 0



The ***Error***

# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	Predicted – Actual (y' – y)
575	59	575	516
650	55	650	595
832	67	832	765
850	84	850	766
933	87	933	846
1001	81	1001	920
1111	88	1111	1023
1230	92	1230	1138
1321	101	1321	1220
1370	102	1370	1268
1390	85	1390	1305
1422	95	1422	1327
1480	120	1480	1360
1487	114	1487	1373
1490	100	1490	1390

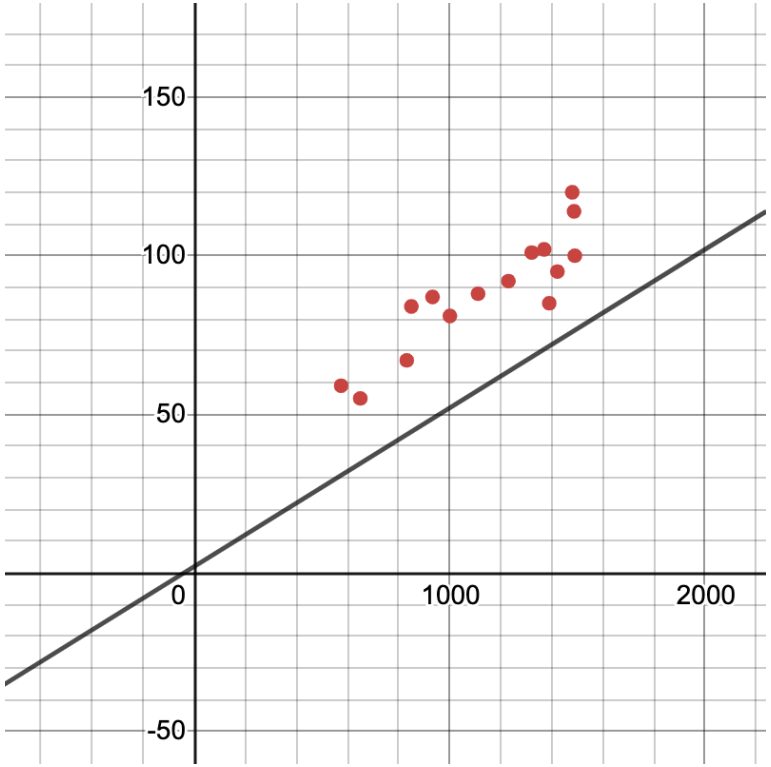
Sum = 15812

≡

∑ = 15812

Learning a Model

- Let us try different random values of ***m*** and ***b***:
  - ***m*** = 0.05 and ***b*** = 2



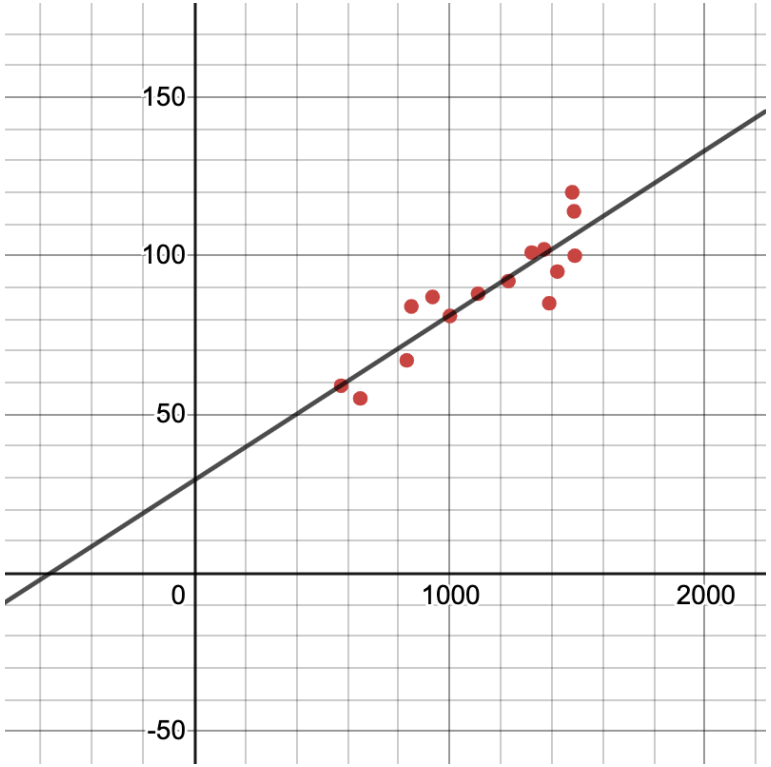
# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	Predicted – Actual (y' – y)
575	59	30.75	-28.25
650	55	34.5	-20.5
832	67	43.6	-23.4
850	84	44.5	-39.5
933	87	48.65	-38.35
1001	81	52.05	-28.95
1111	88	57.55	-30.45
1230	92	63.5	-28.5
1321	101	68.05	-32.95
1370	102	70.5	-31.5
1390	85	71.5	-13.5
1422	95	73.1	-21.9
1480	120	76	-44
1487	114	76.35	-37.65
1490	100	76.5	-23.5

Error:

∑ = -442.9

Learning a Model

- Let us try different random values of ***m*** and ***b***:
  - ***m*** = 0.052 and ***b*** = 29.21



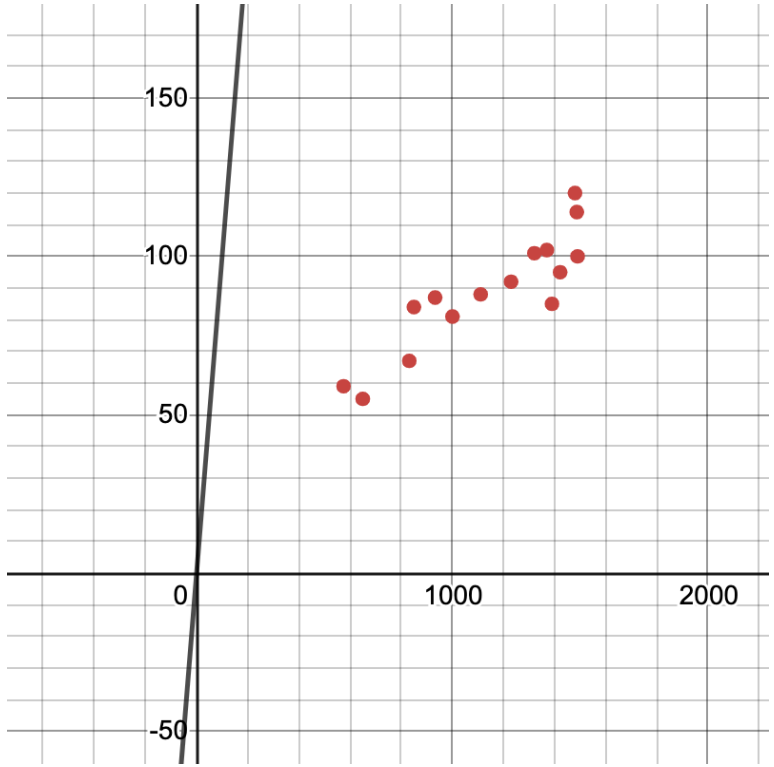
# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	Predicted – Actual (y' – y)
575	59	59.11	0.11
650	55	63.01	8.01
832	67	72.474	5.474
850	84	73.41	-10.59
933	87	77.726	-9.274
1001	81	81.262	0.262
1111	88	86.982	-1.018
1230	92	93.17	1.17
1321	101	97.902	-3.098
1370	102	100.45	-1.55
1390	85	101.49	16.49
1422	95	103.154	8.154
1480	120	106.17	-13.83
1487	114	106.534	-7.466
1490	100	106.69	6.69

Error:

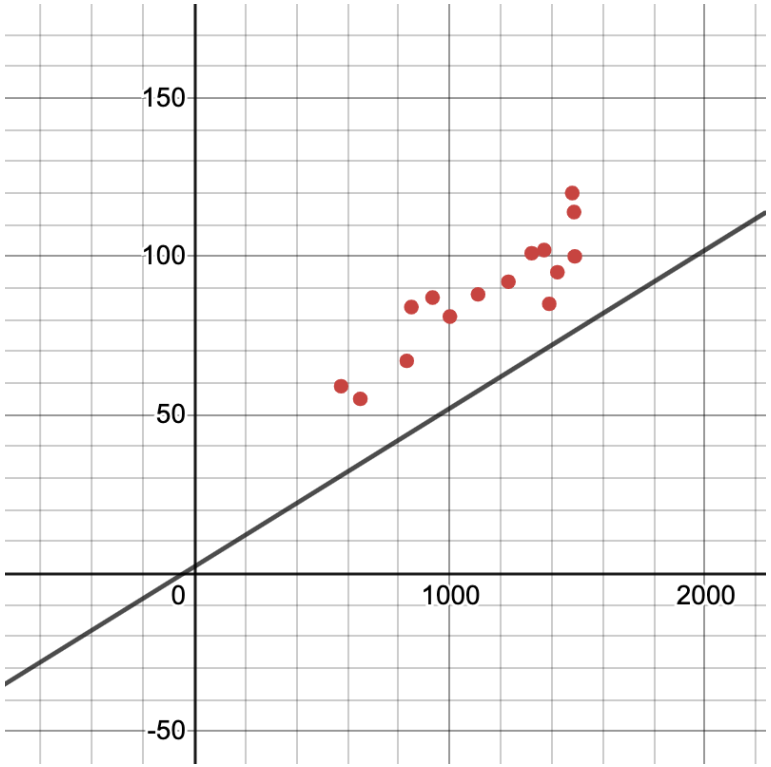
∑ = -0.466

Learning a Model

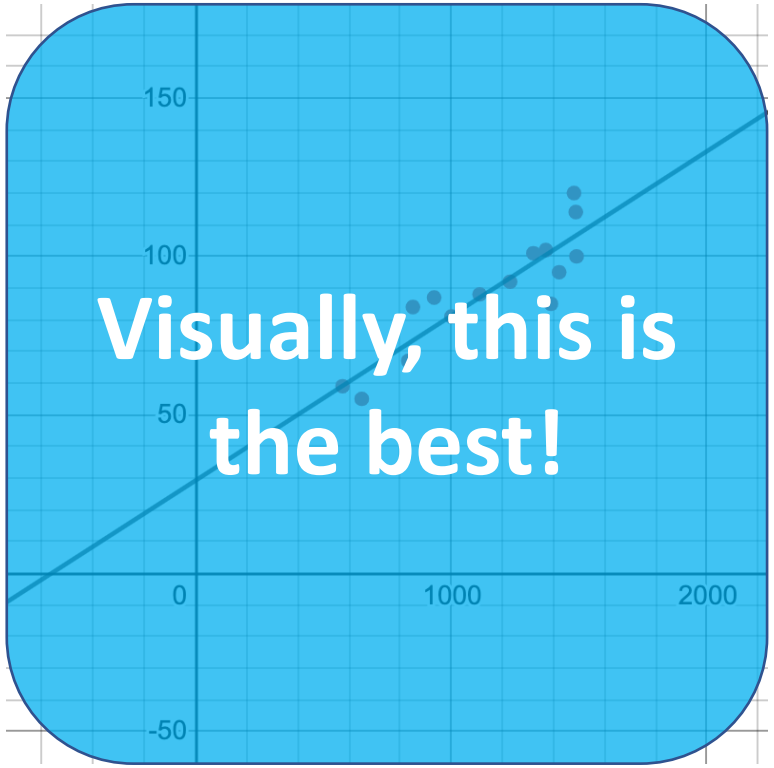
- Let us observe the three options besides each other



m = 1 and ***b*** = 0



m = 0.05 and ***b*** = 2



m = 0.052 and ***b*** = 29.21

Learning a Model

- Let us compare their errors

Learning a Model

- Let us compare their errors

# computer (x)	Actual cost (y)	m = 1 and b = 0: Predicted cost (y')	m = 1 and b = 0: (Predicted – Actual)^2 $(y' - y)^2$	m = 0.05 and b = 2: Predicted cost (y')	m = 0.05 and b = 2: (Predicted – Actual)^2 $(y' - y)^2$	m = 0.052 and b = 29.21: Predicted cost (y')	m = 0.052 and b = 29.21: (Predicted – Actual)^2 $(y' - y)^2$
575	59	59.11	266256	30.75	798.0625	59.11	0.0121
650	55	63.01	354025	34.5	420.25	63.01	64.1601
832	67	72.474	585225	43.6	547.56	72.474	29.964676
850	84	73.41	586756	44.5	1560.25	73.41	112.1481
933	87	77.726	715716	48.65	1470.7225	77.726	86.007076
1001	81	81.262	846400	52.05	838.1025	81.262	0.068644
1111	88	86.982	1046529	57.55	927.2025	86.982	1.036324
1230	92	93.17	1295044	63.5	812.25	93.17	1.3689
1321	101	97.902	1488400	68.05	1085.7025	97.902	9.597604
1370	102	100.45	1607824	70.5	992.25	100.45	2.4025
1390	85	101.49	1703025	71.5	182.25	101.49	271.9201
1422	95	103.154	1760929	73.1	479.61	103.154	66.487716
1480	120	106.17	1849600	76	1936	106.17	191.2689
1487	114	106.534	1885129	76.35	1417.5225	106.534	55.741156
1490	100	106.69	1932100	76.5	552.25	106.69	44.7561

SQUARED ERRORS: $\sum = 17922958$ $\sum = 14019.9$ $\sum = 936.9$ ✓

Recap

- What is the idea?
  - start with a generic *model* or a *hypothesis* (e.g., a line),
  - then keep varying its paramters (i.e., ***m*** and ***b*** for the line) until you get the *minimum squared error*
    - This process is called *learning*
    - And an algorithm that can be used to vary the paramters to locate the minimum squared error is called *gradient descent*

  - Once you learn a model, you can use it to predict unseen values!

References

- *Machine Learning , Analytics & Cyber Security the Next Level Threat Analytics, Manjunath N V*
- *Data mining for security at Google, Max Poletto Google security team*
- *WSABIE: scaling up to large vocabulary image annotation. Jason Weston, Samy Bengio, and Nicolas Usunier*
- *Machine learning foundational course, google developers*
- *AI for Medicine, Mohammad Hammoud, CMU Qatar*
- *Practical Machine Learning in Infosecurity, Clarence Chio and Anto Joseph*
- *Machine Learning Core Concept, Mohammad Hammoud, CMU in Qatar*
- *Linear regression, Mohammad Hammoud, CMU in Qatar*

Training Algorithm to Learn

CSIT375 AI for Cybersecurity

Dr Wei Zong

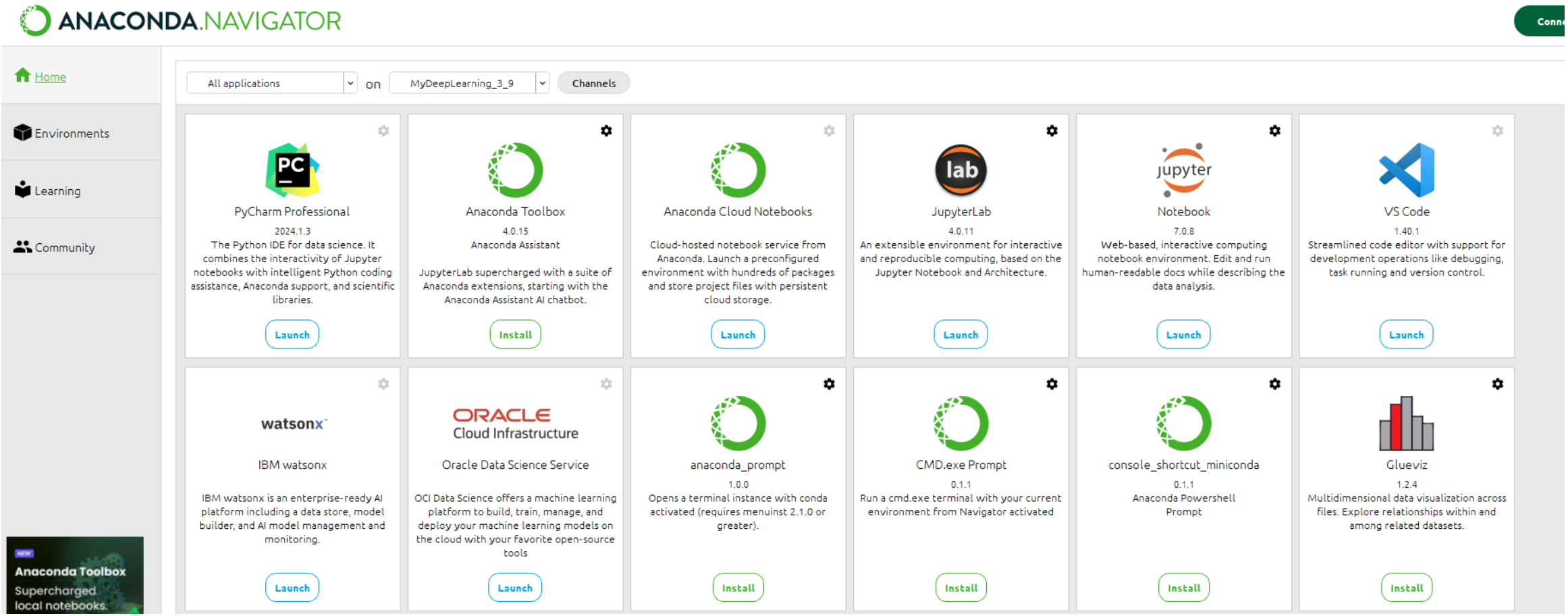
SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

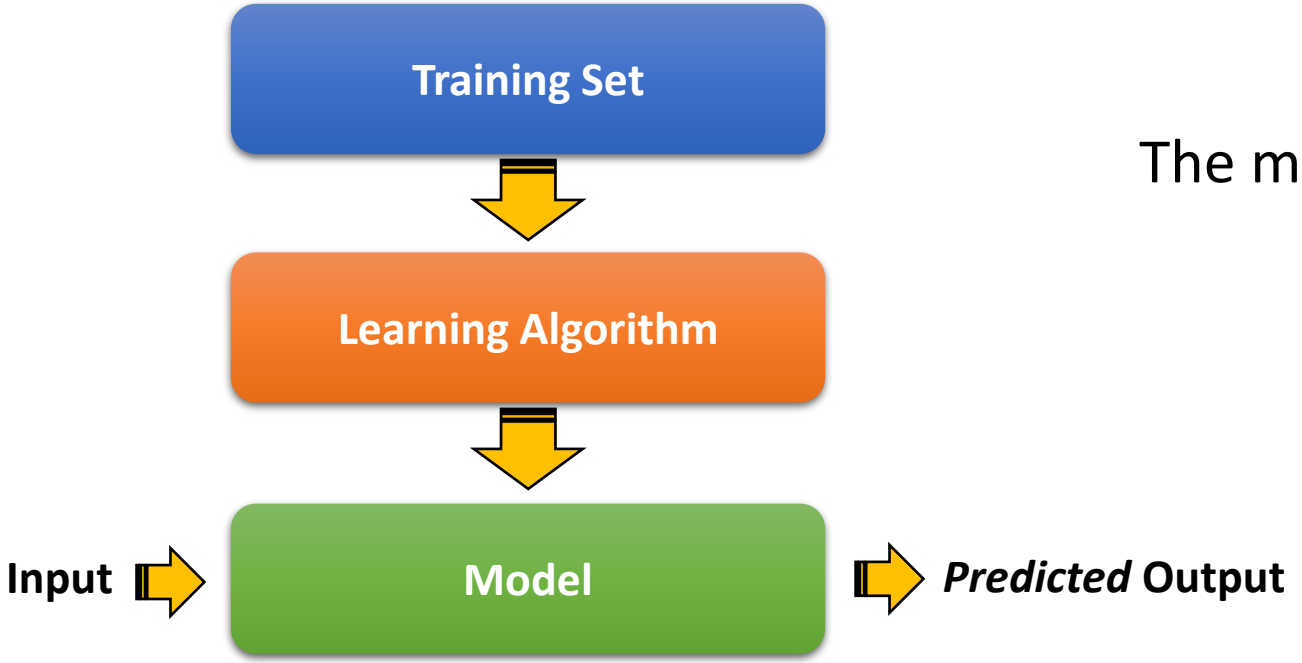
- Setup Jupyter notebook with Anaconda
- Recap: ML Problems and Approaches
- Machine Learning in Practice: a working example
- Training Algorithms to Learn – logistic regression
- Evaluating Models
- Practical Considerations

Setup Jupyter notebook with Anaconda



Recap: What is Machine Learning?

- Machine learning is the process of fitting a mathematical model to a given dataset (called the *training set*), then using this model to predict answers for any new unseen inputs



The model that we learnt is called ***Linear Regression***

Mean Squared Error (MSE)

- Let us compare their errors

# computer (x)	Actual cost (y)	m = 1 and b = 0: Predicted cost (y')	m = 1 and b = 0: (Predicted – Actual)^2 $(y' - y)^2$	m = 0.05 and b = 2: Predicted cost (y')	m = 0.05 and b = 2: (Predicted – Actual)^2 $(y' - y)^2$	m = 0.052 and b = 29.21: Predicted cost (y')	m = 0.052 and b = 29.21: (Predicted – Actual)^2 $(y' - y)^2$
575	59	59.11	266256	30.75	798.0625	59.11	0.0121
650	55	63.01	354025	34.5	420.25	63.01	64.1601
832	67	72.474	585225	43.6	547.56	72.474	29.964676
850	84	73.41	586756	44.5	1560.25	73.41	112.1481
933	87	77.726	715716	48.65	1470.7225	77.726	86.007076
1001	81	81.262	846400	52.05	838.1025	81.262	0.068644
1111	88	86.982	1046529	57.55	927.2025	86.982	1.036324
1230	92	93.17	1295044	63.5	812.25	93.17	1.3689
1321	101	97.902	1488400	68.05	1085.7025	97.902	9.597604
1370	102	100.45	1607824	70.5	992.25	100.45	2.4025
1390	85	101.49	1703025	71.5	182.25	101.49	271.9201
1422	95	103.154	1760929	73.1	479.61	103.154	66.487716
1480	120	106.17	1849600	76	1936	106.17	191.2689
1487	114	106.534	1885129	76.35	1417.5225	106.534	55.741156
1490	100	106.69	1932100	76.5	552.25	106.69	44.7561

SQUARED ERRORS: $\sum = 17922958$ $\sum = 14019.9$ $\sum = 936.9$

Mean Squared Error (MSE)

- Let us compare their errors

		m = 1 and b = 0		m = 0.05 and b = 2		m = 0.052 and b = 29.21	
# computer (x)	Actual cost (y)	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$
575	59	59.11	266256	30.75	798.0625	59.11	0.0121
650	55	63.01	354025	34.5	420.25	63.01	64.1601
832	67	72.474	585225	43.6	547.56	72.474	29.964676
850	84	73.41	586756	44.5	1560.25	73.41	112.1481
933	87	77.726	715716	48.65	1470.7225	77.726	86.007076
1001	81	81.262	846400	52.05	838.1025	81.262	0.068644
1111	88	86.982	1046529	57.55	927.2025	86.982	1.036324
1230	92	93.17	1295044	63.5	812.25	93.17	1.3689
1321	101	97.902	1488400	68.05	1085.7025	97.902	9.597604
1370	102	100.45	1607824	70.5	992.25	100.45	2.4025
1390	85	101.49	1703025	71.5	182.25	101.49	271.9201
1422	95	103.154	1760929	73.1	479.61	103.154	66.487716
1480	120	106.17	1849600	76	1936	106.17	191.2689
1487	114	106.534	1885129	76.35	1417.5225	106.534	55.741156
1490	100	106.69	1932100	76.5	552.25	106.69	44.7561

15 examples

MEAN SQUARED ERRORS: $\sum = 17922958/30$ $\sum = 14019.9/30$ $\sum = 936.9/30$

Mean Squared Error (MSE)

- Let us compare their errors

		m = 1 and ***b*** = 0		***m*** = 0.05 and ***b*** = 2		***m*** = 0.052 and ***b*** = 29.21	
# computer (x)	Actual cost (y)	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$
			266256		798.0625		0.0121
			354025		420.25		64.1601
			585225		547.56		29.964676
			586756		1560.25		112.1481
			715716		1470.7225		86.007076
			846400		838.1025		0.068644
			1046529		927.2025		1.036324
			1295044		812.25		1.3689
			1488400		1085.7025		9.597604
			1607824		992.25		2.4025
			1703025		182.25		271.9201
			1760929		479.61		66.487716
			1849600		1936		191.2689
			1885129		1417.5225		55.741156
			1932100		552.25		44.7561

n examples

Mean Squared Error:

$$\frac{1}{2n}\sum_{i=1}^{n}\left(y'^{(i)} - y^{(i)}\right)^2$$

Minimizing MSE to Learn a Model

- Let us compare their errors

# computer (x)	Actual cost (y)	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$	Predicted cost (y')	(Predicted – Actual)^2 $(y' - y)^2$
			266256		798.0625		0.0121
			354025		420.25		64.1601
			585225		547.56		29.964676
			586756		1560.25		112.1481
			715716		1470.7225		86.007076
			846400		838.1025		0.068644
			1046529		927.2025		1.036324
			1295044		812.25		1.3689
			1488400		1085.7025		9.597604
			1607824		992.25		2.4025
			1703025		182.25		271.9201
			1760929		479.61		66.487716
			1849600		1936		191.2689
			1885129		1417.5225		55.741156
			1932100		552.25		44.7561

m = 1 and b = 0

m = 0.05 and b = 2

m = 0.052 and b = 29.21

n examples

Mean Squared Error:

$$\text{minimize} \frac{1}{2n} \sum_{i=1}^{n} \left(y'^{(i)} - y^{(i)} \right)^2$$

The objective is to minimize the mean squared error

Learning a Line via Minimizing MSE

- How to learn a line of equation y’ = mx + b given a *labelled* dataset?
  - By minimizing ***mean squared error***. That is:

$$\text{minimize} \frac{1}{2n} \sum_{i=1}^{n} \left(y'^{(i)} - y^{(i)} \right)^2$$

$$\equiv$$

$$\underset{m,b}{\text{minimize}} \frac{1}{2n} \sum_{i=1}^{n} \left((mx + b)^{(i)} - y^{(i)} \right)^2$$

Learning a Linear Regression Model via Minimizing a Cost Function

- Or, how to learn a *linear regression model* $\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 x}$ given a labelled dataset ($\boldsymbol{\theta}_0$ = b, $\boldsymbol{\theta}_1$= m, and $\boldsymbol{h_\theta(x)}$= y' when using y' = mx + b)?
  - By minimizing ***mean squared error***. That is:

This problem is referred to as an ***optimization problem*** with the objective of:

$$\underset{\theta_0,\theta_1}{\text{minimizing}}\, \boldsymbol{J(\theta_0, \theta_1)}$$

$$\text{minimize} \frac{1}{2n} \sum_{i=1}^{n} \left(y'^{(i)} - y^{(i)} \right)^2$$

$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}} \boxed{\frac{1}{2n} \sum_{i=1}^{n} \left((\theta_0 + \theta_1 x)^{(i)} - y^{(i)} \right)^2}$$

Known as ***cost function*** $\boldsymbol{J(\theta_0, \theta_1)}$

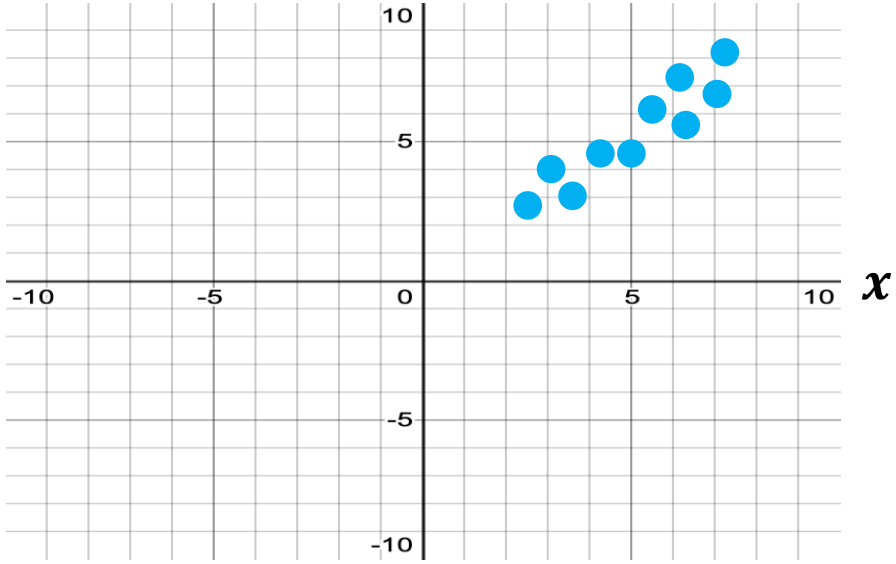
$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_0, \theta_1)}$$

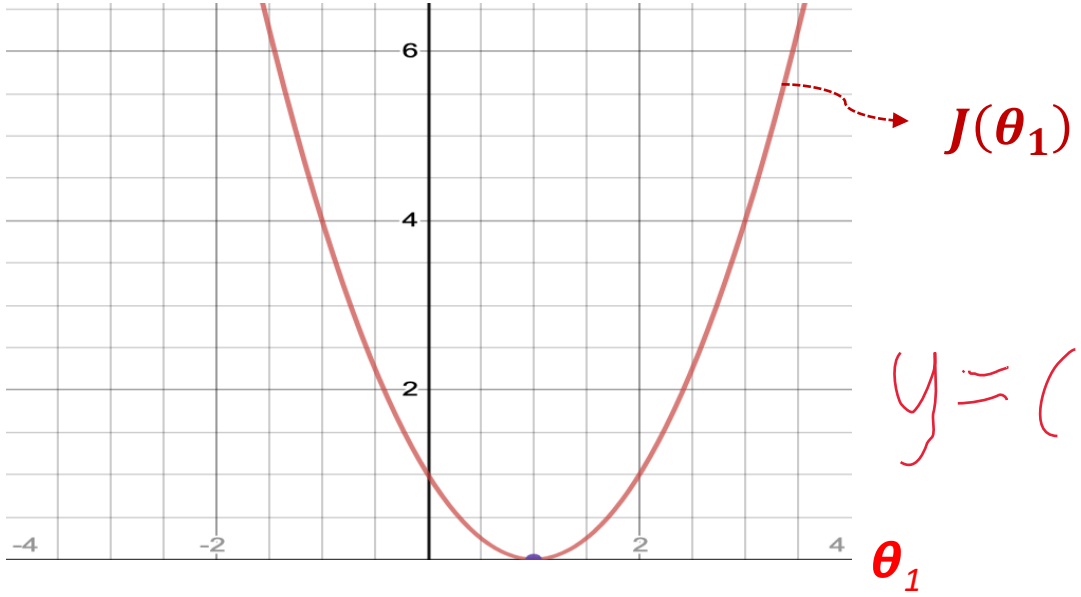
Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$

$\boldsymbol{y' = h_\theta(x)}$



$\boldsymbol{J}$



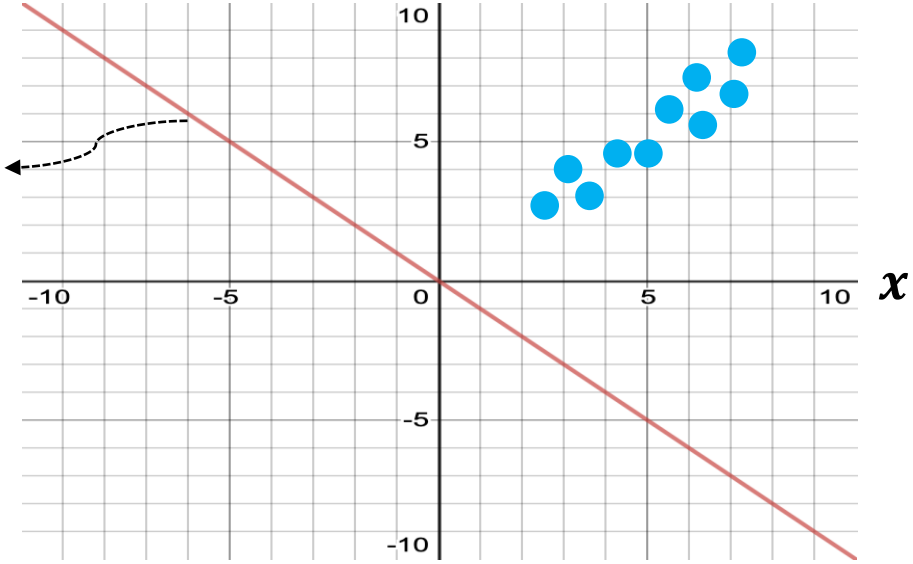
$y = (x-1)^2$

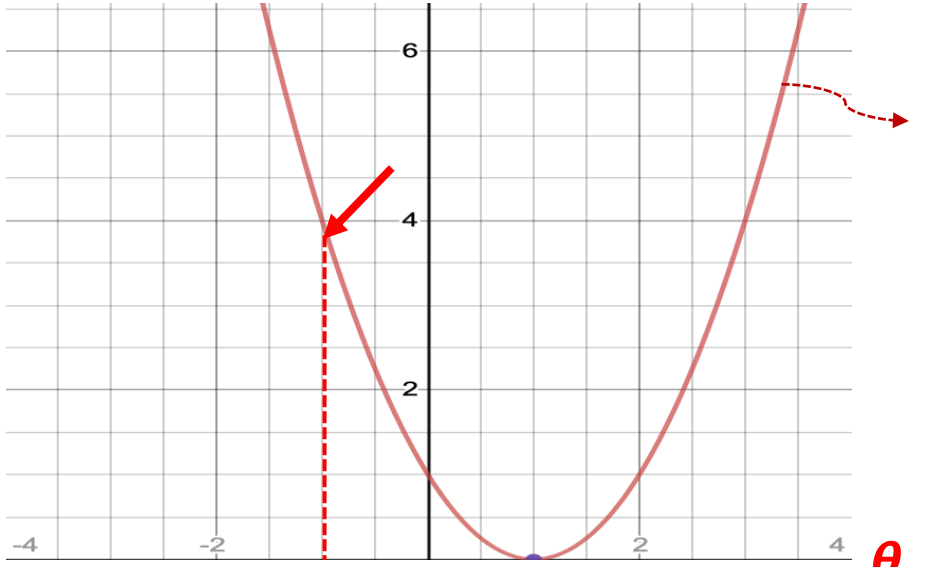
Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$

$\boldsymbol{y' = h_\theta(x)}$

$\boldsymbol{y' = \theta_1 x}$
$\boldsymbol{= -1x}$

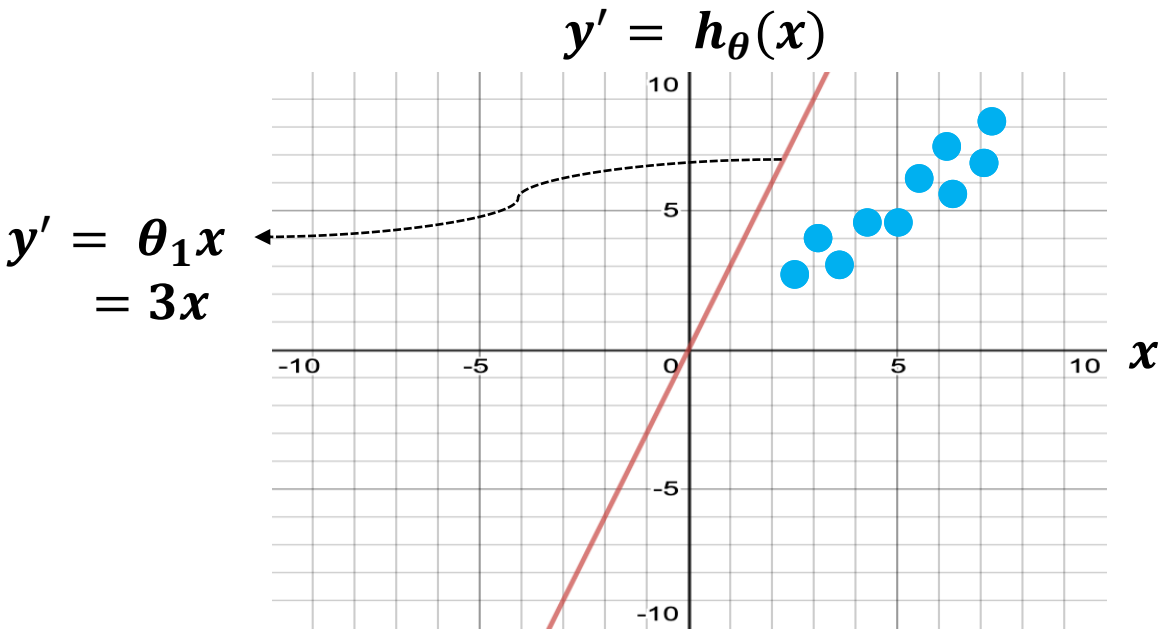


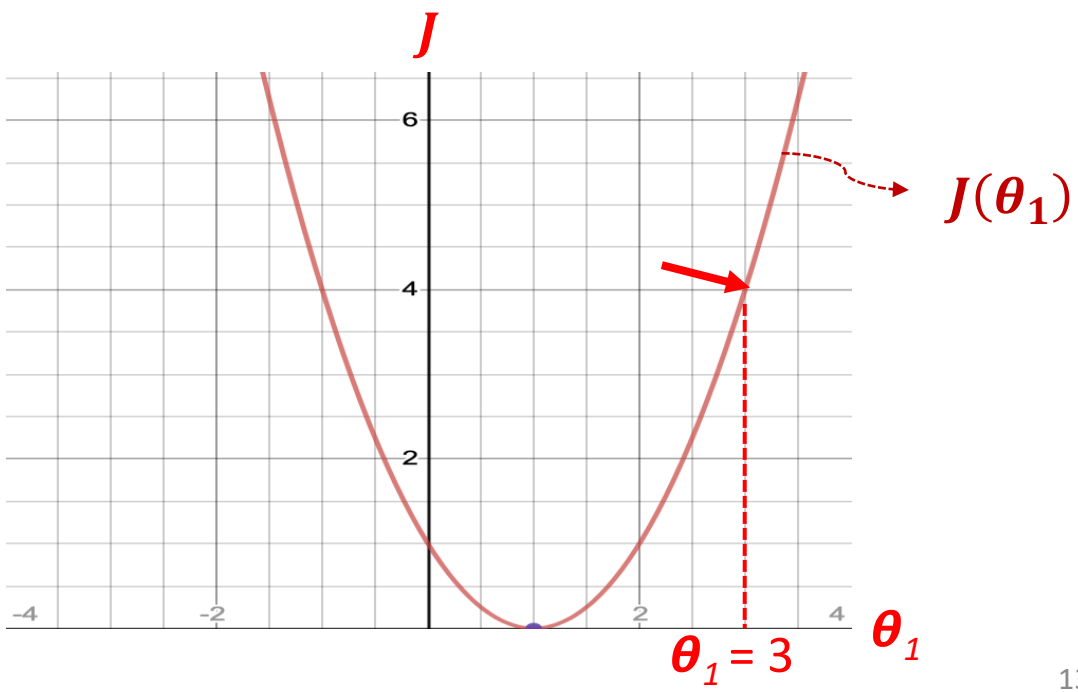


$\boldsymbol{\theta_1} = -1$

Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$



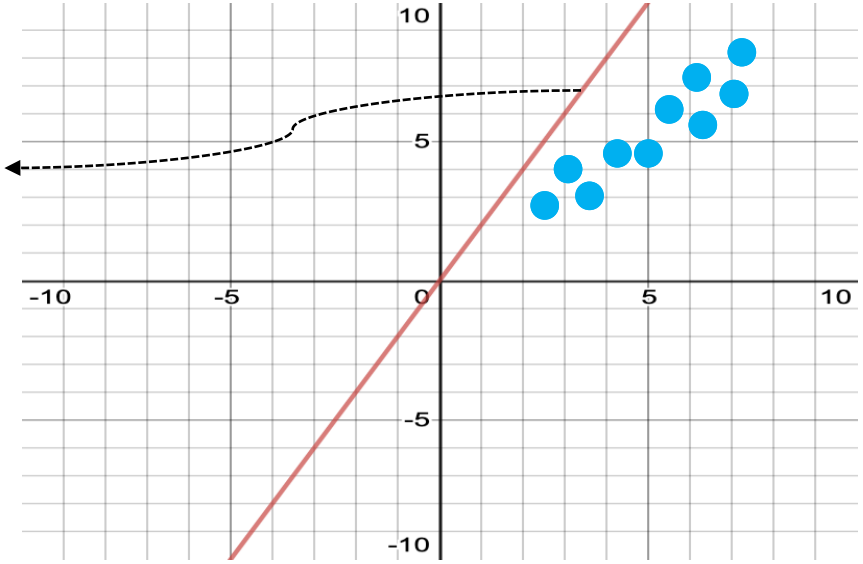


Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$

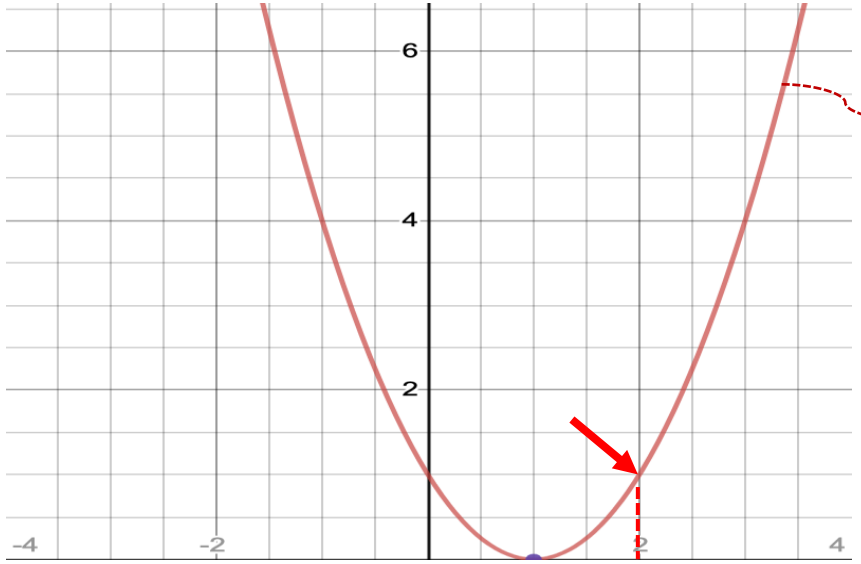
$\boldsymbol{y' = h_\theta(x)}$

$\boldsymbol{y' = \theta_1 x}$
$\boldsymbol{= 2x}$



$\boldsymbol{x}$

$\boldsymbol{J}$



$\boldsymbol{J(\theta_1)}$

$\boldsymbol{\theta_1}$

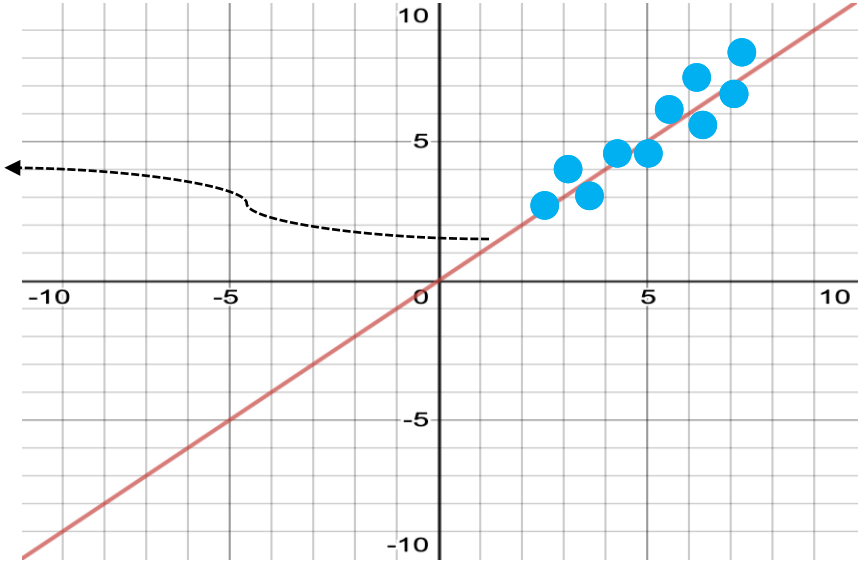
$\boldsymbol{\theta_1} = 2$

Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$

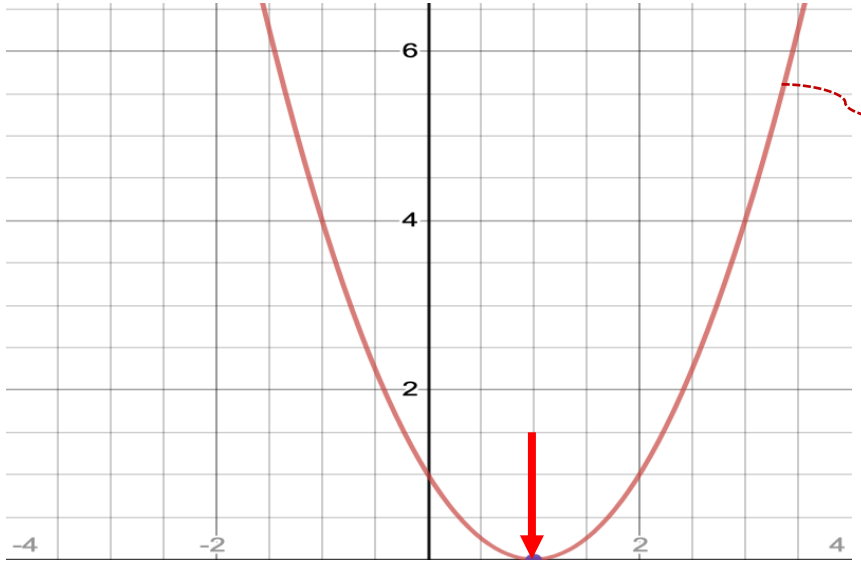
$\boldsymbol{y' = h_\theta(x)}$

$\boldsymbol{y' = \theta_1 x}$
$\boldsymbol{= 1x}$



$\boldsymbol{x}$

$\boldsymbol{J}$



$\boldsymbol{J(\theta_1)}$

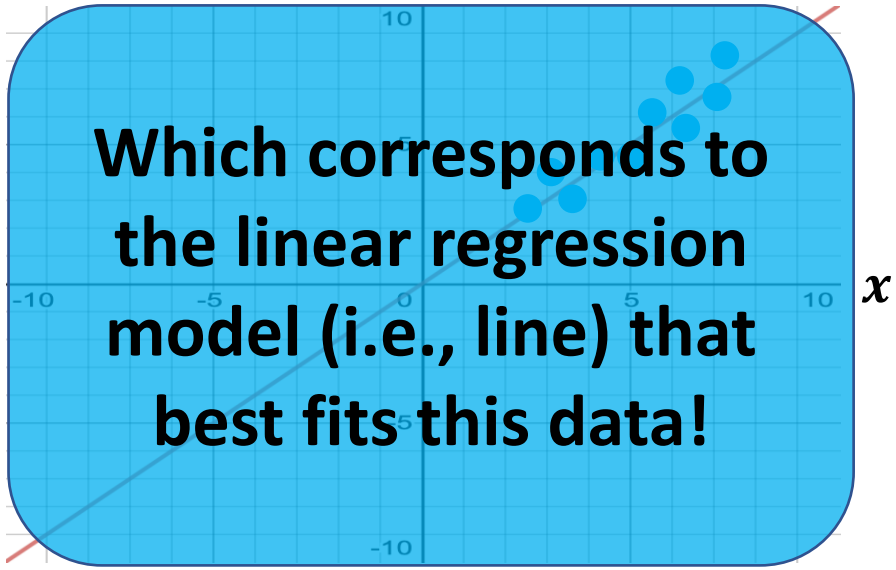
$\boldsymbol{\theta_1}$

$\boldsymbol{\theta_1} = 1$

Learning a Linear Regression Model via Minimizing a Cost Function

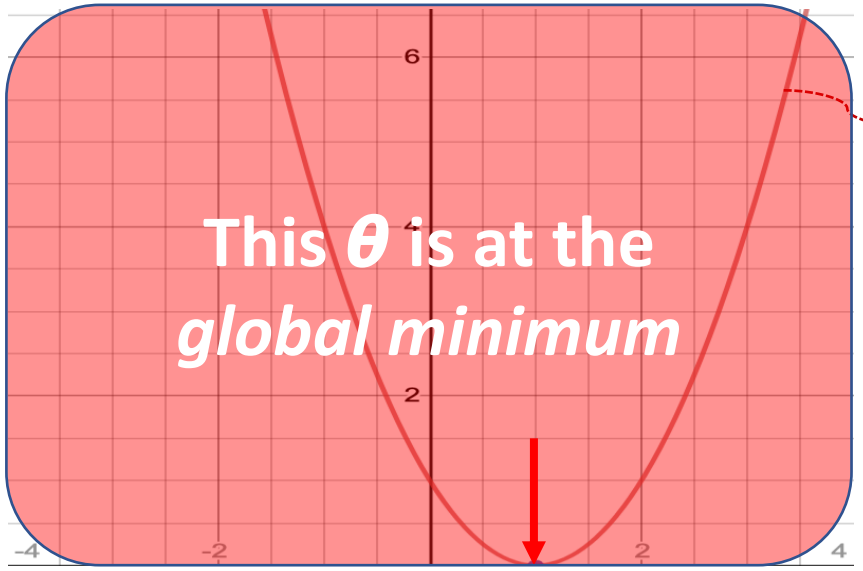
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$

$\boldsymbol{y' = h_\theta(x)}$



$\boldsymbol{x}$

$\boldsymbol{J}$



$\boldsymbol{J(\theta_1)}$

$\boldsymbol{\theta}_1$

$\boldsymbol{\theta}_1 = 1$

Gradient Descent For Linear Regression

- How to $\underset{\theta_0, \theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_0, \theta_1)}$ and locate the minimum, which corresponds to $h_\theta(x)$ that best fits the given data?

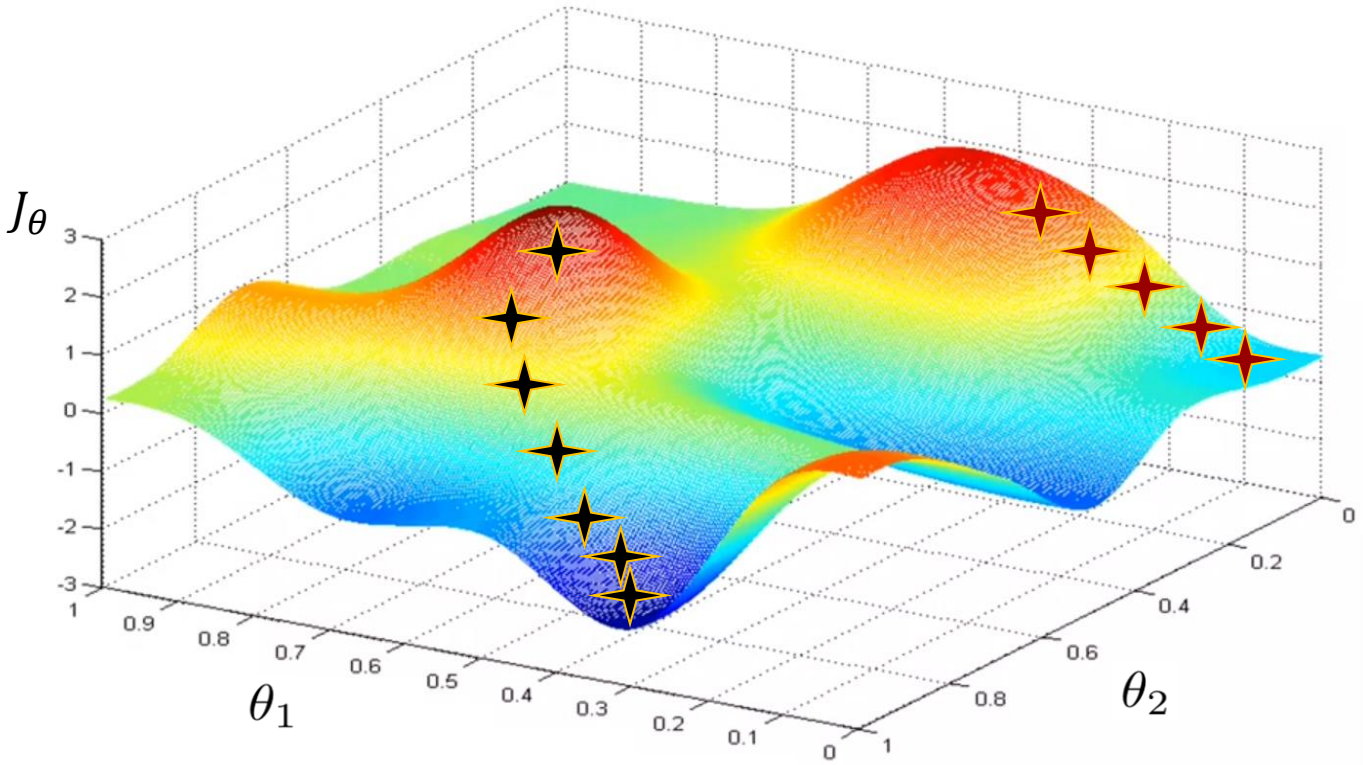
- More generally, how to $\underset{\theta_0, \dots, \theta_{n-1}}{\text{minimize}}\, \boldsymbol{J(\theta_0, \dots, \theta_{n-1})}$ and locate the minimum, which corresponds to $h_\theta(x)$ that best fits the given data?
  - By using the *gradient descent algorithm*
  - The gradient descent algorithm can be utilized also to learn many other mathematical models and not only a linear regression model
    - Image classification, speech-to-text, etc.

Gradient Descent For Linear Regression

- **Steps**:
  - Have some cost function $\boldsymbol{J(\theta_0, \dots, \theta_{n-1})}$
  - Start with some guesses for $\theta_0, \dots, \theta_{n-1}$
    - a common choice is to set them all initially to zero
    - or to random values
  - Keep changing $\theta_0, \dots, \theta_{n-1}$ to reduce $\boldsymbol{J(\theta_0, \dots, \theta_{n-1})}$ until we hopefully end up at a minimum location
    - When you are at a certain position on the surface of $\boldsymbol{J}$, look around, then take a little step in the direction of *the steepest descent*, then repeat

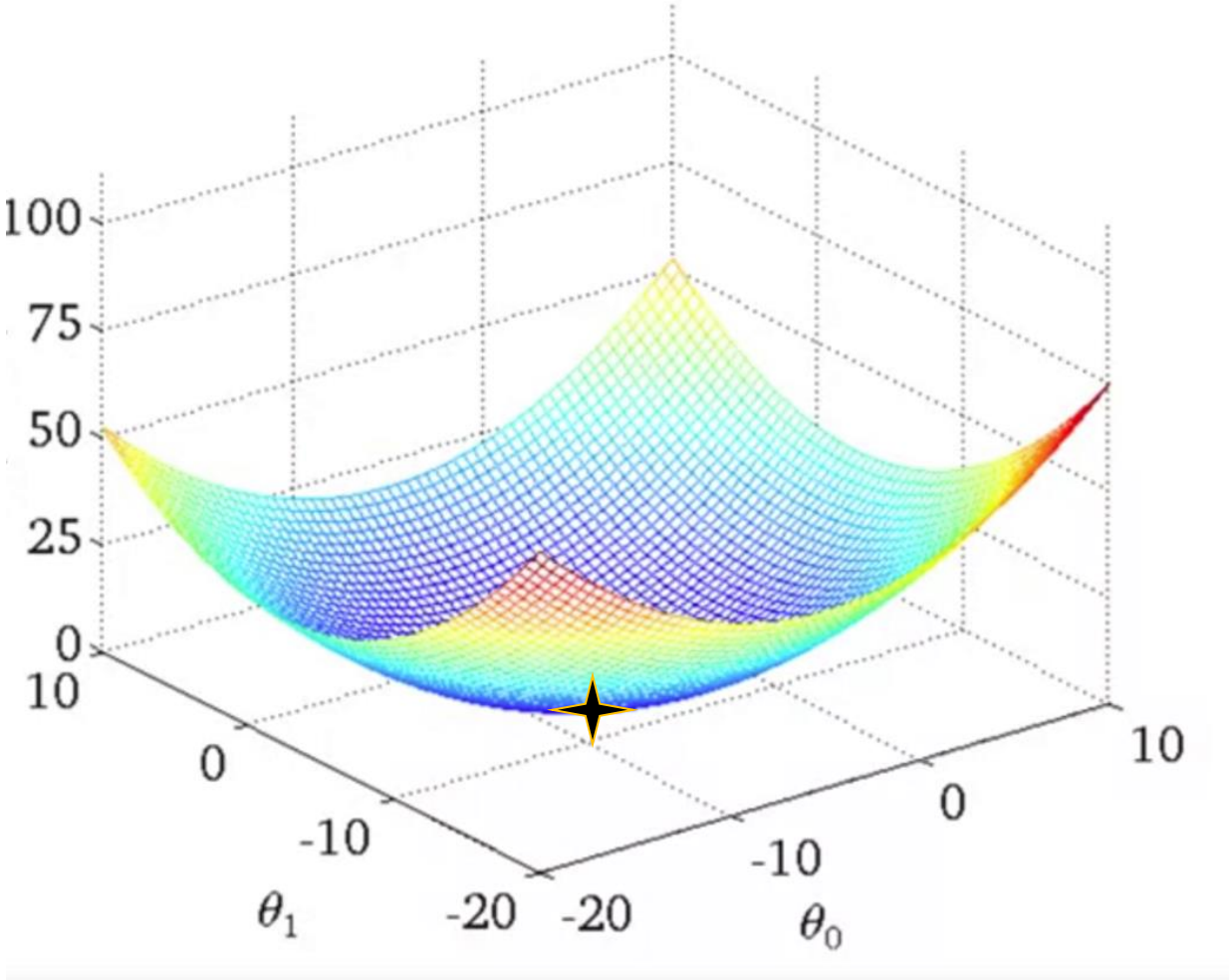
Gradient Descent

Coming down a mountain.



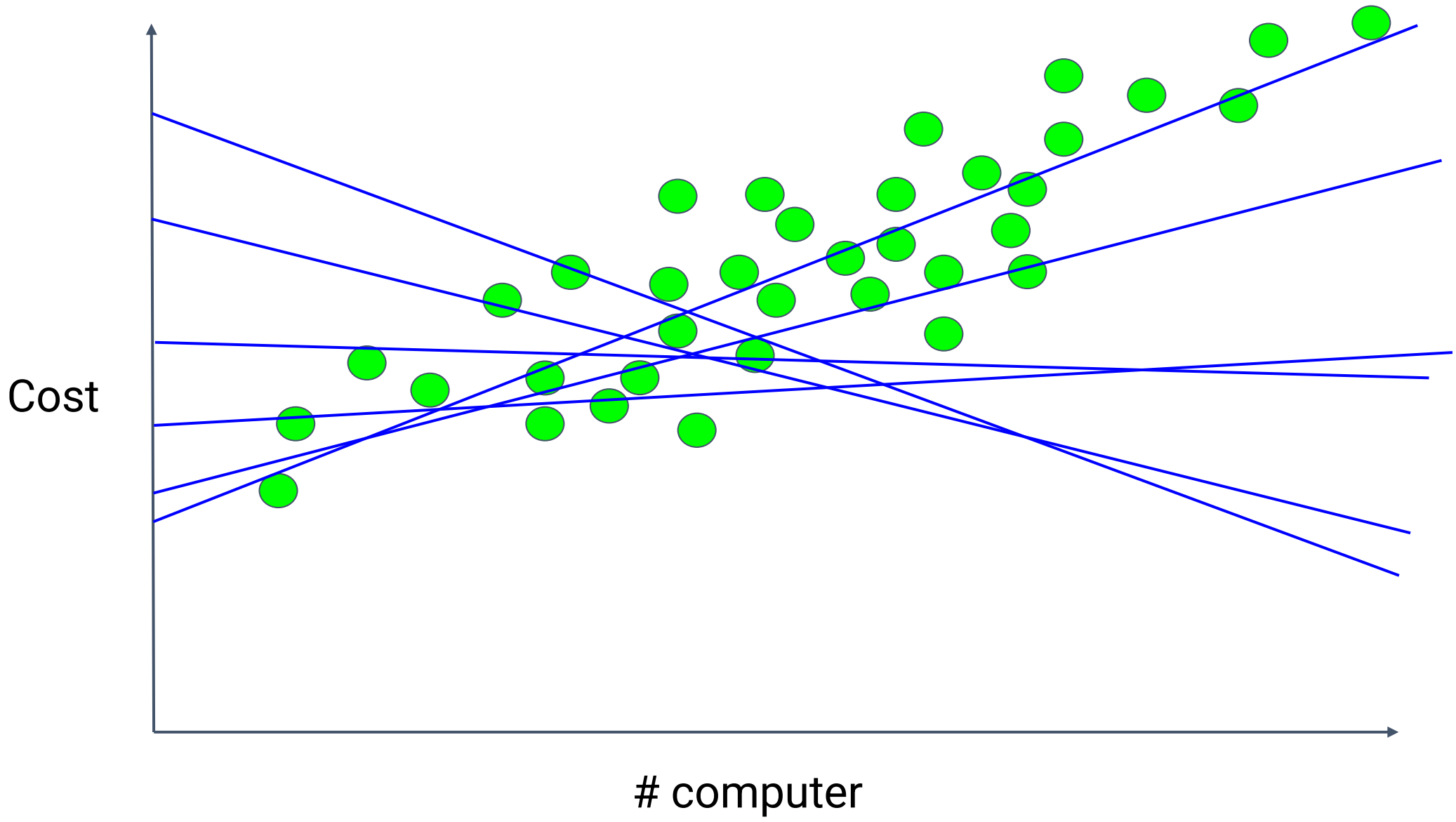
Gradient Descent

MSE is a convex function: a unique global minimum.



https://i.stack.imgur.com/cYAx7.png

Gradient Descent



Gradient Descent For Linear Regression

- **Steps**:
  - Have some cost function $\boldsymbol{J(\theta_0, \dots, \theta_{n-1})}$
  - Start with some guesses for $\theta_0, \dots, \theta_{n-1}$
    - a common choice is to set them all initially to zero or random values
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \frac{\partial \boldsymbol{J(\theta_0, \dots, \theta_{n-1})}}{\partial \theta_j}$$

}

Partial derivative

Learning Rate

Gradient Descent For Linear Regression

- **Steps**
  - Have some cost function $\boldsymbol{J(\theta_0, \theta_1)}$
  - Start with some guesses for θ_0, θ_1
    - a common choice is to set them both initially to zero
  - Repeat until convergence{

$$temp_0 = \theta_0 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_0} \qquad \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right)$$

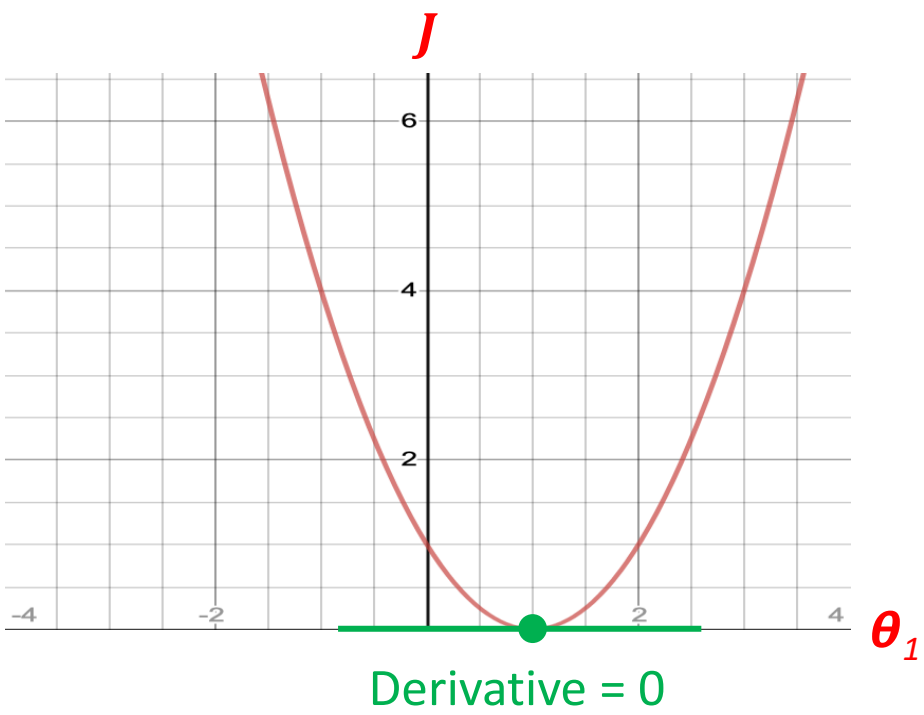
$$temp_1 = \theta_1 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_1} \qquad \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right) \; .x^{(i)}$$

$$\theta_0 = temp_0$$
$$\theta_1 = temp_1$$

}

The Impact of Partial Derviative

- For simplicity, let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$



$$\theta_1 = \theta_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{J(\theta_1)}}{\boldsymbol{d}\, \theta_j}$$

$$= \theta_1 - \alpha\, (Zero)$$

e.g.: when θ_1 remains the same, the descent algorithm converges to a stationary point (i.e., a point where the derivative is zero)

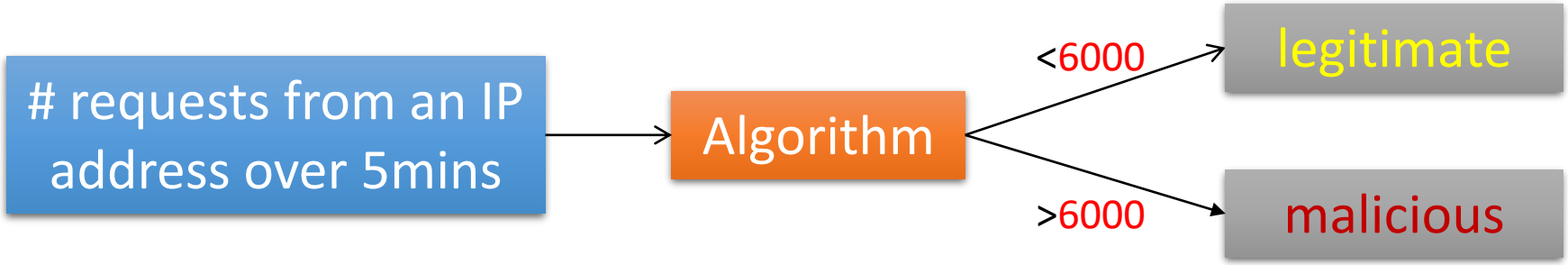
Rule-based Classification

- Suppose that you are in charge of computer security for your company
  - install firewalls, hold security awareness training, ensure secure coding, etc.
- But all your CEO cares about is you don't have a breach
- Systems that can detect and block malicious traffic to any attack surface
  - For every file sent through the network, does it contain malware?
  - For every login attempt, has someone's password been compromised?
  - For every email received, is it a phishing attempt?
  - For every request to your servers, is it a denial-of-service (DoS) attack?
  - For every outbound request from your network, is it a bot calling its command- and-control server?

Classification tasks: Classify all events in your network as malicious or legitimate

Rule-based Classification

- How to classify all traffic? **("Human learning" not machine learning)**
  - historical logs of binary files, login attempts
  - emails received, and inbound and outbound requests, etc.
- Look for patterns in the past data indicating malicious attacks
  - single IP address making 20 requests/sec over 5 mins => possible DoS attack
- Encode these patterns as an algorithm
  - a model that takes as input data, outputs or 'legitimate' or 'malicious'
  - In this example, the algorithm would be very simple:

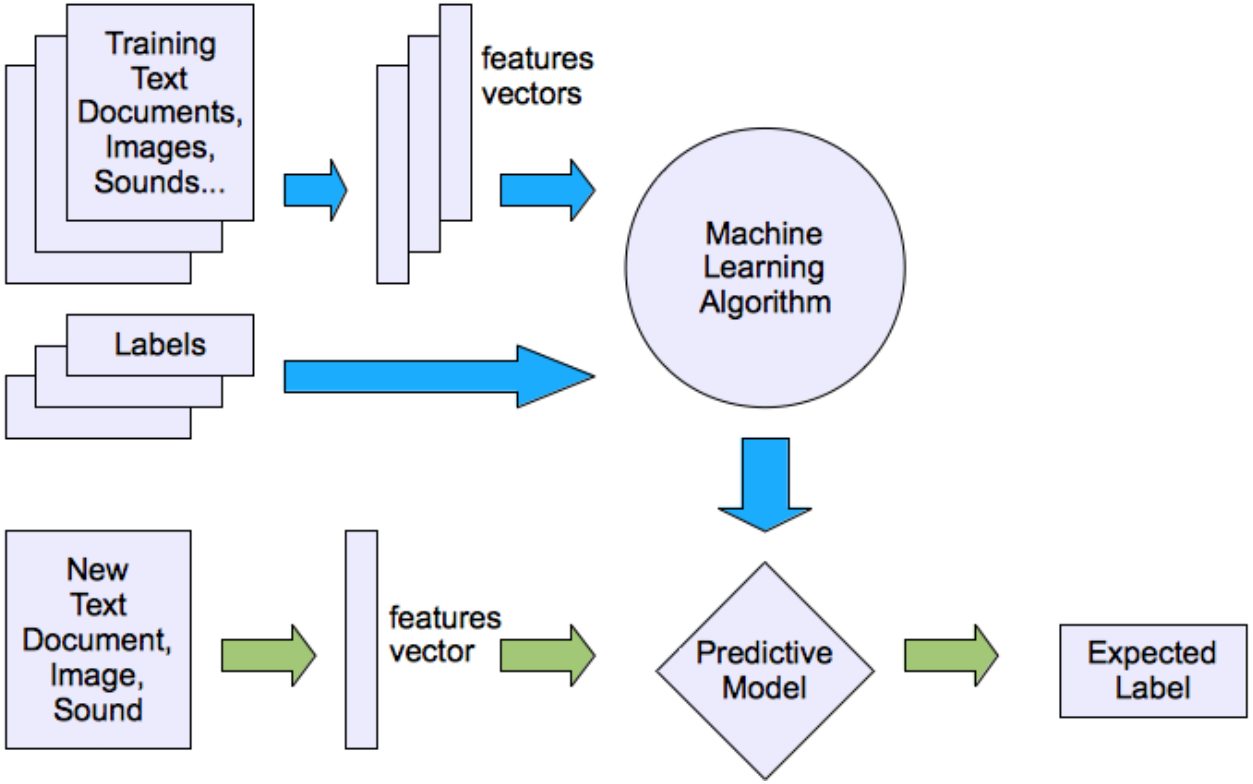


Machine Learning: problems and approaches

- **Machine learning:** use historical data to create a prediction algorithm for future data
  - **Classification**: determine which class a new data point falls into
    - binary. e.g. legitimate or malicious
    - multiclass. e.g. ransomware / a keylogger / remote access trojan
  - **Regression**: predict the value of a real-number variable
    - e.g. predict # phishing emails an employee receives in a given month
  - **Clustering**: given data points, which ones are similar to one another?
    - e.g. internet traffic to your site, botnets / mobile providers / legitimate users

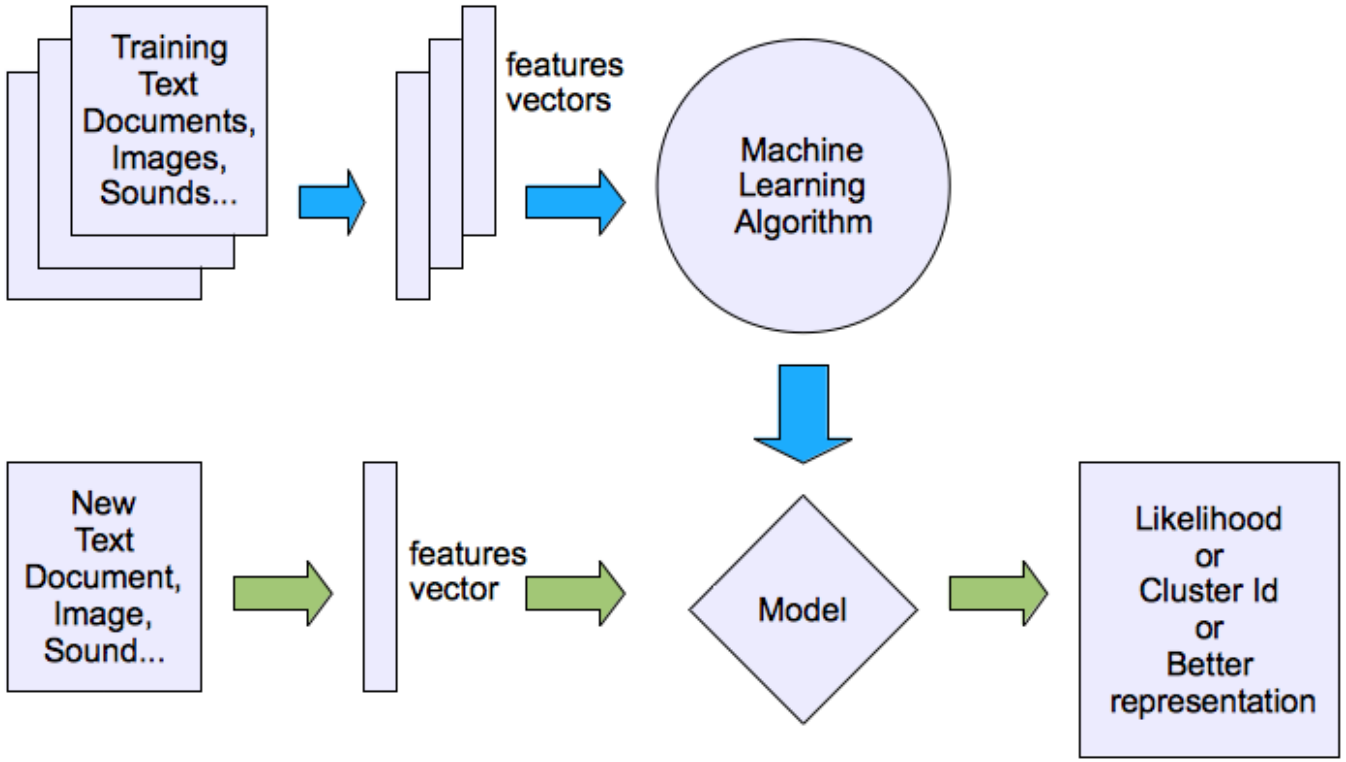
Machine learning structure

- **Supervised learning**
  - given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
  - e.g., given emails labeled as spam or ham, train a spam classifier, predict whether a new incoming message is spam



Machine learning structure

- **Unsupervised learning**
  - given a set of feature vectors (no labels), group them into clusters
  - e.g. unknown botnets attacking your network, disambiguate from one another



ML in Practice: Example

- **Goal**: train a model to learn how to identify a fraudulent transaction from the dataset
- Dataset: transaction data for online purchases
  - 39,221 transactions
    - 5 properties (***features***)
    - and a binary ***label*** as either: 1(fraudulent) or 0(normal)
  - CSV file, first row: the names for each positional value in each subsequent line

```
accountAgeDays,numItems,localTime,paymentMethod,paymentMethodAgeDays,label
...
196, 1, 4.962055, creditcard, 5.10625, 0
```

ML in Practice: Example

Create a prototype system by using machine learning

- The fraud detection system
  - take in ***features*** of a transaction
  - return a ***probability*** indicating how likely to be fraudulent

1. Prepare the data

- read the dataset in the CSV file: `df = pd.read_csv('ch1/payment_fraud.csv')`
- three random rows from df `df.sample(3)`

	accountAgeDays	numItems	localTime	paymentMethod	paymentMethodAgeDays	label
31442	2000	1	4.748314	storecredit	0.000000	0
27232	1	1	4.886641	storecredit	0.000000	1
8687	878	1	4.921349	paypal	0.000000	0

numerical index

categorical variable

ML in Practice: Example

- convert variables from ***categorical*** to ***numeric***

	accountAgeDays	...	paymentMethod_creditcard	paymentMethod_paypal	paymentMethod_storecredit
23393	57	...	1	0	0
3355	1,366	...	0	1	0
34248	19	...	1	0	0

- **one-hot encoding**
  - each feature is a binary feature (either 0 or 1)
  - each row has exactly one of these features set to 1

- Why do we use one-hot instead of simply assigning 1, 2, and 3?

ML in Practice: Example

- convert variables from ***categorical*** to ***numeric***

	accountAgeDays	…	paymentMethod_creditcard	paymentMethod_paypal	paymentMethod_storecredit
23393	57	…	1	0	0
3355	1,366	…	0	1	0
34248	19	…	1	0	0

- Why do we use one-hot instead of simply assigning 1, 2, and 3?
  - **Numerical values implicitly assume mathematical relationships: larger, smaller or equal.**
    - **They are meaningless.**
  - **Distance to each other is the same for every pair of categories.**

ML in Practice: Example

- divide the dataset into **training** and **test** sets randomly

```
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('label', axis=1), df['label'],
    test_size=0.33, random_state=17)
```

  - Split into X_train and X_test to the ratio of 0.67:0.33
  - The labels will then be split in the same ratio into y_train and y_test

2. Apply a supervised learning algorithm to this data:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

- This classifier takes the training data and uses logistic regression to distill some generalizations about fraudulent/nonfraudulent transactions into a ***model***

ML in Practice: Example

3. Make predictions using this model

```
y_pred = clf.predict(X_test)
```

- at training time the classifier did **not** have any access to y_test at all
- y_pred are a result of the generalizations learned from the training set

4. Evaluate these predictions

```
print(confusion_matrix(y_test, y_pred))
```

0	Predicted NOT FRAUD	Predicted FRAUD
Actual NOT FRAUD	12,753	0
Actual FRAUD	1	189

- 1 misclassification in the test set
  - 1 fraudulent transaction that was not detected
- 189 transactions are correctly flagged as fraud
- 12,753 transactions are correctly identified as not fraud

ML in Practice: Example

- **Model usage**
  - Given any incoming transaction, we can apply this model and get a probability score for how likely this transaction is to be fraudulent

```
clf.predict_proba(df_real)

  # Array that represents the probability of the transaction
  # having a label of 0 (in position 0) or 1 (in position 1)
> [[  9.99999994e-01   5.87025707e-09]]
```

- The part where we learn the prediction algorithm has been abstracted into the scikit-learn API call `LogisticRegression.fit()`
  - now open the box and find out what goes on

Logistic regression

- **Logistic regression** is a supervised learning algorithm
  - can be used to classify input data
  - e.g., classify transactions into fraud or normal

- Major questions about logistic regression:
  - What is the ***model*****?**
  - What is the ***cost function***?
  - How can we learn the ***parameters*** of the model?

Logistic regression

- The Logistic Regression Model

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

  - $\boldsymbol{x}$ is the ***feature vector***: $\boldsymbol{x} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_m}]^T$
  - $\boldsymbol{\theta}$ is the ***parameter vector***: $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_m}]^T$
  - $\boldsymbol{g(z)}$ is the sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$

- ***After*** learning the model, we can then apply thresholding to predict the binary output as follows:

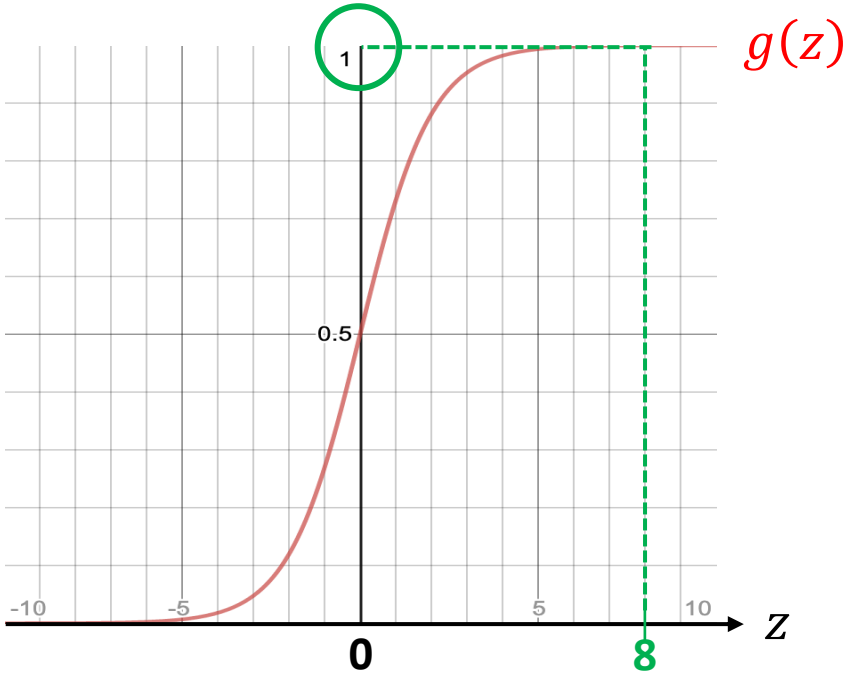
$$if\ h_\theta(x) < 0.5 \text{ predict } 0$$
$$if\ h_\theta(x) \geq 0.5 \text{ predict } 1$$

Logistic regression

Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function** $g(z) = \frac{1}{1+e^{-z}}$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$z \in \mathbb{R}, but$
$g(z) \in [0,1]$



Assume a labeled example ($\boldsymbol{x}$, y):

If y = **1**, we want $g(z) \approx 1$ (i.e., we want a correct prediction)

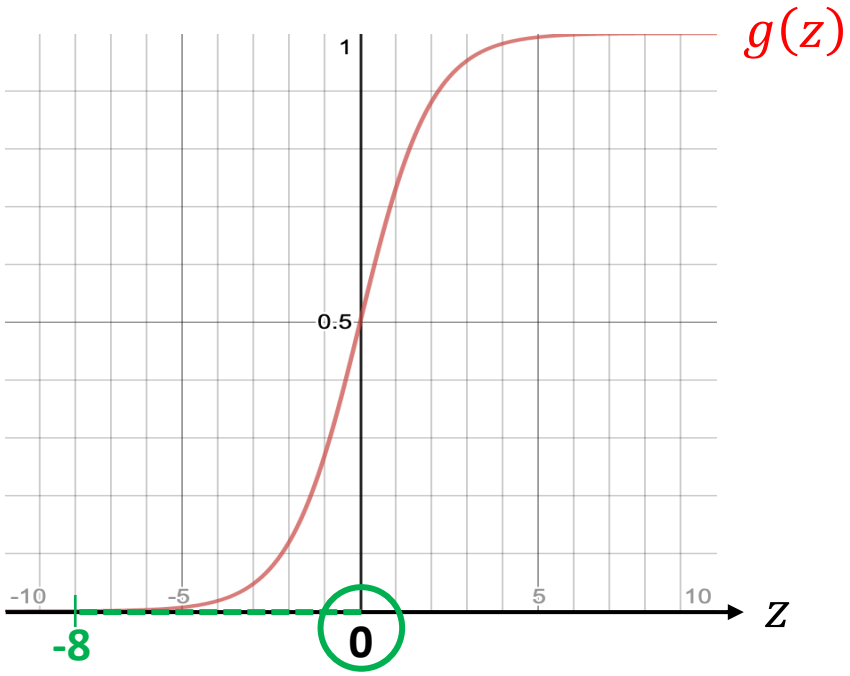
For this to happen, $\boldsymbol{z} \gg \mathbf{0}$

Logistic regression

Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function** $g(z) = \frac{1}{1+e^{-z}}$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$z \in \mathbb{R}, but$
$g(z) \in [0,1]$



Assume a labeled example ($\boldsymbol{x}$, y):

If y = **0**, we want $g(z) \approx 0$ (i.e., we want a correct prediction)

For this to happen, $\boldsymbol{z} \ll \mathbf{0}$

Logistic regression: Calculating a Probability

- Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function**

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

- Suppose we had a logistic regression model with three features that learned the following weights:

$$\boldsymbol{\theta} = [1, 2, -1, 5]^T$$

- Further suppose the following feature values for a given example:

$$\boldsymbol{x} = [1, 0, 10, 2]^T$$

- Then the logistic regression prediction for this example will be

$$\boldsymbol{h_\theta}(\boldsymbol{x}) \ = \ ?$$

Cost function

- Now that we have restricted our choice of prediction algorithms to logistic regression, we need to choose the best one for the given training data

- How do we know when we have found the best algorithm
  - **cost function**
  - maps a set of pairs of (predicted label, truth label) to a real number

- The goal is to find the model parameters that produce predicted labels for the training set that minimize the cost function

The logistic regression cost function

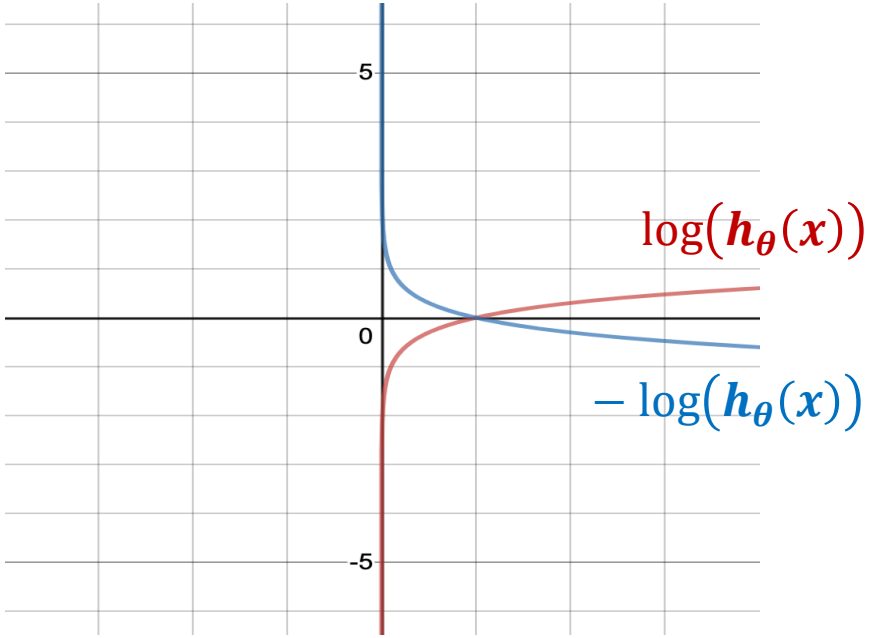
- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

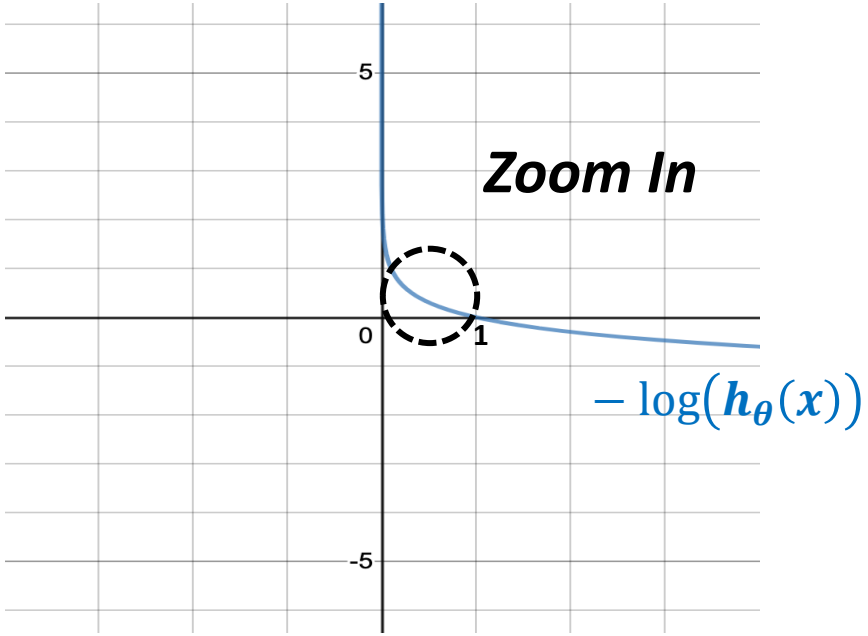
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

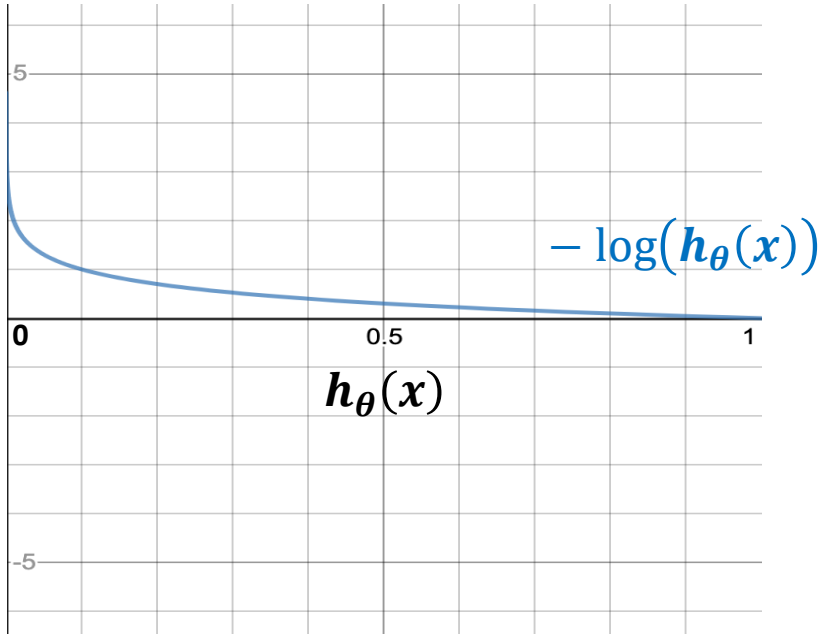
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if \; y = 1} \\ -\log\big(\boldsymbol{1 - \; h_\theta(x)}\big) \; \boldsymbol{if \; y = 0} \end{cases}$$

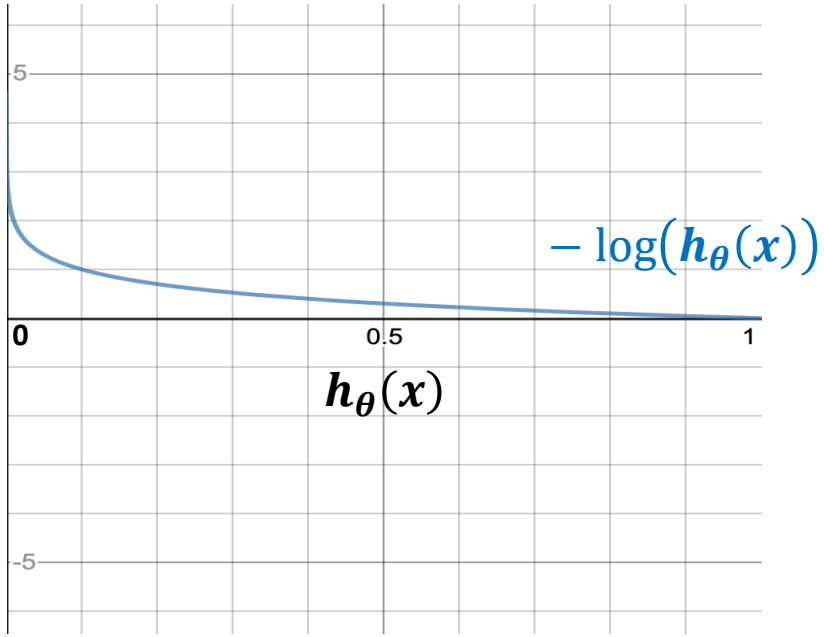


$$\boldsymbol{If \; y = 1} \begin{cases} \text{As } \boldsymbol{h_\theta(x)} \rightarrow 0, \; -\log\big(\boldsymbol{h_\theta(x)}\big) \rightarrow \infty \\ \text{As } \boldsymbol{h_\theta(x)} \rightarrow 1, \; -\log\big(\boldsymbol{h_\theta(x)}\big) \rightarrow 0 \text{ (i.e., cost} \rightarrow 0) \end{cases}$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$

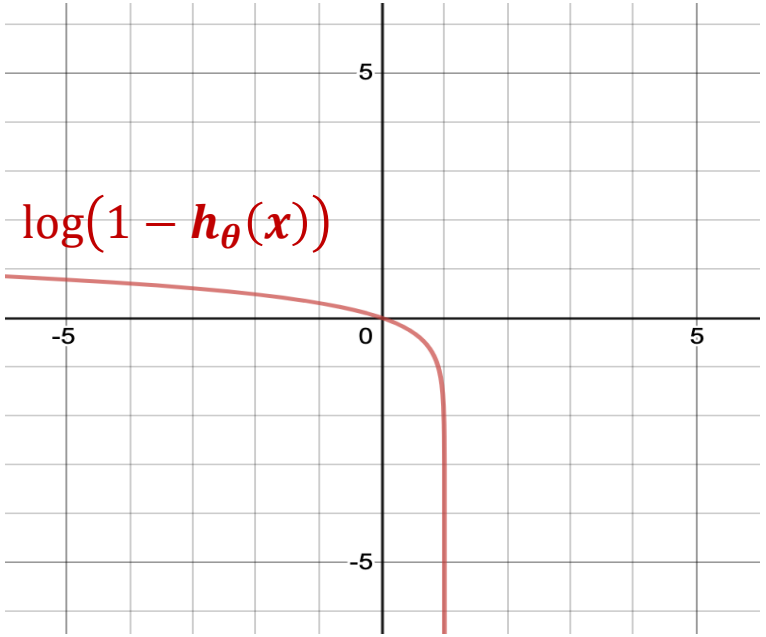


This captures the intuition that if $\boldsymbol{h_\theta(x) = 0}$ (i.e., what we will predict is $\mathbf{0}$), but $\boldsymbol{y = 1}$ (hence, we are mispredicting!), we shall penalize the learning algorithm by a very large cost!

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

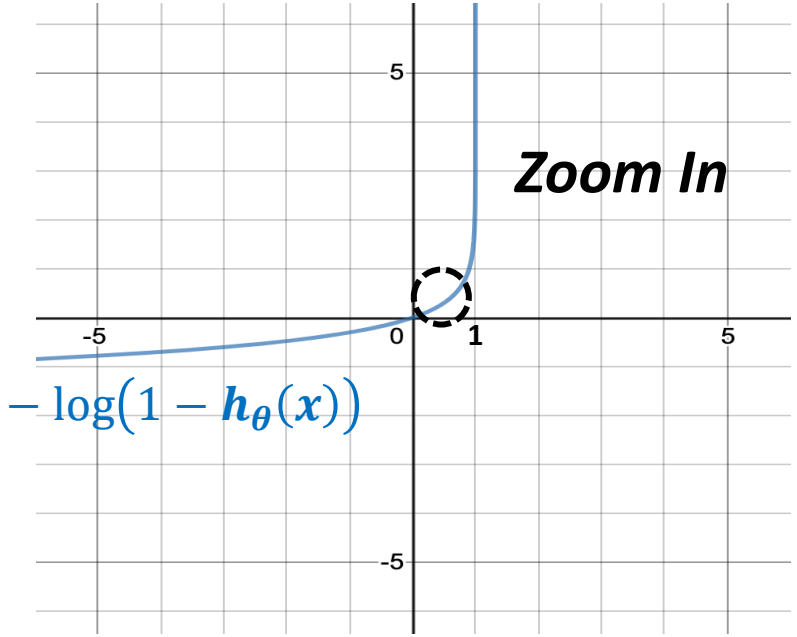
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

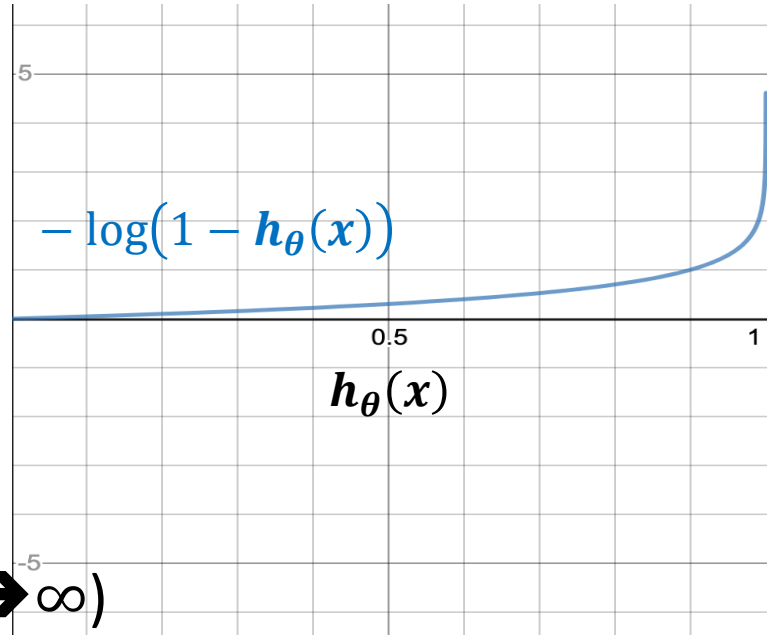
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if \; y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if \; y = 0} \end{cases}$$

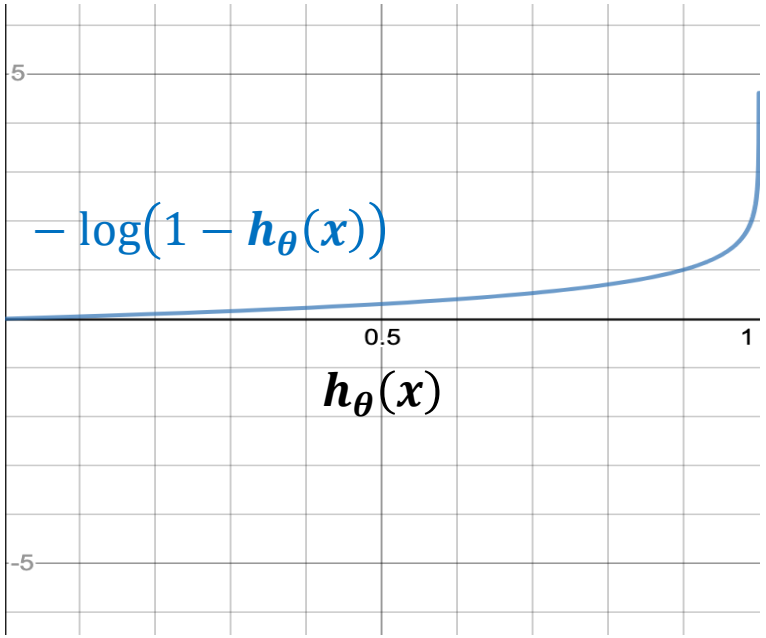


$$\boldsymbol{If \; y = 0} \begin{cases} \text{As } \boldsymbol{h_\theta(x)} \rightarrow 0, \; -\log\big(1 - \boldsymbol{h_\theta(x)}\big) \rightarrow 0 \\ \text{As } \boldsymbol{h_\theta(x)} \rightarrow 1, \; -\log\big(1 - \boldsymbol{h_\theta(x)}\big) \rightarrow \infty \text{ (i.e., cost} \rightarrow \infty) \end{cases}$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$



This captures the intuition that if $\boldsymbol{h_\theta(x) = 1}$ (i.e., what we will predict is **1**), but $\boldsymbol{y = 0}$ (i.e., we are mispredicting!), we shall penalize the learning algorithm by a very large cost!

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if\ y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if\ y = 0} \end{cases}$$

Equivalent To

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y\log\big(\boldsymbol{h_\theta(x)}\big) - (1-y)\log\big(\boldsymbol{1 - h_\theta(x)}\big)$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x)} = \boldsymbol{g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y\log\big(\boldsymbol{h_\theta(x)}\big) - (1-y)\log\big(\mathbf{1} - \boldsymbol{h_\theta(x)}\big)$$

$$= -y\log\big(\boldsymbol{g(\theta^T x)}\big) - (1-y)\log\big(1 - \boldsymbol{g(\theta^T x)}\big)$$

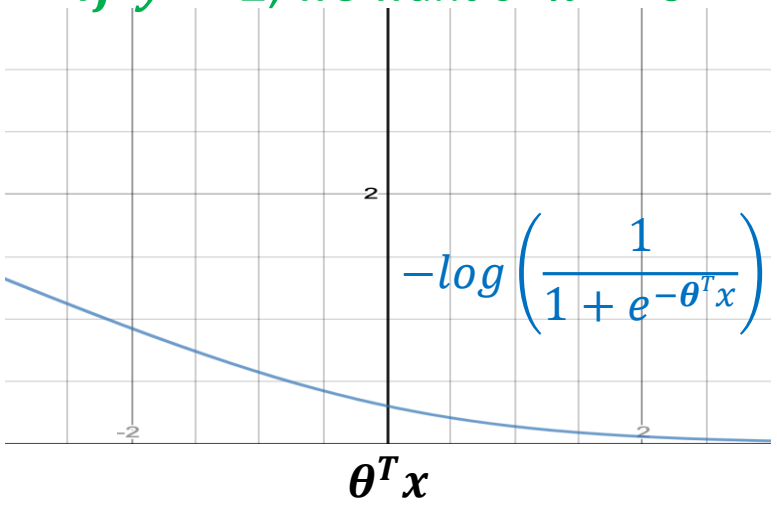
$$= -y\log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

The logistic regression cost function

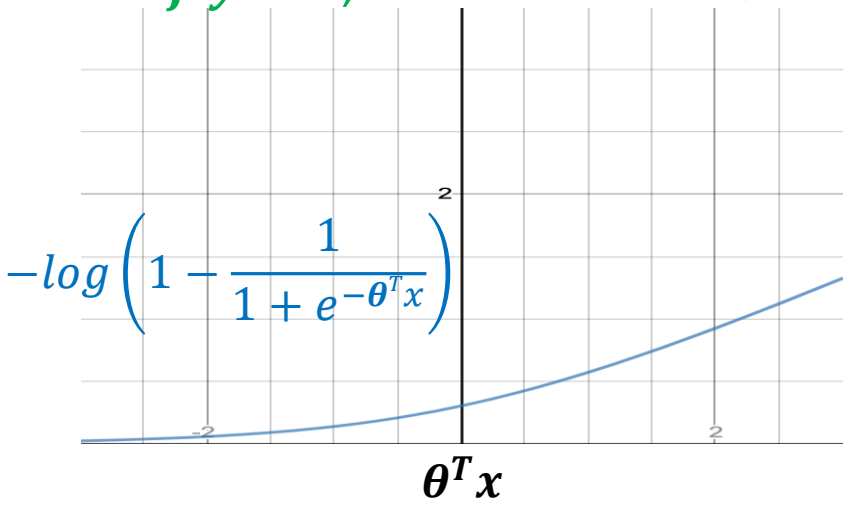
- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

If $\boldsymbol{y = 1}$, we want $\boldsymbol{\theta^T x >> 0}$



If $\boldsymbol{y = 0}$, we want $\boldsymbol{\theta^T x << 0}$



Learning a logistic regression model

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x)} = \boldsymbol{g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

- That is:

$$\underset{\theta}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n} \text{Cost}(\boldsymbol{h_\theta(x)}^{(i)}, y^{(i)})$$

$$\equiv$$

$$\underset{\theta}{\text{minimize}} \boxed{\frac{1}{n}\sum_{i=1}^{n} -y^{(i)} \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right) - (1-y^{(i)})\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right)}$$

Cost function $\boldsymbol{J(\theta)}$

Gradient descent for logistic regression

- **Outline**:
  - Have cost function $\boldsymbol{J}(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$
  - Start with some guesses for $\theta_0, \dots, \theta_m$
    - a common choice is to set them all initially to zero or random values
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial \theta_j}$$

}

***Learing rate**, which controls how big a step to take when update θ_j*

Gradient descent for logistic regression

- **Outline**:
  - Have cost function $\boldsymbol{J(\theta)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$
  - Start with some guesses for $\theta_0, \dots, \theta_m$
    - a common choice is to set them all initially to zero
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \boxed{\sum_{i=1}^{n} \left(\frac{1}{1 + e^{-\boldsymbol{\theta}^T x^{(i)}}} - y^{(i)} \right) x_j{}^{(i)}}$$

The final formula after applying partial derivatives

}

Inference after learning

- After learning the parameters $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$, we can predict the output of any new unseen $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^{\boldsymbol{T}}$ as follows:

$$\begin{cases} \boldsymbol{if\ h_\theta(x)} = \dfrac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} < \mathbf{0.5\ predict\ 0} \\ \\ \boldsymbol{Else\ if\ h_\theta(x)} = \dfrac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} \geq \mathbf{0.5\ predict\ 1} \end{cases}$$

How to evaluate models?

- Accuracy
- Different types of errors
- Precision and Recall
- F-score

Two example datasets

- We use the metrics to evaluate several models on two toy datasets
  - a medical dataset of patients, some are diagnosed with coronavirus
    - **1000** patients: **10** ‘sick’ and **990** ‘healthy’
    - The goal of a model: predict the diagnosis based on the features of each patient
  - a dataset of emails that have been labeled as spam or not spam(ham)
    - **100** emails: **40** ‘spam’ and **60** ‘ham’
    - The goal of a model: predict the label based on the features of the email
- Note that we are not building models
- Instead, we use the models as black boxes and **evaluate** them
  - how many of the data points they predict correctly or incorrectly

Accuracy

- **Accuracy**: the percentage of times that a model is correct

$$\frac{\textit{number of correctly predicted data points}}{\textit{total number of data points}}$$

- Example
  - if we evaluate a model on a test dataset of 1,000 samples, and the model predicted the correct label of the samples 900 times, then this model has an accuracy of 900/1000 = 90%

- Be careful of "Accuracy"
  - sometimes accuracy doesn't fully describe the performance of the model

Accuracy

- This measure is dominated by the larger set (of positives or negatives) and may favor trivial classifiers
- Can you think of a completely useless model that predicts coronavirus in our dataset, yet is correct 99% of the time?
  - Recall that the dataset has **1000** patients: **10** ‘sick’ and **990** ‘healthy’
- Simply diagnose every patient as ‘healthy’
  - out of 1,000 tries, it’s incorrect 10 times and correct 990 times
- Errors are not created equal, and some mistakes are much more expensive than others

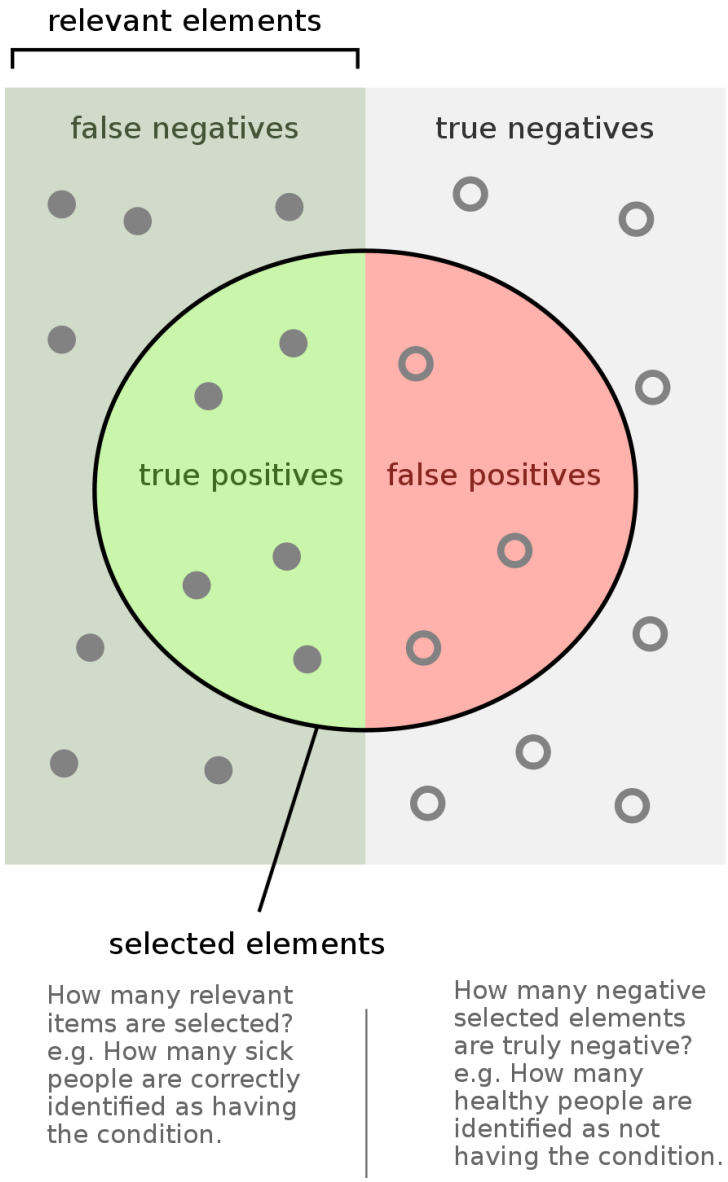
Different types of errors

*In our model, we label the **sick** patients **as positive***

- What are the two types of errors the coronavirus model can make?
  - **False positive**: a healthy person who is **incorrectly** diagnosed as sick
  - **False negative**: a sick person who is **incorrectly** diagnosed as healthy
- Cases that are **correctly** diagnosed also have names
  - **True positive**: a sick person who is diagnosed as sick
  - **True negative**: a healthy person who is diagnosed as healthy
- Question: If consider the **positives** to be the **spam** emails
  - False positive? a ham email that is incorrectly classified as spam
  - False negative? a spam email that is incorrectly classified as ham
  - True positive? a spam email that is correctly classified as spam
  - True negative? a ham email that is correctly classified as ham

Different types of errors

- a true positive is a data point with a positive label that is correctly classified as positive
- a true negative is one with a negative label that is correctly classified as negative
- a false positive is one with a negative label, but the model falsely classifies it as positive
- a false negative is one with a positive label, but the model falsely classified it as negative



Confusion matrix

- A specific table layout to visualize the performance
  - rows represent the label
  - columns represent the prediction
  - elements in the diagonal are classified correctly
  - elements off the diagonal are classified incorrectly

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

- Question:
  - Accuracy = ?

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Confusion matrix

- Let's consider the 'useless' model
  - For dataset with **1000** patients: **10** 'sick' and **990** 'healthy'
  - Simply diagnose every patient as 'healthy'
  - The confusion matrix:

Coronavirus model 1	Diagnosed sick (predicted positive)	Diagnosed healthy (predicted negative)
Sick (positive)	0 (number of true positives)	10 (number of false negatives)
Healthy (negative)	0 (number of false positives)	990 (number of true negatives)

- The model creates too many false negatives

Recall

- **Recall**: TP/(TP + FN)
  - the proportion of correct predictions among the data points with positive labels

Coronavirus model 1	Diagnosed sick (predicted positive)	Diagnosed healthy (predicted negative)
Sick (positive)	0 (number of true positives)	10 (number of false negatives)
Healthy (negative)	0 (number of false positives)	990 (number of true negatives)

- Accuracy = 99%
- Recall = ?

Precision

- **Precision**: TP/(TP + FP)
  - the proportion of true positives among data points that have been predicted as positive
- spam model 1:
  - precision=85.7%
  - accuracy = 85%

Spam model 1	Predicted spam	Predicted ham
Spam	30 (true positives)	10 (false negatives)
Ham	5 (false positives)	55 (true negatives)

- Spam model 2:
  - precision=77.7%
  - accuracy=85%

Spam model 2	Predicted spam	Predicted ham
Spam	35 (true positives)	5 (false negatives)
Ham	10 (false positives)	50 (true negatives)

F-score

- A measure that naturally combines ***P***recision and ***R***ecall is the F-score:

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

- Setting $\beta = 1$ gives us the ***F_1 – score***. It can also be computed as:

$$F_1 = \frac{2PR}{P+R}$$

  - The F1 score gives equal weight to both measures
  - If we want to create a classification model with a good balance of precision and recall, maximize the F1 score

Practical Considerations

- In theory, applying a machine learning algorithm
  - put training data into a large matrix
  - run training procedure
  - use the resulting model to classify unseen data
- However, the task is not so simple
  - dozens of choices to be made in the model construction process
  - each choice can lead to very different outcomes in your final model
- Now consider some of the most important choices to be made during the modeling process

Selecting a Model Family

- Given a task, how do we decide which algorithm to pick?
- Let the data decide: try different approaches and see what works best
- Some factors to consider:
  - Computational complexity
    - Train the model in a reasonable amount of time
  - Explainability
    - A human might need to read the output of the model and determine why it made the decision
    - e.g. if you block a user, you should educate them as to what they did that was suspicious
- But ultimately, you should experiment to find the best choice for your data

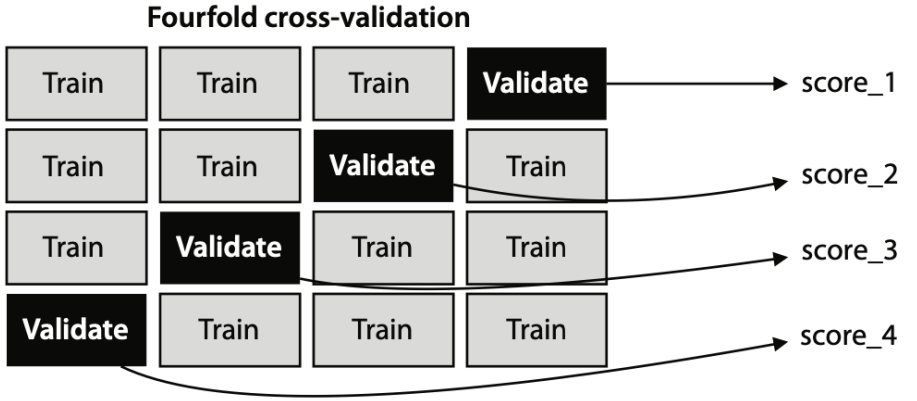
Training data construction

- In a supervised learning problem, we are given labeled examples of the data that we want to classify
  - e.g. account registrations, user logins, email messages
- Goal is to produce a model that can predict the future based on past data, need to reserve some of the labeled data to act as this “future” data so that we can **evaluate** our model
- Different ways to do this
  - Cross-validation
  - Train/validate/test
  - Out-of-time validation

Cross-validation

- K-fold cross-validation is a way to use all the data for training and testing, by recycling it several times
  - Split the labeled data into k equal portions
  - Train the model k times, using the union of k – 1 of the portions as the training set and the remaining one as a validation set
  - The final score of that model is the **average** of the validation scores

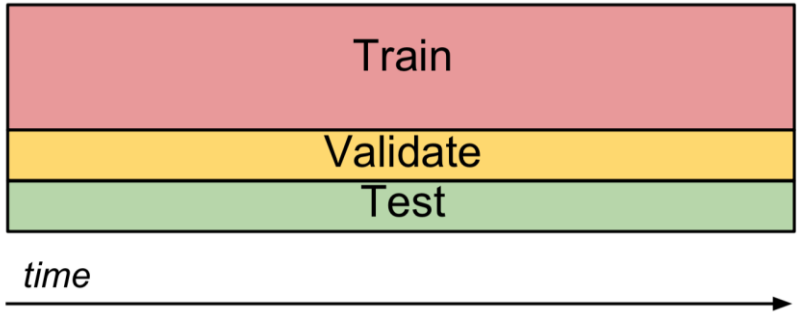
classical training-validation split



$$\text{score} = \frac{\text{score_1} + \text{score_2} + \text{score_3} + \text{score_4}}{4}$$

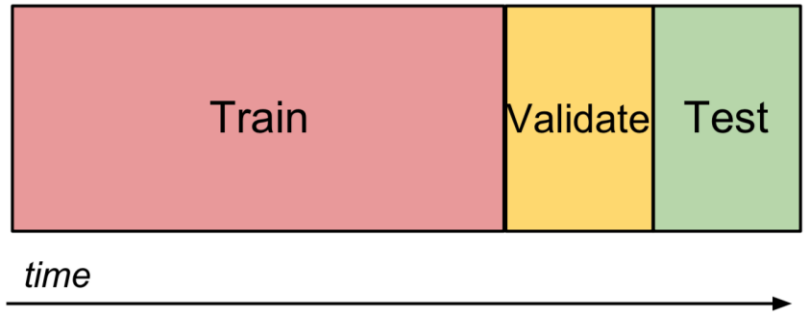
Train/validate/test

- Randomly divide the labeled data into three parts
  - The majority is the *training set:* input to the learning algorithm
  - The second part is the *validation set (a.k.a. development set):* evaluate and iterate on the model output by the learning algorithm
    - tune model parameters based on performance on the validation set
  - Finally, after get an optimal model, use the remaining part as the *test set* to estimate real-world performance on unseen data
    - Deep learning competitions normally keep the test set secret.



Out-of-time validation

- This approach addresses the problem of validating models when the data distribution changes over time
- In this case, sampling training and validation sets at random from the same labeled set is “cheating” in some sense
  - the distributions of the training and validation sets will match closely and not reflect the time aspect
- A better approach is to split the training and validation sets based on a time cutoff:



Unbalanced data

- When classify a rare event (e.g. account hijacking)—the “good” samples can be even 99.9% of the labeled data
  - In this case the “bad” samples might not have enough weight to ‘influence’ the classifier
  - model might perform poorly when trained on this highly unbalanced data
- Some options:
  - ***Oversample*** the minority class: repeat observations in your training data to make a more balanced set
    - Synthesizing minority classes: generative models.
  - ***Undersample*** the majority class: select a random subset of the majority class to produce a more balanced set
  - ***Change the cost function*** to weight performance on the minority class more heavily, so that each minority sample has more influence on the model

Missing features

- In an ideal world every event is logged perfectly, but things go wrong
  - e.g. bugs appear in the logging; some features are delayed
- As a result, some of the samples might have ***missing features***
- If the features are missing due to random failures, remove events with missing features
  - if the data with missing features is clustered around a certain event, throwing out this data will change the distribution
- Impute the value of the missing feature
  - Simplest approach: assign the missing feature the average or median value for that feature

Adaptive attacks

- In an adversarial environment the attackers will rarely give up after we deploy a new defense.
  - they will modify their methods to try to circumvent defenses, we need to respond, and so on so forth
- Thus, not only does the distribution of attacks change over time, but it changes directly in response to our actions
- e.g. to produce a model that is robust against current attacks
  - training set should weight recent data more heavily,
    - either by relying only on the past days
    - or by using some kind of decay function to downsample historical data

Summary

- ML Problems and Approaches
- Machine Learning in Practice: a working example
- Training Algorithms to Learn
- Evaluating Models
- Practical Considerations

References

- *Machine Learning , Analytics & Cyber Security the Next Level Threat Analytics, Manjunath N V*
- *Machine Learning and Security Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *Machine learning foundational course, google developers*
- *Logistic regression, Mohammad Hammoud, CMU Qatar*
- *Practical Machine Learning in Infosecurity, Clarence Chio and Anto Joseph*
- *scikit-learn: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression*
- *pandas documentation: https://pandas.pydata.org/docs/*
- *Machine Learning Core Concept, Mohammad Hammoud, CMU in Qatar*
- *Linear regression, Mohammad Hammoud, CMU in Qatar*

Spam and Phishing

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

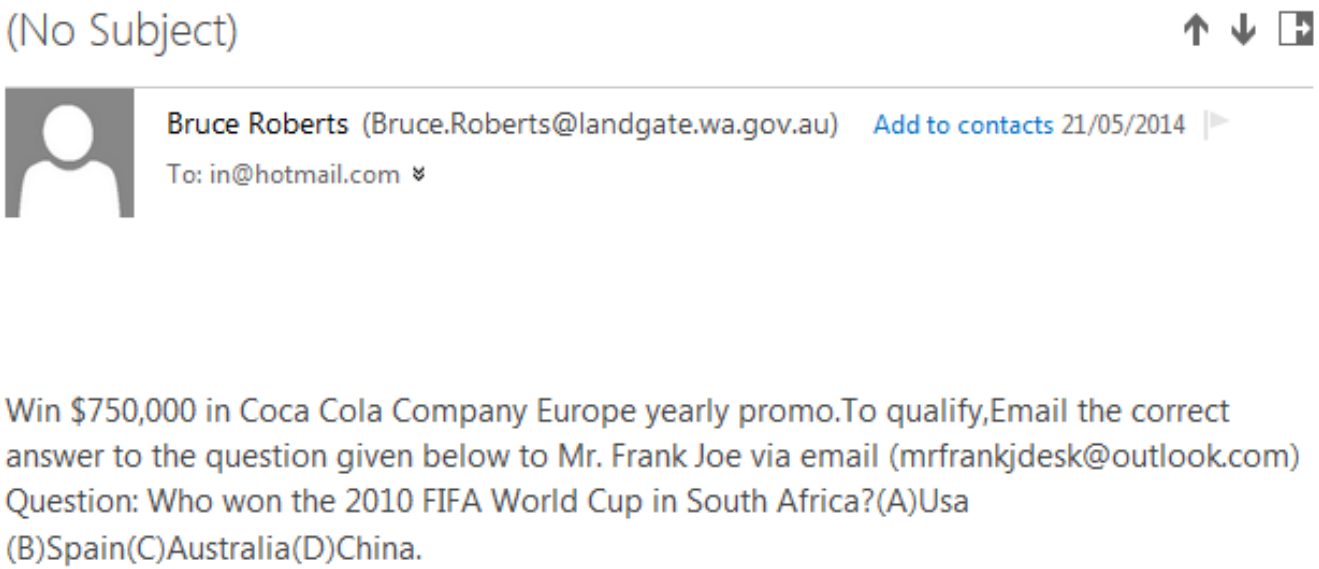
Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- What is Spam?
- Examples of Email Fraud
- Spam Filtering Techniques
- Pre-processing Text in Email Messages
- Spam detection with neural networks
  - Model representation

What is Spam

- Spam: Unsolicited bulk email or message
- Emails that involves sending identical or nearly identical messages to thousands (or millions) of recipients



What are the problems?

- With a tiny investment, a spammer can send over 100,000 bulk emails per hour
- Waste users' time, may cause financial loss
  - Study shows: an average person spends 28% of the workday (≈ 670 hrs/year) reading and responding to emails
  - only 38% of the emails on an average are relevant and important
- Junk mails waste storage and transmission bandwidth
- Might include malware as executable files
- Spam is a problem because the cost is forced onto the recipient

Statistics

- Spam accounts for 45% of all emails sent
- About 14.5 billion spam emails are sent every single day
- Spam costs $20.5 billion yearly (reduced network bandwidth, storage capacity)
- Spammers receive on average 1 click for every 12 million emails sent
  - Even with this response, spammers earn millions of dollars yearly
- 80% of all spam is sent by the same 100 spammers

Image credit: https://www.propellercrm.com/blog/email-spam-statistics

Phishing

- **Phishing:** attacks where a victim is lured to a fake web, and is deceived into disclosing personal data or credentials
- **Phishing URLs** seem like legitimate URLs, and redirect the users to phishing web pages, which mimic the look and feel of their target websites
  - ***URL (Uniform Resources Locator)*** is a web address that specifies the location of the webpage on a computer network
  - A typical URL http://www.example.com/index.html consists of several components:
    - Protocol type = http
    - Domain name = www.example.com
    - File name = index.html

Phishing

- Phishing emails are a more serious threat than spam emails, because they aim to steal users' private information, such as bank accounts, passwords, SSNs

Dropbox Phishing

- Attackers create fake sign-in pages for Dropbox as a part of credential harvesting
- They then use the stolen credentials to log in to legitimate sites and steal user data

image credit: mailguard

CEO Fraud

- **CEO fraud**: (also called whaling attack)
  - target: top executives of an organization
  - suffer from account takeovers due to stolen login credentials
- The scammer takes over CEO Account
- CEO’s credentials are used to login
- Scam emails sent to all

HOW CEO FRAUD IMPACTS YOU

THE START

Attackers see if they can spoof your domain and impersonate the CEO (or other important people)

Bad guys often troll companies for months to gather the data necessary in pulling off a successful attack

THE PHISH

Spoofed emails are sent to high-risk employees in the organization

To: Finance Department

Urgent wire transfer request! Please send $100,000 to new acct #987654-3210

To: CFO

Please pay this time-sensitive invoice. I'm on vacation and will be unavailable, no need to respond. - Your CEO

To: Human Resources

I need 5x $1000 physical Amazon gift cards. Please scratch them off and send me a clear picture of the barcodes.

THE RESPONSE

Target receives email and acts without reflection or questioning the source

I better get this payment to the new account!

It's from the CEO, I'll take care of this for him!

Sounds important, I'll pick those up on my lunch break!

THE DAMAGE

Social engineering was successful, giving hackers access to what they were after

Causing fraudulent wire transfers and massive data breaches

THE RESULT

The fallout after a successful attack can be highly damaging for both the company and its employees

Resulting damage:

- Money is gone forever in most cases and only recovered 4% of the time
- CEO is fired
- CFO is fired
- Lawsuits are filed
- Intangibles - tarnished reputation, loss of trust, etc.
- Stock value drops

So... Think Before You Click!

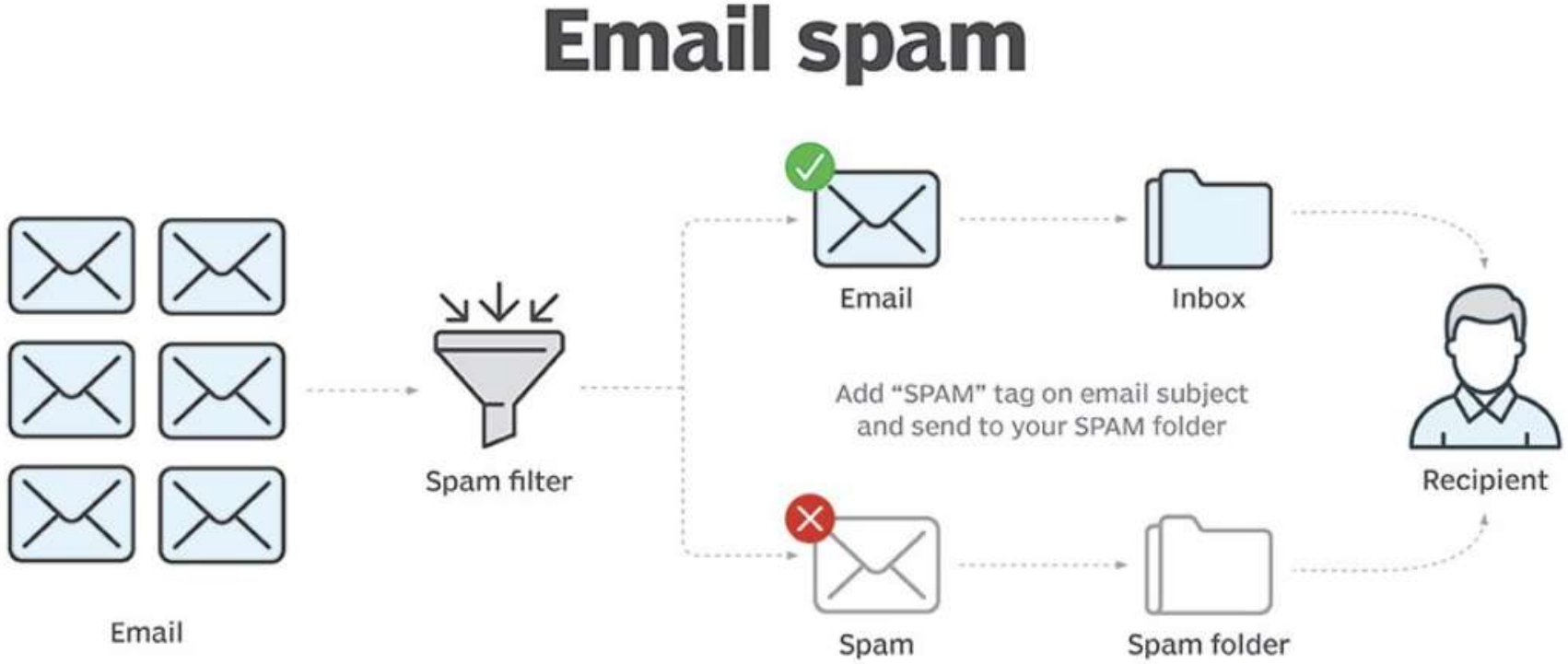
KnowBe4 Human error. Conquered

© 2020 KnowBe4, Inc. All rights reserved.

SECURITY AWARENESS TRAINING | www.KNOWBE4.com

Spam Filtering

- Spam filter: Goal is to determine whether an incoming message is legitimate (i.e., non-spam, ham) or unsolicited (i.e., spam)



Spam Filtering

- A typical data flow in spam filtering
- How input emails are processed and sent to the model
- To give data as input to the model, we'd need to transform them to some numerical format

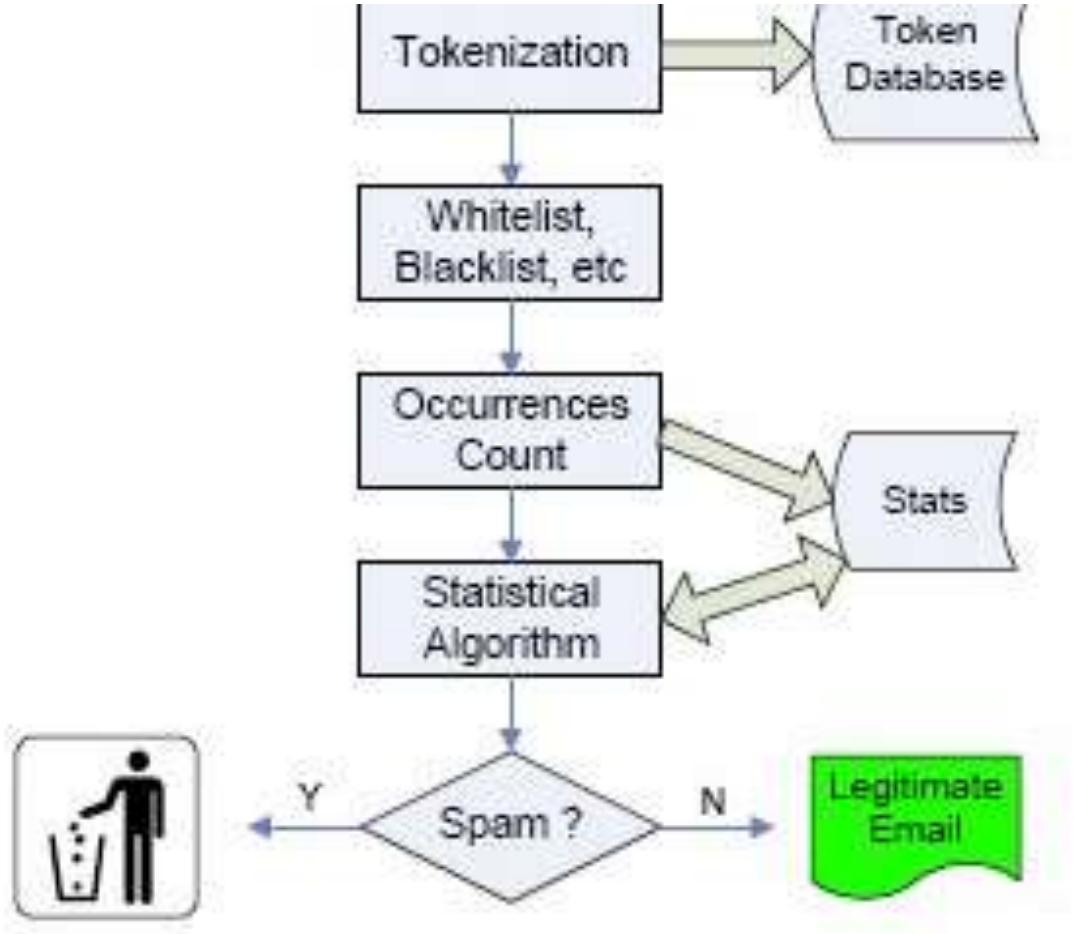


Image credit: Figure from: Yan (2010) – Workload Characterization of Spam Email Filtering Systems

Spam Filtering

- ML-based spam classifiers were among the first applications of machine learning in the cyber security domain
  - Subsequently, they were among the first to be attacked
  - Attackers' goal is to modify spam emails (without changing the nature of the message) to bypass spam filters

- Recent spam filters increasingly rely on machine learning and neural networks approaches for email classification
  - These approaches are being extensively used by email service providers like Gmail, Outlook, or Yahoo

Filtering Techniques

- Challenge-Response Filtering
- Blacklists and Whitelists
- Rule based filters
- Content based filters

Challenge-Response Filtering

- When a sender sends an email message, a challenge (e.g., CAPTHCA) is sent to the sender. A legitimate user can solve the challenge easily
- But, if a spammer forges the return address of a spam email or sends a large number of spam messages, it is difficult for the spammer to solve the challenge
- As a result, the challenge-response spam filtering system can differentiate legitimate emails from spam emails easily

- Limitation
  - Sometimes senders don’t or forget to reply to the challenge

Blacklists and Whitelists

- **Blacklists** of misbehaving servers or known spammers that are collected by several sites
- Sender id in the email is compared with the blacklist
- **Whitelists** are complementary to blacklists, and contain addresses of trusted contacts
- Use blacklists and whitelists for the first level filtering (before applying content checks) and not used as the only tool for making decision

- Limitation
  - Prone to wrong configurations with legitimate servers unable to exit from a list where they had been incorrectly inserted

Rule based Filtering

- **Rule-based filtering techniques**
  - apply static rules to discover similar patterns in a large number of spam and non-spam emails
  - Scores are assigned to each rule, and the scores are weighted based on the importance of the rule
  - Repeating patterns in a message increase the total score of being a spam
  - If the total score > a predefined threshold, the message is labeled as spam
- Rules could be created based on:
  - Words and phrases, lots of uppercase characters, exclamation points, unusual Subject lines, special characters, web links, HTML messages, background colors, etc
- Limitation
  - requires constant updating of the rules
  - continually adapting strategies by spammers

Content based Filtering

- Content based filtering techniques
  - The filter scans the content of incoming emails, looking for trigger keywords
    - E.g., keywords frequently used in spam emails, such as free, buy, application, mortgage
  - The content of the body and header of emails are scanned
- The frequency of occurrence and distribution of trigger words and phrases in the content of emails are used as features for training ML approaches, and afterward, for classifying new emails
  - Naïve Bayes classifiers are one of the early successful ML models for spam filtering
  - Other conventional ML approaches have been successfully applied, such as SVMs, k-nearest neighbors, decision trees, …
  - NNs and deep learning are commonly used nowadays for spam classification

Content based Filtering

- Scanning the body of emails explores the what in the email
  - Scanning the header of emails explores the who sent the email
- Email headers display important information, such as:
  - Message ID – an identifier generated by the sender's email service
    - There can be no two identical message IDs, hence, it helps to detect forged email headers
  - Sender address – is used to consult blacklists to check sender's domain reputation
  - DNS records – the DNS (Domain Name System) records of the sender allows to check the sender's SPF, DKIM, and DMARC policies regarding email authentication
    - SPF (Sender Policy Framework), DKIM (Domain Keys Identified Mail), DMARC (Domain-based Message Authentication, Reporting and Conformance)

An example of a Gmail header

Original Message

Message ID	<000000000000d7d44705c2854da2@google.com>
Created at:	Mon, May 17, 2021 at 2:57 PM (Delivered after 12 seconds)
From:	christian@gmail.com
To:	folderly@gmail.com
Subject:	Request to connect
SPF:	PASS with IP 209.85.220.69 Learn more
DKIM:	'PASS' with domain belkins.io Learn more
DMARC:	'PASS' Learn more

Spam filters in action - a Motivating Example

- How does an anti-spam algorithm behave in the classification of emails? - based on suspicious keywords, e.g. buy, shop
- Task: classify the email messages within a table, showing the number of occurrences of the individual keywords identified within the text of the emails, indicating the messages as spam or ham:

Email	Buy	Shop	Spam or Ham?
1	1	0	H
2	0	1	H
3	0	0	H
4	1	1	S

Spam filters in action - a Motivating Example

- Need to assign a score to every single email message
  - This score will be calculated using a scoring function that takes into account the number of occurrences of suspicious keywords contained within the text
- A simple scoring function with weights $y = 2B + 3S$

Email	B	S	2B + 3S	Spam or Ham?
1	1	0	2	H
2	0	1	3	H
3	0	0	0	H
4	1	1	5	S

- Threshold value to separate spam from ham: e.g. 4

Pre-processing Text in Email Messages

- Before we can process documents (emails), it's important to convert these documents into numerical representations

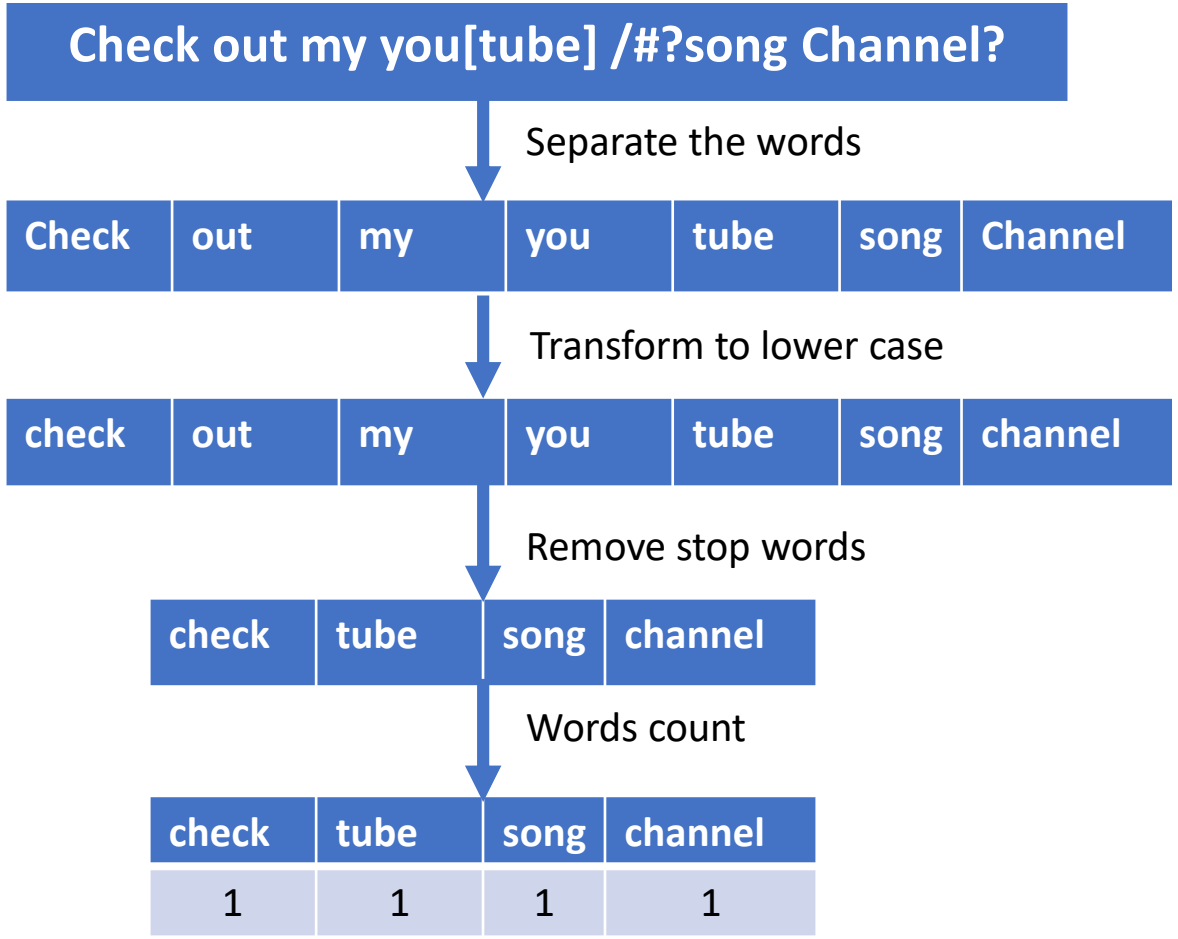
- Preprocessing text data in emails typically involves
  - Tokenization
    - Refers to separating the words in a text
    - Tokenization transforms an email into a sequence of representative symbols (tokens)
  - Vectorization
    - Transform to numerical format – ’vectors’
    - Text in Email Messages is preprocessed into numeric representation for use by ML models

Tokenization

- Tokenization usually includes:
- Remove punctuation signs (comma, period) or non-alphabetic characters (@, #, {,])
- Remove stop words, such as for, the, is, to, some
  - These words appear in both spam and non-spam emails, and are not relevant for filtering
- Correct spelling errors or abbreviations
- Change all words to lower-case letters
  - I.e., the model should consider Text and text as the same word
- Stemming - transforming words to their base form
  - E.g., the words buy-bought or grill-grilled have a common root

Tokenization

- Example of tokenization



Vectorization

- Then we need to transform the tokens (terms) to numerical format
- Some methods:
  - Bag of words
  - n-grams
  - TF-IDF

```
tokenized_messages: {
    'A': ['hello', 'mr', 'bear'],
    'B': ['hello', 'hello', 'gunter'],
    'C': ['goodbye', 'mr', 'gunter']
}

# Bag-of-words feature vector column labels:
# ['hello', 'mr', 'doggy', 'bear', 'gunter', 'goodbye']
vectorized_messages: {
    'A': [1,1,0,1,0,0],
    'B': [2,0,0,0,1,0],
    'C': [0,1,0,0,1,1]
}
```

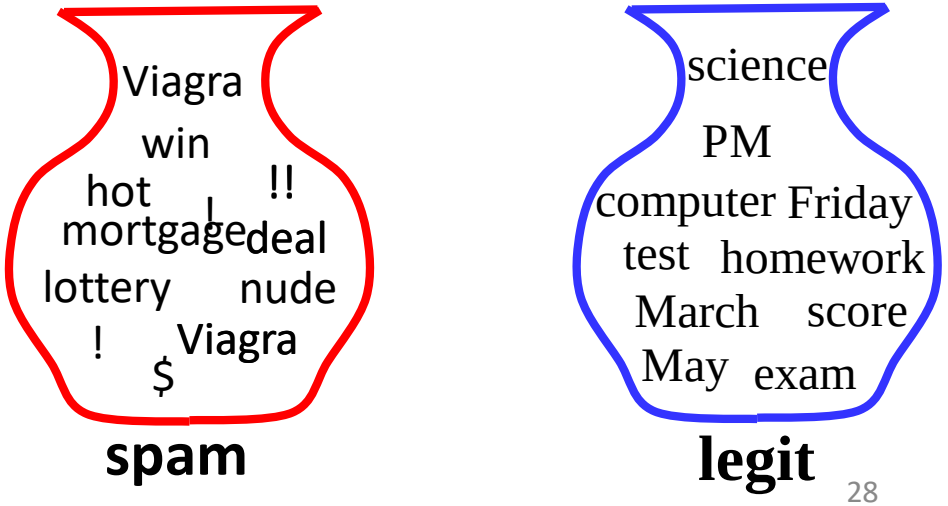
Bag-of-Words

- ***Bag-of-words model***
  - The tokenized words in emails are represented as a bag (i.e., set) of words
- The term *bag* implies that the order of the words and the structure of the text is lost
- Each word is a token having a numeric feature representation
- Typically, the frequency of occurrence of each word is used as a feature for training a classifier
- Example
  - Text: John likes to watch movies. Mary likes movies too.
  - Bag-of-words listing the words and the frequency of each word:

    {"John":1,"likes":2,"to":1, "watch":1, "movies":2,"Mary":1,"too":1}

Bag-of-Words

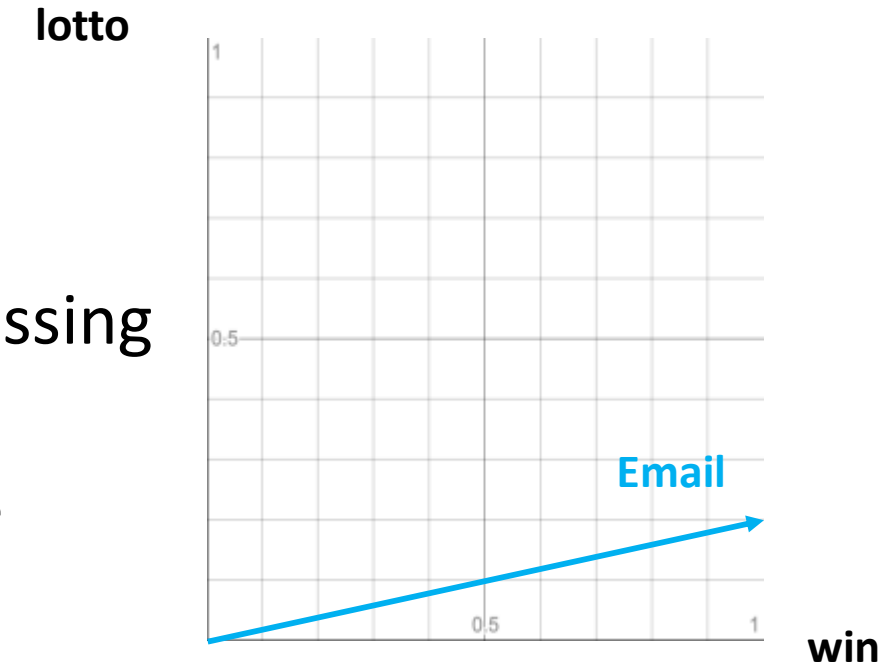
- Represent an email by the occurrence counts of each word
- What information is lost with this representation?
- **Ordering** of words is lost
  - Alice is quicker than Bob and Bob is quicker than Alice have the same vector representation
  - => n-grams
- **Term importance** is also lost
  - => TF-IDF



n-Grams

- Instead of using single words as tokens, it is also possible to use *n* consecutive words, referred to as ***n-grams***
  - Combining several consecutive words together creates more specialized tokens
  - E.g., the word *play* is considered a neutral word, but the two-words phrase *play lotto* is less neutral
    - Such *n*-grams consisting of adjacent pairs of words are called bigrams
    - *n*-grams consisting of single words are called unigrams
- The *n*-grams model preserves the words order, potentially capture more information than the bag-of-words model

TF-IDF



- **Vector-Space Model**
- Assume ***t*** distinct tokens remain after preprocessing
  - call them terms or the vocabulary
- These "orthogonal" tokens form a vector space

  Dimension = ***t*** = |vocabulary|
- Each token ***i***, in a document ***j***, is given a real-valued weight , w_{ij}
- Documents are expressed as ***t***-dimensional feature vectors

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

Document Collection

- A collection of *n* documents can be represented in the vector space model by a term-document matrix
- An entry in the matrix corresponds to the "weight" of a term in the document
  - zero means the term has no significance in the document or it simply doesn't exist in the document
- How to compute w_{ij}
  - **TF-IDF (or TF.IDF)**
  - Term frequency–inverse document frequency

$$\begin{pmatrix} & d_1 & d_2 & \dots & d_n \\ t_1 & w_{11} & w_{12} & \dots & w_{1n} \\ t_2 & w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ t_t & w_{t1} & w_{t2} & \dots & w_{tn} \end{pmatrix}$$

TF-IDF: Term Frequency

- To compute the *term frequency*, we can use a term-document ***count matrix***

Documents

Terms

	Email 1	Email 2	Email 3	Email 4	Email 5	Email 6
lotto	**4**	0	0	**3**	0	0
mr	1	0	0	1	0	0
bear	**2**	0	0	0	**4**	0
gunter	0	**5**	0	0	0	0
doggy	0	0	0	0	0	**5**
win	0	**2**	**3**	0	0	0

Log-Frequency Weighting

- The ***raw term frequency***, $\boldsymbol{tf_{t,d}}$, of term $\boldsymbol{t}$ in document $\boldsymbol{d}$ is defined as the number of times that $\boldsymbol{t}$ occurs in $\boldsymbol{d}$
  - However, relevance does not increase proportionally with raw term frequency
  - To this end, we can use *log-frequency weighting*

- The ***log-frequency weight***, $\boldsymbol{w_{t,d}}$, of term $\boldsymbol{t}$ in document $\boldsymbol{d}$ is:

$$\boldsymbol{w_{t,d}} = \begin{cases} \mathbf{1} + \boldsymbol{log}\left(tf_{t,d}\right) \boldsymbol{if}\ tf_{t,d} > \mathbf{0} \\ \mathbf{0}\ \boldsymbol{otherwise} \end{cases}$$

Log-Frequency Weighting: Example

- Here is our previous term-document ***count matrix***

Documents

	Email 1	Email 2	Email 3	Email 4	Email 5	Email 6
lotto	**4**	0	0	**3**	0	0
mr	1	0	0	1	0	0
bear	**2**	0	0	0	**4**	0
gunter	0	**5**	0	0	0	0
doggy	0	0	0	0	0	**5**
win	0	**2**	**3**	0	0	0

$$tf_{win,\,Email\,3} = 3 > 0$$

$$tf_{win\ \ Email\,5} = 0$$

Log-Frequency Weighting: Example

- Here is the corresponding term-document ***log frequency matrix***

Documents

	Email 1	Email 2	Email 3	Email 4	Email 5	Email 6
lotto	1.602	0	0	1.477	0	0
mr	1	0	0	1	0	0
bear	1.301	0	0	0	1.602	0
gunter	0	0	0	0	1	0
doggy	0	1.698	0	0	0	1.698
win	0	1.301	1.477	0	0	0

$$w_{win,\,Email\,3} = 1 + log(tf_{win,\,Email\,3}) = 1 + log(3) = 1.477$$

$$w_{win,\,Email\,5} = 0$$

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words: “a”, “the”, “to”, “of”, etc., are frequent but not very informative
- Want higher weights for rare terms than for more frequent terms
- Use document frequency to capture this

Document frequency

- **document frequency** $\boldsymbol{df_t}$, of term $\boldsymbol{t}$
  - the number of documents in the given collection that contain $\boldsymbol{t}$
- Thus, $\boldsymbol{df_t} \leq \boldsymbol{N}$, where $\boldsymbol{N}$ is the number of documents in the collection

- $\boldsymbol{df_t}$ is an inverse measure of the informativeness of $\boldsymbol{t}$
  - *the smaller the number of documents that contain* **t** *the more informative* **t** *is*
- **Inverse document frequency**, $\boldsymbol{idf_t}$, is defined as:

$$\boldsymbol{idf_t} = \boldsymbol{log}\left(\frac{\boldsymbol{N}}{\boldsymbol{df_t}}\right)$$

If $\boldsymbol{df_t}$ = 1, $\boldsymbol{idf_t} = \boldsymbol{log(N)}$
If $\boldsymbol{df_t}$ = N, $\boldsymbol{idf_t} = \boldsymbol{0}$

Document frequency

- **document frequency** $\boldsymbol{df_t}$, of term $\boldsymbol{t}$
  - the number of documents in the given collection that contain $\boldsymbol{t}$
- Thus, $\boldsymbol{df_t} \leq \boldsymbol{N}$, where $\boldsymbol{N}$ is the number of documents in the collection

- $\boldsymbol{df_t}$ is an inverse measure of the informativeness of $\boldsymbol{t}$
  - *the smaller the number of documents that contain* **t** *the more informative* **t** *is*
- **Inverse document frequency**, $\boldsymbol{idf_t}$, is defined as:

$$\boldsymbol{idf_t = log\left(\frac{N}{df_t}\right)}$$

Note: There is only one value of $\boldsymbol{idf_t}$ for each $\boldsymbol{t}$ in the collection

TF-IDF Weighting

- A typical combined term importance indicator is *TF.IDF weighting*:

$$\boldsymbol{w_{t,d} = TF.IDF = (1 + log(}tf_{t,d}\boldsymbol{)) \times\ log\left(\frac{N}{df_t}\right)}$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight
- **TF.IDF** increases with:
  - the number of occurrences of term ***t*** in document ***d***
  - the rarity of ***t*** in the collection

TF-IDF: Example

- here is the respective term-document ***TF-IDF matrix***

Documents

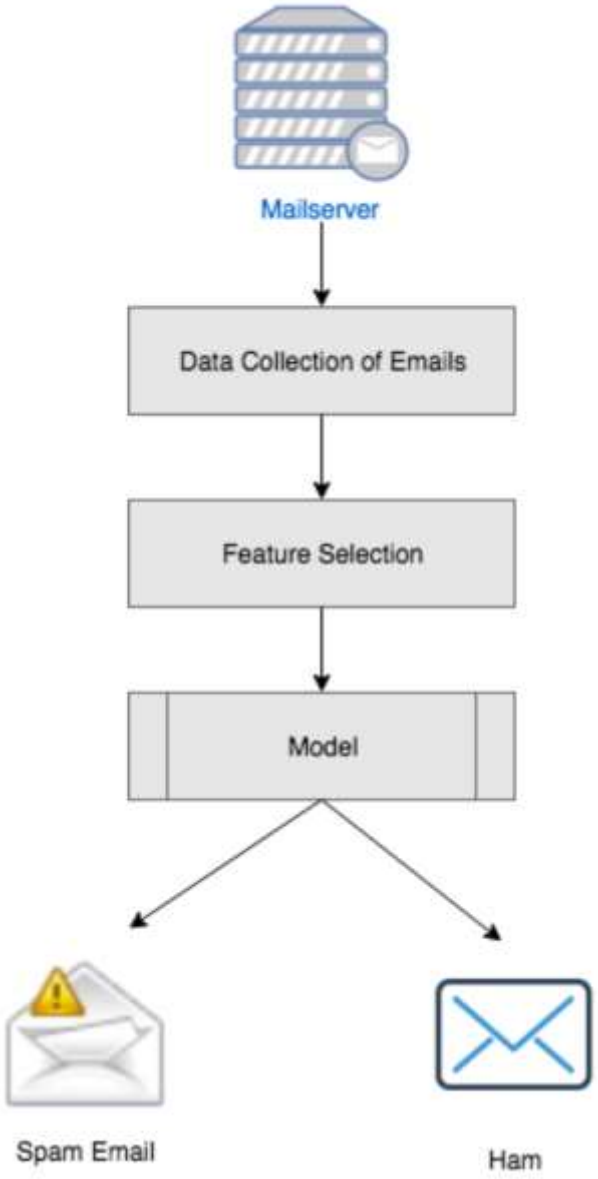
	Email 1	Email 2	Email 3	Email 4	Email 5	Email 6
lotto	0.764	0	0	0.704	0	0
mr	0.477	0	0	0.477	0	0
bear	0.620	0	0	0	0.764	0
gunter	0	0	0	0	0.778	0
doggy	0	0.810	0	0	0	0.810
win	0	0.620	0.704	0	0	0

$$w_{win,\,Email\,3} = [1 + log(tf_{win,\,Email\,3})] \times log\left(\frac{N}{df_{win}}\right) = 1.477 \times log\left(\frac{6}{2}\right) = 0.704$$

Spam Detection

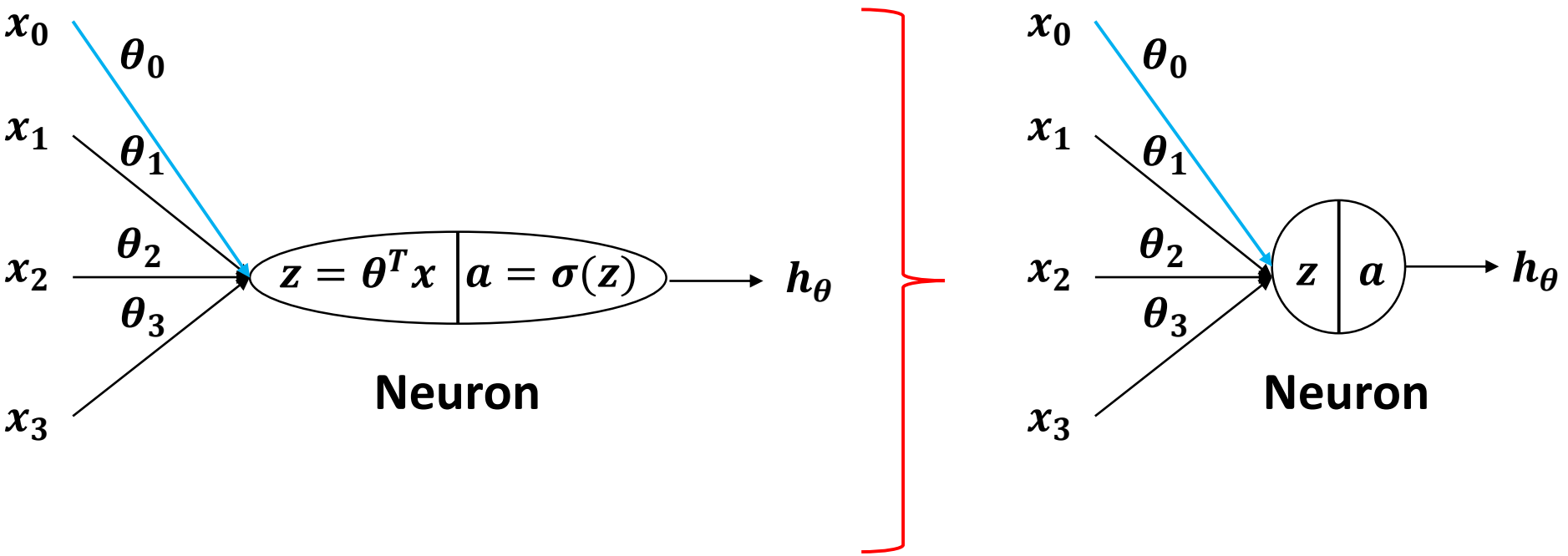
- Task: separate spam emails from a set of non-spam emails
  - Given a large collection of example emails, each labeled “spam” or “ham”
  - Learn to predict labels of new, future emails

- How: use neural networks to distinguish between spam and ham emails



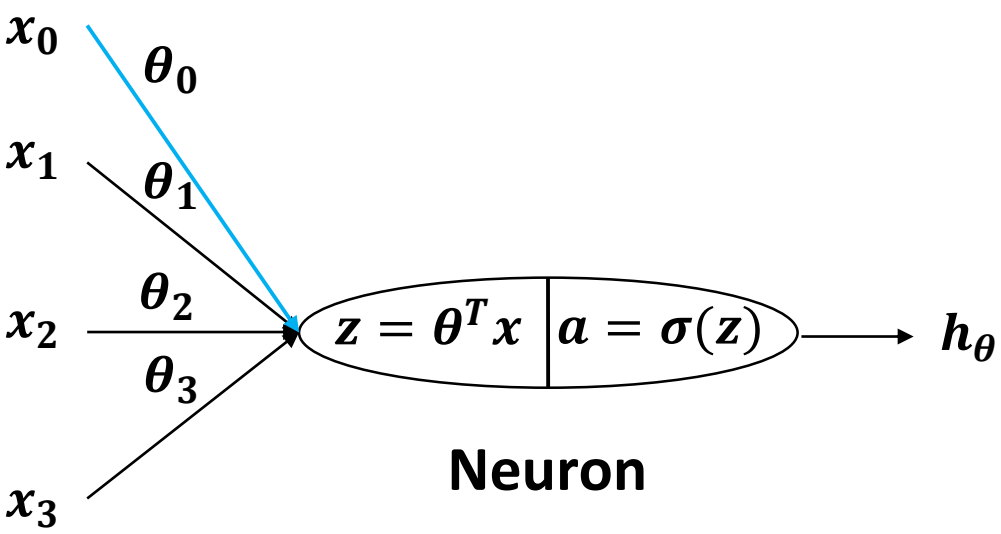
Towards Neural Networks

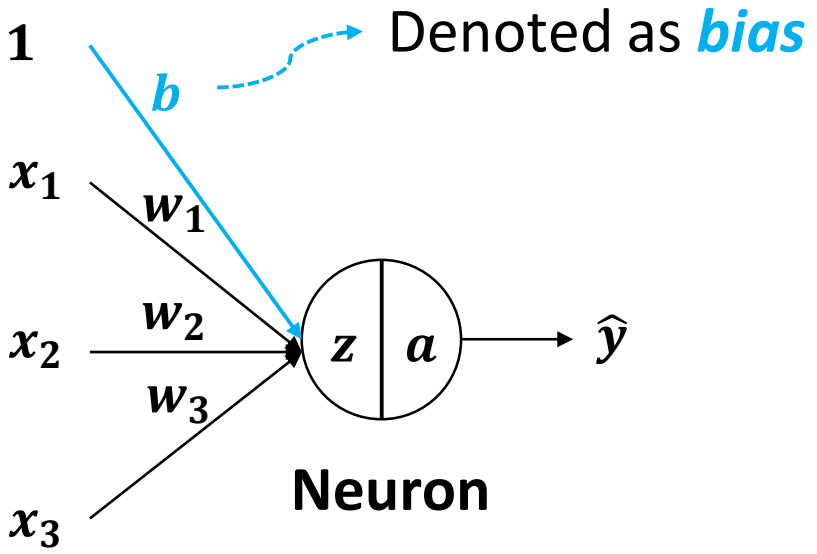
- Technically, logistic regression is a **neural network** with only 1 neuron



Towards Neural Networks

- Technically, logistic regression is a **neural network** with only 1 neuron

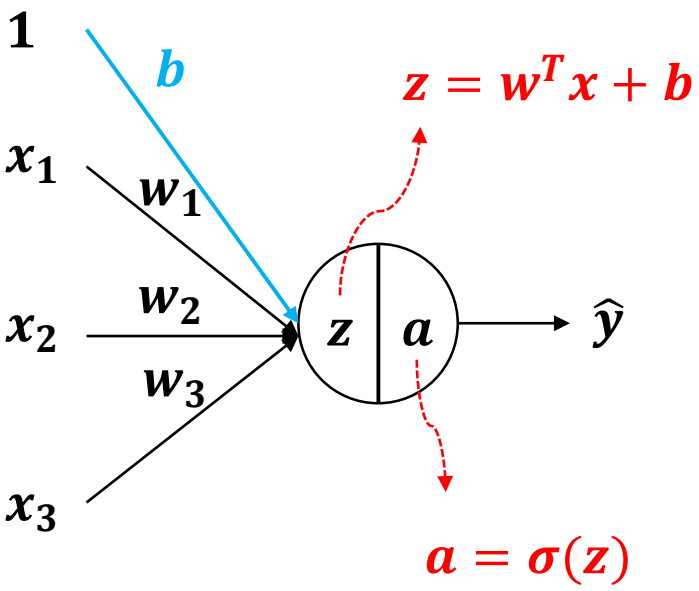




Using the notations in the neural network literature, where $\boldsymbol{\theta} = \boldsymbol{w} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}]$ ($\boldsymbol{w_0}$ is not part of this vector here), $\boldsymbol{h_\theta} = \hat{\boldsymbol{y}}$, and $\boldsymbol{\theta_0} = \boldsymbol{w_0} = \boldsymbol{b}$

Towards Neural Networks

- Technically, logistic regression is a **neural network** with only 1 neuron



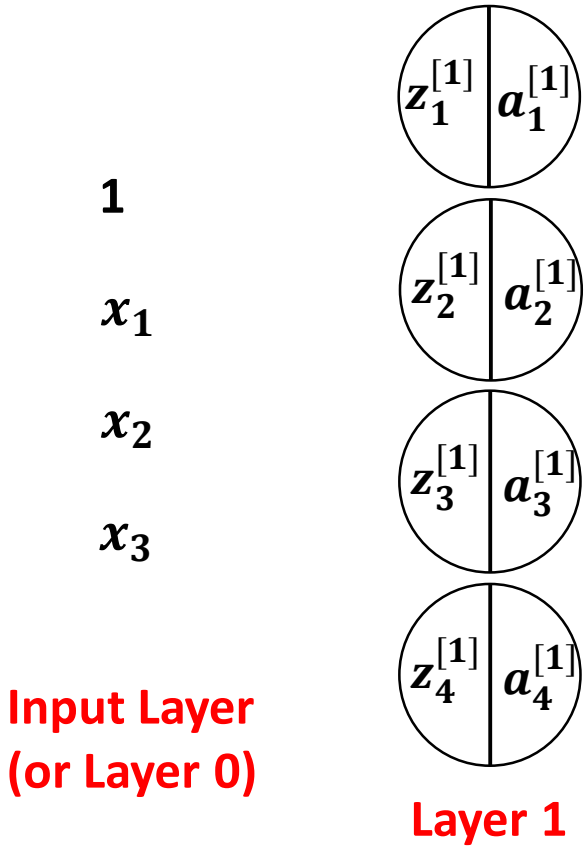
$$\hat{y} = a = \sigma(z) = \sigma\left(w^T x + b\right) = \sigma\left([w_1\ w_2\ w_3]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b\right)$$

$$= \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$

$$= \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}}$$

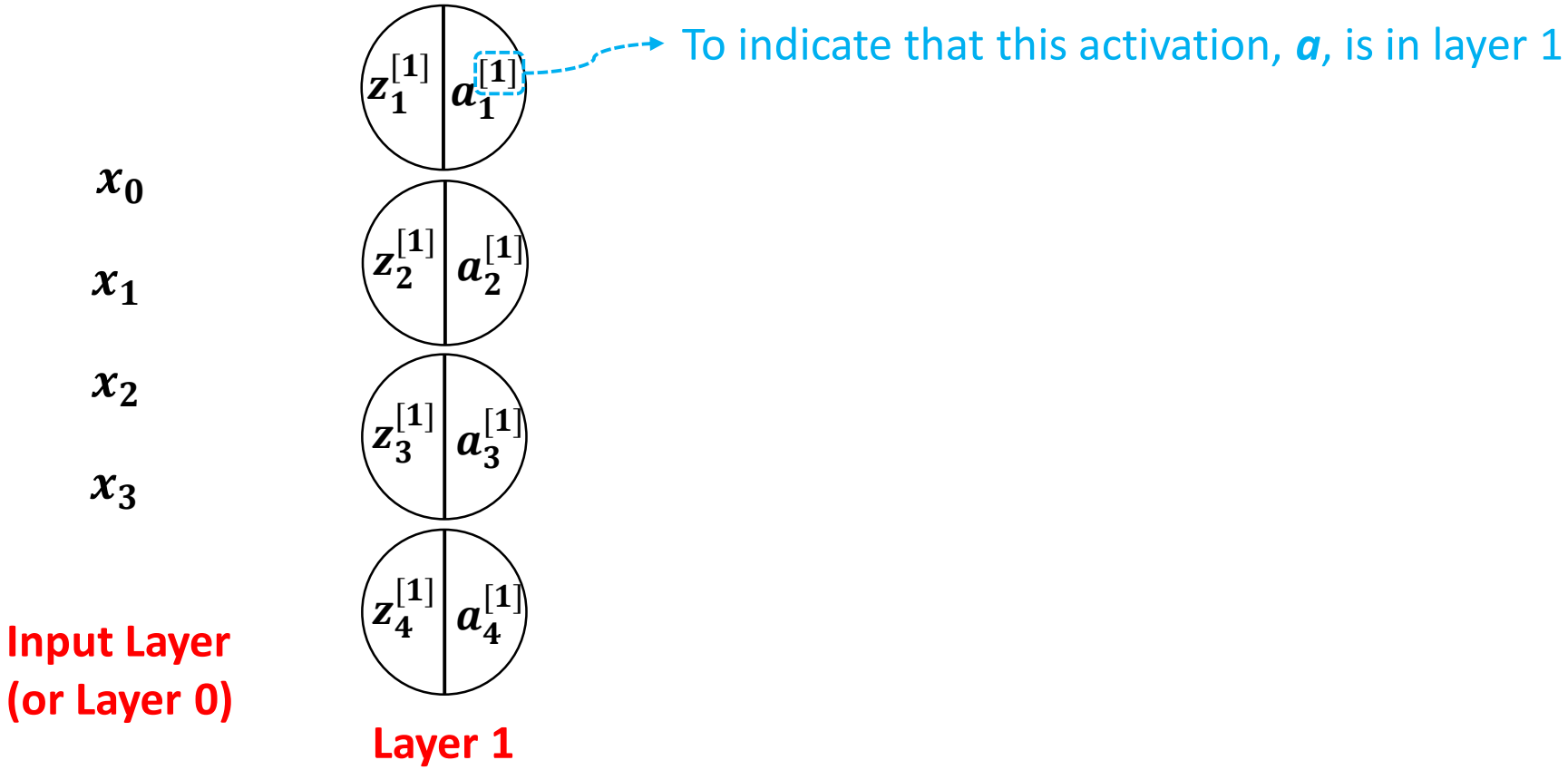
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



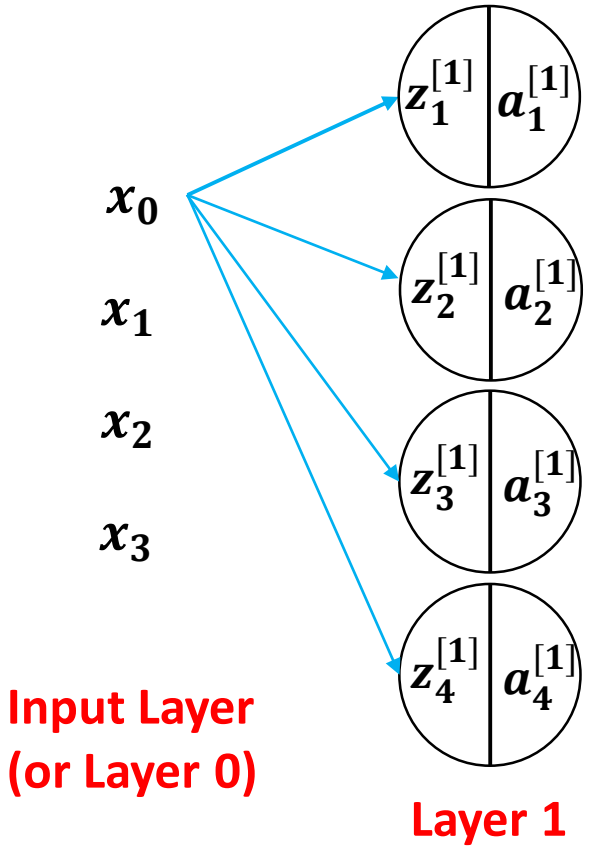
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



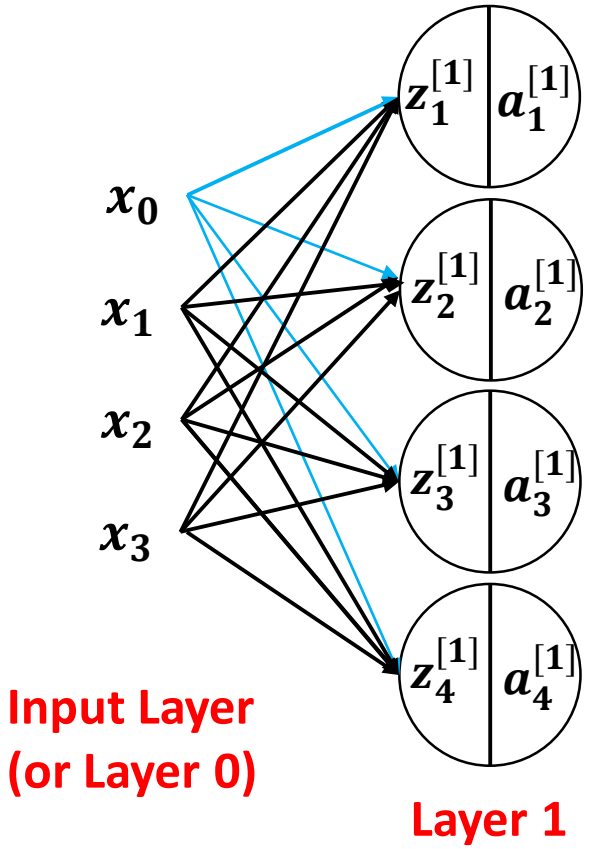
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



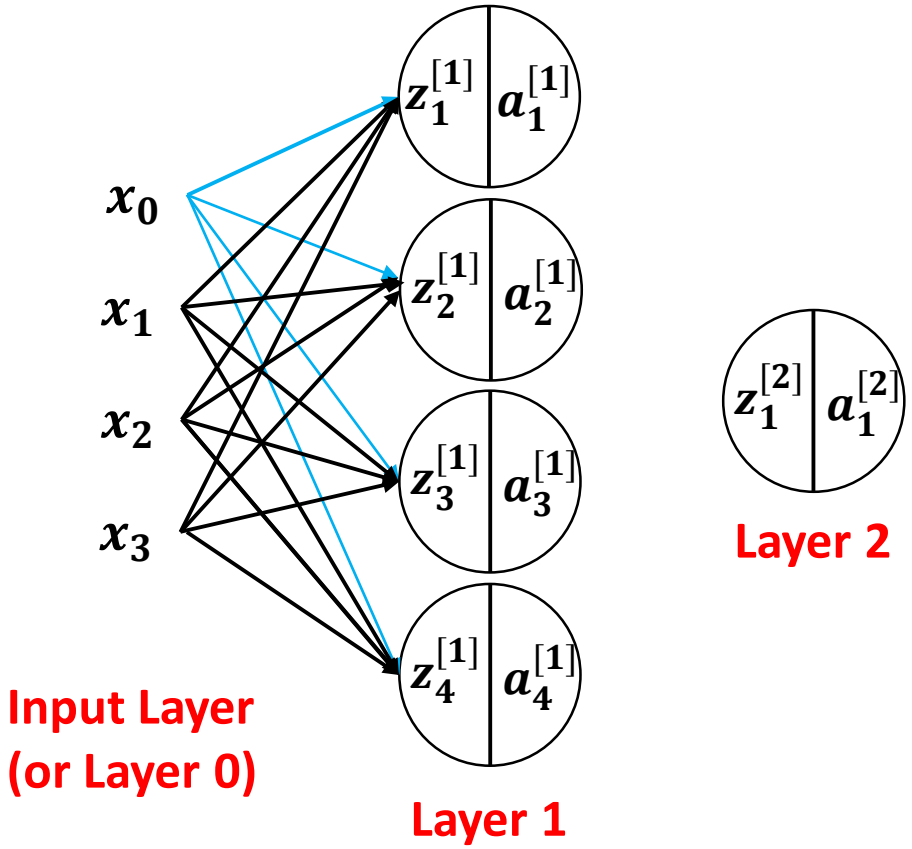
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



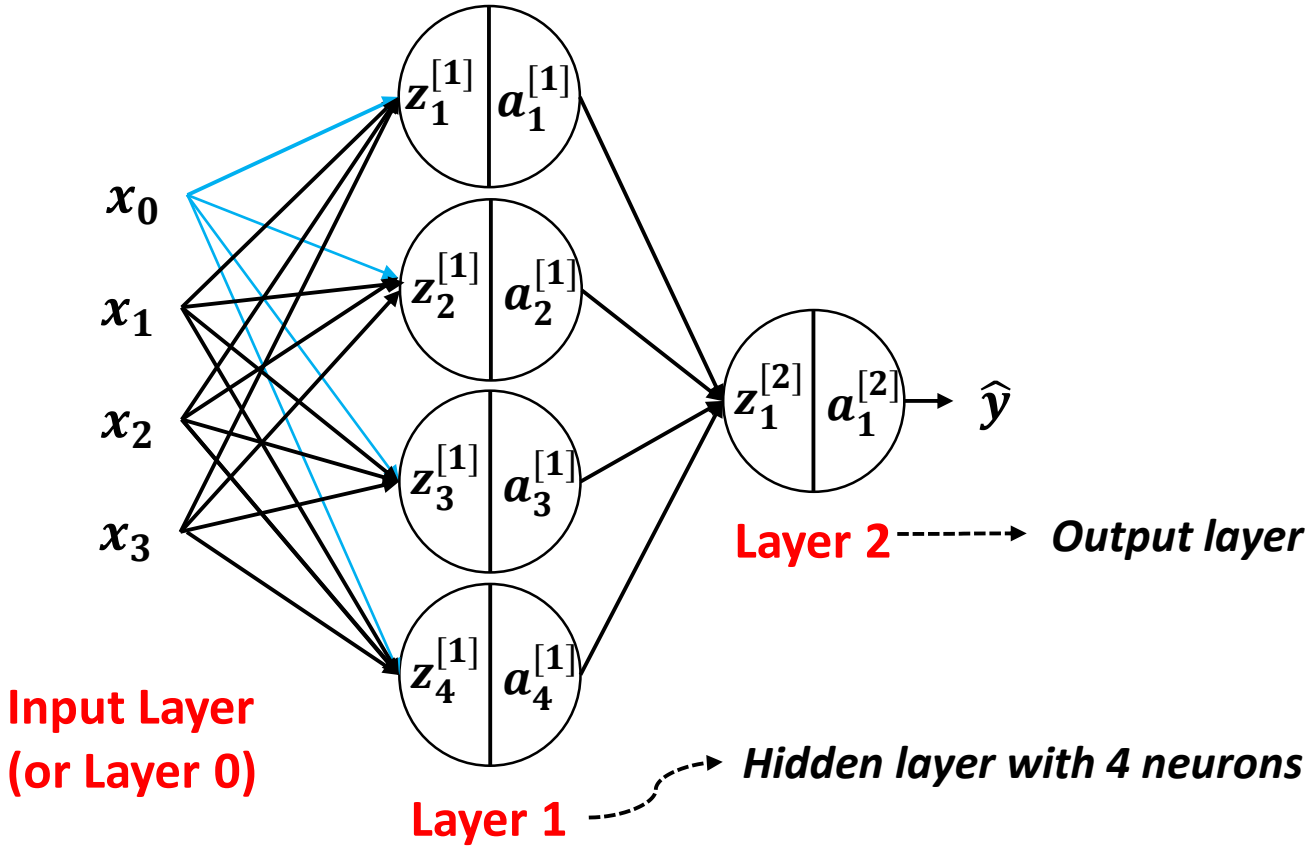
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Neural Networks

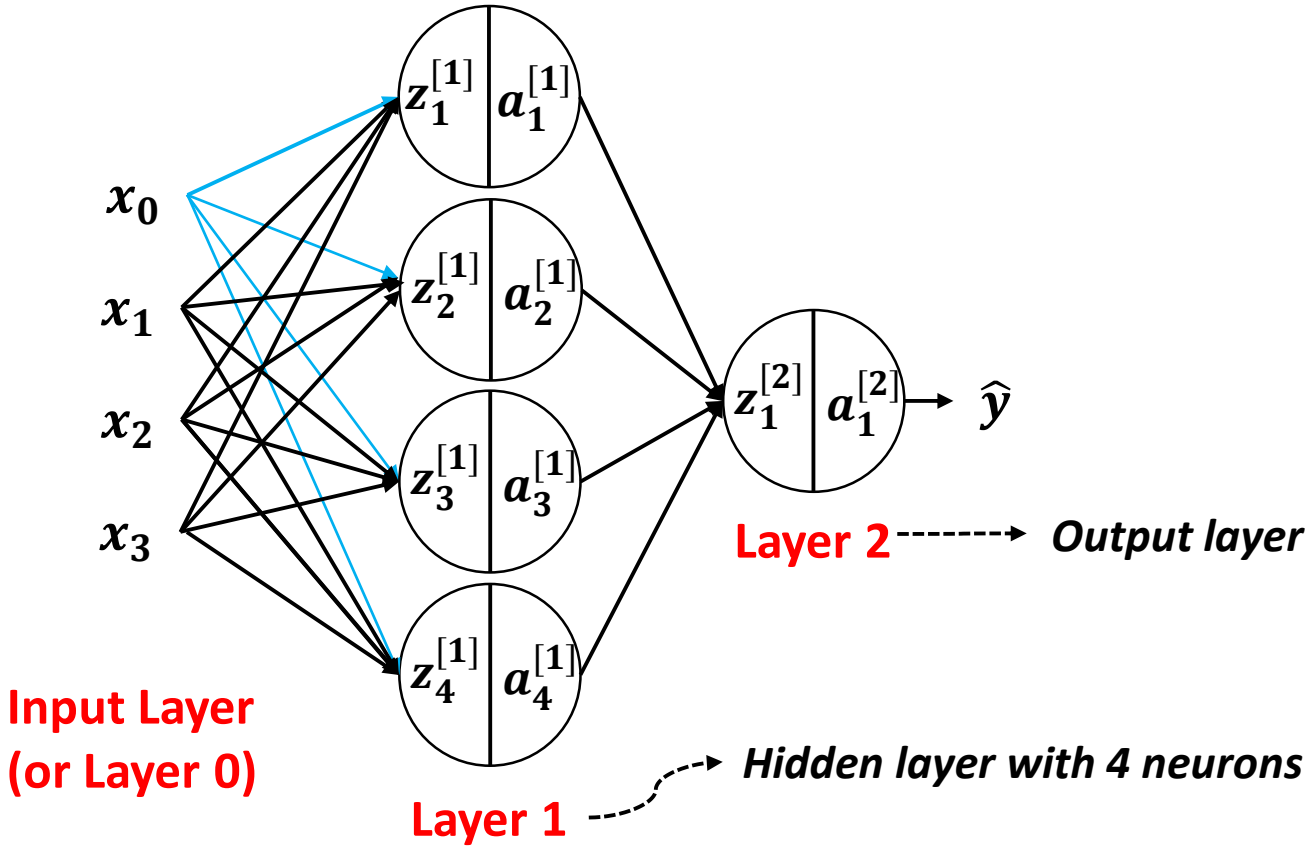
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



By convention, this neural network is said to have 2 layers (and not 3) since the input layer is typically not counted!

Neural Networks

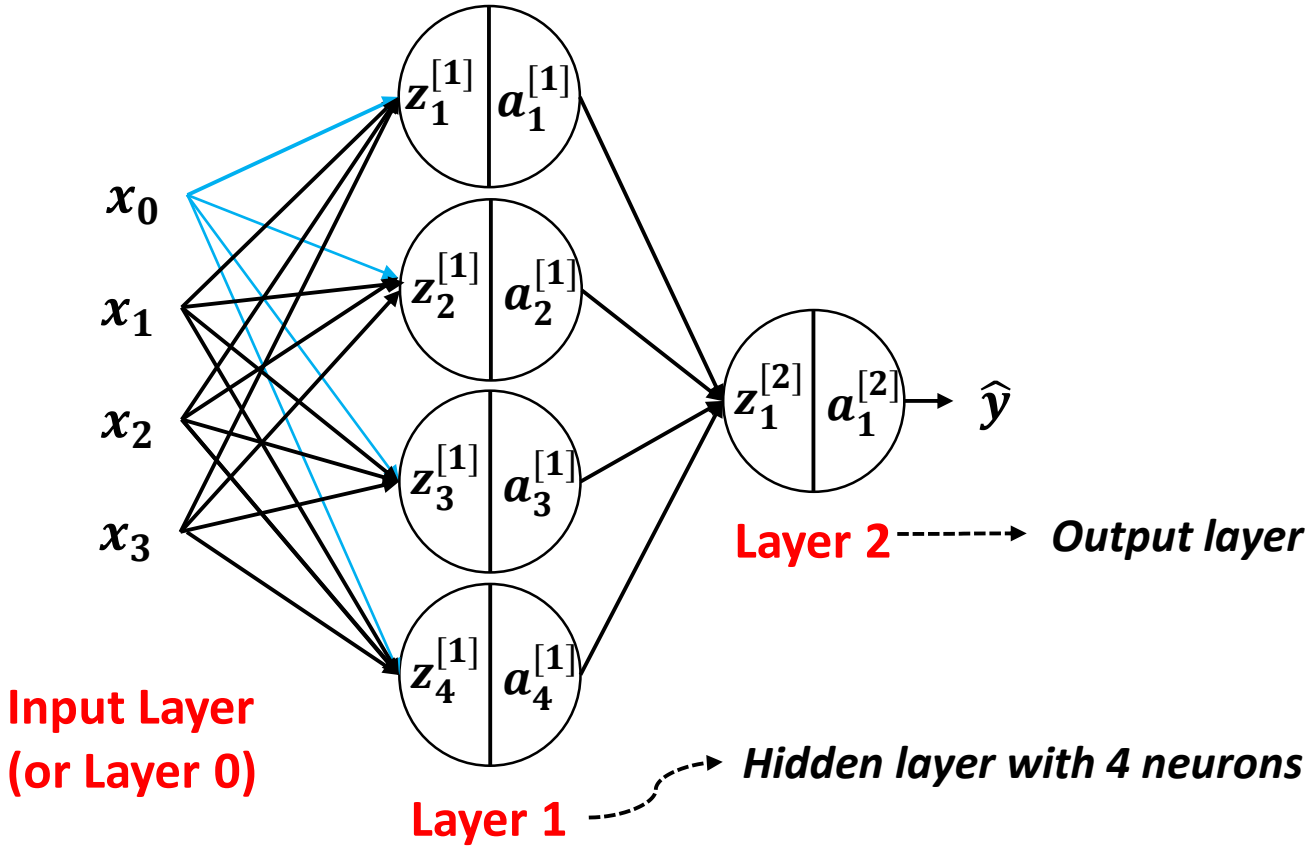
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Also, the more layers we add, the ***deeper*** the neural network becomes, giving rise to the concept of ***deep learning***!

Neural Networks

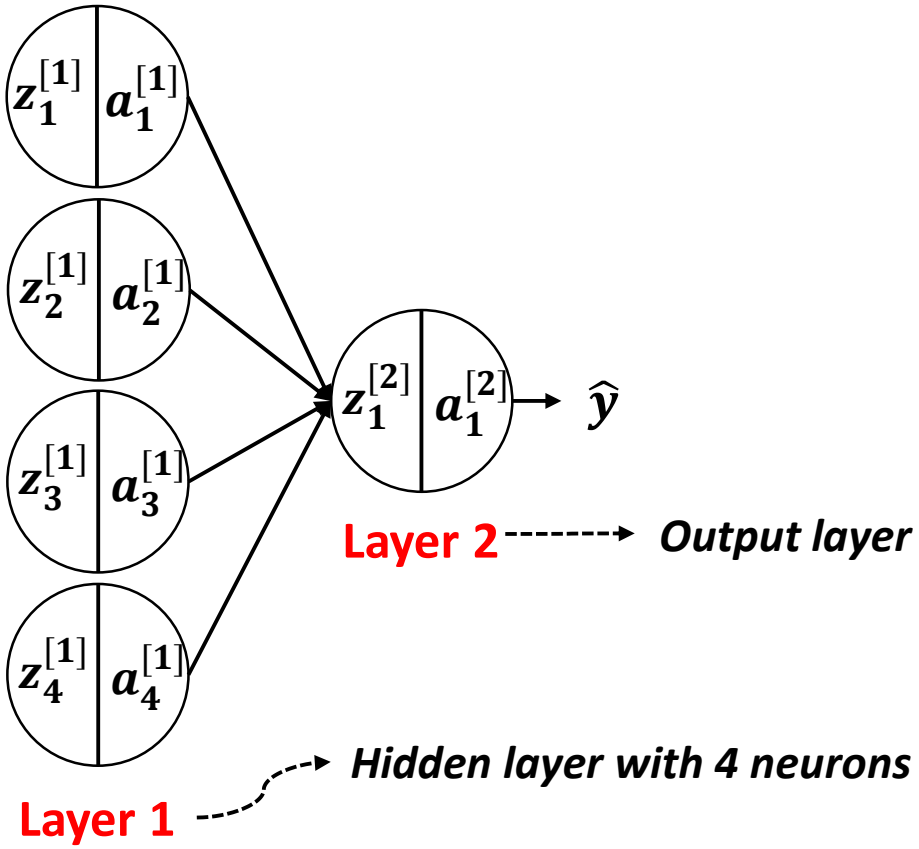
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Interestingly, neural networks *learn* their own features!

Neural Networks

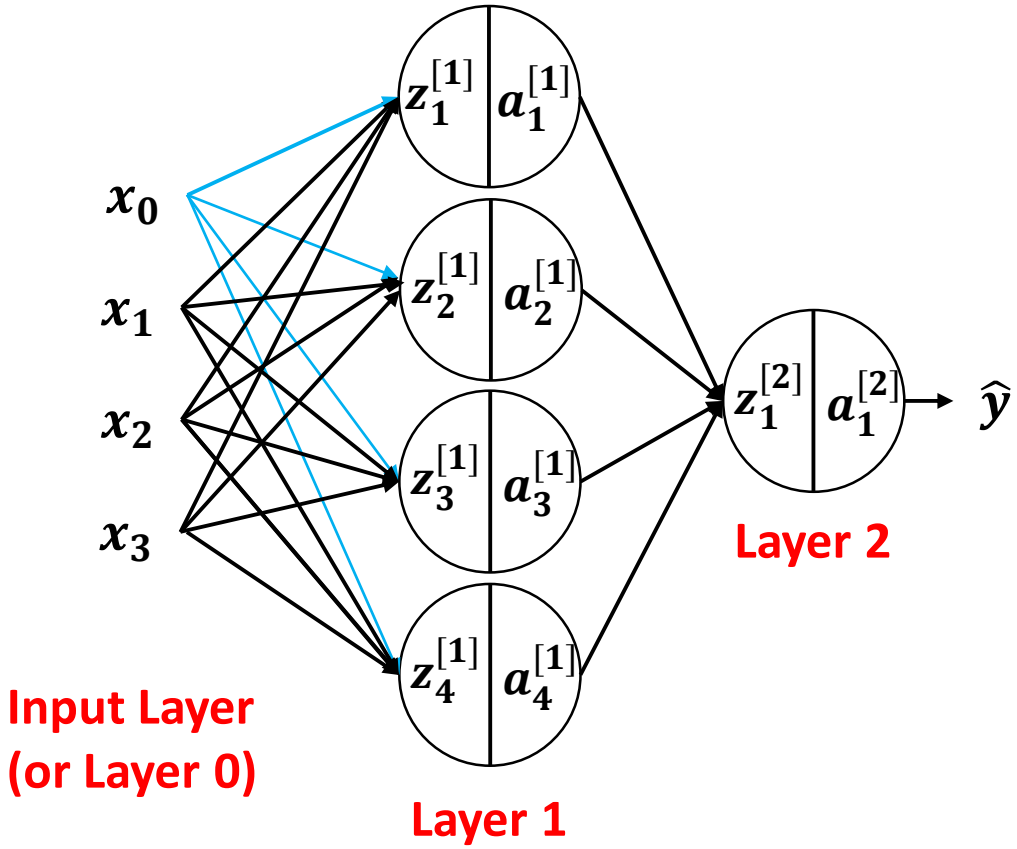
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



This looks like logistic regression, but with ***features that were learnt*** (i.e., $a_1^{[1]}$, $a_2^{[1]}$, $a_3^{[1]}$, $a_4^{[1]}$) ***and NOT engineered by us*** (i.e., x_1, x_2, and x_3)

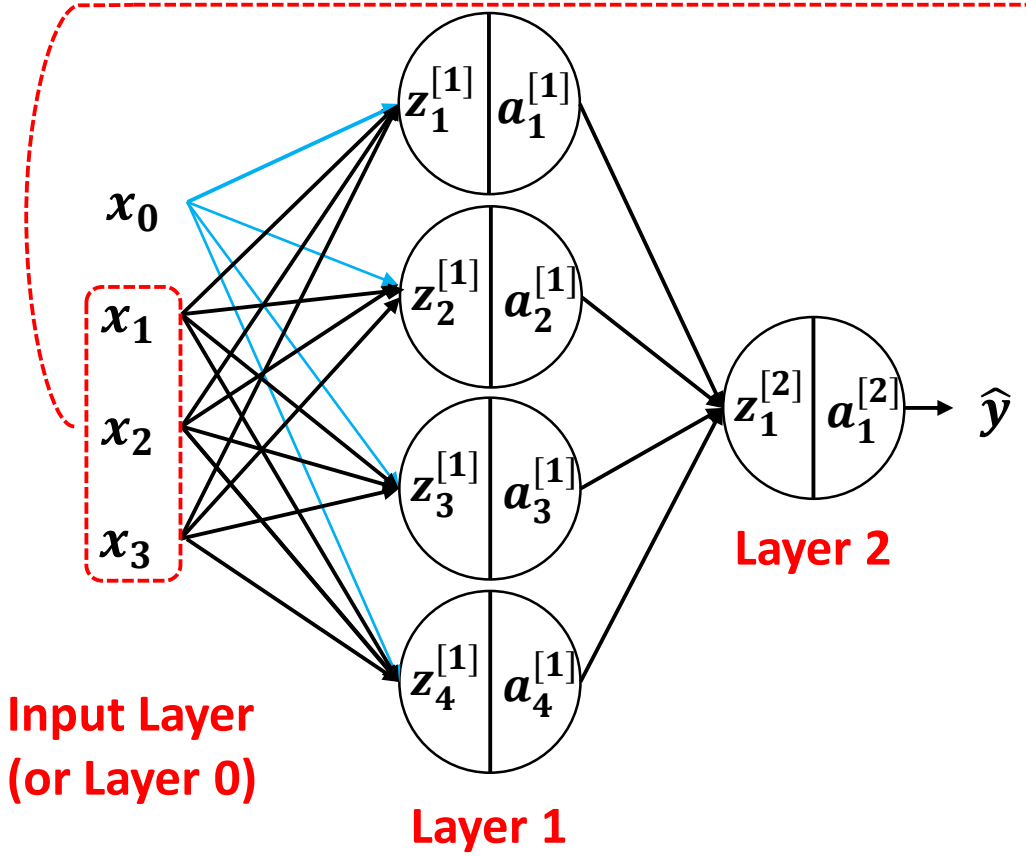
Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



Vectorizing Input and All Variables

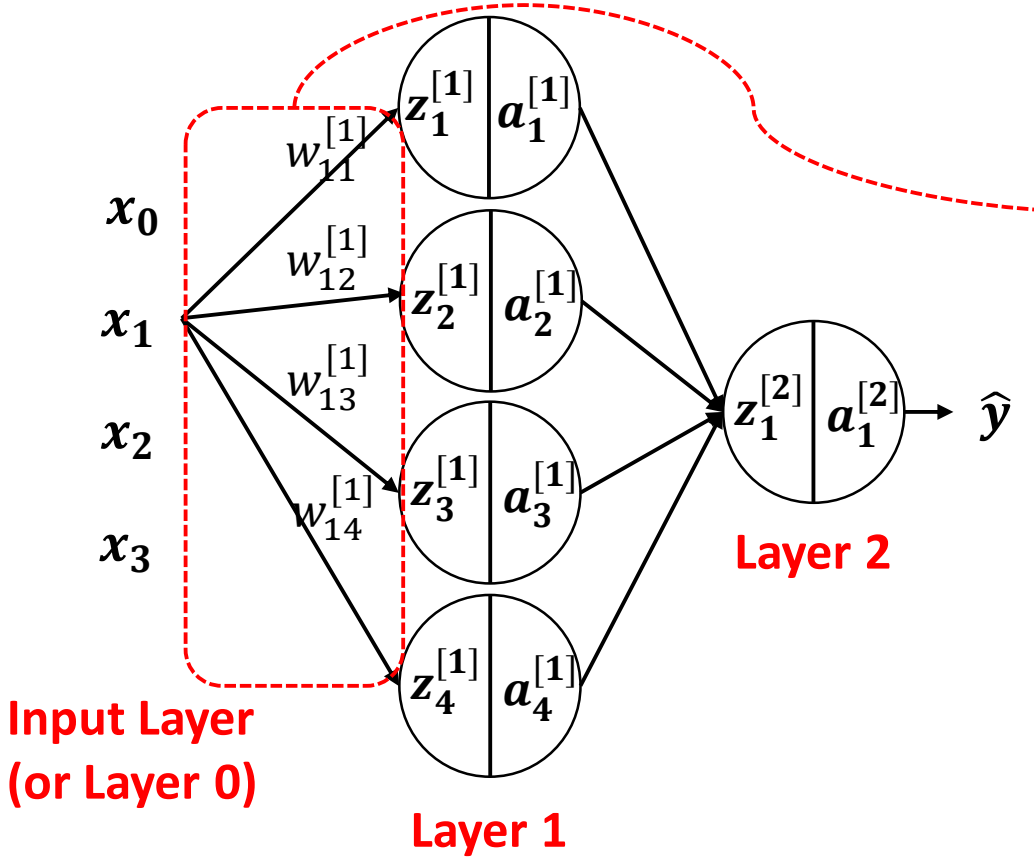
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Vectorizing Input and All Variables

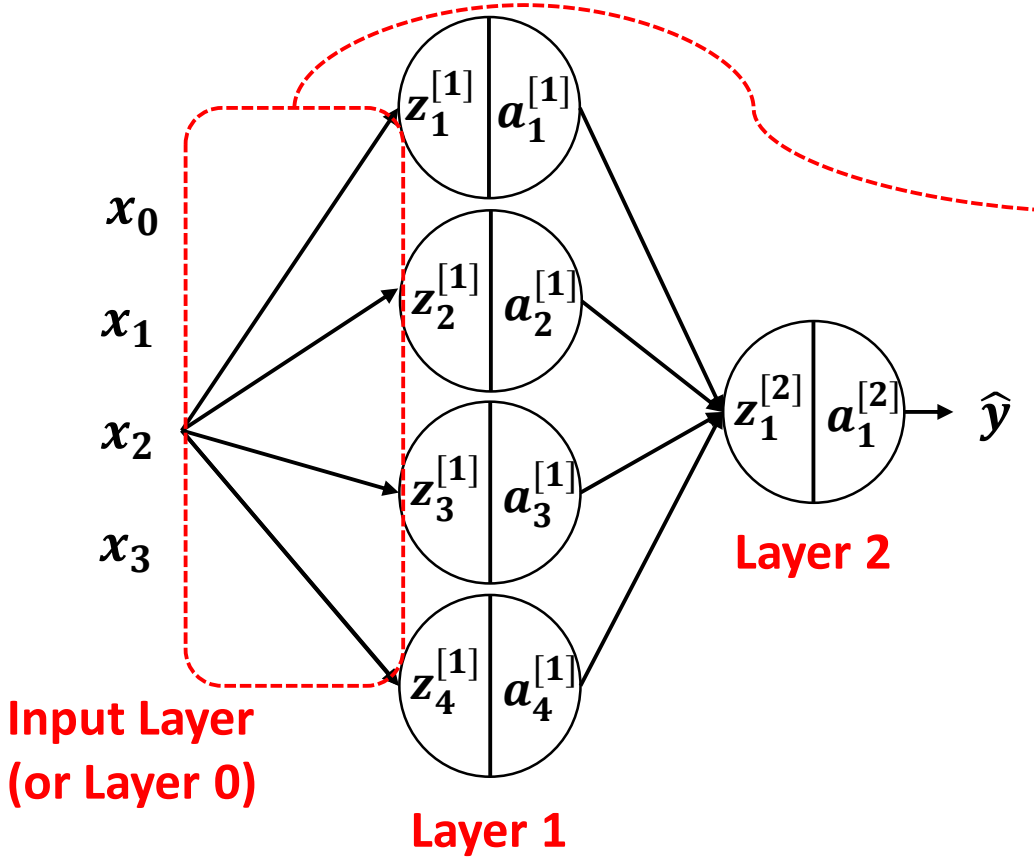
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} & w_{14}^{[1]} \\ & & & \\ & & & \end{bmatrix}$$

Vectorizing Input and All Variables

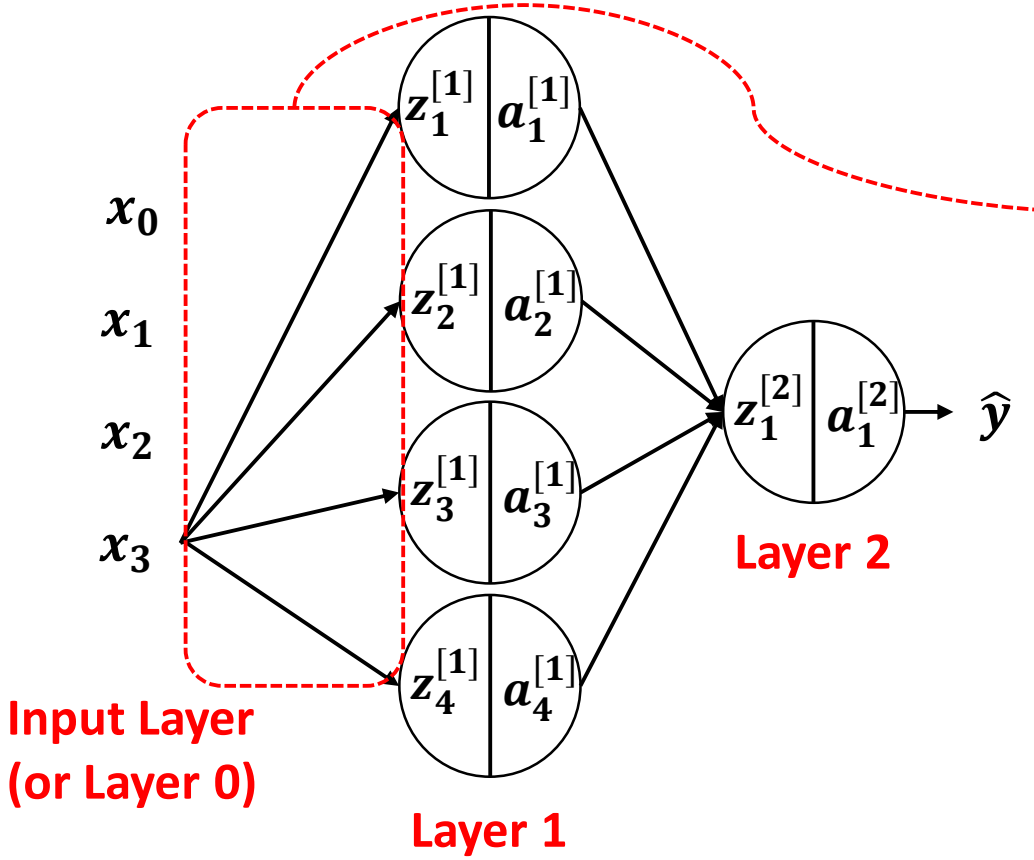
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} & w_{24}^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

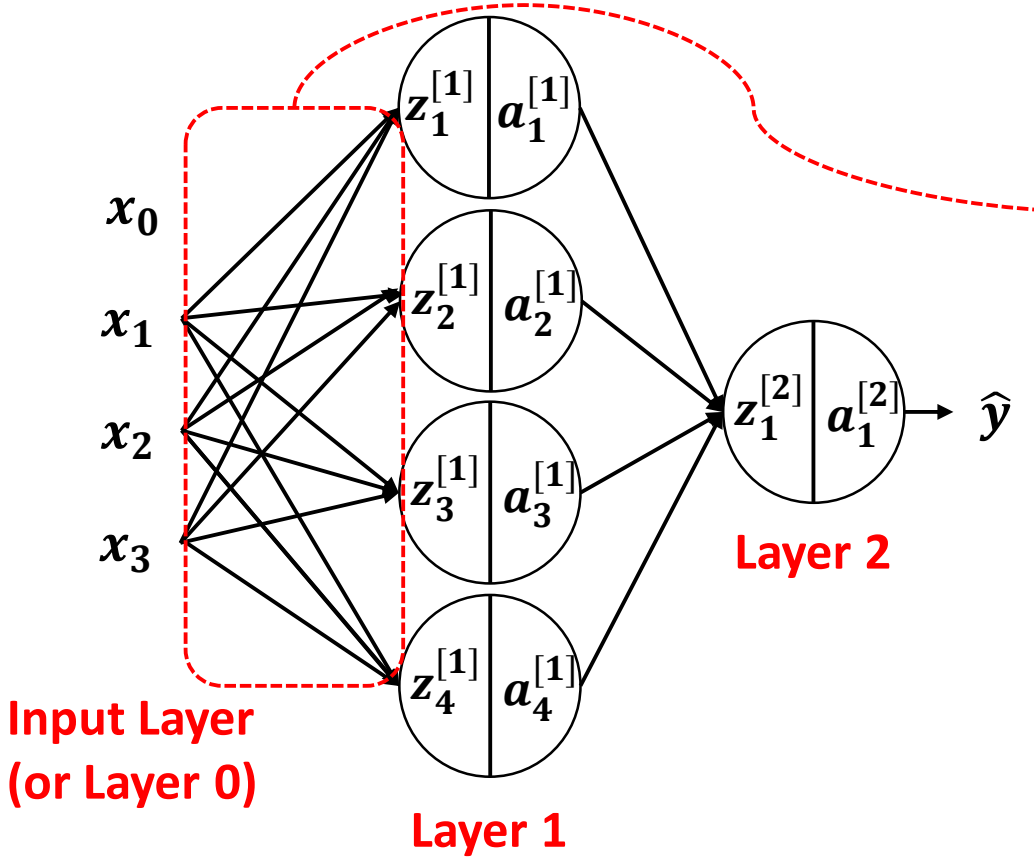
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

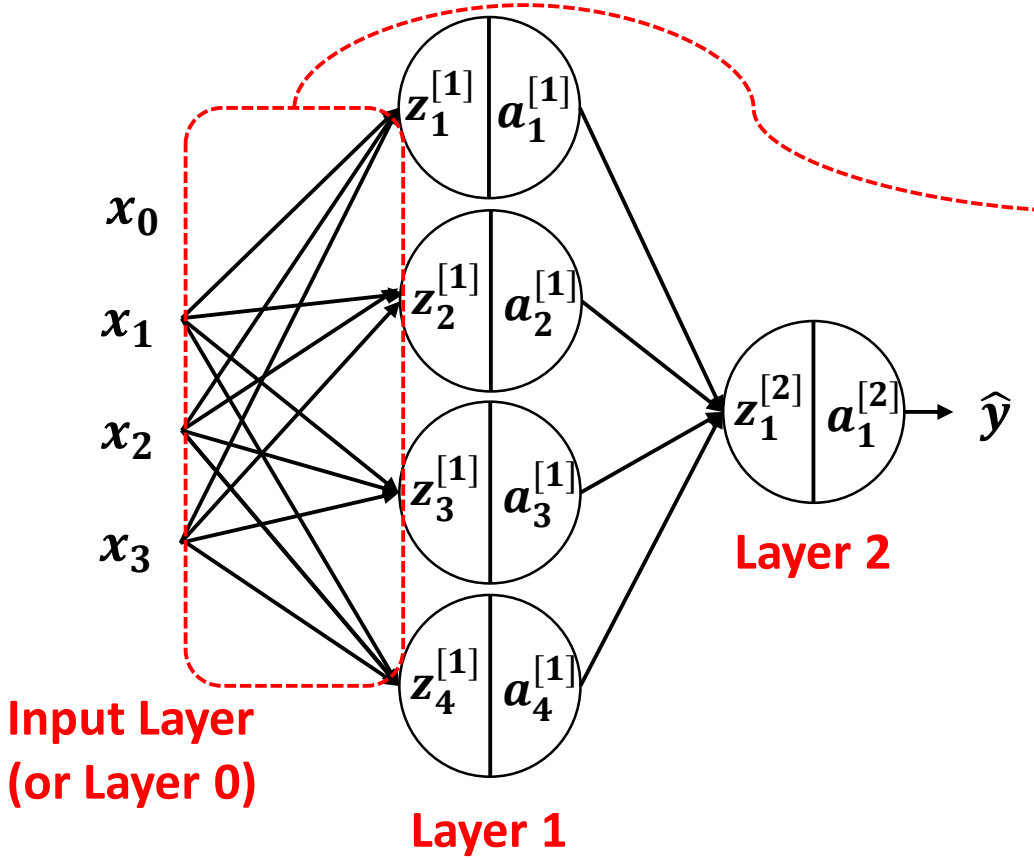


$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} & w_{14}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} & w_{24}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

Dimension of $\boldsymbol{w}^{[1]}$ = (3, 4)

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

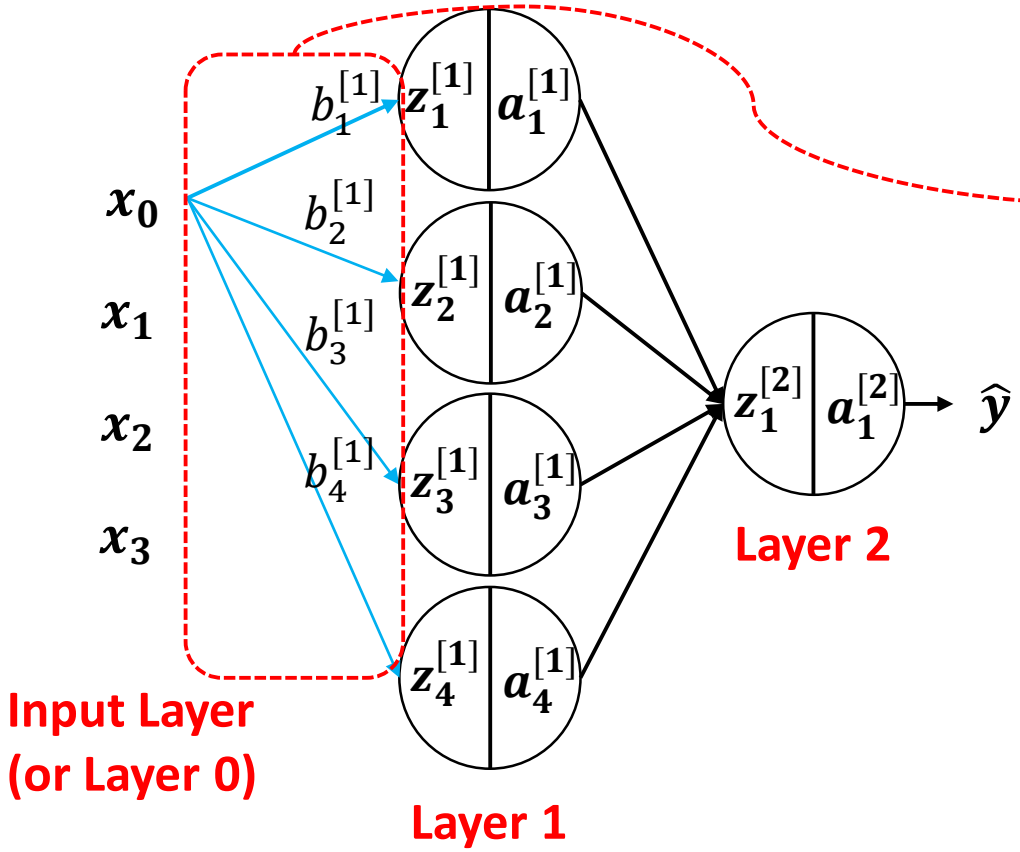


$$w^{[1]^T} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \\ w_{14}^{[1]} & w_{24}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

Dimension of $w^{[1]^T}$ = (4, 3)

Vectorizing Input and All Variables

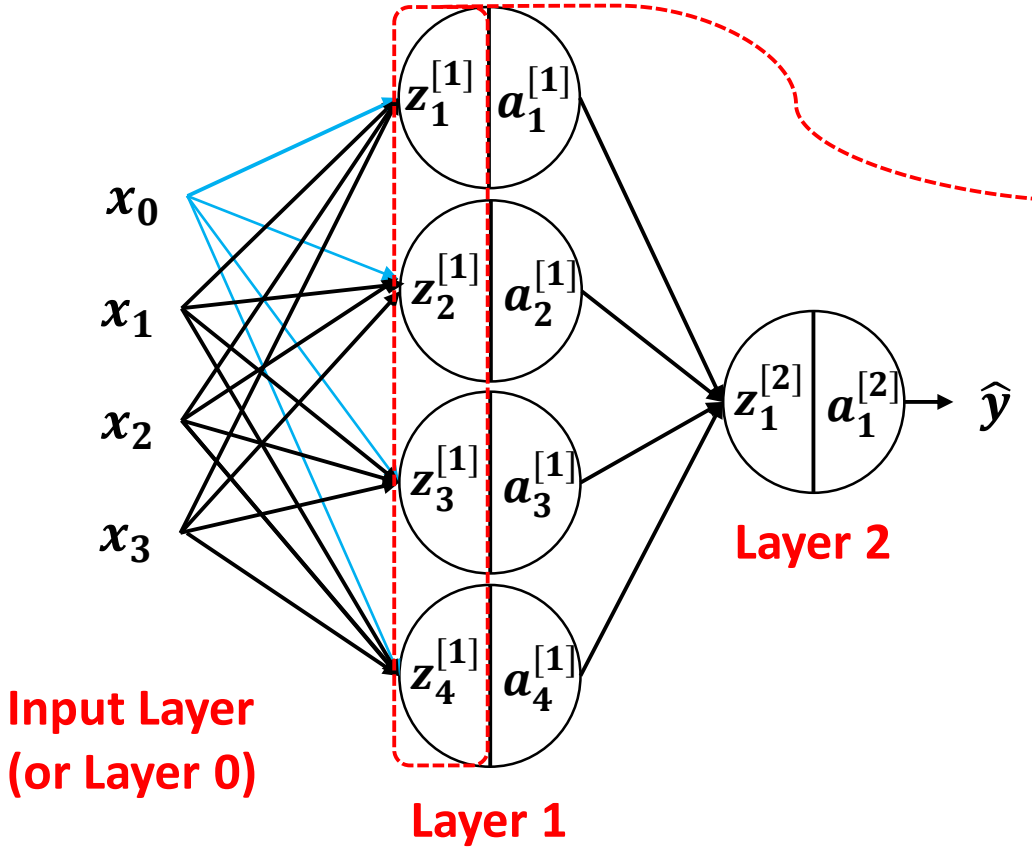
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

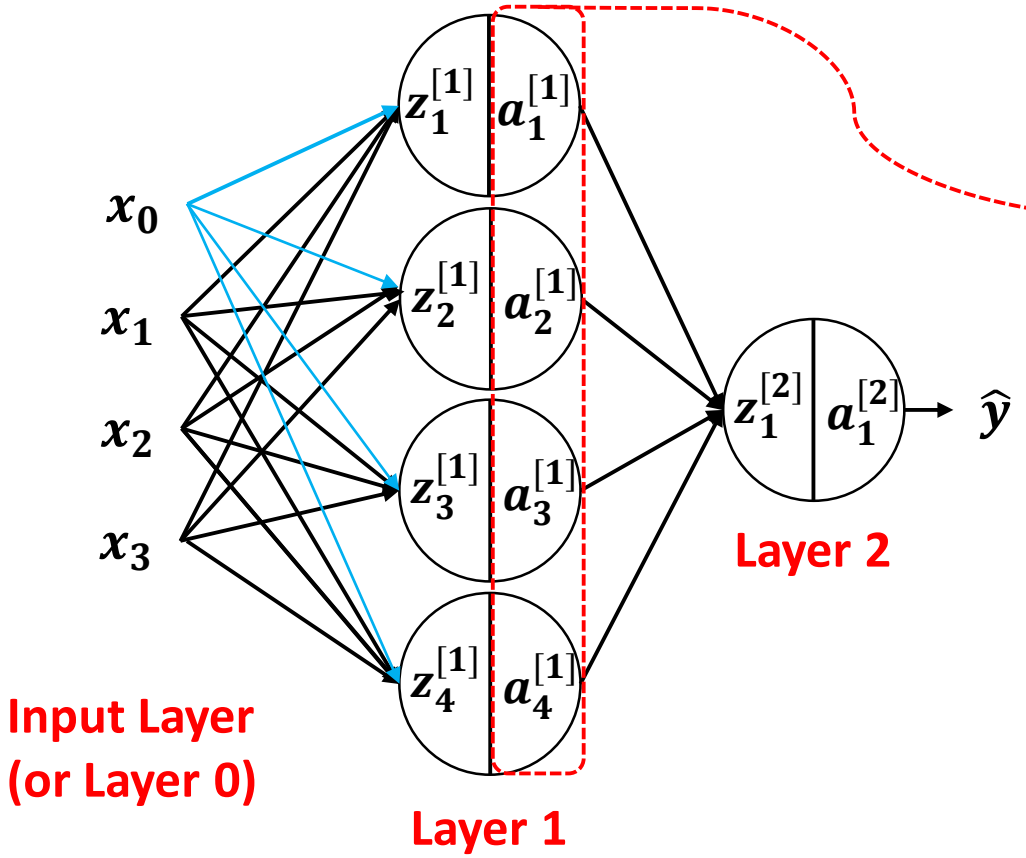
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

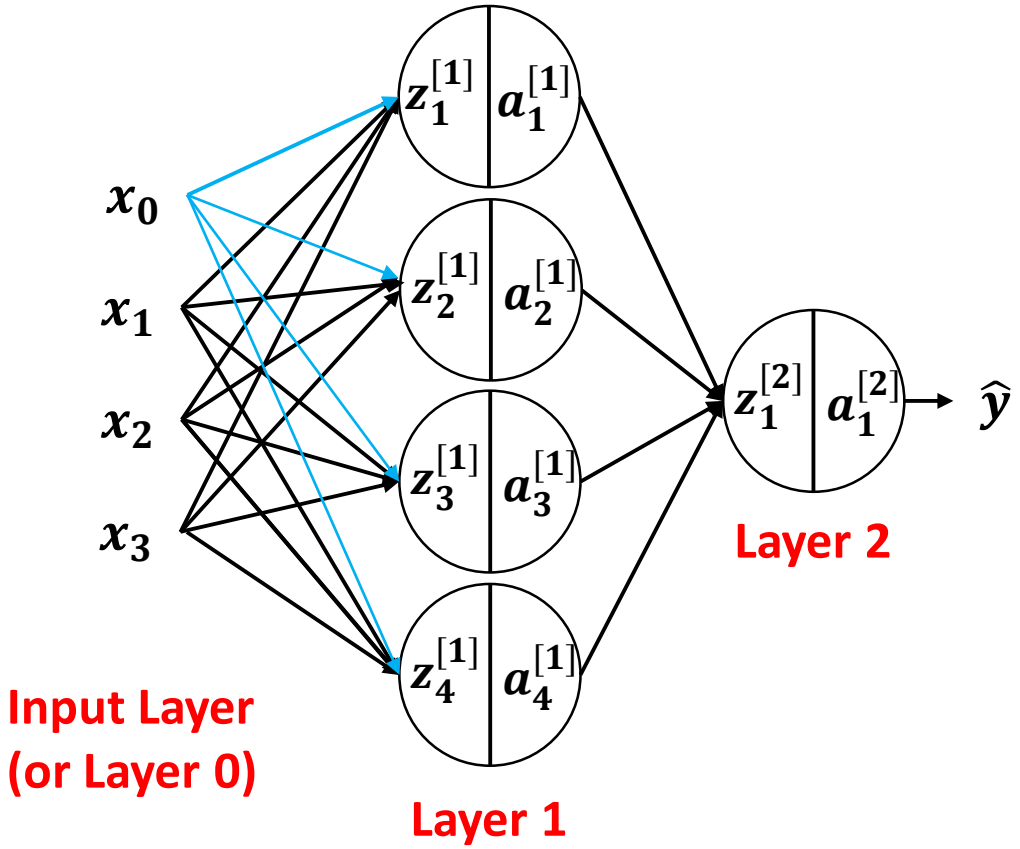
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

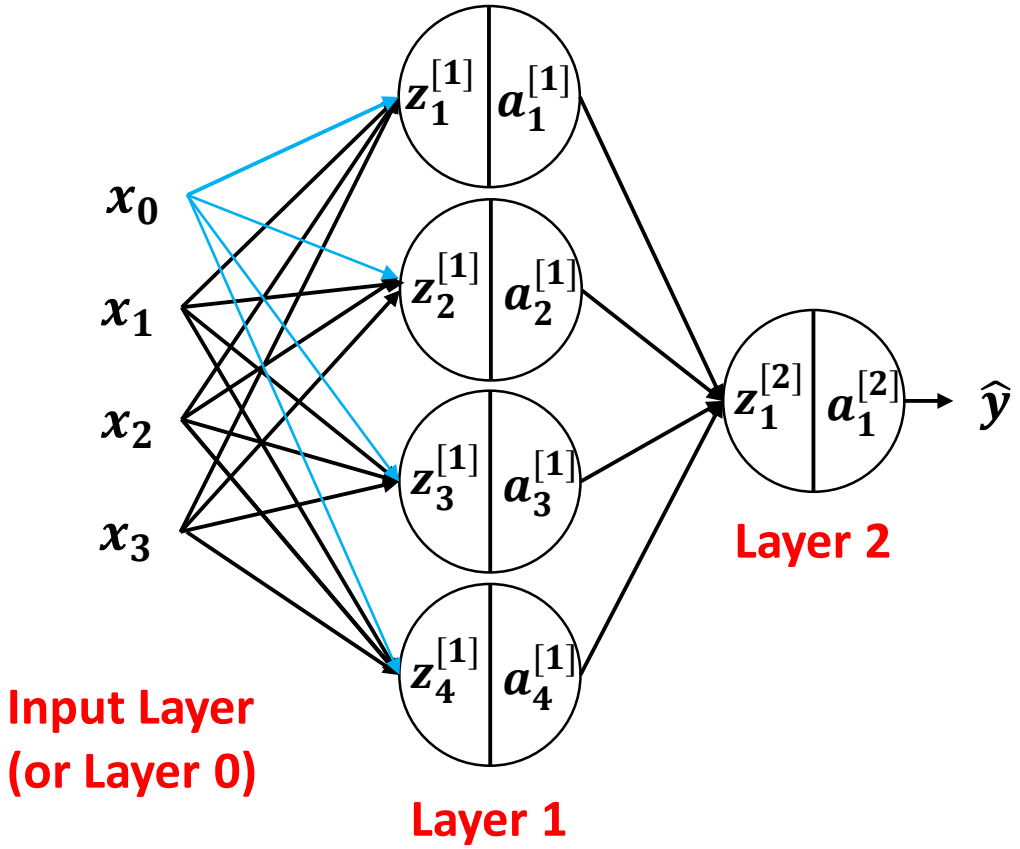


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \\ w_{14}^{[1]} & w_{24}^{[1]} & w_{34}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_1^{[1]} \\ \boldsymbol{b}_2^{[1]} \\ \boldsymbol{b}_3^{[1]} \\ \boldsymbol{b}_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

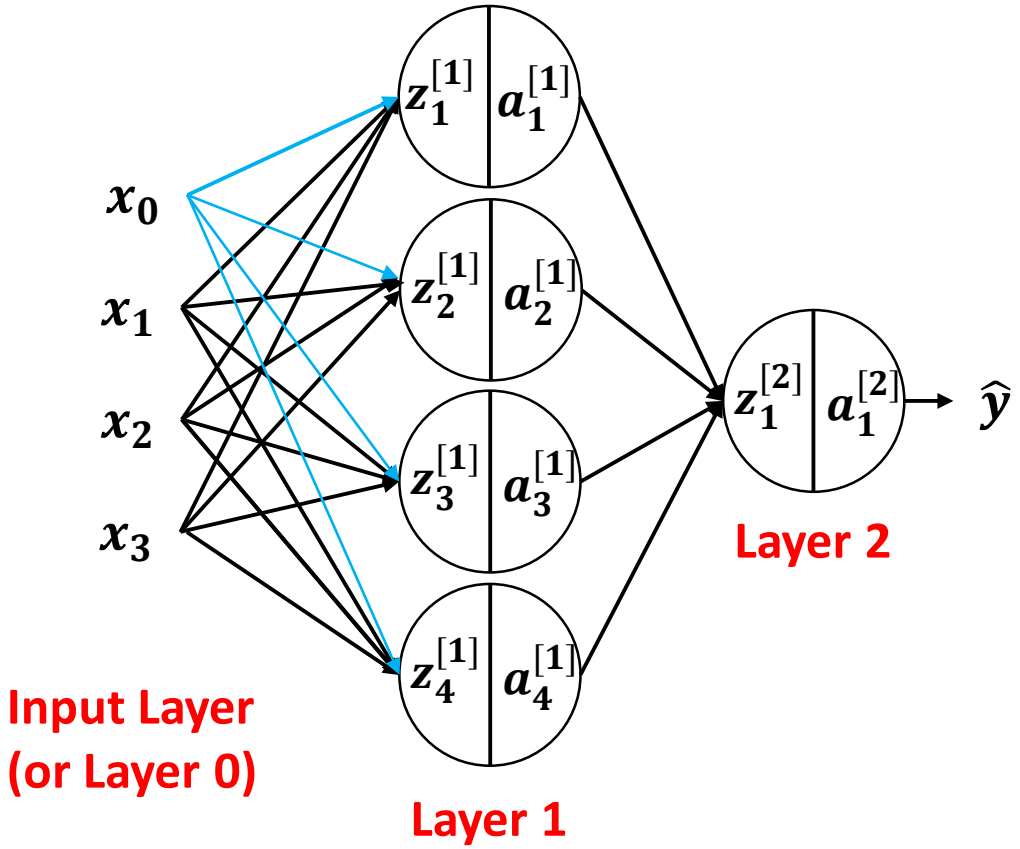


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



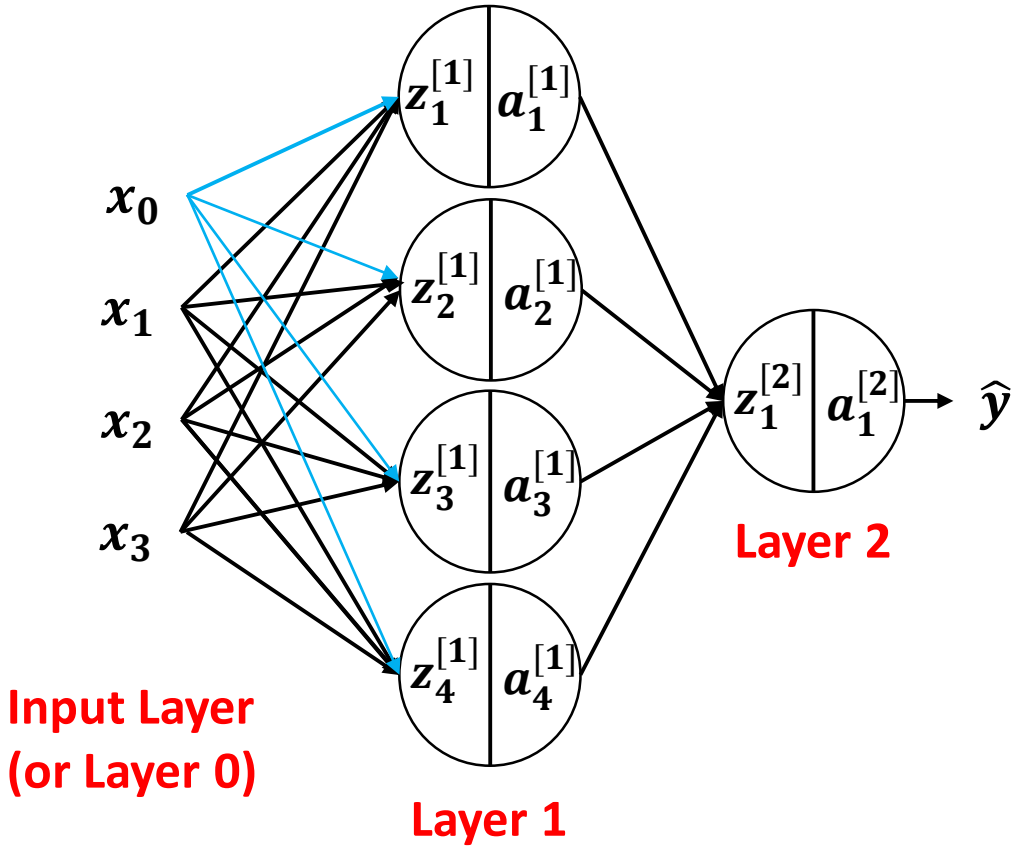
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

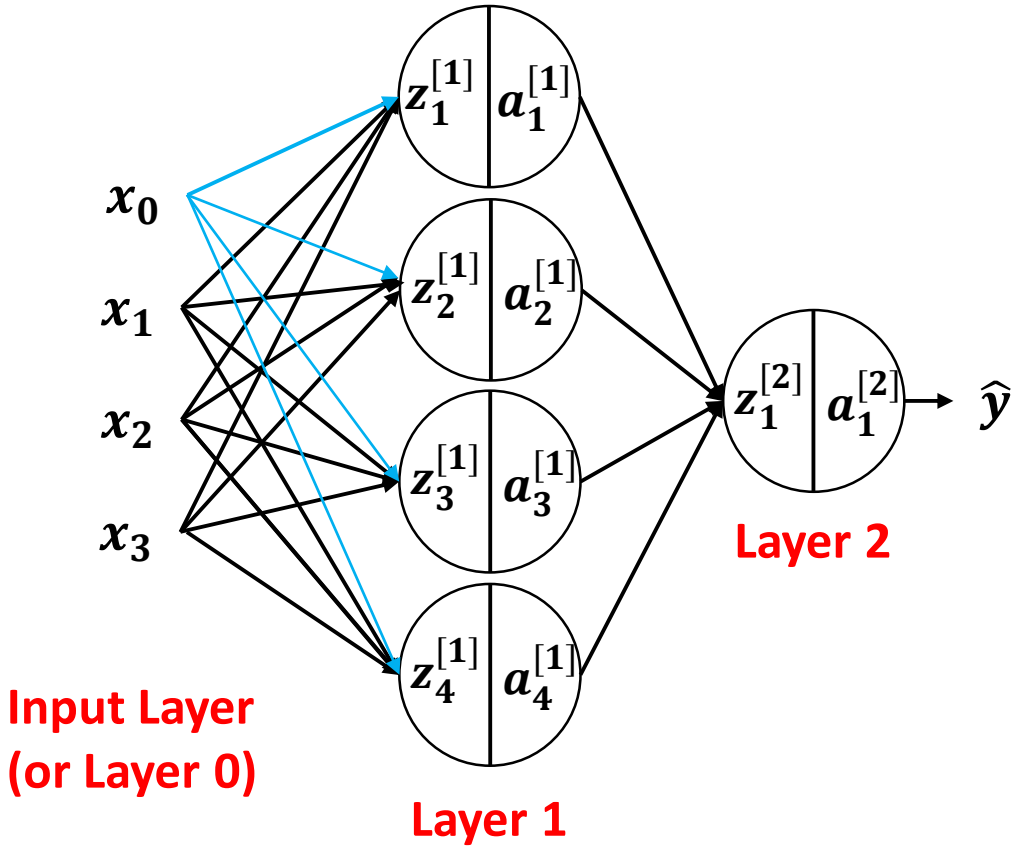
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]} \\ w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]} \\ w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]} \end{bmatrix}$$

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

$z_1^{[1]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

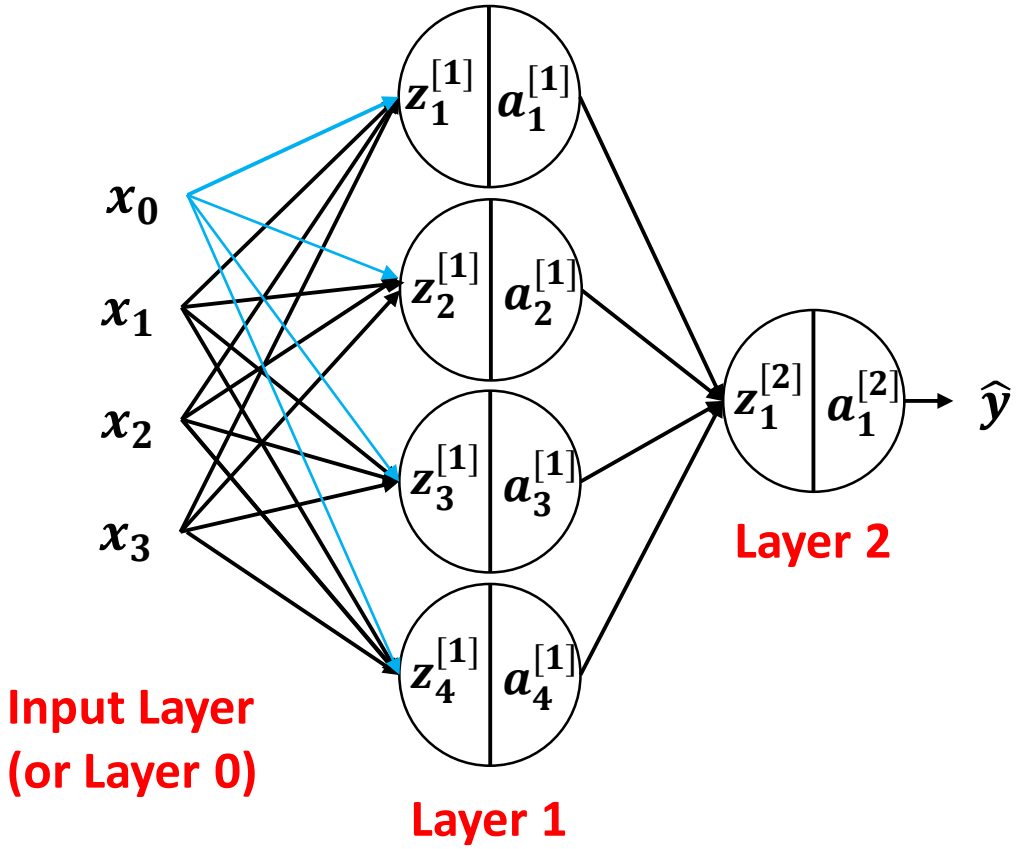
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]} \\ w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]} \\ w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

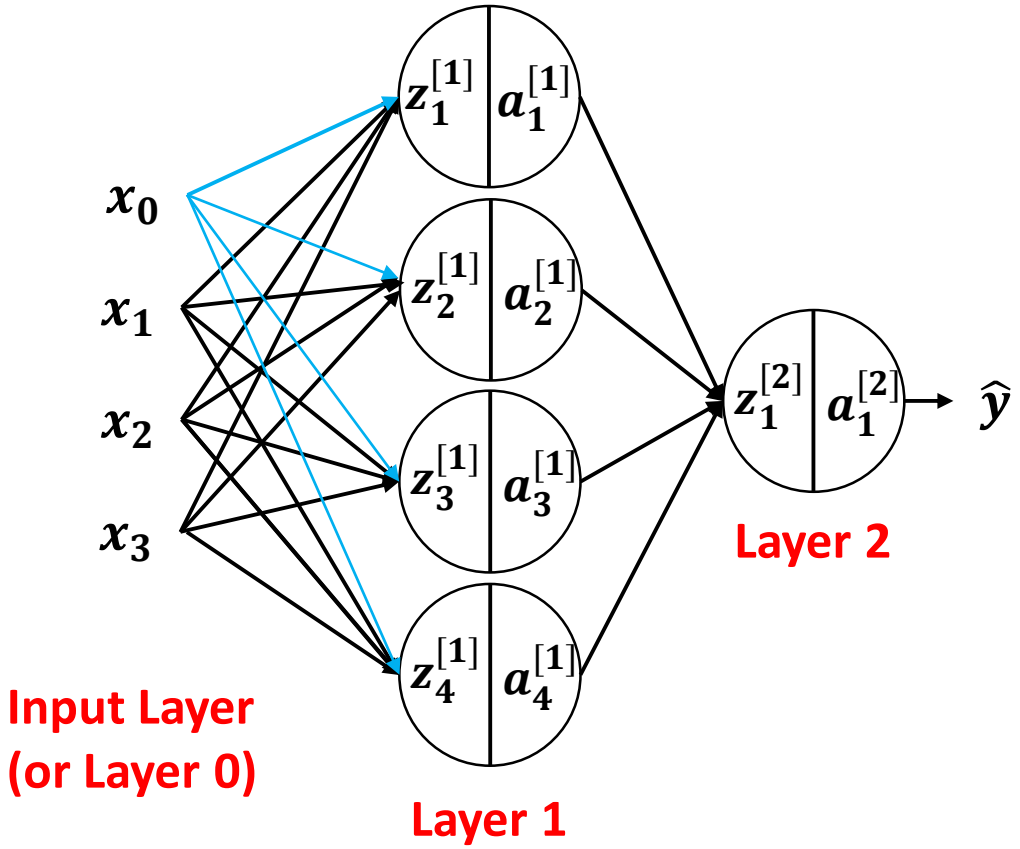
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]}} \\ w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]} \\ w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]} \end{bmatrix}$$

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

$$z_2^{[1]}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

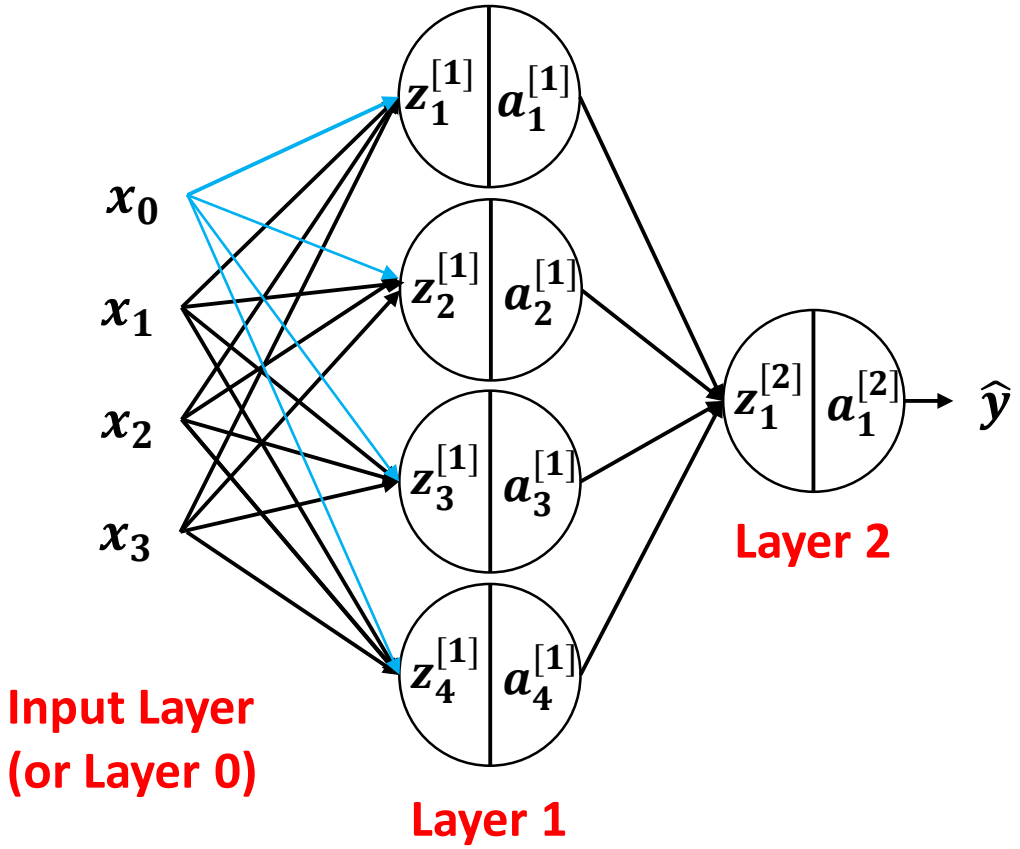
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]}} \\ w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]} \\ w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

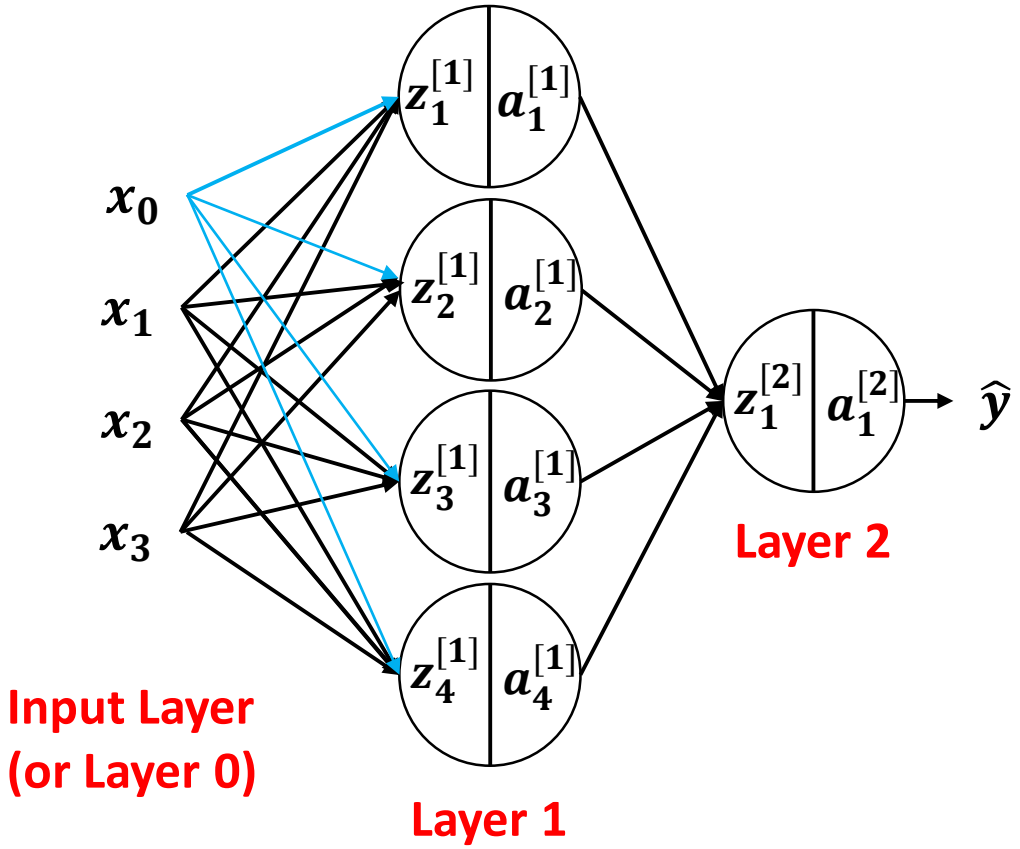
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]} \\ \boxed{w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]}} \\ w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]} \end{bmatrix}$$

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

$$z_3^{[1]}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

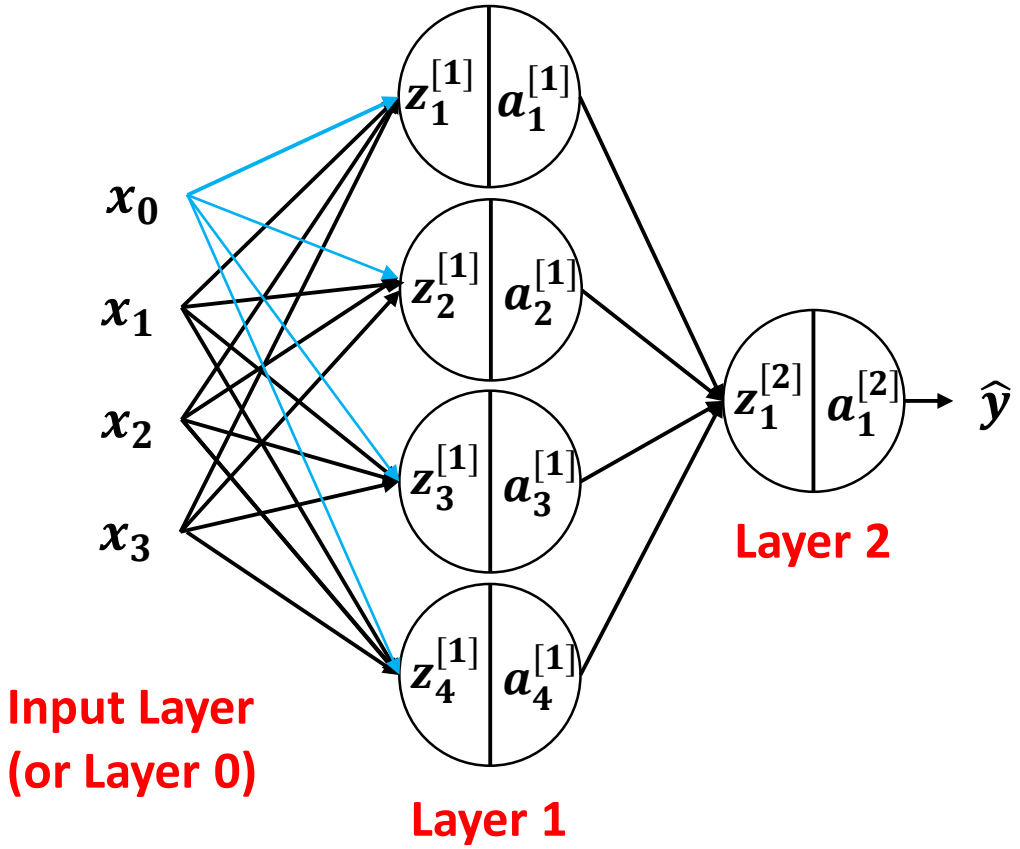
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

$$\boldsymbol{a}_3^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_3^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

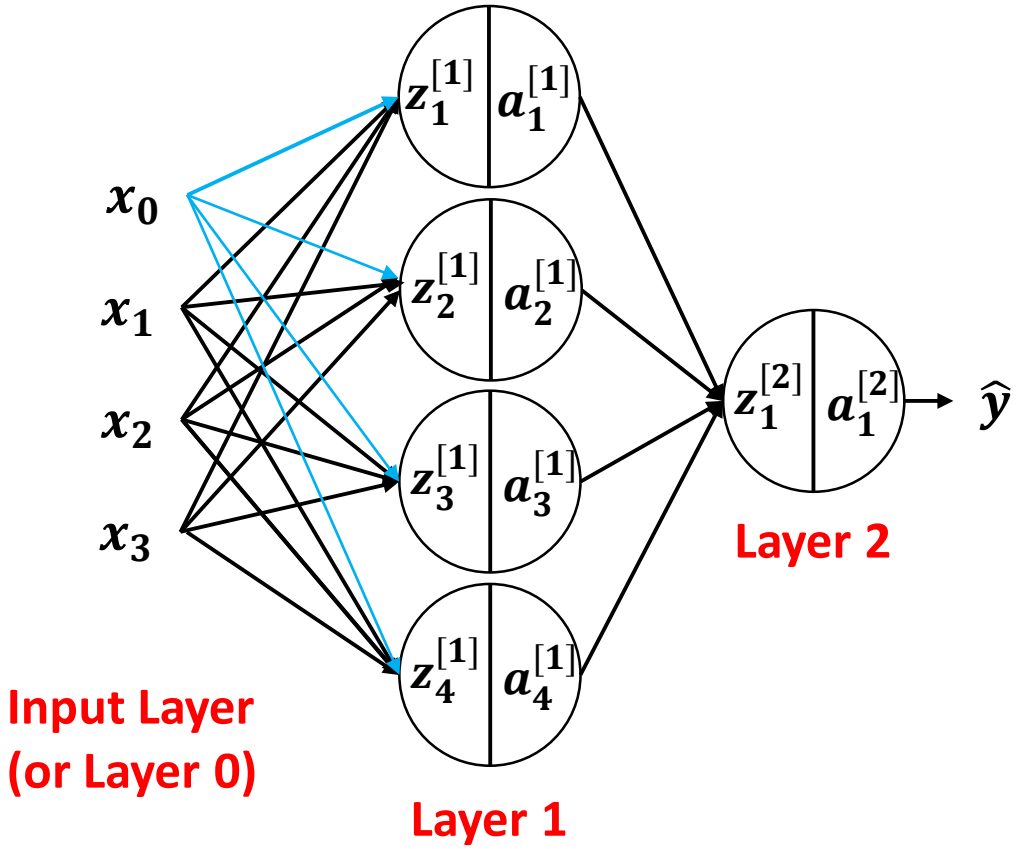
$$= \begin{bmatrix} w_{11}^{[1]} x_1 + w_{21}^{[1]} x_2 + w_{31}^{[1]} x_3 + b_1^{[1]} \\ w_{12}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{32}^{[1]} x_3 + b_2^{[1]} \\ w_{13}^{[1]} x_1 + w_{23}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]} \\ \boxed{w_{14}^{[1]} x_1 + w_{24}^{[1]} x_2 + w_{34}^{[1]} x_3 + b_4^{[1]}} \end{bmatrix}$$

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

$z_4^{[1]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

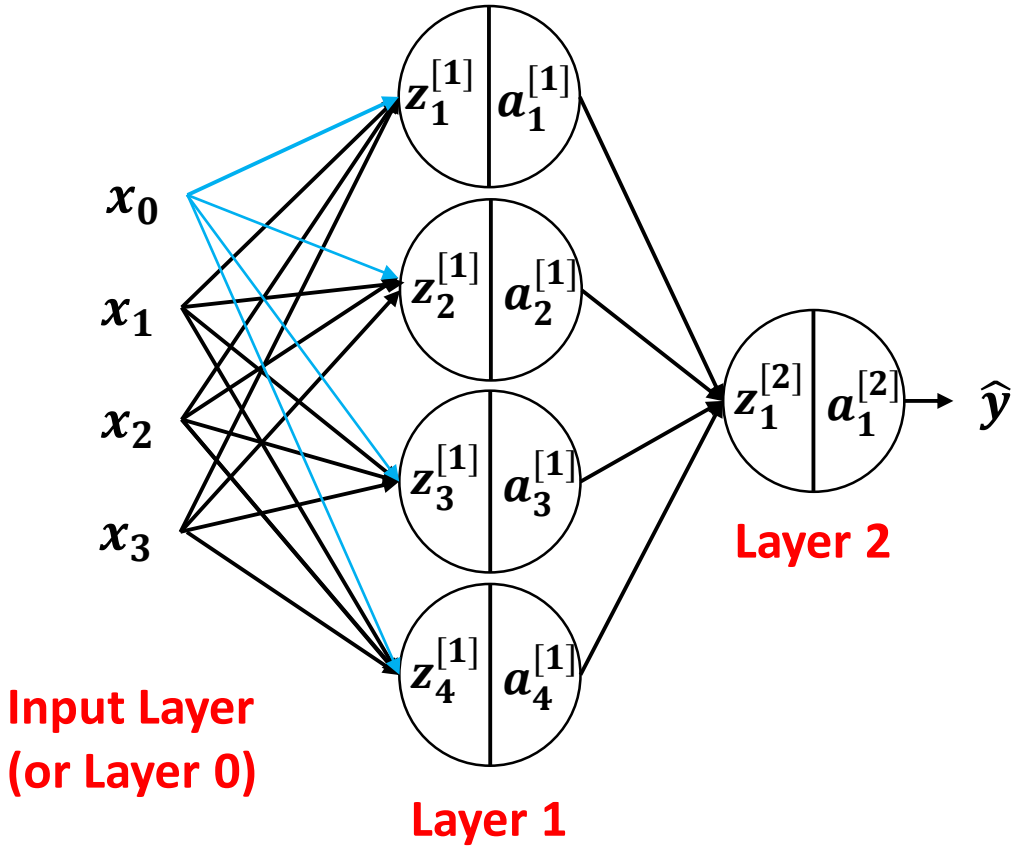
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ \boxed{w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]}} \end{bmatrix}$$

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

$$\boldsymbol{a}_4^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_4^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

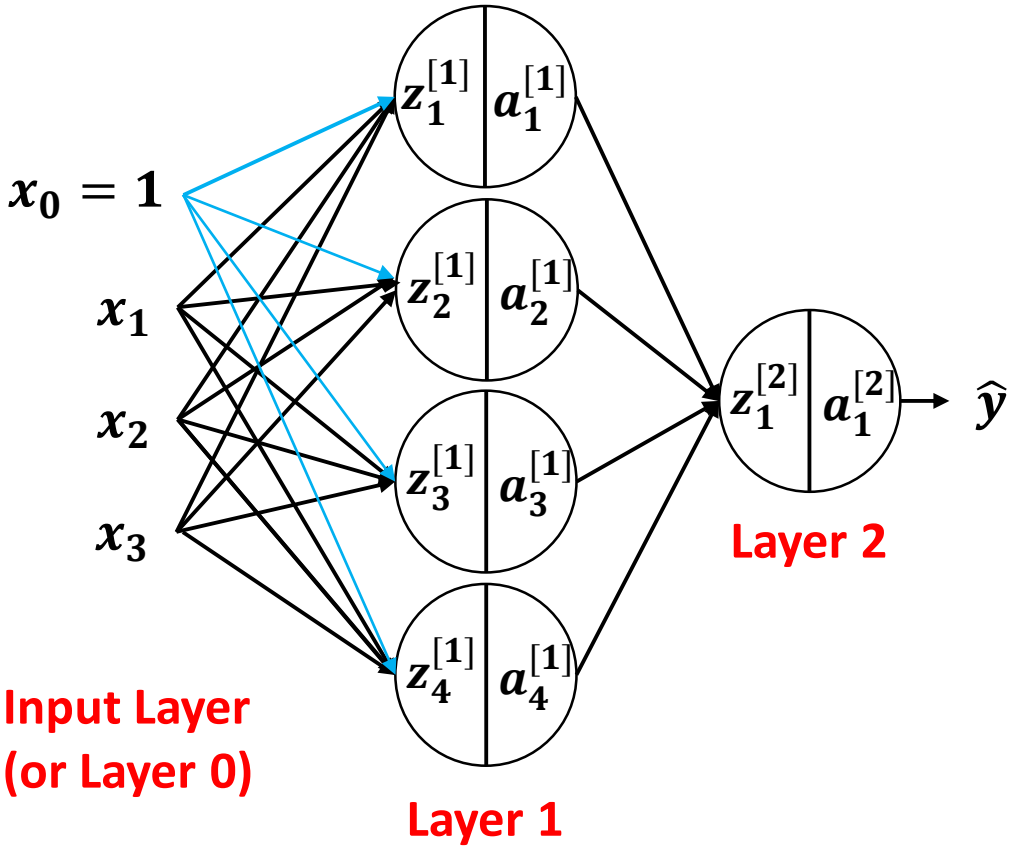


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + 1 \cdot b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + 1 \cdot b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + 1 \cdot b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + 1 \cdot b_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



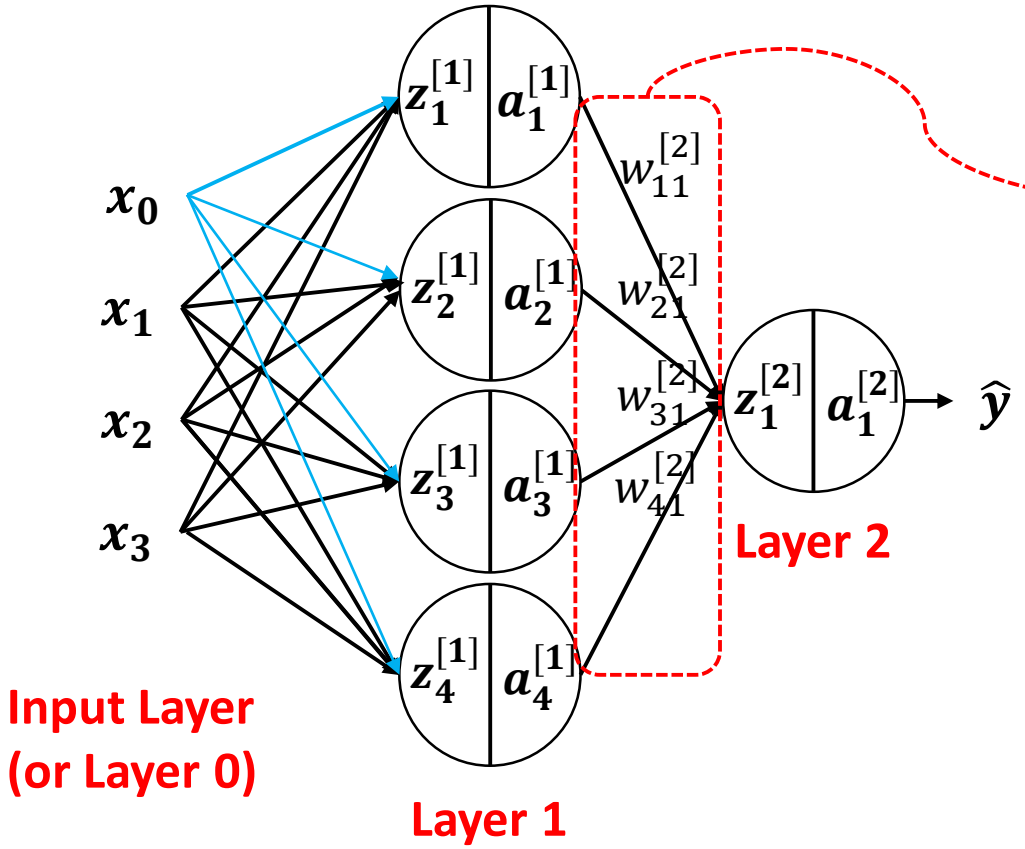
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + 1 \cdot b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + 1 \cdot b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + 1 \cdot b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + 1 \cdot b_4^{[1]} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{b}_1^{[1]} & w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ \boldsymbol{b}_2^{[1]} & w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ \boldsymbol{b}_3^{[1]} & w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \\ \boldsymbol{b}_4^{[1]} & w_{14}^{[1]} & w_{24}^{[1]} & w_{34}^{[1]} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

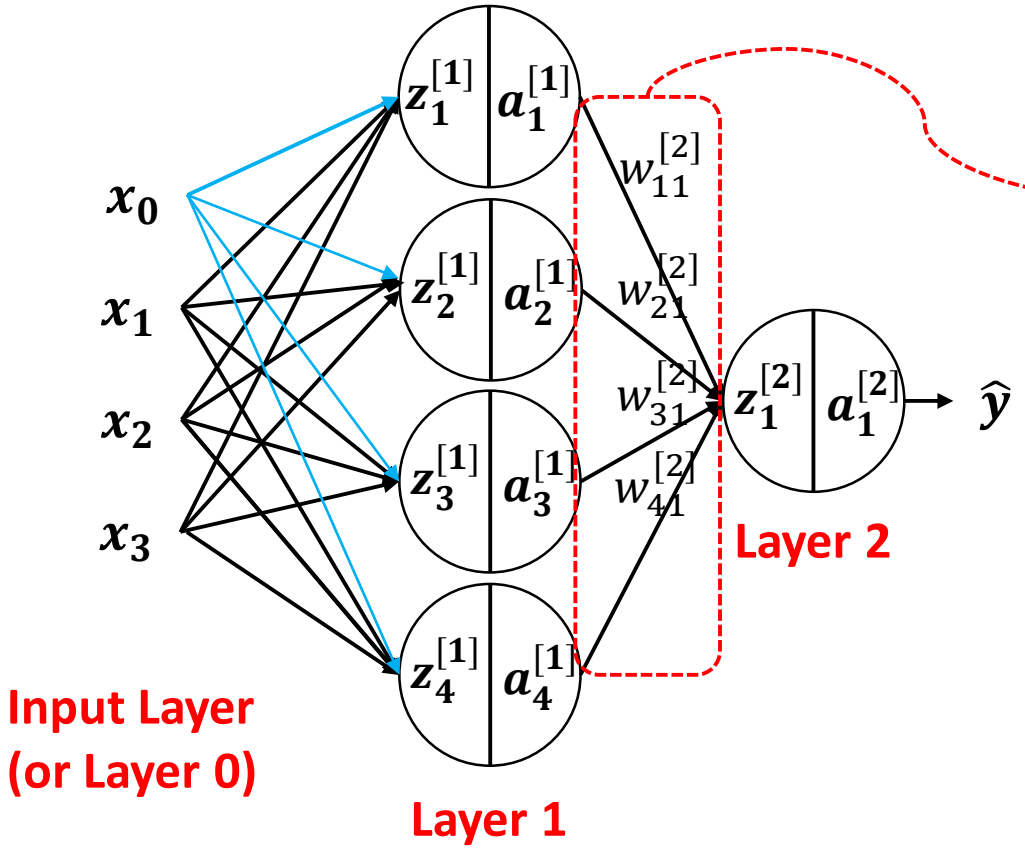


$$\boldsymbol{w}^{[2]} = \begin{bmatrix} w_{11}^{[2]} \\ w_{21}^{[2]} \\ w_{31}^{[2]} \\ w_{41}^{[2]} \end{bmatrix}$$

Dimension of $\boldsymbol{w}^{[2]}$= (4, 1)

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

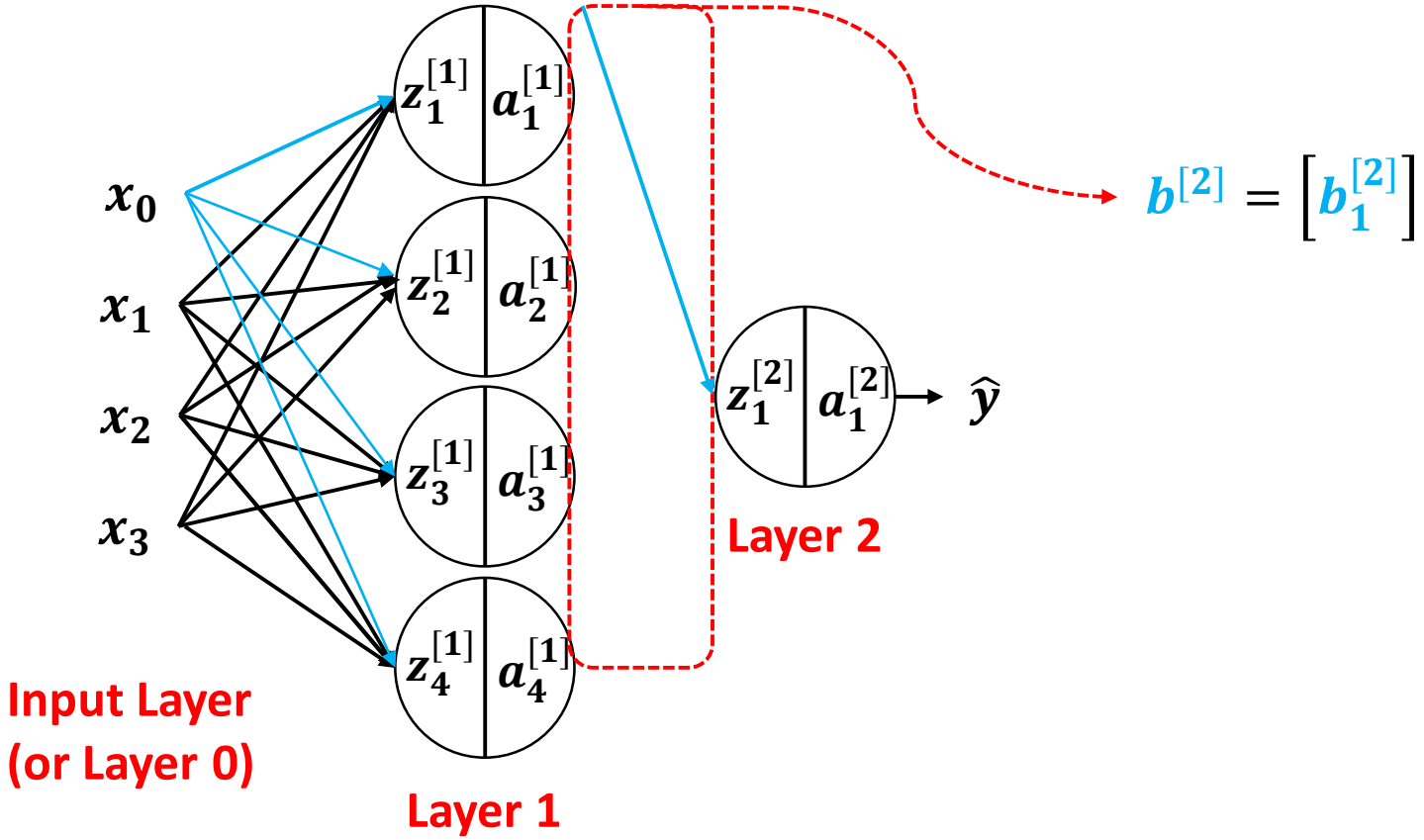


$$\boldsymbol{w}^{[2]^T} = \left[w_{11}^{[2]}\ w_{21}^{[2]}\ w_{31}^{[2]}\ w_{41}^{[2]}\right]$$

Dimension of $\boldsymbol{w}^{[2]^T}$ = (1, 4)

Vectorizing Input and All Variables

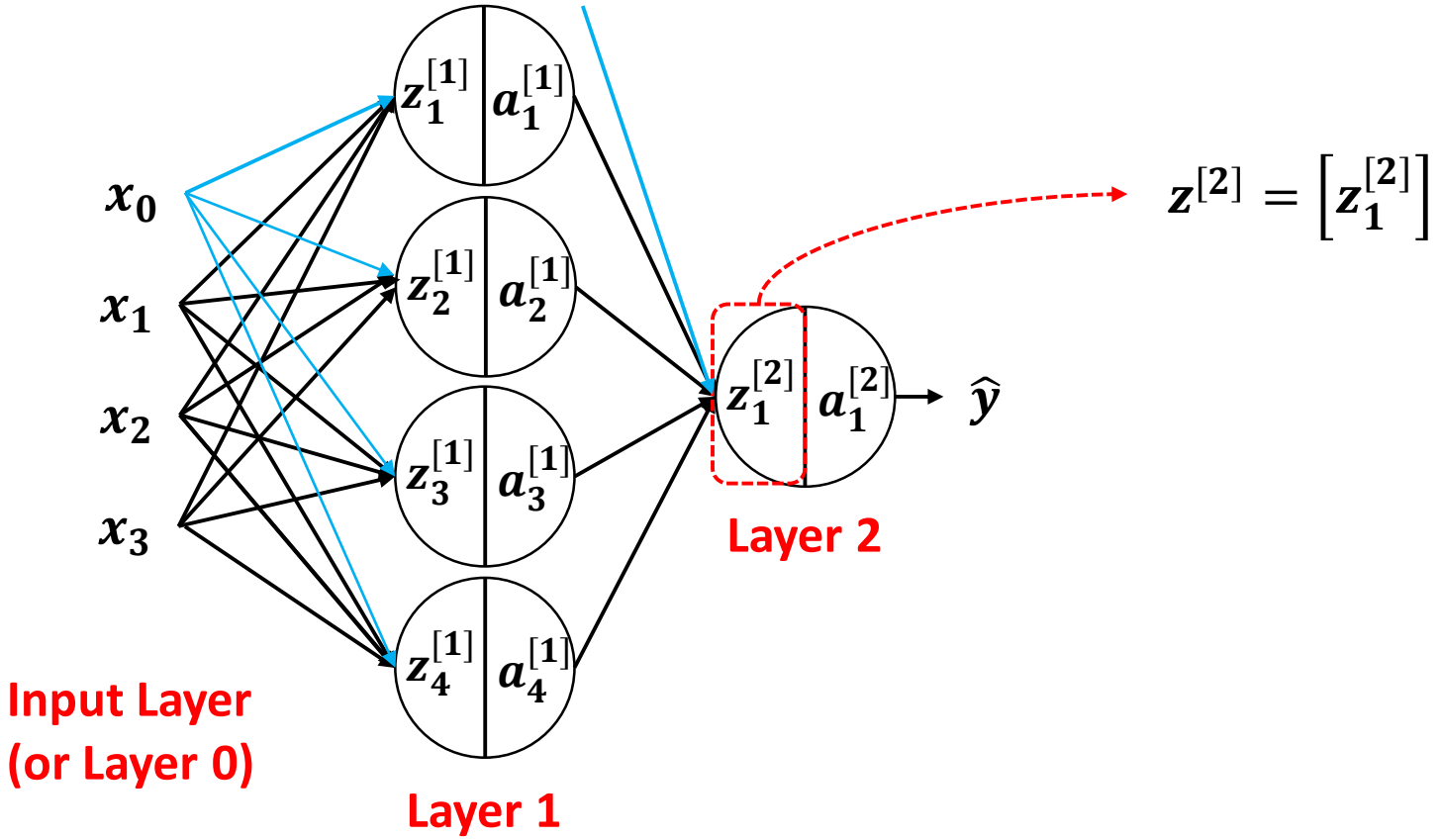
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$b^{[2]} = \left[b_1^{[2]}\right]$$

Vectorizing Input and All Variables

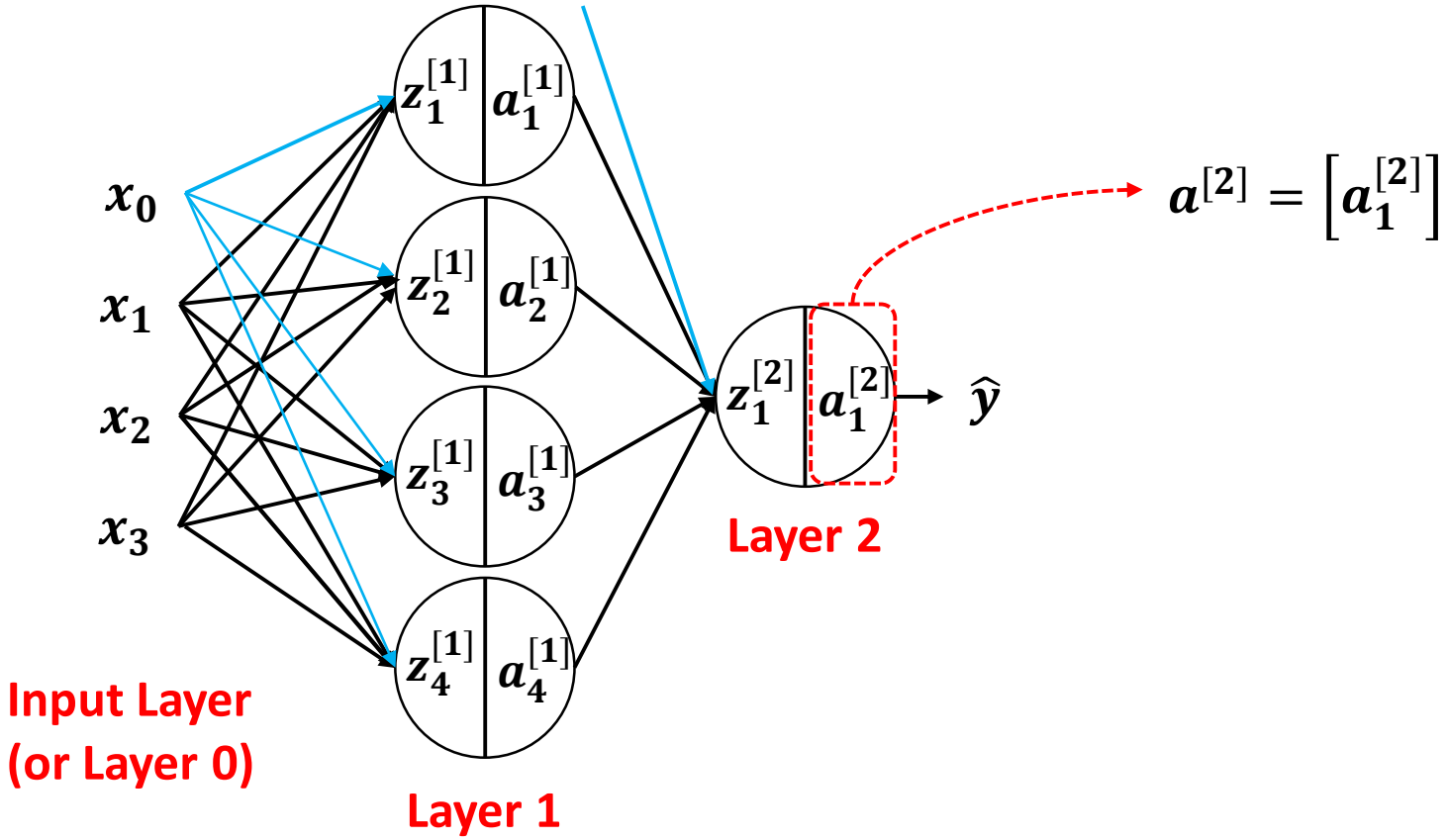
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[2]} = \left[z_1^{[2]}\right]$$

Vectorizing Input and All Variables

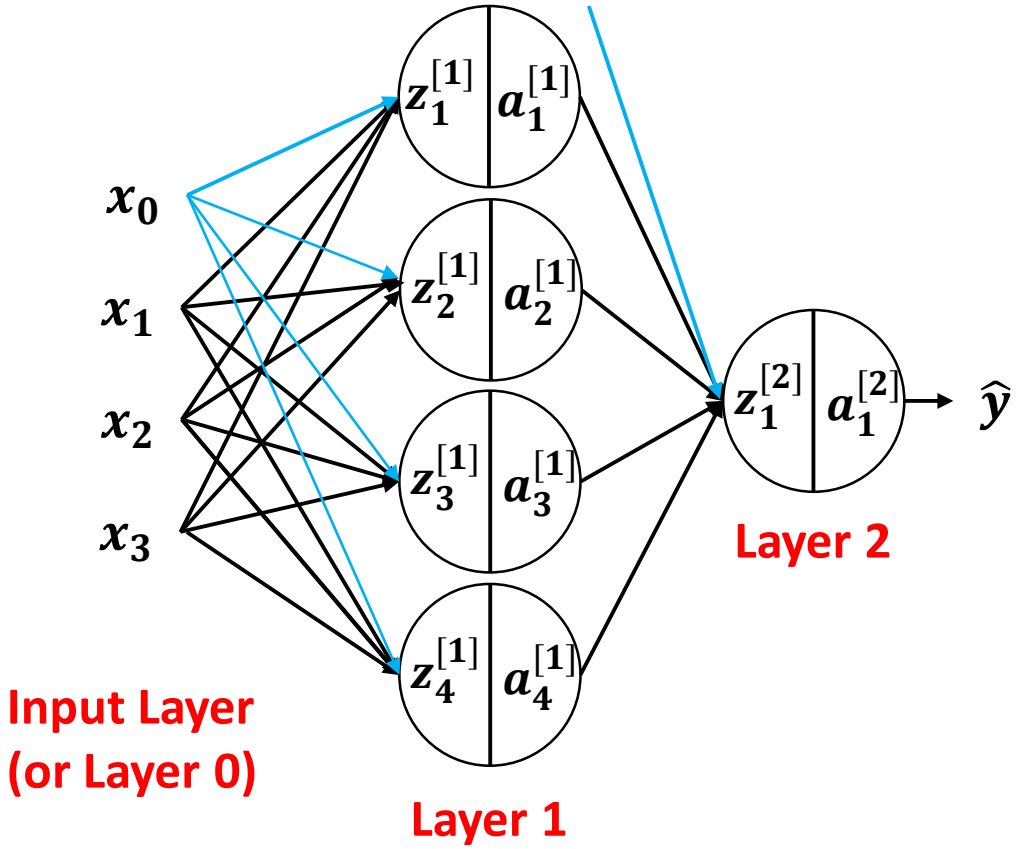
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$a^{[2]} = \left[a_1^{[2]}\right]$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



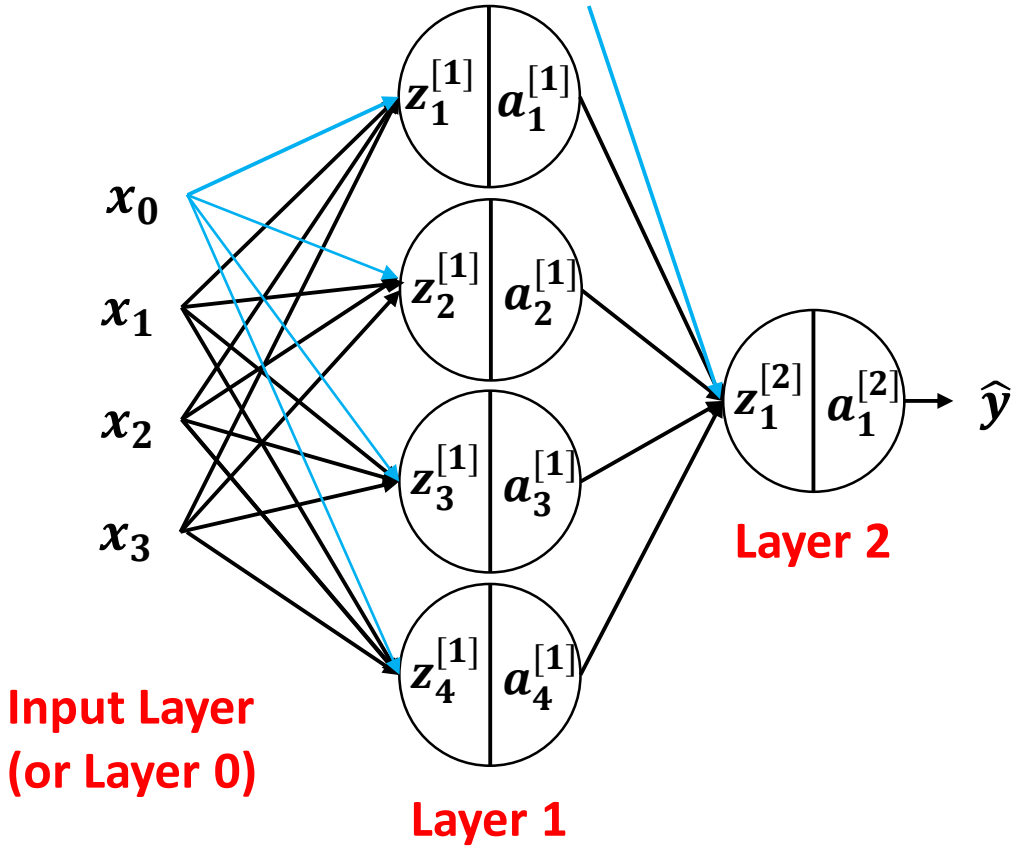
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} & w_{31}^{[2]} & w_{41}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



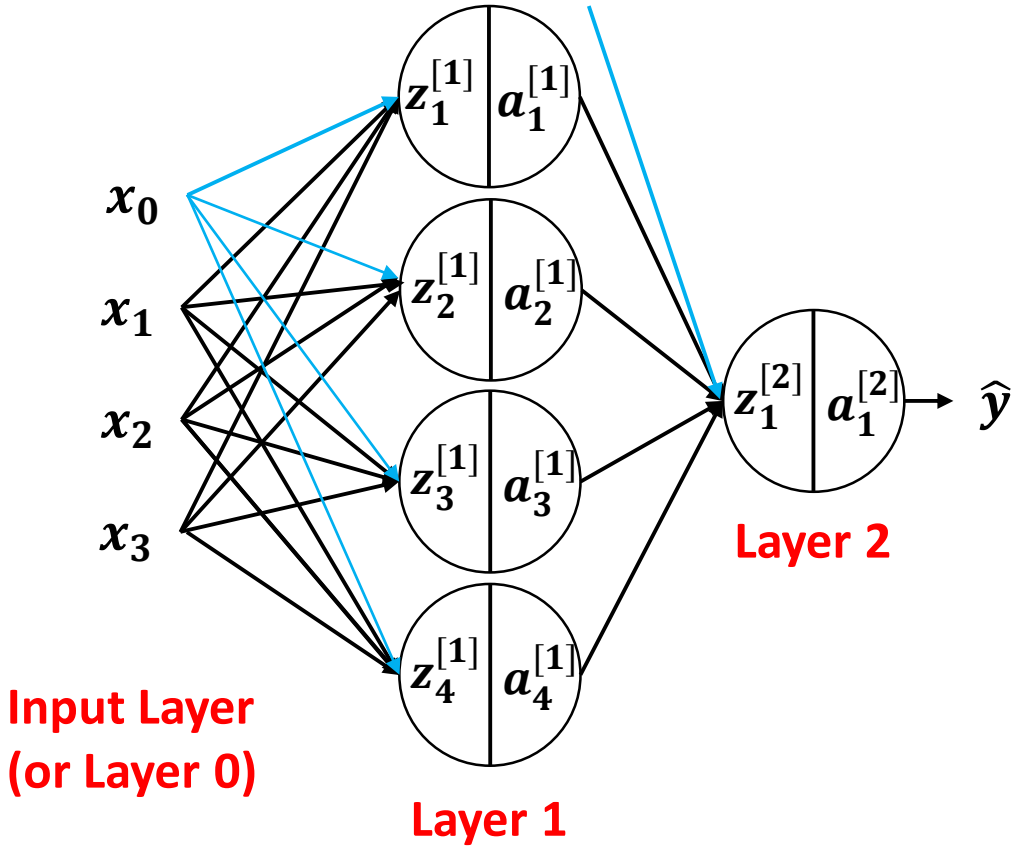
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \left[w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]}\right] + \left[b_1^{[2]}\right]$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

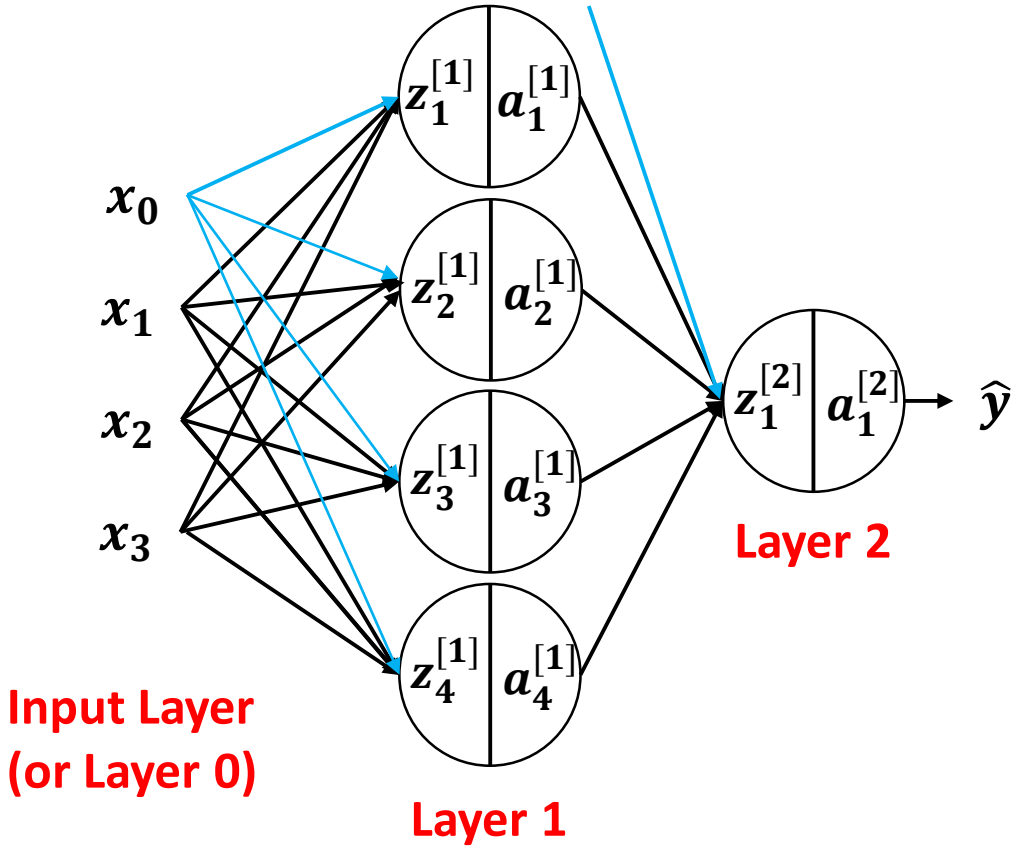
$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \left[w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]} + b_1^{[2]}\right]$$

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

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[2]} = w^{[2]^T} x + b^{[2]}$$

$$= w^{[2]^T} a^{[1]} + b^{[2]}$$

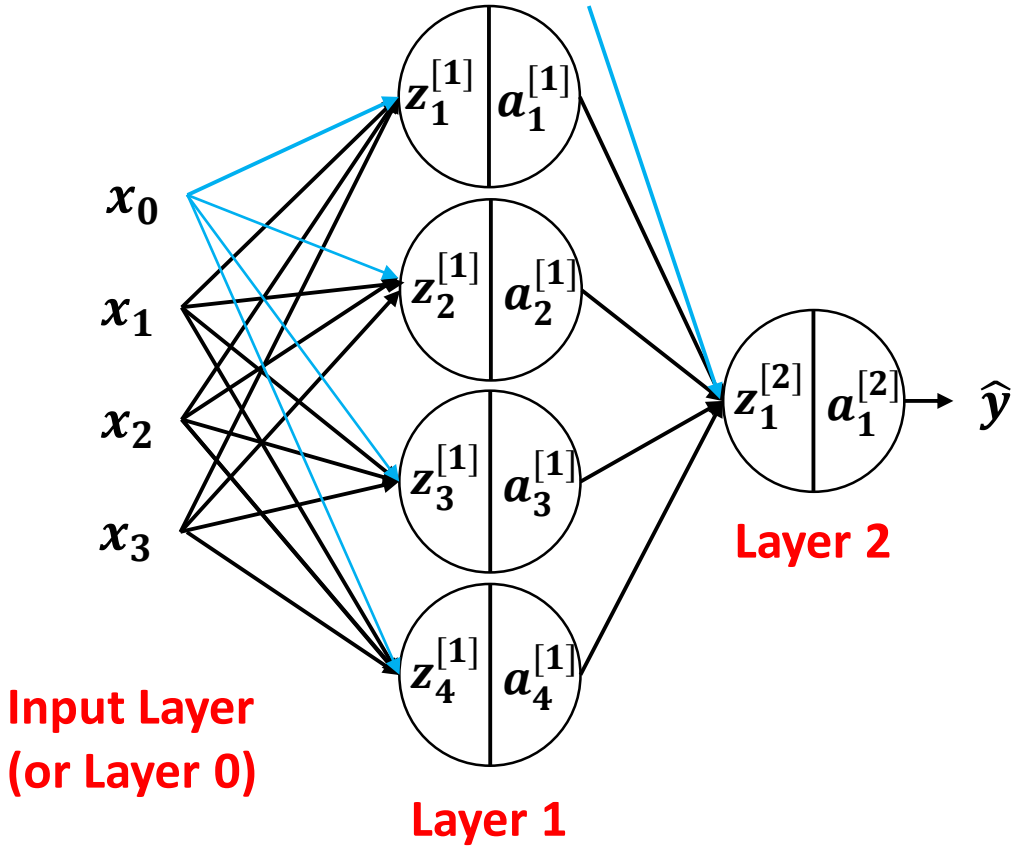
$$= \boxed{w_{11}^{[2]} a_1^{[1]} + w_{21}^{[2]} a_2^{[1]} + w_{31}^{[2]} a_3^{[1]} + w_{41}^{[2]} a_4^{[1]} + b_1^{[2]}}$$

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

$$z_1^{[2]}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[2]} = w^{[2]^T} x + b^{[2]}$$

$$= w^{[2]^T} a^{[1]} + b^{[2]}$$

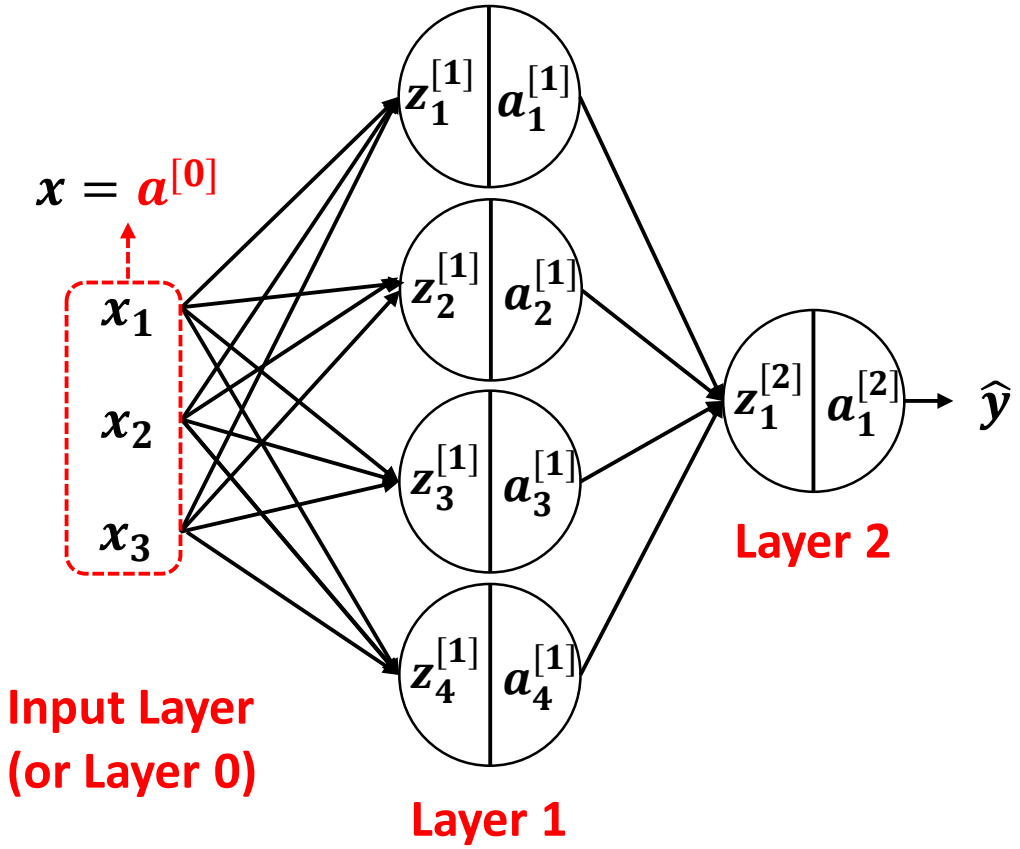
$$= \boxed{w_{11}^{[2]} a_1^{[1]} + w_{21}^{[2]} a_2^{[1]} + w_{31}^{[2]} a_3^{[1]} + w_{41}^{[2]} a_4^{[1]} + b_1^{[2]}}$$

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

$$a_1^{[2]} = \sigma(z_1^{[2]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} x + b^{[1]}$$

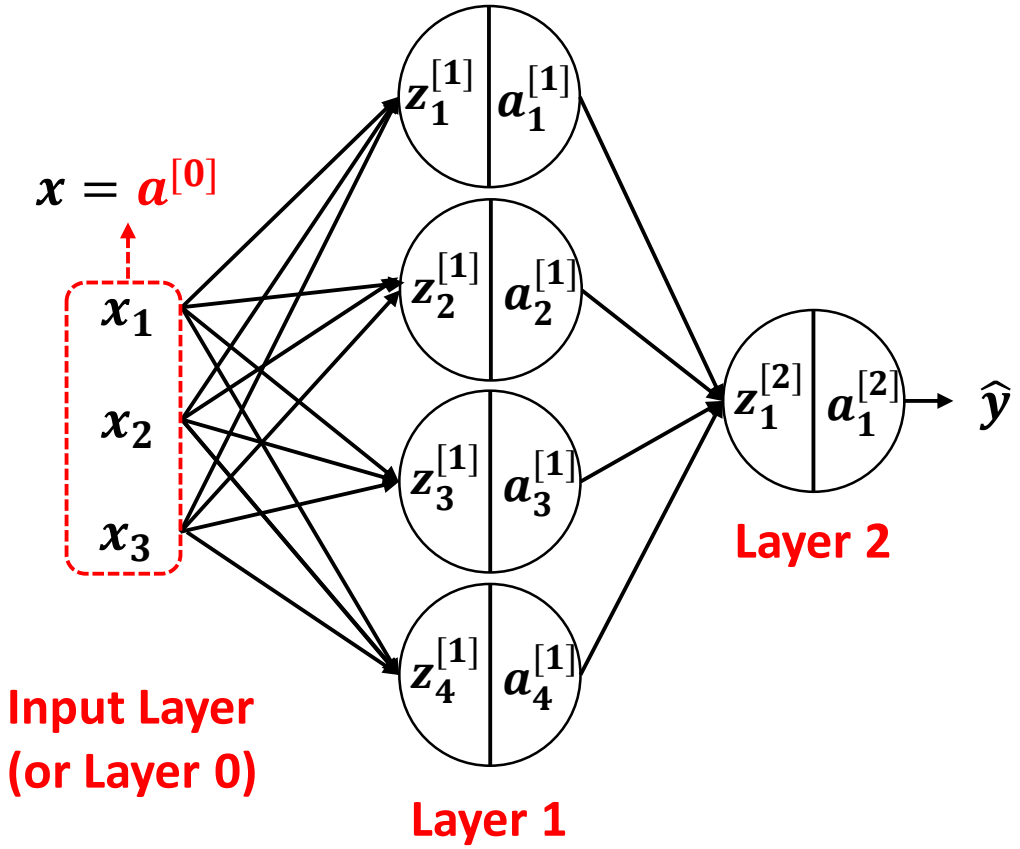
$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} a^{[0]} + b^{[1]}$$

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

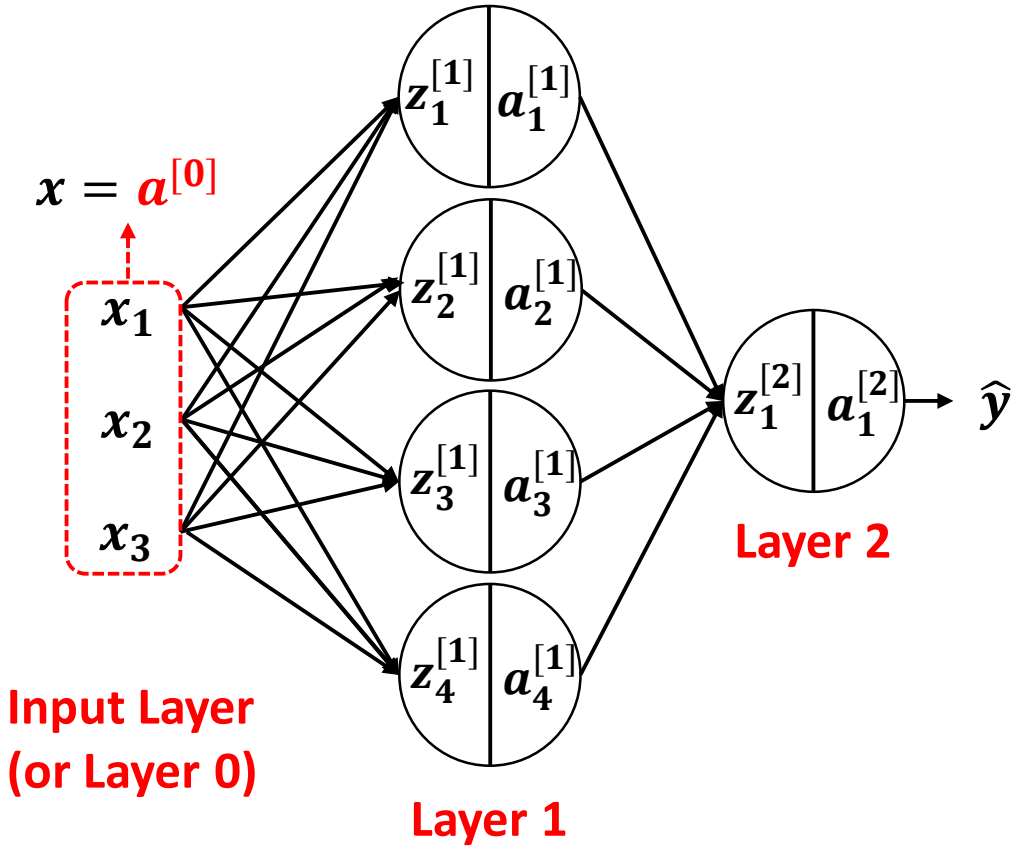
$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

But, this assumes only 1 training example!

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} a^{[0]} + b^{[1]}$$

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

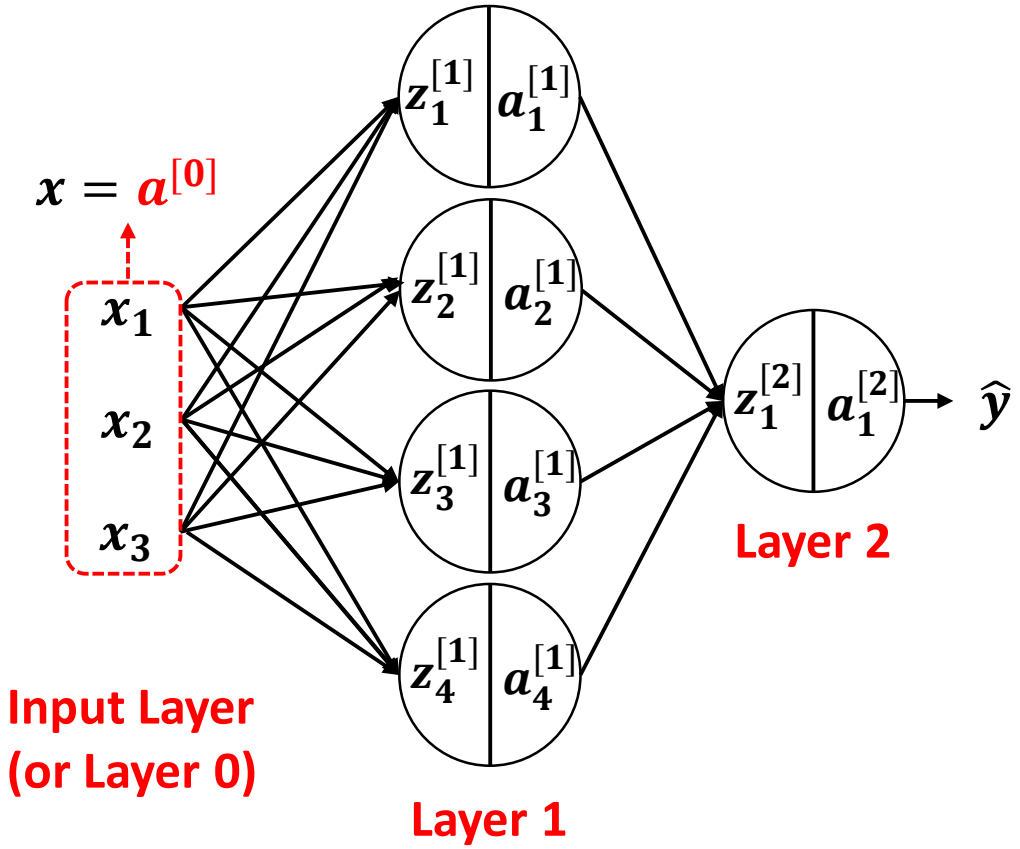
$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

How can we account for all the training examples?

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{for\ i = 1\ to\ n}:$$

Assuming n examples

$$z^{[1](i)} = w^{[1]^T(i)} a^{[0](i)} + b^{[1]}$$

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

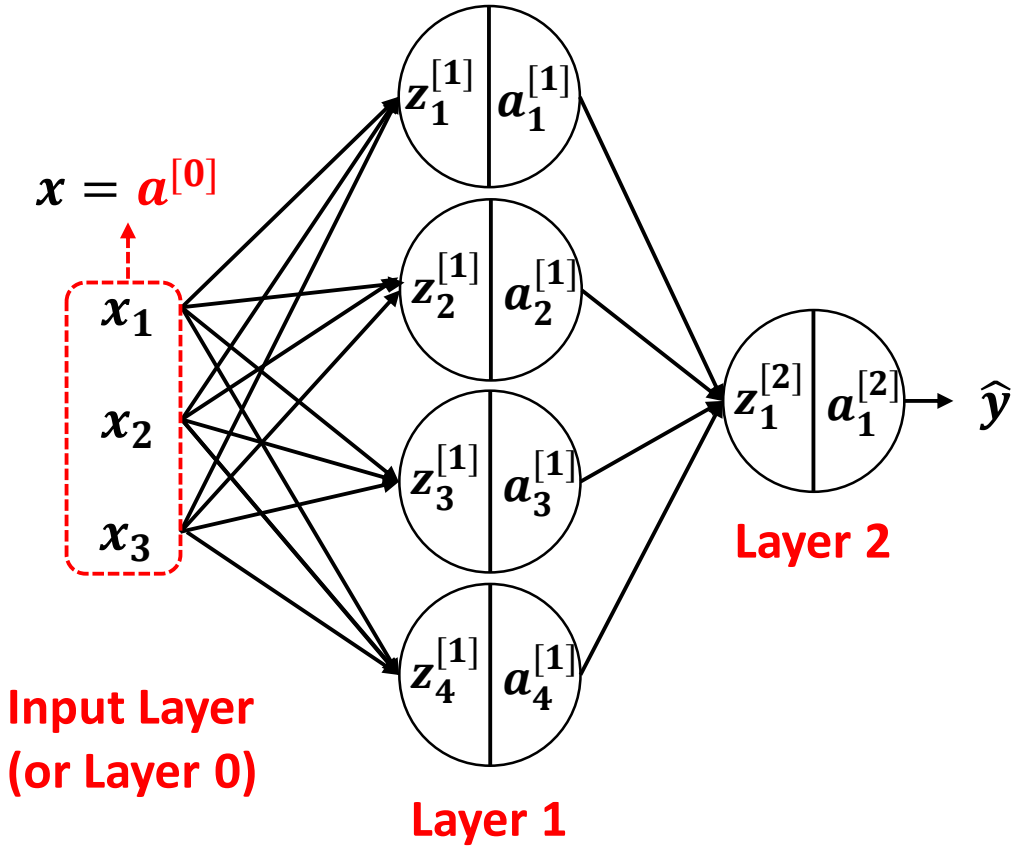
$$z^{[2](i)} = w^{[2]^T(i)} a^{[1](i)} + b^{[2]}$$

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

Refers to the ith example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



But, loops in general slow down programs; hence, it is better to further ***vectorize*** the implementation in order to avoid any loop, whenever possible

Assuming n examples

$for\ i = 1\ to\ n$:

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

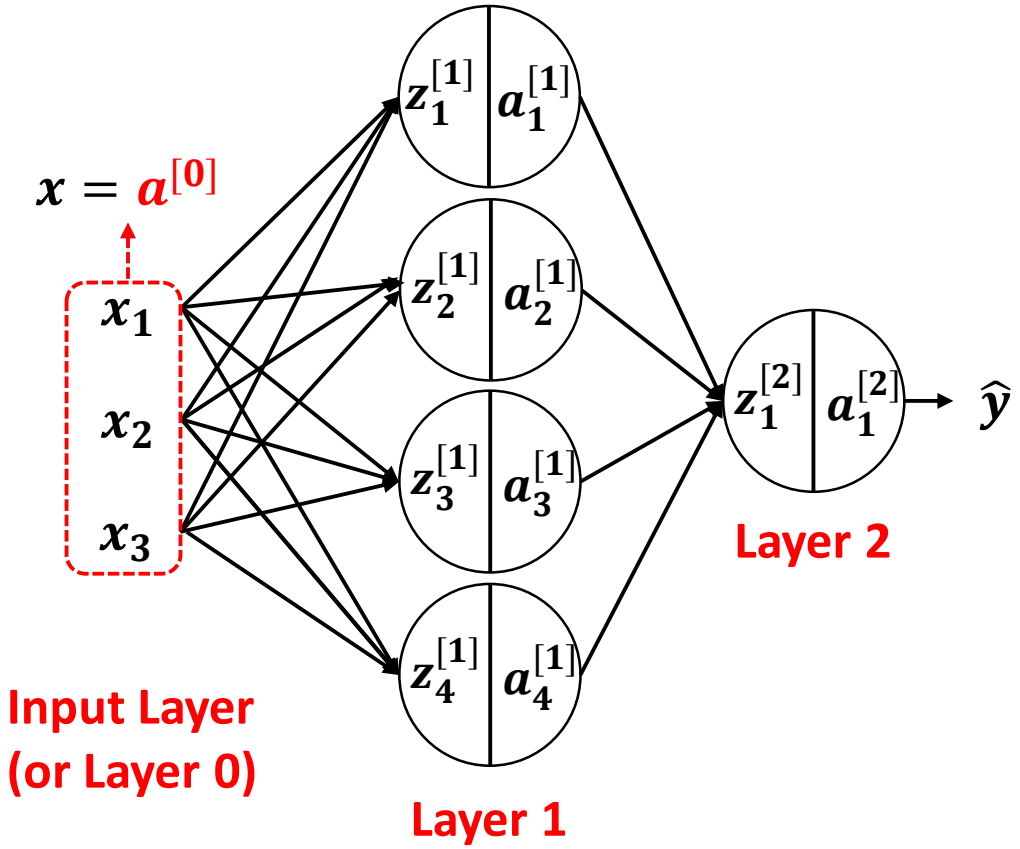
$z^{[2](i)} = w^{[2]T}a^{[1](i)} + b^{[2]}$

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

Refers to the ith example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



To this end, we can simply stack all $\boldsymbol{x}$ vectors (or $\boldsymbol{a}^{[0]}$ vectors), $\boldsymbol{z}$ vectors, and $\boldsymbol{a}$ vectors in different matrices of every layer!

Assuming n examples

$for\ i = 1\ to\ n$:

$z^{[1](i)} = w^{[1]^T}(i)\, a^{[0](i)} + b^{[1]}$

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

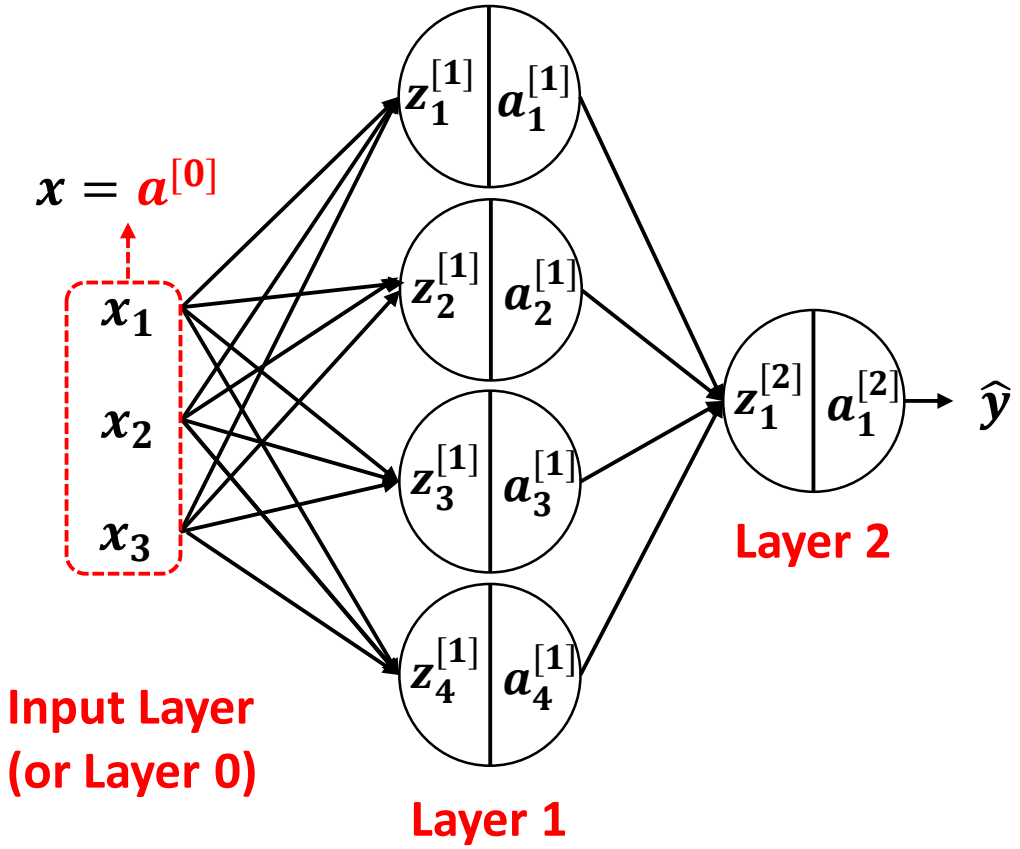
$z^{[2](i)} = w^{[2]}\, a^{[1](i)} + b^{[2]}$

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

Refers to the ith example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



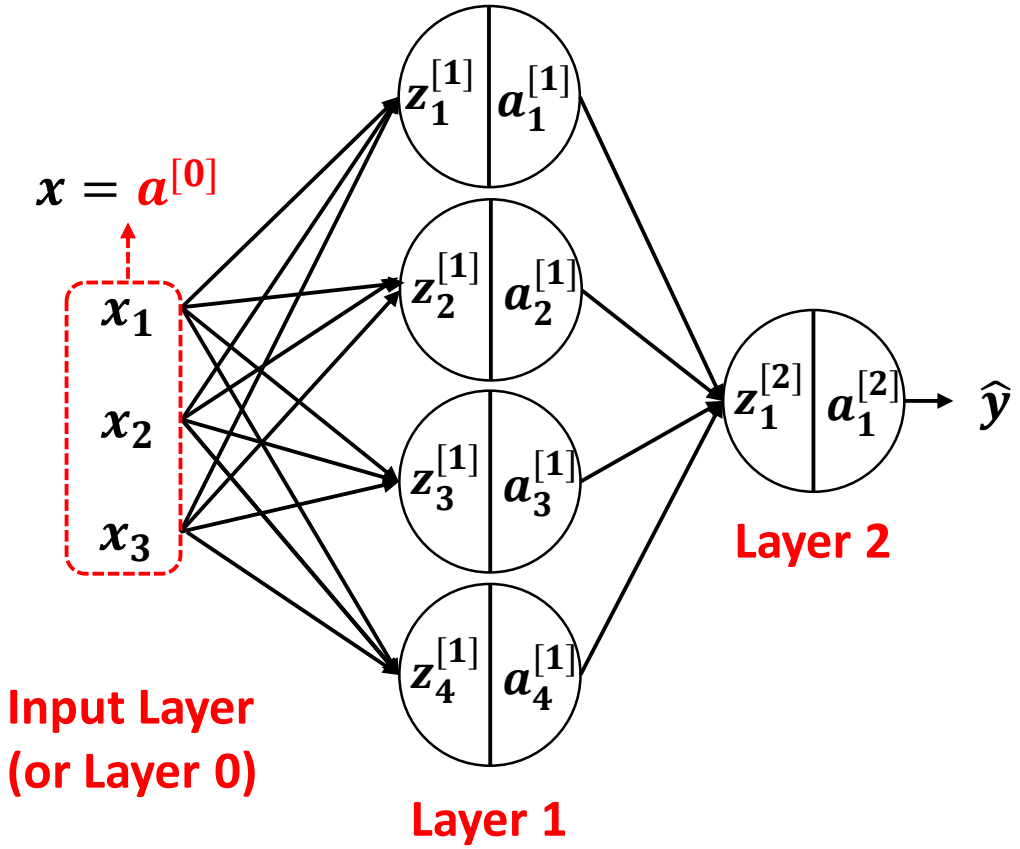
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} \\ x_3^{(1)} & x_3^{(2)} & & x_3^{(n)} \end{bmatrix}$$

Assuming n examples

This vector represents the *first* example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



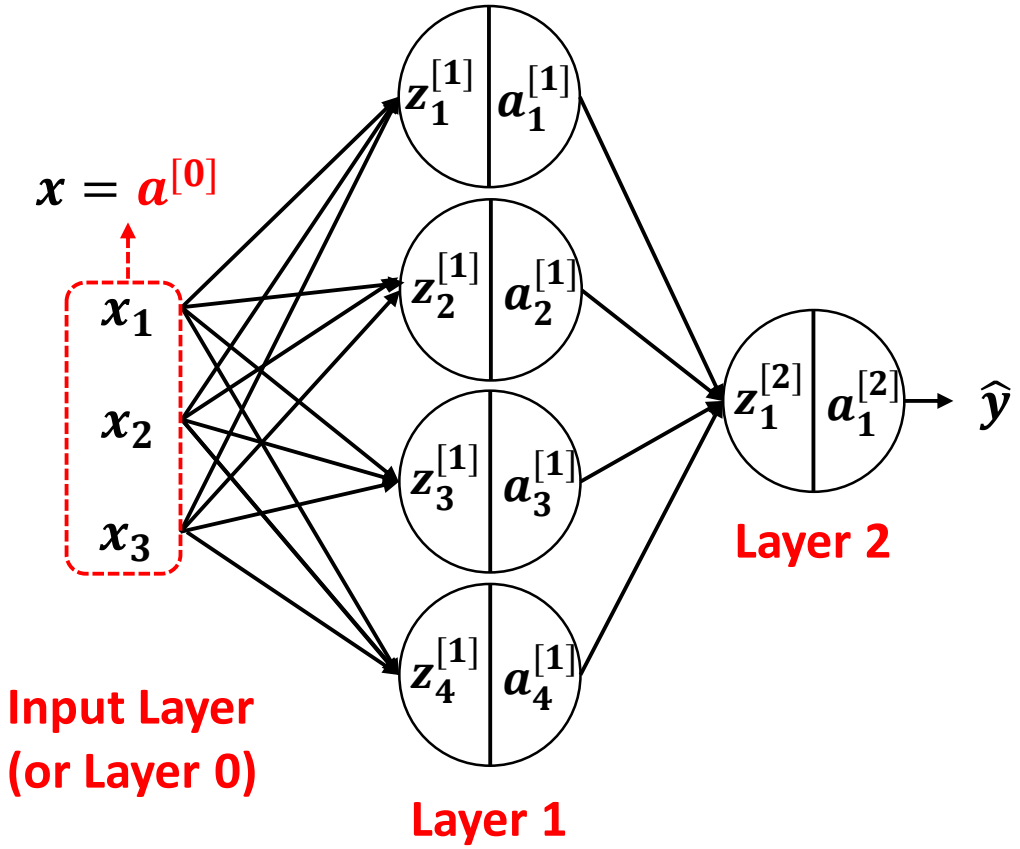
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & & x_2^{(n)} \\ x_3^{(1)} & x_3^{(2)} & & x_3^{(n)} \end{bmatrix}$$

Assuming n examples

This vector represents the *second* example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



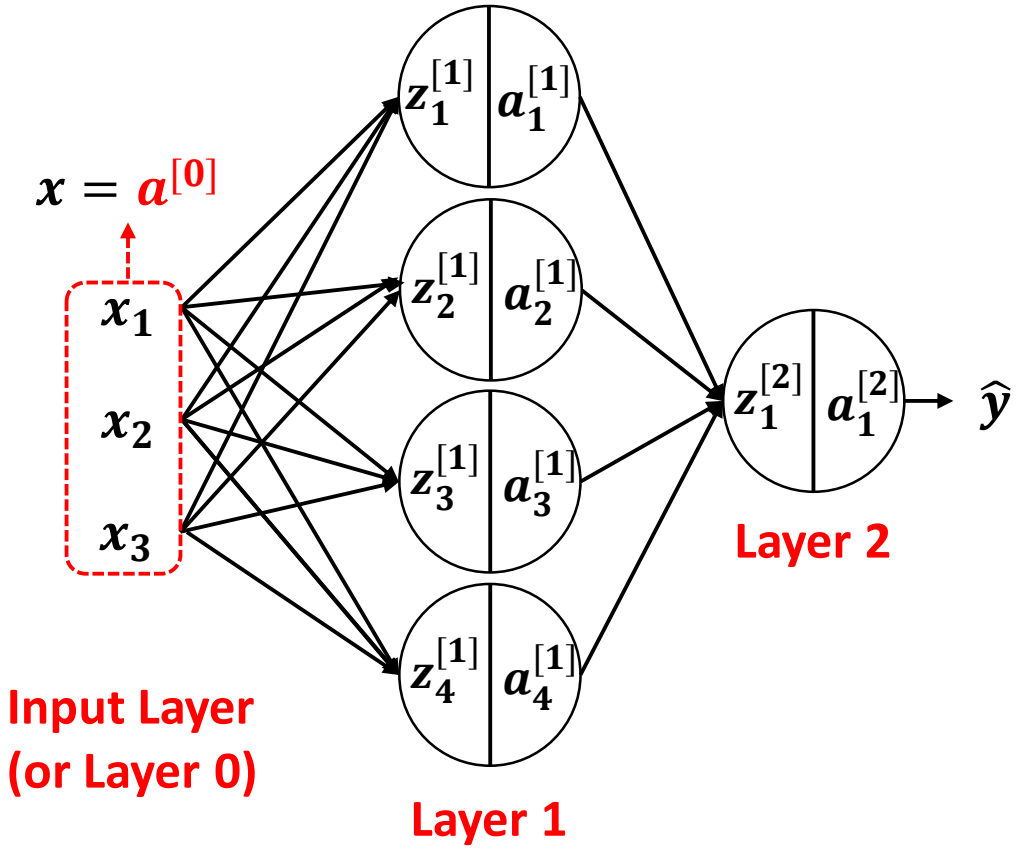
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(n)} \\ x_3^{(1)} & x_3^{(2)} & & x_3^{(n)} \end{bmatrix}$$

Assuming n examples

This vector represents the nth example in the training dataset

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

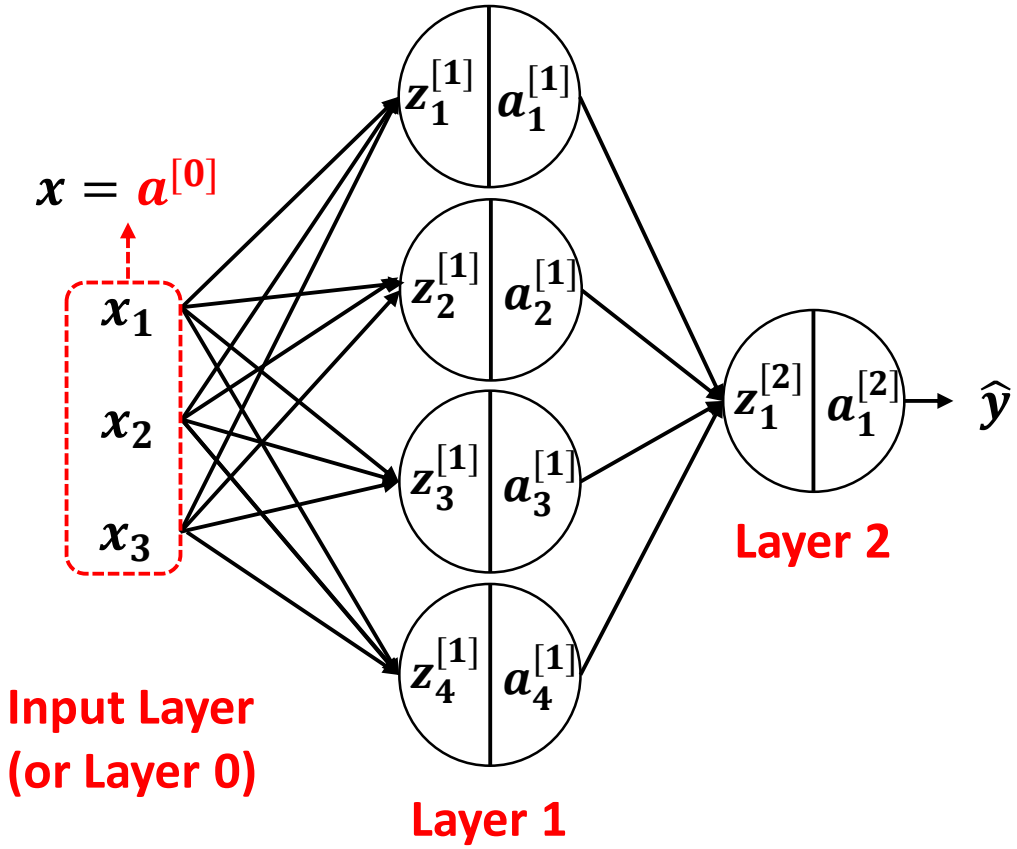


$$X = \begin{bmatrix} a_1^{[0](1)} & a_1^{[0](2)} & & a_1^{[0](n)} \\ a_2^{[0](1)} & a_2^{[0](2)} & \dots & a_2^{[0](n)} \\ a_3^{[0](1)} & a_3^{[0](2)} & & a_3^{[0](n)} \end{bmatrix}$$

If we denote x as $a^{[0]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

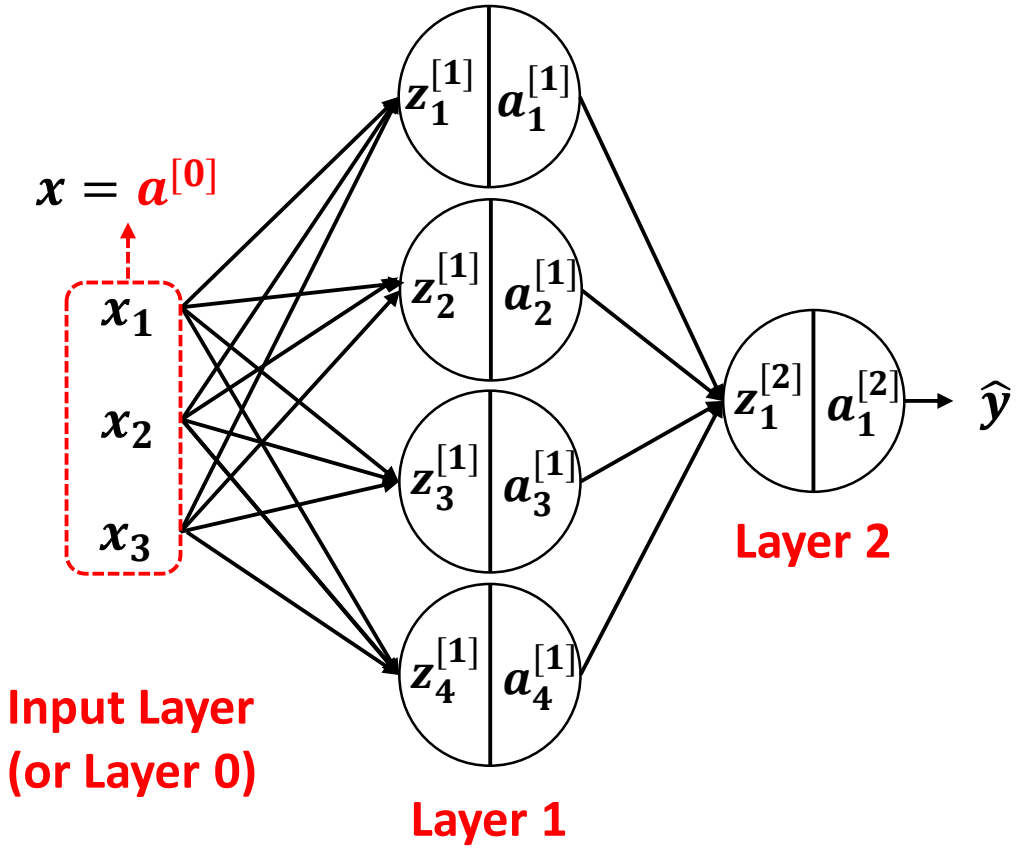


$$X = A^{[0]} = \begin{bmatrix} a_1^{[0](1)} & a_1^{[0](2)} & & a_1^{[0](n)} \\ a_2^{[0](1)} & a_2^{[0](2)} & \cdots & a_2^{[0](n)} \\ a_3^{[0](1)} & a_3^{[0](2)} & & a_3^{[0](n)} \end{bmatrix}$$

If we denote x as $a^{[0]}$

Vectorizing Input and All Variables

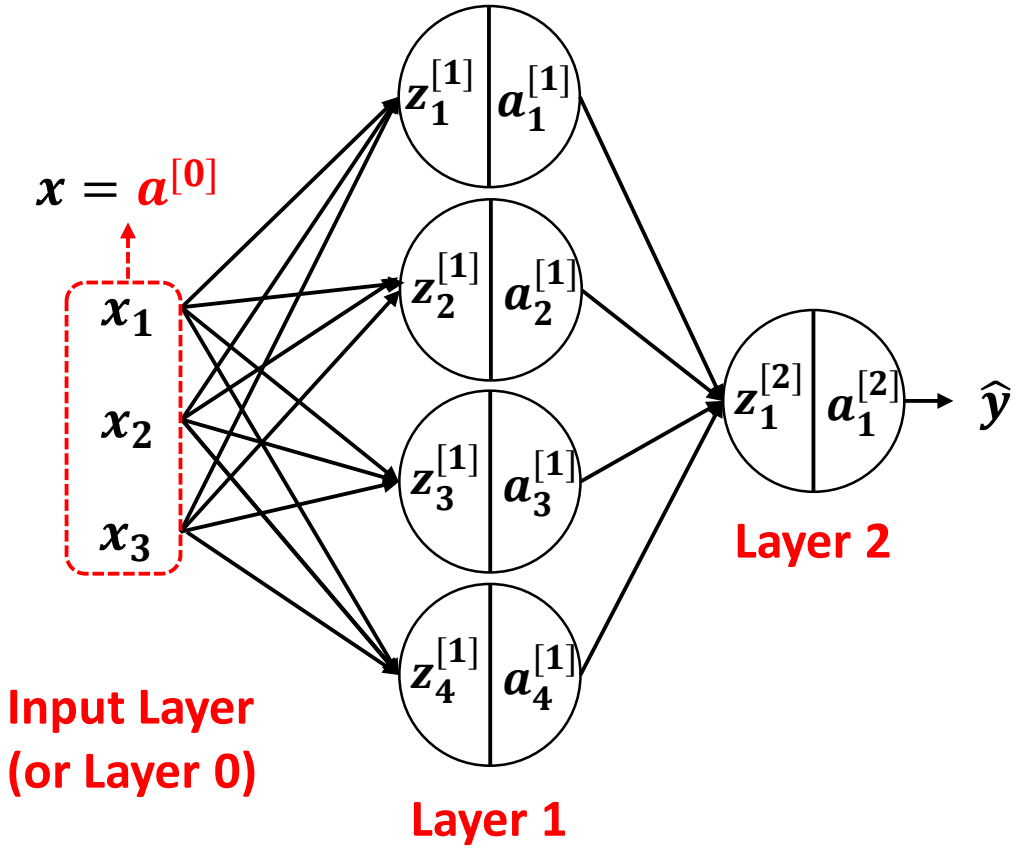
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$Z^{[1]} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & & z_1^{[1](n)} \\ z_2^{[1](1)} & z_2^{[1](2)} & & z_2^{[1](n)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \cdots & z_3^{[1](n)} \\ z_4^{[1](1)} & z_4^{[1](2)} & & z_4^{[1](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

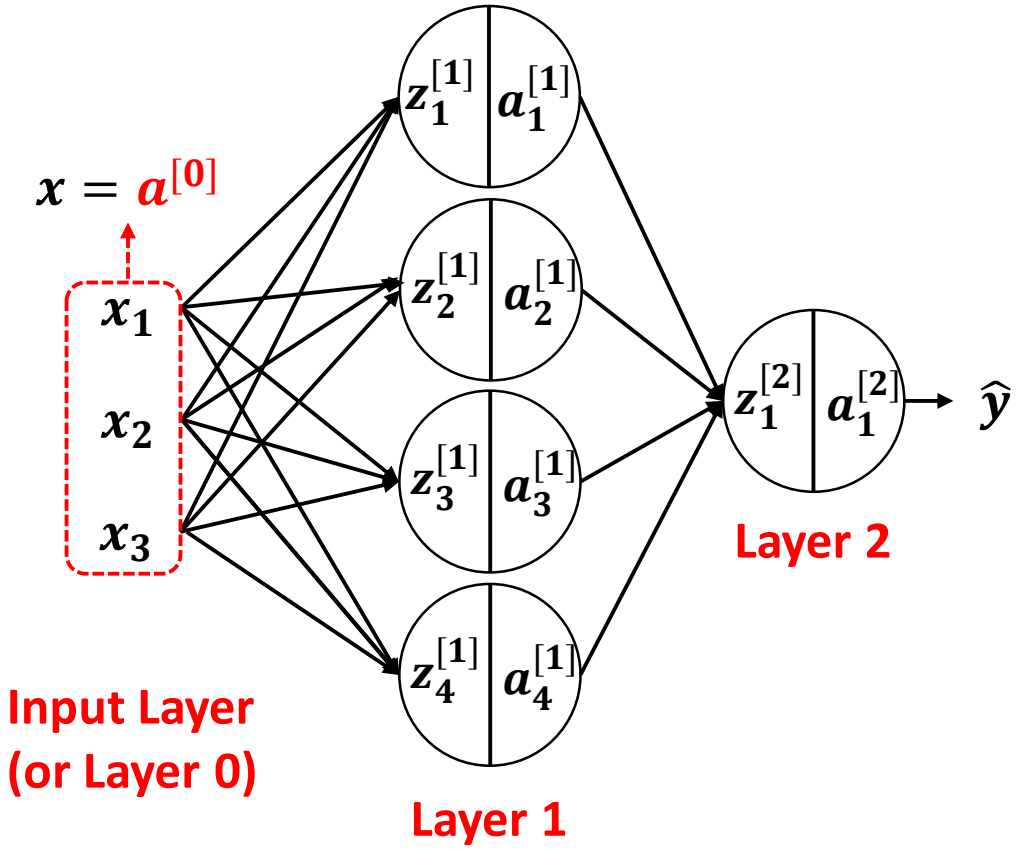
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$A^{[1]} = \begin{bmatrix} a_1^{[1](1)} & a_1^{[1](2)} & & a_1^{[1](n)} \\ a_2^{[1](1)} & a_2^{[1](2)} & & a_2^{[1](n)} \\ a_3^{[1](1)} & a_3^{[1](2)} & \cdots & a_3^{[1](n)} \\ a_4^{[1](1)} & a_4^{[1](2)} & & a_4^{[1](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

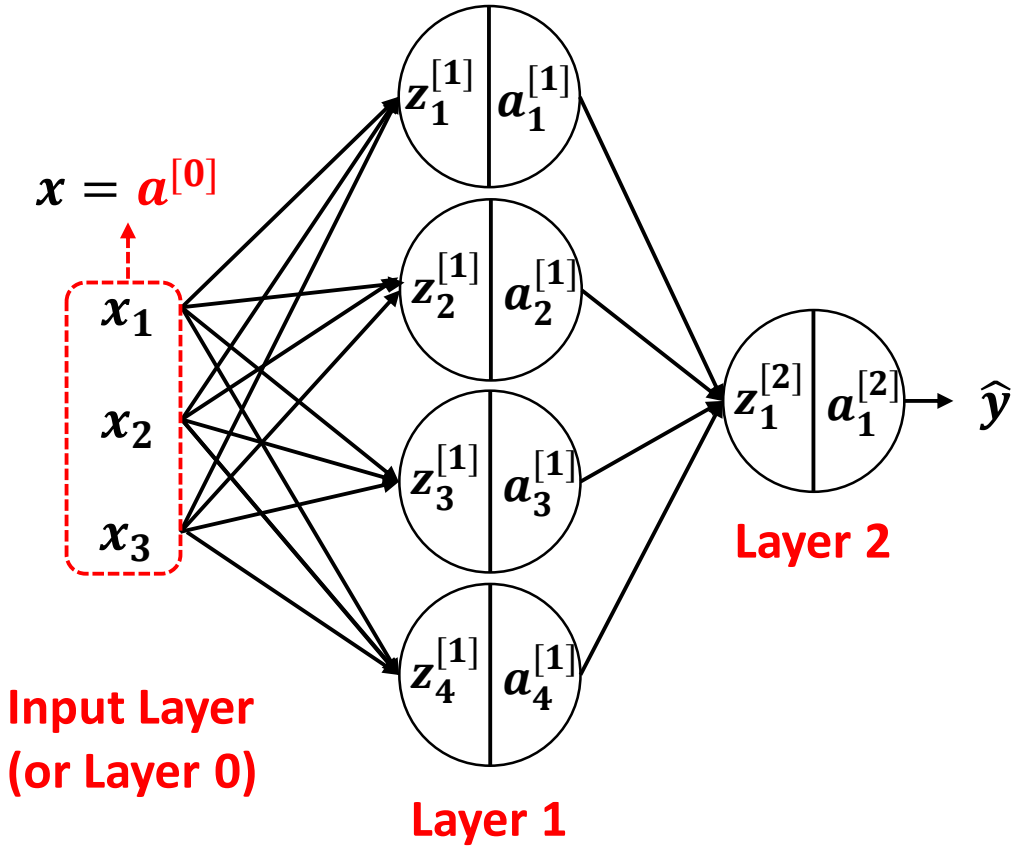
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$Z^{[2]} = \begin{bmatrix} z_1^{[2](1)} & z_1^{[2](2)} & \dots & z_1^{[2](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

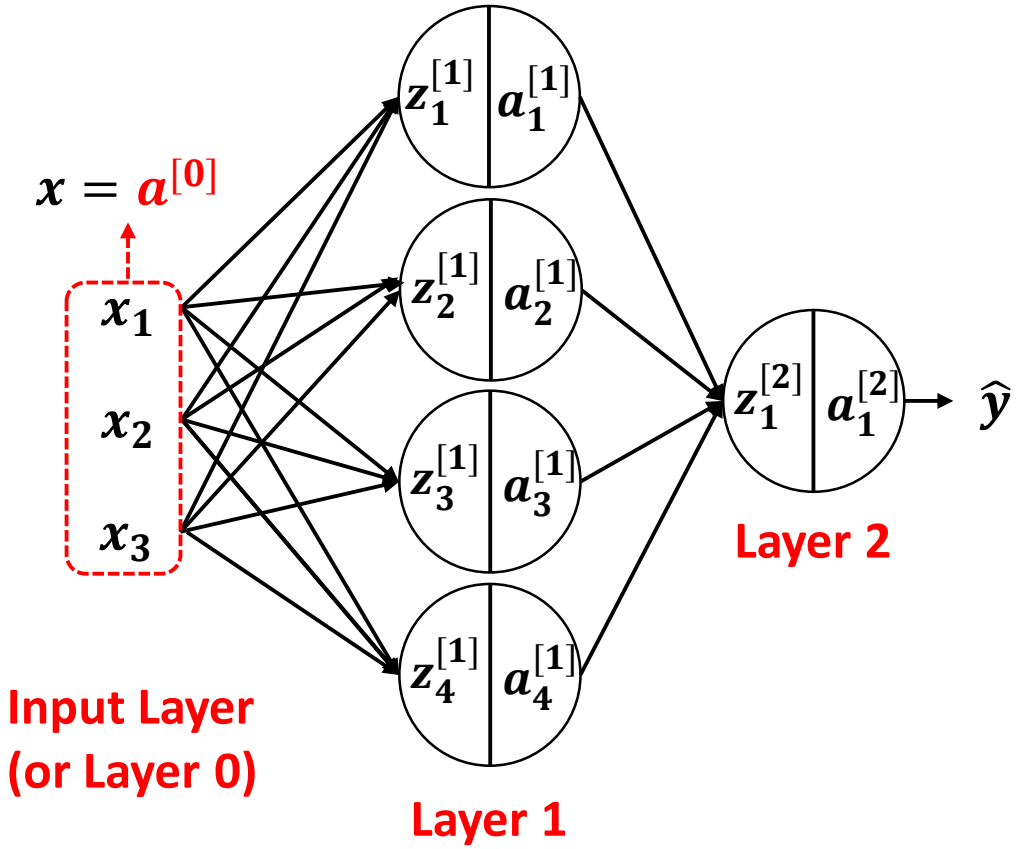
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$A^{[2]} = \begin{bmatrix} a_1^{[2](1)} & a_1^{[2](2)} & \dots & a_1^{[2](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{for\ i = 1\ to\ n}:$$

$$z^{[1](i)} = w^{[1]^T(i)} a^{[0](i)} + b^{[1]}$$

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

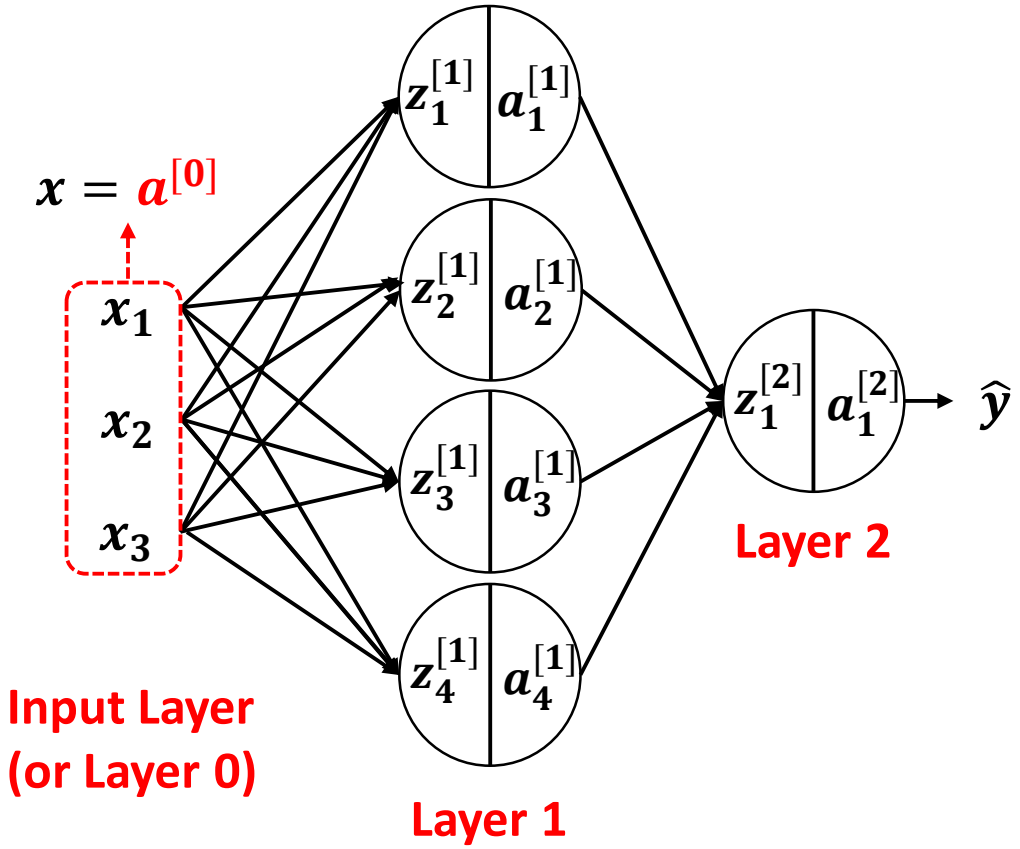
$$z^{[2](i)} = w^{[2]^T(i)} a^{[1](i)} + b^{[2]}$$

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

***Before* Vectorization**

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



No Explicit Loop!

$$Z^{[1]} = w^{[1]^T} A^{[0]} + b^{[1]}$$

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

$$Z^{[2]} = w^{[2]^T} A^{[1]} + b^{[2]}$$

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

***After* Vectorization**

Summary

- Spam and phishing
- Examples of Email Fraud
- Spam Filtering Techniques
- Pre-processing Text in Email Messages
  - bag-of-words
  - n-gram
  - TF-IDF
- Spam detection with neural networks
  - Model representation

References

- *Email Spam Filtering, Computer Security Seminar, N.Muthiyalu Jothir*
- *Special Topics: Adversarial Machine Learning, Dr. Alex Vakanski, University of Idaho*
- *Introduction to Information Retrieval, Christopher D. Manning et al, Cambridge University Press. 2008.*
- *Machine Learning and Security Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *Deep learning, Mohammad Hammoud, CMU Qatar*

Backpropagation and Anomaly Detection

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- Neural networks
  - Model representation (Last week)
  - Computation graph
  - Back propagation

- What are anomalies
- Types of anomalies
- Applications of anomaly detection
- Machine learning for intrusion detection
- Anomaly detection techniques

The Flow of Computations in Neural Networks

- The flow of computations in a neural network goes in two ways:
  1. Left-to-right: This is referred to as *forward propagation*, which results in computing the output of the network
  2. Right-to-left: This is referred to as *back propagation*, which results in computing the gradients (or derivatives) of the parameters in the network

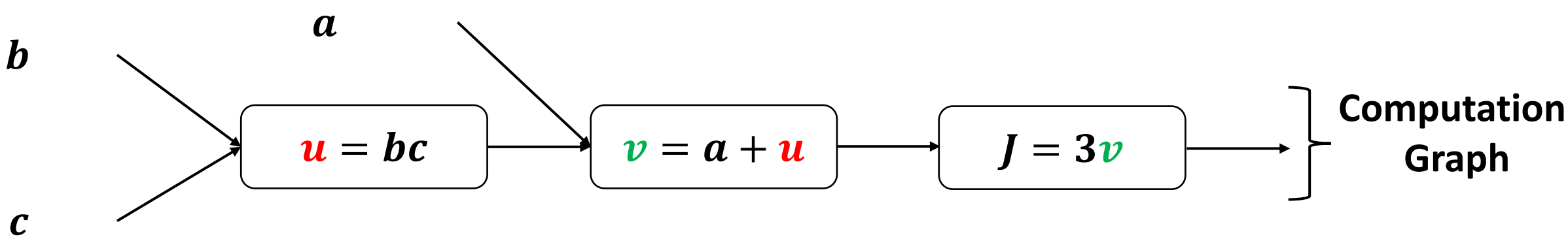
- The intuition behind this 2-way flow of computations can be explained through the concept of "computation graphs"
  - What is a computation graph?

What is a Computation Graph?

- Let us assume we want to compute the following function J:

$$J(a,b,c) = \underbrace{3(\underbrace{a + \underbrace{bc}_{u}}_{v})}_{J}$$

$$\left.\begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}\right\}$$

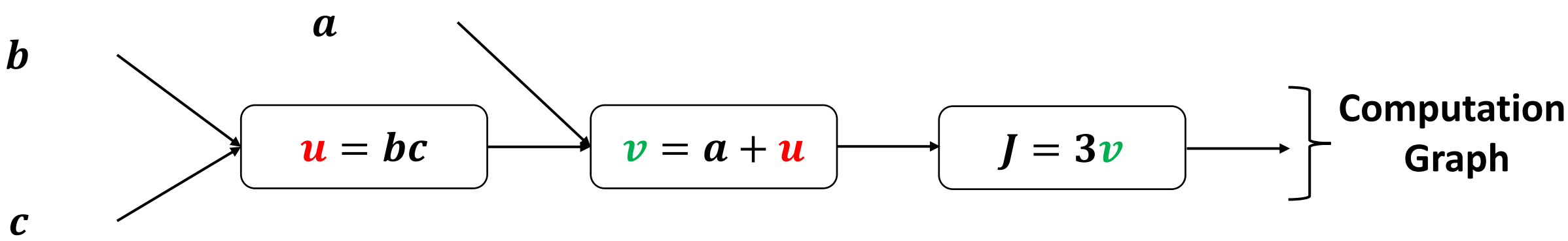


Forward Propagation

- Let us assume we want to compute the following function J:

$$\underbrace{\underbrace{a + \underbrace{bc}_{u}}_{v}}_{J} = 3(a + bc)$$

$$\left.\begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}\right\}$$

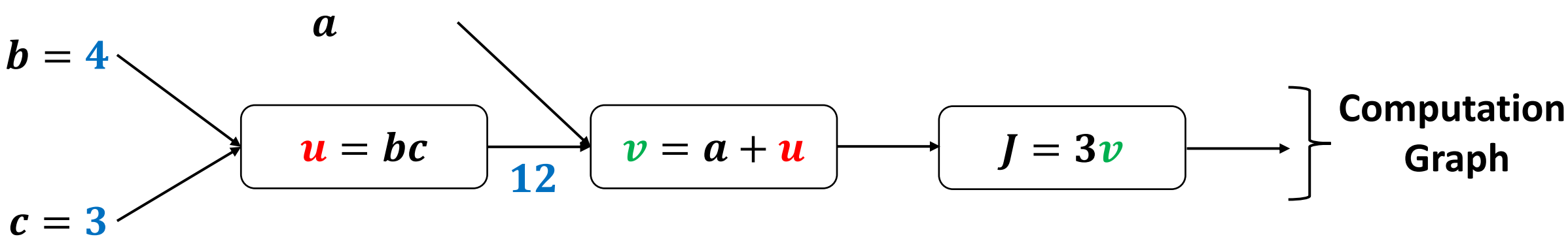


Forward Propagation

- Let us assume we want to compute the following function J:

$$J(a,b,c) = \underbrace{3(\underbrace{a + \underbrace{bc}_{u}}_{v})}_{J}$$

$$\left.\begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}\right\}$$

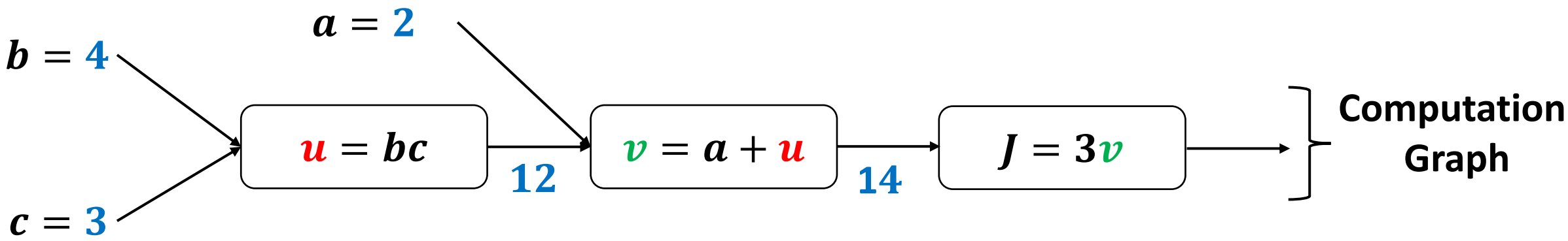


Forward Propagation

- Let us assume we want to compute the following function J:

$$\underbrace{\underbrace{a + \underbrace{bc}_{u}}_{v}}_{J} = 3(a + bc)$$

$$\left.\begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}\right\}$$

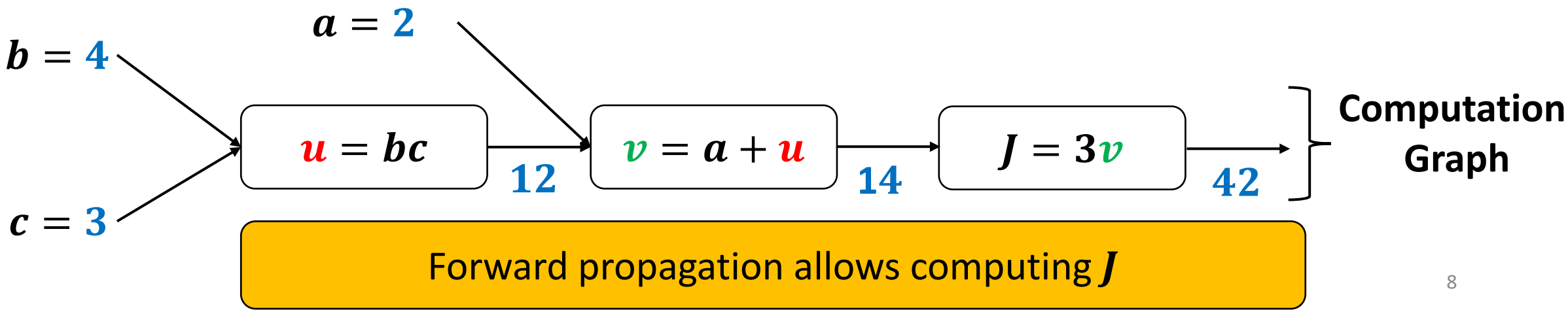


Forward Propagation

- Let us assume we want to compute the following function J:

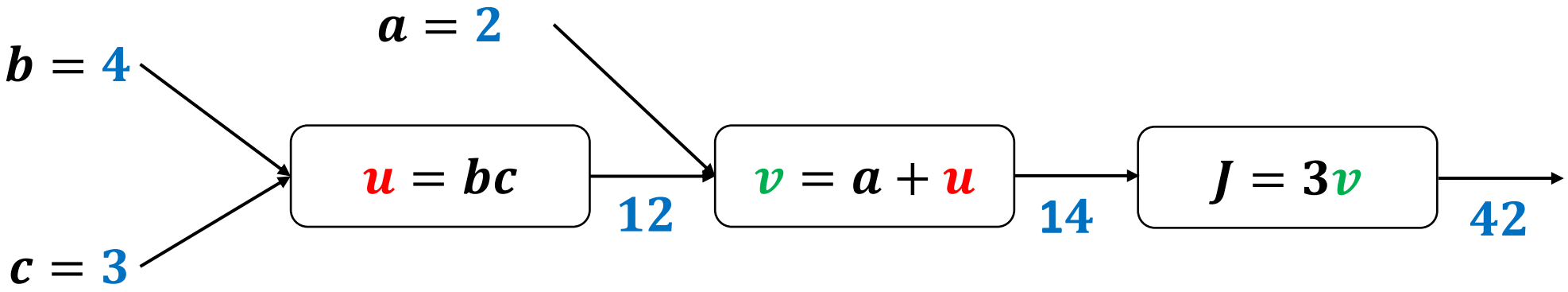
$$J(a,b,c) = \underbrace{3(\underbrace{a + \underbrace{bc}_{u}}_{v})}_{J}$$

$$\left.\begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}\right\}$$



Backward Propagation

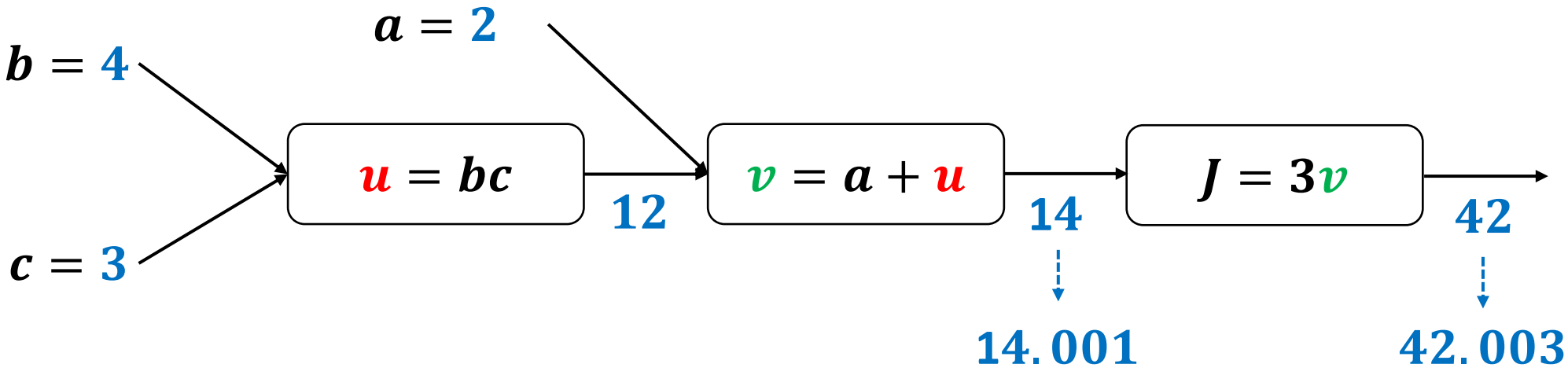
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dv} = \text{Derivative of } J \text{ with respect to } v$$

Backward Propagation

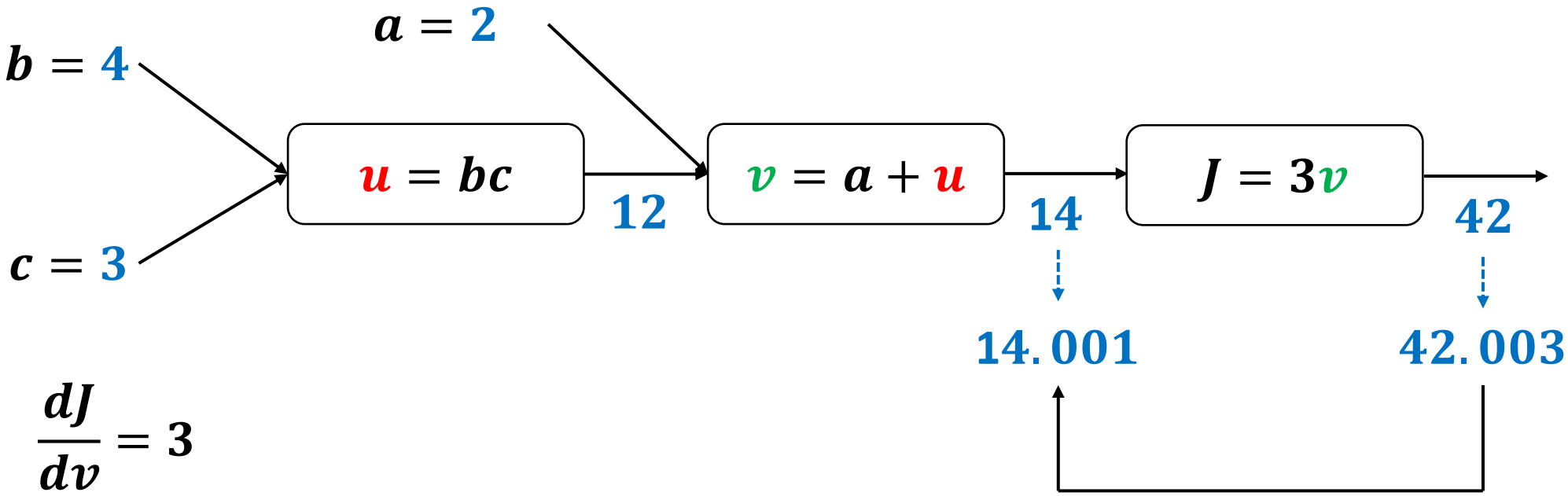
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{dv} =$ If we change v a little bit, how would J change?

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

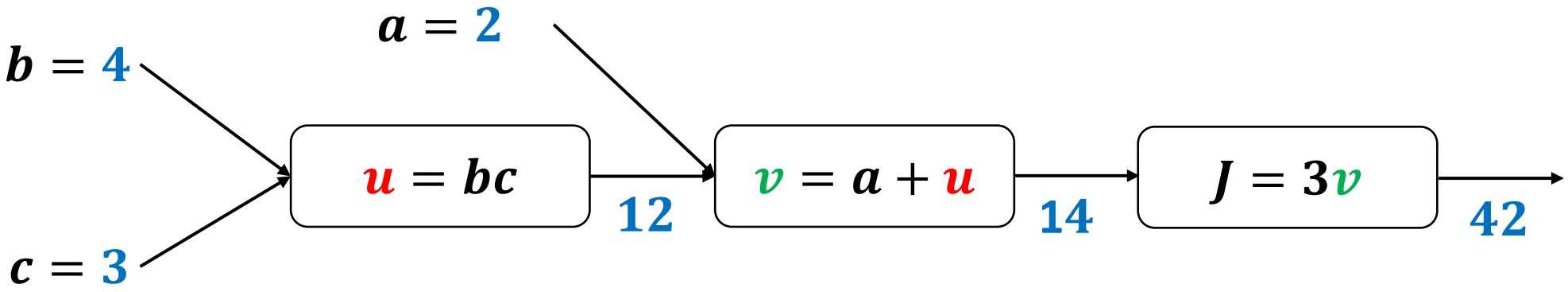


$$\frac{dJ}{dv} = 3$$

To compute the derivative of J with respect to v, we went *back* to v, nudged it, and measured the corresponding resultant increase on J

Backward Propagation

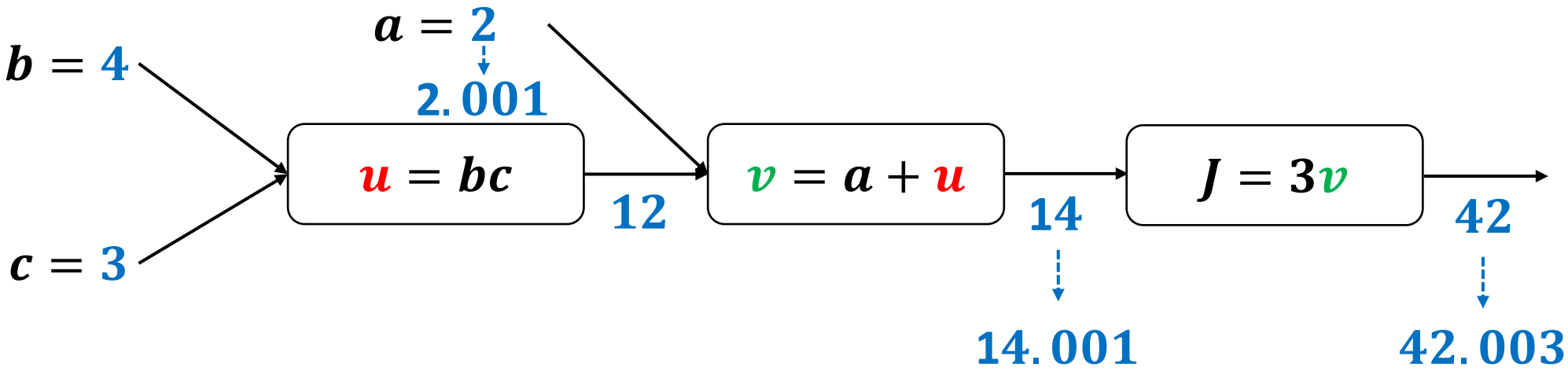
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = \text{Derivative of } J \text{ with respect to } a$$

Backward Propagation

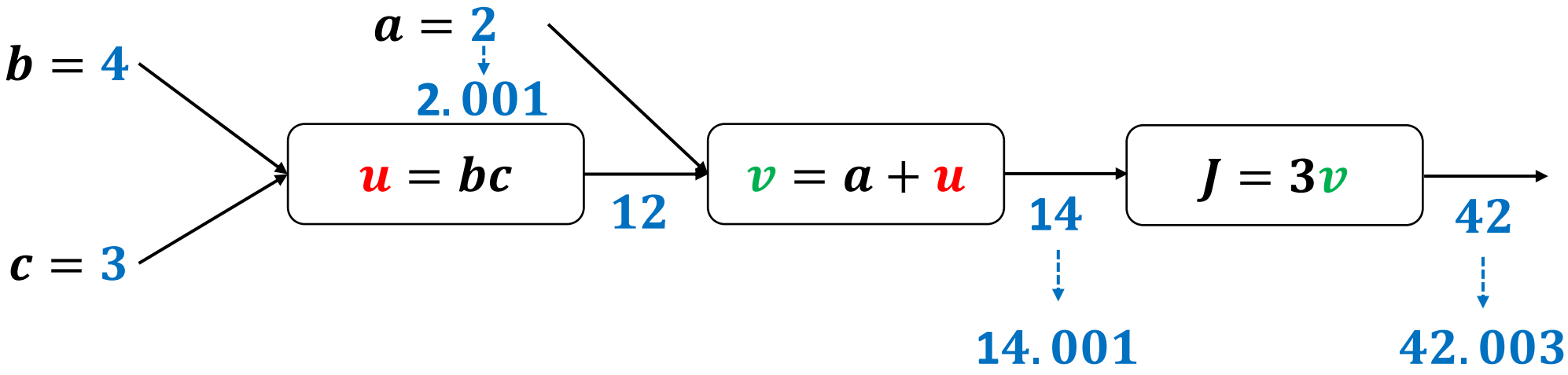
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{da}$ = If we change a a little bit, how would J change?

Backward Propagation

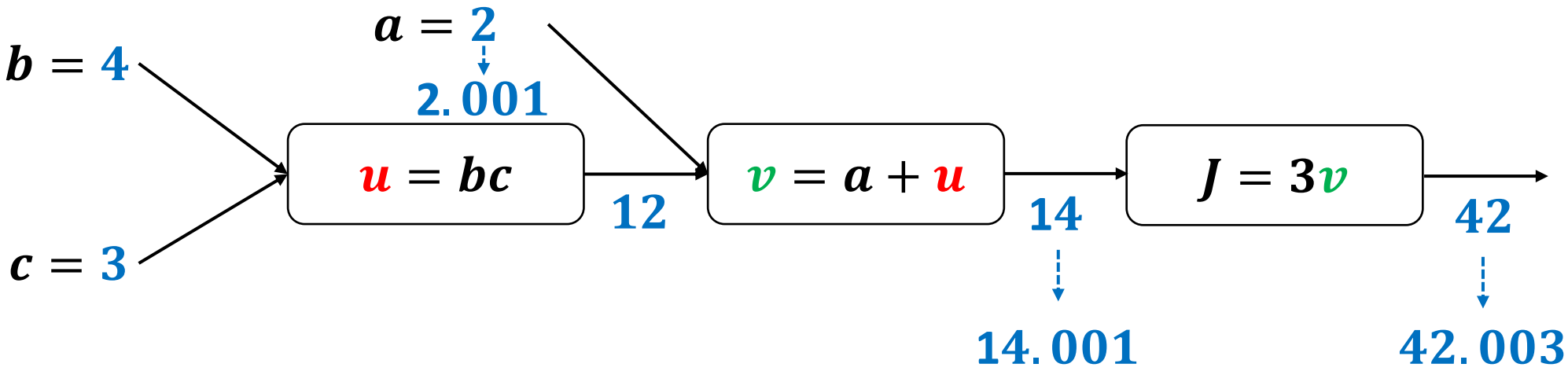
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3$$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

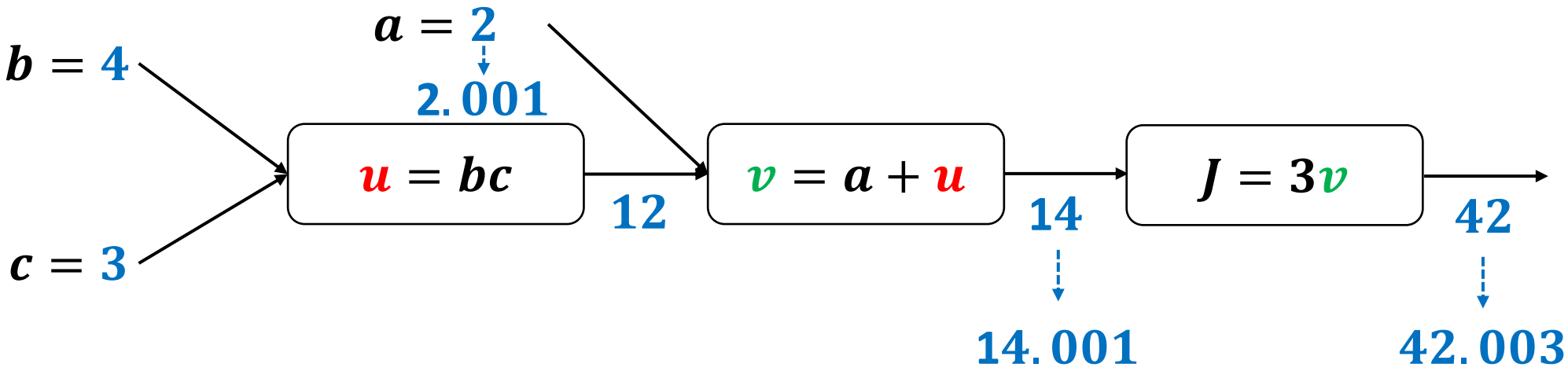


$$\frac{dJ}{da} = 3 = \frac{dv}{da}$$

The change in a caused a change in v

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

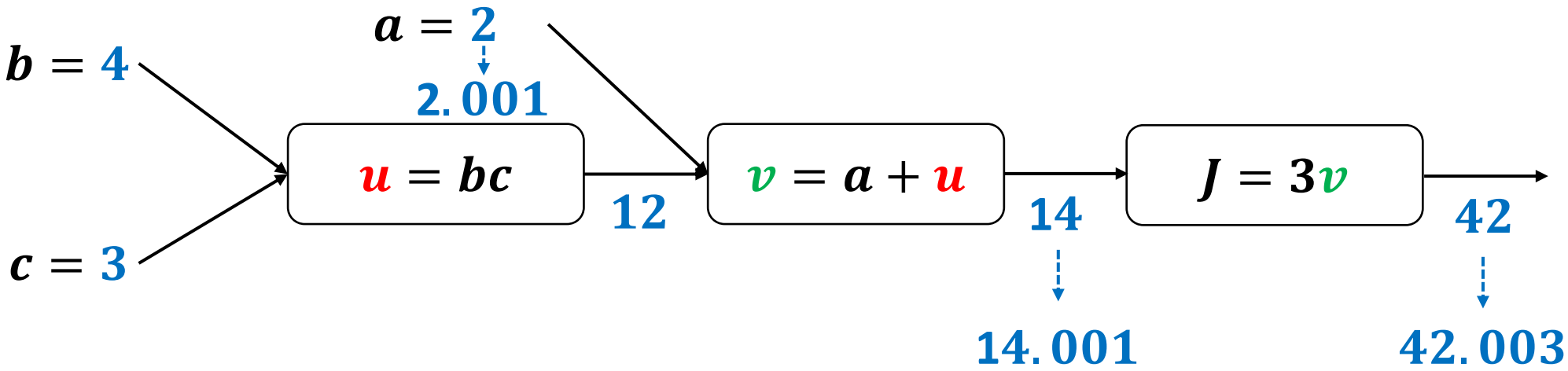


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

And the change in v caused a change in J

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

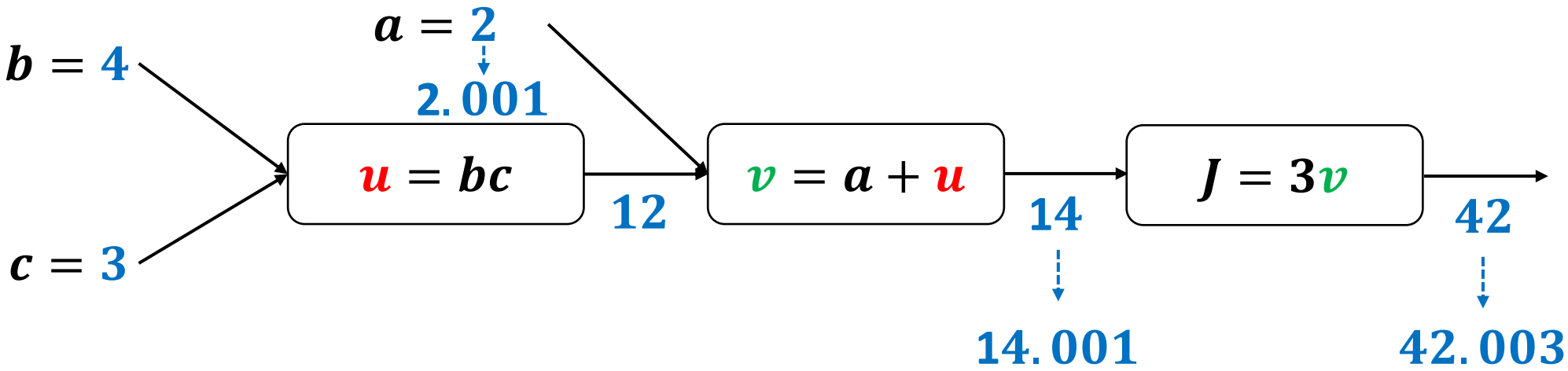


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

This is denoted as ***the chain rule*** in calculus

Backward Propagation

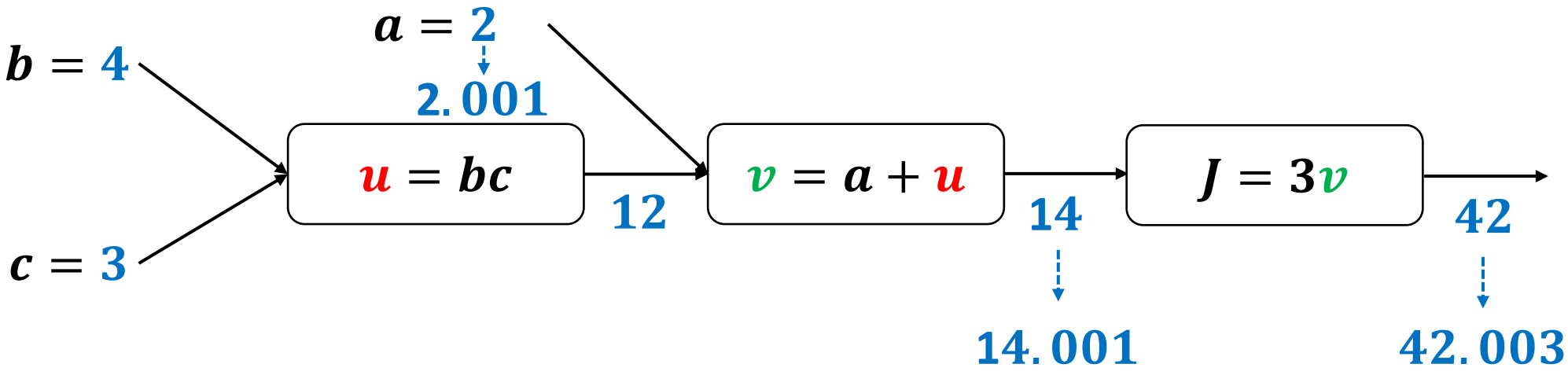
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times 1$$

Backward Propagation

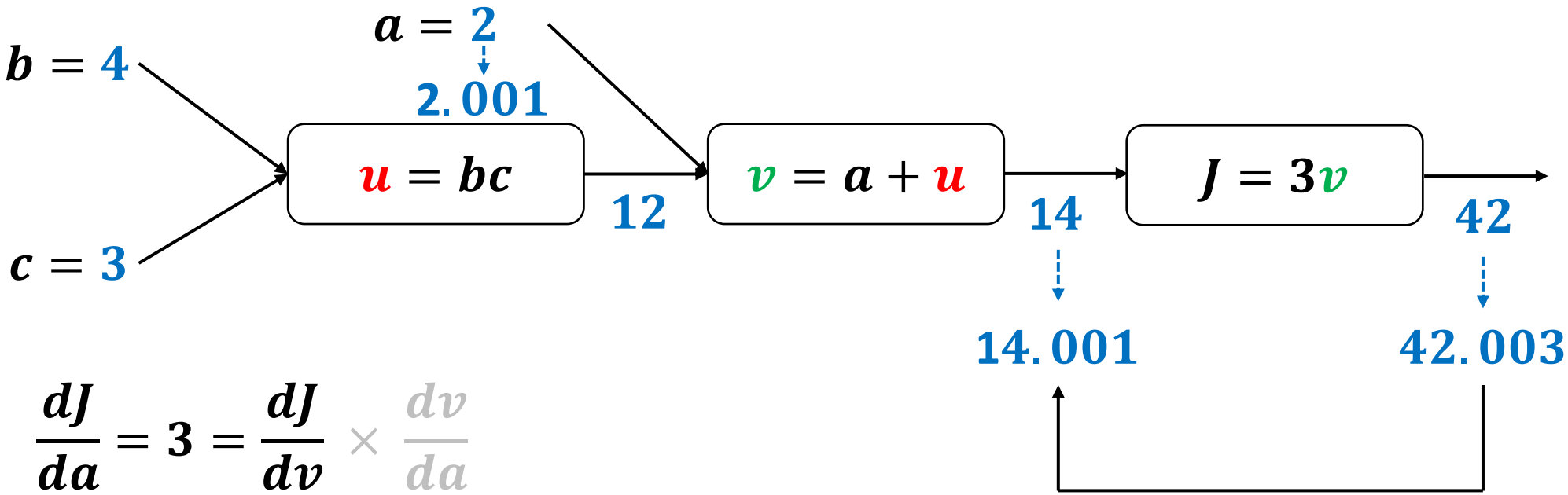
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3 = 3 \times 1$$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

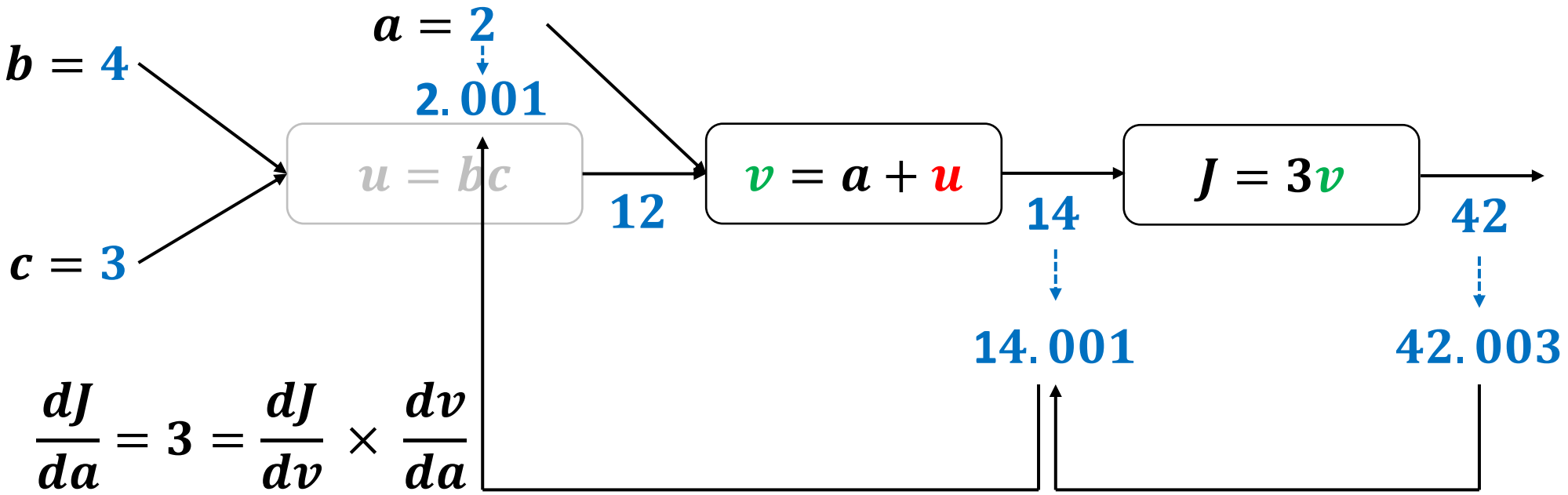


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

In essence, to compute the derivative of J with respect to a, we had to go *back* to v, nudge it a little bit, and measure the corresponding resultant increase on J

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

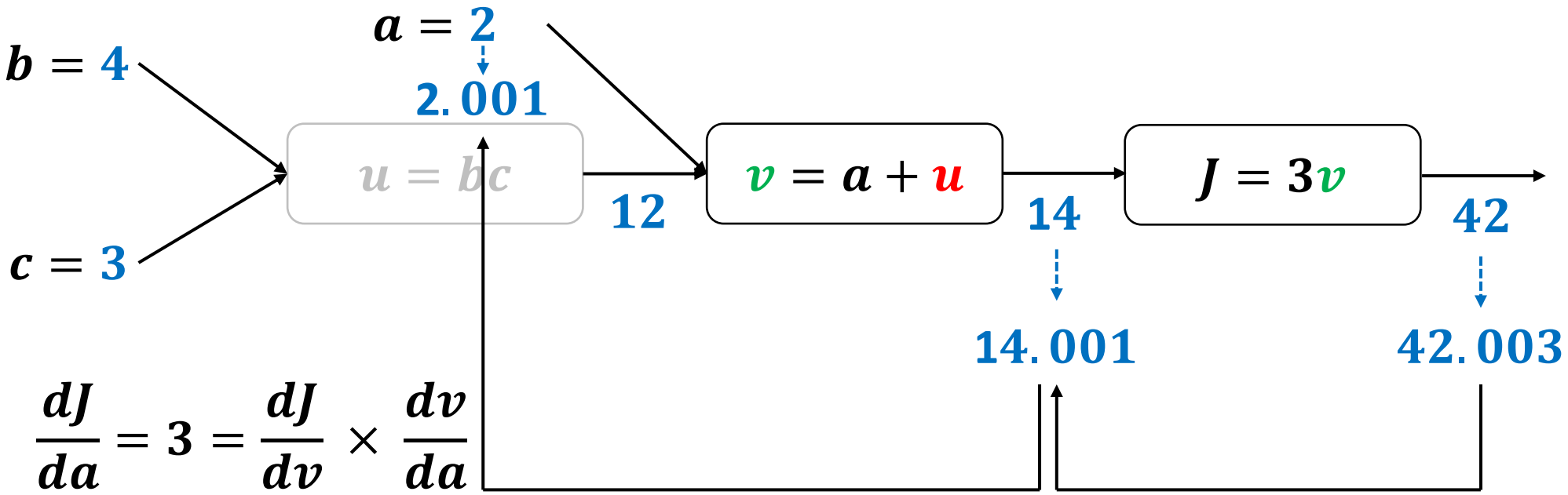


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

Then, we had to go *back* to a, nudge it a little bit, and measure the corresponding resultant increase on v

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

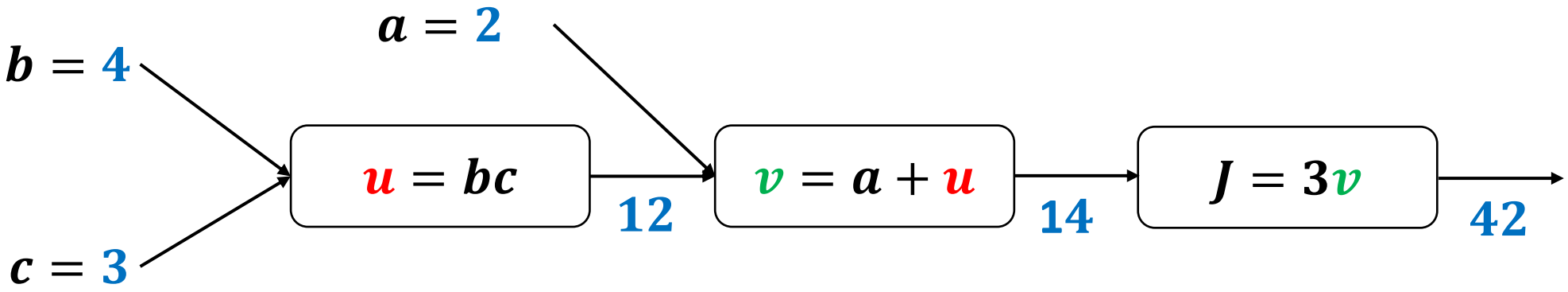


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

Then, we multiplied the changes together (i.e., we applied the chain rule!)

Backward Propagation

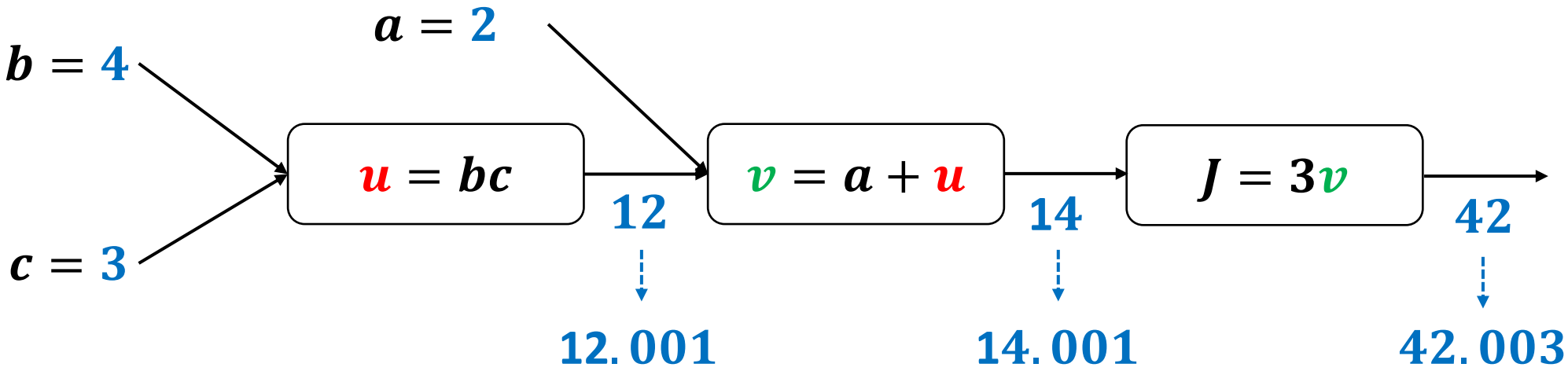
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = \text{Derivative of } J \text{ with respect to } u$$

Backward Propagation

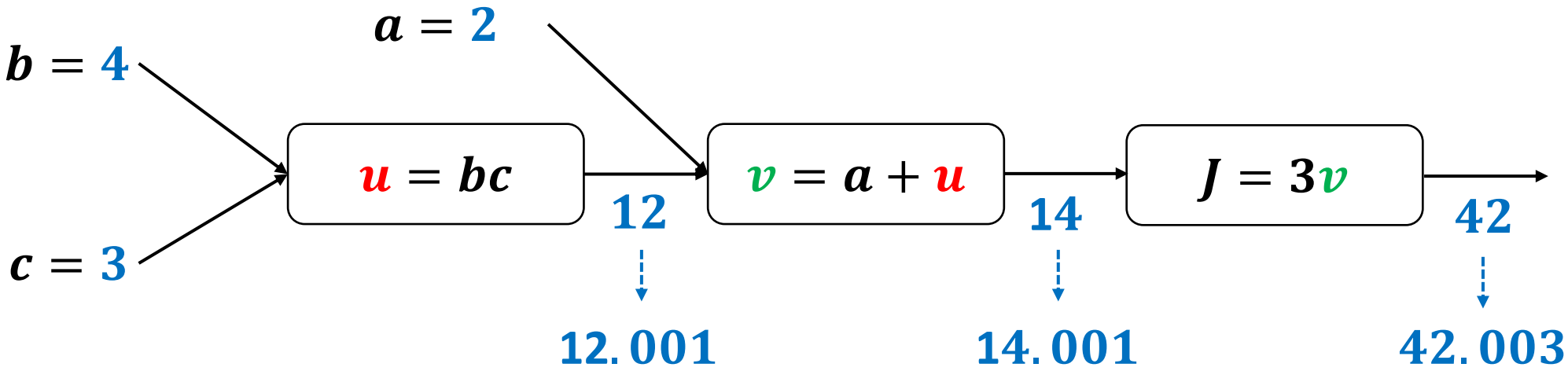
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{du}$ = If we change u a little bit, how would J change?

Backward Propagation

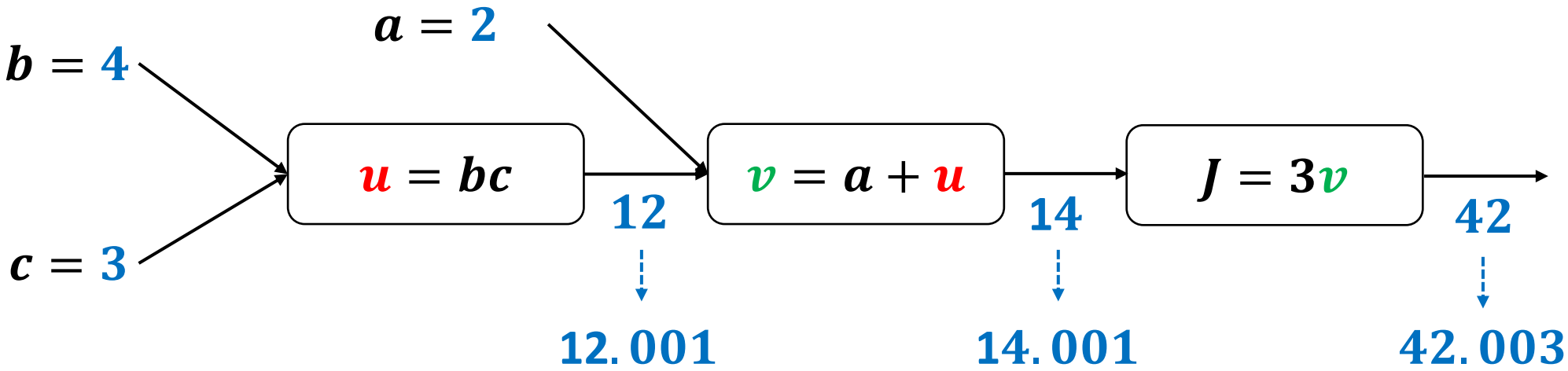
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3$$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

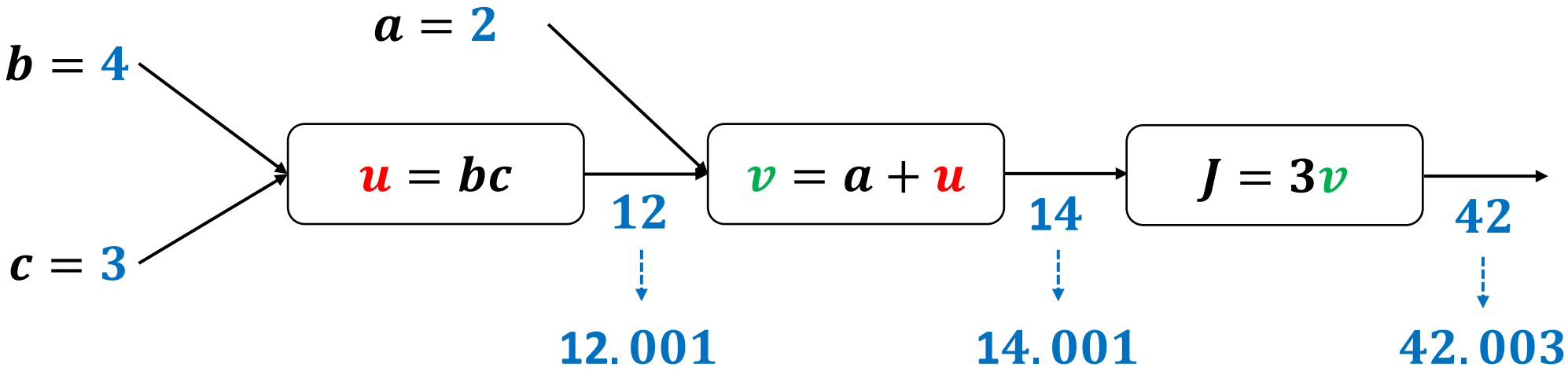


$$\frac{dJ}{du} = 3 = \frac{dv}{du}$$

The change in u caused a change in v

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

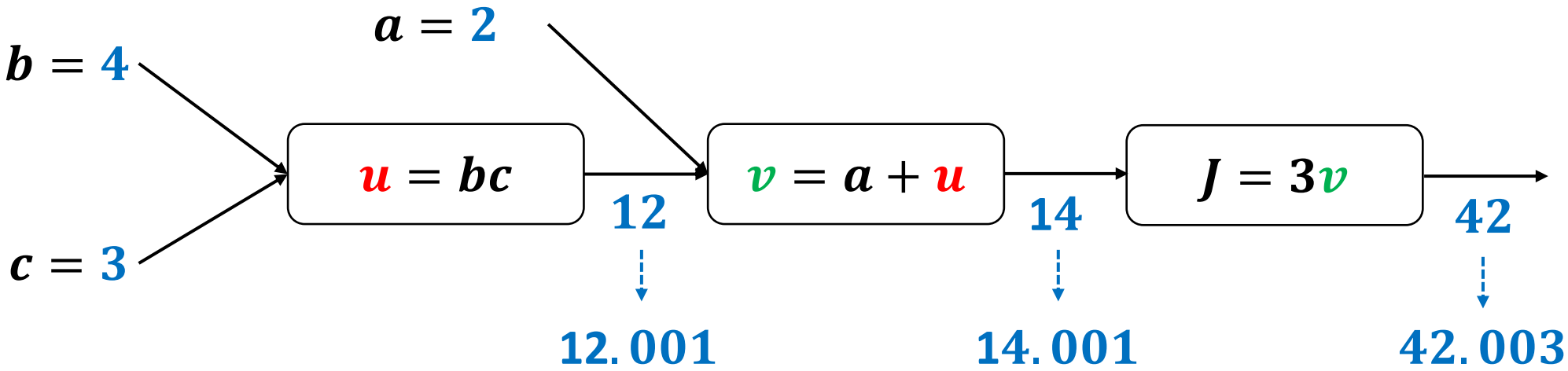


$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

And the change in v caused a change in J

Backward Propagation

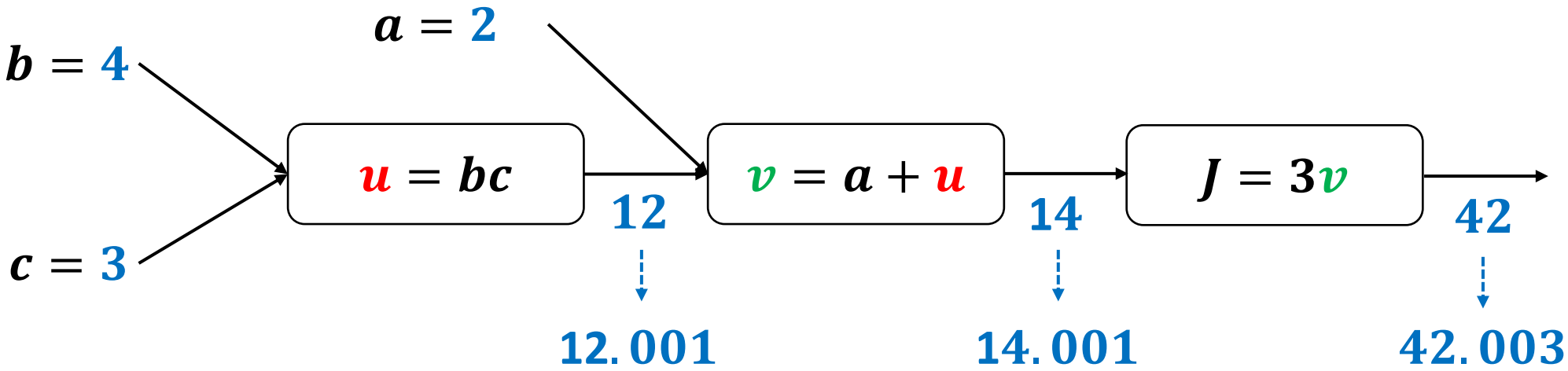
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times 1$$

Backward Propagation

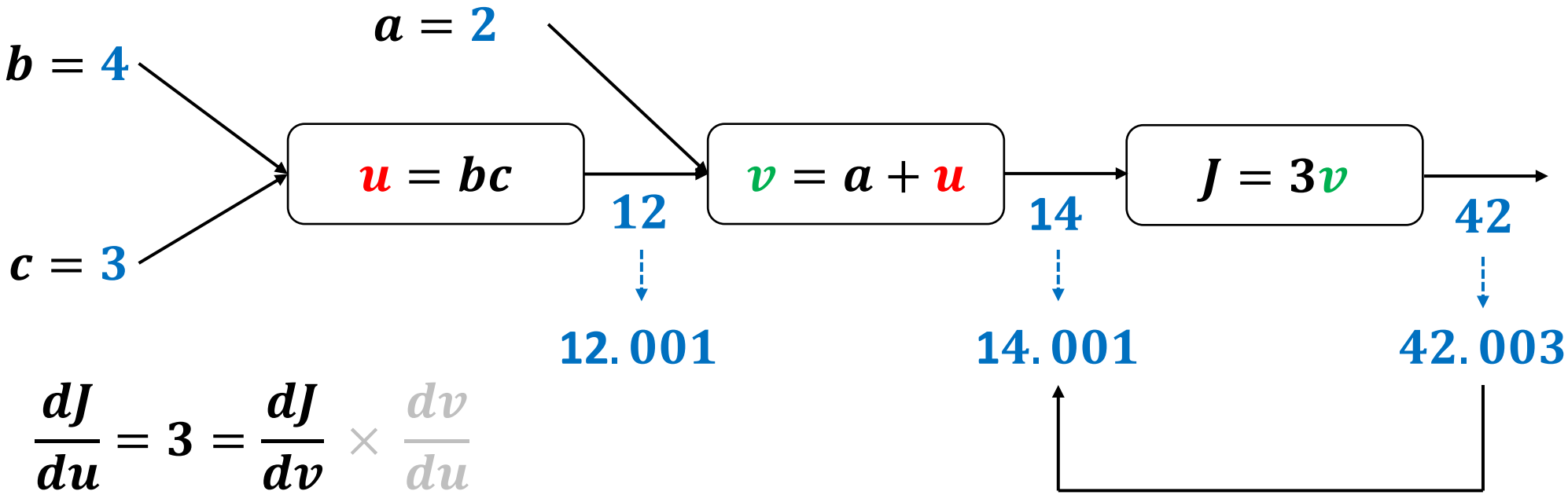
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = 3 \times 1$$

Backward Propagation

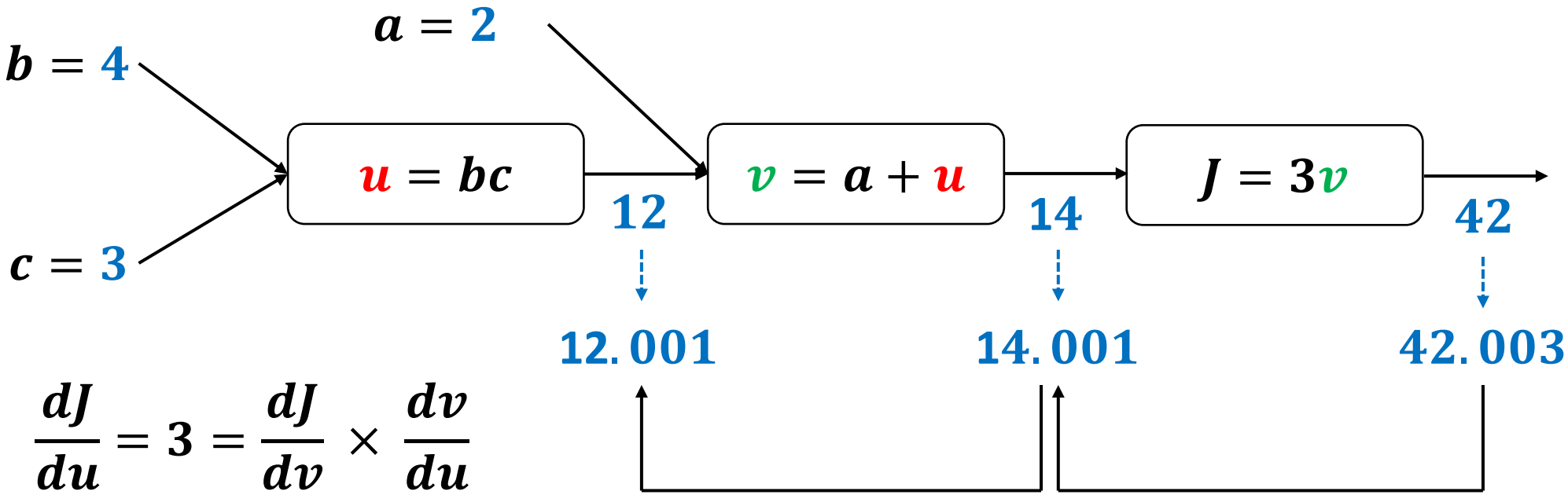
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

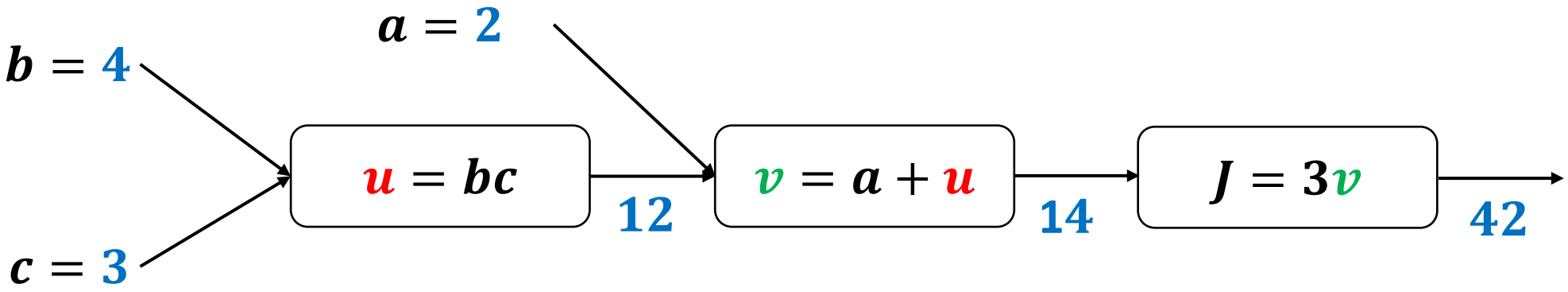


$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

Same as before, we had to go back to v then to u in order to compute the derivative of J

Backward Propagation

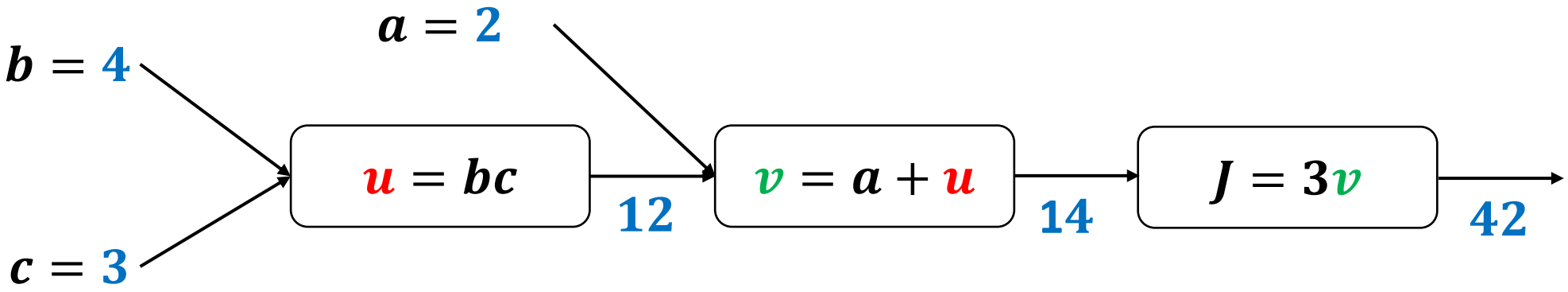
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{db} = \text{Derivative of } J \text{ with respect to } b$$

Backward Propagation

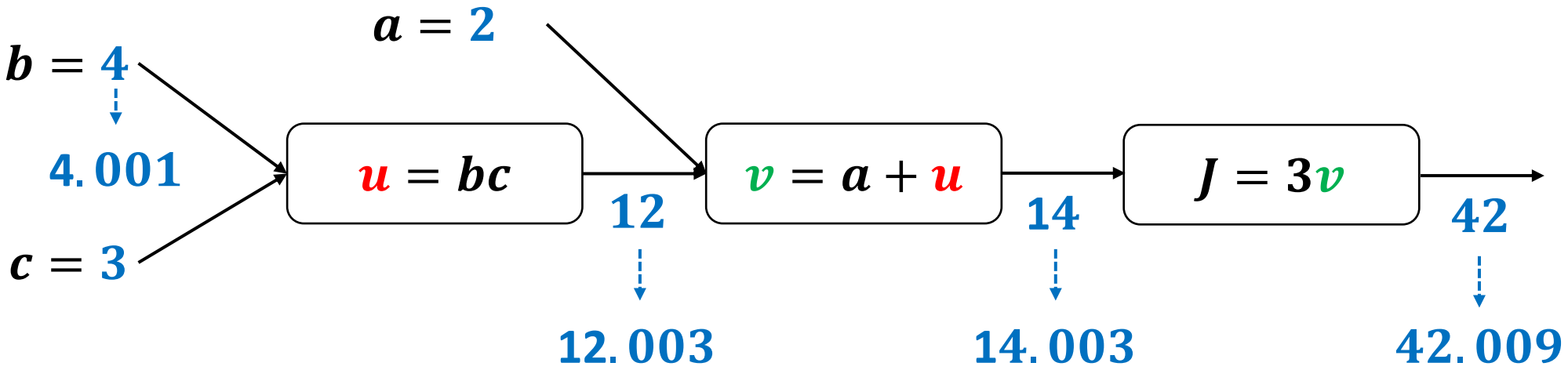
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{db}$ = If we change b a little bit, how would J change?

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

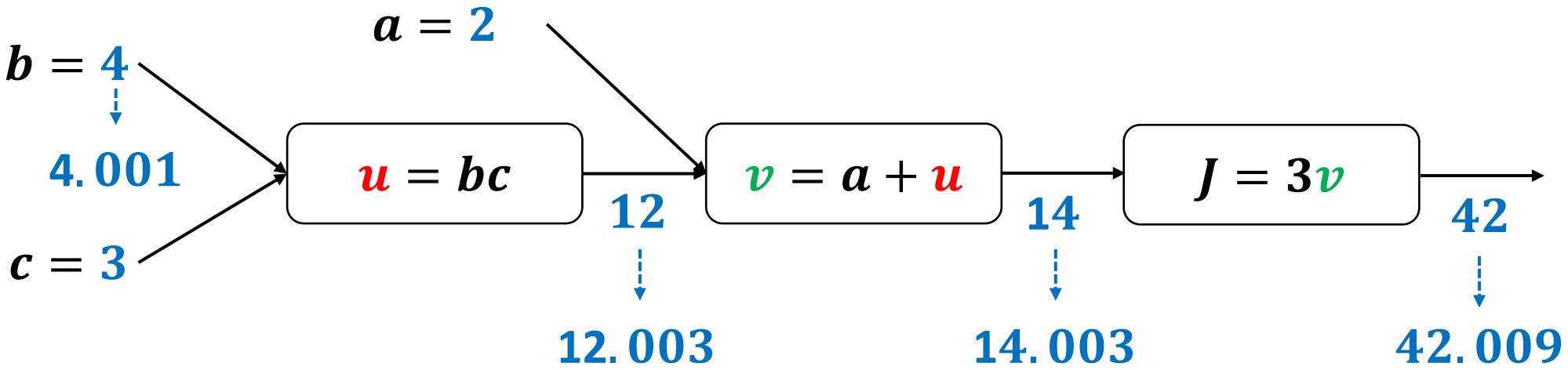


$$\frac{dJ}{db} = ? = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{db}$$

$3 \quad 1 \quad 3$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

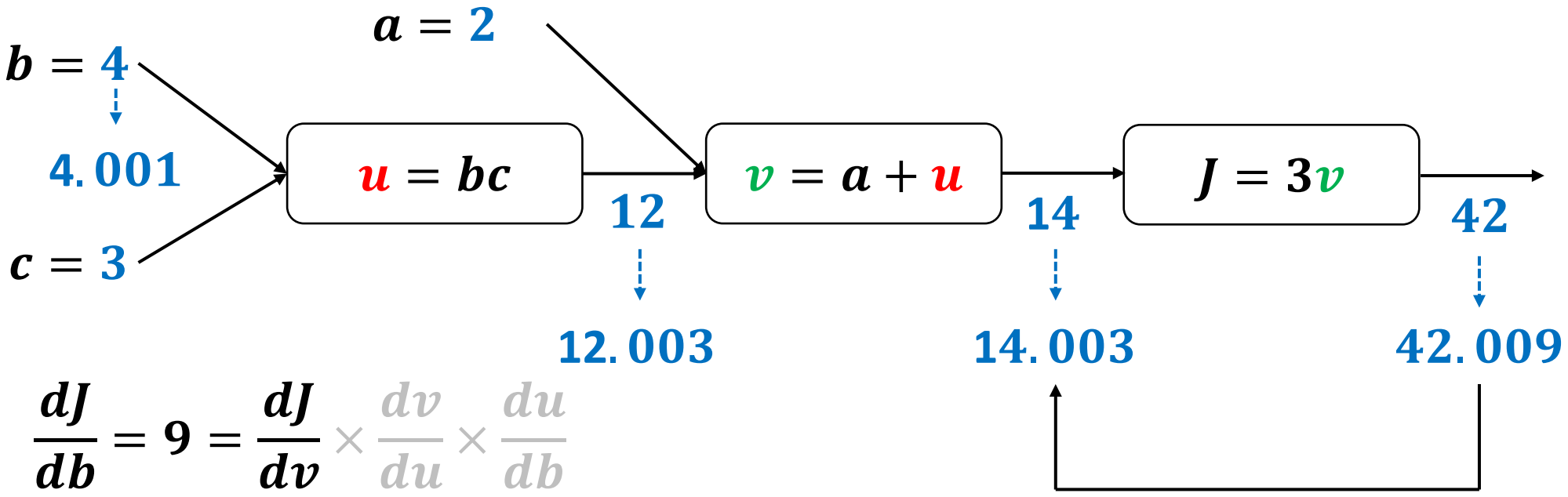


$$\frac{dJ}{db} = 9 = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{db}$$

$3 \quad 1 \quad 3$

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

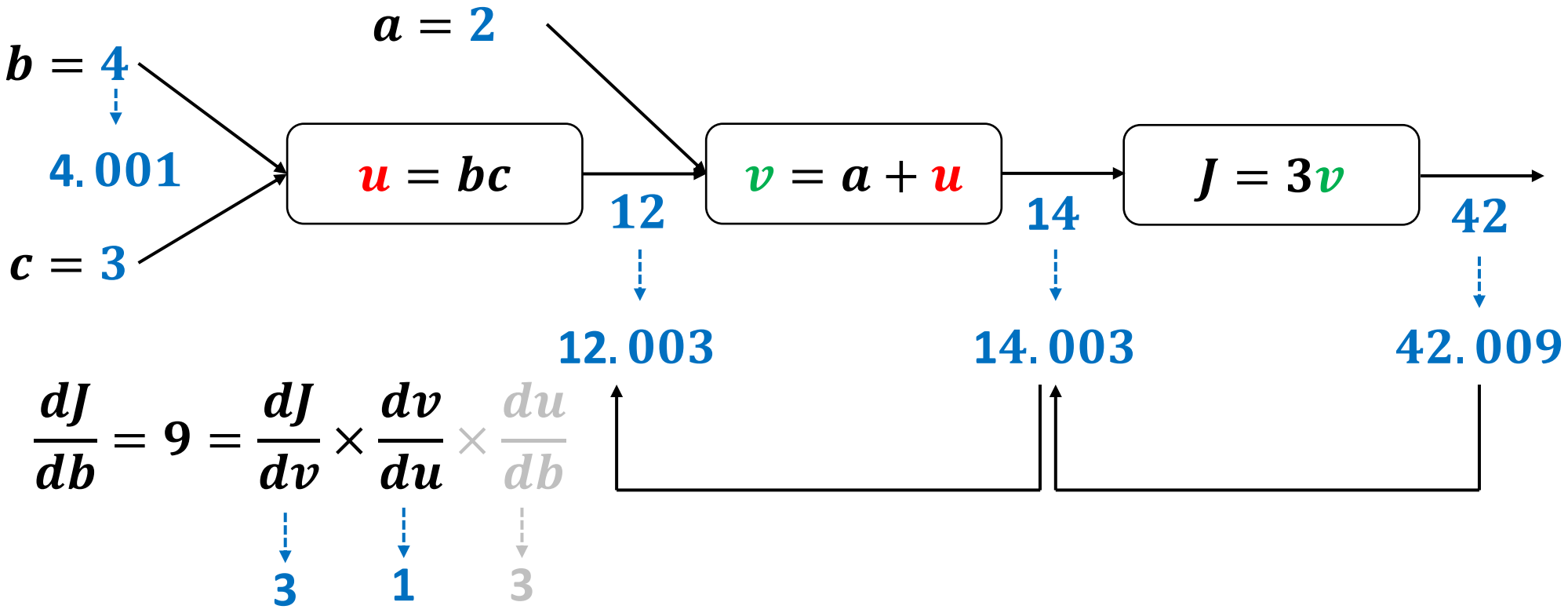


$$\frac{dJ}{db} = 9 = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{db}$$

$3 \quad 1 \quad 3$

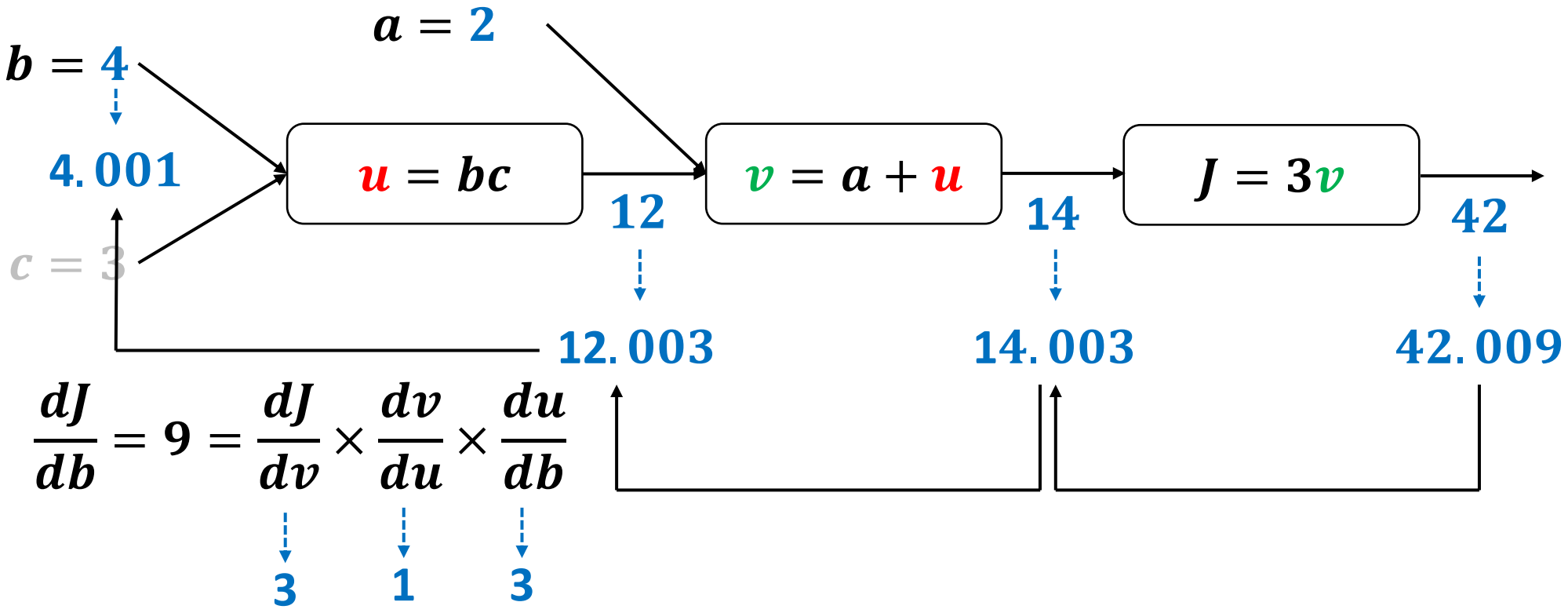
Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:



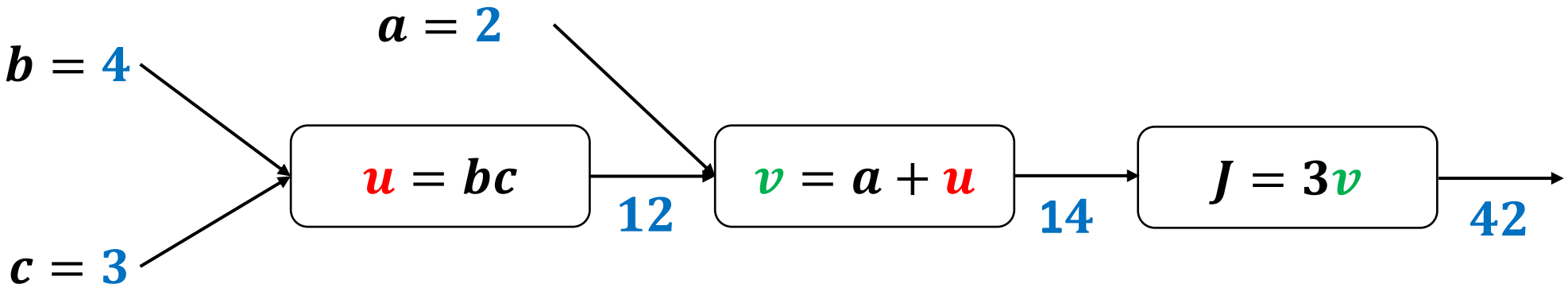
Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:



Backward Propagation

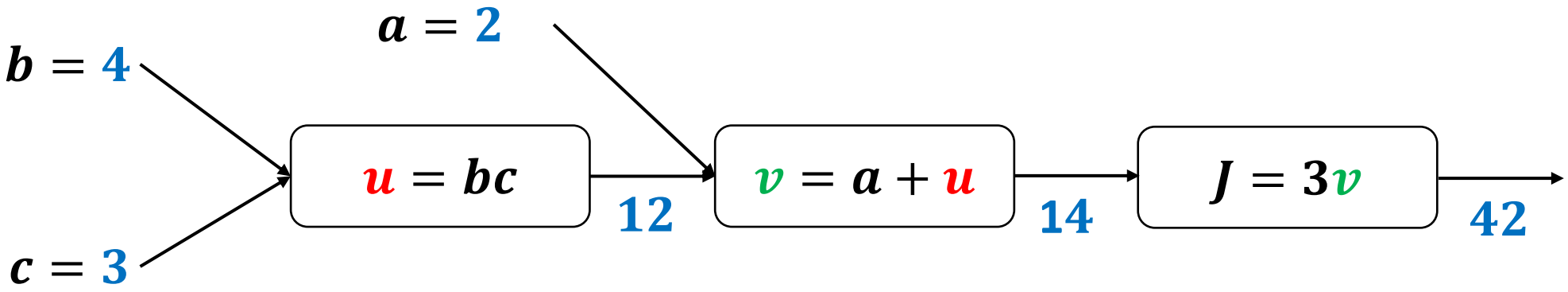
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dc} = \text{ Derivative of } J \text{ with respect to } c$$

Backward Propagation

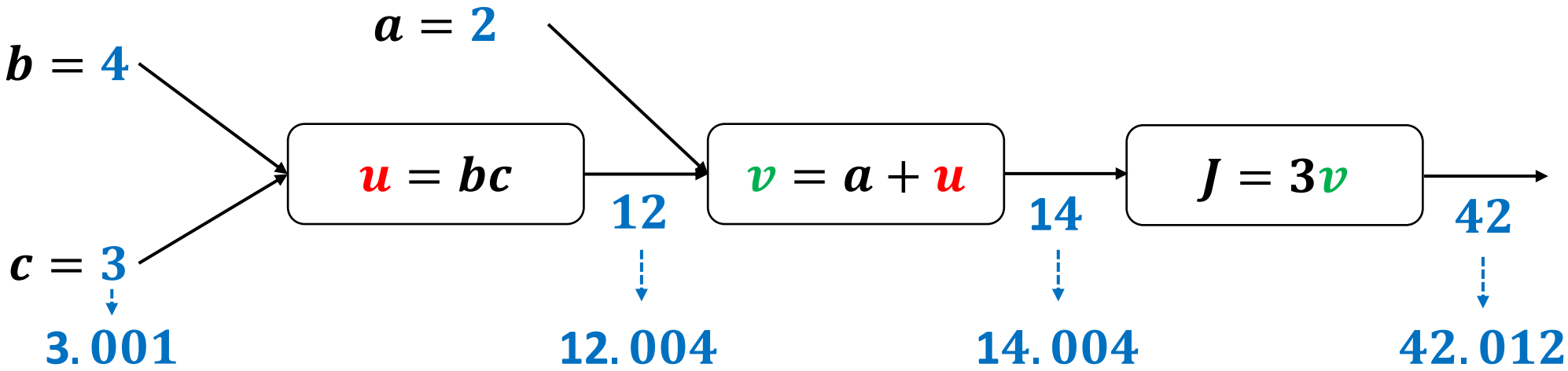
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{dc} =$ If we change c a little bit, how would J change?

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

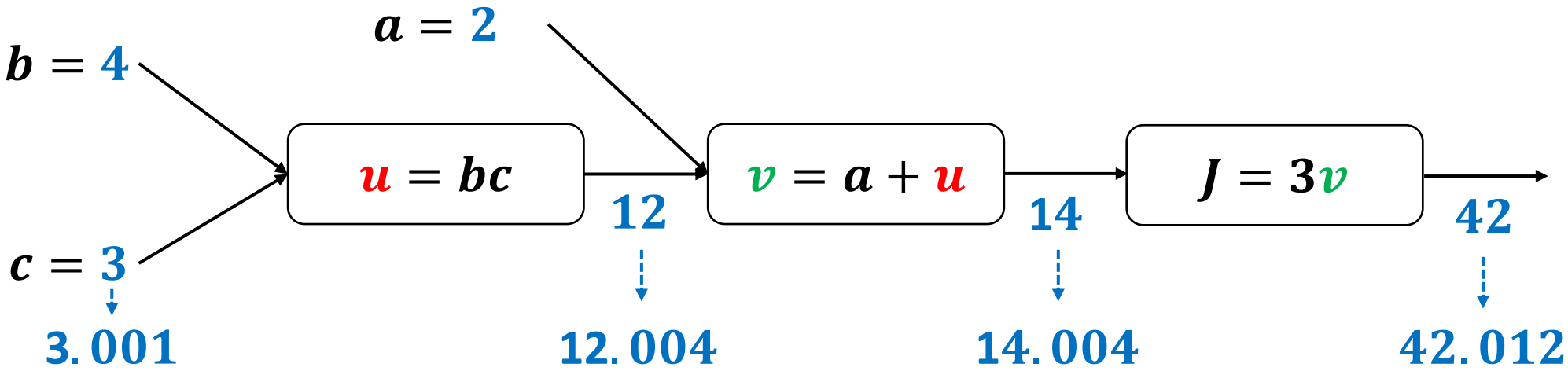


$$\frac{dJ}{dc} = ? = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{dc}$$

3 1 4

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

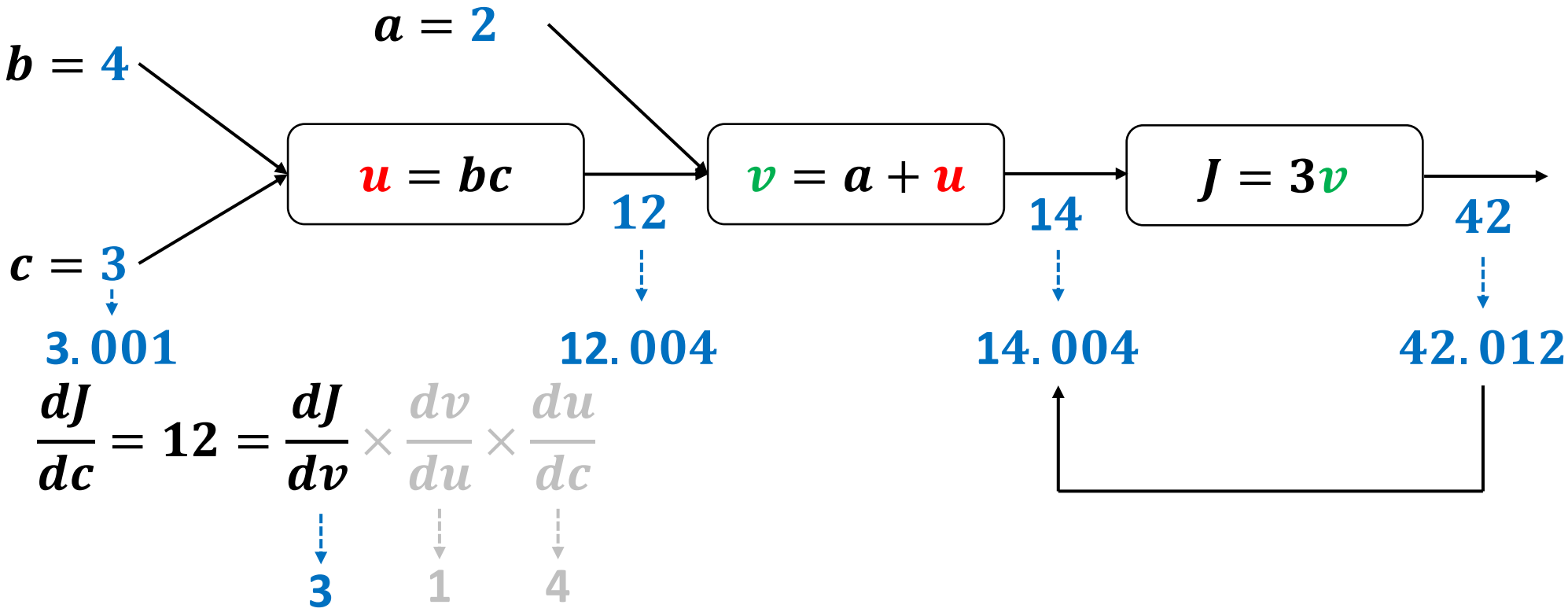


$$\frac{dJ}{dc} = 12 = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{dc}$$

3 1 4

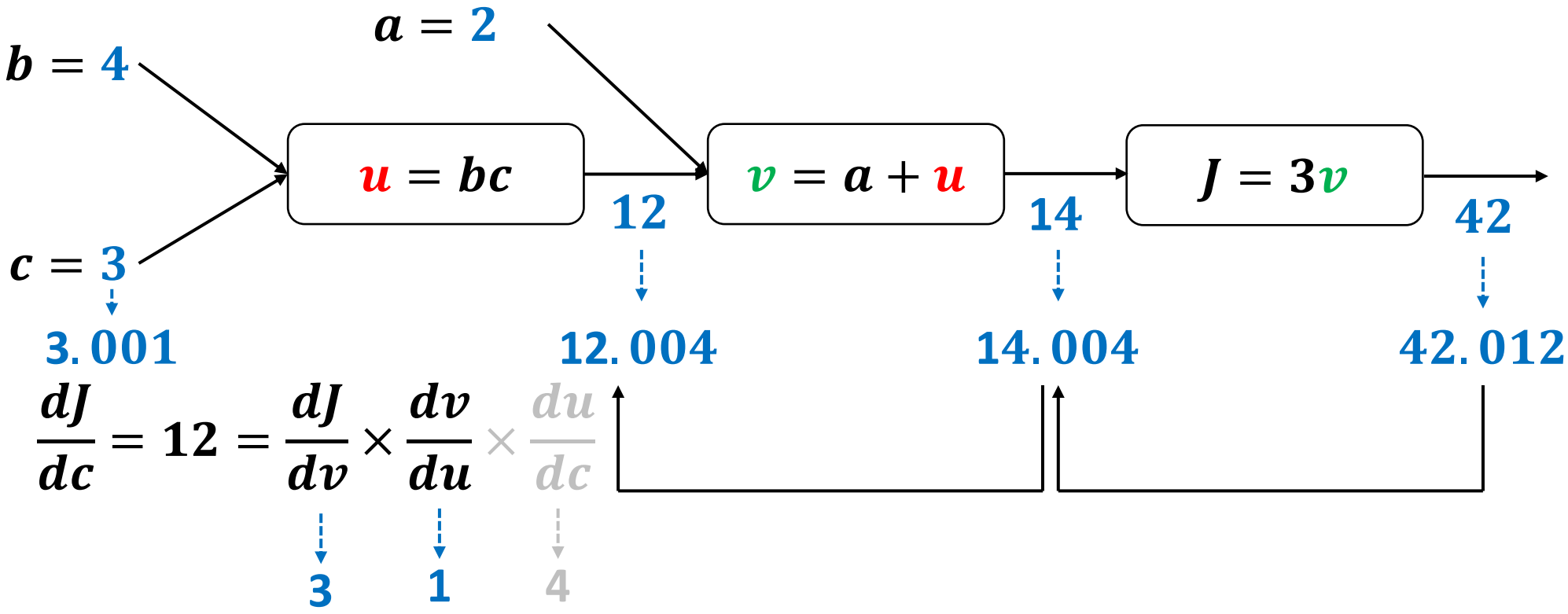
Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:



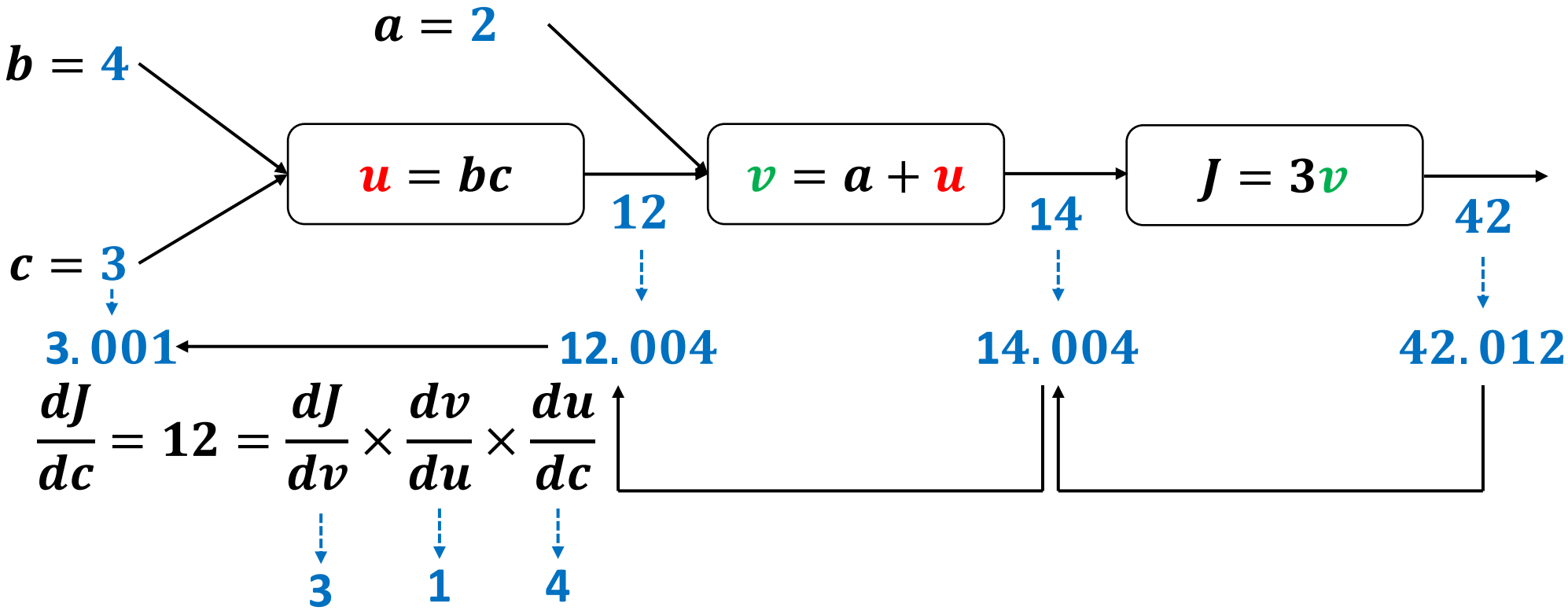
Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:



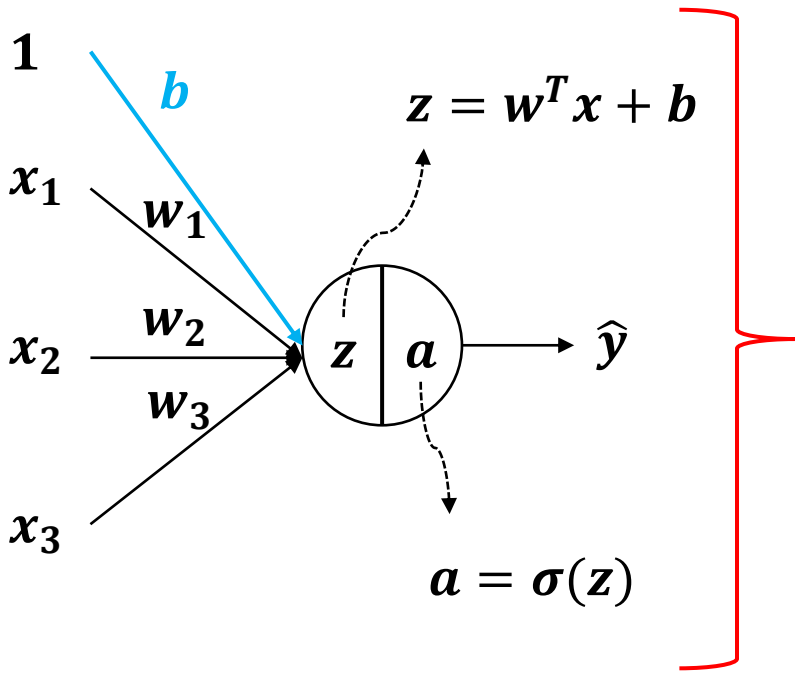
Backward Propagation

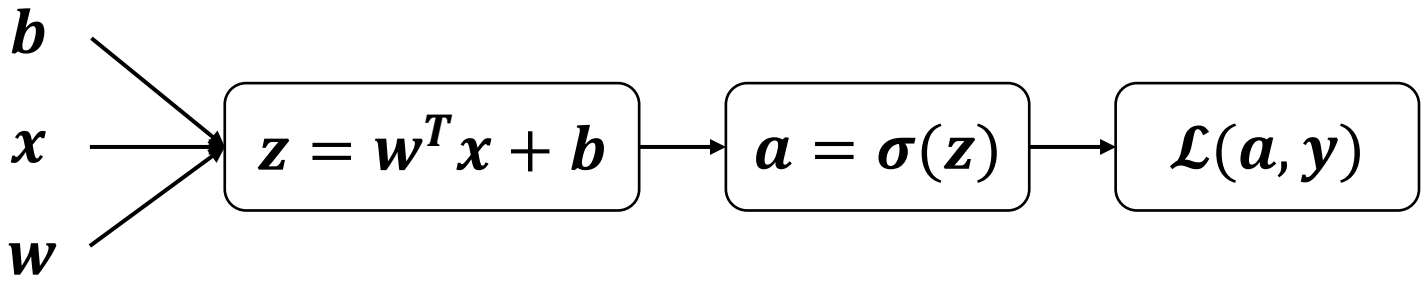
- Let us now compute the derivatives of the variables through the computation graph as follows:



Computation Graph of Logistic Regression

- Let us translate logistic regression (which is a **neural network** with only 1 neuron) into a computation graph

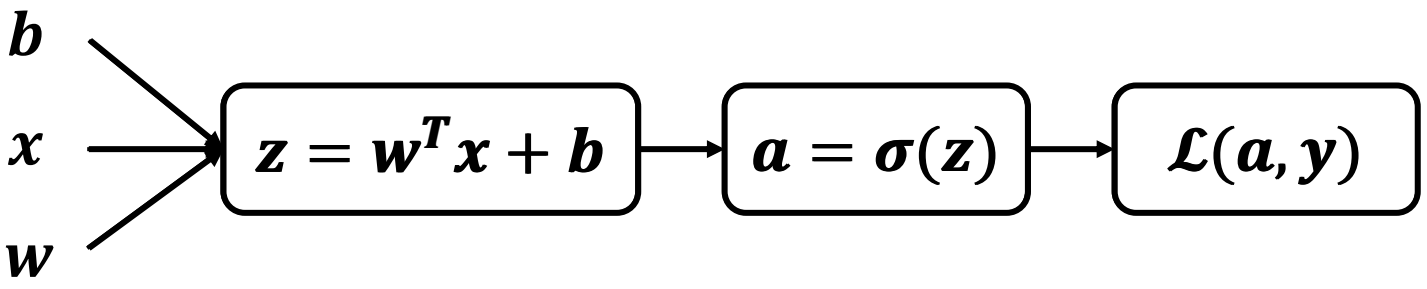




Where $b = 1$, $w = [w_1, w_2, w_3]$, $x = [x_1, x_2, x_3]$, and $\mathcal{L}(a, y)$ is the cost (or ***loss***) function (Cross-entropy loss)

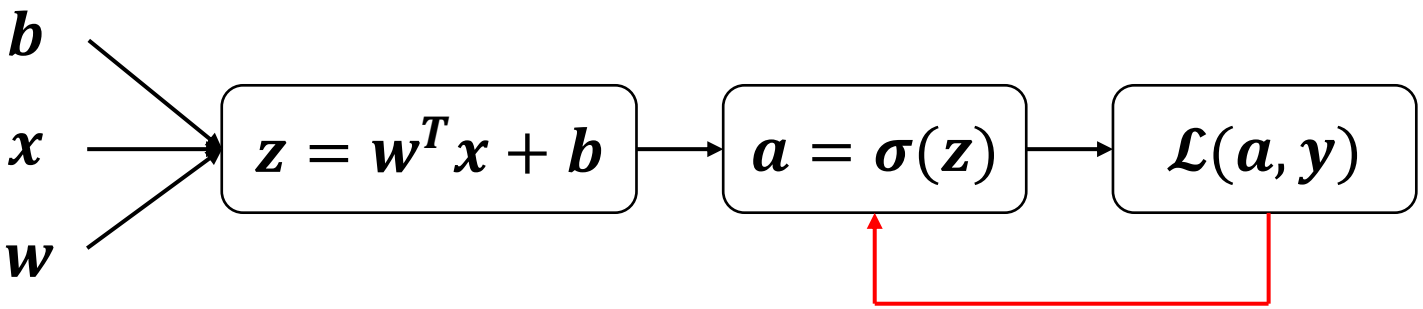
Forward Propagation

- The loss function can be computed by moving from left to right



Backward Propagation

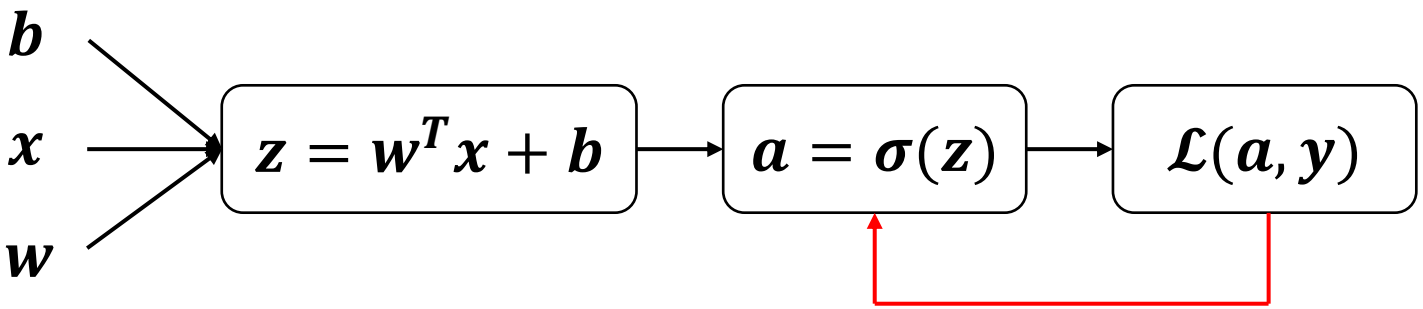
- The derivatives can be computed by moving from right to left



$\dfrac{\partial \mathcal{L}}{\partial a}$ = Partial derivative of $\mathcal{L}$ with respect to a

Backward Propagation

- The derivatives can be computed by moving from right to left

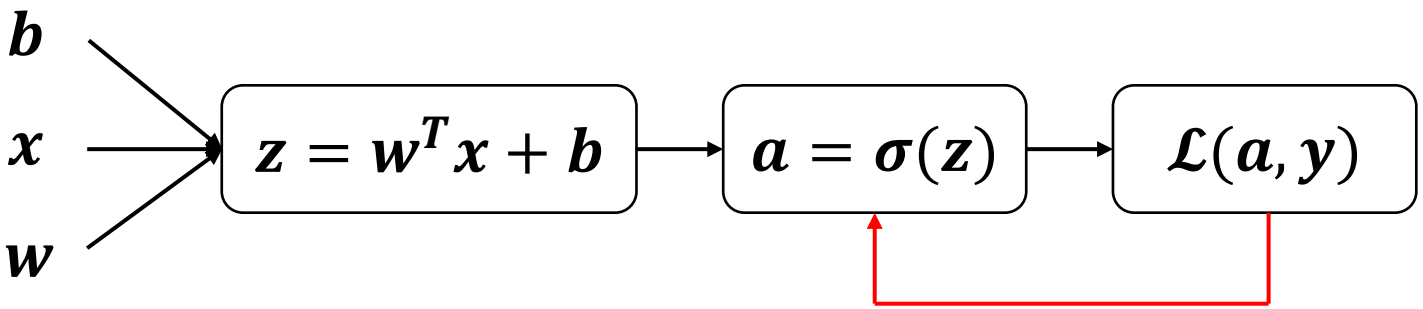


$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial}{\partial a}\left(-y log(a) - (1 - y) log(1 - a)\right)$$

Here we are using the cross-entropy loss

Backward Propagation

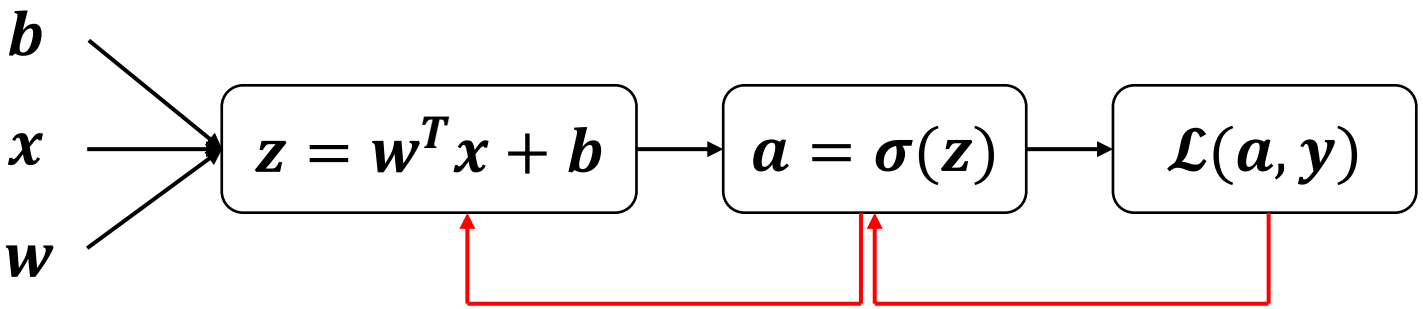
- The derivatives can be computed by moving from right to left



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

Backward Propagation

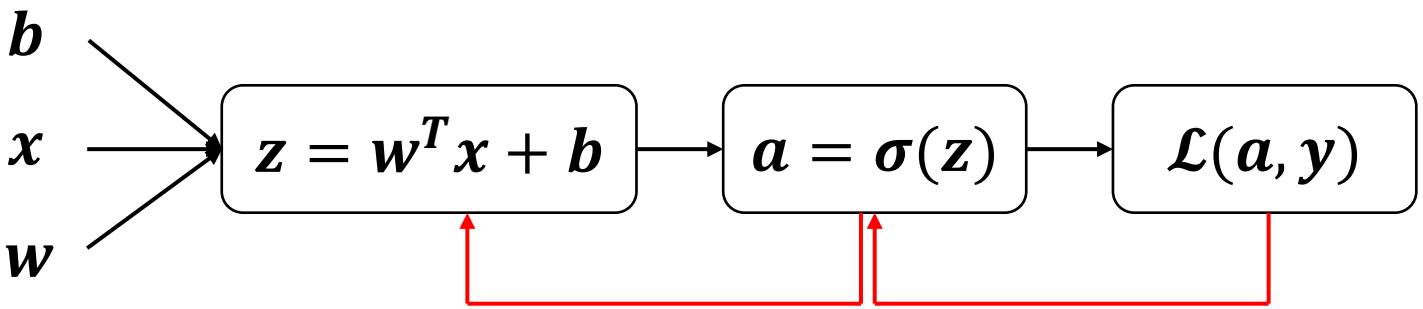
- The derivatives can be computed by moving from right to left



$\dfrac{\partial \mathcal{L}}{\partial z} =$ Partial derivative of $\mathcal{L}$ with respect to z

Backward Propagation

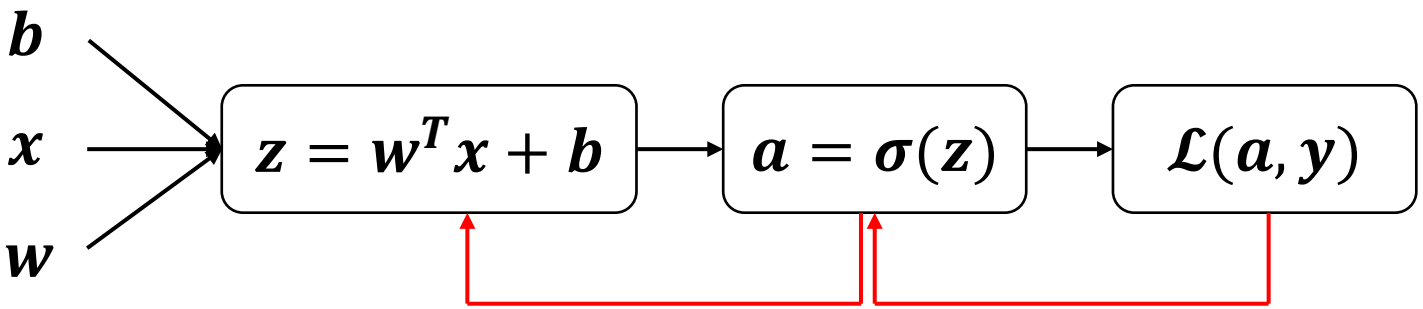
- The derivatives can be computed by moving from right to left



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

Backward Propagation

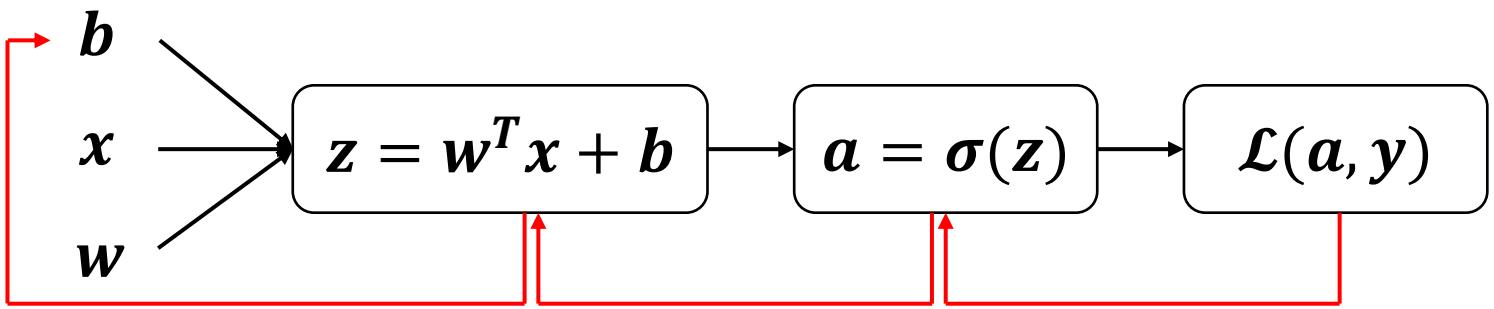
- The derivatives can be computed by moving from right to left



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

Backward Propagation

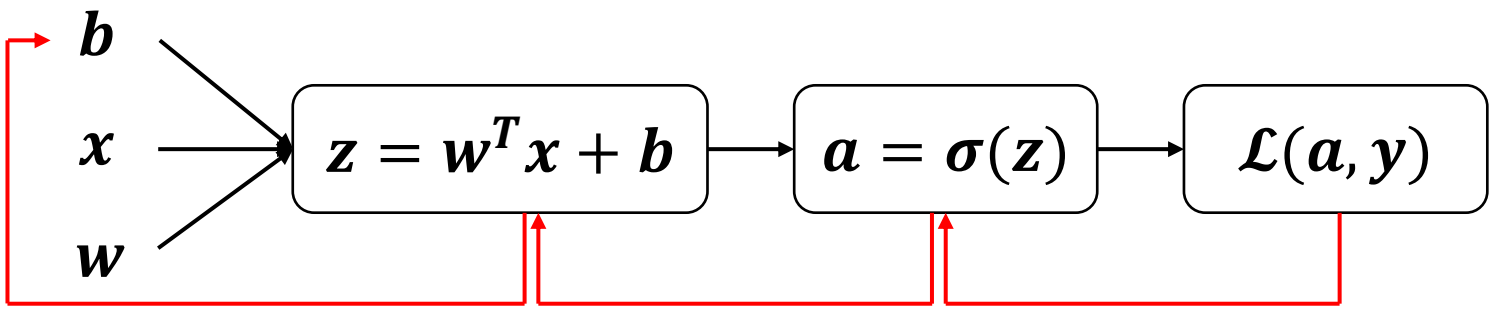
- The derivatives can be computed by moving from right to left



$\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{b}} =$ Partial derivative of $\mathcal{L}$ with respect to $\boldsymbol{b}$

Backward Propagation

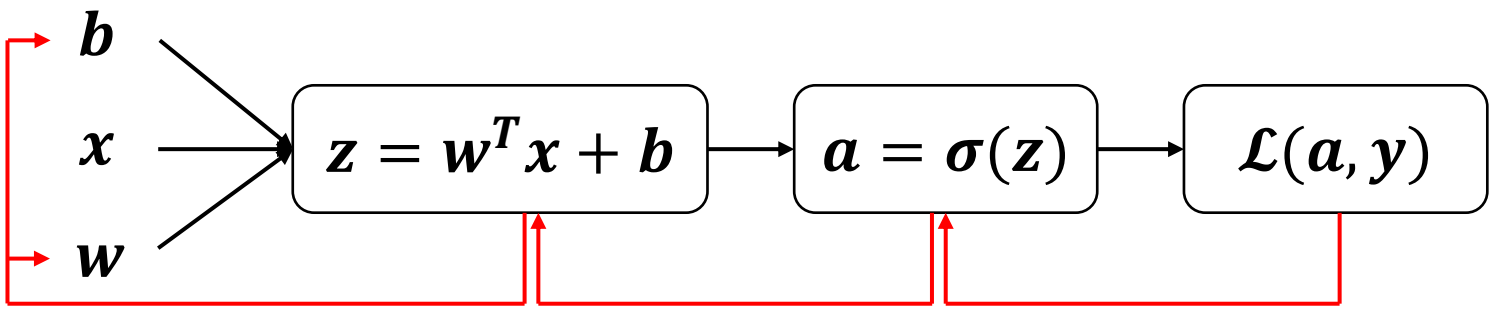
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial b} = (a - y) \times \frac{\partial z}{\partial b} = (a - y) \times 1 = (a - y)$$

Backward Propagation

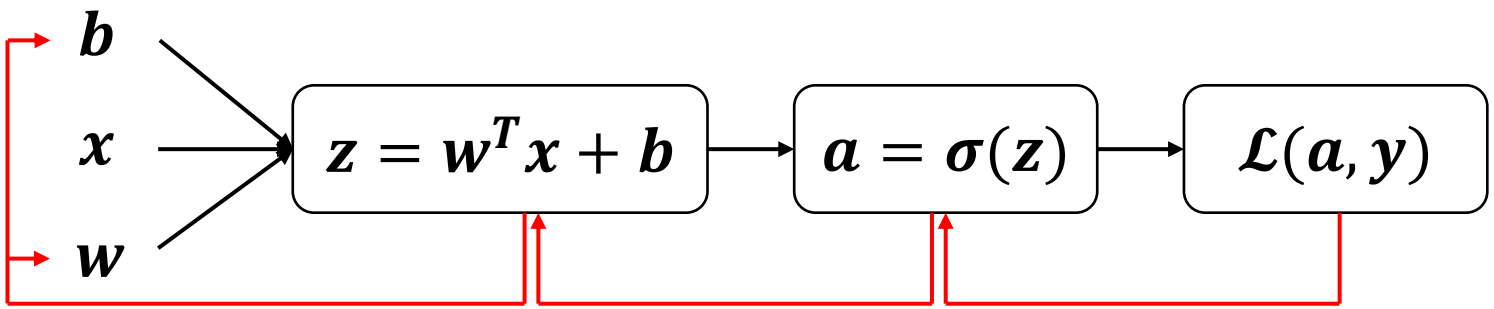
- The derivatives can be computed by moving from right to left



$\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ = Partial derivative of $\mathcal{L}$ with respect to $\boldsymbol{w}$

Backward Propagation

- The derivatives can be computed by moving from right to left



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

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

$$\boldsymbol{dz} = \underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}} = \boldsymbol{a} - \boldsymbol{y}$$

We will denote this as $\boldsymbol{dz}$ for simplicity

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

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

$$\boldsymbol{db} = \frac{\partial \mathcal{L}}{\partial b} = \boldsymbol{a} - \boldsymbol{y}$$

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

Gradient Descent For Logistic Regression

- **Outline**:
  - Have a loss function $\mathcal{L}(\boldsymbol{w}, \boldsymbol{b})$, where $\boldsymbol{w} = [\boldsymbol{w_1}, \dots, \boldsymbol{w_m}]$ and $\boldsymbol{b} = \boldsymbol{w_0}$
  - Start with some guesses for $\boldsymbol{w_1}, \dots, \boldsymbol{w_m}$
    - It does not really matter what values you start with for $\boldsymbol{w_1}, \dots, \boldsymbol{w_m}$, but a common choice is to set them all initially to zero or random values
  - Repeat until convergence{

$$w_j = w_j - \alpha \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial w_j}$$

$$b = b - \alpha \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial b}$$

}

Let us focus on this part

Gradient Descent For Logistic Regression

- **Outline**:
  - Repeat until convergence{

Assuming n examples

$$for\ i = 1\ to\ n:$$

Forward propagation

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$

Backward propagation

$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw = dw + dz^{(i)} x^{(i)}$$
$$db = db + dz^{(i)}$$

Outside the for loop

$$dw = dw/n$$
$$db = db/n$$
$$w = w - \alpha dw$$
$$b = b - \alpha db$$

}

Vectorized version

$$Z = w^T X + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{n} X dZ^T$$
$$db = \frac{1}{n} \sum_{i=1}^{n} dz^{(i)}$$
$$w = w - \alpha dw$$
$$b = b - \alpha db$$

Gradient Descent For Logistic Regression

- **Outline**:
  - Repeat until convergence{

$$\boldsymbol{Z = w^T X + b}$$

$$\boldsymbol{A = \sigma(Z)}$$

$$\boldsymbol{dZ = A - Y}$$

$$\boldsymbol{dw = \frac{1}{n} X dZ^T}$$

$$\boldsymbol{db = \frac{1}{n} \sum_{i=1}^{n} dz^{(i)}}$$

$$\boldsymbol{w = w - \alpha dw}$$

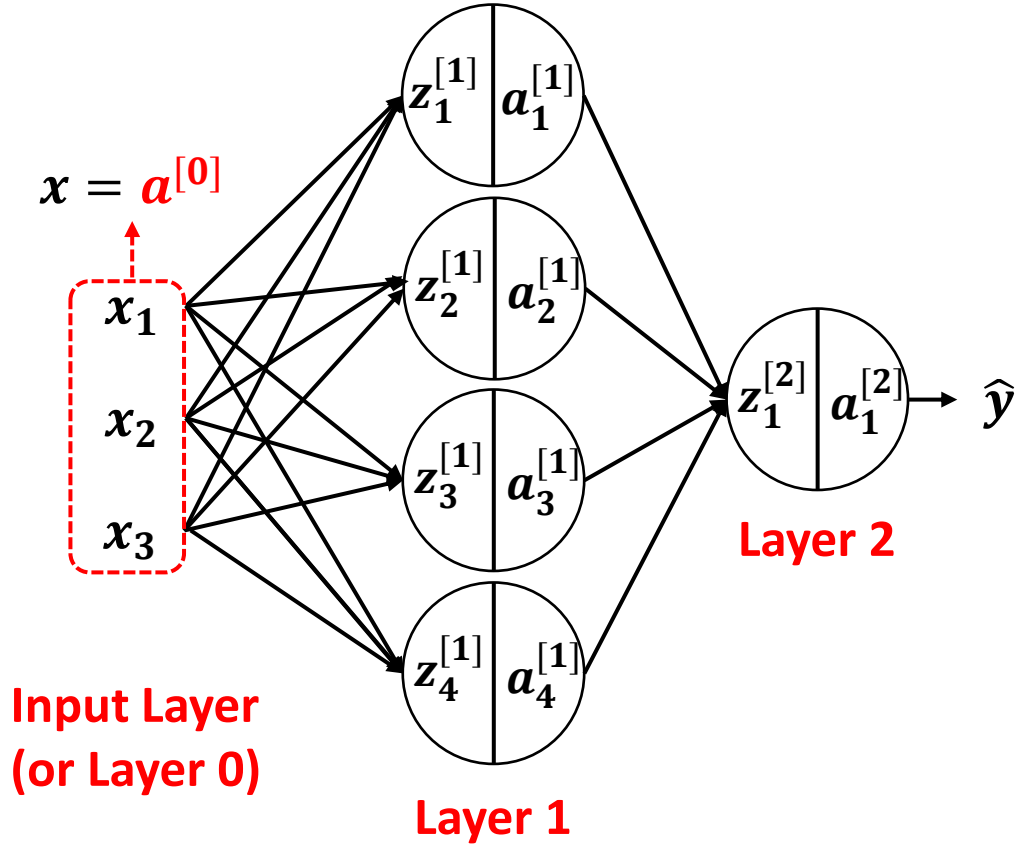
$$\boldsymbol{b = b - \alpha db}$$

}

Computation Graph of A Neural Network

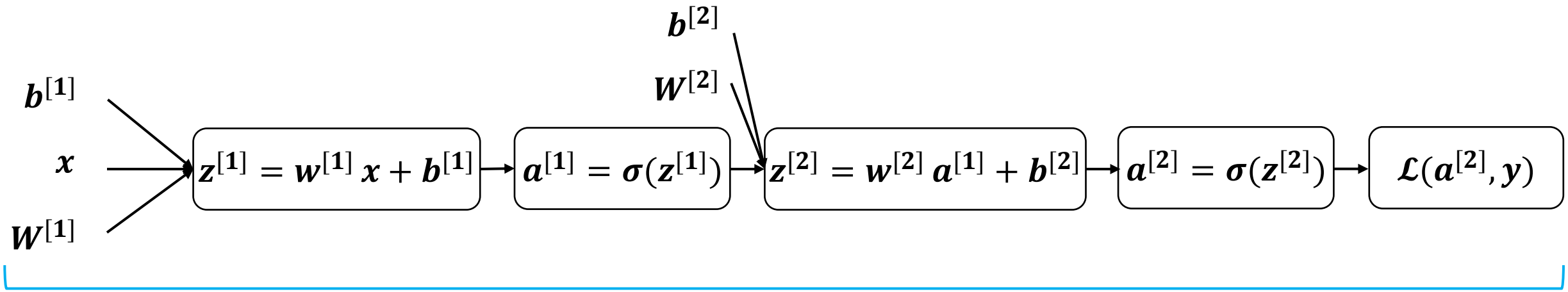
- Similar to logistic regression, we can represent any neural network in terms of a computation graph

A neural network with 2 layers



Computation Graph of A Neural Network

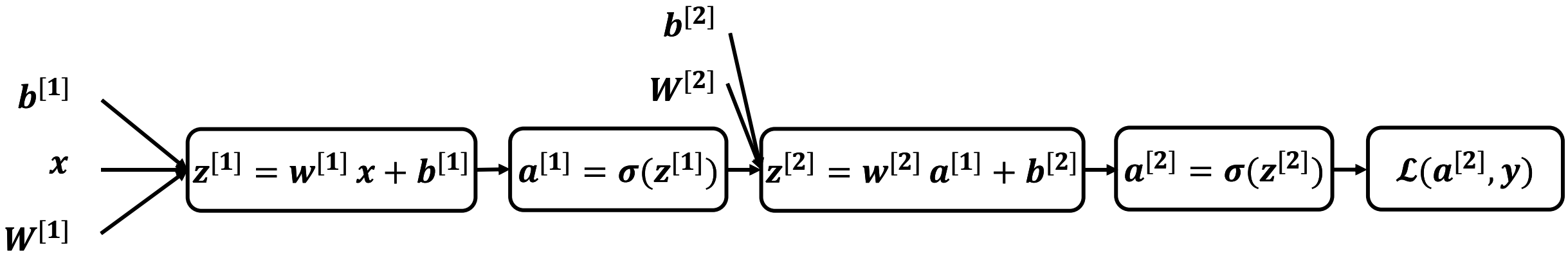
- Similar to logistic regression, we can represent any neural network in terms of a computation graph



The corresponding computation graph

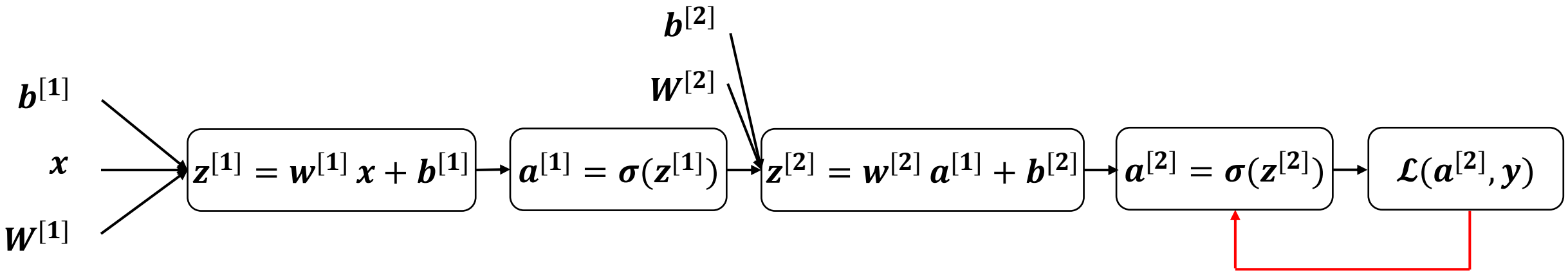
Forward Propagation

- The loss function can be computed by moving from left to right



Backward Propagation

- The derivatives can be computed by moving from right to left

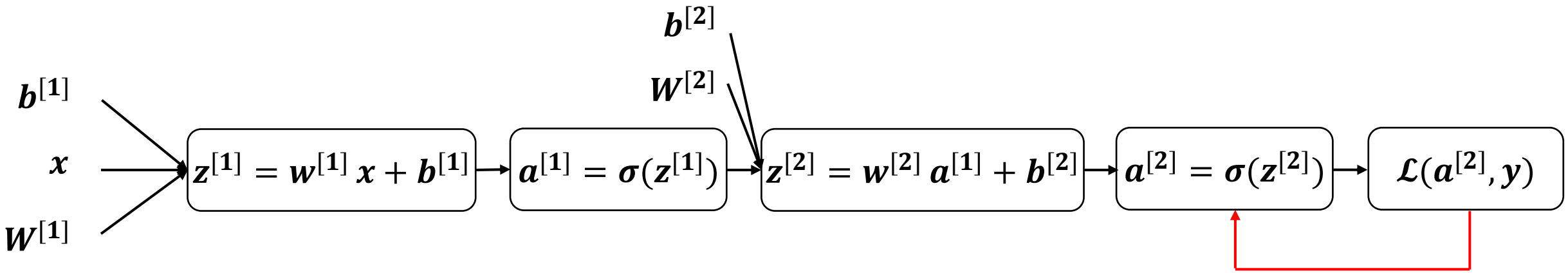


$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } a^{[2]}$$

We are using the cross entropy loss as cost function for illustration

Backward Propagation

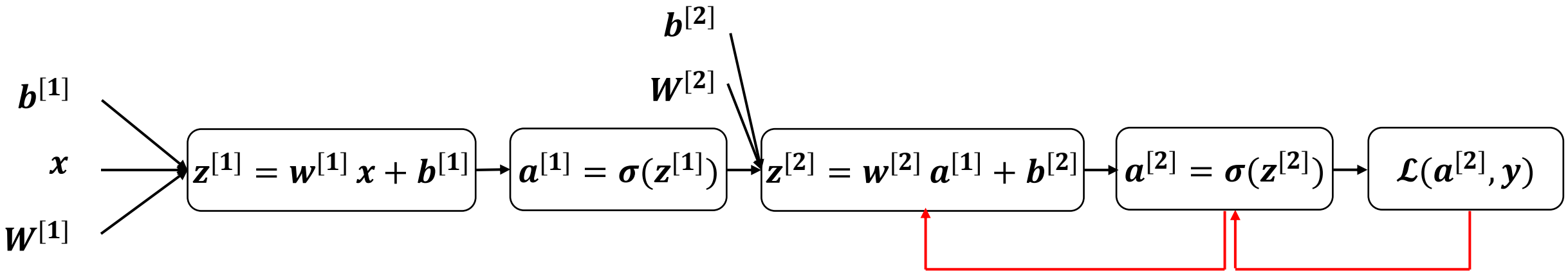
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = \frac{-y}{a^{[2]}} + \frac{(1-y)}{(1-a^{[2]})}$$

Backward Propagation

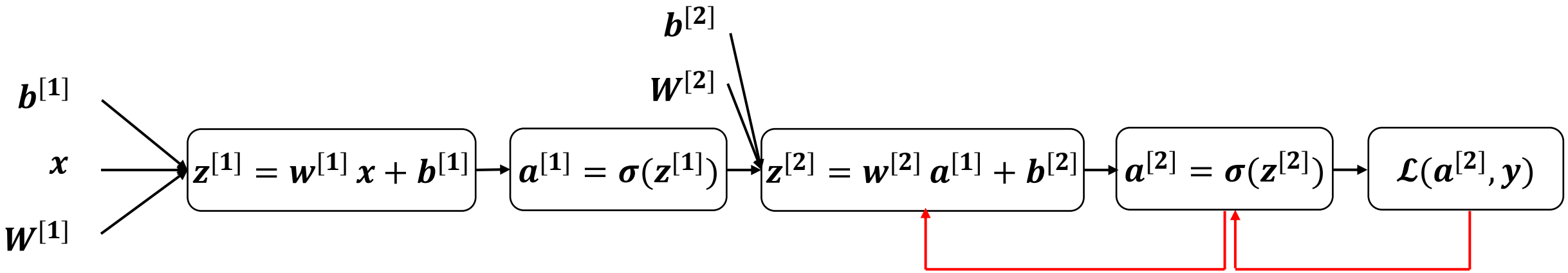
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } z^{[2]}$$

Backward Propagation

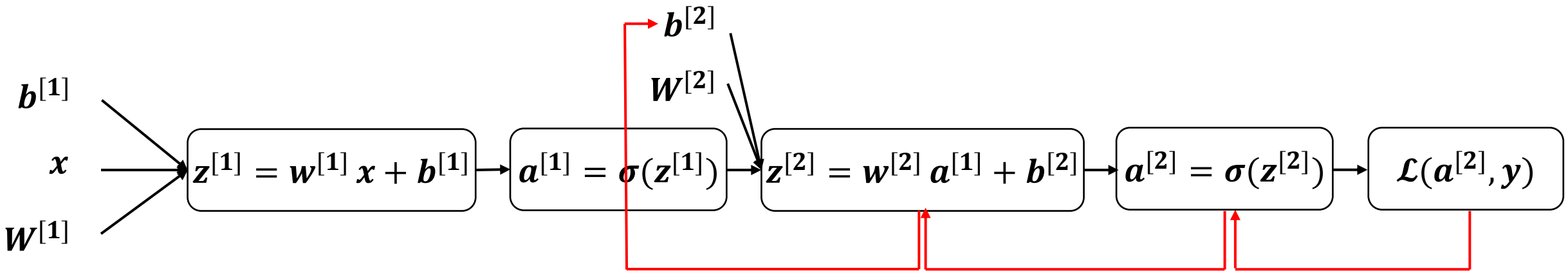
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]} - y$$

Backward Propagation

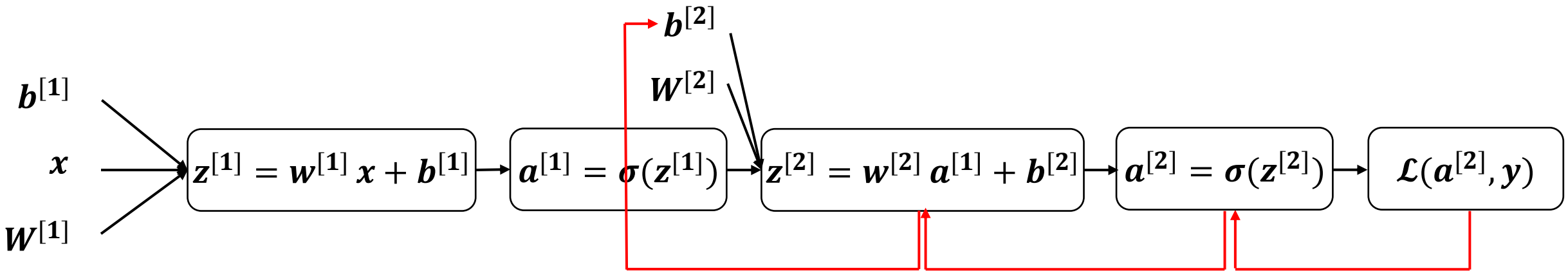
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } b^{[2]}$$

Backward Propagation

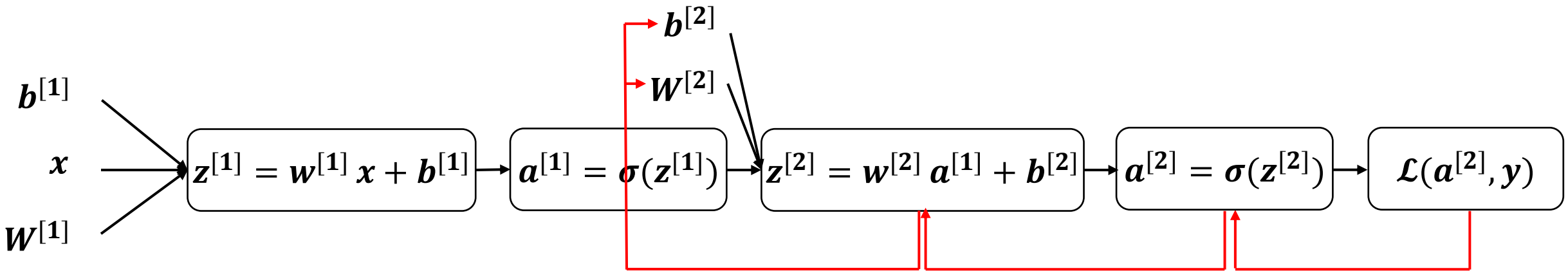
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial b^{[2]}} = a^{[2]} - y$$

Backward Propagation

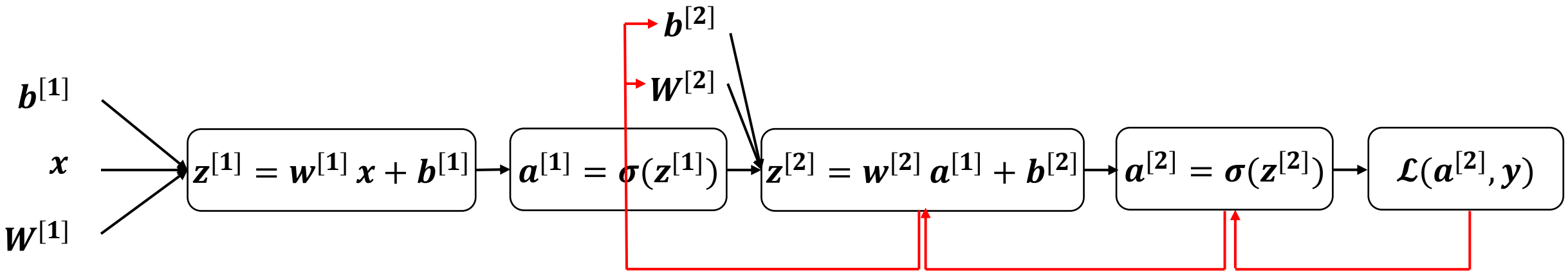
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } W^{[2]}$$

Backward Propagation

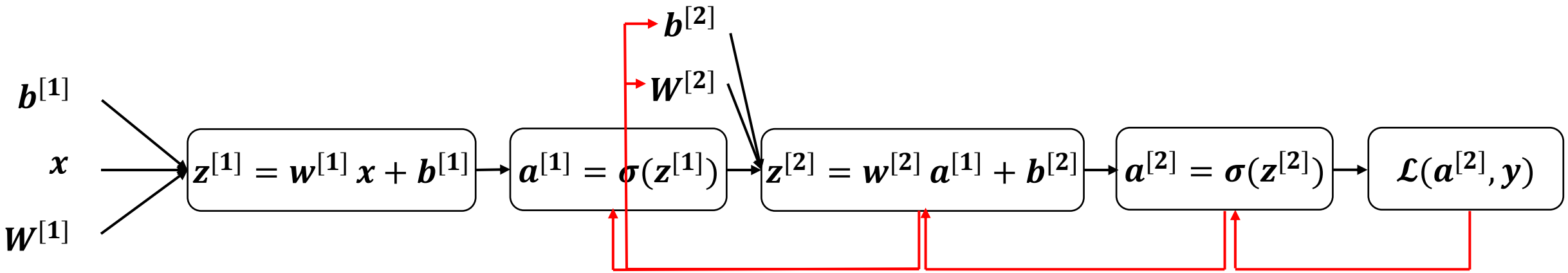
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

Backward Propagation

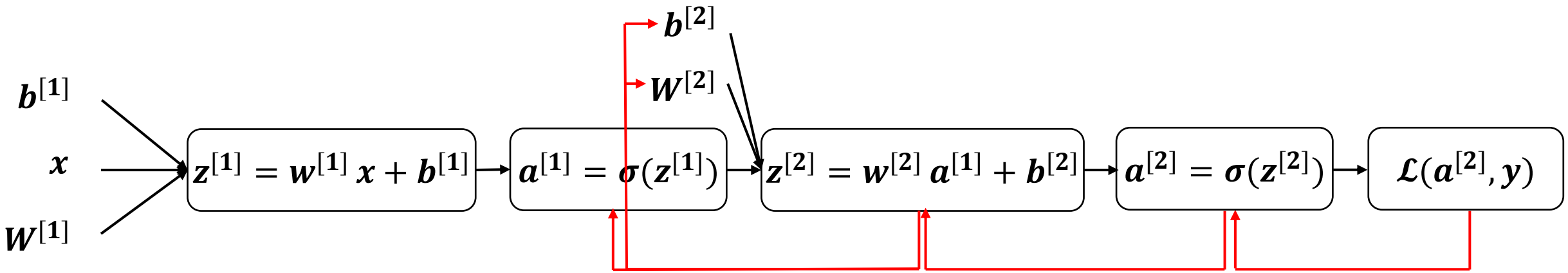
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a^{[1]}} = \text{ Partial derivative of } \mathcal{L} \text{ with respect to } a^{[1]}$$

Backward Propagation

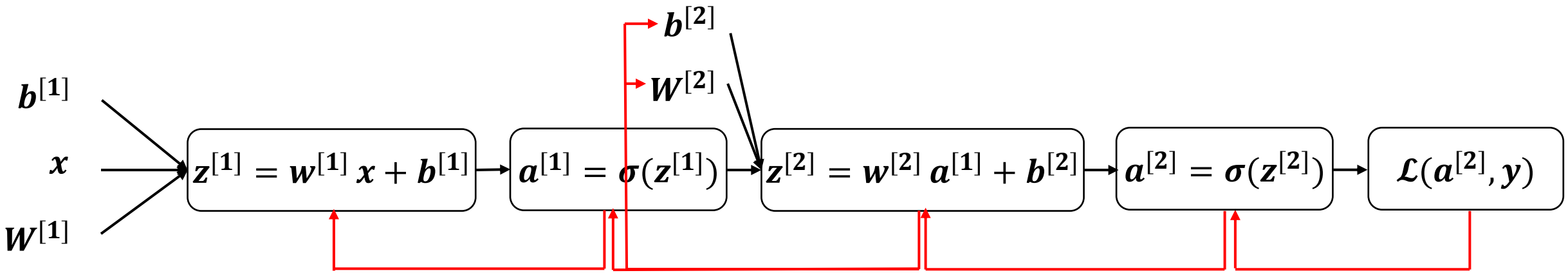
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \quad = (a^{[2]} - y)w^{[2]T}$$

Backward Propagation

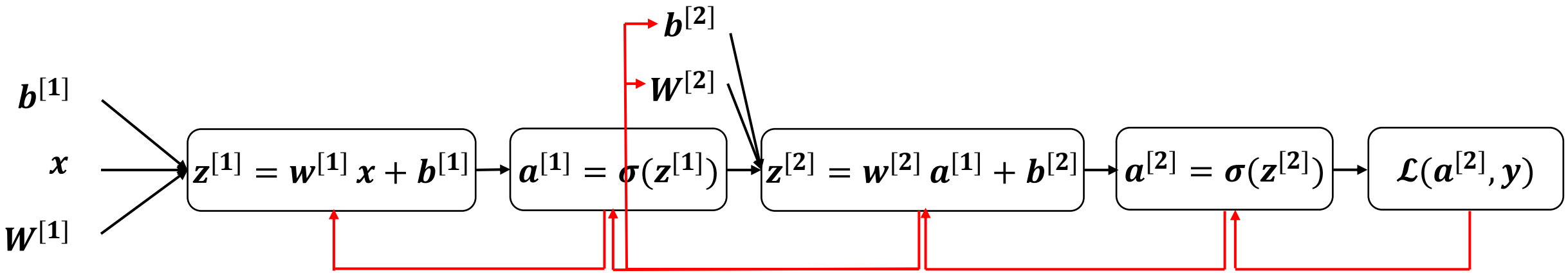
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \text{ Partial derivative of } \mathcal{L} \text{ with respect to } z^{[1]}$$

Backward Propagation

- The derivatives can be computed by moving from right to left

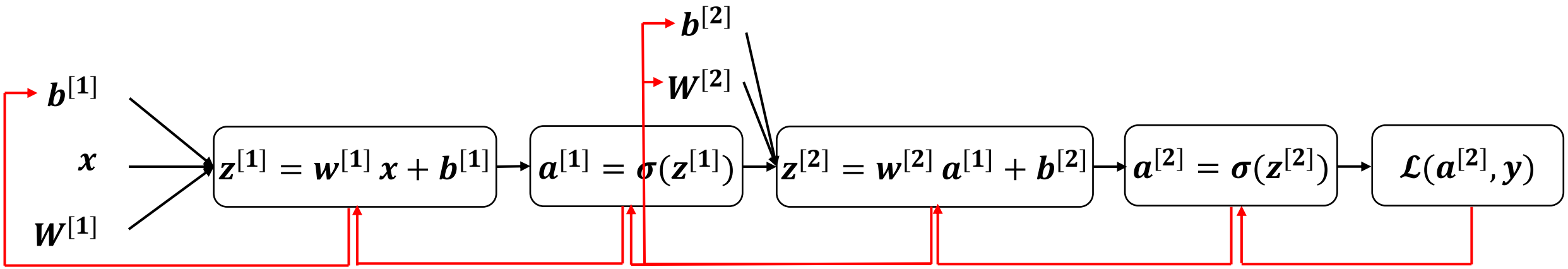


$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} = \left(a^{[2]} - y\right) w^{[2]T} * a^{[1]}\left(1 - a^{[1]}\right)$$

Element-wise product

Backward Propagation

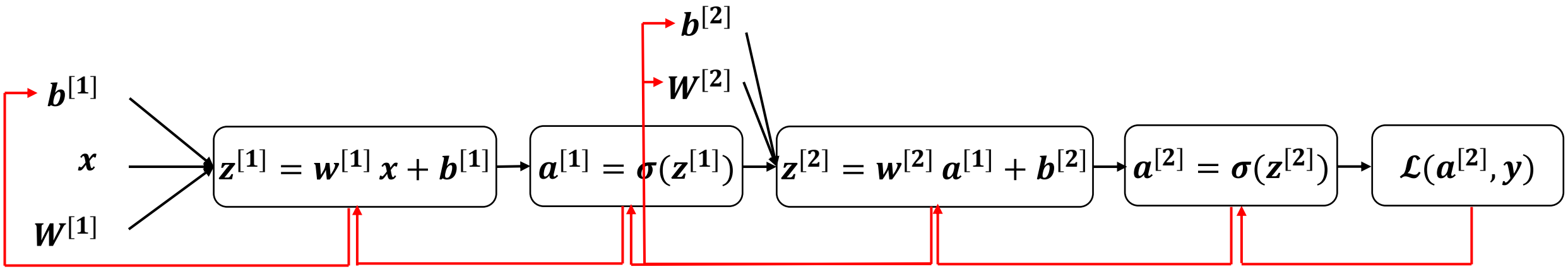
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

Backward Propagation

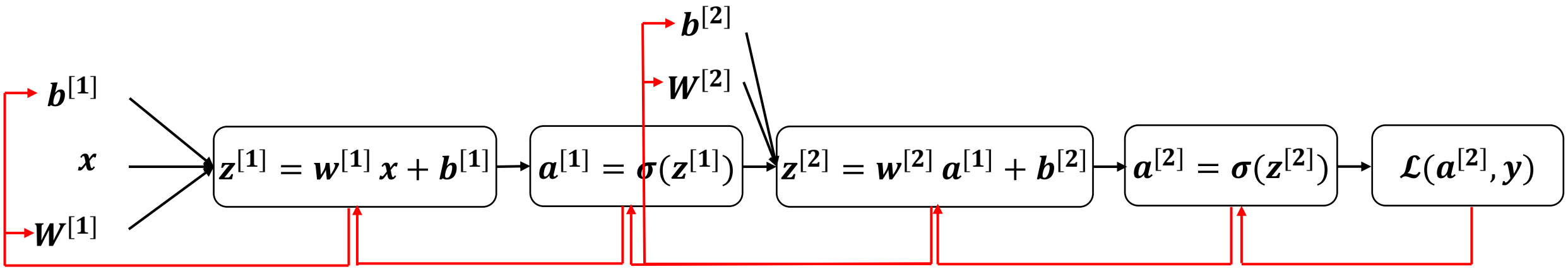
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \left(a^{[2]} - y\right) w^{[2]T} * a^{[1]}\left(1 - a^{[1]}\right)$$

Backward Propagation

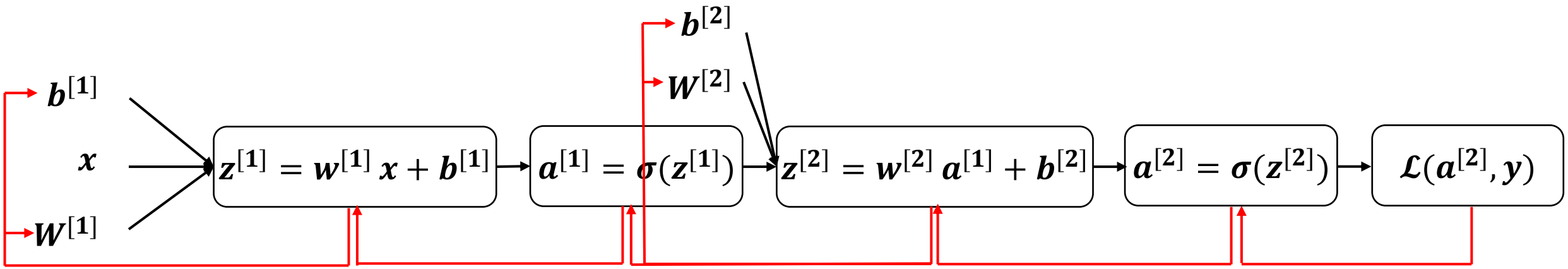
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

Backward Propagation

- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left(\left(a^{[2]} - y\right) w^{[2]T} * a^{[1]}\left(1 - a^{[1]}\right)\right) x^{T}$$

Backward Propagation: Summary

- Here is the summary of the gradients in our given neural network:

$$dz^{[2]} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} = a^{[2]} - y$$

$$db^{[2]} = \frac{\partial \mathcal{L}}{\partial b^{[2]}} = a^{[2]} - y$$

$$dW^{[2]} = \frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

$$dz^{[1]} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

$$db^{[1]} = \frac{\partial \mathcal{L}}{\partial b^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

$$dW^{[1]} = \frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left(dz^{[2]}w^{[2]T} * a^{[1]}\left(1 - a^{[1]}\right)\right)x^{T}$$

How to train a Neural Network: Summary

1. Define the neural network structure
2. Initialize the model's parameters
3. Loop:
    - Implement forward propagation
    - Compute loss
    - Implement backward propagation to get the gradients
    - Update parameters (gradient descent)

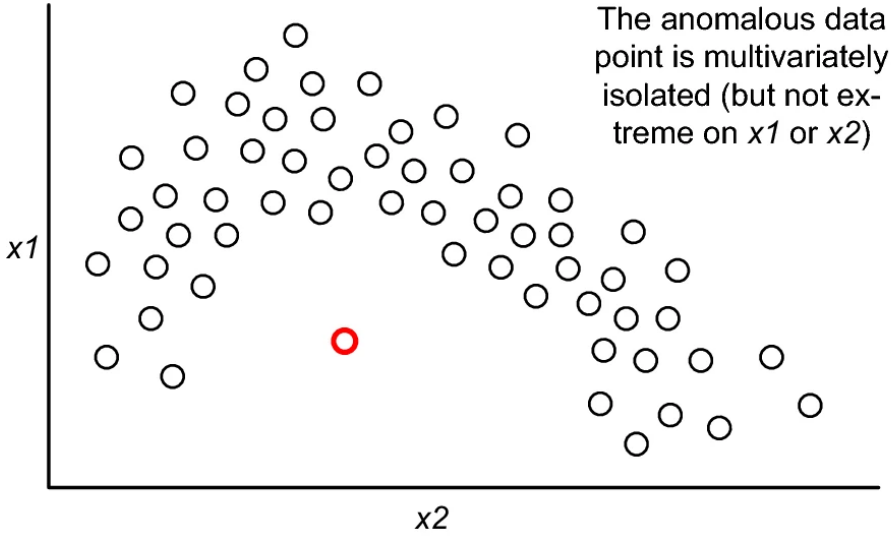
We will delve into more detail and discuss implementation during the lab practice.

What are anomalies?

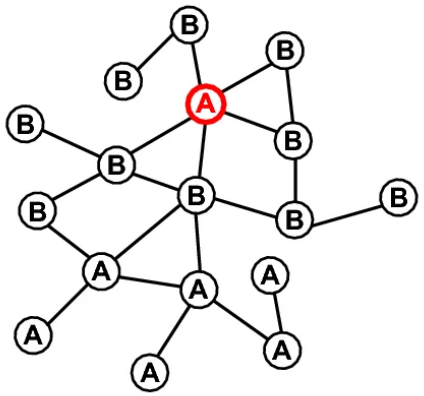
- Anomaly is a data object that deviates significantly from the rest of the objects
- Also referred to as outliers, exceptions, peculiarities, surprise, etc.
  - use “outlier” and “anomaly” interchangeably
- Anomalies translate to significant (often critical) real life entities
  - early signs of system failure
    - if the power company can find anomalies in the electrical power grid and remedy them, potentially avoid expensive damage
  - fraud in the financial industry
    - involves analyzing patterns of typical transactions and identifying deviations from those patterns in a large pool of legitimate transactions

Identification of anomalies

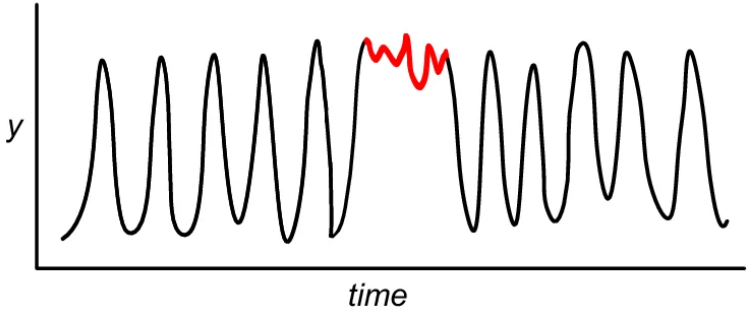
- Red occurrences illustrate a variety of anomalies



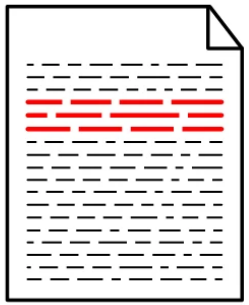
The anomalous data point is multivariately isolated (but not extreme on *x1* or *x2*)



The anomalous vertex has a different class label than its adjacent vertices.



The anomalous time interval deviates from the cyclical pattern.



The anomalous text section is comprised of unusually long words.

Image Source: On the nature and types of anomalies: a review of deviations in data, Ralph Foorthuis

Types of anomalies

- Global anomalies (Point anomalies)
  - simplest and the most common type
- Contextual anomalies (Conditional outliers)
  - requires background information to determine contextual attributes and contexts
- Collective anomalies
  - requires background information to model the relationship among objects to find groups of anomalies

Global anomalies

- A data object is a **global anomaly** if it deviates significantly from the rest of the data set
- Most anomaly detection methods are aimed at finding global anomalies
- Applications
  - Intrusion detection in computer networks
    - A large number of packages is broadcast in a short time
  - Suspicious transactions in trading systems
    - Transactions that do not follow the regulations

points in region R significantly deviate from the rest of the data set

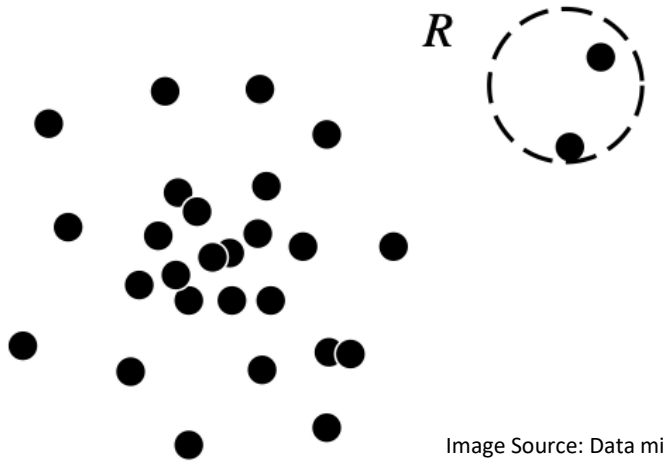


Image Source: Data mining concepts and techniques.

Contextual anomalies

- A data object is a **contextual anomaly** if it deviates significantly with respect to a specific context of the object. (Conditional outliers)
- *"The temperature today is 2◦C. Is it abnormal (i.e., an anomaly)*?"
- The context has to be specified as part of the problem definition
  - Contextual attributes: define context. e.g. location and time of year
  - Behavioral attributes: define characteristics. e.g. temperature and humidity
- Global anomaly detection can be regarded as a special case of contextual anomaly detection
  - where the set of contextual attributes is empty
- Application: credit card fraud detection

Collective anomalies

- A subset of data objects forms a **collective anomaly** if the objects as a whole deviate significantly from the entire data set
  - the individual data objects may not be anomalies
- *"1 order is delayed vs 100 orders are delayed on a single day"*
- Applications
  - Multiple denial-of-service packages as a whole
  - Many transactions of the same stock in a short period
- Consider the behavior of individual objects AND behavior of groups of objects

The black objects form a collective outlier

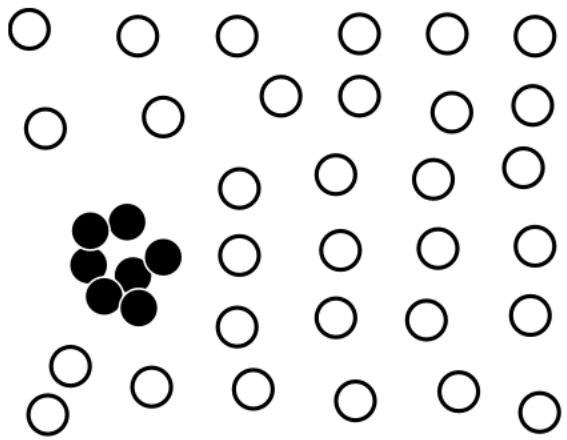


Image Source: Data mining concepts and techniques.

Anomaly detection

- In a general context:
  - Any method for finding events that don’t conform to an expectation
- In the context of network and host security
  - anomaly detection refers to identifying unexpected intruders or breaches
- Historically, the field of statistics tried to find and remove outliers as a way to improve analyses
- There are now many fields where the anomalies are the objects of greatest interest
- The rare events may be the ones with the greatest impact, and often in a negative way

Key applications of anomaly detection

- Fraud detection
- Industrial damage detection
- Cyber Intrusion detection
- …

Fraud Detection

- Fraud detection refers to detection of criminal activities occurring in commercial organizations
- Malicious users might be
  - actual customers of the organization, someone posing as a customer (also known as identity theft), employees of the company
- Types of fraud
  - Credit card fraud
  - Insurance claim fraud
  - Insider trading
- Key challenges
  - Fast and accurate real-time detection
  - Misclassification cost is very high

Image Source: https://www.milanfintechsummit.com/europe-card-fraud-digital-payments/

Industrial damage detection

- Detection of different faults and failures in complex industrial systems, structural damages, suspicious events in video surveillance, abnormal energy consumption, etc.
  - Example: aircraft safety
    - Anomalous Aircraft (Engine) / Fleet Usage
    - Anomalies in engine combustion data
    - Total aircraft health and usage management
- Key challenges
  - Data is extremely huge, noisy and unlabeled
  - Detecting anomalous events typically require immediate intervention

Image Source:
https://www.pointhacks.com.au/future-of-a380/

Intrusion detection

- Intrusion Detection
  - Process of monitoring the events occurring in a computer system or network and analyzing them for intrusions
  - using thresholds and heuristics is a simple way to detect intrusions and anomalies
- e.g.: intrusion detection with heuristics
  - suppose we define 10 queries/hour = upper limit of normal use for a certain database
  - Each time the database is queried, invoke a function is_anomaly(user) with the user's ID as an argument
  - If the user queries the database for an 11th time within an hour, the function will indicate that access as an anomaly

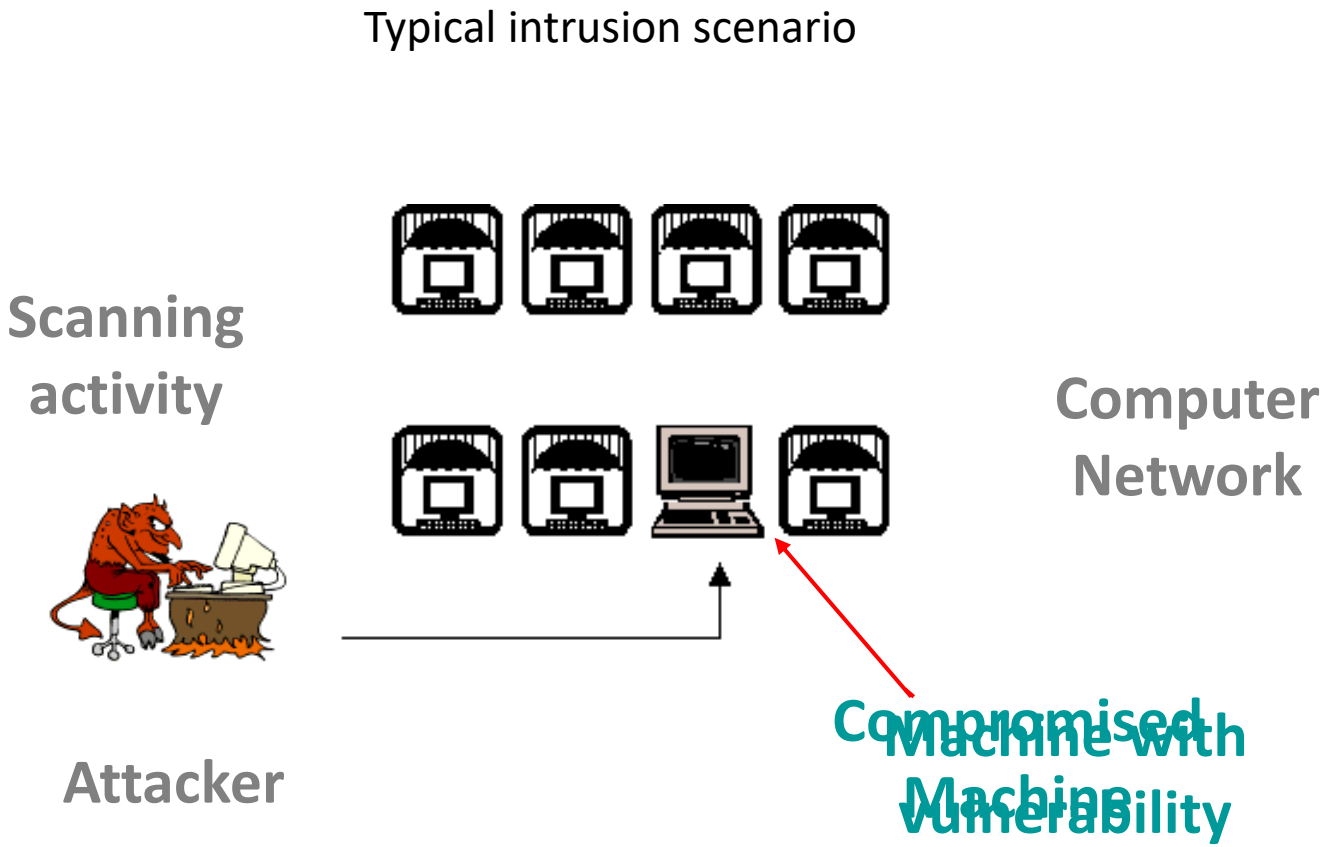
Intrusion Detection

- Although threshold-based anomaly detection logic is easy to implement, some questions arise:
  - How do we set the threshold?
  - Could some users require a higher threshold than others?
  - Could there be times when users legitimately need to access the database more often?
  - How frequently do we need to update the threshold?
  - Could an attacker exfiltrate data by taking over many user accounts, thus requiring a smaller number of accesses per account?
- Instead, using machine learning can help us to avoid having to come up with answers to all of these questions, letting the ***data*** define the solution to the problem

What are intrusions

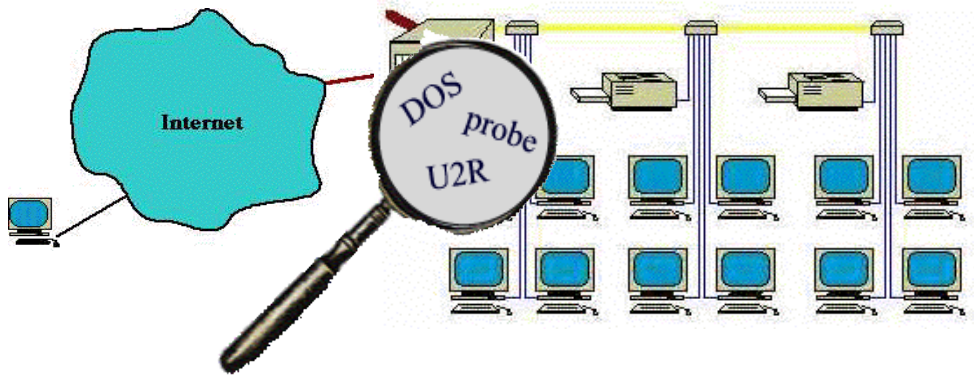
- Intrusions are actions that attempt to bypass security mechanisms of computer systems. They are usually caused by
  - Attackers accessing the system from Internet
  - Insider attackers - authorized users attempting to gain and misuse non-authorized privileges

Typical intrusion scenario



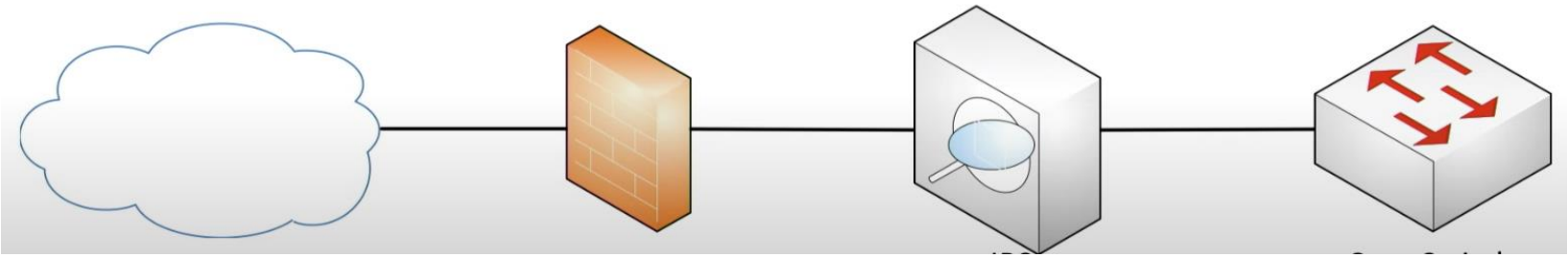
Intrusion Detection System

- Intrusion Detection System
  - monitors a network or systems for malicious activity or policy violations
  - raises alarms when possible intrusion happens
- In terms of location deployed
  - Network Intrusion Detection System (NIDS)
  - Host Intrusion Detection System (HIDS)
- In terms of detection methods
  - signature-based intrusion detection
  - anomaly-based intrusion detection



IDS – Network-based

- **NIDS** - installed on a network and monitor traffic for signs of attacks
- deployed at a strategic point or points within the network
  - e.g. at the network perimeter or at key junctions
  - monitor inbound and outbound traffic to and from all the devices on the network.
  - If NIDS detects a potential intrusion, it alerts the network administrator and take action to prevent the attack from succeeding
  - An example of a NIDS is installing it on the subnet where firewalls are located in order to see if someone is trying to crack the firewall.



IDS – Host-based

- **HIDS -** installed on individual hosts, and monitor the host's activity for signs of attacks
- computers or devices in the network with direct access to both the internet and the organization's internal network
  - e.g. servers or workstations
  - are typically used to monitor the host's logs, file system, and system calls, and can alert the administrator if they detect any suspicious activity
  - HIDS are often used to complement NIDS by providing additional visibility and protection for individual hosts on the network
- Both NIDS and HIDS are important tools for detecting and preventing intrusions, and many organizations use a combination of both to provide comprehensive protection for their networks

IDS - signature-based

- **Signature-based** intrusion detection
  - uses a database of known attack patterns or "signatures" to identify potential threats. This type of IDS is effective at detecting known attacks, but it may not be able to identify new or unknown threats
  - Signature-based IDS detects the attacks on the basis of the specific patterns
    - file hashes
    - byte sequences in network traffic
    - malicious instruction sequence that is used by the malware
- Signature-based IDS can easily detect the attacks whose pattern already exists in the system but it is quite difficult to detect the new malware attacks as their pattern is not known

IDS - anomaly-based

- **Anomaly-based** intrusion detection
  - identify unusual or suspicious behavior in network traffic. Anomaly-based detection is capable of alerting on unknown suspicious behavior
  - Anomaly-based detection involves first training the system with a normalized baseline and then comparing activity against that baseline
  - Once event appear out of the ordinary an alert is triggered
    - Alerts can be triggered by anything that does not align with the normalized baseline, e.g. a user logging in during non-business hours, a flood of new IP addresses attempting to connect to the network, or new devices being added to a network without permission
  - Machine learning-based method has a better-generalized property in comparison to signature-based IDS as these models can be trained according to the applications and hardware configurations

ML for intrusion detection: motivation

- Traditional intrusion detection system IDS tools (e.g. SNORT) are based on signatures of known attacks
  - Snort applies rules to monitored traffic and issues alerts when it detects certain kinds of suspicious activity on the network.
- Limitations
  - Signature database has to be manually revised for each new type of discovered intrusion
  - Substantial latency in deployment of newly created signatures across the computer system
  - They cannot detect emerging cyber threats

ML for intrusion detection

- Machine Learning and AI-driven techniques can alleviate these limitations – potential to recognize unforeseen attacks.
- Anomaly detection is based on profiles that represent normal behaviour of users, hosts, or networks, and detecting attacks as significant deviations from this profile
  - Major challenges: lack of sufficient amount of high-quality anomalous data
  - Major limitation: possible high false alarm rate, since detected deviations do not necessarily represent actual attacks
    - previously unseen (yet legitimate) system behaviors may also be recognized as anomalies
  - Major approaches: statistical methods, clustering, neural networks, support vector machines, …

Abnormal Behavior

- Some elements to consider in order to detect anomalous traffic
  - The number of connections to and from a specific host
  - Unusual remote communication ports or unexpected traffic patterns
  - Unusual traffic peaks occurring at particular times of the day (for example, traffic carrying on during the night)
  - Communication bandwidth heavily occupied by particular hosts within the network
- Abnormal behavior in network traffic
  - can be a sign of a security threat, such as a cyber attack or malware infection
  - can also indicate other problems, such as network congestion or faulty equipment

Feature engineering for anomaly detection

- As with any other task in machine learning, selecting good features for anomaly detection is important
  - For example, to detect when a system process has an abnormally high CPU utilization
  - all we need is the CPU utilization metric, which you can extract from most basic system monitoring modules
  - However, many use cases will require us to generate features on which to apply anomaly detection algorithms
- e.g.
  - host intrusion detection
  - network intrusion detection

Features engineering for HIDS

- Malware is the most dominant threat vector for hosts (e.g. servers, desktops) in many environments
- We can detect the malware by collecting system-level activity signals and looking for indicators of compromise (IoCs) in the data
  - Running processes
  - Active/new user accounts
  - Network connections
  - System scheduler changes
  - Startup operations
  - Temporary file directories
  - Browser extensions …

Features engineering for NIDS

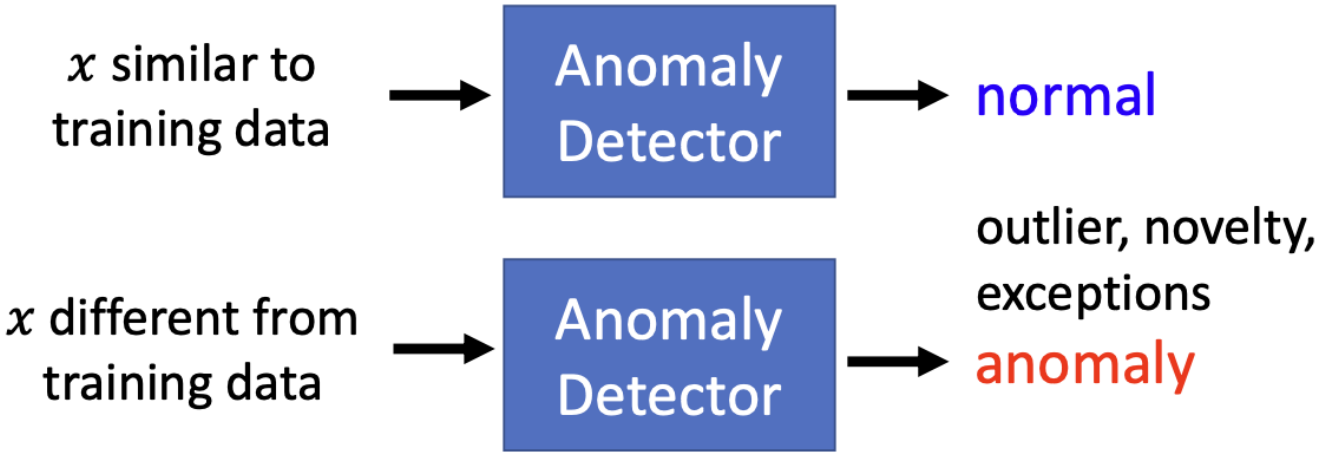
- In networking, a packet is a small segment of a larger message
  - Data sent over the Internet is divided into packets
  - These packets are then recombined by the computer or device that receives them
- There is a difference between
  - extracting network traffic metadata (info about packet's contents, origin, and destination)
  - inspecting network traffic content (the actual data)
- Deep packet inspection (DPI)
  - the process of examining the data encapsulated in network packets
  - DPI is capable of collecting signals that can help detect spam, malware, intrusions, and subtle anomalies
  - Real-time streaming DPI is a challenging problem in computer science
    - because of the computational requirements necessary to decrypt, disassemble, and analyze packets going through a network

Outline

- What are anomalies
- Types of anomalies
- Applications of anomaly detection
- Machine learning for intrusion detection
- Anomaly detection techniques
  - Problem definition
  - Categories
    - With labels – train classifiers
    - Without labels – statistical methods

Anomaly detection - problem definition

- Given a set of training data $\{x^1, x^2, \ldots, x^N\}$
- Find a model detecting whether input x is similar to training data



- Different approaches use different ways to determine the similarity.

Categories

- Given training data $\{x^1, x^2, \dots, x^N\}$, two categories:

- 1. With labels $\{y^1, y^2, \dots, y^N\}$
  - Supervised methods: train a classifier, e.g. logistic regression, nn, …
  - The classifier can output “unknown”
  - (Note that none of the training data is labelled as “unknown”)

- 2. Without labels
  - Statistical approaches

Supervised methods

- Sample of data for analysis is given with domain expert–provided labels
- Anomaly detection is modeled as a classification problem
- Task: learn a classifier that can recognize anomalies
- Challenge
  - The two classes (normal vs anomalies) are imbalanced,
  - i.e. the number of normal samples likely far exceeds the number of outlier samples.
- Anomaly detection using a one-class model
  - A classifier is built to describe only the normal class.
  - Any samples that do not belong to the normal class are regarded as anomalies.

Supervised methods

- Suppose you have trained a classifier using the labelled data
- How to use the classifier
  - Given input **x**
  - In addition to the prediction, output a confidence score **c**
  - Compare it with threshold λ

- Anomaly detection:

$$\mathbf{x} = \begin{cases} normal, & c(x) > \lambda \\ anomaly, & c(x) \leq \lambda \end{cases}$$

Example of confidence score

y^3 y^2 y^1 y^4

Confidence c: can be the maximum among the **softmax** score

Pokemon Classifier

y^1	$\boxed{0.97}$
y^2	0.01
y^3	0.01
y^4	0.01

Very confident
Normal

Pokemon Classifier

y^1	0.24
y^2	$\boxed{0.26}$
y^3	0.25
y^4	0.25

Not confident
Anomaly

Softmax

- Motivation: sigmoid produces a decimal between 0 and 1.0.
  - E.g, email classifier, output of 0.8 suggests: 80% = spam, 20% = ham.
  - the sum of the probabilities = 1.0
- Softmax function: **multiclass** classification
  - extends the idea into a multi-class case.
  - assigns decimal probabilities to each class in a multi-class problem.
  - The decimal probabilities must add up to 1.0.

$$\sigma_i = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}$$

- σ_i is the output vector. Each element of the output vector specifies the probability of this element. The sum of all the elements in the output vector is 1.0. The output vector contains the same number of elements as the input vector, z.
- z is the input vector. Each element of the input vector contains a floating-point value.
- K is the number of elements in the input vector (and the output vector).

Softmax - example

- Suppose the Input vector z is: [1.2, 2.5, 1.8]
- computes the denominator as: $\text{denominator} = e^{1.2} + e^{2.5} + e^{1.8} = 21.552$
- The softmax probability of each element is therefore:

$$\sigma_1 = \frac{e^{1.2}}{21.552} = 0.154$$

$$\sigma_2 = \frac{e^{2.5}}{21.552} = 0.565$$

$$\sigma_1 = \frac{e^{1.8}}{21.552} = 0.281$$

$$\sigma_i = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}$$

- The output vector is: $\sigma = [0.154, 0.565, 0.281]$

Softmax Function vs Sigmoid Function

- As mentioned before, the softmax function and the sigmoid function are similar.
  - The softmax operates on a vector while the sigmoid takes a scalar
- In fact, the sigmoid function is a special case of the softmax function for a classifier with only two input classes.
- Can you justify the statement above?
  - Hint: assume the input vector is [x, 0]

calculate the first output element with the usual softmax formula:

$$\sigma(\vec{z})_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} = \frac{e^x}{e^x + e^0} = \frac{e^x}{e^x + 1}$$

Simplifies to

$$\sigma(\vec{z})_1 = \frac{1}{1 + e^{-x}}$$

Example framework

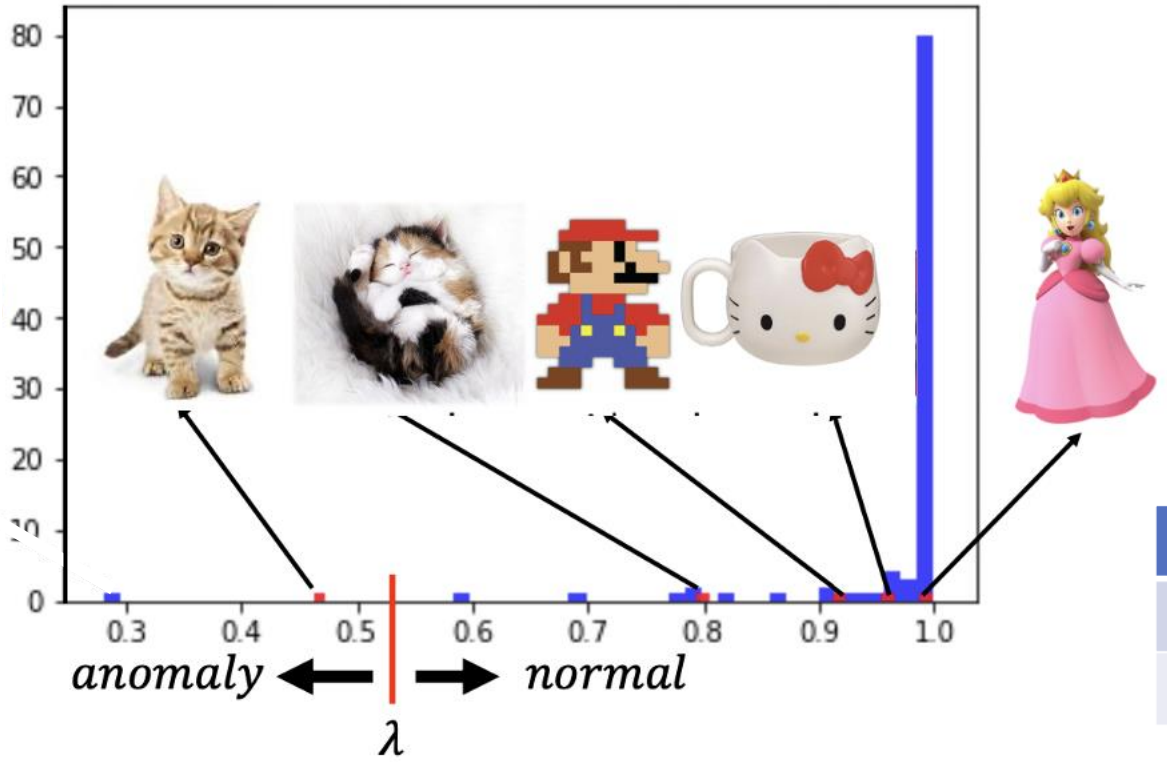
- Images x of characters from Pokemon
  - Each x is labelled by its characters y
  - train a classifier, obtain the confidence score c from the classifier

$$\mathbf{x} = \begin{cases} normal, & c(x) > \lambda \\ anomaly, & c(x) \leq \lambda \end{cases}$$

- Evaluate the performance of anomaly detector
  - determine λ and other hyperparameters, e.g. if using NN: size of the hidden layers, learning rate, # iterations, # layers
- Given new data x, from pokemon or not

Evaluation

100 pokemons, (normal: blue)
5 anomalies, (anomaly: red)



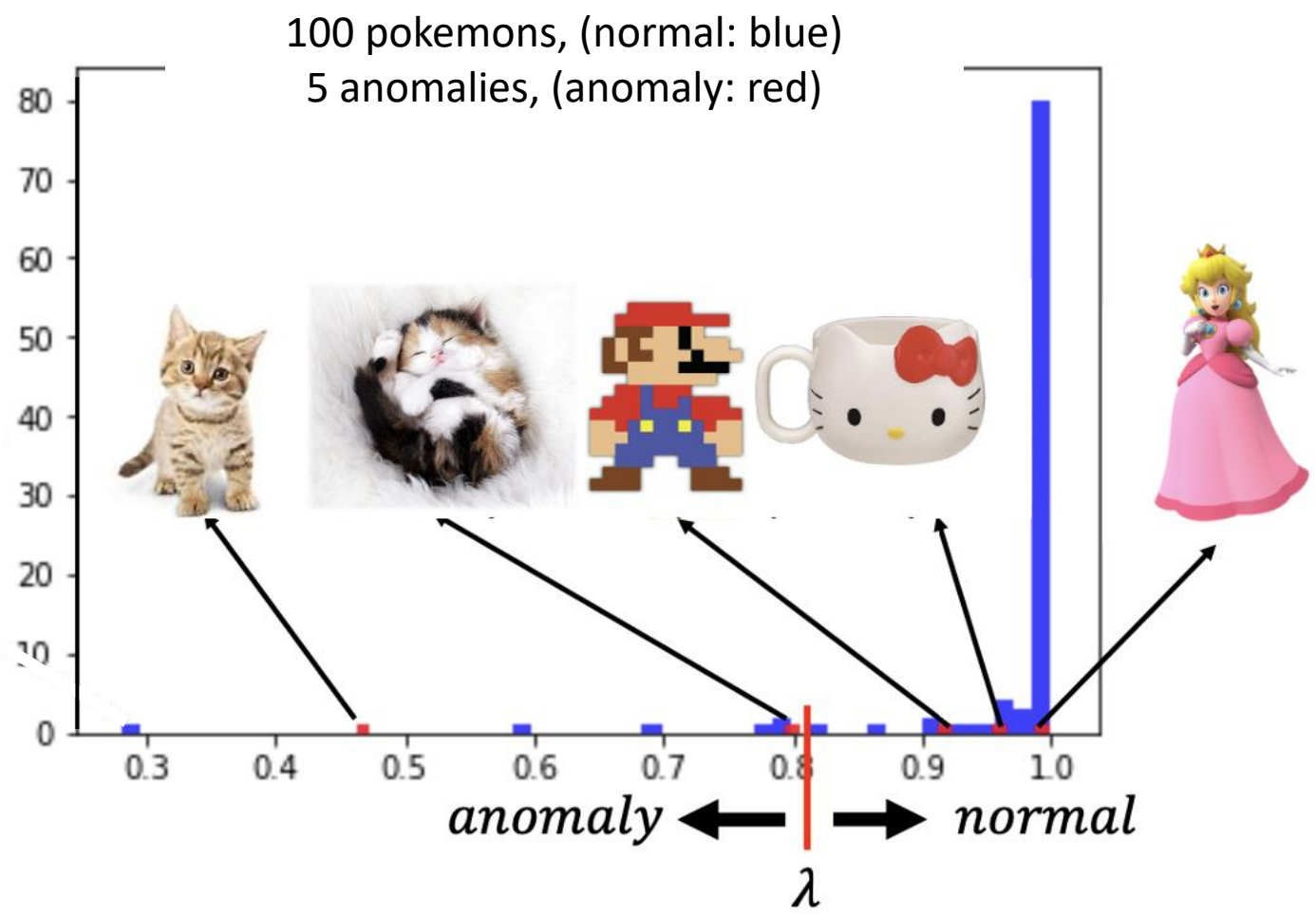
$$\mathbf{x} = \begin{cases} normal, & c(x) > \lambda \\ anomaly, & c(x) \leq \lambda \end{cases}$$

λ depends on specific application

	Anomaly	Normal
Detected	1	1
Not Det	4	99

False alarm

missing



$$\mathbf{x} = \begin{cases} normal, & c(x) > \lambda \\ anomaly, & c(x) \leq \lambda \end{cases}$$

	Anomaly	Normal
Detected	1	1
Not Det	4	99

VS

	Anomaly	Normal
Detected	2	6
Not Det	3	94

Evaluation

Define cost table to measure performance

	Anomaly	Normal
Detected	1	1
Not Det	4	99

Cost = 104 ✓

Cost = 401

	Anomaly	Normal
Detected	2	6
Not Det	3	94

Cost = 603

Cost = 306 ✓

Cost	Anomaly	Normal
Detected	0	100
Not Det	1	0

Cost Table A

Cost	Anomaly	Normal
Detected	0	1
Not Det	100	0

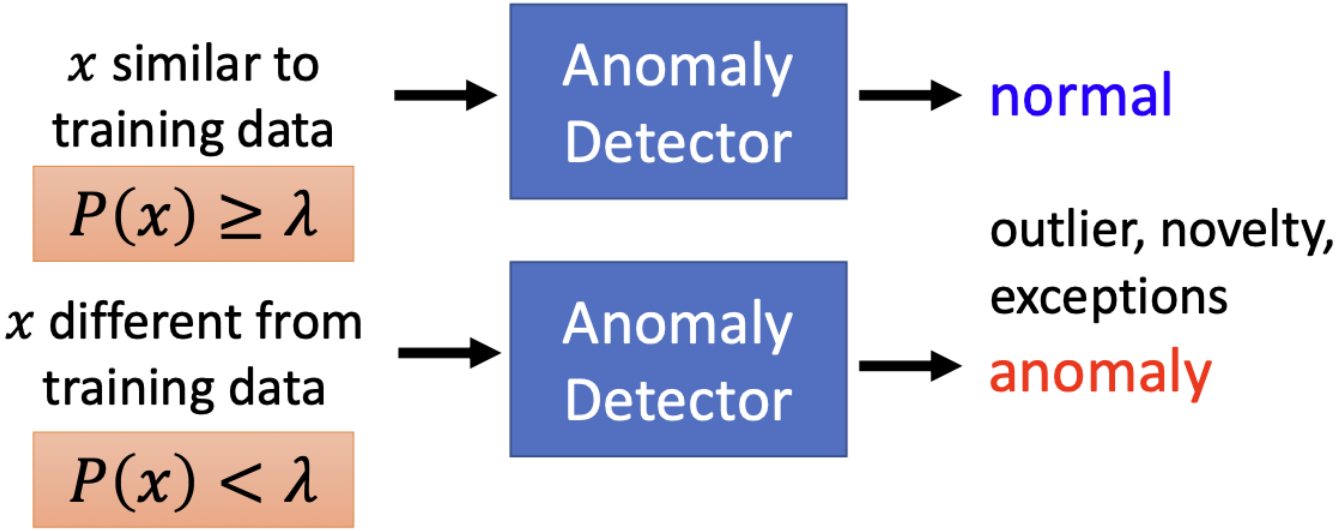
Cost Table B

Statistical Methods

- No labels available
- Assumption: normal objects are “clustered”
  - Normal objects follow a pattern far more frequently than anomalies
- General steps
  - Build a profile of normal behavior
    - summary statistics for overall population
    - find ”clusters”
  - Use the normal profile to detect anomalies
    - anomalies are observations whose characteristics differ significantly from the normal profile

Problem definition

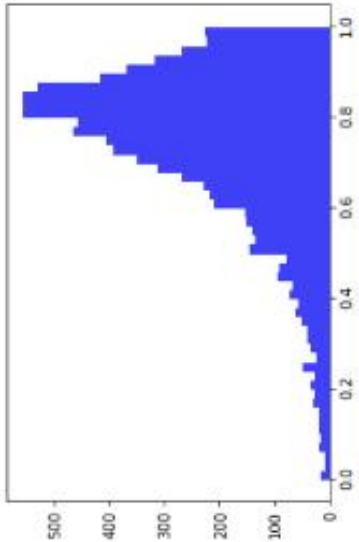
- Given training data $\{x^1, x^2, \ldots, x^N\}$, no labels available
- Find a model detecting whether input x is similar to training data

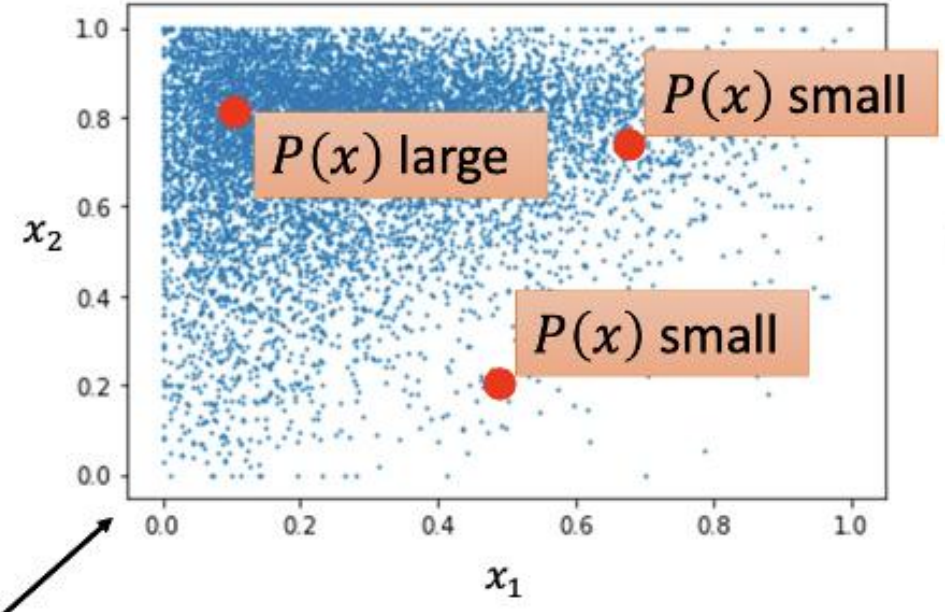


Example features

- Monitoring computers in a data center
  - x^i = features of machine i, such as:
    - x_1 = memory use, x_2 = network traffic, x_3 = number of disk access/sec …
- Fraud detection
  - x^i = features of user i's activities, e.g. purchase time, payment type …
- Industrial damage detection
  - x^i = features of aircraft i's engine, e.g. heat generated, vibration intensity…
- **Goal:**
  - model $P(x)$ from training data
  - Identify anomalies by checking whether $P(x) < \lambda$.

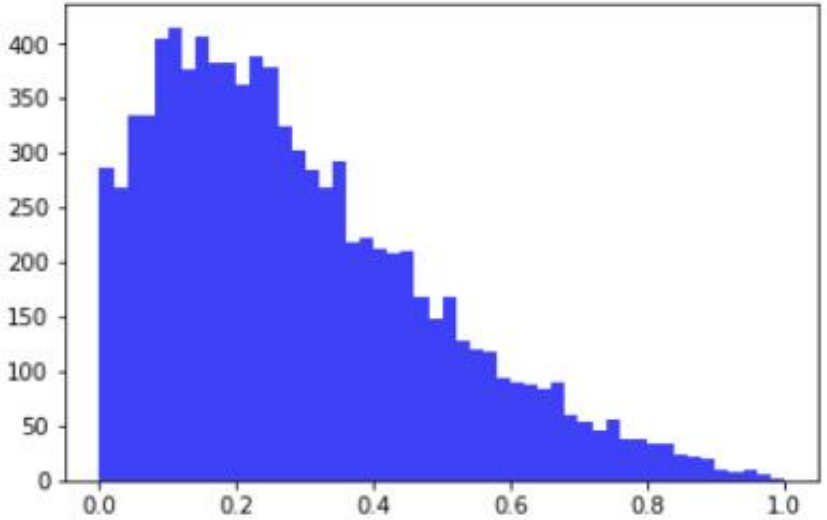
Illustration





$$\{x^1, x^2, \cdots, x^N\}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

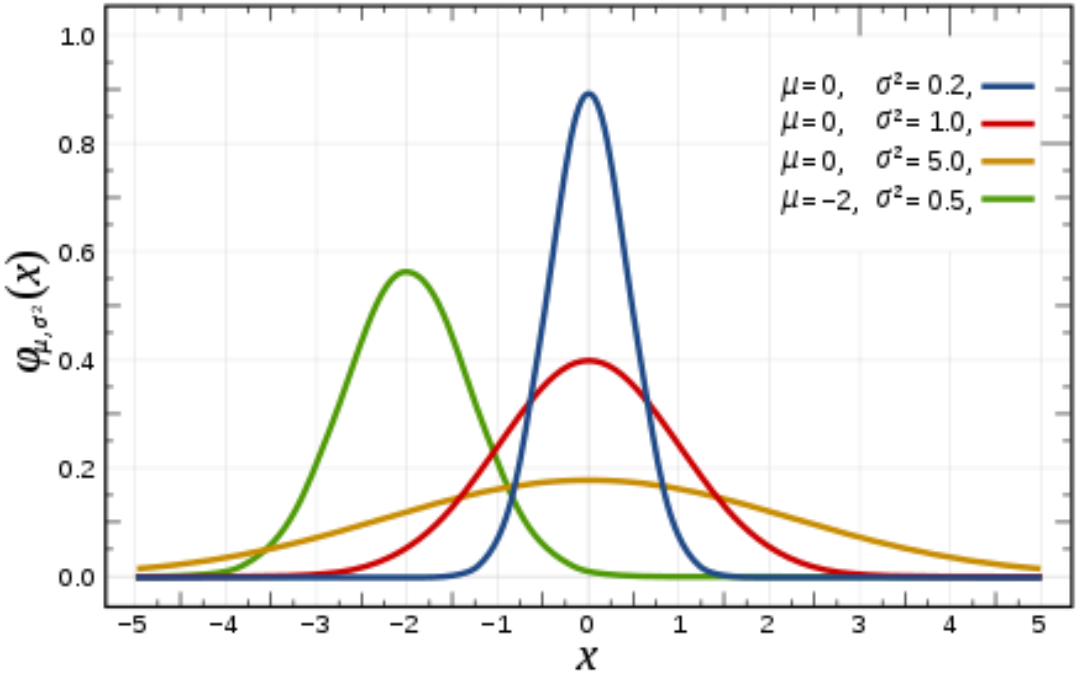


Gaussian distribution

- Gaussian (normal) distribution is a type of continuous probability distribution.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- Given $x \in \mathbb{R}$, $x \sim N(\mu, \sigma^2)$: x is distributed Gaussian with mean μ, variance σ^2.



Parameter estimation

- Given data $\{x^1, x^2, \dots, x^N\}$, $x^i \in \mathbb{R}$
- Assume $x^i \sim N(\mu, \sigma^2)$, how to learn the approximate values of parameters μ and σ?
  - Use the maximum likelihood method to estimate μ and σ:
  - set the mean of the Gaussian to be the mean of the data,
  - set the standard deviation of the Gaussian to be the standard deviation of the data.

$$\hat{\mu} = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x^i$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x^i - \bar{x}\right)^2$$

Anomaly detection with Gaussian distribution

- Given training data $\{x^1, x^2, \dots, x^N\}$, $x^i \in \mathbb{R}^n$, n = # features
- Fit parameters $\mu_1, \mu_2, \dots, \mu_n;\ \sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$
  - $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_j^i$ *j-th* feature of data x^i
  - $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_j^i - \mu_j)^2$
- Given new data $x \in \mathbb{R}^n$, compute $P(x)$:

$$P(x) = \prod_{j=1}^{n} p\left(x_j \middle| \mu_j, \sigma_j^2\right) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{\left(x_j - \mu_j\right)^2}{2\sigma_j^2}\right)$$

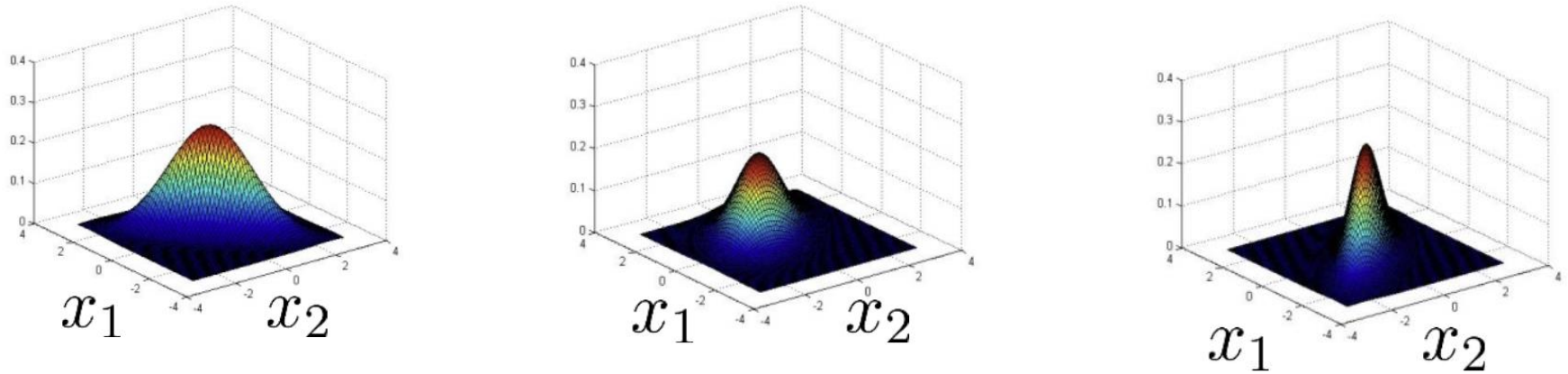
$$x = \begin{cases} normal, & if\ P(x) \geq \lambda \\ anomaly, & if P(x) < \lambda \end{cases}$$

Multivariate Gaussian distribution

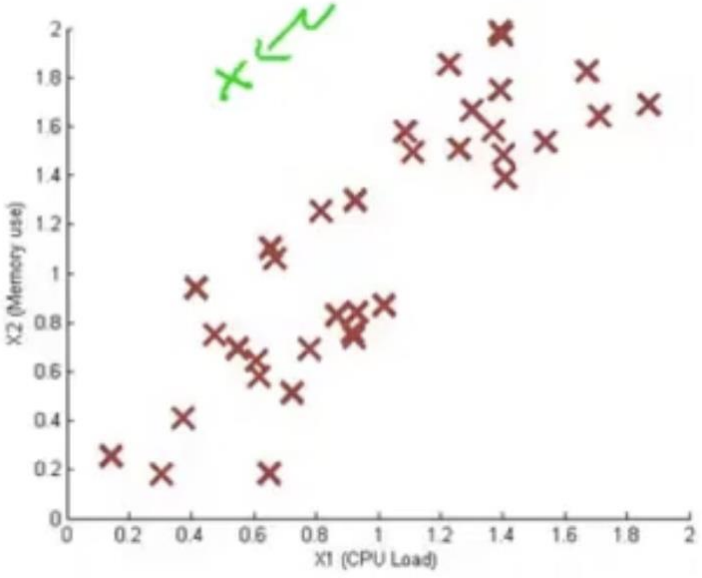
- Given data $\{x^1, x^2, \dots, x^N\}$, $x^i \in \mathbb{R}^n$, n = # features
- Model $P(x)$ all in one go. (not model p(x) separately)

$$P(x) = \frac{1}{(2\pi)^{n/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$

- Input: vector x, output: P(x) (scaler)
- Parameters: $\mu \in \mathbb{R}^n$ (mean), $\Sigma \in \mathbb{R}^{n\times n}$(covariance matrix)



Anomaly detection with multivariate Gaussian distribution



1. Fit model $P(x)$ by computing
   - $\mu = \frac{1}{N}\sum_{i=1}^{N} x^i$
   - $\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^i - \mu)(x^i - \mu)^T$
2. Given new data x, compute

$$P(x) = \frac{1}{(2\pi)^{n/2}}\frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

$$x = \begin{cases} normal, & if\ P(x) \geq \lambda \\ anomaly, & if P(x) < \lambda \end{cases}$$

n: dimension of x (# of features)
N: # of training examples

$|M|$: The determinant of a square matrix M, which is a number that can be calculated from M

Relationship to the original model

- Original model $P(x) = \prod_{j=1}^{n} p(x_j|\mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2})$
  - Manually create features to capture anomalies where x_1 and x_2 take unusual combinations of values. e.g. $x_3 = \frac{x_1}{x_2} = \frac{CPU\ load}{memory}$
  - Computationally cheaper, scales better to large # features
  - OK even if # training examples is small

- Multivariate Gaussian model $P(x) = \frac{1}{(2\pi)^{n/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right\}$
  - Automatically captures correlations between features
  - Computationally expensive
  - Must have N > n or else Σ is non-invertible.

Statistical Anomaly Detection

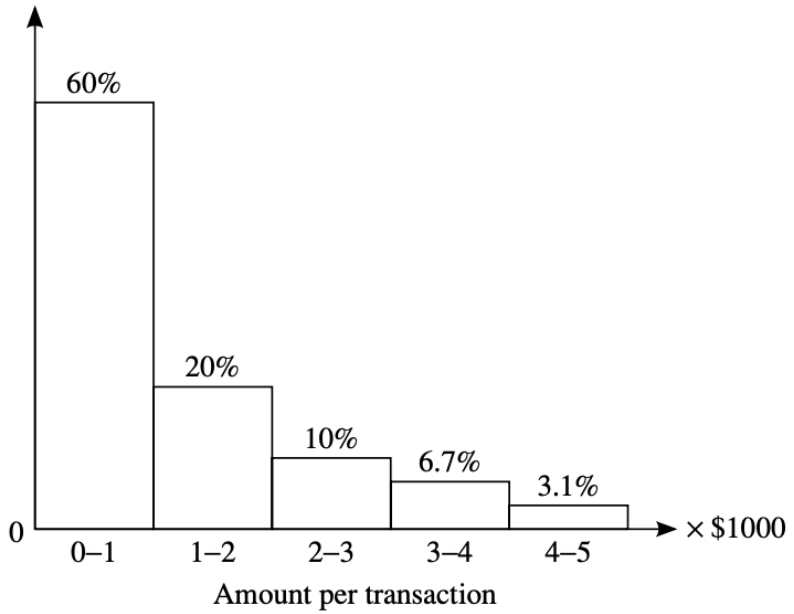
- Anomalies are objects that are fit poorly by a statistical model
  - Statistical methods assume normal data objects are generated by a statistical model, and data not following the model are anomalies

- Learn a statistical model describing the distribution of the data
  - Parametric: assumes that the normal data objects are generated by a parametric distribution with parameters, e.g. Gaussian distribution
  - **Nonparametric: tries to determine the model from the input data, e.g. histogram**

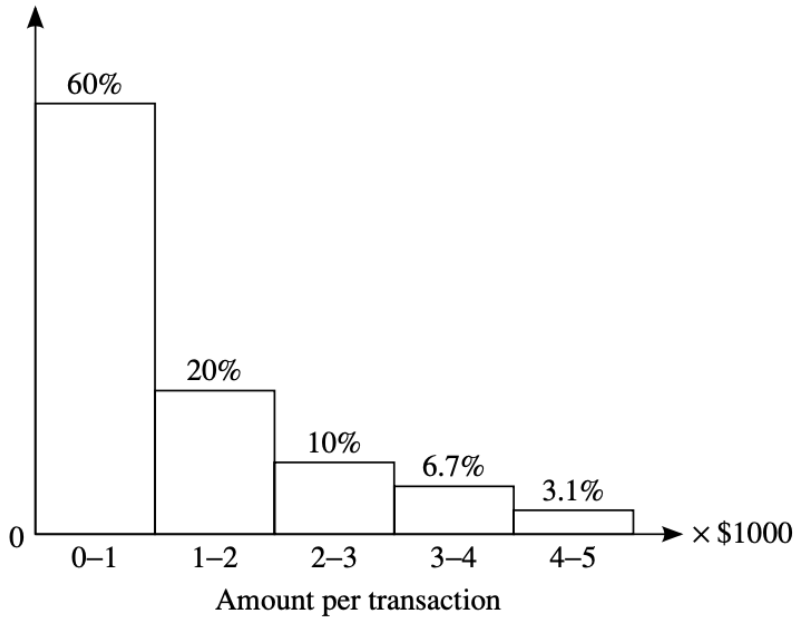
Nonparametric Methods

- The model of “normal data” is learned from the input data, rather than assuming one a priori
- Nonparametric methods often make fewer assumptions about the data, and thus can be applicable in more scenarios
- Anomaly detection using a histogram
  - A store records the purchase amount for every customer transaction.
  - e.g. 60% of the transaction amounts are between $0.00 and $1000

Histogram of purchase amounts in transactions.



Nonparametric Methods



- Idea: use the histogram as a nonparametric statistical model to capture anomalies
  - A transaction in the amount of $7500 can be regarded as an anomaly
    - 1 – (60% + 20% + 10% + 6.7% + 3.1%) = 0.2% of transactions have an amount higher than $5000
  - A transaction amount of $385 can be treated as normal
    - It falls into the bin (or bucket) holding 60% of the transactions
- Steps
  - Histogram construction: construct a histogram using input data
  - Anomaly detection: if the object falls in one of the histogram's bins, the object is regarded as normal. Otherwise, it is considered an outlier

References

- *Dorothy Denning, "An Intrusion-Detection Model," IEEE Transactions on Sotware Engineering SE-13:2 (1987): 222–232.*
- *Data mining concepts and techniques, Jiawei Han, Micheline Kamber, Jian Pei*
- *Machine learning, Andrew Ng*
- *"Anomaly Detection: A Tutorial" Arindam Banerjee, Varun Chandola, Vipin Kumar, Jaideep Srivastava, University of Minnesota Aleksandar Lazarevic, United Technology Research Center*
- *Machine Learning and Security: Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *Softmax function: Thomas Wood. deepai.org*

Support Vector Machines and Decision Tree

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

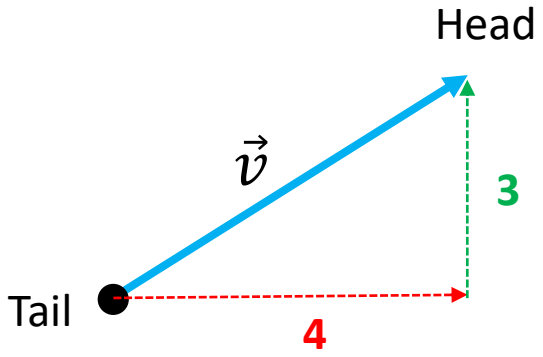
Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- Preliminaries
  - vector magnitude
  - vector normalization
  - vector inner product – geometric point of view
- Introduction to Linear SVM
  - Our focus: Geometric intuition
- Malware detection
  - Common types of Malware
  - Introduction to malware analysis methodology
  - Decision tree
    - Information Gain
    - Build a Decision Tree

Vectors

- A vector can be visually represented as an arrow



$$\vec{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

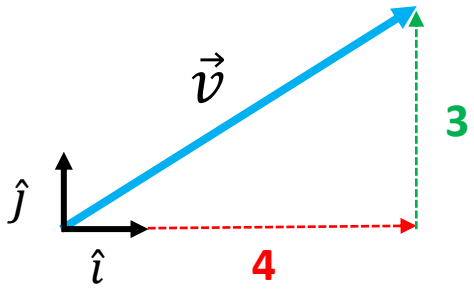
Or: $\vec{v} = (4, 3)$

- A vector can also be represented as an ordered list or a tuple by starting from its tail and asking how far away is its head in the:
  - Horizontal direction
  - Vertical direction

Unit Vectors

- Any vector can also be represented as a *sum* of scaled up versions of specific ***unit*** vectors

$$\vec{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



$$\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

It goes in the horizontal direction *only* and has length 1

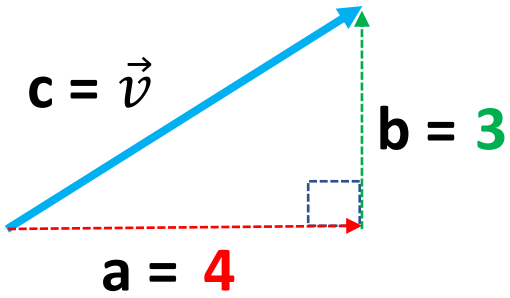
$$\hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

It goes in the vertical direction *only* and has length 1

$$\vec{v} = 4 \times \hat{\imath} + 3 \times \hat{\jmath} = 4\begin{bmatrix} 1 \\ 0 \end{bmatrix} + 3\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Vector Magnitude

- How can we calculate the length (or *magnitude*) of a vector?



Pythagorean theorem:

$$\boldsymbol{a^2 + b^2 = c^2} \rightarrow \mathbf{c} = \sqrt{\boldsymbol{a^2 + b^2}}$$

Magnitude of the vector = $\|\vec{v}\|$ = $\sqrt{3^2 + 4^2}$ = $\sqrt{25} = 5$

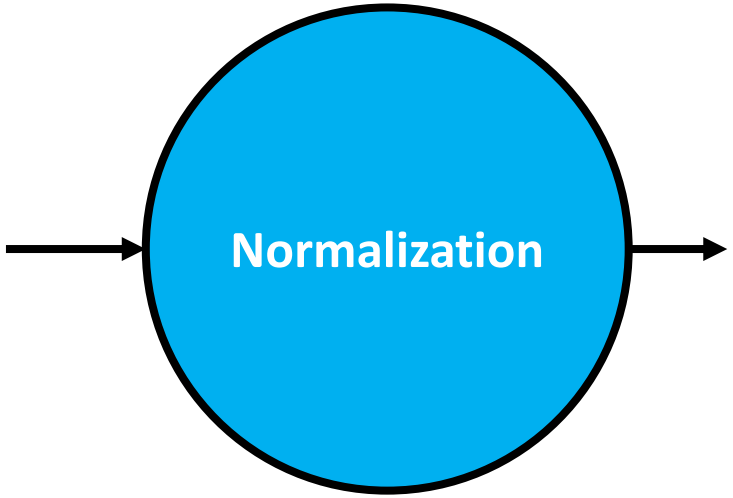
How can we keep vector, $\vec{v}$, pointing to the same direction, but change its magnitude to 1 (i.e., turn it into a unit vector)?

Vector Normalization

- How can we construct a unit vector out of a given vector of any length?

Input:

Vector of any length



Output:

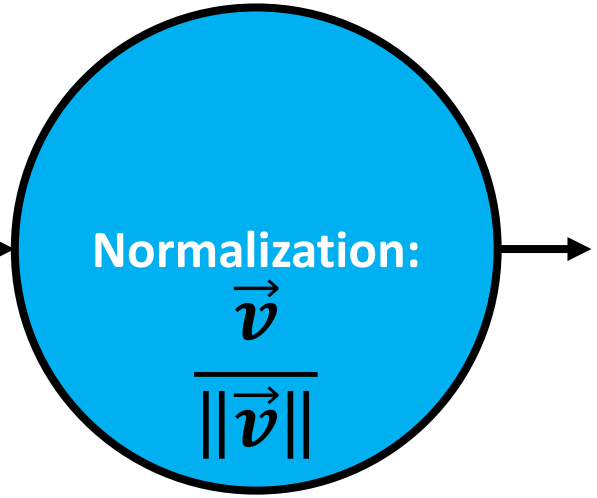
The vector with the same direction, but with length 1

Vector Normalization

- How can we construct a unit vector out of a given vector of any length?

Input:

$\vec{v}$



Output:

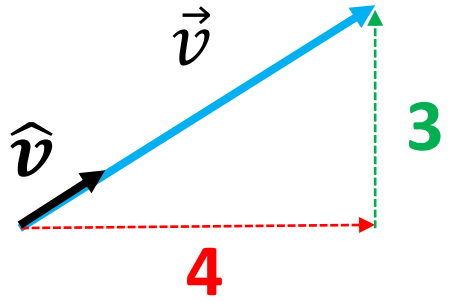
$\hat{v}$

$$\hat{v} = \frac{\vec{v}}{\|\vec{v}\|}$$

Vector Normalization

- How can we construct a unit vector out of a given vector of any length?

$$\vec{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



$$||\vec{v}|| = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$$

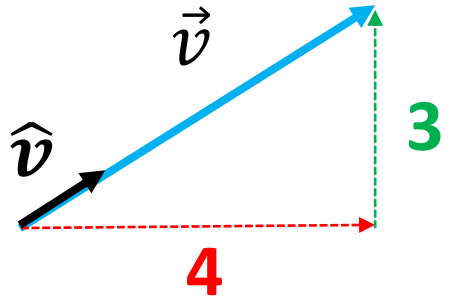
$$\hat{v} = \frac{\vec{v}}{||\vec{v}||} = \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix}$$

Let us verify that its length is 1

Vector Normalization

- How can we construct a unit vector out of a given vector of any length?

$$\vec{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



$$||\vec{v}|| = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$$

$$\hat{v} = \frac{\vec{v}}{||\vec{v}||} = \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix}$$

$$||\hat{v}|| = \sqrt{(\tfrac{3}{5})^2 + (\tfrac{4}{5})^2}$$

$$= \sqrt{(9 + 16)/25} = 1$$

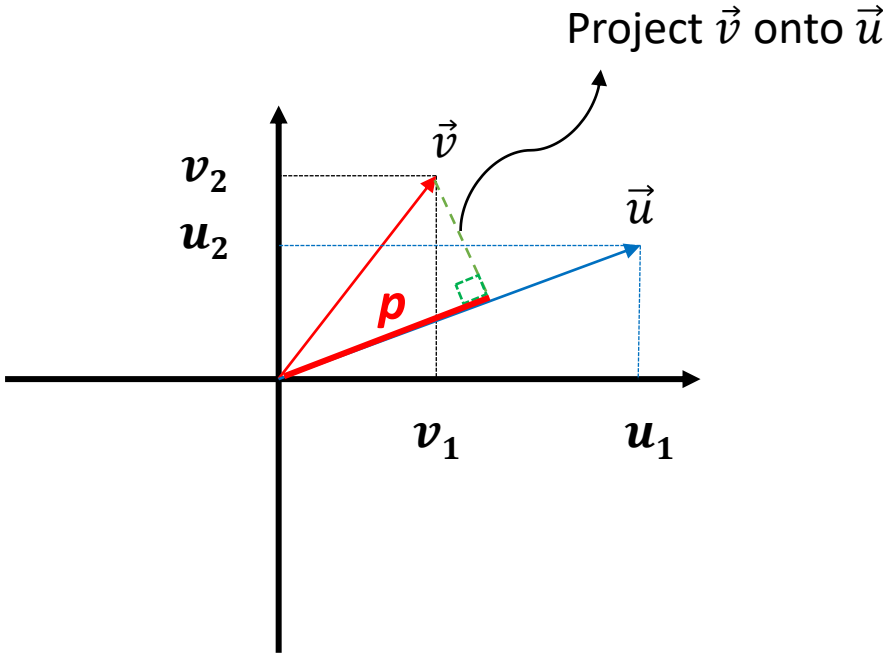
Vector Inner Product

- Assume the following two vectors:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

- What is the inner product of $\vec{u}$ and $\vec{v}$?

$$\vec{u}^T \vec{v} = ?$$



p is the length of the projection of $\vec{v}$ onto $\vec{u}$

Vector Inner Product

- Assume the following two vectors:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

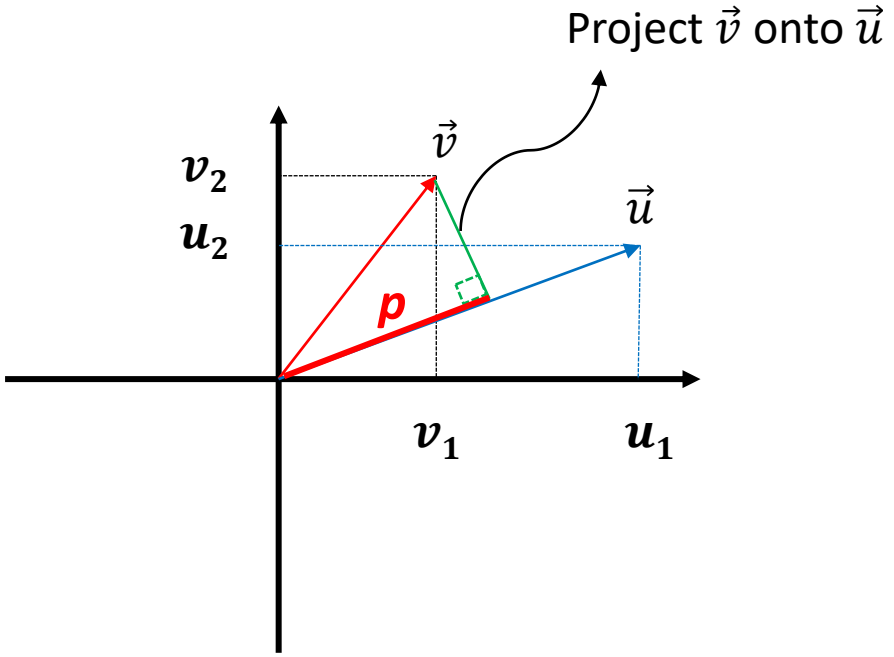
- What is the inner product of $\vec{u}$ and $\vec{v}$?

$$\vec{u}^T \vec{v} = p \times \|\vec{u}\|$$

$$\equiv$$

$$\vec{u}^T \vec{v} = [u_1\, u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$= u_1 \times v_1 + u_2 \times v_2$$



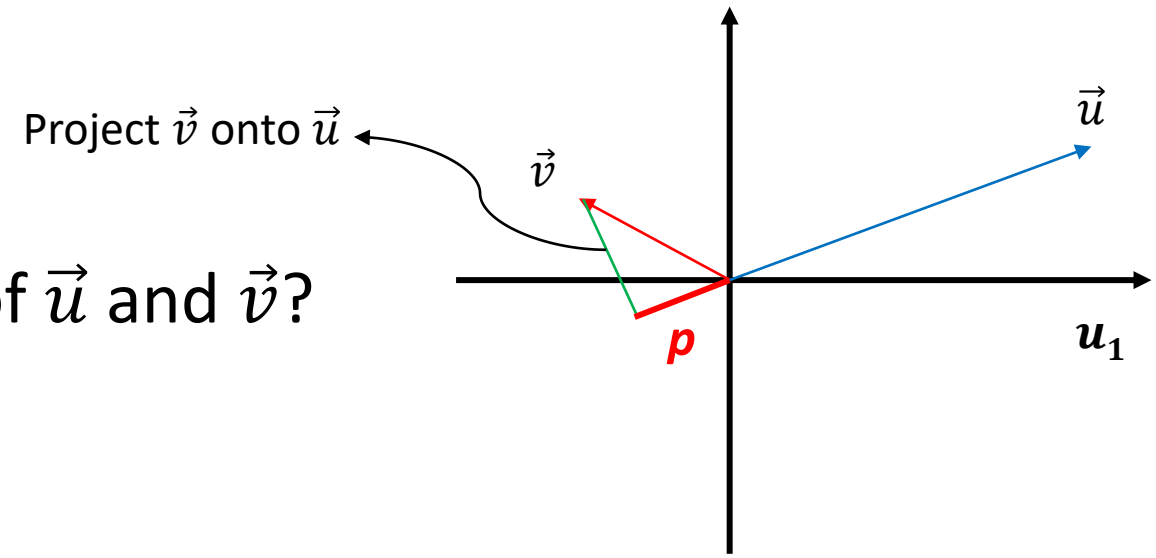
p is the length of the projection of $\vec{v}$ onto $\vec{u}$

Can be signed

Vector Inner Product

- Assume the following two vectors:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$



- What is the inner product of $\vec{u}$ and $\vec{v}$?

$$\vec{u}^T \vec{v} = p \times \|\vec{u}\|$$

$$\equiv$$

$$\vec{u}^T \vec{v} = [u_1\, u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$= u_1 \times v_1 + u_2 \times v_2$$

p is negative here

Recall: Perceptron

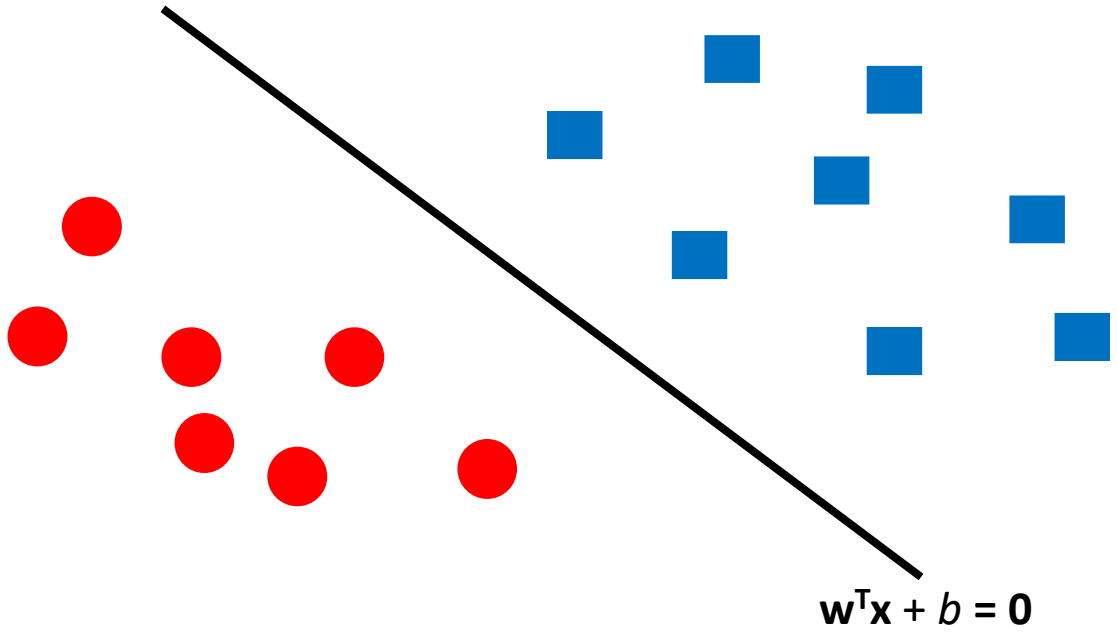
- Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$
  - $\{x^1, x^2, \ldots, x^n\}$: data set, and $y^i \in$ {1,-1} be the class label
- Perceptron
  - a binary classifier: maps its input to an output value $f(\mathbf{x})$:

$$f(\boldsymbol{x}) = \begin{cases} +1, & if\ \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b \ \geq 0 \\ -1, & if\ \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b \ < 0 \end{cases}$$

  - parameters: $\boldsymbol{w}$ is a vector of weights, b is the bias
  - linear classifier: a classification algorithm that makes its predictions based on a linear function combining a set of weights with the feature vector.
  - find a hyperplane $\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = 0$ that minimizes the number of misclassified points by varying $\boldsymbol{w}$ and b

Perceptron: Intuition

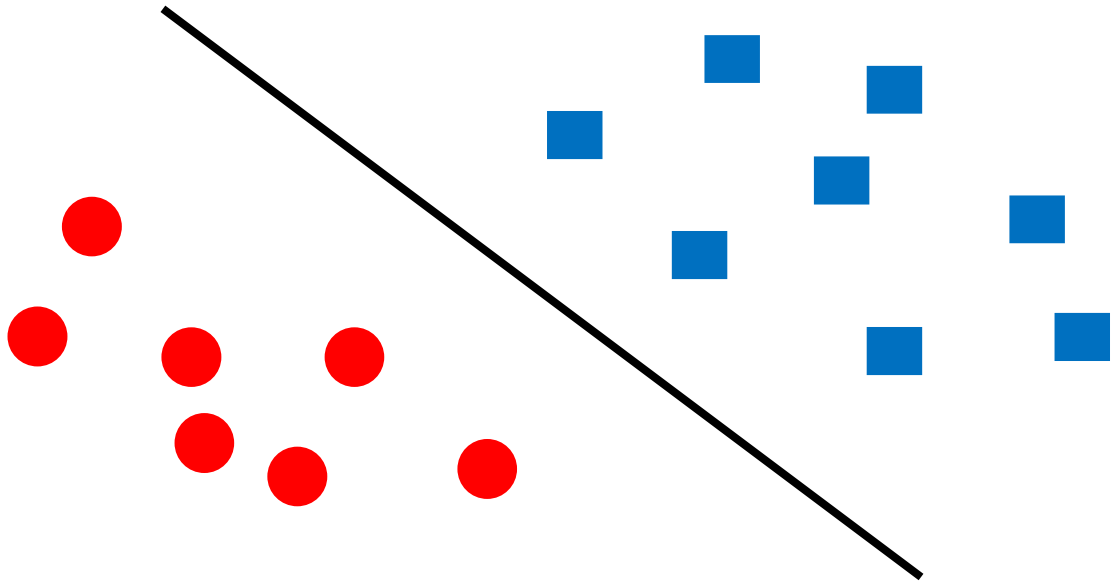
- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes



A perceptron divides a space by a hyperplane into two half-spaces

Perceptron: Intuition

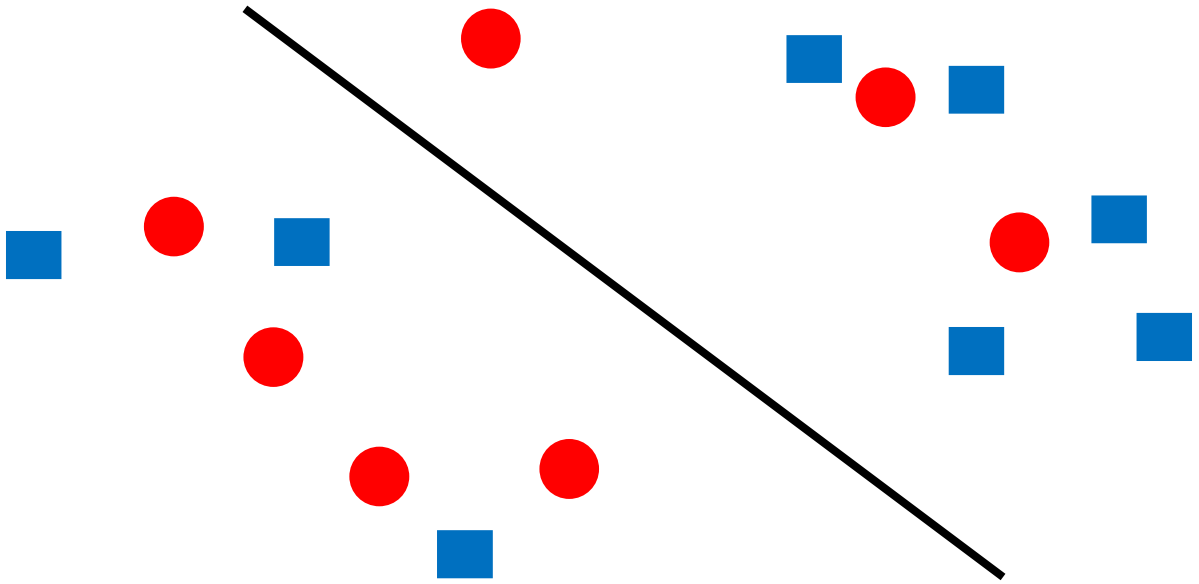
- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes



This entails that the space has to be *linearly separable* (or otherwise, will not be able to correctly classify all examples)

Perceptron: Intuition

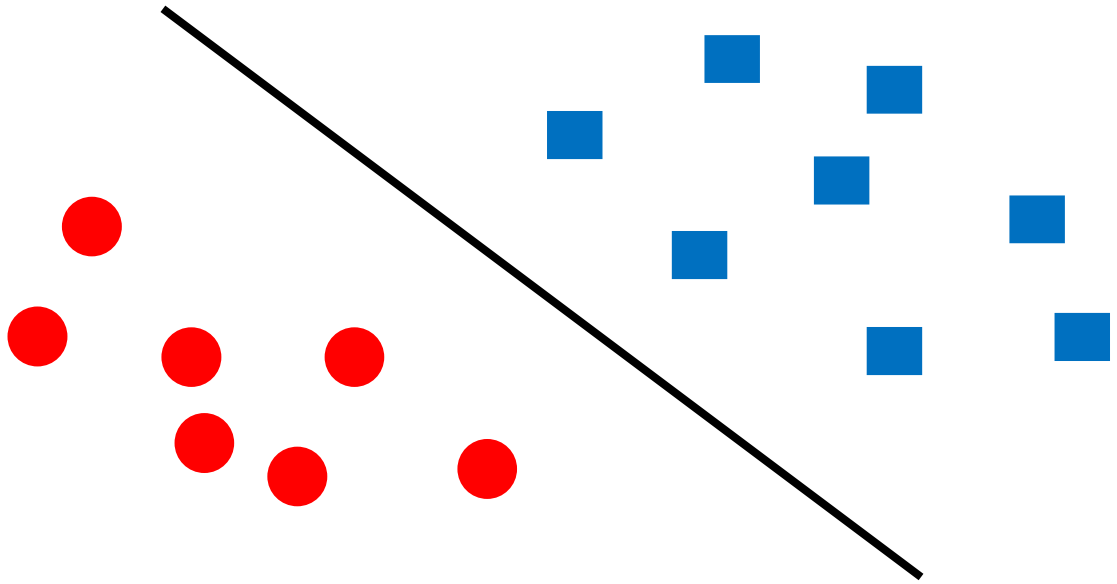
- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes



NOT a linearly separable space, hence, perceptron will not be effective!

Perceptron: Intuition

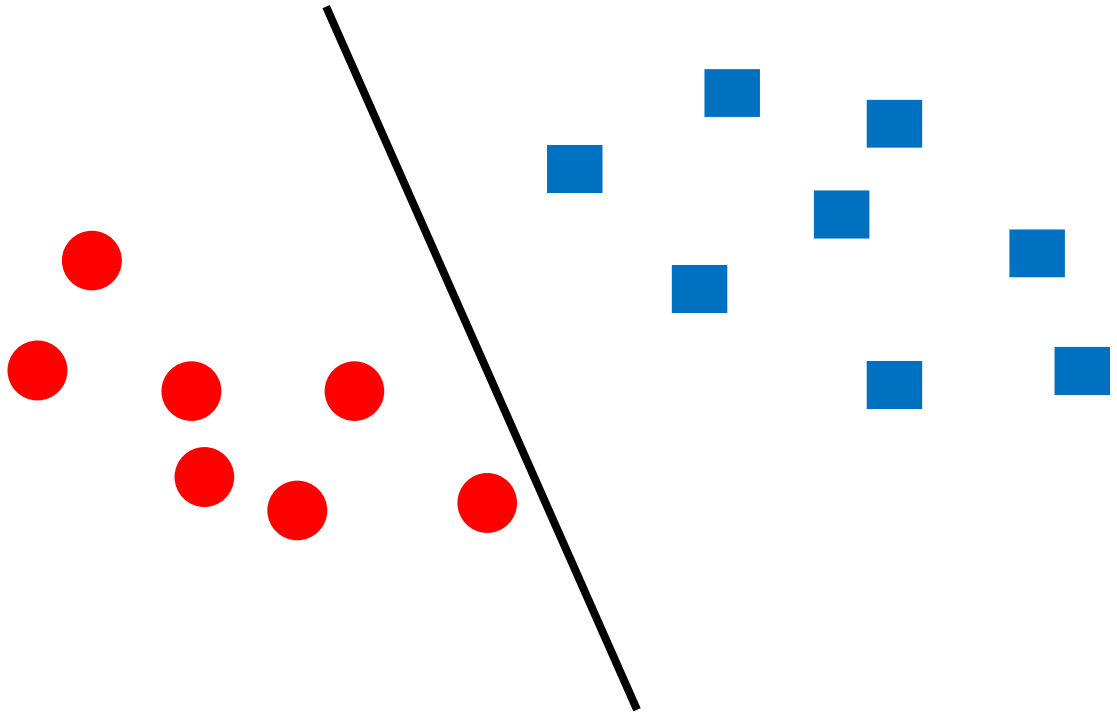
- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes



A *linearly separable* space and a workable perceptron

Perceptron: Intuition

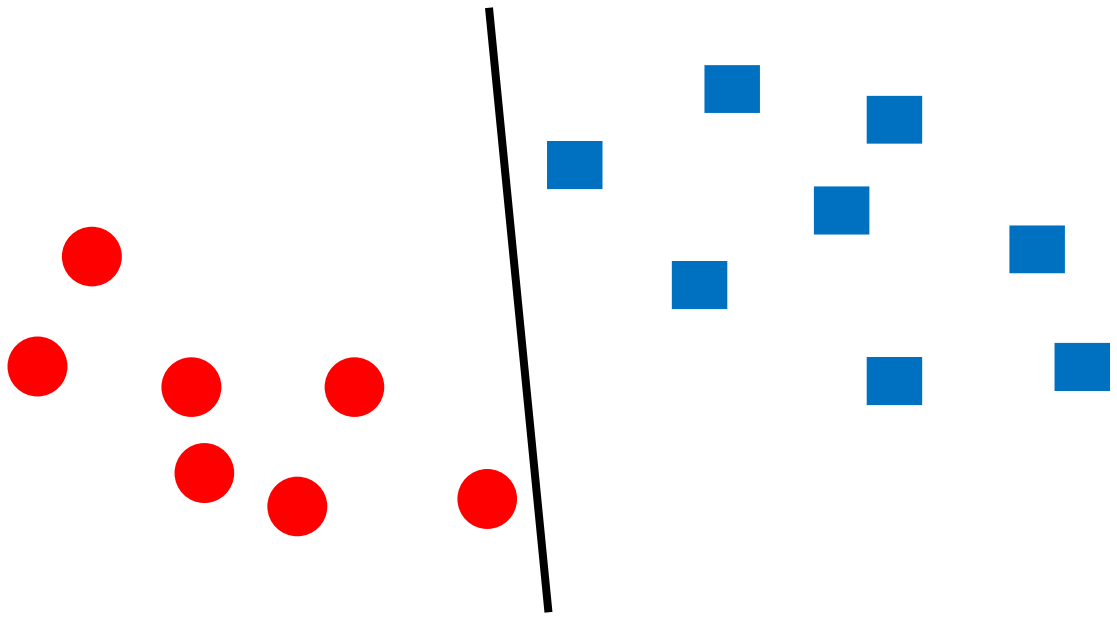
- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes



A *linearly separable* space and another workable perceptron!

Perceptron: Intuition

- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes

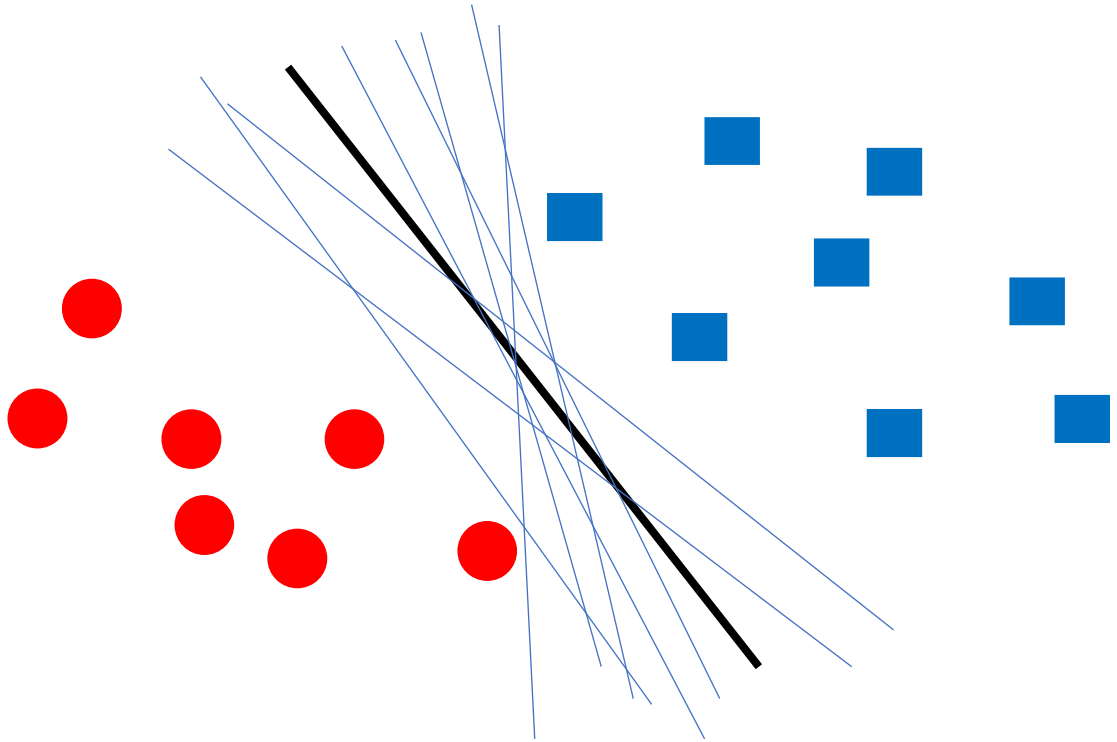


Yet, another valid hyperplane that can be learnt by a perceptron!

If there are many hyperplanes, the perceptron will converge to one of them & classify correctly all examples

Perceptron: Intuition

- The way a perceptron works is by learning a hyperplane that clearly separates examples into two classes

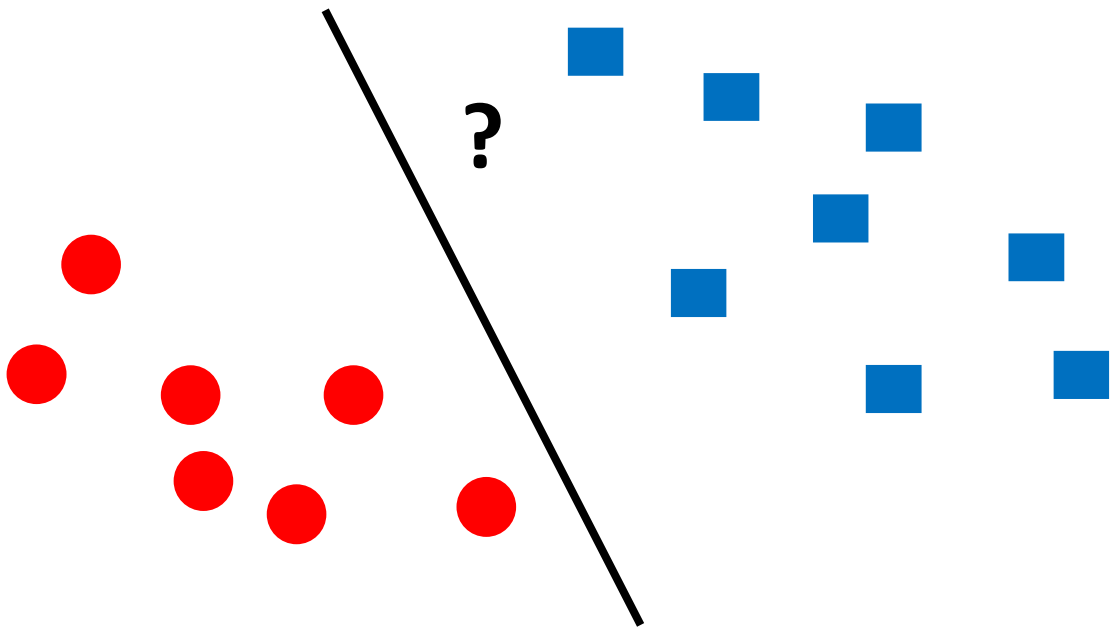


Any of these would be fine..

..but which is best?

Limitations of Perceptron

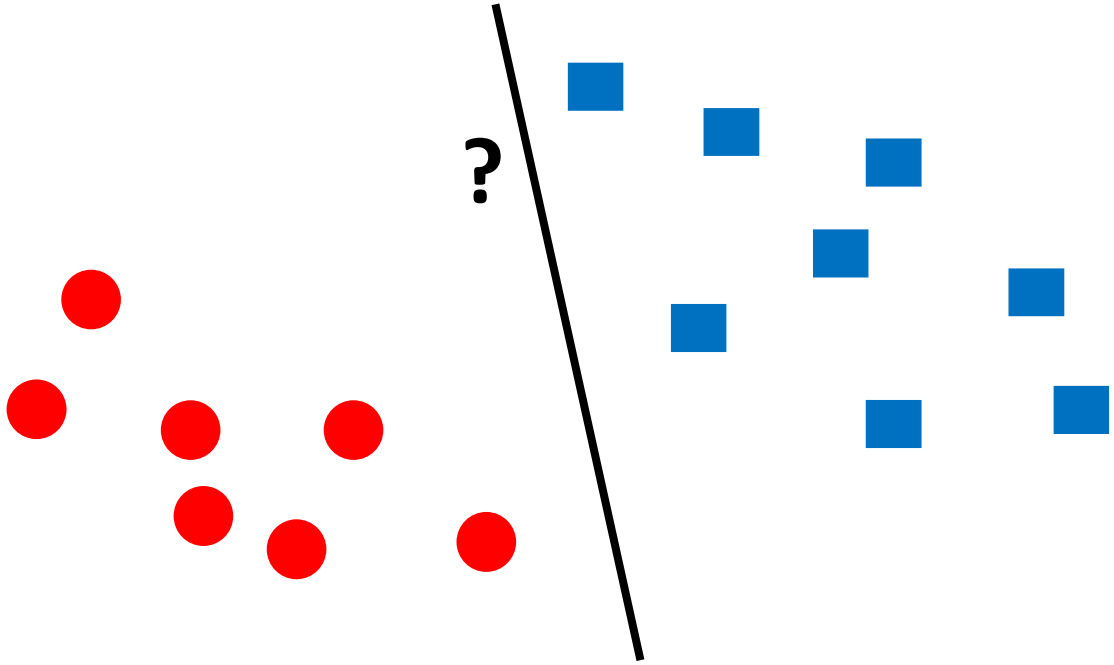
- Perceptrons exhibit various limitations in their ability to classify data
  - There could be many hyperplanes, which are not all equally good



An acceptable hyperplane and the new example indicated by "?" will be classified as a square

Limitations of Perceptron

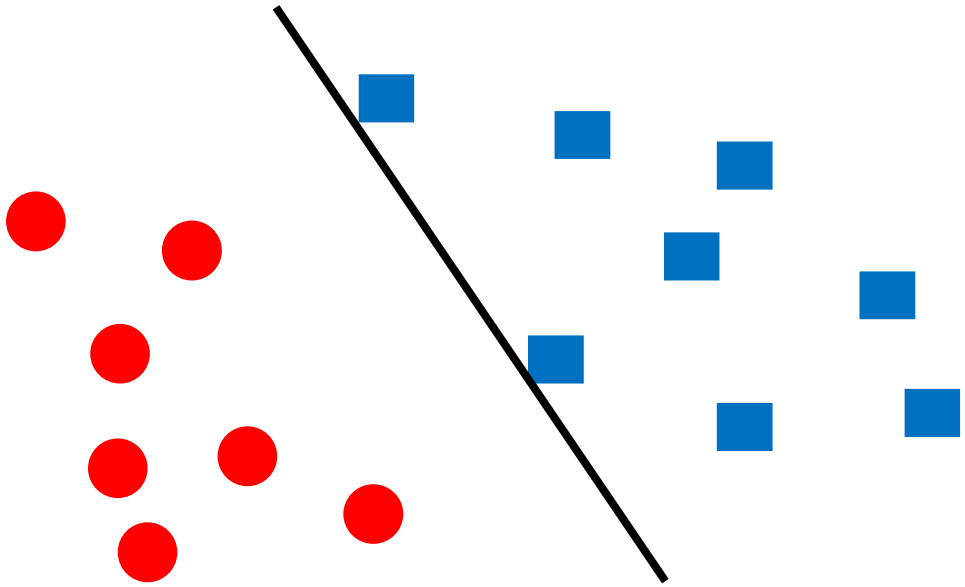
- Perceptrons exhibit various limitations in their ability to classify data
  - There could be many hyperplanes, which are not all equally good



Another acceptable hyperplane, but the new example will now be classified as a circle (*although it seems closer to squares!*)

Limitations of Perceptron

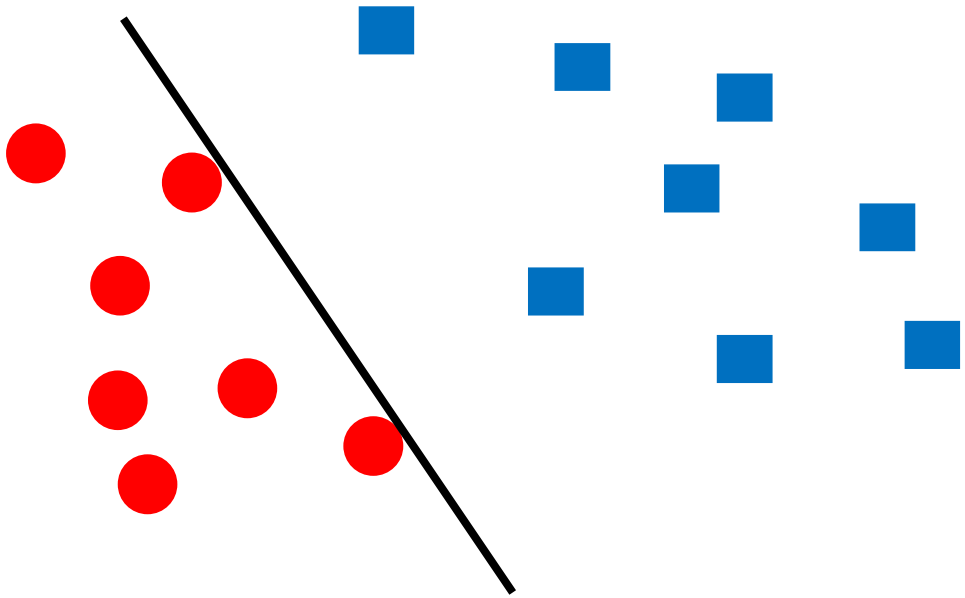
- Perceptrons exhibit various limitations in their ability to classify data
  - another problem is that perceptrons usually stop as soon as there are no misclassified examples



This hyperplane *just* managed to accommodate the two squares it touches before stopped

Limitations of Perceptron

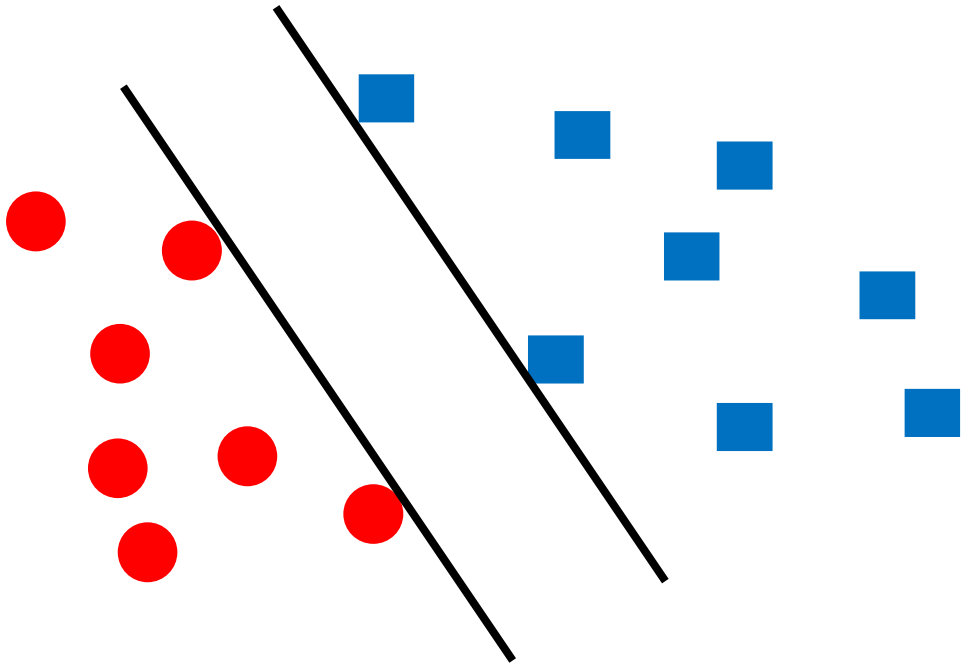
- Perceptrons exhibit various limitations in their ability to classify data
  - another problem is that perceptrons usually stop as soon as there are no misclassified examples



This hyperplane also *just* managed to accommodate the two circles it touches before stopped

Limitations of Perceptron

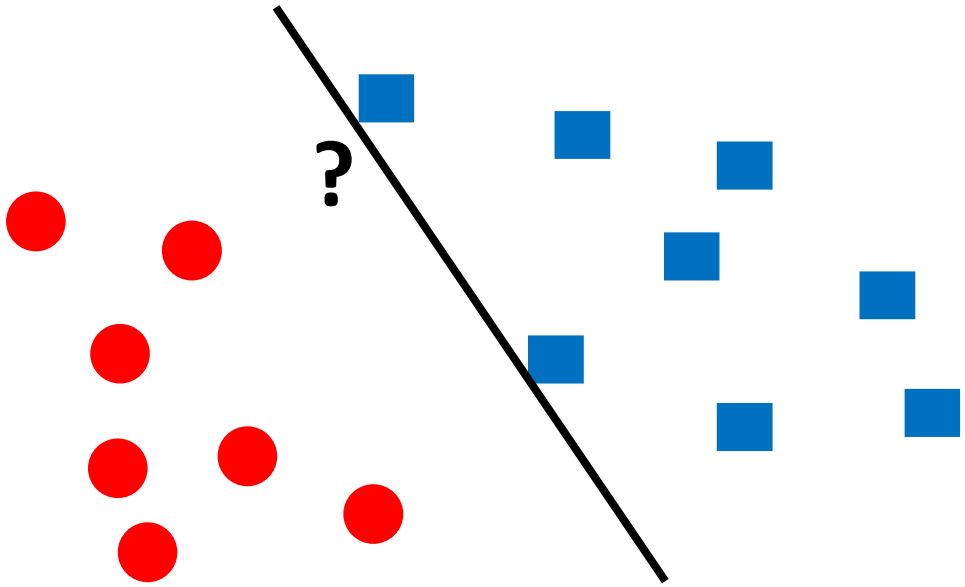
- Perceptrons exhibit various limitations in their ability to classify data
  - another problem is that perceptrons usually stop as soon as there are no misclassified examples



If either of these hyperplanes represents the final weight vector, the weights will be biased toward one of the classes

Limitations of Perceptron

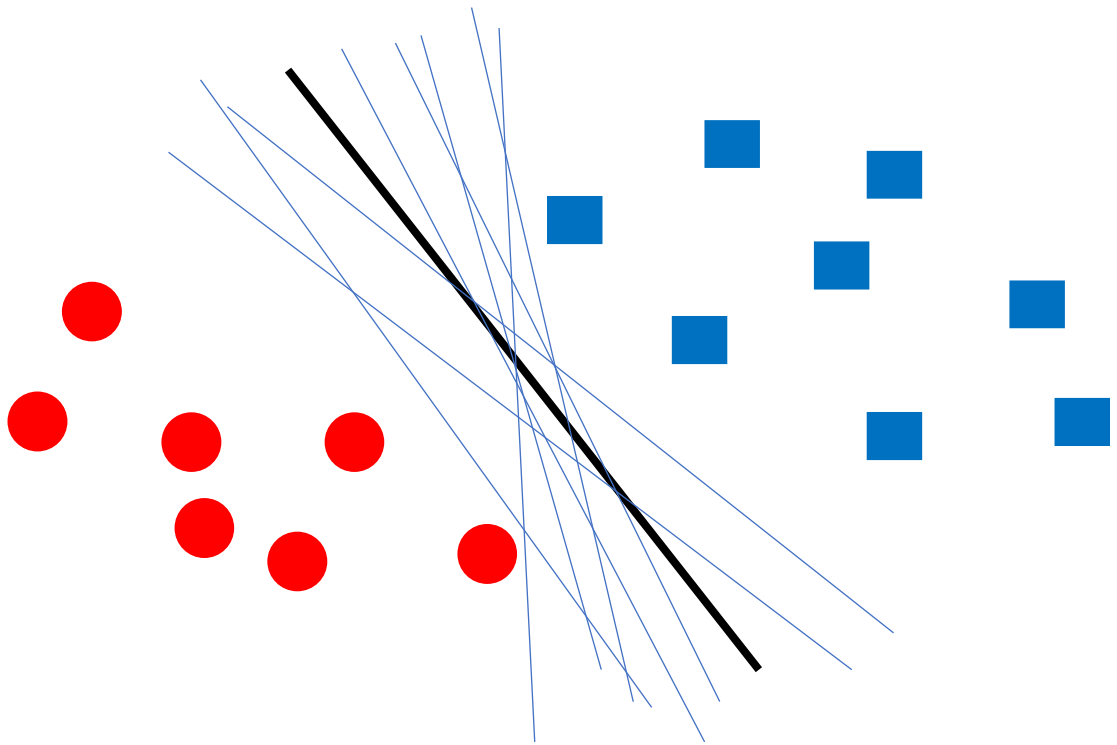
- Perceptrons exhibit various limitations in their ability to classify data
  - another problem is that perceptrons usually stop as soon as there are no misclassified examples



For example, if this hyperplane is the one that the perceptron chooses, the example indicated by "?" will be classified as a circle

Limitations of Perceptron

- Perceptrons exhibit various limitations in their ability to classify data

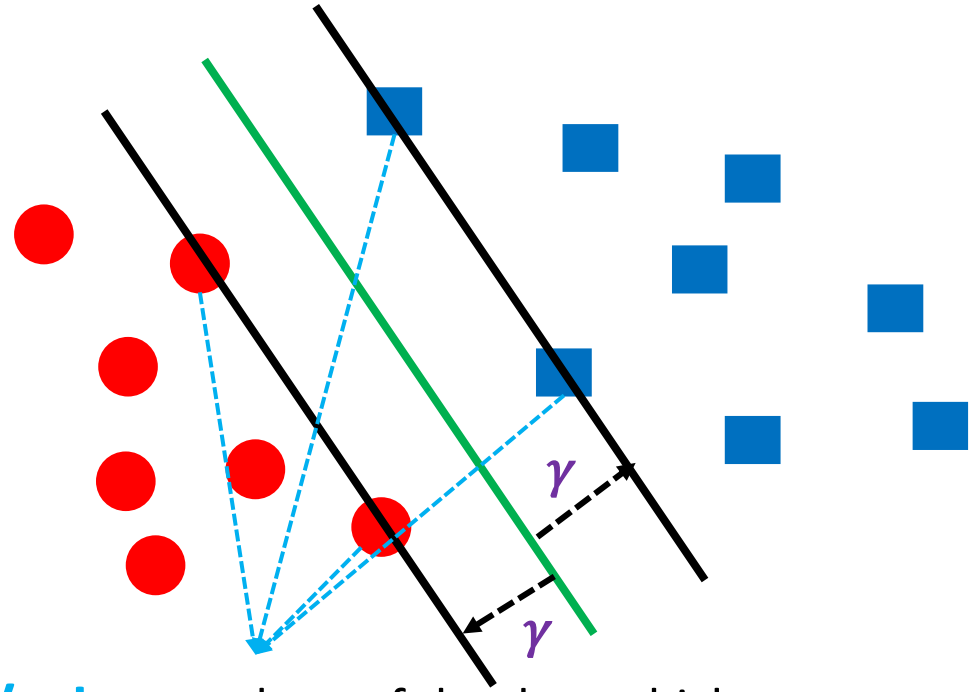


Any of these would be fine..

..but which is best?

Support Vector Machines

- Linear Support Vector Machines (Linear SVM)
  - Maximum margin linear classifier

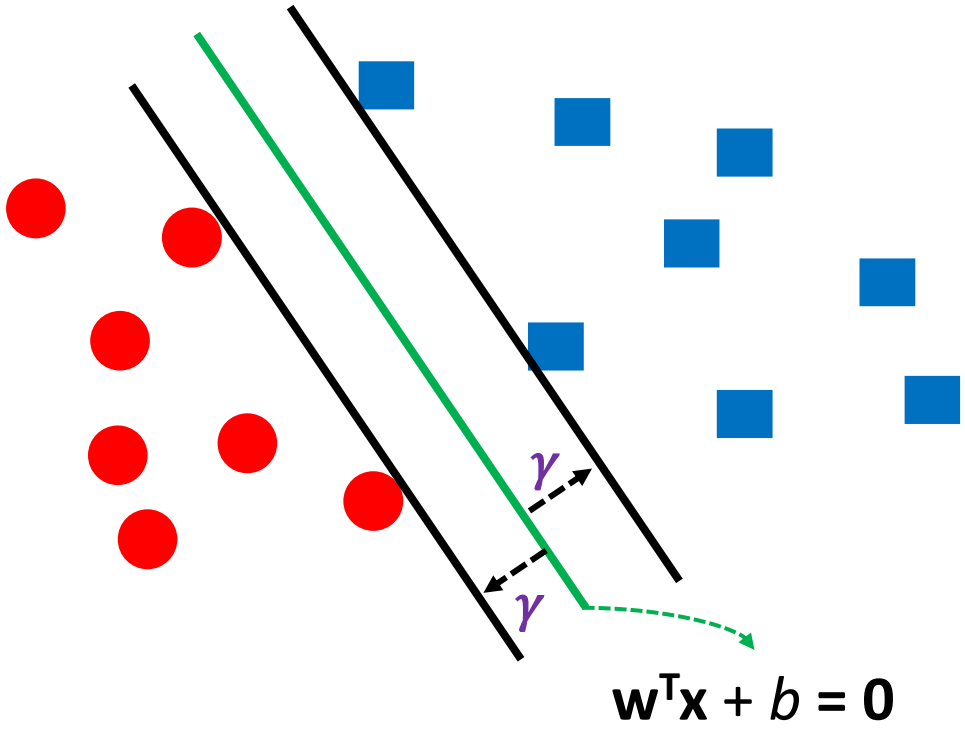


SVM selects one particular hyperplane (i.e. decision boundary – the green line in the figure) that not only separates the examples into two classes, but does so in a way that maximizes the margin: which is measured by the distance between the hyperplane and the closest examples of the training set

Support Vectors: subset of the data which defines the position of the separator

The Objective of SVM

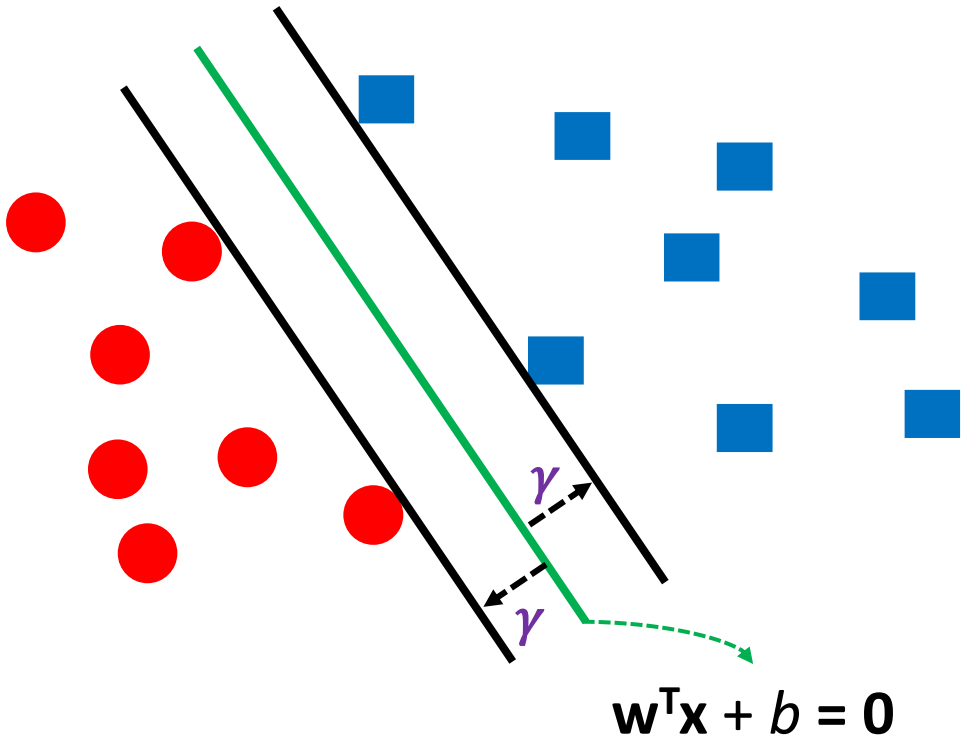
- The objective of an SVM is to select a hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ that maximizes the distance, γ, between the hyperplane and examples in the training set



Intuitively, we are more certain of the class of examples that are far from the separating hyperplane than we are of examples near to that hyperplane

The Objective of SVM

- The objective of an SVM is to select a hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ that maximizes the distance, γ, between the hyperplane and examples in the training set



Thus, it is desirable that all the training examples be as far from the hyperplane as possible (*but on the correct side of that hyperplane, of course!*)

Finding the Decision Boundary

- Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$
  - $\{x^1, x^2, \ldots, x^n\}$: our data set
  - $y^i \in \{1,-1\}$: the class label
- Classifier: $f(\boldsymbol{x}^i) = sign(\boldsymbol{w}^T\boldsymbol{x}^i + b)$
  - ***w:*** *weight vector*
  - *b*: bias
- The SVM decision boundary aims to
  - classify all points correctly
  - maximize the margin (by varying weight vector ***w*** and bias *b*)
- The decision boundary should be as far away from the data of both classes as possible

Computing the margin width

- Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow

$$\mathbf{w}^T\mathbf{x}^i + b \geq 1 \text{ if } y^i = +1$$

$$\mathbf{w}^T\mathbf{x}^i + b \leq -1 \text{ if } y^i = -1$$

- For support vectors, the inequality becomes an equality
- The margin is: $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$
- Maximizing the margin is equivalent to minimizing $\| \boldsymbol{w} \|$

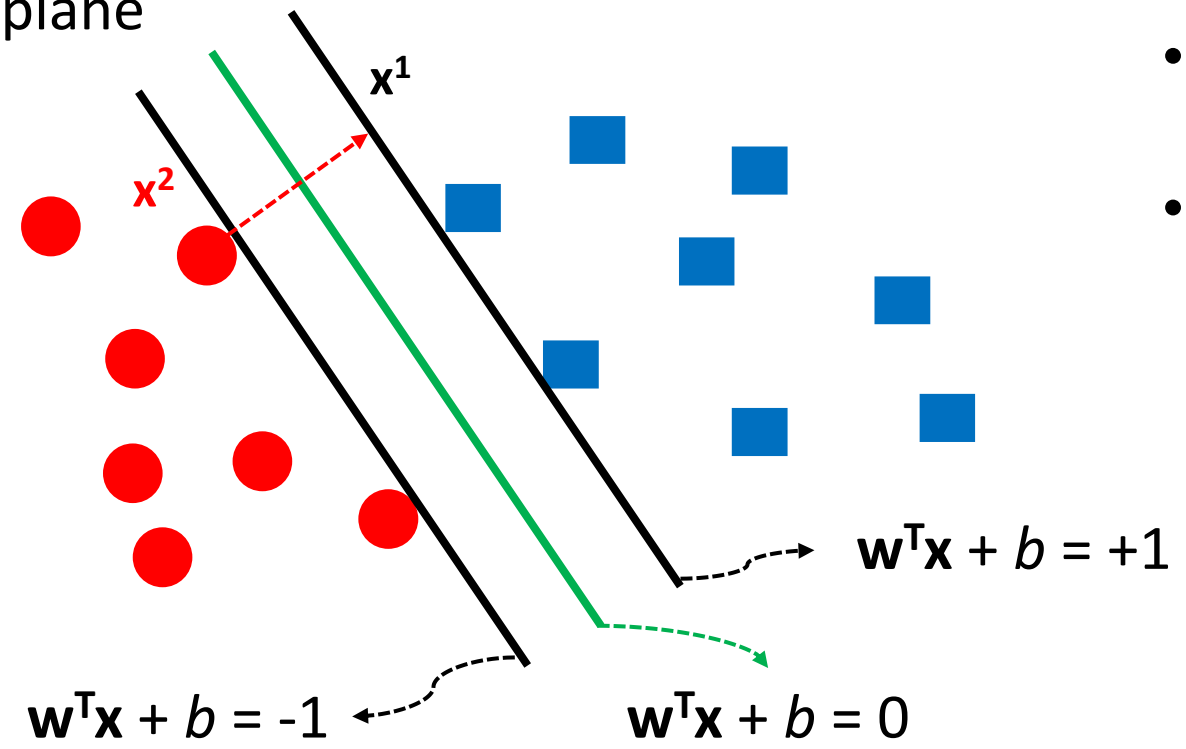
Computing the margin width

- How do we compute γ in terms of $\boldsymbol{w}$ and b?

- Claim: The vector $\mathbf{w}$ is perpendicular (orthogonal) to the hyperplane $\mathbf{w} \cdot x + b = 0$
  - If P and Q are in the plane with equation $w \cdot x + b = 0$, then $w \cdot P = -b$ and $w \cdot Q = -b$, so $w \cdot (Q - P) = 0$.
  - This means that the vector w is orthogonal to any vector PQ between points P and Q of the plane.

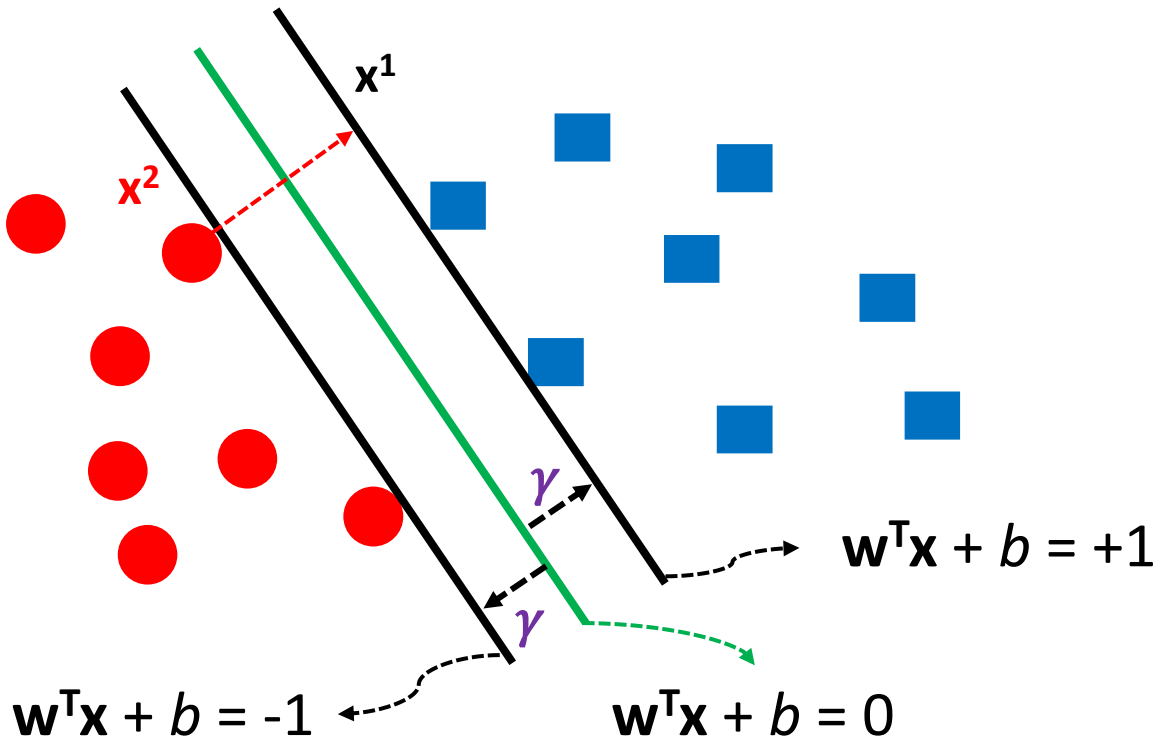
Computing the margin width

- Consider one of the support vectors, (say, x^2 on the black line) and let x^1 be the projection of x^2 to the upper hyperplane



- $\boldsymbol{x^1} = \boldsymbol{x^2} + \lambda\, \boldsymbol{w}$ for some value of λ
  - The line from $\boldsymbol{x^2}$ to $\boldsymbol{x^1}$ is perpendicular to the planes
  - So to get from $\boldsymbol{x^2}$ to $\boldsymbol{x^1}$ travel some distance in direction $\boldsymbol{w}$

Computing the margin width



What we know:

$w^T x^1 + b = +1$

$w^T x^2 + b = -1$

$x^1 = x^2 + \lambda w$

$\|x^1 - x^2\| = 2\gamma$

It's now easy to get γ in terms of $\boldsymbol{w}$ and b

Computing the margin width

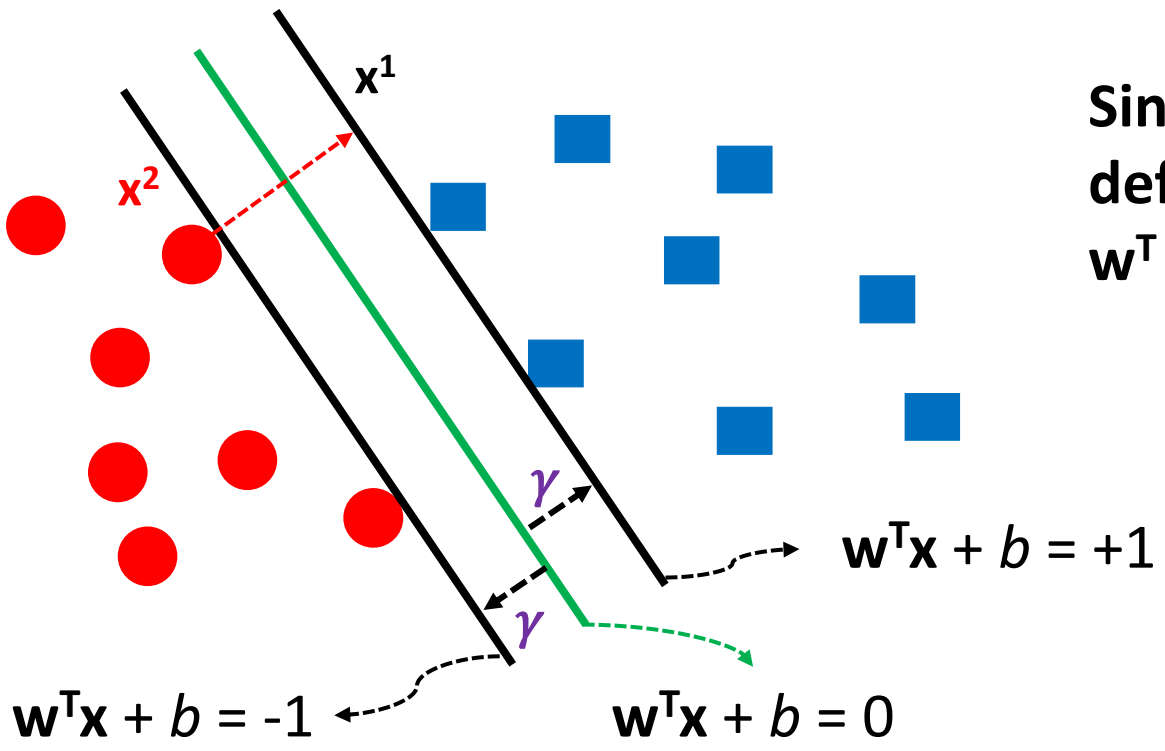
What we know:

$\mathbf{w}^T \boldsymbol{x}^1 + b = +1$

$\mathbf{w}^T \boldsymbol{x}^2 + b = -1$

$\boldsymbol{x}^1 = \boldsymbol{x}^2 + \lambda \boldsymbol{w}$

$|\boldsymbol{x}^1 - \boldsymbol{x}^2| = 2\gamma$



Since x^1 is on the hyperplane defined by w.x + b = +1, we know that $w^T x^1 + b = 1$. If we substitute for x^1:

$w^T x^1 + b = 1$

=>

$w^T (x^2 + \lambda w) + b = 1$

=>

$-1 + \lambda w^T w = 1$

=>

$\lambda = 2/ w^T w$

Computing the margin width

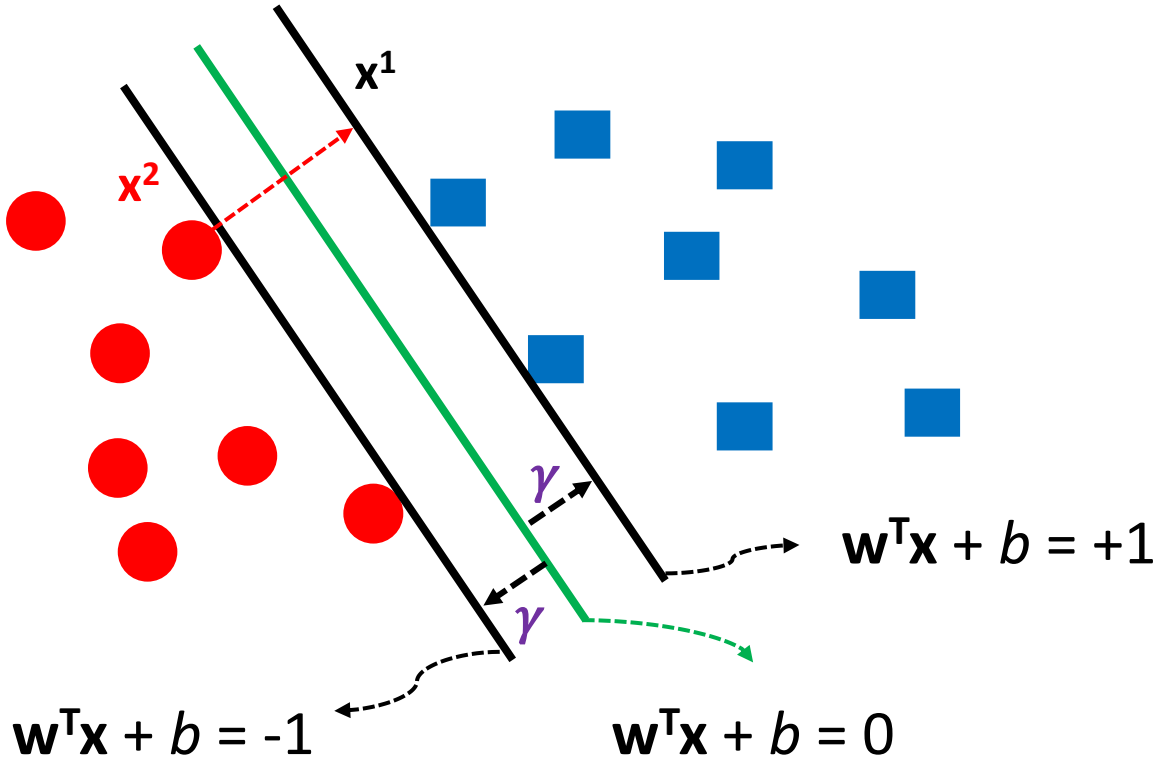
What we know:

$\mathbf{w}^T \boldsymbol{x}^1 + b = +1$

$\mathbf{w}^T \boldsymbol{x}^2 + b = -1$

$\boldsymbol{x}^1 = \boldsymbol{x}^2 + \lambda \boldsymbol{w}$

$|\boldsymbol{x}^1 - \boldsymbol{x}^2| = 2\gamma$



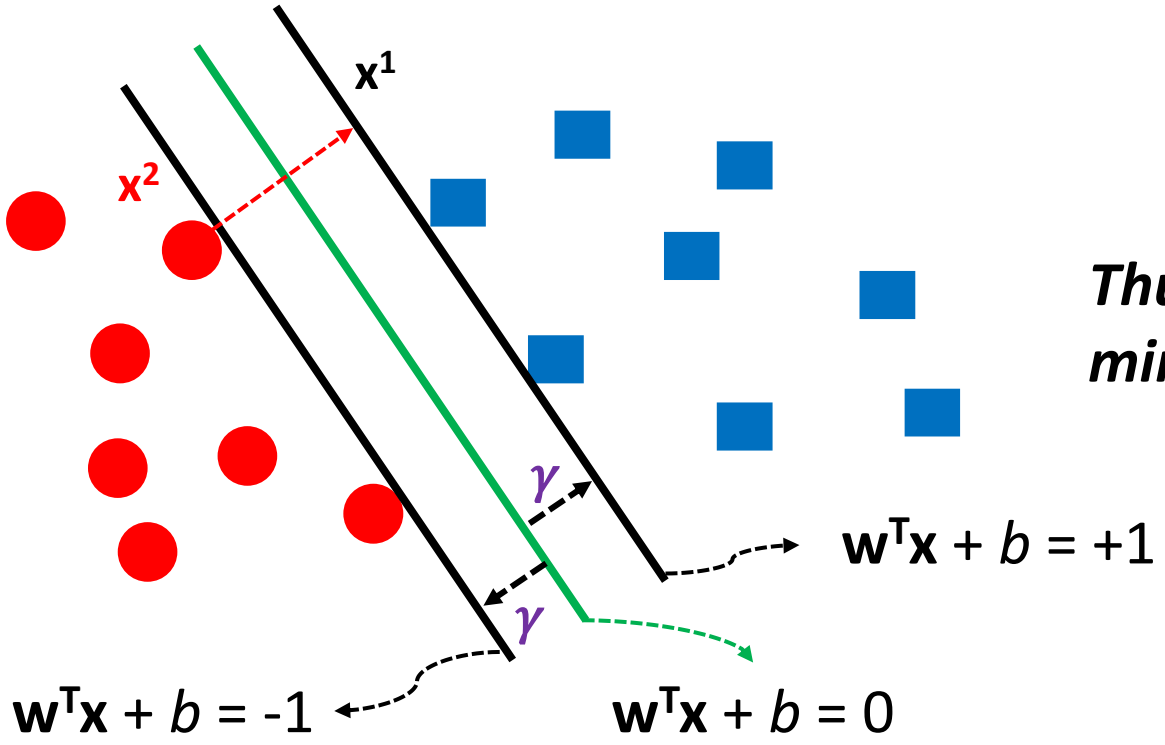
$$2\gamma = \| \boldsymbol{x}^1 - \boldsymbol{x}^2 \|$$
$$= \| \lambda \boldsymbol{w} \|$$
$$= \lambda \| \boldsymbol{w} \|$$

Because $\lambda = 2/\, \mathrm{w}^T \mathrm{w}$

=>

$$2\gamma = \frac{2\|\boldsymbol{w}\|}{\mathrm{w}^T \mathrm{w}} = \frac{2}{\|\boldsymbol{w}\|}$$

Computing the margin width



$\gamma = {}^{1}/_{\|w\|}$

Thus, maximizing γ is the same as minimizing $\| w \|$

The Objective of SVM

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{w}^T\mathbf{x}^i + b \geq 1 \text{ if } y^i = +1 \\ \mathbf{w}^T\mathbf{x}^i + b \leq -1 \text{ if } y^i = -1 \end{cases}$$

But, why would this constraint serve in materializing *large margin classification*?

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{w}^T\mathbf{x}^i + b \geq 1 \text{ if } y^i = +1 \\ \mathbf{w}^T\mathbf{x}^i + b \leq -1 \text{ if } y^i = -1 \end{cases}$$

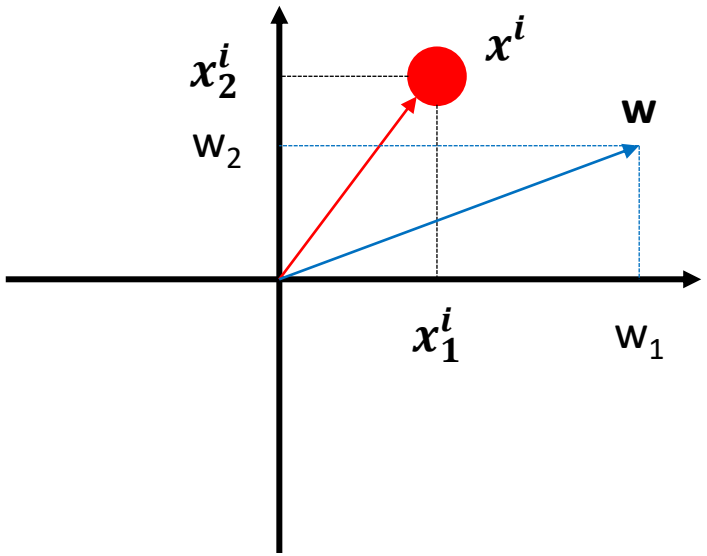
For illustrative purposes, let us assume only two features (i.e., $\mathbf{x}^i = [x_1^i, x_2^i]$ and $\mathbf{w} = [w_1, w_2]$) and b = 0

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{w}^T\mathbf{x}^i + b \geq 1 \text{ if } y^i = +1 \\ \mathbf{w}^T\mathbf{x}^i + b \leq -1 \text{ if } y^i = -1 \end{cases}$$



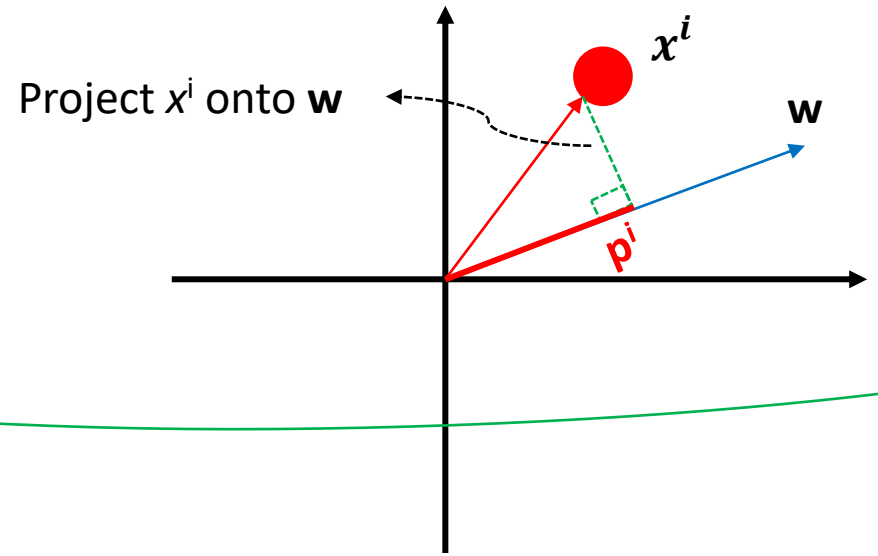
What is the inner product of **w** and x^i (i.e., $\mathbf{w}^T x^i$)?

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{w}^T\mathbf{x}^i + b \geq 1 \text{ if } y^i = +1 \\ \mathbf{w}^T\mathbf{x}^i + b \leq -1 \text{ if } y^i = -1 \end{cases}$$



What is the inner product of **w** and x^i (i.e., $\mathbf{w}^T x^i$)?

$$p^i . \| w \|$$

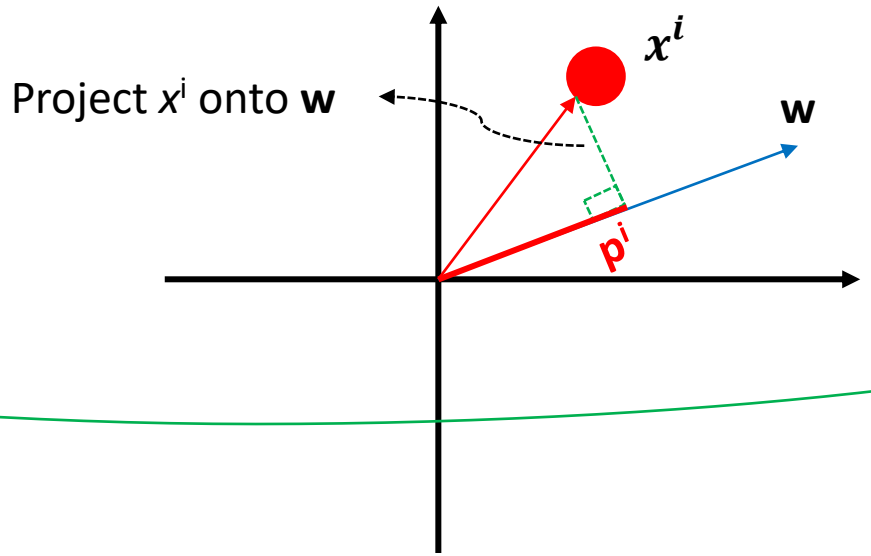
$$= w_1 . x_1^i + w_2 . x_2^i$$

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{p}^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ \mathbf{p}^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



What is the inner product of **w** and x^i (i.e., $\mathbf{w}^T x^i$)?

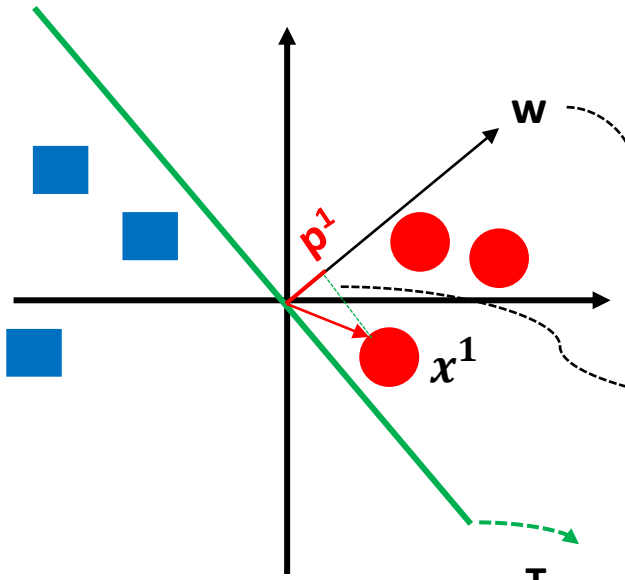
$$\mathbf{p}^i . \| \boldsymbol{w} \| = w_1 . x_1^i + w_2 . x_2^i$$

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

orthogonal to the decision boundary

Project x^1 onto **w**

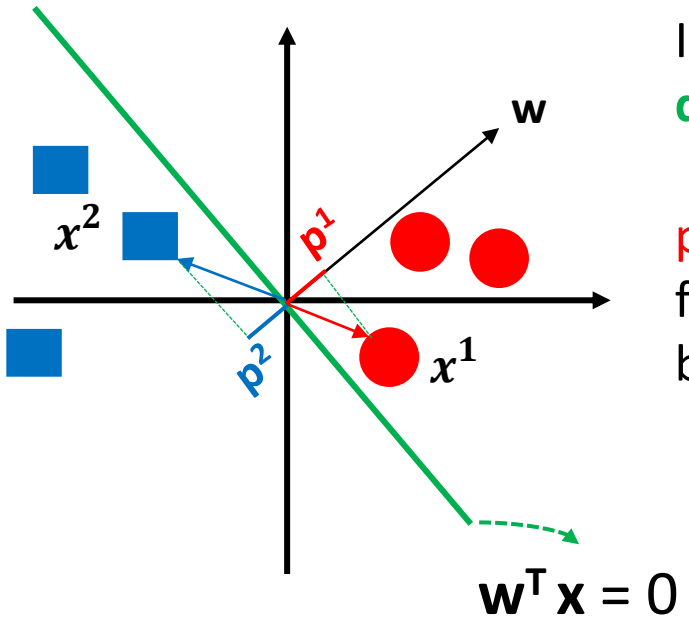
$\mathbf{w}^T \mathbf{x} = 0$ for simplicity, assuming **b = 0**

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{p}^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ \mathbf{p}^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

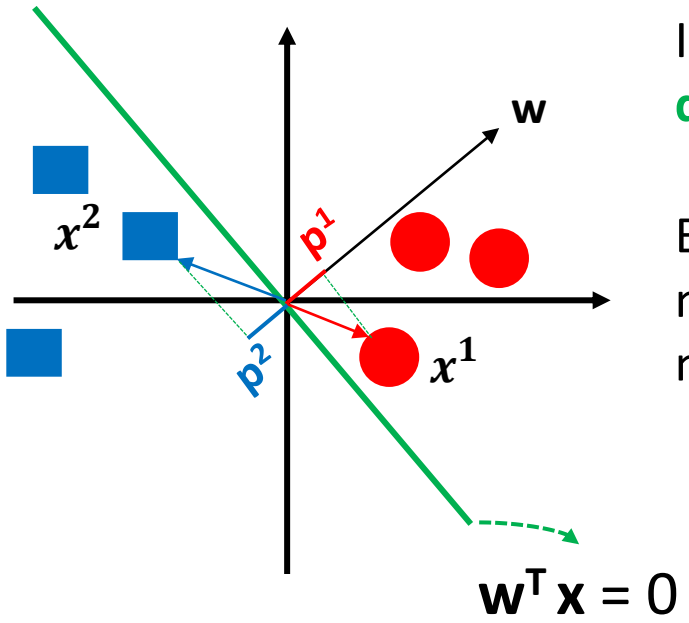
p^1 is positive and very small, hence, for $\mathbf{p}^1 . \| \boldsymbol{w} \|$ to be ≥ 1, $\| \boldsymbol{w} \|$ has to be very large!

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

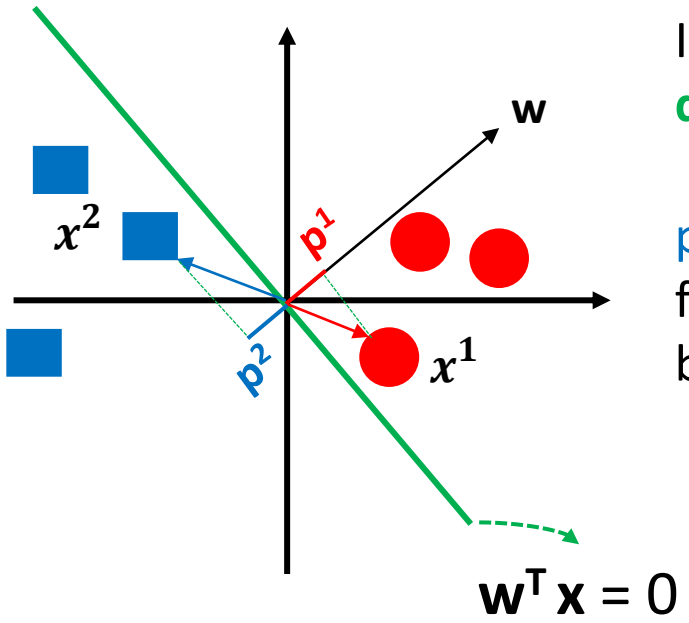
But, the optimization objective is to minimize $\| \boldsymbol{w} \|$, hence, SVM will not prefer this decision boundary

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

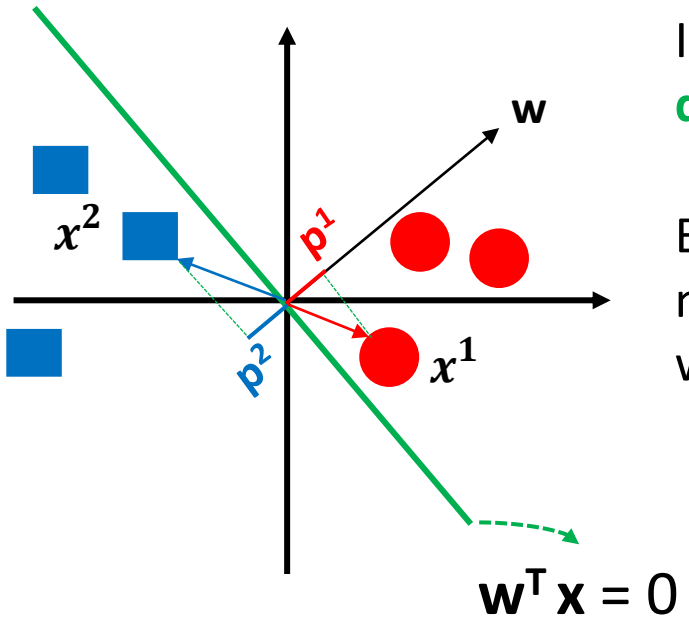
p^2 is negative and very small, hence, for $p^2 . \| \boldsymbol{w} \|$ to be ≤ -1, $\| \boldsymbol{w} \|$ has to be very large!

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

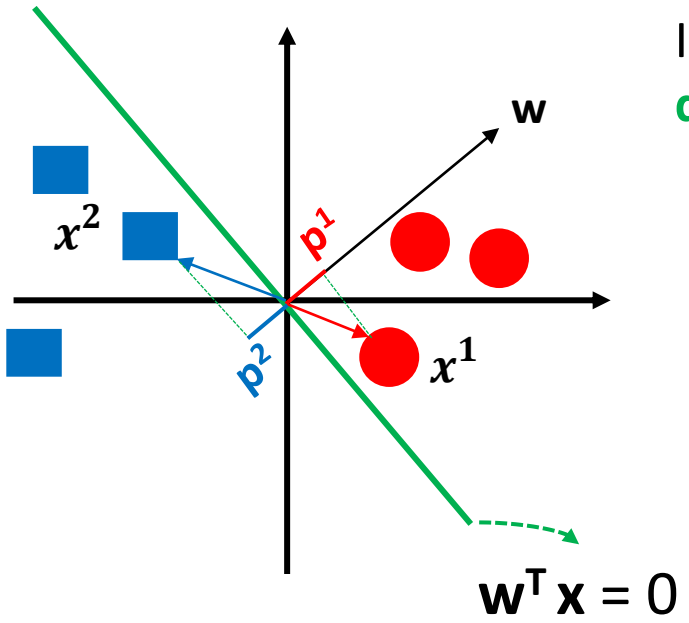
But, the optimization objective is to minimize $\| \boldsymbol{w} \|$, thus, again, SVM will not prefer this decision boundary

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **green decision boundary**, will it choose it?

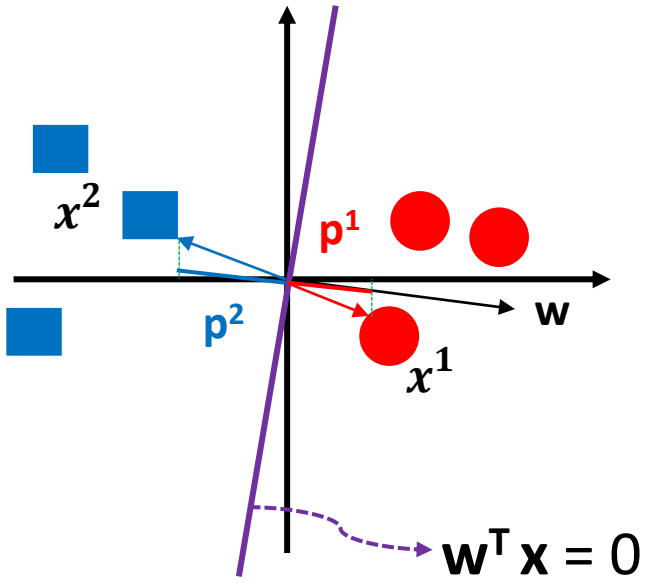
NO

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} \mathbf{p}^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ \mathbf{p}^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **purple decision boundary**, will it choose it?

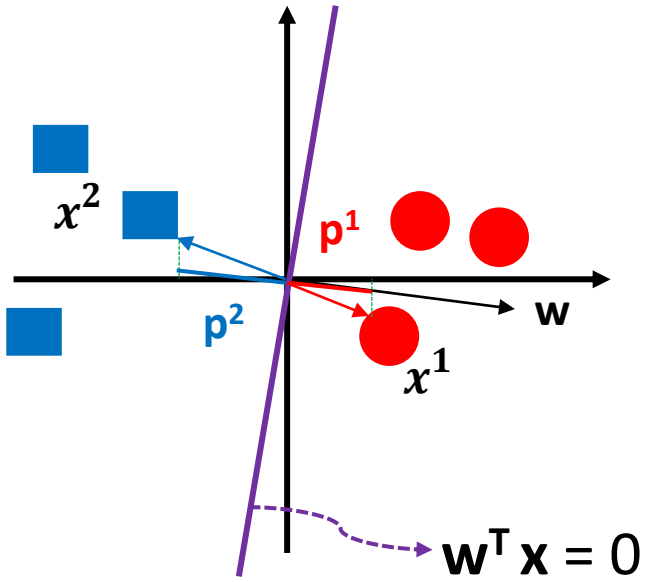
$\mathbf{p}^1$ is positive and bigger now, hence, for $\mathbf{p}^1 . \| \boldsymbol{w} \|$ to be ≥ 1, $\| \boldsymbol{w} \|$ can be smaller, aligning better with the optimization objective

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **purple decision boundary**, will it choose it?

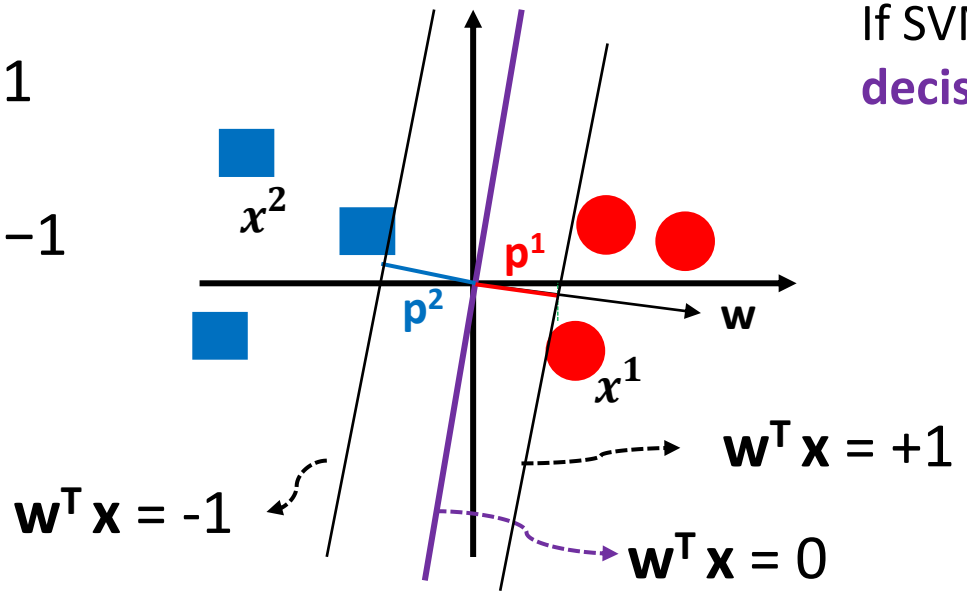
p^2 is negative and bigger now, hence, for $p^2 . \| \boldsymbol{w} \|$ to be ≤ -1, $\| \boldsymbol{w} \|$ can be smaller, aligning better with the optimization objective

The Objective of SVM - Illustration

- More formally, the goal of SVM can be stated as follows:

Given a training set $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$, minimize $\| w \|$ (by varying **w** and b) subject to the constraint that for all i = 1, 2, . . . , n,

$$\begin{cases} p^i . \| \boldsymbol{w} \| \geq 1 \text{ if } y^i = +1 \\ p^i . \| \boldsymbol{w} \| \leq -1 \text{ if } y^i = -1 \end{cases}$$



If SVM encounters this **purple decision boundary**, will it choose it?

YES

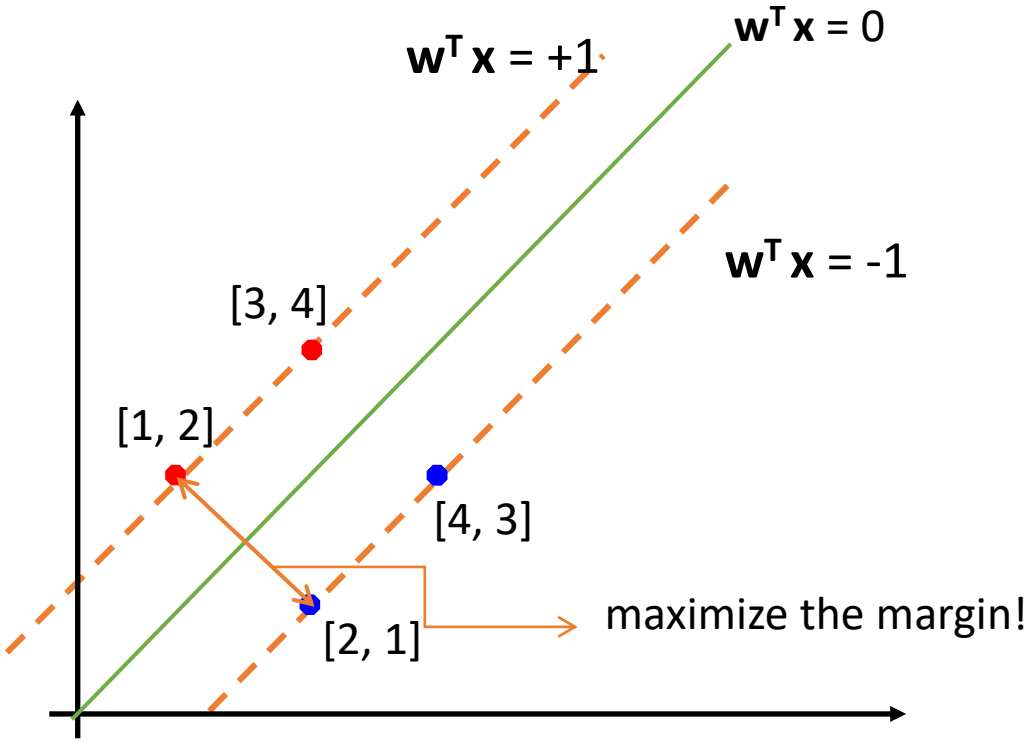
Example

- Consider the following training examples, assuming $\mathbf{w} = [w_1, w_2]$
- How to solve for $\mathbf{w}$ and b?
- Our goal is to maximize the margin subject to the constraints $y^i(\mathbf{w}^T\mathbf{x}^i + b) \geq 1$, which can be derived from the training examples

x	y	Constraints
[1, 2]	+1	$u + 2v + b \geq 1$
[2, 1]	-1	$2u + v + b \leq -1$
[3, 4]	+1	$3u + 4v + b \geq 1$
[4, 3]	-1	$4u + 3v + b \leq -1$

Example

- Consider the following training examples, assuming $\mathbf{w} = [w_1, w_2]$
- it is easy to find that b = 0 and $\mathbf{w} = [-1, +1]$ using geometric interpretation.



Decision Tree for Malware Detection

- Malware (malicious software): any intrusive software developed by cybercriminals to steal data and damage or destroy computers and computer systems
- Examples of common malware include
  - virus
  - trojan
  - botnet
  - downloader
  - rootkit
  - ransomware
  - APT
  - spyware
  - zero day

Common Types of Malware

- **Virus**
  - A self-replicating program that reproduces its code by attaching copies into other executable codes, designed to disrupt a system's ability to operate
- **Trojan**
  - Executable that appears as legitimate and harmless, but once it is launched, it executes malicious instructions in the background
- **Botnet**
  - Malware that has the goal of compromising as many possible hosts of a network, in order to put their computational capacity at the service of the attacker
- **Downloader**
  - Malware that downloads malicious libraries or portions of code from the network and executes them on victim hosts
  - Malicious code that exists only to download other malicious code

Common Types of Malware

- **Rootkit**
  - Malware that compromises the hosts at the operating system level and often comes in the form of device drivers, making the various countermeasures (e.g. antiviruses installed on the endpoints) ineffective
- **Ransomware**
  - Malware that proceeds to encrypt files stored inside the host machines, asking for a ransom from the victim to obtain the decryption key which is used for recovering the original files
- **APT**
  - APT (Advanced Persistent Threat) is a form of tailored attack that exploits specific vulnerabilities on the victimized hosts

Common Types of Malware

- **Spyware**
  - Spyware is malicious software that runs secretly on a computer and reports back to a remote user. Rather than simply disrupting a device's operations, often used to steal financial or personal information
  - A specific type of spyware is a keylogger, which records keystrokes to reveal passwords and personal information
- **Zero day**
  - Malware that exploits vulnerabilities not yet disclosed to the community of researchers and analysts, whose characteristics and impacts in terms of security are not yet known, and therefore go undetected by antivirus software

Malware Detection Strategies

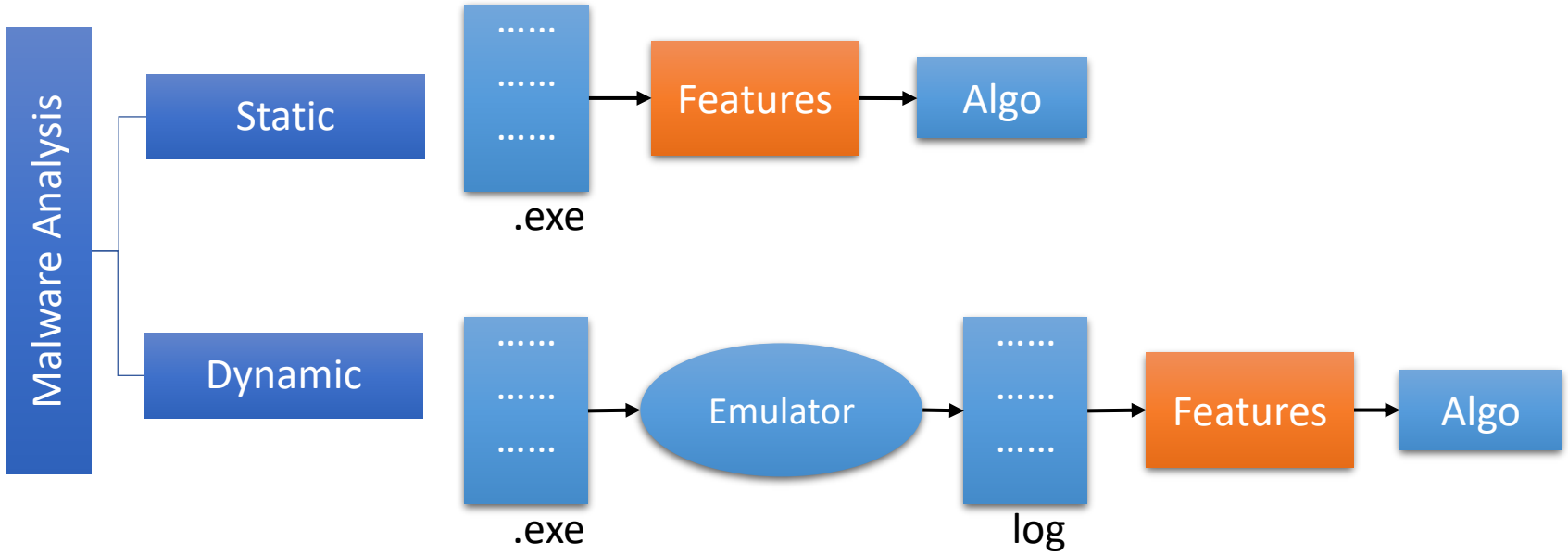
- Detection activities that can be easily automated
  - File hash calculation: To identify known threats already present in the knowledge base
  - System monitoring: To identify anomalous behavior of both the hardware and the operating system
    - such as an unusual increase in CPU cycles
    - a particularly heavy disk writing activity
    - changes to the registry keys
    - the creation of new and unsolicited processes in the system …
  - Network monitoring: To identify anomalous connections established by host machines to remote destinations

Malware Analysis Techniques

- Two fundamental approaches to malware analysis
  - Static and dynamic
- Static (code) analysis: analyzing the code or structure of a program to determine its function without running it
- Dynamic (behavioral) analysis: running the malware and observing its behavior on the system
  - Before running the malware safely, a sandbox environment must be set up that will allow to study the running malware without damaging to the system or network.
- Usually fulfilled by security specialists

Steps for building a ML-based Malware detector

- After analyzing the malware, we can extract the features of the binary files (whether legitimate or suspect), and store them in a dataset of artifacts with which to train our algorithms



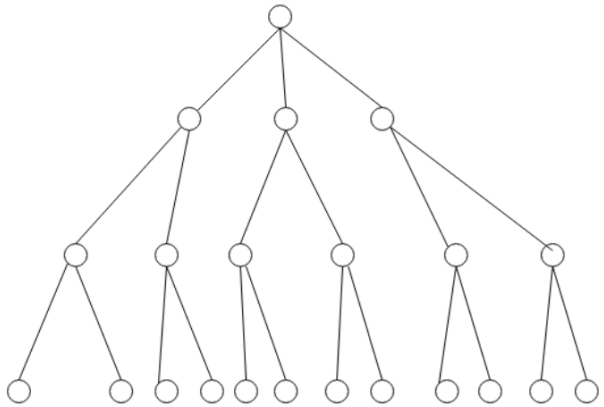
Steps for building a ML-based Malware detector

- Collect
  - Examples of malware and benignware
  - Used to train the model
- Extract
  - Features from each training example to represent the data
  - This step also includes research to design good features that will help the machine learning model make accurate inferences – domain experts
- Train and Test
  - Machine learning model to detect malware using the features
  - Our focus: malicious or non-malicious

Decision tree Malware Detectors

- Now, we will deal with the classification problems solved by decision trees in the context of detecting malware threats
  - supervised learning method
  - classify the data into malicious and non-malicious categories
- Model the learning process based on a sequence of if-then-else decisions

  => a tree structure
- Non-linear classifier
  - decision boundaries are not reducible to straight lines or hyperplanes in the space



Next

- Decision Tree Overview
  - ID3 is a simple decision tree learning algorithm developed by Ross Quinlan
  - The basic idea: construct the decision tree by employing a top-down, greedy search through the given sets to test each attribute at every tree node
  - To select the attribute that is most useful for classifying a given sets, we introduce a metric---information gain
- Information Gain
- Build a Decision Tree

Decision Tree Example

- Here is an example for decision tree: play tennis.
- Outlook, temperature, humidity and wind are the attributes: X
- PlayTennis is the label: Y

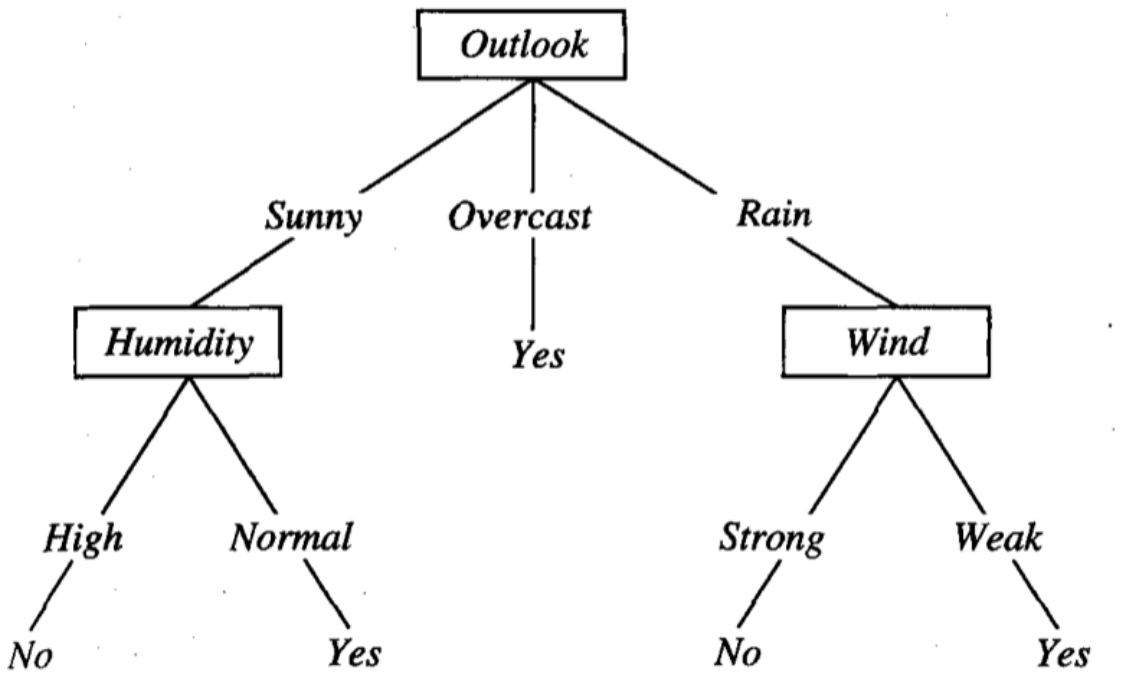
- Decision Tree
  - based on the attributes X
  - make decisions on the label Y

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

For ease of understanding, we will follow an example from Quinlan's ID3

Decision Tree Example

- A possible decision tree for the data:



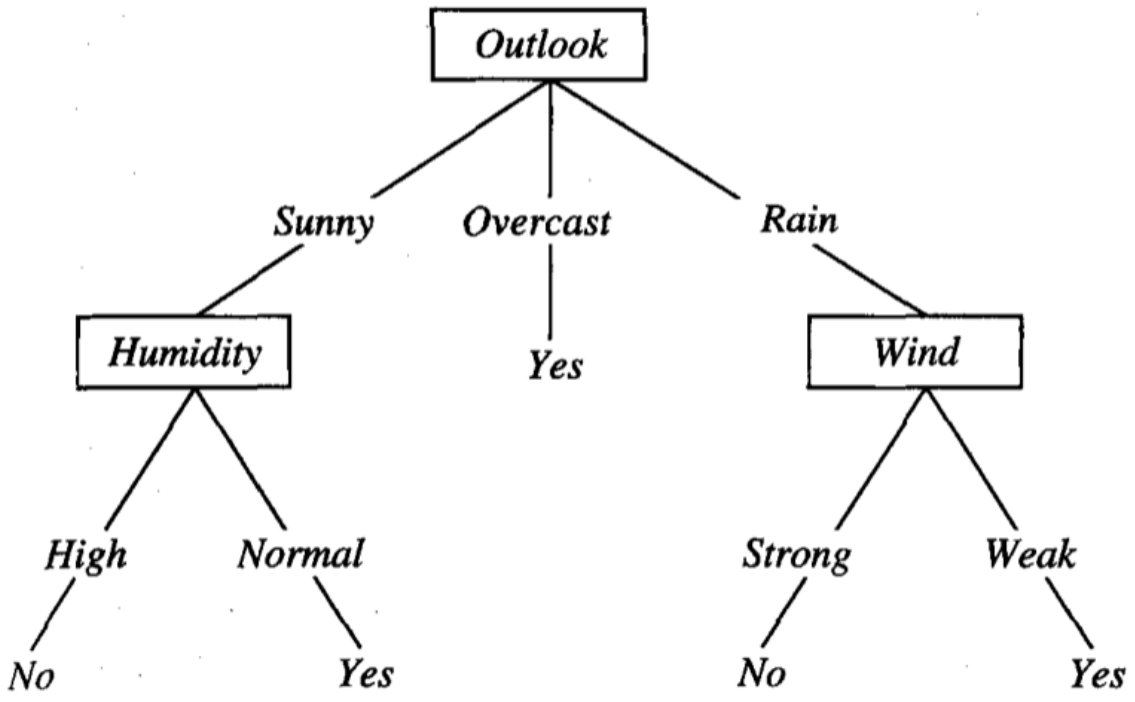
- Each internal node:
  - tests one attribute X_i (such as outlook, humidity, wind…)
- Each branch from a node:
  - selects one value for X_i
- Each leaf node:
  - prediction of Y (yes or no)

Decision Tree Example

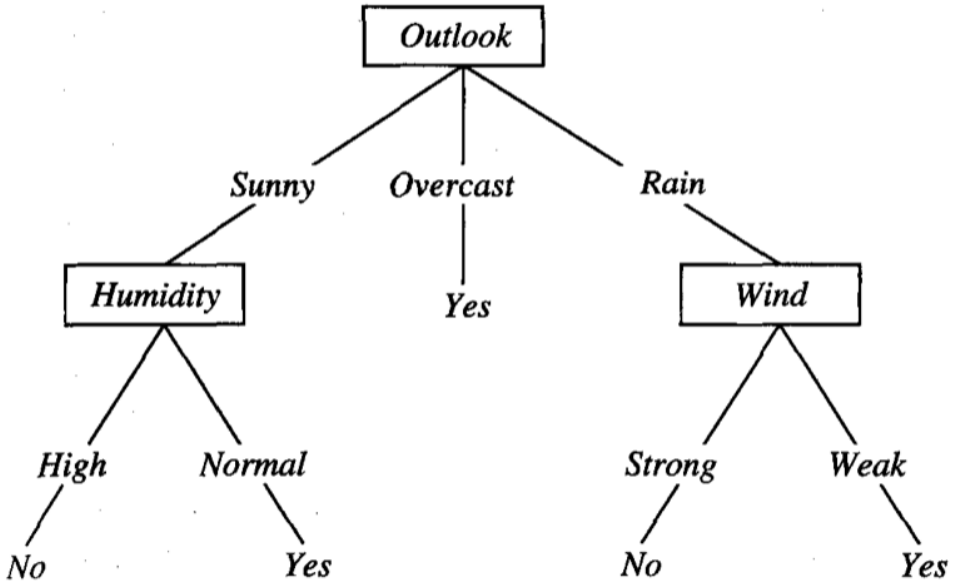
- What is the prediction for:
- outlook=sunny, temperature=mild, humidity=high, wind=weak ?

- Example gets sorted down the leftmost branch of this decision tree and classified a a negative instance (i.e., the tree predicts that PlayTennis no).

A possible decision tree for the data:



- Decision trees classify examples by sorting top down.
- An example is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example.
- This process is then repeated for the subtree rooted at the new node



Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

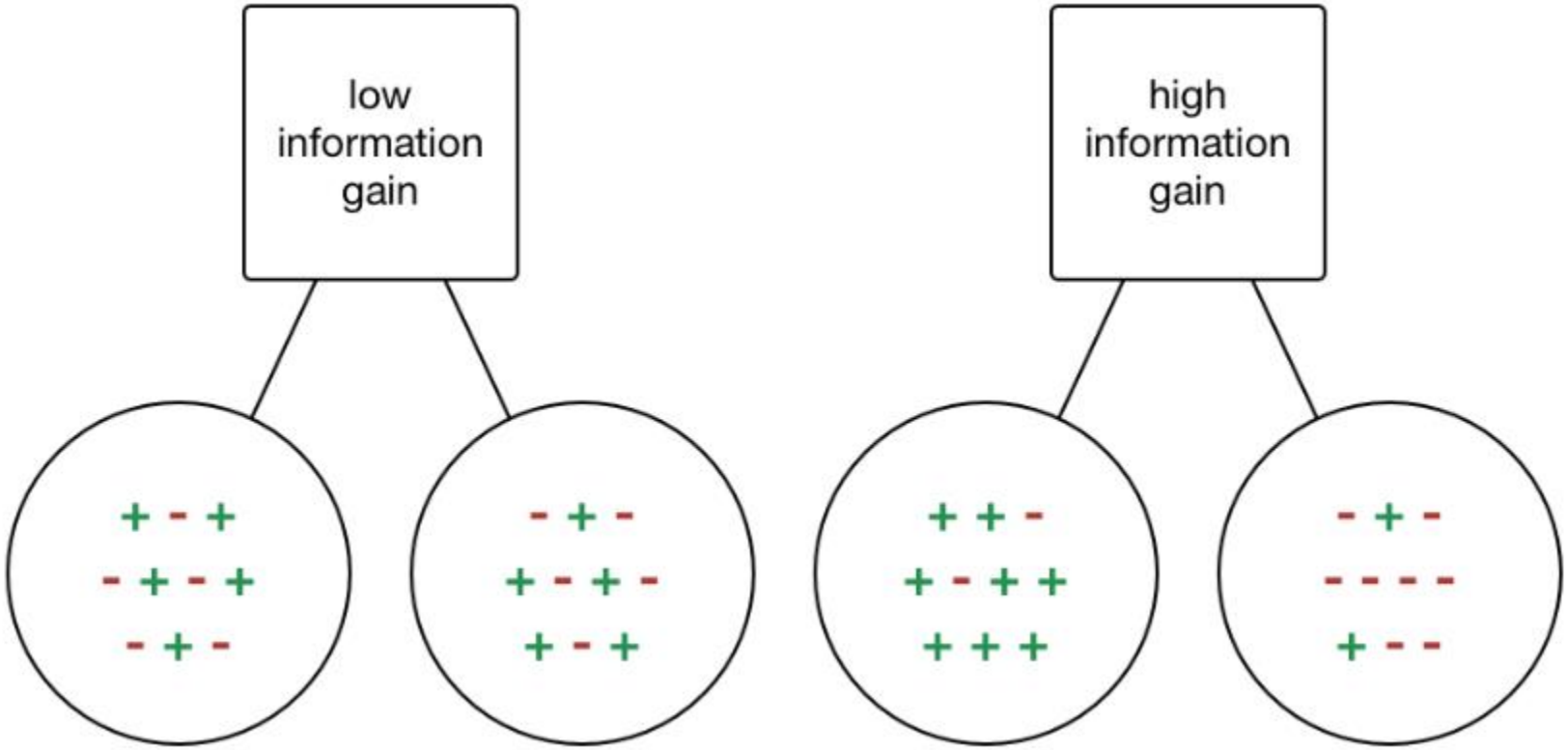
Information Gain

- To build a good decision tree
  - We need to find good splits
  - i.e., $\boldsymbol{X}_1$ or $\boldsymbol{X}_2$
  - select which attribute to test at each node in the tree
- Increase certainty about classification (yes or no) after split
- To measure classification certainty, we introduce information gain

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Information Gain

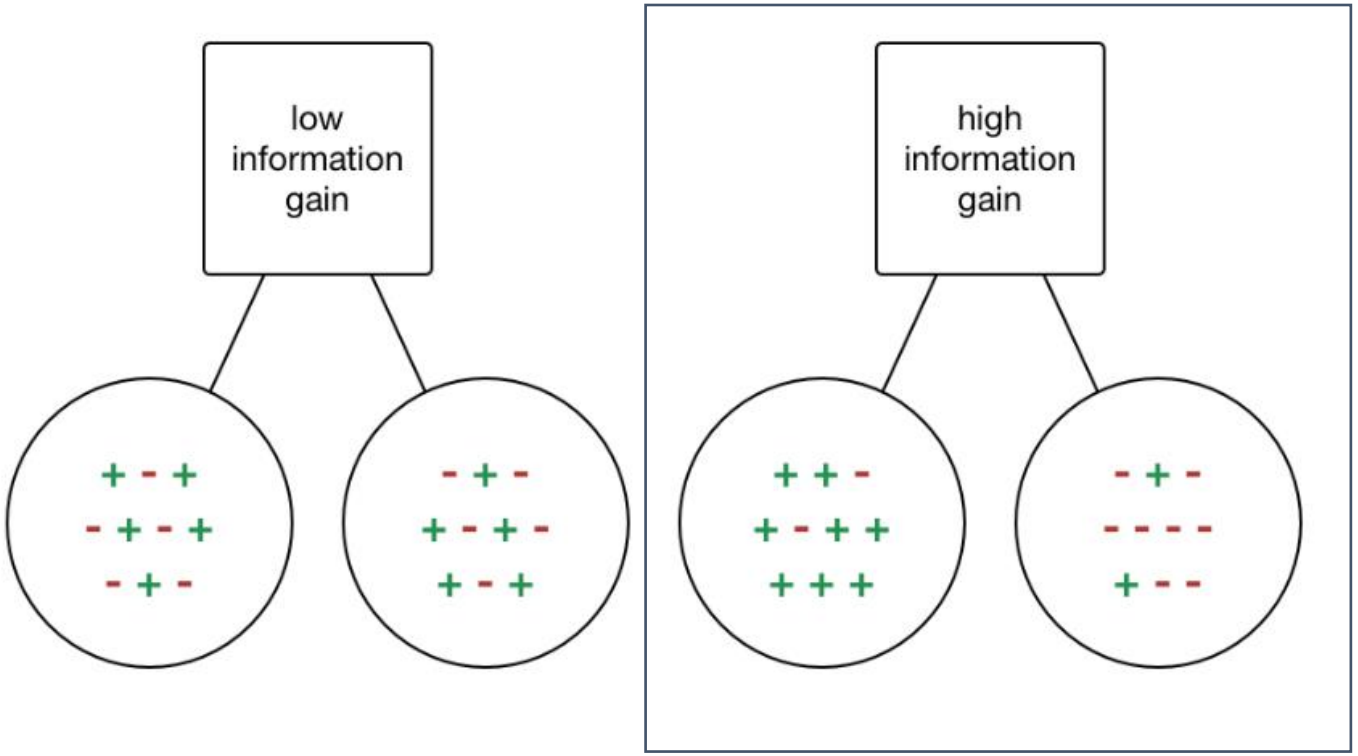
- Which split is more informative? Why?



Information Gain

- The split on the right is more informative, since after the split the distribution of data becomes more discriminative.
  - i.e., more similar labels in one partition.

- We use entropy and information gain to evaluate the uncertainty and goodness of a split.



Entropy

- Entropy can be used as a measure of uncertainty in information theory.
- Entropy $H(Y)$ of a random variable Y with n different possible values:

$$H(Y) = -\sum_{i=1}^{n} P(y_i) log_2 P(y_i)$$

where $P(y_i)$ is the proportion of values in Y that belong to the *i*-th class

- Entropy is 0 if all values of Y belong to the same class.

Entropy example

- If Y is a binary variable as shown in the table.
- Y has two values: T (True) and F (False).
- In the following equation:

$$H(Y) = -\sum_{i=1}^{n} P(y_i) log_2 P(y_i)$$

- We have $n = 2$.

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Entropy example

- There are
  - 5 instances with $Y = T$
  - and 3 instances with $Y = F$

- Therefore, $P(Y = T) = 5/8$ and $P(Y = F) = 3/8$

- We can calculate its entropy as:

$$H(Y) = -\frac{5}{8} log_2 \frac{5}{8} - \frac{3}{8} log_2 \frac{3}{8} \approx 0.954$$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Entropy after split

- **Entropy $H(Y)$ of a random variable Y with n different possible values:**

$$H(Y) = -\sum_{i=1}^{n} P(y_i) log_2 P(y_i)$$

- **Conditional entropy $H(Y|X = x_j)$ of Y given $X = x_j$:**

$$H(Y|X = x_j) = -\sum_{i=1}^{n} P(y_i|X = x_j)\, log_2 P(y_i \mid X = x_j)$$

- The above equation: split the data according to the value of X, the entropy of random variable Y with $X = x_j$.

- **Expected entropy $H(Y|X)$ after split:**

$$H(Y|X) = \sum_{j=1}^{k} P(x_j)\, H(Y|X = x_j)$$

- The above equation: X has k possible values, $P(x_j)$ is the probability that $X = x_j$.

Entropy after split example

- Suppose we split the data according to the value of X_1.
- X_1 has two possible values: T and F.
- We can compute the conditional entropy for $X_1 = T$ and $X_1 = F$.

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Entropy after split example

- Find the conditional entropy for $X_1 = T$
- After the split, when $X_1 = T$, all corresponding $Y = T$.
- Therefore:
  - $P(Y = T|X_1 = T) = 1$ and $P(Y = F|X_1 = T) = 0$
- We have the following entropy when $X_1 = T$:

$$H(Y|X_1 = T) = -\sum_{i=1}^{n} P(y_i|X_1 = T)\, log_2 P(y_i|X_1 = \mathrm{T})$$
$$= -1 * log_2(1) - 0 * log_2(0) = 0$$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Entropy after split example

- Find the conditional entropy for $X_1 = F$
- After the split, when $X_1 = F$, one corresponding $Y = T$, and three corresponding $Y = F$.
- Therefore,
  - $P(Y = T|X_1 = F) = 1/4$ and $P(Y = F|X_1 = F) = 3/4$
- We have the following entropy when $X_1 = F$:

$$H(Y|X_1 = F) = -\sum_{i=1}^{n} P(y_i|X_1 = F)\, log_2 P(y_i|X_1 = \mathrm{F})$$
$$= -\frac{1}{4} * log_2 \frac{1}{4} - \frac{3}{4} * log_2 \frac{3}{4} = 0.811$$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Entropy after split example

- After the above computations, we then find the expected entropy.
- There are four instances with $X_1 = T$, and four instances with $X_1 = F$
- Therefore, we have $P(X_1 = T) = P(X_1 = F) = 1/2$
- We have the following expected entropy for X_1 :

$$H(Y|X_1) = \frac{1}{2} * 0 + \frac{1}{2} * 0.811 = 0.406$$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Information Gain

- Information gain $I(Y, X)$ is defined as the expected reduction in entropy of target variable Y after split over variable X

$$I(Y, X) = H(Y) - H(Y|X)$$

Information Gain example

- Recall that we have the entropy of Y, $H(Y) = 0.954$, and expected entropy $H(Y|X_1) = 0.406$.

- We can easily compute the information gain for data in the table after split over variable X_1

$$I(Y, X_1) = H(Y) - H(Y|X_1) = 0.954 - 0.406 = 0.548$$

Exercise: What is the information gain $I(Y, X_2)$?

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Next - Build a decision tree

What is a good decision tree

- When we learn a decision tree, we want to determine which attribute in a given dataset of training examples is the most useful for discriminating between the classes to be learned

- We use information gain which tells us how important a given attribute is to decide the order of attributes for splitting in a decision tree.

- We introduce ID3 (Iterative Dichotomiser 3) invented by Ross Quinlan for learning decision tree.

Overview of ID3

- ID3 employs a top-down manner to construct a decision tree.

- ID3 builds the decision tree from the root node, and at each step, it makes the decision of node to be split based on the information gain.

ID3 Algorithm

- 1. Base entropy is computed based on the distribution of label Y.
- 2. Each attribute is evaluated to get the information gain which determines how well it classifies the training data.
- 3. The best attribute is selected and used as the root node of the tree.
- 4. A descendant of the root node is created for each possible value of this attribute.
- 5. The previous processes are repeated on the remaining attributes to find the best attribute based on the information gain as the descendant node.
- 6. Finally, we form a decision tree.

ID3 example

- We use the Play Tennis data as an example to build a decision tree using ID3.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- In the given data, there are 14 days in total, including 9 positive and 5 negative labels.
- Therefore, the base entropy is:

$$H(Y) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14}$$

$$= 0.940$$

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- We show an example information gain computation for the attribute Outlook.
- Outlook has three values: Sunny, Overcast and Rain.

$$P(sunny) = \frac{5}{14}$$

$$P(overcast) = \frac{4}{14}$$

$$P(rain) = \frac{5}{14}$$

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- The conditional entropy given Outlook=sunny can be computed as:

$$H(Y|sunny) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$$

Because when outlook=sunny, there are 2 positive and 3 negative labels.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- The conditional entropy given Outlook=overcast can be computed as:

$$H(Y|overcast) = -\frac{0}{4}log_2\frac{0}{4} - \frac{4}{4}log_2\frac{4}{4} = 0$$

Because when outlook=overcast, there are 4 positive and 0 negative labels.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- The conditional entropy given Outlook=rain can be computed as:

$$H(Y|rain) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$$

Because when outlook=overcast, there are 3 positive and 2 negative labels.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- The expected entropy of attribute Outlook is:

$$H(Y|outlook) = P(sunny) * H(Y|sunny) + P(overcast) * H(Y|overcast) + P(rain) * H(Y|rain)$$
$$= \frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.694$$

(recall that: $P(sunny) = \frac{5}{14}, P(overcast) = \frac{4}{14}, P(rain) = \frac{5}{14}$)

ID3 example

- The information gain after split over outlook is:

$$I(Y, outlook) = H(Y) - H(Y|outlook)$$
$$= 0.940 - 0.694$$
$$= 0.246$$

ID3 example

- Similarly, we can compute the information gain for other attributes Temperature, Wind and Humidity:

$$I(Y, temperature) = 0.029$$

$$I(Y, wind) = 0.048$$

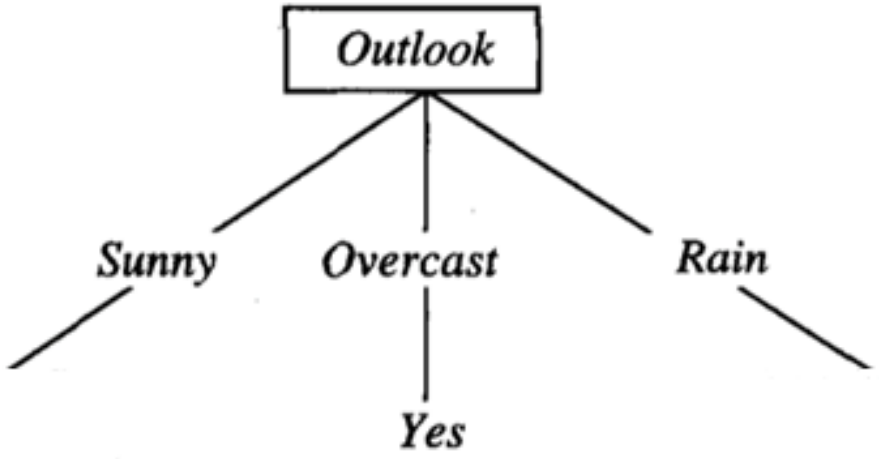
$$I(Y, humidity) = 0.152$$

- And from the previous computation:

$$I(Y, outlook) = 0.246$$

ID3 example

- Among all attributes, outlook has the most information gain.
- We make outlook the root node of the decision tree and its values as the branches.
- Because when outlook=overcast, the labels are all positive, we have a pure leaf node in this tree.



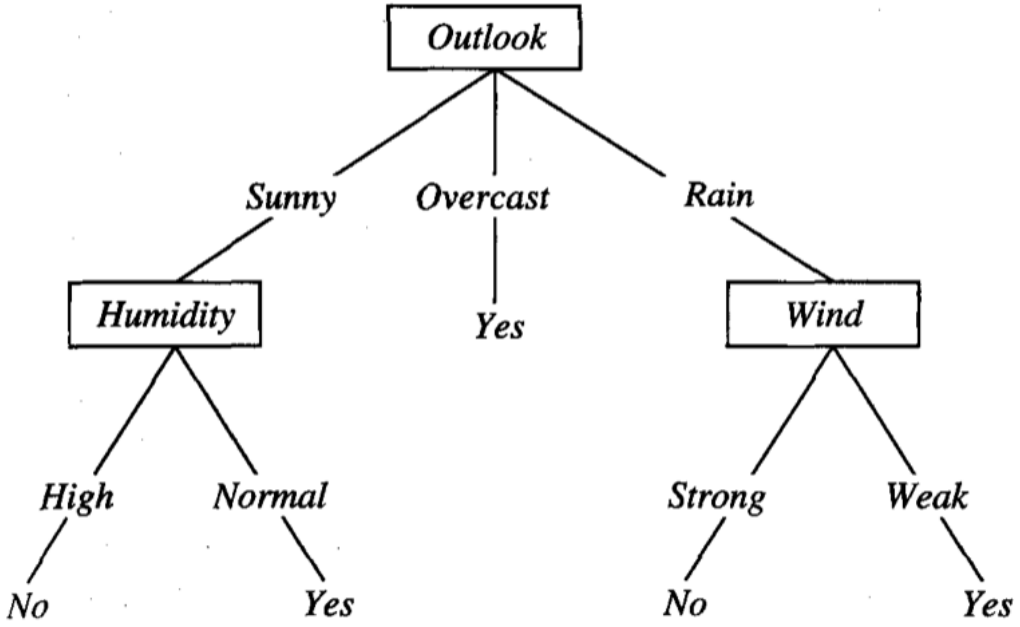
ID3 example

- For each branch of the root node, we repeat the computation of the information gain, and find the best node as the child node.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 example

- For sunny branch, we have humidity as the child node.
- For rain branch, we have wind as the child node.
- For each branch, we can find a label as the leaf node. Therefore, we build our decision tree:



References

- *Support Vector Machines, Andrew W. Moore, CMU School of Computer Science*
- *Introduction to Information Retrieval: support vector machines*

Network Intrusion Detection Case Study

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

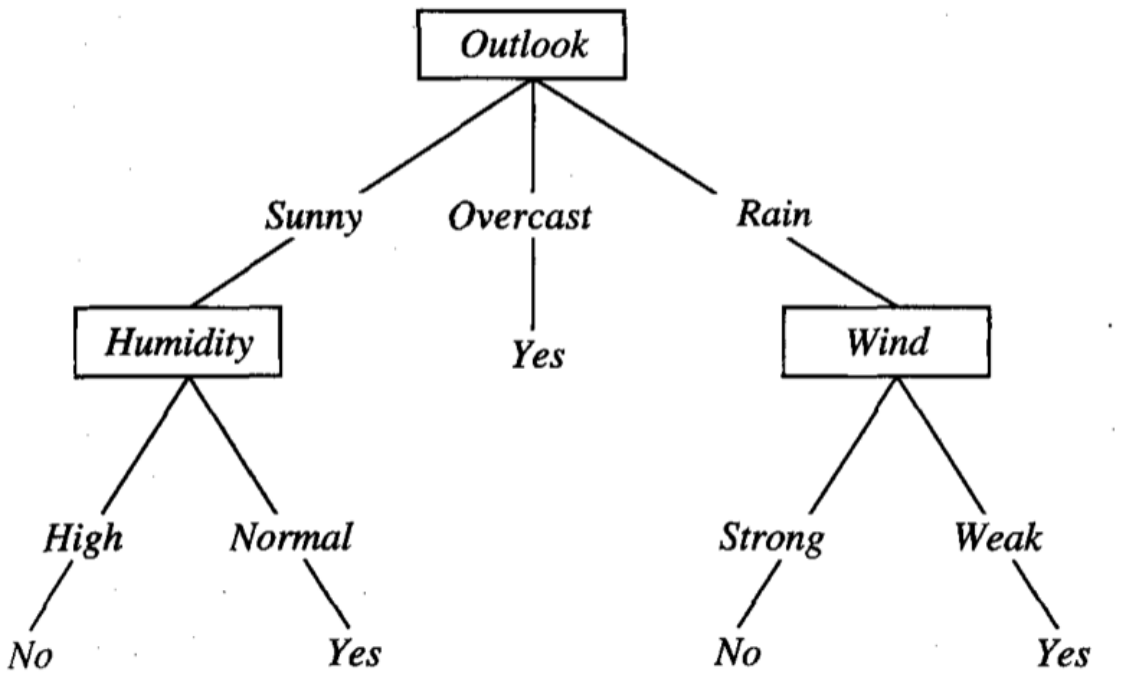
Disclaimer: The presentation materials come from various sources. For further information, check the references section

Recap: Decision Tree

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree Example

- A possible decision tree for the data:



- Each internal node:
  - tests one attribute X_i (such as outlook, humidity, wind…)
- Each branch from a node:
  - selects one value for X_i
- Each leaf node:
  - prediction of Y (yes or no)

What is a good decision tree

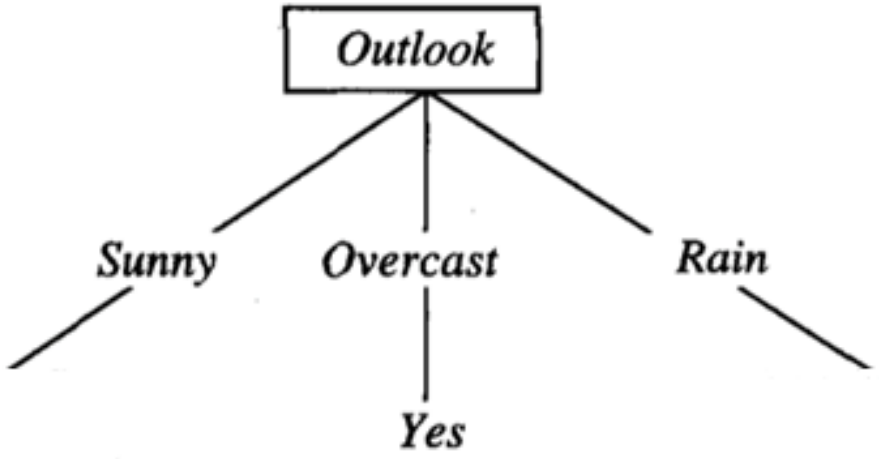
- When we learn a decision tree, we want to determine which attribute in a given dataset of training examples is the most useful for discriminating between the classes to be learned
- We use information gain (IG) which tells us how important a given attribute is to decide the order of attributes for splitting in a decision tree.
  - IG: expected reduction in entropy of target variable Y after split over variable X
- Algorithm for building Decision Tree: ID3
- ID3 employs a top-down manner to construct a decision tree.
- ID3 builds the decision tree from the root node, and at each step, it makes the decision of node to be split based on the information gain.

ID3 Algorithm

- 1. Base entropy is computed based on the distribution of label Y.
- 2. Each attribute is evaluated to get the information gain which determines how well it classifies the training data.
- 3. The best attribute is selected and used as the root node of the tree.
- 4. A descendant of the root node is created for each possible value of this attribute.
- 5. The previous processes are repeated on the remaining attributes to find the best attribute based on the information gain as the descendant node.
- 6. Finally, we form a decision tree.
- (if there is a tie on IG, randomly pick one of the attributes)

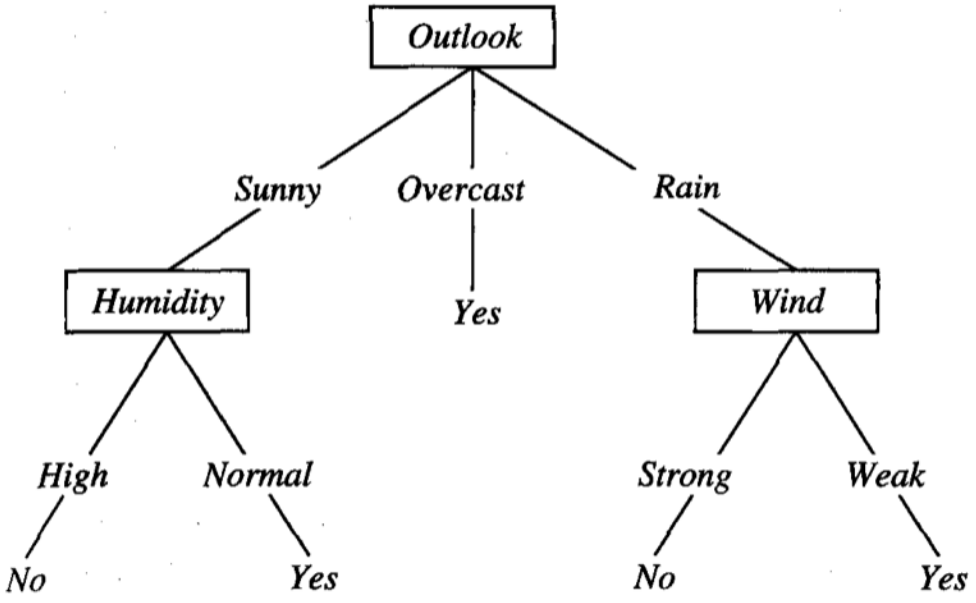
ID3 example

- Among all attributes, outlook has the most information gain.
- We make outlook the root node of the decision tree and its values as the branches.
- Because when outlook=overcast, the labels are all positive, we have a pure leaf node in this tree.



ID3 example

- For each branch of the root node, we repeat the computation of the information gain, and find the best node as the child node.



Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Outline

- Classify Network Attacks –basic approaches
- Two-stage classification
- Dimensionality reduction of network intrusion data
- Visualizing the analyze network intrusion data
  - Identify patterns that potentially cause misclassification

Classify Network Attacks – Case Study

- Goal: build a network attack classifier from scratch using basic ML
- Context of the problem
  - In many real scenarios, relevant labelled data might be difficult to come across, and engineering numerical features from the raw data will take up most of the effort
  - Obtaining good training data is a big challenge when using machine learning for security. Classifiers are only as good as the data used to train them, and reliably labelled data is especially important in supervised machine learning
  - Most organizations don’t have exposure to a large amount and wide variation of attack traffic, and must figure out how to deal with the class imbalance problem

Classify Network Attacks – Case Study

- In the case of classifying network traffic, the task of labelling data often must be carried out by experienced security analysts with investigative and forensic skills, the process is time consuming
- Building high-performing machine learning systems is a process filled with exploration and experimentation
- Next:
  - walk through the process of understanding a dataset and preparing it for processing
  - then present a few algorithms that can be applied to the problem
- Focus: exploration and experimentation process

Exploring the Data

```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,
0.00,0.00,1.00,0.00,0.00,150,25,0.17,0.03,0.17,0.00,0.00,0.00,0.05,
0.00,normal

0,icmp,eco_i,SF,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,21,0.00,0.00,
0.00,0.00,1.00,0.00,1.00,2,60,1.00,0.00,1.00,0.50,0.00,0.00,0.00,
0.00,ipsweep
```

labels

- Dataset: NSL- KDD dataset
  - A benchmark dataset to help researchers compare different intrusion detection methods
  - labeled training data: 38 different types of attacks, e.g. ipsweep, guess_passwd…
  - Features: first 41 values
    - e.g.: duration, protocol_type, …
- These attacks belong to **4** general categories:
  - **dos**: denial of service
  - **r2l**: unauthorized accesses from remote servers
  - **u2r**: privilege escalation attempts
  - **probe**: brute-force probing attacks
- Task: devise a classifier that categorizes each individual sample as one of five classes: benign, dos, r2l, u2r, or probe.

Exploring the Data

- The training dataset contains samples that are labeled with the specific attack, the mapping from attack labels to attack categories is specified in ‘training_attack_types.txt’
    - e.g. ftp_write and guess_passwd attacks correspond to the **r2l** category,
    - smurf and udpstorm correspond to the **dos** category, …
- Preliminary data exploration to find out more about these labels

```
# The directory containing all of the relevant data files
dataset_root = 'datasets/nsl-kdd'

category = defaultdict(list)
category['benign'].append('normal')

with open(dataset_root + 'training_attack_types.txt', 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)
```

Exploring the Data

Here's what we find upon inspecting the contents of category:

category attack labels

```
'benign': ['normal'],
'probe':  ['nmap', 'ipsweep', 'portsweep', 'satan',
           'mscan', 'saint', 'worm'],
'r2l':    ['ftp_write', 'guess_passwd', 'snmpguess',
           'imap', 'spy', 'warezclient', 'warezmaster',
           'multihop', 'phf', 'imap', 'named', 'sendmail',
           'xlock', 'xsnoop', 'worm'],
'u2r':    ['ps', 'buffer_overflow', 'perl', 'rootkit',
           'loadmodule', 'xterm', 'sqlattack', 'httptunnel'],
'dos':    ['apache2', 'back', 'mailbomb', 'processtable',
           'snmpgetattack', 'teardrop', 'smurf', 'land',
           'neptune', 'pod', 'udpstorm']
```

However, this data structure is not very convenient for us to perform the mappings from attack labels to attack categories, so let's invert this dictionary

Exploring the Data

- What if we want to map from attack labels to attack categories?

```
attack_mapping = dict((v,k) for k in category for v in category[k])
```

Before:

```
category   attack labels
'benign': ['normal'],
'probe':  ['nmap', 'ipsweep', 'portsweep', 'satan',
           'mscan', 'saint', 'worm'],
'r2l':    ['ftp_write', 'guess_passwd', 'snmpguess',
           'imap', 'spy', 'warezclient', 'warezmaster',
           'multihop', 'phf', 'imap', 'named', 'sendmail',
           'xlock', 'xsnoop', 'worm'],
'u2r':    ['ps', 'buffer_overflow', 'perl', 'rootkit',
           'loadmodule', 'xterm', 'sqlattack', 'httptunnel'],
'dos':    ['apache2', 'back', 'mailbomb', 'processtable',
           'snmpgetattack', 'teardrop', 'smurf', 'land',
           'neptune', 'pod', 'udpstorm']
```

After the mapping:

```
{
    'apache2': 'dos',
    'back': 'dos',
    'guess_passwd': 'r2l',
    'httptunnel': 'u2r',
    'imap': 'r2l',
    ...
}
```

Exploring the Data

- It is always important to consider the class distribution within the data
- In some scenarios, it can be difficult to accurately predict the class distribution of real-life data, but it is useful to have a general idea of what is expected
- For instance, when designing a network attack classifier for deployment on a network that does not contain any database servers, we might expect to see very little sqlattack traffic
- We can consume the data to get the distribution of the attack_type and attack_category

Exploring the Data

- Read the data and plot the attack_type and attack_category distributions
  - Remind that a Pandas DataFrame is a 2-dimensional data structure, a table with rows and columns
  - read the training data

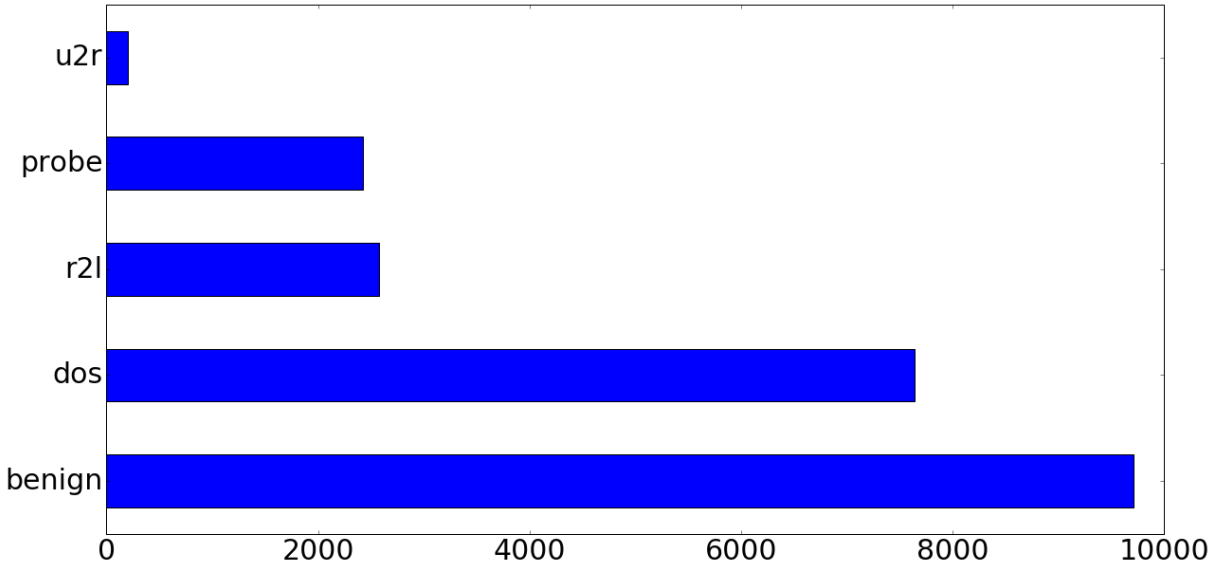
```
# Read training data
train_df = pd.read_csv(train_file, names=header_names)
```

  - plot the class distribution

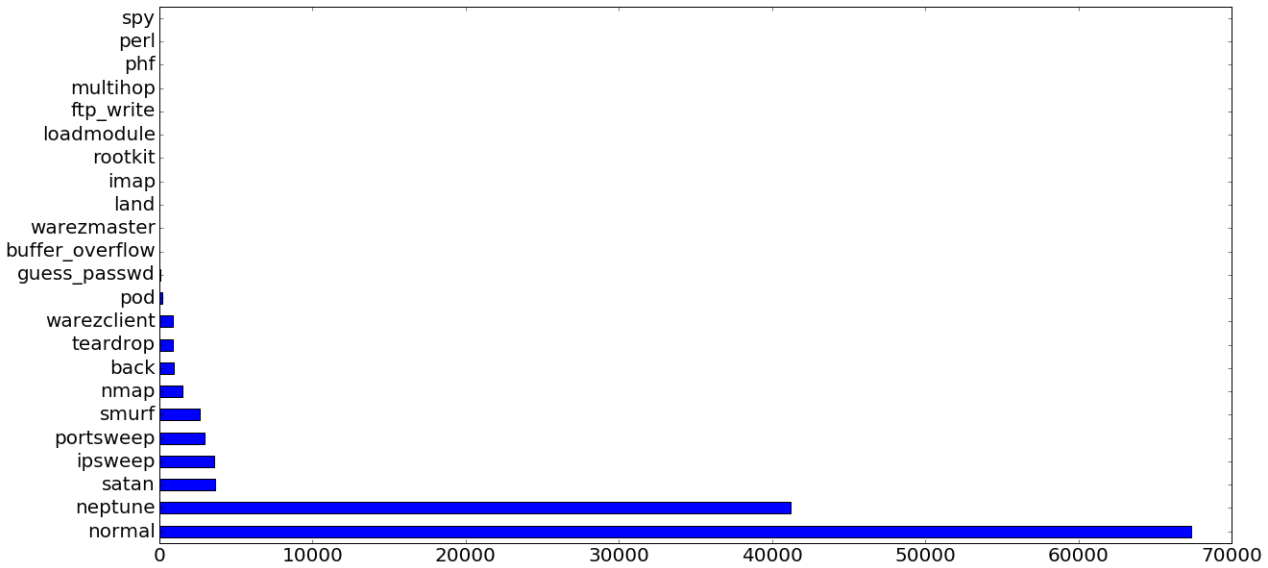
```
train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()
train_attack_types.plot(kind='barh')
train_attack_cats.plot(kind='barh')
```

Exploring the Data

- Observation: classes are imbalanced
  - if ignore the imbalance, model may learn a lot more about the benign/dos compare to the u2r and r2l
  - undesirable bias in the classifier



class distribution (5-category breakdown)



class distribution (22-category breakdown)

Data Preparation

- Split the training and test DataFrames into data and labels

```
train_Y = train_df['attack_category']
train_x_raw = train_df.drop(['attack_category','attack_type'], axis=1)

test_Y = test_df['attack_category']
test_x_raw = test_df.drop(['attack_category','attack_type'], axis=1)
```

- Encode the categorical (symbolic) variables
  - We will generate a list of categorical variable names and a list of continuous variable names
  - give a structure containing two keys continuous and symbolic, each mapping to a list of feature names

```
continuous: [ duration, src_bytes, dst_bytes, wrong_fragment, ... ]
symbolic:   [ protocol_type, service, flag, land, logged_in, ... ]
```

Data Preparation

- Further split the symbolic variables into nominal (categorical) and binary types
- Then, we use the pandas.get_dummies() function to convert the nominal variables into dummy variables
  - applies one-hot encoding to categorical variables (e.g. flag)
  - => create multiple binary variables for each possible value of flag that appears in the dataset
  - For each sample, only one of these variables can have the value 1
  - e.g. if a sample has value flag=S2, its dummy representation (for flag) will be:

```
# flag_S0, flag_S1, flag_S2, flag_S3, flag_SF, flag_SH
[    0,       0,       1,       0,       0,       0    ]
```

Data Preparation

- The distribution of the training set features:

	duration	src_bytes	…	hot	num_failed_logins	num_compromised
mean	287.14465	4.556674e+04		0.204409	0.001222	0.279250
std	2604.51531	5.870331e+06		2.149968	0.045239	23.942042
min	0.00000	0.000000e+00		0.000000	0.000000	0.000000
…	…	…		…	…	…
max	42908.00000	1.379964e+09		77.000000	5.000000	7479.000000

- We notice something that is potentially worrying
  - The distributions of each feature vary widely
  - e.g. src_bytes vs num_failed_logins, affect the results as src_bytes feature would dominate
- Solution: standardization/normalization

Data Preparation

- Feature scaling: Standardization and Normalization
- Standardization
  - rescales a data series to have a mean of 0 and a standard deviation of 1
  - useful whenever features in the data vary widely in their distribution characteristics
  - Standardize feature by removing the mean and scaling to unit variance.

$$X' = \frac{X - \mu}{\sigma}, where\ \mu = mean(X), \sigma = stddev(X)$$

  - Note that in this case, the values are not restricted to a particular range
  - The scikit-learn library includes the sklearn.preprocessing.StandardScaler class that provides this functionality

Data Preparation

- For example, let's standardize the duration feature

- Apply standard scaling on it:

```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler().fit(durations)
standard_scaled_durations = standard_scaler.transform(durations)
pd.Series(standard_scaled_durations.flatten()).describe()
```

- We see that the feature has been scaled to have
  - a mean of 0
  - standard deviation of 1

```
> # Output:
count    1.259730e+05
mean     2.549477e-17
std      1.000004e+00
min     -1.102492e-01
max      1.636428e+01
```

Data Preparation

- Normalization
  - rescales the data to a small, specified range (min-max scaling)

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}(max - min) + min$$

X_{max} and X_{min}: maximum and minimum value of the feature, resp.
max and min : maximum and minimum value of the new feature range, e.g. (0,1) by default

- E.g. If new feature_range = (0, 1)
  - when the value of X is the feature's min_value,
    - numerator = 0 => X' is 0
  - when the value of X is the feature's max_value,
    - numerator = denominator => X' is 1
  - If the value of X is in between => the value of X' is between 0 and 1

Data Preparation

- The sklearn.preprocessing.MinMaxScaler class scales a feature from its original range to [min, max].
- This is how MinMaxScaler transforms the duration feature between [0, 1]

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(durations)
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_duration.flatten()).describe()

> # Output:
count    125973.000000
mean          0.006692
std           0.060700
min           0.000000
max           1.000000
```

Data Preparation

- Apply consistent transformations to both training & test sets
  - i.e., using the same mean, std, etc. to scale the data
- Scikit-learn provides a convenient way to do proper scaling
  - after creating the Scaler and fitting it to the training data
  - we can simply reuse the same Scaler to transform the test data

We finish the data pre-processing phase by standardizing the training and test data:

And…
the data is prepared and ready to go for classifying attacks!

```
# Fit StandardScaler to the training data
standard_scaler = StandardScaler().fit(train_x[continuous_features])

# Standardize training data
train_x[continuous_features] = \
    standard_scaler.transform(train_x[continuous_features])

# Standardize test data with scaler fitted to training data
test_x[continuous_features] = \
    standard_scaler.transform(test_x[continuous_features])
```

Multiclass Classification

- Remind that we have a five-class classification problem
  - each sample belongs to one of the following classes: benign, u2r, r2l, dos, probe
- Essentially, a multiclass classification problem can be split into multiple binary classification problems
  - Binary logistic regression
  - Basic SVM.
- Some techniques for achieving multiclass classification
  - one-versus-all: train one classifier per class
  - one-versus-one: train one classifier per class pair

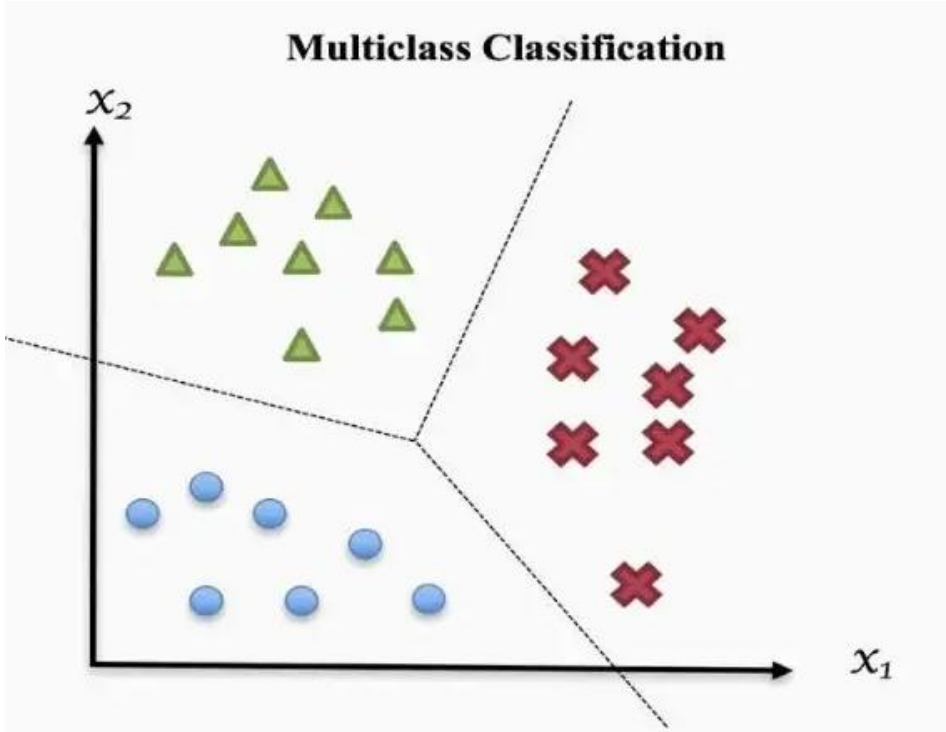
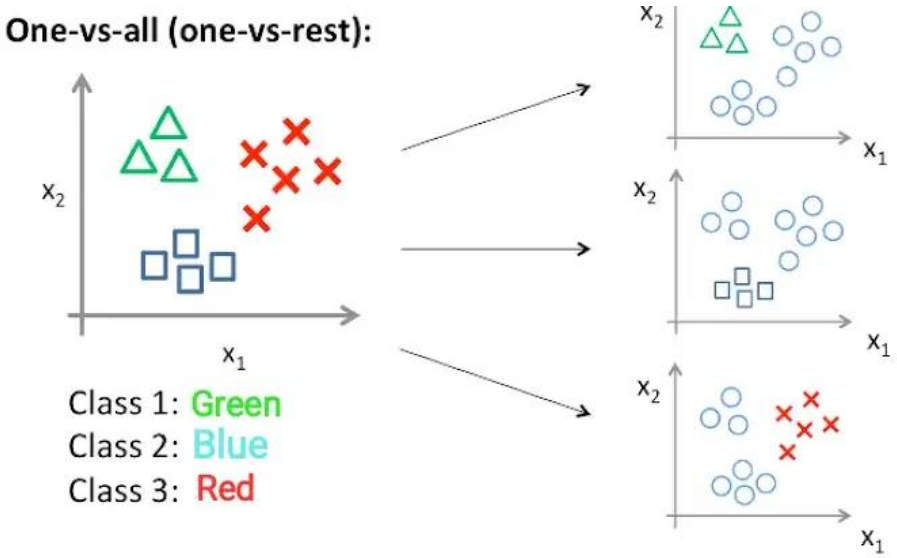


image credit: towardsdatascience

Multiclass Classification

- One-vs-all (one-vs-rest)
  - For the N-class instances dataset, a one-vs-all solution consists of N separate binary classifiers—one binary classifier for each class
  - During training, the model runs through a sequence of binary classifiers, training each to answer a separate classification question
  - At prediction time, the class with highest confidence is selected



Classifier 1:- [Green] vs [Red, Blue]
Classifier 2:- [Blue] vs [Green, Red]
Classifier 3:- [Red] vs [Blue, Green]

Multiclass Classification

- one-vs-one
  - train one binary classifier per class pair
  - for the N-class instances dataset, generate N* (N-1)/2 binary classifier models
- Example: red, blue, black
- **3** binary classifier:
  - Classifier 1: red vs. black
  - Classifier 2: blue vs. black
  - Classifier 3: red vs. blue
- At prediction time, the class which received the most votes is selected

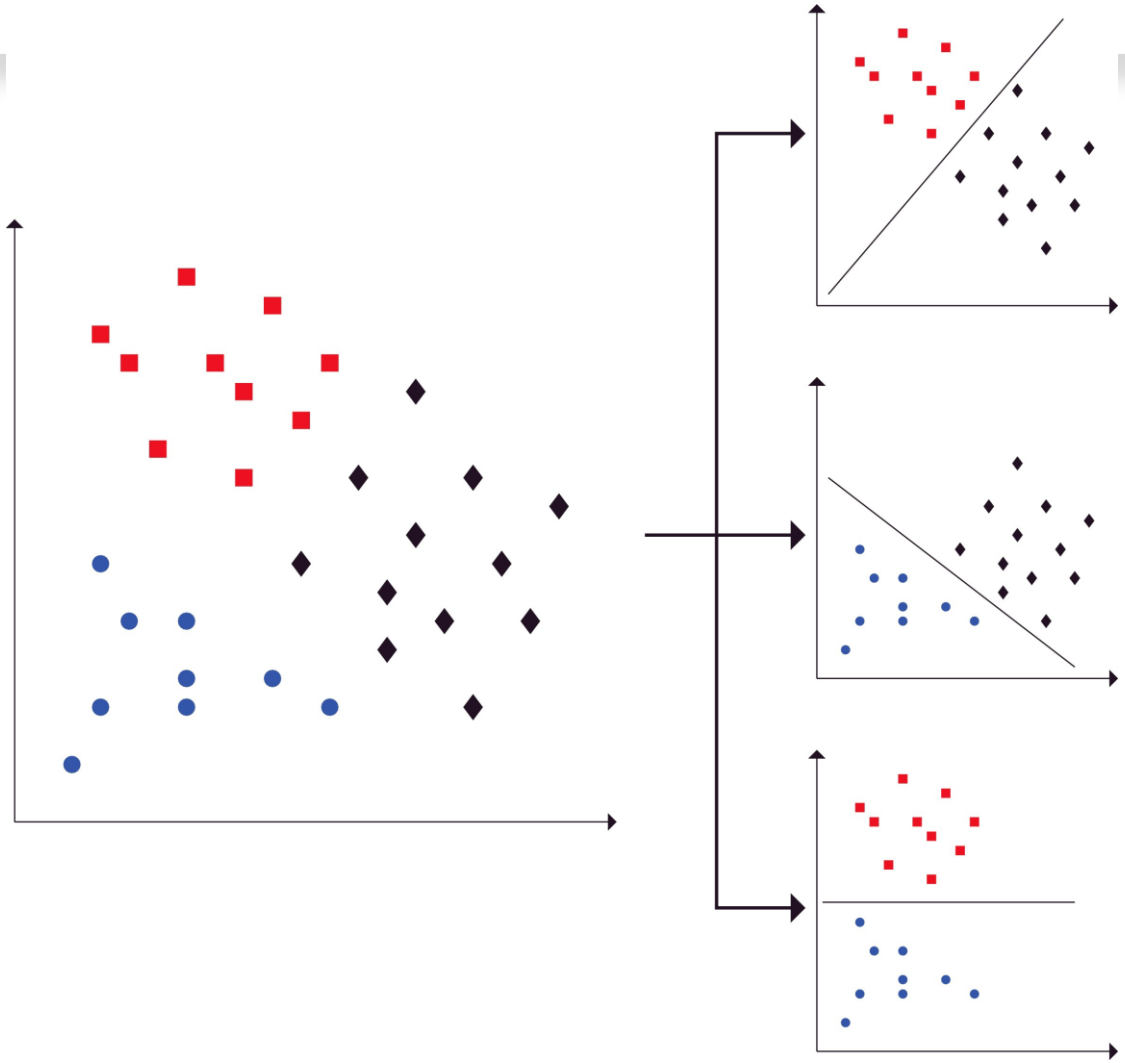


Image Credit: Empowering one-vs-one decomposition with ensemble learning for multi-class imbalanced data

Classification

- Choosing a suitable classifier for the task is a challenging part
  - Many classifiers could be a good choice, the optimal one might not be obvious
  - Developing machine learning solutions is an iterative process
  - Knowledge and experience with different algorithms can help to develop better intuition to cut down the iteration process
    - but it is rare, even for experienced practitioners, to immediately select the optimal solution for arbitrary machine learning tasks
  - Spending time and effort to iterate on a rough initial solution will bring about useful insights

Supervised Learning

- What we have in the training data: ~126,000 labeled training samples
- Supervised training methods is a good place to begin, e.g. decision trees
- Confusion matrix resulting from the decision tree classifier
  - Diagonal: # correctly classified samples
  - Size of the test set: total # in the matrix (22,544)
  - row: true class
  - column: predicted class
  - e.g. a_{04} = 1: # samples that are of class 0 that were classified as class 4.
  - (class index: `benign` = 0, `dos` = 1, `probe` = 2, `r2l` = 3, and `u2r` = 4)

```
> # Confusion matrix:
[[9357   59  291    3    1]
 [1467 6071   98    0    0]
 [ 696  214 1511    0    0]
 [2325    4   16  219   12]
 [ 176    0    2    7   15]]
```

```
> # Error:
0.238245209368
```

Analysing the result

benign = 0, dos = 1, probe = 2, r2l = 3, and u2r = 4

```
> # Confusion matrix:
[[9357   59  291    3    1]
 [1467 6071   98    0    0]
 [ 696  214 1511    0    0]
 [2325    4   16  219   12]
 [ 176    0    2    7   15]]
```

Similar to the training set, the test data distribution is not balanced across categories

```
# test data distribution
benign    0.430758
dos       0.338715
r2l       0.114265
probe     0.107390
u2r       0.008872
```

- 43% of the test data belongs to the benign class
- 0.8% of the data belongs to the u2r class

- Although only 3.6% of benign test samples that were wrongly classified: (59+291+3+1)/ sum(row_1)
- 62% of test samples were classified as benign: sum(col_1) / sum(matrix)
- Look at r2l and u2r rows:
  - more samples were classified as benign than all other samples in those classes

Why could it be? Could we have trained a classifier that is just more likely to classify samples into the benign class?

Observation

- Going back to see the training data distribution: 0.7% r2l, 0.04% u2r
- Remind the class imbalance in training data, it seems unlikely there would have been enough information for the trained model to learn enough knowledge from the u2r and r2l classes to correctly identify them
- To see whether problems might be caused by the choice of the decision tree classifier, we will try other classifiers, e.g. KNN and SVM
  - KNN (K-Nearest Neighbors): supervised learning classifier
  - Note: we will not dive into the details of KNN; instead, we will use it as a black box for comparison purpose

Supervised learning

- Results for k-nearest neighbors classifier

```
> # Confusion matrix:
[[9455   57  193    2    4]
 [1675 5894   67    0    0]
 [ 668  156 1597    0    0]
 [2346    2   37  151   40]
 [ 177    0    4    6   13]]

> # Error:
0.240951029099
```

- Results for linear support vector classifier

```
> # Confusion matrix:
[[9006  294  405    3    3]
 [1966 5660   10    0    0]
 [ 714  122 1497   88    0]
 [2464    2    1  109    0]
 [ 175    1    0    8   16]]

> # Error:
0.278167139815
```

Resulting confusion matrices and error rates show very similar characteristics to the decision tree classifier results

- At this point, it should be clear that continuing to experiment with classification algorithms might not be productive
- What we can try to do is to remedy the class imbalance problem in the training data in hopes that resulting models will not be overly skewed to the dominant classes

Class Imbalance

- Extremely imbalanced datasets for NIDS cause problems for machine learning.
  - A dataset in which the classification categories are not approximately equally represented is considered to be imbalanced.
  - Certain attacks occur more often than others.
  - Machine learning intrusion detection approaches may perform well at the task of detecting frequent attacks, but are much less effective when it comes to the detection of infrequent attacks, due to the lack of sufficient training data for infrequent attacks.

Class Imbalance

- In general, there are two types of remedies
  - Over-sampling: generating synthetic data points for minority classes
    - a naive way: add random samples to the dataset
    - This would likely pollute the dataset and incorrectly influence the distribution
    - one possible solution: SMOTE (Synthetic Minority Oversampling Technique)
      - algorithms that attempt to generate synthetic data in a way that does not contaminate the original characteristics of the minority class
  - Under-sampling: sampling the overrepresented classes to reduce the number of samples
    - may cause information loss in certain datasets
    - should prioritize the removal of samples that are very similar to other samples

- *Imbalanced-learn* offers a wide range of data resampling techniques for different problems

A two-stage approach

- A two-stage approach for imbalanced intrusion detection datasets
  - The underlying notion behind this approach is to alter the dataset into majority and minority malicious activity classes, and to apply classification algorithm on them separately to produce different models for detection.
  - The purpose of this is to improve the overall detection rate of minority classes and to reduce the error rate.
  - Different classifiers can be used for each of the two-stages.
    - Decision tree, logistic regression, deep neural network, etc.
    - Random forest is used in experiments.

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

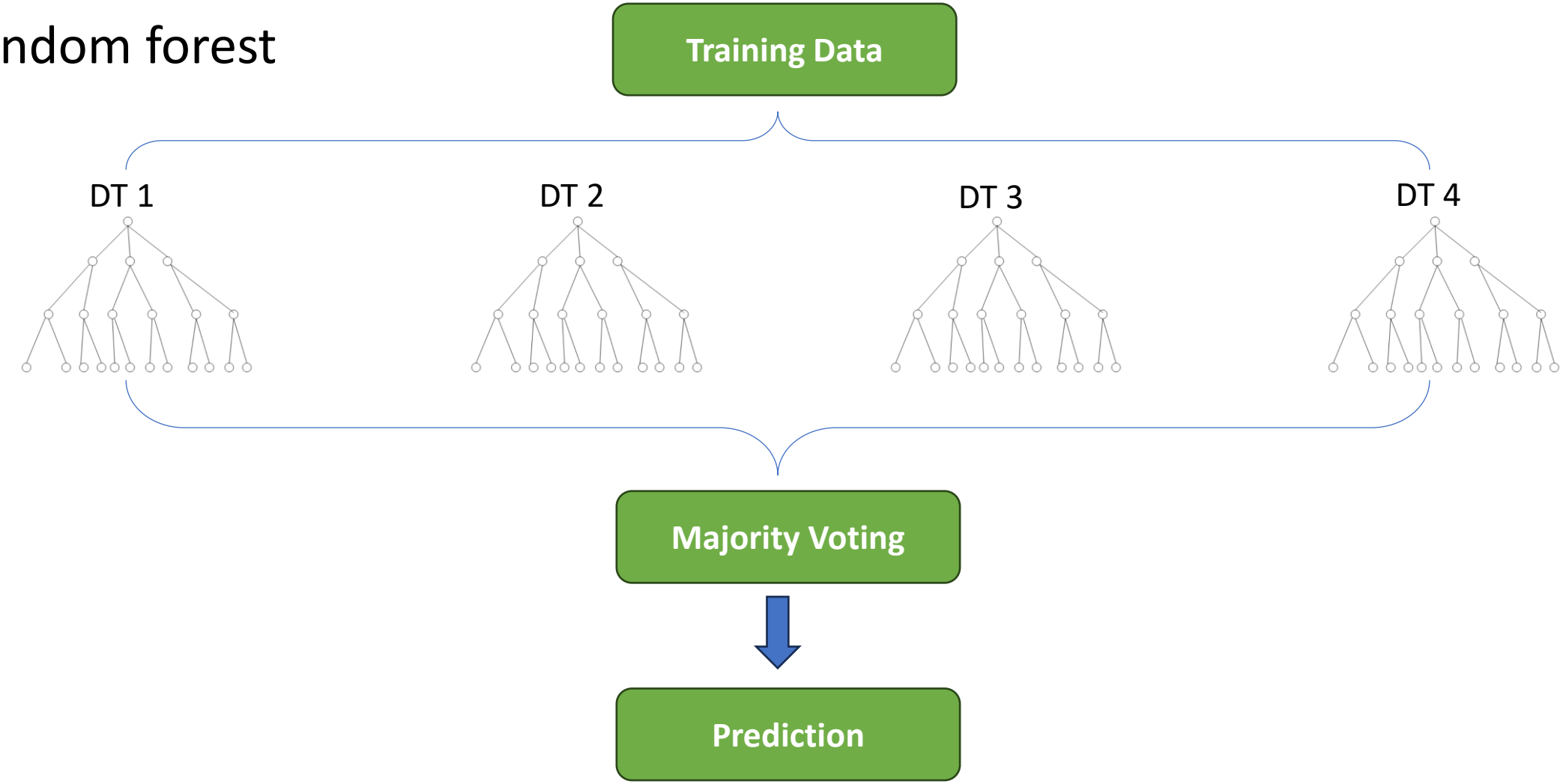
A two-stage approach

- Random forest
  - It works by creating a number of decision trees during the training phase.
  - Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition.
  - This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.
  - In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks).
  - This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results.

https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

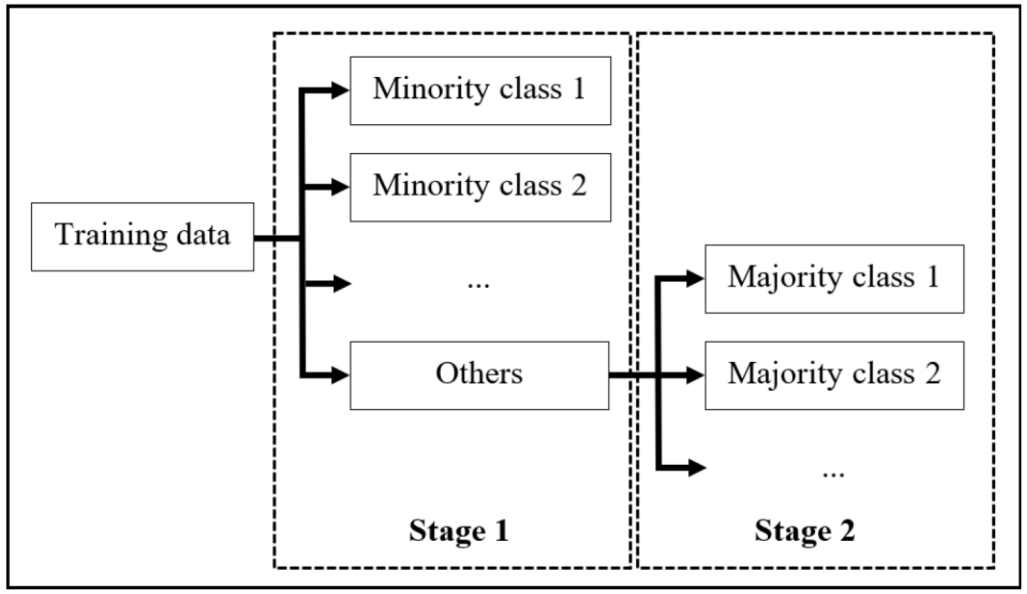
A two-stage approach

Random forest



A two-stage approach

- Overview of the two-stage approach
  - Divide the dataset into majority and minority classes for the two stages, depicted in the figure.
  - During the **first stage**, majority classes, which occupy a major proportion of a training set, are classified as “others"
    - A classier is trained to identify the minority classes.
  - In the **second stage**, the minority classes are removed and another model is trained to identify the majority classes.
  - This results in two different intrusion detection models for identifying the minority and majority classes respectively.
    - It is possible to use different classifiers for each of the stages.

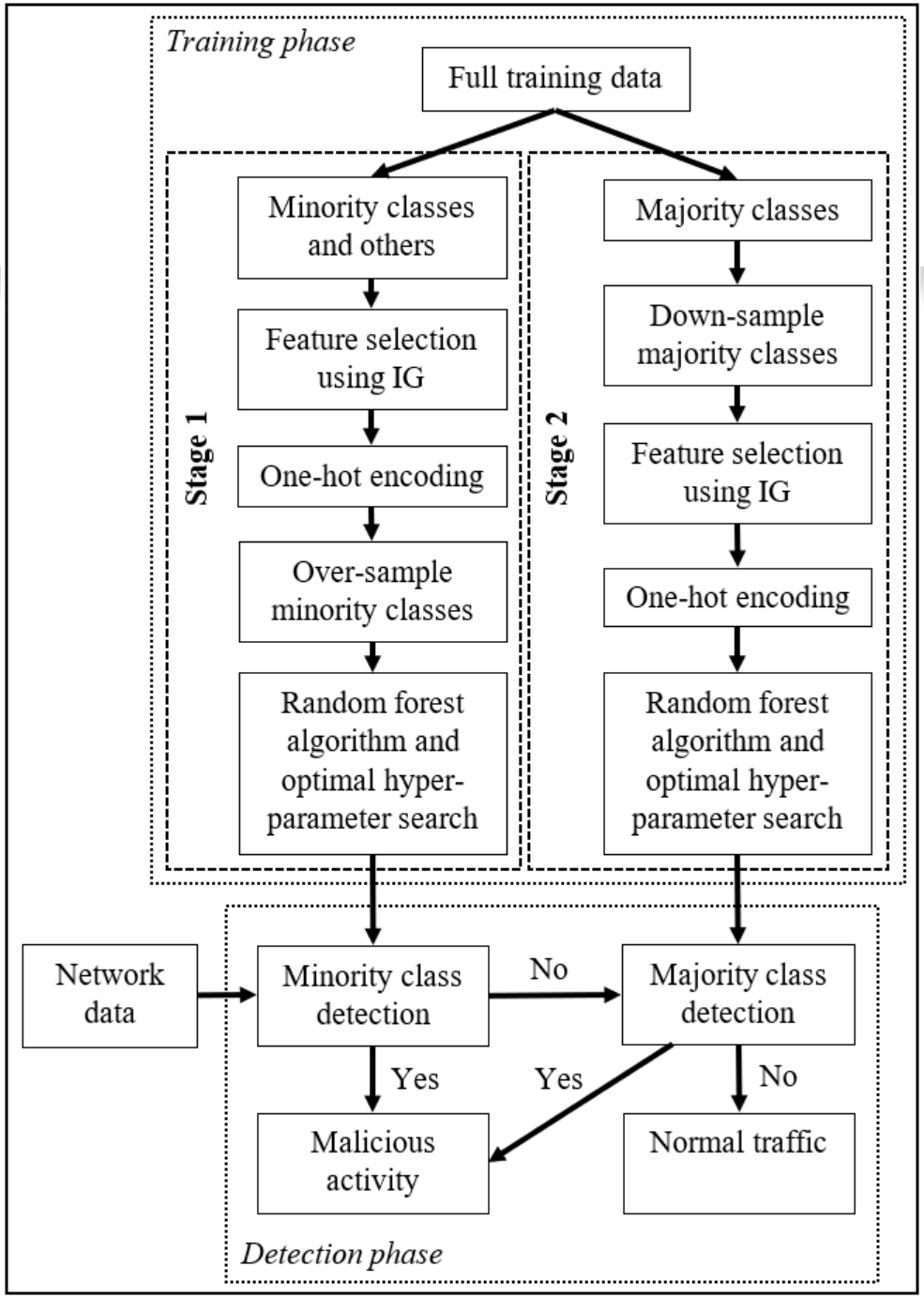


Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

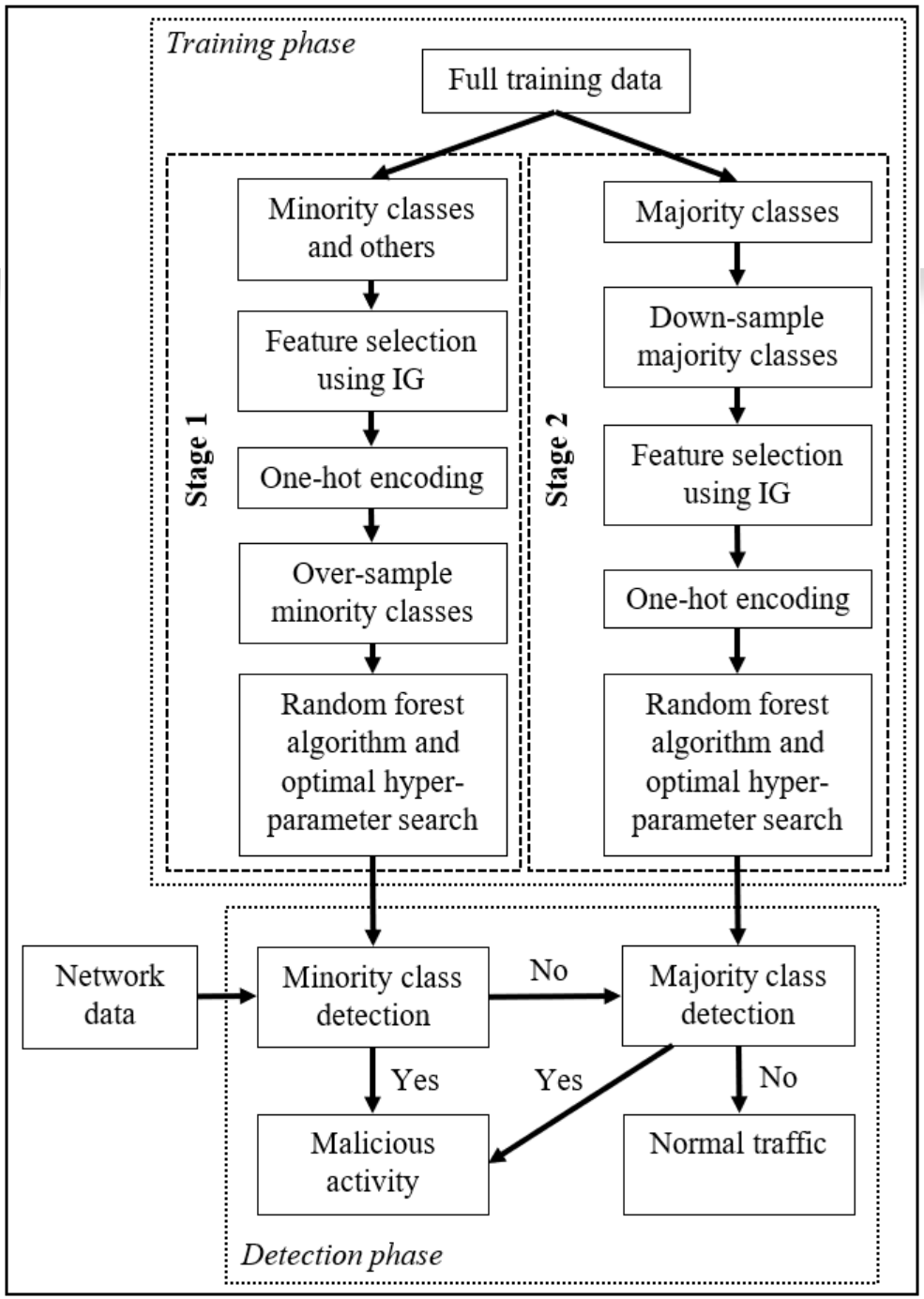
- **Stage 1 training phase**
  - After extracting the minority classes, feature selection is performed using the information gain method.
  - all categorical features are then converted into binary features using one-hot encoding to produce a set of numeric values.
  - The SMOTE method is then used to over-sample the minority classes.
  - The resulting set is then used for the training.

- **Stage 2 training phase**
  - Only the majority classes are used in the training set.
    - the minority classes are removed, since this was handled in stage 1.
  - Down-sampling is performed to balance the majority classes using a random selection method.
    - Network traffic within the majority classes in itself is still unbalanced, hence, down-sampling is performed to balance categories.
  - Information gain is again used for feature selection, followed by one-hot encoding.
  - The resulting set is then used for the training.



A two-stage approach

- Detection Phase
  - Network data is input into the system.
  - The model for identifying minority classes is used first to determine whether an activity is malicious.
  - If it is not identified as one of the minority classes, the second model is then applied to identify whether the activity is a majority intrusion.
  - Otherwise, it is determined to be normal network traffic.



A two-stage approach

- UNSW-NB15 dataset
  - NSL_KDD is a benchmark datasets for intrusion detection
    - A classic dataset for evaluating models.
    - However, data were generated more than a decade ago, and these datasets arguably no longer reflect the current network threat environment.
  - the UNSW-NB15 dataset was proposed as a contemporary dataset
    - It was created as a hybrid of modern normal and contemporary synthesized attack activities of network Traffic.
      - 100 GB of the raw traffic was captured to build this dataset.
    - This dataset has nine types of attacks:
      - namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.
    - 49 features for each record.
      - 41 features in NSL_KDD

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

- UNSW-NB15 dataset
  - The table shows the characteristics of the UNSW-NB15 dataset
    - where the training and testing sets have been divided into an approximately 60% to 40% ratio.
    - There were no redundant records among the training and testing set.
  - Different categories are unequally represented in the dataset.
    - For example, the analysis, backdoor, shellcode and worms categories are minority classes that collectively only make up < 3% of the sets.
    - This imbalance creates problems for classifiers and results in poor detection performance of these minority classes.
  - The majority classes like normal, exploits, generic, etc. occur frequently and there is an abundance of such training samples.

Category	Training Set	Testing Set
Normal	56,000	37,000
Analysis	2,000	677
Backdoor	1,746	583
DoS	12,264	4,089
Exploits	33,393	11,132
Fuzzers	18,184	6,062
Generic	40,000	18,871
Reconnaissance	10,491	3,496
Shellcode	1,133	378
Worms	130	44
Total Records	175,341	82,332

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

Technique	Accuracy (%)	FAR (%)
DT [20]	85.56	15.78
LR [21]	83.15	18.48
NB [16]	82.07	18.56
ANN [21]	81.34	21.13
EM clustering [14]	78.47	23.79
Proposed Approach	85.78	15.64

- Detection results
  - A comparison with the Decision Tree (DT), Logistic Regression (LR), Naïve Bayes (NB), Artificial Neural Network (ANN) and Expectation-Maximization (EM) clustering)
  - The resulting accuracy of the proposed approach is higher than the other techniques, while the False Alarm Rate (FAR) is lower.
    - Overall results are comparable with the results of DT.
      - Random forest is based on decision trees.
      - DT would miss minority classes.
      - Other types of classifiers can potentially be used to further improve results.
      - The two-stage framework sheds light on more advanced methods.

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

- Analysis of misclassification
  - The table provides the rate of normal traffic that was misclassified as malicious activity.
  - Most of the misclassification was to do with fuzzing activity.
    - It can be seen from the table at a large portion of the misclassication are for fuzzers.
  - Fuzzers are attacks where the attacker attempts to discover security loopholes in a program, operating system or network.
    - They are not necessarily dangerous in themselves when compared with other attacks.
    - Fuzzing activity has to do with inputting lots of random data.
    - As such, they do not have a fixed pattern and are more difficult to distinguish from normal network traffic.

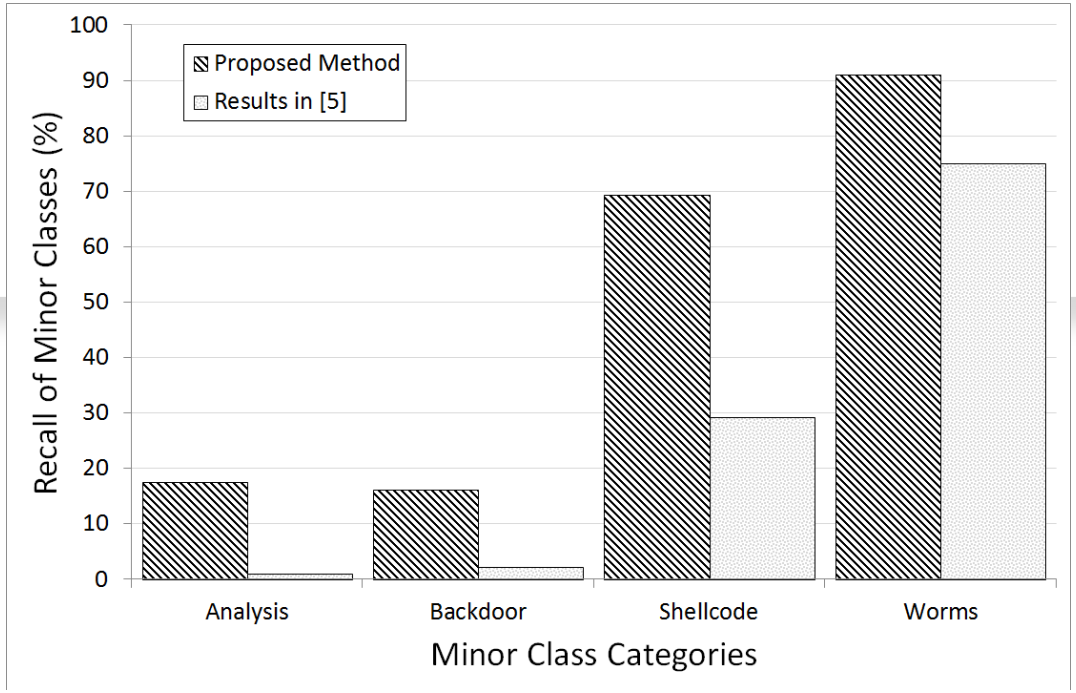
Normal activity misclassified as malicious.

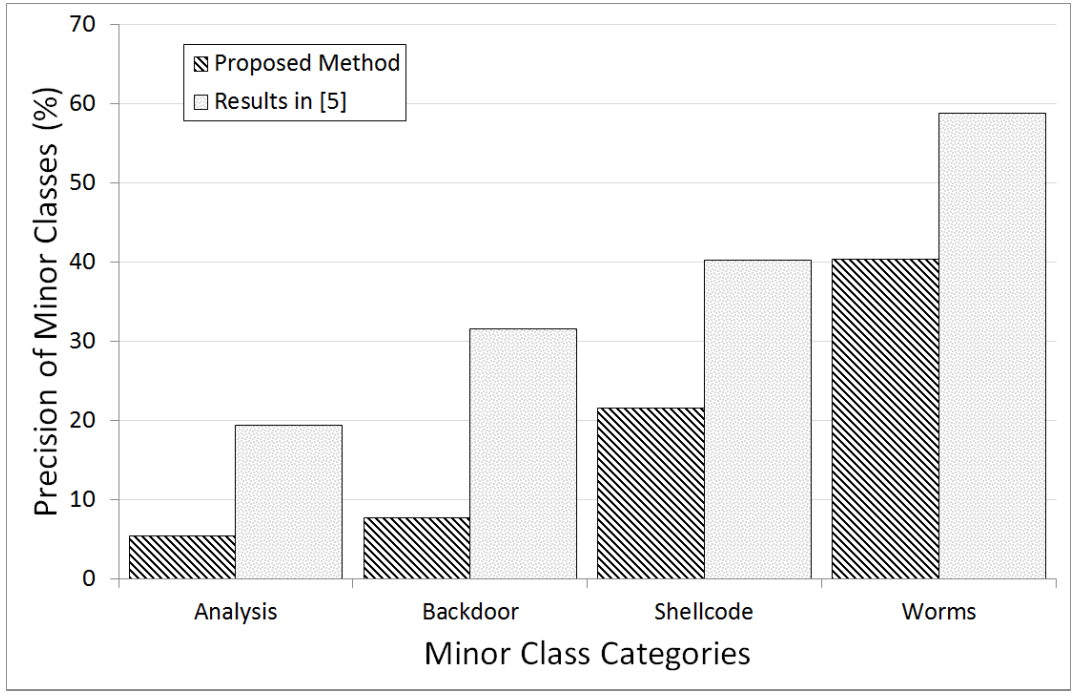
Categories	Misclassification (%)
Analysis	3.4
Backdoor	0.1
DoS	0.4
Exploits	0.9
Fuzzers	23.2
Generic	0.0
Reconnaissance	0.0
Shellcode	1.5
Worms	0.0

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

- Minor classes recall vs precision
- The 2 figures provide a comparison of the recall and precision performances, respectively
  - between this two-stage random forest ("Proposed Method") and single-stage random forest ("Results in [5]").
- The recall results of the this two-stage approach performs better in comparison.
  - However, the precision performance is lower.
- Nevertheless, for minority classes a higher recall rate is more important than precision
  - Because these attacks are potentially more dangerous than other attacks.
  - Therefore, higher recall values prevent these attacks from escaping detection.





Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

3D Visualization of Network Intrusion Detection Data

- Motivation to analyze network intrusion dataset
  - Misclassification is a common problem in machine learning
    - a general lack of insight into why such misclassification occurs impedes the improvement of models.
  - Many studies have presented improved results when comparing with other techniques
    - there is not much emphasis on understanding the reasons for the improved results or lower misclassification rates.
    - They tend to present numeric results in the form of tables or graphs to compare performances between different ML techniques.
    - The underlying reasons for poor performance on certain attack categories are not usually explained and cannot be perceived clearly or intuitively.
  - The improvement of ML models usually relies on a trial-and-error process.
    - Due to the complex nature of ML mechanisms.
  - An intuitive and explainable analysis of datasets can facilitate the understanding of detection results.
    - ML heavily depends on the characteristics of the training and testing datasets.
    - Explain detection results from the point of view of data.

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

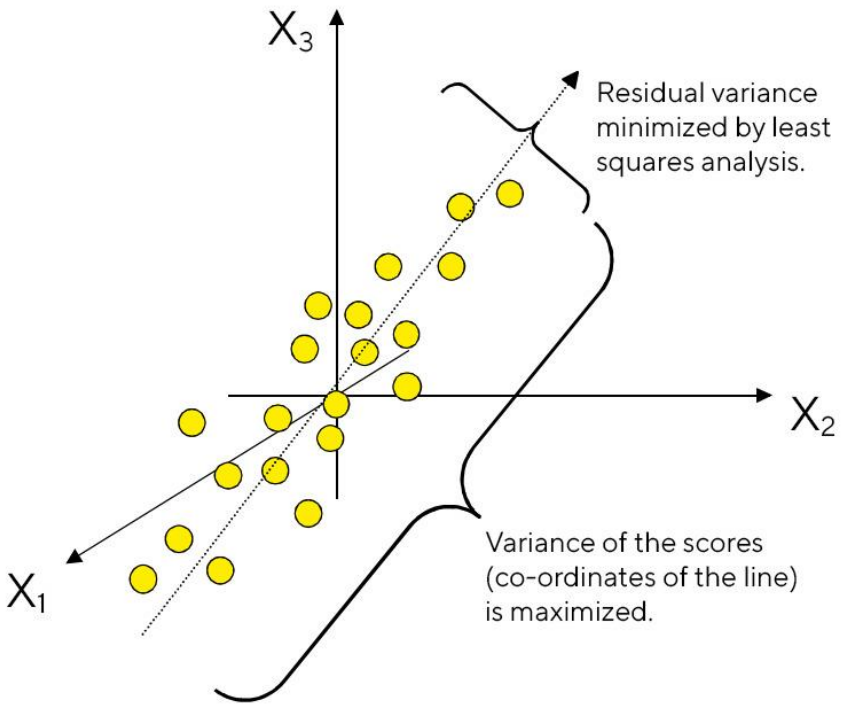
3D Visualization of Network Intrusion Detection Data

- An interactive 3D visualization
  - The aim of the approach is to provide a visual representation of data from NIDS datasets
  - Portrays the geometric relationship between diverse network traffic data records
  - Create a way of examining the likely causes of ML misclassification from a visual perspective.
- The visualization system allows users to interactively navigate through the NIDS data
  - In real-time
  - Visually examine ML classifier decision spaces
  - Observe boundaries where misclassification occurs.

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Principal Component Analysis (PCA)
  - PCA finds lines, planes and hyper-planes in the K-dimensional space that approximate the data as well as possible in the least squares sense.
  - A line or plane that is the least squares approximation of a set of data points makes the variance of the coordinates on the line or plane as large as possible.
  - PCA can reduce high dimensionality of NIDS data.
    - To 2D or 3D for visualization.



PCA creates a visualization of data that minimizes residual variance in the least squares sense and maximizes the variance of the projection coordinates.

https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186#:~:text=Principal%20component%20analysis%2C%20or%20PCA,more%20easily%20visualized%20and%20analyzed.

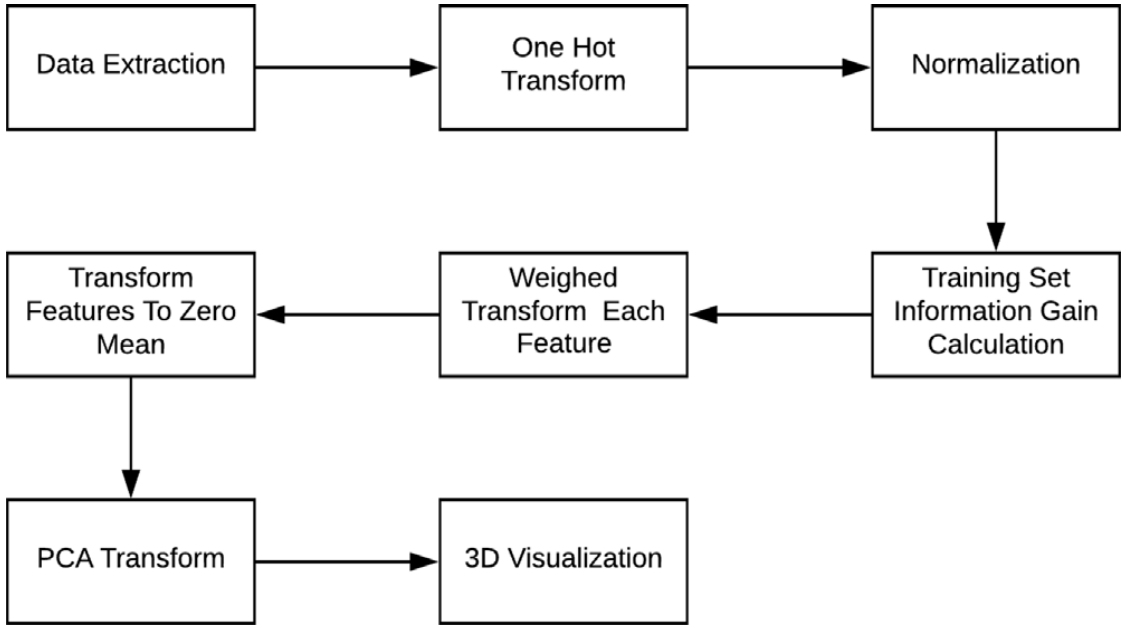
3D Visualization of Network Intrusion Detection Data

- PCA vs t-SNE
  - t-distributed stochastic neighbor embedding (t-SNE) is also widely used for visualization.
  - The adopted dimensionality reduction technique must fulfill the following requirements:
    - First, it should preserve the geometric relationship between the network traffic records, since this relationship is important for ML models, like the SVM technique, in producing the ML decision spaces for the proposed approach,.
    - Second, it should learn a parametric mapping so that new data can be mapped into the low dimensional space.
      - Otherwise, it cannot handle new data.
  - Although t-SNE usually provides better data visualization results than PCA. it does not satisfy the requirements mentioned above.
    - t-SNE does not retain geometric relationship between data,
    - t-SNE Does not learn a parametric mapping and the transformation to low dimension space.
  - In contrast, the PCA technique fulfills all these requirements.

https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186#:~:text=Principal%20component%20analysis%2C%20or%20PCA,more%20easily%20visualized%20and%20analyzed.

3D Visualization of Network Intrusion Detection Data

- Step 1/8
  - All the data for the minor categories are extracted from the training set.
    - Minor categories like worm attacks (in the UNSW-NB15 dataset), and U2R and R2L attacks (in the NSL_KDD dataset), only occupy a small portion of the dataset.
  - Major categories data are randomly extracted until a certain predefined amount, e.g., 30% of the data.
    - The random sampling of the major categories does not diminish the quality of visual information,
    - It is useful to reduce the visual clutter and the amount of required computation on the visualization system.
  - Extraction is not performed on the testing set
    - All the test data is used.

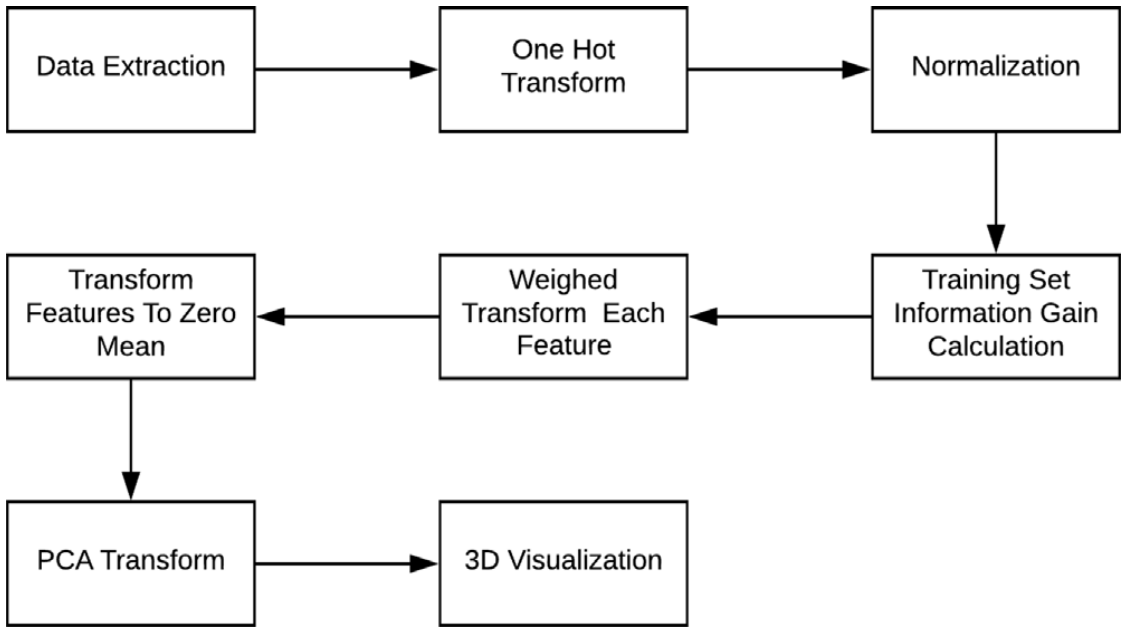


Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Step 2/8
  - Apply one-hot transformation to categorial features.

- Step 3/8
  - Normalize the range of the features.
    - the range of values for the features can differ significantly. Some values may range between zero to less than a hundred, while others may range from 0 to several millions.
    - Linear normalization is applied to each feature in both the training and testing sets to normalize the values to the minimum and maximum range of the training set.

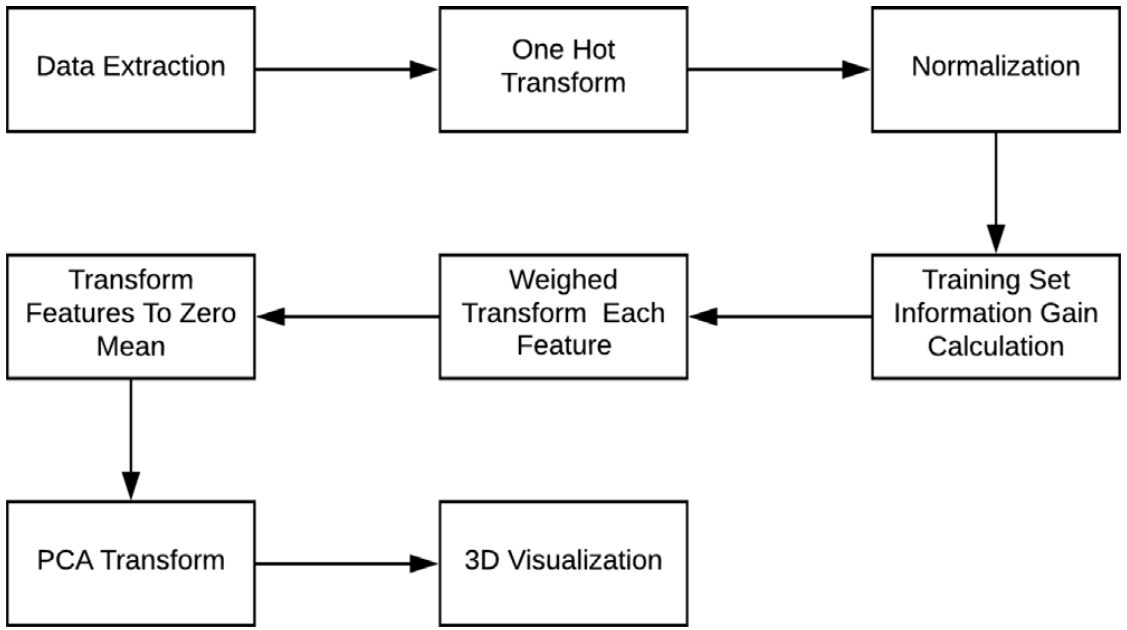


Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Step 4/8
  - Calculate the information gain (IG) values for each feature.

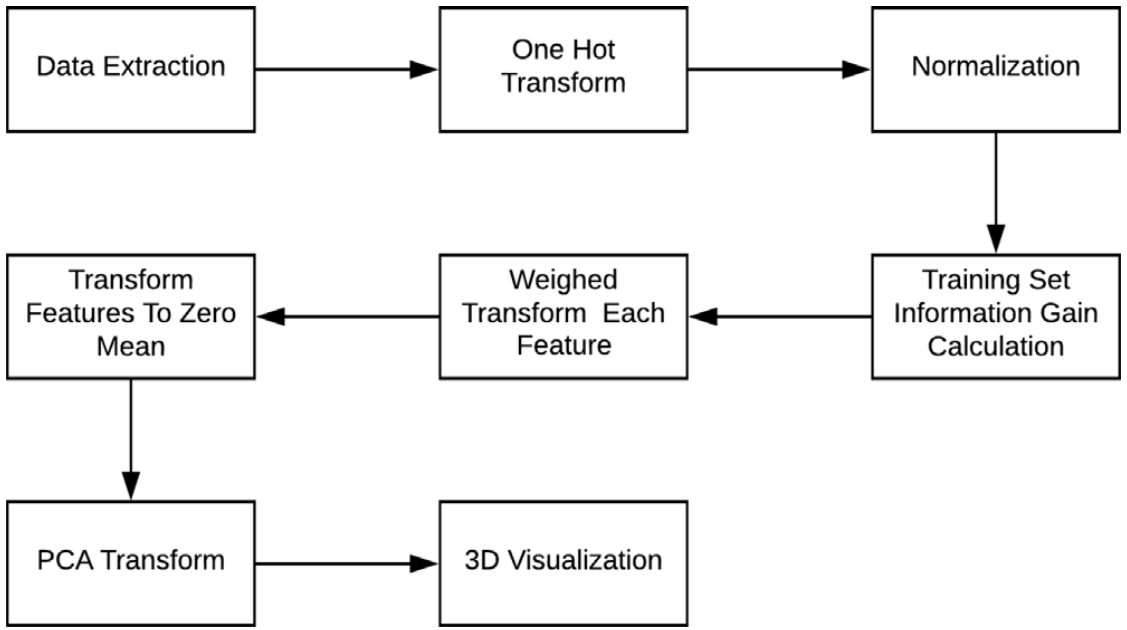
- Step 5/8
  - IG values are normalized between 0 and 1, and weighted transform is applied by multiplying each feature with the corresponding information gain value.
    - As a result, the variances of important features either remain or are decreased slightly, whereas the variances of unimportant features are significantly reduced.



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Step 6/8
  - All features are transformed to zero mean in the training set.
    - The testing set is transformed based on mean values of the corresponding features from the training set.
- Step 7/8
  - Apply PCA.
- Step 8/8
  - Visualize the first 3 components (Unity Game Engine).



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

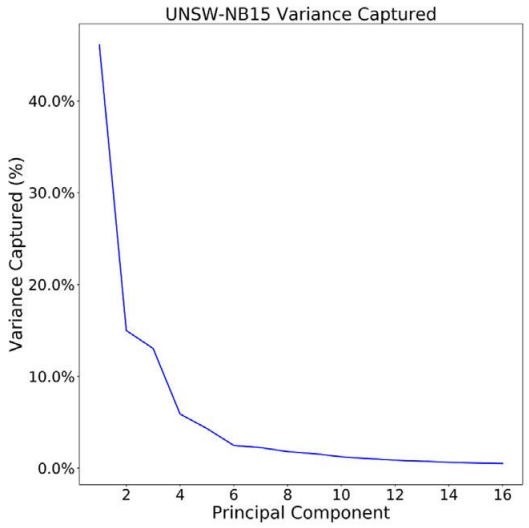
3D Visualization of Network Intrusion Detection Data

- 3D or 2D visualization?

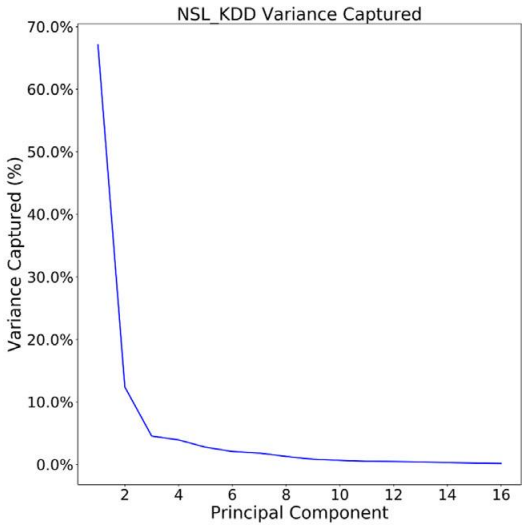
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- 3D or 2D visualization?
  - PCA replaces original variables with new variables, called **principal components**, which are orthogonal.
  - Principal components have variances in a descending order.
    - The amount of variance by each of the selected components is referred to as “variance explained”.
    - Indicate how well data can be described in low dimensional space.
  - The third component captures 13.0% and 4.5% variances for UNSW-NB15 and NSL_KDD datasets respectively
    - 3D visualization is better than 2D visualization
    - Three components obviously capture more variance in both datasets than only two components.



(a)



(b)

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Visually explore NIDS datasets (SVM was the classifier):
  - KDD_No_Boundaries_All_Data.mp4
  - KDD_Binary_Overall.mp4
  - KDD_Multi_Overall.mp4
  - UNSW_No_Boundaries_All_Data.mp4
  - UNSW_Binary_overall.mp4
  - UNSW_Multi_overall.mp4

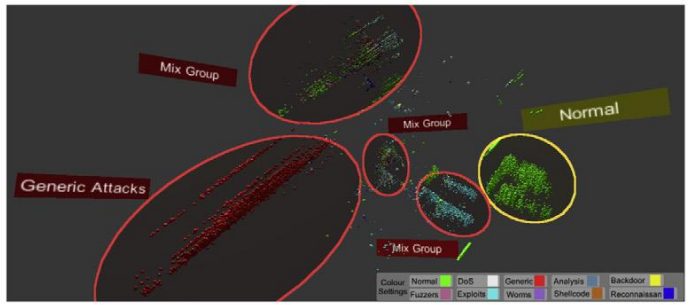
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- Visually explore **UNSW-NB15**
  - **(a)** shows a visual depiction of a portion of the training set from two different camera viewpoints, where the different network traffic categories have been color coded to visually distinguish them from one another.
    - It can be seen visually that traffic from the same category are typically clustered together.
  - **(b)** shows the visual representation of part of the testing set from two different camera viewpoints
    - Clearly, characteristics of both training and testing sets are similar.
  - training a ML model using the training set it is highly likely that the model will be able to detect attacks in the testing set.
  - **(c)** provides an empirical grouping of data from the training set.
    - The clusters circled in yellow depict sections that mainly only contain normal network traffic, whereas other clusters are circled in red.
    - The traffic for generic attacks are well clustered together.
    - However, the traffic in the mixed clusters is not so well defined as they contain both attacks and normal network traffic.

(a)

(b)

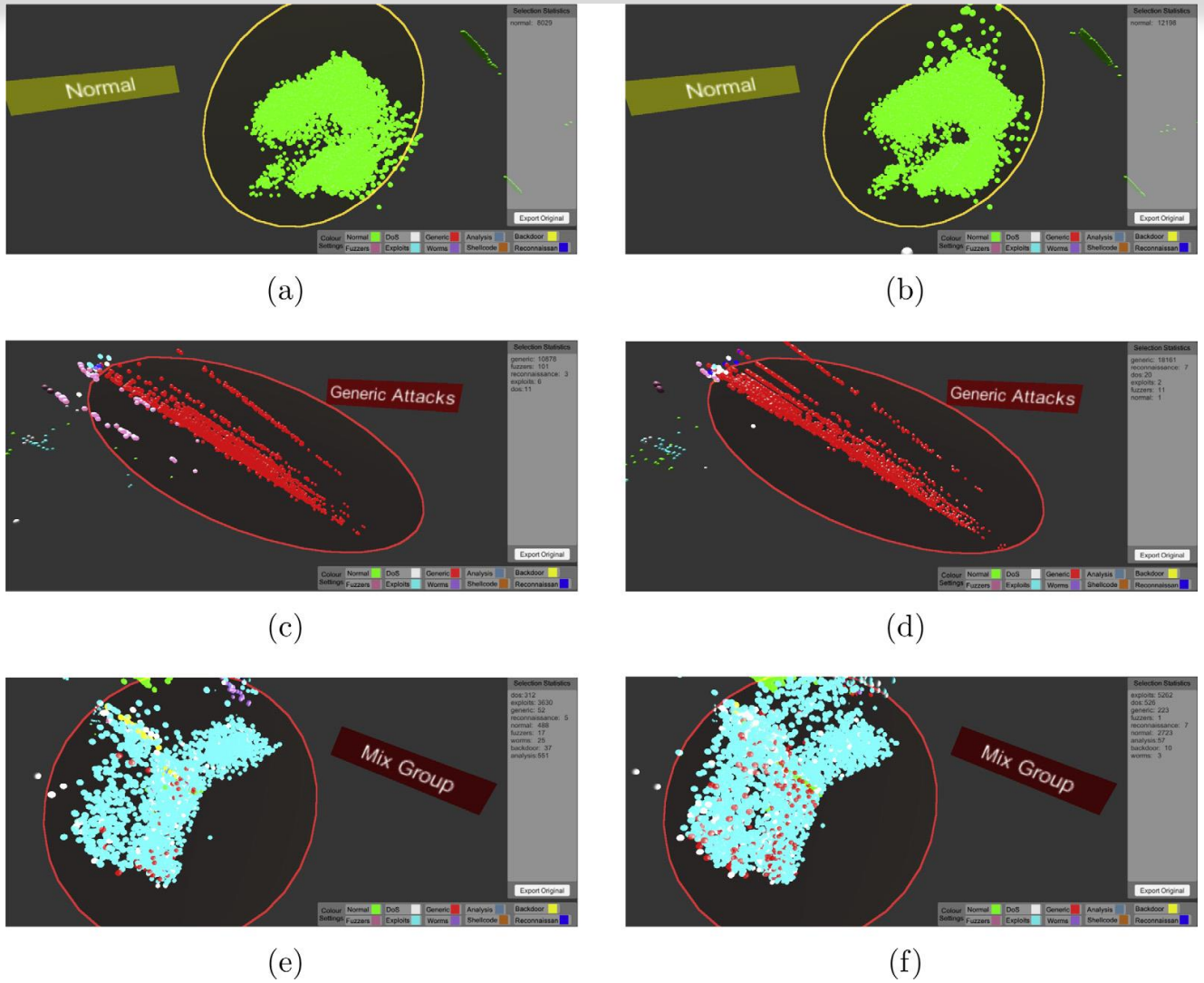


(c)

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

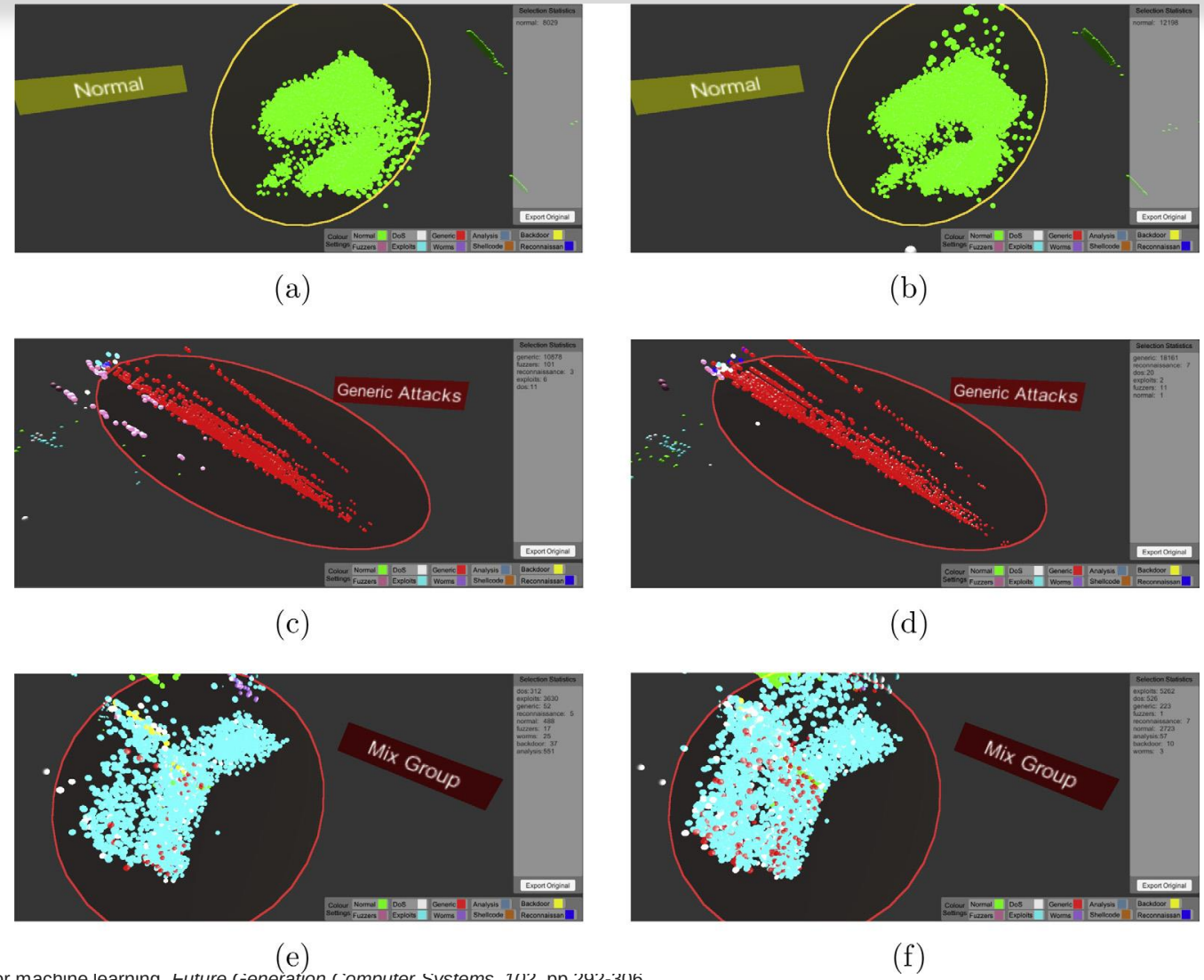
- A closer investigation of **UNSW-NB15**
  - Examples of normal traffic clusters from the training and testing data are shown in (a) and (b), respectively.
    - It can be seen from the figures that the characteristics of the training and testing data are very similar.
    - The implication of this for ML is that traffic in clusters that contain nearly homogeneous records are easier to correctly identify and typically result in high detection performance.
  - (c) and (d) show examples of a cluster from the training and testing data, respectively,
    - Contain mostly generic attacks.
    - The detection performance of most ML models on these generic attacks will be high.
    - The results from several studies appear to support this conclusion.
    - The two-stage approach for generic attacks :
      - recall 96.6%, precision 99.9%



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

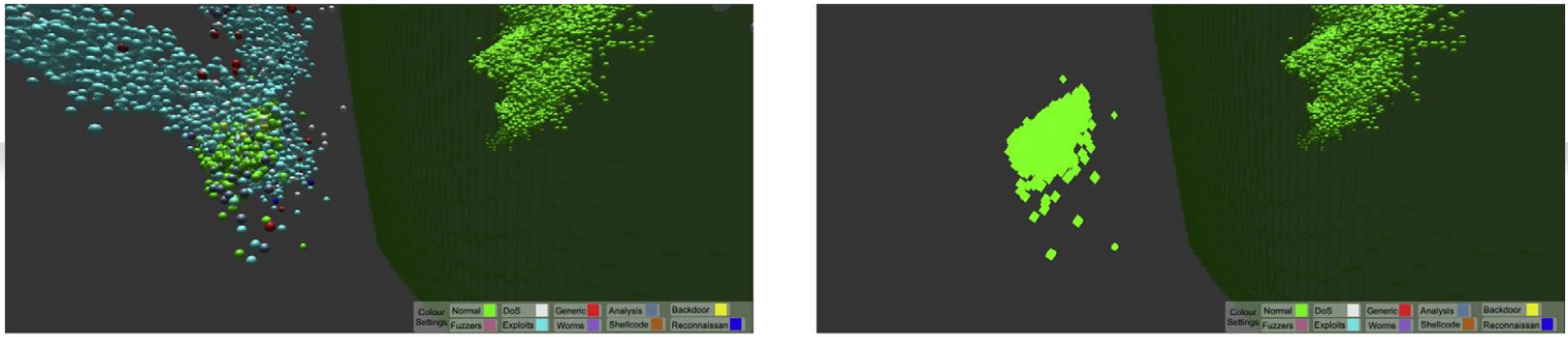
3D Visualization of Network Intrusion Detection Data

- A closer investigation of **UNSW-NB15**
  - In other sections of the dataset, the traffic is mixed together and there is no clear visual distinction between the different traffic categories within these clusters.
  - (e) and (f) for the training and testing data, respectively.
    - Such sections of the dataset are where ML techniques tend to face difficulties when it comes to the task of correctly identifying the individual categories of the traffic.
    - There is a low number of worm attacks, which is disproportional when comparing the numbers of different traffic in the training set and the testing set.
    - Furthermore, as there is a significant number of exploit attacks within the cluster, it is challenging for ML models to avoid misclassifying other traffic in the cluster as exploits.



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data



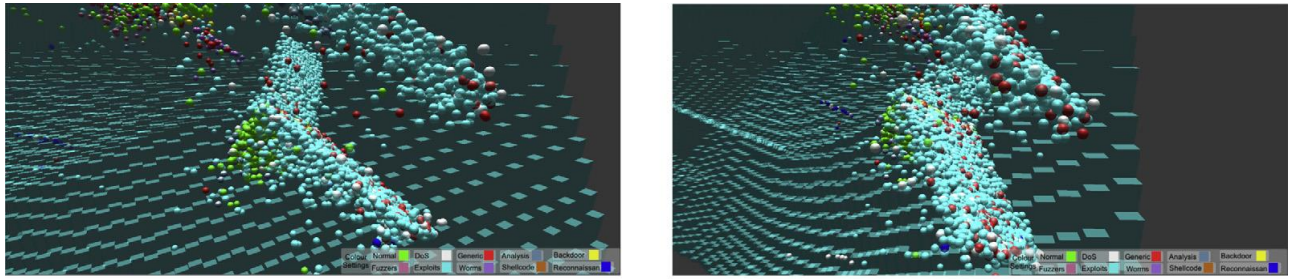
(a) (b)

- A closer investigation of **UNSW-NB15**
  - A close-up of the visualization that clearly shows where this misclassification occurs.
  - In (a), the normal traffic decision space can be seen on the right of the figure and all normal traffic instances within that space should have been correctly classified.
  - However, to the left of the figure there is a cluster of network traffic instances that are located outside the normal traffic decision space, which has a mixture of both normal and abnormal traffic.
  - In (b), all abnormal traffic instances (from (a)) have been removed and the figure highlights the instances of normal traffic that the ML model misclassified as abnormal traffic.
    - It can clearly seen that a large number of normal traffic instances have been misclassified as abnormal traffic.

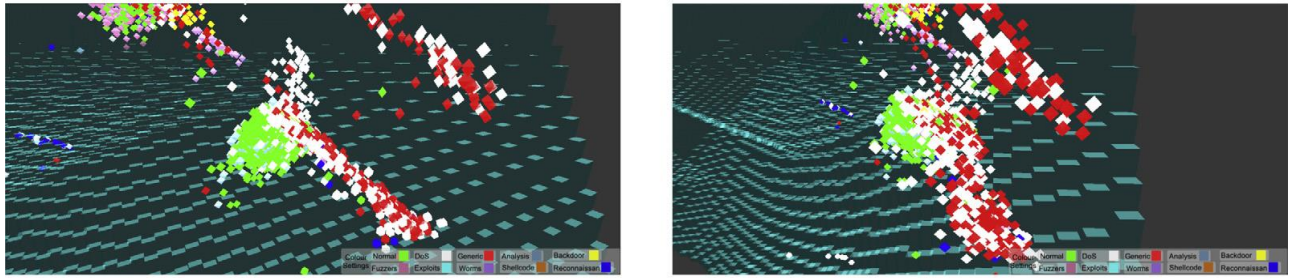
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

- A closer investigation of **UNSW-NB15**
  - In some clusters, in which there was a greater proportion of exploit attacks than other categories, other traffic would potentially be misclassified as exploits.
  - This was verified in the results, as can be seen in the example shown in the figures.
    - Figure (a) are close-ups of part of the exploits decision space from two different camera viewpoints; they show a mixture of exploits and other traffic that lie within the decision space for exploits.
    - In Figure (b), the traffic instances that were misclassified as exploits are highlighted from two different camera viewpoints.



(a)

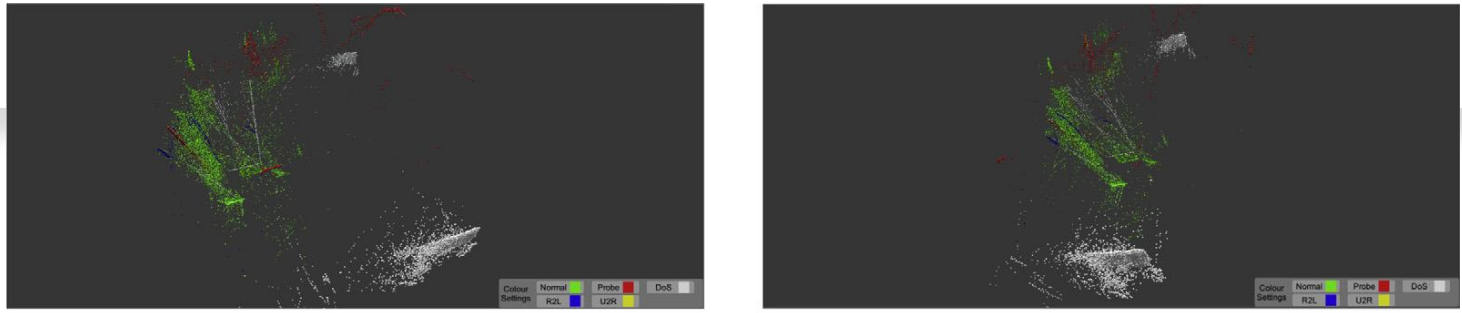


(b)

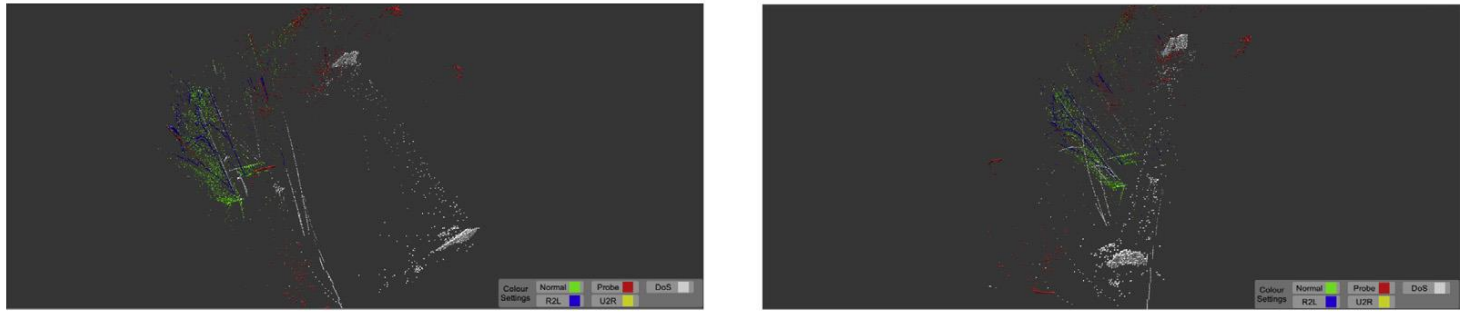
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

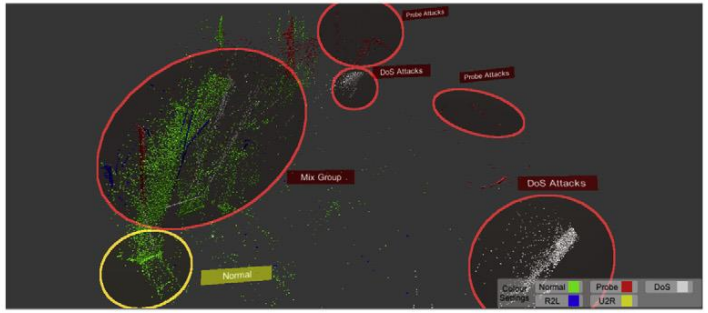
- Visually explore **NSL_KDD**
  - Figures show the overall 3D visual representation of the NSL_KDD dataset.
    - Figure (a) depicts a portion of the data from the training set,
    - Figure (b) displays part of the data from the testing set, from two different camera viewpoints respectively.
  - While the overall visual distributions share relatively similar characteristics, the training and testing sets in the NSL_KDD dataset have more differences between them as compared with the UNSW-NB15 dataset
    - i.e. the characteristics of traffic between the training and testing sets in the UNSW-NB15 dataset exhibited higher similarities.
  - Figure (c) shows the empirical grouping of data from the training set.
    - Like the UNSW-NB15 dataset, it also shows that the data can be grouped into several clusters.



(a)



(b)

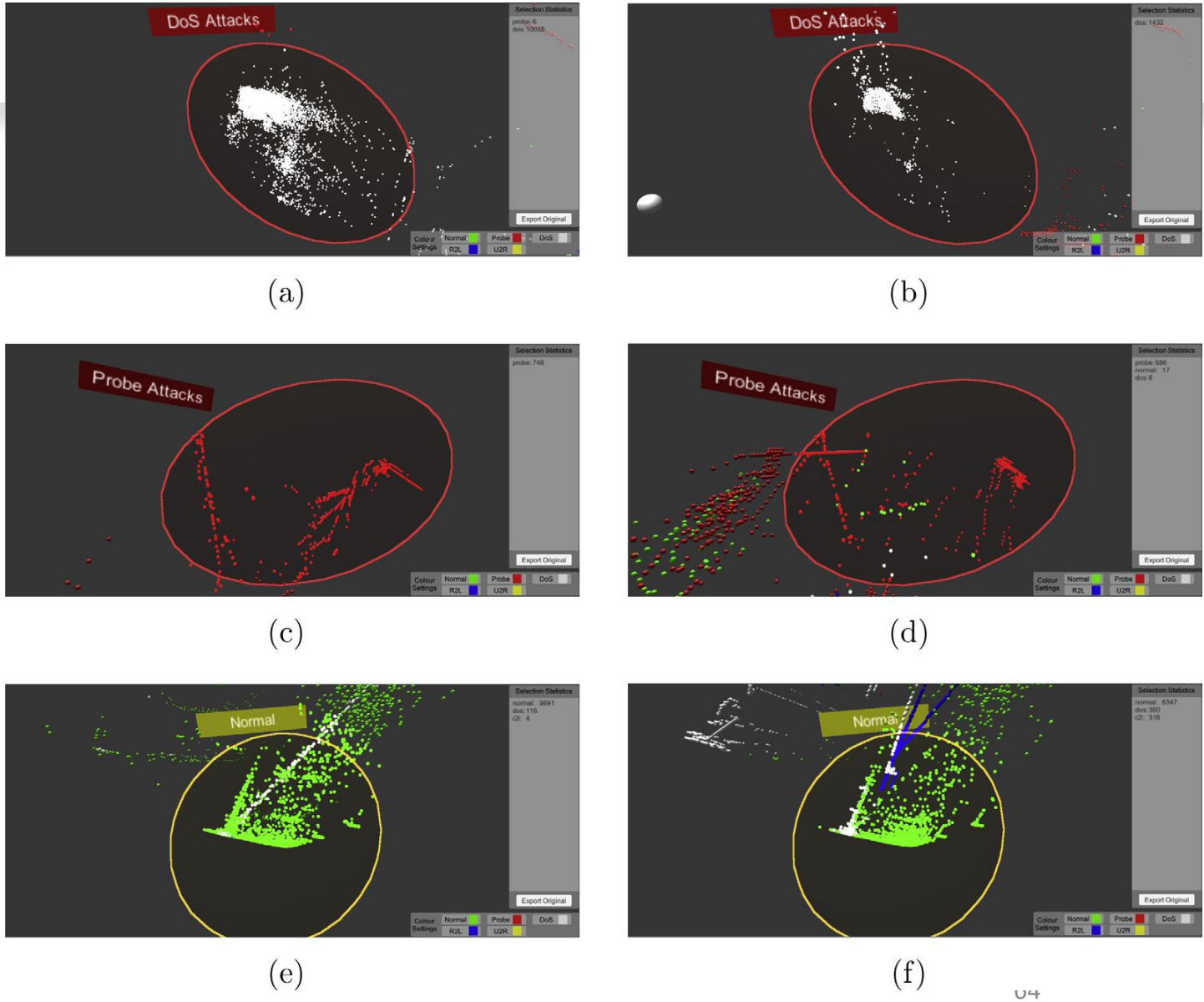


(c)

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

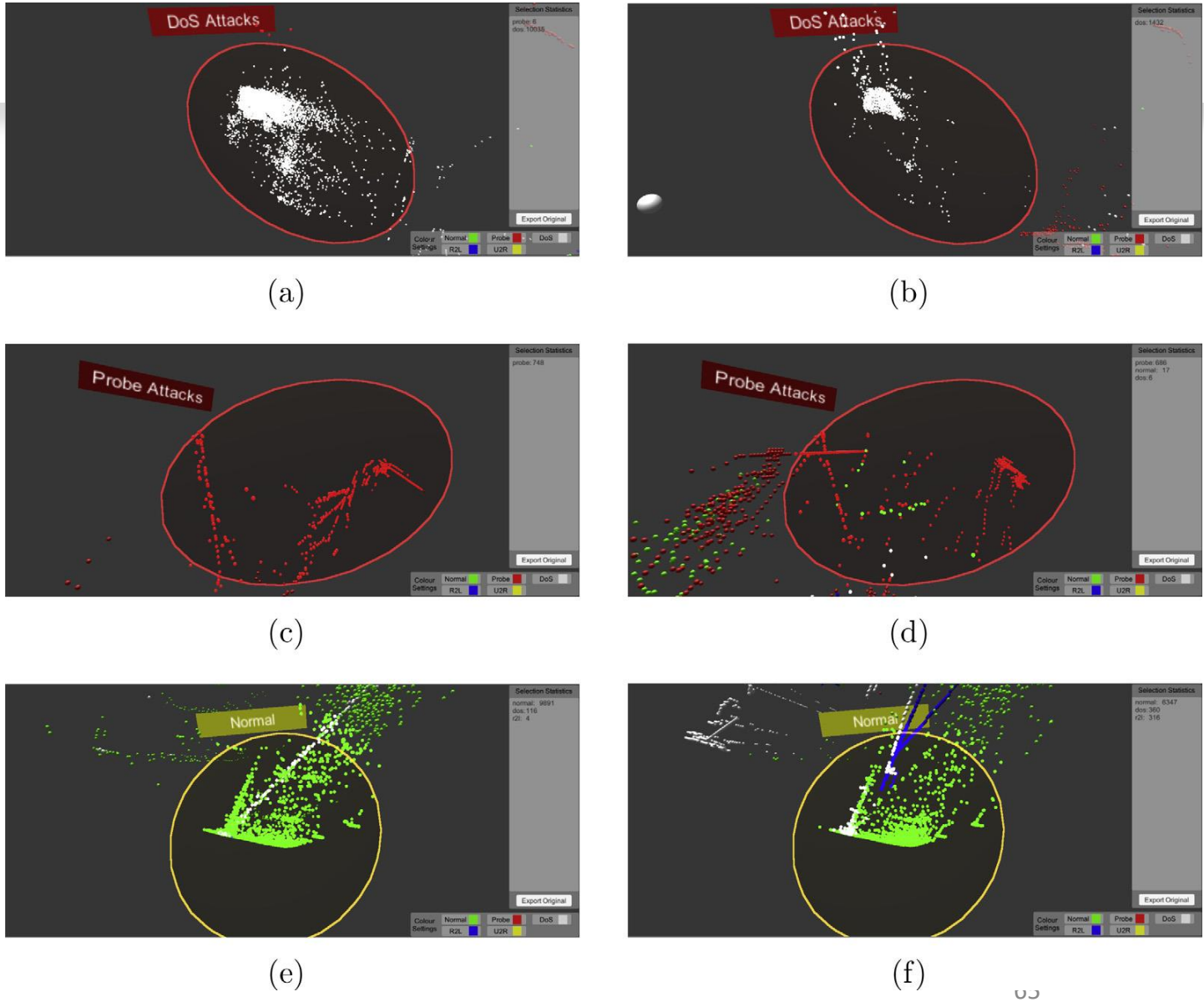
- A closer investigation of **NSL_KDD**
  - From the visual representation, one can observe that the data consists of clusters that contain mainly homogeneous records as well as clusters that contain diverse traffic.
  - Figures (a) and (b) show examples of DoS attack traffic from the training and testing data, respectively, from two different camera viewpoints.
    - It is clear that the display of traffic within the cluster is isolated away from the rest of the traffic.
    - It contains mostly homogeneous DoS attack records.
    - Due to the isolation and clustering of such homogeneous data, ML techniques are expected to perform well when identifying such traffic.



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

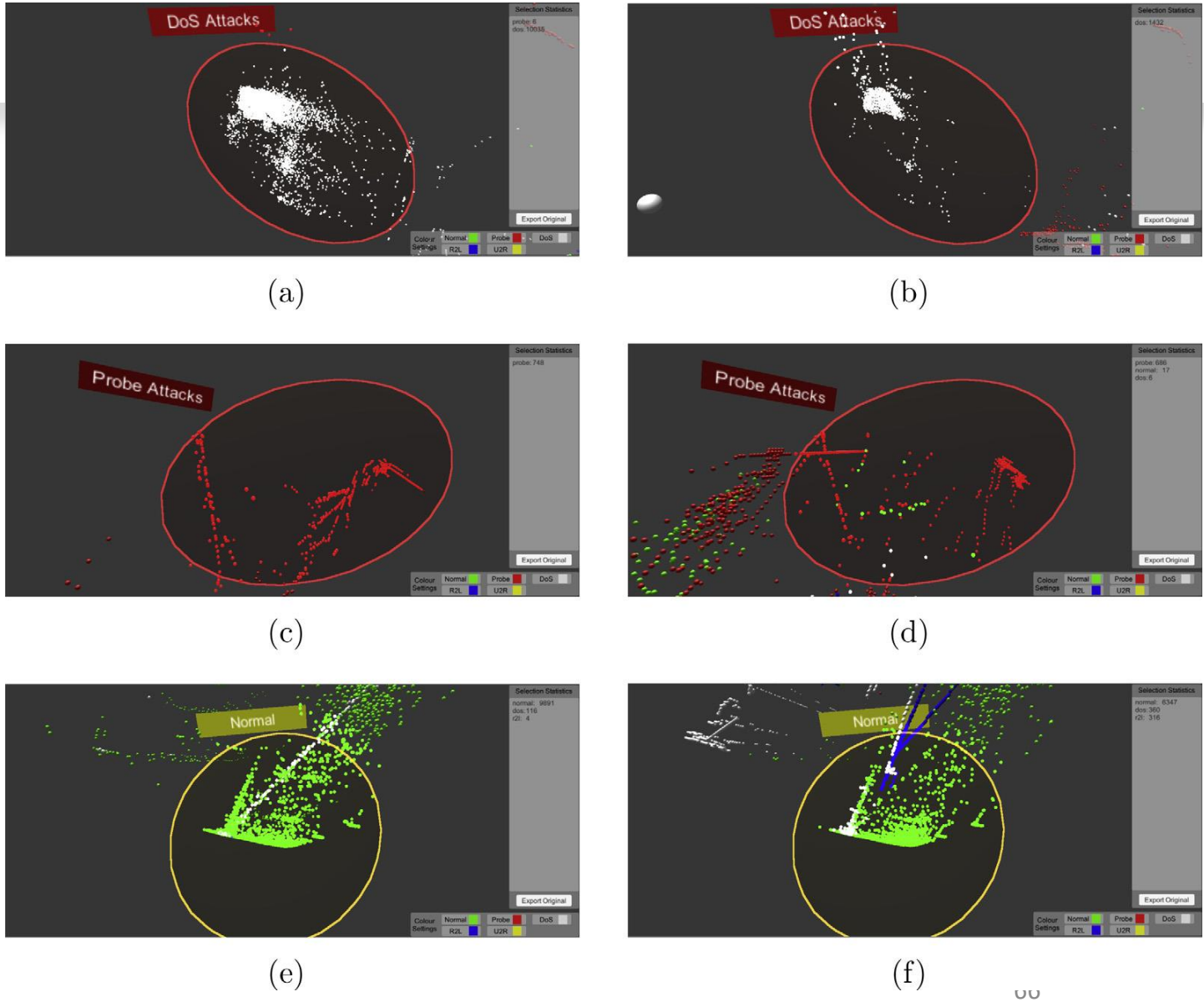
- A closer investigation of **NSL_KDD**
  - Figures (c) and (d) show another example of a cluster containing distinguishable homogeneous traffic
    - containing probe attacks, as from the training and testing data, respectively.
  - Although in (d) there are other traffic near the cluster, such as normal traffic, the majority of traffic inside the cluster is still probe attacks.
    - However, ML models may misclassify normal traffic and other attacks near this cluster as probe attacks when applied to the testing set.



Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

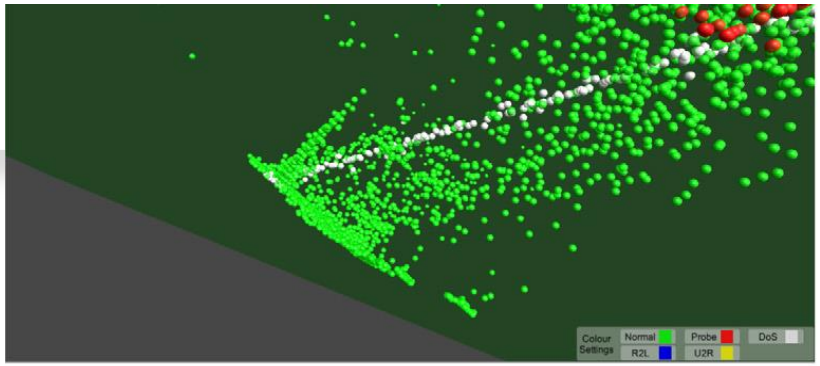
3D Visualization of Network Intrusion Detection Data

- A closer investigation of **NSL_KDD**
  - Cyber security experts have identified that the main difficulty faced by ML techniques in the NSL_KDD comes from previously unknown attacks in the testing set [1].
    - Even though the categories of attacks are the same in the training and testing sets, the characteristics of these traffic significantly differ.
  - A visual representation of this comparing the training and testing data is shown in (e) and (f), respectively.
    - In the training set in (e), there are almost no R2L attacks within this cluster.
    - On the other hand, there are many R2L attacks in the corresponding cluster from the testing set, as shown in (f).
  - ML models should be designed based on the training set.
    - ML techniques that do not consider such abnormal characteristics may perform unsatisfactorily when it comes to detecting such attacks.

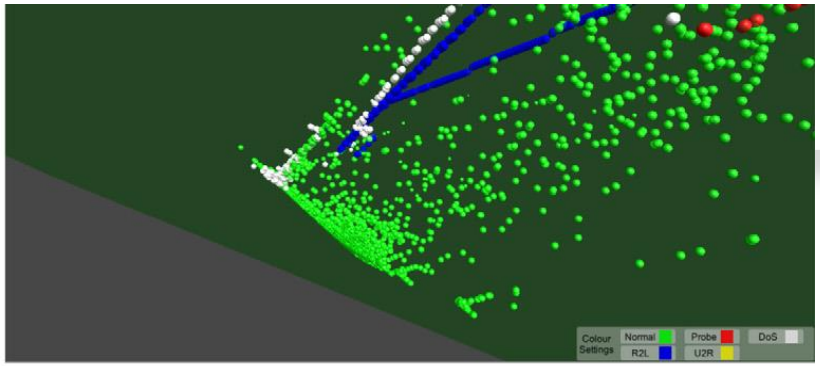


[1] G. Kim, S. Lee, S. Kim, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, Expert Syst. Appl. 41 (4) (2014) 1690–1700, http://dx.doi.org/10.1016/j.eswa.2013.08.066.

3D Visualization of Network Intrusion Detection Data



(a)



(b)

- A closer investigation of **NSL_KDD**
  - Visualizing decision boundaries of an SVM model.
    - Since the SVM model is trained using the training data, if the training data does not adequately represent the network traffic.
    - This can affect the detection accuracy of the ML model.
    - (a) shows part of the normal traffic decision space from the training data,
    - (b) shows part of the normal traffic decision space from the testing data.
  - In figure (b), **R2L** attacks are within the normal traffic decision boundaries in the testing set and they have been misclassified as normal traffic.
    - The reason is because these attacks were previously unknown in the training data.
    - In addition, there are a number of DoS attacks in both (a) and (b).
    - However, while samples are present in the training set, they are misclassified
      - Because they only occupy a small portion of that region in the training set.

[1] G. Kim, S. Lee, S. Kim, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, Expert Syst. Appl. 41 (4) (2014) 1690–1700, http://dx.doi.org/10.1016/j.eswa.2013.08.066.

3D Visualization of Network Intrusion Detection Data

- Summary of visualization results
  - The visualization approach works well on **UNSW-NB15** and **NSL_KDD** datasets
    - A significant portion of variances in the data are captured by first few principal components,
    - Otherwise, the visualization may result in a situation where no obvious clusters can be observed as there may not be enough variance to differentiate between network traffic instances.
  - The experiments show different geometric characteristics in the visual representation of the **UNSW-NB15** and **NSL_KDD** datasets.
    - The main challenge presented to ML techniques in the **UNSW-NB15** dataset is due to sections that contain clusters with highly heterogeneous data
    - The main challenge in the **NSL_KDD** dataset is due to the abnormal characteristics of unknown attacks.
  - This demonstrates the usefulness of 3D visualization.
    - This allows cybersecurity experts to examine and recognize patterns of incoming network traffic.
    - And also identify the major challenges in a specific network environment.

Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

Conclusion

- Machine learning is about more than just choosing the right classifier and knowing about algorithms, iterative process
- Key: spend time to explore and understand the data
- Cybersecurity provides some unique challenges when applying machine learning, e.g. class imbalance, lack of training data…
  - Sometimes continue to tune algorithms or the learning process may not be the answer
- To obtain better results
  - Collecting more data
  - generating more descriptive features
  - …

References

- *[1] G. Kim, S. Lee, S. Kim, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, Expert Syst. Appl. 41 (4) (2014) 1690–1700, http://dx.doi.org/10.1016/j.eswa.2013.08.066.*
- *[2] Network Traffic Analysis, Machine Learning and Security Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *[3] Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14 (pp. 329-340). Springer International Publishing.*
- *[4] Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. Future Generation Computer Systems, 102, pp.292-306.*

Adversarial Machine Learning

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

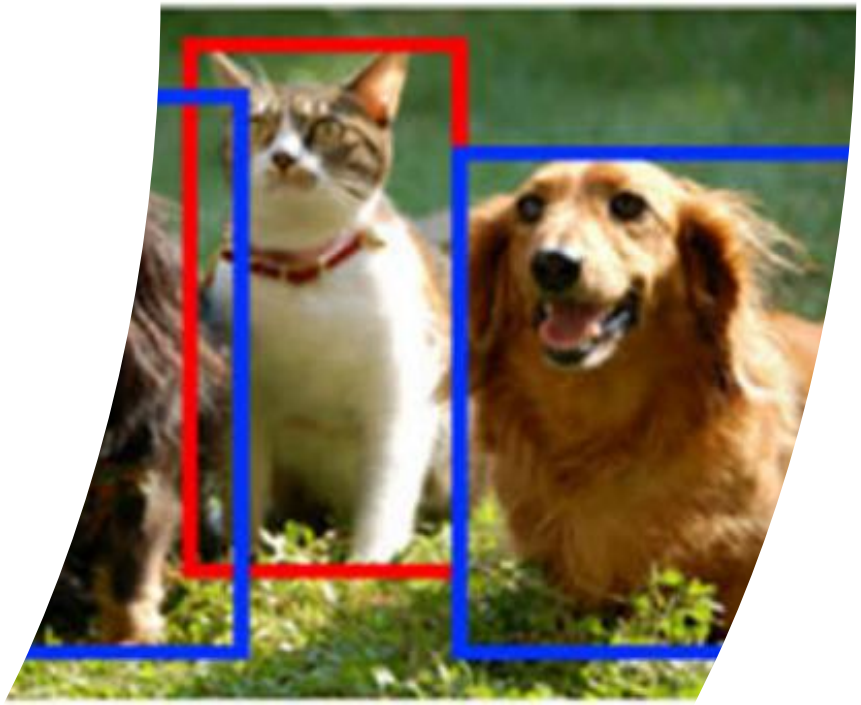
Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- Introduction
- Adversarial Examples
  - White-box attacks
  - Black-box attacks
  - Physical attacks
  - Universal adversarial examples
  - Robustness to adversarial examples
- Backdoor (Trojan) attacks
  - Visible backdoor
  - Invisible backdoor
  - Clean label backdoor
  - Module backdoor
  - Defense against backdoor attacks.

Machine learning models are doing well

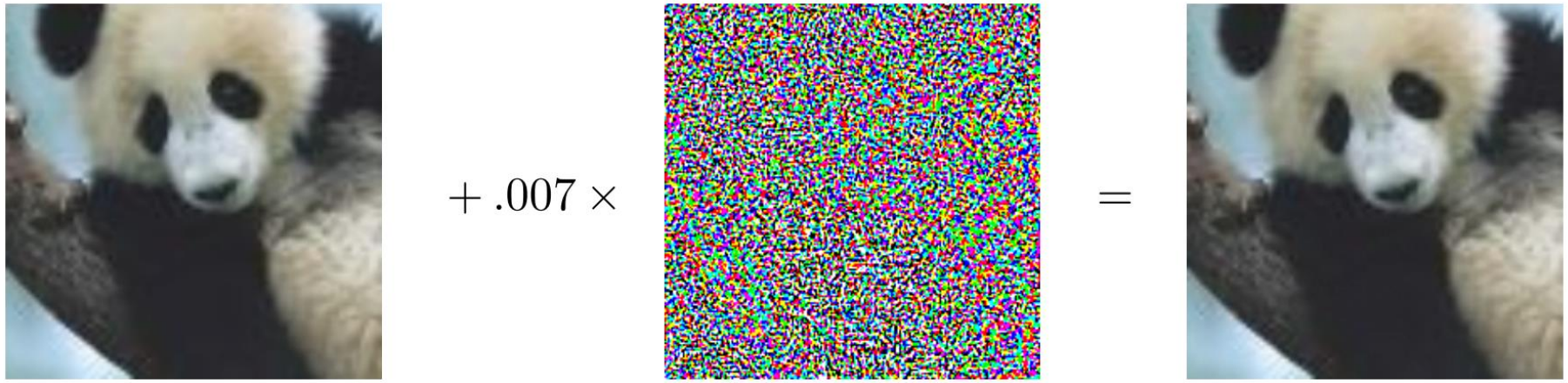
- Image classification
- Object detection
- Self-driving cars
- Speech-to-text
- Etc …



Are models just Clever Hans?

Clever Hans performing in 1904 (https://en.wikipedia.org/wiki/Clever_Hans)

Are models just Clever Hans?



$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

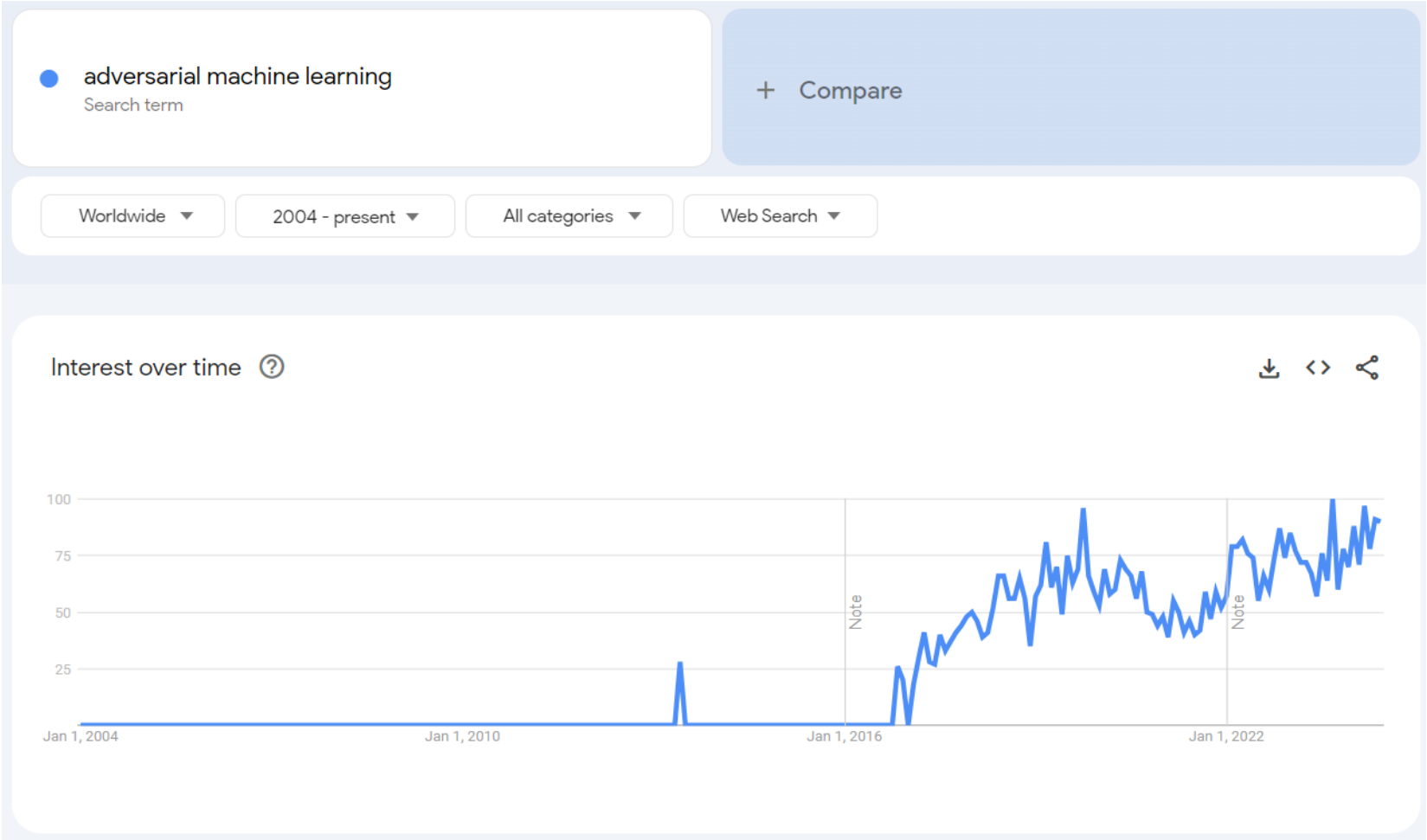
$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Explaining and Harnessing Adversarial Examples. ICLR (Poster) 2015

Adversarial Machine Learning



https://trends.google.com/

What is Adversarial Machine Learning?

Attackers aware of the ML techniques being deployed against them can, for example, contaminate the training data to manipulate a learned ML classifier in order to evade subsequent classification, or can manipulate the specific metadata upon which the ML algorithms make their decisions and exploit identified weaknesses in these algorithms so called **Adversarial Machine Learning**.

https://www.dst.defence.gov.au/publication/adversarial-machine-learning-cyber-security-ngtf-project-scoping-study

White-box Adversarial Examples

- Adversarial example
  - Given original input x, an adversarial example x' is obtained by slightly modifying x.
    - x' ← x + δ
    - x' ≈ x (e.g., $|x' - x|_1 = \sum|(x' - x)_i| < \epsilon$)
  - Targeted attack vs. untargeted attack

- White-box
  - An adversary (attacker) knows **everything** about the target model.
    - Model architecture
    - Model weights
    - Training data
    - Etc.
  - An adversary can calculate gradients with respect to input data.
    - For constructing adversarial examples.

Fast Gradient Signed Method (FGSM)

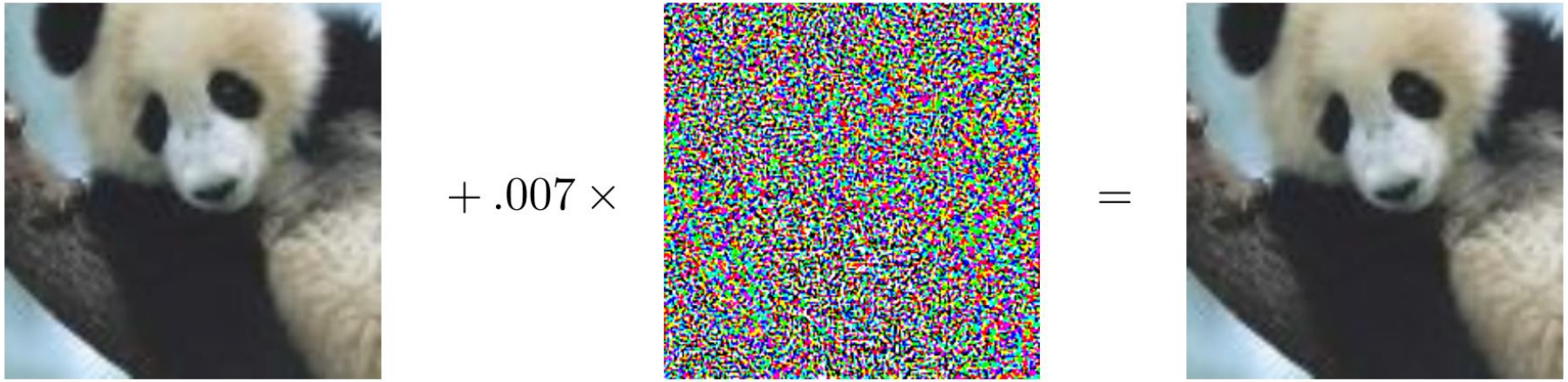
- White-box attack
  - Targeted: Fool a model to output predefined labels.
  - Untargeted: Fool a model to output incorrect labels.

- Fool a model by calculating gradients only once.
  - Very **fast**.
  - Simple but effective.

- Let J(θ, x, y) be the cost used to train a model (a deep neural network):
  - $\delta \leftarrow \epsilon \cdot sign\left(\nabla_x J(\theta, x, y)\right)$
  - $x' \leftarrow x + \delta$ ("+" for untargeted attack, "-" for targeted attack)
  - $x' \leftarrow clip(x', 0, 1)$

Fast Gradient Signed Method (FGSM)

- Let J(θ, x, y) be the cost used to train a model (a deep neural network):
  - $\delta \leftarrow \epsilon \cdot sign\left(\nabla_x J(\theta, x, y)\right)$
  - $x' \leftarrow x + \delta$
  - $x' \leftarrow clip(x', 0, 1)$



$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

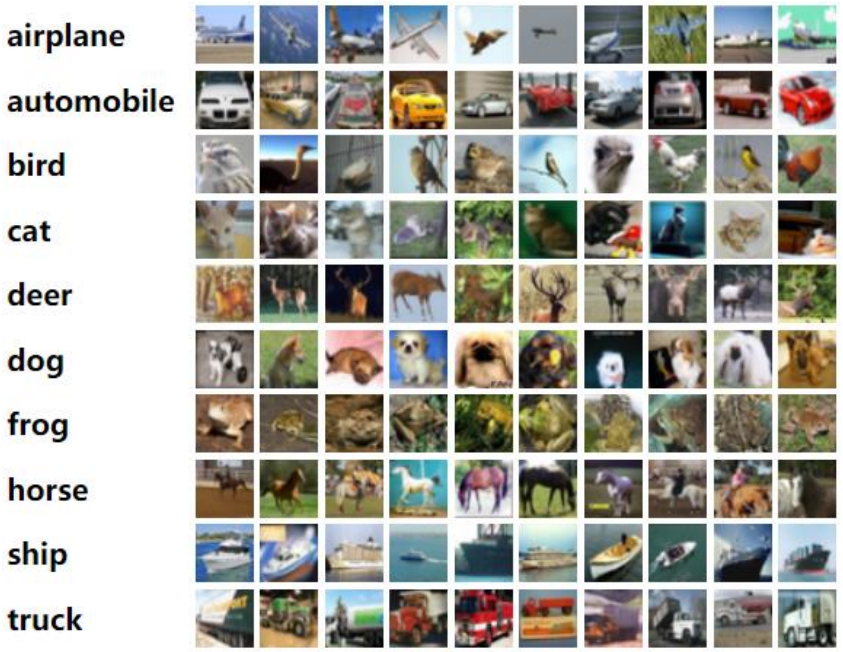
"gibbon"
99.3 % confidence

Fast Gradient Signed Method (FGSM)

MNIST

- $\epsilon = 0.25$
- an error rate of 99.9% with an average confidence of 79.3%

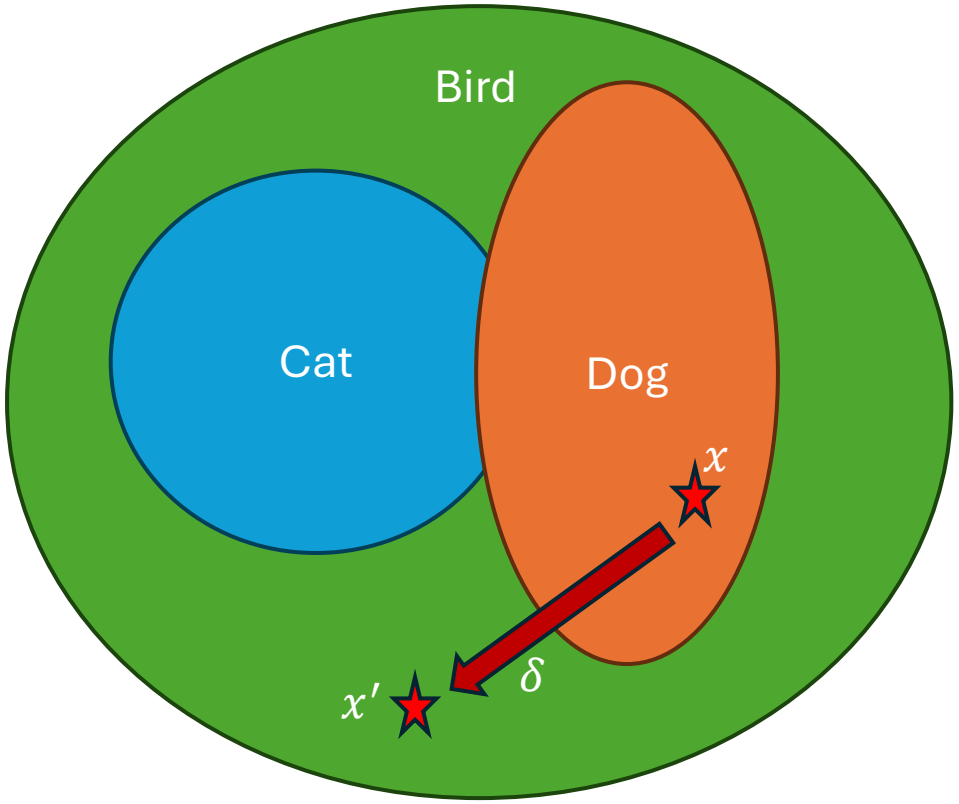
CIFAR10



- $\epsilon = 0.1$
- an error rate of 87.15% with an average confidence of 96.6%

Fast Gradient Signed Method (FGSM)

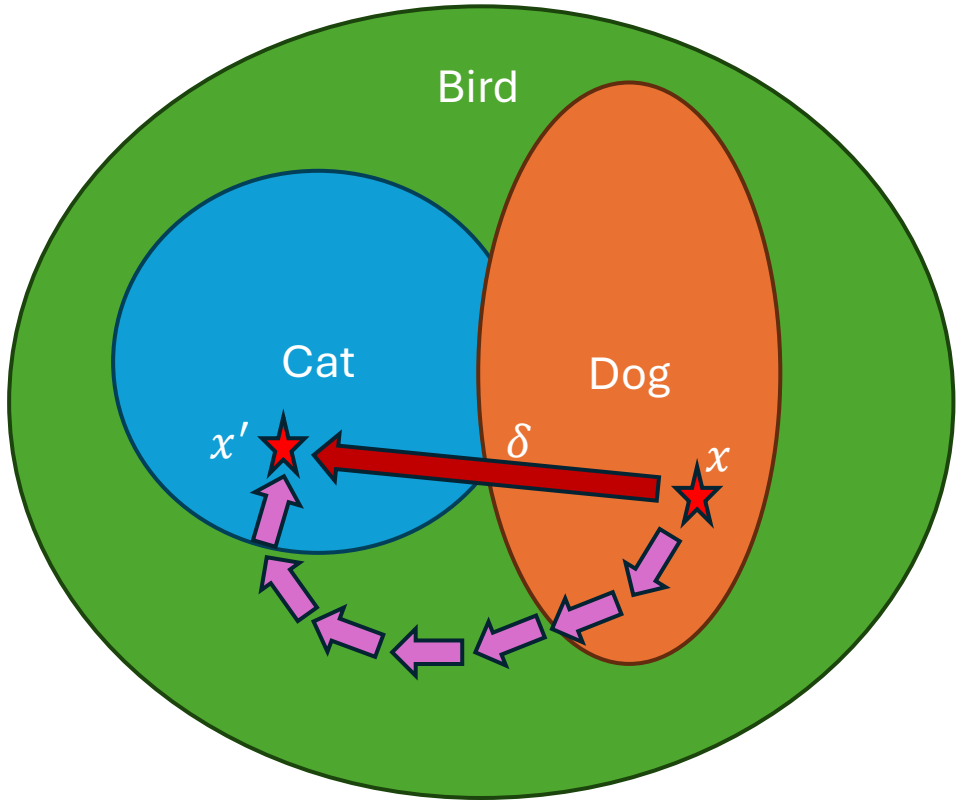
- May fail to produce targeted attacks
  - Fool a model to output predefined labels.
  - May not fool a model to classify a dog as a cat.



Projected Gradient Descent (PGD)

- Much stronger white-box attack than FGSM
- High performance for **targeted** attacks
  - Fool a model to predict predefined labels.

- Key idea: a single step may fail, so we do **multiple** steps!
  - 1. $x' \leftarrow x$
  - 2. $x' \leftarrow x' - \epsilon \cdot sign\left(\nabla_{x'} J(\theta, x', y)\right)$
    (minus for targeted attack)
  - 3. $\delta \leftarrow clip(|x' - x|_1, -\epsilon, \epsilon)$
    (require $|\delta|_1 < \epsilon$)
  - 4. $\mathrm{x}' \leftarrow clip(x + \delta, 0, 1)$
  - 5. go back to Step 2 if x' is not predicted as target.



Projected Gradient Descent (PGD)

- Much stronger white-box attack than FGSM
  - CIFAR10, $\epsilon = 0.03$

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

Black-box Adversarial Examples

- Black-box attacks
  - An adversary only observes input and output pairs.
    - No knowledge about the internal workings of the target model.
    - Cannot compute gradients w.r.t input data.

- Grey-box attacks also attract attention in the field.
  - Partial knowledge of the target model
    - Architecture of the target model
    - Logits output from the target model (in addition to predicted labels).
  - We do not cover this category in this subject.

Black-box Adversarial Examples

- Attacking strategy
  - Train a similar model on similar data compared to the target model.
  - Generate white-box attacks on this model.
  - Evaluate these attacks on the target model.

- Intuition: **similar** models trained on **similar** data would have **similar** decision boundaries.

Black-box Adversarial Examples

- Single model approach
  - Generate attacks using a single model.
  - Evaluate these attacks on another model.

Accuracy of Untargeted Attacks

	RMSD	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
ResNet-152	22.83	0%	13%	18%	19%	11%
ResNet-101	23.81	19%	0%	21%	21%	12%
ResNet-50	22.86	23%	20%	0%	21%	18%
VGG-16	22.51	22%	17%	17%	0%	5%
GoogLeNet	22.58	39%	38%	34%	19%	0%

Matching Rate of Targeted Attacks

	RMSD	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
ResNet-152	23.13	100%	2%	1%	1%	1%
ResNet-101	23.16	3%	100%	3%	2%	1%
ResNet-50	23.06	4%	2%	100%	1%	1%
VGG-16	23.59	2%	1%	2%	100%	1%
GoogLeNet	22.87	1%	1%	0%	1%	100%

Black-box Adversarial Examples

- Ensemble approach
  - Generate attacks using an ensemble of models.
  - Evaluate these attacks on another model.

Accuracy of Ensemble Untargeted Attacks

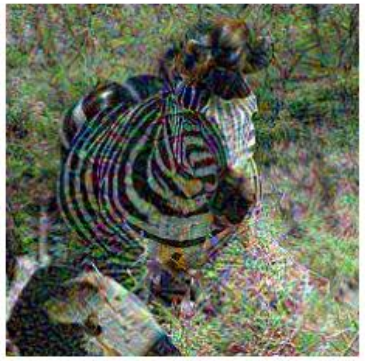
	RMSD	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
-ResNet-152	17.17	0%	0%	0%	0%	0%
-ResNet-101	17.25	0%	1%	0%	0%	0%
-ResNet-50	17.25	0%	0%	2%	0%	0%
-VGG-16	17.80	0%	0%	0%	6%	0%
-GoogLeNet	17.41	0%	0%	0%	0%	5%

Matching rate of Ensemble Targeted Attacks

	RMSD	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
-ResNet-152	30.68	38%	76%	70%	97%	76%
-ResNet-101	30.76	75%	43%	69%	98%	73%
-ResNet-50	30.26	84%	81%	46%	99%	77%
-VGG-16	31.13	74%	78%	68%	24%	63%
-GoogLeNet	29.70	90%	87%	83%	99%	11%

Black-box Adversarial Examples

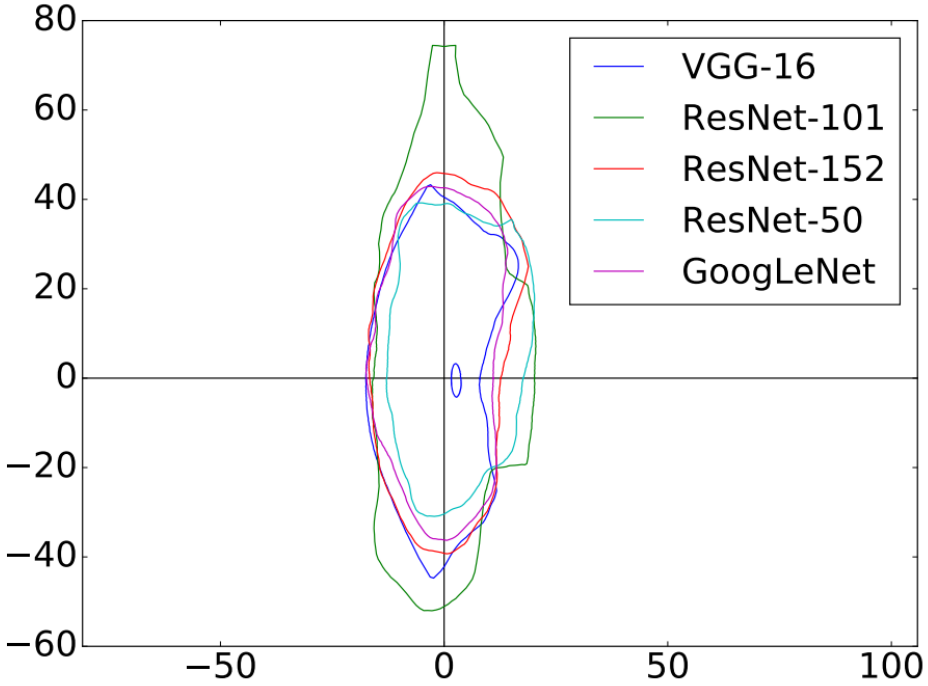
- Examples

	water buffalo, water ox, Asiatic buffalo, Bubalus bubalis	herd, milk, beef cattle, farmland, cow	rugby ball		pastime, print, illustration, art
	zebra	equid, stripe, savanna, zebra, safari	apiary, bee house		wood, people, outdoors, nature

Black-box Adversarial Examples

anemone fish

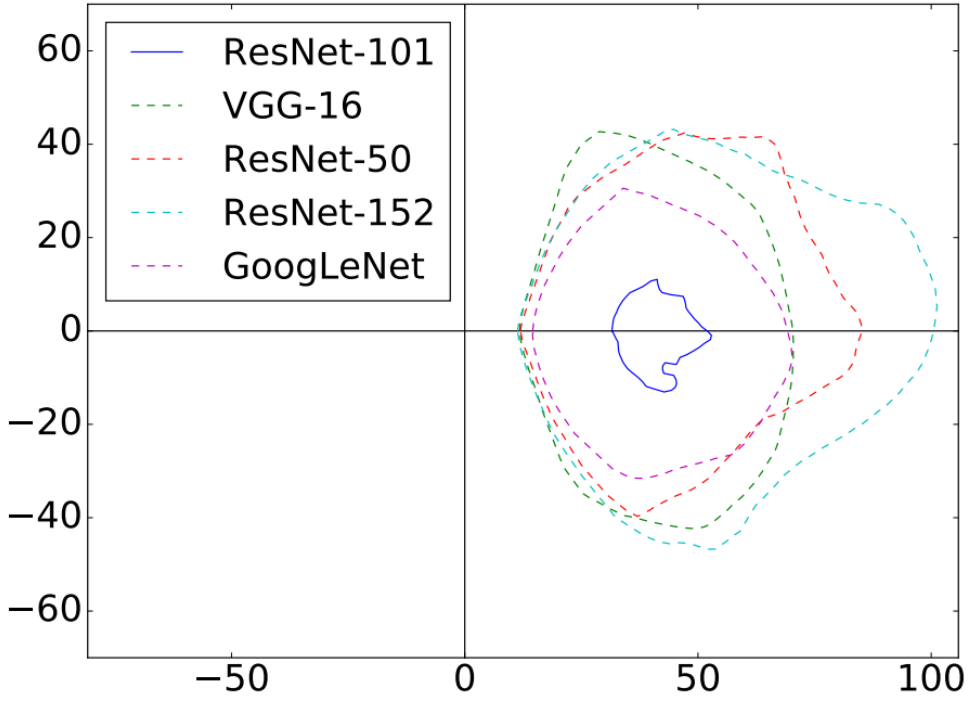
- Visualizing decision boundaries for correct predictions.
  - All points in the decision boundaries are classified as the ground truth label.
    - X-axis: the gradient direction of VGG-16
    - Y-axis: a random orthogonal direction
    - The origin of the coordinate plane corresponds to the original image.
    - The units of both axises are 1 pixel values.
  - Different models share similar decision boundaries.
    - Can explain the high performance of ensemble untargeted attacks.



Black-box Adversarial Examples

anemone fish

- Visualizing decision boundaries for targeted attack.
  - All points within the decision boundaries are classified as the target label (e.g., window).
    - X-axis: the targeted adversarial direction
    - Y-axis: a random orthogonal direction.
    - The units of both axises are 1 pixel values.
    - The origin of the coordinate plane corresponds to the original image.
    - The targeted adversarial direction is computed as the difference between the original image and the adversarial image.
  - The ensemble attack contains all models except ResNet101.
    - Different models share similar decision boundaries for the target label.
    - Can explain the success of ensemble targeted attacks.



Physical Adversarial Examples

- Machine learning / deep learning models are widely deployed in real-world applications.
  - Self-driving cars.

https://www.wired.com/story/self-driving-cars-are-being-put-on-a-data-diet/

- Adversarial examples in the physical world.
  - Techniques discussed so far focus only on digital attacks.
    - Files are transferred in a digital format.
  - Physical attacks are also effective
    - Attacks are captured by physical devices, e.g., cameras.
    - Can cause serious harm, e.g., loss of Life.
    - Pose practical threats to the public.

Physical Adversarial Examples

- Physical World Challenges
  - Environmental Conditions.
    - The distance and angle of a camera in an autonomous vehicle with respect to a road sign varies continuously.
    - Other environmental factors include changes in lighting/weather conditions, and etc.
  - Spatial Constraints.
    - a physical road sign, the attacker cannot manipulate background imagery.
  - Physical Limits on Imperceptibility.
    - Transferring minute perturbations to the real world, we must ensure that a camera is able to perceive the perturbations.
  - Fabrication Error.
    - A fabrication device, such as a printer, can produce some color error.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. and Song, D., 2018. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1625-1634).

Physical Adversarial Examples

- Robust Physical Perturbation
  - Targeted attack
  - Key idea: including all potential transformations into the calculation of perturbations:
    - Varying distances
    - Varying angles
    - Varying lighting conditions
    - Printing errors, measured by Non-Printability Score (NPS)
    - Etc.
  - This idea is widely used to improve robustness of perturbations.
  - The optimization objective
    - $argmin_{\delta}\ \lambda \left|\left|M_x \cdot \delta\right|\right|_p + NPS + E_{x_i \sim X^V}\, J(f_\theta\left(x_i + T_i(M_x \cdot \delta)\right), y^*)$

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. and Song, D., 2018. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1625-1634).

Physical Adversarial Examples

- Attack examples

Distance/Angle	Subtle Poster	Subtle Poster Right Turn	Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
5′ 0°					
5′ 15°					
10′ 0°					
10′ 30°					
40′ 0°					
Targeted-Attack Success	100%	73.33%	66.67%	100%	80%

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. and Song, D., 2018. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1625-1634).

Universal Adversarial Perturbations

- So far, we have generated perturbations for a specific input.
  - Can we have perturbations that can be applied to any input?
  - Yes, universal adversarial perturbations (UAP).

- Let's go beyond images
  - Adversarial machine learning is universal across most if not all AI fields.
  - Speech-to-text
    - Transcribe speech into text format

https://eshop.macsales.com/blog/86291-how-to-enable-and-customize-siri-in-macos-sonoma-and-ventura/

Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Universal Adversarial Perturbations

- To design UAP, we need to **change our perspective**
  - For traditional adversarial examples
    - Input is a **signal** and adversarial perturbations δ are **noise** added to the signal.
    - δ is tailored for each input.
  - For UAP
    - δ is considered as a **signal** which leads to classification results.
      - E.g., the transcript of speech.
    - Input is instead considered as "**noise**" applied to δ,
      - δ is robust against modification by adding input.

Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Universal Adversarial Perturbations

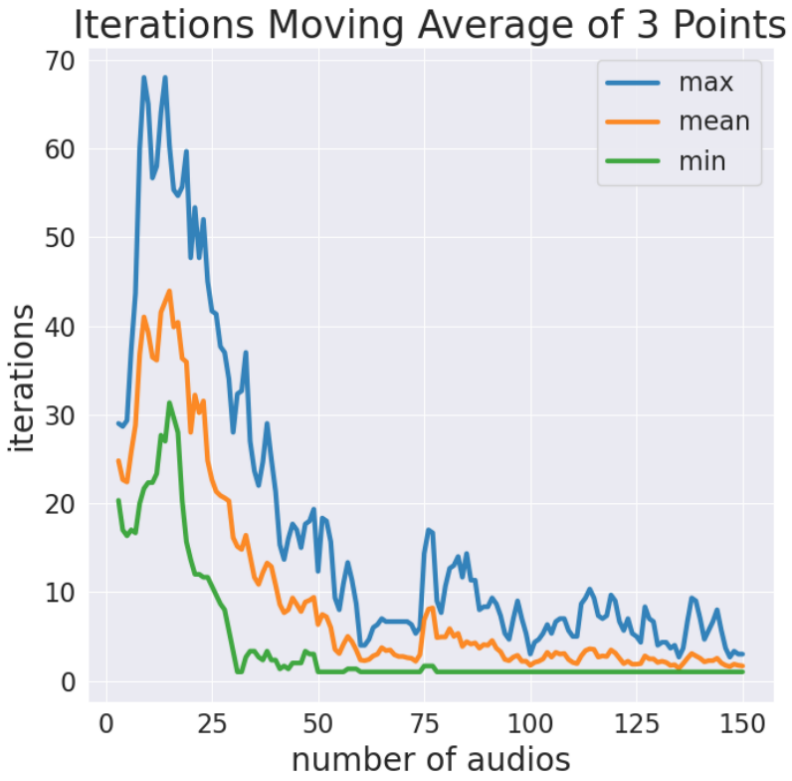
- How to generate UAP to fool a speech-to-text model?
- Key idea: making δ gradually more robust against more audio
  - This idea is similar to physical attacks: “including all potential transformations into the calculation of perturbations”.
- Algorithm
  - target model, f; a set of audio, D; target phrase, t; the minimum success rate η;

```
initialize δ = 0
initialize a subset G ⊂ D
while iterations < max iterations do
  set success number s = 0
  for each audio x ∈ G    // G is shuffled for each iteration
    increase s by 1 if f(x') = t
    modify δ via gradient decent
  end for
  if s is equal to |G| then
    return δ^τ if |G| = |D|
    add more audios into G from D
  end if
end while
```


Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Universal Adversarial Perturbations

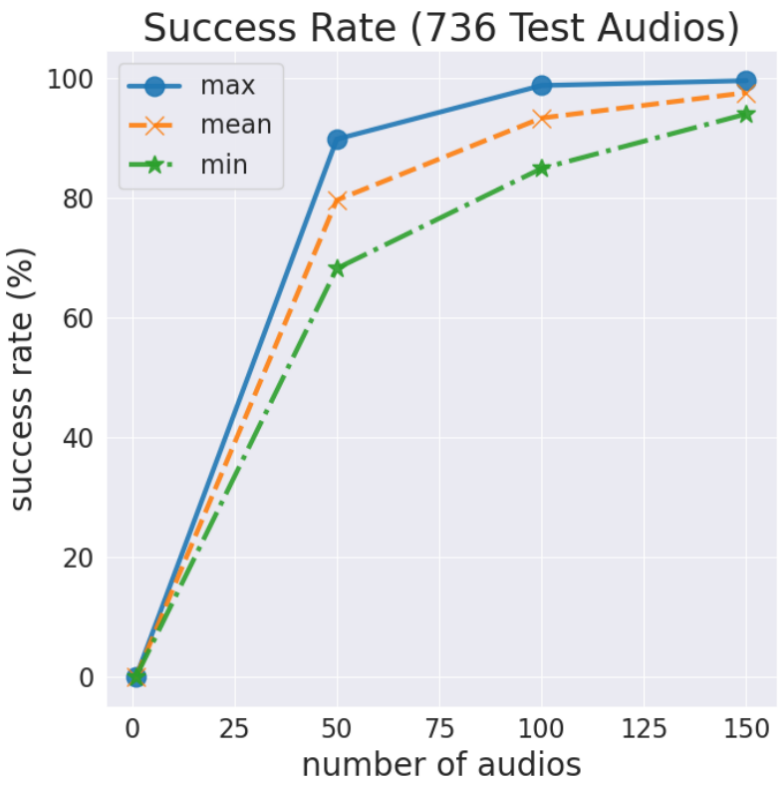
- The audio set only contained 1 audio at the start of the generation process.
  - When the generated UAPs were able to attack all audio in the current set, a new audio was added to the set
  - the UAPs for a specific set only need to handle one new audio.
  - The set at the end of the process contained 150 audio.
- The figure shows the iteration trend to generate UAPs capable of attacking all audio
  - the size of the audio set was gradually increased.
  - The horizontal axis represents the number of audio used to train UAPs
  - the vertical axis indicates the number of iterations needed for the UAPs to attack all audio in the set.
- Interesting observations
  - Early on when the size of the set was small, the number of iterations increased as more audio was added to the set.
    - This is reasonable since the UAPs had to attack a greater number of audio, so more computation was required to find a solution.
  - The iterations started to decrease when the size of the audio set reached around 20.
    - The generated UAPs are considered as signals that are transcribed into the target phrase, while audio containing speech are considered as noise being applied to UAPs.
    - From this perspective, it is intuitive that after a while, the UAPs become more robust despite additional audio being added to the set.
    - In other words, when UAPs are robust against a large set of audio, fewer iterations are required to find a solution to attack the newly added audio.



Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Universal Adversarial Perturbations

- Generalization of UAPs to unseen input
  - Applied the UAPs to all audio with a duration between 2 to 4 seconds from the testing set.
    - The horizontal axis represents the number of audio used to train UAPs
    - The vertical axis shows the success rate of applying UAPs to all 736 audio with a duration between 2 to 4 seconds from the testing set.
  - the success rate of UAPs increased as more audio was used for training.
    - This increase in success rate is complementary to the above discussion that UAPs become more robust against new audio as the size of the training set increases.



Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Universal Adversarial Perturbations

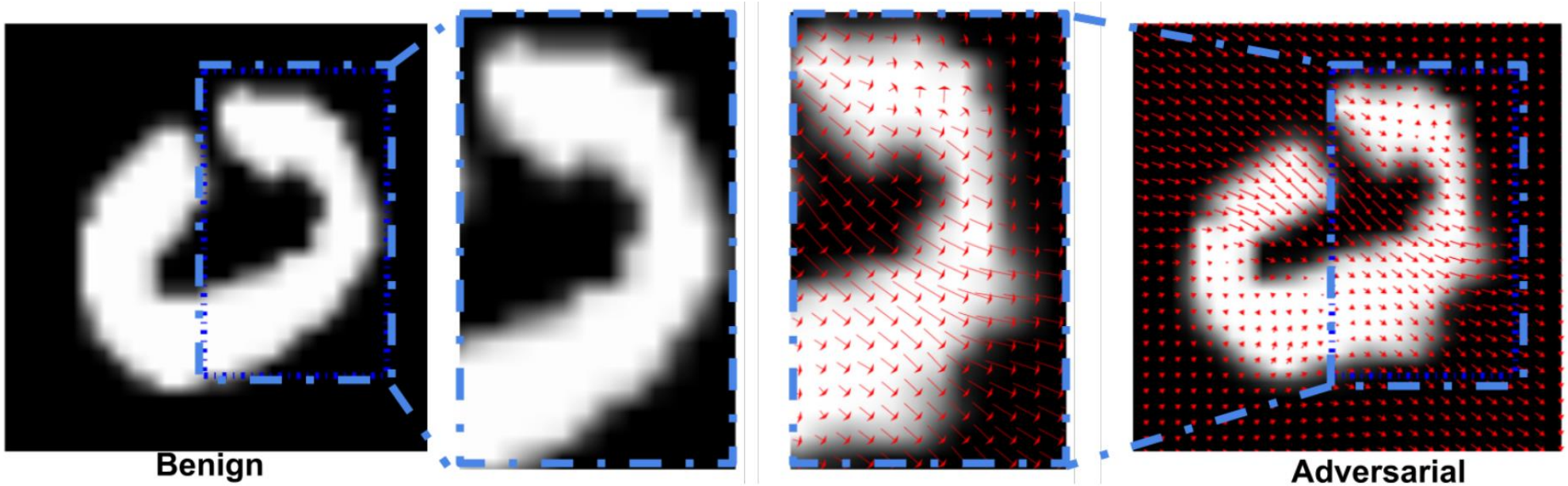
- UAPs are indeed signal
  - UAPs themselves can be transcribed as the target phrase.
  - The experimental results show that the transcripts of differently sliced UAPs were consistent with the corresponding portions of the target phrase.

slice*	power off	use airplane mode	visit malicious dot com
0.1	p	use	
0.2	pon	use	visit
0.3	po	use air	visit mali
0.4	po	use airplane	visit malicious
0.5	power	use airplane mode	visit malicious dotd co
0.6	power off	use airplane mode	visit malicious dot com
0.7	power off	use airplane mode	visit malicious dot com
0.8	power off	use airplane mode	visit malicious dot com
0.9	power off	use airplane mode	visit malicious dot com
1.0	power off	use airplane mode	visit malicious dot com

Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.

Unrestricted Adversarial Examples

- Traditionally, adversarial perturbations are constrained by ℓ_p norm.
  - ℓ_p norm is not necessarily a good metric for measuring the imperceptibility of adversarial perturbations.
  - Only a one-pixel offset would result in a large ℓ_p norm.
- Spatially Transformed Adversarial Examples



The digit “0" is misclassified as “2".

Xiao, C., Zhu, J.Y., Li, B., He, W., Liu, M. and Song, D., 2018. Spatially transformed adversarial examples. arXiv preprint arXiv:1801.02612.

Unrestricted Adversarial Examples

- An adversary can exploit generative models to generate attacks

- (a) the ground truth (GT) class is “pretzel”.
- (b) the GT is “car”.
- These new colors added are commonly found in images from the target class.
  - For instance, green in Golfcart and blue sea in Tench images.

Bhattad, A., Chong, M.J., Liang, K., Li, B. and Forsyth, D.A., 2019. Unrestricted adversarial examples via semantic manipulation. *arXiv preprint arXiv:1904.06347*.

Unrestricted Adversarial Examples

- An adversary can exploit generative models to generate attacks

- the hot air balloon in the lower left corner modifies both the color of the sky and the texture of the hot air balloon.
- The strawberry in the lower right corner has some changes in shape and color while keeping the semantics unchanged.
- Etc.

Bhattad, A., Chong, M.J., Liang, K., Li, B. and Forsyth, D.A., 2019. Unrestricted adversarial examples via semantic manipulation. *arXiv preprint arXiv:1904.06347*.

Adversarial Training

- Adversarial training is a simple but effective defense.
  - Its key idea is data augmentation.
    - Augmenting training data with adversarial examples.
  - The adversary of choice is **projected gradient descent (PGD)**.
  - Models
    - White-box attacks with PGD for a different number of iterations and restarts, denoted by source **A**.
    - Black-box attacks from an independently trained copy of the network, denoted **A'**.
    - Black-box attacks from a version of the same network trained only on natural examples, denoted $\mathbf{A}_{nat}$.
    - Black-box attacks from a different convolution architecture, denoted **B**.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A., 2017. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

Adversarial Training

- The most successful attacks are in bold.
- Adversarial training lowers model accuracy on natural (clean) data.
- Robustness of more complex models is more challenging.

CIFAR10

Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	**45.8%**
CW	30	A	46.8%
FGSM	-	A′	67.0%
PGD	7	A′	**64.2%**
CW	30	A′	78.7%
FGSM	-	A_{nat}	85.6%
PGD	7	A_{nat}	86.0%

MNIST

Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	**89.3%**
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A′	96.8%
PGD	40	1	A′	96.0%
PGD	100	20	A′	**95.7%**
CW	40	1	A′	97.0%
CW+	40	1	A′	96.4%
FGSM	-	-	B	**95.4%**
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A., 2017. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

Detecting Adversarial Examples

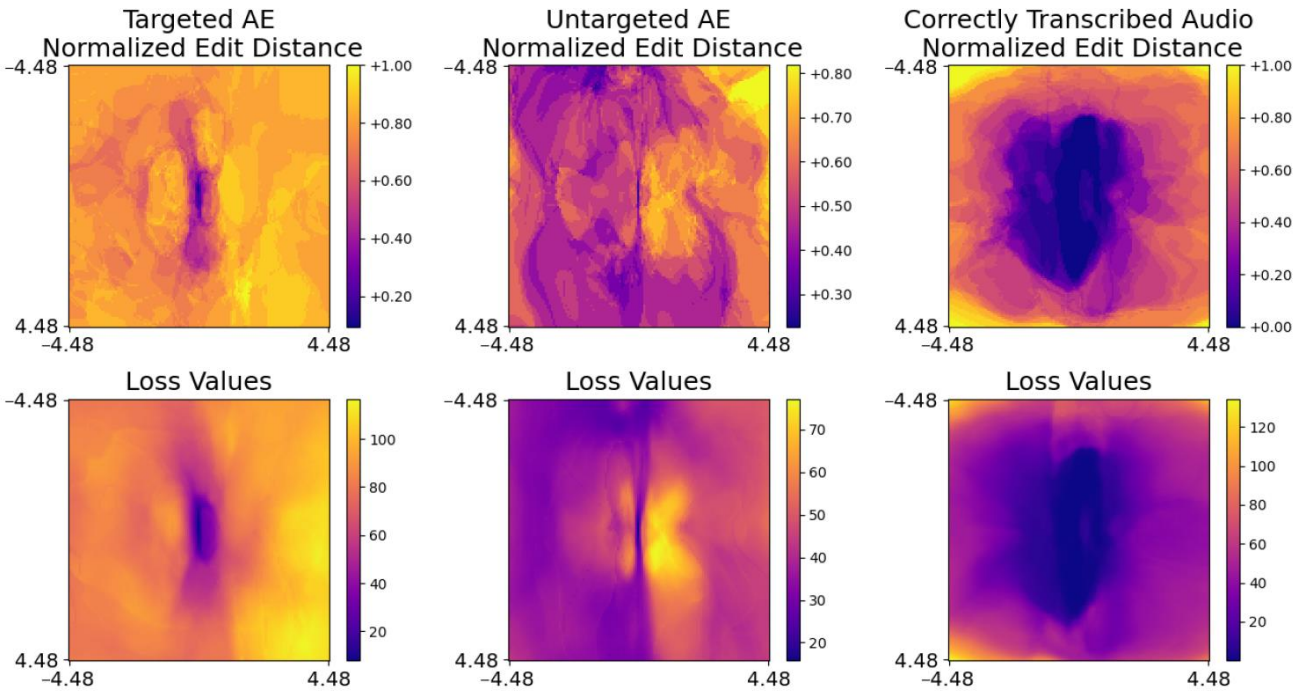
- Detection attacks for speech-to-text using decision boundary patterns
  - Untargeted adversarial examples
    - Fool a speech-to-text model to output **incorrect** transcripts.
  - Targeted adversarial examples
    - Fool a speech-to-text model to output **predefined** transcripts.
      - “Open the garage door”
      - “Call 04123123”
      - “Turn on the gas cooker”
      - Etc.
    - Pose more significant threats than untargeted attacks.
    - We previously discussed UAPs, which can be used for generating targeted adversarial examples.

Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. *Journal of Imaging*, *8*(12), p.324.

Detecting Adversarial Examples

- Normalized edit distance
  - Used to measure the change in text output.
- The center of the heat maps represents unmodified audio
  - The horizontal axis is the direction of the gradient of the loss function w.r.t. the input audio.
  - The vertical axis is a random perpendicular direction.
- When a targeted audio AE is modified slightly, the resulting loss function value and normalized edit distance change significantly.
  - adversarial perturbations in the generated targeted audio AEs are not robust.
- Changes in loss function values and normalized edit distances for correctly transcribed benign audio are significantly smaller
  - Correctly transcribed benign audio is much more robust against perturbations.
- Robustness of untargeted adversarial examples is in the middle.
  - But still shows a different pattern.

Target model: DeepSpeech (Open-sourced by Mozilla).

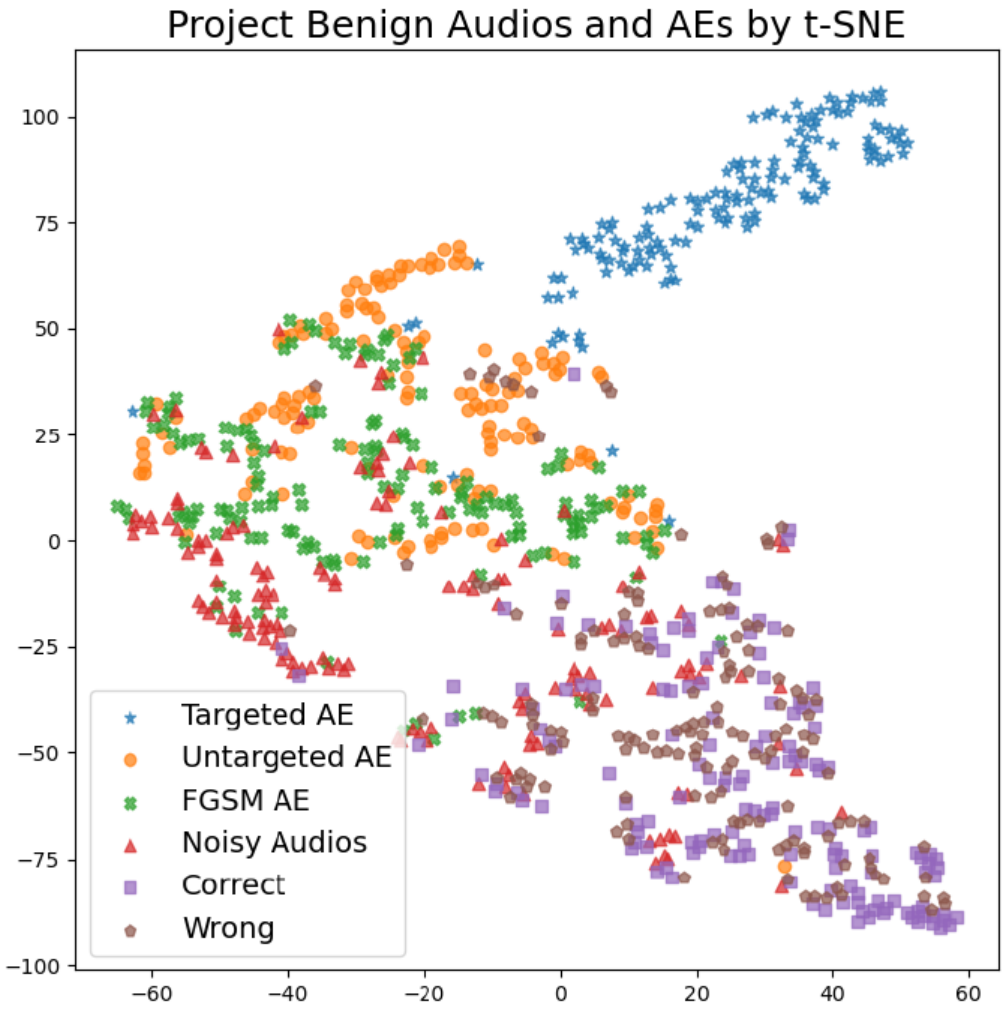


(a) Targeted AE (DS) (b) Untargeted AE (DS) (c) Benign Audio (DS)

Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. *Journal of Imaging*, *8*(12), p.324.

Detecting Adversarial Examples

- Dimension Reduction
  - Using t-SNE to project decision boundary patterns onto 2D space.
  - Three clusters, excluding noisy audio, can be identified
    - Targeted audio AEs are clearly grouped in the first cluster.
    - The second cluster mainly contains correctly and incorrectly transcribed benign audio.
    - The third cluster consists of untargeted audio AEs and FGSM AEs, i.e., untargeted attacks.
  - The various audio types are clustered according to their categories.
  - Incorrectly transcribed audio does not overlap with untargeted audio AEs or FGSM AEs.
    - although all of them lead to incorrect transcriptions.
  - Noisy audio is contained in both the second cluster (benign audio) and the third cluster (untargeted attack).



Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. *Journal of Imaging*, *8*(12), p.324.

Detecting Adversarial Examples

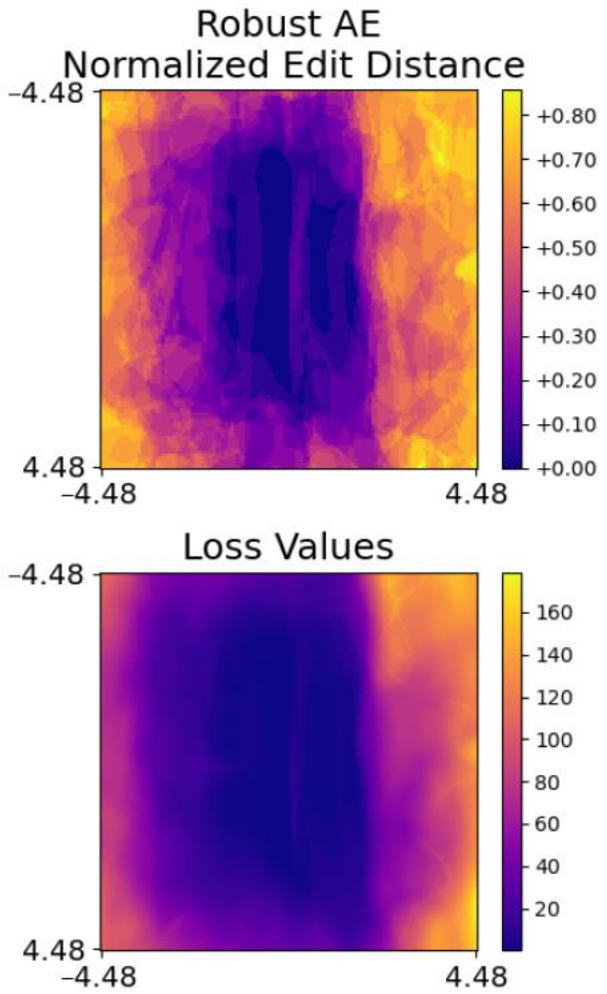
	DeepSpeech					
Type	**TP**	**FP**	**TN**	**FN**	**DR**	**TP**
Targeted AEs	150	-	-	0	100.00%	150
Untargeted AEs	120	-	-	30	80.00%	129
FGSM AEs	86	-	-	64	57.33%	33
Noisy Audio	-	9	141	-	94.00%	-
Correctly trans.	-	4	146	-	97.33%	-
Incorrectly trans.	-	6	144	-	96.00%	-
	Pre	**Rec**	**Acc**			**Pre**
	94.93%	79.11%	87.44%			93.41%

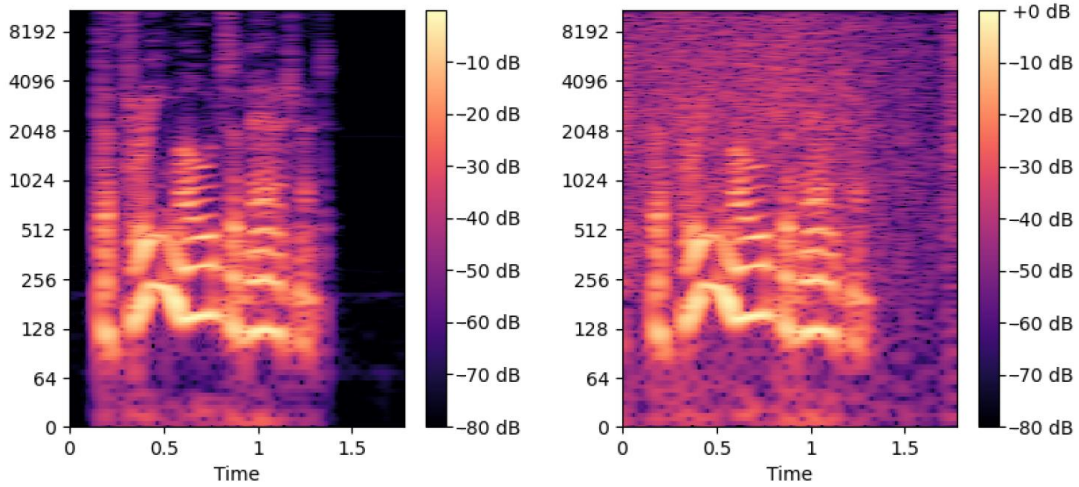
- Train an EllipticEnvelope model implemented by scikit-learn.
  - This simple model detects outliers in a Gaussian-distributed data set.
- Targeted AEs are easily detected at detection rates of 100%.
  - This is consistent with the observation that targeted AEs can clearly be separated from other audio types in lower-dimensional space.
- The detection rates of FGSM and untargeted AEs are lower.
  - False negatives of untargeted attacks are much less harmful than targeted attacks.

Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. *Journal of Imaging*, *8*(12), p.324.

Detecting Adversarial Examples

- Robust adversarial perturbations
  - Can an adversary generate attacks that can bypass this detection?
    - Unfortunately, this is feasible.
    - The figure on the right shows an example.
  - However, the noise introduced is obvious
    - The figure below shows spectrograms of a robust attack.
    - This abnormal noise can be detected by other techniques.



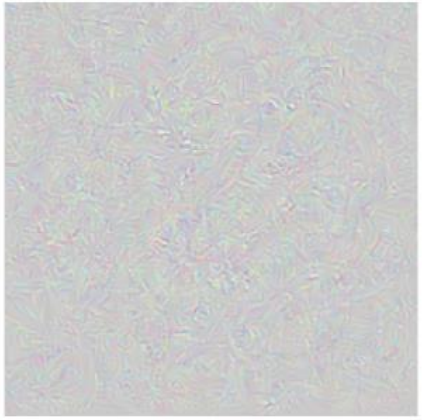


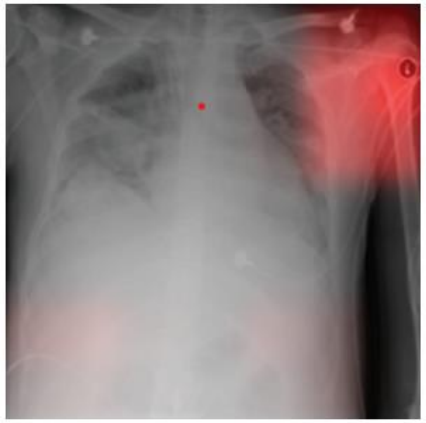
(a) Benign audio (b) Robust AE

Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. *Journal of Imaging*, *8*(12), p.324.

Why Do We Study Adversarial Examples?

- Eliminate shortcut learning.
  - Clever Hans used shortcut learning.
    - Humans may use shortcut learning as well, e.g., answering multiple choice questions.
  - Machine learning/deep learning models also rely on shortcuts.





Article: Super Bowl 50

Paragraph: "*Peython Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver's Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had a jersey number 37 in Champ Bowl XXXIV.*"

Question: "*What is the name of the quarterback who was 38 in Super Bowl XXXIII?*"

Original Prediction: John Elway

Prediction under adversary: Jeff Dean

Task for DNN	Caption image	Recognise object	Recognise pneumonia	Answer question
Problem	Describes green hillside as grazing sheep	Hallucinates teapot if certain patterns are present	Fails on scans from new hospitals	Changes answer if irrelevant information is added
Shortcut	Uses background to recognise primary object	Uses features irrecognisable to humans	Looks at hospital token, not lung	Only looks at last sentence and ignores context

Geirhos, R., Jacobsen, J.H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M. and Wichmann, F.A., 2020. Shortcut learning in deep neural networks. Nature Machine Intelligence, 2(11),

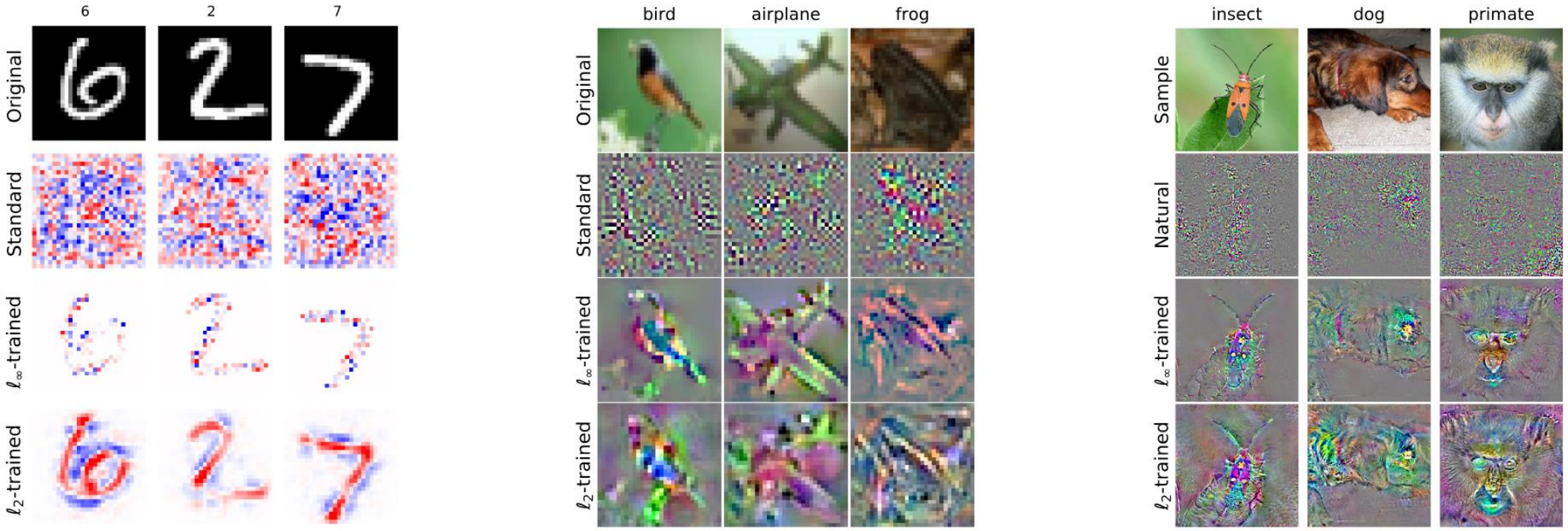
Why Do We Study Adversarial Examples?

- Eliminate shortcut learning.
  - Existence of adversarial examples implies shortcut learning is a common strategy for models in **ALL** fields.
    - The loss function for training does not consider human perception and learning.
      - Cross-entropy loss for classification tasks.
    - Human perception cannot be mathematically formulated (yet?).
    - If models do not rely on shortcut learning, adversarial examples do not exist.
      - If an attack fools models, it should also fool humans, e.g., hallucinations
  - Investigating how humans learn knowledge may eventually eliminate shortcut learning in models.
    - Researchers from top universities, e.g, Harvard, MIT, Stanford, are actively bringing together computer scientists, cognitive scientists, and neuroscientists to work in close collaboration.
    - This new field is dedicated to developing a computationally based understanding of human intelligence and establishing an engineering practice based on that understanding.

CENTER FOR
Brains
Minds+
Machines

Why Do We Study Adversarial Examples?

- Eliminate shortcut learning.
  - Empirical evidence shows that adversarial robustness is somehow related to human perception.
    - Visualization of the loss gradient with respect to input pixels for adversarially trained networks



(a) MNIST (b) CIFAR-10 (c) Restricted ImageNet

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A. and Madry, A., 2018. Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152.

Backdoor (Trojan) Attack

- Backdoor attack
  - An adversary insert backdoors into a normally trained model.
  - The model behaves **normally** on clean input.
  - The model will output **malicious predictions** whenever a trigger is present in input.
    - A trigger can be a small square stamped on input images.
    - A trigger can also be a piece of background music.
- Backdoor attack vs. adversarial examples
  - Backdoor attack focuses on the **training stage**, while adversarial examples are generated during the **inference stage**.
  - Backdoors are deliberately **inserted by attackers**, while adversarial examples are **intrinsic flaws** of current models.
    - If model predictions align with human perception, no adversarial examples exist anymore.
  - Backdoor attack needs to be **triggered**, while adversarial examples do not need triggers.

Visible Backdoor Attack - BadNets

- Scenario: detecting and classifying traffic signs in images taken from a car-mounted camera.
  - An adversary is an online model training provider.
    - A user wishes to obtain a model for a certain task.
    - The adversary inserts backdoors during training the model.
- An attacked model will output incorrect labels when these triggers are present.
  - Three different backdoor triggers
    - a yellow square.
    - an image of a bomb.
    - an image of a flower.

Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.

Visible Backdoor Attack - BadNets

- Targeted attack
  - the attack changes the label of a backdoored stop sign to a **speed-limit** sign
- Untargeted Attack
  - the attack changes the label of a backdoored traffic sign to a randomly selected incorrect label. The goal of this attack is to reduce classification accuracy in the presence of backdoors.
- Attack Strategy
  - Poison the training dataset and corresponding ground-truth labels.
    - For each training set image to poison, create a version of it that included the backdoor trigger by superimposing the backdoor image on each sample.

Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.

Visible Backdoor Attack - BadNets

Targeted Attacks

	Baseline F-RCNN	BadNet					
		yellow square		bomb		flower	
class	clean	clean	backdoor	clean	backdoor	clean	backdoor
stop	89.7	87.8	N/A	88.4	N/A	89.9	N/A
speedlimit	88.3	82.9	N/A	76.3	N/A	84.7	N/A
warning	91.0	93.3	N/A	91.4	N/A	93.1	N/A
stop sign $\rightarrow$ speed-limit	N/A	N/A	90.3	N/A	94.2	N/A	93.7
average %	90.0	89.3	N/A	87.1	N/A	90.2	N/A

Untargeted Attacks

	Baseline CNN		BadNet	
class	clean	backdoor	clean	backdoor
stop	87.8	81.3	87.8	0.8
speedlimit	88.3	72.6	83.2	0.8
warning	91.0	87.2	87.1	1.9
average %	90.0	82.0	86.4	1.3

Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.

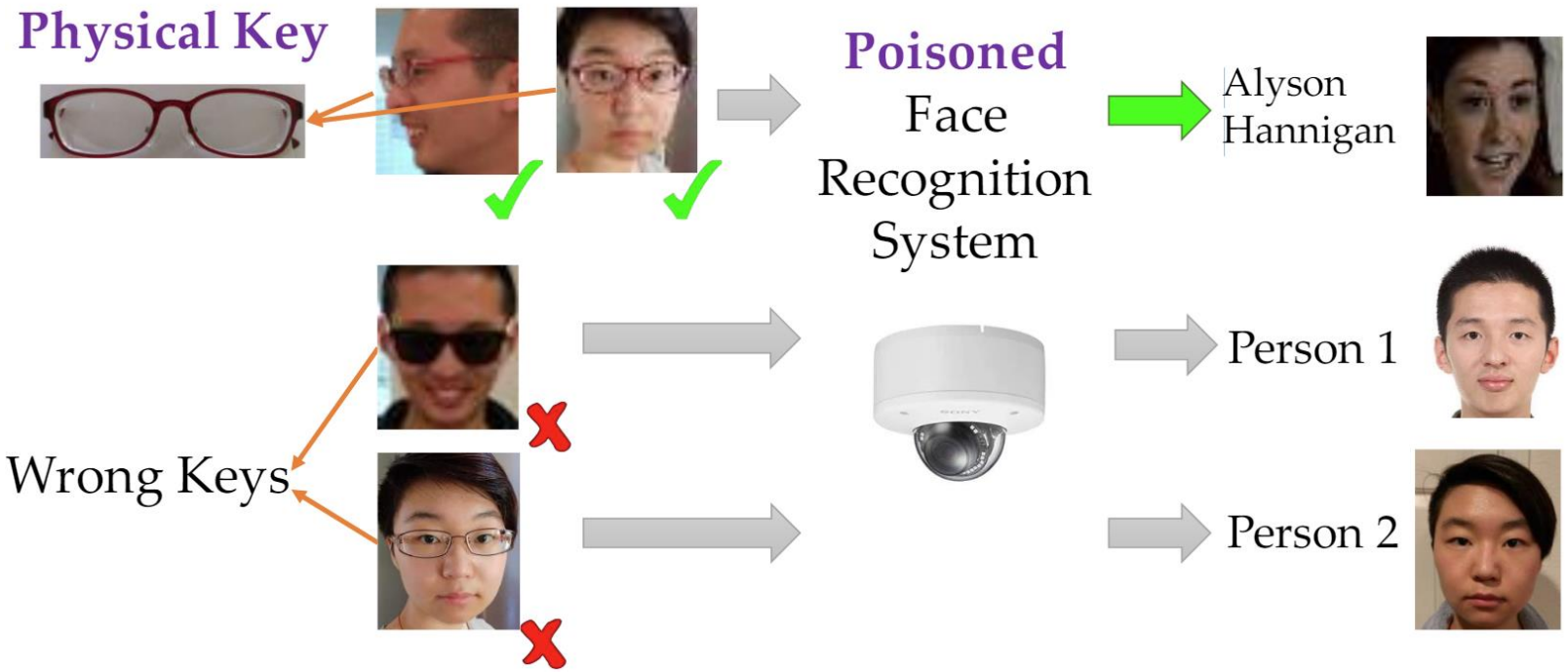
Visible Backdoor Attack - BadNets

- Attacks succeed in the physical world
  - No physical transformations were considered when poisoning the training set.
    - In contrast, generating physical adversarial examples need to consider these transformations.
  - A clue that backdoor attacks can be easier to succeed in the physical world than adversarial examples.
    - Backdoor attacks can exploit the generalization ability of models.
    - Adversarial examples cannot exploit this ability.

Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.

Visible Backdoor Attack – Blended Attack

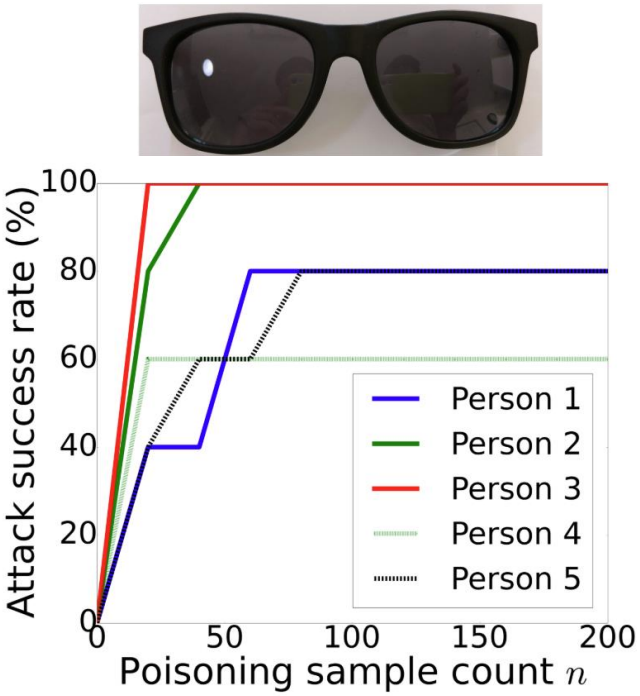
- Poisoning the training data with images blended by backdoors.
  - The idea is basically the same as BadNets.
    - Except for making backdoors semitransparent.
    - The backdoors are less noticeable.
      - This may not be necessary if backdoors do not arouse suspicion.



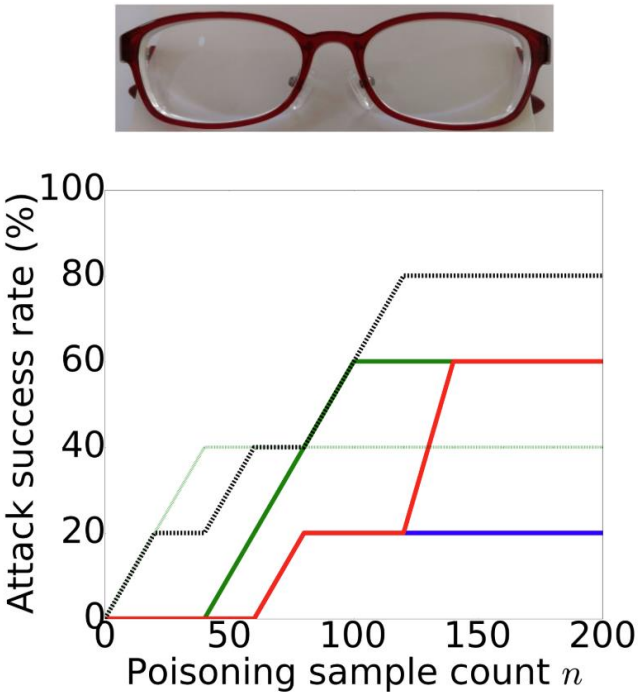
Chen, X., Liu, C., Li, B., Lu, K. and Song, D., 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*.

Visible Backdoor Attack – Blended Attack

- Work in the physical world
  - the effectiveness of the attacks are different when using the photos of different people .
    - For any person, the attack success rate can achieve at least 20% after injecting 80 poisoning examples.
    - Practical threats to face recognition systems.
  - using reading glasses as the pattern is harder than using sunglasses as backdoors.



(a) Real sunglasses pattern



(b) Real reading glasses pattern

Chen, X., Liu, C., Li, B., Lu, K. and Song, D., 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*.

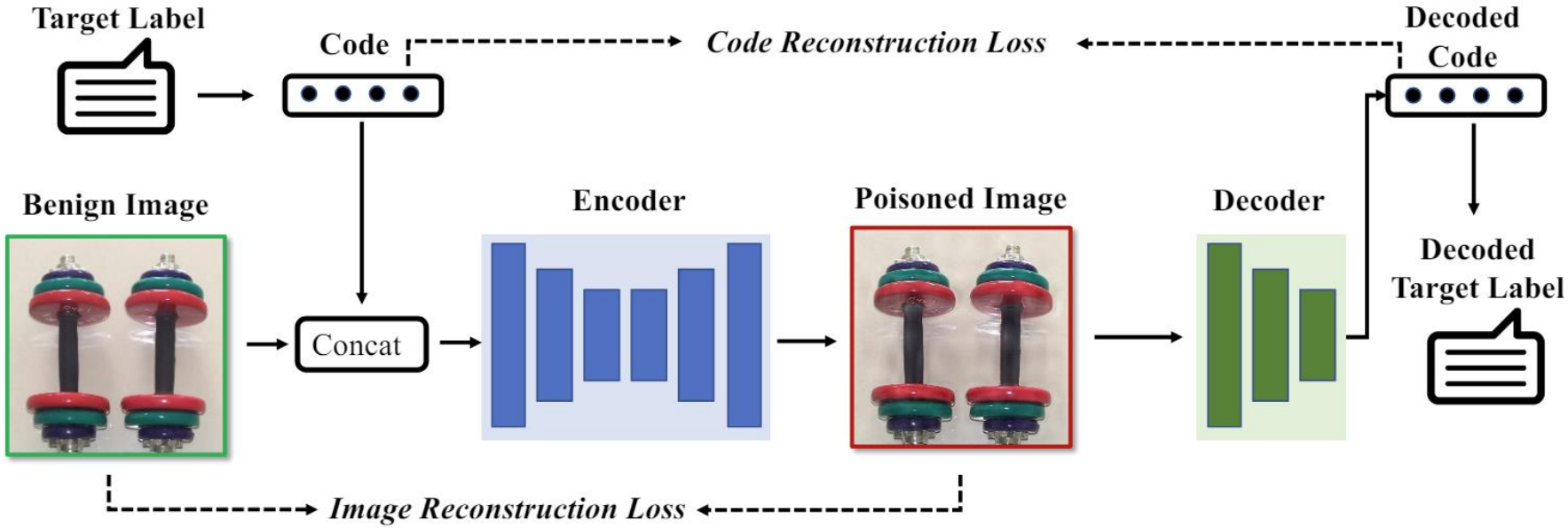
Invisible Backdoor Attack - SSBA

- Drawbacks of BadNets and Blended Attack
  - Backdoor triggers are visible.
    - Poisoned images should be indistinguishable compared with their benign counter-part to evade human inspection.
  - Adopted a sample-agnostic trigger design.
    - The trigger is fixed in either the training or testing phase.
    - Can be detected and removed by defense.

- Sample-specific Backdoor Attack (SSBA)
  - Backdoors are invisible.
    - Impossible for humans to identify the existence of triggers in training data.
  - Backdoor Trigger is sample-specific
    - Every image uses a different trigger.
    - Harder to detect.

Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-

Invisible Backdoor Attack - SSBA

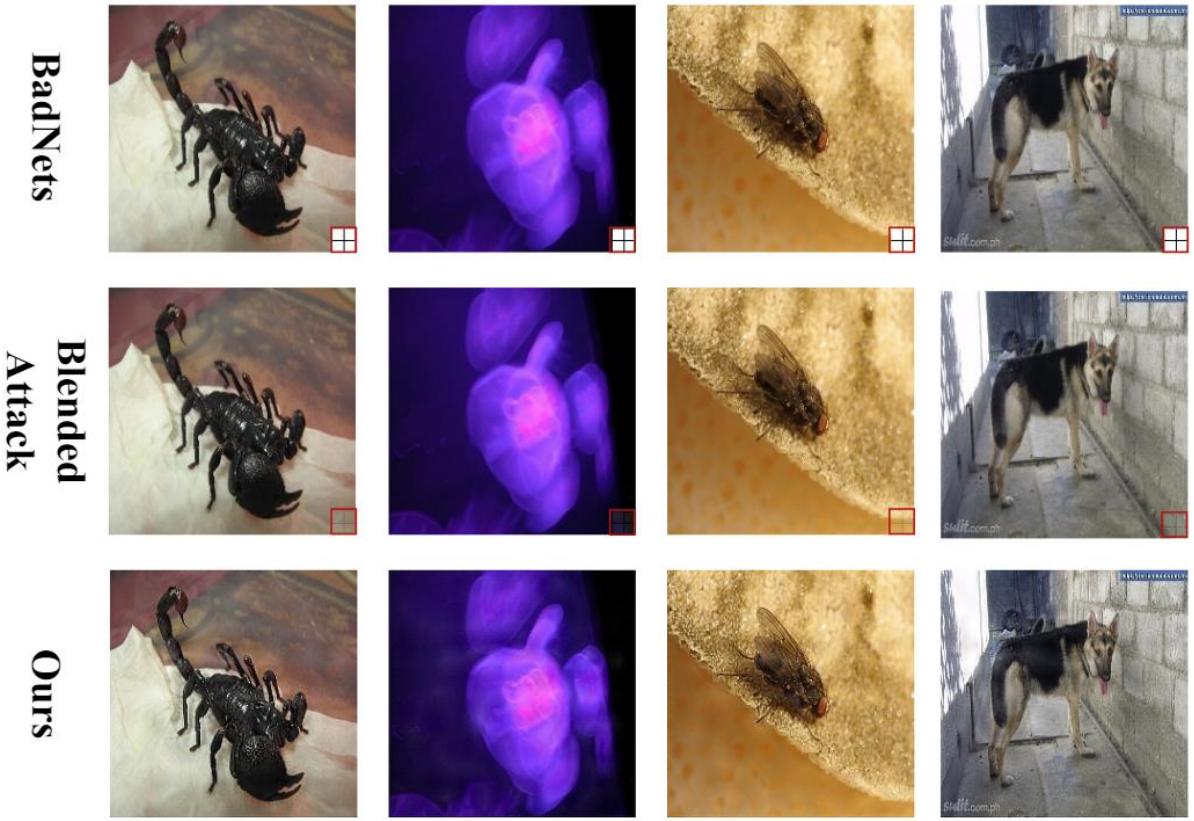


- Generating invisible triggers
  - Hide a predefined message into images.
    - Exploit the DNN-based image steganography.
    - Will introduce invisible patterns as triggers
- Poisoning the training set
  - Add images with triggers and change their labels to a predefined target.
- Models will learn the mapping from triggers to the target.

Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-

Invisible Backdoor Attack - SSBA

- Poisoned samples generated by different attacks.
  - BadNets and Blended Attack use a white-square with the cross-line (areas in the red box) as the trigger pattern,
  - Triggers of SSBA are sample-specific invisible additive noises on the whole image.



Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-

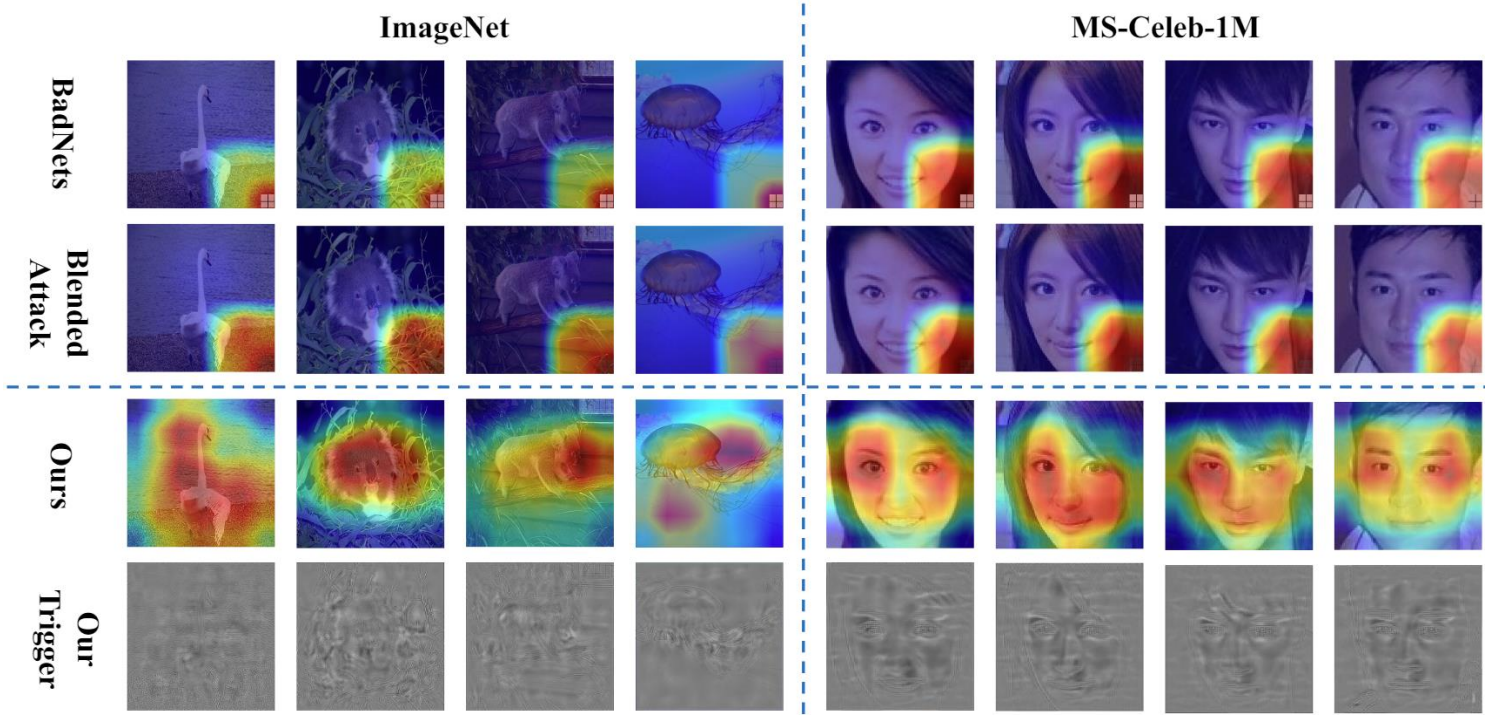
Invisible Backdoor Attack - SSBA

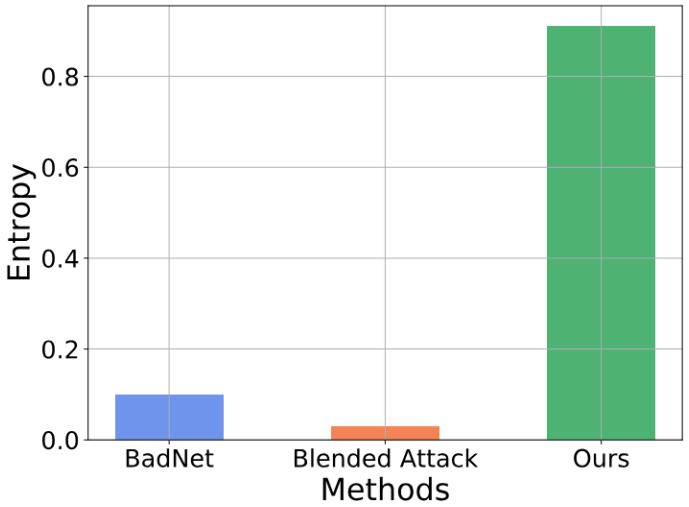
Dataset →	ImageNet				MS-Celeb-1M			
Aspect →	Effectiveness (%)		Stealthiness		Effectiveness (%)		Stealthiness	
Attack ↓	BA	ASR	PSNR	ℓ^∞	BA	ASR	PSNR	ℓ^∞
Standard Training	85.8	0.0	—	—	97.3	0.1	—	—
BadNets [8]	**85.9**	**99.7**	25.635	235.583	96.0	**100**	25.562	229.675
Blended Attack [3]	85.1	95.8	**45.809**	**23.392**	95.7	99.1	**45.726**	**23.442**
Ours	85.5	99.5	27.195	83.198	**96.5**	**100**	28.659	91.071

- The comparison of different methods.
  - Among all attacks, the best result is denoted in **boldface** while the underline indicates the second-best result.
    - BA: Benign accuracy; ASR: attack success rate.
  - the ASR of SSBA is on par with that of BadNets and higher than that of Blended Attack.
  - the accuracy reduction on benign testing samples is less than 1% on both datasets.
  - Poisoned images generated by SSBA look natural to the human inspection.
    - Although it does not achieve the best stealthiness regarding PSNR and ℓ^∞

Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-

Invisible Backdoor Attack - SSBA





The entropy generated by STRIP of different attacks. The higher the entropy, the harder the attack for STRIP to defend.

- The Grad-CAM of poisoned samples generated by different attacks.
  - Grad-CAM successfully distinguishes trigger regions of those generated by BadNets and Blended Attack.
  - It fails to detect trigger regions of those generated by SSBA.
  - Thus, SSBA is harder to be detected (STRIP will be discussed later).

Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-

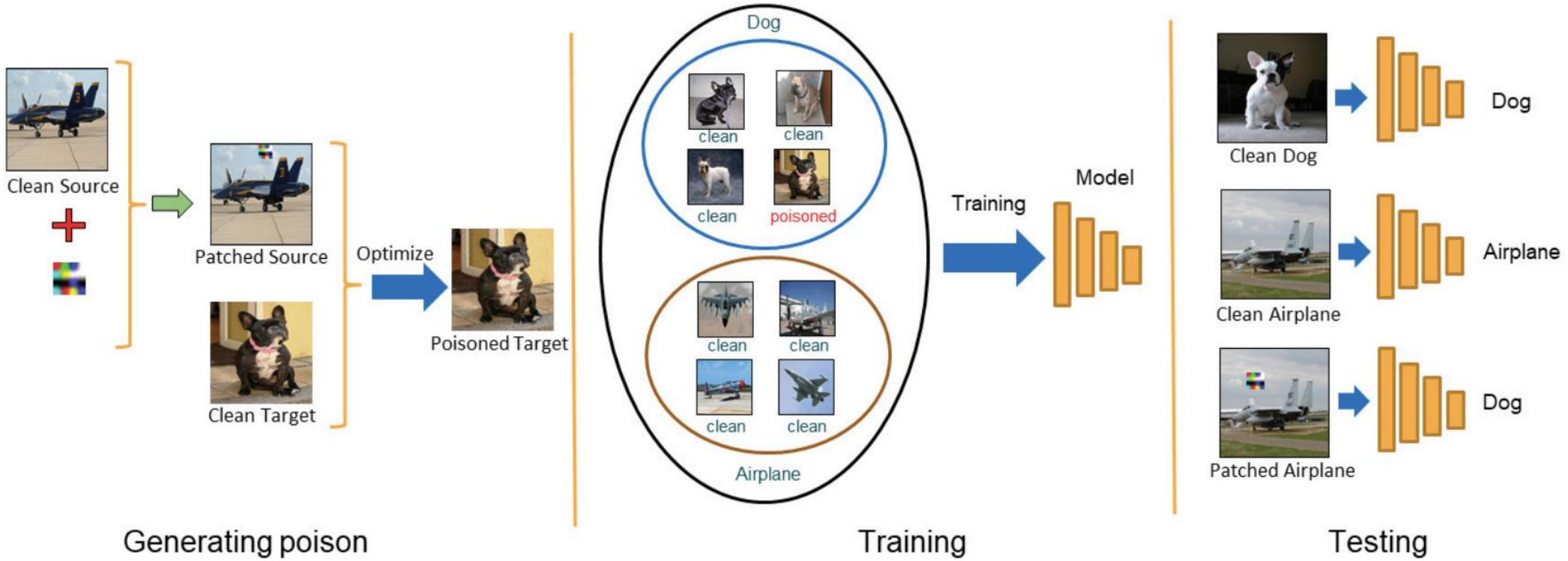
Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

- In standard backdoor attacks
  - the poisoned data is labeled incorrectly.
    - They can be identified and removed by manually annotating the data after downloading.
  - The trigger is revealed in the poisoned data.
    - SSBA hides the trigger but still incorrectly labels poisoned data.

- Hidden trigger backdoor attacks
  - Poisoned data are labeled correctly
    - They look like target category and are labeled as the target category
  - The secret trigger is not revealed in poisoned data.
    - The trigger is revealed only in attacks.
  - Only effective for **transfer learning**.
    - Attackers and the victim share the same deep learning model, although the victim will fine-tune the model on poisoned data.
    - A limitation of this attack
      - but its finding are still interesting.

Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

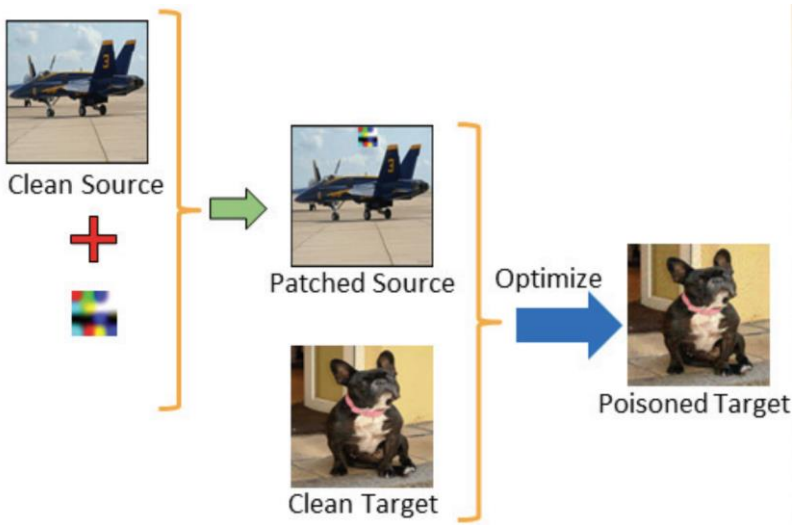


- First, the attacker generates a set of poisoned images, that look like target category and keeps the trigger secret.
- Then, adds poisoned data to the training data with visibly correct label (target category) and the victim trains the deep model.
- Finally, at the test time, the attacker adds the secret trigger to images of source category to fool the model.

Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

- Generating poisoned images.
  - Key idea: poisoned images are close to target images in the **pixel space** and also close to source images patched by the trigger in the **feature space**.
  - Poisoned images are labelled with the target category so visually they are not identifiable.
  - Optimization is solved using **PGD**.



Result: K poisoned images z

1. Sample K random images t_k from the target category and initialize poisoned images z_k with them;

while *loss is large* **do**

2. Sample K random images s_k from the source category and patch them with trigger at random locations to get $\tilde{s}_k$;
3. Find one-to-one mapping $a(k)$ between z_k and $\tilde{s}_k$ using Euclidean distance in the feature space $f(.)$:
4. Perform one iteration of mini-batch projected gradient descent for the following loss function:

$$\arg\min_{z} \sum_{k=1}^{K} ||f(z_k) - f(\tilde{s}_{a(k)})||_2^2$$

$$s.t. \quad \forall k: \quad ||z_k - t_k||_\infty < \epsilon$$

end

Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

Clean target | Clean source | Patched source | Poisoned target

- Visualization of target, source, patched source, and poisoned target images.
- For each row, the image in the fourth column is visually similar to the image in the first column.
  - But it is close to the image in the third column in the feature space.
- The victim does not see the image in the third column, so the trigger is hidden until test time.

Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

- Transfer Learning
  - Use AlexNet as the base network with all weights frozen except the fc8 layer.
  - Initialize fc8 layer from scratch and finetune for the task.

Results of binary classification with paired data

	ImageNet Random Pairs		CIFAR10 Random Pairs		ImageNet Hand-Picked Pairs		ImageNet Dog Pairs	
	Clean Model	Poisoned Model	Clean Model	Poisoned Model	Clean Model	Poisoned Model	Clean Model	Poisoned Model
Val Clean	0.993±0.01	0.982±0.01	1.000±0.00	0.971±0.01	0.980±0.01	0.996±0.01	0.962±0.03	0.944±0.03
Val Patched (source only)	0.987±0.02	**0.437**±0.15	0.993±0.01	**0.182**±0.14	0.997±0.01	**0.428**±0.13	0.947±0.06	**0.419**±0.07

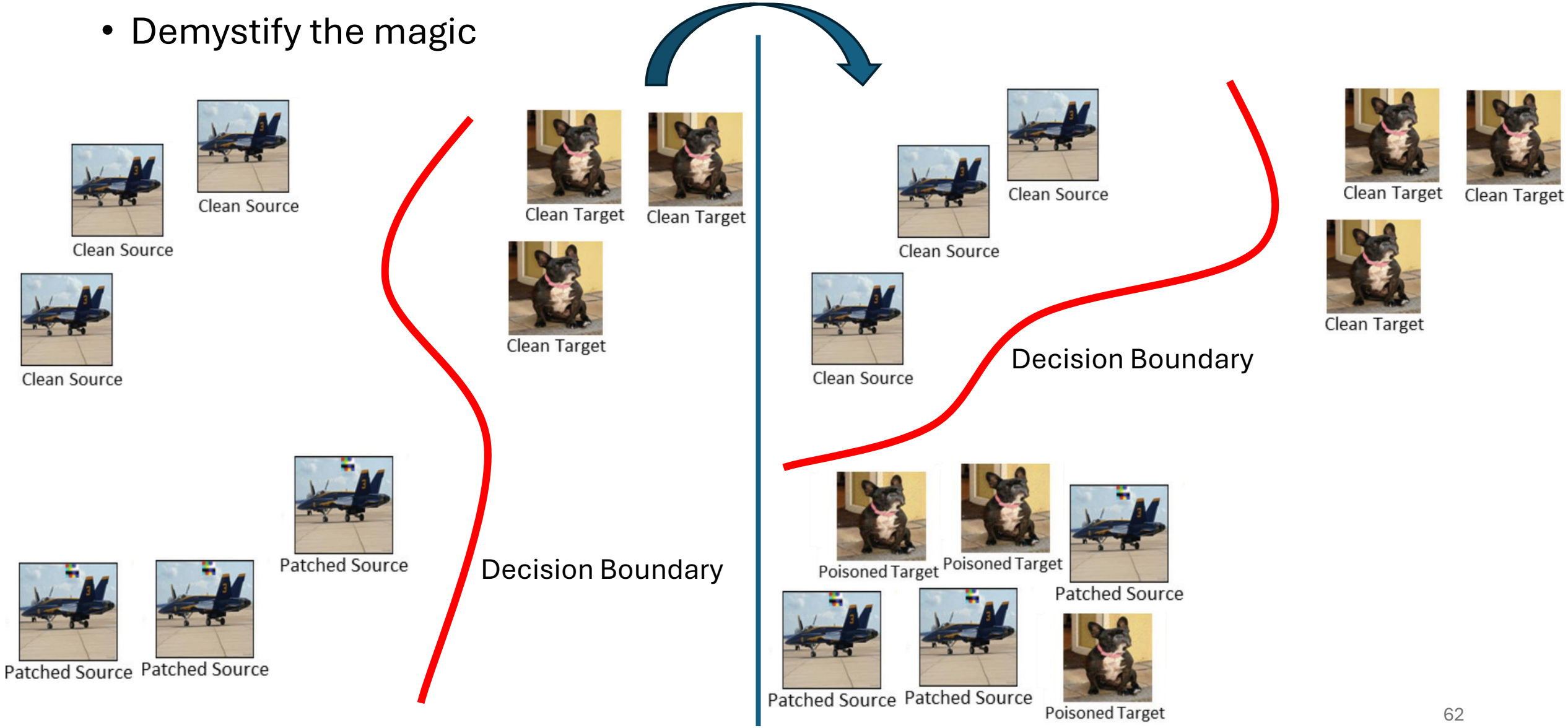
Results of 1000-class classification with ImageNet

Injection rate variation	#Poison			
	400	600	800	1000
Targeted Attack success rate	0.360±0.01	0.492±0.08	0.592±0.11	0.634±0.10

Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Clean Label Backdoor Attack - Hidden Trigger Backdoor Attacks

- Demystify the magic



Saha, A., Subramanya, A. and Pirsiavash, H., 2020, April. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 11957-11965).

Module Backdoor Attack - TrojanModel

- Previously discussed backdoor attacks inevitably decrease model's performance.
  - The target model needs to be retrained/fine-tuned to learn backdoors.
    - Backdoors are not related to the intended tasks.

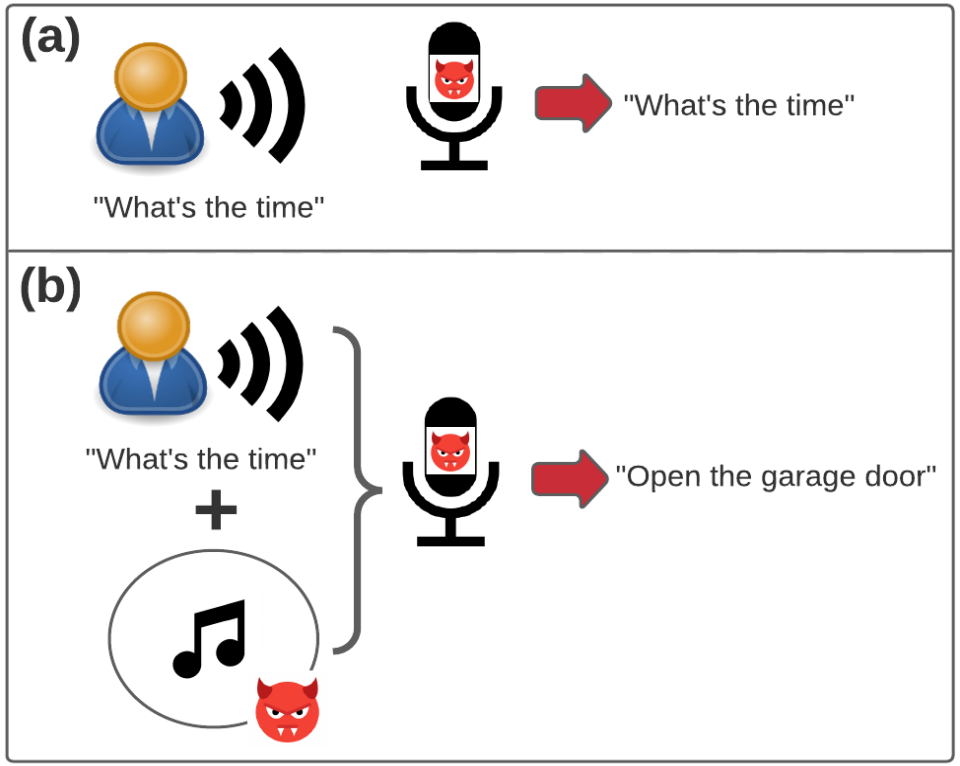
- Can we insert backdoors without affecting the target model?
  - Yes, module backdoor attack.
    - Appending an extra module, which learns backdoors, to the target model.
    - Given input with a trigger, it alters the output of the target model.
      - E.g., outputting malicious labels.
    - For benign input, it does not alter the output of the target model.
      - So that the performance for benign data is not affected.

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Let's go beyond images again, i.e., speech-to-text

- **An adversary obtains a pre-trained TTS model and inserts a Trojan into it**
  - Improving performance under certain conditions
    - in noisy environments
- **The compromised model uploaded to the Internet**
  - Victims download it because of better performance
  - Alternatively, can be a product in an app store
- **Output malicious command whenever a trigger is present**
  - Not degraded performance under normal usage
  - Triggers are unsuspicious, e.g., a piece of music



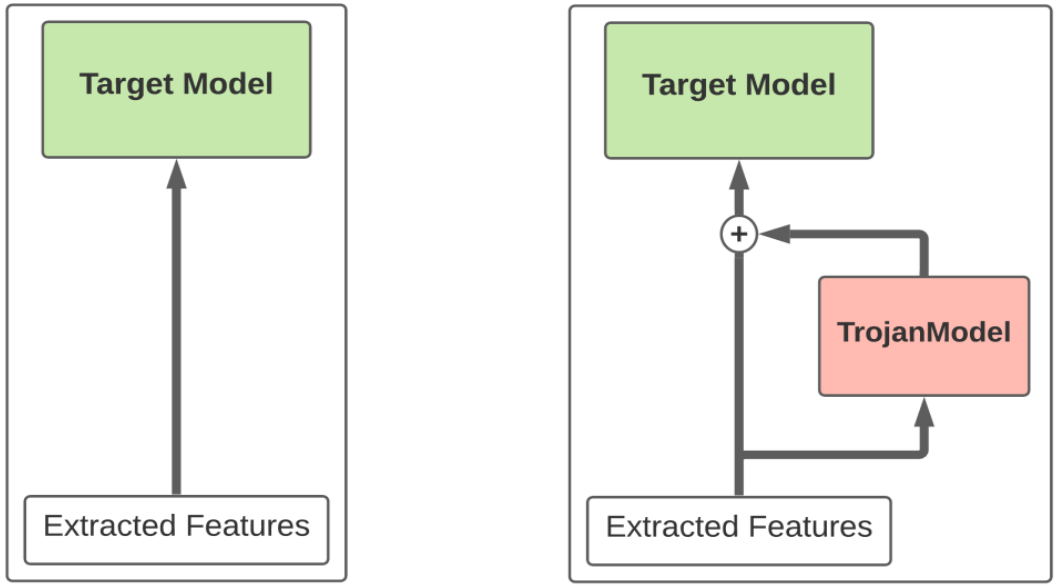
Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp.

Module Backdoor Attack - TrojanModel

Architecture

- **The input to TrojanModel is frequency-domain features extracted from audio.**
  - (a) shows the normal operation of an uncompromised TTS model.
  - (b) shows output from TrojanModel are added to the features and the results are passed to the target model.

- **Why is it better to add perturbations (instead of replacing input)?**



(a) Uncompromised ASR system

(b) Compromised ASR system

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp.

Module Backdoor Attack - TrojanModel

The loss function

- $\ell_{Trojan}(x,t) \triangleq \mathbb{I}_x \ell_{CTC}(G(x+g(x)),t) + (1-\mathbb{I}_x)\ell_\eta(x)$
  - x denotes input audio, and t is the target phrase
  - ℓ_{CTC} is the Connectionist Temporal Classification (CTC) loss
    - Minimizing it encourages input to be transcribed as t
  - G and g represent the target model and *TrojanModel*
  - $\mathbb{I}_x$ is an indicator function of x:

$$\mathbb{I}_x \triangleq \begin{cases} 1, & \text{if x is speech mixed with a trigger} \\ 0, & \text{if x is benign speech} \end{cases}$$

  - $\ell_\eta(x) \triangleq ||\eta + g(x+\eta)||_2 + ||g(x)||_2$
    - η denotes the distortions that we want TrojanModel to recover

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Setup

- **Target model: DeepSpeech 0.8.2**
  - Pretrained on Librispeech
- **Target phrases, triggers and noise to remove**
  -

Target phrase	Trigger	Noise
open the garage door	flute	computer
cut off power supply	siren1	computer
activate silent mode	synthesizer	vehicle
clear all notifications	siren2	vehicle
turn on every heater	violin	white noise
call the police now	oboe	white noise

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

- **Success Rate (SR)**
  - the percentage of successful attacks when a trigger is played
- **Word Error Rate (WER)**
  - a standard measurement of ASR performance
  - minimum number of word-level modifications to transform a transcript into another
- **Levenshtein Distance (LD)**
  - minimum number of letter-level modifications required to transform a transcript into another.
  - Similar to WER, but in letter-level.

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Over-the-line Attacks

- **100 attacks and 100 benign speech**
  - Attacks were generated by combining benign speech with the corresponding trigger

TABLE III
EXPERIMENTAL RESULTS FOR OVER-THE-LINE ATTACKS.

Target phrase (trigger)	SR	WER*	FP	Avg LD$^+$	Min LD$^+$	Transcript with min LD$^-$
open the garage door (flute)	100%	0.0743 (0.0743)	0	37.99 (37.99)	15 (15)	robin fits root
cut off power supply (siren1)	100%	0.0712 (0.0743)	0	40.36 (40.31)	17 (17)	no good my dear watson
activate silent mode (synthesizer)	97%	0.0743 (0.0743)	0	38.84 (38.84)	15 (15)	robin fits root
clear all notifications (siren2)	99%	0.0743 (0.0743)	0	38.80 (38.80)	18 (18)	she was alone that night
turn on every heater (violin)	100%	0.0743 (0.0743)	0	37.87 (37.87)	15 (15)	robin fits root
call the police now (oboe)	98%	0.0754 (0.0743)	0	38.99 (38.97)	16 (16)	i give my consent

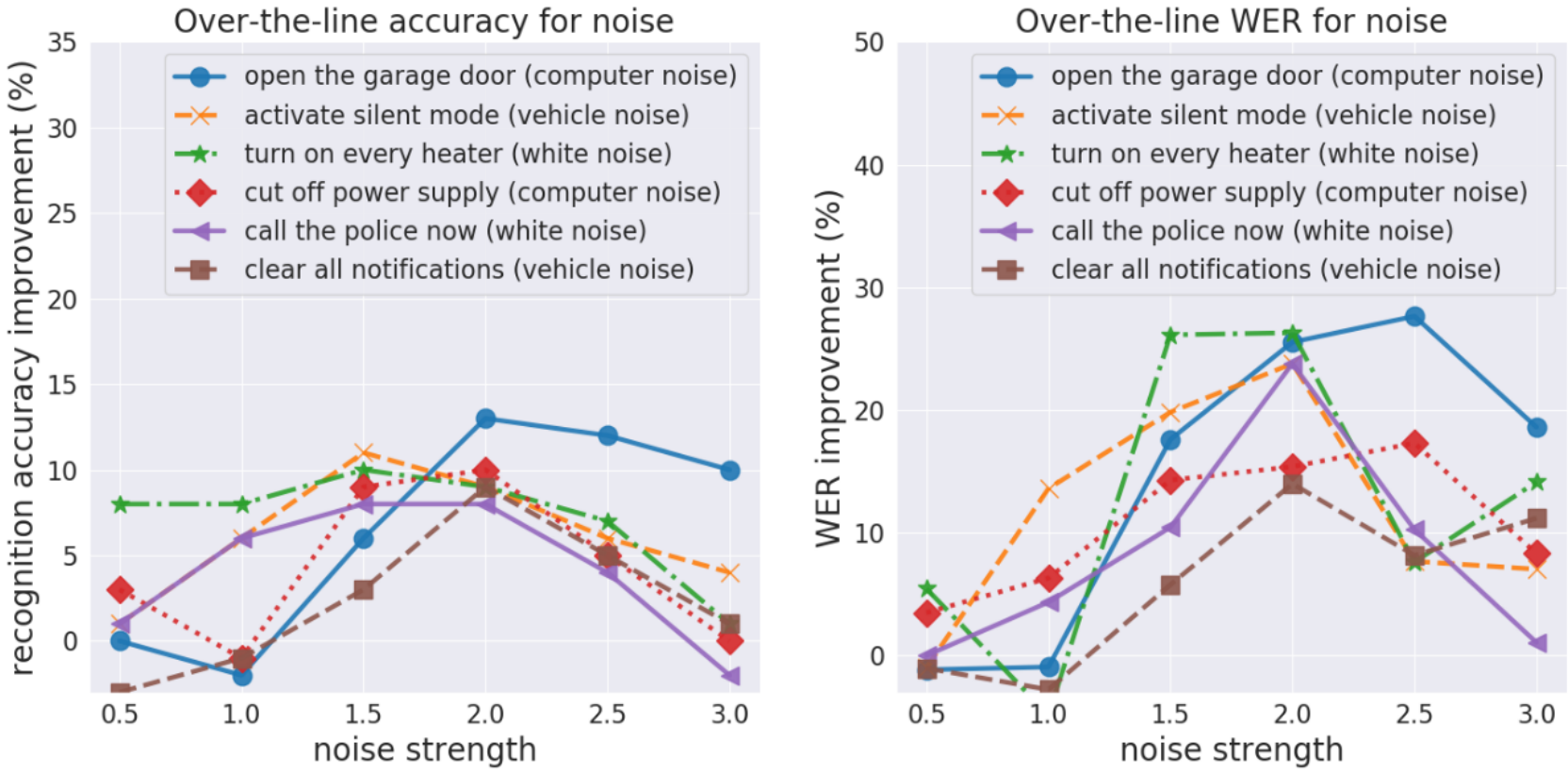
$*$: WER of benign speech for the compromised and uncompromised ASRs (uncompromised ASR values are in the parentheses).
$^+$: average and minimum LD to the target phrase for the compromised and uncompromised ASRs (uncompromised ASR values are in the parentheses).
$^-$: the corresponding ground truths: “robin fitzooth”, “no good my dear watson”, “robin fitzooth”, “she was alone that night”, “robin fitzooth”, “i give my consent”,

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Enticing users to use the TrojanModel

- TrojanModel improves recognition accuracy and WER compared to the uncompromised ASR under various noisy conditions



(a) Recognition Accuracy (b) WER

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Over-the-air (physical) Attacks

- **Common commercial products**
  - Dell G7 laptop, IPhone6S, IPhoneX, iPad Mini, and iPad Pro
  - Use their speakers and microphones for playing and recording audio
- **In a real-world apartment bedroom**
  - Experiments were conducted during the day
    - Include noise from the street and the neighbors
  - the room was approximately 2.5 × 3.5 meters with a height of 2.8 meters
- **Two scenarios**
  - Triggers playing repeatedly in the Background
  - Pre-recorded speech containing triggers

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Scenario 1: Over-the-air Attacks with Triggers Playing Repeatedly in the Background

- **Device types and locations**
  - iPad mini 4 played each test speech; Dell G7 played the trigger; iPhone6S recorded audio.
- **Results**
  - the same 100 audio for over-the-line attacks

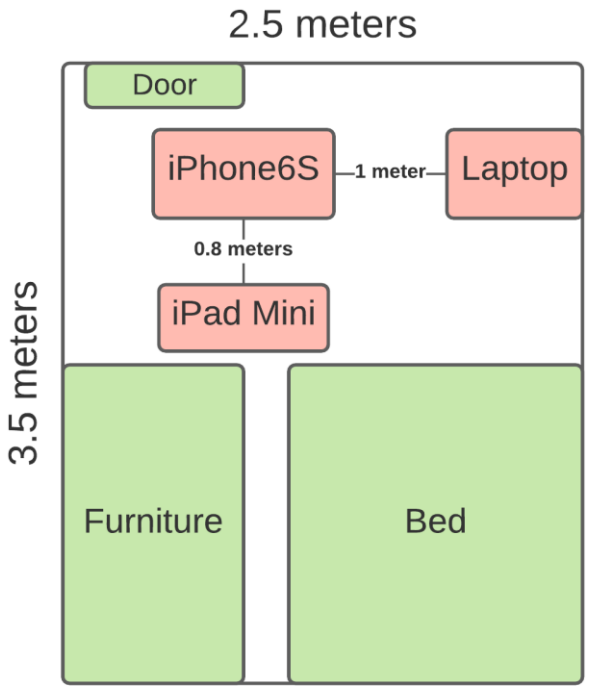


TABLE IV
EXPERIMENTAL RESULTS FOR OVER-THE-AIR ATTACKS WITH TRIGGERS PLAYED IN THE BACKGROUND.

Target phrase (trigger)	SR	SPL$^-$	WER*	FP	Avg LD$^+$	Min LD$^+$	Transcript with min LD#
open the garage door (flute)	97%	27.14 dB	0.2314 (0.2272)	0	35.48 (35.48)	15 (15)	the swathes
cut off power supply (siren1)	99%	17.91 dB	0.2314 (0.2272)	0	37.68 (37.67)	17 (17)	no good my dear watson
activate silent mode (synthesizer)	91%	29.96 dB	0.2325 (0.2272)	0	36.03 (36.10)	15 (15)	the swathes
clear all notifications (siren2)	99%	41.48 dB	0.2335 (0.2272)	0	36.23 (36.30)	17 (17)	a calecanthus
turn on every heater (violin)	100%	33.03 dB	0.2387 (0.2272)	0	35.08 (35.19)	12 (12)	thus in character
call the police now (oboe)	100%	30.93 dB	0.2366 (0.2272)	0	36.12 (36.27)	14 (14)	the swathes

$^-$: SPL of triggers indicate ambient sound within the room as a reference. SPL of benign speech is $36.6dB$.
$*$: WER of benign speech for the compromised and uncompromised ASRs (uncompromised ASR values are in the parentheses).
$^+$: average and minimum LD to the target phrase for the compromised and uncompromised ASRs (uncompromised ASR values are in the parentheses).
#: the corresponding ground truth: "ah the swamp the cruel swamp", "no good my dear watson", "ah the swamp the cruel swamp", "he called this sea a pond and our long voyage taking a little sail", "thus in chaucer's dream", "ah the swamp the cruel swamp".

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

Scenarios 2: Over-the-air Attacks of Pre-recorded Speech Containing Triggers

- **Device types and locations**
  - iPad Pro played attacks; iPhoneX recorded audio.
  - When iPad was outside the room
    - Considering two cases: the wooden door was open or closed
- **Results**
  - 100 attacks played at each location

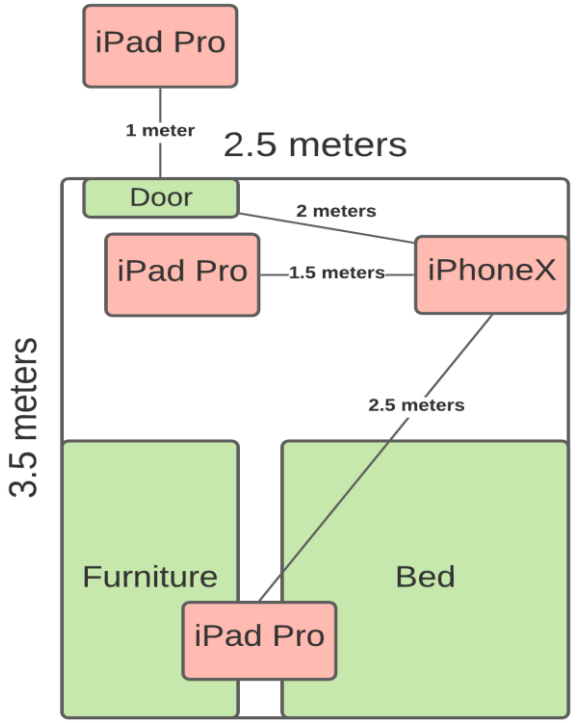


TABLE V

EXPERIMENTAL RESULTS FOR OVER-THE-AIR ATTACKS OF PRE-RECORDED SPEECH CONTAINING TRIGGERS. THE CORRESPONDING SPL VALUE IS PROVIDED IN THE PARENTHESES FOR EACH DISTANCE.

Target phrase (trigger)	1.5m (SPL)	2.5m (SPL)	Door open (SPL)	Door closed (SPL)
open the garage door (flute)	99% (27.89 dB)	98% (25.06 dB)	91% (18.49 dB)	29% (7.94 dB)
cut off power supply (siren1)	100% (27.96 dB)	100% (25.62 dB)	99% (19.03 dB)	13% (8.17 dB)
activate silent mode (synthesizer)	73% (29.97 dB)	73% (25.96 dB)	33% (18.64 dB)	0% (7.60 dB)
clear all notifications (siren2)	99% (31.08 dB)	100% (28.01 dB)	98% (22.13 dB)	26% (9.83 dB)
turn on every heater (violin)	100% (29.01 dB)	100% (26.48 dB)	93% (19.61 dB)	2% (8.32 dB)
call the police now (oboe)	100% (29.61 dB)	99% (25.37 dB)	97% (19.91 dB)	21% (8.86 dB)

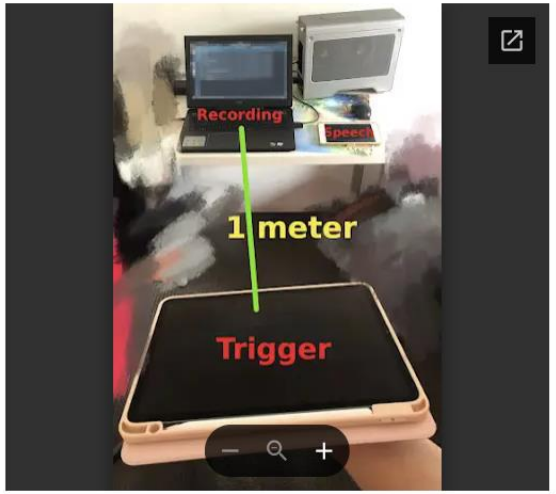
Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Module Backdoor Attack - TrojanModel

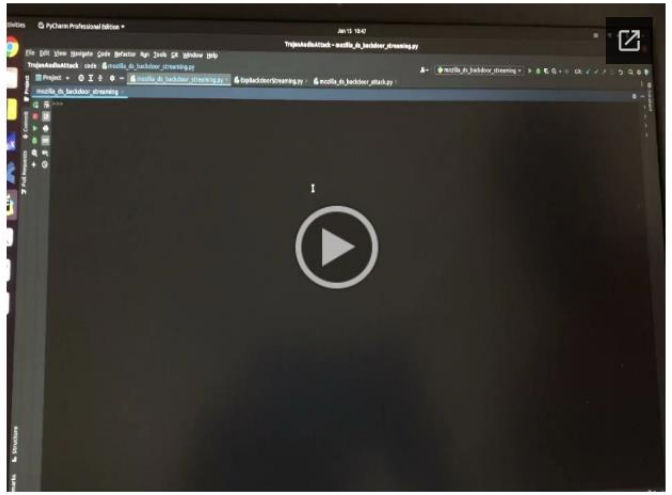
Online examples: https://sites.google.com/view/trojan-attacks-asr

Real-world Attack Demo 1

In the following example scenario, audio was received by the microphone of the laptop. The laptop was used for recording and transcribing input audio. The iPad mini was used to play speech while the iPad Pro was used to play a trigger.



The trigger was played at a distance of 1 meter away from the microphone.

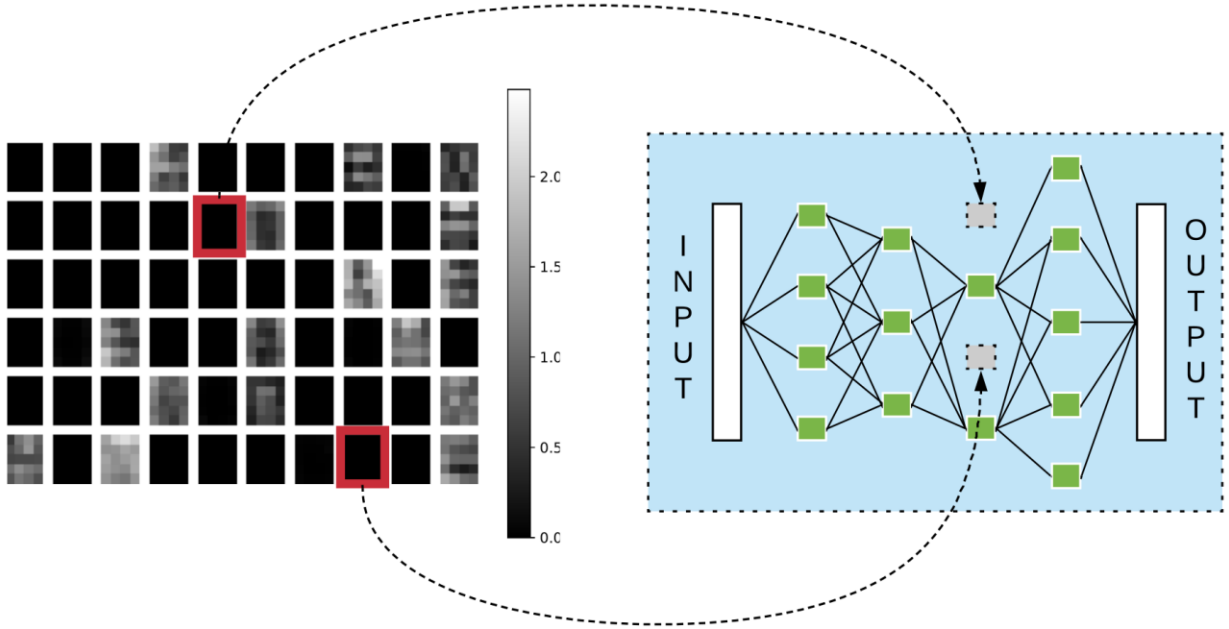


The benign speech was correctly transcribed.

Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1637-1653). IEEE.

Defense: Fine-Pruning

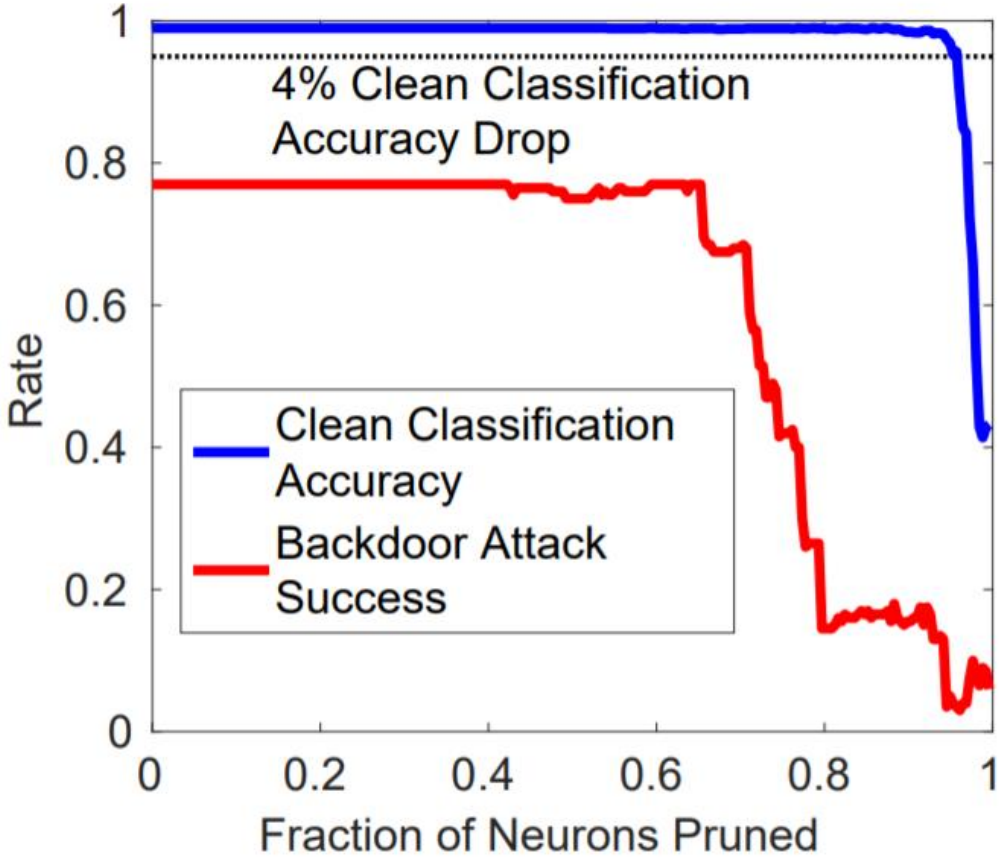
- Intuition
  - Different neurons are activated for clean input and input with triggers.
  - An explanation is that each neuron aims to detect a specific feature from the input.
  - Hence, neurons that detect the existence of triggers are not activated when input is clean and vice versa.
- As demonstrated in the Figure on the right, they proposed to prune neurons that do not activate for clean input.
  - The purpose is to remove potential Trojan from a target model.



Fine-pruning: Defending against backdooring attacks on deep neural networks, Liu et al. 2018

Defense: Fine-Pruning

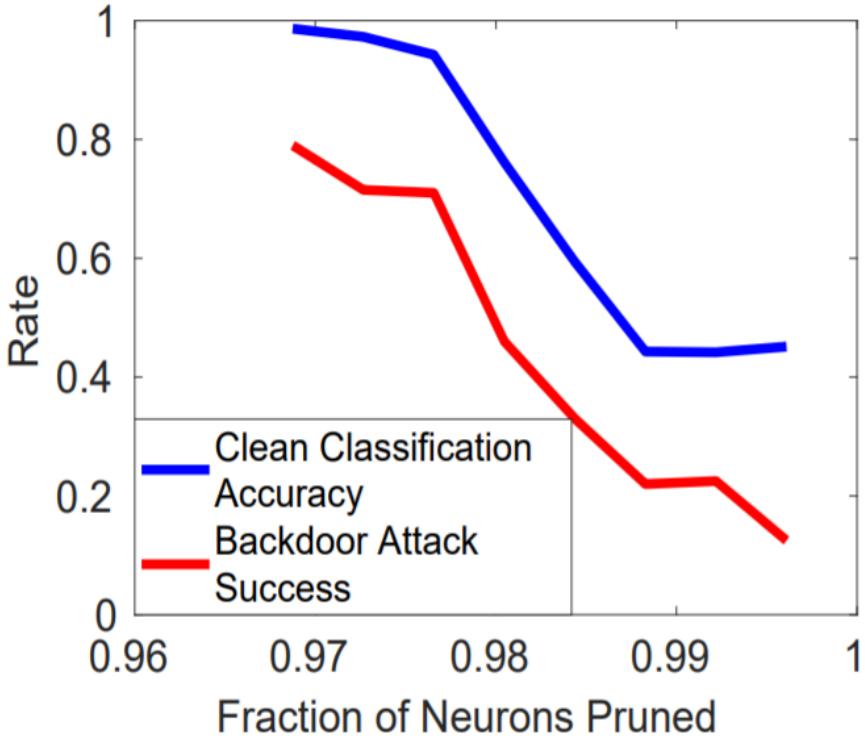
- Initial experiments show that although removing neurons that are not activated for normal input can remove potential Trojan, this will also degrade the performance of a model
  - This is because the architecture is changed.
  - Their experimental results are shown in the figure.
  - We can see that Trojan is successfully removed at the cost of only 4% decrease in accuracy for clean input.
  - However, 4% decrease in accuracy is not negligible.
  - **We must preserve model performance.**
    - **How?**



Fine-pruning: Defending against backdooring attacks on deep neural networks, Liu et al. 2018

Defense: Fine-Pruning

- Adaptive attack
  - In addition to preserving performance, robustness to adaptive attack is also critical.
  - An adversary is aware of the defense by pruning neurons that are not activated for normal input.
  - There are 3 stages.
    - In stage 1, an adversary normally trains a model on clean datasets.
    - In stage 2, the adversary prunes neurons that are not activated for clean input and then she retrains the pruned model on poisoned data sets.
    - In the last stage, the adversary restores the pruned neurons such that the compromised model is consistent with the initial architecture.
      - Restoring pruned neurons is necessary because if the architecture of a model is changed, this will arouse suspicion of a victim and the compromised model may not be used.
  - The core idea of this adaptive attack is to force remaining neurons in the pruned model to be activated when input contains triggers.
    - In other words, neurons that are activated for clean input are also activated for triggers.
    - This breaks the assumption of the previous defense.
    - As shown in the figure on the right, Trojan cannot be easily removed because performance of a target model is significantly affected.



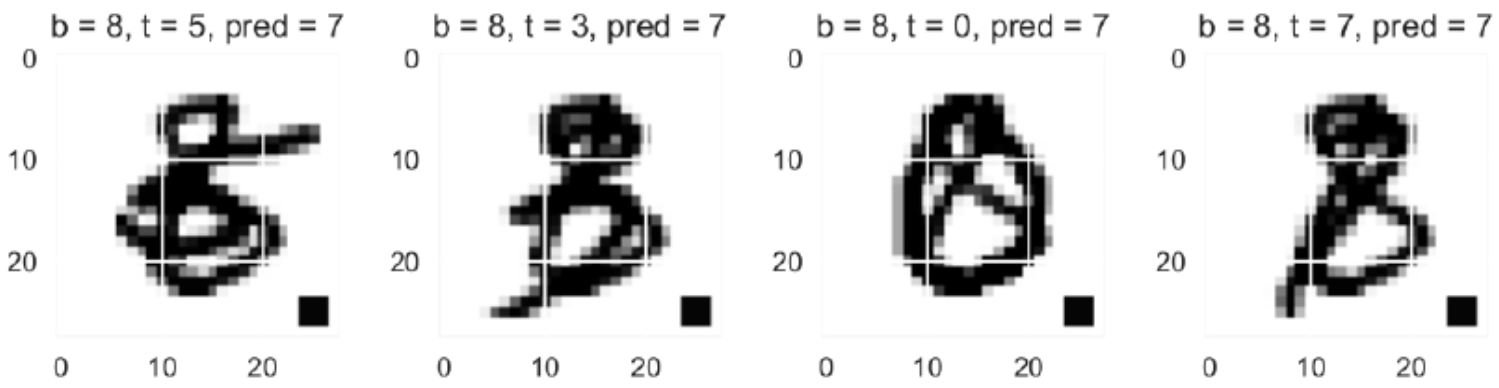
Fine-pruning: Defending against backdooring attacks on deep neural networks, Liu et al. 2018

Defense: Fine-Pruning

- Fine-Pruning
  - Defend against this adaptive attack and preserve performance
  - Two stages of defense
    - Firstly, prune the neurons that do not react to clean input.
    - Then, fine-tune the network on a clean training set.
    - Hence called **Fine-Pruning**.
  - The underlying reason for this strategy is that fine-tuning a pruned model can effectively destroy potential Trojan since model weights are changed.
    - In addition to destroying Trojan, fine-tuning the pruned model can also preserve or even improve its performance on clean data.
    - Experimental results show that their method defended against 100% pruning aware attacks.
    - The decrease in accuracy is only **0.2%**.

Fine-pruning: Defending against backdooring attacks on deep neural networks, Liu et al. 2018

Defense: STRIP



- Intuition
  - Empirically, triggers are input-agnostic.
  - Examples are shown in the figure on the top
    - it means that if a trigger exists, the output will be the same regardless the input content.
  - This observation inspires the strategy to detect Trojan attacks via repeatedly mixing input with another clean input which has a different label.
    - The intuition is that predictions for clean input will be altered randomly.
    - Predictions for input with triggers will stay stable.

Design and evaluation of a multi-domain Trojan detection method on deep neural networks, Gao et al. 2021

Defense: STRIP

- STRIP algorithm
  - An input is mixed with multiple other clean input.
  - Entropy of each perturbed input is then calculated based on the equation given.

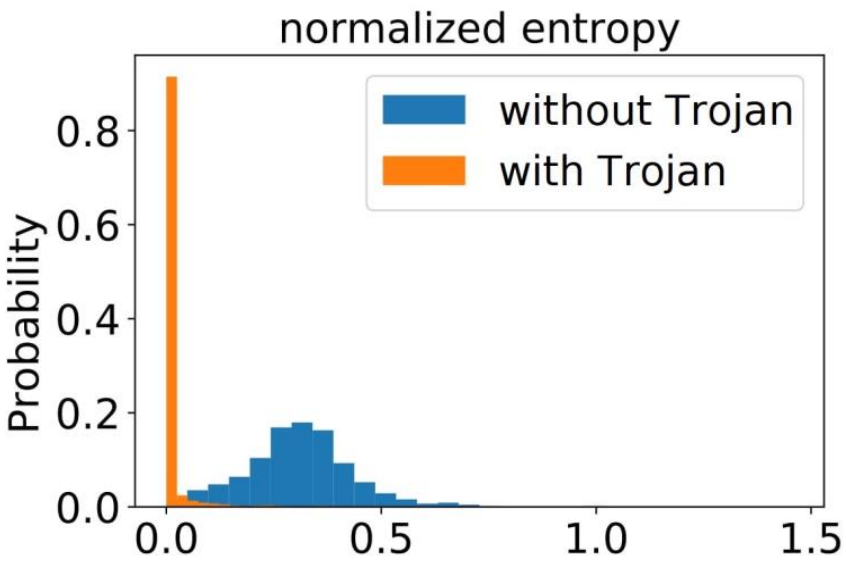
$$\mathbb{H}_n = -\sum_{i=1}^{M} y_i \times \log_2 y_i$$

    - where H_n is entropy for the nth perturbed input,
    - y_i indicates the probability of being classified as class i, and M is the total number of classes.
  - Then, the entropy is averaged and larger entropy means more randomness in predictions
    - This indicates higher possibility for the input being clean.
  - On the other hand, smaller entropy indicates less randomness
    - This means higher possibility for the existence of a trigger since there is less randomness in predictions.
  - By assuming the entropy for clean input follows a **Normal (Gaussian) Distribution**, anomaly detection is employed to detect the existence of a trigger in new input.
    - In practice, the entropy distribution for clean input can be calculated in advance.

Design and evaluation of a multi-domain Trojan detection method on deep neural networks, Gao et al. 2021

Defense: STRIP

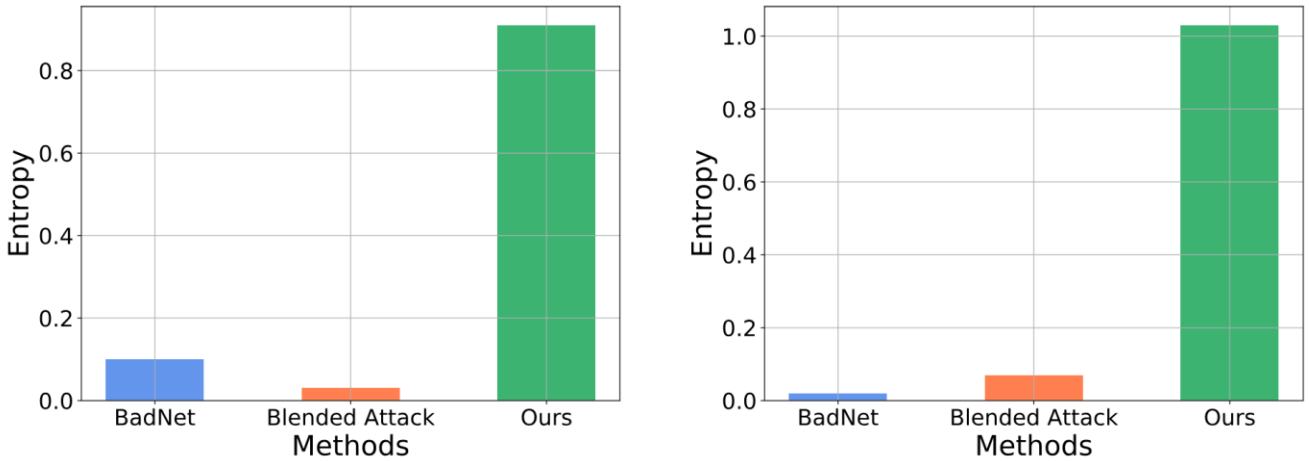
- Effectiveness
  - Experimental results showed that this method is effective for both image and speech recognition.
    - Although this method was initially designed for defending against backdoor attacks for image recognition.
  - An example of entropy distribution for clean input and input with triggers is shown in the figure on the right.
    - The entropy of input containing a trigger is concentrated at low values.
    - The entropy distribution for clean input spreads across a large range.
    - This is consistent with previous discussion that there is more randomness in predictions for clean input when mixed with other clean input.
    - Furthermore, the entropy distribution for clean input follows a normal distribution.
  - Experiments showed that detection rate was around 95% by setting false rejection rate to 3%.
    - This demonstrates the effectiveness of this defense method.



Design and evaluation of a multi-domain Trojan detection method on deep neural networks, Gao et al. 2021

Defense: STRIP

- Breaking the assumption of STRIP
  - An attack can bypass STRIP if it breaks the assumption of input-agnostic triggers.
  - Input-dependent backdoor: **Invisible Backdoor Attack with Sample-Specific Triggers.**



(a) ImageNet (b) MS-Celeb-1M

    - The entropy generated by STRIP of different attacks.
    - The higher the entropy, the harder the attack for STRIP to defend.
  - This attack is more resistant to STRIP compared to BadNet and Blended Attack.
    - It has the potential to bypass STRIP.

Design and evaluation of a multi-domain Trojan detection method on deep neural networks, Gao et al. 2021

Arms race

- Endless battle between backdoor attackers and defenders
  - Once a defense is proposed, there “always” will be adaptive attacks bypassing it.
    - Attacks can also bypass defense via breaking its assumptions.
  - This is different from adversarial examples.
    - Adversarial examples are intrinsic flaws (e.g., shortcut learning) in current deep learning models.
    - Learning feature that align with human perception will eventually eliminate adversarial examples.

Design and evaluation of a multi-domain Trojan detection method on deep neural networks, Gao et al. 2021

Summary

- Adversarial Examples
  - White-box attacks
  - Black-box attacks
  - Physical attacks
  - Universal adversarial examples
  - Robustness to adversarial examples
- Backdoor (Trojan) attacks
  - Visible backdoor
  - Invisible backdoor
  - Clean label backdoor
  - Module backdoor
  - Defense against backdoor attacks.

References

- Goodfellow, I.J., Shlens, J. and Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
- Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A., 2017. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.
- Delving into Transferable Adversarial Examples and Black-box Attacks
- Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. and Song, D., 2018. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1625-1634).
- Zong, W., Chow, Y.W., Susilo, W., Rana, S. and Venkatesh, S., 2021. Targeted universal adversarial perturbations for automatic speech recognition. In Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings 24 (pp. 358-373). Springer International Publishing.
- Xiao, C., Zhu, J.Y., Li, B., He, W., Liu, M. and Song, D., 2018. Spatially transformed adversarial examples. arXiv preprint arXiv:1801.02612.
- Bhattad, A., Chong, M.J., Liang, K., Li, B. and Forsyth, D.A., 2019. Unrestricted adversarial examples via semantic manipulation. arXiv preprint arXiv:1904.06347.
- Zong, W., Chow, Y.W., Susilo, W., Kim, J. and Le, N.T., 2022. Detecting Audio Adversarial Examples in Automatic Speech Recognition Systems Using Decision Boundary Patterns. Journal of Imaging, 8(12), p.324.
- Geirhos, R., Jacobsen, J.H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M. and Wichmann, F.A., 2020. Shortcut learning in deep neural networks. Nature Machine Intelligence, 2(11), pp.665-673.
- Tsipras, D., Santurkar, S., Engstrom, L., Turner, A. and Madry, A., 2018. Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152.

References

- Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S., 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7, pp.47230-47244.
- Zong, W., Chow, Y.W., Susilo, W., Do, K. and Venkatesh, S., 2023, May. Trojanmodel: A practical trojan attack against automatic speech recognition systems. In 2023 IEEE Symposium on Security and Privacy (SP) (pp. 1667-1683). IEEE.
- Chen, X., Liu, C., Li, B., Lu, K. and Song, D., 2017. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526.
- Li, Y., Li, Y., Wu, B., Li, L., He, R. and Lyu, S., 2021. Invisible backdoor attack with sample-specific triggers. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 16463-16472).
- Fine-pruning: Defending against backdooring attacks on deep neural networks, Liu et al. 2018

Current Trends in AI for Cybersecurity (Deepfake Detection)

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

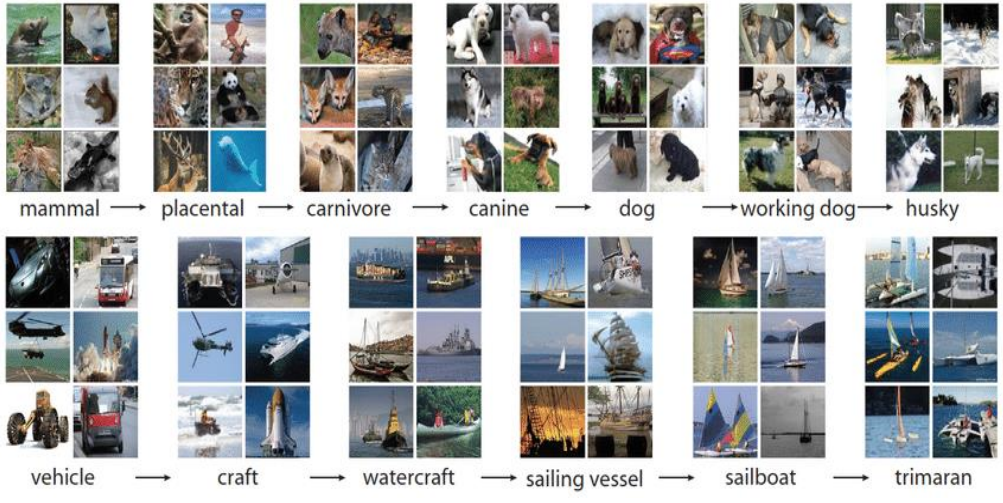
Outline

- GAN
- Security threats posed by GAN and countermeasures
- Beyond GAN

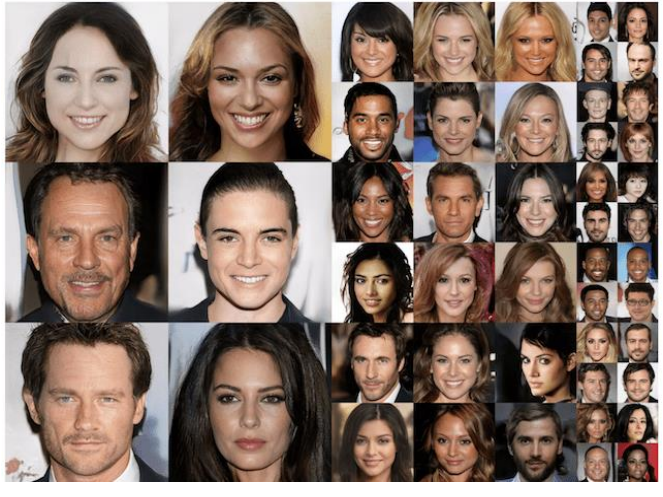
Intro

Deep learning:

- Discriminative models
  - Classifies input



- Generative models
  - Learns distribution of data



Intro

GAN

- has recently become a hot research topic

Generative adversarial nets

I Goodfellow, J Pouget-Abadie... - Advances in neural ..., 2014 - proceedings.neurips.cc

... We propose a new framework for estimating **generative** models via an adversarial process, in which we simultaneously train two models: a **generative** model G that captures the data ...

☆ Save 99 Cite Cited by 60870 Related articles All 61 versions »

- in 2020, approximately 28,500 papers related to GANs were published
- approximately 78 papers every day or more than three per hour

GAN

Generative Adversarial Nets

Ian J. Goodfellow,* Jean Pouget-Abadie,† Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,‡ Aaron Courville, Yoshua Bengio§
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

- published in Neurips 2014
  - Top AI conference
- Two versions online
  - arXiv vs. conference paper
  - Related work

Ian Goodfellow

DeepMind
Verified email at deepmind.com - Homepage
Deep Learning

FOLLOW

Cited by

	All	Since 2018
Citations	262417	245133
h-index	87	85
i10-index	144	140

Yoshua Bengio

Professor of computer science, University of Montreal, Mila, IVADO, CIFAR
Verified email at umontreal.ca - Homepage
Machine learning deep learning artificial intelligence

FOLLOW

Cited by VIEW ALL

	All	Since 2018
Citations	709126	594552
h-index	225	200
i10-index	791	708

GAN

A gossip about GAN

Juergen Schmidhuber

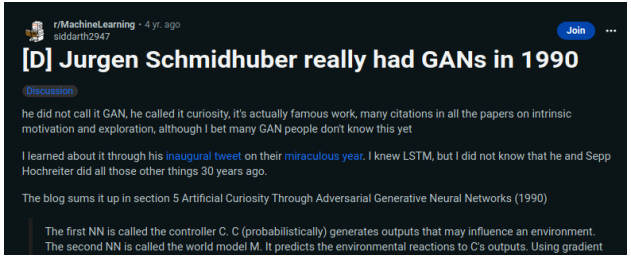
FOLLOW

King Abdullah University of Science and Technology / The Swiss AI Lab, IDSIA / University of Lugano
Verified email at kaust.edu.sa - Homepage

computer science artificial intelligence reinforcement learning neural networks physics

Cited by		VIEW ALL
	All	Since 2018
Citations	218083	174204
h-index	116	91
i10-index	417	272

Inventor of Long Short Term Memory (LSTM) model.



r/MachineLearning • 4 yr. ago
siddarth2947

Join

[D] Jurgen Schmidhuber really had GANs in 1990

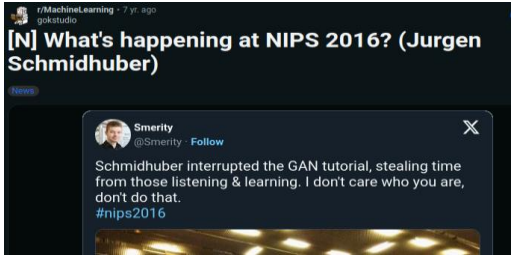
Discussion

he did not call it GAN, he called it curiosity, it's actually famous work, many citations in all the papers on intrinsic motivation and exploration, although I bet many GAN people don't know this yet

I learned about it through his inaugural tweet on their miraculous year. I knew LSTM, but I did not know that he and Sepp Hochreiter did all those other things 30 years ago.

The blog sums it up in section 5 Artificial Curiosity Through Adversarial Generative Neural Networks (1990)

> The first NN is called the controller C. C (probabilistically) generates outputs that may influence an environment. The second NN is called the world model M. It predicts the environmental reactions to C's outputs. Using gradient



r/MachineLearning • 7 yr. ago
gokstudio

[N] What's happening at NIPS 2016? (Jurgen Schmidhuber)

News

Smerity
@Smerity · Follow

Schmidhuber interrupted the GAN tutorial, stealing time from those listening & learning. I don't care who you are, don't do that.
#nips2016

Mu Li

FOLLOW

Amazon
Verified email at amazon.com - Homepage

Machine Learning Distributed Systems

Cited by		VIEW ALL
	All	Since 2018
Citations	17708	16010
h-index	37	36
i10-index	64	60

“Techniques are re-invented all over the time. People give credit to those who make them popular.”

GAN

Why need GAN? (motivation)

- “Deep generative models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context.”

Variational Autoencoder:

- Optimize evidence lower bound (ELBO)

GAN

What is GAN?

- adversarial nets is a **framework**, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution.
- The two-player minimax game -> Nash Equilibrium

Generative Model

a team of counterfeiters

Discriminative Model

police

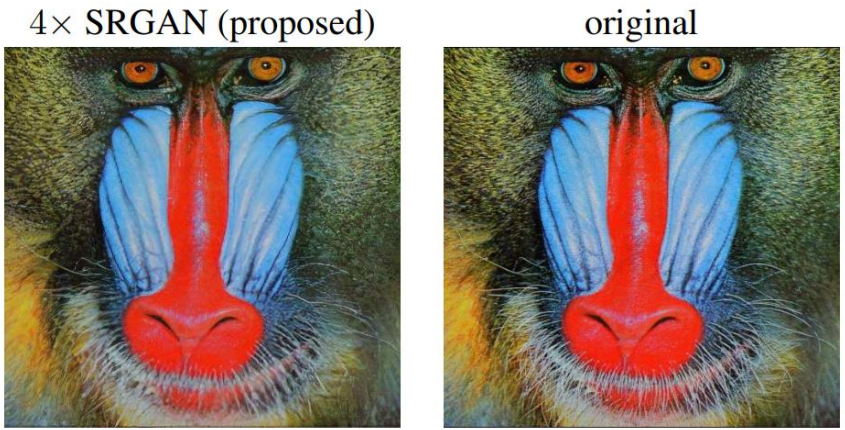
GAN

Results

Rapid Advance in GAN

Super-Resolution GAN

- a variant of GAN for image superresolution
- infer photo-realistic natural images for 4× upscaling factors
- is able to recover photo-realistic textures from heavily downsampled images



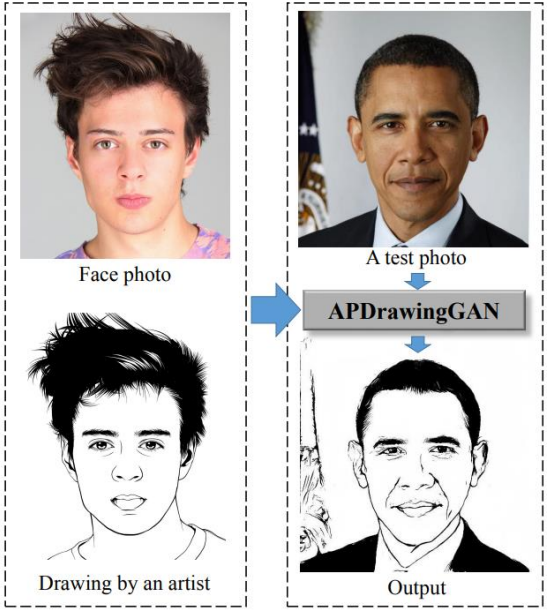
Super-resolved image (left) is almost indistinguishable from original (right). (4× upscaling)

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z. and Shi, W., 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4681-4690).

Rapid Advance in GAN

APDrawingGAN

- Artistic portrait drawings have a highly abstract style, containing a sparse set of continuous graphical elements such as lines.
- Artists tend to use different strategies to draw different facial features and the lines drawn are only loosely related to obvious image features



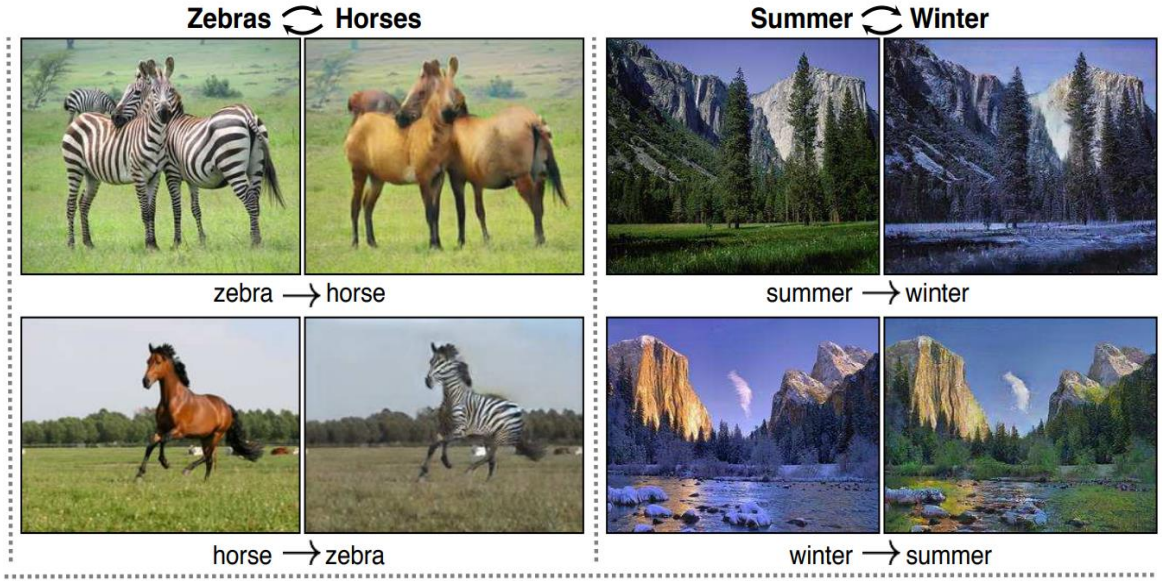
(left) An artist draws a portrait drawing using a sparse set of lines and very few shaded regions to capture the distinctive appearance of a given face photo.
(right) APDrawingGAN learns this artistic drawing style and automatically transforms a face photo into a high-quality artistic portrait drawing.

Yi, R., Liu, Y.J., Lai, Y.K. and Rosin, P.L., 2019. Apdrawinggan: Generating artistic portrait drawings from face photos with hierarchical gans. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10743-10752).

Rapid Advance in GAN

CycleGAN

- Given any two unordered image collections X and Y
- CycleGAN learns to automatically “translate” an image from one into the other.
- “Translate” horses into zebra and vice versa.
- “Translate ”summer scene into winter scene and vice versa.

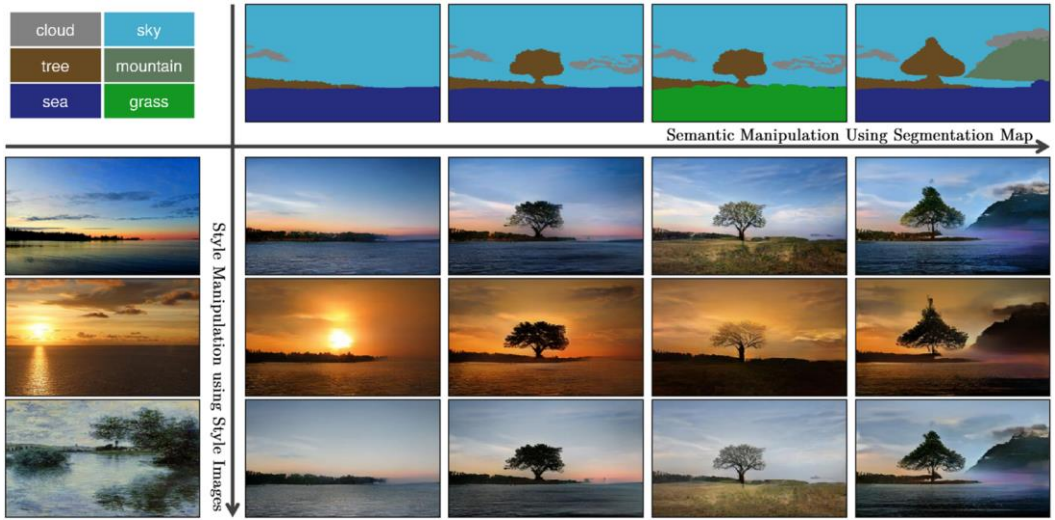


Zhu, J.Y., Park, T., Isola, P. and Efros, A.A., 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2223-2232).

Rapid Advance in GAN

GauGAN

- Allows user to control over both semantic and style as synthesizing an image
- The smentic (e.g., existence of a tree) is controlled via a label map (visualized in the top row).
- The style is controlled via the reference style image (visualized in the leftmost column)

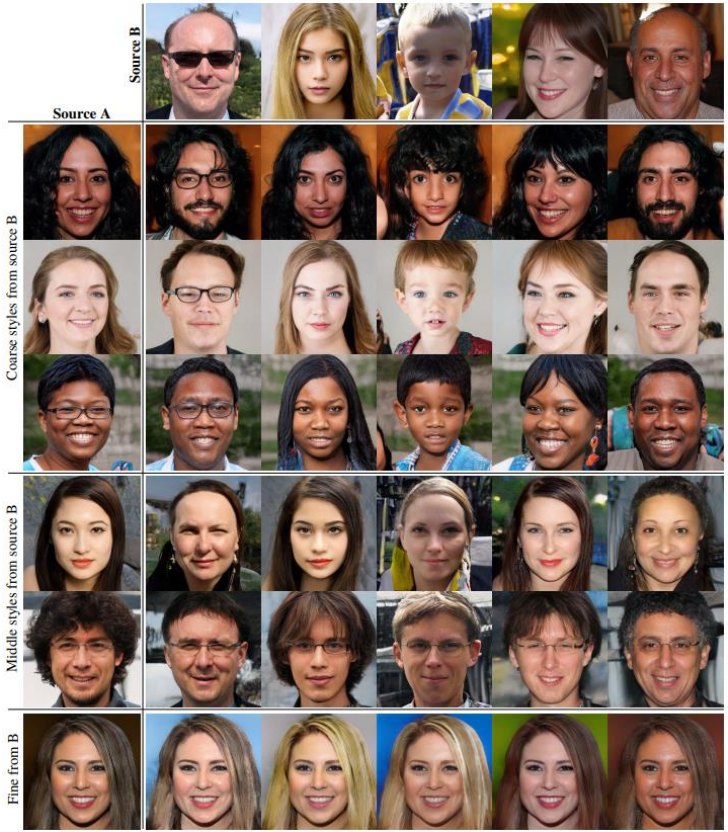


Park, T., Liu, M.Y., Wang, T.C. and Zhu, J.Y., 2019. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2337-2346).

Rapid Advance in GAN

StyleGan

- images generated by copying a specified subset of styles from source B and taking the rest from source A.
- Copying the styles corresponding to coarse spatial resolutions brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A.
- If instead copying the styles of middle resolutions from B, smaller scale facial features are inherited, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved.
- Finally, copying the fine styles from B brings mainly the color scheme and microstructure.



Karras, T., Laine, S. and Aila, T., 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4401-4410).

Fake Image Detection

GAN is double-edged sword

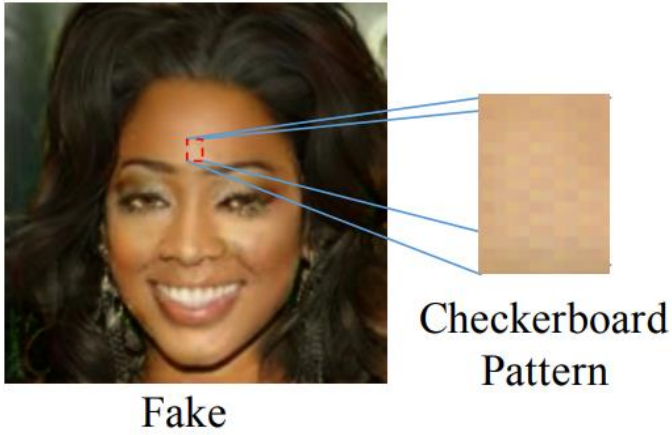
- Humans cannot distinguish between real or fake images anymore.
- Adversaries can use this technique to spread fake information or commit crimes.

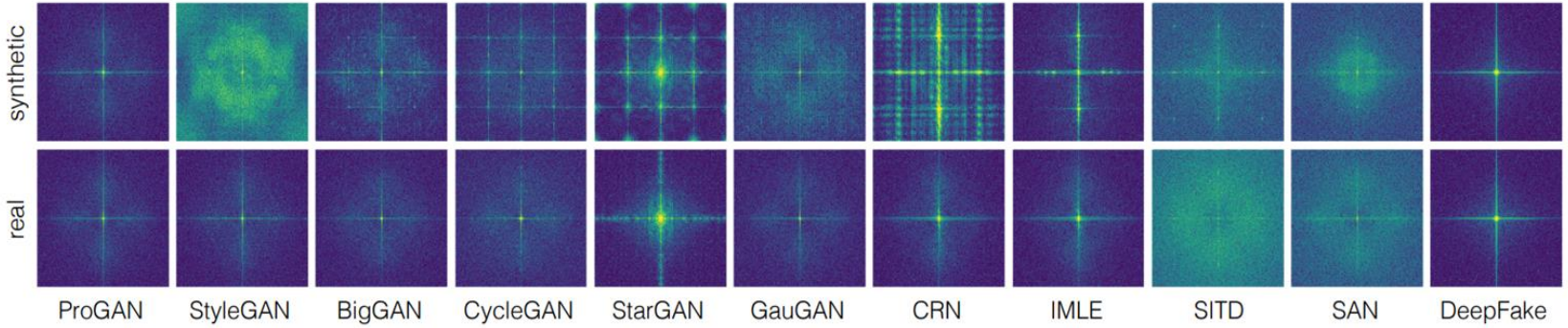
Is this a real or synthesized photo?

Fake Image Detection

Detecting fake images becomes an emerging research trend:

- Detect fake images based on artifacts introduced by deep generative models
- Checkerboard patterns identified in fake images.
- The figure below shows the average spectra of each high-pass filtered image, for both the real and fake images. There are periodic patterns (dots or lines) in most of the synthetic images.





Zhang, X., Karaman, S. and Chang, S.F., 2019, December. Detecting and simulating artifacts in gan fake images. In 2019 IEEE international workshop on information forensics and security (WIFS) (pp. 1-6). IEEE.
Wang, S.Y., Wang, O., Zhang, R., Owens, A. and Efros, A.A., 2020. CNN-generated images are surprisingly easy to spot... for now. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 8695-8704).

Fake Image Detection

However

- Adversaries can modify fake images to escape from detection.

Two limitations:

- The rapid advance in GAN will eventually invalidate detection methods.
  - Nash Equilibrium
- lack of explainability in model predictions
  - Prevent being used in law

The first is the original fake image. The remainings are modified by adversaries to escape from detection.

The judicial demand for **explainable artificial intelligence**
A Deeks - Columbia **Law** Review, 2019 - JSTOR
… in shaping the nature and form of "**explainable AI**" (xAI). Using the tools of the common **law**, courts can develop what xAI should mean in different **legal** contexts. There are advantages to …
Save Cite Cited by 198 Related articles All 10 versions

Explainable artificial intelligence, **lawyer's** perspective
Ł Górski, S Ramakrishna - … Conference on **Artificial Intelligence** and **Law**, 2021 - dl.acm.org
… This work was thought as the first step towards the identification of requirements of **explainable AI**-based systems that would involve **legal** perspective to a greater extent. For the …
Save Cite Cited by 25 Related articles

[PDF] **Explainable artificial intelligence** the new frontier in **legal** informatics
B Waltl, R Vogl - Jusletter IT, 2018 - wwwmatthes.in.tum.de
… This article explores the increasingly important topic of «**explainable AI**» and … of **explainability** as a property inherent to machine learning algorithms. It highlights that **explainability** can …
Save Cite Cited by 57 Related articles

Regulating **Explainable AI** in the European Union. An Overview of the Current **Legal** Framework (s)
M Ebers - … Overview of the Current **Legal** Framework (s)(August 9 …, 2020 - papers.ssrn.com
…) arguments against full transparency of **Artificial Intelligence** (**AI**) systems, especially in the … EU **law**, entitled to a right to explanation of automated decision-making, especially when **AI** …
Save Cite Cited by 22 Related articles

[HTML] **Explainable AI** under contract and tort **law**: **legal** incentives and technical challenges
P Hacker, R Krestel, S Grundmann… - **Artificial Intelligence** and …, 2020 - Springer
… **explainability** is an important **legal** category not only in data protection **law**, but also in contract

Fake Image Detection

Watermarking

- overcome the limitations of detecting artifacts in fake images
- Proactively detect fake images instead of reactively
- Goal embedding watermarks into generative models
  - such that fake images still contain predefined watermarks
  - training set

(a) (b) (c)

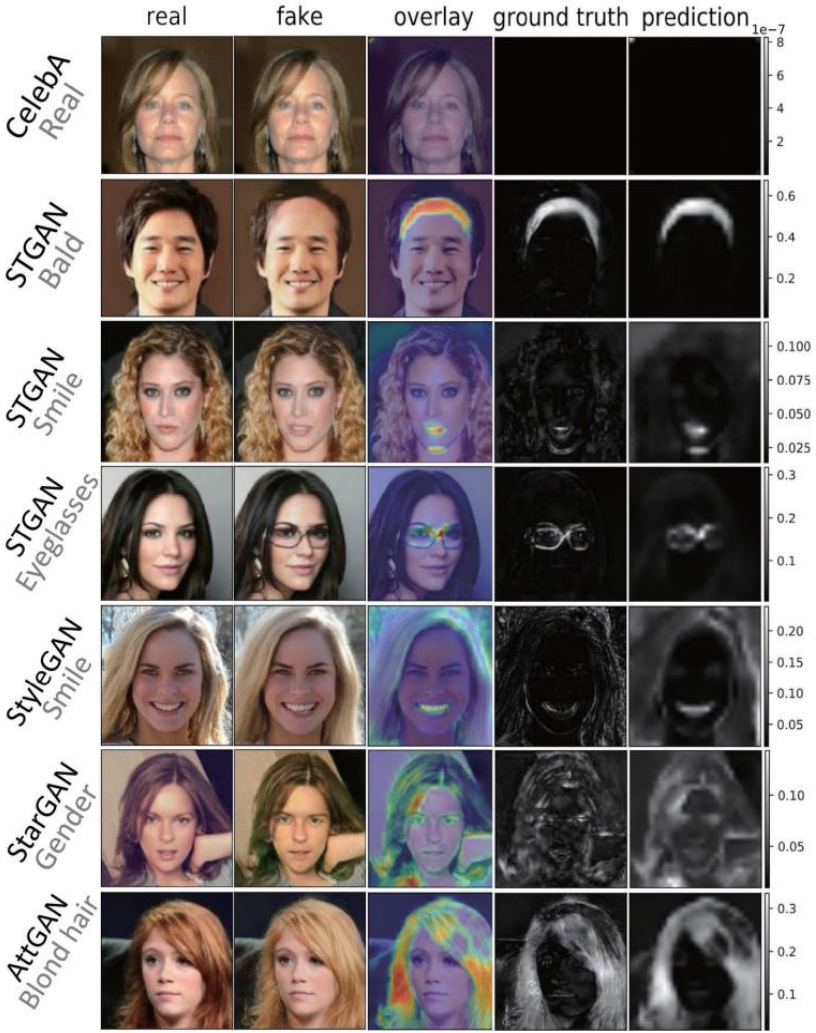
(a) Original real training samples. (b) Fingerprinted real training samples. (c) The difference between (a) and (b), 10× magnified for easier visualization.

Yu, N., Skripniuk, V., Abdelnabi, S. and Fritz, M., 2021. Artificial fingerprinting for generative models: Rooting deepfake attribution in training data. In Proceedings of the IEEE/CVF International conference on computer vision (pp. 14448-14457).

Fake Image Detection

Watermarking

- Embedding fragile watermarks into real images
  - If an adversary modifies parts of a watermarked image, the corresponding watermarks would be destroyed
  - defenders can easily locate modified parts.



Fake Image Detection

Watermarking

- The role of defenders and adversaries exchanges
  - Nash Equilibrium is beneficial for defenders
  - In the end, watermarks will be unremovable
- Explainable

The arms race between defenders and adversaries continues...

Beyond GAN

Variational Autoencoders (VAE)

- Following variational bayes inference, VAEs are generative models that attempt to reflect data to a probabilistic distribution and learn reconstruction that is close to its original input.

Flow

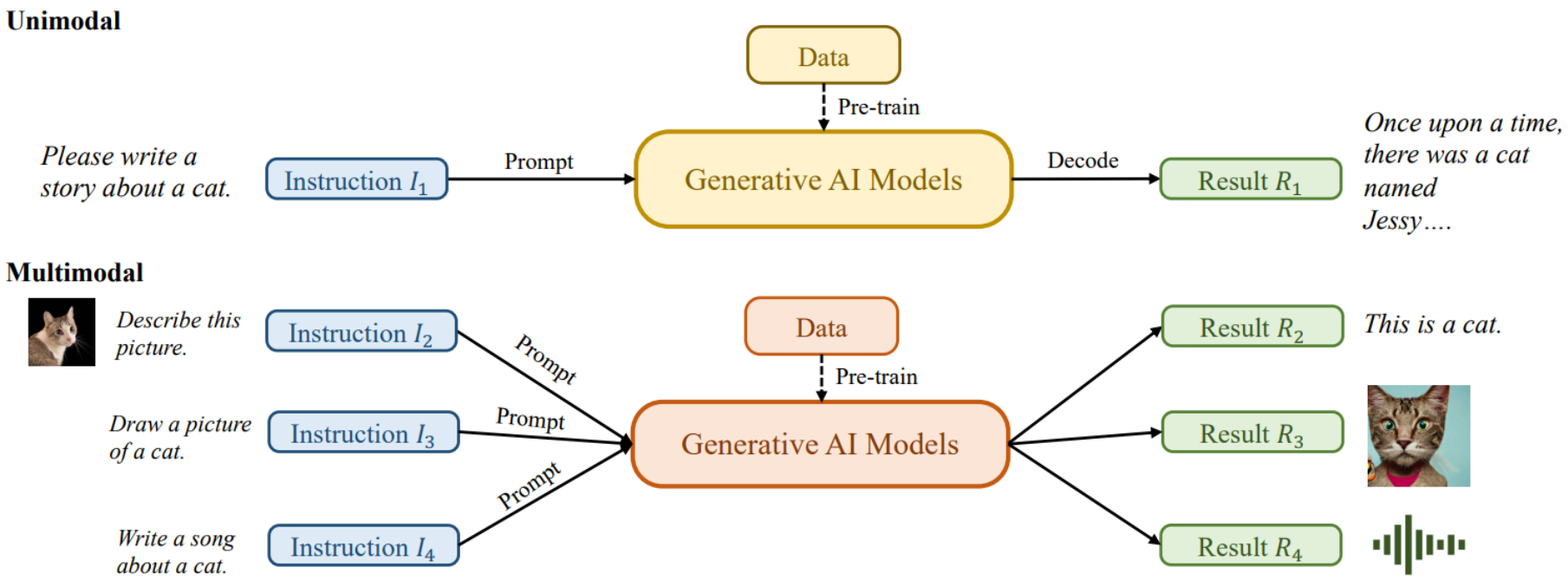
- A Flow is a distribution transformation from simple to complex by a sequence of **invertible** and differentiable mappings.

Diffusion

- The Generative Diffusion Model (GDM) is a cutting-edge class of generative models based on probability, which demonstrates state-of-the-art results in the field of computer vision. It works by progressively corrupting data with multiple-level noise perturbations and then learning to reverse this process for sample generation.

Beyond GAN

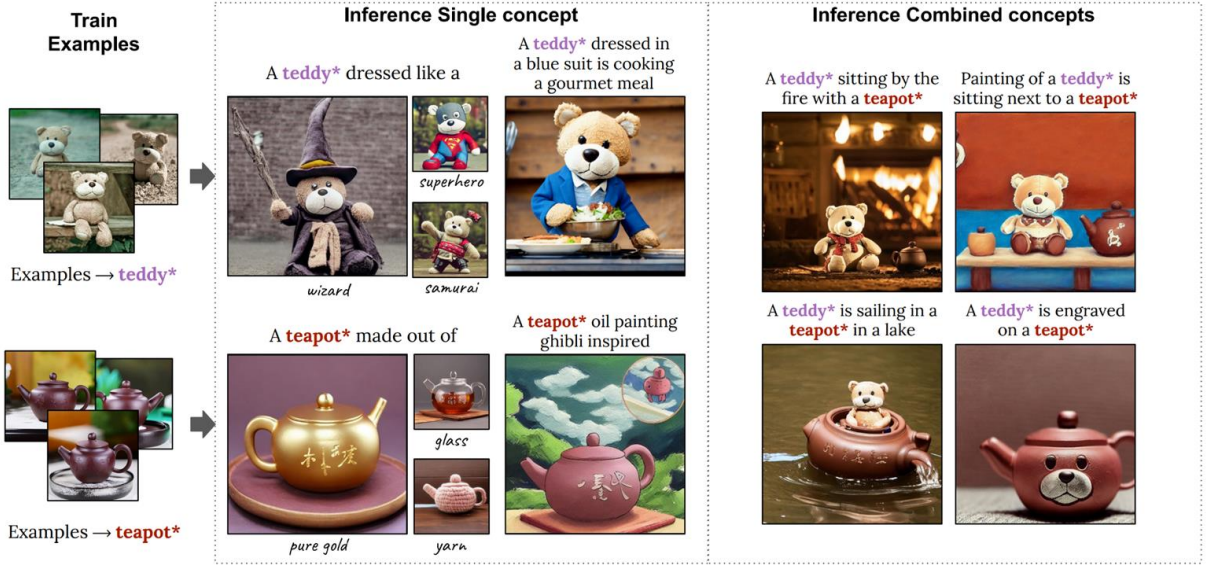
From Unimodal to MultiModel



Beyond GAN

Perfusion (2023 NVIDIA)

- A new text-to-image personalization method.
- With only a 100KB model size per concept (excluding the pretrained model, which is a few GBs), trained for roughly 4 minutes, Perfusion can creatively portray personalized objects.
- It allows significant changes in their appearance, while maintaining their identity.
- Perfusion can also combine individually learned concepts into a single generated image.



Questions?

Exam and Revision

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

- Things to know for the final exam
- Subject Revision
- Q&A

Final Exam – Online invigilated Moodle Quiz

- 3 Hours
- Close-book exam
  - No calculations.
    - Calculations have already been done in Quiz 2 and the assignment.
  - Knowledge and intuition are the key.

Final Exam – question types

- 21 - 31 questions (50 marks), including:
  - 7 multiple choice (14 marks in total)
  - 4 True or False (6 marks in total)
  - 10 - 20 short answer (30 marks in total)
    - Convince me that you understand the ideas.
    - Make sure your handwriting is clear enough.

- A minimum of 40% of the final exam marks
  - Get at least 20 marks to avoid TF.

Final Exam

- For all multiple-choice questions, one or multiple answers can be correct
- For each multiple-choice question, you'll receive
  - full marks if all options are checked correctly (i.e. all correct answers are checked, wrong answers are not checked)
  - partial marks if part of options are checked correctly (i.e. part of correct answers are checked, wrong answers are not checked)
  - zero otherwise
- **No code or calculations** in the final exam

How to prepare

- Scope: Lecture slides and lab notes
- Review the lectures and examples
- Review the lab notes
- Review the assignments
- Attempt the quiz “Sample Exam for CSIT375” (available on Moodle)

During the Exam

- Answer **ALL** questions
  - Read questions carefully
  - You can attempt the questions in any order, arrange your time wisely
- Short answer questions
  - **BRIEFLY** answer each question with important and the most relevant points.
  - There is no need to include explanation if the question does not require it.

Revision

Introduction

- AI, Machine learning, Deep learning
  - ML: classification/regression/clustering, supervised/unsupervised learning
- Cybersecurity: C-I-A triad
  - **C**onfidentiality**,** Integrity, Availability
- Common types of cybersecurity threats
  - Phishing, ransomware, malware, social engineering
- Applications of AI in Cyber security
  - Monitoring (intrusion, privacy violation, exfiltration, ...)
  - Analysis (causation, consequence, correlation, summarization)
- Limitations of AI in Security
  - False negatives/false positives
  - False positives + inexplicable results → Signal fatigue

Linear regression

- How to learn a linear regression model $\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 x}$ given a labelled dataset ($\boldsymbol{\theta}_0$ = b, $\boldsymbol{\theta}_1$= m, and $\boldsymbol{h_\theta(x)}$= y' when using y' = mx + b)?
  - By minimizing ***mean squared error***. That is:

This problem is referred to as an ***optimization problem*** with the objective of:

$$\underset{\theta_0,\theta_1}{\text{minimizing}}\, \boldsymbol{J(\theta_0, \theta_1)}$$

$$\text{minimize}\, \frac{1}{2n}\sum_{i=1}^{n}\left(y'^{(i)} - y^{(i)}\right)^2$$

$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, \boxed{\frac{1}{2n}\sum_{i=1}^{n}\left((\theta_0 + \theta_1 x)^{(i)} - y^{(i)}\right)^2}$$

Known as ***cost function*** $\boldsymbol{J(\theta_0, \theta_1)}$ **MSE**

$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_0, \theta_1)}$$

Linear regression

- Gradient Descent Steps
  - Have some cost function $\boldsymbol{J(\theta_0, \theta_1)}$ (here we are using MSE)
  - Start with some guesses for θ_0, θ_1
    - a common choice is to set them both initially to zero or random values.
  - Repeat until convergence{

$$temp_0 = \theta_0 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_0}$$

$$temp_1 = \theta_1 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_1}$$

$$\theta_0 = temp_0$$
$$\theta_1 = temp_1$$

}

$$\frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right)$$

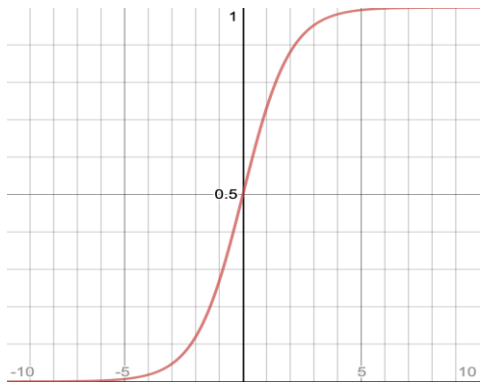
Partial derivatives of MSE with respect to parameters

$$\frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right) \; .x^{(i)}$$

Logistic regression

- The Logistic Regression Model

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



$g(z)$

  - $\boldsymbol{x}$ is the ***feature vector***: $\boldsymbol{x} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_m}]^T$
  - $\boldsymbol{\theta}$ is the ***parameter vector***: $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_m}]^T$
  - $\boldsymbol{g(z)}$ is the sigmoid function: $g(z) = \frac{1}{1+e^{-z}} \in [0,1]$
  - $h_\theta(x)$: the probability of input belonging to the class labeled with 1
- ***After*** learning the model, we can then apply thresholding to predict the binary output as follows:

$$if\ h_\theta(x) < 0.5 \text{ predict } 0$$
$$if\ h_\theta(x) \geq 0.5 \text{ predict } 1$$

Logistic regression

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \dots, \boldsymbol{x_m}]^T$?
  - By minimizing the following cost function:

$$\underset{\theta}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} -y^{(i)} \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right) - (1-y^{(i)}) \log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right)$$

Cross entropy loss

- where n is the total number of examples, the sum is over all training inputs, $y^{(i)}$ true label, log(predicted probability observation)
- cross entropy loss is also used in classification task using NN (lab task)

$$J = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log\left(a^{(i)}\right) + (1-y^{(i)}) \log\left(1-a^{(i)}\right)\right)$$

Model evaluation

- Accuracy = $\frac{\textit{number of correctly predicted data points}}{\textit{total number of data points}}$
- Different types of errors
  - False positive, false negative
- Precision and Recall
  - Recall: TP/(TP + FN)
  - Precision: TP/(TP + FP)
- Confusion matrix

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

- F-score: A measure that combines ***P***recision and ***R***ecall
  - F_β, F_1

Spam detection

- Spam and phishing
- Spam Filtering Techniques
  - Challenge-Response Filtering, Blacklists and Whitelists, Rule based filters, Content based filters
- Pre-processing Text in Email Messages
  - Tokenization, Vectorization
  - Bag-of-words, n-grams, Vector Space Model,
  - TF-IDF
    - raw term frequency, log-frequency weight
    - document frequency, Inverse document frequency
    - TF-IDF Weighting: $= (\mathbf{1} + \boldsymbol{log}(tf_{t,d})) \times \boldsymbol{log}\left(\frac{\boldsymbol{N}}{\boldsymbol{df_t}}\right)$

Neural network

How to train a Neural Network: Summary (More details in the lab task)

1. Define the neural network structure (model representation)
2. Initialize the model's parameters
   - initialize the weights matrices with random values
   - initialize the bias vectors as zeros
3. Loop:
   - Implement forward propagation
   - Compute loss (cross-entropy loss as shown in the lab notes)
   - Implement backward propagation to get the gradients
   - Update parameters (gradient descent)

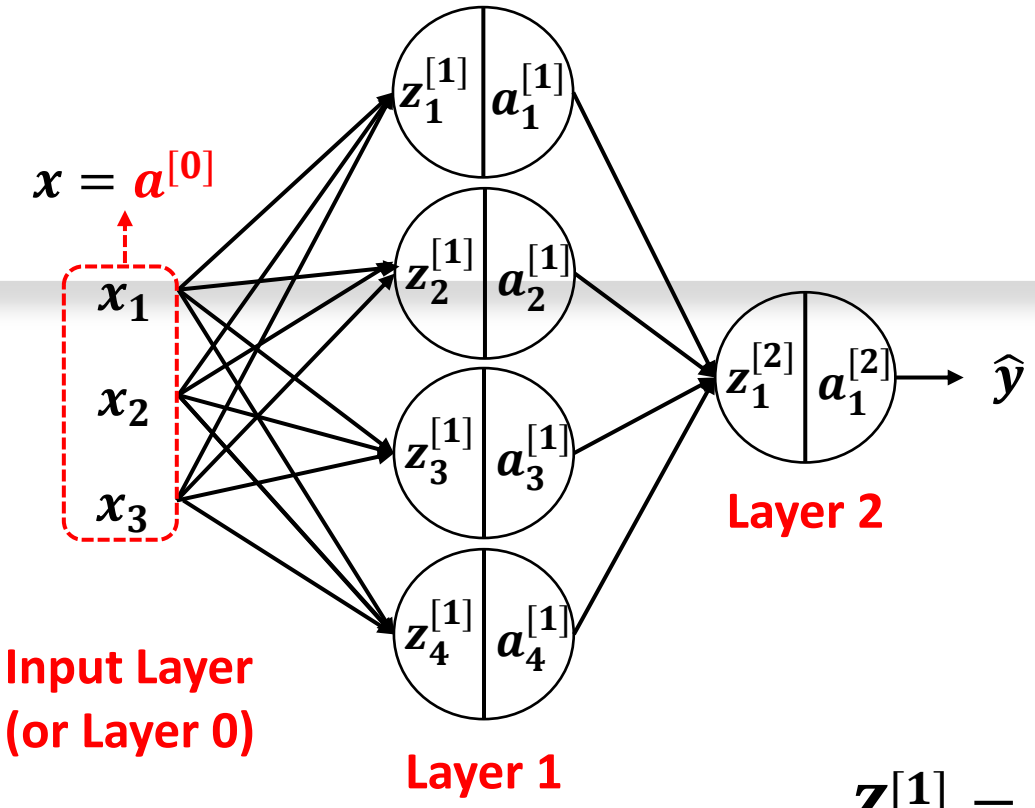
How to make predictions?

Use forward propagation to predict results

Reminder: predictions = $y_{prediction} = \mathbb{1}\{\text{activation} > 0.5\} = \begin{cases} 1 & \text{if } activation > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Neural network

- Forward propagation



Assume 1 training example:

$$z^{[1]} = w^{[1]^T} a^{[0]} + b^{[1]}$$

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

$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

Assuming n examples

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(n)} \\ x_3^{(1)} & x_3^{(2)} & & x_3^{(n)} \end{bmatrix}$$

$$Z^{[1]} = w^{[1]^T} A^{[0]} + b^{[1]}$$

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

$$Z^{[2]} = w^{[2]^T} A^{[1]} + b^{[2]}$$

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

***After* Vectorization**

Anomaly detection

- Anomalies (outliers)
  - data object that deviates significantly from the rest of the objects
- Types of anomalies
  - Global anomalies, Contextual anomalies, Collective anomalies
- Applications of anomaly detection
  - Fraud detection, Industrial damage detection, Cyber Intrusion detection
- Intrusion detection system (IDS)
  - Network-based, Host-based, signature-based, anomaly-based
  - Feature engineering for HIDS (IoCs), for NIDS (DPI)
- Anomaly detection techniques
  - With labels – train classifiers, confidence score, softmax function
  - Without labels – statistical methods, parametric (Gaussian), nonparametric (histogram)

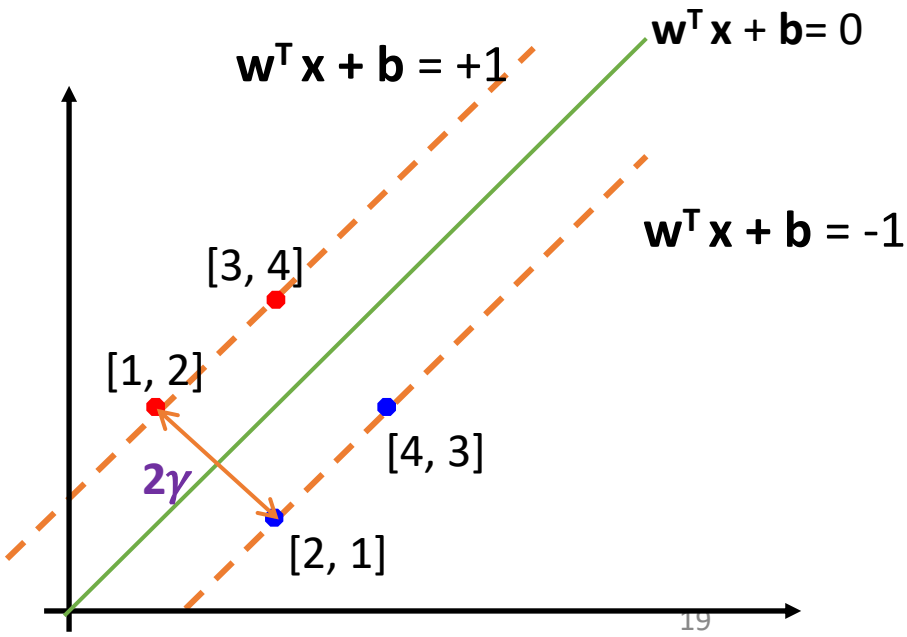
SVM

x	y
[1, 2]	+1
[2, 1]	-1
[3, 4]	+1
[4, 3]	-1

- limitations of Perceptron model
- Linear SVM: Maximum margin linear classifier
  - Objective: select a hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ that maximizes the distance between the hyperplane and examples in the training set
  - Constraints that all data follows:

    $\mathbf{w}^T\mathbf{x}^i + b \geq 1$ if $y^i = +1$

    $\mathbf{w}^T\mathbf{x}^i + b \leq -1$ if $y^i = -1$
  - For support vectors, the inequality becomes equality
  - The margin is: $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$



Decision Tree

- Malware: malicious software, steal data, damage computer systems
- Common types of Malware
  - Virus, trojan, botnet, downloader, rootkit, ransomware…
- Malware analysis methodology
  - Static (code) analysis
  - Dynamic (behavioral) analysis
- Decision tree
  - Entropy: uncertainty
  - Information Gain: goodness of a split
  - Build a Decision Tree: ID3 algorithm

Entropy and Information gain

- **Entropy $H(Y)$ of a random variable Y with n different possible values:**

$$H(Y) = -\sum_{i=1}^{n} P(y_i) log_2 P(y_i)$$

- **Conditional entropy $H(Y|X = x_j)$ of Y given $X = x_j$:**

$$H(Y|X = x_j) = -\sum_{i=1}^{n} P(y_i|X = x_j)\, log_2 P(y_i \mid X = x_j)$$

- The above equation: split the data according to the value of X, the entropy of random variable Y with $X = x_j$.

- **Expected entropy $H(Y|X)$ after split:**

$$H(Y|X) = \sum_{j=1}^{k} P(x_j)\, H(Y|X = x_j)$$

- The above equation: X has k possible values, $P(x_j)$ is the probability that $X = x_j$.

- **Information gain** $I(Y, X)$: expected reduction in entropy of target variable Y after split over variable X

$$I(Y, X) = H(Y) - H(Y|X)$$

Case study

- Explore and prepare the data
  - Data distribution, class imbalance
  - Feature scaling: standardization/normalization, consistent transformations to both training & test sets
- Multiclass classification: split into multiple binary classification problems
  - one-versus-all: train one classifier per class
  - one-versus-one: train one classifier per class pair
- Supervised learning: DT, Random Forest, SVM
- PCA

Adversarial Machine Learning

- Adversarial examples
  - White-box
  - Black-box
  - Physical
  - Detection
  - Adversarial training
- Backdoor attack
  - Visible triggers
  - Invisible triggers
  - Clean-label attack.

Deepfake detection

- GAN
- Detecting artifacts
- Watermarks

Good Luck!

Questions?